

AIRLINE RESERVATION SYSTEM

A MINI-PROJECT REPORT

Submitted by

SIVATHANU K P **220701280**

SUKESH RAJ V **220701291**

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

An Autonomous Institute

CHENNAI-602105

NOV-DEC

2023

BONAFIDE CERTIFICATE

Certified that this project “**AIRLINE RESERVATION SYSTEM**” is the bonafide work of “ **SIVATHANU K P (220701280), SUKESH RAJ V (220701291)**” who carried out the project work under my supervision.

SIGNATURE

Dr.R.SABITHA

Professor and Academic Head,
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai-602 105

SIGNATURE

Ms.V.JANANEE

Assistant Professor(SG)
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai-602 105

Submitted for the Practical examination to be Held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The Airline Management System is a software application designed to streamline the process of airline booking and management. Developed using Python and Tkinter for the user interface and MySQL for database management, this system provides users with the ability to sign up, log in, view available flights, make bookings, and manage their bookings.

The system comprises three main modules: User Authentication, Flight Management, and Booking Management. The User Authentication module allows users to sign up for new accounts or log in with existing credentials. Upon successful authentication, users are directed to the User Dashboard, where they can view their bookings and book new flights. The Flight Management module displays a list of available flights, including details such as flight name, arrival, departure, and date. Users can select from these flights and proceed to make bookings through the Booking Management module.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 SQL

2.2.2 PYTHON

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6. CONCLUSION

7. REFERENCES

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

The Airline Management System is designed to streamline the process of booking flights and managing user information. It provides a user-friendly interface for customers to sign up, log in, view available flights, book flights, and manage their bookings. The system is implemented using Python for the application logic, Tkinter for the graphical user interface, and MySQL for database management

1.2 OBJECTIVES

The primary objectives of the Airline Management System are:

- To provide a seamless user experience for flight booking.
- To manage customer data securely.
- To display available flights and bookings efficiently.
- To integrate with a MySQL database for data storage and retrieval.

1.3 MODULES

The system is divided into several key modules:

1. **Signup Module:** Allows new users to register by providing their details.
2. **Login Module:** Authenticates existing users based on their credentials.
3. **User Dashboard:** Provides options for viewing bookings, booking new flights, and viewing available flights.
4. **Booking Module:** Enables users to book flights by selecting from available options.
5. **Flight Display Module:** Shows available flights to users with detailed information.

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The Airline Management System is a software application designed to simplify and manage the process of booking flights and handling user information. This section provides a concise overview of the various software components, languages, and technologies used in the development of the system.

The technologies used in this project are:

- **Python:** A high-level, interpreted programming language known for its ease of use and readability. It is used for the application logic.
- **Tkinter:** The standard GUI library for Python, used to create the graphical user interface for the system.
- **MySQL:** A popular relational database management system used for storing and managing data securely and efficiently

2.2 LANGUAGES

2.2.1 SQL

SQL (Structured Query Language) is used for managing and manipulating relational databases. In this project, SQL is used to interact with the MySQL database for operations such as inserting, updating, and retrieving data. SQL ensures data integrity and allows for efficient data manipulation.

2.2.2PYTHON

Python is a versatile programming language used for both the application logic and the graphical user interface (GUI). Tkinter, a standard GUI library in Python, is utilized to create the user interface for this project. Python's readability and extensive libraries make it an ideal choice for developing the Airline Management System.

CHAPTER 3

REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

The system requires the following functionalities:

- **User registration:** Allows new users to create an account by providing necessary details.
- **User login:** Authenticates existing users based on their credentials.
- **Displaying available flights:** Shows a list of available flights to users.
- **Booking flights:** Enables users to book flights by selecting from available options.
- **Viewing user bookings:** Allows users to view their existing bookings and related details.

3.2 Hardware and Software Requirements

- **Hardware:**
 - A computer with a minimum of 2GB RAM.
 - Hard disk space of at least 500MB.
- **Software:**
 - Python 3.x.
 - MySQL database server.
 - Tkinter library for Python.

3.3 ARCHITECTURE DIAGRAM

1. User Interface Layer:

- **Components:**

- Signup Window
- Login Window
- User Dashboard
- Booking Details Window
- Book Flights Window
- Available Flights Window

- **Description:**

- The User Interface Layer is built using Tkinter, a Python GUI library. It includes various windows that facilitate user interactions with the system. Users can sign up, log in, view their dashboard, see their bookings, book new flights, and view available flights.

2. Application Logic Layer:

- **Components:**

- Signup Logic

- Login Logic
 - Dashboard Logic
 - Booking Management Logic
 - Flight Management Logic
- **Description:**
 - This layer contains the core functionality of the system, implemented as functions in Python. It handles user actions from the UI layer, such as registering a new user, authenticating a user, displaying the user dashboard, managing bookings, and managing flight data. This layer interacts with the Database Layer to perform CRUD (Create, Read, Update, Delete) operations.

3. Database Layer:

- **Components:**
 - MySQL Database
 - **Tables:**
 - User
 - Bookings
 - Flights
- **Description:**
 - The Database Layer is built using MySQL, a relational database management system. It stores all the data related to users, bookings, and flights. This layer ensures data persistence and provides a structured way to store and

retrieve information.

4. Interaction Flow:

- **User Actions:**

- Users interact with the UI Layer by performing actions such as signing up, logging in, and managing bookings.

- **UI to Application Logic:**

- The UI Layer captures user inputs and sends them to the corresponding functions in the Application Logic Layer.

- **Application Logic to Database:**

- The Application Logic Layer interacts with the Database Layer to execute SQL queries based on user actions. It retrieves or modifies data as needed.

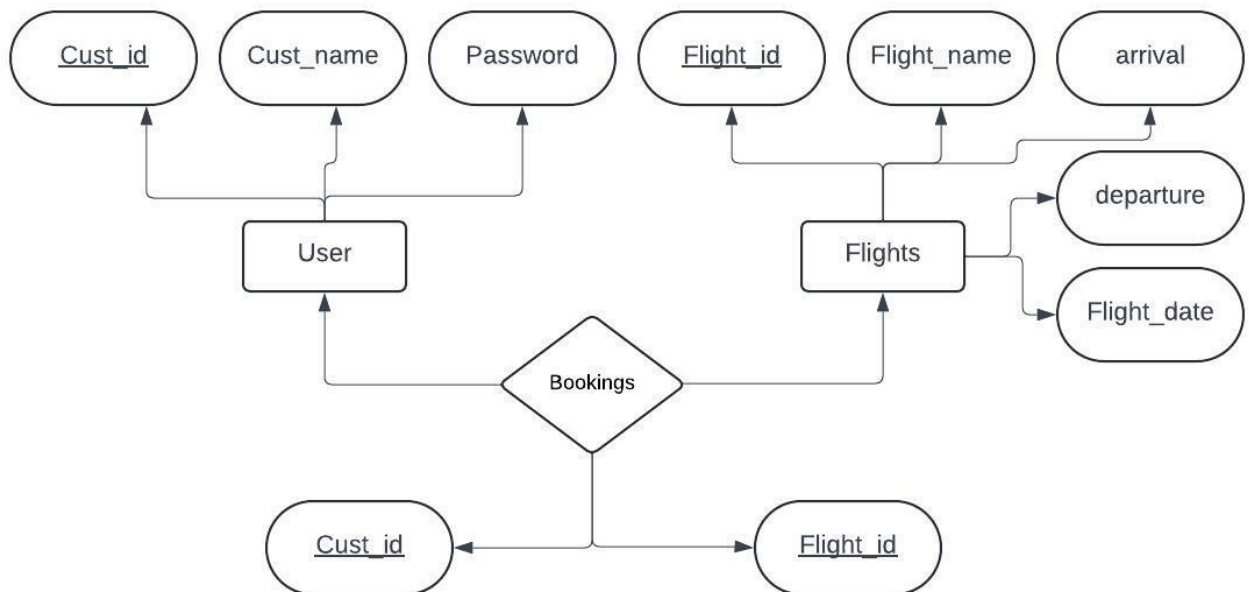
- **Database Response:**

- The Database Layer processes the SQL queries and returns the results to the Application Logic Layer.

- **Application Logic to UI:**

- The Application Logic Layer processes the data and updates the UI Layer to reflect the changes or display the requested information.

3.4 ER DIAGRAM



3.5 NORMALIZATION

1NF (First Normal Form)

1NF ensures that the table is organized in such a way that each column contains atomic (indivisible) values, and each column contains values of a single type.

Users Table:

- Cust_id (Primary Key)
- Cust_name
- Password

Flights Table:

- Flight_id (Primary Key)

- Flight_name
- Arrival
- Departure
- Flight_date

Bookings Table:

- Cust_id (Foreign Key)
- Flight_id (Foreign Key)

2NF (Second Normal Form)

2NF ensures that the table is in 1NF and that all non-key attributes are fully functional dependent on the primary key. Since all tables are already in 1NF and all non-key attributes are fully functional dependent on the primary key, they are also in 2NF.

3NF (Third Normal Form)

3NF ensures that the table is in 2NF and that all the attributes are functionally dependent only on the primary key.

Since the tables do not have any transitive dependencies, they are already in 3NF. However, for clarity, the normalized tables can be presented as follows:

Users Table:

- Cust_id (Primary Key)
- Cust_name
- Password

Flights Table:

- Flight_id (Primary Key)
- Flight_name
- Arrival
- Departure
- Flight_date

Bookings Table:

- Cust_id (Foreign Key referencing Users.Cust_id)
- Flight_id (Foreign Key referencing Flights.Flight_id)

Explanation:

- **First Normal Form (1NF):**
 - Ensure that each table has a primary key and that all fields contain atomic values. In this case, each table (Users, Flights, Bookings) has a primary key and all fields are atomic.
- **Second Normal Form (2NF):**
 - Ensure that the table is in 1NF and that all non-key attributes are fully functional dependent on the primary key. In this case, all tables (Users, Flights, Bookings) have non-key attributes that are fully dependent on the primary key.
- **Third Normal Form (3NF):**
 - Ensure that the table is in 2NF and that all attributes are functionally dependent only on the primary key. In this case, all

tables (Users, Flights, Bookings) have attributes that are functionally dependent only on the primary key.

CHAPTER 4

PROGRAM CODE

```
import tkinter as tk

from tkinter import messagebox, ttk

import mysql.connector

from mysql.connector import Error


# MySQL configuration
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'root',
    'database': 'airline'
}


def get_db_connection():
    try:
        connection = mysql.connector.connect(**db_config)

        if connection.is_connected():
            return connection

    except Error as e:
```

```
    print(f"Error: '{e}'")

    return None


def signup():

    def register_user():

        cust_name = entry_cust_name.get()

        cust_id = entry_cust_id.get()

        password = entry_password.get()


        connection = get_db_connection()

        if connection is None:

            messagebox.showerror("Error", "Failed to connect to the database.")

            return

        cursor = connection.cursor()


        try:

            cursor.execute("INSERT INTO User (Cust_name, Cust_id,
password) VALUES (%s, %s, %s)",

                (cust_name, cust_id, password))

            connection.commit()

            messagebox.showinfo("Success", "Signup successful! Please login.")

            signup_window.destroy()

        except Error as e:
```

```
messagebox.showerror("Error", f"Error: {e}")
```

```
finally:
```

```
    cursor.close()
```

```
    connection.close()
```

```
signup_window = tk.Toplevel(root)
```

```
signup_window.title("Signup")
```

```
tk.Label(signup_window, text="Customer Name").grid(row=0, column=0)
```

```
entry_cust_name = tk.Entry(signup_window)
```

```
entry_cust_name.grid(row=0, column=1)
```

```
tk.Label(signup_window, text="Customer ID").grid(row=1, column=0)
```

```
entry_cust_id = tk.Entry(signup_window)
```

```
entry_cust_id.grid(row=1, column=1)
```

```
tk.Label(signup_window, text="Password").grid(row=2, column=0)
```

```
entry_password = tk.Entry(signup_window, show="*")
```

```
entry_password.grid(row=2, column=1)
```

```
tk.Button(signup_window, text="Signup",  
command=register_user).grid(row=3, column=1)
```

```
def login():  
    def authenticate_user():  
        cust_id = entry_cust_id.get()  
        password = entry_password.get()  
  
        connection = get_db_connection()  
        if connection is None:  
            messagebox.showerror("Error", "Failed to connect to the database.")  
            return  
        cursor = connection.cursor()  
  
        try:  
            cursor.execute("SELECT * FROM User WHERE Cust_id = %s AND  
password = %s", (cust_id, password))  
            user = cursor.fetchone()  
            if user:  
                messagebox.showinfo("Success", "Login successful!")  
                login_window.destroy()  
                open_user_dashboard(cust_id)  
            else:  
                messagebox.showerror("Error", "Invalid credentials. Please try  
again.")  
        except Error as e:
```

```
        messagebox.showerror("Error", f"Error: {e}")

    finally:

        cursor.close()

        connection.close()


login_window = tk.Toplevel(root)
login_window.title("Login")


tk.Label(login_window, text="Customer ID").grid(row=0, column=0)
entry_cust_id = tk.Entry(login_window)
entry_cust_id.grid(row=0, column=1)


tk.Label(login_window, text="Password").grid(row=1, column=0)
entry_password = tk.Entry(login_window, show="*")
entry_password.grid(row=1, column=1)


tk.Button(login_window,
command=authenticate_user).grid(row=2, column=1)
text="Login",


def open_user_dashboard(cust_id):

    dashboard_window = tk.Toplevel(root)

    dashboard_window.title("User Dashboard")
```

```
tk.Button(dashboard_window, text="Your Bookings", command=lambda:
your_bookings(cust_id)).pack(pady=10)
```

```
tk.Button(dashboard_window, text="Book Now", command=lambda:
book_now(cust_id)).pack(pady=10)
```

```
tk.Button(dashboard_window, text="Show Flights",
command=show_flights).pack(pady=10)
```

```
def your_bookings(cust_id):
```

```
    def view_booking_details(cust_id, flight_id):
```

```
        booking_window = tk.Toplevel(root)
```

```
        booking_window.title("Booking Details")
```

```
        tk.Label(booking_window, text="Booking Details", font=("Helvetica",
16, "bold")).pack(pady=10)
```

```
        # Fetch and display booking details
```

```
        connection = get_db_connection()
```

```
        if connection is None:
```

```
            messagebox.showerror("Error", "Failed to connect to the database.")
```

```
            return
```

```
        cursor = connection.cursor()
```

```
        try:
```

```
cursor.execute("SELECT * FROM Bookings WHERE Cust_id = %s
AND flight_id = %s", (cust_id, flight_id))
```

```
booking = cursor.fetchone()
```

```
if booking:
```

```
    tk.Label(booking_window, text=f"Customer ID: {cust_id}").pack()
```

```
    tk.Label(booking_window, text=f"Flight ID: {flight_id}").pack()
```

```
    # Fetch flight details
```

```
    cursor.execute("SELECT * FROM Flights WHERE flight_id =
%s", (flight_id,))
```

```
    flight = cursor.fetchone()
```

```
    if flight:
```

```
        tk.Label(booking_window, text=f"Flight Name:
{flight[0]}").pack()
```

```
        tk.Label(booking_window, text=f"Arrival: {flight[2]}").pack()
```

```
        tk.Label(booking_window, text=f"Departure:
{flight[3]}").pack()
```

```
        tk.Label(booking_window, text=f"Flight Date:
{flight[4]}").pack()
```

```
    else:
```

```
        tk.Label(booking_window, text="Flight details not
found.").pack()
```

```
    else:
```

```
        tk.Label(booking_window, text="Booking details not
found.").pack()
```

```
except Error as e:

    messagebox.showerror("Error", f"Error: {e}")

finally:

    cursor.close()

    connection.close()


bookings_window = tk.Toplevel(root)

bookings_window.title("Your Bookings")


columns = ('Customer ID', 'Flight ID', 'Flight Name', 'Arrival', 'Departure',
'Flight Date')

tree      =      ttk.Treeview(bookings_window,      columns=columns,
show='headings')


for col in columns:

    tree.heading(col, text=col)

    tree.column(col, minwidth=0, width=100)


tree.pack(fill=tk.BOTH, expand=True)


# Fetch and display bookings for the logged-in user
connection = get_db_connection()

if connection is None:
```



```

messagebox.showerror("Error", "Failed to connect to the database.")

return

cursor = connection.cursor()

try:
    cursor.execute("""
        SELECT b.Cust_id, b.flight_id, f.flight_name, f.arrival, f.departure,
f.flight_date
        FROM Bookings b
        JOIN Flights f ON b.flight_id = f.flight_id
        WHERE b.Cust_id = %s
    """, (cust_id,))

    bookings = cursor.fetchall()

    for booking in bookings:
        tree.insert("", tk.END, values=booking)

def on_treeview_select(event):
    selected_item = tree.selection()[0]
    cust_id, flight_id = tree.item(selected_item, 'values')[:2]
    view_booking_details(cust_id, flight_id)

    tree.bind("<Double-1>", on_treeview_select)

except Error as e:

```

```

        messagebox.showerror("Error", f"Error: {e}")

    finally:

        cursor.close()

        connection.close()


def book_now(cust_id):

    def book_flight(flight_id):

        connection = get_db_connection()

        if connection is None:

            messagebox.showerror("Error", "Failed to connect to the database.")

            return

        cursor = connection.cursor()

        try:

            cursor.execute("INSERT INTO Bookings (Cust_id, flight_id)
VALUES (%s, %s)", (cust_id, flight_id))

            connection.commit()

            messagebox.showinfo("Success", f"Flight {flight_id} booked
successfully!")

        except Error as e:

            messagebox.showerror("Error", f"Error: {e}")

    finally:

```

```
cursor.close()

connection.close()


book_window = tk.Toplevel(root)
book_window.title("Book Flights")


columns = ('flight_name', 'flight_id', 'arrival', 'departure', 'flight_date',
'book')

tree = ttk.Treeview(book_window, columns=columns, show='headings')

for col in columns[:-1]:
    tree.heading(col, text=col)
    tree.column(col, minwidth=0, width=100)

tree.heading('book', text='Book')
tree.column('book', minwidth=0, width=100)

tree.pack(fill=tk.BOTH, expand=True)


connection = get_db_connection()

if connection is None:
    messagebox.showerror("Error", "Failed to connect to the database.")

return
```

```

cursor = connection.cursor()

try:
    cursor.execute("SELECT flight_name, flight_id, arrival, departure,
flight_date FROM Flights")
    rows = cursor.fetchall()
    for row in rows:
        flight_name, flight_id, arrival, departure, flight_date = row
        tree.insert("", tk.END, values=row + ("Book",))
except Error as e:
    messagebox.showerror("Error", f"Error: {e}")
finally:
    cursor.close()
    connection.close()

def on_treeview_select(event):
    selected_item = tree.selection()[0]
    flight_id = tree.item(selected_item, 'values')[1]
    book_flight(flight_id)

tree.bind("<ButtonRelease-1>", on_treeview_select)

def show_flights():

```

```
flights_window = tk.Toplevel(root)

flights_window.title("Available Flights")


columns = ('flight_name', 'flight_id', 'arrival', 'departure', 'flight_date')
tree = ttk.Treeview(flights_window, columns=columns, show='headings')


for col in columns:

    tree.heading(col, text=col)

    tree.column(col, minwidth=0, width=100)


tree.pack(fill=tk.BOTH, expand=True)


connection = get_db_connection()

if connection is None:

    messagebox.showerror("Error", "Failed to connect to the database.")

    return


cursor = connection.cursor()

try:

    cursor.execute("SELECT flight_name, flight_id, arrival, departure,
flight_date FROM Flights")

    rows = cursor.fetchall()

    for row in rows:
```

```
        tree.insert("", tk.END, values=row)

except Error as e:

    messagebox.showerror("Error", f"Error: {e}")

finally:

    cursor.close()

    connection.close()


# Main application window

root = tk.Tk()

root.title("Airline System")


tk.Button(root, text="Signup", command=signup).pack(pady=10)

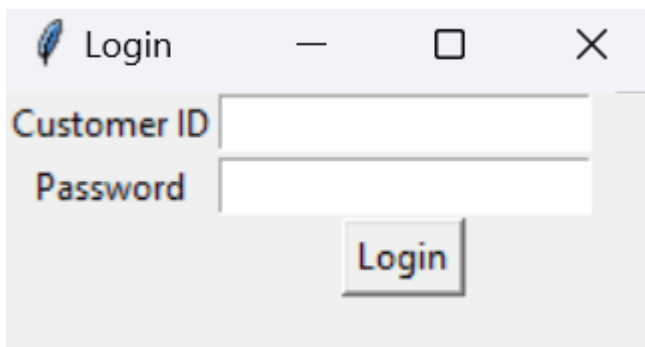
tk.Button(root, text="Login", command=login).pack(pady=10)


root.mainloop()
```

CHAPTER 5

RESULTS AND DISCUSSION

LOGIN MODULE

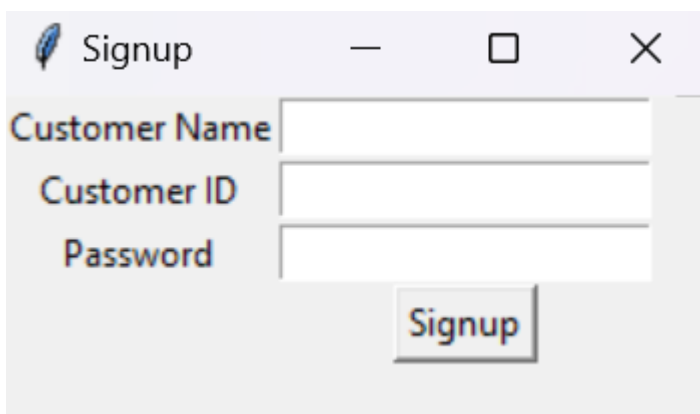


A screenshot of a web application window titled "Login". The window has a light purple header bar with a feather icon, the title "Login", and standard window controls (minimize, maximize, close). Below the header, there are two input fields: "Customer ID" and "Password". The "Password" field is masked with dots. A "Login" button is positioned below the input fields.

Field	Value
Customer ID	
Password	

Login

SIGN UP MODULE

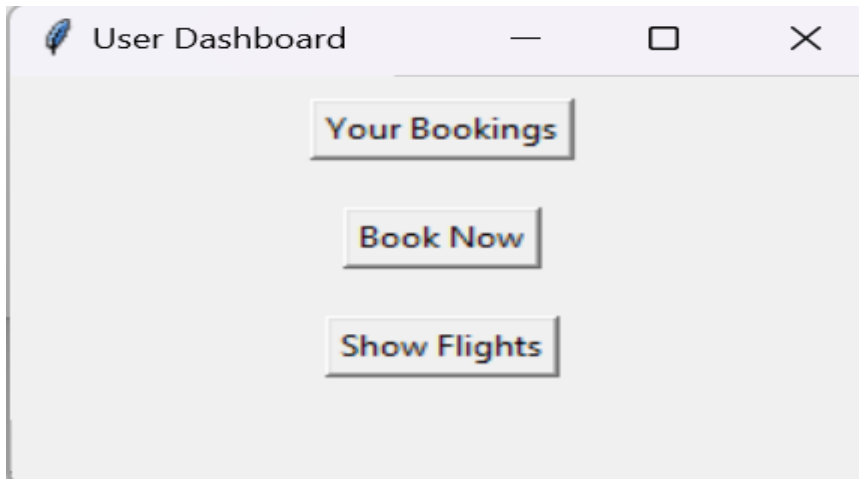


A screenshot of a web application window titled "Signup". The window has a light purple header bar with a feather icon, the title "Signup", and standard window controls (minimize, maximize, close). Below the header, there are three input fields: "Customer Name", "Customer ID", and "Password". The "Password" field is masked with dots. A "Signup" button is positioned below the input fields.

Field	Value
Customer Name	
Customer ID	
Password	

Signup

USER DASHBOARD MODULE



YOUR BOOKINGS MODULE

Your Bookings					
Customer ID	Flight ID	Flight Name	Arrival	Departure	Flight Date
323	101	Flight A	8:00:00	10:00:00	2024-06-01
323	104	Flight D	6:00:00	8:00:00	2024-06-04

BOOK NOW MODULE

Book Flights					
flight_name	flight_id	arrival	departure	flight_date	Book
Flight A	101	8:00:00	10:00:00	2024-06-01	Book
Flight B	102	9:00:00	11:00:00	2024-06-02	Book
Flight C	103	7:00:00	9:00:00	2024-06-03	Book
Flight D	104	6:00:00	8:00:00	2024-06-04	Book
Flight E	105	5:00:00	7:00:00	2024-06-05	Book
Flight F	106	4:00:00	6:00:00	2024-06-06	Book
Flight G	107	3:00:00	5:00:00	2024-06-07	Book
Flight H	108	2:00:00	4:00:00	2024-06-08	Book
Flight I	109	1:00:00	3:00:00	2024-06-09	Book
Flight J	110	0:00:00	2:00:00	2024-06-10	Book

SHOWFLIGHTS-MODULE

Available Flights				
flight_name	flight_id	arrival	departure	flight_date
Flight A	101	8:00:00	10:00:00	2024-06-01
Flight B	102	9:00:00	11:00:00	2024-06-02
Flight C	103	7:00:00	9:00:00	2024-06-03
Flight D	104	6:00:00	8:00:00	2024-06-04
Flight E	105	5:00:00	7:00:00	2024-06-05
Flight F	106	4:00:00	6:00:00	2024-06-06
Flight G	107	3:00:00	5:00:00	2024-06-07
Flight H	108	2:00:00	4:00:00	2024-06-08
Flight I	109	1:00:00	3:00:00	2024-06-09
Flight J	110	0:00:00	2:00:00	2024-06-10

Cust_name	Cust_id	password
sibi sundar	277	chan
vipin	323	rec

flight_name	flight_id	arrival	departure	flight_date
Flight A	101	08:00:00	10:00:00	2024-06-01
Flight B	102	09:00:00	11:00:00	2024-06-02
Flight C	103	07:00:00	09:00:00	2024-06-03
Flight D	104	06:00:00	08:00:00	2024-06-04
Flight E	105	05:00:00	07:00:00	2024-06-05
Flight F	106	04:00:00	06:00:00	2024-06-06
Flight G	107	03:00:00	05:00:00	2024-06-07
Flight H	108	02:00:00	04:00:00	2024-06-08
Flight I	109	01:00:00	03:00:00	2024-06-09
Flight J	110	00:00:00	02:00:00	2024-06-10

Cust_id	flight_id
323	101
323	104

CHAPTER 6

CONCLUSION

The Airline Management System presented in this project offers an effective solution for managing airline bookings and operations. Through the integration of Python, Tkinter, and MySQL, the system provides users with a user-friendly interface for signing up, logging in, viewing available flights, and making bookings.

The system's modular design ensures clear separation of concerns, with distinct modules for user authentication, flight management, and booking management. This modular approach enhances the system's maintainability and scalability, allowing for easy expansion and addition of new features in the future.

By normalizing the database schema up to the third normal form (3NF), data integrity is ensured, and redundancy is minimized. The normalized schema facilitates efficient data storage and retrieval, contributing to the system's overall performance and reliability.

Overall, the Airline Management System streamlines the airline booking process, providing users with a seamless and hassle-free experience. Through its intuitive interface and robust functionality, the system enhances customer satisfaction and contributes to the efficient management of airline operations. With further development and refinement, the system has the potential to become a valuable asset for airlines and passengers alike.

CHAPTER 7

REFERENCES

- Tkinter Documentation. (n.d.). Retrieved from <https://docs.python.org/3/library/tkinter.html>
- MySQL Connector/Python Developer Guide. (n.d.). Retrieved from <https://dev.mysql.com/doc/connector-python/en/>
- Python Software Foundation. (n.d.). Python 3.9.6 documentation. Retrieved from <https://docs.python.org/3/>

