Audrey Long

JHU Web Security

Client-Side Module Assignment

02/09/2021

Introduction
In this assignment you will implement two REST endpoint methods (GET, POST) using the Flask microframework (Python). Flask is a very lightweight and flexible web framework and makes endpoint development very simple. Your endpoint will maintain a" to do" list and allow you to make an HTTP GET request to read a list of tasks as well as make an HTTP POST request to add a task to the list. Our application will return the task data to the caller using the JSON format we discussed in the lecture.

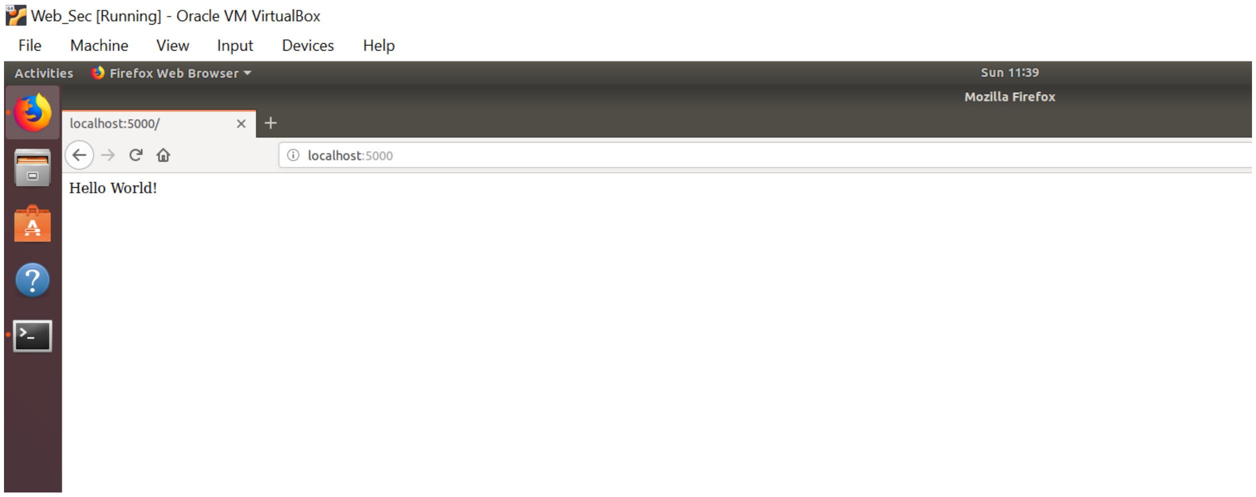
Assignment Details
You will need an Ubuntu VM for this assignment so feel free to use your VM from last week or spin up a brand new one. Instructions for spinning up a new VM can be found in the Week 1 homework.

- **The submission section asks for you to capture screenshots of meaningful output each time you perform either a GET or POST to the endpoint, so keep that in mind as you proceed.**
- **Visit: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask There you will find a terrific guide for creating your first web service. Read the first couple of sections for a nice, brief discussion of what REST endpoints are. In the section entitled "A brief introduction to the Flask microframework" you will install Flask. Feel free to follow the instructions as they are, but using a virtual environment is optional.**
  - **If your VM does not have Python and pip installed, just use sudo apt-get install ¡name¿ to install the appropriate packages.**
  - **For the next sections you can use your favorite text editor (vim, nano, emacs, etc.) or, if you prefer an IDE, you can try installing PyCharm.**
  - **Complete each of the sections and take a screenshot for each 'curl' command you use to test along the way. Store the screenshots in a Word document or PDF.**
- ****STOP WHEN YOU GET TO THE SECTION ENTITLED "Improving the web service interface". Finally, skip down to the section called "Securing a RESTful web service".**
- **Read about some of the security concerns surrounding our somewhat naive 1 REST endpoint.**

Submission Deliverables
In One PDF please provide:
1. Please document the screenshots with one line descriptions of each time you perform either a GET or POST to the endpoint (aka, anytime there is meaningful output).

• Please also submit your Python source code through Blackboard in a file called

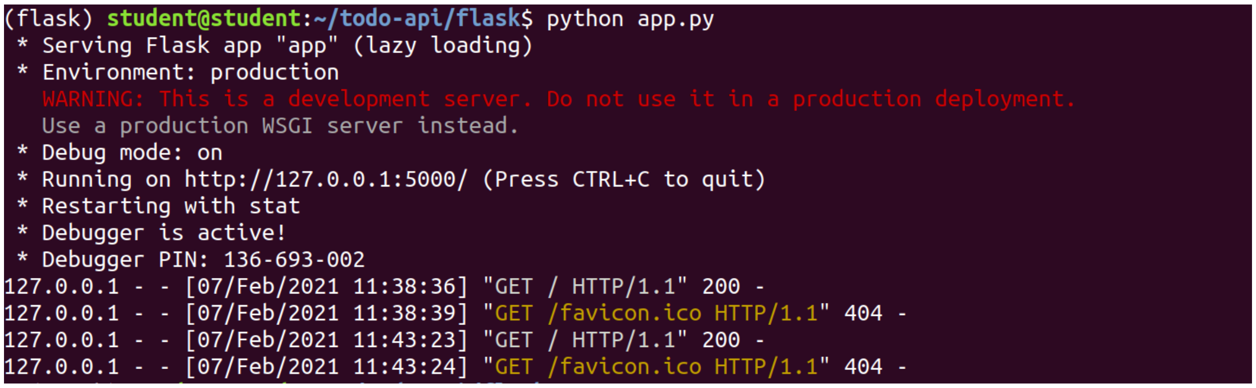
endpoint.py.txt (Please do not zip.)



Hello World!

Figure 1: web server up and running

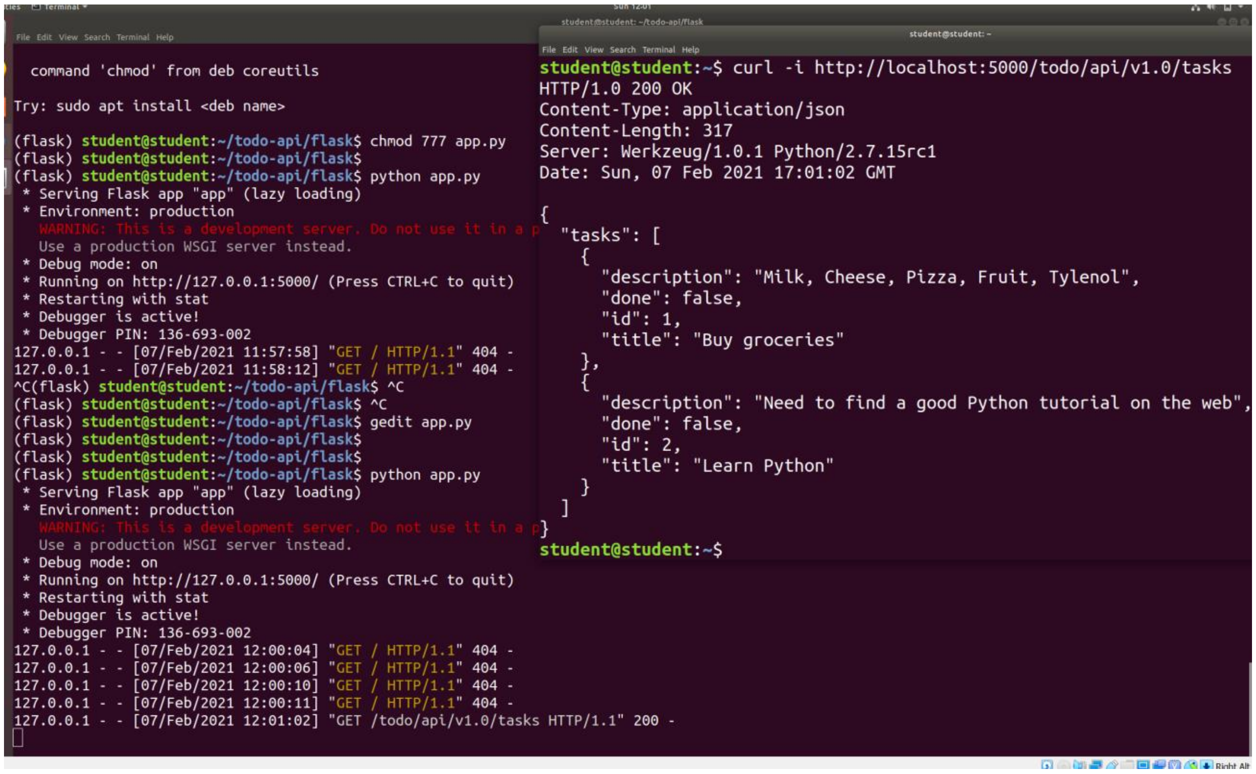
The local web server up and running from python flask commands.



```
(flask) student@student:~/todo-api/flask$ 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: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 11:38:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Feb/2021 11:38:39] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 11:43:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Feb/2021 11:43:24] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 2: web server up and running

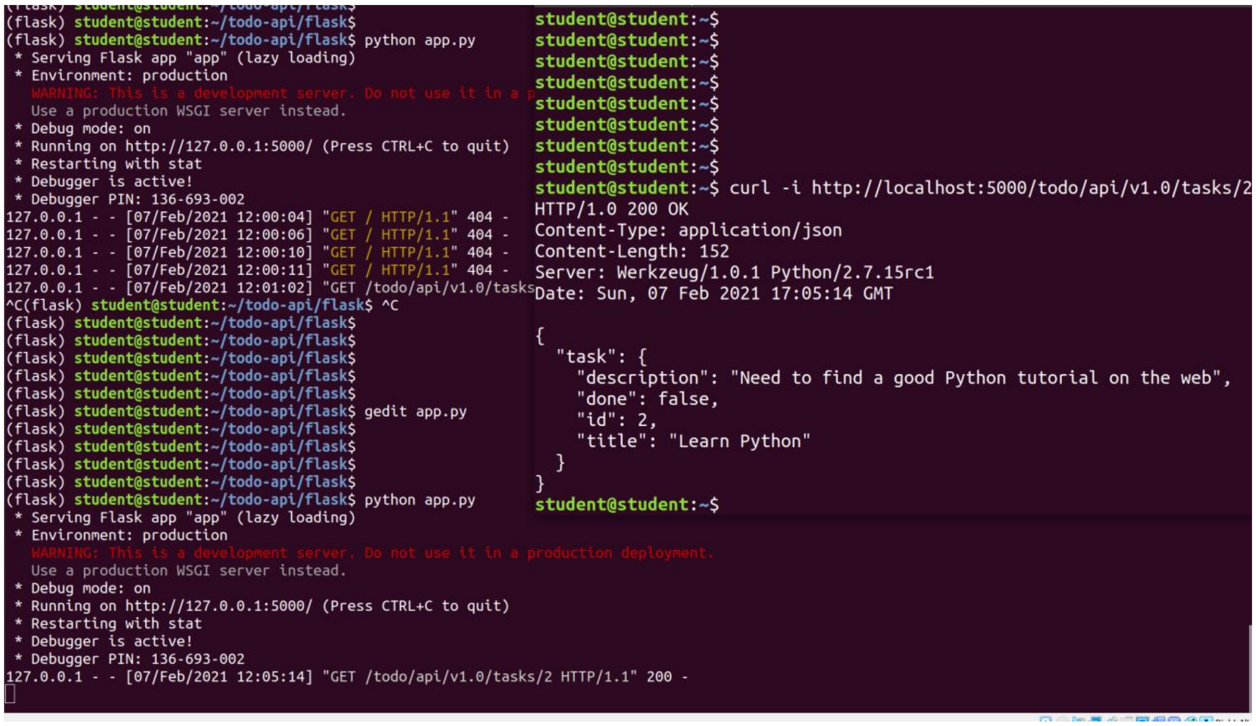
The CLI screenshot of the up and running server



```
 command 'chmod' from deb coreutils

Try: sudo apt install <deb name>

(flask) student@student:~/todo-api/flask$ chmod 777 app.py
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a p
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 11:57:58] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 11:58:12] "GET / HTTP/1.1" 404 -
^C(flask) student@student:~/todo-api/flask$ ^C
(flask) student@student:~/todo-api/flask$ ^C
(flask) student@student:~/todo-api/flask$ gedit app.py
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a p
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 12:00:04] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:06] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:10] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:11] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:01:02] "GET /todo/api/v1.0/tasks HTTP/1.1" 200 -
```

```
student@student:~$ curl -i http://localhost:5000/todo/api/v1.0/tasks
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 317
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 17:01:02 GMT

{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    }
  ]
}
student@student:~$
```

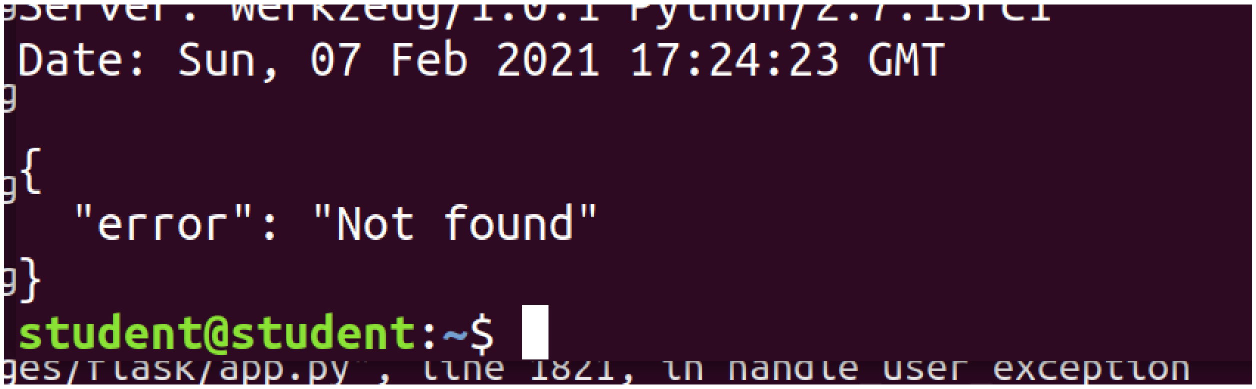
Figure 3: Nominal GET request
Curl command and getting response from web server for a GET request.



```
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a p
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 12:00:04] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:06] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:10] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:00:11] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2021 12:01:02] "GET /todo/api/v1.0/tasks
^C(flask) student@student:~/todo-api/flask$ ^C
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ gedit app.py
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ 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: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 12:05:14] "GET /todo/api/v1.0/tasks/2 HTTP/1.1" 200 -
```

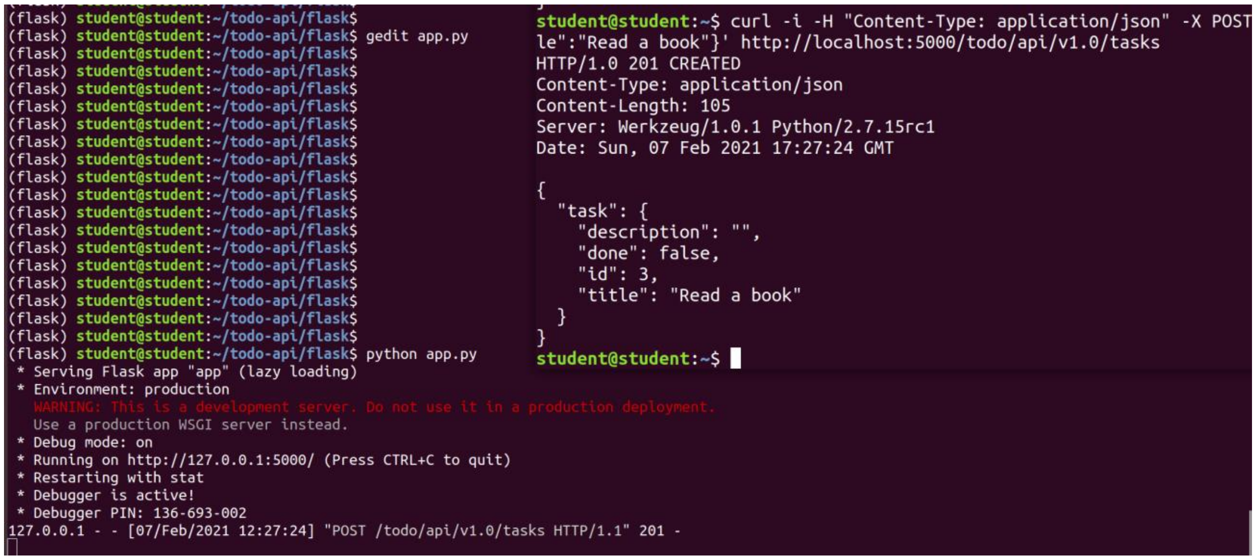
```
student@student:~$
student@student:~$
student@student:~$
student@student:~$
student@student:~$
student@student:~$
student@student:~$
student@student:~$
student@student:~$ curl -i http://localhost:5000/todo/api/v1.0/tasks/2
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 152
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 17:05:14 GMT

{
  "task": {
    "description": "Need to find a good Python tutorial on the web",
    "done": false,
    "id": 2,
    "title": "Learn Python"
  }
}
student@student:~$
```

Figure 4: Nominal GET request v2
The updated GET request to get the information from a single list with an id.



```
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 17:24:23 GMT

{
  "error": "Not found"
}
student@student:~$
ges/flask/app.py", line 1821, in handle user exception
```

Figure 5: error handling
 Improving the error not found exception for a GET command.

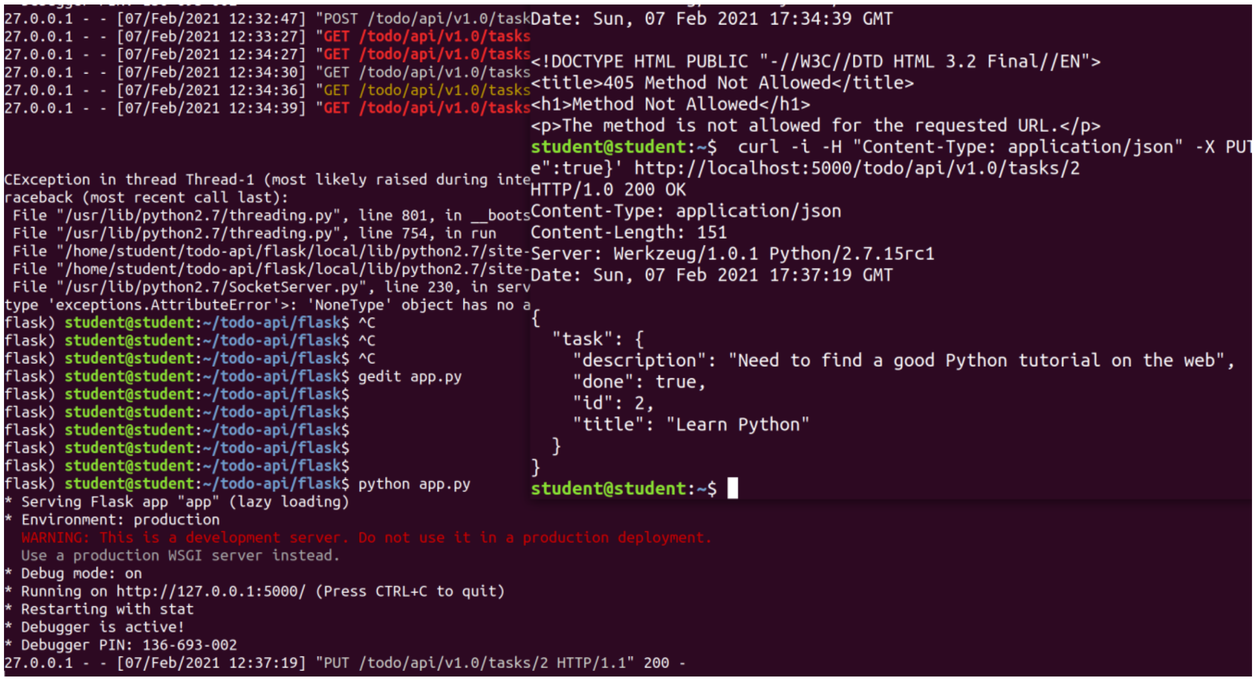


```
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ gedit app.py
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$
(flask) student@student:~/todo-api/flask$ 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: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-693-002
127.0.0.1 - - [07/Feb/2021 12:27:24] "POST /todo/api/v1.0/tasks HTTP/1.1" 201 -
```

```
student@student:~$ curl -i -H "Content-Type: application/json" -X POST
le":"Read a book"}' http://localhost:5000/todo/api/v1.0/tasks
HTTP/1.0 201 CREATED
Content-Type: application/json
Content-Length: 105
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 17:27:24 GMT

{
  "task": {
    "description": "",
    "done": false,
    "id": 3,
    "title": "Read a book"
  }
}
student@student:~$
```

Figure 6: Successful post request

Successfully implemented and working POST request to add information to the server list.

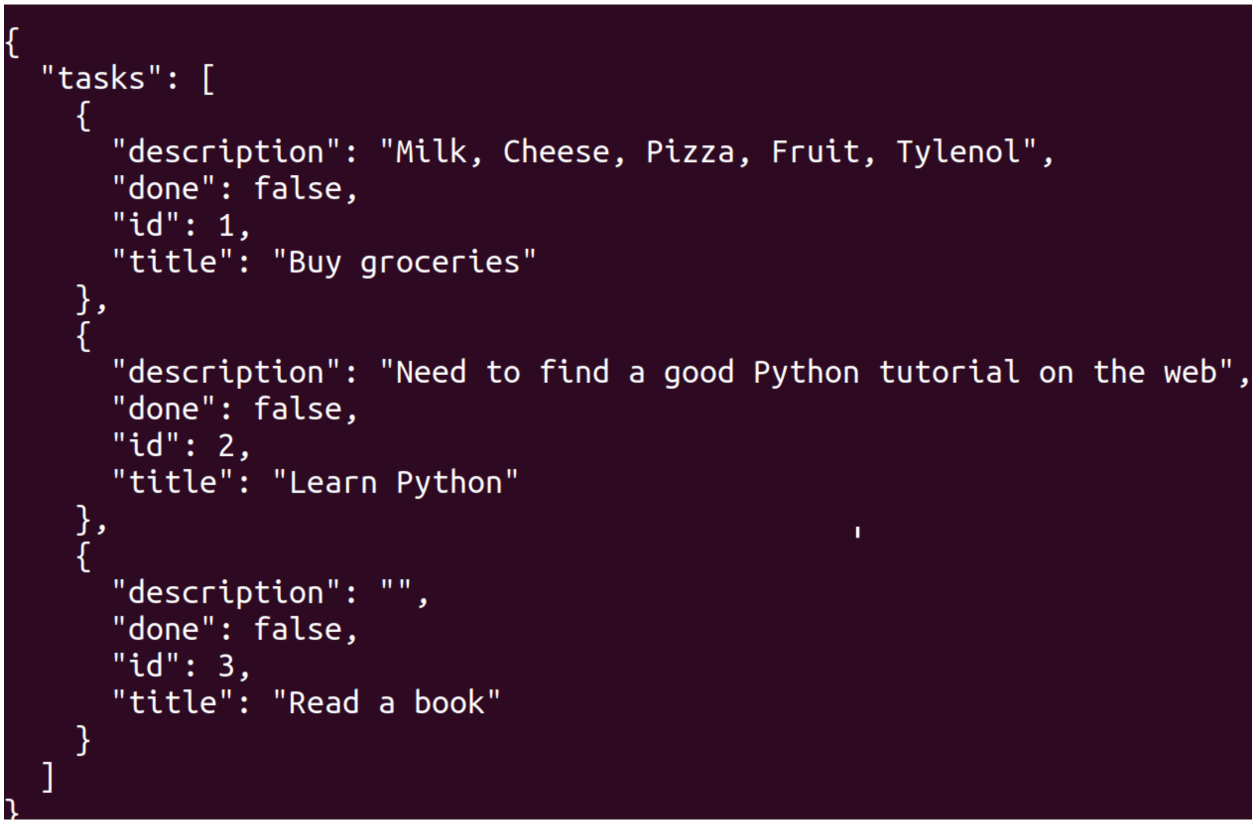


```
27.0.0.1 - - [07/Feb/2021 12:32:47] "POST /todo/api/v1.0/task
27.0.0.1 - - [07/Feb/2021 12:33:27] "GET /todo/api/v1.0/tasks
27.0.0.1 - - [07/Feb/2021 12:34:27] "GET /todo/api/v1.0/tasks
27.0.0.1 - - [07/Feb/2021 12:34:30] "GET /todo/api/v1.0/tasks
27.0.0.1 - - [07/Feb/2021 12:34:36] "GET /todo/api/v1.0/tasks
27.0.0.1 - - [07/Feb/2021 12:34:39] "GET /todo/api/v1.0/tasks


CException in thread Thread-1 (most likely raised during inte
raceback (most recent call last):
 File "/usr/lib/python2.7/threading.py", line 801, in __boots
 File "/usr/lib/python2.7/threading.py", line 754, in run
 File "/home/student/todo-api/flask/local/lib/python2.7/site-
 File "/home/student/todo-api/flask/local/lib/python2.7/site-
 File "/usr/lib/python2.7/SocketServer.py", line 230, in serv
type 'exceptions.AttributeError'>: 'NoneType' object has no a
flask) student@student:~/todo-api/flask$ ^C
flask) student@student:~/todo-api/flask$ ^C
flask) student@student:~/todo-api/flask$ ^C
flask) student@student:~/todo-api/flask$ gedit app.py
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$ 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: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 136-693-002
27.0.0.1 - - [07/Feb/2021 12:37:19] "PUT /todo/api/v1.0/tasks/2 HTTP/1.1" 200 -
```

```
Date: Sun, 07 Feb 2021 17:34:39 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
student@student:~$  curl -i -H "Content-Type: application/json" -X PUT
e":true}' http://localhost:5000/todo/api/v1.0/tasks/2
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 151
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 17:37:19 GMT

{
  "task": {
    "description": "Need to find a good Python tutorial on the web",
    "done": true,
    "id": 2,
    "title": "Learn Python"
  }
}
student@student:~$
```

Figure 7: example of PUT API

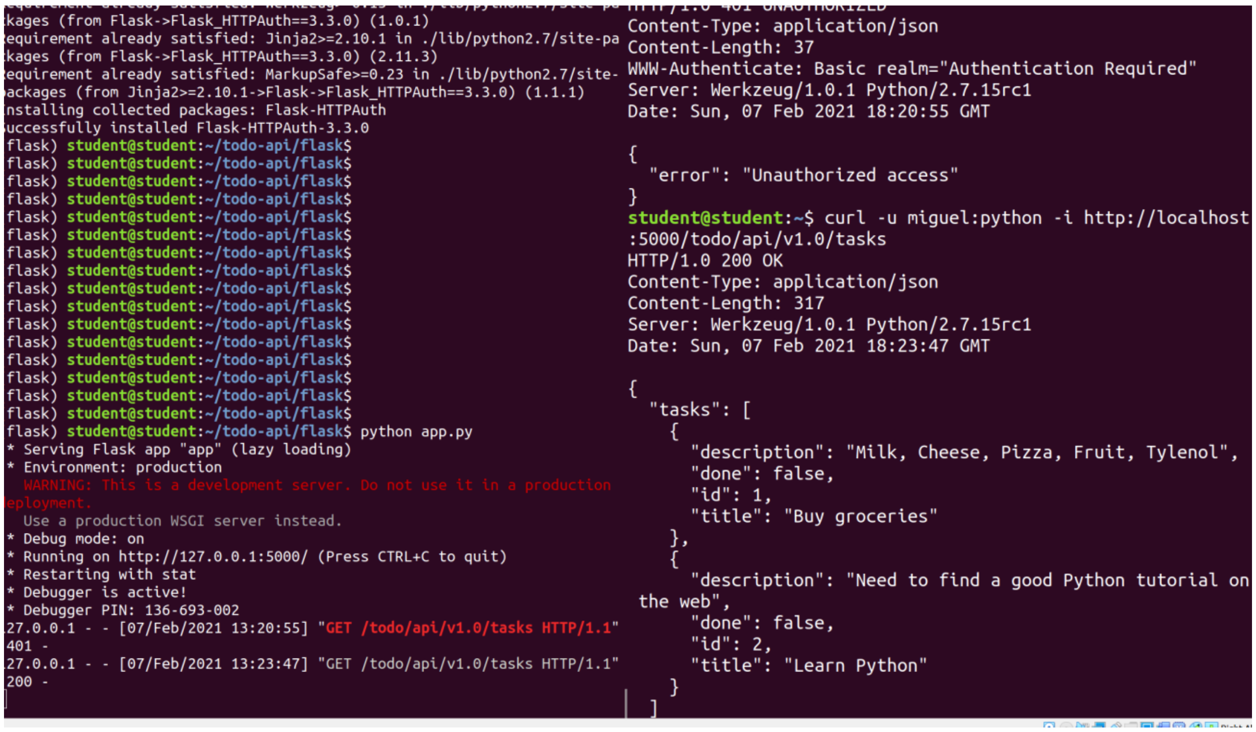
Implemented and working example of the PUT API on our web server.



```
{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    },
    {
      "description": "",
      "done": false,
      "id": 3,
      "title": "Read a book"
    }
  ]
}
```

Figure 8: Nominal GET API Request

An example of the GET request still working after adding more items to the list and modifying the program.

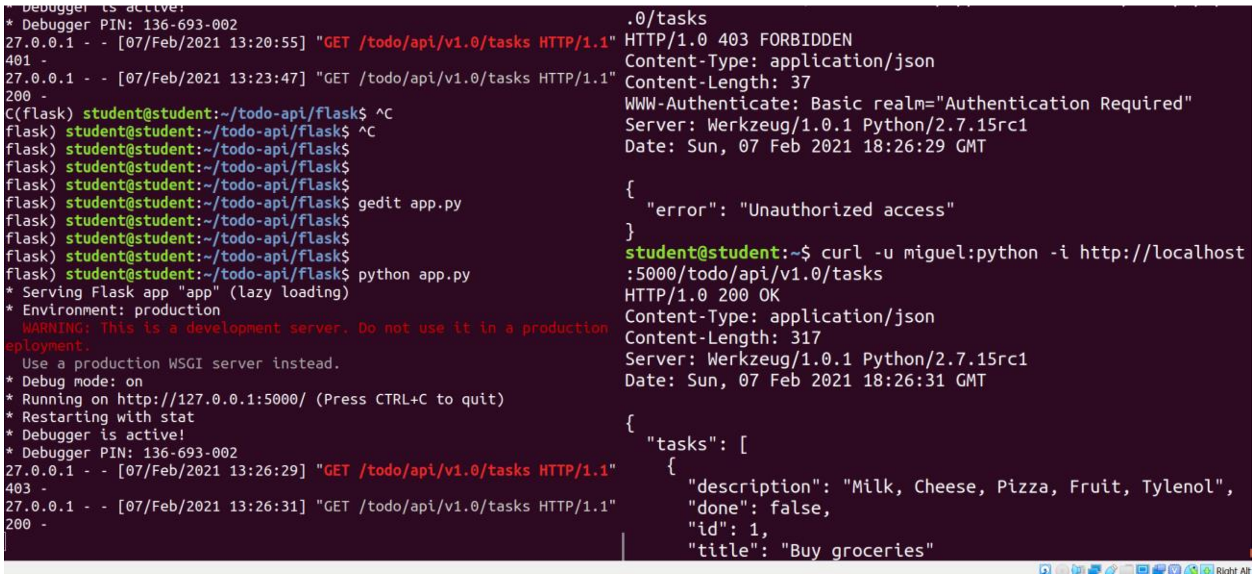


```
kages (from Flask->Flask_HTTPAuth==3.3.0) (1.0.1)
equirement already satisfied: Jinja2>=2.10.1 in ./lib/python2.7/site-pa
kages (from Flask->Flask_HTTPAuth==3.3.0) (2.11.3)
equirement already satisfied: MarkupSafe>=0.23 in ./lib/python2.7/site-
ackages (from Jinja2>=2.10.1->Flask->Flask_HTTPAuth==3.3.0) (1.1.1)
nstalling collected packages: Flask-HTTPAuth
uccessfully installed Flask-HTTPAuth-3.3.0
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$ python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production
eployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 136-693-002
27.0.0.1 - - [07/Feb/2021 13:20:55] "GET /todo/api/v1.0/tasks HTTP/1.1"
401 -
27.0.0.1 - - [07/Feb/2021 13:23:47] "GET /todo/api/v1.0/tasks HTTP/1.1"
200 -
```

```
Content-Type: application/json
Content-Length: 37
WWW-Authenticate: Basic realm="Authentication Required"
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 18:20:55 GMT

{
  "error": "Unauthorized access"
}
student@student:~$ curl -u miguel:python -i http://localhost
:5000/todo/api/v1.0/tasks
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 317
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 18:23:47 GMT

{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on
 the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    }
  ]
```

Figure 9: Authorization denied and accepted.

This figure shows a failed GET request without credentials and a successful curl command with credentials.



```
* Debugger PIN: 136-693-002
27.0.0.1 - - [07/Feb/2021 13:20:55] "GET /todo/api/v1.0/tasks HTTP/1.1"
401 -
27.0.0.1 - - [07/Feb/2021 13:23:47] "GET /todo/api/v1.0/tasks HTTP/1.1"
200 -
C(flask) student@student:~/todo-api/flask$ ^C
flask) student@student:~/todo-api/flask$ ^C
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$ gedit app.py
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$
flask) student@student:~/todo-api/flask$ python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production
eployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 136-693-002
27.0.0.1 - - [07/Feb/2021 13:26:29] "GET /todo/api/v1.0/tasks HTTP/1.1"
403 -
27.0.0.1 - - [07/Feb/2021 13:26:31] "GET /todo/api/v1.0/tasks HTTP/1.1"
200 -
```

```
.0/tasks
HTTP/1.0 403 FORBIDDEN
Content-Type: application/json
Content-Length: 37
WWW-Authenticate: Basic realm="Authentication Required"
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 18:26:29 GMT

{
  "error": "Unauthorized access"
}
student@student:~$ curl -u miguel:python -i http://localhost
:5000/todo/api/v1.0/tasks
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 317
Server: Werkzeug/1.0.1 Python/2.7.15rc1
Date: Sun, 07 Feb 2021 18:26:31 GMT

{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
```

Figure 10: changing the error to 403 for unauthorized access.

This figure shows a failed GET request without credentials throwing a 403 error and a successful curl command with credentials.

2. Using your security mindset, write a brief paragraph about some of the security concerns applying surrounding our endpoint and your suggestions for how to remedy them.

Web APIs have many significant attacks that can be used against them, to name a few: injection, XSS, DDoS, MitM, credential stuffing, among many others. Data sanitization and validation is by far one of the most secure things we can do as engineers to prevent malicious code to be extracted or injected. Ensuring our backend is not only validating the data its receiving, but also ensuring the input is properly coded and sanitized to ensure sensitive data cannot piggyback on the response can also be crucial. Rate limiting and limiting the payload size of the GET and POST request can also help mitigate against DDoS attacks since the flooding of messages can be put into a queue. MitM attacks can be prevented by ensuring the data is encrypted in transit, encrypting the data at rest is also a safeguard that is recommended by many security experts. Using safeguards mentioned above will also prevent credential stuffing and can also help against some brute force attacks. At the end we can see that authentication and authorization can help prevent unwanted users from using the endpoints properly, to ensure the data is truly safeguarded we can practice the principal of least privilege, and adopt strong protocols and practices to prevent unwanted adversaries from getting in. Another easy thing we can do as engineers is ensure we are using HTTPs with the TLS/SSL protocols to ensure everything is encrypted between transactions.

3. Also, you will notice that our code must convert between string and JSON data using the 'jsonify' function. What does the 'jsonify' function do and why is it necessary in Python?

The jsonify function is included in the Flask module to serialize data into the JSON mime type format. At a high level in Python the jsonify constructor is the same as a python dictionary constructor and it will return a dictionary of attribute name/value pairs and will eventually return a python object. The json encoder then adds some metadata to the python object along with other data transforming to return a json encodable representation of an object which then allows the python object to serialize with jsonify.

References

https://www.f5.com/labs/articles/education/securing-apis--10-best-practices-for-keeping-your-data-and-infra [1]

https://www.fullstackpython.com/flask-json-jsonify-examples.html#:~:text=jsonify%20is%20a%20function%20in,with%20the%20application%2Fjson%20mimetype. [2]

https://pyphi.readthedocs.io/en/latest/api/jsonify.html [3]