Computer Science Project Report

TravelWise

a Travel Itinerary Planner

By:
Arnav Yayavaram
Siddharth Yayavaram
Shailesh Chandra Rao
XII 'A'

Sri Kumaran Children's Home, CBSE

Bangalore

Project Certificate

This is to certify that

…….....

Of Class XII has successfully completed the project work titled ----------

---------- under the guidance

of ----------during the academic year _____in

partial fulfillment of _____practical examination conducted

*by **AISSCE.***

_____ _____

Internal Examiner ***External Examiner***

_____

Principal

Date of Practical Examination : _____

Name of the Candidate: _____

Registration Number: _____

Examination Centre: _____

Index

Acknowledgement

The success of a project depends on the persistent efforts of an individual projecting it and the sustained support received from a few others who are equally responsible for their precious appreciation of such endeavours. My strength is all due to my honourable Principal **Mrs. Pushkala Parasuraman**, who has been an unending source of inspiration and support towards accomplishment of this project.

I would like to express my deepest sense of gratitude to my computer teachers **Mrs. Padma Pavani Kanukollu** and **Mrs. Smitha Ravindran** without whom I could not have successfully completed this project.

I would also like to thank all my friends who helped me create such a project.

My personal gratitude is extended towards my parents, who have been a constant source of encouragement and support in the success of the project. Last but not the least I want to thank the Almighty for enlightening, strengthening, and guiding me in the completion of this project.

Synopsis

TravelWise is an itinerary planner which allows one to truly focus on enjoying the trip and not on the stress riddled task of planning it.

The carefully crafted algorithm takes your preferences into consideration and delivers a meticulously constructed guide which caters to your every need. We consider the weather, as well as the type of attraction which takes precedence.

The site offers you three different carefully laid out plans within your budget with only the most popular and magnificent attractions.

We even provide the means of transportation that one will require for their trip.

TravelWise is truly an essential tool which brings back the essence of a vacation which is to appreciate the attraction and to immerse oneself in the culture and heritage of the destination and not to worry about the planning of the trip.

Hardware and Software Specifications

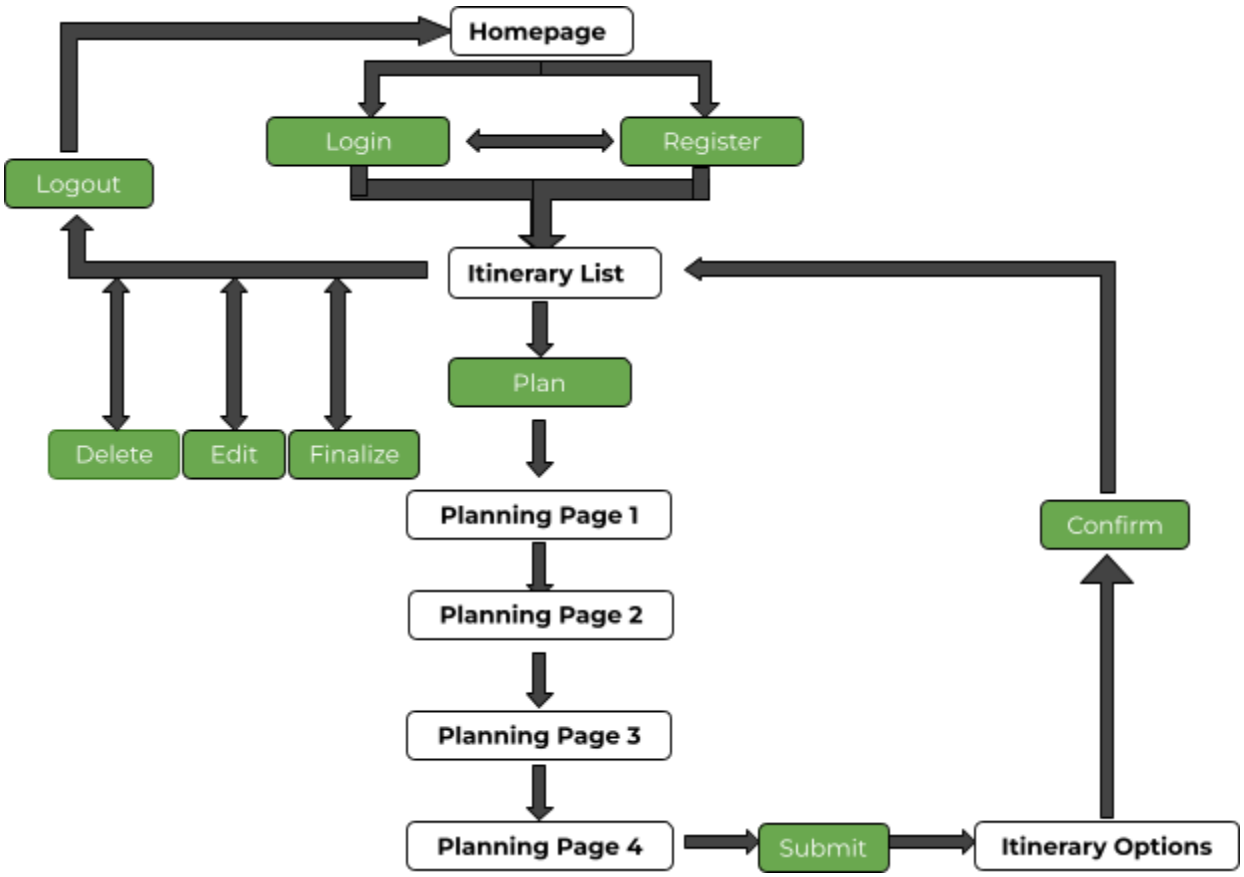
Hardware:

- Processor: 1.60 Ghz or more
- RAM: 8GB or more
- VDU: LCD/LED

Software:

- Operating System: Windows 7, 8 or 10
- Python 3-8-0 64 bit
- MySQL installed

Flow Diagram/State Transition Diagram



Library Modules and their Purpose:

S. No.	Library Modules	Purpose
1.	random	Implements pseudo-random number generators for various distributions
2.	copy	Provides generic shallow and deep copy operations
3.	pickle	To create and edit binary files
4.	os	Provides functions for interacting with the operating system
5.	datetime	Supplies classes for manipulating dates and times
6.	math	Provides access to mathematical functions
7.	mysql.connector	Enables access to mysql databases

Functions and their Purpose

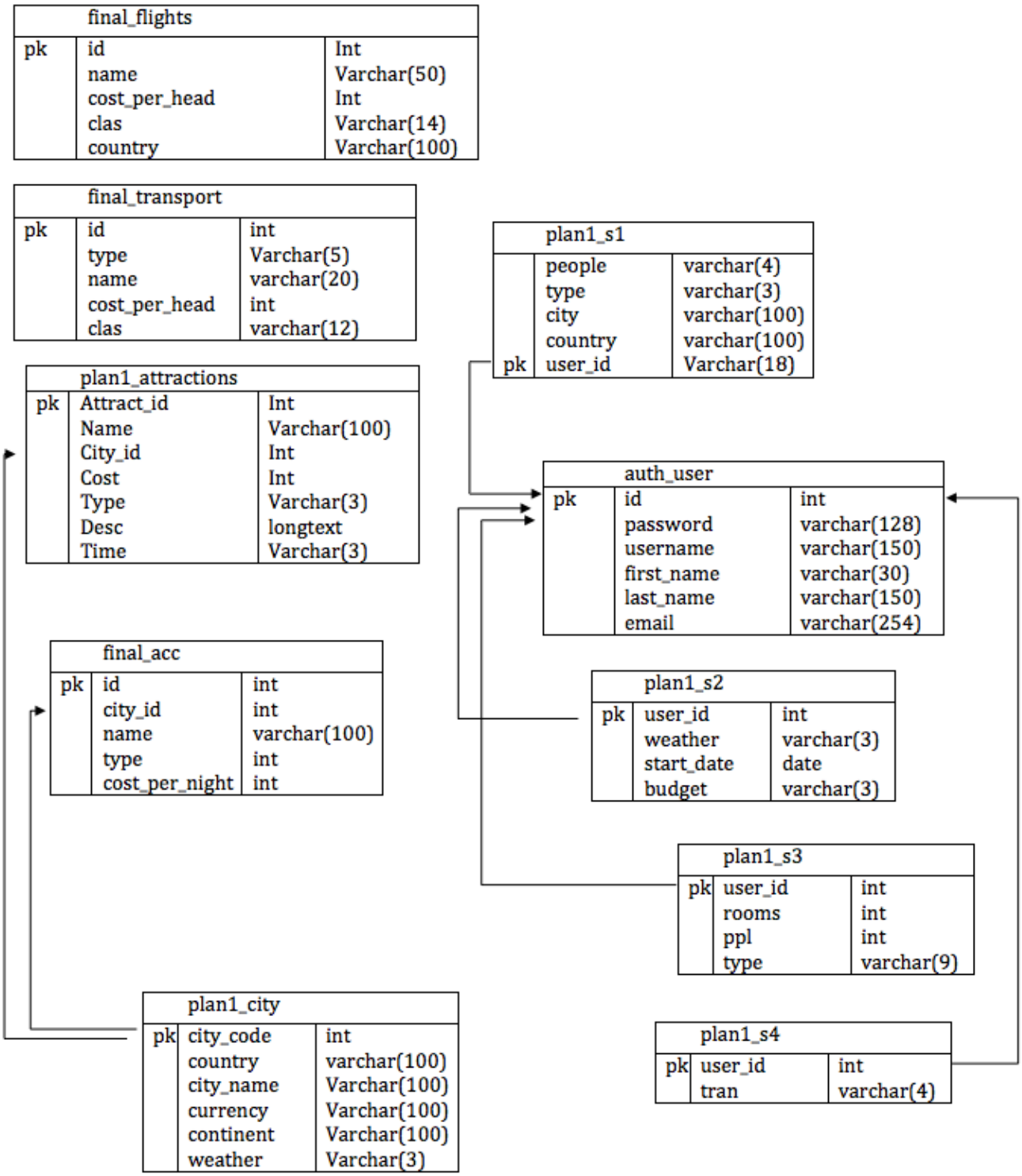
S.No	File	Function	Purpose
1	addUser\ views.py	**index()**	directs to landing page
2	addUser\ views.py	**login()**	authenticates the user, checks validity of credentials
3	addUser\ views.py	**add_User()**	takes user info from the form and loads it into the database
4	plan1\ views.py	**plan_1(), plan_2(), plan_3(), plan_4()**	directs user to planning pages
5	final\ views.py	**show()**	calls all planning functions, fetches user inputs and shows final itinerary
6	final\ views.py	**f1()**	decides the country based on the weather and decides the flight details based on budget range and country
7	final\ views.py	**f2()**	calculates attraction + accommodation costs, decides attractions based on type input, decides accommodation based on budget and also adds the timeline info.
8	final\ views.py	**timeline_info()**	organizes the attractions and schedules the trip and also adds transport details
9	final\ views.py	**edit()**	edits the itinerary, reorganizes the timeline
10	final\ views.py	**delete()**	deletes itineraries
11	tip_func s.py	**city_validate()**	validates cities based on attraction types
12	tip_func s.py	**country_validate()**	validates countries based on whether cities are valid

Data Files And Their Purpose

The data file which has been used in the project is:

S.No	Data File	Purpose
1.	itinerary_files.dat	Used to store all the itineraries planned for each user along with the country, flights, attraction details in the form of dictionaries.

Entity Relationship Diagram



Source Code

addUser\views.py:

```
from django.shortcuts import render
from django.contrib.auth.models import User as User_New
from django.contrib.auth import authenticate
from django.contrib.auth import login as log
import pickle

# Create your views here.
def index(request):  # directs to landing page
  return render(request, 'homepage.html')

def add_User(request):  # takes user info from the form and loads it into the database
  if request.method == 'GET':
    return render(request, 'add_user1.html')
  elif request.method == 'POST':
    username = request.POST['username']
    passwd = request.POST['passwd']
    email = request.POST['email']
    f_name = request.POST['f_name']
    l_name = request.POST['l_name']
    if request.GET['a'] == 'db':
      user = User_New.objects.create_user(username, email, passwd)  # creating the user
      user.first_name = f_name
      user.last_name = l_name
      user.save()
      return render(request, 'homepage.html')
    else:
      return render(request, 'homepage.html')

def login(request):  # authenticates the user, checks validity of credentials
  if request.method == 'GET':
    return render(request, 'login.html')
  elif request.method == 'POST':
    user = request.POST['username']
    pwd = request.POST['passwd']
    if request.GET['b'] == 'ck':
      user = authenticate(request, username=user, password=pwd)
      if user is not None:
        log(request, user)
        with open("itinerary_files.dat", "rb") as handle:
          itinerary_list = []
```

```
            while True:
                try:
                    itinerary = pickle.load(handle)
                    if itinerary['id'] == request.user.id:
                        itinerary_list.append(itinerary)  # extracting user's previously planned
itineraries
                    else:
                        pass
                except EOFError:
                    break
        return render(request, 'itinlist.html',  # leads to home page or itinerary list
                      {'lst': itinerary_list, 'len': len(itinerary_list), 'logged_in': request.user.first_name})
    else:
        return render(request, 'login_error.html')  # error page
```

plan1\views.py:

```
from django.shortcuts import render
from plan1.models import s1  # tables which contain choices of users
from plan1.models import s2
from plan1.models import s3
from plan1.models import s4
from django import forms

# Create your views here.
def plan_1(request):
  if request.method == 'GET':
    return render(request, 'mainpage.html')  # directs user to first planning page
  elif request.method == 'POST':
    u = request.user  # instance of currently logged in user
    pp = request.POST['people']  # number of people
    t = request.POST['type']  # scheduling
    city = request.POST['city']  # city of residence
    country = request.POST['country']  # country of residence
    if request.GET['a'] == 's1':
      a = s1()
      a.user = u
      a.people = pp
      a.type = t
      a.city = city
      a.country = country
      a.save()
      return render(request, 'plan2.html')
    else:
      return render('login_error.html')
```

```
def plan_2(request):
  if request.method == 'GET':
    return render(request, 'plan2.html')  # directs user to second planning page
  elif request.method == 'POST':
    u = request.user  # instance of currently logged in user
    w = request.POST.get('weather')  # preferred weather
    s_d = request.POST.get('start_date')  # start date of trip
    bud = request.POST.get('budget')  # budget range
    if request.GET['b'] == 's2':
      b = s2()
      b.user = u
      b.weather = w
      b.start_date = s_d
      b.budget = bud
      b.save()
      val = s1.objects.get(user_id=request.user.id)
      return render(request, 'plan3.html', {'val': val})
    else:
      return render(request, 'login_error.html')

def plan_3(request):
  if request.method == 'GET':
    return render(request, 'plan3.html')  # directs user to third planning page
  elif request.method == 'POST':
    u = request.user  # instance of currently logged in user
    t = request.POST.getlist('type')  # list of attraction types preferred by user
    type = ''
    for i in t:
      type += i  # converting list to string to store in database
    x = s1.objects.get(user_id=request.user.id)
    val = x.people
    if val == 'fami':  # asks for number of people if 'family' option is selected
      rooms = request.POST.get('rooms')  # number of rooms
      ppl = request.POST.get('ppl')  # number of people
    elif val == 'coup':
      rooms = 1
      ppl = 2
    else:
      rooms = 1
      ppl = 1
    if request.GET['b'] == 's3':
      b = s3()
      b.user = u
      b.rooms = rooms
```

```
        b.ppl = ppl
        b.type = type
        b.save()
        return render(request, 'plan4.html')
    else:
        return render(request, 'login_error.html')

def plan_4(request):
  if request.method == 'GET':
    return render(request, 'plan4.html')  # directs user to fourth planning page
  elif request.method == 'POST':
    u = request.user  # instance of currently logged in user
    t2 = request.POST.getlist('tran')  # list of transportation types (between cities)
    tran = ''
    for j in t2:
        tran += j
    if request.GET['b'] == 's4':
        b = s4()
        b.user = u
        b.tran = tran
        b.save()
        return render(request, 'int.html')
    else:
        return render(request, 'login_error.html')

# Classes for datepickers
class Date_Input(forms.DateInput):
  input_type = "date"

class form1(forms.Form):
  start_date = forms.DateField(widget=Date_Input)
  end_date = forms.DateField(widget=Date_Input)
```

plan1\models.py:

```
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class s1(models.Model):
  user=models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)
  people=models.CharField(max_length=4)
  type=models.CharField(max_length=3)
  city=models.CharField(max_length=100)
  country = models.CharField(max_length=100)
```

```
class city(models.Model):
  city_code=models.IntegerField(primary_key=True)
  country=models.CharField(max_length=100)
  city_name = models.CharField(max_length=100)
  currency=models.CharField(max_length=100)
  continent=models.CharField(max_length=100)
  weather=models.CharField(max_length=3)

class attractions(models.Model):
  attract_id=models.IntegerField(primary_key=True)
  name=models.CharField(max_length=100)
  city=models.ForeignKey(city, to_field="city_code", on_delete=models.CASCADE)
  cost = models.IntegerField()
  type = models.CharField(max_length=3)
  time = models.CharField(max_length=3)
  desc = models.TextField(max_length=250)

class s2(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)
  weather = models.CharField(max_length=3)
  start_date = models.DateField()
  budget = models.CharField(max_length=3)

class s3(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)
  rooms = models.IntegerField()
  ppl = models.IntegerField()
  type = models.CharField(max_length=9)

class s4(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE, primary_key=True)
  tran = models.CharField(max_length=4)
```

final\models.py:

```
from django.db import models
from plan1.models import city
# Create your models here.

class acc(models.Model):
  city = models.ForeignKey(city, to_field = "city_code", on_delete = models.CASCADE)
  name = models.CharField(max_length=100)
  type = models.IntegerField()
  cost_per_night = models.IntegerField()

class transport(models.Model):
```

```
  id = models.IntegerField(primary_key = True)
  type = models.CharField(max_length = 5)
  name = models.CharField(max_length = 20)
  cost_per_head = models.IntegerField()
  clas = models.CharField(max_length = 12)

class flights(models.Model):
  id = models.IntegerField(primary_key = True)
  name = models.CharField(max_length = 50)
  cost_per_head = models.IntegerField()
  clas = models.CharField(max_length = 14)
  country=models.CharField(max_length=100)
```

final\views.py:

```
from django.shortcuts import render
from plan1.models import s1
from plan1.models import s2
from plan1.models import s3
from plan1.models import s4
from plan1.models import attractions
from plan1.models import city
from final.models import acc
from final.models import transport
from final.models import flights
from utils import tip_funcs as tf
import os
import random
import datetime
import math
import pickle
import copy

# Create your views here.
countries_visited = []  # countries already visited by the user
country_false = []  # invalid countries: countries not preferred by the user
handler = open("itinerary_files.dat", "wb")  # creating the file used to store final itineraries
handler.close()


def show(request):
  if request.method == 'GET':
    logged_in = request.user
    val1 = s1.objects.get(user_id=request.user.id)  # retrieving inputs given by current user
    val2 = s2.objects.get(user_id=request.user.id)
    val3 = s3.objects.get(user_id=request.user.id)
```

```
    val4 = s4.objects.get(user_id=request.user.id)
    city_val = list(city.objects.values_list('city_name', 'country'))  # fetching all cities from
database
    if (val1.city, val1.country) in city_val:
        country, flight_info, flightcost, flag = f1(val2, val3)  # feeding data to f1()
        city_info, cost1, last, new_Desc, ax = f2(val1, val2, val3, val4, country, flag)  # feeding data
to f2()
        cit = list(city_info.keys())
        first = cit[0]  # first city visited
        COST = flightcost + cost1  # total cost of entire trip
        user_dict = {'id': request.user.id,  # creating dictionary to store in 'itinerary_files.dat'
                'country': country,
                'countryedit': country + '_edit',
                'dict': city_info,
                'new': new_Desc,
                'ax': ax,
                'cost': COST,
                'f_info': flight_info,
                'last': last
                }
        handle = open("itinerary_files.dat", "ab")
        pickle.dump(user_dict, handle)  # dumping dict into file
        handle.close()
        return render(request, 'final.html', {'logged_in': logged_in,  # redirects to final itinerary
page
                                'data1': val1,
                                'data2': val2,
                                'data3': val3,
                                'data4': val4,
                                'country': country,
                                'f_info': flight_info,
                                'info': city_info,
                                'first': first,
                                'last': last,
                                'cost': COST
                                }
                )
    else:
        return render(request, 'city_error.html')

def f1(val2, val3):
  attr_list = tf.get_types(val3.type)  # attraction types preferred by user
  country_info, city_info = tf.queries()  # fetching data from database
  country_list = tf.get_country(val2.weather, country_info)
  country_list.sort()
```

```
  city_val = list(city.objects.values_list('city_name', 'country'))
  while True:
    country_false.sort()  # list of invalid countries
    country_list.sort()  # list of countries which have the chosen weather
    if country_false != country_list:
      country = country_list[random.randint(0, len(country_list) - 1)]  # choosing a random
country
      if country not in countries_visited:  # if country is not yet chosen as valid
        cities_in_country = [i[0] for i in city_val if i[1] == country]  # list of cities in chosen
country
        if tf.country_validate(cities_in_country, attr_list, city_info):
          country_final = country  # validating, finalising the country
          if country not in countries_visited:
            countries_visited.append(country_final)
            country_false.append(country)  # adds it to list of invalid countries
            flag = False
            break
        else:
          if country not in country_false:
            country_false.append(country)  # adds it to list of invalid countries
          else:
            pass
      else:
        pass
    else:  # executes if no country of given weather is valid
      country = country_list[random.randint(0, len(country_list) - 1)]  # selecting the country
by brute force
      country_final = country
      flag = True
      if country not in countries_visited:
        countries_visited.append(country_final)
        break
  flightlist = list(flights.objects.values_list('name', 'cost_per_head', 'clas', 'country'))  # fetching
data from database
  if val2.budget == 'low':
    flight_obj = [i for i in flightlist if i[2] == 'Economy Class' and i[3] == country_final]  #
deciding flight details based on budget
  elif val2.budget == 'med':
    flight_obj = [i for i in flightlist if
            (i[2] == 'Business Class' or i[2] == 'Economy Class') and i[3] == country_final]
  else:
    flight_obj = [i for i in flightlist if
            i[2] == ('Business Class' or i[2] == 'First Class') and i[3] == country_final]
  flight = flight_obj[random.randint(0, len(flight_obj) - 1)]
  flight_dict = {'Name': flight[0], 'Class': flight[2], 'Cost_per_head': flight[1]}
```

```
  flightcost = val3.ppl * flight[1]  # flight costs, multiply by number of seats
  return country_final, flight_dict, flightcost, flag

def f2(val1, val2, val3, val4, country, flag):
  city_val = list(city.objects.values_list('city_code', 'city_name', 'country'))  # fetching data
from database
  cities_in_country = [[i[0], i[1]] for i in city_val if
                    i[2] == country]  # list of cities in the country:[city code, city name]
  print(country)
  info_city = tf.get_dict(cities_in_country)  # {city name: city code}
  codelist = [i[0] for i in cities_in_country]  # list of city codes
  attr_list = tf.get_types(val3.type)  # list of attraction types preferred by users
  if flag:
    attr_list = random.sample(['nat', 'her', 'mon', 'bea', 'mus', 'wil', 'pil', 'adv', 'amu'],
                              4)  # attraction types if country is chosen by brute force
  f_dict = {}
  acc_val = list(acc.objects.values_list('city_id', 'name', 'type', 'cost_per_night'))  # fetching
data from database
  acclist = [{'name': a[1], 'type': tf.get_acctype(a[2]), 'price': a[3], 'id': a[0]} for a in acc_val if
          a[0] in codelist]  # reorganizing fetched data into dict
  acc_cost = 0  # total cost of accommodation
  for i in acclist:
    acc_cost += math.ceil((i['price']) * val3.rooms / 2)
  country_info, city_info = tf.queries()  # fetching data from database
  for i in cities_in_country:
    c_dict = {}  # attraction info (only valid ones)
    valid_acc = [q for q in acclist if q['id'] == info_city[i[1]]]  # accommodations in city per city in
country
    accc = valid_acc[random.randint(0, len(valid_acc) - 1)]  # choosing accommodation for a
particular city
    adict = city_info[i[1]]
    for k, v in adict.items():
      if v in attr_list:
        c_dict[k] = v
      else:
        pass
    f_dict[i[1]] = {'attrs': c_dict, 'accs': accc}  # combining attraction and accommodation
details
  temp = {}
  for k, v in f_dict.items():
    if len(v['attrs']) != 0:  # eliminating empty dicts if there are no attractions in any city
      temp[k] = v
    else:
      pass
  f_dict = temp
```

```
  cities = list(f_dict.keys())
  ff = {}
  for i in cities:
    ff[i] = f_dict[i]
  f_dict = ff
  f_dict = tf.shorten_timeline(f_dict)  # restricting length of trip
  k = list(f_dict.keys())
  print('k', k)
  v = list(f_dict.values())
  ax = [i['accs'] for i in v]
  desc_val = list(attractions.objects.values_list('name', 'desc', 'cost'))  # fetching data from
database
  desc_dict = {i[0]: i[1] for i in desc_val}  # description of attractions
  new_desc = []
  for cit in v:
    tempd = {}
    for att in (cit['attrs']).keys():
      tempd[att] = {'desc': desc_dict[att], 'img': tf.get_image(att)}  # complete attraction
details
    new_desc.append(tempd)  # list of attrs per city for all cities in the country
  subdict = [{'attrs': new_desc[i], 'accs': ax[i]} for i in range(len(new_desc))]
  final_dict = dict(zip(k, subdict))  # creating final itinerary dict
  print('new', new_desc)
  edit_desc = copy.deepcopy(new_desc)  # list of dicts of dicts, each elem is a city, sub-elem is
an attraction
  print('edit', edit_desc)
  attcost = 0
  for cc in new_desc:
    for atr in cc:
      tempv = attractions.objects.get(name=atr)
      attcost += tempv.cost * val3.ppl
  total = attcost + acc_cost  # adding accommodation and attraction costs
  t_info, last = timeline_info(new_desc, val1, val2, val4)  # adding timeline info
  for j in new_desc:
    for ii, jj in j.items():
      jj['day'] = t_info[ii]['day']
      jj['date'] = t_info[ii]['date']
      jj['time'] = t_info[ii]['time']
      jj['mode'] = t_info[ii]['mode']
  return final_dict, total, last, edit_desc, ax

def timeline_info(dictt, val1, val2, val4):
  attrlist = []  # list of all attractions chosen (whole country)
  start_date = val2.start_date  # fetching start date
  rh = val1.type  # type of scheduling (relaxed/hectic)
```

```
  tranlist = tf.tran_func(val4.tran)  # transport between cities
  tran_val = list(transport.objects.values_list('name', 'type', 'clas'))
  tranlist1 = [i for i in tran_val if i[1] in tranlist]
  for i in dictt:
     attrlist.extend(list(i.keys()))
  if rh == 'rel':  # scheduling of attractions for relaxed trip
     info = {}
     for i in range(1, len(attrlist) + 1):
        att = attractions.objects.get(name=attrlist[i - 1])
        tran = tranlist1[random.randint(0, len(tranlist1) - 1)]
        if tran[2] is None:
           tran = list(tran)
           tran[2] = 'General'
            tran = tuple(tran)
        if att.time == 'mor':
           time = 'Morning'
         else:
           time = 'Evening'
         info[att.name] = {'day': i,
                           'date': start_date + datetime.timedelta(days=i, seconds=0, microseconds=0,
milliseconds=0,
                                                                   minutes=0, hours=0, weeks=0),
                           'time': time,
                           'mode': {'tran_name': tran[0], 'type': tran[1], 'clas': tran[2]}
                           }
     last = start_date + datetime.timedelta(days=i + 1, seconds=0, microseconds=0,
milliseconds=0, minutes=0,
                                          hours=0, weeks=0)  # end date
  else:  # scheduling of attractions for hectic trip
     info = {}
     for i in range(1, len(attrlist) + 1):
        att = attractions.objects.get(name=attrlist[i - 1])
        tran = tranlist1[random.randint(0, len(tranlist1) - 1)]
        if tran[2] is None:
           tran = list(tran)
           tran[2] = 'General'
            tran = tuple(tran)
        if att.time == 'mor':
           time = 'Morning'
         else:
           time = 'Evening'
         info[att.name] = {'day': math.ceil(i / 2),
                           'date': start_date + datetime.timedelta(days=math.ceil(i / 2), seconds=0,
microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0),
                           'time': time,
```

```
                    'mode': {'tran_name': tran[0], 'type': tran[1], 'clas': tran[2]}
                    }
    last = start_date + datetime.timedelta(days=math.ceil(i / 2) + 1, seconds=0,
microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)  # end date
  return info, last

def delete(request):
  if request.method == 'GET':
    it_list = []
    with open("itinerary_files.dat", "rb") as handle:
      while True:
        try:
          itinerary = pickle.load(handle)
          it_list.append(itinerary)
        except EOFError:
          break
    return render(request, 'delete.html', {'ilist': it_list})  # redirects to deletion page
  elif request.method == 'POST':
    trip = request.POST['deleted_itinerary']  # fetching user input (deleted itinerary)
    if request.GET['d'] == 'delete':
      cty = trip.split()[2]
      fh1 = open("itinerary_files.dat", "rb") # removing itinerary from itinerary_files.dat by
temporary file method
      temp = open("temp.dat", "wb")
      while True:
        try:
          itinerary = pickle.load(fh1)
          if itinerary['id'] == request.user.id and itinerary['country'] != cty:
            pickle.dump(itinerary, temp)
          else:
            pass
        except EOFError:
          break
      fh1.close()
      temp.close()
      os.remove('itinerary_files.dat')
      os.rename('temp.dat', 'itinerary_files.dat')
      itinerary_list = []
      logged_in = request.user.first_name
      with open("itinerary_files.dat", "rb") as handle:  # fetching itineraries already finalized
by user
        while True:
          try:
            itinerary = pickle.load(handle)
            if itinerary['id'] == request.user.id:
```

```
                itinerary_list.append(itinerary)
            else:
                pass
        except EOFError:
            break
    return render(request, 'itinlist.html',
                 {'lst': itinerary_list, 'len': len(itinerary_list), 'logged_in': logged_in})  # returning to
itinerary list
  else:
    return render(request, 'city_error.html')

def edit(request):
  if request.method == 'POST':
    if request.GET['e'] == 'edit':
      dit_attrs = request.POST.getlist('edit')  # fetching list of deleted attractions
      s = dit_attrs[0]
      edit_attrs = []  # list of attraction without (Country)
      for i in dit_attrs:
        ls = i.split()
        word = ''
         for j in range(0, len(ls) - 1):
          word += (ls[j] + ' ')
        edit_attrs.append(word[:-1])
      cost_red = 0
      val3 = s3.objects.get(user_id=request.user.id)
      for atr in edit_attrs:
        tempv = attractions.objects.get(name=atr)
        cost_red += tempv.cost * val3.ppl  # cost to be reduced
      country = s.split()[-1][1:len(s.split()[-1]) - 1]  # fetching country name
      with open("itinerary_files.dat", "rb") as handle:  # retrieving itinerary for editing
        while True:
          try:
            itinerary = pickle.load(handle)
            if itinerary['id'] == request.user.id and itinerary['country'] == country:
              fin_dict = itinerary
            else:
              pass
          except EOFError:
            break
      val1 = s1.objects.get(user_id=request.user.id)  # user inputs
      val2 = s2.objects.get(user_id=request.user.id)
      val4 = s4.objects.get(user_id=request.user.id)
      newdesc = fin_dict['new']
      axx = fin_dict['ax']
```

```
        city_val = list(city.objects.values_list('city_code', 'city_name', 'country'))  # fetching data
from database
        k = [i[1] for i in city_val if
            i[2] == country]  # list of cities in the country:[city code, city name]
        intdesc = []
        for cur_city in newdesc:
            for attr in edit_attrs:
                try:
                    del cur_city[attr]  # deleting selected attractions
                except KeyError:
                    pass
            intdesc.append(cur_city)
        finaldesc = []
        for thing in intdesc:
            if len(thing) != 0:
                finaldesc.append(thing)
            else:
                axx.pop(intdesc.index(thing))
                k.pop(intdesc.index(thing))
        subdict = [{'attrs': finaldesc[i], 'accs': axx[i]} for i in range(len(finaldesc))]
        final_dict = dict(zip(k, subdict))
        t_info, last = timeline_info(finaldesc, val1, val2, val4)  # reorganising timeline info
        for j in finaldesc:
            for ii, jj in j.items():
                jj['day'] = t_info[ii]['day']
                jj['date'] = t_info[ii]['date']
                jj['time'] = t_info[ii]['time']
                jj['mode'] = t_info[ii]['mode']
        fh1 = open("itinerary_files.dat", "rb")
        temp = open("temp.dat", "wb")
        while True:
            try:
                itinerary = pickle.load(fh1)
                if itinerary['id'] == request.user.id and itinerary['country'] == country:
                    itinerary['dict'] = final_dict
                    itinerary['last'] = last
                    itinerary['cost'] = itinerary['cost'] - cost_red  # reducing the cost
                    pickle.dump(itinerary, temp)
                else:
                    pickle.dump(itinerary, temp)
            except EOFError:
                break
        fh1.close()
        temp.close()
        os.remove('itinerary_files.dat')
```

```
        os.rename('temp.dat', 'itinerary_files.dat')
        itinerary_list = []
        logged_in = request.user.first_name
        with open("itinerary_files.dat", "rb") as handle:
            while True:
                try:
                    itinerary = pickle.load(handle)
                    if itinerary['id'] == request.user.id:
                        itinerary_list.append(itinerary)
                    else:
                        pass
                except EOFError:
                    break
        return render(request, 'itinlist.html', {'lst': itinerary_list,  # returning to itinerary list
                                                 'len': len(itinerary_list),
                                                 'logged_in': logged_in
                                                 })
    else:
        return render(request, 'itinlist.html')
else:
    return render(request, 'itinlist.html')
```

tip_funcs.py:

```
def get_country(weather, d):
    country_list = list(d.keys())
    col_list = [i for i in country_list if d[i] == 'col']  # list of cold countries
    hot_list = [i for i in country_list if d[i] == 'hot']  # list of hot countries
    mod_list = [i for i in country_list if d[i] == 'mod']  # list of moderate countries
    if weather == 'col':
        return col_list
    elif weather == 'hot':
        return hot_list
    elif weather == 'mod':
        return mod_list
    else:
        Return
def city_validate(city, attr_list, city_dict):
    adict = city_dict[city]  # attraction info per city in the form: {'attr': 'type'}
    alist = list(adict.values())  # fetching their types
    tr, fa = 0, 0
    for i in alist:
        if i in attr_list:  # preferred attraction types
            tr += 1
        else:
            fa += 1
    if tr >= fa:  # if there are more valid than invalid attractions, city is valid
```

```
        return True
    else:
        return False
def country_validate(cities_in_country, attr_list, city_dict):
    bool_list = []
    for i in cities_in_country:  # validating cities within the country
        bool_list.append(city_validate(i, attr_list, city_dict))
    if bool_list.count(True) > bool_list.count(False):  # if there are more valid than invalid cities, country is valid
        return True
    else:
        Return False
```

urls.py:

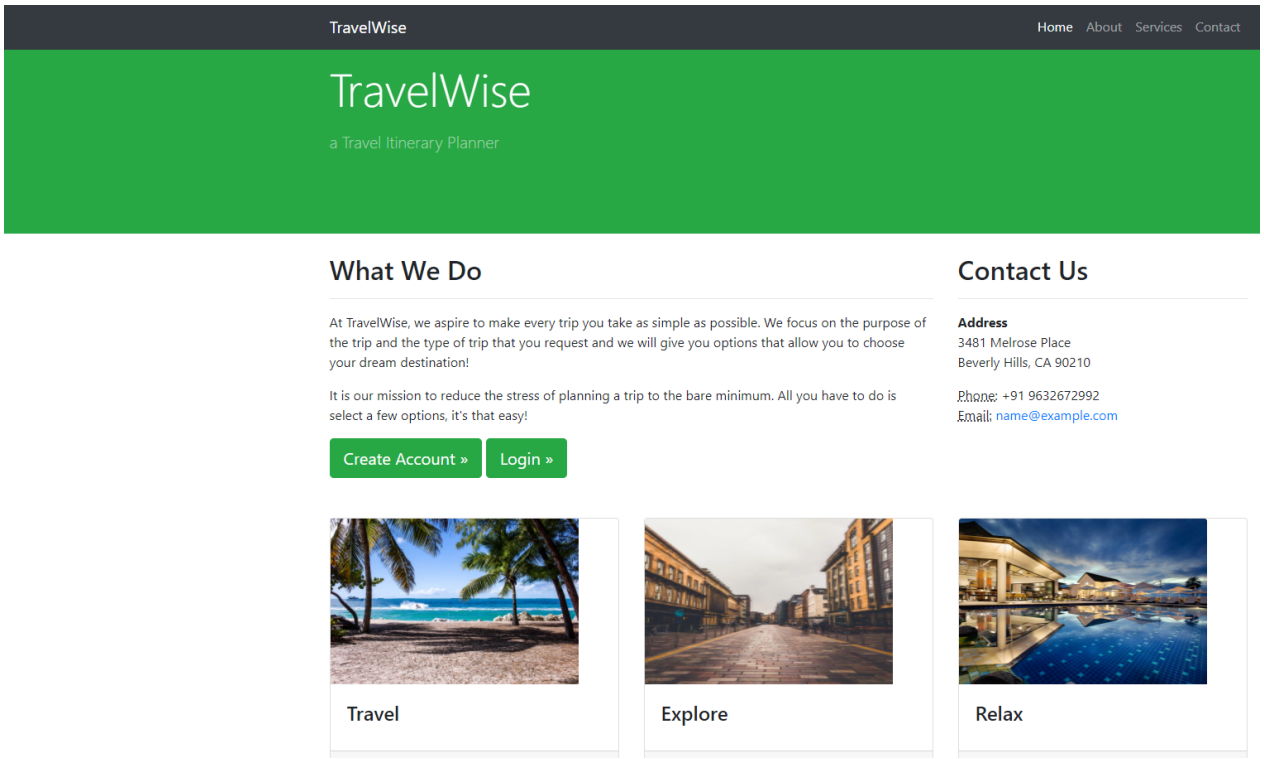
```
from django.contrib import admin
from django.urls import path
from addUser import views as addviews
from plan1 import views as plan1views
from final import views as finalviews

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', addviews.index),
    path('add', addviews.add_User),
    path('login', addviews.login),
    path('plan_1', plan1views.plan_1),
    path('plan_2', plan1views.plan_2),
    path('plan_3', plan1views.plan_3),
    path('plan_4', plan1views.plan_4),
    path('final_it', finalviews.show),
    path('final2', finalviews.show2),
    path('final3', finalviews.show3),
    path('prev', finalviews.itin_list),
    path('del', finalviews.delete),
    path('edit', finalviews.edit),
    path('USA', finalviews.USA),
    path('Canada', finalviews.Canada),
    path('India', finalviews.India),
    path('England', finalviews.England),
    path('Spain', finalviews.Spain),
    path('Italy', finalviews.Italy),
    path('France', finalviews.France),
    path('Russia', finalviews.Russia),
    path('Sweden', finalviews.Sweden),
```

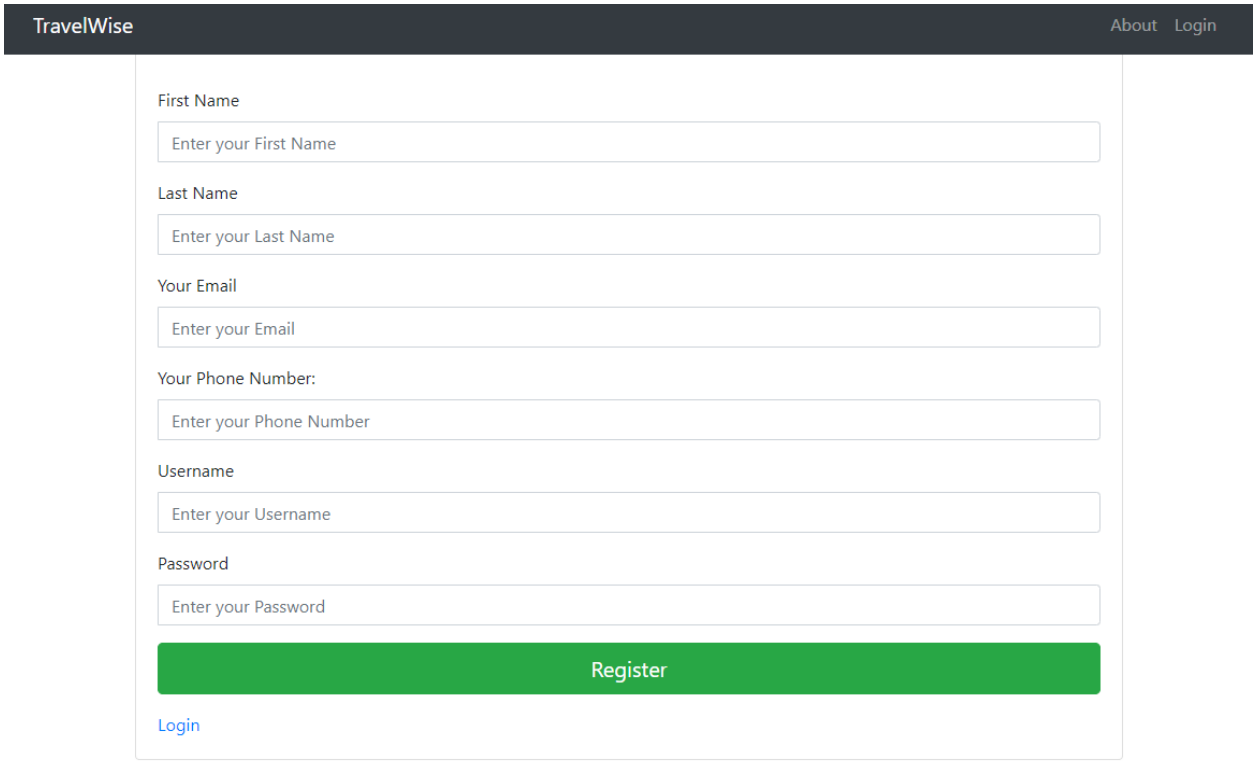
```
    path('Germany', finalviews.Germany),
    path('Norway', finalviews.Norway),
    path('Switzerland', finalviews.Switzerland),
    path('Netherlands', finalviews.Netherlands),
    path('Belgium', finalviews.Belgium),
    path('Denmark', finalviews.Denmark),
    path('Japan', finalviews.Japan),
    path('China', finalviews.China),
    path('Singapore', finalviews.Singapore),
    path('UAE', finalviews.UAE),
    path('Egypt', finalviews.Egypt),
    path('Morocco', finalviews.Morocco),
    path('Mexico', finalviews.Mexico),
    path('Australia', finalviews.Australia),
    path('NewZealand', finalviews.NZ),
    path('USA_edit', finalviews.USA_edit),
    path('Canada_edit', finalviews.Canada_edit),
    path('India_edit', finalviews.India_edit),
    path('England_edit', finalviews.England_edit),
    path('Spain_edit', finalviews.Spain_edit),
    path('Italy_edit', finalviews.Italy_edit),
    path('France_edit', finalviews.France_edit),
    path('Russia_edit', finalviews.Russia_edit),
    path('Sweden_edit', finalviews.Sweden_edit),
    path('Germany_edit', finalviews.Germany_edit),
    path('Norway_edit', finalviews.Norway_edit),
    path('Switzerland_edit', finalviews.Switzerland_edit),
    path('Netherlands_edit', finalviews.Netherlands_edit),
    path('Belgium_edit', finalviews.Belgium_edit),
    path('Denmark_edit', finalviews.Denmark_edit),
    path('Japan_edit', finalviews.Japan_edit),
    path('China_edit', finalviews.China_edit),
    path('Singapore_edit', finalviews.Singapore_edit),
    path('UAE_edit', finalviews.UAE_edit),
    path('Egypt_edit', finalviews.Egypt_edit),
    path('Morocco_edit', finalviews.Morocco_edit),
    path('Mexico_edit', finalviews.Mexico_edit),
    path('Australia_edit', finalviews.Australia_edit),
    path('NewZealand_edit', finalviews.NZ_edit),
]
```

Output Screenshots

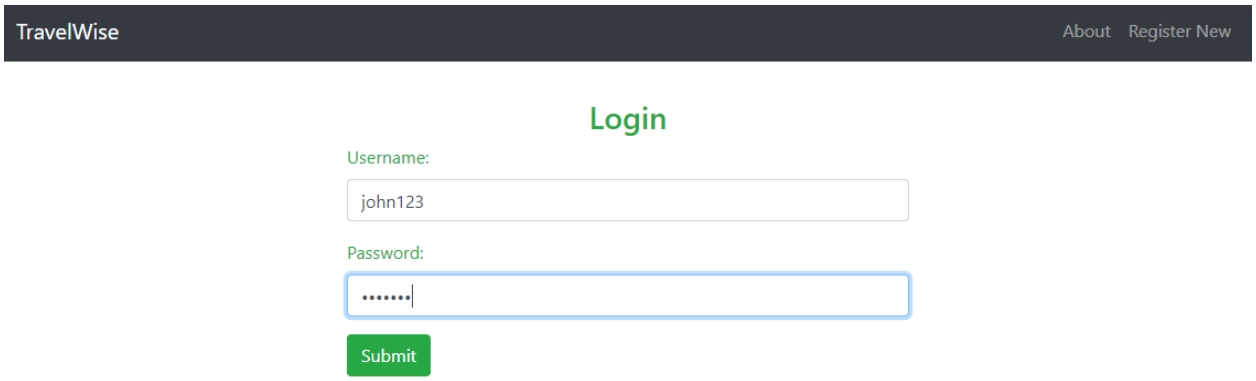
Landing Page:



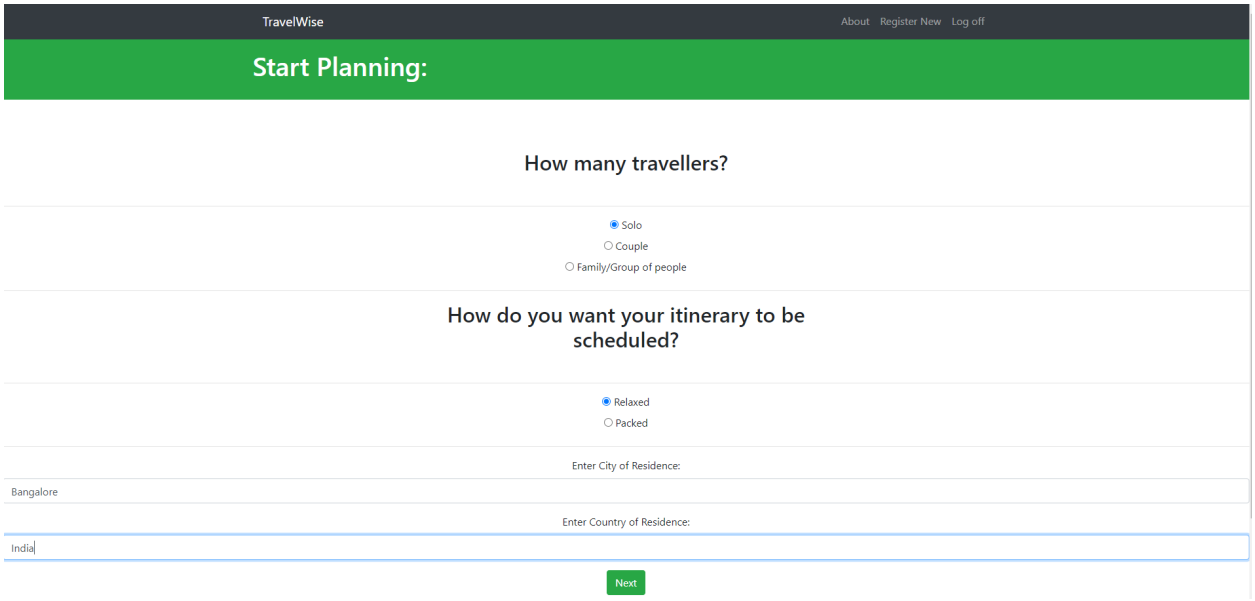
Create user:

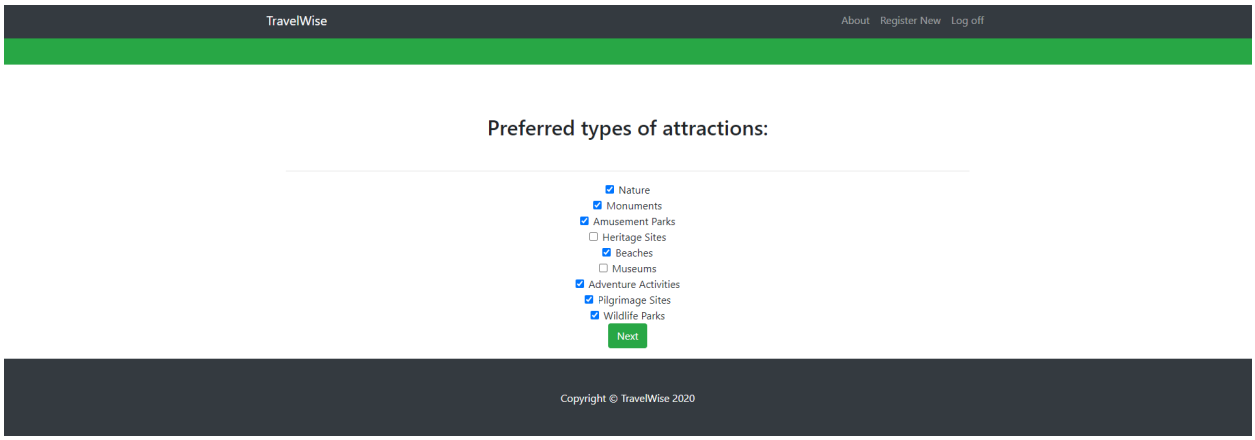


Login:



Planning pages:





Itinerary options:

TravelWise

About Register New Log off

John, here are your itinerary options:

Itinerary 1: Trip to Germany

Flight details: Lufthansa from Bangalore to Munich
(Economy Class)
Arrival: Jan. 30, 2021

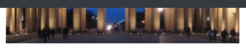
Munich

Accomodation details:
Hotel: Hotel NH München City Süd
Price: $50 per night

Time	Mode of Transport	Attraction	Description	Picture
Day 1: Jan. 31, 2021 Morning	EZbuses (bus) General	Frauenkirche	The Frauenkirche is a church in Munich, Bavaria, Germany.	

TravelWise

About Register New Log off

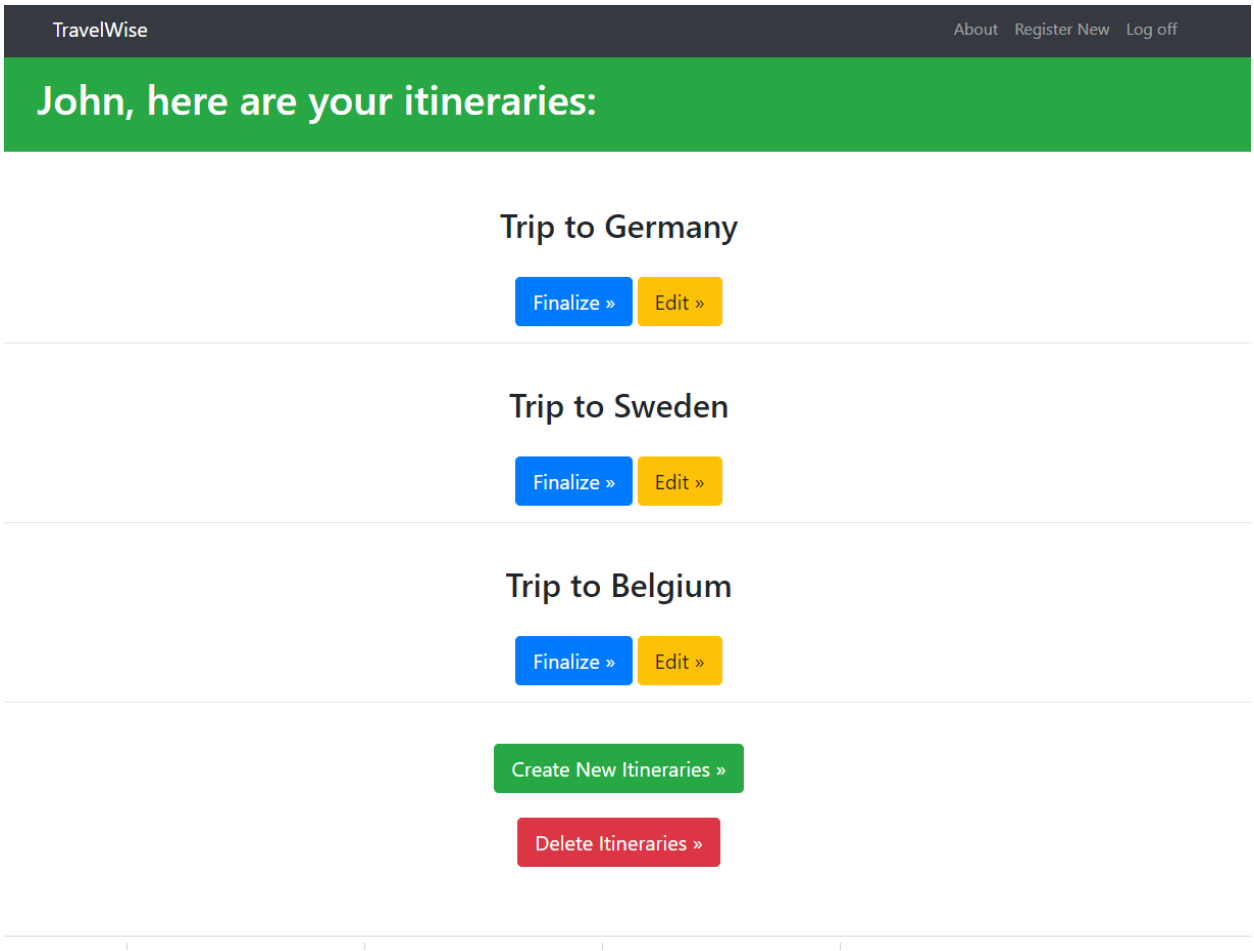
				
Day 11: Feb. 10, 2021 Evening	Transit 1 (train) Second Class	The Britzer Garten	The Britzer Garten is a large park in the south of Berlin, named after Britz, a neighborhood of the borough of Neukolln.	
Day 12: Feb. 11, 2021 Evening	Transit 1 (train) Second Class	The Tiergarten	The Tiergarten is Berlins most popular inner-city park, located completely in the district of the same name.	

Return Flight: Lufthansa from Munich to Bangalore
(Economy Class)
Arrival: Feb. 12, 2021

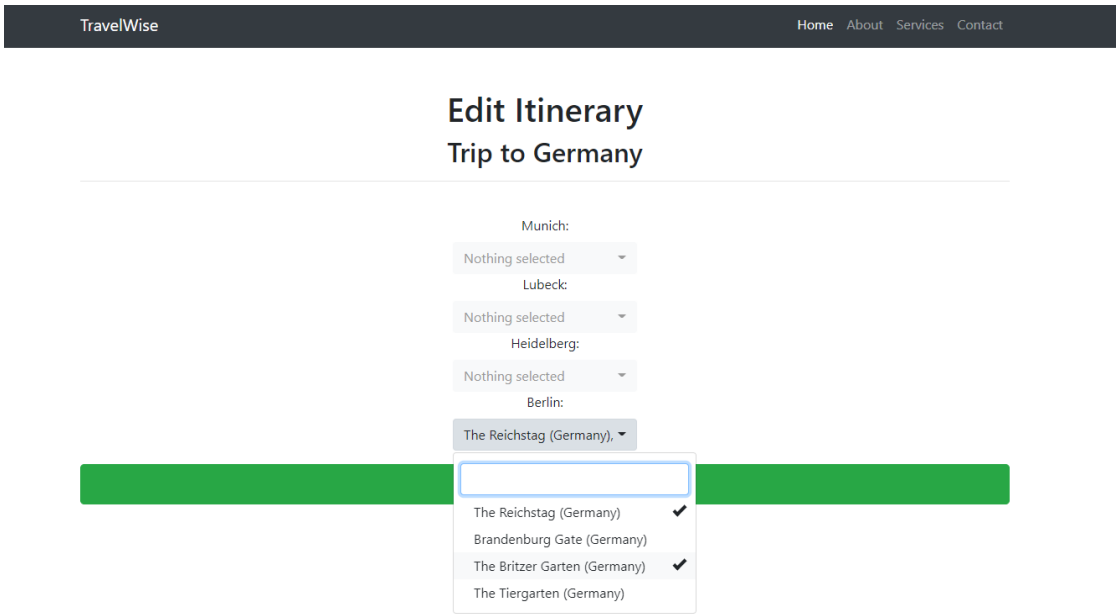
Total cost: $1584.00 (USD)

View Next »

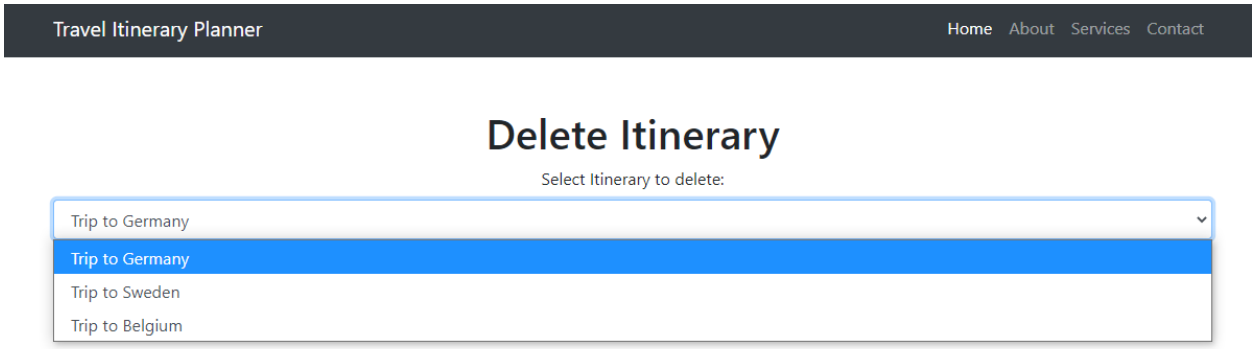
Itinerary list:



Edit:


TravelWise
Home About Services Contact
Edit Itinerary
Trip to Germany
Munich:
Nothing selected
Lubeck:
Nothing selected
Heidelberg:
Nothing selected
Berlin:
The Reichstag (Germany),
The Reichstag (Germany)
Brandenburg Gate (Germany)
The Britzer Garten (Germany)
The Tiergarten (Germany)

Delete:


Travel Itinerary Planner
Home About Services Contact
Delete Itinerary
Select Itinerary to delete:
Trip to Germany
Trip to Germany
Trip to Sweden
Trip to Belgium

Scope for Improvement

1. Trips to multiple countries cannot be planned.

2. The prices of transportation are not dependent on distance between cities or countries.

3. Payment and booking options can be included.

4. Weather of individual cities is not considered while planning.

5. User interface can be improved.

Bibliography

1. docs.djangoproject.com
2. www.geeksforgeeks.org
3. docs.python.org
4. stackoverflow.com
5. www.w3schools.com
6. getbootstrap.com
7. startbootstrap.com
8. en.wikipedia.org
9. themeforest.net
10. freshdesignweb.com
11. bootsnipp.com
12. templatemo.com
13. pynative.com
14. www.themezy.com