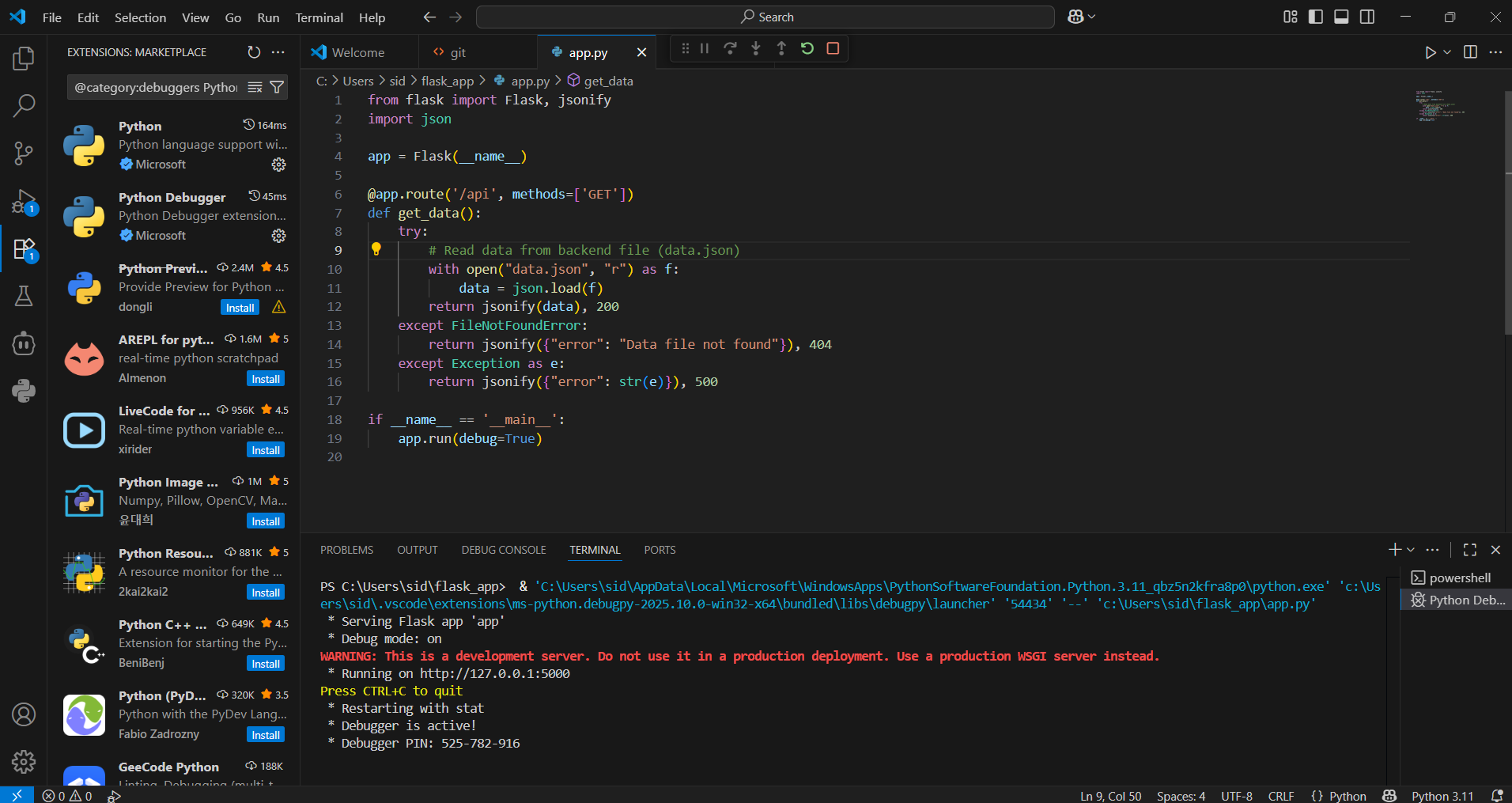
**Assignment #03**

1Q. Create a Flask application with an /api route. When this route is accessed, it should return a JSON list. The data should be stored in a backend file, read from it, and sent as a response.



**Solution:**

[

{"id": 1, "name": "Alice", "role": "Developer"},

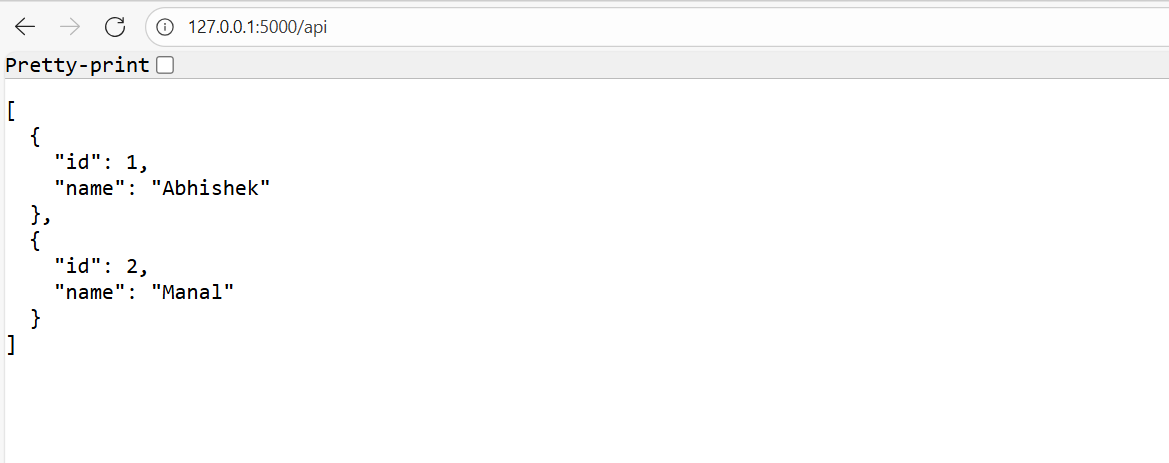
{"id": 2, "name": "Bob", "role": "DevOps Engineer"},

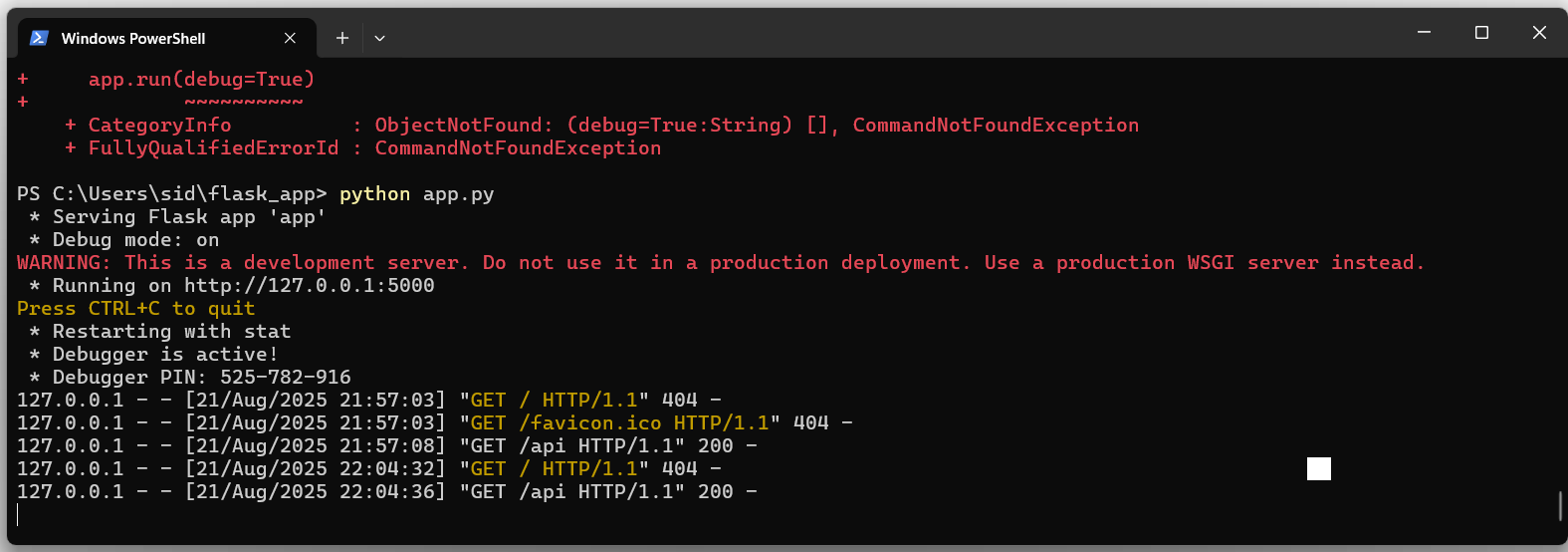
{"id": 3, "name": "Charlie", "role": "Tester"}

]



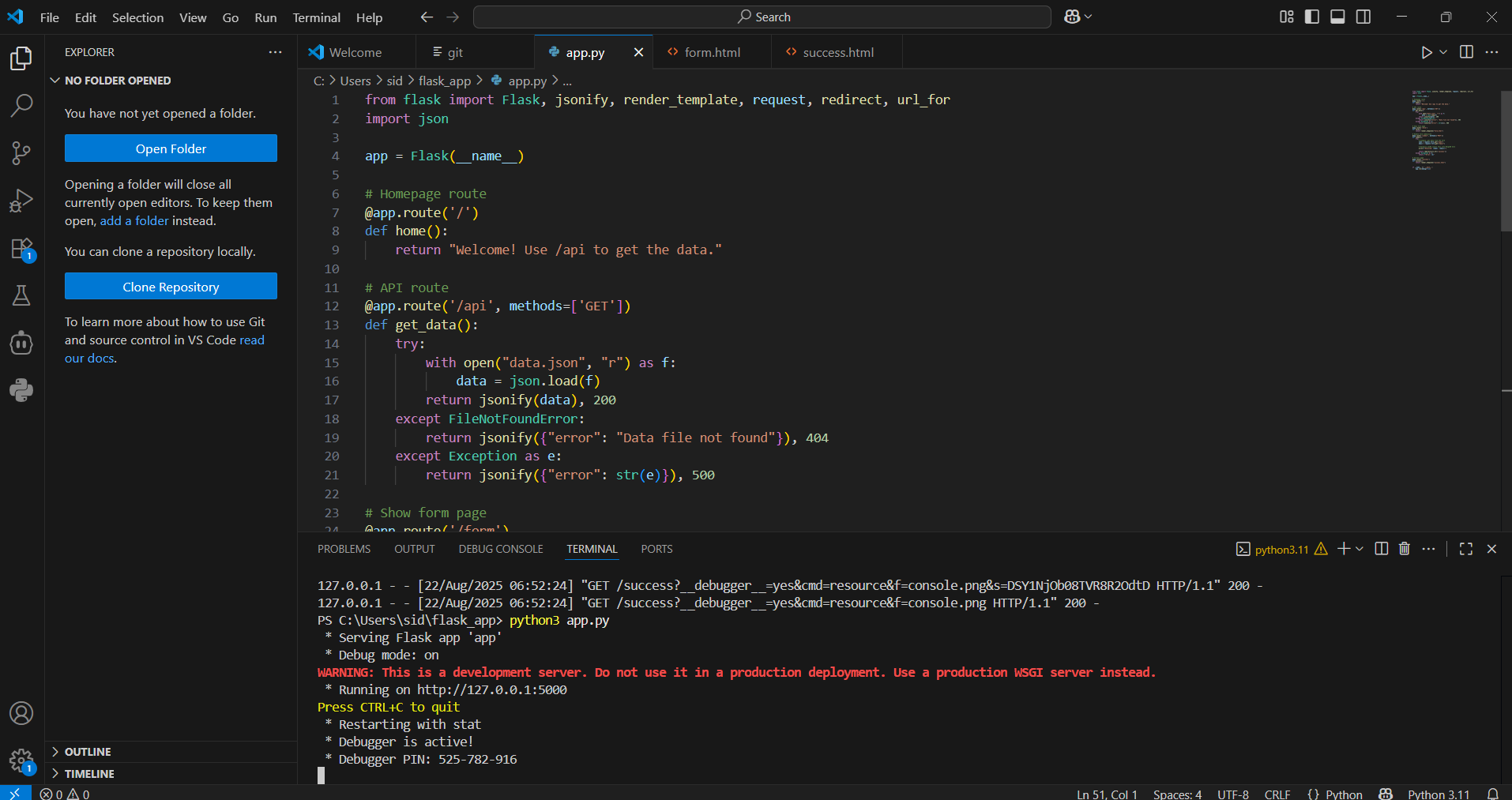
<http://127.0.0.1:5000/api>

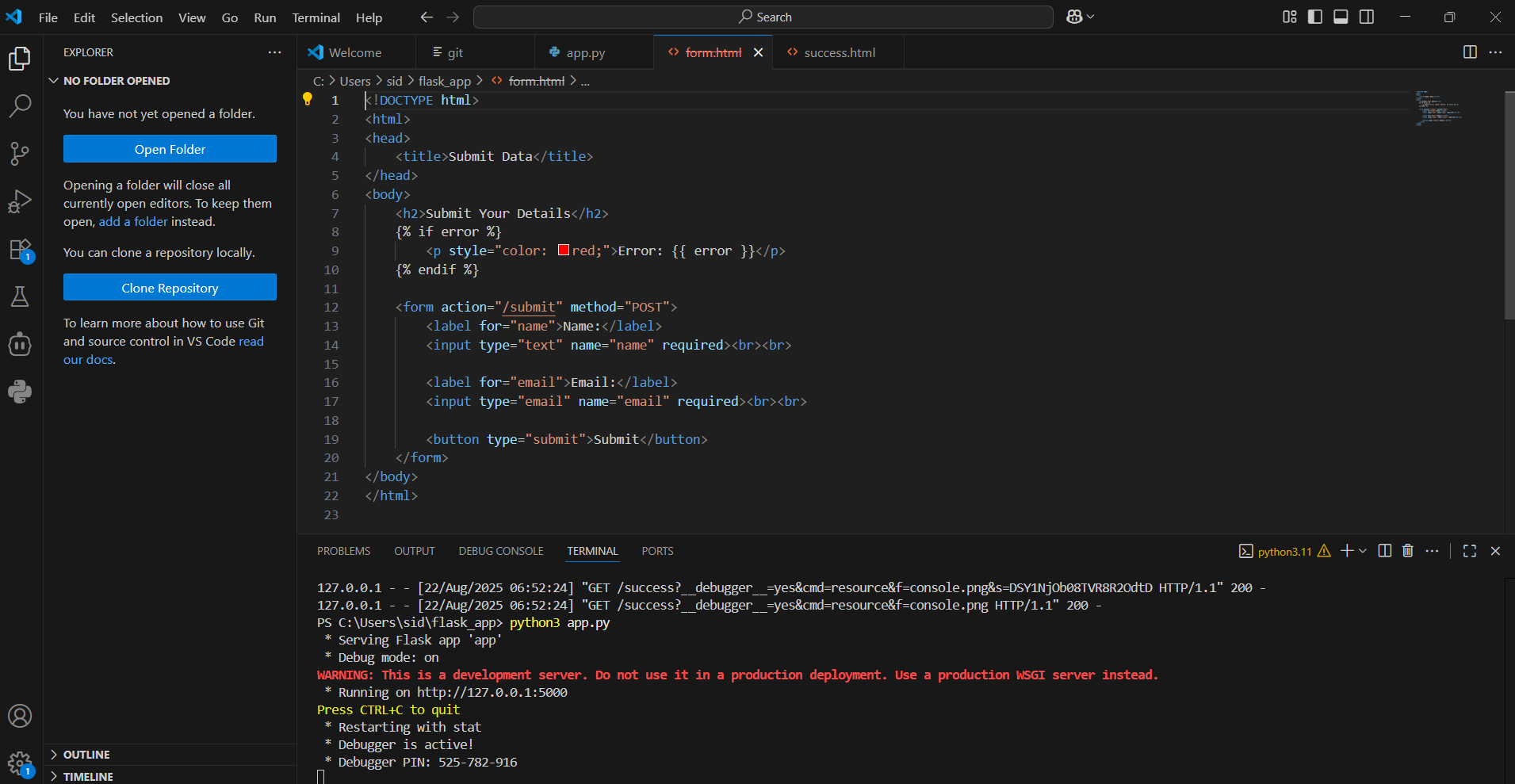


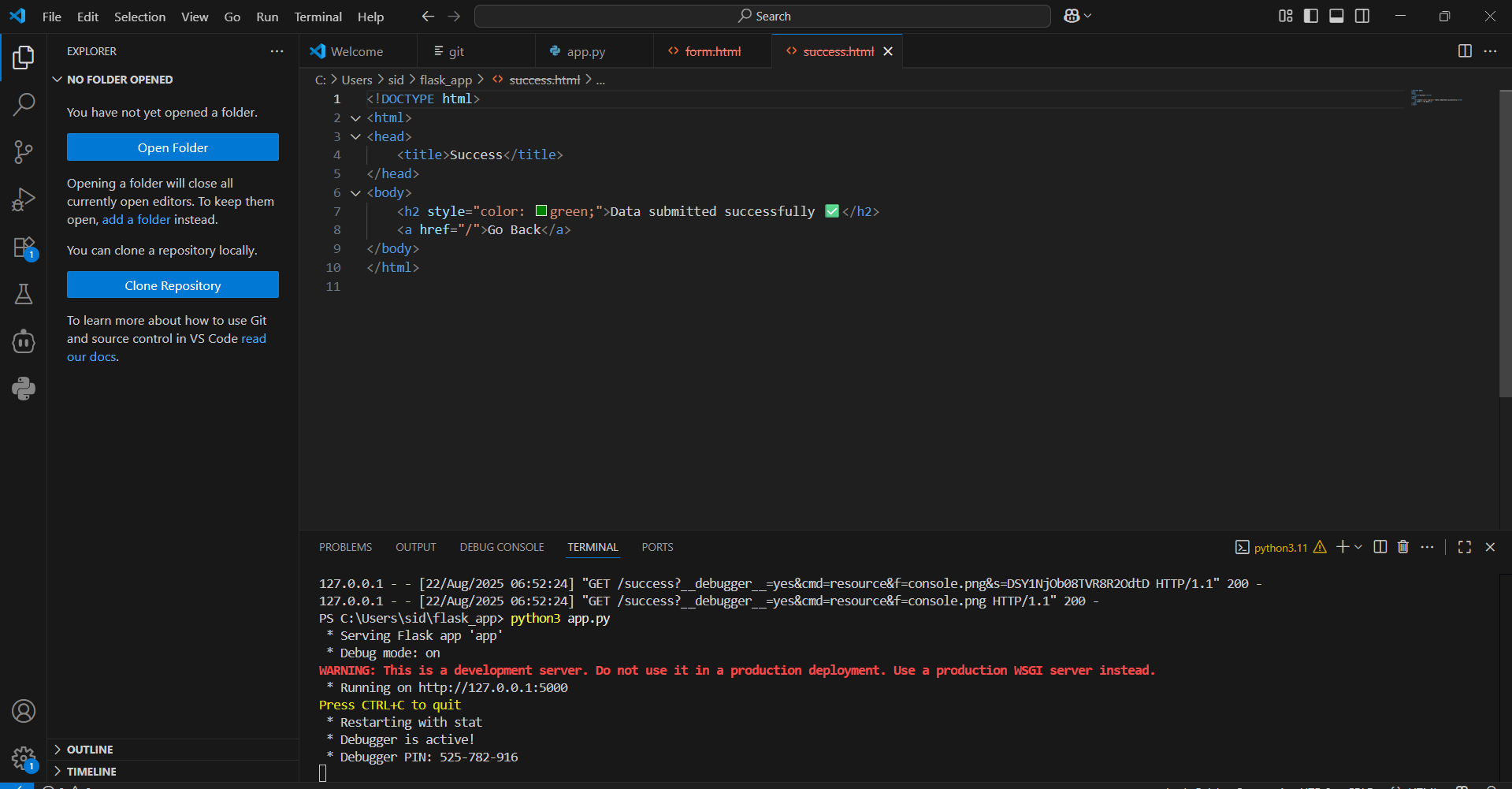


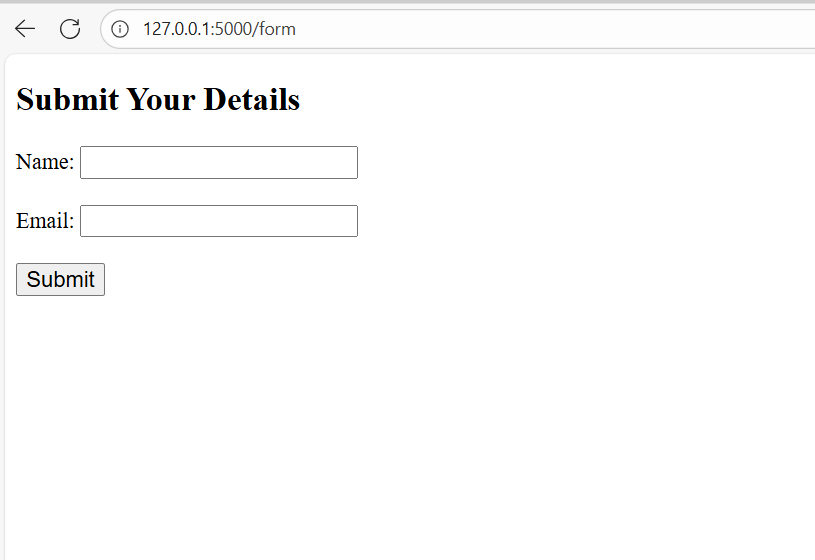
2Q) 2. Create a form on the frontend that, when submitted, inserts data into MongoDB Atlas. Upon successful submission, the user should be redirected to another page displaying the message **"Data submitted successfully"**. If there's an error during submission, display the error on the same page without redirection.

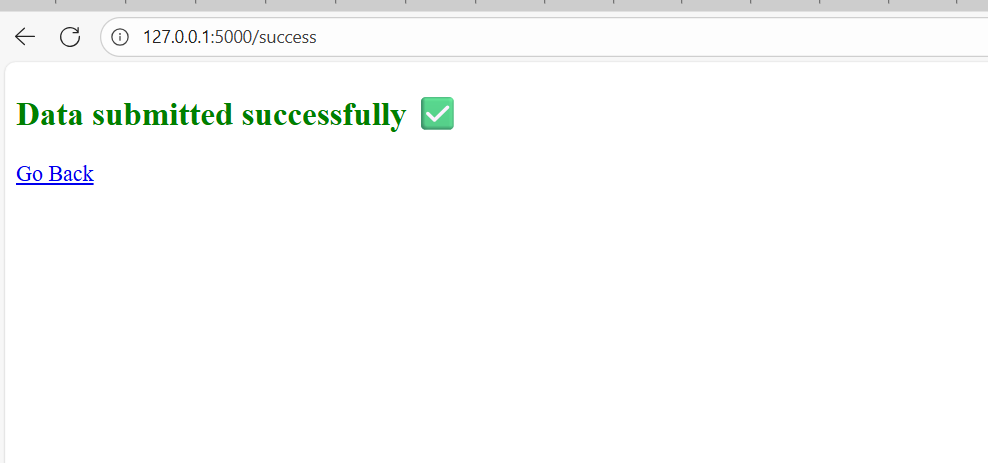
Solution for Question #02











-------------------------------------------------------------------------------------------------

**Linux Assignments:**

1Q) Creating and Renaming Files/Directories

Create a directory named test\_dir using mkdir.

Inside test\_dir, create an empty file called example.txt.

Rename example.txt to renamed\_example.txt using mv

# 1. Create a directory named test\_dir

mkdir test\_dir

# 2. Go inside test\_dir

cd test\_dir

# 3. Create an empty file called example.txt

touch example.txt

# 4. Rename example.txt to renamed\_example.txt

mv example.txt renamed\_example.txt

Explanation:

mkdir test\_dir → makes a new directory.

cd test\_dir → navigates into that directory.

touch example.txt → creates an empty file.

mv example.txt renamed\_example.txt → renames the file.

-----------------------------------------------------------------------------------------

2Q) Viewing File Contents

Use cat to display the contents of /etc/passwd.

Display only the first 5 lines of /etc/passwd using head.

Display only the last 5 lines of /etc/passwd using tail.

Solutioin:

# 1. Display the entire contents of /etc/passwd

cat /etc/passwd

# 2. Display only the first 5 lines of /etc/passwd

head -n 5 /etc/passwd

# 3. Display only the last 5 lines of /etc/passwd

tail -n 5 /etc/passwd

---------------------------------------------------------------------------------------

3Q) Searching for Patterns

Use grep to find all lines containing the word "root" in /etc/passwd.

Solution:

grep "root" /etc/passwd

---------------------------------------------------------------------------------

4Q) Zipping and Unzipping

Compress the test\_dir directory into a file named test\_dir.zip using zip.

Unzip test\_dir.zip into a new directory named unzipped\_dir.

Solution:

zip -r test\_dir.zip test\_dir

Explanation:

zip → command to create a zip file.

-r → recursive, includes all files and subdirectories.

test\_dir.zip → name of the zip archive.

test\_dir → directory to compress.

unzip test\_dir.zip -d unzipped\_dir

Explanation:

unzip → command to extract a zip file.

-d unzipped\_dir → specifies the destination directory.

5Q) Downloading Files

Use wget to download a file from a URL (e.g., https://example.com/sample.txt).

Solution:

wget https://example.com/sample.txt

6Q) Changing Permissions

Create a file named secure.txt and change its permissions to read-only for everyone using chmod.

Solution:

touch secure.txt

chmod 444 secure.txt

-----------------------------------------------------------------------------------------------------

7Q)Working with Environment Variables

Use export to set a new environment variable called MY\_VAR with the value "Hello, Linux!".

Solution:

export MY\_VAR="Hello, Linux!"

echo $MY\_VAR

Hello, Linux!

printenv MY\_VAR

-----------------------------------------------------------------------------------------

Q1) 1. Grade Checker

Take a score as input and print the grade based on the following:

90+ : "A"

80-89 : "B"

70-79 : "C"

60-69 : "D"

Below 60 : "F"

here we used a basic if else statement to carry out marks and all.

**Solution:**

# Grade Checker Program

# Take score input from the user

score = int(input("Enter your score: "))

# Check grade using if-else

if score >= 90:

grade = "A"

elif score >= 80:

grade = "B"

elif score >= 70:

grade = "C"

elif score >= 60:

grade = "D"

else:

grade = "F"

# Print result

print(f"Your grade is: {grade}")

**Explanaton:**

The program takes an integer score as input.

Uses if-elif-else conditions to decide the grade.

Prints the grade.

2Q) Student Grades

Create a dictionary where the keys are student names and the values are their grades. Allow the user to:

Add a new student and grade.

Update an existing student’s grade.

Print all student grades.

Used dictionary and basic operations. Using if else:

**Solution:**

# Student Grades Program

# Dictionary to store student names and grades

student\_grades = {}

while True:

print("\n--- Student Grades Menu ---")

print("1. Add a new student and grade")

print("2. Update an existing student's grade")

print("3. Print all student grades")

print("4. Exit")

choice = int(input("Enter your choice (1-4): "))

if choice == 1:

name = input("Enter student name: ")

grade = input("Enter grade: ")

student\_grades[name] = grade

print(f"{name} added with grade {grade}.")

elif choice == 2:

name = input("Enter student name to update: ")

if name in student\_grades:

grade = input("Enter new grade: ")

student\_grades[name] = grade

print(f"{name}'s grade updated to {grade}.")

else:

print(f"Student {name} not found.")

elif choice == 3:

if student\_grades:

**Explanation:**

Keeps student names + grades in a dictionary.

Lets you **add**, **update**, or **print** all grades.

Runs in a loop until you choose **Exit (4)**.

3Q).Write to a File

Write a program to create a text file and write some content to it.

Using file functions like write and open.

**Solution:**

# Write to a File Program

# Open a file in write mode ("w")

# If the file doesn't exist, it will be created

with open("myfile.txt", "w") as file:

file.write("Hello, this is my first file in Python!\n")

file.write("I am learning file handling using open() and write().\n")

print("File created and content written successfully by Hasan!")

**Explanation:**

open("myfile.txt", "w") → opens/creates a file in **write mode**.

file.write("text") → writes text into the file.

with is used so the file is **automatically closed** after writing.

4Q). Read from a File We used open in read mode and file.read to read and print to display.

# Read from a File Program

# Open the file in read mode ("r")

with open("myfile.txt", "r") as file:

content = file.read()

# Print the content of the file

print("File Content:\n")

print(content)

**Explanation:**

open("myfile.txt", "r") → opens the file in **read mode**.

file.read() → reads the **entire content** of the file.

with automatically closes the file after reading.

Assingnment – 2 Flask and mongodb

1Q). Create a Flask application with an /api route. When this route is accessed, it should return a JSON list. The data should be stored in a backend file, read from it, and sent as a response.

**Solution:**

**from flask import Flask, jsonify**

**import json**

**app = Flask(\_\_name\_\_)**

**# API route**

**@app.route('/api', methods=['GET'])**

**def get\_data():**

**try:**

**# Read data from backend file**

**with open("data.json", "r") as file:**

**data = json.load(file)**

**return jsonify(data), 200**

**except Exception as e:**

**return jsonify({"error": str(e)}), 500**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug=True)**

2Q). Create a form on the frontend that, when submitted, inserts data into MongoDB Atlas. Upon successful submission, the user should be redirected to another page displaying the message **"Data submitted successfully"**. If there's an error during submission, display the error on the same page without redirection.

**Project Structure:**

project/

│── app.py

│── templates/

│ ├── form.html

│ └── success.html

**App.py:**

from flask import Flask, render\_template, request

from pymongo import MongoClient

from bson.objectid import ObjectId

app = Flask(\_\_name\_\_)

# MongoDB Atlas Connection

# Replace <username>, <password>, and <cluster-url> with your details

client = MongoClient("mongodb+srv://<username>:<password>@<cluster-url>/test")

db = client["mydatabase"]

collection = db["students"]

@app.route('/', methods=['GET', 'POST'])

def form():

error = None

if request.method == 'POST':

try:

name = request.form['name']

email = request.form['email']

# Insert into MongoDB

collection.insert\_one({"name": name, "email": email})

# Redirect to success page

return render\_template("success.html")

except Exception as e:

error = str(e) # Show error on the same page

return render\_template("form.html", error=error)

@app.route('/success')

def success():

return render\_template("success.html")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Templates/form.html:**

<!DOCTYPE html>

<html>

<head>

<title>Submit Data</title>

</head>

<body>

<h2>Enter Your Details</h2>

<form method="POST">

<label>Name:</label>

<input type="text" name="name" required><br><br>

<label>Email:</label>

<input type="email" name="email" required><br><br>

<button type="submit">Submit</button>

</form>

{% if error %}

<p style="color:red;">Error: {{ error }}</p>

{% endif %}

</body>

</html>

**Templates/success.html:**

<!DOCTYPE html>

<html>

<head>

<title>Success</title>

</head>

<body>

<h2 style="color:green;">Data submitted successfully</h2>

<a href="/">Go Back</a>

</body>

</html>

Open / → you see the form.

Fill in details and submit → data is inserted into **MongoDB Atlas**.

If successful → redirected to /success page with message.

If error (like connection issue) → error is shown on **same form page**.