•

1 Exercise 3: Wedding Planner

1.1 General Instructions

IMPORTANT NOTE: The submission deadline for this exercise is a THURSDAY (not a Friday!) on 02/MAY/2024 at 18:00, due to the holiday of the first of May.

This exercise is divided into two main parts. You may choose to develop the first part, referred to as **Session 1**, during the initial laboratory session of the exercise. The second part, known as **Session 2**, can be developed during the final lab session.

Before submitting your work, ensure that you test it on one of the lab machines. It's important to note that your code will be evaluated in the Kilburn's Linux environment, so make sure it runs successfully in this setting.

1.2 Introduction

In this section, we describe the problem you are supposed to solve, which represents the motivation for the development of your SOA-based client-server application. Note that you are being asked to use Web programming technologies (i.e., Web Services, the HTTP protocol and the JSON markup language) in the development of your solutions to this exercise.

1.2.1 Problem Description

You need to arrange, for the earliest time possible, a wedding. You are required to make two reservations — a hotel to host and a band to play at the reception. The aim of this exercise is for you to experience the inherent difficulties in building distributed applications that need to cope with concurrency, message delays and failures. Enjoy!

The hotel and the band both advertise slots 1 to 550 where slot 1 represents the earliest slot available while slot 550 represents the latest one.

As you need the band to play at the reception, the reservation for the hotel and the band must match — that is, if you reserve slot i for the hotel then you must also reserve slot i for the band. Note that the hotel and band restrict a user to hold at most two reservations at any one time.

Unlike the two previous exercises, for this exercise, you will *not* need to devise and implement a protocol to allow your client application to communicate with the server being provided to you. Instead, you will write your client using the HTTP protocol (for content exchange) and the JSON language (for content description); your client will compete with other clients (your fellow classmates) to reserve the hotel and the band.

The hotel and the band both provide a reservation service on the Web. They receive JSON messages over HTTP and allow you to send in requests to reserve a slot, cancel a reservation, query to find available slots and query to find slots reserved by you.

You will need to contend with the service being unavailable to receive your messages, with the fact that both the hotel and the band may not process your messages in the order in which they are received and delays before messages are processed. When your client tries to retrieve the results from a request, the message may not be available so your client will need to try again later. The services have been configured to randomly make themselves unavailable to incoming requests and to introduce arbitrary delays in processing requests. However, once the server acknowledges the

receipt of your message you may assume your message will not be lost.

Finally, your client should not overwhelm the server with requests and should never send more than one request per second.

1.2.2 Getting Started

Get started by downloading the skeleton code from the Blackboard page of this course unit. And putting them in a suitably named subdirectory of your `~/COMP28112` directory.

You should have the following files:

- `api.ini`: a basic configuration file
- `exceptions.py`: a Python module containing exceptions that are thrown by the `ReservationApi`.
- `reservationapi.py` : a simple reservation API wrapper.
- `session1.py` : an empty skeleton for your client code.

1.3 Session 1 Tasks

1.3.1 Task 3.1: Checking and reserving a slot

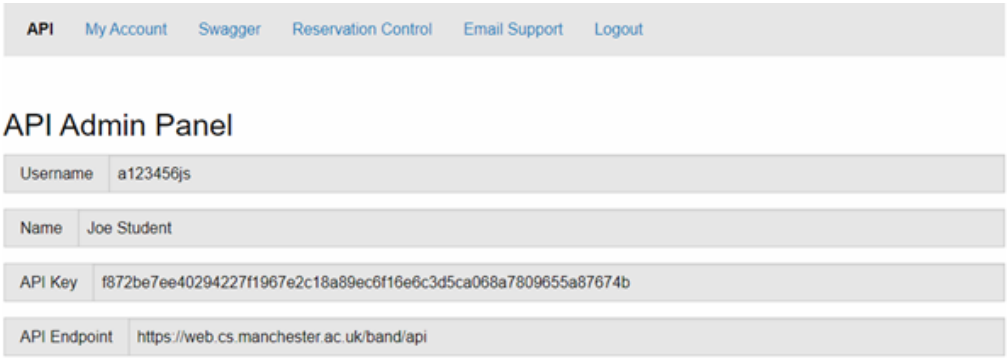
Write a simple client using Python3 to send a reservation request to the band (or the hotel).

You must understand `ReservationApi` (see `reservationapi.py`) before you start working on your own code.

Before you can communicate with either the band or hotel APIs, you must obtain an authentication token for each system. These tokens are unique to you, and they are used to identify you in API requests without you having to store or enter a username or password in your client code. To obtain a token for the band API, you should open

`https://web.cs.manchester.ac.uk/band/`

in your web browser. When prompted to log in enter your University of Manchester username and password. You will then be taken to a page that looks like the following:



Your API Key is shown on that page. All requests to the band API must include your API Key in the request header field "Authorization" in the following format:

`Authorization: Bearer <API Key>`

For example:

```
Authorization:  Bearer f872be7ee40294....87674b
```

This header must be created and returned by the `'headers'` method defined within the `reservationapi.py` file.

Note: You **can not** use the same API Key for the hotel API. You must obtain another API Key for the hotel API by going to

`https://web.cs.manchester.ac.uk/hotel/`

in your web browser, logging in as before, and making a note of the API Key you are given by that system.

Both APIs provide a Swagger interface through which you can try out API calls "by hand" and this is accessible via the "Swagger" link in the menu bar at the top of both the band and hotel API Admin Panels. The Swagger page also includes information about the expected arguments for each API endpoint, and the possible response values you can receive from each endpoint.

To send a reservation request to the band you need to perform a `HTTP POST`[1] operation message to the URL

`https://web.cs.manchester.ac.uk/band/api/reservation/{slotid}`

Where `{slotid}` is the numeric ID (between 1 and 550) of the slot you wish to book. Similarly, if you wish to send your requests to the hotel then send your POST request to

`https://web.cs.manchester.ac.uk/hotel/api/reservation/{slotid}`

As noted above, the message must include the "Authorization" header, otherwise you will receive a permission error when making the request.
There are several possible responses from the server. Each response will contain a HTTP response code, and a JSON response body containing information about the server's response to your request. For a list of defined http response codes see `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html` - only a subset of possible response codes will be generated by the service:

- If your request has been processed successfully, you will receive a 200 response code, and the response body will contain JSON — see Listing 1.
- If the service is currently unavailable, you will receive a 503 response code and the response body will contain JSON — see Listing 2.
- If your request is malformed, you will receive a 400 response code and the response body will contain JSON — see Listing 3.
- If you have attempted to perform an API request with a missing or invalid token, you will receive a 401 response code and the response body will contain JSON — see Listing 4.
- If you have requested a slot that does not exist, you will receive a 403 response code and the response body will contain JSON— see Listing 5.
- If your request has not been processed for some reason, you will receive a 404 response code and the response body will contain JSON— see Listing 6.
- If the slot you have requested is already held by another user, or is otherwise unavailable, you will receive a 409 response code and the response body will contain JSON — see Listing 7.

[1]For more information on how to perform such operations in Python, check this link:
`https://requests.readthedocs.io/en/master/`

- If you already hold two reservations on the system, you will receive a 451 response code and the response body will contain JSON — see Listing 8.
- You may also receive a 500 response code if there has been a server error while processing your request.

Listing 1: Successful Reservation

```
{ "id":"56" }
```

Listing 2: Service not available

```
{"code": 503, "message": "Service unavailable"}
```

Listing 3: Bad request

```
{"code": 400, "message": "Bad request."}
```

Listing 4: Invalid or missing token

```
{"code": 401, "message": "The API token was invalid or missing."}
```

Listing 5: Slot does not exist

```
{"code": 403, "message": "SlotId does not exist."}
```

Listing 6: Request not processed

```
{"code": 404, "message": `The request has not been processed'}
```

Listing 7: Slot is not free

```
{"code": 409, "message": "Slot is not available."}
```

Listing 8: Maximum permitted number of reservations

```
{"code": 451, "message": "The client already holds the maximum
    number of reservations."}
```

1.3.2 Task 3.2: Cancelling a reservation

Extend your client to request the cancellation of a reservation. This is done by sending a DELETE request to the URL
`https://web.cs.manchester.ac.uk/band/api/reservation/{slotid}`

Where `{slotid}` is the numeric ID (between 1 and 550) of the slot you wish to cancel your reservation for. Similarly, if you wish to send your cancellation requests to the hotel then send your POST request to

`https://web.cs.manchester.ac.uk/hotel/api/reservation/{slotid}`

You will receive the same set of responses as before. However, the result of processing your request is a little different to before. The possible responses are:

- The reservation has been cancelled — code 200;
- The cancellation failed as the slot was not reserved by you — code: 409;
- The cancellation failed as the username password was invalid — code: 401.
- The cancellation failed because the slot does not exist, code 403;
- You may also receive 500, 503, or 400 errors.

1.3.3 Task 3.3: Finding free slots

Extend your client to find free slots. This time, send a `GET` request to the URL

`https://web.cs.manchester.ac.uk/band/api/reservation/available`

As before, if you want to find the free slots available for the hotel, you should use the same endpoint for the hotel API by sending a `GET` request to

`https://web.cs.manchester.ac.uk/hotel/api/reservation/available`

A successful response from this endpoint will contain a JSON array representing the available slots in the system, an example of which is shown in Listing 9.

Listing 9: Body of a 200 response

```
[{"id":"10"},
 {"id":"15"},
 {"id":"17"},
 {"id":"22"}]
```

As with the previously discussed endpoints, you may receive several response codes in response to your request, and you should check whether the response you get is a 200 code or not, and take appropriate action if not (which may be retrying or exiting with an error message).

1.3.4 Task 3.4: Checking slots reserved by you

You can also send a request to find all the slots that are reserved by you by sending a `GET` request to the URL

`https://web.cs.manchester.ac.uk/band/api/reservation`

As before, if you want to get the list of slots you have reserved for the hotel, you should use the same endpoint for the hotel API by sending a `GET` request to

`https://web.cs.manchester.ac.uk/hotel/api/reservation`

A successful response from this endpoint will contain a JSON array representing the list of slots you have in the system, an example of which is shown in Listing 10.

Listing 10: Body of a 200 response

```
[{"id":"56"},
 {"id":"22"}]
```

1.4 Session 2 Tasks—Reserving identical slots (as early as possible) for the hotel and the band

Now the fun and games start. You now need to design and implement a strategy to reserve the same numbered slot for both the hotel and the band — that is, if you reserve slot 10 for the band then you also need to reserve slot 10 for the hotel. Remember, you are permitted to have at most two reservations for the hotel and two reservations for the band.

This strategy should try to reserve the earliest available matching slots. Note, we will sometimes block slots from reservations and then release them during the labs.

Important:. It is quite easy, by having a number of clients fire requests at the server as quickly as they can, for this to turn into a denial of service attack! It is strongly recommended that client

code which loops around interrogating the server should include a deliberate delay (of say one second) to reduce the danger of this. A code which ignores this advice will be penalised in the assessment.

To get full marks for Session 2 you need to have a fully operational program (client) that implements a strategy for booking matching slots that relies on availability, does not give up when a booking request fails, and will cancel unmatched bookings. Furthermore, your strategy should find the earliest matching slot, not any matching slot; and, once a matching slot is found, it should also check for a better booking (remember, the availability data for the band and the hotel is out-of-date as soon as it is received!); you need to make sure that you cancel bookings after obtaining better ones, but you need to maintain always at least one booking; also, your code should not give up if no matching slots are available. Finally, there must be a delay of at least 1 second between successive requests to the server (even if it is a retry).

To get full marks you should always assume that slot availability changes frequently and messages may be delayed or lost!

Remember that you are not allowed to give anybody your username and password (even more you should not put a username/password or any code online). If you ignore this or the 1-second-delay rule you may be penalised.

1.5 Deliverables (Total marks: 15)

The deliverables for this exercise are:

- **Session 1 tasks [5 marks]:**
  - Complete the code for the following relevant methods that are declared in the file `reservationapi.py`:
    - `get_slots_available` method: this method should be used to check for free slots
    - `reserve_slot` method: this method should be used to reserve a slot
    - `release_slot` method: this method should be used to cancel a reserved slot
    - `get_slots_held` method : this method should be used to retrieve booked slot(s)
    - `_send_request` method: this method should be used by the above-mentioned methods to send requests and also to handle errors and failures from the API sensibly
  - Use the code provided in `session1.py` and create a Python client `mysession1.py` that should perform operations as explained in Tasks 3.1, 3.2, 3.3 and 3.4 in an appropriate sequence by calling the newly defined methods. It should also display messages on the screen for each of the operations.
- **Session 2 tasks [10 marks]:**
  Create a Python client `mysession2.py` that should:
  - check the availability of common slots and display the first 20 common slots on the screen
  - book earliest common slot, release other booked slots if any and display booked slots on the screen,
  - recheck at least once for better bookings
  - displays appropriate messages on the screen
  - always adds a one-second delay
  - fully working i.e. free of errors and bugs

1.6 Submission instructions

You should submit your work via Blackboard. You are expected to submit a **zip folder** with name `comp28112_ex3_username.zip` containing :

- `api.ini`
- `exceptions.py`
- `reservationapi.py`
- `mysession1.py`
- `mysession2.py`

Don't forget to replace `username` with your own username consisting of 8 alphanumeric characters (e.g. a12345zz) before submitting your work.