Rest Api with Flask in Python

Amal Mathews Antu 201AEM021

May 2021

1 Introduction

REST API: Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. REST API is a way of accessing the web services in a simple and flexible way without having any processing. REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses the less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web services. All communication done via REST API used only HTTP request.

2 Creating Simple REST API with flask

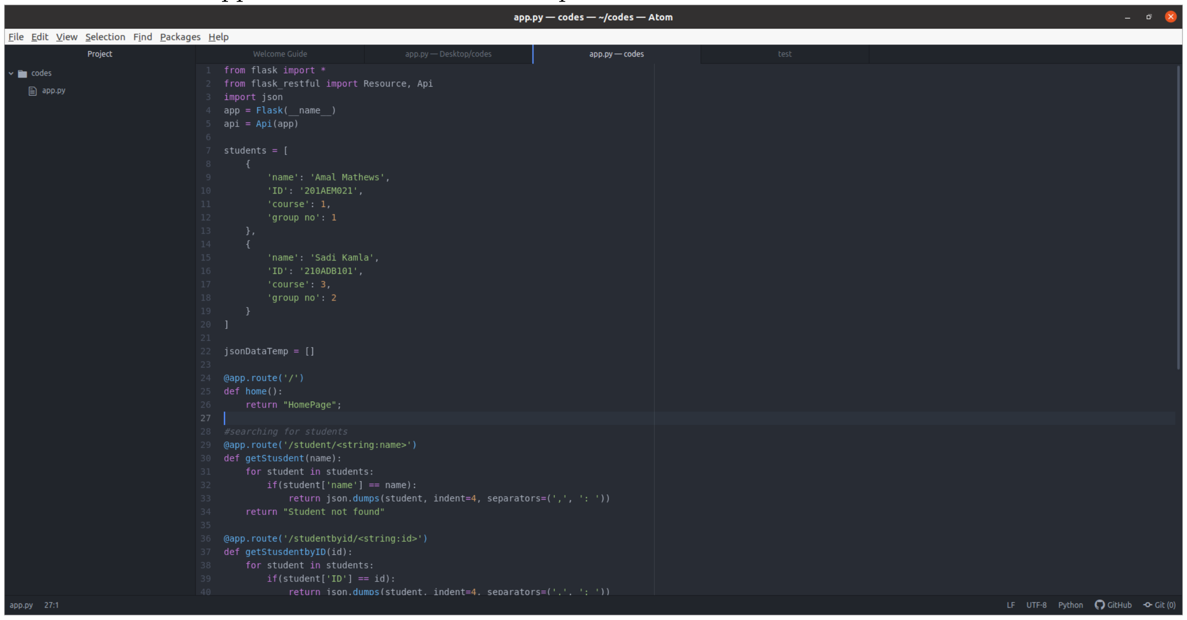
Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins.

2.1 Creating a virtual environment using miniconda

Once miniconda installation is successful, created virtual environment using command below conda create -n test1. We can Check the new environment in the list using command: conda env list. Switching to newly-created environment, using command: source activate test1 Install a flask package using command conda install flask

2.2 Creating a new file giving it a name app.py

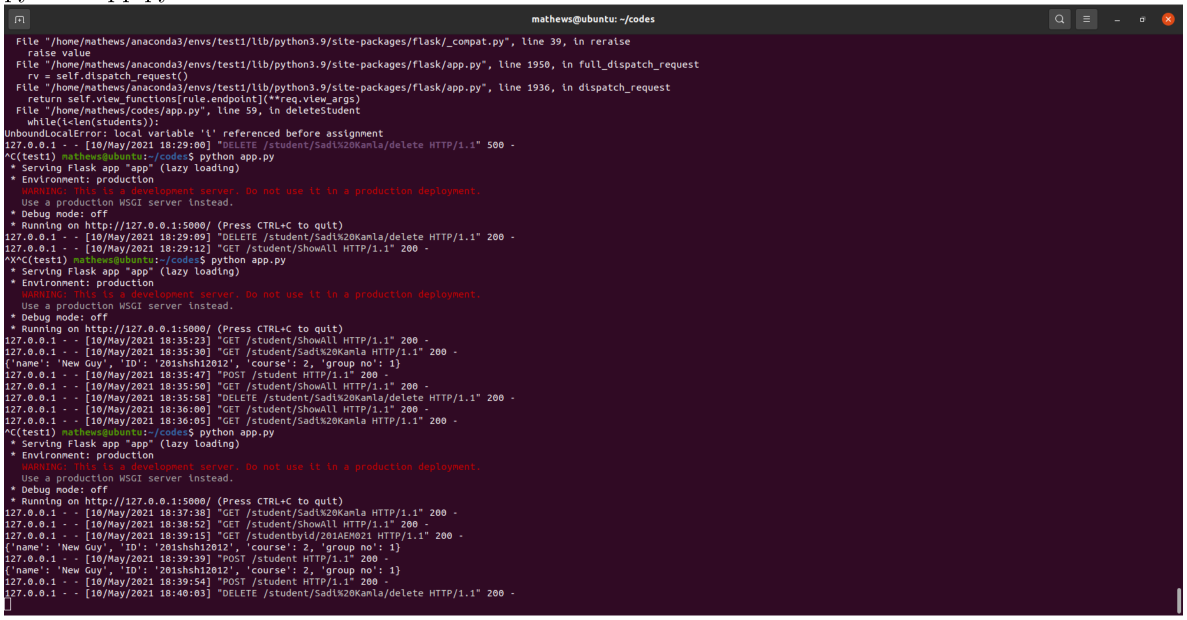
When Atom is running we created a new file giving it a name app.py. Important to remember that we will be running this file from virtual environments so flask and flask restful packages should be already installed in this environment. Flask restful package (pip install flask-restful and conda install flask) The code for our application is shown in below picture.


app.py — codes — ~/codes — Atom
File Edit View Selection Find Packages Help
Project
codes
app.py
Welcome Guide
app.py — Desktop/codes
app.py — codes
test
app.py 27:1
LF UTF-8 Python GitHub Git (0)

```python
from flask import *
from flask_restful import Resource, Api
import json
app = Flask(__name__)
api = Api(app)

students = [
    {
        'name': 'Amal Mathews',
        'ID': '201AEM021',
        'course': 1,
        'group no': 1
    },
    {
        'name': 'Sadi Kamla',
        'ID': '210ADB101',
        'course': 3,
        'group no': 2
    }
]

jsonDataTemp = []

@app.route('/')
def home():
    return "HomePage";

#searching for students
@app.route('/student/<string:name>')
def getStusdent(name):
    for student in students:
        if(student['name'] == name):
            return json.dumps(student, indent=4, separators=(',', ': '))
    return "Student not found"

@app.route('/studentbyid/<string:id>')
def getStusdentbyID(id):
    for student in students:
        if(student['ID'] == id):
            return json.dumps(student, indent=4, separators=(',', ': '))
```

2.3 Terminal

In virtual environment we are working in and launching app with a command:
python app.py



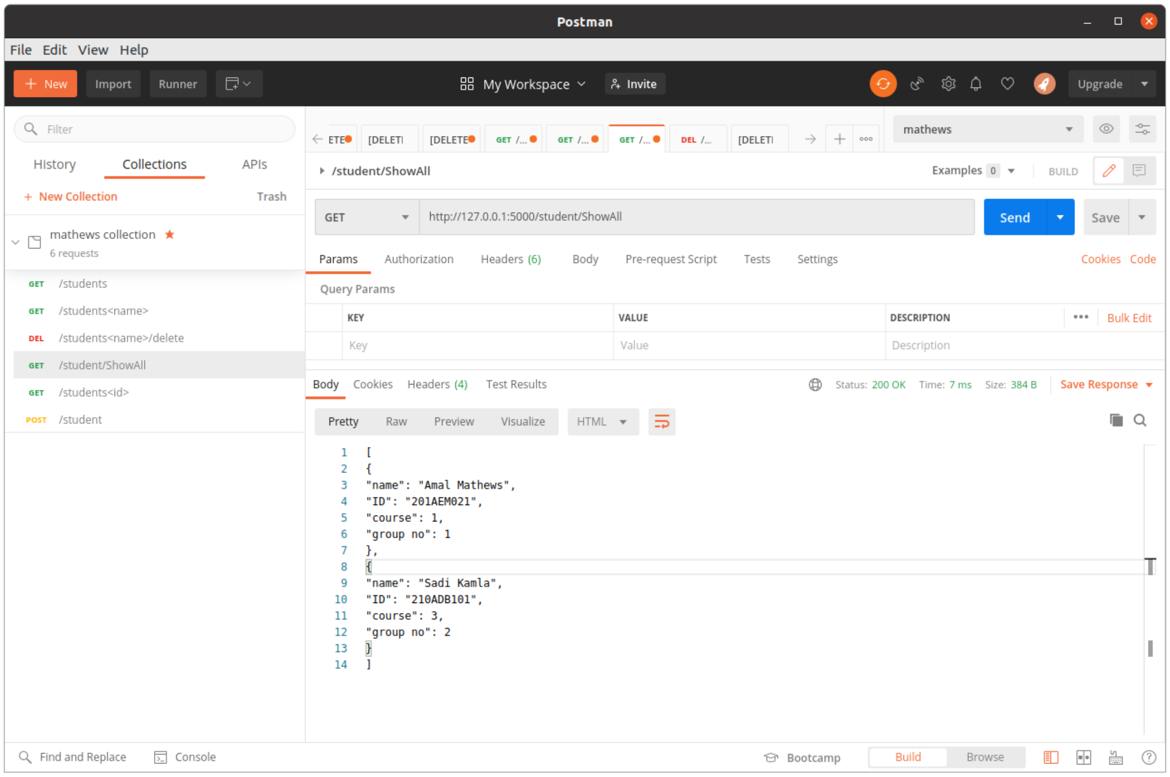
```
  File "/home/mathews/anaconda3/envs/test1/lib/python3.9/site-packages/flask/_compat.py", line 39, in reraise
    raise value
  File "/home/mathews/anaconda3/envs/test1/lib/python3.9/site-packages/flask/app.py", line 1950, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/mathews/anaconda3/envs/test1/lib/python3.9/site-packages/flask/app.py", line 1936, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "/home/mathews/codes/app.py", line 59, in deleteStudent
    while(i<len(students)):
UnboundLocalError: local variable 'i' referenced before assignment
127.0.0.1 - - [10/May/2021 18:29:00] "DELETE /student/Sadi%20Kamla/delete HTTP/1.1" 500 -
^C(test1) mathews@ubuntu:~/codes$ 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 - - [10/May/2021 18:29:09] "DELETE /student/Sadi%20Kamla/delete HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:29:12] "GET /student/ShowAll HTTP/1.1" 200 -
^X^C(test1) mathews@ubuntu:~/codes$ 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 - - [10/May/2021 18:35:23] "GET /student/ShowAll HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:35:30] "GET /student/Sadi%20Kamla HTTP/1.1" 200 -
{'name': 'New Guy', 'ID': '201shsh12012', 'course': 2, 'group no': 1}
127.0.0.1 - - [10/May/2021 18:35:47] "POST /student HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:35:50] "GET /student/ShowAll HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:35:58] "DELETE /student/Sadi%20Kamla/delete HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:36:00] "GET /student/ShowAll HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:36:05] "GET /student/Sadi%20Kamla HTTP/1.1" 200 -
^C(test1) mathews@ubuntu:~/codes$ 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 - - [10/May/2021 18:37:38] "GET /student/Sadi%20Kamla HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:38:52] "GET /student/ShowAll HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:39:15] "GET /studentbyid/201AEM021 HTTP/1.1" 200 -
{'name': 'New Guy', 'ID': '201shsh12012', 'course': 2, 'group no': 1}
127.0.0.1 - - [10/May/2021 18:39:39] "POST /student HTTP/1.1" 200 -
{'name': 'New Guy', 'ID': '201shsh12012', 'course': 2, 'group no': 1}
127.0.0.1 - - [10/May/2021 18:39:54] "POST /student HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2021 18:40:03] "DELETE /student/Sadi%20Kamla/delete HTTP/1.1" 200 -
```

3 Design and creating my RESTful Web-service

3.1 Application for student register

This is simple Student register and try to apply all of the methods: GET, POST, PUT and DELETE

3.2 Showing All students



GET /students
GET /students<name>
DEL /students<name>/delete
GET /student/ShowAll
GET /students<id>
POST /student

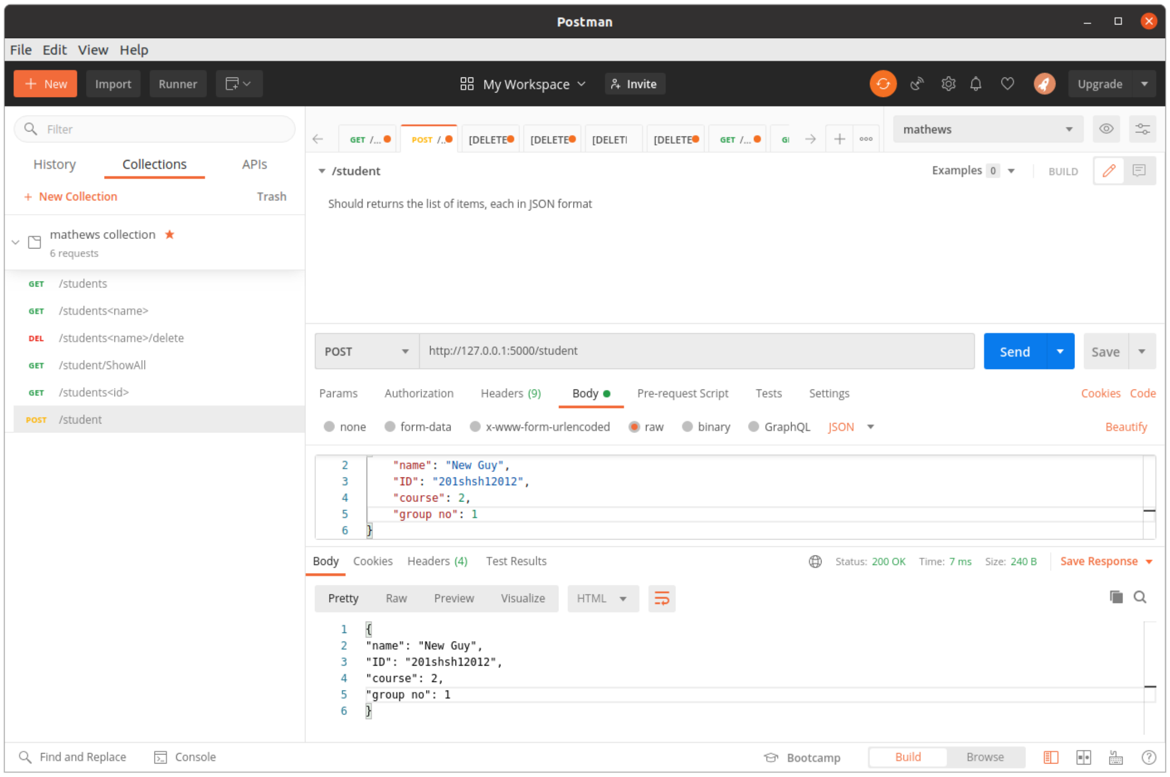
/student/ShowAll

GET http://127.0.0.1:5000/student/ShowAll

Status: 200 OK Time: 7 ms Size: 384 B

```
[
{
"name": "Amal Mathews",
"ID": "201AEM021",
"course": 1,
"group no": 1
},
{
"name": "Sadi Kamla",
"ID": "210ADB101",
"course": 3,
"group no": 2
}
]
```

3.3 Adding new student



/student

Should returns the list of items, each in JSON format

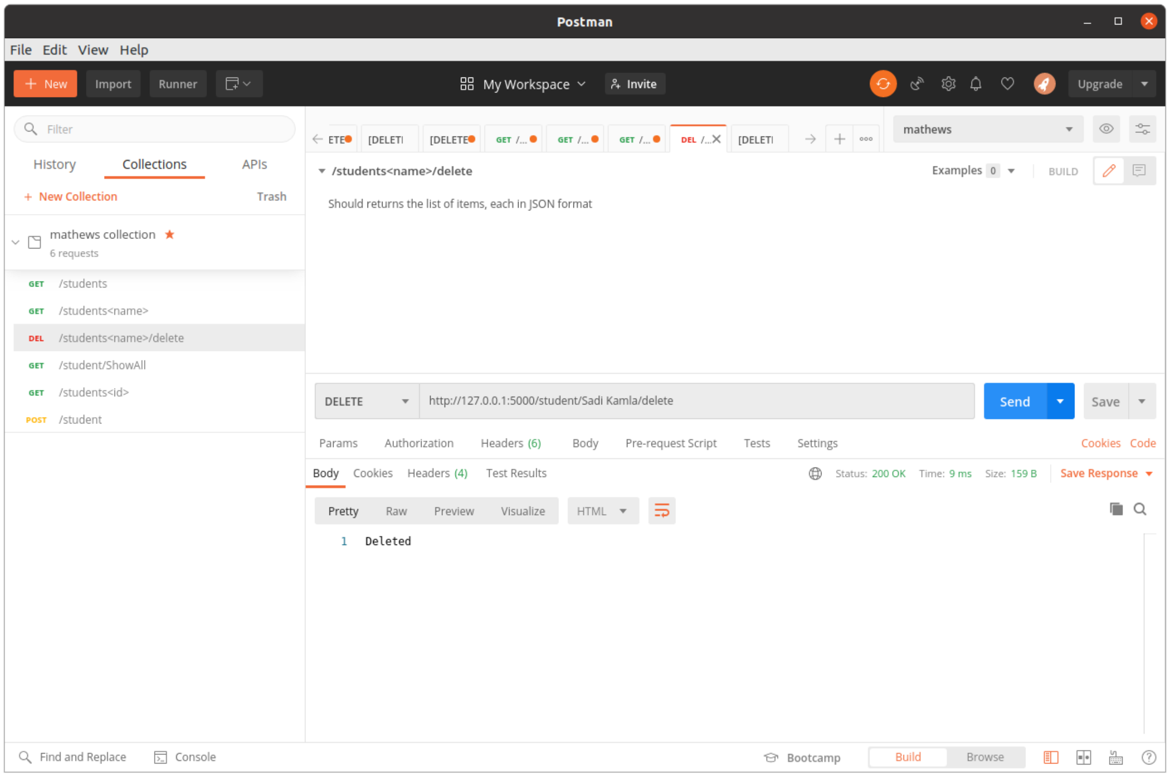
POST http://127.0.0.1:5000/student

```
    "name": "New Guy",
    "ID": "201shsh12012",
    "course": 2,
    "group no": 1
}
```

Status: 200 OK Time: 7 ms Size: 240 B

```
{
"name": "New Guy",
"ID": "201shsh12012",
"course": 2,
"group no": 1
}
```

3.4 Deleting students



/students<name>/delete

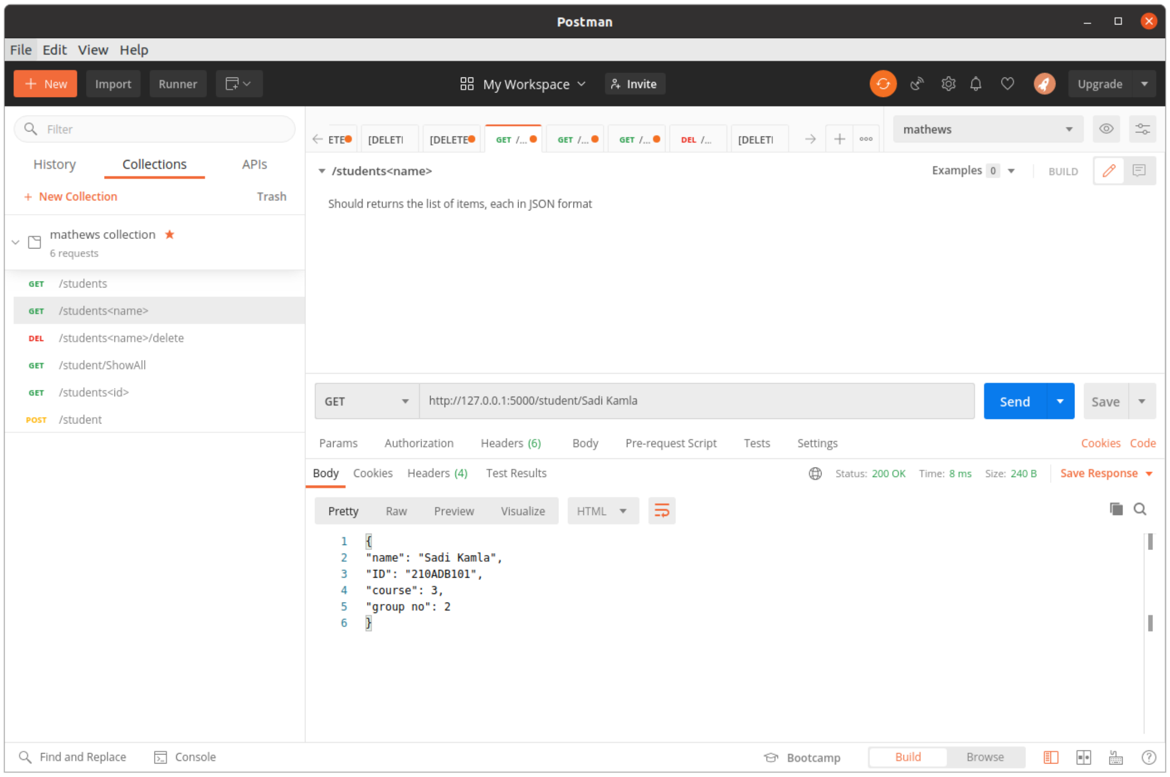
Should returns the list of items, each in JSON format

DELETE http://127.0.0.1:5000/student/Sadi Kamla/delete

Status: 200 OK Time: 9 ms Size: 159 B

```
Deleted
```

3.5 Finding students

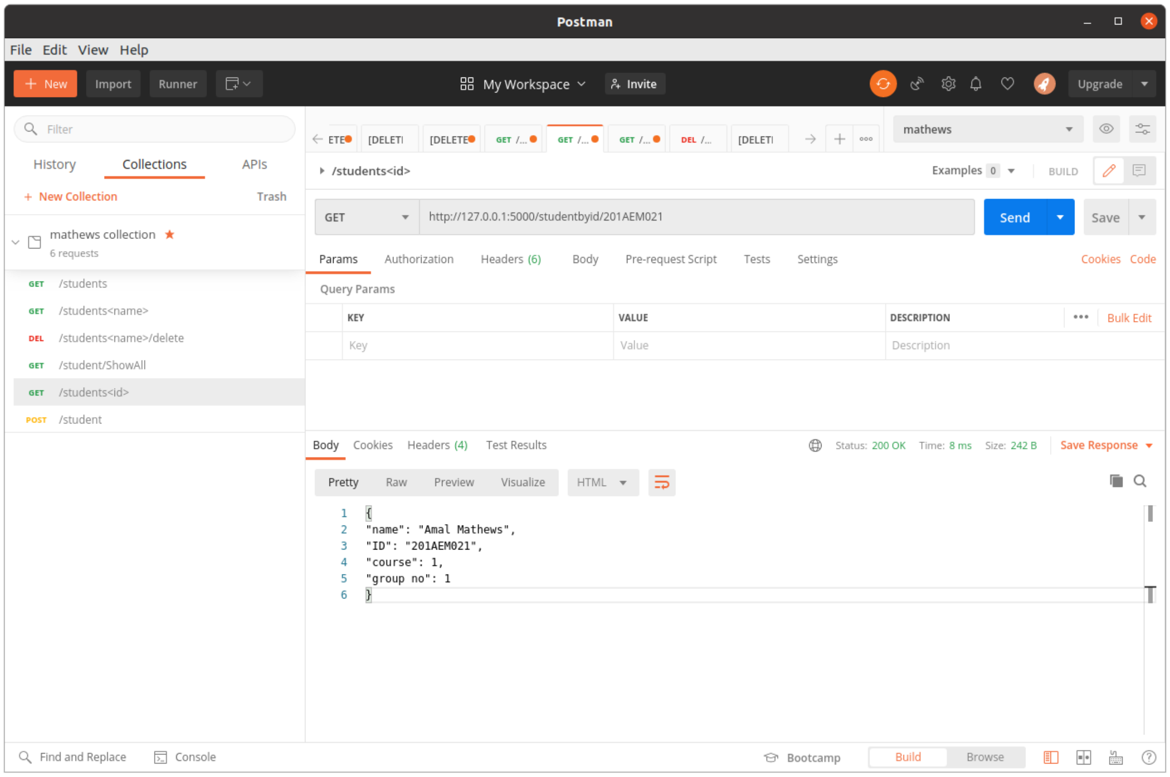


Should returns the list of items, each in JSON format

GET http://127.0.0.1:5000/student/Sadi Kamla

```
{
"name": "Sadi Kamla",
"ID": "210ADB101",
"course": 3,
"group no": 2
}
```

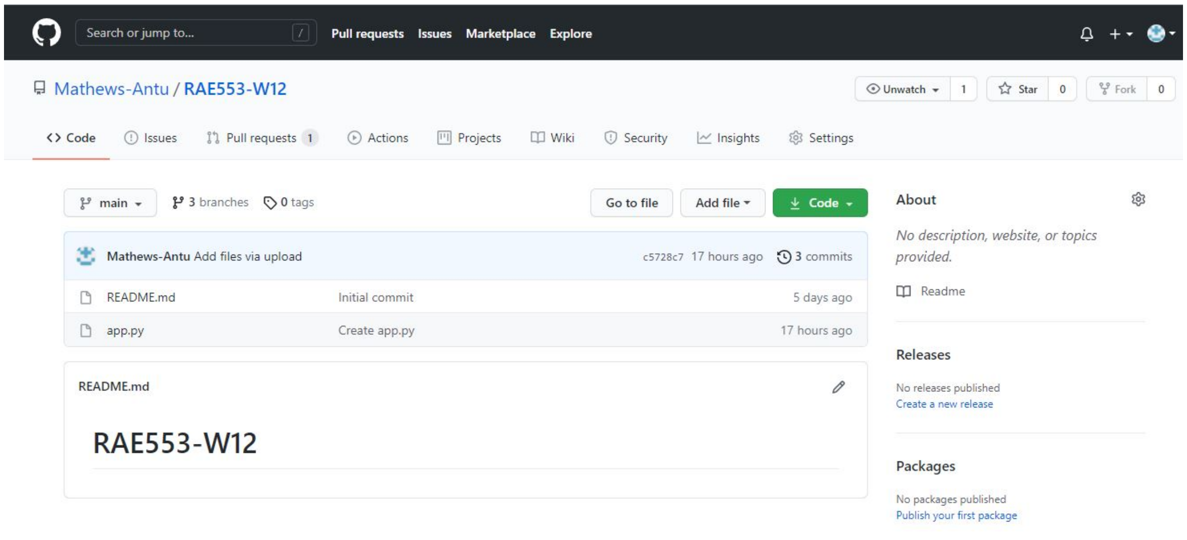
3.6 Finding students by ID



GET http://127.0.0.1:5000/studentbyid/201AEM021

```
{
"name": "Amal Mathews",
"ID": "201AEM021",
"course": 1,
"group no": 1
}
```

3.7 Out Put of Github



Mathews-Antu Add files via upload	c5728c7 17 hours ago	3 commits
README.md	Initial commit	5 days ago
app.py	Create app.py	17 hours ago

README.md

RAE553-W12

About

No description, website, or topics provided.

Readme

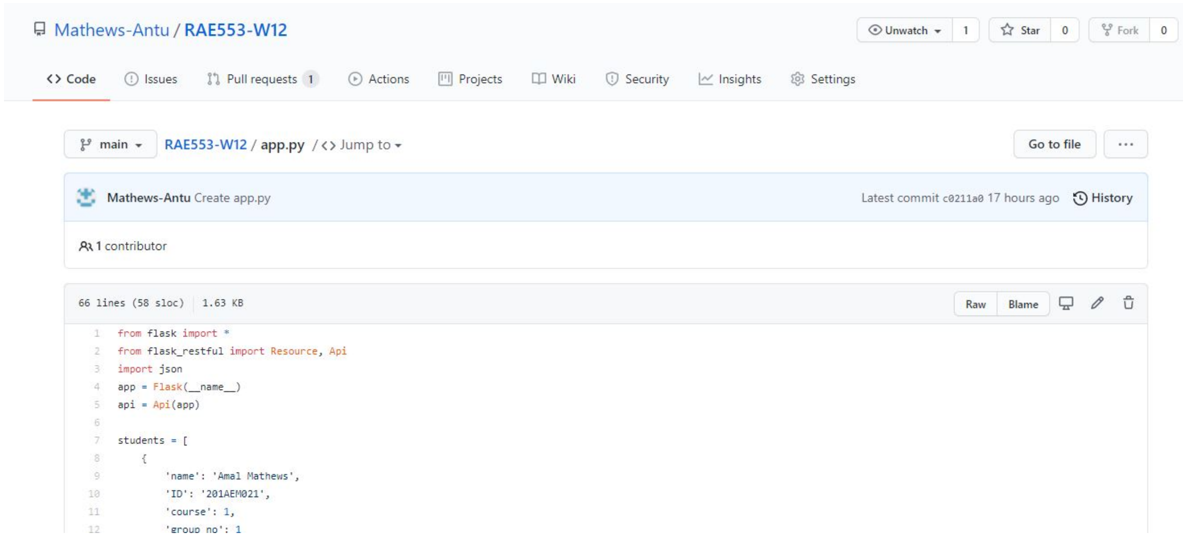
Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

3.8 Github Code



Mathews-Antu Create app.py   Latest commit c0211a0 17 hours ago   History

1 contributor

66 lines (58 sloc) | 1.63 KB

```
from flask import *
from flask_restful import Resource, Api
import json
app = Flask(__name__)
api = Api(app)

students = [
    {
        'name': 'Amal Mathews',
        'ID': '201AEM021',
        'course': 1,
        'group no': 1
```

3.9 How RESTful APIs work

A RESTful API breaks down a transaction to create a series of small modules. Each module addresses a particular underlying part of the transaction. This modularity provides developers with a lot of flexibility, but it can be challenging for developers to design their REST API from scratch. Currently, several companies provide models for developers to use; the models provided by Amazon S3, Cloud Data Management Interface (CDMI) and OpenStack Swift are the most popular. A RESTful API uses existing HTTP methodologies defined by the RFC 2616 protocol. They use 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. With REST, networked components are a resource the user requests access to – a black box whose implementation details are unclear. All calls are stateless; nothing can be retained by the RESTful service between executions.

3.10 USES

Because the calls are stateless, REST is useful in cloud applications. Stateless components can be freely redeployed if something fails, and they can scale to accommodate load changes. This is because any request can be directed to any instance of a component; there can be nothing saved that has to be remembered by the next transaction. That makes REST preferable for web use, but the RESTful model is also helpful in cloud services because binding to a service through an API is a matter of controlling how the URL is decoded. Cloud computing and microservices are almost certain to make RESTful API design the rule in the future

3.11 DISCUSSION

In this course work we studied mainly about creating different application program interfaces.different tools for helping this to work properly.mainly familiar with ubuntu.And the main topics are rest api,Oauth,web server and servies.we got familiarized with github,postman and learned the different method like get,put,post,delete. Web server is basically a program that uses HTTP to serve files that create web pages to users in based on their requests, which are forwarded by their computer's HTTP connection. ... In other words, a Web server is an Internet server that responds to HTTP requests to deliver the content and services. To implement our knowledge we gained through lecture classes several lab sessions were provided as assignment. The final part of the course work we developed an API of our own.And all the corresponding works are gitrepository.