Name - Gaurang A Raorane Div - D15A Roll no - 49 Batch - C

**Experiment - 5**

Aim - To study Flask.

Theory - List some of the core features of Flask

1. Why do we use Flask(\_\_name\_\_) in Flask?

Flask(\_\_name\_\_) used to create an instance of the Flask application. But why exactly is \_\_name\_\_ used here?

There are two main reasons:

1. Finding Resources: Flask utilizes \_\_name\_\_ to determine the location of your application's resources. This includes things like templates, static files (images, CSS, etc.), and potentially configuration files. By knowing the application's name (which \_\_name\_\_ provides), Flask can search the appropriate directories relative to the application's location.
2. Improved Debugging: Some Flask extensions can use the \_\_name\_\_ to enhance debugging information. This can be helpful in pinpointing issues within your application.

Here's a breakdown of how \_\_name\_\_ works:

* When Executed Directly: If you run the Python file containing your Flask application directly, \_\_name\_\_ will be set to "\_\_main\_\_". This allows Flask to locate resources in the same directory as the script.
* When Imported as a Module: If you import your Flask application into another Python module, \_\_name\_\_ will be set to the actual import name of the module. This ensures Flask searches for resources in the correct location relative to the imported module.

By using \_\_name\_\_, Flask gains flexibility in how your application is run and ensures it can find the necessary resources regardless of the execution method.

In essence, \_\_name\_\_ acts like a handy reference point for Flask to navigate your application's structure and locate the building blocks it needs to function.

1. What is Template (Template Inheritance) in Flask?

In Flask, templates are HTML files with special placeholders that allow you to incorporate dynamic content into your web pages. Template inheritance is a powerful feature that builds upon this concept to promote code reuse and consistency across your application's web pages.

Here's a breakdown of template inheritance:

Benefits:

* Reduced Code Duplication: Imagine your website has a common layout for the header, navigation bar, and footer that appears on every page. Without template inheritance, you'd rewrite this code in each individual template. Template inheritance allows you to define this common layout in a single base template and then have child templates inherit and override specific sections.
* Consistent Look and Feel: By inheriting from a base template, child templates automatically share the same base layout. This ensures a consistent user experience throughout your website.

How it Works:

1. Base Template: Create a base template (often named base.html) that defines the overall structure of your web pages. This typically includes the HTML document structure, common elements like headers and footers, and placeholder blocks for content that will vary across pages.
2. Block Definition: Within the base template, use the {% block %} and {% endblock %} tags to define sections that child templates can override. These blocks act like designated areas where child templates can insert their unique content.
3. Child Templates: Create individual HTML templates (like home.html or about.html) for specific pages in your application. These child templates inherit from the base template using the {% extends "base.html" %} statement at the beginning.
4. Overriding Blocks: Child templates can override the content within blocks defined in the base template. They achieve this by using the same block name and placing their specific content between {% block block\_name %} ... {% endblock %} tags.
5. Accessing Base Content: Child templates can still access and render content defined directly in the base template using the {{ super() }} Jinja template expression. This is useful for including common elements from the base template that aren't meant to be overridden.

By following these steps, you can create a modular and maintainable template structure for your Flask application. Changes to the base template are automatically reflected in all child templates that inherit from it, promoting code efficiency and a consistent user experience.

1. What methods of HTTP are implemented in Flask.

Flask itself doesn't implement the HTTP methods directly, but it provides mechanisms to handle them within your web application. Here's a breakdown of how Flask works with HTTP methods:

Supported Methods:

Flask applications can handle a variety of HTTP methods, including the most common ones:

* GET: Used to retrieve data from the server. This is the default method for route definitions in Flask.
* POST: Used to submit data to the server, often through HTML forms or file uploads.
* PUT: Used to update an existing resource on the server with a complete representation of the new data.
* DELETE: Used to delete a resource from the server.
* HEAD: Similar to GET, but only retrieves the header information, not the actual content.
* OPTIONS: Used to determine the communication options supported by the server for a specific resource.

Specifying Methods in Routes:

You define routes in Flask using the @app.route() decorator. This decorator can optionally take a second argument, methods, which is a list of HTTP methods your route can handle

1. What is difference between Flask and Django framework

Flask and Django are both popular Python frameworks for web development, but they have key differences in their approach and suitability for projects:

Philosophy:

* Flask: Microframework - Lightweight and minimalistic, offering flexibility and a "build-from-scratch" approach. You have more control over the application structure and libraries used.
* Django: Full-featured framework - Batteries-included approach with a collection of built-in features to jumpstart development. It enforces a specific project structure and conventions.

Complexity:

* Flask: Simpler to learn due to its lightweight nature. Ideal for smaller projects or those requiring more customization.
* Django: Has a steeper learning curve due to its comprehensive features and structure. More suitable for complex, large-scale applications.

Features:

* Flask: Offers core functionalities like routing, templating, and request handling. Relies on extensions for additional features like database interaction or user authentication.
* Django: Includes a wider range of built-in features like an ORM (Object-Relational Mapper) for database access, admin panel, user authentication, and more.

Development Speed:

* Flask: Can be faster for development of simple applications due to its flexibility and ease of customization.
* Django: Encourages rapid development for complex applications with its built-in features and streamlined approach.

Security:

* Flask: Requires more manual effort to implement security measures as it doesn't have built-in security features.
* Django: Offers built-in security features like user authentication, CSRF protection, and XSS prevention.

Here's a table summarizing the key differences:

| Feature | Flask | Django |
| --- | --- | --- |
| Philosophy | Microframework (build-from-scratch) | Full-featured framework (batteries-included) |
| Complexity | Simpler | Steeper learning curve |
| Features | Core functionalities + extensions | Built-in features (ORM, admin panel, etc.) |
| Development Speed | Faster for simple applications | Faster for complex applications |
| Security | Requires manual implementation | Built-in security features |

drive\_spreadsheetExport to Sheets

In essence, choosing between Flask and Django depends on your project requirements:

* Use Flask for: Simple web apps, prototypes, projects requiring high customization, or when you prefer more control over the application structure.
* Use Django for: Complex web applications, rapid development of large-scale projects, or when you need built-in features and security.

Code -

**app.py**

# app.py

from flask import Flask, render\_template, request

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

@app.route('/')

def index():

return render\_template('index.html')

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

def contact():

if request.method == 'POST':

# Handle form submission

name = request.form['name']

email = request.form['email']

message = request.form['message']

# You can add code to send the form data via email or store it in a database

return 'Thank you for your message!'

return render\_template('contact.html')

@app.route('/thankyou')

def thankyou():

return render\_template('thankyou.html')

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

app.run(debug=True)

**Index.html-**

<!-- templates/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My Portfolio</title>

<link rel="stylesheet" href="/static/css/index.css">

</head>

<body>

<header>

<nav>

<a href="/">Home</a>

<a href="/contact">Contact</a>

</nav>

</header>

<main>

<section class="hero">

<img src="https://t4.ftcdn.net/jpg/04/91/45/15/360\_F\_491451522\_ufXPQSzdSxohOBUWiYTK33tUgx2Zwbrk.jpg" alt="Scenery Image">

<div class="hero-text">

<h1>Welcome to My Portfolio</h1>

<p>I am a software developer with experience in Python, JavaScript, and more.</p>

</div>

</section>

</main>

<footer>

<p>&copy; 2023 My Portfolio</p>

</footer>

</body>

</html>

**Contact.html-**

<!-- templates/contact.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contact Me</title>

<link rel="stylesheet" href="static/css/contact.css">

</head>

<body>

<header>

<nav>

<a href="index.html">Home</a>

<a href="contact.html">Contact</a>

</nav>

</header>

<main>

<section class="contact-form">

<h2>Contact Me</h2>

<form method="POST" action="{{ url\_for('contact') }}">

<label for="name">Name:</label>

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

<label for="email">Email:</label>

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

<label for="message">Message:</label>

<textarea id="message" name="message" required></textarea>

<button type="submit">Send Message</button>

</form>

</section>

</main>

<footer>

<p>&copy; 2023 My Portfolio</p>

</footer>

</body>

</html>

**Thankyou.html-**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Thank You</title>

<style>

body {

margin: 0;

padding: 0;

height: 100%;

display: flex;

justify-content: center;

align-items: center;

background: linear-gradient(135deg, #6a11cb, #2575fc);

color: #fff;

}

h1 {

text-align: center;

align-items: center;

}

p {

text-align: center;

font-size: 20px;

color: #f0f0f0;

}

</style>

</head>

<body>

<h1>Thank You,</h1>

<p>We have received your message. We will get back to you shortly.</p>

</body>

</html>

**Index.css-**

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background: linear-gradient(135deg, #6a11cb, #2575fc);

}

header {

background-color: #333;

color: #fff;

padding: 20px;

display: flex;

justify-content: space-between;

align-items: center;

}

nav {

display: flex;

align-items: center;

}

nav a {

color: #fff;

text-decoration: none;

margin-right: 20px;

font-size: 18px;

padding: 10px 20px; /\* Adjust padding for larger buttons \*/

border-radius: 5px; /\* Add rounded corners \*/

transition: background-color 0.3s ease; /\* Smooth transition for hover effect \*/

}

nav a:hover {

background-color: #555; /\* Change background color on hover \*/

}

nav a[href="contact.html"],

nav a[href="#"] {

font-size: 20px; /\* Enlarge the Home and Contact links \*/

}

.hero {

position: relative;

height: 500px;

overflow: hidden;

}

.hero img {

width: 100%;

height: 100%;

object-fit: cover;

}

.hero-text {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

text-align: center;

color: #fff;

background-color: rgba(0, 0, 0, 0.5);

padding: 20px;

}

.contact-form {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #f2f2f2;

border-radius: 5px;

}

.contact-form label {

display: block;

margin-bottom: 5px;

}

.contact-form input,

.contact-form textarea {

width: 100%;

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 3px;

}

.contact-form button {

background-color: #333;

color: #fff;

border: none;

padding: 10px 20px;

border-radius: 3px;

cursor: pointer;

}

footer {

background-color: #333;

color: #fff;

text-align: center;

padding: 10px;

}

.projects,

.skills {

padding: 50px 0;

color: #fff;

}

.projects {

background-color: rgba(0, 0, 0, 0.7);

}

.skills {

background-color: rgba(0, 0, 0, 0.5);

}

h2 {

text-align: center;

margin-bottom: 20px;

}

**Contact.css-**

/\* static/css/contact.css \*/

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

header {

background-color: #333;

color: #fff;

padding: 20px;

display: flex;

justify-content: space-between;

align-items: center;

}

nav a {

color: #fff;

text-decoration: none;

margin-right: 20px;

font-size: 18px;

transition: color 0.3s ease;

}

nav a:hover {

color: #ccc;

}

.contact-form {

max-width: 600px;

margin: 50px auto;

padding: 30px;

background-color: #f2f2f2;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);

text-align: center;

}

.contact-form h2 {

margin-top: 0;

}

.contact-form label {

display: block;

margin-bottom: 10px;

text-align: left;

}

.contact-form input,

.contact-form textarea {

width: 100%;

padding: 15px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

font-size: 16px;

box-sizing: border-box;

}

.contact-form button {

background-color: #4CAF50; /\* Green \*/

color: #fff;

border: none;

padding: 15px 30px;

border-radius: 3px;

cursor: pointer;

font-size: 18px;

transition: background-color 0.3s ease;

}

.contact-form button:hover {

background-color: #45a049;

}

footer {

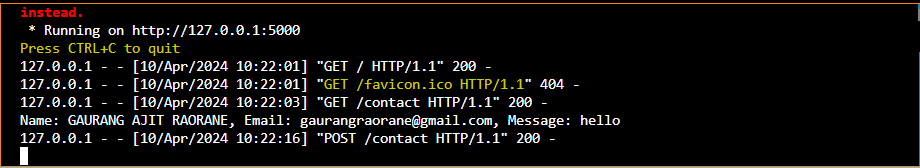
background-color: #333;

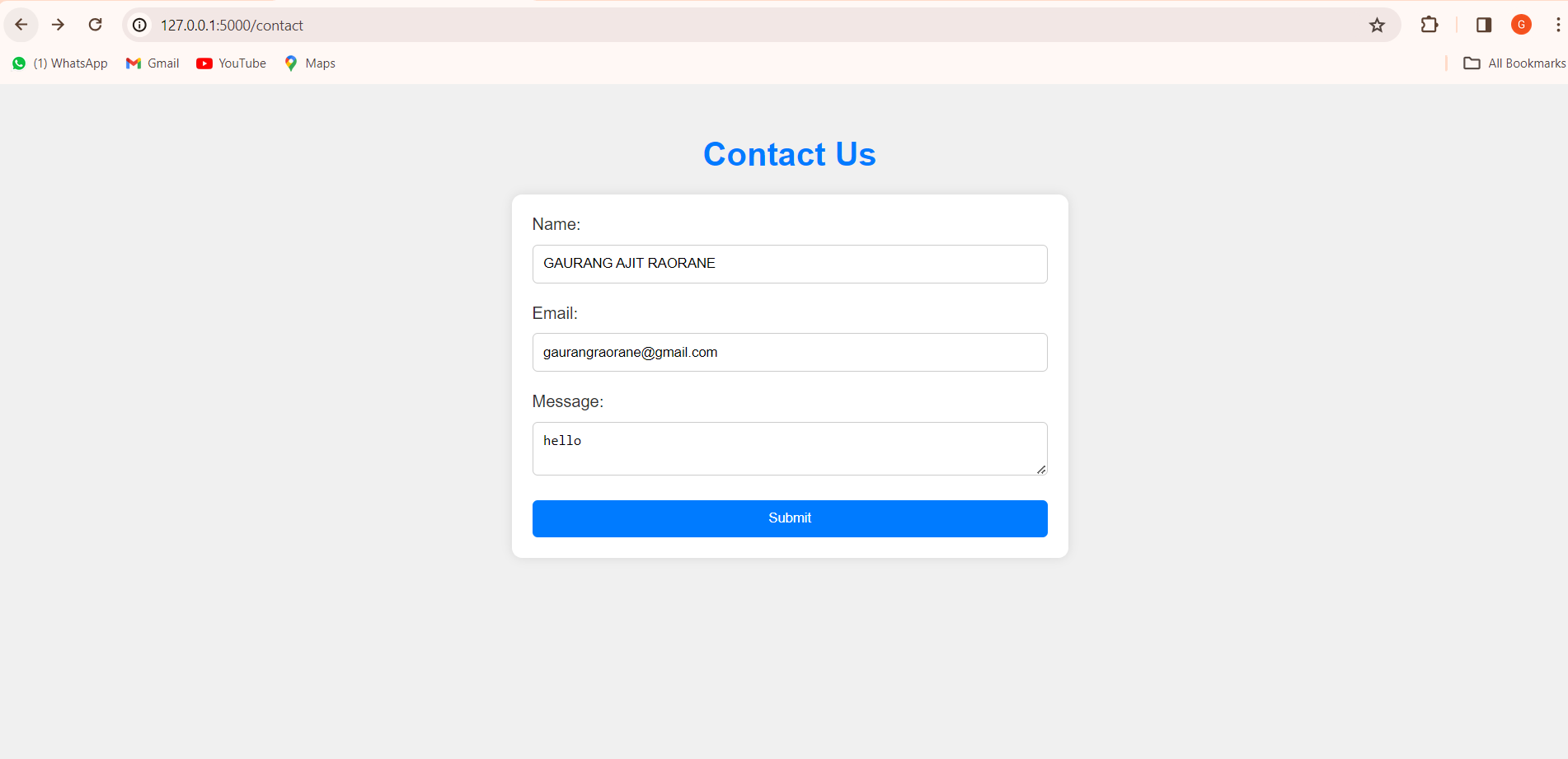
color: #fff;

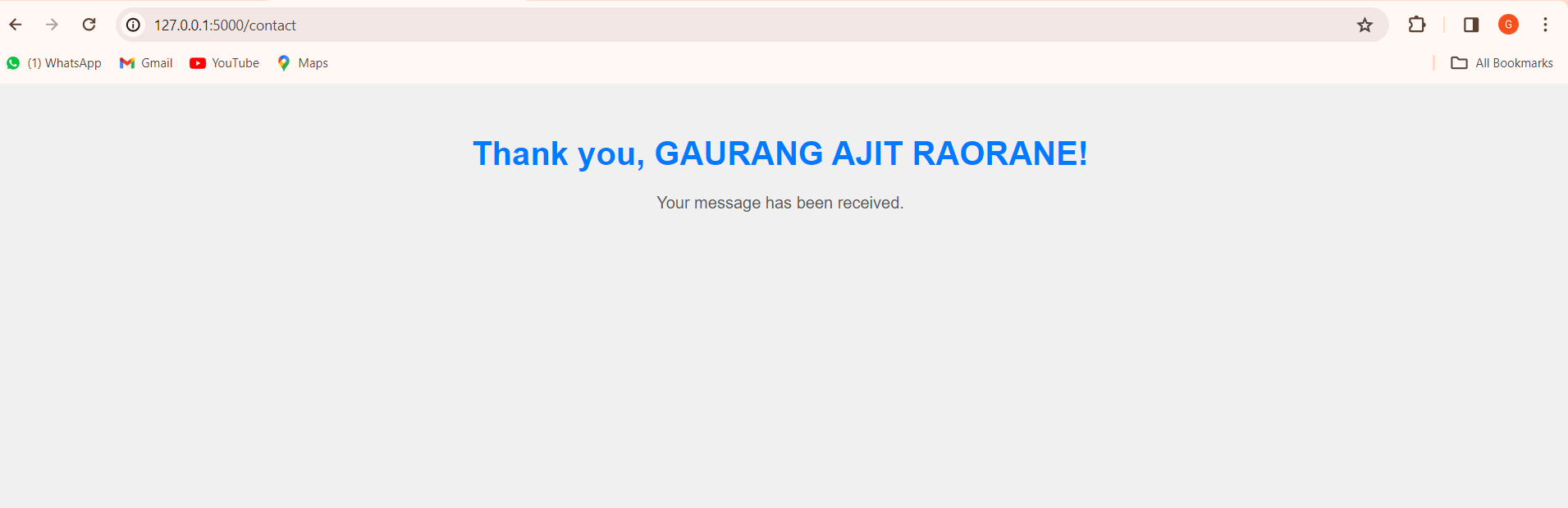
text-align: center;

padding: 20px;

}







Conclusion - We understood what is flask and implemented basic functionalities