SE 226 23/24 PROJECT REPORT: Best Hotels For You

Alper Özgür Şahin 20210601055

May 18, 2024

1. Introduction

The project goal is to use Python to create a hotel listing application with a graphical user interface (GUI). Users of the project can choose between Euro and TL units, enter check-in and check-out dates, and choose a city. Using web scraping techniques, “Booking.com” data elements including hotel title, address, proximity to the city center, hotel rating, and price are extracted. After that, the application sorts the hotels it has collected according to predetermined standards and shows the user the top 5. In addition, the hotel data is kept for later use in a text or CSV or TXT file. The project's overall goal is to improve the user experience when looking for and booking hotels.

2. Libraries

The libraries and modules used in this project are used in Python programming language to perform various functions. You can find their functions and purposes below:

datetime: This module is used for date and time operations. In particular, it is used to get the date and time information of the moment the program runs.

tkinter: Tkinter is Python's standard GUI (Graphical User Interface) library. Tkinter is used to create and manage GUI components.

bs4 (BeautifulSoup): BeautifulSoup is a library used to analyze HTML and XML documents. It is often used to extract and analyze data from web pages.

requests: This library is used to make HTTP requests. It is used to retrieve data from web pages. In particular, it is often used to download web pages and retrieve their content.

csv: It is a module for working with CSV files. It is used to read and write data in CSV format.

tkcalendar: Provides a calendar widget for Tkinter. It is used to make it easier for users to select dates.

tkinter.messagebox: Tkinter is used to create dialog boxes. It is used to display error messages and informational messages.

These libraries are used for different functions in different parts of the hotel search application and are put together to ensure the functionality of the application.

3. City Destination IDs

In this section, a dictionary has been created containing target IDs to be used in hotel searches in various cities. Destination IDs collected from Booking.com are taken from the "dest_id" section in the URL. The destination ID corresponding to each city represents that city on the hotel booking site Booking.com. These target IDs are used to focus on the correct city during web scraping.

For example, a target ID is specified for the city "Rome". This ID is used to search for hotels in Rome. Similarly, unique destination IDs have been defined for other cities.

These destination IDs allow the application to find the correct hotels based on the city selected by the user. When the user selects the city he wants to search, the correct target ID is obtained using this dictionary and then the hotel search is performed.

In this way, a separate target ID is defined for each city, allowing the application to work on a wide range of cities.

```
# Different cities have different destination id's
dest_id = {
    "Roma": 126693,
    "Stockholm": 2524279,
    "Venice": 132007,
    "Warsaw": 534433,
    "Helsinki": 1364995,
    "Amsterdam": 2140479,
    "Vilnius": 2620663,
    "Paris": 1456928,
    "Fulda": 1772866,
    "Liverpool": 2601422,
    "Berlin": 1746443,
    "Seoul": 716583,
    "Frankfurt": 1771148,
    "Kyoto": 235402,
    "Rotterdam":2152403,
    "Brussels":1955538,
    "Manchester":2602512,
    "Hamburg":1785434,
    "Dortmund":1761123
}
```

4. Hotel Scraper App Class Functionalities

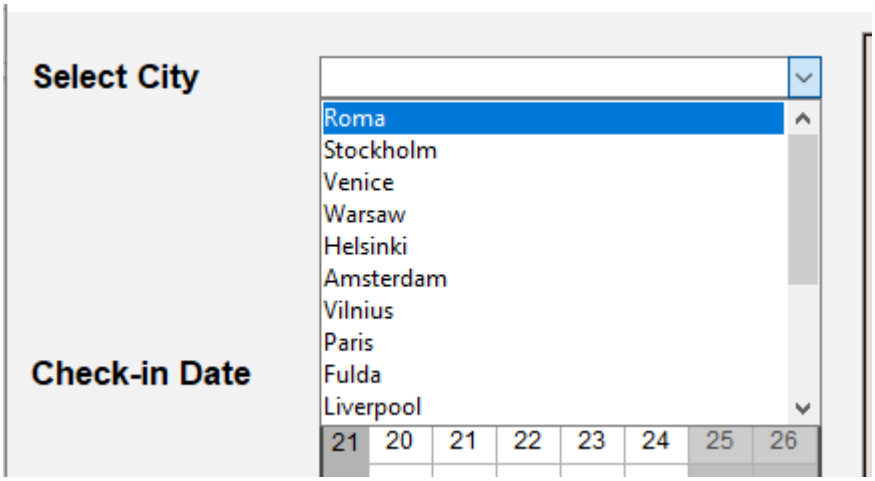
This class is a central structure that forms the basis of the hotel search application and creates a user interface using Tkinter. This class performs the following functions:

A) __init__ Method:

This initializer method is called during class initialization. It starts the Tkinter window and renders the main interface components of the application. These components include user input fields such as city selection, date selection, currency selection.

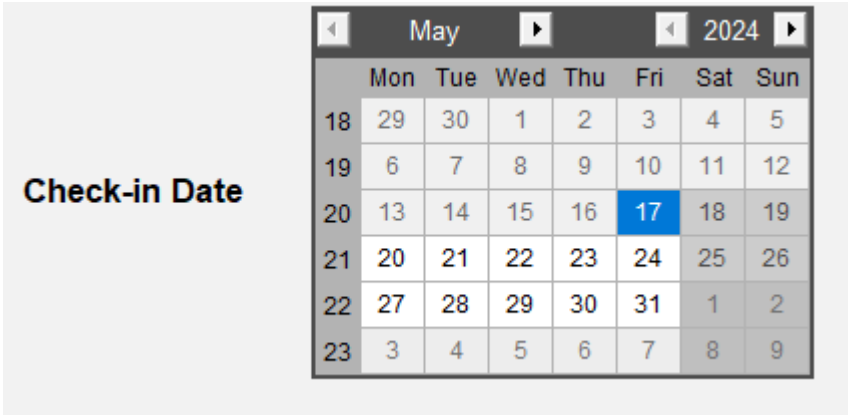
City Selection Dropdown (Combobox):

- This component allows the user to select the city they want to search for hotels.
- Created using Tkinter's **ttk.Combobox** widget.
- The city options are displayed as a list of pre-defined cities.
- The user can select a desired city from the list.



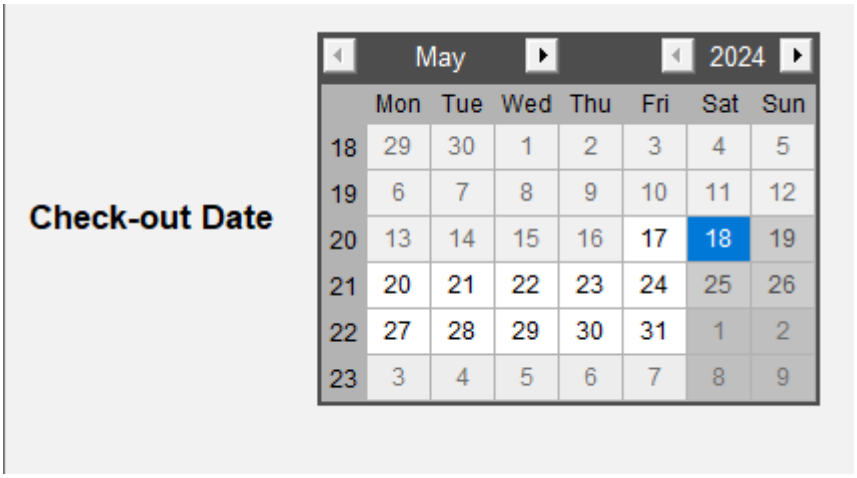
Check-in Date Entry (Calendar):

- This component enables the user to select the start date of their stay at the hotel.
- Created using the **tkcalendar.Calendar** module in Tkinter.
- The user can choose the desired date from the calendar interface.



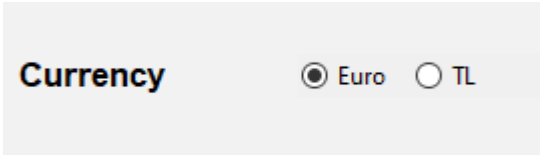
Check-out Date Entry (Calendar):

- This component allows the user to select the end date of their stay at the hotel.
- Similar to the check-in date, it's created using the **tkcalendar.Calendar** module.
- The user can choose the desired date from the calendar interface.



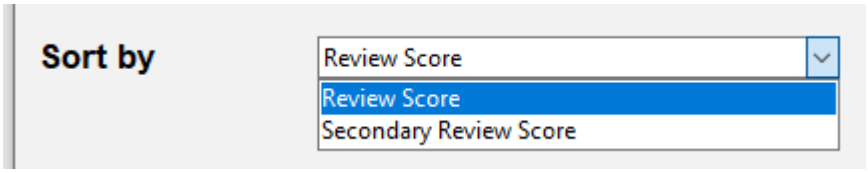
Currency Selection (Radio Buttons):

- This component allows the user to choose whether they want to see prices in Euro or TL.
- Created using Tkinter's **ttk.Radiobutton** widgets.
- The user can make a selection between the two options.



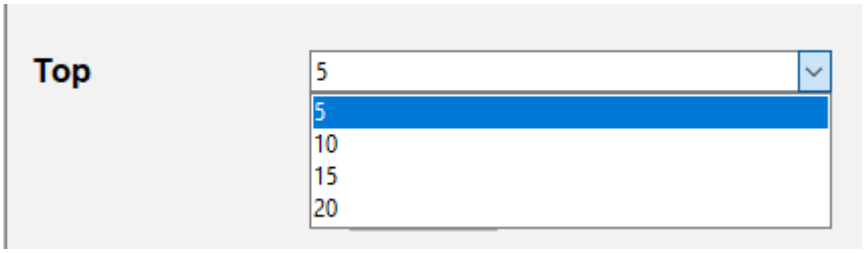
Sort by Selection Dropdown (Combobox):

- This component provides a dropdown menu for users to sort hotels based on a specific criterion.
- Created using Tkinter's **ttk.Combobox** widget.
- Sorting criteria are predefined as "Review Score" or "Secondary Review Score,(which is desired in description)" based on user preference.



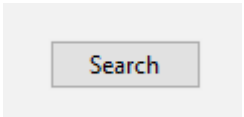
Number of Top Hotels Selection (Combobox):

- This component allows the user to select the number of top hotels to be displayed.
- Created using Tkinter's **ttk.Combobox** widget.
- The user can choose a number to specify the count of hotels to be shown.



Search Button (Button):

- This component triggers the hotel search process when clicked by the user.
- Created using Tkinter's **ttk.Button** widget.
- When clicked, the search button initiates the hotel search process, and the results are displayed in the user interface.



Text Box (Text):

- This component displays the search results to the user.
- Created using Tkinter's **tk.Text** module.
- Search results, including hotel names, addresses, prices, and other attributes, are listed in this text box

The all GUI Development Requirements is satisfied.

B) save_hotels_to_csv Method:

This method is used to save hotel data into a CSV file. Names, prices, addresses and other features of hotels are written into this CSV file. This file can be used by users to review or export hotel data later.

```python
def save_hotels_to_csv(self, hotels_list, filename):
    # Create the CSV file and write the header line
    with open(filename, mode='w', newline='', encoding='utf-8') as file:
        fieldnames = ['name', 'review_score','secondary_review_score','price', 'address', 'distance', 'city','review_number']
        writer = csv.DictWriter(file, fieldnames=fieldnames)

        writer.writeheader()

        # Write data for each hotel
        for hotel in hotels_list:
            writer.writerow(hotel)
```

Csv file:

HotelScraper.py | myhotels.csv

C:\Users\alper\Desktop\HotelScraper\HotelScraper.py myhotels.csv

```
name,review_score,secondary_review_score,price,address,distance,city,review_number
ARCAsuite Tritone Luxury,10 Exceptional,Location 10,143779,"Trevi, Rome",0.7 km from centre,Roma,7
Palazzo Ripetta,9.5 Exceptional,Location 9.8,71714,"Spagna, Rome",1.5 km from centre,Roma,324
"Palazzo Velabro, a Member of Design Hotels",9.4 Superb,Cleanliness 9.8,52901,"Colosseum, Rome",0.7 km from centre,Roma,207
Eitch Borromini Palazzo Pamphilj,8.6 Fabulous,Location 9.7,47926,"Navona, Rome",0.9 km from centre,Roma,698
Spagna Royal Suite,8.4 Very Good,Location 9.7,69570,"Spagna, Rome",1 km from centre,Roma,919
```

C) date_validation Method:

This method is used to verify the dates entered by the user. It specifically prevents past dates from being selected, and also checks whether the check-in date is before the check-out date and that the dates should not be more than 90 days apart. This validation prevents users from selecting invalid date ranges.

```
def date_validation(self, checkin_date, checkout_date):
    current_date = datetime.now() - timedelta(days=1)

    if checkin_date < current_date or checkout_date < current_date:
        messagebox.showerror("Error", "Check-in or Check-out date cannot be in the past.")
        return False

    # Checkin date should be after the checkout date
    if checkin_date >= checkout_date:
        messagebox.showerror("Error", "Check-in date must be before Check-out date.")
        return False

    # Check-in and check-out dates should not be more than 90 days apart
    if (checkout_date - checkin_date).days > 90:
        messagebox.showerror("Error", "The difference between check-in and check-out dates cannot be more than 90 days.")
        return False

    return True
```

D. search_hotels Method:

This method performs the hotel search. It pulls data from Booking.com and finds available hotels based on the city, date range, currency and other options the user selects. Using web scraping, the name, address, price, rating and other attributes of each hotel are obtained and this information is displayed in the user interface.

a. Clear Previous Text:

- This method begins by clearing any previous text displayed in the text box where the search results are shown.
- Achieved using the **`delete`** method of the **`tk.Text`** widget, specifying the range from the beginning (1.0) to the end (tk.END).

```
self.text_box.delete(1.0, tk.END)  # Clear any previous text
```

b. Input Retrieval and Validation:

- Retrieves user inputs such as selected city, check-in and check-out dates, currency preference, and sorting method.
- Validates the selected city to ensure it exists in the predefined dictionary of destination IDs (**`dest_id`**).
- Validates the check-in and check-out dates to ensure they are valid and not in the past.

```
# Date validation
checkin_date = self.checkin_calendar.get_date()
formatted_checkin_date = datetime.strptime(checkin_date, "%m/%d/%y")
checkout_date = self.checkout_calendar.get_date()
formatted_checkout_date = datetime.strptime(checkout_date, "%m/%d/%y")

if not self.date_validation(formatted_checkin_date, formatted_checkout_date):
    return
currency = self.currency_var.get()
sort_by = self.sort_by_var.get()  # Get the selected sorting method


if (city not in dest_id.keys()):
    messagebox.showerror("Error","City Not Found!")
    return
```

c. Web Scraping:

- Constructs the URL for scraping hotel data from Booking.com based on the selected city, check-in and check-out dates.
- Sends an HTTP GET request to the constructed URL using the **`requests`** library, including appropriate headers.
- Parses the HTML response using BeautifulSoup (**`BeautifulSoup`**) to extract relevant hotel data.

```
# Web scraping area
url = f"https://www.booking.com/searchresults.en-gb.html?ss=Roma&ssne=Roma&ssne_untouche

headers = {
    'User-Agent': 'Mozilla/5.0 (X11; CrOS x86_64 8172.45.0) AppleWebKit/537.36 (KHTML,
    'Accept-Language': 'en-US, en;q=0.5'
}

response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, 'html.parser')
hotels = soup.findAll('div', {'data-testid': 'property-card'})

hotels_data = []
sorted_hotels = []
```

d. Data Extraction:

- Loops through each hotel element found in the parsed HTML response.
- Extracts various attributes of each hotel, such as name, distance to center, address, review score, price, and secondary review score.

```python
# Loop over the hotel elements and extract the desired data
for hotel in hotels:
    # Extract the hotel name
    name_element = hotel.find('div', {'data-testid': 'title'})
    name = name_element.text.strip()

    # Extract the distance to center
    distance_element = hotel.find("span",{"data-testid": "distance"})
    distance_to_center = distance_element.text.strip()

    # Extract the address
    address_element = hotel.find("span", {"data-testid" : "address"})
    address = address_element.text.strip()

    # Extract the review score
    review_score_element = hotel.find("div", {"data-testid":"review-score"})
    if review_score_element:
        review_score_text = review_score_element.text.strip()
        parts = review_score_text.split("Scored ")
        review_score = parts[1].split()[0]
        review_number = parts[1].split()[-2]
        review_info = parts[1].split(maxsplit=2)[1]  # just take review info
        review_type = review_info.split()[0]  # review type
    else:
        review_score = "N/A"
        review_type = ""
    if review_type == "Very":
        review_type = "Very Good"
```

```python
# Extract the price
price_tag = hotel.find("span", {"data-testid": "price-and-discounted-price"})

if price_tag:
    price = price_tag.text
    price = price.replace("TL", "").replace(",", "").replace("\xa0", "").replace(".", "").strip()
    price = int(price)
else:
    price = "N/A"


# Extract the secondary review score(rating)
secondary_review_score = hotel.find('span', class_='a3332d346a')
if secondary_review_score:
    secondary_review_score_text = secondary_review_score.text.strip()

else:
    secondary_review_score_text = "N/A"
```

e. Data Formatting:

- Formats the extracted data into a dictionary format for each hotel, containing all relevant attributes.

f. Appending Data:

- Appends the formatted hotel data to a list (**`hotels_data`**) containing information about all the hotels scraped from the website.

```
# Append hotels_data with info about hotel
hotels_data.append({
    'name': name,
    'review_score': review_score + " " + review_type,
    'secondary_review_score': secondary_review_score_text,
    'price': price,
    'address': address,
    'distance': distance_to_center,
    'city': city,
    "review_number": review_number
})
```

g. Displaying Results:

- Displays the retrieved hotel information in the text box of the GUI, presenting details such as hotel name, address, distance to center, review score, secondary review score, and price.
- Includes error handling to display a message if the city is not found.

```
# Write the best hotels in the text box
counter = 1
for hotel in sorted_hotels:
    self.text_box.insert(tk.END, f"\n HOTEL {counter}\n", "bold")
    self.text_box.insert(tk.END, f" Name: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['name']}\n")
    self.text_box.insert(tk.END, f" Address: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['address']}\n")
    self.text_box.insert(tk.END, f" Distance: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['distance']}\n")
    self.text_box.insert(tk.END, f" Review Score: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['review_score']}\n")
    self.text_box.insert(tk.END, f" Secondary Review Score: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['secondary_review_score']}\n")
    self.text_box.insert(tk.END, f" Review Number: ", "bold")
    self.text_box.insert(tk.END, f"{hotel['review_number']}\n")
    hotel_price = hotel["price"]
    if currency == "Euro":
        converted_price = round(hotel_price / 30)
        self.text_box.insert(tk.END, f" Price: ", "bold")
        self.text_box.insert(tk.END, f"{converted_price} Euro\n")
    else:
        self.text_box.insert(tk.END, f" Price: ", "bold")
        self.text_box.insert(tk.END, f"{hotel_price} TL\n")
```

h. CSV File Saving:

- Saves the top hotels' information (based on user selection) to a CSV file named **`myhotels.csv`**.
- Utilizes the **`save_hotels_to_csv`** method within the class to accomplish this task.

The all Web Scraping Requirements is satisfied.

Selecting Top Hotels:

- The code selects the top hotels based on the user's choice, which is determined by the value of GUI dropdown selection.
- It slices the **`sorted_hotels`** list up to the number specified by the user, indicating how many top hotels to display.

```
# Sort hotels_data based on the selected sorting method
if sort_by == "Review Score":
    sorted_hotels = sorted(hotels_data, key=lambda x: x['review_score'], reverse=True)
elif sort_by == "Secondary Review Score":
    sorted_hotels = sorted([hotel for hotel in hotels_data if hotel['secondary_review_score'] != 'N/A'],
                           key=lambda x: float(x['secondary_review_score'].split(' ')[-1]), reverse=True)

# Select Top User Choice Hotels
sorted_hotels = sorted_hotels[:self.top_var.get()]
```

Displaying Hotel Information in Text Box:

- After selecting the top hotels, the code iterates over each hotel in the **`sorted_hotels`** list.
- It inserts the hotel information into the text box of the GUI, including the hotel name, address, distance, review score, secondary review score(rating), and price.
- If the user selected Euro as the currency, the price is converted from TL to Euro using a conversion rate of 1 Euro = 30 TL and displayed accordingly.

Saving Top Hotels to CSV File:

- Once the hotel information is displayed in the GUI, the code saves the top hotels to a CSV file named **`'myhotels.csv'`**.
- It calls the **`save_hotels_to_csv`** method within the class, passing the sorted hotels list and the file name as arguments.

The all Hotel Information Storage/Display Requirements is satisfied.

Main Function and GUI Initialization:

- The code checks if the script is being run directly (**`__name__ == "__main__"`**).
- If so, it initializes the Tkinter root window (**`tk.Tk()`**), creates an instance of the **`HotelScraperApp`** class, and starts the GUI event loop with **`root.mainloop()`**.

```python
if __name__ == "__main__":
    root = tk.Tk()
    app = HotelScraperApp(root)
    root.mainloop()
```

GUI View