SPRINT 03

Date	12 November 2022
Team ID	PNT2022TMID11066
Project Name	Smart solutions for railways
Maximum Marks	20 marks

PAYMENT:

```
#Payment
from django.contrib.auth.base_user import AbstractBaseUser
from django.db import models


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",
    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",
    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.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")
```

NOTIFICATION:

```
#Notification
import pyttsx3
from plyer import notification
import time


# 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")

        if "no" in question:
                Speak("We will automatically start after 5 Mins Sir.")
                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 mins",
                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 as you have"
                    "been continuously using it for 45 mins and it will affect your eyes",
                    timeout=10)


# Driver's Code
if __name__ == '__main__':
          Take_break()
```

TICKET GENERATION:

```
#Ticket_Generation
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
```

CONFIRMATION:

```
#Confirmation
 import module
import requests
from bs4 import BeautifulSoup
import pandas as pd

# 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/"+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'])
```