FLIGHT BOOKING SYSTEM

15IT322E - MINOR PROJECT REPORT

Submitted by

AKSHAY KUMAR(RA1511003010552)

for the course

15IT322E – Python Programming

in partial fulfillment for the award of the degree

of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY

KATTANKULATHUR

OCTOBER 2018

SRM UNIVERSITY

KATTANKULATHUR

BONAFIDE CERTIFICATE

Certified that this project report **"FLIGHT BOOKING SYSTEM"** is the bonafide work of "**AKSHAY KUMAR (RA1511003010552) "** who carried out the project work as part of their course **15IT322E - Python Programming**.

SIGNATURE

Course Instructor

SIGNATURE

Dr. B. Amutha

HEAD OF THE DEPARTMENT
Computer Science & Engg

INTERNAL EXAMINER

ABSTRACT

The project is a basic online flight booking system which uses Python scraping modules to scrape data off flight booking websites, show them to the user in GUI based program view and then the user can book a particular flight by filling the required parameters which will show up all the flights satisfying the parameters and the user will select the flight according to his/her needs and it will be stored in a database. This program uses selenium which is a browser automation testing package. Using selenium, a simulation of google chrome browser is created and then beautifulsoup, a python web scraping package is used to scrape flight data off google flights websites. The inputs are complete names of cities to be traveled, date of journey and number of passengers. The GUI contains a list box that helps user in writing the correct name. The list box only shows the correct name, that cannot be selected from the list. After filling in the parameters, the flight details are collected and then all the possible flights for the given parameter are shown and a flight is selected and using sqlite3, a python database package , the flight booked is stored in a database. The GUI built is by using tkinter, a python GUI package. Overall, the project is a simple flight booking system using selenium, beautifulsoup, tkinter and sqlite3 python packages.

INTRODUCTION

A computer reservation system is used for the reservations of a particular airline and interfaces with a global distribution system (GDS) which supports travel agencies and other distribution channels in making reservations for most major airlines in a single system. Based on this idea, this project was developed to showcase the usage of python programming and how easily its packages can be used to create a flight booking system which takes input parameters from the user and then takes data from the web and then based on the user's selection, it stores the data in a database. The project showcases some of the basic scraping, browser simulation and database management techniques which can be used in python. These techniques are very relevant in many of the modern web applications and that is why this project was taken up so that these interesting aspects of python programming could be learnt. Although, the project is pretty basic and simple, it does provide as a platform for one to learn about web scraping and database management.

RELATED WORK

This project is an improvement over most of the projects which are found online for airline reservation system as this project uses real time data to show the results using multiple features of python programming like web scraping, browser simulation, GUI creation and database management. One of the projects, which this project is slightly based on but is an improved version of it is airline booking system by Hieu Do and Michael Chen (https://github.com/hieusydo/Airline-Reservation-System).

Improvements:

1. Real time data retrieval
2. Dynamic database creation for flight information
3. Web scraping for airport information retrieval
4. Used SQLite3 rather than MySQL as SQLite3 is lightweight unlike MySQL which is heavy.
5. Proper GUI based project, program interface isn't used for booking

MODEL

This project consists of all the basic modules required to make an online reservation system. The modules in this project are namely:

- ➢ Database creation module
- ➢ Web scraping module
- ➢ Browser simulation module
- ➢ GUI module

1. Database creation module: This module is the most important as it creates database for storing flight details and airport codes and cities.

```python
def make_IATA_database():
    """
    Method to create the database of all available IATA coded airports. This method is to be called only once
    and the database created will act as a base database to search destinations.
    :return None:
    """
    database = sqlite3.connect('Flights')
    db = database.cursor()
    query = "CREATE TABLE IF NOT EXISTS IATA(CODE VARCHAR(3) PRIMARY KEY, CITY VARCHAR UNIQUE, NAME VARCHAR,COUNTRY VARCHAR)"
    db.execute(query)
```

Table
- BOOKED
- FLIGHTS
- IATA

View
Index
Trigger
Syntax

```
1 SELECT * FROM 'IATA';
```

CODE	CITY	NAME	COUNTRY
AAA	Anaa	Anaa	French Polynesia
AAB	Arrabury	Arrabury	Australia
AAC	Arish	El Arish International Airport	Egypt
AAD	Ad-Dabbah	Ad-Dabbah	Sudan
AAE	Annaba	Les Salines	Algeria
AAF	FL	Municipal	USA
AAG	Arapoti	Arapoti	Brazil
AAH	Aachen	Aachen/Merzbrück	Germany
AAI	Arraias	Arraias	Brazil
AAJ	Awaradam	Cayana Airstrip	Suriname

2. Web scraping module: This module scrapes data off from google flights and airports list using beautifulsoup python web scraping package.

```
alpha = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
for x in alpha:
    url = r'http://www.nationsonline.org/oneworld/IATA_Codes/IATA_Code_{}.htm'.format(x)
    source_code = requests.get(url, headers=headers, verify=True).text
    soup = BeautifulSoup(source_code, 'html.parser')
    for code in soup.findAll('tr'):
        c = 0
        t = tuple()
        for info in code.findAll('td', {'class': 'border1'}):
            if info.string == None and c == 0:
                break
            elif info.string == None and c == 1:
                try:
                    t += (rabsolute(info.findAll('a')),)
                except TypeError or AttributeError:
                    t += (findname(t[0]),)

            else:
                t += (info.string,)
            c += 1
        if len(t) == 4 and len(t[0]) == 3:
            # print(t)
            db.execute("INSERT OR REPLACE INTO IATA VALUES(?,?,?,?)", t)
database.commit()
database.close()
```

3. Browser Simulation Module: Browser simulation is done to retrieve flight data using selenium package.


```
alpha = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
for x in alpha:
    url = r'http://www.nationsonline.org/oneworld/IATA_Codes/IATA_Code_{}.htm'.format(x)
    source_code = requests.get(url, headers=headers, verify=True).text
    soup = BeautifulSoup(source_code, 'html.parser')
    for code in soup.findAll('tr'):
        c = 0
        t = tuple()
        for info in code.findAll('td', {'class': 'border1'}):
            if info.string == None and c == 0:
                break
            elif info.string == None and c == 1:
                try:
                    t += (rabsolute(info.findAll('a')),)
                except TypeError or AttributeError:
                    t += (findname(t[0]),)

            else:
                t += (info.string,)
            c += 1
        if len(t) == 4 and len(t[0]) == 3:
            # print(t)
            db.execute("INSERT OR REPLACE INTO IATA VALUES(?,?,?,?)", t)
database.commit()
database.close()
```

4. GUI Module: A GUI is important to interact with user and tkinter is used to create GUI in this project.


```
Label1 = tk.Label(root, text="Source ")
Label1.grid(row=0, column=0)
entry1 = tk.Entry(root)
entry1.grid(row=0, column=1)
entry1.bind('<KeyRelease>', on_keyrelease1)

Label2 = tk.Label(root, text="Destination ")
Label2.grid(row=0, column=3)
entry2 = tk.Entry(root)
entry2.grid(row=0, column=4)
entry2.bind('<KeyRelease>', on_keyrelease2)

listbox1 = tk.Listbox(root)
listbox2 = tk.Listbox(root)
listbox1.grid(row=1, column=1)
listbox2.grid(row=1, column=4)

listbox1_update(city_list)
listbox2_update(city_list)

Label3 = tk.Label(root, text="Date of Journey\n(YYYY-MM-DD)")
Label3.grid(row=0, column=5)
entry3 = tk.Entry(root)
entry3.grid(row=0, column=6)

Label4 = tk.Label(root, text="Passengers")
Label4.grid(row=1, column=5, sticky='N')
entry4 = tk.Entry(root)
entry4.grid(row=1, column=6, sticky='N')

search = tk.Button(root, text="SEARCH FLIGHTS", justify='left', padx=2, width=20, command=searchflight)
search.grid(row=3, column=6, sticky='NW')
```

Available Flights

Flight Name	Departure	Arrival	Duration	Price
Jet Airways 9W798	2:00 PM	5:00 PM	5.92	6031
Jet Airways 9W845	5:35 PM	8:30 PM	4.75	6629
SpiceJet SG105	8:15 PM	11:20 PM	3.08	3787
IndiGo 6E2162	8:35 PM	11:25 PM	2.83	3953
Jet Airways 9W739	9:15 PM	12:10 AM	2.92	4656
Air India AI429	9:45 AM	12:40 PM	2.92	4755
SpiceJet SG107	6:10 AM	9:00 AM	2.83	4501
IndiGo 6E2985	6:30 AM	9:20 AM	2.83	4528
IndiGo 6E2315	4:20 PM	7:15 PM	2.92	4900
IndiGo 6E2694	6:05 PM	8:55 PM	2.83	4900

BOOK FLIGHT

PROGRAM

flightdatabase.py

```
import sqlite3
import requests

from bs4 import BeautifulSoup
import time
from dateutil.parser import parse

from selenium import webdriver
from selenium.webdriver.support.ui import
WebDriverWait
from selenium.webdriver.support import
expected_conditions as EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import
TimeoutException

headers = {
    'User-Agent':
        'Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/42.0.2311.90 Safari/537.36'}


def clean(s):
    """
    Method to remove Titles from given string based
on requirement.
    :param s: Input string
    :return s: Modified string
    """
    return s[s.index(':') + 2:-4]


def rabsolute(s):
    """
    Method to concatenate string parts of a HTML tag
    :param s: HTML Tags
    :return k: String
    """
    k = ""
    for x in s:
        k += x.string
    return k


def findname(code):
    """
    Method to find name of the city. Some cities were
missing from the website database.
    :param code:  IATA code
    :return city: Name of the city
    """
    url = r'https://airports-list.com/airport/' + code
    source_code = requests.get(url, headers=headers,
verify=True).text
    soup = BeautifulSoup(source_code, 'html.parser')
    details = soup.find('div', {'class': 'view-content'})
    city = clean(str(details.find('div', {'class':
'views-field views-field-field-gorod-eng'}).find('p')))
    return city


def make_IATA_database():
    """
    Method to create the database of all available
IATA coded airports. This method is to be called
only once
    and the database created will act as a base database
to search destinations.
    :return None:
    """
    database = sqlite3.connect('Flights')
    db = database.cursor()
    query = "CREATE TABLE IF NOT EXISTS
IATA(CODE VARCHAR(3) PRIMARY KEY,
CITY VARCHAR UNIQUE, NAME
VARCHAR,COUNTRY VARCHAR)"
    db.execute(query)

    alpha =
list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
    for x in alpha:
        url =
r'http://www.nationsonline.org/oneworld/IATA_Cod
es/IATA_Code_{}.htm'.format(x)
        source_code = requests.get(url,
headers=headers, verify=True).text
        soup = BeautifulSoup(source_code,
'html.parser')
        for code in soup.findAll('tr'):
            c = 0
            t = tuple()
            for info in code.findAll('td', {'class':
'border1'}):
                if info.string == None and c == 0:
```

```
                break
            elif info.string == None and c == 1:
                try:
                    t += (rabsolute(info.findAll('a')),)
                except TypeError or AttributeError:
                    t += (findname(t[0]),)

            else:
                t += (info.string,)
            c += 1
        if len(t) == 4 and len(t[0]) == 3:
            # print(t)
            db.execute("INSERT OR REPLACE INTO
IATA VALUES(?,?,?,?)", t)
    database.commit()
    database.close()


# make_IATA_database()


def get_segment_data(segment):
    """
    This method extracts the reqired data from the
HTML tag of google flights website.
    Information like Departure time, Airport name,
Arrival time, Airline code, Flight number etc.
    The nomenclature is easy enough to understand.
    :param segment: '.gws-flights-results__leg' object
from BeautifulSoup
    :return data: Dictionary containing data
    """
    data = {}
    departure_data = list(
        segment.find(

class_='gws-flights-results__leg-departure').stripped_
strings)
    data['dep_time'] = departure_data[0]
    data['dep_airport_long'] = departure_data[1]
    data['dep_airport_code'] = departure_data[2]

    arrival_data = list(
        segment.find(

class_='gws-flights-results__leg-arrival').stripped_stri
ngs)
    if len(arrival_data) == 3:
        data['arr_time'] = arrival_data[0]
        data['arr_airport_long'] = arrival_data[1]
        data['arr_airport_code'] = arrival_data[2]
    else:
        data['arr_time'] = arrival_data[0]
        data['arr_airport_long'] = arrival_data[2]
        data['arr_airport_code'] = arrival_data[3]

    flight_data = list(
        segment.find(

class_='gws-flights-results__leg-flight').stripped_stri
ngs)
    data['airline'] = flight_data[0]
    data['seat_class'] = flight_data[1]
    data['airplane'] = flight_data[2]
    data['airline_code'] = flight_data[3]
    try:
        data['flight_number'] = flight_data[4] +
flight_data[5]
    except IndexError:
        data['flight_number'] = flight_data[3] +
flight_data[4]

    return data


def scrape(url):
    """
    Driver for scraping. Contains code to save the
collected information into the database. Contains the
main crawler.
    Using Selenium and BeautifulSoup4 to collect data
from website and parsing. Using the web driver
PhantomJS.
    :param url: URL for scraping
    :return:
    """
    # ['flight_date', 'full_duration', 'segments', 'stops',
'flight_number', 'arr_airport_long', 'arr_airport_code',
    # 'dep_time', 'airplane', 'arr_time', 'airline_code',
'dep_airport_code', 'dep_airport_long',
    # 'airline', 'seat_class']
    database = sqlite3.connect('Flights')
    db = database.cursor()
    query = "CREATE TABLE IF NOT EXISTS
FLIGHTS(FLIGHT_NUMBER VARCHAR
PRIMARY KEY,FLIGHT_DATE
DATE,DURATION FLOAT," \
            " STOPS INTEGER,
DEPT_AIRPORT_NAME VARCHAR,
DEPT_AIRPORT_CODE VARCHAR,
DEPART_TIME VARCHAR ,AIRLINE
VARCHAR," \
            "AIRPLANE VARCHAR,ARR_TIME
VARCHAR, ARR_AIRPORT_CODE VARCHAR,
ARR_ARIPORT_NAME VARCHAR," \
            " SEAT VARCHAR,PRICE INTEGER )"
    db.execute(query)
```

```
    # driver = webdriver.PhantomJS('phantomjs.exe')
    options=webdriver.ChromeOptions()
    options.add_argument('headless');
    driver =
webdriver.Chrome(chrome_options=options)
    driver.get(url)
    timeout = 10
    try:
        element_present =
EC.text_to_be_present_in_element(
            (By.CSS_SELECTOR,
'.gws-flights-results__unpriced-airlines'),
            'Prices are not available for: Southwest.
Flights with unavailable prices are at the end of the
list.'
        )
        WebDriverWait(driver,
timeout).until(element_present)
        time.sleep(1)
    except TimeoutException:
        print("Timed out waiting for page to load")

    soup = BeautifulSoup(driver.page_source, 'lxml')
    flights = []
    all_results = soup.find_all(

class_='gws-flights-widgets-expandablecard__body')
    for n in range(len(all_results)):
        dict = {}
        stops = list(
            driver.find_elements_by_css_selector(
                '.gws-flights-results__stops'))[n].text.strip()
        try:
            n_stops = int(stops[0])
        except ValueError:
            n_stops = 0
        dict['stops'] = n_stops

        collapsed_itinerary = list(
            soup.find_all(

class_='gws-flights-results__collapsed-itinerary'))[n]
        full_duration = collapsed_itinerary.find(

class_='gws-flights-results__duration').get_text()
        full_duration = str(full_duration).split(" ")
        hours = int(full_duration[0][:-1])
        try:
            minutes = int(full_duration[1][:-1])
        except IndexError:
            minutes = 0
            hours = hours/60
        dict['full_duration'] = round(hours + (minutes /
60), 2)

        flight_date = list(
            list(
                soup.find_all(
                    class_=

'gws-flights-results__itinerary-details-heading-text'))[
                n].stripped_strings)[1]
        dict['flight_date'] = parse(flight_date).date()

        price = list(
            collapsed_itinerary.find(

class_='gws-flights-results__itinerary-price')
                .stripped_strings)[0][1:]
        price = str(price).replace(',', '')

        dict['price'] = int(price[1:])

        dict['segments'] = []
        segments = list(all_results)[n].find_all(
            class_='gws-flights-results__leg')
        for segment in segments:
            dict['segments'] = get_segment_data(segment)

        flights.append(dict)
        database_tuple =
(dict['segments']['flight_number'], dict['flight_date'],
dict['full_duration'],
                    dict['stops'],
dict['segments']['dep_airport_long'],
                    dict['segments']['dep_airport_code'],
dict['segments']['dep_time'],
                    dict['segments']['airline'],
                    dict['segments']['airplane'],
dict['segments']['arr_time'],
                    dict['segments']['arr_airport_code'],
                    dict['segments']['arr_airport_long'],
dict['segments']['seat_class'], dict['price'])
        # Inserting into database
        db.execute("INSERT OR REPLACE INTO
FLIGHTS VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)",
database_tuple)
    database.commit()
    database.close()
    return


def makeurl(origin, destination, dep_date,
passengers):
    """
    Converts source, destination, date and passengers
into valid flights.google URL
    :param origin: Source passenger
```

```
    :param destination: Destination of passenger
    :param dep_date: Departure Date
    :param passengers: Number of passengers 0<p<10
    :return None:
    """
    url = \
        'https://www.google.com/flights#' \
        + 'flt=' \
        + origin \
        + '.' \
        + destination \
        + '.' \
        + dep_date \
        + ';c:INR' \
        + ';e:1;sd:1;t:f;tt:o'

    if passengers != 1:
        url = url + ';px:' + str(passengers)
    scrape(url)
    return


import sqlite3
import requests

from bs4 import BeautifulSoup
import time
from dateutil.parser import parse

from selenium import webdriver
from selenium.webdriver.support.ui import
WebDriverWait
from selenium.webdriver.support import
expected_conditions as EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import
TimeoutException

headers = {
    'User-Agent':
        'Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/42.0.2311.90 Safari/537.36'}


def clean(s):
    """
    Method to remove Titles from given string based
on requirement.
    :param s: Input string
    :return s: Modified string
    """
    return s[s.index(':') + 2:-4]


def rabsolute(s):
    """
    Method to concatenate string parts of a HTML tag
    :param s: HTML Tags
    :return k: String
    """
    k = ""
    for x in s:
        k += x.string
    return k


def findname(code):
    """
    Method to find name of the city. Some cities were
missing from the website database.
    :param code:  IATA code
    :return city: Name of the city
    """
    url = r'https://airports-list.com/airport/' + code
    source_code = requests.get(url, headers=headers,
verify=True).text
    soup = BeautifulSoup(source_code, 'html.parser')
    details = soup.find('div', {'class': 'view-content'})
    city = clean(str(details.find('div', {'class':
'views-field views-field-field-gorod-eng'}).find('p')))
    return city


def make_IATA_database():
    """
    Method to create the database of all available
IATA coded airports. This method is to be called
only once
    and the database created will act as a base database
to search destinations.
    :return None:
    """
    database = sqlite3.connect('Flights')
    db = database.cursor()
    query = "CREATE TABLE IF NOT EXISTS
IATA(CODE VARCHAR(3) PRIMARY KEY,
CITY VARCHAR UNIQUE, NAME
VARCHAR,COUNTRY VARCHAR)"
    db.execute(query)

    alpha =
list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
    for x in alpha:
        url =
r'http://www.nationsonline.org/oneworld/IATA_Cod
es/IATA_Code_{}.htm'.format(x)
```

```
        source_code = requests.get(url,
headers=headers, verify=True).text
        soup = BeautifulSoup(source_code,
'html.parser')
        for code in soup.findAll('tr'):
            c = 0
            t = tuple()
            for info in code.findAll('td', {'class':
'border1'}):
                if info.string == None and c == 0:
                    break
                elif info.string == None and c == 1:
                    try:
                        t += (rabsolute(info.findAll('a')),)
                    except TypeError or AttributeError:
                        t += (findname(t[0]),)

                else:
                    t += (info.string,)
                c += 1
            if len(t) == 4 and len(t[0]) == 3:
                # print(t)
                db.execute("INSERT OR REPLACE INTO
IATA VALUES(?,?,?,?)", t)
    database.commit()
    database.close()


# make_IATA_database()


def get_segment_data(segment):
    """
    This method extracts the reqired data from the
HTML tag of google flights website.
    Information like Departure time, Airport name,
Arrival time, Airline code, Flight number etc.
    The nomenclature is easy enough to understand.
    :param segment: '.gws-flights-results__leg' object
from BeautifulSoup
    :return data: Dictionary containing data
    """
    data = {}
    departure_data = list(
        segment.find(

class_='gws-flights-results__leg-departure').stripped_
strings)
    data['dep_time'] = departure_data[0]
    data['dep_airport_long'] = departure_data[1]
    data['dep_airport_code'] = departure_data[2]

    arrival_data = list(
        segment.find(
class_='gws-flights-results__leg-arrival').stripped_stri
ngs)
    if len(arrival_data) == 3:
        data['arr_time'] = arrival_data[0]
        data['arr_airport_long'] = arrival_data[1]
        data['arr_airport_code'] = arrival_data[2]
    else:
        data['arr_time'] = arrival_data[0]
        data['arr_airport_long'] = arrival_data[2]
        data['arr_airport_code'] = arrival_data[3]

    flight_data = list(
        segment.find(

class_='gws-flights-results__leg-flight').stripped_stri
ngs)
    data['airline'] = flight_data[0]
    data['seat_class'] = flight_data[1]
    data['airplane'] = flight_data[2]
    data['airline_code'] = flight_data[3]
    try:
        data['flight_number'] = flight_data[4] +
flight_data[5]
    except IndexError:
        data['flight_number'] = flight_data[3] +
flight_data[4]

    return data


def scrape(url):
    """
    Driver for scraping. Contains code to save the
collected information into the database. Contains the
main crawler.
    Using Selenium and BeautifulSoup4 to collect data
from website and parsing. Using the web driver
PhantomJS.
    :param url: URL for scraping
    :return:
    """
    # ['flight_date', 'full_duration', 'segments', 'stops',
'flight_number', 'arr_airport_long', 'arr_airport_code',
    # 'dep_time', 'airplane', 'arr_time', 'airline_code',
'dep_airport_code', 'dep_airport_long',
    # 'airline', 'seat_class']
    database = sqlite3.connect('Flights')
    db = database.cursor()
    query = "CREATE TABLE IF NOT EXISTS
FLIGHTS(FLIGHT_NUMBER VARCHAR
PRIMARY KEY,FLIGHT_DATE
DATE,DURATION FLOAT," \
```

```
        " STOPS INTEGER,
DEPT_AIRPORT_NAME VARCHAR,
DEPT_AIRPORT_CODE VARCHAR,
DEPART_TIME VARCHAR ,AIRLINE
VARCHAR," \
        "AIRPLANE VARCHAR,ARR_TIME
VARCHAR, ARR_AIRPORT_CODE VARCHAR,
ARR_ARIPORT_NAME VARCHAR," \
        " SEAT VARCHAR,PRICE INTEGER )"
    db.execute(query)

    # driver = webdriver.PhantomJS('phantomjs.exe')
    options=webdriver.ChromeOptions()
    options.add_argument('headless');
    driver =
webdriver.Chrome(chrome_options=options)
    driver.get(url)
    timeout = 10
    try:
        element_present =
EC.text_to_be_present_in_element(
        (By.CSS_SELECTOR,
'.gws-flights-results__unpriced-airlines'),
        'Prices are not available for: Southwest.
Flights with unavailable prices are at the end of the
list.'
        )
        WebDriverWait(driver,
timeout).until(element_present)
        time.sleep(1)
    except TimeoutException:
        print("Timed out waiting for page to load")

    soup = BeautifulSoup(driver.page_source, 'lxml')
    flights = []
    all_results = soup.find_all(

class_='gws-flights-widgets-expandablecard__body')
    for n in range(len(all_results)):
        dict = {}
        stops = list(
            driver.find_elements_by_css_selector(
                '.gws-flights-results__stops'))[n].text.strip()
        try:
            n_stops = int(stops[0])
        except ValueError:
            n_stops = 0
        dict['stops'] = n_stops

        collapsed_itinerary = list(
            soup.find_all(

class_='gws-flights-results__collapsed-itinerary'))[n]
        full_duration = collapsed_itinerary.find(
class_='gws-flights-results__duration').get_text()
        full_duration = str(full_duration).split(" ")
        hours = int(full_duration[0][:-1])
        try:
            minutes = int(full_duration[1][:-1])
        except IndexError:
            minutes = 0
            hours = hours/60
        dict['full_duration'] = round(hours + (minutes /
60), 2)

        flight_date = list(
            list(
                soup.find_all(
                    class_=

'gws-flights-results__itinerary-details-heading-text'))[
                n].stripped_strings)[1]
        dict['flight_date'] = parse(flight_date).date()

        price = list(
            collapsed_itinerary.find(

class_='gws-flights-results__itinerary-price')
                .stripped_strings)[0][1:]
        price = str(price).replace(',', '')

        dict['price'] = int(price[1:])

        dict['segments'] = []
        segments = list(all_results)[n].find_all(
            class_='gws-flights-results__leg')
        for segment in segments:
            dict['segments'] = get_segment_data(segment)

        flights.append(dict)
        database_tuple =
(dict['segments']['flight_number'], dict['flight_date'],
dict['full_duration'],
                    dict['stops'],
dict['segments']['dep_airport_long'],
                    dict['segments']['dep_airport_code'],
dict['segments']['dep_time'],
                    dict['segments']['airline'],
                    dict['segments']['airplane'],
dict['segments']['arr_time'],
                    dict['segments']['arr_airport_code'],
                    dict['segments']['arr_airport_long'],
dict['segments']['seat_class'], dict['price'])
        # Inserting into database
        db.execute("INSERT OR REPLACE INTO
FLIGHTS VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)",
database_tuple)
```

```
    database.commit()
    database.close()
    return


def makeurl(origin, destination, dep_date,
passengers):
    """
    Converts source, destination, date and passengers
into valid flights.google URL
    :param origin: Source passenger
    :param destination: Destination of passenger
    :param dep_date: Departure Date
    :param passengers: Number of passengers 0<p<10
    :return None:
    """
    url = \
        'https://www.google.com/flights#' \
        + 'flt=' \
        + origin \
        + '.' \
        + destination \
        + '.' \
        + dep_date \
        + ';c:INR' \
        + ';e:1;sd:1;t:f;tt:o'

    if passengers != 1:
        url = url + ';px:' + str(passengers)
    scrape(url)
    return
make_IATA_database()
```

GUI.py

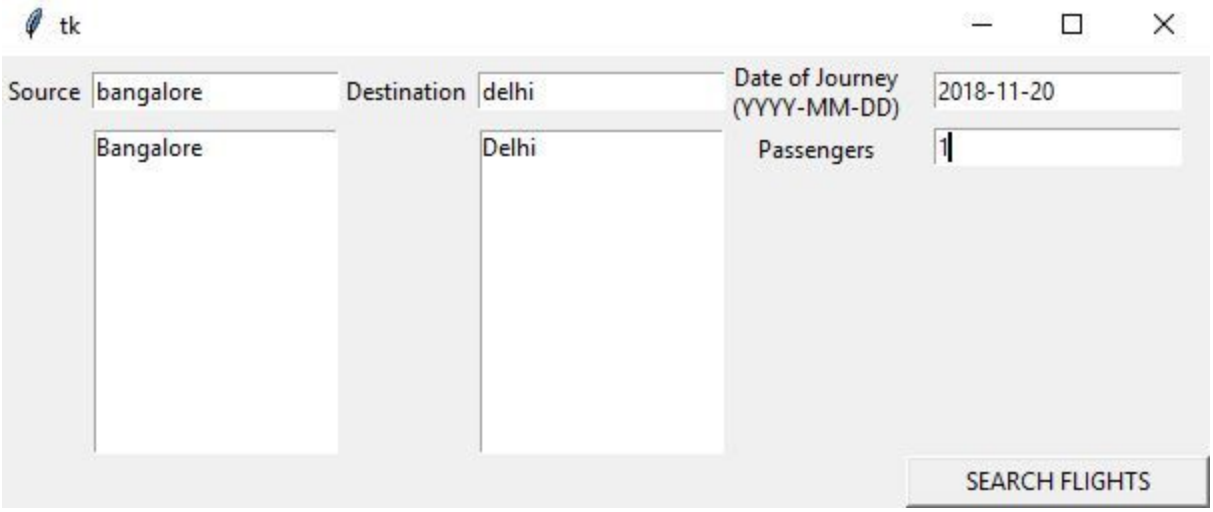
```
"""
GUI script. Created a GUI using tkinter package.

1 window contains 4 entry  feilds, 2 listboxex to help
the use correctly enter the source and destinations.
Date Entry box in the stict format (YYYY-MM-DD)
Number of passengers in the range of 1 - 9, a limit
because of Google
A button to check for available flights.

2 window contains a table with the list of all available
flights along with its number, depart time, arrival
time,
duration and Price.
It contains a button when pressed after selecting a
flight books the flight and add them to the booked
table
in the database. All transactions occur through
database.
"""

import tkinter as tk
from tkinter import messagebox
from tkinter.ttk import *
from FlightDatabase import makeurl,
make_IATA_database
import sqlite3

database = sqlite3.connect('Flights')
db = database.cursor()
# make_IATA_database()
'''
print(scrape(makeurl('BOM', 'MAA', '2018-11-11',
11)))

'''



def on_keyrelease1(event):
    """
    Feeds the first list box based on the input in the
entry box
    :param event:
    :return None:
    """
    # get text from entry
    value = event.widget.get()
    value = value.strip().lower()

    # get data from test_list
    if value == '':
        data = city_list
    else:
        data = []
        for item in city_list:
            if value in item.lower():
                data.append(item)

    # update data in listbox
    listbox1_update(data)


def listbox1_update(data):
    """
    Updates the first list box based on the input in the
entry box
    :param data:
    :return None:
    """
    # delete previous data
    listbox1.delete(0, 'end')

    # sorting data
    data = sorted(data, key=str.lower)

    # put new data
    for item in data:
        listbox1.insert('end', item)


def on_keyrelease2(event):
    """
    Feeds the second list box based on the input in the
entry box
    :param event:
    :return None:
    """

    # get text from entry
    value = event.widget.get()
    value = value.strip().lower()

    # get data from test_list
    if value == '':
        data = city_list
    else:
        data = []
        for item in city_list:
            if value in item.lower():
                data.append(item)

    # update data in listbox
    listbox2_update(data)


def listbox2_update(data):
```

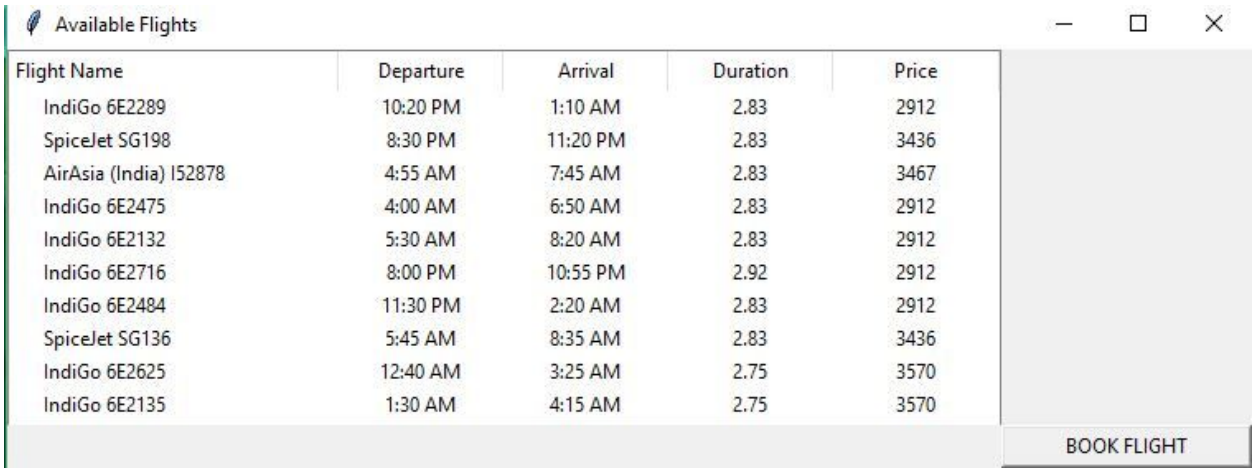
```
    """
    Updates the second list box based on the input in
the entry box
    :param event:
    :return None:
    """

    # delete previous data
    listbox2.delete(0, 'end')

    # sorting data
    data = sorted(data, key=str.lower)

    # put new data
    for item in data:
        listbox2.insert('end', item)


def searchflight():
    """
    Secomnd Window program. Collects the available
flights and feed them in the table,
    :return:
    """
    origin = (entry1.get().lower(),)
    destination = (entry2.get().lower(),)
    date = entry3.get()
    passengers = int(entry4.get())
    ocode = db.execute("SELECT CODE from IATA
WHERE CITY=? COLLATE NOCASE",
origin).fetchall()[0][0]
    dcode = db.execute("SELECT CODE from IATA
WHERE CITY=? COLLATE NOCASE",
destination).fetchall()[0][0]
    if passengers > 0 and ocode in codes and dcode in
codes:
        makeurl(ocode, dcode, date, passengers)
    elif passengers < 0:
        raise Exception("Passengers cannot be less than
1")
    elif ocode not in codes:
        raise Exception("Select valid Source")
    else:
        raise Exception("Select valid Destination")
    newWin = tk.Toplevel()
    newWin.title("Available Flights")
    table = Treeview(newWin)

    def selectItem(a):
        return

    def bookflight():
        curItem = table.focus()
        d = table.item(curItem)
        t = (str(d['text']).split(" ")[-1],)

        db.execute("CREATE TABLE if not exists
BOOKED AS select * FROM FLIGHTS WHERE
0")
        t = db.execute('select * from FLIGHTS where
FLIGHT_NUMBER=?', t).fetchall()[0]
        db.execute("INSERT OR REPLACE INTO
BOOKED VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)", t)

        database.commit()
        messagebox.showinfo(title="Success!",
message="Flight booked succesfully!")
        newWin.destroy()

    #table creation
    table['columns'] = ['Departure', 'Arrival', 'Duration',
'Price']
    table.heading("#0", text="Flight Name",
anchor='w')
    table.column("#0", anchor='w')
    table.heading("#0", text="Flight Name",
anchor='w')
    table.heading('Departure', text='Departure')
    table.column('Departure', anchor='center',
width=100)
    table.heading('Arrival', text='Arrival')
    table.column('Arrival', anchor='center', width=100)
    table.heading('Duration', text='Duration')
    table.column('Duration', anchor='center',
width=100)
    table.heading('Price', text='Price')
    table.column('Price', anchor='center', width=100)
    table.grid(sticky=('N', 'S', 'W', 'E'))
    table.bind('<ButtonRelease-1>', selectItem)
    search = tk.Button(newWin, text="BOOK
FLIGHT", justify='left', padx=2, width=20,
command=bookflight)
    search.grid(sticky='s', row=1, column=1)

    flights = db.execute(
        "SELECT
AIRLINE,FLIGHT_NUMBER,DEPART_TIME,AR
R_TIME,DURATION,PRICE from FLIGHTS where
DEPT_AIRPORT_CODE=? and
ARR_AIRPORT_CODE=?",
        (ocode, dcode,)).fetchall()
    for x in flights:
        table.insert(', 'end', text=x[0] + " " + x[1],
values=(x[2], x[3], x[4], x[5]))


# --- main ---#
#first window creation and operations
```

```
city_list = sorted([x[0] for x in db.execute('SELECT
CITY from IATA ORDER BY CITY
DESC').fetchall()])
codes = sorted([x[0] for x in db.execute('SELECT
CODE from IATA ORDER BY CITY
DESC').fetchall()])


root = tk.Tk()

Label1 = tk.Label(root, text="Source ")
Label1.grid(row=0, column=0)
entry1 = tk.Entry(root)
entry1.grid(row=0, column=1)
entry1.bind('<KeyRelease>', on_keyrelease1)

Label2 = tk.Label(root, text="Destination ")
Label2.grid(row=0, column=3)
entry2 = tk.Entry(root)
entry2.grid(row=0, column=4)
entry2.bind('<KeyRelease>', on_keyrelease2)

listbox1 = tk.Listbox(root)
listbox2 = tk.Listbox(root)
listbox1.grid(row=1, column=1)
listbox2.grid(row=1, column=4)

listbox1_update(city_list)
listbox2_update(city_list)

Label3 = tk.Label(root, text="Date of
Journey\n(YYYY-MM-DD)")
Label3.grid(row=0, column=5)
entry3 = tk.Entry(root)
entry3.grid(row=0, column=6)

Label4 = tk.Label(root, text="Passengers")
Label4.grid(row=1, column=5, sticky='N')
entry4 = tk.Entry(root)
entry4.grid(row=1, column=6, sticky='N')

search = tk.Button(root, text="SEARCH FLIGHTS",
justify='left', padx=2, width=20,
command=searchflight)
search.grid(row=3, column=6, sticky='NW')
root.mainloop()
```

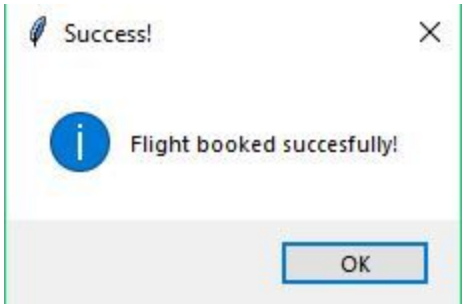
RESULTS

Screenshots:





Flight Name	Departure	Arrival	Duration	Price
IndiGo 6E2289	10:20 PM	1:10 AM	2.83	2912
SpiceJet SG198	8:30 PM	11:20 PM	2.83	3436
AirAsia (India) I52878	4:55 AM	7:45 AM	2.83	3467
IndiGo 6E2475	4:00 AM	6:50 AM	2.83	2912
IndiGo 6E2132	5:30 AM	8:20 AM	2.83	2912
IndiGo 6E2716	8:00 PM	10:55 PM	2.92	2912
IndiGo 6E2484	11:30 PM	2:20 AM	2.83	2912
SpiceJet SG136	5:45 AM	8:35 AM	2.83	3436
IndiGo 6E2625	12:40 AM	3:25 AM	2.75	3570
IndiGo 6E2135	1:30 AM	4:15 AM	2.75	3570



CONCLUSION

The project was successful in accomplishing the goals aimed for as all the flight details are successfully retrieved from the google flights website and the user can easily choose among the available options to book a website which is stored in the database. Although, the project achieves its goals, there are still some improvements which can be done, for eg., the GUI can be improved by making a selection box rather than a list box and furthermore parameters like one way trip or return trip can be taken into account to show more specific results making it easy for the user to select the type of journey they want to choose. The evaluation metrics that can be used for this project is ease of usage, time taken to show the results, GUI quality and throughput of data retrieved per time unit. The project is better in these aspects from the previous work but by implementing the suggested improvements, it will be slightly better than its current iteration. Overall the project satisfies its intended goals and is technically sound in the technologies stated except the GUI which can be improved the most out of all the modules. Although there are some improvements that can be done, the project completely satisfies the job requirement taken up and that too in an effective manner.

REFERENCES

1. Hieu Do and Michael Chen, (https://github.com/hieusydo/Airline-Reservation-System)
2. SQLite database viewer, (https://sqliteonline.com/)
3. Stack Overflow, (www.stackoverflow.com)
4. Google (www.google.co.in)