# PROJECT DOCUMENTATION

**Smart Solution for Railways**

**TEAM ID : PNT2022TMID16033**

**Project Report**

1. **INTRODUCTION**
   1. Project Overview
   2. Purpose
2. **LITERATURE SURVEY**
   1. Existing problem
   2. References
   3. Problem Statement Definition
3. **IDEATION & PROPOSED SOLUTION**
   1. Empathy Map Canvas
   2. Ideation & Brainstorming
   3. Proposed Solution
   4. Problem Solution fit
4. **REQUIREMENT ANALYSIS**
   1. Functional requirement
   2. Non-Functional requirements
5. **PROJECT DESIGN**
   1. Data Flow Diagrams
   2. Solution & Technical Architecture
   3. User Stories
6. **PROJECT PLANNING & SCHEDULING**
   1. Sprint Planning & Estimation
   2. Sprint Delivery Schedule
   3. Reports from JIRA
7. **CODING & SOLUTIONING (Explain the features added in the project along with code)**
   1. Feature 1
   2. Feature 2
   3. Database Schema (if Applicable)
8. **TESTING**
   1. Test Cases
   2. User Acceptance Testing
9. **RESULTS**
   1. Performance Metrics
10. **ADVANTAGES & DISADVANTAGES**
11. **CONCLUSION**
12. **FUTURE SCOPE**
13. **APPENDIX**

Source Code GitHub Link

# INTRODUCTION

## PROJECT OVERVIEW

The "QUEUE" when purchasing our tickets for the suburban railway is one of the greatest problems with the current ticketing facilities. With just a smartphone application, you may purchase our augmented reality ticket and carry your suburban railway tickets on your phone as QR (Quick Response) codes. Once the user arrives at their destination, it uses the "GPS" capability of their smart phone to automatically validate and destroy their ticket after a set amount of time. For security reasons, user ticket information is kept in a CLOUD database instead of the current suburban system. Additionally, the ticket checker is given access to a checker program that allows them to look up the user's ticket in the cloud database using the ticket number for checking reasons.

In this project prototype is used to book tickets from the web UI of the railways with some customized feature for users by booking tickets for them. By booking ticket from the web UI users can get special ID for their specific purpose and also creates QR code for the checking purpose and to be easier in checking the originality of the tickets. After booking they get unique ID and QR code. From the ticket checker point they may get a special login from the web-UI and they get a QR code scanner to check the tickets, By scanning the QR code they get the details of the passenger directly from the cloud IoT server. This is very useful in checking the ticket originality and to confirm the passenger’s details for future and security purpose.

## PURPOSE

The Internet is essentially a network of computers that are connected. However, as the world changes, its use is expanding beyond just email and web browsing. The creation of smart homes, smart rural communities, and e-health are all products of today's internet, which also deals with embedded sensors.

The idea of IoT was introduced by care's etc. Without human-to-human or human-to- computer interaction, the Internet of Things refers to the connection or communication between two or more devices. With the use of sensors or actuators, connected devices may sense their environment. Sensing the device, gaining access to the device, processing the device's data, and offering applications and services make up the four main parts of IOT. Along with this, it also offers data security and privacy. All facets of our daily life have been impacted by automation. In order to save time and reduce human effort, more advancements are being made in practically every industry. The same is being considered while attempting to automate track testing. Railroad track is a crucial component of every company's asset base since it enables them to conduct business as usual. Problems brought on by issues with railroads must be solved. The Indian railroad's most recent technique involves following the train tracks, which takes a lot of time and labor.

# LITERATURE SURVEY

## EXISTING SYSTEM

Tommy Kuncara et al.,(2021) presents the hardest tickets to purchase using the QR e-ticket technology. Only a smartphone can be used to purchase this bus pass ticket, and users can carry their smart phone bus pass tickets as QR (Quick Response). Customers who want to purchase the pass can do so by indicating the source and destination. Mrs. Om Prakash Yadav et al.,(2020) presents that today PRS is available at 8000 counters morethan 2380 locations throughout the countries, including all major stations, and important non- railhead locations such as tourist centers and district headquarters. The PRS services are available to passengers for 23 hours in a day. Karthick et al.,(2012) presents Using Java, SQLite, MySQL, and PHP on the server side, we designed a mobile ticket application for Android 1.5 that could alter how users purchase tickets in the future. Any type of transportation system can use this form of ticketing program. Our Android software is unique in its class and has become a very popular way to purchase tickets for suburban trains using an Android smartphone. Man Mohan Swarup et al.,(2012) presents that in addition to all of these services offered by Indian Railway, an effort was made to cut down on paper use as much as possible. Passengers are requested to use their Mobile Phones as journey tickets, because the ticket is stored in Mobile Phone either as an SMS sent by IRCTC or in the PDF format store in memory card in the case of e-ticket. Ankita Bhander et al.,(2016) presents the A QR code is any code that users find on most of any items that they buy from the store. QR codes have come a long way and now that they are integrated into the online world it’s a true phenomenon. It makes searching for online products, shopping and buying much easier. Now, users are going to use it for buying tickets. Createsan image in real world and acts like a web link for the smart phones.

## REFERENCES

* + 1. Kuncara, T., Putra, A. S., Aisyah, N., & Valentino, V. H. (2021). Effectiveness of the E-Ticket System Using QR Codes For Smart Transportation Systems. *International Journal of Science, Technology & Management*, *2*(3), 900-907.
    2. Yadav, M. O., Fernandes, R., Tiwari, R., & Kaul, S. (2014). Online reservation system using qr code based android application system. *International Journal of Scientific and Research Publications*, *4*(1–2).
    3. Karthick, S., & Velmurugan, A. (2012, August). Android suburban railway ticketing with GPS as ticket checker. In 2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT) (pp. 63-66). IEEE.
    4. Swarup, M. M., Dwivedi, A., Sonkar, C., Prasad, R., Bag, M., & Singh, V. (2012). A QR code based processing for dynamic and transparent seat allocation in Indian railway. International Journal of Computer Science Issues (IJCSI), 9(3), 338.
    5. Thombare, S., Kulkarni, T., Ghuge, K., & Bhadkumbhe, S. M. Android Railway Ticketing with GPS as Ticket Checker.

## PROBLEM STATEMENT DEFINITION

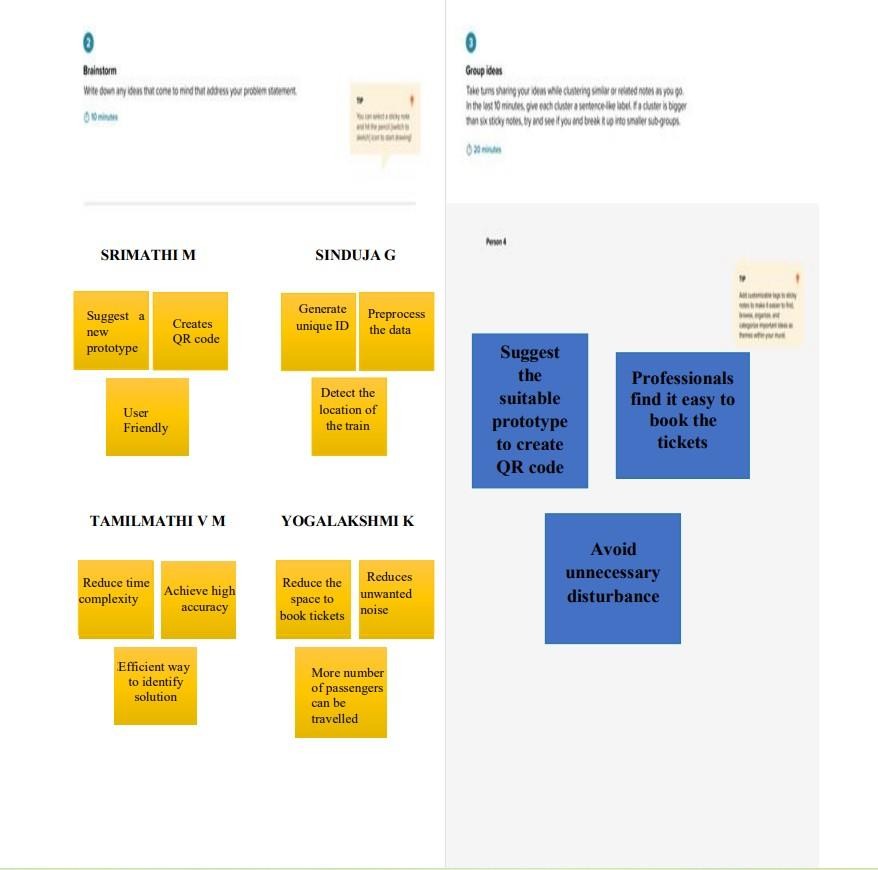
Railways are one of the world's major modes of transportation among the many ones. The issue statement is intended to provide an answer to the question "What are the problems faced by the passengers while travelling by rail at the station and on board?" despite competition from airlines, upscale buses, public transportation, and customised transportation. It has been deduced that every previous study on the subject either uses a system where the tools necessary to gather real-time data, like cameras and sensors, are all positioned outside of the train tracks and train detection. Systems that have been proposed, created, and put into use thus far scarcely allow us to detect crack. There are very few crack detection systems that employ techniques like breaking intensity detection and networking, among others. Despite the fact that they may have been innovators, the rail industry has just begun to scratch the surface of what is feasible. The information offered to operators is becoming more standardised, open, and scalable as IoT develops. Operators get knowledge of how their assets are functioning, which creates several new opportunities to use big data in more innovative and efficient ways. But when it comes to deploying IoT systems, the fact that trains move at such high speeds through tunnels and adverse weather conditions creates significant hurdles.

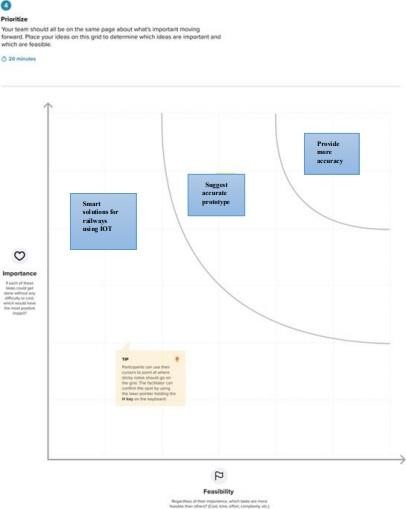
# IDEATION AND PROPOSED SOLUTION

## EMPATHY MAP CANVAS

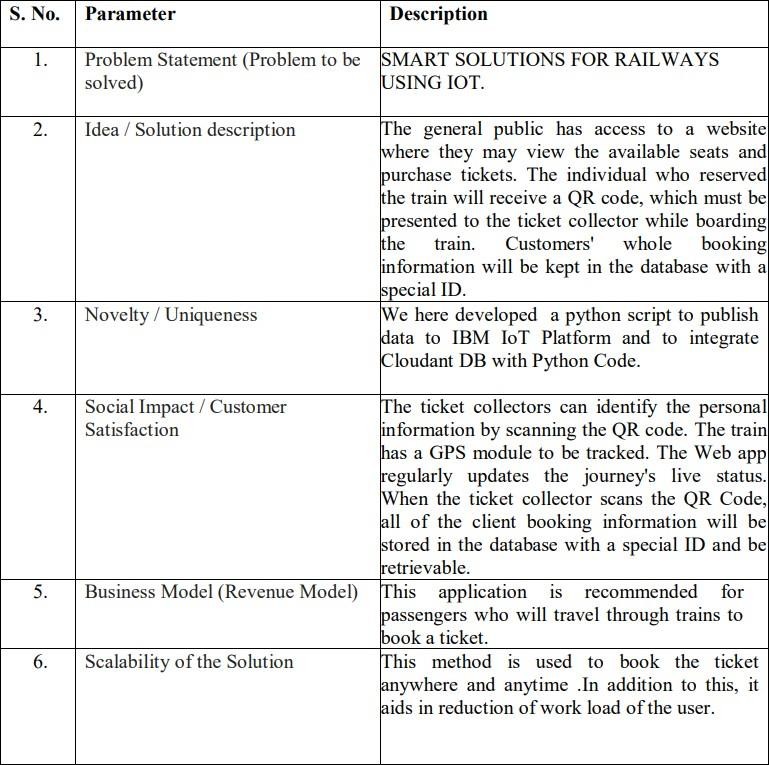


* 1. **IDEATION AND BRAINSTORMING**

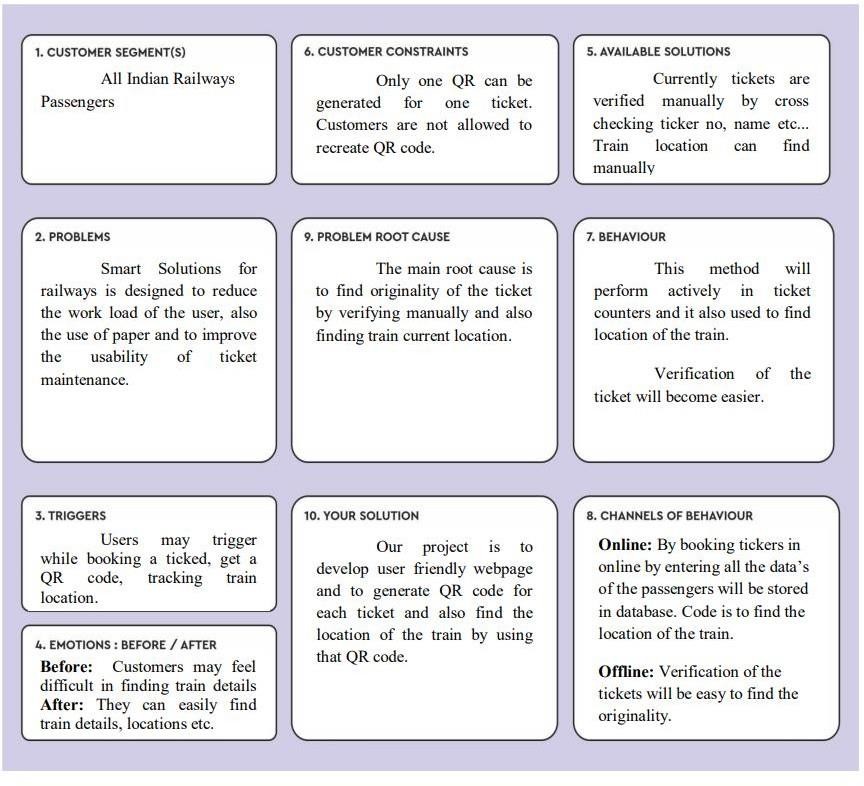




## PROPOSED SOLUTION



* 1. **PROBLEM SOLUTION FIT**



# REQUIREMENT ANALYSIS

## FUNCTIONAL REQUIREMENTS

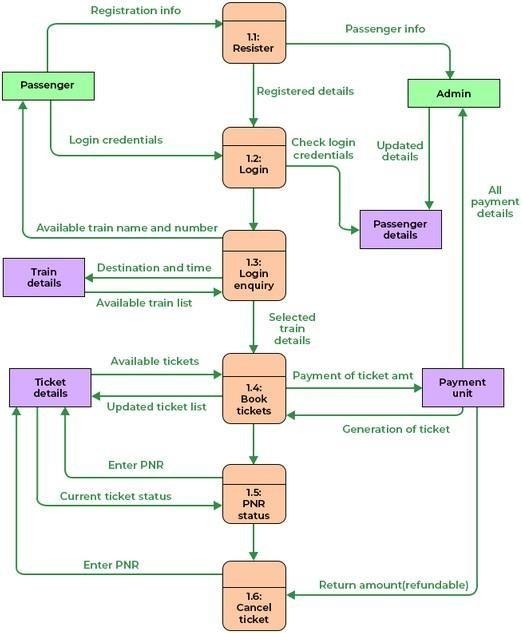
|  |  |  |
| --- | --- | --- |
| **FR**  **No.** | **Functional Requirement**  **(Epic)** | **Sub Requirement (Story / Sub-Task)** |
| FR-1 | Unique accounts | * Every online booking needs to be associated with an account * One account cannot be associated with multiple users |
| FR-2 | Booking options | * Search results should enable users to find the most recent and relevant booking options |
| FR-3 | Mandatory fields | * System should only allow users to move to payment only when mandatory fields such as date, time, location has been mentioned |
| FR-4 | Synchronization | * System should consider time zone synchronization when accepting bookings from different time zones |
| FR-5 | Authentication | * Booking confirmation should be sent to user to the specified contact details |

* 1. **NON – FUNCTIONAL REQUIREMENTS**

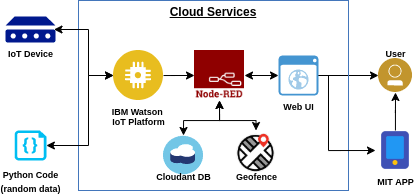
|  |  |  |
| --- | --- | --- |
| **FR**  **No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | * Search results should populate within acceptable time limits |
| NFR-2 | **Security** | * System should visually confirm as well as send booking confirmation   to the user's contact |
| NFR-3 | **Reliability** | * System should accept payments via different payment methods, like   PayPal, wallets, cards, vouchers, etc |
| NFR-4 | **Performance** | * Search results should populate within acceptable time limits |
| NFR-5 | **Availability** | * User should be helped appropriately to fill in the mandatory fields, incase   of invalid input |
| NFR-6 | **Scalability** | * Use of captcha and encryption to avoid bots from booking tickets |

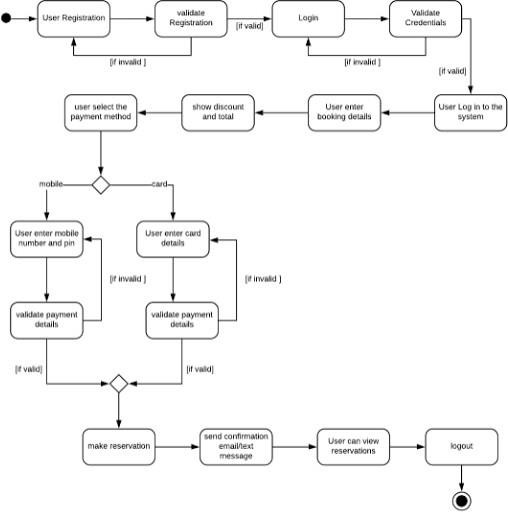
## PROJECT DESIGN

* 1. **DATA FLOW DIAGRAM**



## SOLUTION AND TECHNICAL ARCHITECTURE





* 1. **USER STORIES**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Acceptance criteria** | **Priority** | **Release** |
| Customer  (Mobile user, Web user) | Registration | USN-1 | As a user, I can register through the form by Filling in my details | I can register and create my account / dashboard | High | Sprint-1 |
|  |  | USN-2 | As a user, I can register through phone numbers, Gmail,  Facebook or other social sites | I can register & create my dashboard with  Facebook login or other social sites | High | Sprint-2 |
|  | Conformation | USN-3 | As a user, I will receive confirmation through email or OTP once registration is successful | I can receive confirmation email & click confirm. | High | Sprint-1 |
|  | Authentication/Login | USN-4 | As a user, I can login via login id and password or through OTP received on register phone number | I can login and access my account/dashboard | High | Sprint-1 |
|  | Display Train details | USN-5 | As a user, I can enter the start and destination to get the list of trains available connecting the above | I can view the train details  (name & number), corresponding routes it passes through based on the start and  destination entered. | High | Sprint-1 |
|  | Booking | USN-6 | As a use, I can provide the basic details such  as a name, age, gender etc… | I will view, modify or confirm the details enter. | High | Sprint-1 |
|  |  | USN-7 | As a user, I can choose the class, seat/berth.  If a preferred seat/berth isn’t available I can be allocated based on the availability. | I will view, modify or confirm the seat/class berth selected | High | Sprint-1 |
|  | Payment | USN-8 | As a user, I can choose to pay through credit Card/debit card/UPI. | I can view the payment  Options available and select my desirable choice To proceed with the payment | High | Sprint-1 |
|  |  | USN-9 | As a user, I will be redirected to the selected  Payment gateway and upon successful | I can pay through the payment portal and confirm the booking if any changes need to | High | Sprint-1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Acceptance criteria** | **Priority** | **Release** |
|  |  |  | completion of payment I’ll be redirected to the booking website. | be done  I can move back to the initial payment page |  |  |
|  | Ticket generation | USN-10 | As a user, I can download the generated  e-ticket for my journey along with the QR code which is used for authentication during my journey. | I can show the generated QR code so that authentication can be done quickly. | High | Sprint-1 |
|  | Ticket status | USN-11 | As a user, I can see the status of my ticket Whether it’s confirmed/waiting/RAC. | I can confidentially get the Information and arrange alternate transport if the ticket isn’t Confirmed | High | Sprint-1 |
|  | Remainders notification | USN-12 | As a user, I get remainders about my journey  A day before my actual journey. | I can make sure that I don’t miss the journey because of the constant notifications. | Medium | Sprint-2 |
|  |  | USN-13 | As a user, I can track the train using GPS and can get information such as ETA,  Current stop and delay. | I can track the train and get to know about the delays pian accordingly | Medium | Sprint-2 |
|  | Ticket cancellation | USN-14 | As a user, I can cancel my tickets if there’s any Change of plan | I can cancel the ticket and get a refund based on how close the date is to the journey. | High | Sprint-1 |
|  | Raise queries | USN-15 | As a user, I can raise queries through the query box or via mail. | I can view my pervious queries. | Low | Sprint-2 |
| Customer care Executive | Answer the queries | USN-16 | As a user, I will answer the questions/doubts Raised by the customers. | I can view the queries and make it once resolved | Medium | Sprint-2 |
| Administrator | Feed details | USN-17 | As a user, I will feed information about the trains delays and add extra seats if a new compartment is added. | I can view and ensure the corrections of the information fed. | High | Sprint-1 |

# PROJECT PLANNING AND SCHEDULING

* 1. **SPRINT PLANNING AND ESTIMATION**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| Sprint-1 | Registration | USN-1 | As a user, I can register through the form by  Filling in my details | 2 | High | Srimathi |
| Sprint-1 |  | USN-2 | As a user, I can register through phone  numbers, Gmail, Facebook or other social sites | 1 | High | Sinduja |
| Sprint-1 | Conformation | USN-3 | As a user, I will receive confirmation through  email or OTP once registration is successful | 2 | Low | Tamilmathi |
| Sprint-1 | login | USN-4 | As a user, I can login via login id and password  or through OTP received on register phone number | 2 | Medium | Yogalakshmi |
| Sprint-1 | Display Train details | USN-5 | As a user, I can enter the start and destination to get the list of trains available connecting the above | 1 | High | Srimathi |
| Sprint-2 | Booking | USN-6 | As a use, I can provide the basic details such as  a name, age, gender etc… | 2 | High | Sinduja |
| Sprint-2 |  | USN-7 | As a user, I can choose the class, seat/berth. If a preferred seat/berth isn’t available I can be  allocated based on the availability | 1 | Low | Yogalkshmi |
| Sprint-2 | Payment | USN-8 | As a user, I can choose to pay through credit  Card/debit card/UPI. | 1 | High | Tamilmathi |
| Sprint | Functional Requirement (Epic) | User Story Number | User Story / Task | Story Points | Priority | Team Members |
| Sprint-2 |  | USN-9 | As a user, I will be redirected to the selected | 2 | High | Tamilmathi |
| Sprint-3 | Ticket generation | USN-10 | As a user, I can download the generated e- ticket for my journey along with the QR code which is used for authentication during my journey. | 1 | High | Srimathi |
| Sprint-3 | Ticket status | USN-11 | As a user, I can see the status of my ticket | 2 | High | Sinduja |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Whether it’s confirmed/waiting/RAC. |  |  |  |
| Sprint-3 | Remainders notification | USN-12 | As a user, I get remainders about my journey A  day before my actual journey. | 1 | High | Srimathi |
| Sprint-3 | Ticket cancellation | USN-13 | As a user, I can track the train using GPS and can get information such as ETA,  Currentstop and delay | 2 | High | Tamilmathi |
| Sprint-4 |  | USN-14 | As a user, I can cancel my tickets if there’s any  Change of plan | 1 | High | Sinduja |
| Sprint-4 | Raise queries | USN-15 | As a user, I can raise queries through the query  box or via mail. | 2 | Medium | Yogalakshmi |
| Sprint-4 | Answer the queries | USN-16 | As a user, I will answer the questions/doubts Raised by the customers. | 2 | High | Sinduja |
| Sprint-4 | Feed details | USN-17 | As a user, I will feed information about the trains delays and add extra seats if a new  compartment is added. | 1 | High | Srimathi |

* 1. **SPRINT DELIVERY SCHEDULE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Total Story Point s** | **Duratio n** | **Sprint StartDate** | **Sprint EndDate (Planned)** | **Story Points Completed (ason Planned End Date)** | **Sprint Release Date(Actual)** |
| Sprint-1 | 20 | 6 Days | 24 Oct 2022 | 29 Oct 2022 | 20 | 29 Oct 2022 |
| Sprint-2 | 20 | 6 Days | 31 Oct 2022 | 05 Nov 2022 | 20 | 5 Nov 2022 |
| Sprint | Total Story Points | Duration | Sprint Start Date | Sprint End Date(Planned) | Story Points Completed (as onPlanned End Date) | Sprint Release Date(Actual) |
| Sprint-3 | 20 | 6 Days | 07 Nov 2022 | 12 Nov 2022 | 20 | 12 Nov 2022 |
| Sprint-4 | 20 | 6 Days | 14 Nov 2022 | 19 Nov 2022 | 20 | 19 Nov2022 |

# CODING AND SOLUTIONING

* 1. **FEATURE 1**
     + IoT Device
     + IBM Watson platform
     + Node Red
     + Cloudant DB
     + Web UI
     + Geofence
     + MIT App
     + Python Code

# FEATURE 2

* + - Registration
    - Login
    - Verification
    - Ticket Booking
    - Payment
    - Ticket Cancellation
    - Adding Queries

labl\_0 = Label(base, text="Registration form", width=20,font=("bold", 20)) labl\_0.place(x=90,y=53)

lb1= Label(base, text="Enter Name", width=10, font=("arial",12)) lb1.place(x=20, y=120)

en1= Entry(base) en1.place(x=200, y=120)

lb3= Label(base, text="Enter Email", width=10, font=("arial",12)) lb3.place(x=19, y=160)

en3= Entry(base) en3.place(x=200, y=160)

lb4= Label(base, text="Contact Number", width=13,font=("arial",12)) lb4.place(x=19, y=200)

en4= Entry(base) en4.place(x=200, y=200)

lb5= Label(base, text="Select Gender", width=15, font=("arial",12)) lb5.place(x=5, y=240)

var = Int Var()

Radio button(base, text="Male", pad x=5,variable=var, value=1).place(x=180, y=240)

Radio button(base, text="Female", pad x =10,variable=var, value=2).place(x=240,y=240)

Radio button(base, text="others", pad x=15, variable=var, value=3).place(x=310,y=240)

list\_of\_cntry = ("United States", "India", "Nepal", "Germany") cv = String Var() drplist= Option Menu(base, cv, \*list of\_cntry) drplist.config(width=15) cv.set("United States")

lb2= Label(base, text="Select Country", width=13,font=("arial",12)) lb2.place(x=14,y=280)

drplist.place(x=200, y=275)

lb6= Label(base, text="Enter Password", width=13,font=("arial",12)) lb6.place(x=19, y=320)

en6= Entry(base, show='\*') en6.place(x=200, y=320)

lb7= Label(base, text="Re-Enter Password", width=15,font=("arial",12)) lb7.place(x=21, y=360)

en7 =Entry(base, show='\*') en7.place(x=200, y=360)

Button(base, text="Register", width=10).place(x=200,y=400) base.mainloop() def generateOTP() :

# Declare a digits variable # which stores all digits digits = "0123456789" OTP = ""

# length of password can be changed # by changing value in range for i in range(4) :

OTP += digits[math.floor(random.random() \* 10)] return OTP # Driver code

if name == " main " :

print("OTP of 4 digits:", generateOTP()) digits="0123456789"

OTP=""

for i in range(6): OTP+=digits[math.floor(random.random()\*10)] otp = OTP + " is your OTP" msg= otp

s = smtplib.SMTP('smtp.gmail.com', 587) s.starttls()

s.login("Your Gmail Account", "You app password") email id = input("Enter your email: ")

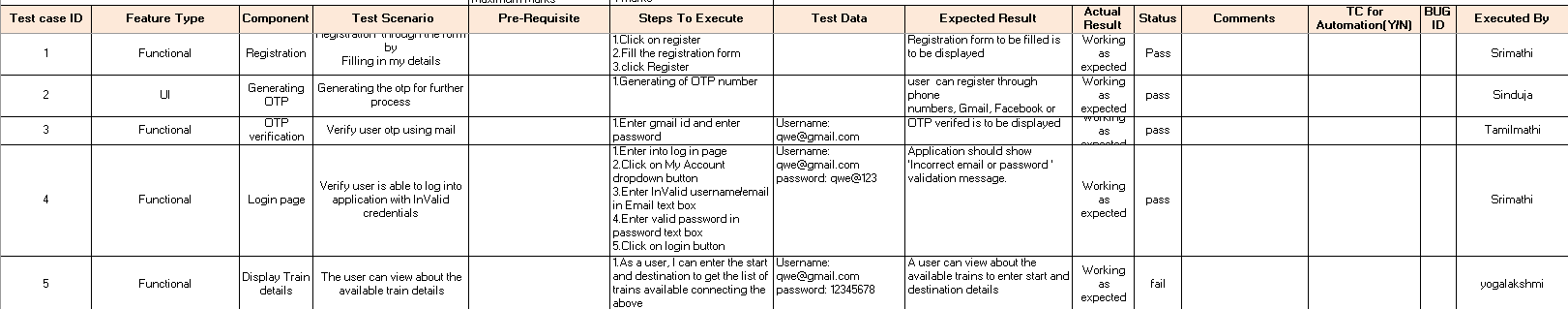
s.sendmail('&&&&&&&&&&&',emailid,msg) a = input("Enter Your OTP >>: ") if a == OTP: print("Verified")

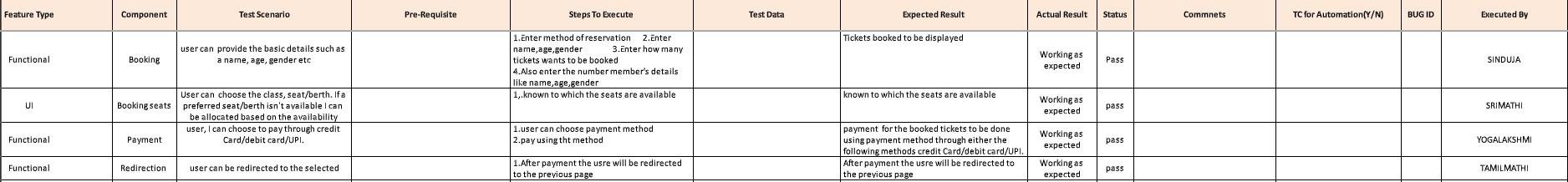
else:

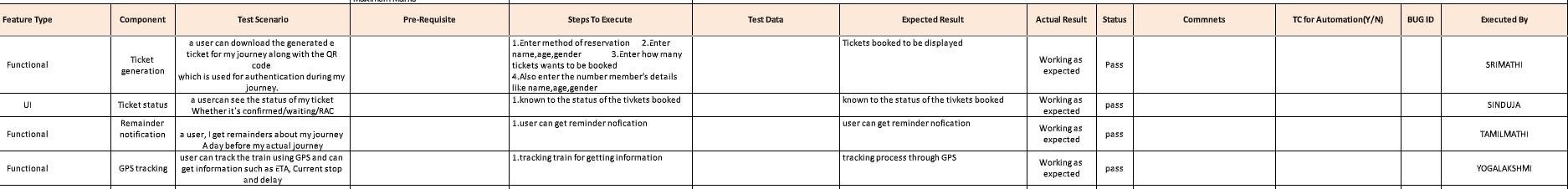
print("Please Check your OTP again") roo

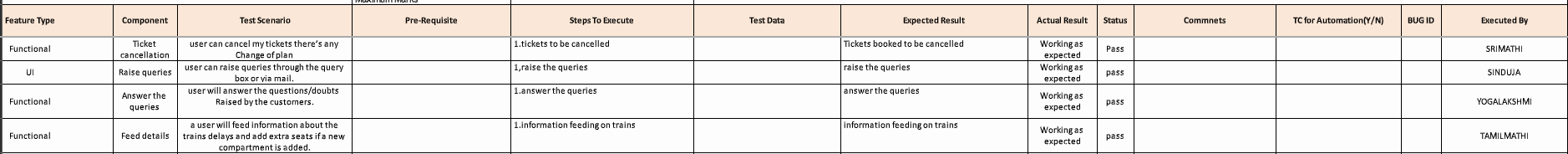
# TESTING

**8.1 TEST CASES**









# RESULTS

* 1. **PERFORMANCE METRICES**



# ADVANTAGES AND DISADVANTAGES

## ADVANTAGES

* + - Openness – compatibility between different system modules, potentially from different vendors;
    - Orchestration – ability to manage large numbers of devices, with full visibility over them;
    - Dynamic scaling – ability to scale the system according to the application needs, through resource virtualization and cloud operation;
    - Automation – ability to automate parts of the system monitoring application, leading to better performance and lower operation costs.

## DISADVANTAGES

* + - Approaches to flexible, effective, efficient, and low-costdata collection for both railway vehicles and infrastructure monitoring, using regular trains;
    - Data processing, reduction, and analysis in local controllers, and subsequent sending of that data to the cloud, for further processing;
    - Online data processing systems, for real-time monitoring, using emerging communication technologies;
    - Integrated, interoperable, and scalable solutions for railway systems preventive maintenance.

# CONCLUSION

A significant number of lives are lost as a result of accidents in the rail transportation system. Thus, this system aids in the prevention of accidents by informing the railroad authorities in advance of any faults or cracks. so that they can be fixed and the number of accidents decreases. This undertaking is economical. They can be improved and enhanced in accordance with their applications by utilizing more strategies. By preventing accidents, this technology can save many lives. Long-term large-scale implementation of the concept is possible to support improved rail track safety requirements and offer efficient testing infrastructure for improved outcomes in the future.

# FUTURE SCOPES

In future CCTV systems with IP based camera can be used for monitoring the visual videos captured from the track. It will also increase security for both passengers and railways. GPS can also be used to detect exact location of track fault area, IP cameras can also be used to show fault with the help of video. Locations on Google maps with the help of sensors can be used to detect in which area track is broken.

* 1. **SOURCE PROGRAM**

# APPENDIX

import math, random import os

import smtplib import sqlite3 import requests

from bs4 import BeautifulSoup

from django.contrib.auth.base\_user import AbstractBaseUser from django.db import models

import logging import pandas as pd import pyttsx3

from plyer import notification import time

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image, ImageDraw from pickle import load,dump import smtplib, ssl

from email.mime.text import MIMEText from email.mime.multipart

import MIMEMultipart import email

from email import encoders

from email.mime.base import MIMEBase import attr

from flask import Blueprint, flash, redirect, request, url\_for from flask.views import MethodView

from flask\_babelplus import gettext as \_

from flask\_login import current\_user, login\_required from pluggy import HookimplMarker

from tkinter import\* base = Tk() base.geometry("500x500") base.title("registration form")

labl\_0 = Label(base, text="Registration form",width=20,font=("bold", 20)) labl\_0.place(x=90,y=53)

lb1= Label(base, text="Enter Name", width=10, font=("arial",12)) lb1.place(x=20, y=120)

en1= Entry(base) en1.place(x=200, y=120)

lb3= Label(base, text="Enter Email", width=10, font=("arial",12)) lb3.place(x=19, y=160)

en3= Entry(base) en3.place(x=200, y=160)

lb4= Label(base, text="Contact Number", width=13,font=("arial",12)) lb4.place(x=19, y=200)

en4= Entry(base) en4.place(x=200, y=200)

lb5= Label(base, text="Select Gender", width=15, font=("arial",12)) lb5.place(x=5, y=240)

var = IntVar()

Radiobutton(base, text="Male", padx=5,variable=var, value=1).place(x=180, y=240)

Radiobutton(base, text="Female", padx =10,variable=var, value=2).place(x=240,y=240)

Radiobutton(base, text="others", padx=15, variable=var, value=3).place(x=310,y=240)

list\_of\_cntry = ("United States", "India", "Nepal", "Germany") cv = StringVar() drplist= OptionMenu(base, cv, \*list\_of\_cntry) drplist.config(width=15) cv.set("United States")

lb2= Label(base, text="Select Country", width=13,font=("arial",12)) lb2.place(x=14,y=280)

drplist.place(x=200, y=275)

lb6= Label(base, text="Enter Password", width=13,font=("arial",12)) lb6.place(x=19, y=320)

en6= Entry(base, show='\*') en6.place(x=200, y=320)

lb7= Label(base, text="Re-Enter Password", width=15,font=("arial",12)) lb7.place(x=21, y=360)

en7 =Entry(base, show='\*') en7.place(x=200, y=360)

Button(base, text="Register", width=10).place(x=200,y=400) base.mainloop() def generateOTP() :

# Declare a digits variable # which stores all digits digits = "0123456789" OTP

= ""

# length of password can be changed # by changing value in range for i in range(4) :

OTP += digits[math.floor(random.random() \* 10)] return OTP # Driver code

If name== "main" :

print("OTP of 4 digits:", generateOTP()) digits="0123456789" OTP=""

for i in range(6): OTP+=digits[math.floor(random.random()\*10)] otp = OTP + " is your OTP" msg= otp

s = smtplib.SMTP('smtp.gmail.com', 587) s.starttls()

s.login("Your Gmail Account", "You app password") emailid = input("Enter your email: ") s.sendmail('&&&&&&&&&&&',emailid,msg)

a = input("Enter Your OTP >>: ") if a == OTP: print("Verified") else:

print("Please Check your OTP again") root = Tk() root.title("Python: Simple Login Application") width = 400 height = 280

screen\_width = root.winfo\_screenwidth() screen\_height = root.winfo\_screenheight() x = (screen\_width/2) - (width/2)

y = (screen\_height/2) - (height/2) root.geometry("%dx%d+%d+%d" % (width, height, x, y)) root.resizable(0, 0)

USERNAME = StringVar() PASSWORD = StringVar()

Top = Frame(root, bd=2, relief=RIDGE) Top.pack(side=TOP, fill=X) Form = Frame(root, height=200) Form.pack(side=TOP, pady=20)

lbl\_title = Label(Top, text = "Python: Simple Login Application", font=('arial', 15))

lbl\_title.pack(fill=X)

lbl\_username = Label(Form, text = "Username:", font=('arial', 14), bd=15) lbl\_username.grid(row=0, sticky="e")

lbl\_password = Label(Form, text = "Password:", font=('arial', 14), bd=15) lbl\_password.grid(row=1, sticky="e") lbl\_text = Label(Form) lbl\_text.grid(row=2, columnspan=2)

username = Entry(Form, textvariable=USERNAME, font=(14)) username.grid(row=0, column=1)

password = Entry(Form, textvariable=PASSWORD, show="\*", font=(14)) password.grid(row=1, column=1) def Database():

global conn, cursor

conn = sqlite3.connect("pythontut.db") cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS `member` (mem\_id INTEGER NOT NULL PRIMARY KEY

AUTOINCREMENT, username TEXT, password TEXT)") cursor.execute("SELECT \* FROM `member` WHERE `username` =

'admin' AND `password` = 'admin'") if cursor.fetchone() is None:

cursor.execute("INSERT INTO `member` (username, password) VALUES('admin', 'admin')")

conn.commit()

def Login(event=None): Database()

if USERNAME.get() == "" or PASSWORD.get() == "":

lbl\_text.config(text="Please complete the required field!", fg="red") else: cursor.execute("SELECT \* FROM `member` WHERE `username`

= ? AND `password` = ?", (USERNAME.get(), PASSWORD.get())) if cursor.fetchone() is not None:

HomeWindow() USERNAME.set("") PASSWORD.set("")

lbl\_text.config(text="")

else:

lbl\_text.config(text="Invalid username or password", fg="red") USERNAME.set("")

PASSWORD.set("")

cursor.close() conn.close()

btn\_login = Button(Form, text="Login", width=45, command=Login) btn\_login.grid(pady=25, row=3, columnspan=2) btn\_login.bind('<Return>', Login)

def HomeWindow(): global Home root.withdraw() Home = Toplevel()

Home.title("Python: Simple Login Application") width = 600 height = 500

screen\_width = root.winfo\_screenwidth() screen\_height = root.winfo\_screenheight() x = (screen\_width/2) - (width/2)

y = (screen\_height/2) - (height/2) root.resizable(0, 0)

Home.geometry("%dx%d+%d+%d" % (width, height, x, y)) lbl\_home =

Label(Home, text="Successfully Login!", font=('times new roman', 20)).pack()

btn\_back = Button(Home, text='Back', command=Back).pack(pady=20, fill=X) def Back():

Home.destroy() root.deiconify()

def getdata(url):

r = requests.get(url) return r.text

# input by geek from\_Station\_code = "GAYA" from\_Station\_name = "GAYA"

To\_station\_code = "PNBE" To\_station\_name = "PATNA" # url url = "https:/[/www.railyatri.in/booking/trains-between-](http://www.railyatri.in/booking/trains-between-)

stations?from\_code="+from\_Station\_code+"&from\_name="+from\_Stat ion\_name+"+JN+&journey\_date=+Wed&src=tbs&to\_code=" + \

To\_station\_code+"&to\_name="+To\_station\_name + \ "+JN+&user\_id=- 1603228437&user\_token=355740&utm\_source=dwebsearch\_tbs\_search\_ trains"

# pass the url

# into getdata function htmldata = getdata(url) soup = BeautifulSoup(htmldata, 'html.parser') # find the Html tag # with find()

# and convert into string data\_str = ""

for item in soup.find\_all("div", class\_="col-xs-12 TrainSearchSection"): data\_str = data\_str + item.get\_text()

result = data\_str.split("\n")

print("Train between "+from\_Station\_name+" and "+To\_station\_name) print("")

# Display the result for item in result:

if item != "": print(item)

print("\n\nTicket Booking System\n") restart = ('Y') while restart != ('N','NO','n','no'):

print("1.Check PNR status") print("2.Ticket Reservation") option = int(input("\nEnter your option : "))

if option == 1:

print("Your PNR status is t3") exit(0)")) elif option == 2:

people = int(input("\nEnter no. of Ticket you want : name\_l = [] age\_l = [] sex\_l = []

for p in range(people):

name = str(input("\nName : ")) name\_l.append(name) age = int(input("\nAge : ")) age\_l.append(age)

sex = str(input("\nMale or Female : ")) sex\_l.append(sex) "))

restart = str(input("\nDid you forgot someone? y/n: if restart in ('y','YES','yes','Yes'): restart = ('Y')

else :

x = 0

print("\nTotal Ticket : ",people) for p in range(1,people+1): print("Ticket : ",p)

print("Name : ", name\_l[x])

print("Age : ", age\_l[x])

print("Sex : ",sex\_l[x]) x += 1

class User(AbstractBaseUser): """ User model. """ USERNAME\_FIELD = "email"

REQUIRED\_FIELDS = ["first\_name", "last\_name"] email = models.EmailField(

verbose\_name="E-mail", unique=True

)

first\_name = models.CharField( verbose\_name="First name", max\_length=30

)

last\_name = models.CharField( verbose\_name="Last name", max\_length=40

)

city = models.CharField( verbose\_name="City", max\_length=40

)

stripe\_id = models.CharField( verbose\_name="Stripe ID", unique=True, max\_length=50, blank=True,

null=True

)

objects = UserManager() @property def get\_full\_name(self):

return f"{self.first\_name} {self.last\_name}" class Meta:

verbose\_name = "User" verbose\_name\_plural = "Users" class Profile(models.Model): """

User's profile. """

phone\_number = models.CharField( verbose\_name="Phone number", max\_length=15)

date\_of\_birth = models.DateField( verbose\_name="Date of birth"

)

postal\_code = models.CharField( verbose\_name="Postal code", max\_length=10,

blank=True

)

address = models.CharField( verbose\_name="Address", max\_length=255, blank=True

)

class Meta: abstract = True class UserProfile(Profile): """ User's profile model. """

user = models.OneToOneField(

to=User, on\_delete=models.CASCADE, related\_name="profile",

)

group = models.CharField( verbose\_name="Group type", choices=GroupTypeChoices.choices(), max\_length=20, default=GroupTypeChoices.EMPLOYEE.name,

)

Def str(self): return self.user.email class Meta:

# user 1 - employer

user1, \_ = User.objects.get\_or\_create( email="[foo@bar.com](mailto:foo@bar.com)", first\_name="Employer", last\_name="Testowy", city="Białystok",

)

user1.set\_unusable\_password() group\_name = "employer"

\_profile1, \_ = UserProfile.objects.get\_or\_create( user=user1,

date\_of\_birth=datetime.now() - timedelta(days=6600), group=GroupTypeChoices(group\_name).name, address="Myśliwska 14",

postal\_code="15-569", phone\_number="+48100200300",

)

# user2 - employee

user2, \_ = User.objects.get\_or\_create() email="[bar@foo.com](mailto:bar@foo.com)", first\_name="Employee", last\_name="Testowy", city="Białystok",

)

user2.set\_unusable\_password() group\_name = "employee"

\_profile2, \_ = UserProfile.objects.get\_or\_create() user=user2,

date\_of\_birth=datetime.now() - timedelta(days=7600), group=GroupTypeChoices(group\_name).name, address="Myśliwska 14",

postal\_code="15-569", phone\_number="+48200300400",

)

response\_customer = stripe.Customer.create() email=user.email,

description=f"EMPLOYER - {user.get\_full\_name}", name=user.get\_full\_name, phone=user.profile.phone\_number,

)

user1.stripe\_id = response\_customer.stripe\_id user1.save()

mcc\_code, url = "1520", "https:/[/www](http://www.softserveinc.com/).[softserveinc.com/](http://www.softserveinc.com/)" response\_ca = stripe.Account.create()

type="custom",

country="PL", email=user2.email, default\_currency="pln", business\_type="individual",

settings={"payouts": {"schedule": {"interval": "manual", }}}, requested\_capabilities=["card\_payments", "transfers", ], business\_profile={"mcc": mcc\_code, "url": url},

individual={

"first\_name": user2.first\_name, "last\_name": user2.last\_name, "email": user2.email,

"dob": {

"day": user2.profile.date\_of\_birth.day, "month": user2.profile.date\_of\_birth.month, "year": user2.profile.date\_of\_birth.year,

},

"phone": user2.profile.phone\_number, "address": {

"city": user2.city,

"postal\_code": user2.profile.postal\_code, "country": "PL", "line1": user2.profile.address,

},

},

)

user2.stripe\_id = response\_ca.stripe\_id user2.save() tos\_acceptance = {"date": int(time.time()), "ip": user\_ip},

stripe.Account.modify(user2.stripe\_id, tos\_acceptance=tos\_acceptance) passport\_front = stripe.File.create(

purpose="identity\_document", file=\_file, # ContentFile object stripe\_account=user2.stripe\_id,

)

individual = { "verification": {

"document": {"front": passport\_front.get("id"),}, "additional\_document":

{"front": passport\_front.get("id"),},

}

}

stripe.Account.modify(user2.stripe\_id, individual=individual)

new\_card\_source = stripe.Customer.create\_source(user1.stripe\_id, source=token)

stripe.SetupIntent.create( payment\_method\_types=["card"], customer=user1.stripe\_id, description="some description", payment\_method=new\_card\_source.id,

)

payment\_method = stripe.Customer.retrieve(user1.stripe\_id).default\_source

payment\_intent = stripe.PaymentIntent.create( amount=amount,

currency="pln", payment\_method\_types=["card"], capture\_method="manual", customer=user1.stripe\_id, # customer payment\_method=payment\_method, application\_fee\_amount=application\_fee\_amount,

transfer\_data={"destination": user2.stripe\_id}, # connect account description=description,

metadata=metadata,

)

payment\_intent\_confirm = stripe.PaymentIntent.confirm( payment\_intent.stripe\_id, payment\_method=payment\_method

)

stripe.PaymentIntent.capture( payment\_intent.id, amount\_to\_capture=amount

)

stripe.Balance.retrieve(stripe\_account=user2.stripe\_id)

stripe.Charge.create( amount=amount, currency="pln", source=user2.stripe\_id, description=description

)

stripe.PaymentIntent.cancel(payment\_intent.id)

unique\_together = ("user", "group") @attr.s(frozen=True, cmp=False, hash=False, repr=True) class UserSettings(MethodView):

form = attr.ib(factory=settings\_form\_factory) settings\_update\_handler = attr.ib(factory=settings\_update\_handler)

decorators = [login\_required] def get(self): return self.render()

def post(self):

if self.form.validate\_on\_submit(): try:

self.settings\_update\_handler.apply\_changeset( current\_user, self.form.as\_change()

)

except StopValidation as e:

self.form.populate\_errors(e.reasons) return self.render() except PersistenceError:

logger.exception("Error while updating user settings") flash(\_("Error while updating user settings"), "danger") return self.redirect()

flash(\_("Settings updated."), "success") return self.redirect() return self.render()

def render(self):

return render\_template("user/general\_settings.html", form=self.form) def redirect(self):

return redirect(url\_for("user.settings"))

@attr.s(frozen=True, hash=False, cmp=False, repr=True) class ChangePassword(MethodView):

form = attr.ib(factory=change\_password\_form\_factory) password\_update\_handler =

attr.ib(factory=password\_update\_handler) decorators = [login\_required]

def get(self):

return self.render() def post(self):

if self.form.validate\_on\_submit(): try:

self.password\_update\_handler.apply\_changeset( current\_user, self.form.as\_change()

)

except StopValidation as e: self.form.populate\_errors(e.reasons) return self.render()

except PersistenceError:

logger.exception("Error while changing password") flash(\_("Error while changing password"), "danger") return self.redirect()

flash(\_("Password updated."), "success") return self.redirect() return self.render()

def render(self):

return render\_template("user/change\_password.html", form=self.form) def redirect(self):

return redirect(url\_for("user.change\_password"))

@attr.s(frozen=True, cmp=False, hash=False, repr=True) class ChangeEmail(MethodView):

form = attr.ib(factory=change\_email\_form\_factory) update\_email\_handler = attr.ib(factory=email\_update\_handler) decorators = [login\_required]

def get(self):

return self.render() def post(self):

if self.form.validate\_on\_submit(): try:

self.update\_email\_handler.apply\_changeset( current\_user, self.form.as\_change()

)

except StopValidation as e: self.form.populate\_errors(e.reasons) return self.render()

except PersistenceError:

logger.exception("Error while updating email") flash(\_("Error while updating email"), "danger") return self.redirect()

flash(\_("Email address updated."), "success") return self.redirect() return self.render()

def render(self):

return render\_template("user/change\_email.html", form=self.form) def redirect(self):

return redirect(url\_for("user.change\_email")) def berth\_type(s): if s>0 and s<73:

if s % 8 == 1 or s % 8 == 4: print (s), "is lower berth"

elif s % 8 == 2 or s % 8 == 5: print (s), "is middle berth" elif s % 8 == 3 or s % 8 == 6:

print (s), "is upper berth" elif s % 8 == 7:

print (s), "is side lower berth" else:

print (s), "is side upper berth" else:

print (s), "invalid seat number" # Driver code s = 10

berth\_type(s) # fxn call for berth type s = 7

berth\_type(s) # fxn call for berth type s = 0

berth\_type(s) # fxn call for berth type class Ticket: counter=0

def init (self,passenger\_name,source,destination): self. passenger\_name=passenger\_name

self. source=source

self. destination=destination self.Counter=Ticket.counter Ticket.counter+=1

def validate\_source\_destination(self):

if (self. source=="Delhi" and (self. destination=="Pune" or self. destination=="Mumbai" or self. destination=="Chennai" or self. destination=="Kolkata")):

return True else:

return False

def generate\_ticket(self ): if True:

ticket\_id=self. source[0]+self. destination[0]+"0"+str(self.Counter) print( "Ticket id will be:", ticket\_id)

else:

return False

def get\_ticket\_id(self): return self.ticket\_id

def get\_passenger\_name(self): return self.\_\_passenger\_name def get\_source(self):

if self.source=="Delhi": return self.source else:

print("you have written invalid soure option") return None def get\_destination(self):

if self.destination=="Pune": return self. destination elif self.destination=="Mumbai": return self. destination

elif self.destination=="Chennai": return self. destination elif self.destination=="Kolkata": return self. destination else:

return None

# user define function # Scrape the data def getdata(url):

r = requests.get(url) return r.text # input by geek

train\_name = "03391-rajgir-new-delhi-clone-special-rgd-to-ndls" # url

url = "https:/[/www.railyatri.in/live-train-status/](http://www.railyatri.in/live-train-status/)"+train\_name # pass the url

# into getdata function htmldata = getdata(url) soup = BeautifulSoup(htmldata, 'html.parser')

# traverse the live status from # this Html code data = []

for item in soup.find\_all('script', type="application/ld+json"): data.append(item.get\_text())

# convert into dataframe df = pd.read\_json(data[2]) # display this column of # dataframe print(df["mainEntity"][0]['name'])

print(df["mainEntity"][0]['acceptedAnswer']['text'])

Speak method

def Speak(self, audio):

# Calling the initial constructor # of pyttsx3 engine = pyttsx3.init('sapi5')

# Calling the getter method

voices = engine.getProperty('voices')

# Calling the setter method engine.setProperty('voice', voices[1].id) engine.say(audio) engine.runAndWait()

def Take\_break():

Speak("Do you want to start sir?") question = input() if "yes" in question:

Speak("Starting Sir") Sir.")

mins",

as you have" affect your eyes", if "no" in question:

Speak("We will automatically start after 5 Mins time.sleep(5\*60) Speak("Starting Sir")

# A notification we will held that

# Let's Start sir and with a message of # will tell you to take a break after 45 # mins for 10 seconds

while(True): notification.notify(title="Let's Start sir", message="will tell you to take a break after 45 timeout=10)

# For 45 min the will be no notification but # after 45 min a notification will pop up. time.sleep(0.5\*60)

Speak("Please Take a break Sir")

notification.notify(title="Break Notification", message="Please do use your device after sometime

"been continuously using it for 45 mins and it will timeout=10) # Driver's Code

If name== ' main ':

Take\_break() data\_path = 'data.csv'

data = pd.read\_csv(data\_path, names=['LATITUDE', 'LONGITUDE'], sep=',') gps\_data = tuple(zip(data['LATITUDE'].values, data['LONGITUDE'].values)) image = Image.open('map.png', 'r') # Load map image. img\_points = []

for d in gps\_data:

x1, y1 = scale\_to\_img(d, (image.size[0], image.size[1])) # Convert GPS coordinates to image coordinates.

img\_points.append((x1, y1)) draw = ImageDraw.Draw(image)

draw.line(img\_points, fill=(255, 0, 0), width=2) # Draw converted records to the map image.

image.save('resultMap.png')

x\_ticks = map(lambda x: round(x, 4), np.linspace(lon1, lon2, num=7)) y\_ticks = map(lambda x: round(x, 4), np.linspace(lat1, lat2, num=8))

y\_ticks = sorted(y\_ticks, reverse=True) # y ticks must be reversed due to conversion to image coordinates.

fig, axis1 = plt.subplots(figsize=(10, 10)) axis1.imshow(plt.imread('resultMap.png')) # Load the image to matplotlib plot. axis1.set\_xlabel('Longitude')

axis1.set\_ylabel('Latitude') axis1.set\_xticklabels(x\_ticks) axis1.set\_yticklabels(y\_ticks) axis1.grid()

plt.show() class tickets:

def init(self): self.no\_ofac1stclass=0 self.totaf=0 self.no\_ofac2ndclass=0 self.no\_ofac3rdclass=0 self.no\_ofsleeper=0 self.no\_oftickets=0 self.name=''

self.age='' self.resno=0 self.status='' def ret(self): return(self.resno)

def retname(self): return(self.name) def display(self): f=0

fin1=open("tickets.dat","rb") if not fin1:

print "ERROR" else:

print

n=int(raw\_input("ENTER PNR NUMBER : ")) print "\n\n" print ("FETCHING DATA . . .".center(80)) time.sleep(1) print

print('PLEASE WAIT...!!'.center(80)) time.sleep(1) os.system('cls') try:

while True: tick=load(fin1) if(n==tick.ret()): f=1

print "="\*80

print("PNR STATUS".center(80)) print"="\*80 print

print "PASSENGER'S NAME :",tick.name print print "PASSENGER'S AGE :",tick.age print print "PNR NO :",tick.resno print

print "STATUS :",tick.status print

print "NO OF SEATS BOOKED : ",tick.no\_oftickets print except:

pass fin1.close() if(f==0): print

print "WRONG PNR NUMBER..!!"

print

def pending(self): self.status="WAITING LIST" print "PNR NUMBER

:",self.resno print time.sleep(1.2)

print "STATUS = ",self.status print

print "NO OF SEATS BOOKED : ",self.no\_oftickets print def confirmation (self): self.status="CONFIRMED"

print "PNR NUMBER : ",self.resno print time.sleep(1.5)

print "STATUS = ",self.status print def cancellation(self): z=0

f=0 fin=open("tickets.dat","rb") fout=open("temp.dat","ab") print r= int(raw\_input("ENTER PNR NUMBER : ")) try:

while(True): tick=load(fin) z=tick.ret() if(z!=r): dump(tick,fout) elif(z==r):

f=1 except:

pass fin.close() fout.close()

os.remove("tickets.dat") os.rename("temp.dat","tickets.dat") if (f==0): print

print "NO SUCH RESERVATION NUMBER FOUND"

print time.sleep(2) os.system('cls') else:

print

print "TICKET CANCELLED" print"RS.600 REFUNDED "

def reservation(self):

trainno=int(raw\_input("ENTER THE TRAIN NO:")) z=0 f=0 fin2=open("tr1details.dat") fin2.seek(0)

if not fin2:

print "ERROR" else:

try:

while True:

n=int(raw\_input("ENTER PNR NUMBER : ")) print "\n\n" print ("FETCHING DATA . . .".center(80)) time.sleep(1) print

print('PLEASE WAIT...!!'.center(80)) time.sleep(1) os.system('cls') try:

while True: tick=load(fin1) if(n==tick.ret()):

f=1

print "="\*80

print("PNR STATUS".center(80)) print"="\*80 print

print "PASSENGER'S NAME :",tick.name print print "PASSENGER'S AGE :",tick.age print print "PNR NO :",tick.resno print

print "STATUS :",tick.status print

print "NO OF SEATS BOOKED : ",tick.no\_oftickets print except:

pass fin1.close() if(f==0):

print

print "WRONG PNR NUMBER..!!"

print

def pending(self): self.status="WAITING LIST" print "PNR NUMBER

:",self.resno print time.sleep(1.2)

print "STATUS = ",self.status print

print "NO OF SEATS BOOKED : ",self.no\_oftickets print def confirmation (self): self.status="CONFIRMED"

print "PNR NUMBER : ",self.resno print

time.sleep(1.5)

print "STATUS = ",self.status print def cancellation(self): z=0

f=0 fin=open("tickets.dat","rb") fout=open("temp.dat","ab") print r= int(raw\_input("ENTER PNR NUMBER : ")) try:

while(True): tick=load(fin) z=tick.ret() if(z!=r): dump(tick,fout) elif(z==r):

f=1 except:

pass fin.close() fout.close()

os.remove("tickets.dat") os.rename("temp.dat","tickets.dat") if (f==0): print

print "NO SUCH RESERVATION NUMBER FOUND"

print time.sleep(2) os.system('cls') else:

print

print "TICKET CANCELLED" print"RS.600 REFUNDED "

def reservation(self):

trainno=int(raw\_input("ENTER THE TRAIN NO:")) z=0 f=0 fin2=open("tr1details.dat") fin2.seek(0)

if not fin2:

print "ERROR" else:

try:

while True: tr=load(fin2) z=tr.gettrainno() n=tr.gettrainname() if (trainno==z): print

print "TRAIN NAME IS : ",n

f=1 print

print "-"\*80 no\_ofac1st=tr.getno\_ofac1stclass() no\_ofac2nd=tr.getno\_ofac2ndclass() no\_ofac3rd=tr.getno\_ofac3rdclass() no\_ofsleeper=tr.getno\_ofsleeper()

if(f==1): fout1=open("tickets.dat","ab") NAME ")

print

self.name=raw\_input("ENTER THE PASSENGER'S print

self.age=int(raw\_input("PASSENGER'S AGE : ")) print print"\t\t SELECT A CLASS YOU WOULD LIKE TO TRAVEL IN :- "

print "1.AC FIRST CLASS"

print

print "2.AC SECOND CLASS"

print

print "3.AC THIRD CLASS"

print

print "4.SLEEPER CLASS"

print

c=int(raw\_input("\t\t\tENTER YOUR CHOICE = ")) os.system('cls') amt1=0 if(c==1):

self.no\_oftickets=int(raw\_input("ENTER NO\_OF FIRST CLASS AC SEATS TO BE BOOKED : "))

i=1 while(i<=self.no\_oftickets):

self.totaf=self.totaf+1 amt1=1000\*self.no\_oftickets i=i+1 print

print "PROCESSING. .",

time.sleep(0.5)

print ".", time.sleep(0.3) print'.' time.sleep(2) os.system('cls')

print "TOTAL AMOUNT TO BE PAID = ",amt1

self.resno=int(random.randint(1000,2546)) x=no\_ofac1st-self.totaf print if(x>0):

self.confirmation() dump(self,fout1) break else:

self.pending() dump(tick,fout1) break

elif(c==2): self.no\_oftickets=int(raw\_input("ENTER NO\_OF SECOND CLASS AC SEATS TO BE BOOKED : "))

i=1

def menu(): tr=train() tick=tickets() print

print "WELCOME TO PRAHIT AGENCY".center(80)

while True:

print

print "="\*80

print " \t\t\t\t RAILWAY" print print "="\*80 print

print "\t\t\t1. \*\*UPDATE TRAIN DETAILS." print print "\t\t\t2. TRAIN DETAILS. " print

print "\t\t\t3. RESERVATION OF TICKETS." print print "\t\t\t4. CANCELLATION OF TICKETS. "

print

print "\t\t\t5. DISPLAY PNR STATUS." print print "\t\t\t6. QUIT." print"\*\* - office use "

ch=int(raw\_input("\t\t\tENTER YOUR CHOICE : ")) os.system('cls')

print "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t\tLOADI NG. .",

time.sleep(1) print ("."), time.sleep(0.5) print (".") time.sleep(2) os.system('cls') if ch==1:

j="\*\*\*\*\*" r=raw\_input("\n\n\n\n\n\n\n\n\n\n\n\t\t\t\tENTER THE

PASSWORD: ")

os.system('cls') if (j==r):

x='y'

while (x.lower()=='y'): fout=open("tr1details.dat","ab") tr.getinput() dump(tr,fout) fout.close()

print"\n\n\n\n\n\n\n\n\n\n\n\t\t\tUPDATING TRAIN LIST PLEASE WAIT . .", time.sleep(1) print ("."), time.sleep(0.5) print ("."), time.sleep(2) os.system('cls') print "\n\n\n\n\n\n\n\n\n\n\n"

x=raw\_input("\t\tDO YOU WANT TO ADD ANY MORE TRAINS DETAILS

? ")

os.system('cls') continue elif(j<>r): print"\n\n\n\n\n"

print "WRONG PASSWORD".center(80) elif ch==2: fin=open("tr1details.dat",'rb') if not fin:

print "ERROR" tick.display() elif ch==6: quit()

raw\_input("PRESS ENTER TO GO TO BACK

MENU".center(80)) os.system('cls') menu()

sender\_email = "[my@gmail.com](mailto:my@gmail.com)" receiver\_email = "[your@gmail.com](mailto:your@gmail.com)" password = input("Type your password and press enter:")

message = MIMEMultipart("alternative") message["Subject"] = "multipart test" message["From"] = sender\_email message["To"] = receiver\_email

# Create the plain-text and HTML version of your message text = """\ Hi,

How are you?

Real Python has many great tutorials: [www.realpython.com](http://www.realpython.com/)""" html = """\

<html>

<body>

<p>Hi,<br>

How are you?<br>

<a href="[http://www.realpython.com](http://www.realpython.com/)">Real Python</a> has many great tutorials.

</p>

</body>

</html> """

# Turn these into plain/html MIMEText objects part1 = MIMEText(text, "plain")

part2 = MIMEText(html, "html")

# Add HTML/plain-text parts to MIMEMultipart message # The email client will try to render the last part first message.attach(part1)

message.attach(part2)

# Create secure connection with server and send email context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as server: server.login(sender\_email, password) server.sendmail(

sender\_email, receiver\_email, message.as\_string()

)

subject = "An email with attachment from Python"

body = "This is an email with attachment sent from Python" sender\_email = "[my@gmail.com](mailto:my@gmail.com)"

receiver\_email = "[your@gmail.com](mailto:your@gmail.com)"

password = input("Type your password and press enter:")

# Create a multipart message and set headers message = MIMEMultipart() message["From"] = sender\_email message["To"] = receiver\_email message["Subject"] = subject

message["Bcc"] = receiver\_email # Recommended for mass emails

# Add body to email message.attach(MIMEText(body, "plain"))

filename = "document.pdf" # In same directory as script # Open PDF file in binary mode

with open(filename, "rb") as attachment:

# Add file as application/octet-stream

# Email client can usually download this automatically as attachment part = MIMEBase("application", "octet-stream") part.set\_payload(attachment.read())

# Encode file in ASCII characters to send by email encoders.encode\_base64(part)

# Add header as key/value pair to attachment part part.add\_header( "Content-Disposition", f"attachment; filename= {filename}",

)

# Add attachment to message and convert message to string message.attach(part)

text = message.as\_string()

# Log in to server using secure context and send email context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as server:

server.login(sender\_email, password) server.sendmail(sender\_email, receiver\_email, text)

api\_key = "Your\_API\_key"

# base\_url variable to store url

base\_url = "https://api.railwayapi.com/v2/pnr-status/pnr/" # Enter valid pnr\_number pnr\_number = "6515483790"

# Stores complete url address

complete\_url = base\_url + pnr\_number + "/apikey/" + api\_key + "/" # get method of requests module # return response object response\_ob = requests.get(complete\_url)

# json method of response object convert

# json format data into python format data result = response\_ob.json()

# now result contains list # of nested dictionaries if result["response\_code"] == 200:

# train name is extracting

# from the result variable data train\_name = result["train"]["name"] # train number is extracting from # the result variable data train\_number = result["train"]["number"]

# from station name is extracting # from the result variable data from\_station = result["from\_station"]["name"]

# to\_station name is extracting from # the result variable data to\_station = result["to\_station"]["name"]

# boarding point station name is

# extracting from the result variable data boarding\_point = result["boarding\_point"]["name"]

# reservation upto station name is

# extracting from the result variable data reservation\_upto = result["reservation\_upto"]["name"]

# store the value or data of "pnr" # key in pnr\_num variable pnr\_num = result["pnr"]

# store the value or data of "doj" key # in variable date\_of\_journey variable date\_of\_journey = result["doj"]

# store the value or data of

# "total\_passengers" key in variable total\_passengers = reult["total\_passengers"]

# store the value or data of "passengers" # key in variable passengers\_list passengers\_list = result["passengers"]

# store the value or data of

# "chart\_prepared" key in variable chart\_prepared = result["chart\_prepared"] # print following values

print(" train name : " + str(train\_name)

+ "\n train number : " + str(train\_number)

+ "\n from station : " + str(from\_station)

+ "\n to station : " + str(to\_station)

+ "\n boarding point : " + str(boarding\_point)

+ "\n reservation upto : " + str(reservation\_upto)

+ "\n pnr number : " + str(pnr\_num)

+ "\n date of journey : " + str(date\_of\_journey)

+ "\n total no. of passengers: " + str(total\_passengers)

+ "\n chart prepared : " + str(chart\_prepared)) # looping through passenger list

for passenger in passengers\_list:

# store the value or data # of "no" key in variable passenger\_num = passenger["no"]

# store the value or data of

# "current\_status" key in variable current\_status = passenger["current\_status"] # store the value or data of

# "booking\_status" key in variable booking\_status = passenger["booking\_status"]

# print following values

print(" passenger number : " + str(passenger\_num)

+ "\n current status : " + str(current\_status)

+ "\n booking\_status : " + str(booking\_status)) else:

print("Record Not Found") NAME ")

tr=load(fin2) z=tr.gettrainno() n=tr.gettrainname() if (trainno==z): print

print "TRAIN NAME IS : ",n

f=1 print

print "-"\*80 no\_ofac1st=tr.getno\_ofac1stclass() no\_ofac2nd=tr.getno\_ofac2ndclass() no\_ofac3rd=tr.getno\_ofac3rdclass() no\_ofsleeper=tr.getno\_ofsleeper()

if(f==1): fout1=open("tickets.dat","ab") print self.name=raw\_input("ENTER THE PASSENGER'S print

self.age=int(raw\_input("PASSENGER'S AGE : ")) print print"\t\t SELECT A CLASS YOU WOULD LIKE TO TRAVEL IN :- "

print "1.AC FIRST CLASS"

print

print "2.AC SECOND CLASS"

print

print "3.AC THIRD CLASS"

print

print "4.SLEEPER CLASS"

print

c=int(raw\_input("\t\t\tENTER YOUR CHOICE = ")) os.system('cls') amt1=0 if(c==1):

self.no\_oftickets=int(raw\_input("ENTER NO\_OF FIRST CLASS AC SEATS TO BE BOOKED : "))

i=1 while(i<=self.no\_oftickets):

self.totaf=self.totaf+1 amt1=1000\*self.no\_oftickets i=i+1 print

print "PROCESSING. .",

time.sleep(0.5) print ".", time.sleep(0.3) print'.' time.sleep(2) os.system('cls') print "TOTAL AMOUNT TO BE PAID = ",amt1

self.resno=int(random.randint(1000,2546)) x=no\_ofac1st-self.totaf print if(x>0):

self.confirmation() dump(self,fout1) break else:

self.pending() dump(tick,fout1) break

elif(c==2): self.no\_oftickets=int(raw\_input("ENTER NO\_OF SECOND CLASS AC SEATS TO BE BOOKED : "))

i=1

def menu(): tr=train() tick=tickets() print

print "WELCOME TO PRAHIT AGENCY".center(80)

while True:

print

print "="\*80

print " \t\t\t\t RAILWAY" print print "="\*80 print

print "\t\t\t1. \*\*UPDATE TRAIN DETAILS." print print "\t\t\t2. TRAIN DETAILS. " print

print "\t\t\t3. RESERVATION OF TICKETS." print print "\t\t\t4. CANCELLATION OF TICKETS. "

print

print "\t\t\t5. DISPLAY PNR STATUS." print

print "\t\t\t6. QUIT." print"\*\* - office use " ch=int(raw\_input("\t\t\tENTER YOUR CHOICE : ")) os.system('cls')

print "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t\tLOADI NG. .",

time.sleep(1) print ("."), time.sleep(0.5) print (".") time.sleep(2) os.system('cls') if ch==1:

j="\*\*\*\*\*" r=raw\_input("\n\n\n\n\n\n\n\n\n\n\n\t\t\t\tENTER THE PASSWORD: ")

os.system('cls') if (j==r):

x='y'

while (x.lower()=='y'): fout=open("tr1details.dat","ab") tr.getinput() dump(tr,fout) fout.close()

print"\n\n\n\n\n\n\n\n\n\n\n\t\t\tUPDATING TRAIN LIST PLEASE WAIT . .", time.sleep(1) print ("."),

time.sleep(0.5) print ("."), time.sleep(2) os.system('cls') print "\n\n\n\n\n\n\n\n\n\n\n"

x=raw\_input("\t\tDO YOU WANT TO ADD ANY MORE TRAINS DETAILS

? ")

os.system('cls') continue elif(j<>r): print"\n\n\n\n\n"

print "WRONG PASSWORD".center(80) elif ch==2: fin=open("tr1details.dat",'rb') if not fin:

print "ERROR" tick.display() elif ch==6: quit()

raw\_input("PRESS ENTER TO GO TO BACK

MENU".center(80)) os.system('cls') menu()

sender\_email = "[my@gmail.com](mailto:my@gmail.com)" receiver\_email = "[your@gmail.com](mailto:your@gmail.com)" password = input("Type your password and press enter:")

message = MIMEMultipart("alternative") message["Subject"] = "multipart test" message["From"] = sender\_email message["To"] = receiver\_email

# Create the plain-text and HTML version of your message text = """\ Hi,

How are you?

Real Python has many great tutorials: [www.realpython.com](http://www.realpython.com/)""" html = """\

<html>

<body>

<p>Hi,<br>

How are you?<br>

<a href="[http://www.realpython.com](http://www.realpython.com/)">Real Python</a> has many great tutorials.

</p>

</body>

</html> """

# Turn these into plain/html MIMEText objects part1 = MIMEText(text, "plain")

part2 = MIMEText(html, "html")

# Add HTML/plain-text parts to MIMEMultipart message # The email client will try to render the last part first message.attach(part1)

message.attach(part2)

# Create secure connection with server and send email context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as server: server.login(sender\_email, password) server.sendmail(

sender\_email, receiver\_email, message.as\_string()

)

subject = "An email with attachment from Python"

body = "This is an email with attachment sent from Python" sender\_email = "[my@gmail.com](mailto:my@gmail.com)"

receiver\_email = "[your@gmail.com](mailto:your@gmail.com)"

password = input("Type your password and press enter:")

# Create a multipart message and set headers message = MIMEMultipart() message["From"] = sender\_email message["To"] = receiver\_email message["Subject"] = subject

message["Bcc"] = receiver\_email # Recommended for mass emails

# Add body to email message.attach(MIMEText(body, "plain"))

filename = "document.pdf" # In same directory as script # Open PDF file in binary mode

with open(filename, "rb") as attachment:

# Add file as application/octet-stream

# Email client can usually download this automatically as attachment part = MIMEBase("application", "octet-stream") part.set\_payload(attachment.read())

# Encode file in ASCII characters to send by email encoders.encode\_base64(part)

# Add header as key/value pair to attachment part part.add\_header( "Content-Disposition", f"attachment; filename= {filename}",

)

# Add attachment to message and convert message to string message.attach(part)

text = message.as\_string()

# Log in to server using secure context and send email context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as server:

server.login(sender\_email, password) server.sendmail(sender\_email, receiver\_email, text)

api\_key = "Your\_API\_key"

# base\_url variable to store url

base\_url = "https://api.railwayapi.com/v2/pnr-status/pnr/" # Enter valid pnr\_number pnr\_number = "6515483790"

# Stores complete url address

complete\_url = base\_url + pnr\_number + "/apikey/" + api\_key + "/" # get method of requests module # return response object response\_ob = requests.get(complete\_url)

# json method of response object convert

# json format data into python format data result = response\_ob.json()

# now result contains list # of nested dictionaries if result["response\_code"] == 200:

# train name is extracting

# from the result variable data train\_name = result["train"]["name"] # train number is extracting from # the result variable data train\_number = result["train"]["number"]

# from station name is extracting # from the result variable data from\_station = result["from\_station"]["name"]

# to\_station name is extracting from # the result variable data to\_station = result["to\_station"]["name"]

# boarding point station name is

# extracting from the result variable data boarding\_point = result["boarding\_point"]["name"]

# reservation upto station name is

# extracting from the result variable data reservation\_upto = result["reservation\_upto"]["name"]

# store the value or data of "pnr" # key in pnr\_num variable pnr\_num = result["pnr"]

# store the value or data of "doj" key # in variable date\_of\_journey variable date\_of\_journey = result["doj"]

# store the value or data of

# "total\_passengers" key in variable total\_passengers = result["total\_passengers"]

# store the value or data of "passengers" # key in variable passengers\_list passengers\_list = result["passengers"]

# store the value or data of

# "chart\_prepared" key in variable chart\_prepared = result["chart\_prepared"] # print following values

print(" train name : " + str(train\_name)

+ "\n train number : " + str(train\_number)

+ "\n from station : " + str(from\_station)

+ "\n to station : " + str(to\_station)

+ "\n boarding point : " + str(boarding\_point)

+ "\n reservation upto : " + str(reservation\_upto)

+ "\n pnr number : " + str(pnr\_num)

+ "\n date of journey : " + str(date\_of\_journey)

+ "\n total no. of passengers: " + str(total\_passengers)

+ "\n chart prepared : " + str(chart\_prepared)) # looping through passenger list

for passenger in passengers\_list:

# store the value or data # of "no" key in variable passenger\_num = passenger["no"]

# store the value or data of

# "current\_status" key in variable current\_status = passenger["current\_status"] # store the value or data of

# "booking\_status" key in variable booking\_status = passenger["booking\_status"]

# print following values

print(" passenger number : " + str(passenger\_num)

+ "\n current status : " + str(current\_status)

+ "\n booking\_status : " + str(booking\_status)) else:

print("Record Not Found")

## GIT HUB LINK

https://github.com/IBM-EPBL/IBM-Project-15060-1659593698