

Table of Contents

aplicatie.....	1
testare.....	2
fisierul Dockerfile pentru utilizare python din docker.....	3
Perplexity.....	5

aplicatie

Folosire chatGPT pt generare cod python folosind

- vscode,
- chatgpt
- browser brave

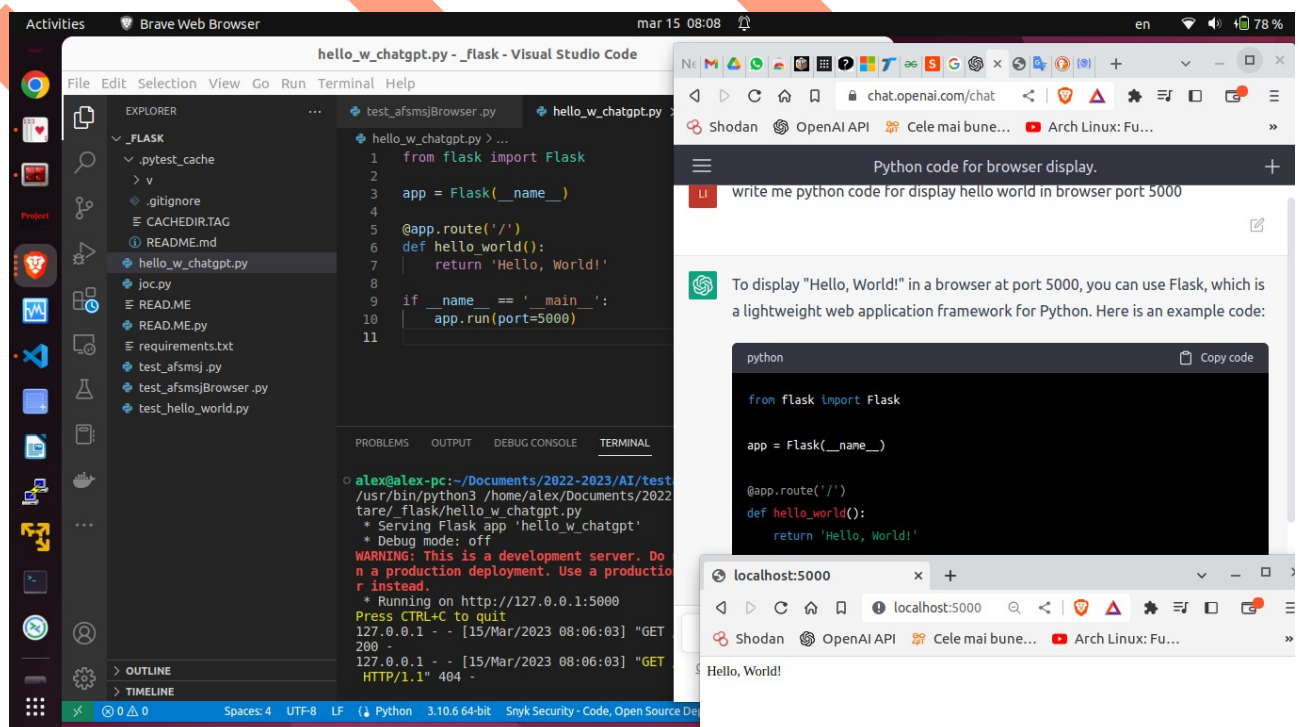
fisierul hello_chatgpt.py

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')  
def hello_world():  
    return 'Hello, World!'
```

```
if __name__ == '__main__':  
    app.run(port=5000)
```



testare

Fisierul python

```
import unittest
from hello import app

class TestHelloWorld(unittest.TestCase):
    def setUp(self):
        app.testing = True
        self.client = app.test_client()

    def test_hello_world(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data, b'Hello, World!')

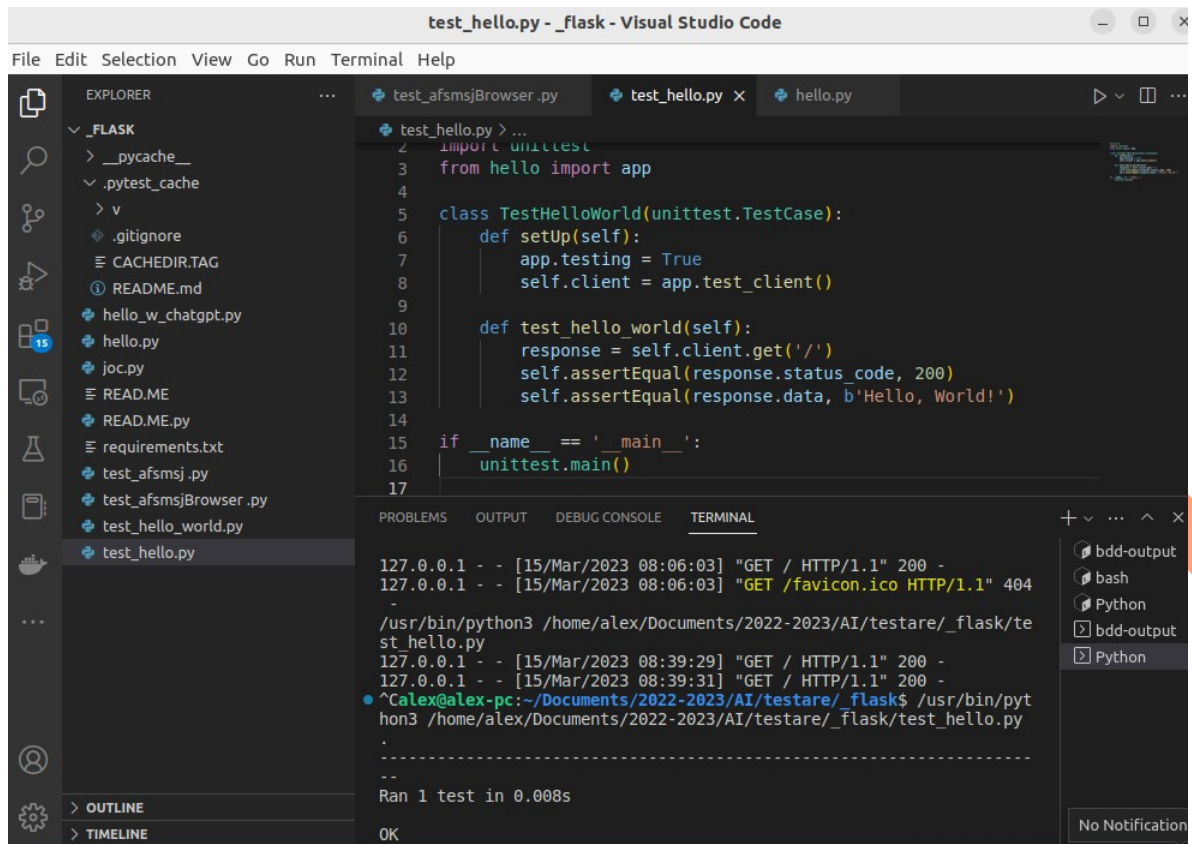
if __name__ == '__main__':
    unittest.main()
```

In this code, we import the `unittest` module and the `app` object from the "hello.py" file. We then define a test class `TestHelloWorld` that inherits from `unittest.TestCase`.

In the `setUp()` method, we set the testing attribute of the `app` object to `True` and create a test client that we can use to make requests to the application.

The `test_hello_world()` method sends a GET request to the root URL ("/") using the test client and checks that the response status code is 200 (indicating a successful request) and that the response data is the expected "Hello, World!" string.

Finally, we use the `unittest.main()` function to run the test suite. When you run this code, you should see that the test passes and outputs "OK" to the console.



```
test_hello.py > ...
2 import unittest
3 from hello import app
4
5 class TestHelloWorld(unittest.TestCase):
6     def setUp(self):
7         app.testing = True
8         self.client = app.test_client()
9
10    def test_hello_world(self):
11        response = self.client.get('/')
12        self.assertEqual(response.status_code, 200)
13        self.assertEqual(response.data, b'Hello, World!')
14
15    if __name__ == '__main__':
16        unittest.main()
17
```

```
127.0.0.1 - - [15/Mar/2023 08:06:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Mar/2023 08:06:03] "GET /favicon.ico HTTP/1.1" 404 -
/usr/bin/python3 /home/alex/Documents/2022-2023/AI/testare/_flask/test_hello.py
127.0.0.1 - - [15/Mar/2023 08:39:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Mar/2023 08:39:31] "GET / HTTP/1.1" 200 -
^Calex@alex-pc:~/Documents/2022-2023/AI/testare/_flask$ /usr/bin/python3 /home/alex/Documents/2022-2023/AI/testare/_flask/test_hello.py
:
-----
Ran 1 test in 0.008s
OK
```

fișierul Dockerfile pentru utilizare python din docker

Desigur, iată un exemplu de fișier Dockerfile pentru utilizarea Python din Docker:

```
pythonCopy code
# Selecția imaginii de bază
FROM python:3.9-slim-buster

# Setarea directorului de lucru
WORKDIR /app

# Copierea fișierelor necesare în container
COPY requirements.txt .
COPY app.py .

# Instalarea dependențelor Python
RUN pip install --no-cache-dir -r requirements.txt

# Exponarea portului 8000 pentru accesul la aplicație
EXPOSE 8000

# Rularea aplicației
CMD ["python", "app.py"]
```

Explicații:

1. Selecția imaginii de bază: se alege o imagine de bază care să conțină Python-ul necesar. În exemplul de mai sus se folosește versiunea 3.9 a limbajului, împachetată într-o imagine minimală (slim) cu sistemul de operare Debian Buster.
2. Setarea directorului de lucru: se specifică că în interiorul container-ului directorul de lucru (working directory) va fi /app.
3. Copierea fișierelor necesare în container: se copiază fișierele necesare pentru rularea aplicației (în cazul de față, requirements.txt și app.py) din directorul local în directorul /app din interiorul container-ului.
4. Instalarea dependențelor Python: se instalează dependențele Python necesare pentru rularea aplicației. Acestea sunt specificate în fișierul requirements.txt. Opțiunile --no-cache-dir sunt folosite pentru a evita stocarea în cache a dependențelor.
5. Exponerea portului 8000 pentru accesul la aplicație: se specifică că aplicația va fi disponibilă prin intermediul portului 8000.
6. Rularea aplicației: se specifică comanda care va fi rulată la pornirea container-ului. În cazul de față, se pornește aplicația Python din fișierul app.py.

Perplexity

fișierul Dockerfile pentru utilizare python din docker

PERPLEXITY

Here is an example Dockerfile for using Python in Docker:

Use an official Python runtime as a parent image

FROM python:3.9-slim-buster

Set the working directory to /app

WORKDIR /app

Copy the current directory contents into the container at /app

COPY . /app

Install any needed packages specified in requirements.txt

RUN pip install --trusted-host pypi.python.org -r requirements.txt

Make port 80 available to the world outside this container

EXPOSE 80

Define environment variable

ENV NAME World

Run app.py when the container launches

CMD ["python", "app.py"]

This Dockerfile starts with an official Python runtime image from Docker Hub[1]. It sets the working directory to /app and copies all files from the current directory into the container. It then installs any required packages specified in requirements.txt using pip. The EXPOSE instruction informs Docker that the container will listen on port 80 at runtime. The ENV instruction sets an environment variable, and finally, it runs app.py when the container launches using CMD.

To build a Docker image from this Dockerfile, navigate to its directory and run:

docker build -t my-python-app .

This command builds a new Docker image with tag my-python-app. The dot (.) at the end of the command specifies that it should use the current directory as context.

Once built, you can run a new container from this image using:

docker run -p 4000:80 my-python-app

This command runs a new container from your image with port mapping of host's port 4000 to container's port 80. You can access your application by visiting <http://localhost:4000> in your web browser.

Note that this is just an example, and you may need to modify it based on your specific needs[2][3][4][5].

5 SOURCES

1

[docker.com favicon](https://docs.docker.com/get-docker/)

[docker](https://docs.docker.com/get-docker/)

2

[stackoverflow.com favicon](https://stackoverflow.com/questions/40176183/python-dockerfile)

stackoverflow

3

visualstudio.com favicon

visualstudio

4

medium.com favicon

medium

5

collabnix.com favicon

collabnix

