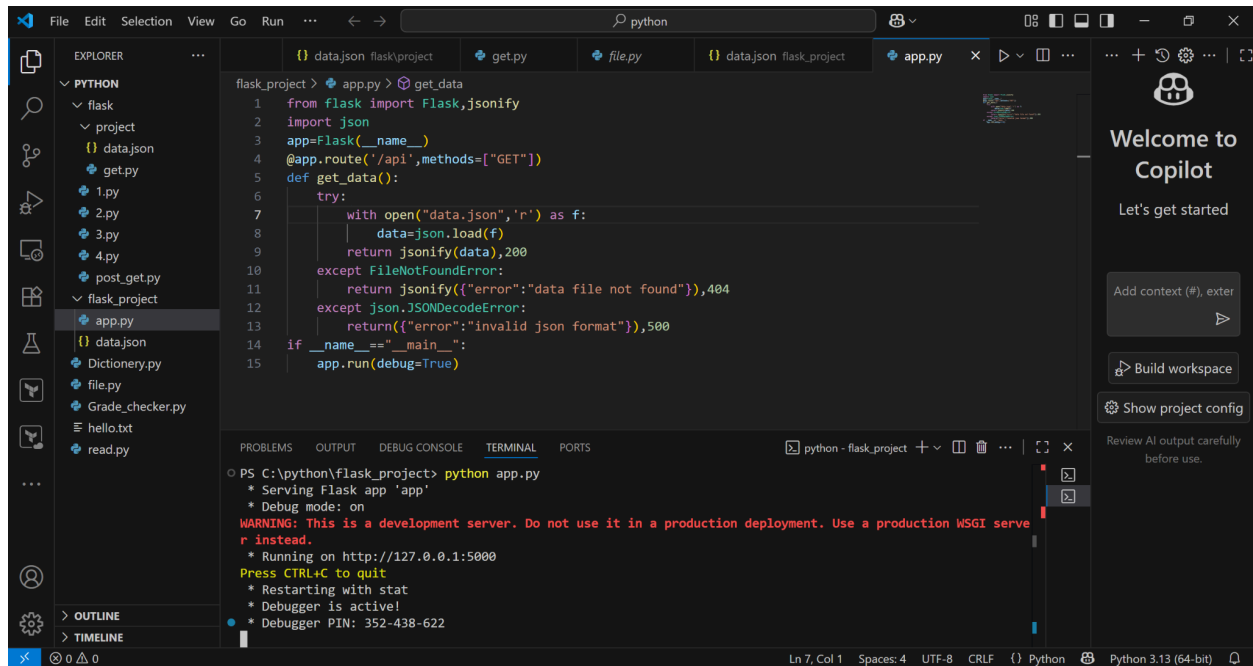


1. Create a Flask application with an `/api` route. When this route is accessed, it should return a JSON list. The data should be stored in a backend file, read from it, and sent as a response.

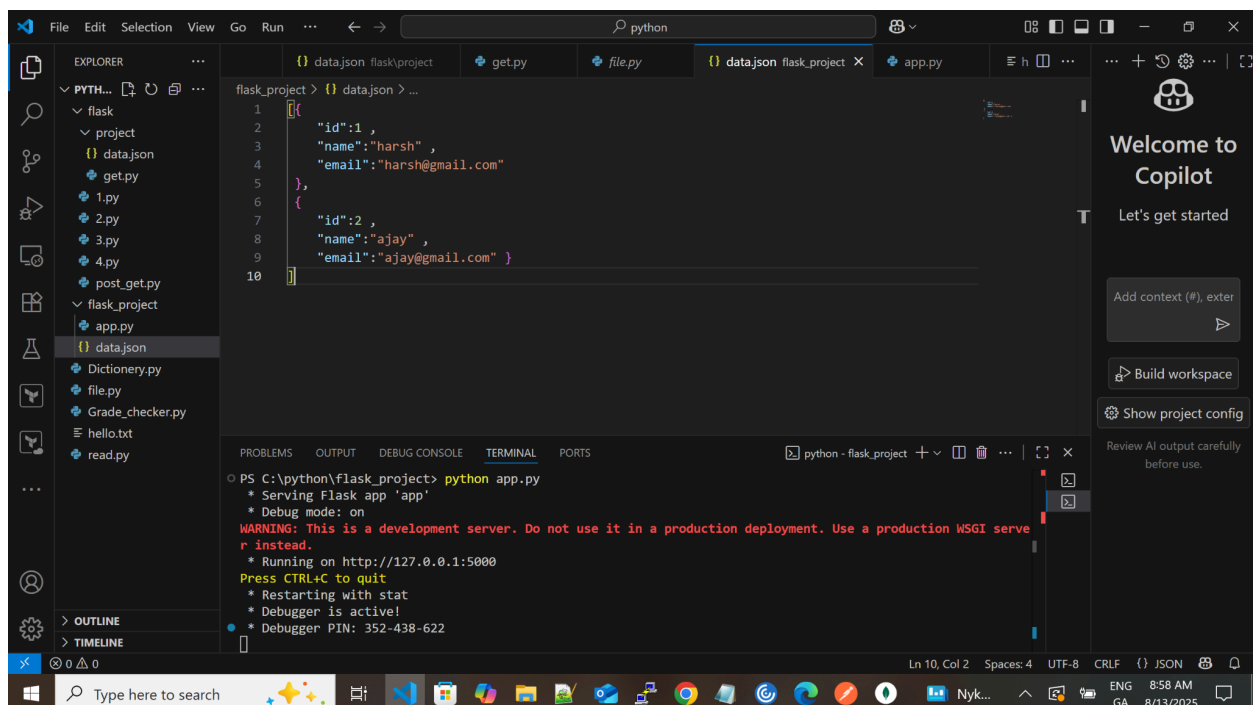
CODE FOR THE PROGRAM:



```
flask_project > app.py > get_data
1 from flask import Flask, jsonify
2 import json
3 app=Flask(__name__)
4 @app.route('/api', methods=["GET"])
5 def get_data():
6     try:
7         with open("data.json", 'r') as f:
8             data=json.load(f)
9             return jsonify(data),200
10    except FileNotFoundError:
11        return jsonify({"error":"data file not found"}),404
12    except json.JSONDecodeError:
13        return jsonify({"error":"invalid json format"}),500
14 if __name__=="__main__":
15     app.run(debug=True)
```

```
PS C:\python\flask_project> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 352-438-622
```

DATA FOR THE PROGRAM SO THE PROGRAM FETCH IT :-



```
flask_project > data.json > ...
1 [{"id":1 ,
2   "name":"harsh" ,
3   "email":"harsh@gmail.com"
4 },
5  {
6   "id":2 ,
7   "name":"ajay" ,
8   "email":"ajay@gmail.com" }
9 ]
10 ]
```

```
PS C:\python\flask_project> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 352-438-622
```

OUTPUT FOR THE PROGRAM BY POSTMAN:

The image shows the Postman application interface. At the top, a tab is labeled "GET Get data". The URL bar contains "http://127.0.0.1:5000/api". The "Body" tab is selected, showing "none" as the content type. Below the request setup, the response is displayed with a status of "200 OK", a time of "5 ms", and a size of "315 B". The response is in JSON format, showing an array of two objects. The first object represents a user with email "harsh@gmail.com", id 1, and name "harsh". The second object represents a user with email "ajay@gmail.com", id 2, and name "ajay".

port GET Get data + No environment

My Collection / Get data Save Share

GET http://127.0.0.1:5000/api Send

Params Authorization Headers (7) Body Scripts Settings Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

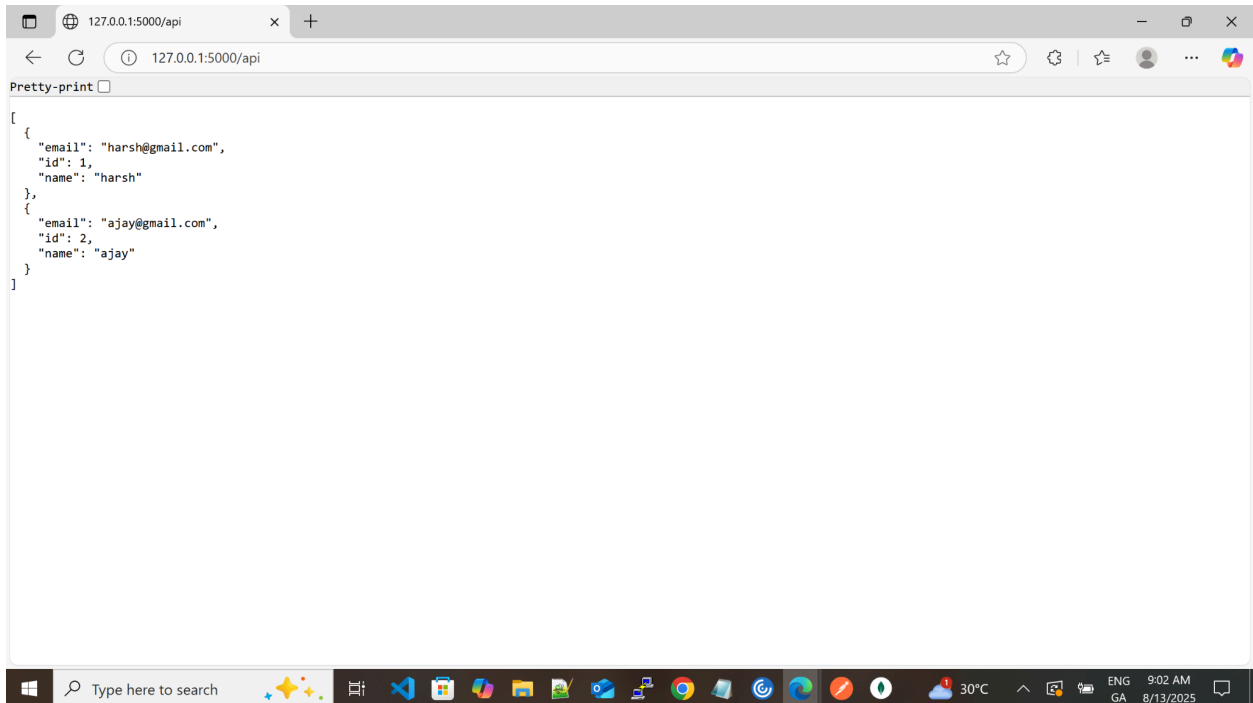
This request does not have a body

Body Cookies Headers (5) Test Results (1/1) 200 OK • 5 ms • 315 B Save Response

{ } JSON Preview Visualize

```
1  [
2    {
3      "email": "harsh@gmail.com",
4      "id": 1,
5      "name": "harsh"
6    },
7    {
8      "email": "ajay@gmail.com",
9      "id": 2,
10     "name": "ajay"
11   }
12 ]
```

OUTPUT FOR THE PROGRAM BY THE URL:



```
[
  {
    "email": "harsh@gmail.com",
    "id": 1,
    "name": "harsh"
  },
  {
    "email": "ajay@gmail.com",
    "id": 2,
    "name": "ajay"
  }
]
```

EXPLANATION OF THE PROGRAM:

- `from flask import Flask, jsonify`

Flask aur jsonify import kar rahe ho.

Flask → Flask app banane ke liye

jsonify → Python ka data (like list or dict) ko JSON response banane ke liye

- `import json`

Python ka built-in json module import kiya — jisse file se JSON data read kar sakein.

- `app = Flask(__name__)`

Initializes your Flask application. `__name__` tells Flask where to look for resources.

- `@app.route('/api', methods=["GET"])`
`def get_data():`

This sets up a route `/api` that only responds to GET requests.

When someone visits `/api`, the `get_data()` function runs.

- **Inside the get() function:**

```
try:  
    with open("data.json", 'r') as f:  
        data = json.load(f)
```

Tries to open a file named data.json in read mode.

If successful, it loads and parses the JSON data from that file into a Python variable called data.

- return jsonify(data), 200

If everything works, it returns the data as a JSON response with HTTP status code 200 OK.

- **Handling Errors:**

```
except FileNotFoundError:  
  
    return jsonify({"error": "data file not found"}), 404
```

If data.json doesn't exist, it returns an error message with status 404 Not Found.

```
except json.JSONDecodeError:  
  
    return({"error": "invalid json format"}), 500
```

If data.json exists but has invalid JSON syntax, this returns a 500 Internal Server Error with an appropriate message.

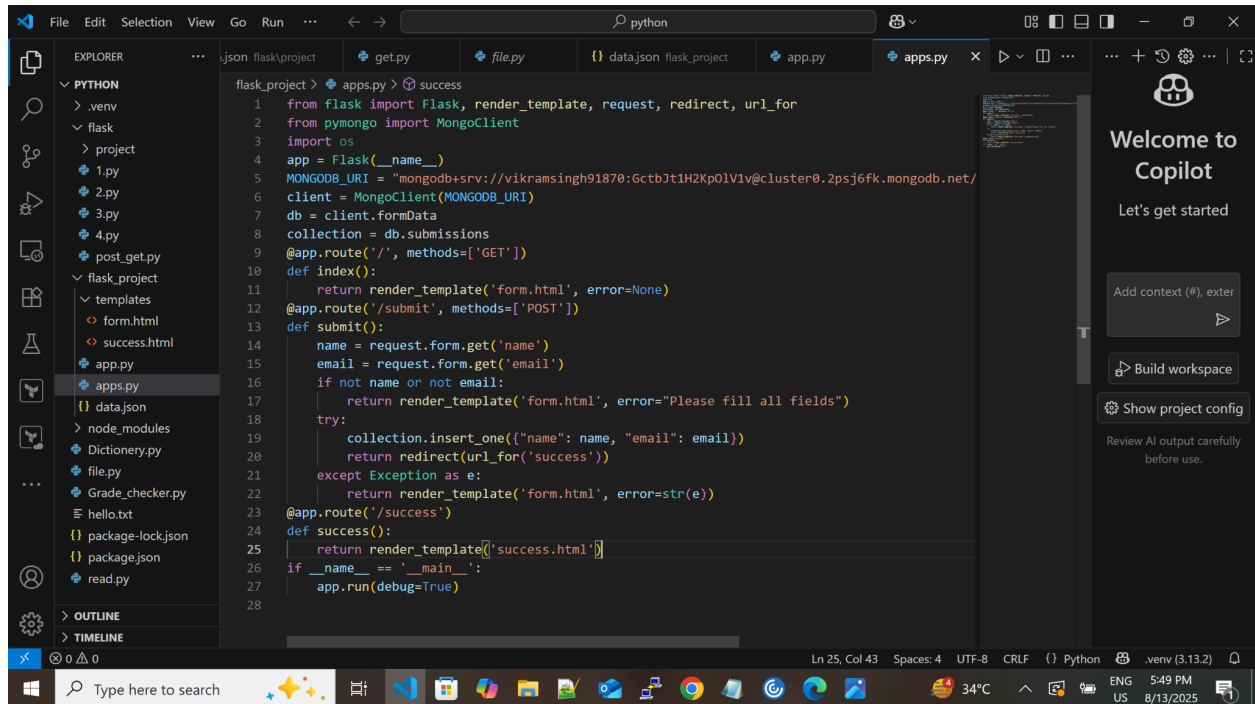
- if `__name__ == "__main__"`:

 app.run(debug=True)

This starts the Flask app in debug mode, so you can see error logs and live reload during development.

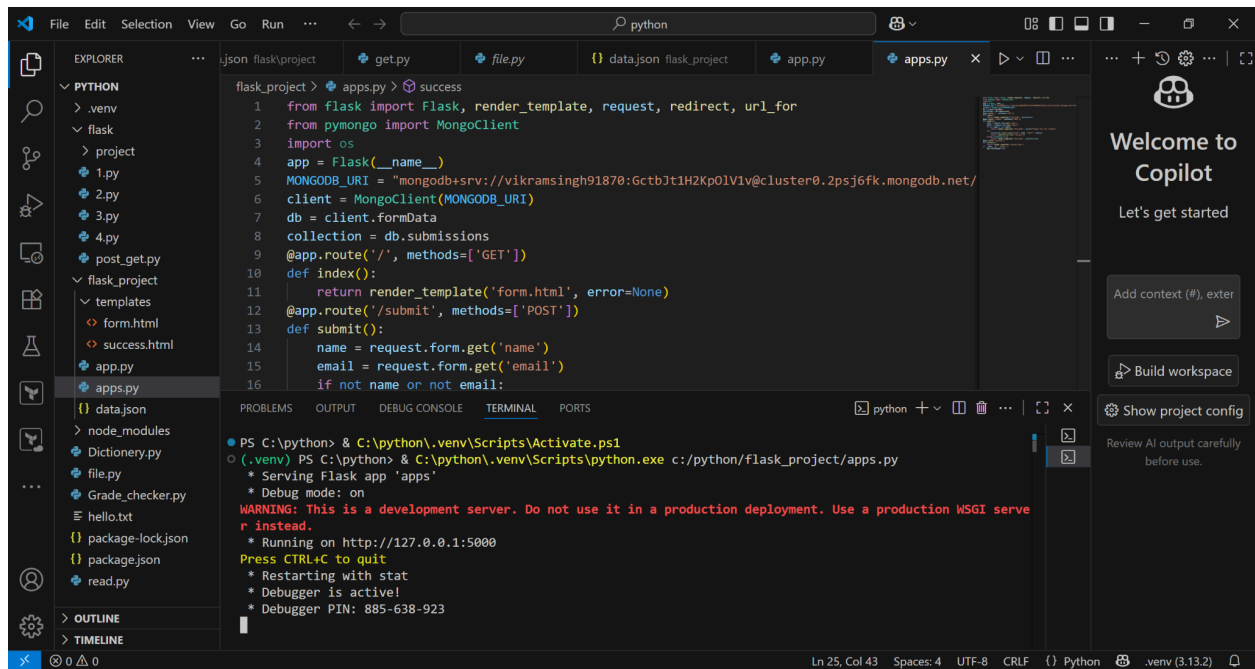
2. Create a form on the frontend that, when submitted, inserts data into MongoDB Atlas. Upon successful submission, the user should be redirected to another page displaying the message **"Data submitted successfully"**. If there's an error during submission, display the error on the same page without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

CODE FOR THE PROGRAM:



```
1 from flask import Flask, render_template, request, redirect, url_for
2 from pymongo import MongoClient
3 import os
4 app = Flask(__name__)
5 MONGODB_URI = "mongodb+srv://vikramsingh91870:Gctb3t1H2Kp0lV1v@cluster0.2psj6fk.mongodb.net/"
6 client = MongoClient(MONGODB_URI)
7 db = client.formData
8 collection = db.submissions
9 @app.route('/', methods=['GET'])
10 def index():
11     return render_template('form.html', error=None)
12 @app.route('/submit', methods=['POST'])
13 def submit():
14     name = request.form.get('name')
15     email = request.form.get('email')
16     if not name or not email:
17         return render_template('form.html', error="Please fill all fields")
18     try:
19         collection.insert_one({"name": name, "email": email})
20         return redirect(url_for('success'))
21     except Exception as e:
22         return render_template('form.html', error=str(e))
23 @app.route('/success')
24 def success():
25     return render_template('success.html')
26 if __name__ == '__main__':
27     app.run(debug=True)
```

OUTPUT OF THE PROGRAM IN THE VSCODE TERMINAL:

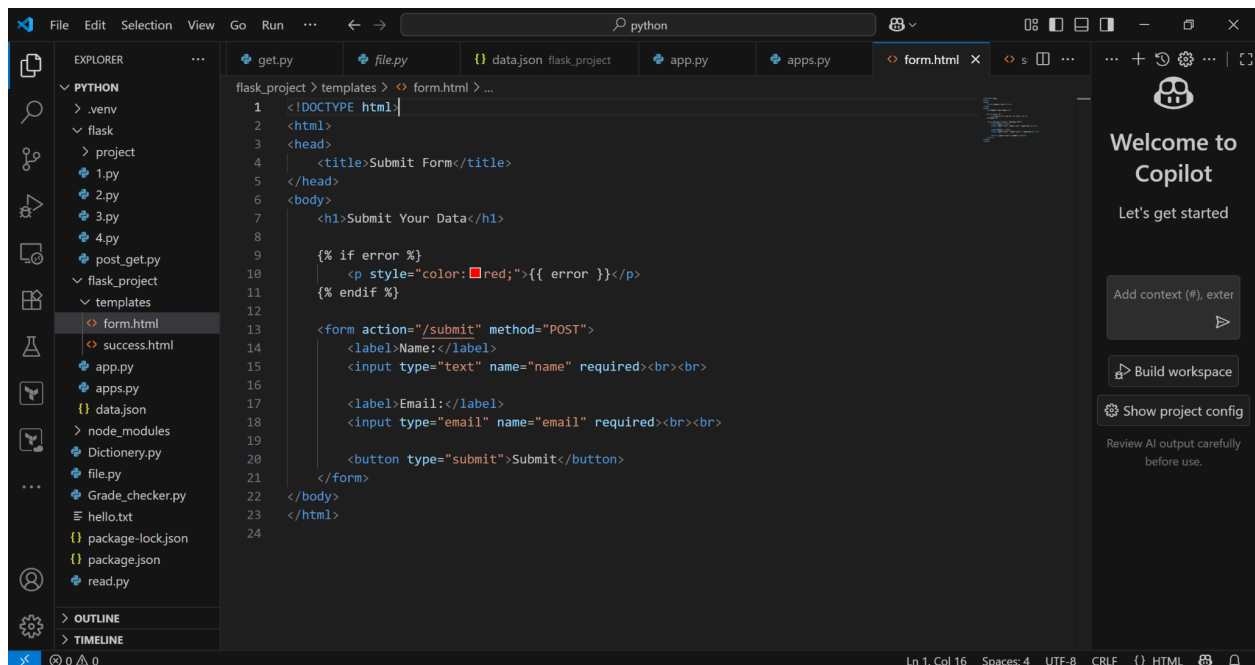


```
flask_project > apps.py > success
1 from flask import Flask, render_template, request, redirect, url_for
2 from pymongo import MongoClient
3 import os
4 app = Flask(__name__)
5 MONGODB_URI = "mongodb+srv://vikramsingh91870:GctbJt1H2Kp0lViv@cluster0.2psj6fk.mongodb.net/"
6 client = MongoClient(MONGODB_URI)
7 db = client.formData
8 collection = db.submissions
9 @app.route('/', methods=['GET'])
10 def index():
11     return render_template('form.html', error=None)
12 @app.route('/submit', methods=['POST'])
13 def submit():
14     name = request.form.get('name')
15     email = request.form.get('email')
16     if not name or not email:
```

PS C:\python> & C:\python\venv\Scripts\Activate.ps1
(.venv) PS C:\python> & C:\python\venv\Scripts\python.exe c:/python/flask_project/apps.py
* Serving Flask app 'apps'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 885-638-923

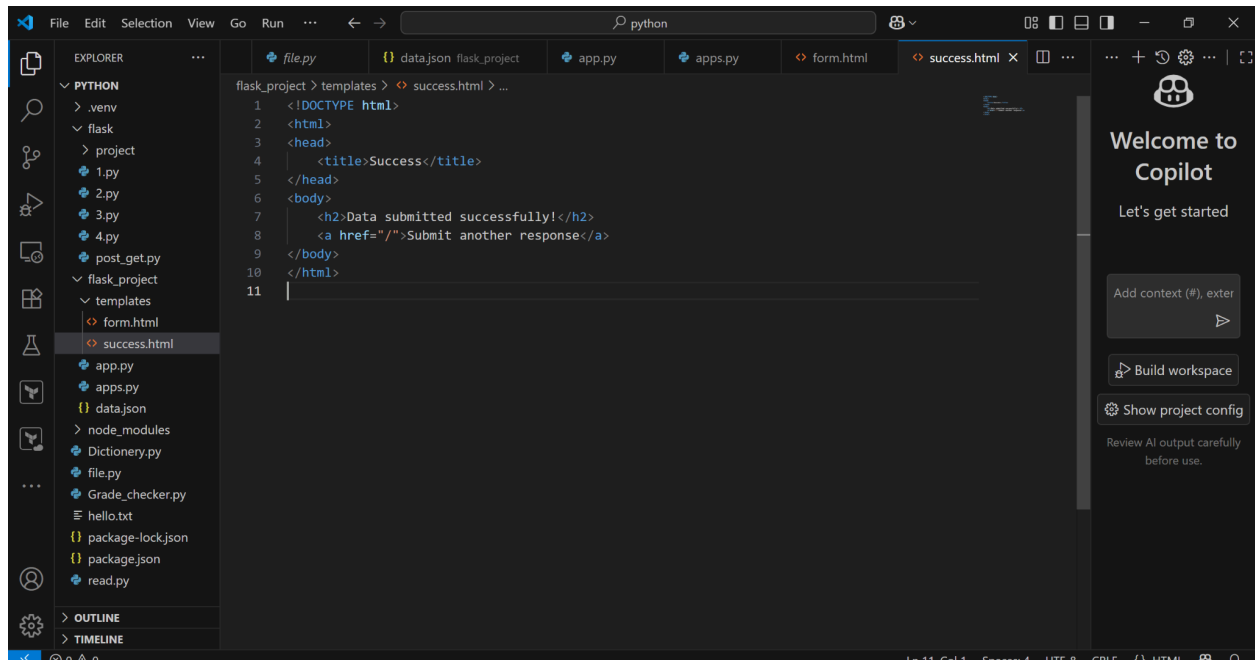
FOR FURTHER PROCESS WE HAVE TO WRITE DIFFERENT HTML CODE IN SAME FOLDER

HTML CODE FOR THE PROGRAM FOR MAKING A FORM IN THE FRONTEND:



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Submit Form</title>
5 </head>
6 <body>
7     <h1>Submit Your Data</h1>
8
9     {% if error %}
10         <p style="color:red;">{{ error }}</p>
11     {% endif %}
12
13     <form action="/submit" method="POST">
14         <label>Name:</label>
15         <input type="text" name="name" required><br>
16
17         <label>Email:</label>
18         <input type="email" name="email" required><br>
19
20         <button type="submit">Submit</button>
21     </form>
22 </body>
23 </html>
```

HTML CODE FOR THE PROGRAM FOR SAYING THAT YOUR DATA IS SUBMITTED SUCCESSFULLY:

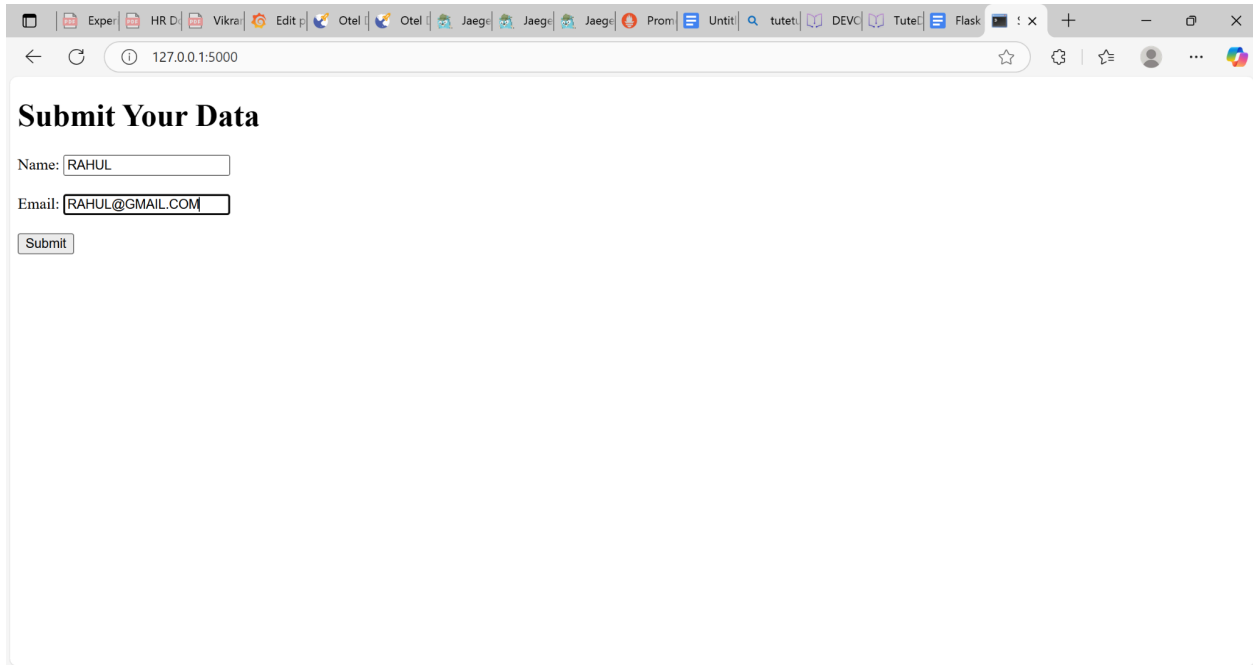


The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays a file tree for a project named 'flask_project'. The file 'success.html' is selected under the 'templates' directory. The main editor window shows the HTML code for 'success.html' with the following content:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Success</title>
5 </head>
6 <body>
7   <h2>Data submitted successfully!</h2>
8   <a href="/">Submit another response</a>
9 </body>
10 </html>
11
```

The right sidebar displays the 'Welcome to Copilot' message, including options to 'Add context (#), enter', 'Build workspace', and 'Show project config'.

THE OUTPUT OF THE PROGRAM IN THE URL:



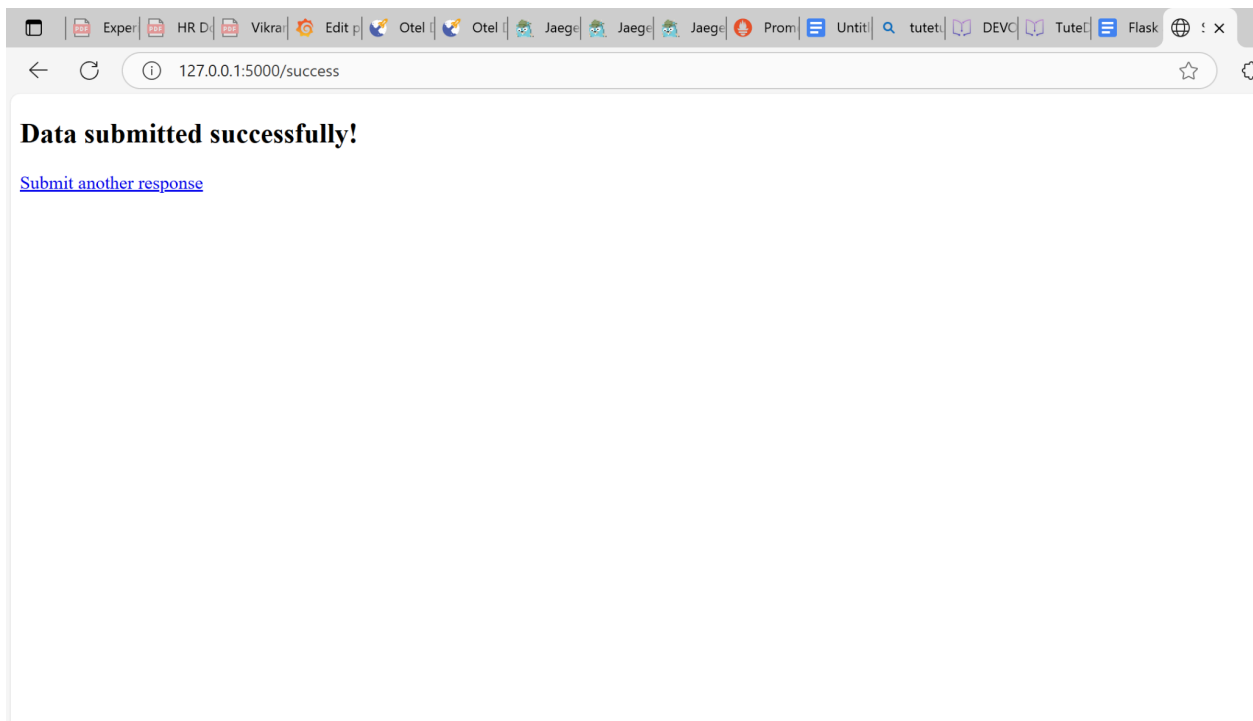
A screenshot of a web browser window. The address bar shows the URL "127.0.0.1:5000". The page title is "Submit Your Data". Below the title, there are two input fields: "Name:" with the value "RAHUL" and "Email:" with the value "RAHUL@GMAIL.COM". Below these fields is a "Submit" button.

Submit Your Data

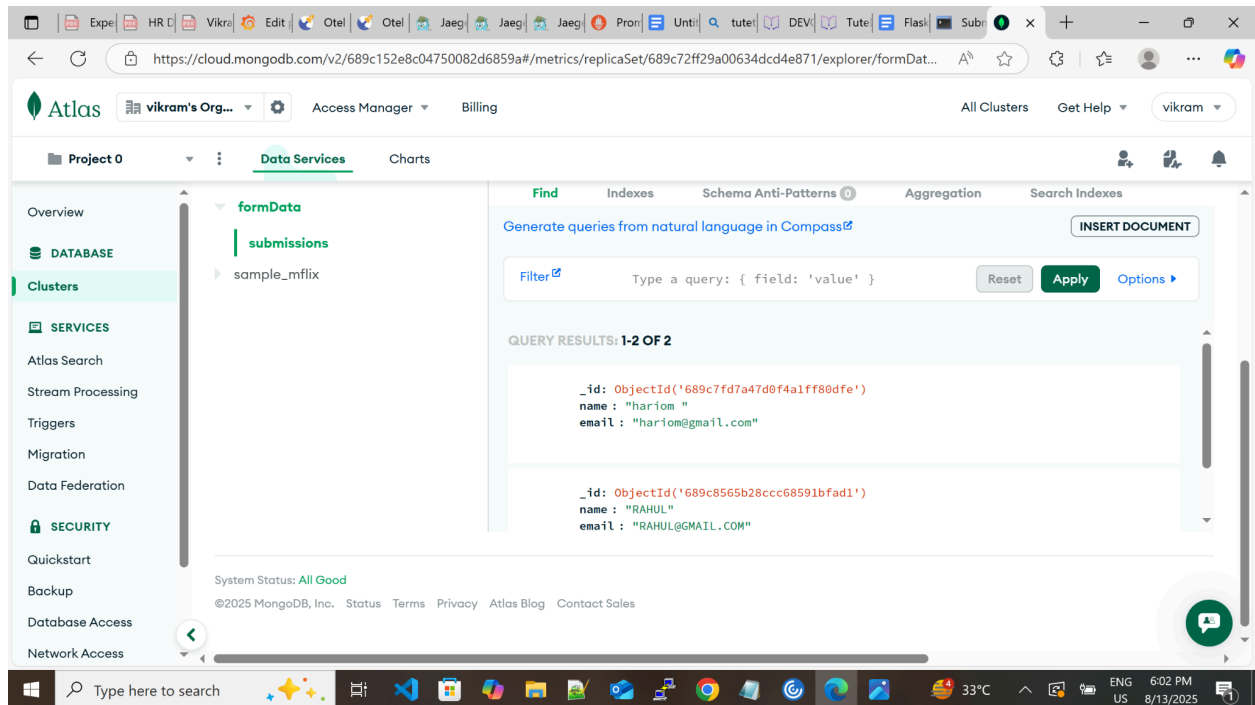
Name:

Email:

AFTER GIVING THE DATA IN THE FIELD IT SHOWS:



NOW CHECK THIS DATA IN THE MONGODB ATLAS:



YES THE DATA IS SAVED IN THE MONGODB ATLAS.

EXPLANATION OF THE PROGRAM apps.py :

◆ 1. Importing Libraries

```
from flask import Flask, render_template, request, redirect, url_for
from pymongo import MongoClient
import os
```

- **Flask**: Micro web framework for Python
- **render_template**: Renders HTML pages (`form.html`, `success.html`)
- **request**: Access form data sent via POST
- **redirect** & **url_for**: For navigating to the success page after form submission

- `MongoClient`: Connects to MongoDB Atlas
- `os`: Used to access environment variables (optional in your current code)

◆ 2. Flask App Initialization

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

Creates a Flask web app instance.

◆ 3. MongoDB Atlas Setup


```
MONGODB_URI =  
"mongodb+srv://vikramsingh91870:GctbJt1H2Kp0lV1v@cluster0.2psj6fk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
```

```
client = MongoClient(MONGODB_URI)
```

```
db = client.formData
```

```
collection = db.submissions
```

- Connects to MongoDB Atlas using your connection string.
- `formData`: The database name (it will be created automatically if it doesn't exist).
- `submissions`: The collection where user data will be stored.

 **Security Tip:** Never expose your MongoDB URI (especially username/password) in public code. Use environment variables in production.

◆ 4. Route: / (GET Request)

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

```
def index():
```

```
    return render_template('form.html', error=None)
```

- Displays the form by rendering `form.html`
- `error=None`: Initially, no error message

◆ 5. Route: `/submit` (POST Request)

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

```
def submit():
```

```
    name = request.form.get('name')
```

```
    email = request.form.get('email')
```

- Collects form data using `request.form.get()`

Form Validation

```
    if not name or not email:
```

```
        return render_template('form.html', error="Please fill all  
fields")
```

- If any field is empty, it returns the same form with an error message

Data Insertion to MongoDB

```
    try:
```

```
        collection.insert_one({"name": name, "email": email})
```

```
        return redirect(url_for('success'))
```

- Inserts a new document into MongoDB with the submitted `name` and `email`
- Redirects to the `/success` route if successful

Error Handling

```
except Exception as e:
```

```
    return render_template('form.html', error=str(e))
```

- If there's an error during insertion, it catches it and displays the error on the same form page

◆ 6. Route: **/success**

```
@app.route('/success')
```

```
def success():
```

```
    return render_template('success.html')
```

- After successful submission, shows a simple success message using `success.html`

◆ 7. Run the App

```
if __name__ == '__main__':
```

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

- Starts the Flask development server on <http://localhost:5000>

EXPLANATION OF THE HTML CODE FORM.HTML:



form.html – User Input Form Template

This is the frontend page served when users visit the root URL (`/`). It contains a simple HTML form where users can submit their name and email.



Code Breakdown

```
<!DOCTYPE html>

<html>

<head>

    <title>Submit Form</title>

</head>

<body>
```

- Standard HTML5 document setup.
 - `<title>` sets the browser tab name.
-

Heading and Error Display

```
<h1>Submit Your Data</h1>

{% if error %}

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

{% endif %}
```

- `{% if error %} ... {% endif %}` is a **Jinja2 template tag** (used in Flask templates).
- It checks if the `error` variable is passed from the Flask backend:
 - If yes → displays the error message in **red**

- If no → nothing is shown

This lets the same form page handle and display validation or database errors.

Form Element

```
<form action="/submit" method="POST">
```

- Submits form data to the `/submit` endpoint on the backend using `POST` (secure for sending data).
-

Input Fields

```
<label>Name:</label>
```

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

- Input field for the user's **name**
- `required` makes sure it cannot be empty

```
<label>Email:</label>
```

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

- Input field for the **email**
 - `type="email"` ensures basic email validation
-

Submit Button

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

- When clicked, the form is sent to `/submit`, triggering the Flask route logic you wrote in `app.py`
-

How It Connects to Flask

This template is rendered by the `/` route in `app.py`:

```
return render_template('form.html', error=None)
```

-

If there's an error (like missing fields or a MongoDB error), it gets passed back to this template:

```
return render_template('form.html', error=str(e))
```

-

This keeps the user on the same page and shows helpful feedback.

EXPLANATION OF THE HTML CODE SUCCESS.HTML:

success.html – Success Confirmation Page

This HTML template is displayed **after a user successfully submits** the form data to the backend and the data is inserted into **MongoDB Atlas**.



Code Breakdown

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <title>Success</title>
```

```
</head>
```

```
<body>
```

- Standard HTML5 document.
 - The title will appear on the browser tab as "Success".
-



Success Message

```
<h2>Data submitted successfully!</h2>
```


- Displays a clear success confirmation to the user.
-

Return Link

```
<a href="/">Submit another response</a>
```

- A hyperlink that takes the user back to the main form page (/route), allowing them to submit a new entry without refreshing or restarting the app.
-

How It Connects to Flask

In your `app.py`, you redirect to this template after successful form submission:

```
return redirect(url_for('success'))
```

The `success` function in Flask renders this template:

```
@app.route('/success')
```

```
def success():
```

```
    return render_template('success.html')
```