L. N. Mishra Institute of Economic Development & Social Change, Patna

AFFILIATED TO ARYABHATTA KNOWLEDGE UNIVERSITY, PATNA

AND APPROVED BY AICTE, GOVT. OF INDIA

Railway Reservation System

Project Report

On

In partial fulfilment for the award of the degree of

Bachelor of Computer Application

Session:2022-2025

UNDER THE GUIDANCE

MR. ALOK KUMAR

Assistant Professor,

LNMI, PATNA

SUBMITTED BY

Student Name	Roll no	Regn.No
Piyush Kumar	226078	22303333027
Shivam Anand	226091	22303333050

STUDENT DECLARATION

I the undersigned, a BCA Student of LNMI, Patna bearing Registration Number: 22303333027 do solemnly declare that the project work titled “Railway Reservation System” is based on my own work, carried out under supervision of guide Mr. Alok Kumar (Assistant Professor, LNMI, Patna).

I assert that the statements made are outcome of my work. I ensure that the project work is original, not a copy paste job. I affirm that this project work has been not submitted whether to this Institute or any other University for fulfilment of the requirement of any course of study.

Signature of the Student

ACKNOWLEDGEMENT

I would like to thank to Organization for giving me an opportunity for **Internship** which gives me a pleasurable experience.

I would like to thanks all Faculties of BCA department for their regular guidance and supervision during the Project and throughout numerous consultations.

I am also grateful to Mr. Alok Kumar (Assistant Professor, LNMI, Patna) and other Assistant Professor members for their consistent guidance and support throughout the project tenure. I than all the people for their help directly and indirectly to complete my Project.

Piyush Kumar

Reg_No-22303333027

Roll_no-226078

BCA 6TH SEM

Table of Content

1.Introduction

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS purpose, scope, definitions, acronyms, abbreviations, references and overview of SRS.A **Software Requirements Specification** (**SRS**) - a requirements specification for a software system - is a complete description of the behaviour of a system to be developed. It includes a set of use cases that describe all the

interactions the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). The aim of this document is to gather and analyse and give an in-depth insight of the complete Marvel Electronics and Home Entertainment software system by defining the problem statement in detail. This is a documentation of the project **Railways Reservation System** done sincerely and satisfactorily by my group members. A Software has to be developed for automating the manual Railway Reservation System.

- RESERVE SEATS – Reservation form has to be filled by passenger. If seats are available entries like train name, number, destination are made.

- CANCEL RESERVATION- The clerk deletes the entry in the System and changes in the Reservation Status.

- VIEW RESERVATION STATUS-The user needs to enter the PIN number printed on ticket.

1.1 Objective:

The purpose of this source is to describe the railway reservation system which provides the train timing details, reservation, billing and cancellation on various types of reservation namely, • Confirm Reservation for confirm Seat.

- Reservation against Cancellation.
- Waiting list Reservation.
- Online Reservation.
- Tatkal Reservation.

The origin of most software systems is in the need of a client, who either wants to automate the existing manual system or desires a new software system. The software system is itself created by the developer. Finally, the end user will use the completed system. Thus, there are three major parties interested in a new system: the client, the user, and the developer. Somehow the requirements for the system that will satisfy the needs of the clients and the concerns of the users have to be communicated to the developer. The problem is that the client doesn't usually design the software or the software development process and the developer does not understand the client's problem and the application area. This causes a communication gap between the parties involved in the development of the project.

The basic purpose of Software Requirement Specification (SRS) is to bridge this communication gap. SRS is the medium through which the client's and the user's needs are accurately specified; indeed SRS forms the basis of software development.

Another important purpose of developing an SRS is helping the clients understanding their own needs. An SRS establishes the basis for agreement between the client and the supplier on what the software product will do.

An SRS provides a reference for validation of the final product. A high-quality SRS is a prerequisite to high quality software and it also reduces the development cost.

A few factors that direct us to develop a new system are given below -:

1. Faster System
2. Accuracy
3. Reliability
4. Informative
5. Reservations and cancellations from anywhere to any place

6. Informative
7. Reservations and cancellations from anywhere to any place

1.2 Scop:

“Railways Reservation System” is an attempt to simulate the basic concepts of an online Reservation system. The system enables to perform the following functions:

- REGISTER\LOGIN
- SEARCH FOR TRAIN
- BOOKING OF A SELECTED TRAIN
- PAYMENT
- CANCELLATION

1.3 Overview:

The remaining sections of this document provide a general description, including characteristics of the users of this project, the product's hardware, and the functional and data requirements of the product. General description of the project is discussed in section 2 of this document. Section 3 gives the functional requirements, data requirements and constraints and assumptions made while designing the E-Store. It also gives the user viewpoint of product. Section 3 also gives the specific requirements of the product. Section functional requirements. Section 4 is for supporting information. 3 also discusses the external interface requirements and gives detailed description of

2.Overall Description

This document contains the problem statement that the current system is facing which is hampering the growth opportunities of the company. It further contains a list of the stakeholders and users of the

proposed solution. It also illustrates the needs and wants of the stakeholders that were identified in the brainstorming exercise as part of the requirements workshop. It further lists and briefly describes the major features and a brief description of each of the proposed system.

2.1 Product Perspective:

Before the automation, the system suffered from the following **DRAWBACKS:**

- The existing system is highly manual involving a lot of paper work and calculation and therefore may be erroneous. This has led to inconsistency and inaccuracy in the maintenance of data.
- The data, which is stored on the paper only, may be lost, stolen or destroyed due to natural calamity like fire and water.
- The existing system is sluggish and consumes a lot of time causing inconvenience to customers and the airlines staff.
- Due to manual nature, it is difficult to update, delete, add or view the data.
- Since the number of passengers have drastically increased therefore maintaining and retrieving detailed record of passenger is extremely difficult.
- Railways has many offices around the world, an absence of a link between these offices lead to lack of coordination and communication.

Hence the railways reservation system is proposed with the following

- The computerization of the reservation system will reduce a lot of paperwork and hence the load on the airline administrative staff.
- The machine performs all calculations. Hence chances of error are nil.
- The passenger, reservation, cancellation list can easily be retrieved and any required addition, deletion or updation can be performed.
- The system provides for user-ID validation, hence unauthorized access is prevented.

2.2 Project Functions:

Booking agents with varying levels of familiarity with computers will mostly use this system.
With this in mind, an important feature of this software is that it be relatively simple to use. The scope of this project encompasses: -

¨ **Search:** This function allows the booking agent to search for train that are available between the two travel cities, namely the "Departure city" and "Arrival city" as desired by the traveller. The system initially prompts the agent for the departure and arrival city, the date of departure, preferred time slot and the number of passengers. It then displays a list of train available with different airlines between the designated cities on the specified date and time.

¨ **Selection**: This function allows a particular train to be selected from the displayed list. All the details of the train are shown: -

1. train Number
2. Date, time and place of departure
3. Date, time and place of arrival
4. TRAIN Duration
5. Fare per head
6. Number of stoppages – 0, 1, 2…

¨ **Review**: If the seats are available, then the software prompts for the booking of train. The train information is shown. The total fare including taxes is shown and flight details are reviewed.

¨ **Passenger Information**: It asks for the details of all the passengers supposed to travel including name, address, telephone number and e-mail id.

¨ **Payment**: It asks the agent to enter the various credit card details of the person making the reservation.

1. Credit card type
2. Credit card number
3. CVC number of the card
4. Expiration date of the card
5. The name on the card
6. CVV number

¨ **Cancellation:** The system also allows the passenger to cancel an existing reservation. This function registers the information regarding a passenger who has requested for a cancellation of his/her ticket. It includes entries pertaining to the train No., Confirmation No., Name, Date of Journey, Fare deducted.

2.3 User Characteristics:

- **EDUCATIONAL LEVEL: -**At least user of the system should be comfortable with English language.

- **TECHNICAL EXPERTISE: -** User should be comfortable using general purpose applications on the computer system.

2.4 Constrains:

Software constraints:

-

2.5 Assumptions and Dependencies:

- Booking Agents will be having a valid user name and password to access the software
- The software needs booking agent to have complete knowledge of railways reservation system.
- Software is dependent on access to internet.

3.1 Function Requirements

3.1.1 performance requirements:

- **User Satisfaction: -** The system is such that it stands up to the user expectations.
- **Response Time: -**The response of all the operation is good. This has been made possible by careful programming.
- **Error Handling: -** Response to user errors and undesired situations has been taken care of to ensure that the system operates without halting.
- **Safety and Robustness: -** The system is able to avoid or tackle disastrous action. In other words, it should be foul proof. The system safeguards against undesired events, without human intervention.
- **Portable: -** The software should not be architecture specific. It should be easily transferable to other platforms if needed.
- **User friendliness: -** The system is easy to learn and understand. A native user can also use the system effectively, without any difficulties.

3.1.2 Design constraints:

There are a number of factors in the client's environment that may restrict the choices of a designer. Such factors include standards that must be followed, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS (Software Requirements Analysis and Specification) should identify and specify all such constraints.

Standard Compliance: - This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting properties.

Hardware Limitations: - The software may have to operate on some existing or predetermined hardware, thus imposing restrictions on the design. Hardware limitations can include the types of machines to be used, operating system available on the system, languages supported and limits on primary and secondary storage.

Reliability and Fault Tolerance: - Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive. Requirements about system behaviour in the face of certain kinds of faults are specified. Recovery requirements are often an integral part here, detailing what the system should do I some failure occurs to ensure certain properties. Reliability requirements are very important for critical applications.

Security: - Security requirements are particularly significant in defence systems and database systems. They place restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people, require the use of passwords and cryptography techniques and maintain a log of activities in the system.

3.1.3 Hardware requirements:

For the hardware requirements the SRS specifies the logical characteristics of each interface b/w the software product and the hardware components. It specifies the hardware requirements like memory restrictions, cache size, the processor, RAM size etc... those are required for the software to run.

Minimum Hardware Requirements

Processor Pentium III

Hard disk drive 40 GB

RAM 128 MB

Cache 512 kb

Preferred Hardware Requirements

Processor Pentium IV

Hard disk drive 80 GB

RAM 256 MB

Cache 512 kb

3.1.4 Software requirements:

- Any window-based operating system with DOS support are primary requirements for software development. Windows XP, FrontPage and dumps are required. The systems must be connected via LAN and connection to internet is mandatory.

3.1.5 other requirements:

Software should satisfy following requirements as well: -

- SECURITY
- PORTABILITY
- CORRECTNESS
- EFFICIENCY • FLEXIBILTY
- TESTABILTY
- REUSABILTY

3.2 Non-Function Requirements

3.2.1 Security:

The system uses SSL (secured socket layer) in all transactions that include any confidential customer information. The system must automatically log out all customers after a period of inactivity. The system should not leave any cookies on the customer's computer containing the user's password. The system's back-end servers shall only be accessible to authenticated management.

3.2.2 Reliability:

The reliability of the overall project depends on the reliability of the separate components. The main pillar of reliability of the system is the backup of the database which is continuously maintained and updated to reflect the most recent changes. Also, the system will be functioning inside a container. Thus, the overall stability of the system depends on the stability of container and its underlying operating system.

3.2.3 Availability:

The system should be available at all times, meaning the user can access it using a web browser, only restricted by the down time of the server on which the system runs. A customer friendly system which is in excess of people around the world should work 24 hours. In case of a of a hardware failure or database corruption, a replacement page will be shown. Also, in case of a hardware failure or database corruption, backups of the database should be retrieved from the server and saved by the Organizer. Then the service will be restarted. It means 24 x 7 availability.

3.2.4 Maintainability:

A commercial database is used for maintaining the database and the application server takes care of the site. In case of a failure, a re-initialization of the project will be done. Also the software design is being done with modularity in mind so that maintainability can be done efficiently.

3.2.5 Supportability:

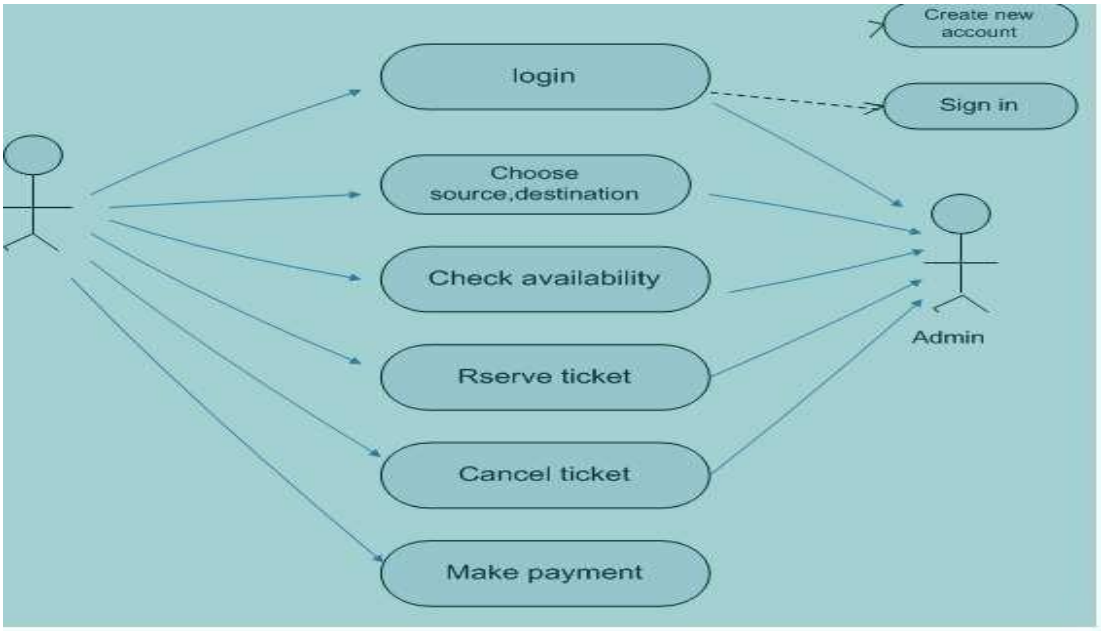
The code and supporting modules of the system will be well documented and easy to understand. Online User Documentation and Help System Requirements.

A **use case diagram** in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Interaction among actors is not shown on the use case diagram. If this interaction is essential to a coherent description of the desired behaviour, perhaps the system or use case boundaries should be re-examined. Alternatively, interaction among actors can be part of the assumptions used in the use case.

- Use cases
  A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.
- Actors
  An actor is a person, organization, or external system that plays a role in one or more interactions with the system.
- System boundary boxes(optional)

A rectangle is drawn around the use cases, called the system boundary box, to indicate its scope of system. Anything within the box represents functionality that is in scope and anything outside the box is not.

4.1 Use-case Diagram



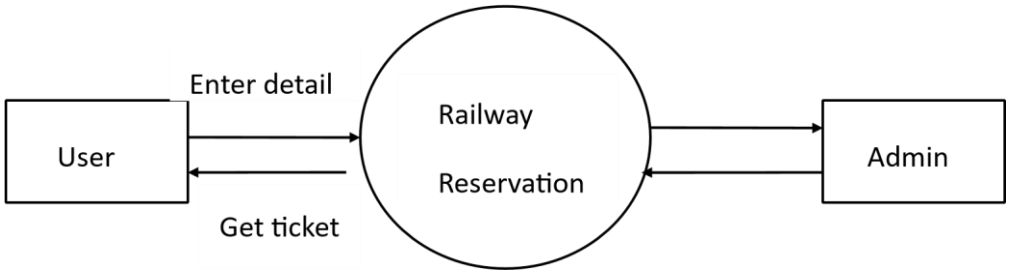
4.4 Data Flow Diagram

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process. A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).

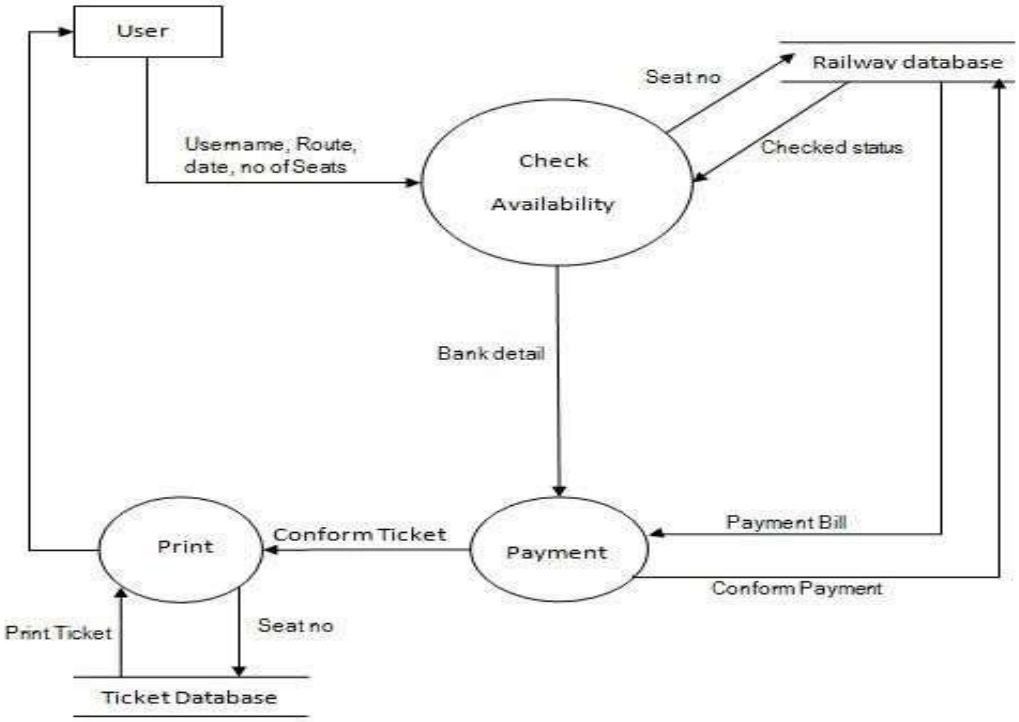
It is common practice to draw a context-level data flow diagram first, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the 'Level 0 DFD') the system's interactions with the outside world are modelled purely in terms of data flows across the *system boundary*. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modelled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

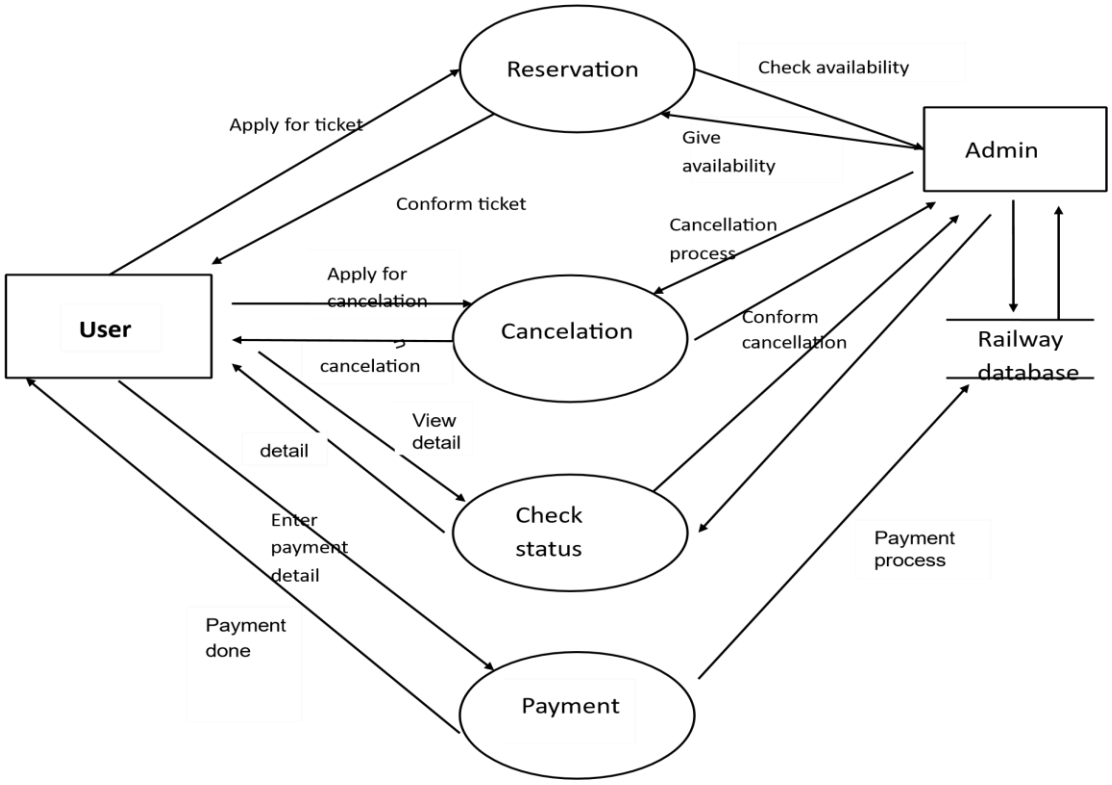
Level 0:



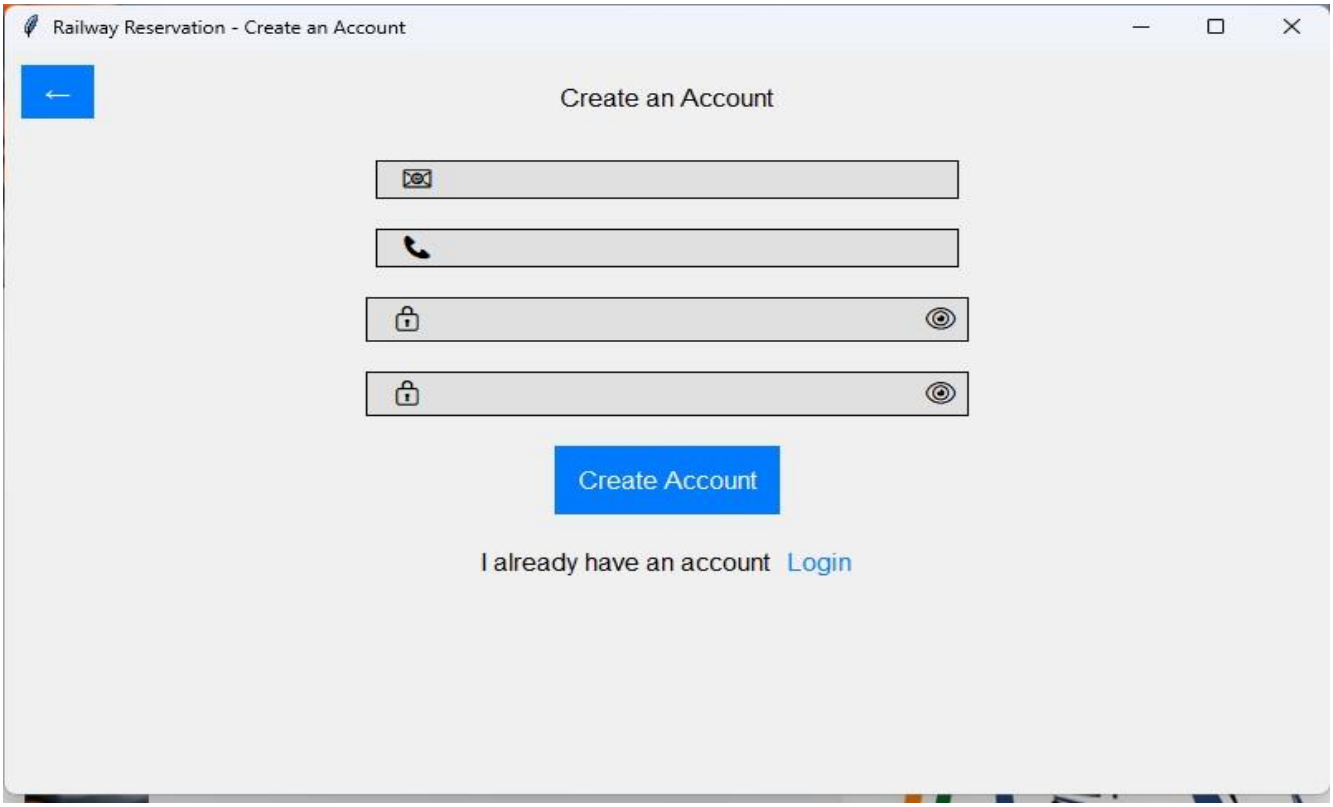
Level 1:

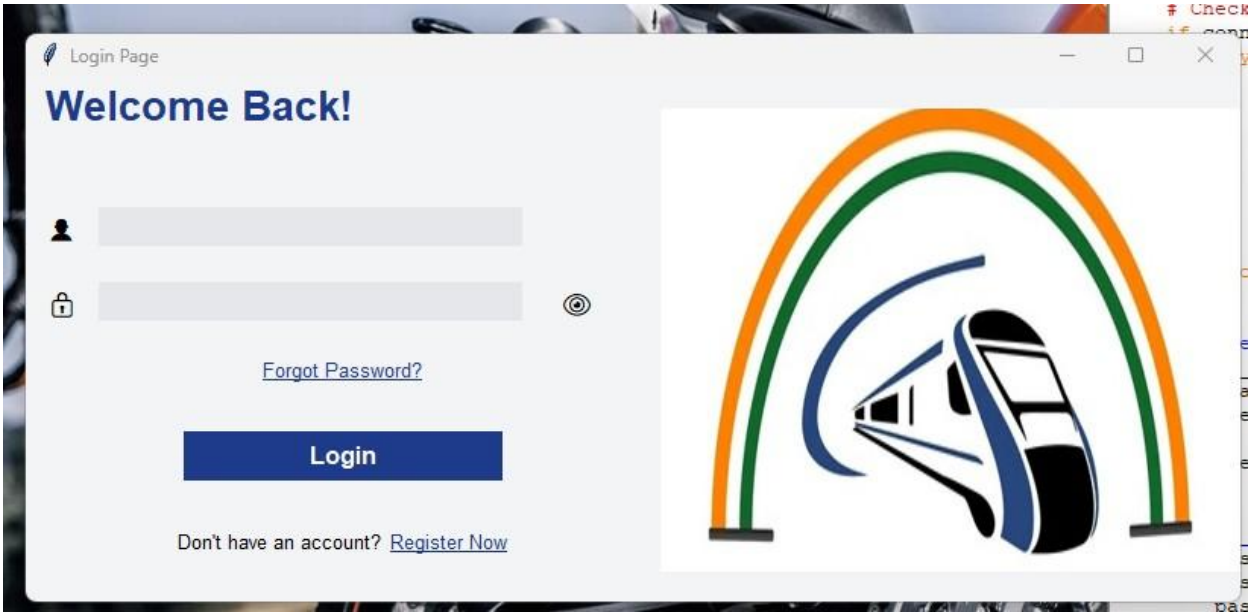


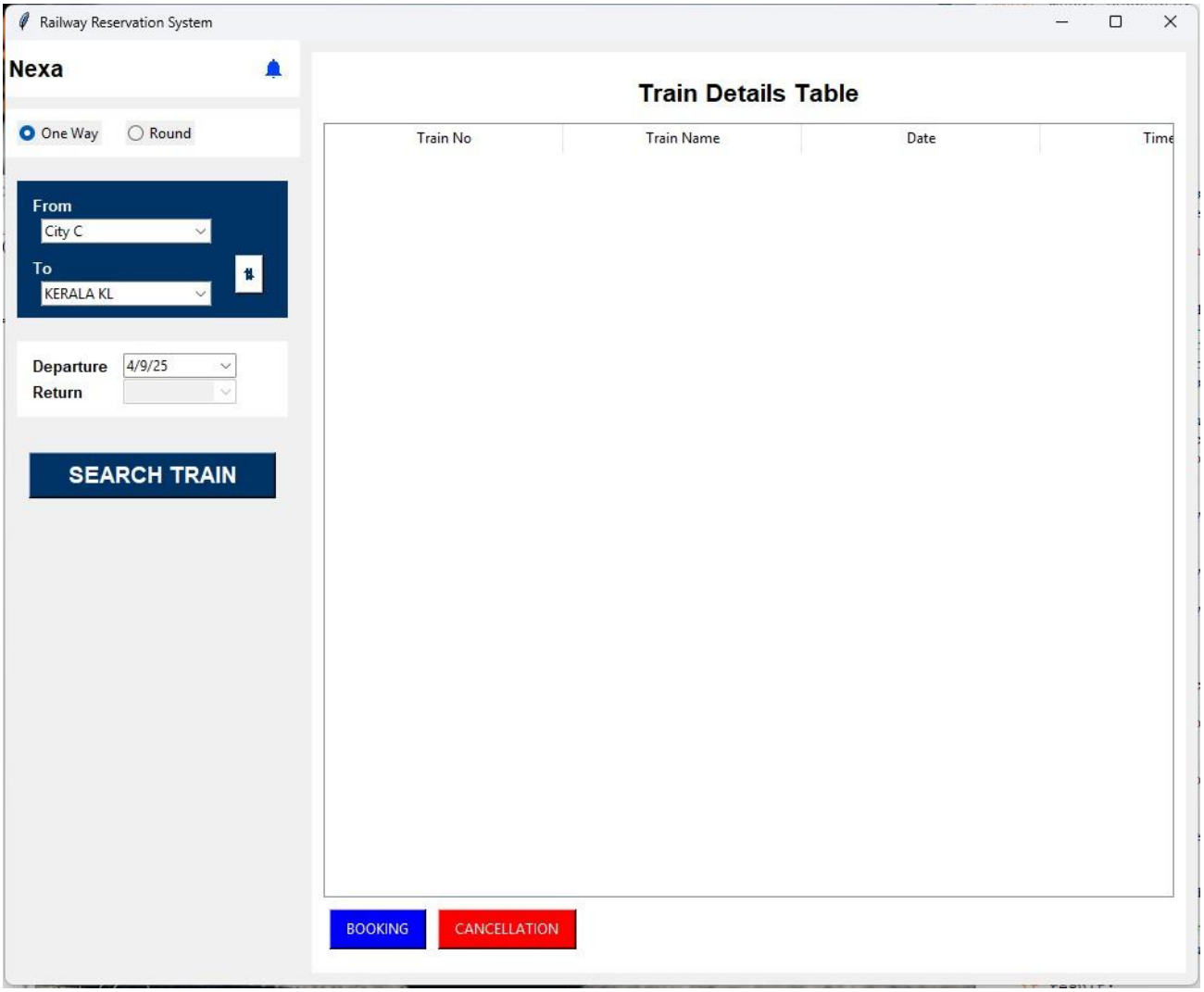
Level 2:

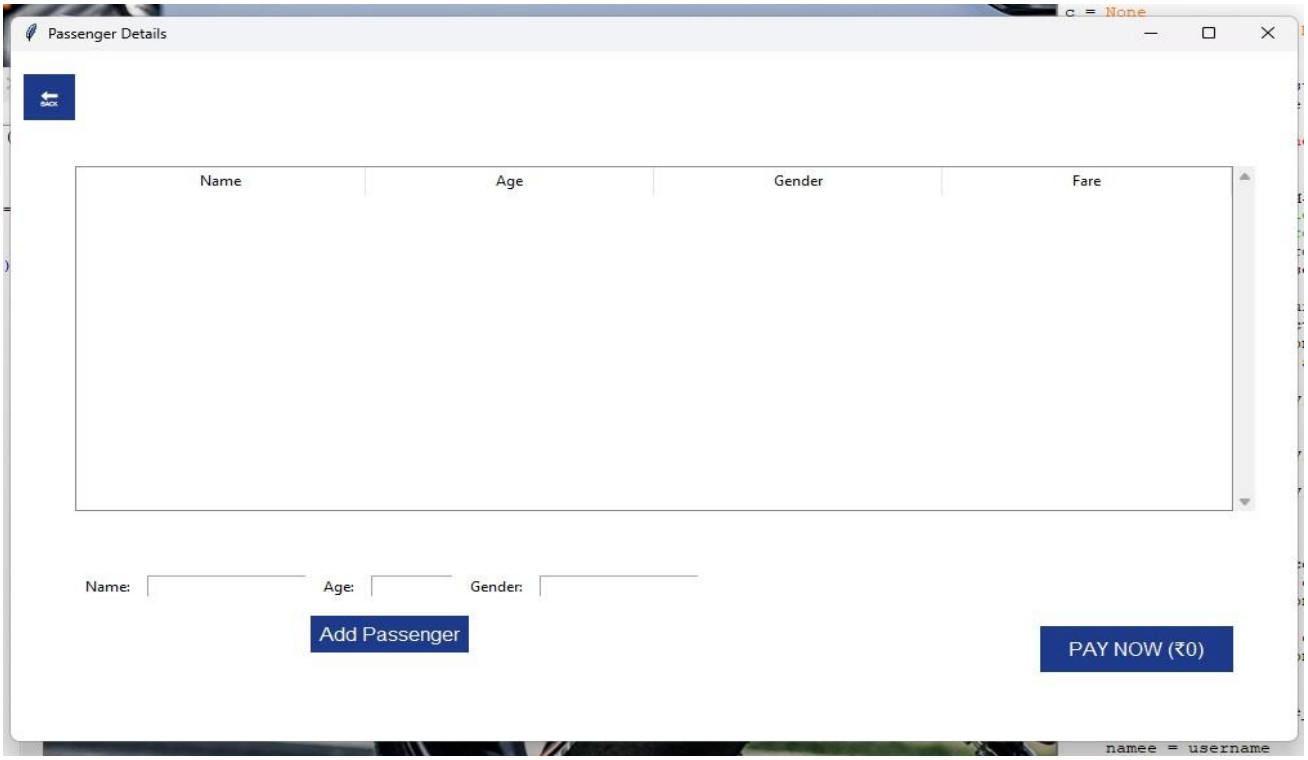


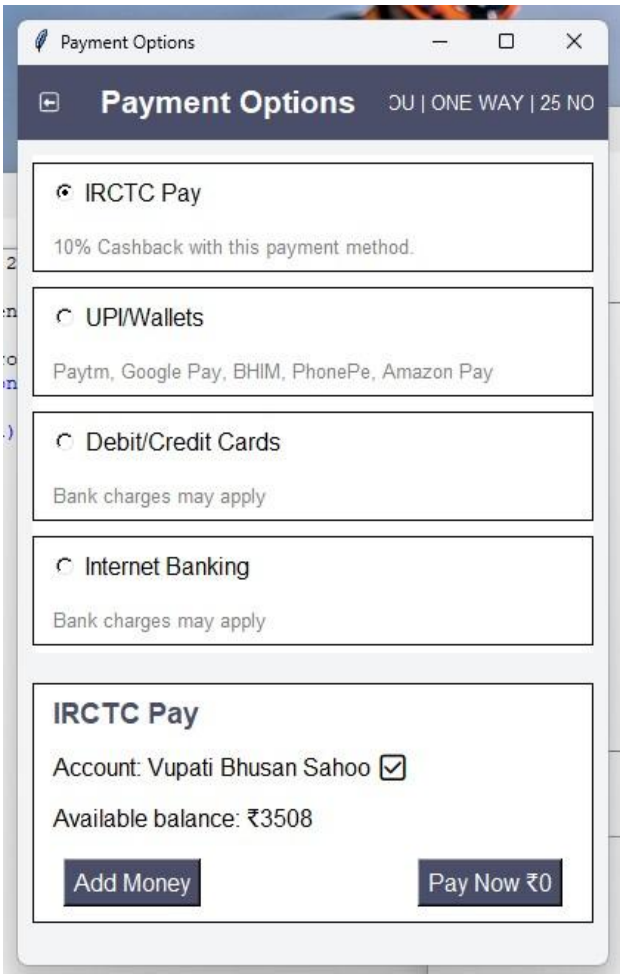
5.1 Screen Short


Railway Reservation - Create an Account
Create an Account
Create Account
I already have an account Login


Login Page
Welcome Back!
Forgot Password?
Login
Don't have an account? Register Now


Railway Reservation System
Nexa
One Way
Round
From
City C
To
KERALA KL
Departure
4/9/25
Return
SEARCH TRAIN
Train Details Table
Train No
Train Name
Date
Time
BOOKING
CANCELLATION


Passenger Details
Name
Age
Gender
Fare
Name:
Age:
Gender:
Add Passenger
PAY NOW (₹0)



6. Code

```
import tkinter as tk

from tkinter import ttk, messagebox

from tkinter import font as tkfont
```

```
from tkcalendar import DateEntry
from PIL import Image, ImageTk
import mysql.connector
import datetime
import time

# Global variables for connection and cursor
conn = None
c = None
last_connection_time = None

def get_db_connection():
global conn, c, last_connection_time
current_time = time.time()

# Check if the connection is None or if 20 minutes have passed
if conn is None or (last_connection_time is not None and (current_time - last_connection_time) > 1200):
try:
conn = mysql.connector.connect(
host="localhost",
user="root",
password="tiger",
database="railway_db"
)
c = conn.cursor()
last_connection_time = current_time
```

```
except mysql.connector.Error as e:

messagebox.showerror("Database Error", f"Error connecting to database: {e}")


def execute_query(query, params=None):

get_db_connection()  # Ensure the connection is active

if params:

c.execute(query, params)

else:

c.execute(query)

return c.fetchall()


def toggle_password():

if password_entry.cget('show') == '':

password_entry.config(show='*')

password_eye.config(text='◎')

else:

password_entry.config(show='')

password_eye.config(text='👁')


def login():

username = username_entry.get()

global namee

namee = username

password = password_entry.get()


query = "SELECT * FROM users WHERE username = %s AND password = %s"
```

```
result = execute_query(query, (username, password))


if result:
messagebox.showinfo("Login Success", "Welcome to the Railway Reservation System!")
log.destroy()
global x2
x2 = Main_window(username)  # Pass the username here
else:
message_label.config(text="Invalid credentials, please try again.", fg="red")


def forgot_password():
messagebox.showinfo("Forgot Password", "Password reset instructions have been sent to your email.")


def go_back():
root.destroy()


def open_forgot_password_window():
win1 = tk.Toplevel(root)
win1.title("Forgot Password")
win1.geometry("800x500")
win1.configure(bg="#f0f0f0")
screen_width = win1.winfo_screenwidth()
screen_height = win1.winfo_screenheight()
width = 800
height = 500
x = (screen_width / 2) - (width / 2)
```

```
y = (screen_height / 2) - (height / 2)

win1.geometry('%dx%d+%d+%d' % (width, height, x, y))


heading_label = tk.Label(win1, text="Reset Your Password", font=("Helvetica", 16, "bold"), bg="#f0f0f0")

heading_label.pack(pady=20)


username_label = tk.Label(win1, text="Username", bg="#f0f0f0")

username_label.pack(pady=5)

username_entry = tk.Entry(win1, width=30)

username_entry.pack(pady=5)

username_entry.insert(0, "Enter your username")


reset_password_button = tk.Button(win1, text="Reset Password", bg="#4CAF50", fg="white", width=15,
command=forgot_password)

reset_password_button.pack(pady=20)


def on_enter(e):

reset_password_button['background'] = '#45a049'


def on_leave(e):

reset_password_button['background'] = '#4CAF50'


reset_password_button.bind("<Enter>", on_enter)

reset_password_button.bind("<Leave>", on_leave)


back_button = tk.Button(win1, text="Back", bg="#f0f0f0", command=go_back)
```

```
back_button.pack(pady=5)

message_label = tk.Label(win1, text="", bg="#f0f0f0")
message_label.pack(pady=5)

def create_main_window():
def toggle_password(password_entry):
if password_entry.cget("show") == "*":
password_entry.config(show="")
else:
password_entry.config(show="*")

def toggle_confirm_password(confirm_password_entry):
if confirm_password_entry.cget("show") == "*":
confirm_password_entry.config(show="")
else:
confirm_password_entry.config(show="*")

def create_account(username_entry, phone_entry, password_entry, confirm_password_entry):
username = username_entry.get()
phone = phone_entry.get()
password = password_entry.get()
confirm_password = confirm_password_entry.get()

if password != confirm_password:
messagebox.showerror("Error", "Passwords do not match!")
```

```
else:
query = "INSERT INTO users (username, phone, password) VALUES (%s, %s, %s)"
execute_query(query, (username, phone, password))
conn.commit()
messagebox.showinfo("Success", "Account created successfully!")
root.destroy()

def on_back():
root.destroy()

def on_login(event):
root.destroy()

# Initialize the main application window
root = tk.Tk()
root.title("Railway Reservation - Create an Account")
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
width = 800
height = 500
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
root.geometry('%dx%d+%d+%d' % (width, height, x, y))
root.configure(bg="#f0f0f0")

header_font = tkfont.Font(family="Helvetica", size=12, weight="bold")
```

```
label_font = tkfont.Font(family="Helvetica", size=10)

button_font = tkfont.Font(family="Helvetica", size=12, weight="bold")


back_button = tk.Button(root, text="←", command=on_back, font=button_font, bg="#007bff",
fg="white", bd=0, padx=10, pady=5)

back_button.place(x=10, y=10)


header = tk.Label(root, text="Create an Account", font=header_font, bg="#f0f0f0")

header.pack(pady=20)


username_frame = tk.Frame(root, bg="#e0e0e0", bd=1, relief="solid")

username_frame.pack(pady=10, padx=20)

username_icon = tk.Label(username_frame, text="📧", font=label_font, bg="#e0e0e0")

username_icon.pack(side="left", padx=10)

username_entry = tk.Entry(username_frame, font=label_font, bd=0, bg="#e0e0e0")

username_entry.pack(side="left", padx=60)


phone_frame = tk.Frame(root, bg="#e0e0e0", bd=1, relief="solid")

phone_frame.pack(pady=10, padx=20)

phone_icon = tk.Label(phone_frame, text="📞", font=label_font, bg="#e0e0e0")

phone_icon.pack(side="left", padx=10)

phone_entry = tk.Entry(phone_frame, font=label_font, bd=0, bg="#e0e0e0")

phone_entry.pack(side="left", padx=60)


password_frame = tk.Frame(root, bg="#e0e0e0", bd=1, relief="solid")

password_frame.pack(pady=10, padx=50)
```

```
password_icon = tk.Label(password_frame, text=" 🔒 ", font=label_font, bg="#e0e0e0")

password_icon.pack(side="left", padx=10)

password_entry = tk.Entry(password_frame, font=label_font, bd=0, bg="#e0e0e0", show="*")

password_entry.pack(side="left", padx=50)

password_toggle = tk.Button(password_frame, text="👁", font=label_font, bg="#e0e0e0", bd=0,
command=lambda: toggle_password(password_entry))

password_toggle.pack(side="right")


confirm_password_frame = tk.Frame(root, bg="#e0e0e0", bd=1, relief="solid")

confirm_password_frame.pack(pady=10, padx=50)

confirm_password_icon = tk.Label(confirm_password_frame, text=" 🔒 ", font=label_font,
bg="#e0e0e0")

confirm_password_icon.pack(side="left", padx=10)

confirm_password_entry = tk.Entry(confirm_password_frame, font=label_font, bd=0, bg="#e0e0e0",
show="*")

confirm_password_entry.pack(side="left", padx=50)

confirm_password_toggle = tk.Button(confirm_password_frame, text="👁", font=label_font,
bg="#e0e0e0", bd=0, command=lambda: toggle_confirm_password(confirm_password_entry))

confirm_password_toggle.pack(side="right")


create_account_button = tk.Button(root, text="Create Account", font=button_font, bg="#007bff",
fg="white", bd=0, padx=10, pady=10, command=lambda: create_account(username_entry, phone_entry,
password_entry, confirm_password_entry))

create_account_button.pack(pady=10, padx=20)


login_frame = tk.Frame(root, bg="#f0f0f0")

login_frame.pack(pady=10)

login_label = tk.Label(login_frame, text="I already have an account ", font=label_font, bg="#f0f0f0")
```

```
login_label.pack(side="left")

login_link = tk.Label(login_frame, text="Login", font=label_font, fg="#007bff", bg="#f0f0f0",
cursor="hand2")

login_link.pack(side="left")

login_link.bind("<Button-1>", on_login)


return root


def Main_window(username):

global trip_type, from_city, to_city, return_date, departure_date, tree

main = tk.Tk()

main.title("Railway Reservation System")

screen_width = main.winfo_screenwidth()

screen_height = main.winfo_screenheight()

width = 1000

height = 800

x = (screen_width / 2) - (width / 2)

y = (screen_height / 2) - (height / 2)

main.geometry('%dx%d+%d+%d' % (width, height, x, y))

main.configure(bg="#f0f0f0")


name = username

# Left Frame

left_frame = tk.Frame(main, bg="#f0f0f0", width=400)

left_frame.pack(side="left", fill="y")
```

```
# Top Section

top_frame = tk.Frame(left_frame, bg="white", pady=10)

top_frame.pack(fill="x")


username_label = tk.Label(top_frame, font=("Arial", 14, "bold"), bg="white")

username_label.config(text=name)

username_label.bind("<Button-1>", lambda e: pro_open())  # Bind the click event to pro_open

username_label.pack(side="left")


bell_icon = Image.open("E:/bell.png").resize((20, 20))

bell_photo = ImageTk.PhotoImage(bell_icon)

bell_label = tk.Label(top_frame, image=bell_photo, bg="white")

bell_label.image = bell_photo

bell_label.pack(side="right", padx=10)


# Trip Type Selection

trip_type_frame = tk.Frame(left_frame, bg="white", pady=10)

trip_type_frame.pack(fill="x", pady=10)

trip_type = tk.StringVar(value="One Way")


one_way_rb = ttk.Radiobutton(trip_type_frame, text="One Way", variable=trip_type, value="One Way")

round_rb = ttk.Radiobutton(trip_type_frame, text="Round", variable=trip_type, value="Round",
command=toggle_return_date)


one_way_rb.pack(side="left", padx=10)

round_rb.pack(side="left", padx=10)
```

```
# Journey Details Section

journey_frame = tk.Frame(left_frame, bg="#003366", pady=10, padx=10)

journey_frame.pack(fill="x", pady=10, padx=10)


from_label = tk.Label(journey_frame, text="From", font=("Arial", 10, "bold"), fg="white", bg="#003366")

from_label.grid(row=0, column=0, sticky="w")


from_city = tk.StringVar()

cities = ["CHENNAI CHE", "KERALA KL", "City A", "City B", "City C", "City D", "City E", "City F",

"City G", "City H", "City I", "City J", "City K", "City L", "City M", "City N",

"City O", "City P", "City Q", "City R", "City S", "City T"]


from_city_combobox = ttk.Combobox(journey_frame, textvariable=from_city, values=cities,
state="readonly")

from_city_combobox.grid(row=1, column=0, sticky="w", padx=10)

from_city_combobox.set("CHENNAI CHE")




to_label = tk.Label(journey_frame, text="To", font=("Arial", 10, "bold"), fg="white", bg="#003366")

to_label.grid(row=2, column=0, sticky="w", pady=(10, 0))


to_city = tk.StringVar()

to_city_combobox = ttk.Combobox(journey_frame, textvariable=to_city, values=cities, state="readonly")

to_city_combobox.grid(row=3, column=0, sticky="w", padx=10)
```

```
to_city_combobox.set("KERALA KL")  # Default value


swap_button = tk.Button(journey_frame, text="↑↓", font=("Arial", 12, "bold"), bg="white", fg="#003366", command=swap_locations)

swap_button.grid(row=2, column=1, rowspan=2, padx=10)


# Journey Information Fields

info_frame = tk.Frame(left_frame, bg="white", pady=10, padx=10)

info_frame.pack(fill="x", pady=10, padx=10)


departure_label = tk.Label(info_frame, text="Departure", font=("Arial", 10, "bold"), bg="white")

departure_label.grid(row=0, column=0, sticky="w")


departure_date = DateEntry(info_frame, width=12, background='darkblue', foreground='white', borderwidth=2)

departure_date.grid(row=0, column=1, padx=10, sticky="w")


return_label = tk.Label(info_frame, text="Return", font=("Arial", 10, "bold"), bg="white")

return_label.grid(row=1, column=0, sticky="w")


return_date = DateEntry(info_frame, width=12, background='darkblue', foreground='white', borderwidth=2, state="disabled")

return_date.grid(row=1, column=1, padx=10, sticky="w")


# Search Button

search_button = tk.Button(left_frame, text="SEARCH TRAIN", font=("Arial", 14, "bold"), bg="#003366", fg="white", command=search_train)

search_button.pack(fill="x", pady=20, padx=20)
```

```
# Right Frame

right_frame = tk.Frame(main, bg='white', padx=10, pady=10)

right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)


table_label = tk.Label(right_frame, text="Train Details Table", font=("Arial", 16, "bold"), bg='white')

table_label.pack(pady=10)


columns = ("Train No", "Train Name", "Date", "Time", "Source", "Destination")

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

for col in columns:

tree.heading(col, text=col)

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

buttons_frame = tk.Frame(right_frame, bg='white')

buttons_frame.pack(fill=tk.X, pady=10)

book_btn = tk.Button(buttons_frame, text="BOOKING", bg='blue', fg='white', padx=10, pady=5)

book_btn.bind("<Button-1>", lambda e: create_passenger_details_app())

book_btn.pack(side=tk.LEFT, padx=5)

cancel_btn = tk.Button(buttons_frame, text="CANCELLATION", bg='red', fg='white', padx=10, pady=5,
command=cancel_ticket)

cancel_btn.pack(side=tk.LEFT, padx=5)


def search_train():

# Ensure the database connection is active

get_db_connection()
```

```
# MySQL connection

try:

query = """

SELECT train_no, train_name, date, time, source, destination

FROM trains

WHERE source = %s AND destination = %s AND date = %s

"""

source = from_city.get()

destination = to_city.get()

date = departure_date.get_date()


# Debugging: Print the values being used in the query

print(f"Searching for trains from {source} to {destination} on {date}")


rows = execute_query(query, (source, destination, date))


# Clear the treeview

for item in tree.get_children():

tree.delete(item)


# Insert data into the treeview

if not rows:

messagebox.showinfo("No Results", "No trains found for the selected criteria.")

else:

for row in rows:

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

```
# Debugging: Print the rows fetched

print(f"Found {len(rows)} trains.")

for row in rows:

print(row)


except mysql.connector.Error as e:

messagebox.showerror("Database Error", f"Error connecting to database: {e}")


def cancel_ticket():

messagebox.showinfo("Cancellation", "Cancelling ticket...")


def toggle_return_date():

if trip_type.get() == "Round":

return_date.config(state="normal")

else:

return_date.config(state="disabled")


def swap_locations():

global from_city, to_city

temp = from_city.get()

from_city.set(to_city.get())

to_city.set(temp)


def BookingSection(name):

book = tk.Tk()
```

```
book.title("Booking History")

screen_width = book.winfo_screenwidth()

screen_height = book.winfo_screenheight()

width = 800

height = 500

x = (screen_width / 2) - (width / 2)

y = (screen_height / 2) - (height / 2)

book.geometry('%dx%d+%d+%d' % (width, height, x, y))

book.configure(bg="#f0f0f0")

bookings = get_booking_data()


# Top Frame for User Information

top_frame = tk.Frame(book, bg="white", pady=10)

top_frame.pack(fill=tk.X)

# User Info

user_info_frame = tk.Frame(top_frame, bg="white")

user_info_frame.pack(side=tk.LEFT, padx=10)

user_name = tk.Label(user_info_frame, font=("Helvetica", 14, "bold"), bg="white")

user_name.config(text=name)

user_name.pack(anchor="w")

last_login = tk.Label(user_info_frame, text=f"{datetime.datetime.now():%I:%M:%S %p, %d-%m-%Y}",

font=("Helvetica", 10), bg="white", fg="gray")

last_login.pack(anchor="w")


# History Button
```

```
history_button = tk.Button(top_frame, text="Back", font=("Helvetica", 10), bg="white", fg="blue", bd=0, cursor="hand2")

history_button.pack(side=tk.RIGHT, padx=10)


# Scrollable Frame

container = tk.Frame(book, bg="#f0f0f0")

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


canvas = tk.Canvas(container, bg="#f0f0f0")

scrollbar = ttk.Scrollbar(container, orient="vertical", command=canvas.yview)

profilescrollable_frame = ttk.Frame(canvas)


profilescrollable_frame.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))


canvas.create_window((0, 0), window=profilescrollable_frame, anchor="nw")

canvas.configure(yscrollcommand=scrollbar.set)


canvas.pack(side="left", fill=tk.BOTH, expand=True)

scrollbar.pack(side="right", fill="y")


# Create Booking Cards

for booking in bookings:

create_booking_card(profilescrollable_frame, booking)


def create_booking_card(parent, booking):

card_frame = tk.Frame(parent, bg="white", bd=1, relief="solid", padx=10, pady=10)
```

```
card_frame.pack(fill=tk.X, pady=5, padx=10)

# Train Info
train_info_frame = tk.Frame(card_frame, bg="white")
train_info_frame.pack(fill=tk.X)
train_name = tk.Label(train_info_frame, text=f"{booking['train_name']} ({booking['train_number']})",
font=("Helvetica", 12, "bold"), bg="white")
train_name.pack(side=tk.LEFT)

# Status Label
status_color = "blue" if booking["status"] == "BOOKED" else "red"
status_label = tk.Label(train_info_frame, text=booking["status"], font=("Helvetica", 10, "bold"),
bg="white", fg=status_color)
status_label.pack(side=tk.RIGHT)

# Departure, Arrival, Duration
time_frame = tk.Frame(card_frame, bg="white")
time_frame.pack(fill=tk.X, pady=5)
departure = tk.Label(time_frame, text=f"Departs: {booking['departure']}", font=("Helvetica", 10),
bg="white")
departure.pack(side=tk.LEFT)
arrival = tk.Label(time_frame, text=f"Arrives: {booking['arrival']}", font=("Helvetica", 10), bg="white")
arrival.pack(side=tk.LEFT, padx=10)
duration = tk.Label(time_frame, text=f"Duration: {booking['duration']}", font=("Helvetica", 10),
bg="white")
duration.pack(side=tk.LEFT)
```

```
# Days of Operation

days_label = tk.Label(card_frame, text=f"Departs on: {booking['days']}", font=("Helvetica", 10),
bg="white")

days_label.pack(fill=tk.X, pady=5)


# Financial Info

financial_frame = tk.Frame(card_frame, bg="white")

financial_frame.pack(fill=tk.X, pady=5)

bank_balance = tk.Label(financial_frame, text=f"Bank balance: {booking['bank_balance']}",

font=("Helvetica", 10), bg="white")

bank_balance.pack(side=tk.LEFT)

ticket_price = tk.Label(financial_frame, text=f"Ticket price + Tax: {booking['ticket_price']}",

font=("Helvetica", 10), bg="white")

ticket_price.pack(side=tk.LEFT, padx=10)

available_balance = tk.Label(financial_frame, text=f"Available balance: {booking['available_balance']}",

font=("Helvetica", 10, "bold"), bg="white", fg="blue")

available_balance.pack(side=tk.LEFT)


def get_booking_data():

return [

{

"train_name": "Godan Express",

"train_number": "00000",

"departure": "00:00PM",

"arrival": "00:00AM",

"duration": "00D 00H 00M",
```

"days": "Monday, Wednesday, Friday, Sunday",

"status": "BOOKED",

"bank_balance": "10,500 ₹",

"ticket_price": "650 ₹ + 50 ₹",

"available_balance": "9,800 ₹"

},

{

"train_name": "Godan Express",

"train_number": "00000",

"departure": "00:00PM",

"arrival": "00:00AM",

"duration": "00D 00H 00M",

"days": "Monday, Wednesday, Friday, Sunday",

"status": "CANCELED",

"bank_balance": "11,000 ₹",

"ticket_price": "650 ₹ - 150 ₹",

"available_balance": "10,500 ₹"

},

{

"train_name": "Godan Express",

"train_number": "00000",

"departure": "00:00PM",

"arrival": "00:00AM",

"duration": "00D 00H 00M",

"days": "Monday, Wednesday, Friday, Sunday",

"status": "CANCELED",

```
"bank_balance": "11,000 ₹",
"ticket_price": "650 ₹ - 150 ₹",
"available_balance": "10,500 ₹"
}
]

def transaction_history():
r = tk.Tk()
r.title("Transaction History")
screen_width = r.winfo_screenwidth()
screen_height = r.winfo_screenheight()
width = 800
height = 500
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
r.geometry('%dx%d+%d+%d' % (width, height, x, y))

# Header Section
header_frame = tk.Frame(r)
header_frame.pack(pady=10, fill=tk.X)

title_label = tk.Label(header_frame, text="Transaction History", font=("Arial", 16, "bold"))
title_label.pack(side=tk.LEFT, padx=10)

close_button = tk.Button(header_frame, text="X", command=r.destroy, font=("Arial", 12), fg="red")
close_button.pack(side=tk.RIGHT, padx=10)
```

```
# Toggle Switch

toggle_var = tk.StringVar(value="Completed")

toggle_frame = tk.Frame(r)

toggle_frame.pack(pady=10)


completed_button = tk.Radiobutton(toggle_frame, text="Completed", variable=toggle_var,
value="Completed", command=lambda: filter_transactions(toggle_var, transaction_list))

completed_button.pack(side=tk.LEFT, padx=5)


pending_button = tk.Radiobutton(toggle_frame, text="Pending", variable=toggle_var, value="Pending",
command=lambda: filter_transactions(toggle_var, transaction_list))

pending_button.pack(side=tk.LEFT, padx=5)


# Transaction History List

transaction_list = ttk.Treeview(r, columns=("Train Name", "Date", "Type", "Amount", "Method", "Status"),
show="headings")

transaction_list.heading("Train Name", text="Train Name & Number")

transaction_list.heading("Date", text="Transaction Date")

transaction_list.heading("Type", text="Transaction Type")

transaction_list.heading("Amount", text="Amount")

transaction_list.heading("Method", text="Payment Method")

transaction_list.heading("Status", text="Transaction Status")

transaction_list.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)


# Scrollbar

scrollbar = ttk.Scrollbar(r, orient="vertical", command=transaction_list.yview)
```

```
transaction_list.configure(yscroll=scrollbar.set)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)


# Footer Section

footer_frame = tk.Frame(r)

footer_frame.pack(side=tk.BOTTOM, pady=10)


terms_label = tk.Label(footer_frame, text="Terms of Service | Privacy Policy")

terms_label.pack(side=tk.LEFT, padx=10)


logo_label = tk.Label(footer_frame, text="Company Logo", font=("Arial", 10, "italic"))

logo_label.pack(side=tk.RIGHT, padx=10)


# Sample Data

transactions = [

("Express Train 101", "2023-10-01", "Ticket Booking", 1500, "UPI", "Successful"),

("Local Train 202", "2023-10-02", "Refund", -500, "Credit Card", "Successful"),

("Express Train 303", "2023-10-03", "Cancellation", -1500, "Net Banking", "Failed"),

("Local Train 404", "2023-10-04", "Ticket Booking", 1200, "UPI", "Pending"),

]


populate_transactions(transaction_list, transactions)


def populate_transactions(transaction_list, transactions):

"""Populate the transaction list with color-coded statuses."""

transaction_list.delete(*transaction_list.get_children())  # Clear old data
```

```
for transaction in transactions:

amount_color = "green" if transaction[2] == "Refund" else "red" if transaction[2] == "Cancellation" else "black"

status_color = "green" if transaction[5] == "Successful" else "red" if transaction[5] == "Failed" else "orange"


row_id = transaction_list.insert("", "end", values=transaction)

transaction_list.tag_configure(row_id, foreground=amount_color)

transaction_list.tag_configure(row_id, foreground=status_color)


def filter_transactions(toggle_var, transaction_list):

"""Filter transactions based on 'Completed' and 'Pending'."""

selected_filter = toggle_var.get()

transaction_list.delete(*transaction_list.get_children())  # Clear old data


for transaction in transactions:

if selected_filter == "Completed" and transaction[5] in ["Successful", "Failed"]:

transaction_list.insert("", "end", values=transaction)

elif selected_filter == "Pending" and transaction[5] == "Pending":

transaction_list.insert("", "end", values=transaction)


class ProfileWindow(tk.Tk):

def __init__(self, username):

super().__init__()

self.title("Profile - Railway Reservation System")

screen_width = self.winfo_screenwidth()
```

```
screen_height = self.winfo_screenheight()

width = 800

height = 500

x = (screen_width / 2) - (width / 2)

y = (screen_height / 2) - (height / 2)

self.geometry('%dx%d+%d+%d' % (width, height, x, y))

self.username = username  # Store the username

self.create_widgets()


def create_widgets(self):

# Top Frame - Profile Section

top_frame = tk.Frame(self, bg="white", bd=2, relief=tk.RIDGE)

top_frame.pack(pady=10, padx=10, fill=tk.X)

# Name, Email, and Phone Number

self.name_var = tk.StringVar()

self.email_var = tk.StringVar()

self.phone_var = tk.StringVar()


name_label = tk.Label(top_frame, text="Name:", font=("Helvetica", 12), bg="white")

name_label.grid(row=0, column=1, sticky="w", padx=10, pady=2)

self.name_entry = tk.Entry(top_frame, textvariable=self.name_var, font=("Helvetica", 12), bg="white",
state="readonly")

self.name_entry.grid(row=0, column=2, sticky="w", padx=10, pady=2)

email_label = tk.Label(top_frame, text="Email:", font=("Helvetica", 12), bg="white")

email_label.grid(row=1, column=1, sticky="w", padx=10, pady=2)

self.email_entry = tk.Entry(top_frame, textvariable=self.email_var, font=("Helvetica", 12), bg="white",
state="readonly")
```

```
self.email_entry.grid(row=1, column=2, sticky="w", padx=10, pady=2)

phone_label = tk.Label(top_frame, text="Phone:", font=("Helvetica", 12), bg="white")

phone_label.grid(row=2, column=1, sticky="w", padx=10, pady=2)

self.phone_entry = tk.Entry(top_frame, textvariable=self.phone_var, font=("Helvetica", 12), bg="white", state="readonly")

self.phone_entry.grid(row=2, column=2, sticky="w", padx=10, pady=2)


# Edit Button

self.edit_button = tk.Button(top_frame, text="Edit", command=self.edit_profile, bg="#007BFF", fg="white", cursor="hand2")

self.edit_button.grid(row=0, column=3, rowspan=3, padx=10, pady=10)


# Middle Frame - Booking History Section

middle_frame = tk.Frame(self, bg="white", bd=2, relief=tk.RIDGE)

middle_frame.pack(pady=10, padx=10, fill=tk.X)

booking_label = tk.Label(middle_frame, text="Booking History", font=("Helvetica", 14, "bold"), bg="white")

booking_label.pack(pady=10)

booking_button = tk.Button(middle_frame, text="View Bookings", command=self.view_bookings, bg="#007BFF", fg="white", cursor="hand2")

booking_button.pack(pady=10)


# Bottom Frame - Transaction History Section

bottom_frame = tk.Frame(self, bg="white", bd=2, relief=tk.RIDGE)

bottom_frame.pack(pady=10, padx=10, fill=tk.X)

transaction_label = tk.Label(bottom_frame, text="Transaction History", font=("Helvetica", 14, "bold"), bg="white")

transaction_label.pack(pady=10)
```

```
transaction_button = tk.Button(bottom_frame, text="View Transactions",
command=self.view_transactions, bg="#007BFF", fg="white", cursor="hand2")

transaction_button.pack(pady=10)



self.load_profile()



def load_profile(self):

query = "SELECT username, phone, email FROM users WHERE username = %s"

result = execute_query(query, (self.username,))

if result:

self.name_var.set(result[0][0])

self.phone_var.set(result[0][1])

self.email_var.set(result[0][2])



def edit_profile(self):

if self.edit_button["text"] == "Edit":

self.name_entry.config(state="normal")

self.email_entry.config(state="normal")

self.phone_entry.config(state="normal")

self.edit_button.config(text="Save")

else:

self.name_entry.config(state="readonly")

self.email_entry.config(state="readonly")

self.phone_entry.config(state="readonly")

self.edit_button.config(text="Edit")

self.save_profile()
```

```
messagebox.showinfo("Profile Updated", "Profile information updated successfully")


def save_profile(self):

name = self.name_entry.get()

email = self.email_entry.get()

phone = self.phone_entry.get()

query = "UPDATE users SET username = %s, email = %s, phone = %s WHERE username = %s"

execute_query(query, (name, email, phone, self.username))

conn.commit()


def view_bookings(self):

BookingSection(self.username)


def view_transactions(self):

transaction_history()


def pro_open():

# Create an instance of ProfileWindow and pass the username

profile_window = ProfileWindow(namee)  # Assuming namee holds the username

profile_window.mainloop()  # Start the profile window's main loop


def create_passenger_details_app():

def go_back():

root.destroy()


def update_total_fare():
```

```
global total_fare

total_fare = 0

for child in tree.get_children():

total_fare += int(tree.item(child, "values")[5])  # Column index 5 is the fare

pay_now_button.config(text=f"PAY NOW (₹{total_fare})")


def add_to_table():

# Check if there are already 8 entries in the table

if len(tree.get_children()) >= 8:

messagebox.showerror("Limit Reached", "You can only add up to 8 passengers.")

return


# Retrieve Data from Entry Fields

name = name_entry.get()

age = age_entry.get()

gender = gender_entry.get()




if name and age and gender :

try:

age = int(age)  # Ensure Age is a number


if (gender.lower() == "male") and age >= 50:

fare = 80

elif (gender.lower() == "male") and 13 <= age < 50:

fare = 100
```

```
elif gender.lower() == "female":

fare = 60

elif age <= 12:  # Allow age 12

fare = 50

else:

messagebox.showerror("Invalid Input", "Invalid passenger type.")

return


# Insert data into the table with the calculated fare

tree.insert("", "end", values=(name, age, gender, fare))


# Update total fare dynamically

update_total_fare()


# Clear Entries

name_entry.delete(0, tk.END)

age_entry.delete(0, tk.END)

gender_entry.delete(0, tk.END)


except ValueError:

messagebox.showerror("Invalid Input", "Age must be a number.")

else:

messagebox.showerror("Missing Information", "All fields are required.")


root = tk.Tk()

root.title("Passenger Details")
```

```
root.configure(bg="white")


screen_width = root.winfo_screenwidth()

screen_height = root.winfo_screenheight()

width = 1000

height = 600

x = (screen_width / 2) - (width / 2)

y = (screen_height / 2) - (height / 2)

root.geometry('%dx%d+%d+%d' % (width, height, x, y))


# Back Button

back_button = tk.Button(root, text=" ⬅ BACK ", command=go_back, bg="#1E3A8A", fg="white", font=("Arial", 12), relief="flat")

back_button.place(x=10, y=20, width=40, height=40)


global tree

global total_fare

total_fare = 0


# Treeview Table with Fare Column

tree = ttk.Treeview(root, columns=("Name", "Age", "Gender", "Fare"), show="headings")

tree.heading("Name", text="Name")

tree.heading("Age", text="Age")

tree.heading("Gender", text="Gender")

tree.heading("Fare", text="Fare")
```

```
for col in tree["columns"]:
tree.column(col, width=100, anchor="center")

tree.place(x=50, y=100, width=900, height=300)

# Scrollbar for Treeview
scrollbar = ttk.Scrollbar(root, orient="vertical", command=tree.yview)
tree.configure(yscroll=scrollbar.set)
scrollbar.place(x=950, y=100, height=300)

# Form for Passenger Details
form_frame = tk.Frame(root, bg="white")
form_frame.place(x=50, y=450)

# Name, Age, Gender
tk.Label(form_frame, text="Name:", bg="white").grid(row=0, column=0, padx=5, pady=5)
name_entry = tk.Entry(form_frame, width=20)
name_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Age:", bg="white").grid(row=0, column=2, padx=5, pady=5)
age_entry = tk.Entry(form_frame, width=10)
age_entry.grid(row=0, column=3, padx=5, pady=5)

tk.Label(form_frame, text="Gender:", bg="white").grid(row=0, column=4, padx=5, pady=5)
gender_entry = tk.Entry(form_frame, width=20)
gender_entry.grid(row=0, column=5, padx=5, pady=5)
```

```
# Add Passenger Button

add_button = tk.Button(form_frame, text="Add Passenger", command=add_to_table, bg="#1E3A8A",
fg="white", font=("Arial", 12), relief="flat")

add_button.grid(row=2, column=0, columnspan=6, pady=10)



# "Pay Now" Button

pay_now_button = tk.Button(root, text="PAY NOW (₹0)", command=lambda:
initialize_payment_app(total_fare), bg="#1E3A8A", fg="white", font=("Arial", 12), relief="flat")

pay_now_button.place(x=800, y=500, width=150, height=40)



def initialize_payment_app(total_fare):

r = tk.Toplevel()  # Use Toplevel for the payment window

r.title("Payment Options")

r.geometry("400x600")

r.configure(bg="#f3f4f6")



# Global variables

selected_payment_method = tk.StringVar(value="")

balance = 3508  # Dynamic balance

required_amount = total_fare



# Function to go back

def go_back():

r.destroy()  # Close the payment options window



# Function to add money
```

```
def add_money():
messagebox.showinfo("Add Money", "Add money functionality.")

# Function to handle payment
def pay_now():
nonlocal balance
if balance >= required_amount:
messagebox.showinfo("Payment Successful", f"Payment of ₹{required_amount} was successful.")
balance -= required_amount
update_balance()
else:
messagebox.showwarning("Insufficient Balance", "Your balance is insufficient to make this payment.")

# Function to check balance and update Pay Now button state
def check_balance():
if balance < required_amount:
pay_now_button.config(state=tk.DISABLED)
else:
pay_now_button.config(state=tk.NORMAL)

# Function to update balance label
def update_balance():
balance_label.config(text=f"Available balance: ₹{balance}")
check_balance()

# Function to handle payment method selection
```

```
def on_payment_method_selected():

if selected_payment_method.get() == "IRCTC Pay":

irctc_pay_frame.pack(fill=tk.X, pady=10, padx=10)

else:

irctc_pay_frame.pack_forget()


# Header section

header_frame = tk.Frame(r, bg="#4a4e69")

header_frame.pack(fill=tk.X)


back_button = tk.Button(header_frame, text="🔙", bg="#4a4e69", fg="white", borderwidth=0,
command=go_back)

back_button.pack(side=tk.LEFT, padx=10, pady=10)


title_label = tk.Label(header_frame, text="Payment Options", bg="#4a4e69", fg="white",
font=("Helvetica", 16, "bold"))

title_label.pack(side=tk.LEFT, padx=10, pady=10)


train_details_label = tk.Label(header_frame, text="BBS-ROU | ONE WAY | 25 NOV 2020", bg="#4a4e69",
fg="white", font=("Helvetica", 10))

train_details_label.pack(side=tk.RIGHT, padx=10, pady=10)


# Payment methods section

payment_methods_frame = tk.Frame(r, bg="white")

payment_methods_frame.pack(fill=tk.X, pady=10, padx=10)


def create_payment_method_option(parent, method, note):
```

```
frame = tk.Frame(parent, bg="white", bd=1, relief=tk.SOLID)

frame.pack(fill=tk.X, pady=5)


radio_button = tk.Radiobutton(frame, text=method, variable=selected_payment_method,
value=method,

bg="white", fg="black", font=("Helvetica", 12), command=on_payment_method_selected)

radio_button.pack(anchor=tk.W, padx=10, pady=5)


note_label = tk.Label(frame, text=note, bg="white", fg="gray", font=("Helvetica", 10))

note_label.pack(anchor=tk.W, padx=10, pady=5)


create_payment_method_option(payment_methods_frame, "IRCTC Pay", "10% Cashback with this
payment method.")

create_payment_method_option(payment_methods_frame, "UPI/Wallets", "Paytm, Google Pay, BHIM,
PhonePe, Amazon Pay")

create_payment_method_option(payment_methods_frame, "Debit/Credit Cards", "Bank charges may
apply")

create_payment_method_option(payment_methods_frame, "Internet Banking", "Bank charges may
apply")


# IRCTC Pay section

irctc_pay_frame = tk.Frame(r, bg="white", bd=1, relief=tk.SOLID)


irctc_pay_label = tk.Label(irctc_pay_frame, text="IRCTC Pay", bg="white", fg="#4a4e69",
font=("Helvetica", 14, "bold"))

irctc_pay_label.pack(anchor=tk.W, padx=10, pady=5)


account_holder_label = tk.Label(irctc_pay_frame, text="Account: Vupati Bhusan Sahoo ☑", bg="white",
fg="black", font=("Helvetica", 12))
```

```
account_holder_label.pack(anchor=tk.W, padx=10, pady=5)


balance_label = tk.Label(irctc_pay_frame, text=f"Available balance: ₹{balance}", bg="white", fg="black", font=("Helvetica", 12))

balance_label.pack(anchor=tk.W, padx=10, pady=5)


button_frame = tk.Frame(irctc_pay_frame, bg="white")

button_frame.pack(fill=tk.X, padx=10, pady=10)


add_money_button = tk.Button(button_frame, text="Add Money", bg="#4a4e69", fg="white", font=("Helvetica", 12), command=add_money)

add_money_button.pack(side=tk.LEFT, padx=10)


pay_now_button = tk.Button(button_frame, text=f"Pay Now ₹{required_amount}", bg="#4a4e69", fg="white", font=("Helvetica", 12), command=pay_now)

pay_now_button.pack(side=tk.RIGHT, padx=10)


update_balance()


def lomain():

global log

log = tk.Tk()

log.title("Login Page")

screen_width = log.winfo_screenwidth()

screen_height = log.winfo_screenheight()

width = 800

height = 350
```

```
x = (screen_width / 2) - (width / 2)

y = (screen_height / 2) - (height / 2)

log.geometry('%dx%d+%d+%d' % (width, height, x, y))

log.configure(bg="#f3f4f6")


left_frame = tk.Frame(log, bg="#f3f4f6")

left_frame.pack(side="left", fill="both", expand=True)


title_font = tkfont.Font(family="Helvetica", size=20, weight="bold")

button_font = tkfont.Font(family="Helvetica", size=12, weight="bold")

link_font = tkfont.Font(family="Helvetica", size=10, underline=True)


title_label = tk.Label(left_frame, text="Welcome Back!", font=title_font, fg="#1e3a8a", bg="#f3f4f6")

title_label.pack(padx=10, side="top", anchor="w")


padding_space = tk.Label(left_frame, bg="#f3f4f6", height=2)

padding_space.pack()


username_frame = tk.Frame(left_frame, bg="#f3f4f6")

username_frame.pack(pady=10, fill="x")

username_icon = tk.Label(username_frame, text=" 👤 ", font=("Helvetica", 16), bg="#f3f4f6")

username_icon.pack(side="left", padx=10)

global username_entry

username_entry = tk.Entry(username_frame, font=("Helvetica", 14), bg="#e5e7eb", relief="flat",
width=25)

username_entry.pack(side="left", padx=(0, 10))
```

```
password_frame = tk.Frame(left_frame, bg="#f3f4f6")

password_frame.pack(pady=10, fill="x")

password_icon = tk.Label(password_frame, text=" 🔓 ", font=("Helvetica", 16), bg="#f3f4f6")

password_icon.pack(side="left", padx=10)

global password_entry

password_entry = tk.Entry(password_frame, font=("Helvetica", 14), bg="#e5e7eb", relief="flat",
width=25, show="*")

password_entry.pack(side="left", padx=(0, 10))



global password_eye

password_eye = tk.Label(password_frame, text="👁", font=("Helvetica", 16), bg="#f3f4f6",
cursor="hand2")

password_eye.pack(side="left", padx=12)

password_eye.bind("<Button-1>", lambda e: toggle_password())



forgot_password_link = tk.Label(left_frame, text="Forgot Password?", font=link_font, fg="#1e3a8a",
bg="#f3f4f6", cursor="hand2")

forgot_password_link.pack(pady=10)

forgot_password_link.bind("<Button-1>", lambda e: open_forgot_password_window())



login_button = tk.Button(left_frame, text="Login", font=button_font, bg="#1e3a8a", fg="white",
relief="flat", width=20, command=login)

login_button.pack(pady=20)



register_frame = tk.Frame(left_frame, bg="#f3f4f6")

register_frame.pack(pady=10)
```

```
register_label = tk.Label(register_frame, text="Don't have an account?", font=("Helvetica", 10), bg="#f3f4f6")

register_label.pack(side="left")

register_link = tk.Label(register_frame, text="Register Now", font=link_font, fg="#1e3a8a", bg="#f3f4f6", cursor="hand2")

register_link.pack(side="left")

register_link.bind("<Button-1>", lambda e: create_main_window())

global message_label

message_label = tk.Label(left_frame, text="", bg="#f3f4f6")

message_label.pack(pady=5)


right_frame = tk.Frame(log, bg="#f3f4f6")

right_frame.pack(side="left", fill="both", expand=True)


image_path = "E:/train.png"

image = Image.open(image_path)

image = image.resize((400, 400))

photo = ImageTk.PhotoImage(image)

image_label = tk.Label(right_frame, image=photo, bg="#f3f4f6")

image_label.image = photo

image_label.pack(pady=20)


log.mainloop()


lomain()
```