VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A Database Management System Mini Project Report
on

"RAILWAY BOOKING MANAGEMENT"

Submitted in Partial fulfillment of the Requirements for the V Semester of the Degree of

Bachelor of Engineering
In
Computer Science & Engineering
By

SAI PRIYADARSHINI
(1CR17CS124)

Under the Guidance of

Mrs. Sagarika Behera
Assoc Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD,

KUNDALAHALLI, BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD,

KUNDALAHALLI, BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CELEBRATING 25 YEARS

CMRIT

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.

ACCREDITED WITH A+ GRADE BY NAAC

CERTIFICATE

This is to certify that the Database Management System Project work entitled **"HOTEL MANAGEMENT"** has been carried out by **Sai Priyadarshini (1CR17CS124)** bonafide students of CMR Institute of Technology in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2019-2020**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. This DBMS Project Report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

-----------------------	**----------------------**
Signature of Guide	**Signature of HOD**
Mrs. Sagarika Behera **Assoc Professor** **Dept. of CSE, CMRIT**	**Dr. Prem Kumar Ramesh** **Professor, Head** **Dept. of CSE, CMRIT**

<u>External Viva</u>

Name of the examiners	Signature with date
1.	
2.	

ABSTRACT

The aim of this project is to create a web-based application for railway management and booking that would guide the user or the customer through the entire process of ticket booking, searching and cancelling said booking.

ACKNOWLEDGEMENT

I am thankful to, and fortunate enough to get constant encouragement, support and guidance from our **Dr. Prem Kumar Ramesh,** HOD and the teaching staff of the Department of CSE, that helped me in successfully completion of my project. Also, I would like to extend our sincere gratitude to all staff in laboratory for their timely support.

I owe my deep gratitude to my project guide **Mrs. Sagarika Behera**, who took keen interest in my project work and guided me all along, till the completion of my project work by providing all the necessary information for developing a good system.

Lastly, I would like to thank all my classmates and friends who helped me finish this project in any way directly, or indirectly.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Chapter 1

INTRODUCTION

A Railway Management interface is very important in today’s world because the travel and hospitality sectors are expanding exponentially, as the world becomes more economically stable. This will require a very elaborate and efficient form of database management system (but also simple to use) that can support the smooth functioning of trains along with its massively growing customers.
The database management system needs a solid interconnection in between the management and customer interaction so that the entire reservation system can run without any problems.
This project will focus on the process of train ticket booking and personalized user experience for their travel.
The following process will take place on the portal for a potential customer:

- The first screen will be a simple GUI showing three functions, viz., book a ticket, search for a reservation, and cancel a reservation.
- When the user clicks on the button to book a ticket, a set of text boxes appear in which the user enters relevant details.
- Once the user submits the information, the details are stored in the passenger table.
- When the user wants to search for a booking, just by clicking the Search Booking button they can access a record by entering the reservation number (which is the primary key for the passenger table)
- When the user wants to cancel a booking, just by clicking the Cancel Booking button they can cancel their reservation by entering the reservation number. The record is then taken off the table.

Chapter 2

SYSTEM REQUIREMENTS

Section 2.1

Hardware Requirements

- **Processor:** Intel i5
- **Processor Speed:** 2.5 GHz
- **RAM:** 8 GB
- **Storage:** 1 TB

Section 2.2

Software Requirements

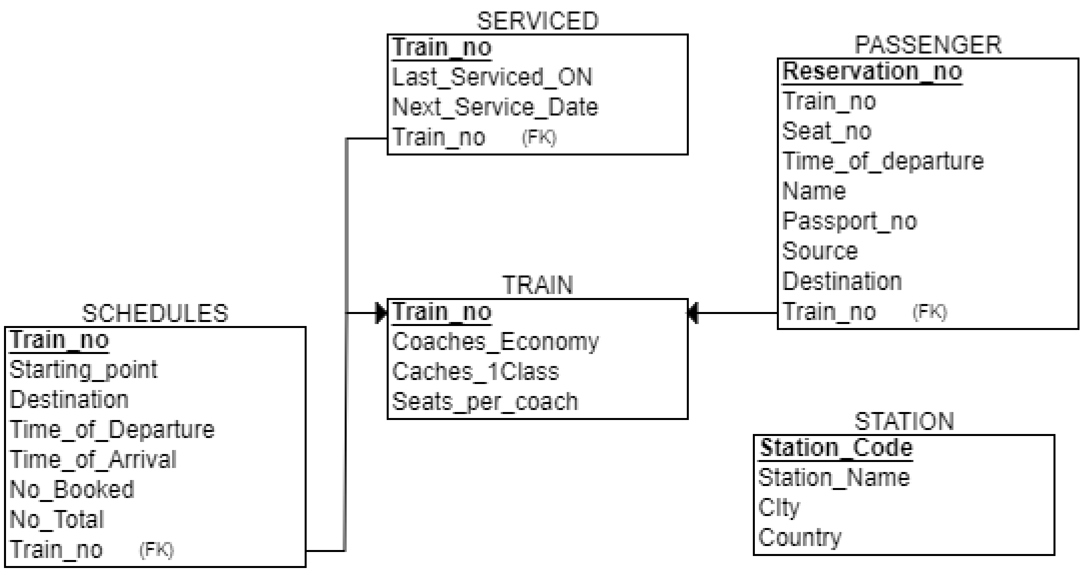
- **Operating System:** Windows 10
- **Other Software:** Python 2.7(VS Code), MySQL and XAMPP for DBMS
- **Web Browser:** Google Chrome/Microsoft Edge

Chapter 3

DESIGN

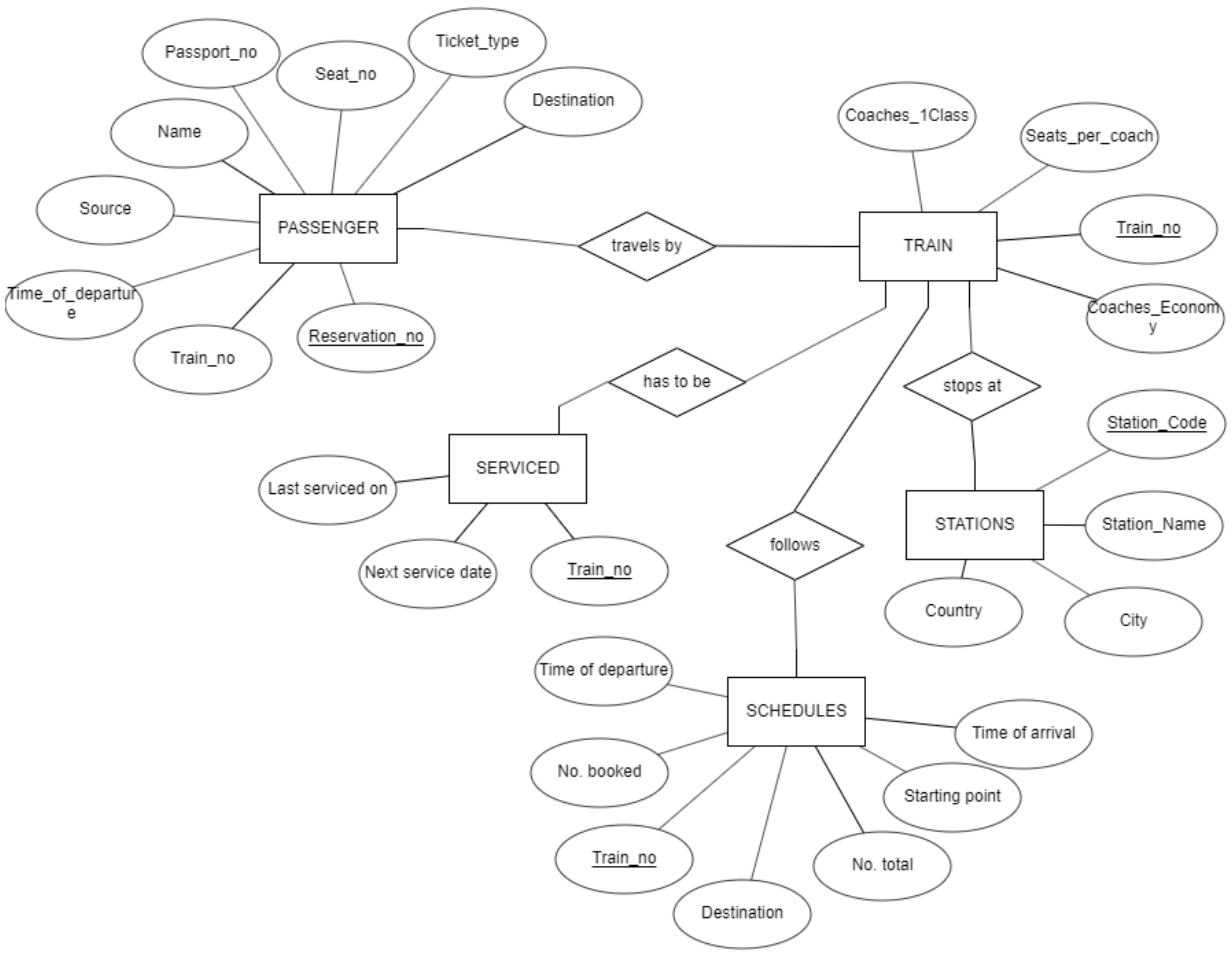
Section 3.1

Schema



Section 3.2

E-R Diagram



Tables Information:

PASSENGER – Stores the details of all reservations made by the passengers.

TRAIN – Stores the details of the commercial trains, like the number of coaches.

SCHEDULES – Stores the timings and locations of trains that are operating.

STATIONS – Stores the details of the stations in operating cities and countries.

SERVICED – Stores the maintenance details of the trains

Chapter 4

Section 4.1
Connection of the database to the code and main function

```
from Tkinter import *

import tkMessageBox

import MySQLdb

# Open database connection

conn = MySQLdb.connect("localhost","root","","database")

# prepare a cursor object using cursor() method

c = conn.cursor()

#tkinter window

class Application:

    def __init__(self, master):

        self.master = master

        #creating the frames in the master

        self.left = Frame(master, width=800, height=720, bg='grey')

        self.left.pack(side = LEFT)

        self.right = Frame(master, width=400, height=720, bg='black')

        self.right.pack(side = RIGHT)
        #labels for the window

        self.heading = Label(self.left, text = "Railway Bookings",
font=('Cocogoose 40'), fg='white', bg='grey')

     self.heading.place(x=0, y=0)

        self.New_Booking = Button(self.right, text="Book New Ticket", width=30,
height=1, bg='grey', fg='black', font=('Cocogoose 10'), command=lambda:
self.new_ticket())

        self.New_Booking.place(x=65, y=100)

        self.Search_Booking = Button(self.right, text="Search Booking", width=30,
height=1, bg='grey', fg='black', font=('Cocogoose 10'), command=lambda:
self.search())

        self.Search_Booking.place(x=65, y=150)

        self.Cancel_Booking = Button(self.right, text="Cancel Ticket", width=30,
height=1, bg='grey', fg='black', font=('Cocogoose 10'), command=lambda:
self.cancel())

        self.Cancel_Booking.place(x=65, y=200)
```

Section 4.2

Booking a new ticket

```
def new_ticket(self):
        self.res_no = Label(self.left, text="Reservation No.",
font=('Cocogoose_Light 15'), fg='white', bg='grey')
        self.res_no.place(x=1, y=100)

        self.train_no = Label(self.left, text="Train No.", font=('Cocogoose_Light
15'), fg='white', bg='grey')
        self.train_no.place(x=1, y=130)

        self.seat_no = Label(self.left, text="Seat No.", font=('Cocogoose_Light
15'), fg='white', bg='grey')
        self.seat_no.place(x=1, y=160)

        self.tt = Label(self.left, text="Ticket Type", font=('Cocogoose_Light
15'), fg='white', bg='grey')
        self.tt.place(x=1, y=190)

        self.time = Label(self.left, text="Time of arrival",
font=('Cocogoose_Light 15'), fg='white', bg='grey')
        self.time.place(x=1, y=220)

        self.name = Label(self.left, text="Name", font=('Cocogoose_Light 15'),
fg='white', bg='grey')
        self.name.place(x=1, y=250)

        self.pp_no = Label(self.left, text="Passport No.", font=('Cocogoose_Light
15'), fg='white', bg='grey')
        self.pp_no.place(x=1, y=280)

        self.From = Label(self.left, text="From", font=('Cocogoose_Light 15'),
fg='white', bg='grey')
        self.From.place(x=1, y=310)

        self.to = Label(self.left, text="To", font=('Cocogoose_Light 15'),
fg='white', bg='grey')
        self.to.place(x=1, y=340)

        #entries for all labels----------------------------------------
        #reservation no.
        self.res_ent = Entry(self.left, width=40)
        self.res_ent.place(x=150, y=105) #y increases by 30
        #train no.
        self.train_ent = Entry(self.left, width=40)
        self.train_ent.place(x=150, y=135)
        #seat no.
        self.seat_ent = Entry(self.left, width=40)
        self.seat_ent.place(x=150, y=165)
        #ticket type
        self.tt_ent = Entry(self.left, width=40)
        self.tt_ent.place(x=150, y=195)
        #time of arrival
        self.toa_ent = Entry(self.left, width=40)
        self.toa_ent.place(x=150, y=225)
        #name
        self.name_ent = Entry(self.left, width=40)
        self.name_ent.place(x=150, y=255)
        #passport no.
        self.pp_ent = Entry(self.left, width=40)
        self.pp_ent.place(x=150, y=285)
        #from
        self.from_ent = Entry(self.left, width=40)
        self.from_ent.place(x=150, y=315)
        #to
        self.to_ent = Entry(self.left, width=40)
        self.to_ent.place(x=150, y=345)

        #button to perform the command
        self.submit = Button(self.left, text="Book Ticket", width=15, height=1,
bg='black', fg='grey', font=('Cocogoose 10'), command= self.add_booking)
        self.submit.place(x=250,y=380)

        #function to call when button is pressed
    def add_booking(self):
        #take all inputs
        #check if seats are available
            #if available, book seat and mention price
            #if not available, display message saying choose another flight

        self.val1 = self.res_ent.get()
        self.val2 = self.train_ent.get()
        self.val3 = self.seat_ent.get()
        self.val4 = self.toa_ent.get()
        self.val5 = self.name_ent.get()
        self.val6 = self.pp_ent.get()
        self.val7 = self.from_ent.get()
        self.val8 = self.to_ent.get()
        self.val9 = self.tt_ent.get()

        #check if user input is empty
        if self.val1=='' or self.val2=='' or self.val3=='' or self.val4=='' or
self.val5=='' or self.val6=='' or self.val7=='' or self.val8=='' or self.val9=='':
            tkMessageBox.showinfo("Warning", "Please fill all the fields.")
        else:
            sql = "INSERT INTO passenger (Reservation_no, Train_no, Seat_no,
Ticket_type, Time_of_departure, Name, Passport_no, Source, Destination) VALUES(%s,
%s, %s, %s, %s, %s, %s, %s, %s)"
            c.execute(sql, (self.val1, self.val2, self.val3, self.val9, self.val4,
self.val5, self.val6, self.val7, self.val8))
            conn.commit()
            tkMessageBox.showinfo("Success", "Reservation made for " +
str(self.val5))
```

Section 4.3

Searching for a booking

```
def search(self):
        #enter res number to search table
        self.to = Label(self.left, text="SEARCH: Enter the Reservation Number",
font=('Cocogoose 15'), fg='white', bg='grey')
        self.to.place(x=5, y=415)
        #entry box
        self.res_ent = Entry(self.left, width=45)
        self.res_ent.place(x=5, y=450)
        #search button that leads you to the next function that does the actual
searching
        self.srch = Button(self.left, text="Search", width=15, height=1,
bg='black', fg='grey', font=('Cocogoose 10'), command= self.actual_search)
        self.srch.place(x=5, y=480)

    def actual_search(self):
        val = self.res_ent.get() #res no is stored in val
        #now you search if the table has the res no and then display details
        query1='SELECT * FROM passenger WHERE Reservation_no=%s'
        c.execute(query1, val)
        data=c.fetchall()
        #print(data)
        if data != []:
            tkMessageBox.showinfo("Reservation Details", data)
        else:
            tkMessageBox.showinfo("Error", "Reservation number does not exist in
the records.")
```

Section 4.4

Cancelling a booking

```
def cancel(self):
        #ask for what you wanna cancel
        #delete the entry in passenger
        #update the number of seats
        self.to = Label(self.left, text="CANCELLATION: Enter the Reservation
Number", font=('Cocogoose 15'), fg='white', bg='grey')
        self.to.place(x=5, y=515)
        #entry box
        self.res_ent = Entry(self.left, width=45)
        self.res_ent.place(x=5, y=550)
        #button
        self.sc = Button(self.left, text="Search and Cancel", width=15, height=1,
bg='black', fg='grey', font=('Cocogoose 10'), command= self.actual_cancel)
        self.sc.place(x=5, y=580)

    def actual_cancel(self):
        val = self.res_ent.get() #res no is stored in val
        #first you increase the seat count



        #now you remove entry from passenger
        query2='DELETE FROM passenger WHERE Reservation_no=%s'
        c.execute(query2, val)
        conn.commit()
        tkMessageBox.showinfo("Alert","Reservation Cancelled")
#creating the object
root = Tk()
b= Application(root)
#resolution of the window
root.geometry("1200x720+0+0")

#preventing the resize feature
root.resizable(False, False)

#end of loop
root.mainloop()
```

Section 4.5

Trigger and Stored Procedure

Section 4.5.1

Trigger

```
     trig = "create trigger valFilled after insert on passenger for each row
update schedules set No_Booked=No_Booked+1 where
passengers.Train_no=schedules.Train_no"
      c.execute(trig)
```

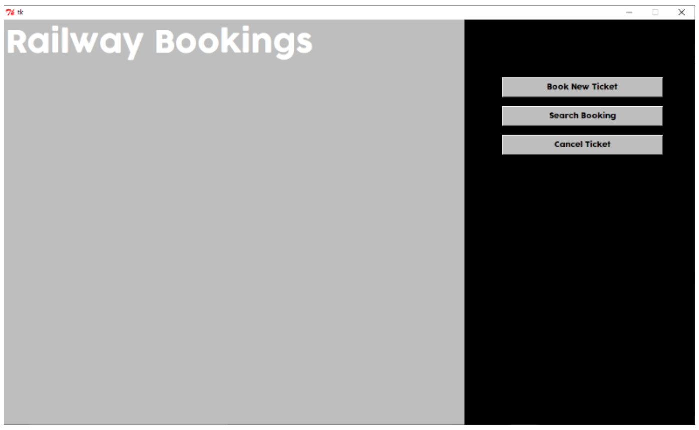
Section 4.4.2

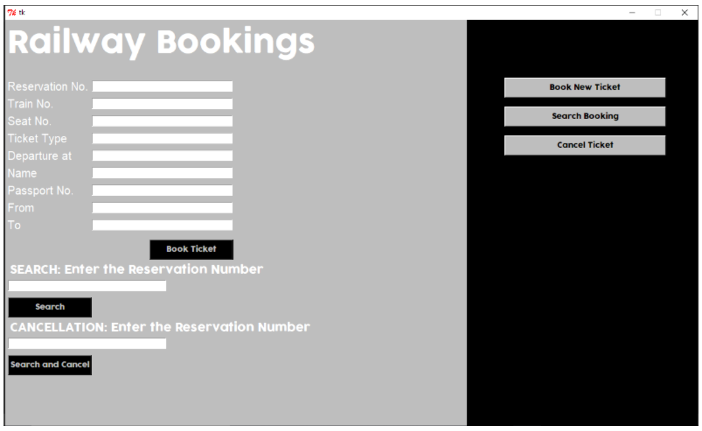
Stored Procedure

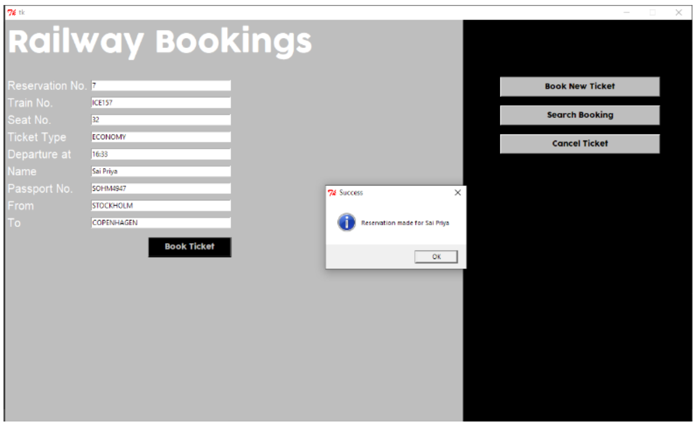
```
query1='SELECT * FROM passenger WHERE Reservation_no=%s'

query2='DELETE FROM passenger WHERE Reservation_no=%s'
```

Chapter 5

SCREENSHOTS

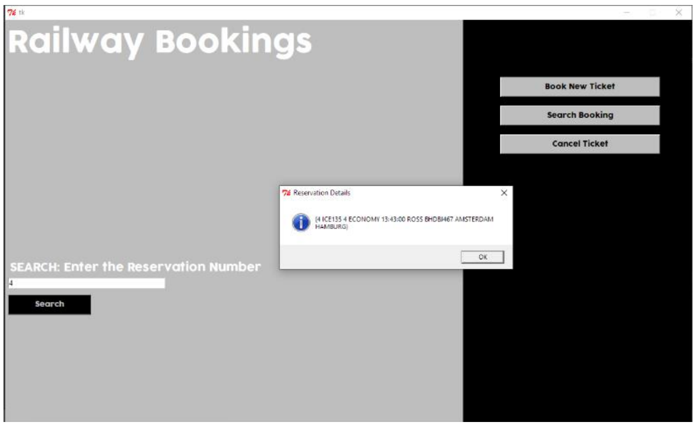






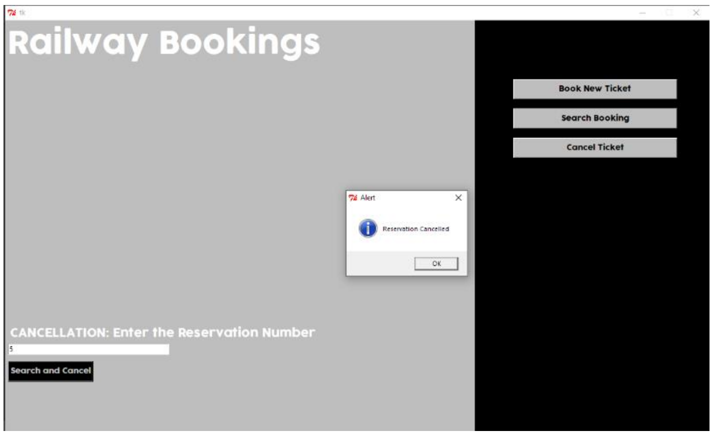
Reservation No. 7
Train No. ICE157
Seat No. 32
Ticket Type ECONOMY
Departure at 16:33
Name Sai Priya
Passport No. SOHM4947
From STOCKHOLM
To COPENHAGEN

Reservation made for Sai Priya



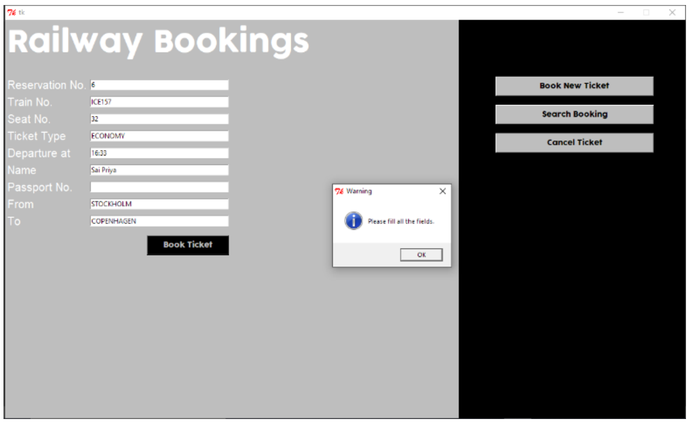
4

(4 ICE135 4 ECONOMY 13:43:00 ROSS BHDBH467 AMSTERDAM HAMBURG)



5

Reservation Cancelled



Reservation No. 6
Train No. ICE157
Seat No. 32
Ticket Type ECONOMY
Departure at 16:33
Name Sai Priya
Passport No.
From STOCKHOLM
To COPENHAGEN

Please fill all the fields.

Chapter 6

CONCLUSION AND FUTURE SCOPE

Section 6.1
Conclusion

As mentioned earlier in the introduction this project perfectly executes the smooth functioning of the railway reservation process and provides a very smooth user interface for the user to go through their reservation over the web.

Section 6.2
Future Scope

While the user interface is very easy to understand, it takes a toll on the complexity of the code itself. Future implementations of the system may include payment information for each reservation, bulk reservations, display of multiple routes and a display of the schedule for people to make reservations based on the same. For a rudimentary implementation, the current application does suffice but there is always room for innovation so as to constantly improve the user experience.

REFERENCES

- **www.youtube.com**
- **www.stackoverflow.com**
- **www.sqltutorial.com**