# Hands-on Lab - Creating a Swagger documentation for REST API

Skills Network

Estimated Time: 45 minutes

In this lab, you will understand how to create a Swagger documentation for your REST APIs.

## Learning Objectives:

After completing this exercise, you should be able to perform the following tasks:

- Use the Swagger Editor to create Swagger documentation for REST API
- Use SwaggerUI to access the REST API endpoints of an application
- Generate code with the Swagger documentation

## Pre-requisites

- You must be familiar with Docker applications and commands
- You must have a good understanding of REST API.
- Knowledge of Python is highly recommended

### Task 1 - Getting your application started

1. Open a terminal window by using the top menu in the IDE: **Terminal > New Terminal**, if you don't have one open already.

2. In the terminal, clone the repository which has the Swagger documentation and the REST API code ready by pasting the following command. The repository that you clone has code that will run a REST API application which can be used to organize tasks.

```
1. 1
1. git clone https://github.com/ibm-developer-skills-network/jmgdo-microservices.git
```

Copied!

3. Change the working directory to **jmgdo-microservices/swagger_example** by running the following command.

```
1. 1
1. cd jmgdo-microservices/swagger_example
```

Copied!

4. Run the following commands to install the required packages.

```
1. 1
1. python3 -m pip install flask_cors
```

Copied!

5. Now start the application which serves the REST API on port number 5000.

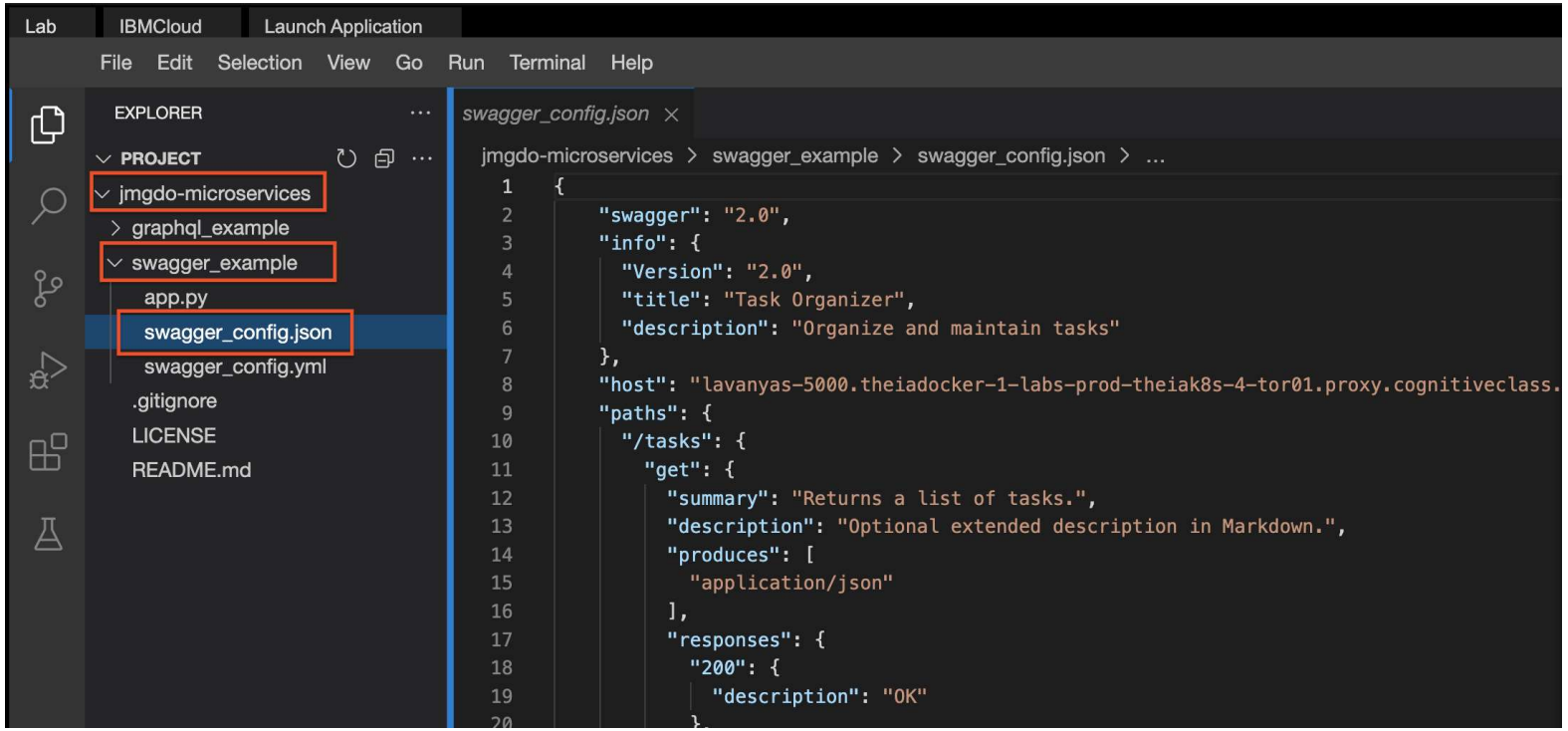
```
1. 1
1. python3 app.py
```

Copied!

6. From the top menu, choose **Launch Application** and enter the port number as 5000. This will open a new browser page, which accesses the application you just ran.

7. Copy the url on the address bar.

8. From the file menu, go to jmgdo-microservices/swagger_example/swagger_config.json to view the file on the file editor.




9. In the file editor, paste the application URL that you copied where it says `**<Your application URL\>**` without the protocol (https://) and save the file.

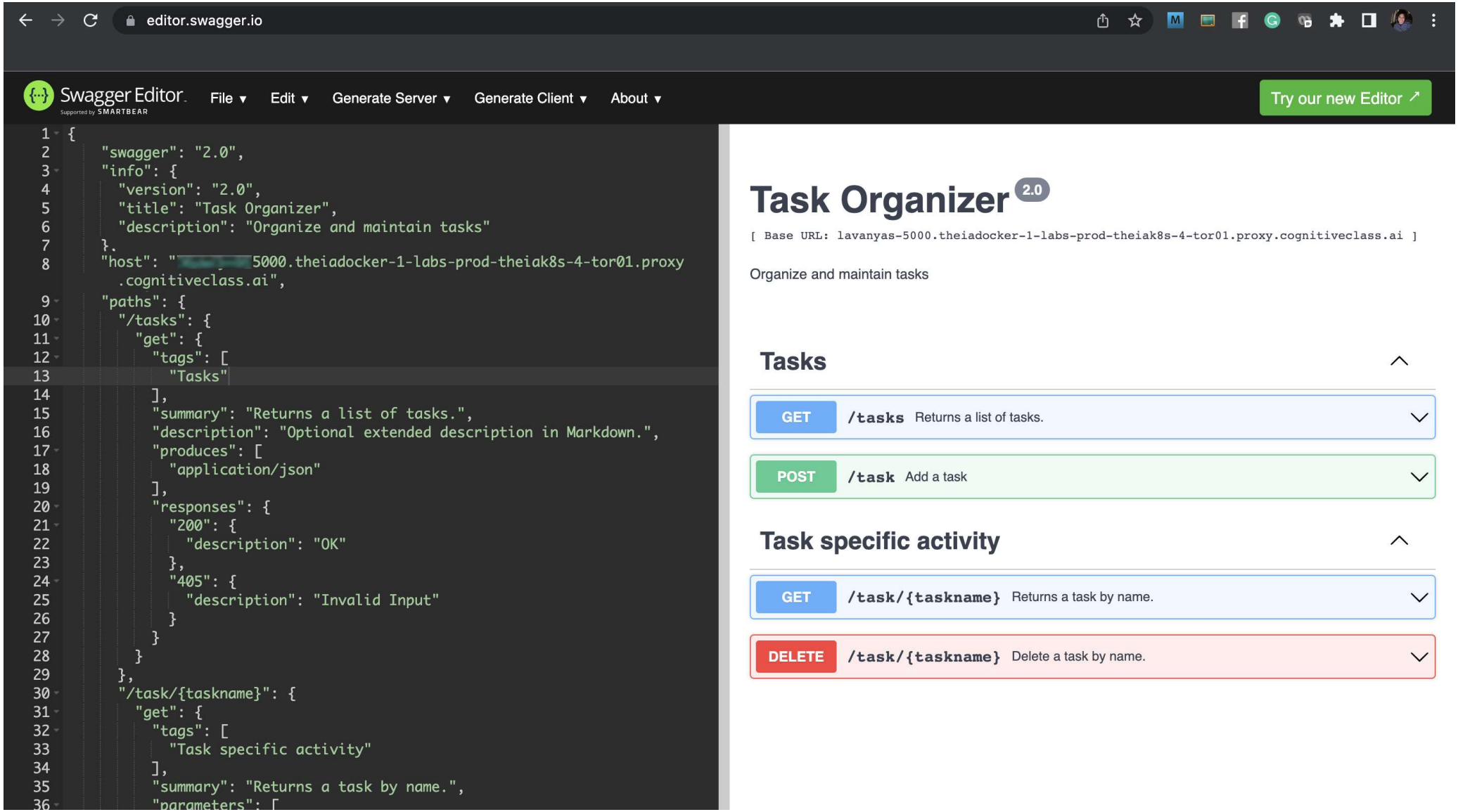
10. Copy the entire content of the file **swagger_config.json**. You will need this copied content to generate SwaggerUI.

11. Click on this link https://editor.swagger.io/ to go to the Swagger Editor.

12. From the File menu, click on Clear Editor to clear the content of the Swagger Editor.

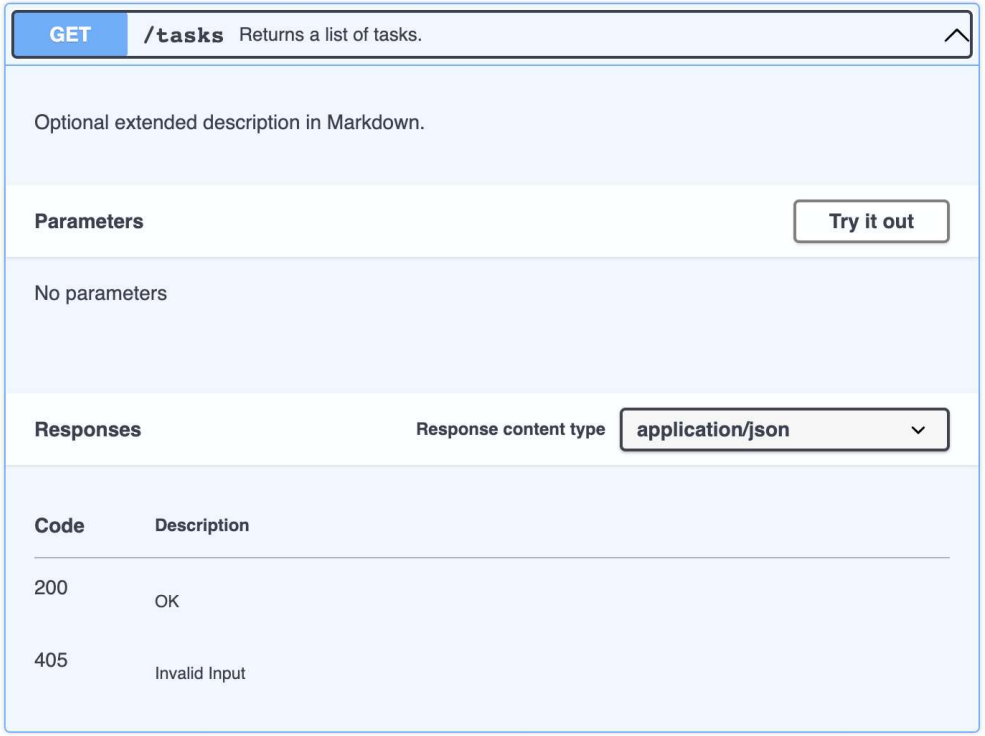
13. Paste the content you copied from **swagger_config.json** on the left side. You will get a prompt which says `Would you like to convert your JSON into YAML?`. Press `Cancel` to paste the content.

14. You will see that the UI is automatically populated on the right.

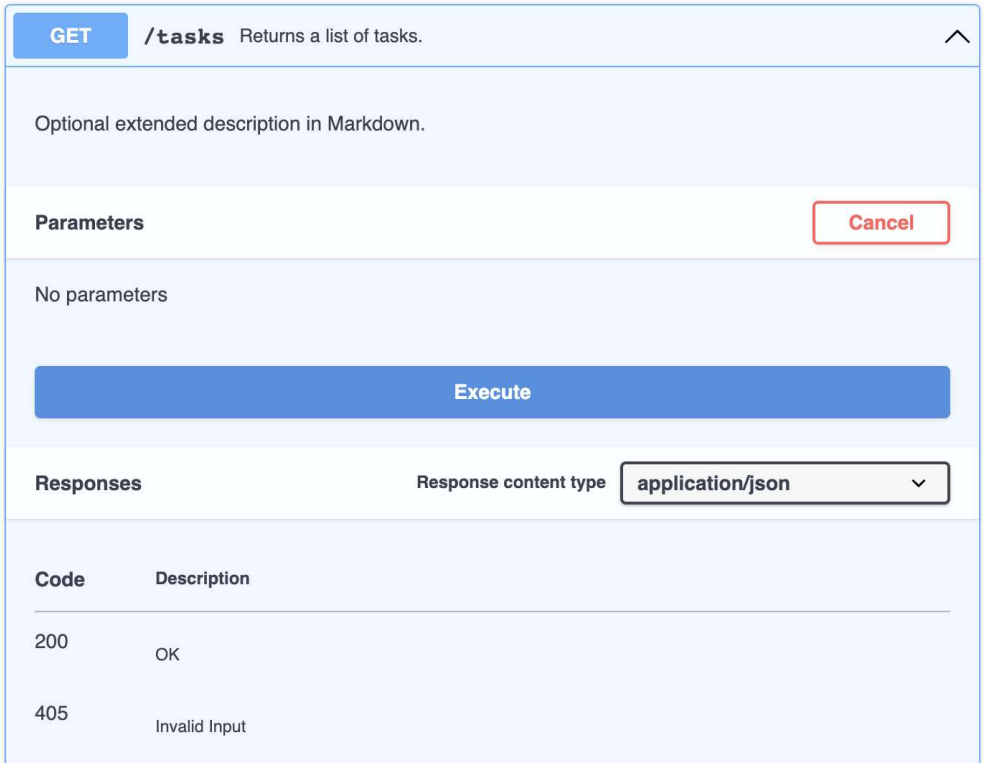



15. Now you can test each of the endpoints. Four tasks have been already added for you, when the application was started. Click on the down arrow next to **GET /tasks**.


16. Click on **Try it out**. This will allow you to try your REST API endpoint.




17. Click on **Execute** to invoke a call to your REST API. This is a **GET** request which does not take any parameters. It returns the task as an **application/json**.




18. You can scroll down to view the output of the API call.

**Curl**

```
curl -X 'GET' \
  'https://lavanyas-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/tasks' \
  -H 'accept: application/json'
```

**Request URL**

```
https://lavanyas-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/tasks
```

**Server response**

| Code | Details |
|---|---|
| 200 | **Response body** |

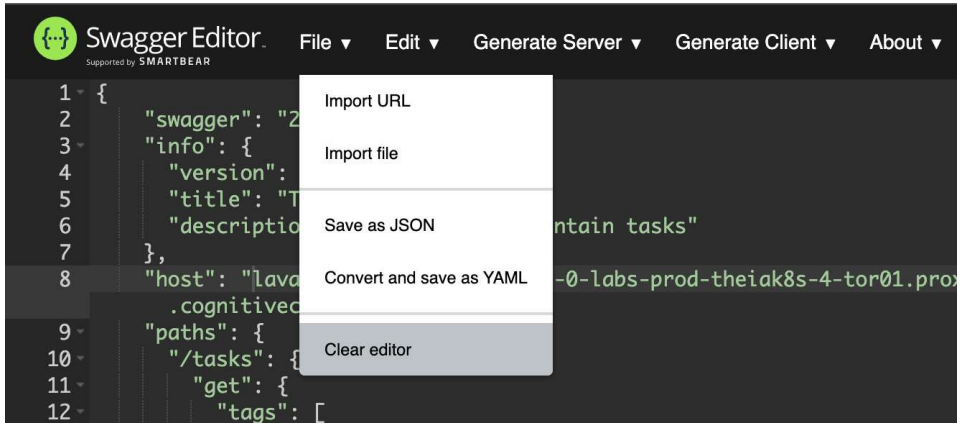
```
{
  "tasks": [
    {
      "description": "Do the laundry this weekend",
      "name": "Laundry"
    },
    {
      "description": "Finish assignment by Friday",
      "name": "Assignment"
    },
    {
      "description": "Call family Sunday morning",
      "name": "Call family"
    },
    {
      "description": "Pay the electricity and water bill",
      "name": "Pay bills"
    }
  ]
}
```

Download

19. Try to do the following:

- Add a task
- Retrieve the tasks to see if your task is added to the list
- Get the details on one task
- Delete a task and check the list to verify that it is deleted.

20. From the **File** menu, click on **Clear Editor** to clear the content of the Swagger Editor.




**Task 2 - Creating Swagger Documentation and Generating Server code**

1. Now you will create a REST API with Swagger documentation. To start with, let's define your application.

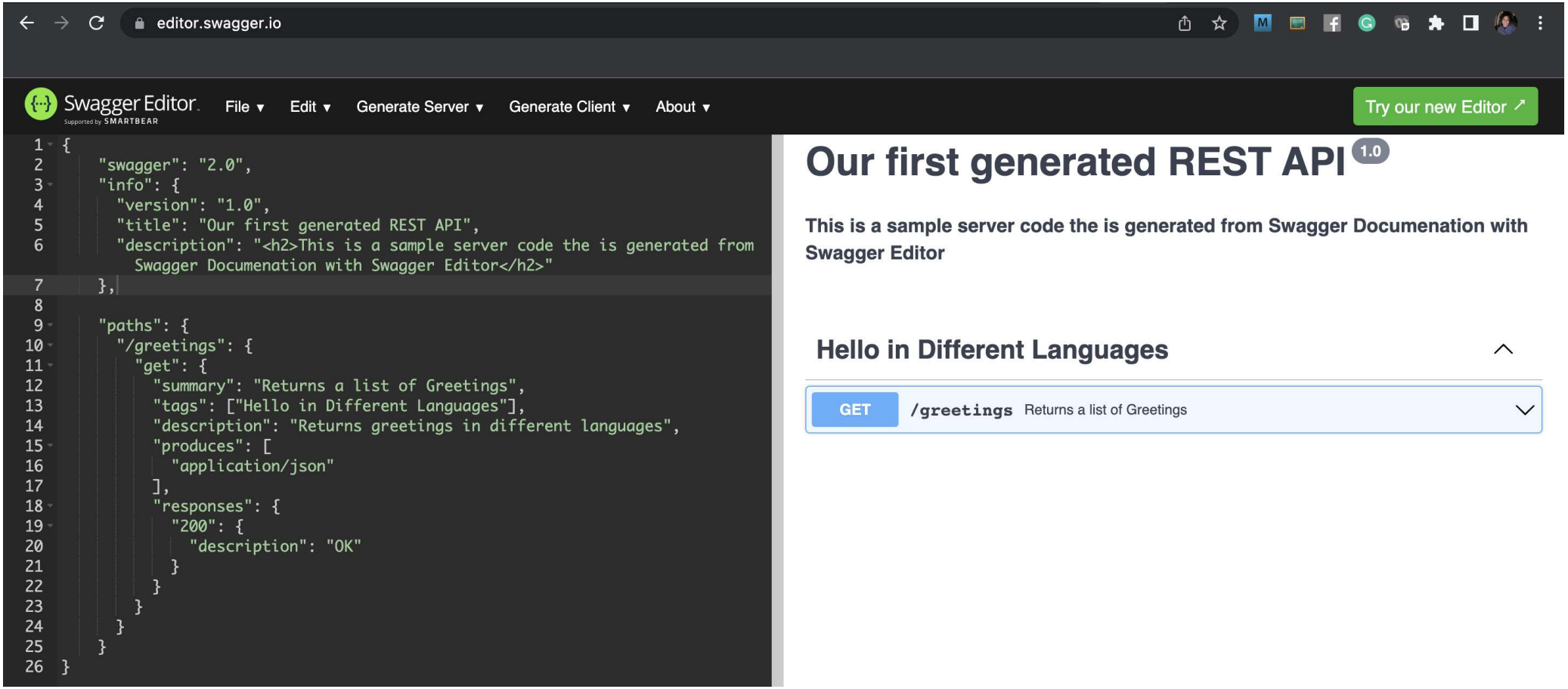
- It will adhere to Swagger 2.0 version
- This is the first version of the application
- It will have one endpoint /greetings, which returns the list of greetings as a JSON object.

2. Copy and paste the following JSON in the Swagger Editor. You will get a prompt which says `Would you like to convert your JSON into YAML?`. Press `Cancel` to paste the content.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

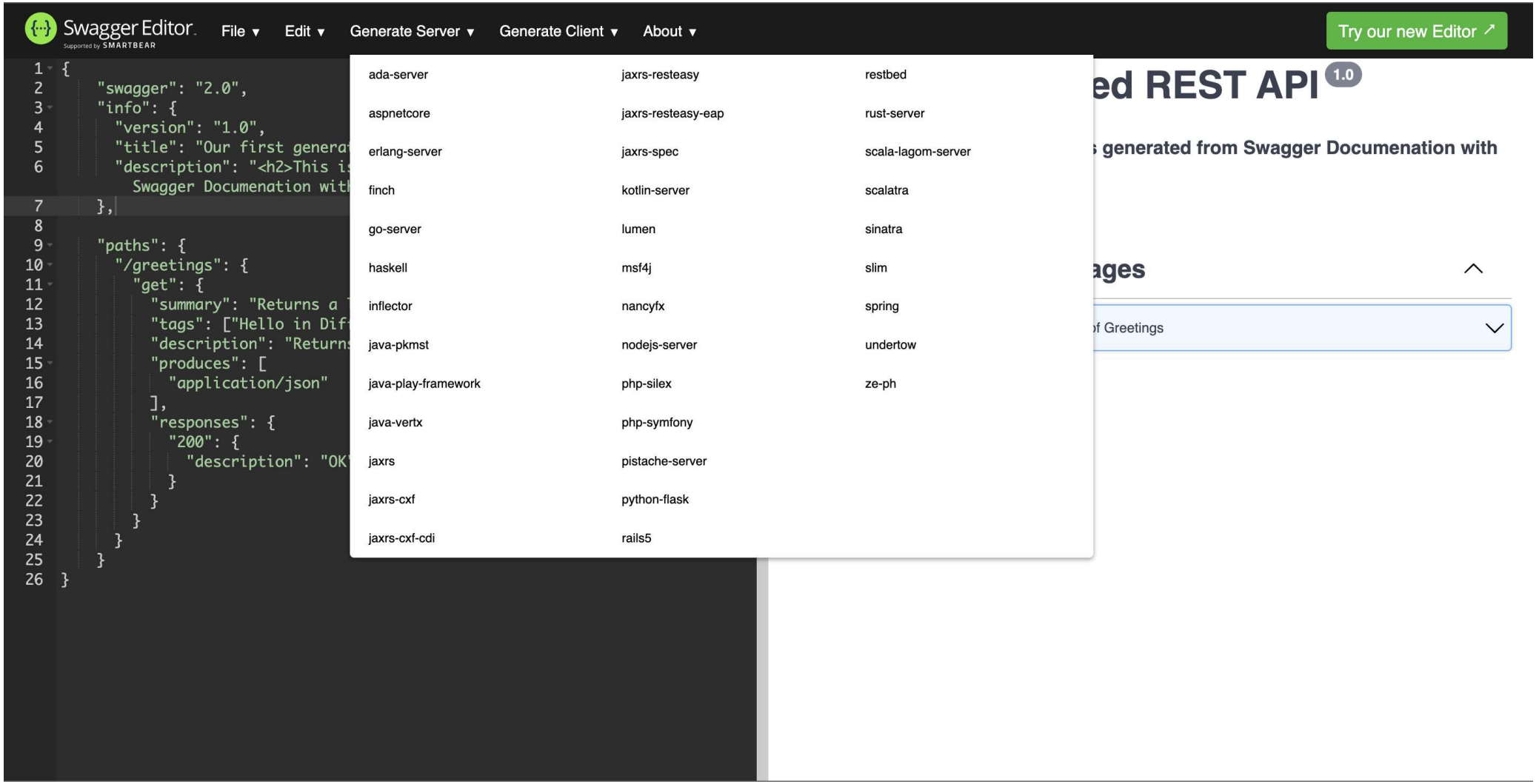
```
{
    "swagger": "2.0",
    "info": {
      "version": "1.0",
      "title": "Our first generated REST API",
      "description": "<h2>This is a sample server code the is generated from Swagger Documenation with Swagger Editor</h2>"
    },

    "paths": {
      "/greetings": {
        "get": {
          "summary": "Returns a list of Greetings",
          "tags": ["Hello in Different Languages"],
          "description": "Returns greetings in different languages",
          "produces": [
            "application/json"
          ],
          "responses": {
            "200": {
              "description": "OK"
            }
          }
        }
      }
    }
}
```

Copied!

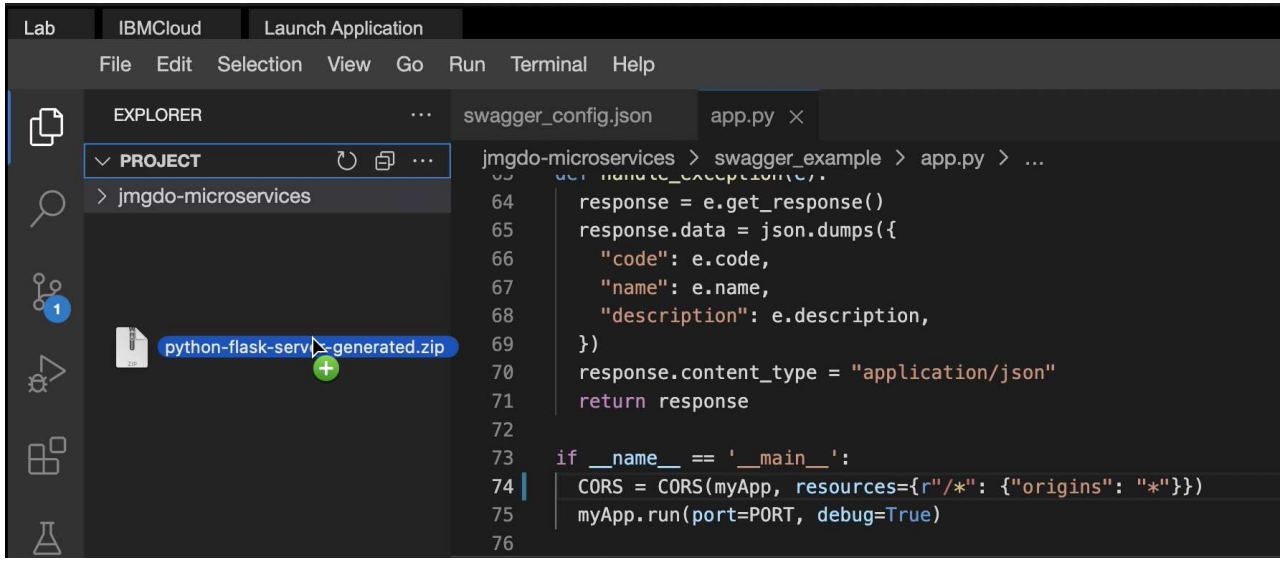



You will see the Swagger UI automatically appearing on the right. You cannot test it yet as your application is not defined and running yet.

3. From the menu on top, click on **Generate Server** and select **python-flask**. This will automatically generate the server code as a zip file named **python-flask-server-generated.zip**. Download the zip file to your system.




4. In your lab envrionment, click on the **PROJECT** folder and drag and drop the zip file there.




5. On the terminal go to the **/home/project** directory.

```
1
cd /home/project
```

Copied!

6. Check to see if the zip file that you just dragged and dropped, exists.

```
1
ls python-flask-server-generated.zip
```

Copied!

7. Unzip the contents of the zip file into a directory named **python-flask-server-generated** by running the following command.

```
1
unzip python-flask-server-generated.zip -d python-flask-server-generated/
```

Copied!

8. Change to the **python-flask-server** folder inside the folder you just extracted the zip file into.

```
1
cd python-flask-server-generated/python-flask-server
```

Copied!

9. The entire server setup along with endpoint is done for you already. Let's build the server code.

```
1
docker build . -t mynewserver
```

Copied!

This takes a while. If the build runs successfully you will have a new container with tag mynewserver.

10. Run the docker application now by running the following command. The server generated code automatically is configured to run on port 8080.

```
1
docker run -dp 8080:8080 mynewserver
```

Copied!

You will get a hex code that indicates the application has started.

11. To confirm that the service is running and your REST API works, execute the following command.

```
1
curl localhost:8080/greetings
```

Copied!

```
theia@theiadocker-lavanyas:/home/project/python-flask-server-generated/python-flask-server$ docker run -dp 8080:8080 mynewserver
7be538de67e2e81d1435c80cccd3e0c11fe03821e68517ed10e43f347ff89a37
theia@theiadocker-lavanyas:/home/project/python-flask-server-generated/python-flask-server$ curl localhost:8080/greetings
"do some magic!"
theia@theiadocker-lavanyas:/home/project/python-flask-server-generated/python-flask-server$
```

What you see in the output is what you have to do. **do some magic!**

▼ Click here for hint in case you encounter an error
First inspect the Docker images you've built by executing the following command:

```
1
```

```
docker images
```

Copied!

This command will provide a list of Docker images, with their respective IMAGE IDs.

Then delete the Docker image of **mynewserver** by using the following command:

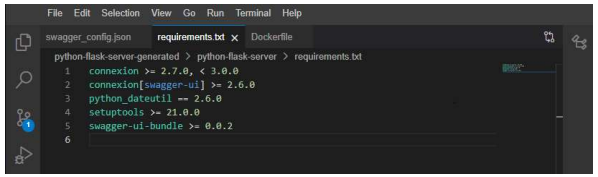
```
1
```

```
docker rmi -f <IMAGE ID>
```

Copied!

Make sure to replace "<IMAGE ID>" with the actual IMAGE ID of **mynewserver** you got in the previous step.

Open the "requirements.txt" file which present inside the unzipped folder named "python-flask-server-generated" and update the connexion version to "connexion >= 2.7.0, < 3.0.0" as shown in the screenshot below.




Once the image is deleted, proceed with rebuilding the docker image and allowing some time before running the docker application.

12. Now you should stop the server. For this you need the docker container id. Run the following command and copy the container id.

```
1
```

```
docker ps | grep mynewserver
```

Copied!

```
$ docker ps | grep mynewserver
7be538de67e2   mynewserver   "python3 -m swagger_…"   44 minutes ago   Up 44 minutes   0.0.0.0:8080->8080/tcp, :::808
0->8080/tcp   priceless_morse
```

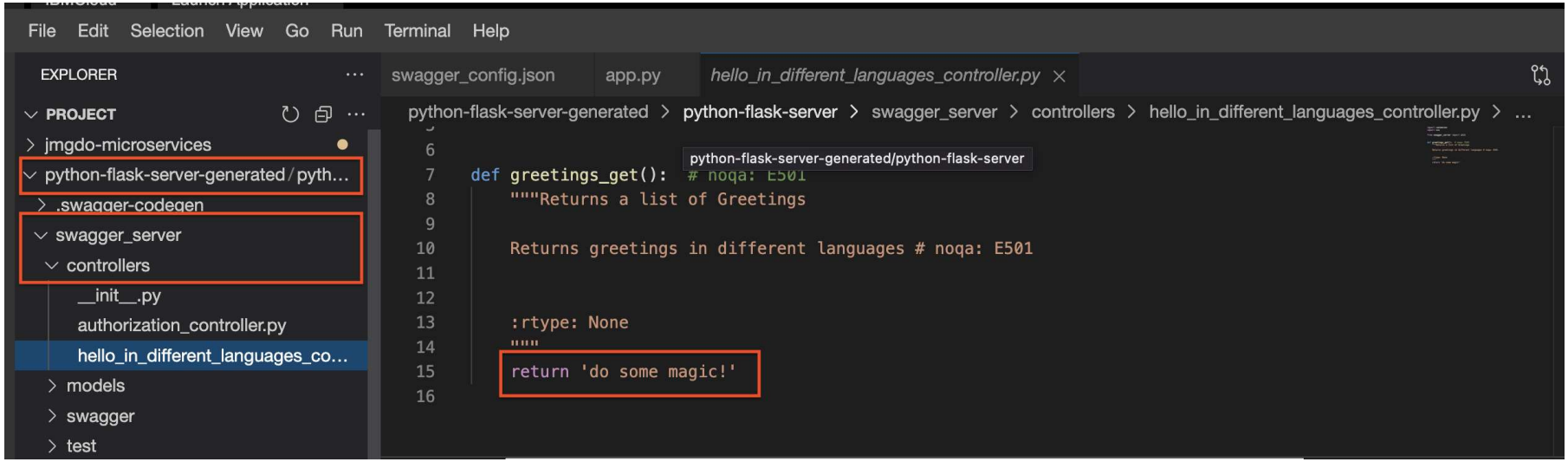
13. To stop the container you need to kill the instance referring to the container id you copied in the last step.

```
1
```

```
docker kill <container_id>
```

Copied!

14. In the file explorer go to, python-flask-server-generated/python-flask-server/swagger_server/controllers/hello_in_different_languages_controller.py. This is where you need to implement your actual response for the REST API.




15. Replace `return 'do some magic!'` with the following code. As this is the python code and the indentation in Python is very important, make sure you check the indentations error.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
15. 15
16. 16

 1. hellos = {
 2.   "English": "hello",
 3.   "Hindi": "namastey",
 4.   "Spanish": "hola",
 5.   "French": "bonjour",
 6.   "German": "guten tag",
 7.   "Italian": "salve",
 8.   "Chinese": "nin hào",
 9.   "Portuguese": "olá",
10.   "Arabic": "asalaam alaikum",
11.   "Japanese": "konnichiwa",
12.   "Korean": "anyoung haseyo",
13.   "Russian": "Zdravstvuyte"
14. }
15.
16. return hellos
```

Copied!

16. Build the docker container again to ensure the changed code is taken in.

```
1. 1

1. docker build . -t mynewserver
```

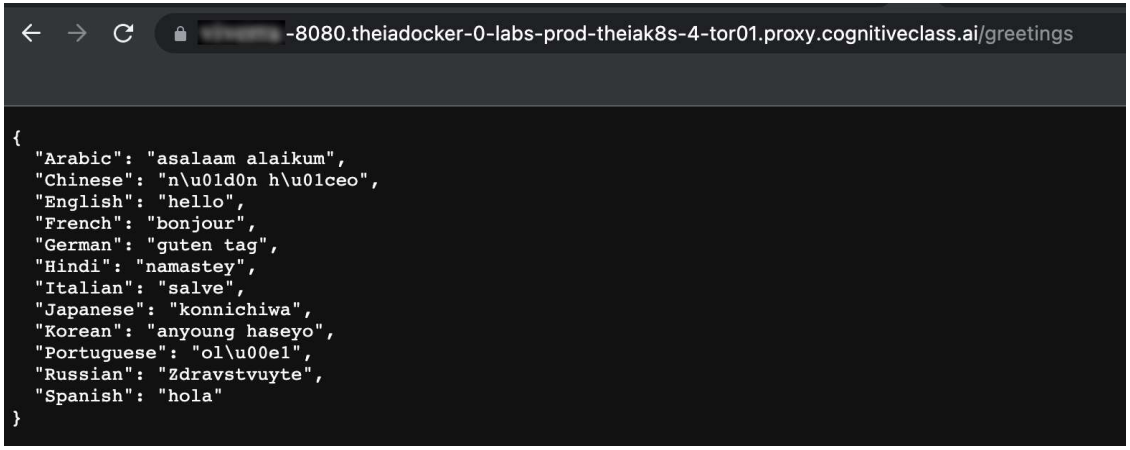
Copied!

17. Run the container now with the following command. You may notice that you are using **-p** instead of **-dp**. This is to ensure the server is not running in **discreet** mode and you are able to see errors if any.

```
1. 1

1. docker run -p 8080:8080 mynewserver
```

Copied!

18. Now click on **Launch Application** and enter the port number 8080. This will open a browser window. Append the path **/greetings** to the URL. You should see the greetings in the page.




Congratulations! You have successfully completed the task.

## Tutorial details

**Author:** Lavanaya T S

**Contributors:** Pallavi Rai

**Change Log**

| Date | Version | Changed by | Change Description |
|---|---|---|---|
| 2022-08-26 | 1.0 | Lavanaya T S | Initial version created |
| 2023-01-18 | 1.1 | K Sundararajan | Instructions updated based on testing |
| 2023-11-08 | 1.2 | Rajashree Patil | Added hints for docker error in Task 2 |