

CSE 4074 Project Report – Socket Programming – Travel Agency

150116825 – Onur Can Yücedağ

150115501 – Emre Erdem

Source Files

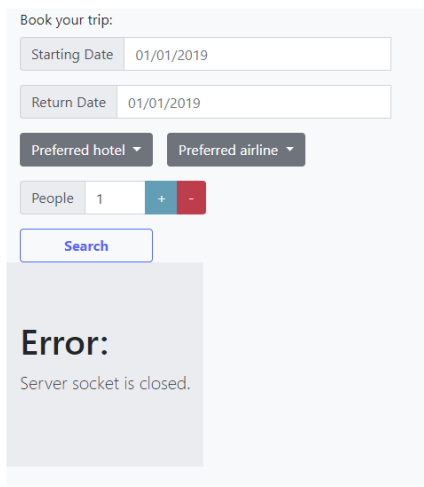
We have 4 different projects in total. They are for Client Web App, Trip Advisor Server, Airline Request Handler and Hotels Request Handler.

Client Web App

It is a Flask project that is used for creating a web app as a GUI to the users. It relatively simple structure consists of single HTML file(index.html), an entry point (app.py), front-end Javascript file (index.js) for handling interactions and lastly a CSS file(style.css) for style. It also has a text file for getting the table names in a dynamic manner, we will talk about this later.

Before we dive in how to use this app, we need to talk about what it does exactly. First, it is a web app as we mentioned above. So, before the actual request handling starts, it will check if the Trip Advisor Server's socket is open for connection. Because it will forward the client's requests to the Trip Advisor Server, we need to establish this connection before any HTML pop up to our screens.

By convention, we decided to put the Trip Advisor Server in localhost:54000 address so Client Web App will try to connect to the that socket. If any reason it can't establish a connection, it will simply output an error to the screen of the user as you can see in Figure 1.



The screenshot shows a web application interface for booking a trip. At the top, it says "Book your trip:". Below this are two input fields for "Starting Date" and "Return Date", both set to "01/01/2019". There are two dropdown menus for "Preferred hotel" and "Preferred airline". Below these is a "People" section with a value of "1" and plus/minus buttons. A "Search" button is located below the "People" section. At the bottom of the form, there is a large grey box with the text "Error: Server socket is closed.".

Figure 1: Server socket error alert.

This error's main reason is that connection refused by Trip Advisor Server. So, we need to execute and initialize listening socket in Trip Advisor Server to resolve the error.

After establishing connection is successful with the Trip Advisor Server, it is good to serve to the clients. In the main route ("/") it will try to read a table_names.txt file for get the airline and hotel names; if it can't find the text file or encounters any errors while trying to read the file, it will simply put static elements in the both lists.

When it is up and running without any errors you will see a screen like in Figure 2.

Book your trip:

Starting Date

Return Date

Preferred hotel

Preferred airline

People

+

-

Search

Figure 2: Opening page without error.

Book your trip:

Starting Date

×

Please provide a valid starting date.

Return Date

MarmaraHotel

TurkishAirlines

People

+

-

Search

Figure 3: Validation from the front-end.

When a client enters Starting Date, Return Date, Preferred hotel, Preferred airline and People inputs in the page; Search button will be active for sending the request to the Trip Advisor Server. If a client inputs wrong format of date or below input the 1 in the People input region, front-end Javascript will warn the user as Figure 3.

It will block all days before today's date for making sense. After a client clicks the Search button; front-end will check if you entered all the inputs in the correct format and change the style of the dropdown buttons for airline and hotels, that is because these dropdowns are selected now and they won't be unselected again. They will turn green for displaying this.

Then, it will post this form data in a request to a backend method, CheckDates. This method simply parses form data and send to the Trip Advisor Server with the method "check_dates". After sending the data it will wait for the response.

Data that Client Web App sends to Trip Advisor Server is fairly simple. It consists of the request components separated by semicolon like in Figure 4.

```
{start_date};{return_date};{preferred_hotel};{preferred_airline};{people_count};{req_method}"
```

Figure 4: Data from Client Web App to Trip Advisor Server.

The return message from Trip Advisor Server consists of two components separated by "|". First one consists of status of the airplane results and second one is for the hotel responses. Each of the message can be either "Success", "Failure" or "Alternative1;Alternative2...". Client Web App will parse the alternatives if they exists.

Trip Advisor Server will most likely send “Failure|Failure”, “Alternatives|Success” or “Success|Success”. These messages represent the airline and hotel responses to the dates and number of people selections. Basically, they represent if they can book it or not. If any of these messages includes “Failure”, it means we can’t book simple as that. Because we can’t sleep on the streets while we are using airplanes to go there.

If the response is “Success|Success” it means that preferred dates and airline, hotels are free to book. In that case user will be prompted to accept or reject the offer. In case of accepting the offer, Client Web App will send the same booking information with the method as “accept_dates” to the Trip Advisor Server.

If there are any alternatives for either airplanes or hotels (and if the other response is not a “Failure”), user will prompt to select one of the alternatives if he/she wants. In case of accepting an alternative offer, Client Web App will send the same booking information with the method as “accept_dates” to the Trip Advisor Server.

Trip Advisor Server

It is a simple python file that has couple of responsibilities: getting the clients’ requests about trips they want to make, contacting with the hotel and airplane handlers to check if it is okay to book, giving the clients response that they need and wait if they accept the offer, send back to airplane and hotel handlers the acceptance request.

First of all, server will define a selector that will act like a mutex. This mutex is essential for concurrency of this server. Basically it will register all the sockets to this selector, including listening and every connection socket. For any socket event it will read the socket identifier so it will accept the requests that is coming from listening socket and register the new connection socket and serve the connection sockets that have been already accepted.

It will connect with the airplane and hotel request handlers through sockets, details of these sockets will explained later.

Sockets that are accepted and registered to selector object will be served. First, incoming message will be parsed to create a User object like in Figure 5.

```
class User():
    def __init__(self, user_no, start_date, return_date, preferred_hotel, preferred_airline, people_count, method_requested):
        self.user_no = user_no
        self.preferred_airline = preferred_airline
        self.preferred_hotel = preferred_hotel
        self.start_date = start_date
        self.return_date = return_date
        self.people_count = people_count
        self.method_requested = method_requested
```

Figure 5: User object for compact data structure for requests.

This object will be a compact representation of the requests. Also, we have to mention that `use_no` variable is connected socket's incoming port number (for proof of concept). After creating of the User object, this object and parsed `method_requested` will be sent to the `CheckPreferredOrOffer` method. This method is the core of all the project. It will generate GET requests by `GenerateGetRequest` method to the airlines and hotels and send them. This GET requests are fairly simple, you can see headers in Figure 6 and body in Figure

```
host = HOST
port = 50000
headers = ""\
GET /{method_requested} HTTP/1.1\r
Content-Type: {content_type}\r
Content-Length: {content_length}\r
Host: {host}\r
Connection: close\r
\r\n""
```

Figure 6: Header field in the GET request from Trip Advisor Server

```
body = 'user_no={user_no}&start_date={start_date}&return_date={return_date}&preferred_airline={preferred_airline}&preferred_hotel={preferred_hotel}&people_count={people_count}'
body_bytes = body.format(
    user_no=user.user_no,
    start_date=user.start_date,
    return_date=user.return_date,
    preferred_airline=user.preferred_airline,
    preferred_hotel=user.preferred_hotel,
    people_count=user.people_count
).encode('ascii')
```

Figure 7: Body field in the GET request from Trip Advisor Server

As you can see in the body, we simply send all the information about the trip preferences. Also we need to mention, we don't have to send `preferred_hotel` to the airline handler but it is airline handler's job to ignore this information in the body. `GenerateGetRequest` method avoids checks like this because of clean code experiments.

There is two `method_requested` variable in debates in here. First one is coming from the Client Web App and second one is going to the airline or hotel handlers. By checking the `method_requested` variable that is coming from the Client Web App we can decide if user only wants to check dates or accept the offered dates.

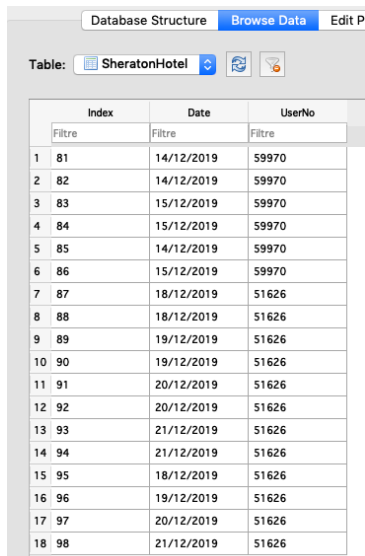
If user only wants to check availabilities, we let the airline and hotel handlers know that by sending them fairly similar parameter, `method_requested` because airline and hotel handlers will use this variable to check if they need to insert a new booking or just query bookings. For airlines we set this parameter as "`check_airline_dates`" and for hotels we set as "`check_hotel_dates`".

After sending these requests to the airline and hotel handlers, we expect a status code as 200 OK or 404 Not Found. After parsing the results we can join these two responses coming from airline and hotel handlers and combine them with "`|`" to send back to the Client Web App.

If user wants to accept the dates offered before, we need to set the `method_requested` parameter again. For airlines we set this parameter as "`accept_airline_dates`" and for hotels we set as "`accept_hotel_dates`". If the insertion is a success for both of them, we send back the responses as "`Success|Success`". In the contrary we send to Client Web App a message that contains a "`Failure`" which stands for insertion is failed abnormally.

Hotel Request Handler and Airlines Request Handler

Since they are fairly similar scripts, we decided to put them in the same heading. Below explanations are for both of them but every one of them runs separately, so airline doesn't have access to hotel database and vice versa. These scripts' main objective is creating a socket where Trip Advisor Server can connect and listen the requests that is coming from it. By the requests, these handlers will decide the response and execute the queries in the SQLite databases. There are two different databases for each of them to execute their queries on them.



The screenshot shows a database browser interface with tabs for 'Database Structure', 'Browse Data', and 'Edit Pr'. The 'Table:' dropdown is set to 'SheratonHotel'. Below it, a table preview is shown with columns 'Index', 'Date', and 'UserNo'. The table contains 18 rows of data, with dates ranging from 14/12/2019 to 21/12/2019 and UserNo values of 59970 and 51626.

	Index	Date	UserNo
1	81	14/12/2019	59970
2	82	14/12/2019	59970
3	83	15/12/2019	59970
4	84	15/12/2019	59970
5	85	14/12/2019	59970
6	86	15/12/2019	59970
7	87	18/12/2019	51626
8	88	18/12/2019	51626
9	89	19/12/2019	51626
10	90	19/12/2019	51626
11	91	20/12/2019	51626
12	92	20/12/2019	51626
13	93	21/12/2019	51626
14	94	21/12/2019	51626
15	95	18/12/2019	51626
16	96	19/12/2019	51626
17	97	20/12/2019	51626
18	98	21/12/2019	51626

Each database consists of tables which are airline or hotel companies in them. For a snapshot a hotel table in Hotels database see Figure 8.

Each table whether in Hotels or Airplane database consists of Index, Date and UserNo columns. Index column is primary integer key with auto-increment, Date column is not null text and UserNo column is not null integer.

First, they will establish a connection with the database, if they can't they will print the error to the console and execute freely because we didn't wanted to crash all socket connections all once. Later, they will create a listening socket with different ports in the localhost to listen Trip Advisor Server. Also this socket will be non-blocking and use the same selector object as Trip Advisor Server.

Figure 8: Sheraton Hotel table preview.

Trip Advisor Server will connect with the both of them and send GET requests as explained before. When these handlers receives a GET request, they will parse the body and the method requested and send to a method called RequestHandler.

RequestHandler method is the core method of these scripts because it will check the method and generate queries for satisfying them, finally it will return the results.

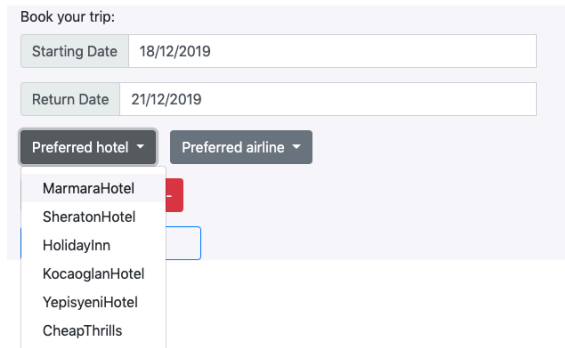
Each hotel and each airline has a 3-person-max limit in their bookings for a specified day. RequestHandler method firstly gets the table names from the its own database. If a user specified an airline or hotel that is not exists in databases, they will try to create the table for that preference. Later, they will check the number of bookings in preferred hotel or airline in a specified dates. We need to mention that between start date and end date of the trip, these scripts will generate a date sequence one day away from each other and use this date sequence to query number of bookings. If any particular day exceeds the maximum number of people with adding the requested people count, we have to check for alternatives.

Alternative checking is fairly simple; for the ever hotel or airline that we can't allow booking we need to query the other airline or hotels with the same date sequence and check again for number of bookings in them. If they can be booked in a specified date sequence, we can add the airline or hotel to alternatives.

If these handlers couldn't find any alternatives they will simply return "Failure" with 404 Not Found as response to the GET request. If they find any alternatives they will return "Success" with 200 Ok status and add the alternative airline or hotel names in the body section of the response with semicolon separated.

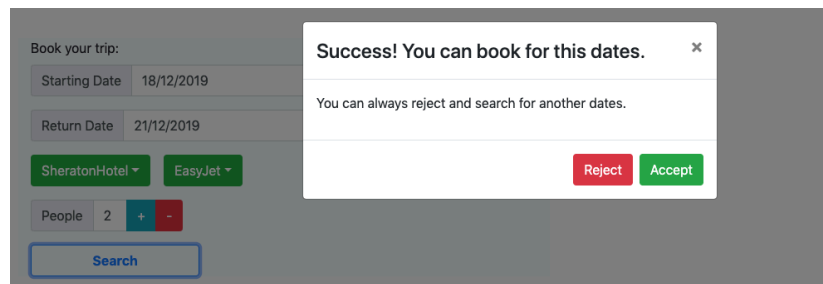
Running the Project

1. Go to localhost:5000 in your browser and fill the Starting Date, Return Date, Preferred hotel, Preferred airline and People inputs:



The screenshot shows a web form titled "Book your trip:". It contains two date input fields: "Starting Date" with the value "18/12/2019" and "Return Date" with the value "21/12/2019". Below these are two dropdown menus: "Preferred hotel" and "Preferred airline". The "Preferred hotel" dropdown is open, showing a list of options: MarmaraHotel, SheratonHotel, HolidayInn, KocaoglanHotel, YepisyeniHotel, and CheapThrills. The "Preferred airline" dropdown is closed.

2. If there are available seats and rooms for your preferred hotel and airline, you will be prompt to accept or reject the booking, if you accept it databases of airline and hotels will be updated by your trip informations:



The screenshot shows the same "Book your trip:" form, but now with "SheratonHotel" selected for the preferred hotel and "EasyJet" for the preferred airline. The "People" input is set to "2" with plus and minus buttons. A "Search" button is at the bottom. A modal dialog box is open on top of the form, displaying the message "Success! You can book for this dates." with a close button (X). Below the message, it says "You can always reject and search for another dates." and there are two buttons: "Reject" (red) and "Accept" (green).

3. If you exceed the maximum people limit either for preferred airline or hotel, you will be prompt to select the alternatives (if exist). If you select alternative hotel and airline and accept the offer, databases will be updated by your booking information:

The screenshot shows a booking form titled "Book your trip:" with fields for "Starting Date" (18/12/2019) and "Return Date" (21/12/2019). Below these are dropdowns for "HolidayInn" and "DutchAirlines", and a "People" counter set to 1. A "Search" button is at the bottom. A modal window titled "Not so lucky! You can try this alternative bookings." is open, containing the text "You can always reject and search for another dates." and two dropdowns: "Select airline" (showing a list with RyanAir, TurkishAirlines, EmiratesAirlines, and Pegasus) and "Select hotel". At the bottom of the modal are "Reject" and "Accept" buttons.

4. If the insertion is successful to the databases you will be alerted like this:

The screenshot shows the same booking form as before, but now with "SheratonHotel" and "TurkishAirlines" selected in the dropdowns. The "People" counter remains at 1, and the "Search" button is still present.

Great Choice Successfully booked your alternative place.

5. If anything goes wrong whether in the insertion or you can't book any hotel or airline because there is no room or seat for you, you will be alerted like this:

The screenshot shows the same booking form, but the "People" counter has been increased to 4. The "Search" button is still present.

No Luck! There is no place for you.

