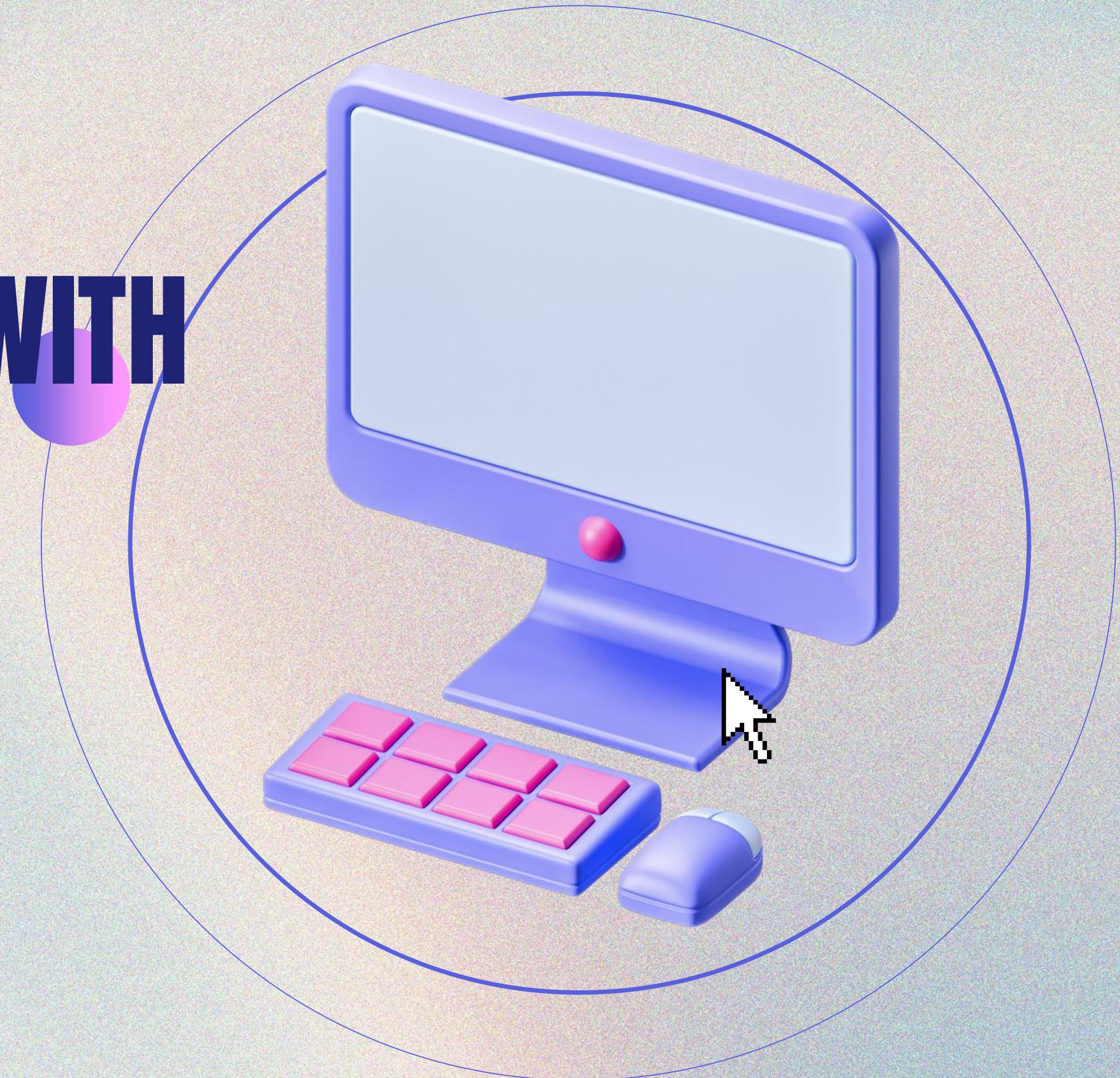


DOCKER COMPOSE WITH MULTI-CONTAINER APPLICATION

GROUP 5



INTRODUCTION

In this project, we learned how to use Docker to run a simple website using Flask and store data in Redis. We created containers for both Flask and Redis so they can work together easily. The project helped us understand how to run multiple services, link them, and test that they work using Docker Compose in a clear and practical way.

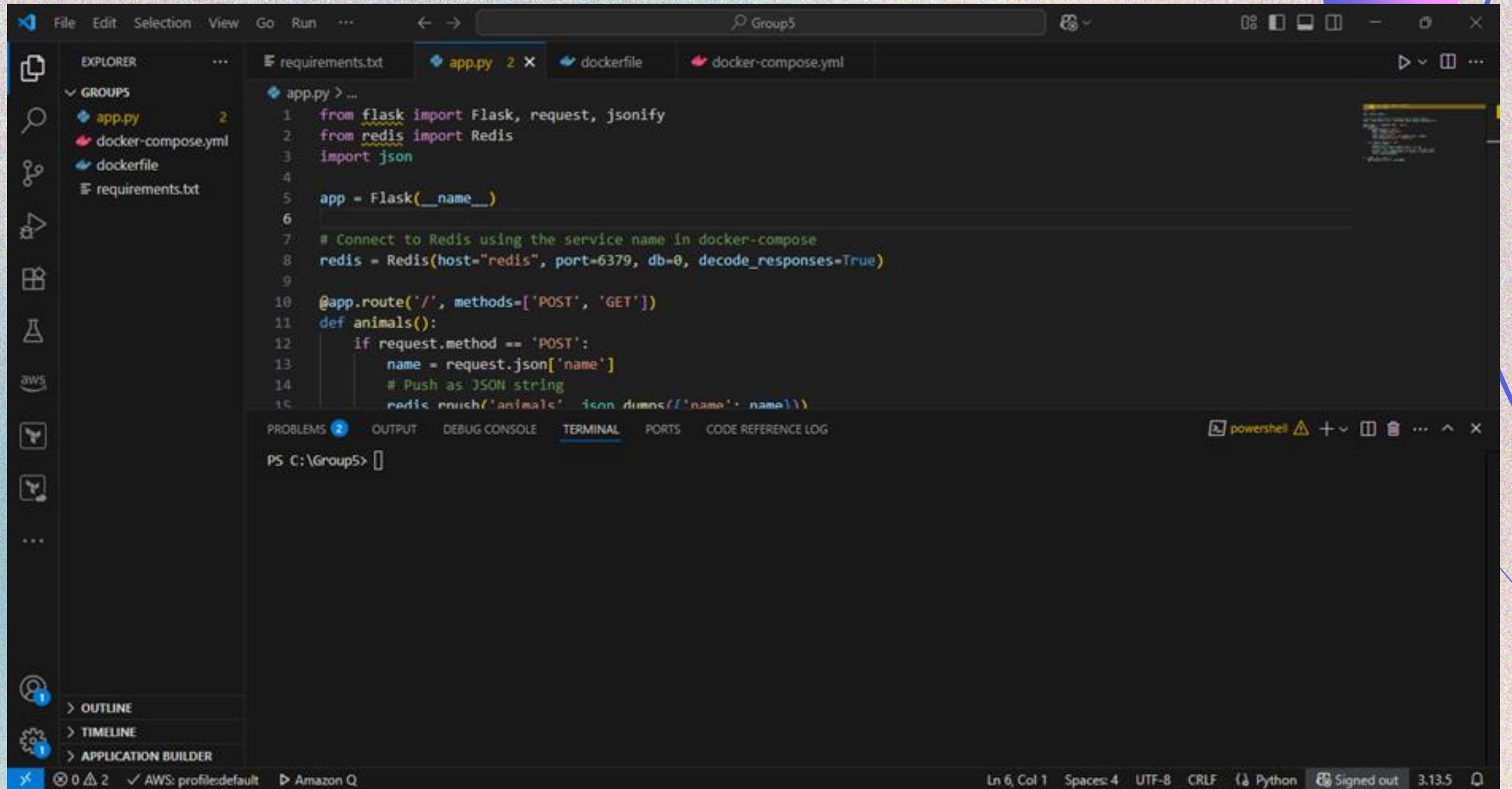
PROJECT OBJECTIVES

The objectives of the project was to;

- Containerize a Python Flask app using Docker.
- Use Redis as a database for data storage and retrieval.
- Link multiple services using Docker Compose.
- Test connectivity between services.

TASK 1: CREATE A PROJECT DIRECTORY

- Created a directory called Group5 to hold our files namely: **app.py**, **requirements.txt**, **dockerfile** and **docker-compose.yml**.



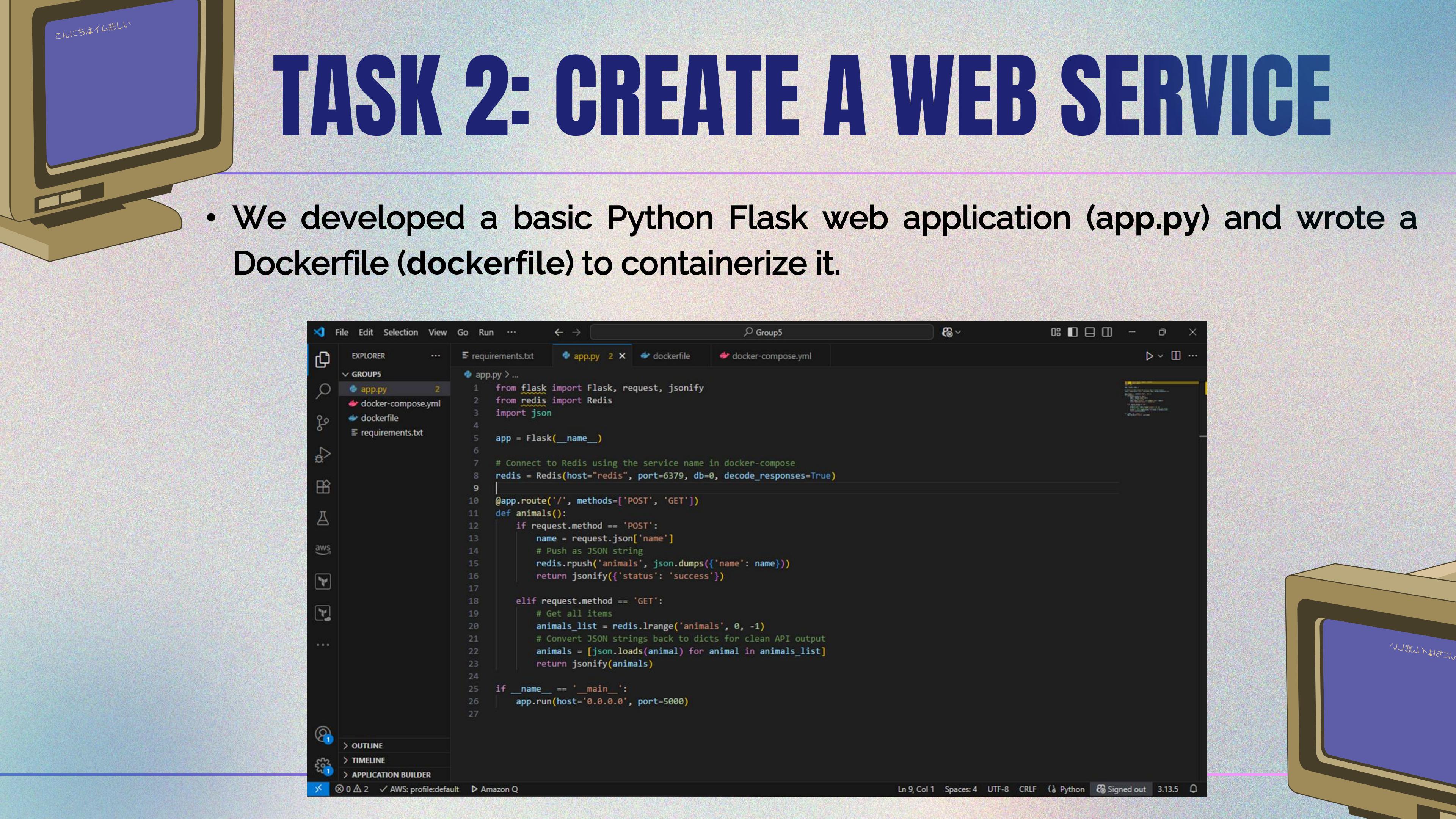
```
File Edit Selection View Go Run ... ← → Group5 08 □ □ □ - □ X  
EXPLORER ... requirements.txt app.py 2 X dockerfile docker-compose.yml  
GROUPS app.py 2 docker-compose.yml dockerfile requirements.txt  
app.py > ...  
1 from flask import Flask, request, jsonify  
2 from redis import Redis  
3 import json  
4  
5 app = Flask(__name__)  
6  
7 # Connect to Redis using the service name in docker-compose  
8 redis = Redis(host="redis", port=6379, db=0, decode_responses=True)  
9  
10 @app.route('/', methods=['POST', 'GET'])  
11 def animals():  
12     if request.method == 'POST':  
13         name = request.json['name']  
14         # Push as JSON string  
15         redis.rpush('animals', json.dumps({'name': name}))  
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG  
PS C:\Group5> [powershell] +v □ ... ^ x  
...  
OUTLINE  
TIMELINE  
APPLICATION BUILDER  
X 0 △ 2 ✓ AWS: profile=default D Amazon Q Ln 6, Col 1 Spaces: 4 UTF-8 CRLF (Python) Signed out 3.13.5
```



こんにちわイム悲しい

TASK 2: CREATE A WEB SERVICE

- We developed a basic Python Flask web application (`app.py`) and wrote a Dockerfile (`dockerfile`) to containerize it.



A screenshot of a dark-themed code editor showing a Python Flask application. The code uses Redis to store animal names and returns them as JSON. It includes routes for POSTing new names and GETting all names.

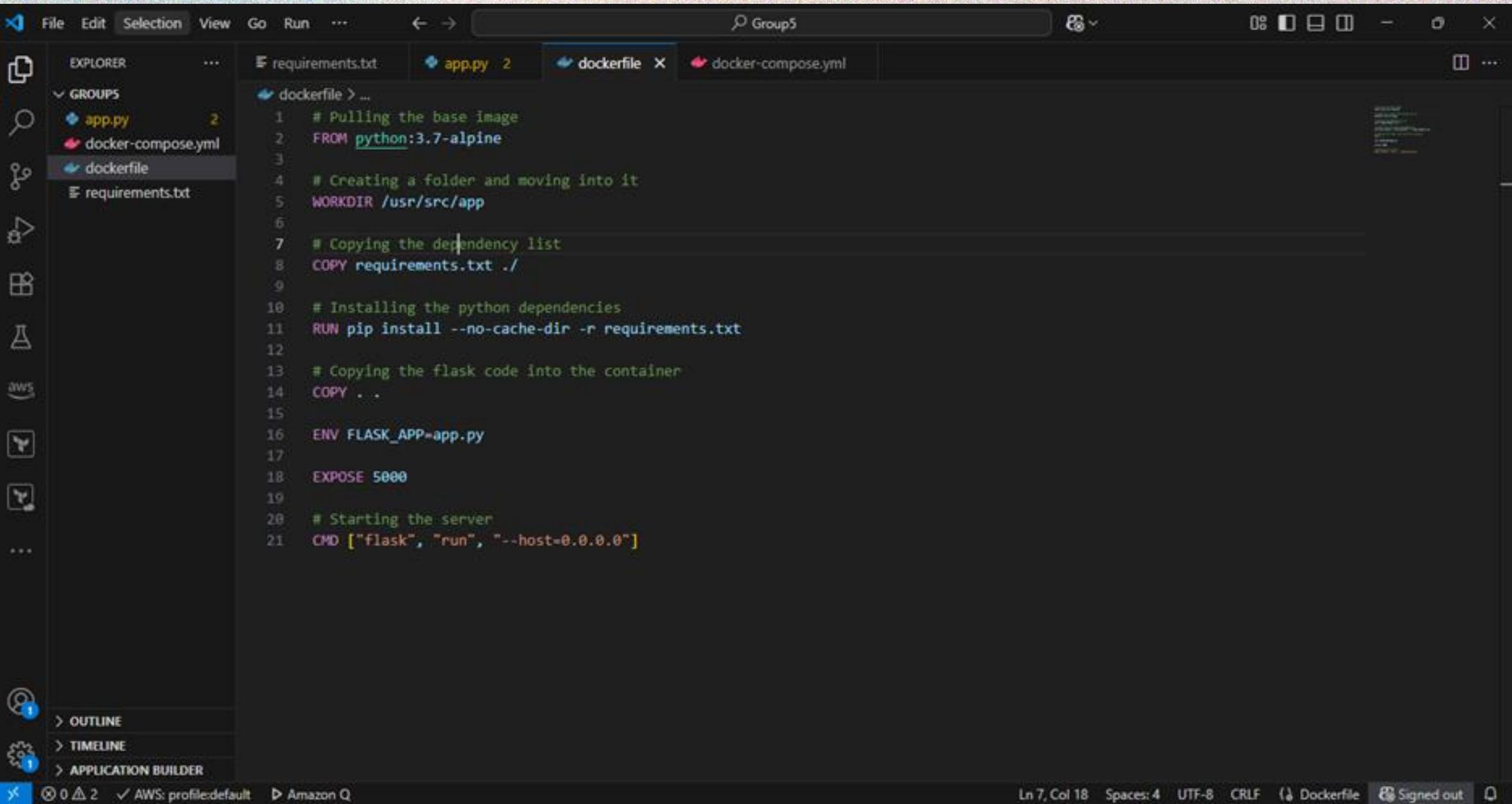
```
File Edit Selection View Go Run ... Group5
EXPLORER ... requirements.txt app.py 2 dockerfile docker-compose.yml
GROUPS
app.py 2
docker-compose.yml
dockerfile
requirements.txt
app.py > ...
1 from flask import Flask, request, jsonify
2 from redis import Redis
3 import json
4
5 app = Flask(__name__)
6
7 # Connect to Redis using the service name in docker-compose
8 redis = Redis(host="redis", port=6379, db=0, decode_responses=True)
9
10 @app.route('/', methods=['POST', 'GET'])
11 def animals():
12     if request.method == 'POST':
13         name = request.json['name']
14         # Push as JSON string
15         redis.rpush('animals', json.dumps({'name': name}))
16         return jsonify({'status': 'success'})
17
18     elif request.method == 'GET':
19         # Get all items
20         animals_list = redis.lrange('animals', 0, -1)
21         # Convert JSON strings back to dicts for clean API output
22         animals = [json.loads(animal) for animal in animals_list]
23         return jsonify(animals)
24
25 if __name__ == '__main__':
26     app.run(host='0.0.0.0', port=5000)
```

Bottom status bar: Ln 9, Col 1 Spaces: 4 UTF-8 CRLF Python Signed out 3.13.5

こんにちわイム悲しい

TASK 2: CREATE A WEB SERVICE

- We developed a basic Python Flask web application (`app.py`) and wrote a Dockerfile (`dockerfile`) to containerize it.



The screenshot shows a dark-themed code editor interface with several files open in tabs at the top: `requirements.txt`, `app.py` (with a count of 2), `dockerfile` (selected), and `docker-compose.yml`. The `EXPLORER` sidebar on the left shows a folder structure with `GROUPS` containing `app.py`, `docker-compose.yml`, and `dockerfile`, along with `requirements.txt`. The `dockerfile` tab contains the following Dockerfile content:

```
# Pulling the base image
FROM python:3.7-alpine

# Creating a folder and moving into it
WORKDIR /usr/src/app

# Copying the dependency list
COPY requirements.txt .

# Installing the python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copying the flask code into the container
COPY .

ENV FLASK_APP=app.py

EXPOSE 5000

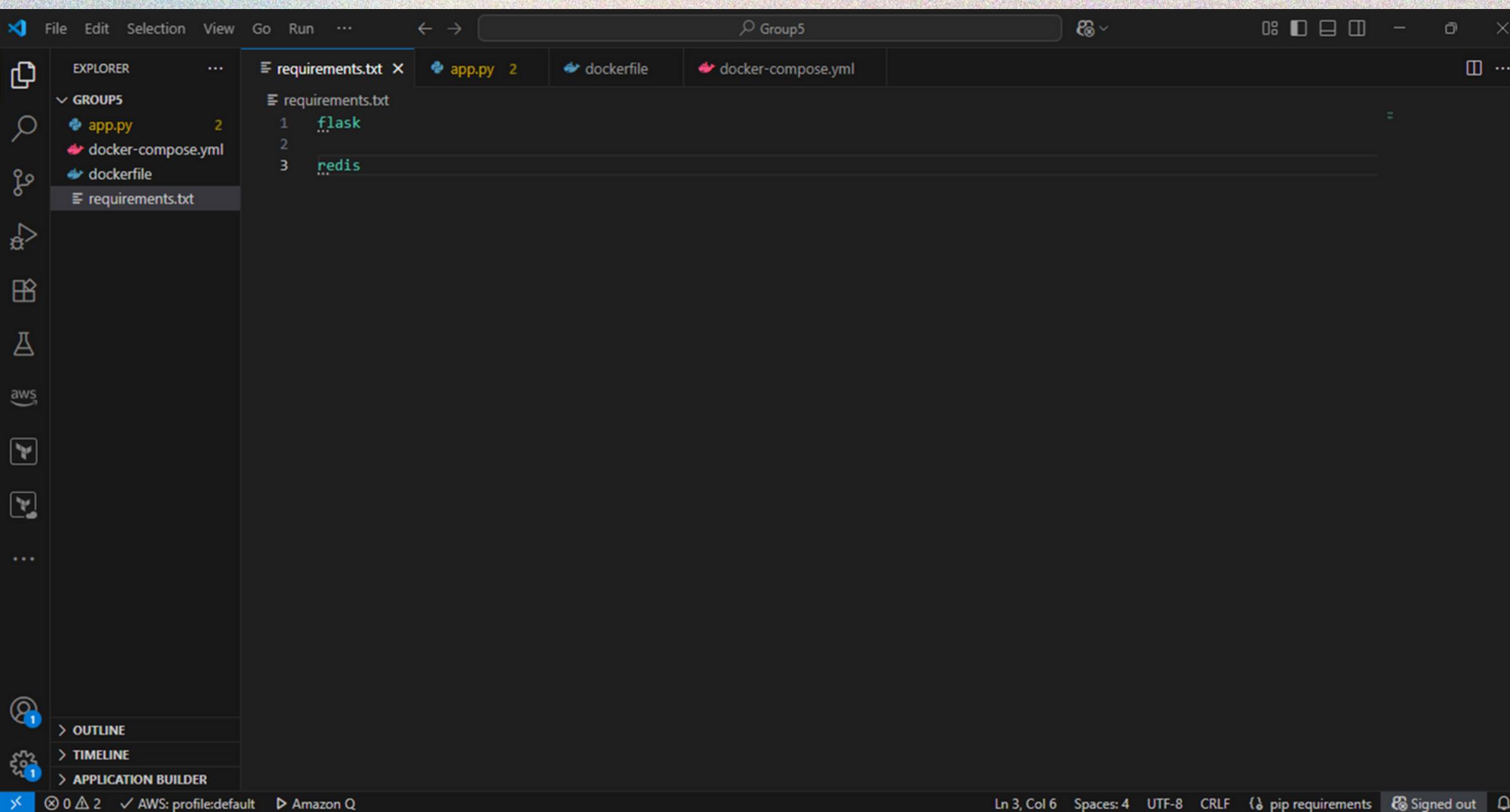
# Starting the server
CMD ["flask", "run", "--host=0.0.0.0"]
```

At the bottom of the editor, status bar items include: Ln 7, Col 18, Spaces: 4, UTF-8, CRLF, Dockerfile, Signed out, and a file icon.

こんにちわイム悲しい

TASK 2: CREATE A WEB SERVICE

- We created **requirements.txt** to list needed packages.



The screenshot shows a dark-themed code editor interface with several tabs at the top: File, Edit, Selection, View, Go, Run, ..., requirements.txt, app.py (2), dockerfile, and docker-compose.yml. The requirements.txt tab is active, displaying the following content:

```
flask
redis
```

The left sidebar features an Explorer view with a GROUPS section containing app.py (2), docker-compose.yml, dockerfile, and requirements.txt. Below this are various icons for AWS services like Lambda, API Gateway, and CloudWatch. The bottom status bar includes information such as AWS profile:default, Amazon Q, Ln 3, Col 6, Spaces: 4, UTF-8, CRLF, pip requirements, Signed out, and a file count of 0.

八月悲しい

TASK 3: SET UP REDIS AS A DATABASE

- We used the official Redis image with no additional configuration.



```
14 ▷ Run Service  
15 redis:  
15   image: "redis:4.0.11-alpine"
```

TASK 4: LINK SERVICES IN DOCKER COMPOSE

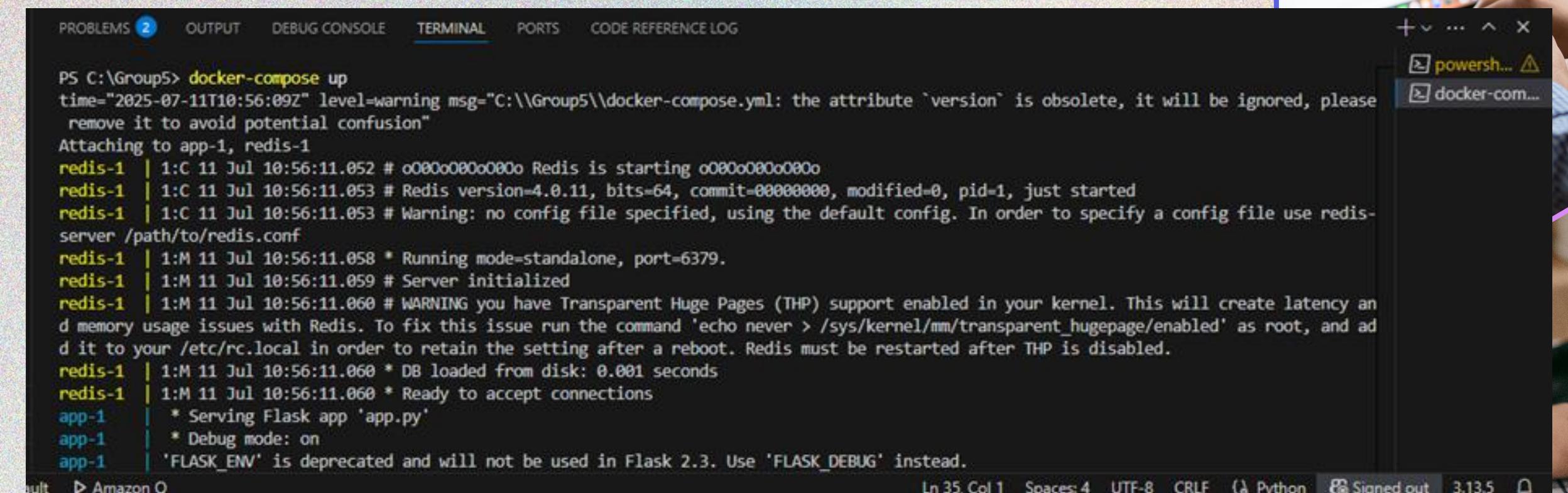


- We created **docker-compose.yml** to connect Flask and Redis.

```
version: "3.8"
services:
  app:
    build: .
    image: gfg-flask-app
    environment:
      - FLASK_ENV=development
    ports:
      - "5000:5000"
    depends_on:
      - redis
  redis:
    image: "redis:4.0.11-alpine"
```

TASK 5: RUN AND TEST

- We used docker-compose up to start the app and Redis.

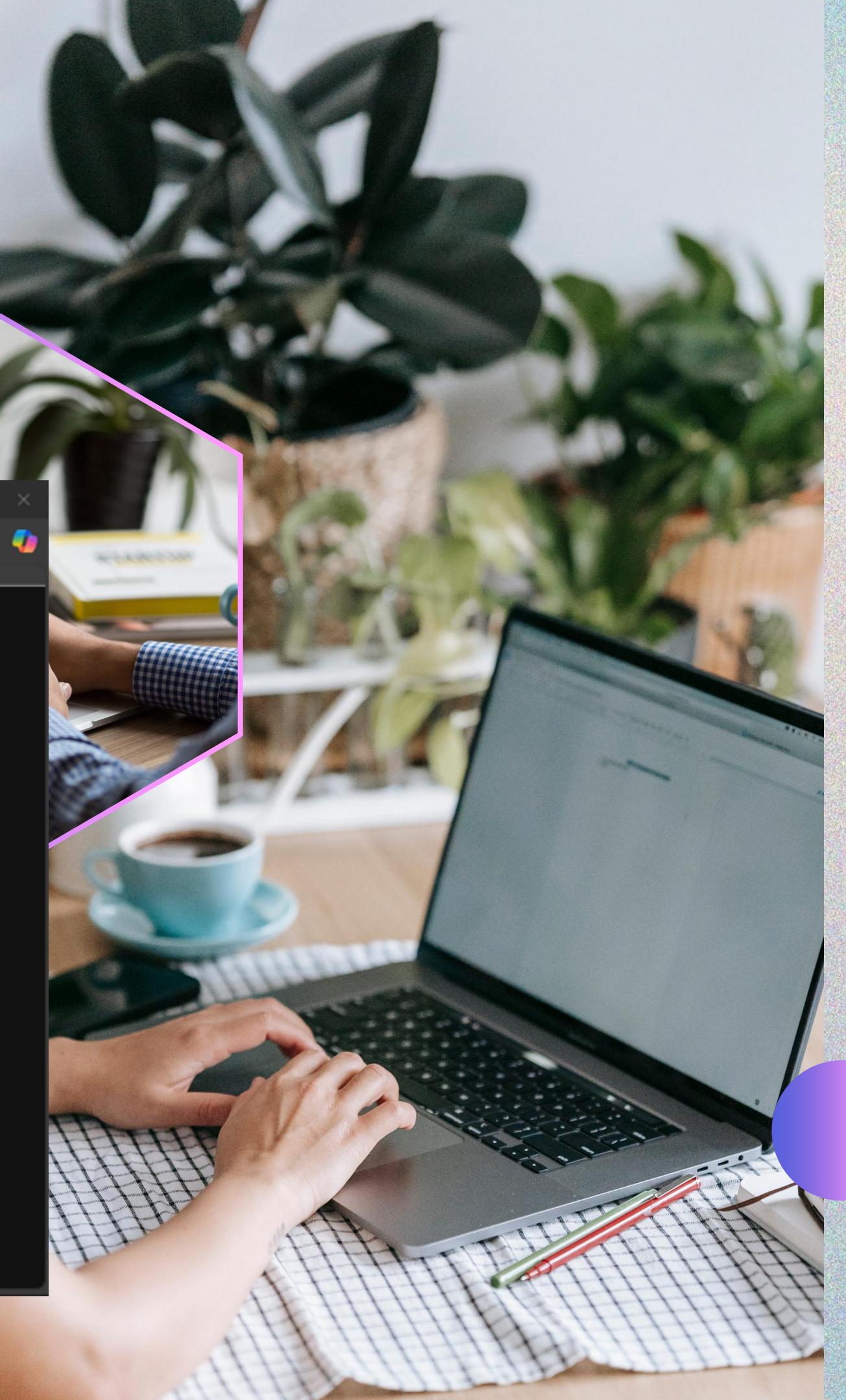
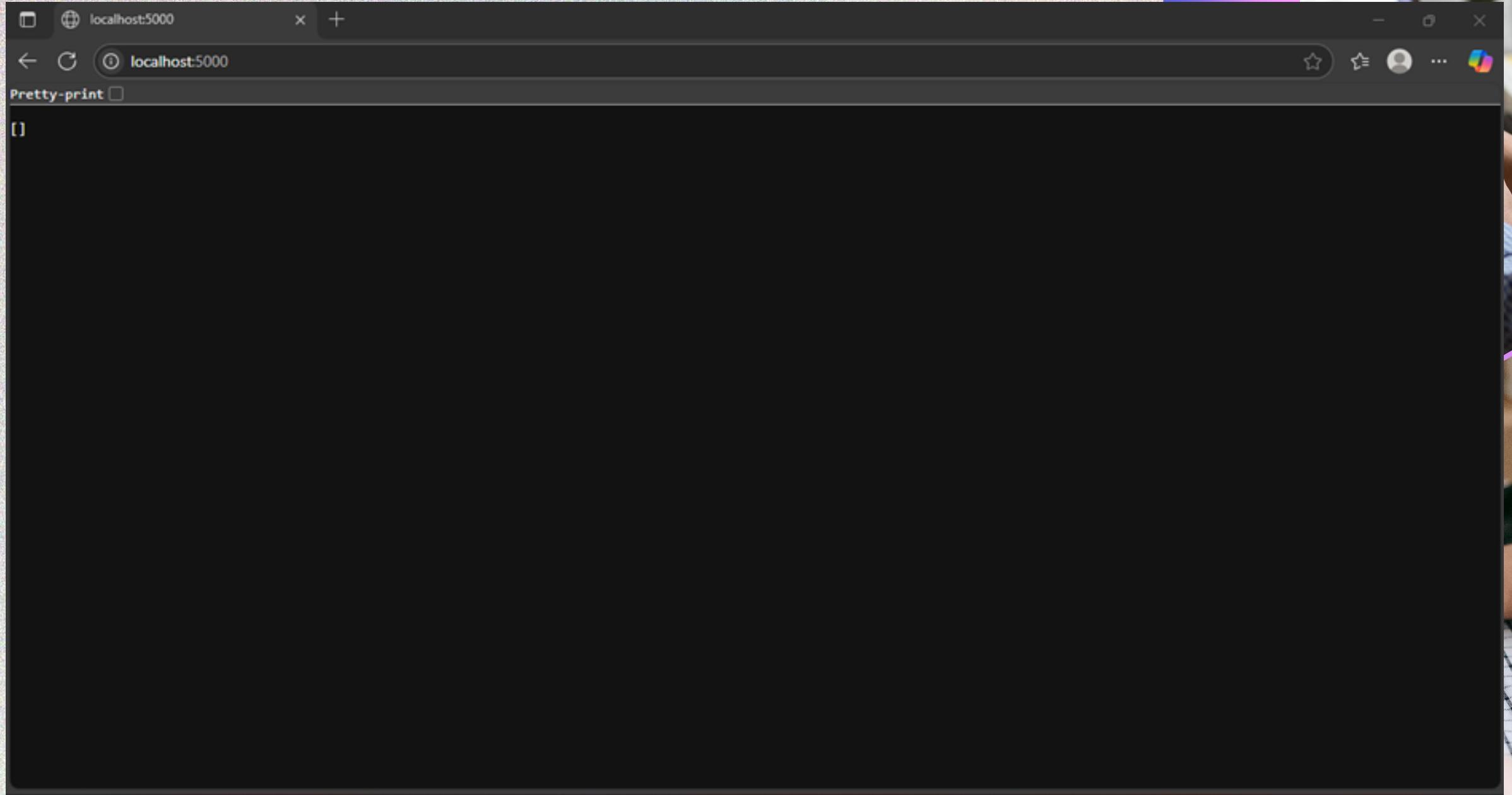


```
PS C:\Group5> docker-compose up
time="2025-07-11T10:56:09Z" level=warning msg="C:\\\\Group5\\\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
Attaching to app-1, redis-1
redis-1 | 1:C 11 Jul 10:56:11.052 # o000o000o000 Redis is starting o000o000o000
redis-1 | 1:C 11 Jul 10:56:11.053 # Redis version=4.0.11, bits=64, commit=00000000, modified=0, pid=1, just started
redis-1 | 1:C 11 Jul 10:56:11.053 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis-1 | 1:M 11 Jul 10:56:11.058 * Running mode=standalone, port=6379.
redis-1 | 1:M 11 Jul 10:56:11.059 # Server initialized
redis-1 | 1:M 11 Jul 10:56:11.060 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
redis-1 | 1:M 11 Jul 10:56:11.060 * DB loaded from disk: 0.001 seconds
redis-1 | 1:M 11 Jul 10:56:11.060 * Ready to accept connections
app-1 | * Serving Flask app 'app.py'
app-1 | * Debug mode: on
app-1 | 'FLASK_ENV' is deprecated and will not be used in Flask 2.3. Use 'FLASK_DEBUG' instead.
```



TASK 5: RUN AND TEST

- We visited `http://localhost:5000` in a browser for verification.



TASK 5: RUN AND TEST

- We tested adding data using a `populate.py` script.



A screenshot of a Microsoft Visual Studio Code interface. The left sidebar shows file navigation with items like 'EXPLORER', 'GROUPS' (containing 'app.py', 'docker-compose.yml', 'dockerfile', 'populate.py', and 'requirements.txt'), and other icons for AWS, Yarn, and GitHub. The main editor area displays a Python script named 'populate.py'. The code imports 'requests' and defines a list of animals to post to a local endpoint. The terminal at the bottom shows the output of running the script, indicating success for a 'Zebra' entry with status 201.

```
import requests

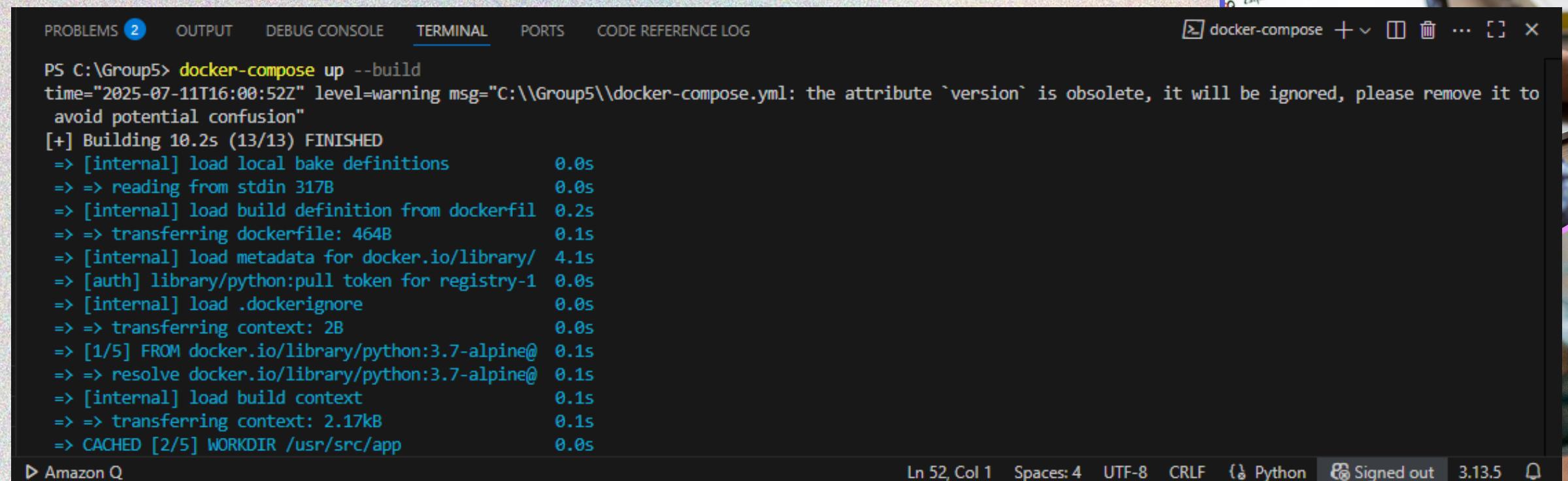
data = [
    {"name": "Lion"},
    {"name": "Elephant"},
    {"name": "Giraffe"},
    {"name": "Zebra"}
]

for animal in data:
    response = requests.post("http://localhost:5000", json=animal)
    print(f"Posted {animal['name']}: Status {response.status_code}")
```

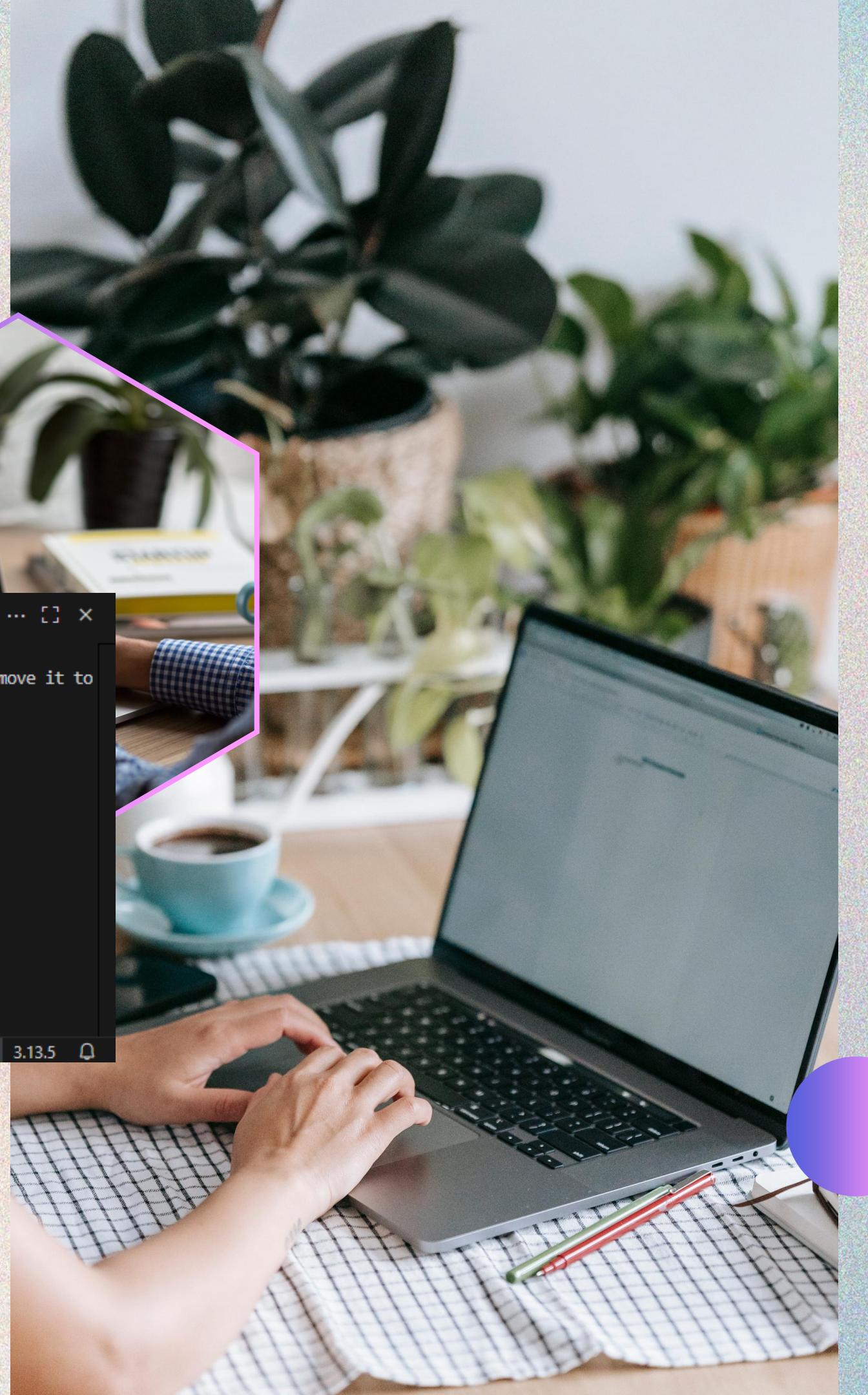
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
Posted Zebra: Status 201
PS C:\Group5>

TASK 5: RUN AND TEST

- We run our multi-container app with the command `docker-compose up --build`.

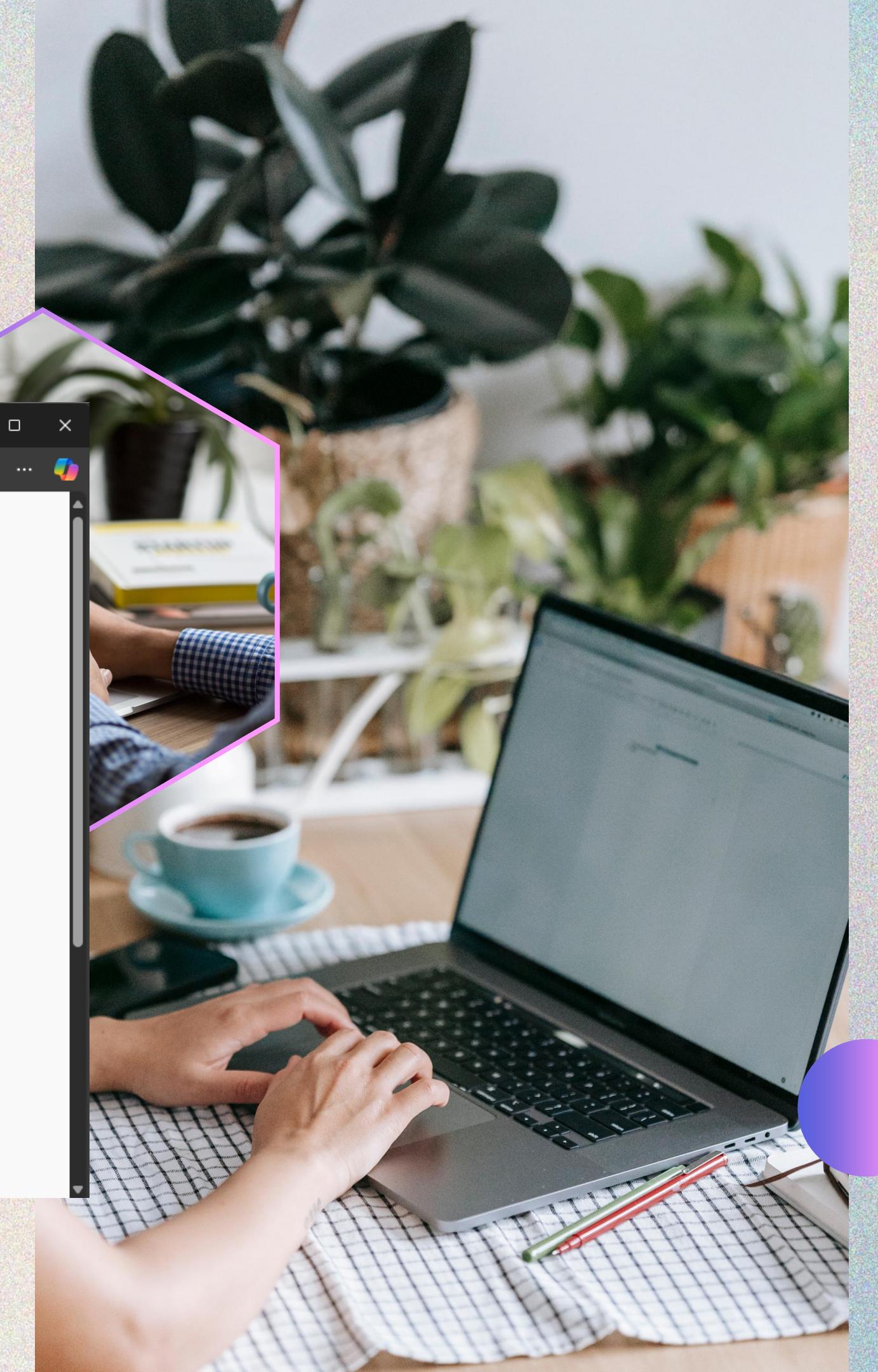
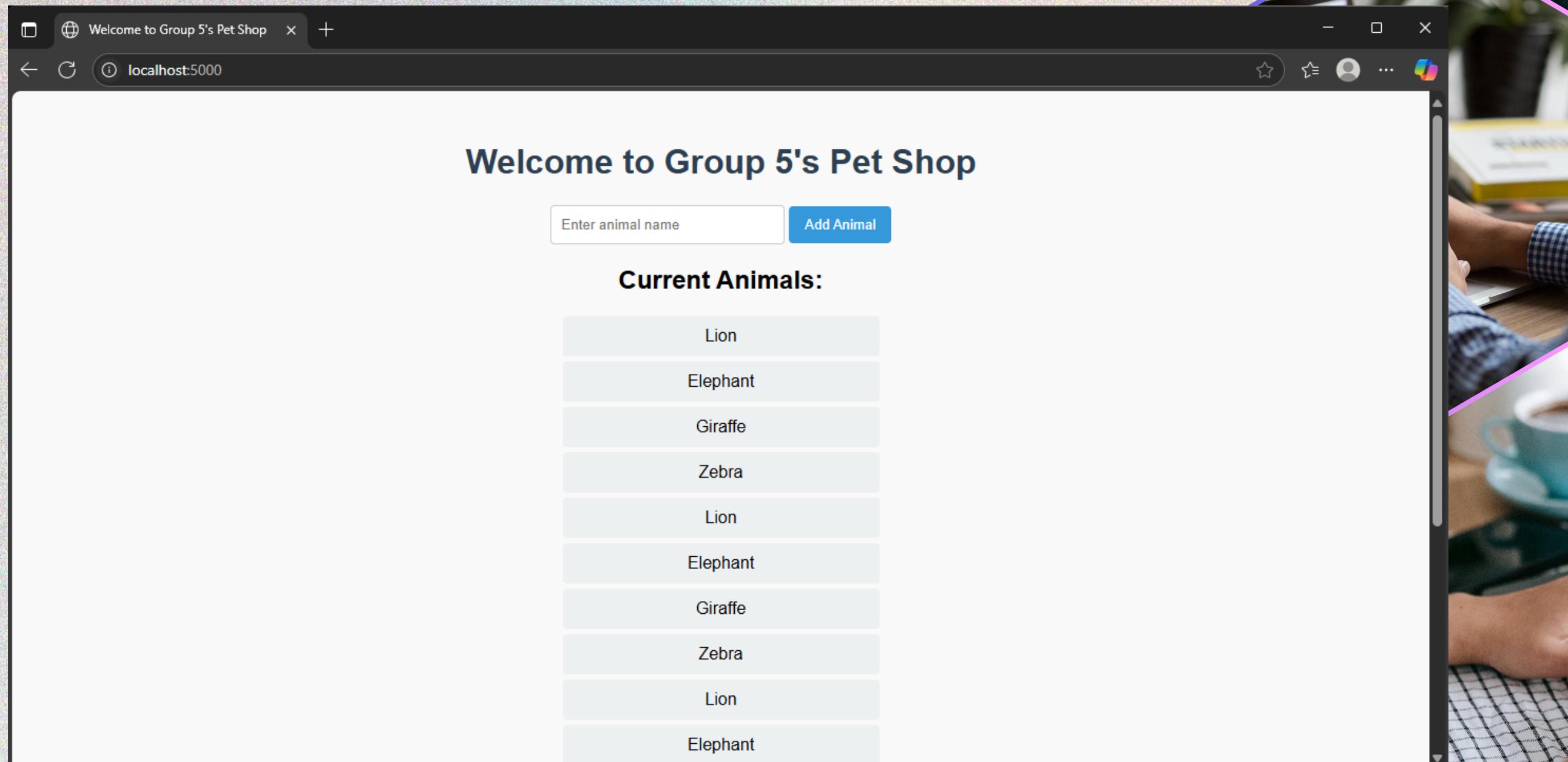


```
PS C:\Group5> docker-compose up --build
time="2025-07-11T16:00:52Z" level=warning msg="C:\\Group5\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 10.2s (13/13) FINISHED
  => [internal] load local bake definitions          0.0s
  => => reading from stdin 317B                      0.0s
  => [internal] load build definition from dockerfile 0.2s
  => => transferring dockerfile: 464B                0.1s
  => [internal] load metadata for docker.io/library/ 4.1s
  => [auth] library/python:pull token for registry-1 0.0s
  => [internal] load .dockerignore                   0.0s
  => => transferring context: 2B                  0.0s
  => [1/5] FROM docker.io/library/python:3.7-alpine@ 0.1s
  => => resolve docker.io/library/python:3.7-alpine@ 0.1s
  => [internal] load build context                 0.1s
  => => transferring context: 2.17kB            0.1s
  => CACHED [2/5] WORKDIR /usr/src/app           0.0s
```



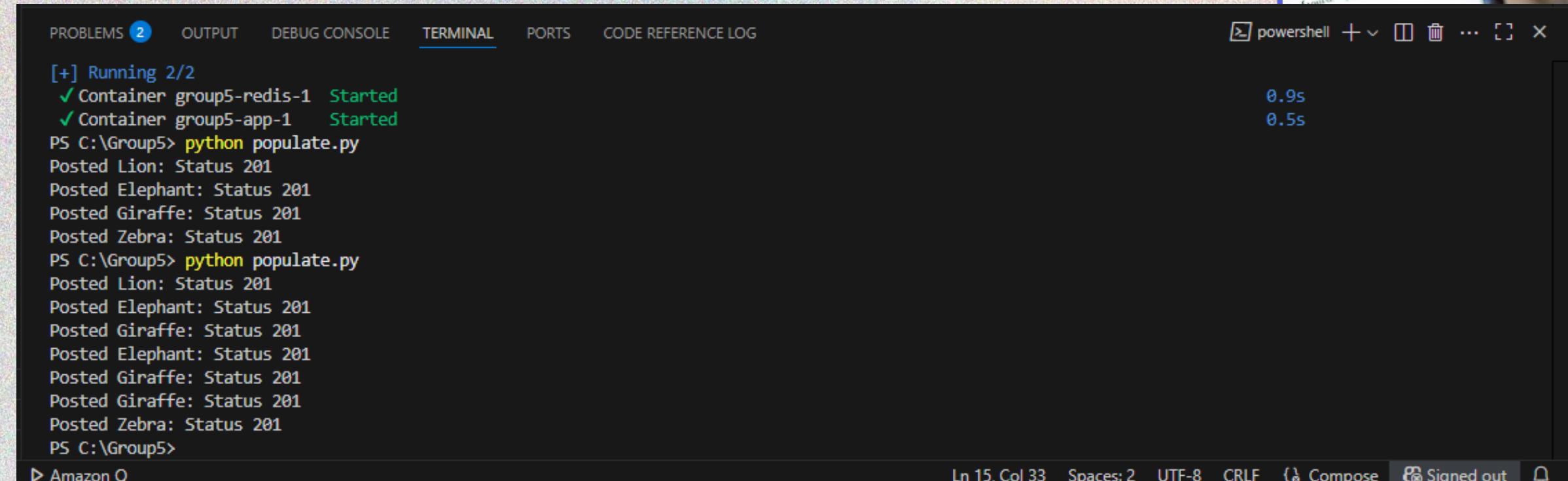
TASK 5: RUN AND TEST

- The output after refresing our browser.



TASK 5: RUN AND TEST

- The output after running the command, `python populate.py`.



```
[+] Running 2/2
✓ Container group5-redis-1 Started
✓ Container group5-app-1 Started
PS C:\Group5> python populate.py
Posted Lion: Status 201
Posted Elephant: Status 201
Posted Giraffe: Status 201
Posted Zebra: Status 201
PS C:\Group5> python populate.py
Posted Lion: Status 201
Posted Elephant: Status 201
Posted Giraffe: Status 201
Posted Elephant: Status 201
Posted Giraffe: Status 201
Posted Giraffe: Status 201
Posted Zebra: Status 201
PS C:\Group5>
```



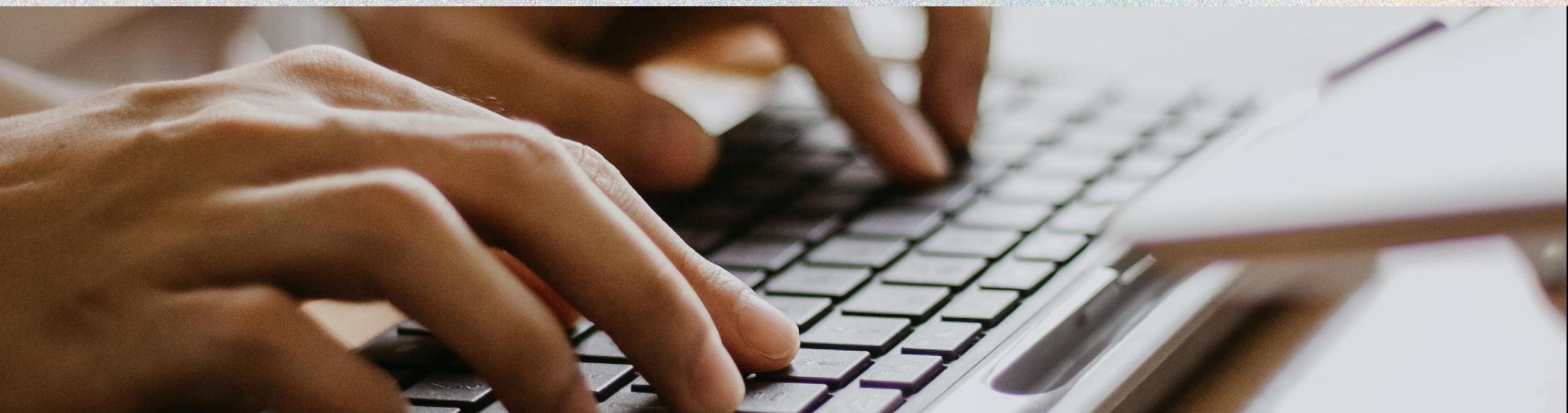
CHALLENGES



- We created another subfolder and placed the Dockerfile inside, causing build path issues.
- Python was not installed initially, requiring us to install Python before proceeding.
- We had to install the requests library to run populate.py successfully.
- Connection issues occurred when Redis was not recognized until the correct host and environment setup were used.

CONCLUSION

This project helped us understand how to build a simple web application and connect it to a database using containers. We practiced using Docker, Docker Compose, and basic Flask app development, preparing us for larger projects in cloud and container-based deployments.





MEET THE TEAM

- Macleana Mensah Oteng
- Esther Acheampong
- Selinam Fudzi-Amesu
- Clement Owusu Bempah
- Abigail Safoa
- Fahad Mohammed Gibrine



**THANK
YOU!**