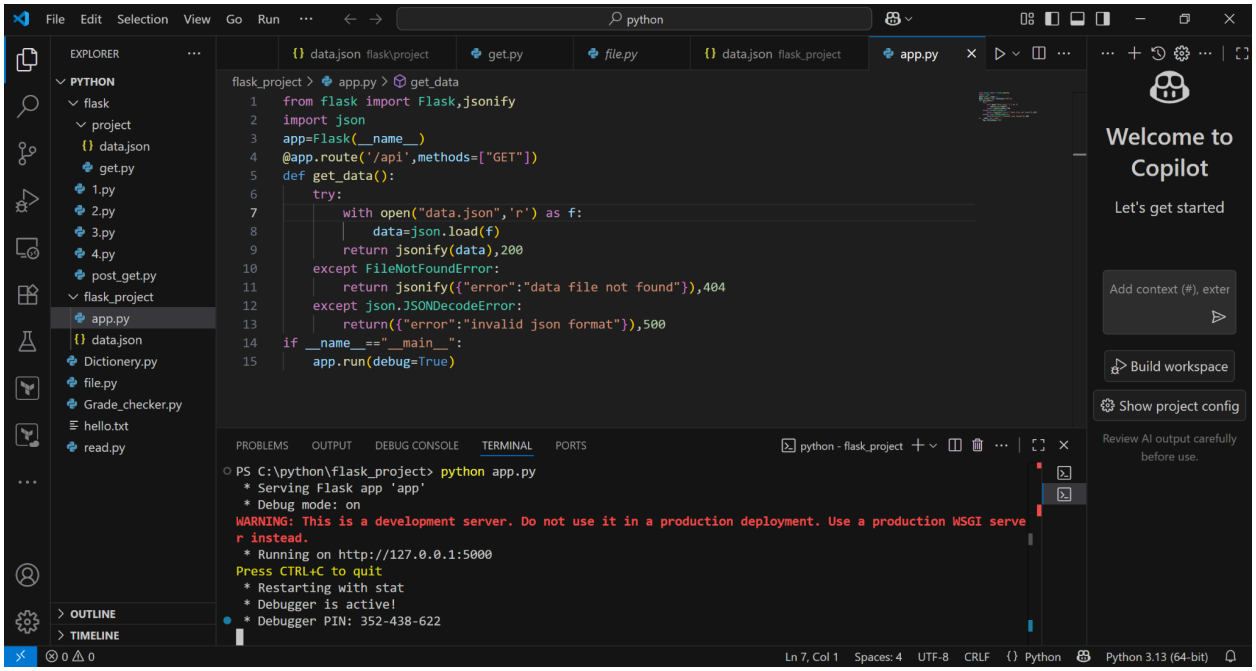
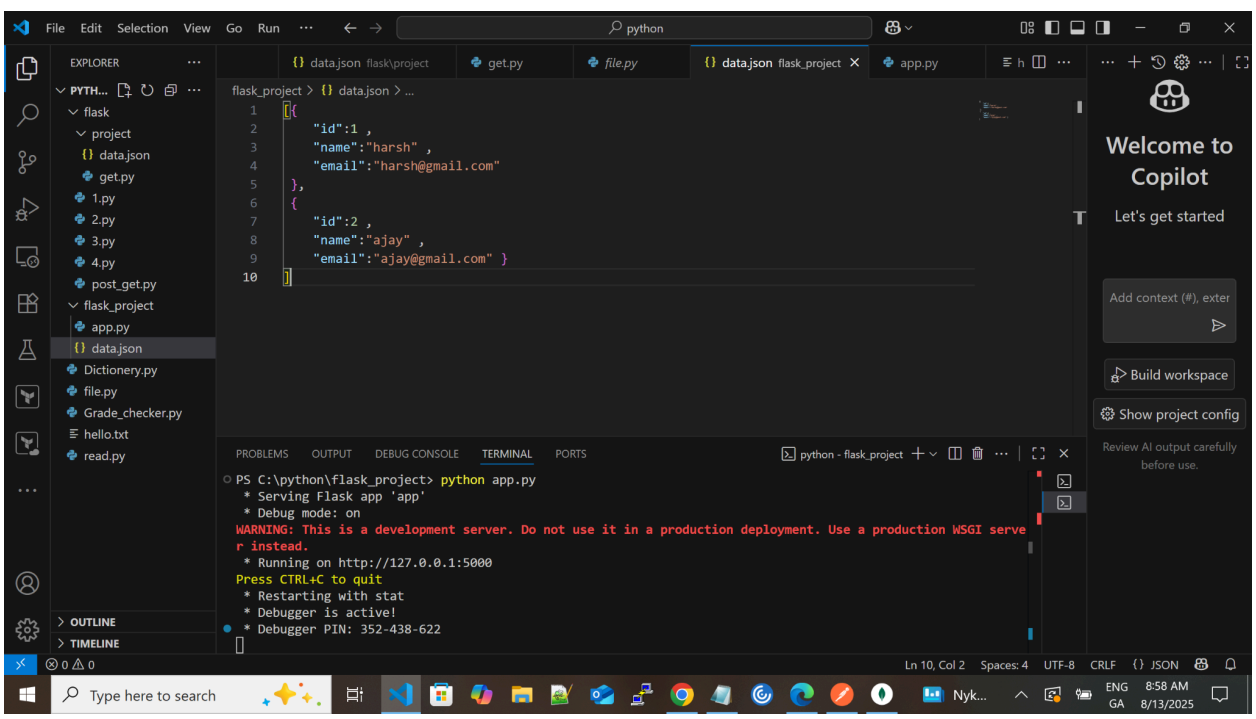
1. Create a Flask application with an `/api` route. When this route is accessed, it should return a JSON list. The data should be stored in a backend file, read from it, and sent as a response.

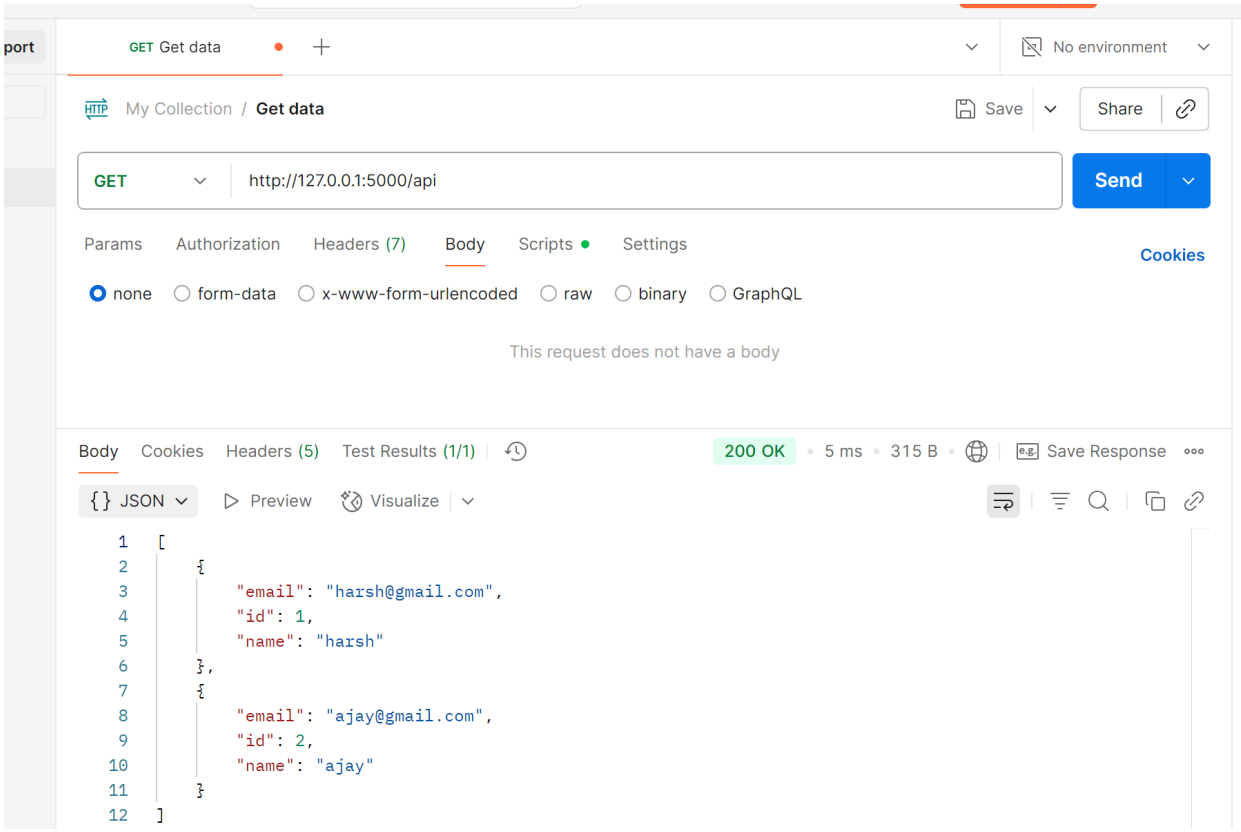
CODE FOR THE PROGRAM:



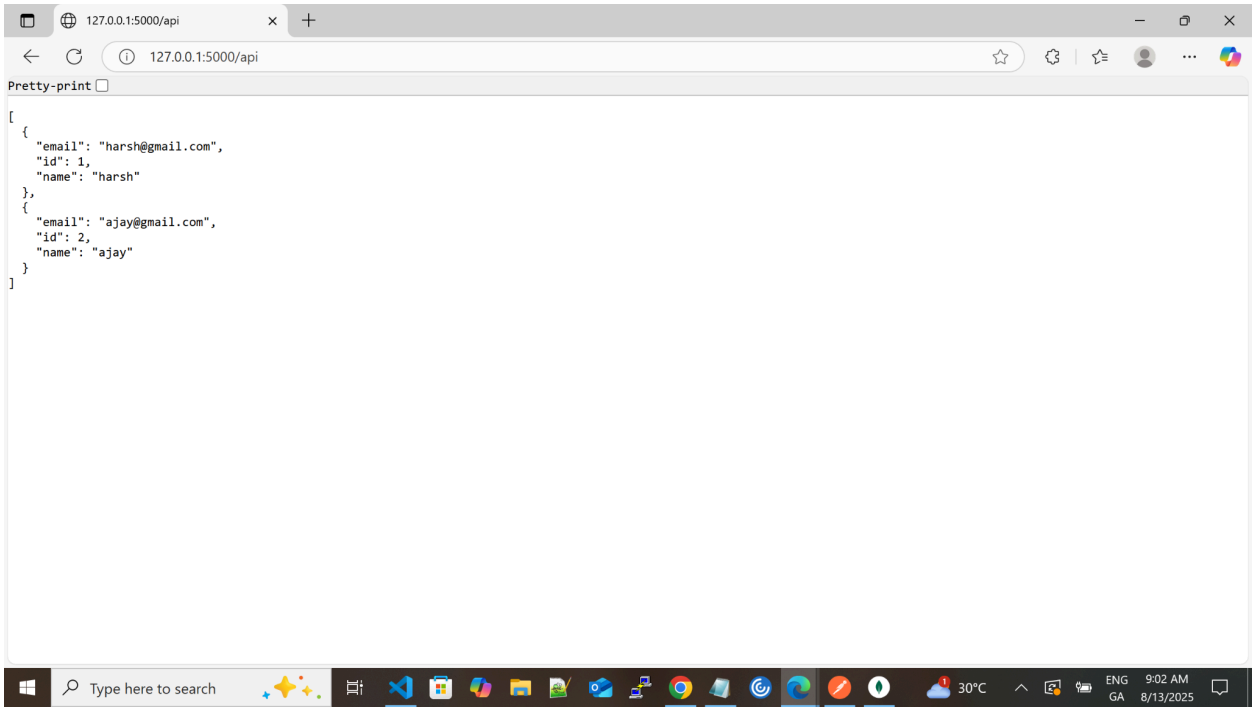
DATA FOR THE PROGRAM SO THE PROGRAM FETCH IT :-



OUTPUT FOR THE PROGRAM BY POSTMAN:



OUTPUT FOR THE PROGRAM BY THE URL:



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.

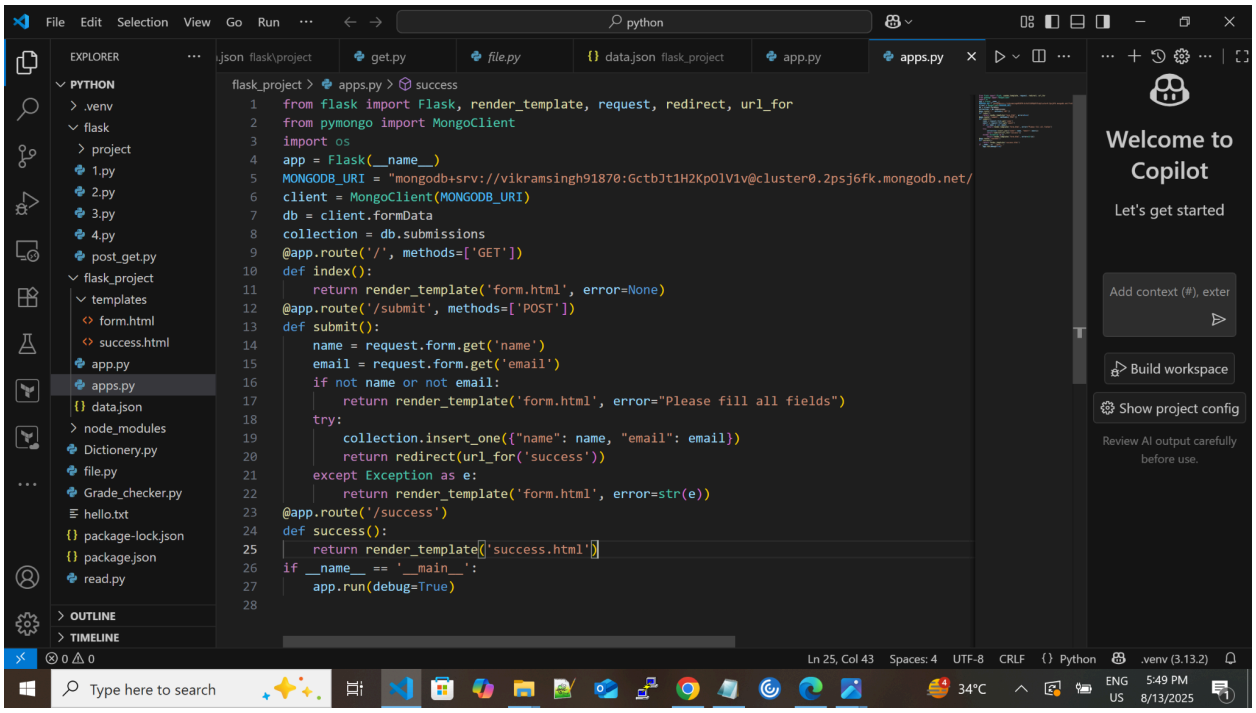
```
• if __name__ == "__main__":

    app.run(debug=True)
```

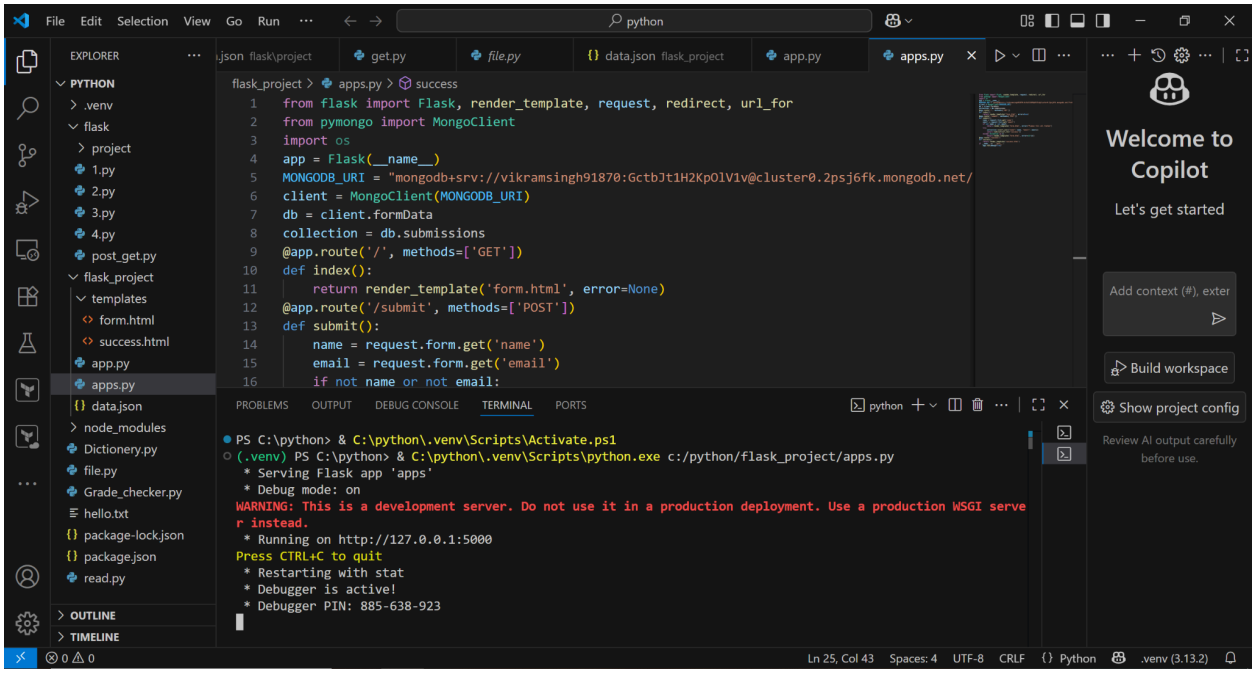
This starts the Flask app in debug mode, so you can see error logs and live reload during development.

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:

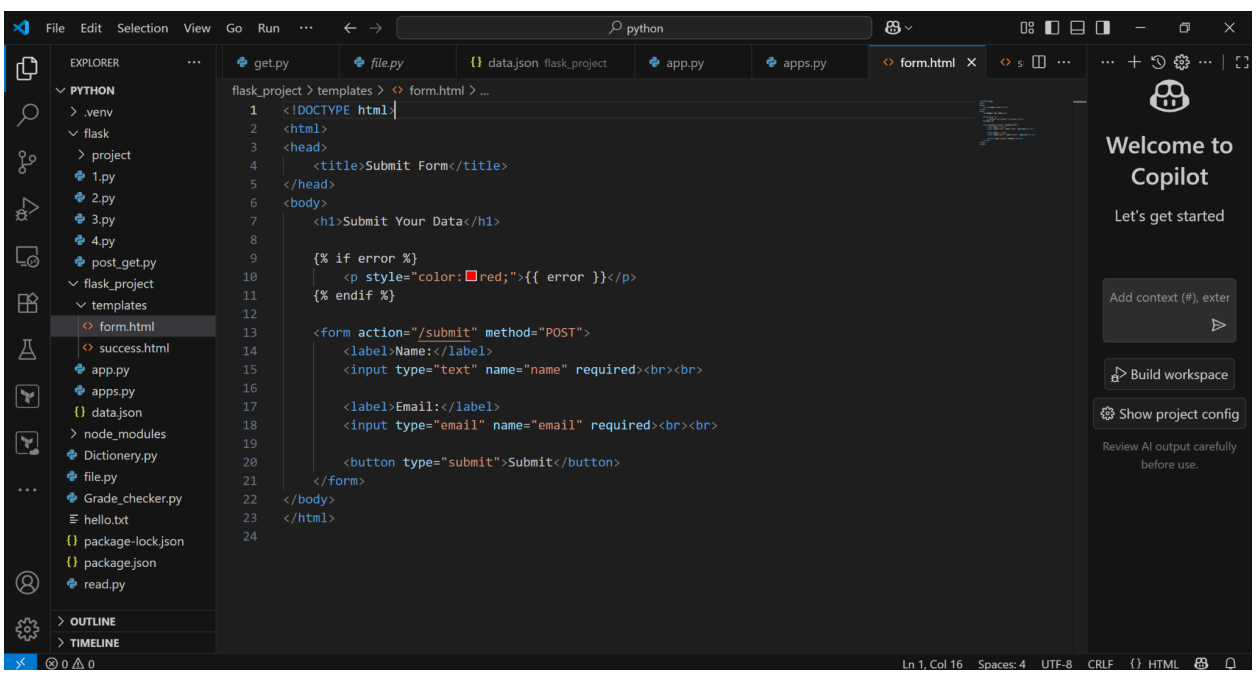


OUTPUT OF THE PROGRAM IN THE VSCODE TERMINAL:

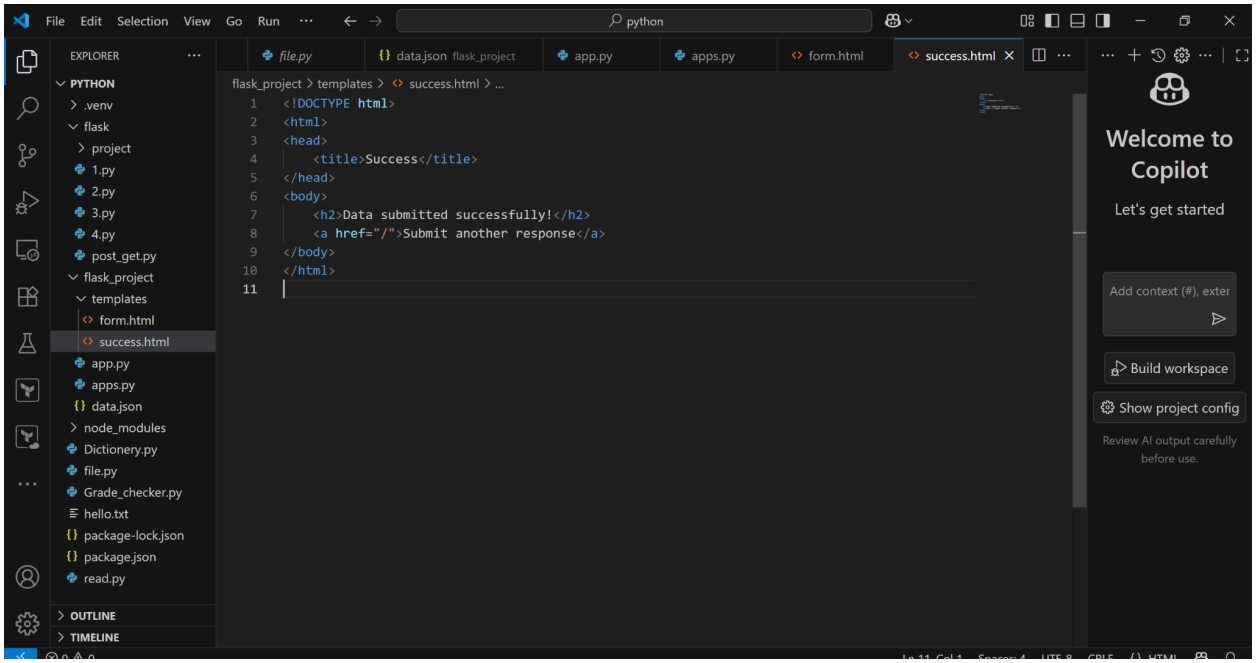


FOR FURTHER PROCESS WE HAVE TO WRITE DIFFERENT HTML CODE IN SAME FOLDER

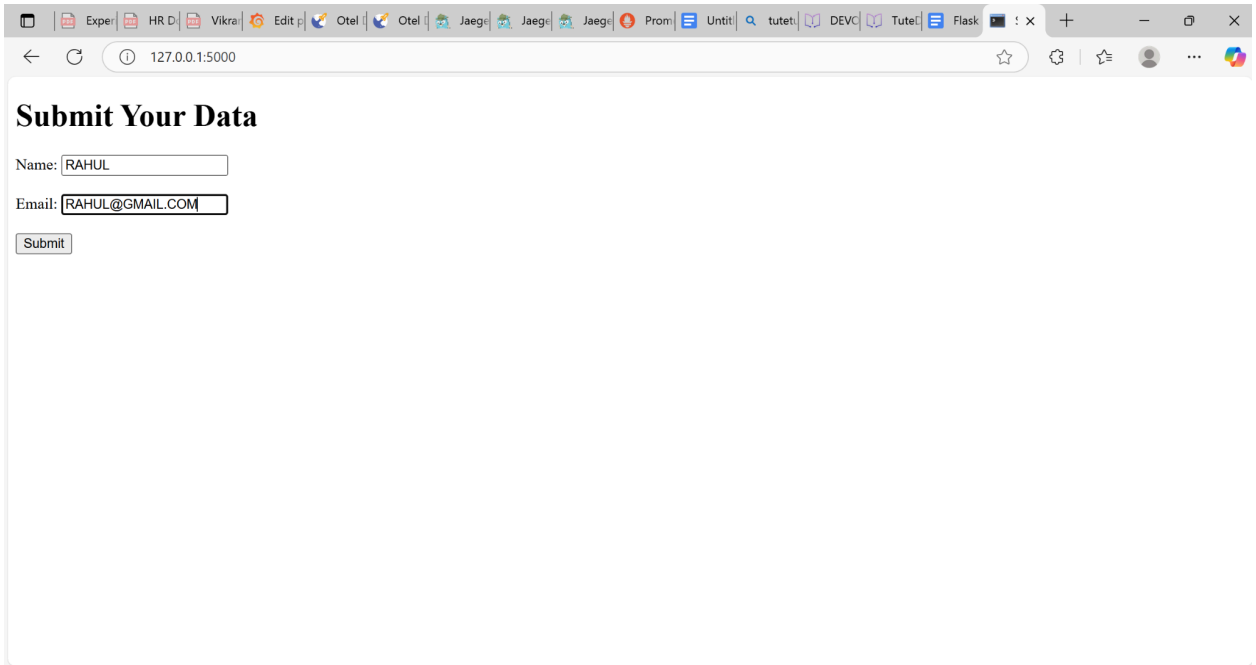
HTML CODE FOR THE PROGRAM FOR MAKING A FORM IN THE FRONTEND:



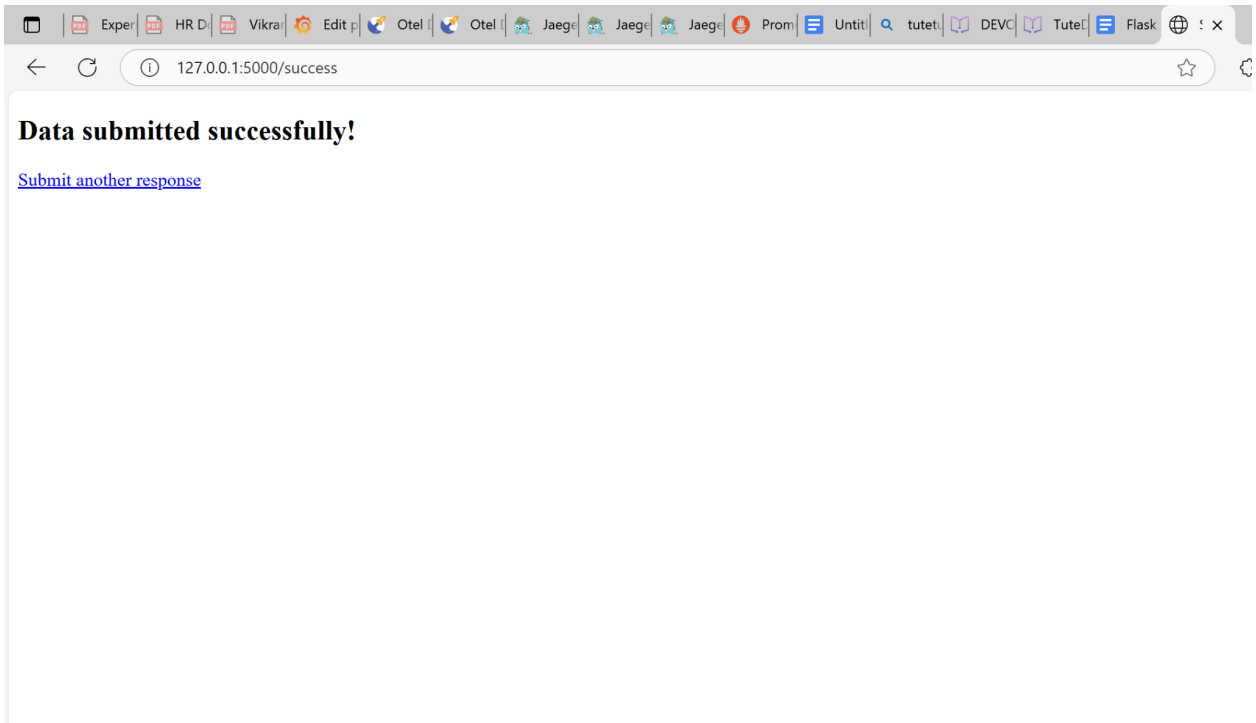
HTML CODE FOR THE PROGRAM FOR SAYING THAT YOUR DATA IS SUBMITTED SUCCESSFULLY:



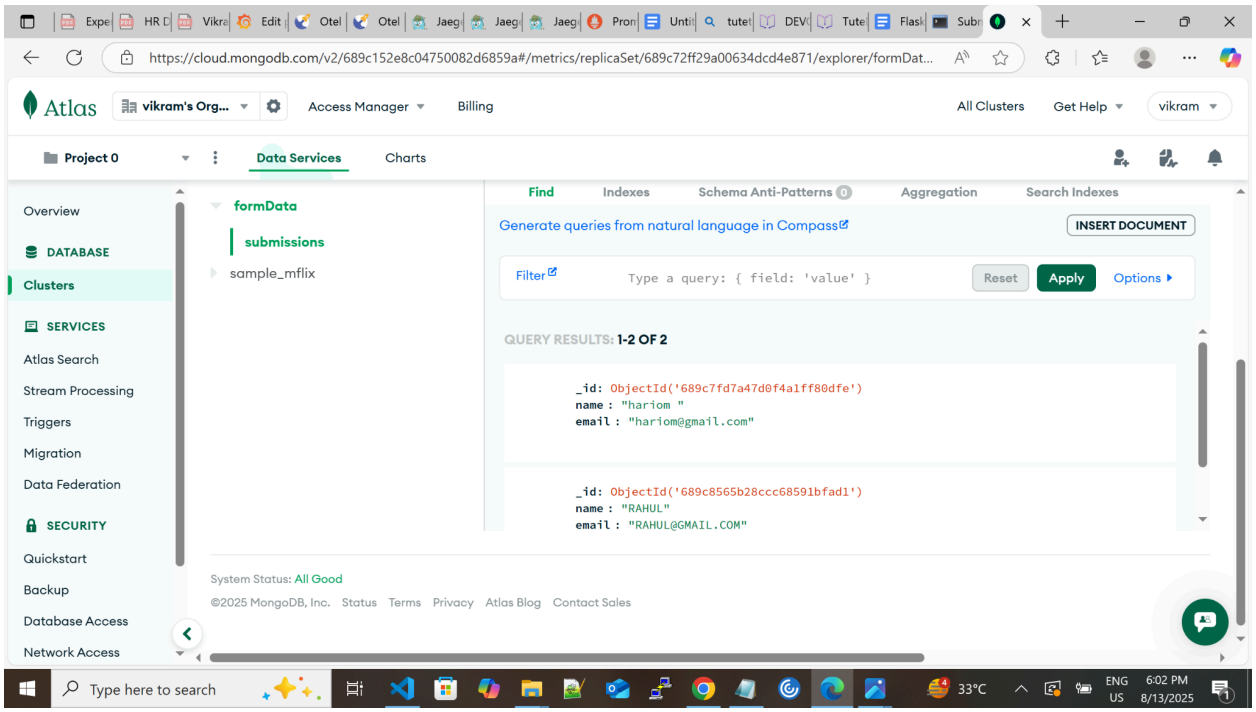
THE OUTPUT OF THE PROGRAM IN THE URL:



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:GctbJt1H2KpOlV1v@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')
```