EXPERIMENT NO. 3

Name of Student	Rohan Lalchandani
Class Roll No	25
D.O.P.	20/2/2024
D.O.S.	27/2/2024
Sign and Grade	

AIM : **To develop a basic Flask application with multiple routes and demonstrate the handling of GET and POST requests.**

PROBLEM STATEMENT :
Design a Flask web application with the following features:

1. A homepage (/) that provides a welcome message and a link to a contact form.
   a. Create routes for the homepage (/), contact form (/contact), and thank-you page (/thank_you).
2. A contact page (/contact) where users can fill out a form with their name and email.
3. Handle the form submission using the POST method and display the submitted data on a thank-you page (/thank_you).
   a. On the contact page, create a form to accept user details (name and email).
   b. Use the POST method to handle form submission and pass data to the thank-you page
4. Demonstrate the use of GET requests by showing a dynamic welcome message on the homepage when the user accesses it with a query parameter, e.g., /welcome?name=<user_name>.
   a. On the homepage (/), use a query parameter (name) to display a personalized welcome message.

THEORY:

1. **List some of the core features of Flask**

   Flask is a lightweight and flexible web framework for Python, often described as "micro" because it provides the essentials to get a web application up and running without imposing unnecessary restrictions. Some core features of Flask include its minimalist design, which allows developers to add only what they need, built-in support for routing, templates, and handling HTTP requests and responses. It uses the Jinja2 templating engine to generate dynamic HTML content and can be extended with various extensions like database support, authentication, and form handling.

2. **Why do we use Flask(__name__) in Flask?**

   The reason we use Flask(__name__) is to create an instance of the Flask class. __name__ tells Flask if the script is being run directly or if it's being imported as a module into another script. When running directly, Flask knows to start the app, and when imported, it avoids running the app code unnecessarily.

3. **What is Template (Template Inheritance) in Flask?**

   In Flask, a template is an HTML file with placeholders for dynamic content, usually populated with data passed from the Flask view functions. Template inheritance allows you to create a base template (with common elements like headers or footers) and extend it in other templates. This promotes reusability and helps keep your code DRY (Don't Repeat Yourself).

4. **What methods of HTTP are implemented in Flask?**

   Flask supports several HTTP methods like GET (to retrieve data), POST (to send data), PUT (to update existing data), DELETE (to remove data), and OPTIONS (to describe communication options for the resource). These methods allow Flask to handle different types of interactions with the server.

5. **What is difference between Flask and Django framework**

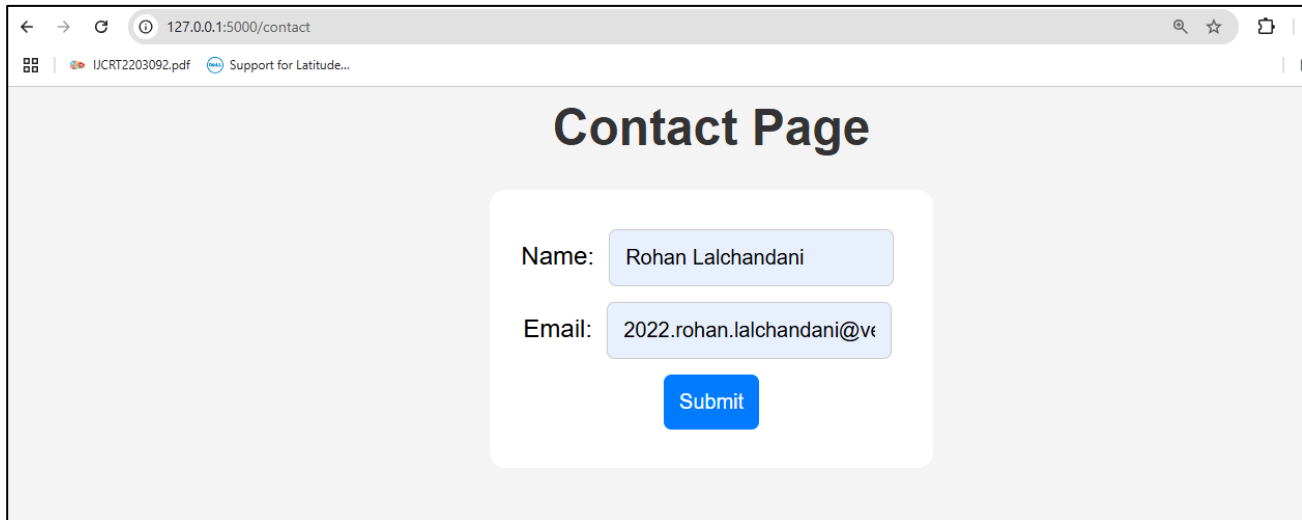| Feature	Flask	Django
Type	Lightweight, micro-framework	Full-stack, "batteries-included" framework
Flexibility	Highly flexible	Less flexible
Best For	Small to medium-sized projects	Larger, more complex projects
Database Integration	No built-in ORM	Built-in ORM
Community	Growing community	Larger community

GITHUB LINK –
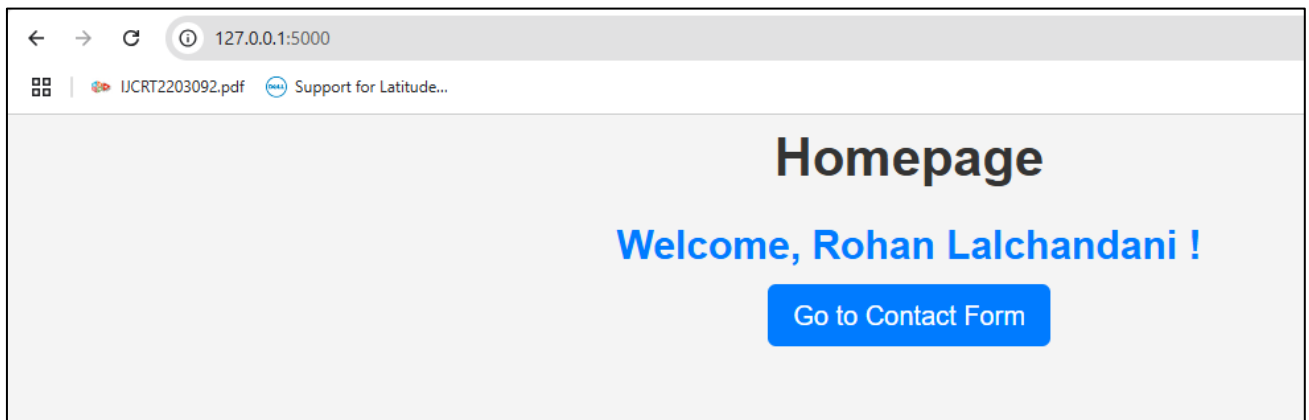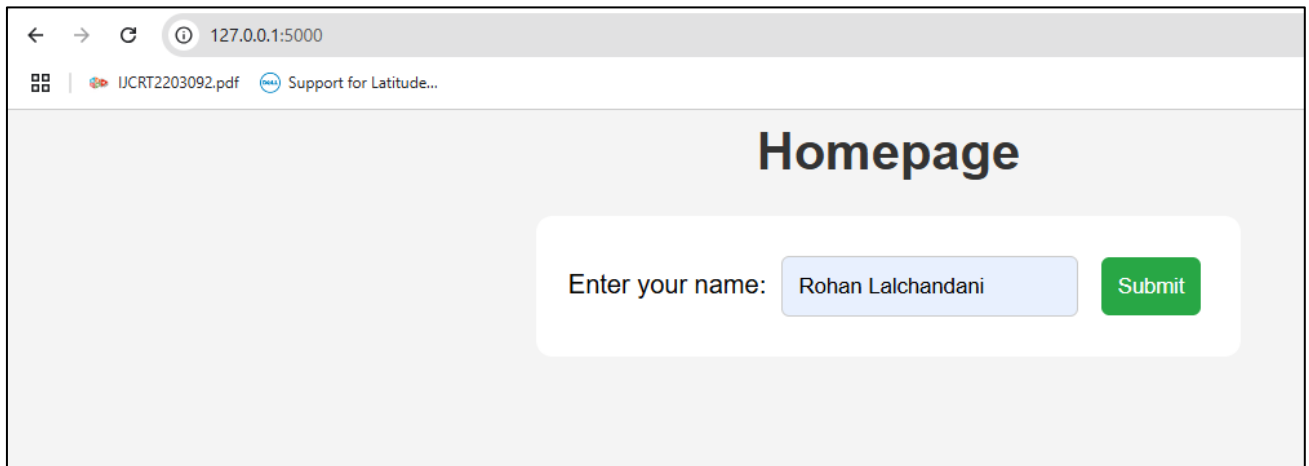
OUTPUT :

Homepage:

This screenshot displays the output of the homepage when accessed by taking query parameter of the name of person. The welcome message, **"Welcome, {"name"}!"**, is shown, along with a hyperlink labeled **"Go to the contact form"** that redirects users to the contact page. This demonstrates the basic functionality of the homepage and how it serves dynamic content by displaying user' name.
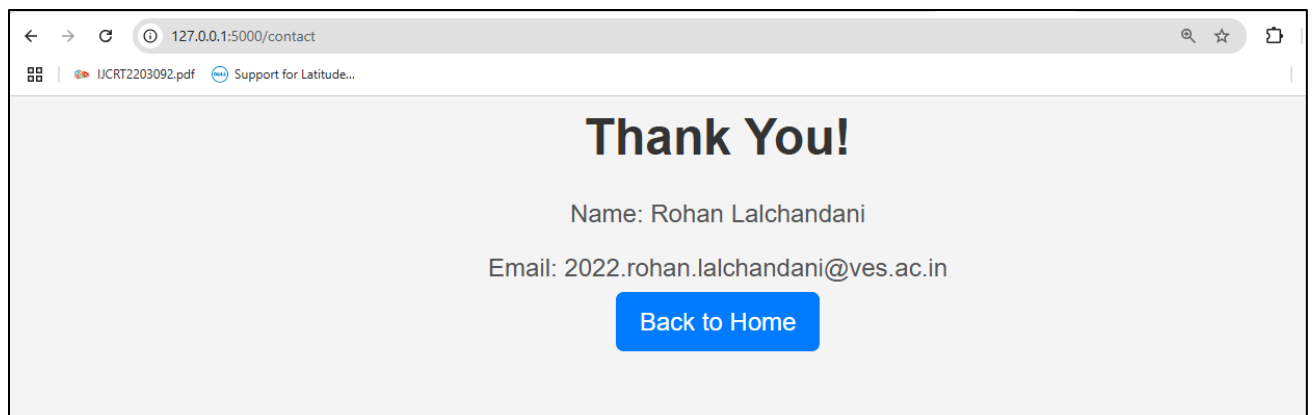
b) Contact Page (`/contact`)

This screenshot shows the contact form page where users can enter their **name** and **email**. The form includes labeled input fields and a submit button. When the user enters their details and submits the form, the data is processed via the **POST** method and redirected to the **thank-you page**. This highlights Flask's ability to handle user input and form submission securely.

c) After form submission:

This screenshot represents the output after successfully submitting the contact form. The **thank-you page** displays a message, **"Thank You for Contacting Us!"**, along with the user's submitted **name and email**. Additionally, a **"Back to homepage"** link is provided for navigation. This demonstrates how Flask processes form data, passes it via the URL query string, and renders it dynamically on the response page.

CONCLUSION

In this experiment, a **Flask web application** was successfully developed to demonstrate the handling of **GET and POST requests** with multiple routes. The application included:

- A **homepage (`/`)** that displayed a static or dynamic welcome message based on query parameters.
- A **contact page (`/contact`)** where users could submit their name and email via a form.
- A **thank-you page (`/thank_you`)** that dynamically displayed the submitted form data.

Through this implementation, key Flask features such as **routing, request handling, form submission, template rendering, and URL parameter passing** were explored. This experiment effectively demonstrated Flask's capability to create **interactive and dynamic web applications** with minimal code.