REST API development with Flask using Python

Sandali Wasana, K A D - 191AEM006

January 2020

1 Objective

The purpose of this micro-project is to develop an REST API application with Flask in Python. Before going through the details of the process, we need to understand the purpose of using a some tools within this project.

2 Introduction

*** What is REST API ?**

REST API - Representational state transfer Application programme Interface, also refers to as RESTful API is a web application which can be used in any protocol. A RESTful API explicitly takes advantage of HTTP to GET to retrieve a resource; PUT to change the state of or update a resource, which can be an object, file or block; POST to create that resource ; and DELETE to remove it. We can see the results of our project APIs in Postman which is a web application created for interacting with HTTP APIs. APIs can also be used for storage and and processing.

REST architecture is made of 6 main characteristics. They are;

i) Client-server - it is a must to clearly differentiate the client and the server.

ii) Stateless - Client request must contain all the necessary information it need to carry out. And the server mustn't store any state about the client which persists from one request to another.

iii) Cache - server responses can be categories as cacheable or non-cacheable. It will help the clients to use cache for optimization purposes.

iv) Uniform interface - Most complex part of REST. It covers the use of unique resource identifiers, resource representations, self-destructive massage between client and server and hypermedia.

v) Layered System

vi) Code on demand - client can download the code from server and then use it to execute in their context.

*** Why use API first ?**

By using API we can design, execute and check our API first before building the rest of the app. When we buils the API fist instead of writing the code we can start with design, planning, mocks, and tests.It helps us to reduce the complexity of working, specially in the cloud.

*** What is Flask ?**

Flask is a web framework based on python language which provides us with tools, libraries and technologies that allow us to build web applications. Those web applications can be anything like a web page, a blog, a wiki or as big as a commercial website. Flask does not need any native support for accessing databases, validating web forms, authenticating users, or other high-level tasks. These and many other key services most web applications need are available through extensions that integrate with the core pack- ages. Examples of the usage of Flask include Pinterest, LinkedIn, and the community web page for Flask itself.

3 Process of the project

The project basically consists of following stages;

i) Creating basic flask-RESTful application;

ii) Training with API design;

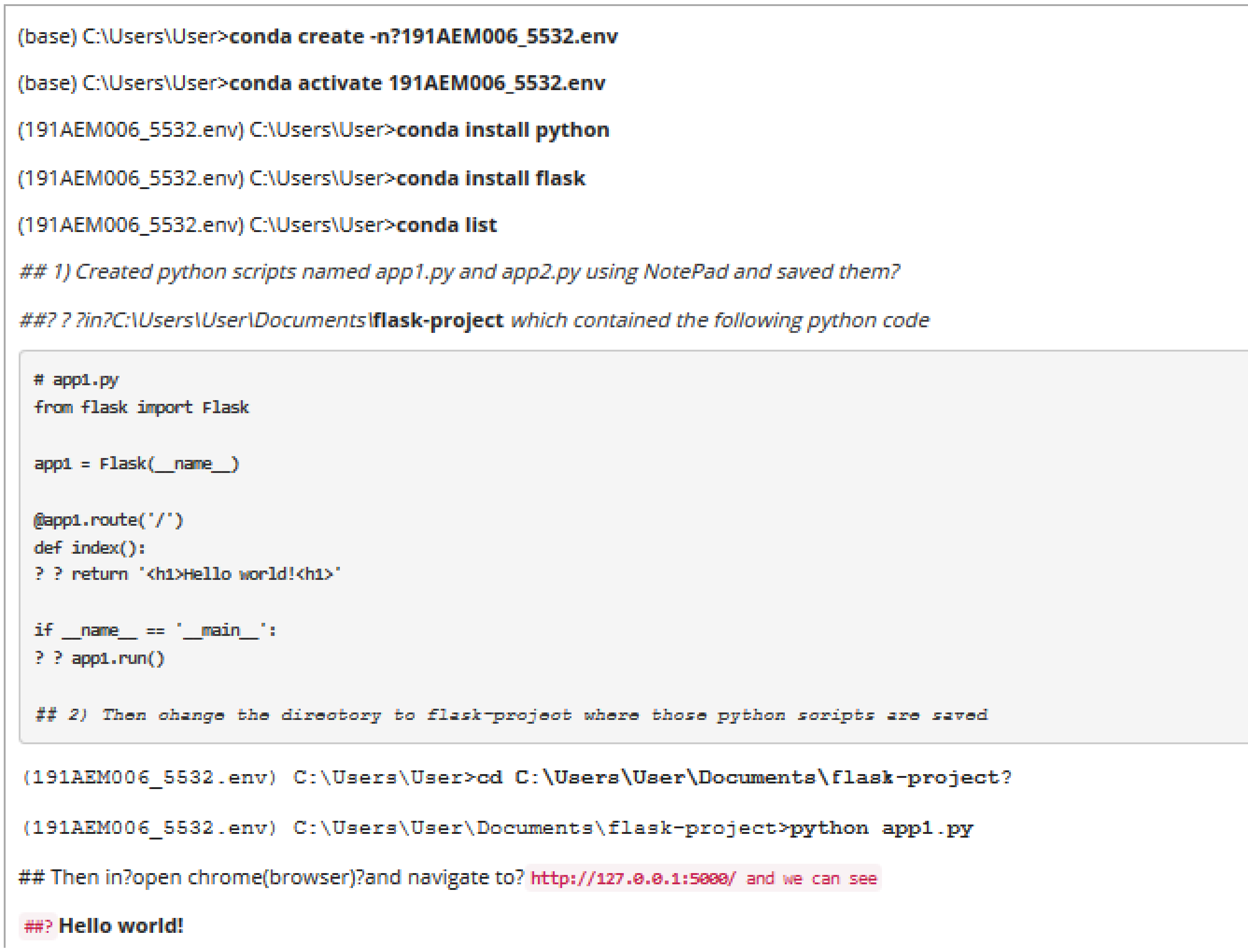
iii) Creating Resources;

iv) ItemList;

v) Authentication and logging;

vi) Resource storage in SQL database.

At the very beginning we created a simple application "Hello world" using flask and execute it in terminal as an example. We save it as **app.py** I used the Anaconda prompt as the terminal. Here are the example. And also we created a git repositary to upload relevant codes which is " **https://github.com/SandaliW/RAI 553-2-summer-2019** ".

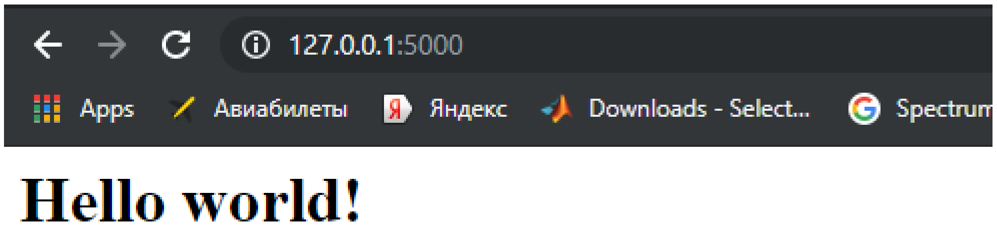


```
(base) C:\Users\User>conda create -n?191AEM006_5532.env

(base) C:\Users\User>conda activate 191AEM006_5532.env

(191AEM006_5532.env) C:\Users\User>conda install python

(191AEM006_5532.env) C:\Users\User>conda install flask

(191AEM006_5532.env) C:\Users\User>conda list

## 1) Created python scripts named app1.py and app2.py using NotePad and saved them?

##? ? ?in?C:\Users\User\Documents\flask-project which contained the following python code
```

```
# app1.py
from flask import Flask

app1 = Flask(__name__)

@app1.route('/')
def index():
? ? return '<h1>Hello world!<h1>'

if __name__ == '__main__':
? ? app1.run()

## 2) Then change the directory to flask-project where those python scripts are saved
```

```
(191AEM006_5532.env) C:\Users\User>cd C:\Users\User\Documents\flask-project?

(191AEM006_5532.env) C:\Users\User\Documents\flask-project>python app1.py

## Then in?open chrome(browser)?and navigate to? http://127.0.0.1:5000/ and we can see

##? Hello world!
```

Figure 1: Code


127.0.0.1:5000

Apps Авиабилеты Яндекс Downloads - Select... Spectrum

Hello world!

Figure 2: Hello World

For the next step we need to install **flask-restful** package which is an extension of Flask. It is a lightweight abstraction that works with our existing libraries which will be usefull to create REST APIs more quickly.In this step of the project we have created a very simple REST API which state that newly created resource "Student" will be accessible via API. Then including GET, we will practice creating requests like PUT, POST, DELETE etc.

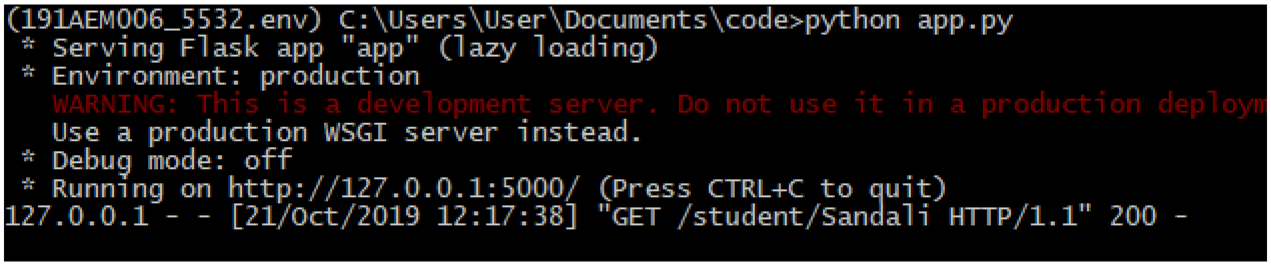
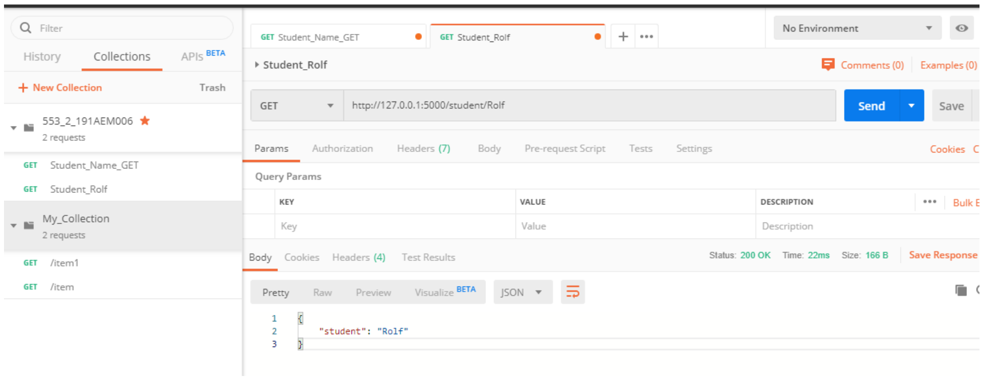

```
(191AEM006_5532.env) C:\Users\User\Documents\code>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deploym
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Oct/2019 12:17:38] "GET /student/Sandali HTTP/1.1" 200 -
```

Figure 3: Terminal

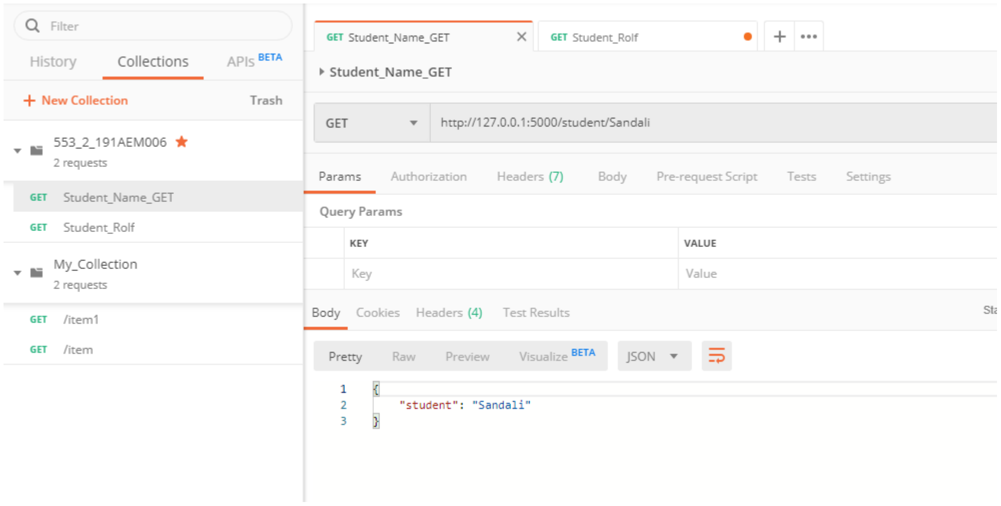


GET http://127.0.0.1:5000/student/Rolf

Status: 200 OK Time: 22ms Size: 166 B

```
{
    "student": "Rolf"
}
```

Figure 4: Student name

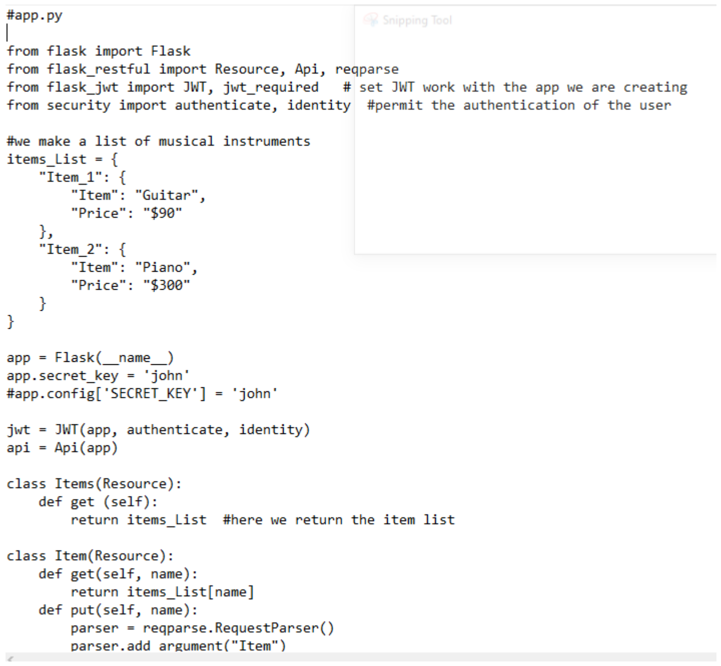


GET http://127.0.0.1:5000/student/Sandali

```
{
    "student": "Sandali"
}
```

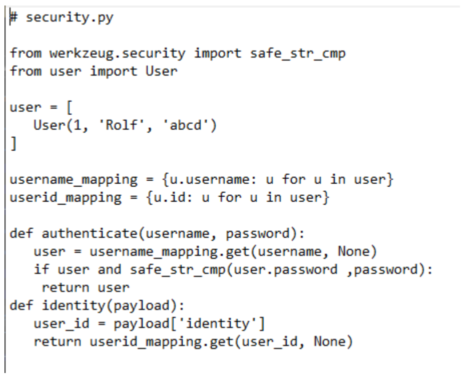
Figure 5: Student name

Next we need to create ItemList using JSON payload and design security part like authentication and logging bu using additional **security.py and user.py**. For that first we need to install,
flask-jwt(adds basic JWT features to our application)
Werkzeug (A debug server)libraries.
And also we will apply advanced request parsing (to make sure we can only pass certain fields through our JSON payload via our endpoints). We also need to apply additional library **rqparse**. Since I am using conda environment i have to use the install flask-jwt by clonning into a git repository using the link;
" git clone https://github.com/mattupstate/flask-jwt.git ./flask-jwt pip install ./flask-jwt ".
After successful installation of above libraries we need to create other two .py files. Here I used app, security and user.py files.
All of them has been pushed to my git repositary.



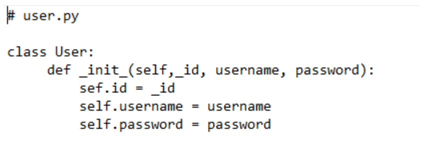
```
#app.py

from flask import Flask
from flask_restful import Resource, Api, reqparse
from flask_jwt import JWT, jwt_required   # set JWT work with the app we are creating
from security import authenticate, identity  #permit the authentication of the user

#we make a list of musical instruments
items_List = {
    "Item_1": {
        "Item": "Guitar",
        "Price": "$90"
    },
    "Item_2": {
        "Item": "Piano",
        "Price": "$300"
    }
}

app = Flask(__name__)
app.secret_key = 'john'
#app.config['SECRET_KEY'] = 'john'

jwt = JWT(app, authenticate, identity)
api = Api(app)

class Items(Resource):
    def get (self):
        return items_List  #here we return the item list

class Item(Resource):
    def get(self, name):
        return items_List[name]
    def put(self, name):
        parser = reqparse.RequestParser()
        parser.add_argument("Item")
```

Figure 6: Part of app.py



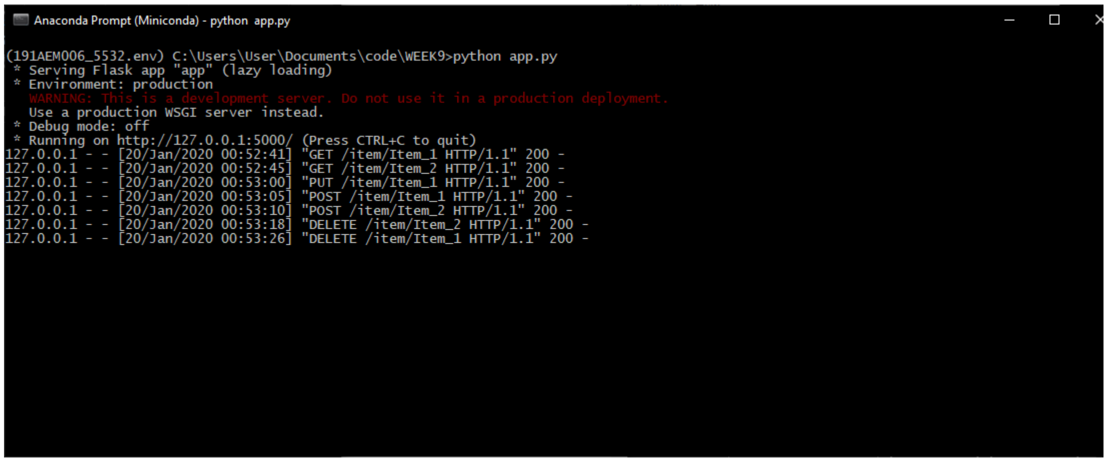
```
# security.py

from werkzeug.security import safe_str_cmp
from user import User

user = [
   User(1, 'Rolf', 'abcd')
]

username_mapping = {u.username: u for u in user}
userid_mapping = {u.id: u for u in user}

def authenticate(username, password):
   user = username_mapping.get(username, None)
   if user and safe_str_cmp(user.password ,password):
    return user
def identity(payload):
   user_id = payload['identity']
   return userid_mapping.get(user_id, None)
```

Figure 7: security.py



```
# user.py

class User:
     def _init_(self,_id, username, password):
         sef.id = _id
         self.username = username
         self.password = password
```

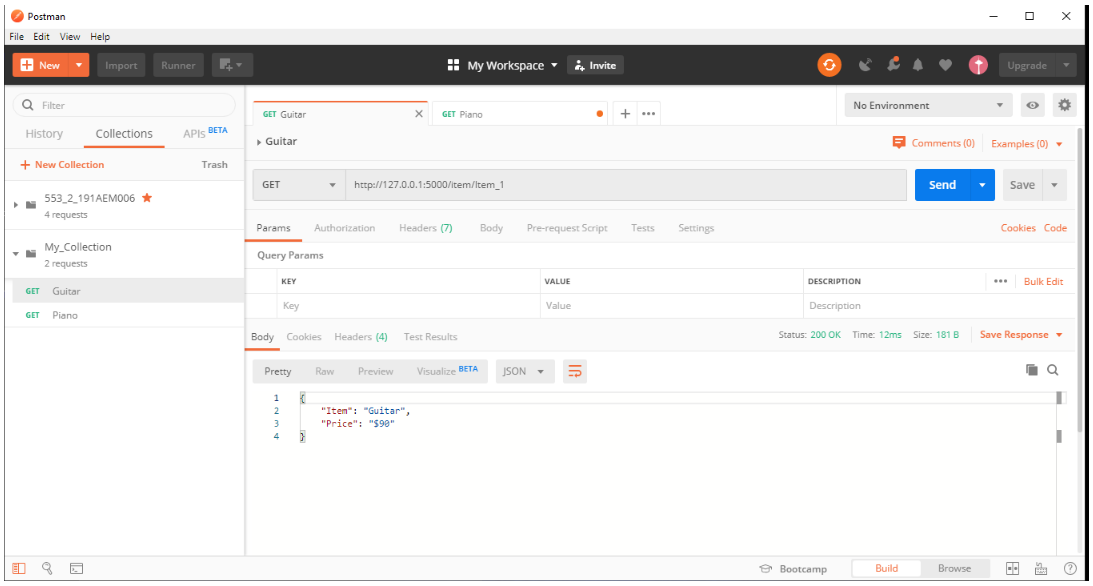
Figure 8: user.py



```
(191AEM006_5532.env) C:\Users\User\Documents\code\WEEK9>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [20/Jan/2020 00:52:41] "GET /item/Item_1 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:52:45] "GET /item/Item_2 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:53:00] "PUT /item/Item_1 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:53:05] "POST /item/Item_1 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:53:10] "POST /item/Item_2 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:53:18] "DELETE /item/Item_2 HTTP/1.1" 200 -
127.0.0.1 - - [20/Jan/2020 00:53:26] "DELETE /item/Item_1 HTTP/1.1" 200 -
```

Figure 9: terminal

And we can get following results.

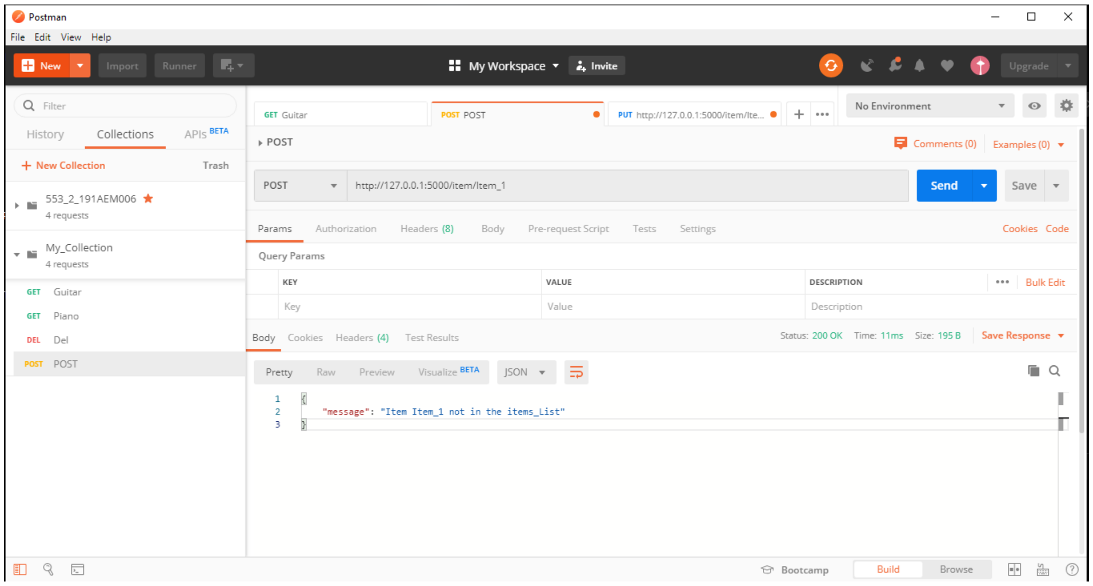


GET http://127.0.0.1:5000/item/Item_1

Status: 200 OK Time: 12ms Size: 181 B

```
{
    "Item": "Guitar",
    "Price": "$90"
}
```

Figure 10: Result shown in GET request



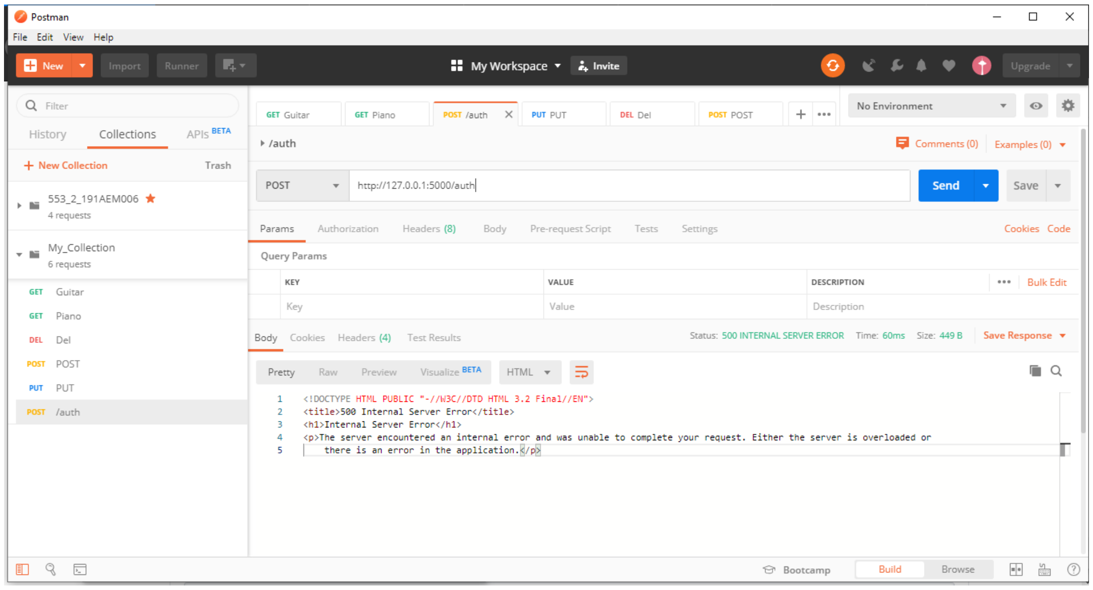
POST http://127.0.0.1:5000/item/Item_1

Status: 200 OK Time: 11ms Size: 195 B

```
{
    "message": "Item Item_1 not in the items_List"
}
```

Figure 11: POST request

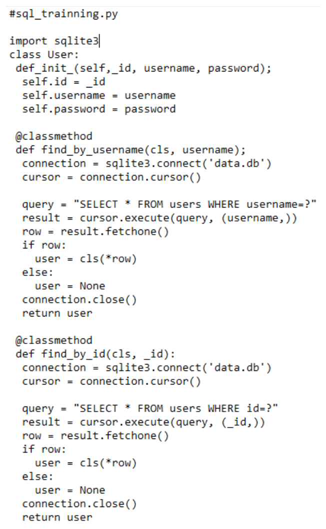
Even though I successfully executed those .py files even got results, I could not get the results for authentication and log in results. I already checked everything including libraries and codes but I could not find any error. I always get this ”internal Server Error” in Postman.



```
POST  http://127.0.0.1:5000/auth
Status: 500 INTERNAL SERVER ERROR   Time: 60ms   Size: 449 B

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or
    there is an error in the application.</p>
```

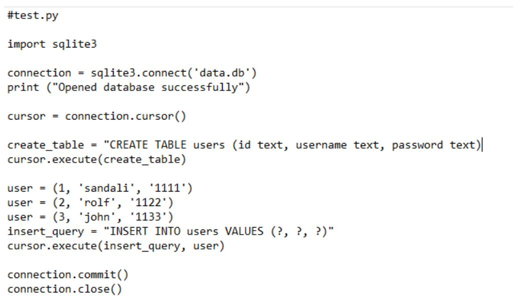
Figure 12: Error

Then we will study how to store and manage resources in SQL database using flask-REST API. We will create an app;ication which will allow users to sign up with an ID, username and password by application of SQLite database. Respectively, we will use Sqlite outside API and integrate with our API afterwards. For that we need to install **sqlite3**.



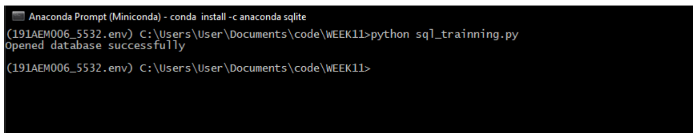
```
#sql_trainning.py

import sqlite3
class User:
 def_init_(self,_id, username, password);
  self.id = _id
  self.username = username
  self.password = password

 @classmethod
 def find_by_username(cls, username);
  connection = sqlite3.connect('data.db')
  cursor = connection.cursor()

  query = "SELECT * FROM users WHERE username=?"
  result = cursor.execute(query, (username,))
  row = result.fetchone()
  if row:
    user = cls(*row)
  else:
    user = None
  connection.close()
  return user

 @classmethod
 def find_by_id(cls, _id):
  connection = sqlite3.connect('data.db')
  cursor = connection.cursor()

  query = "SELECT * FROM users WHERE id=?"
  result = cursor.execute(query, (_id,))
  row = result.fetchone()
  if row:
    user = cls(*row)
  else:
    user = None
  connection.close()
  return user
```

Figure 13: sql-trainning.py



```
#test.py

import sqlite3

connection = sqlite3.connect('data.db')
print ("Opened database successfully")

cursor = connection.cursor()

create_table = "CREATE TABLE users (id text, username text, password text)
cursor.execute(create_table)

user = (1, 'sandali', '1111')
user = (2, 'rolf', '1122')
user = (3, 'john', '1133')
insert_query = "INSERT INTO users VALUES (?, ?, ?)"
cursor.execute(insert_query, user)

connection.commit()
connection.close()
```

Figure 14: test.py



```
Anaconda Prompt (Miniconda) - conda  install -c anaconda sqlite
(191AEM006_5532.env) C:\Users\User\Documents\code\WEEK11>python sql_trainning.py
Opened database successfully

(191AEM006_5532.env) C:\Users\User\Documents\code\WEEK11>
```

Figure 15: Open Database

After running in the terminal a data,db file will be created evrytime and we need to delete the previous data.db file before running the code next time. Now we are in the final stage. Here we need to confirm following steps of our project;
Logging and retrieving Users from a database;
Signing up and writing Users to a database;
Learn how to prevent duplicate usernames when signing users up;
Retrieve our Item resources from a database;
Write our Item resources to a database;
Delete Item resources from the database;
Refactoring insertion of items;
Apply the PUT method with database interaction;
Retrieve many items from the database;

In my case I could not identify the error and I could not have access to the last, log in and storage part.

4 Conclusion

It is very efficient to use REST API in web application development. In this project I had learn a lot about web application and It is very difficult to use Windows operating system for web application development.

5 Referneces

https://github.com/SandaliW/RAE-553-2-2019-191AEM006/tree/code–rest$_api-project$

$https://pythonhosted.org/Flask-JWT/$

$https://anaconda.org/anaconda/sqlite$