Name : Shashwat Tripathi

Class : D15A

Roll no. : 64

**Experiment – 5: Flask**

1. **Aim: To study Flask**
2. **Problem Statement:**

**Design using Flask**

* 1. **A Contact Us Form**
  2. **A Portfolio Website.**

**Demonstrate the usage of Routing, Request Object and Templating**

1. **Theory:**
2. **List some of the core features of Flask**

Flask is a micro web framework for Python, which means it provides the basic functionality for building web applications and leaves much of the additional functionality to third-party extensions. Some core features of Flask include:

Lightweight and minimalist: Flask is designed to be simple and easy to use, with a minimalistic core that allows developers to add only the components they need.

Routing: Flask provides a built-in routing system that allows developers to map URL patterns to view functions, making it easy to define endpoints for handling HTTP requests.

Templating: Flask includes a built-in templating engine called Jinja2, which allows developers to create HTML templates with dynamic content.

HTTP request handling: Flask provides decorators to define view functions for handling different HTTP methods such as GET, POST, PUT, DELETE, etc.

Session management: Flask includes support for session management, allowing developers to store user-specific data across multiple requests.

Extension ecosystem: Flask has a vibrant ecosystem of extensions that add additional functionality such as database integration, authentication, and more.

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

In Flask, Flask(\_\_name\_\_) is used to create an instance of the Flask application. The \_\_name\_\_ variable is a special Python variable that represents the name of the current module. When Flask is run as the main program, \_\_name\_\_ is set to '\_\_main\_\_'. By passing \_\_name\_\_ as an argument to the Flask constructor, Flask can determine the root path of the application, which is necessary for locating templates, static files, and other resources. Additionally, it allows Flask to set up the application context and configure various settings.

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

In Flask, a template refers to an HTML file with placeholders for dynamic content. Templates are typically rendered using the Jinja2 templating engine. Template inheritance is a feature of Jinja2 that allows developers to create a base template with common layout and structure, and then extend or override specific blocks within that template in child templates. This allows for code reusability and helps maintain a consistent look and feel across multiple pages of a web application. Child templates can override specific blocks defined in the base template, while inheriting the rest of its content.

1. **What methods of HTTP are implemented in Flask.**

Flask implements several HTTP methods that correspond to different types of requests:

GET: Used to request data from a specified resource.

POST: Used to submit data to be processed to a specified resource.

PUT: Used to update a resource.

DELETE: Used to delete a specified resource.

PATCH: Used to apply partial modifications to a resource.

OPTIONS: Used to describe the communication options for the target resource.

HEAD: Same as GET, but without the response body, used to obtain metadata about the resource.

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

Flask and Django are both web frameworks for Python, but they have different design philosophies and cater to different use cases:

Flask is a micro-framework, meaning it provides only the core functionality needed to build web applications. It is lightweight and flexible, allowing developers to choose and integrate specific components as needed. Django, on the other hand, is a full-stack framework that comes with many built-in features and conventions for common web development tasks.

Flask is best suited for small to medium-sized applications, where developers want more control over the architecture and components of their application. Django, with its built-in features like ORM, admin interface, and authentication system, is better suited for larger, more complex applications with strict deadlines.

Flask is more minimalist and requires less boilerplate code, making it easier to get started with and learn. Django, while more opinionated, provides more out-of-the-box features and follows the "batteries-included" philosophy, reducing the need for third-party libraries.

Flask allows for greater flexibility and customization, making it suitable for projects with unique requirements. Django, with its built-in components, promotes rapid development by providing solutions to common problems out of the box.

Overall, the choice between Flask and Django depends on the specific requirements and constraints of the project, as well as the preferences and expertise of the development team.

1. **Output:**

**CODE:**

from flask import Flask, render\_template, request, redirect, url\_for

from flask\_mail import Mail, Message

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

# Flask-Mail configuration

app.config['MAIL\_SERVER'] = 'smtp.gmail.com'

app.config['MAIL\_PORT'] = 587

app.config['MAIL\_USE\_TLS'] = True

app.config['MAIL\_USERNAME'] = 'selmonbhoi123@gmail.com'

app.config['MAIL\_PASSWORD'] = 'zsrchkjsadads'

app.config['MAIL\_DEFAULT\_SENDER'] = 'selmonbhoi123@gmail.com'

mail = Mail(app)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/submit\_contact\_form', methods=['POST'])

def submit\_contact\_form():

name = request.form['name']

email = request.form['email']

message = request.form['message']

# Send email

msg = Message(subject='Contact Form Submission',

recipients=['']) # Specify recipient email

msg.body = f"Name: {name}\nEmail: {email}\nMessage: {message}"

mail.send(msg)

return redirect(url\_for('success', name=name, email=email))

@app.route('/success')

def success():

name = request.args.get('name')

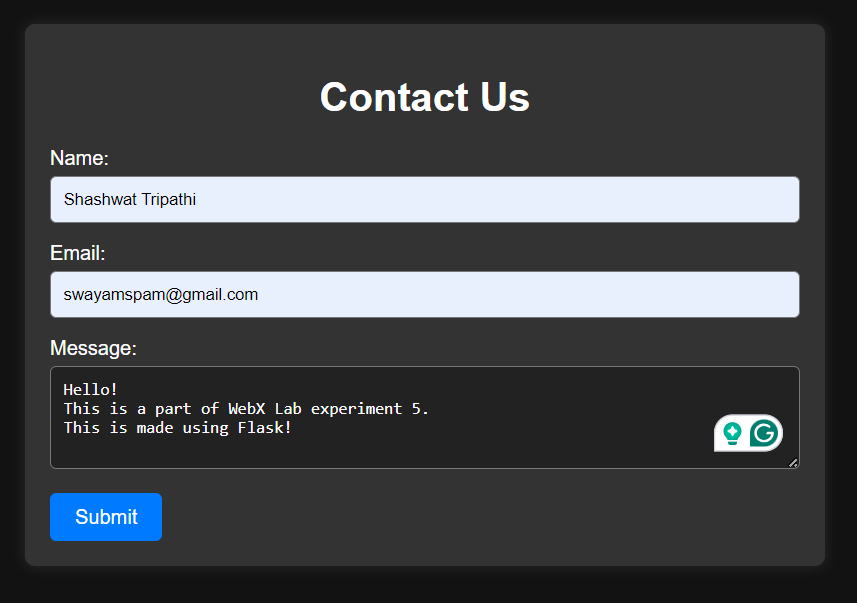
email = request.args.get('email')

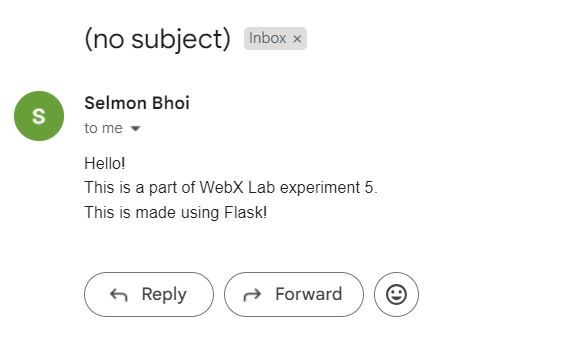
return render\_template('success.html', name=name, email=email)

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

app.run(debug=True)

**OUTPUT:**





**CODE:**from flask import Flask, render\_template, request, url\_for, redirect

from email.mime.text import MIMEText

import smtplib

from email.message import EmailMessage

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

@app.route("/")

def index():

return render\_template("index.html")

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

def sendemail():

if request.method == "POST":

name = request.form['name']

subject = request.form['Subject']

email = request.form['\_replyto']

message = request.form['message']

# Set your credentials

yourEmail = "suraj@geeksforgeeks.org"

yourPassword = "########"

# Logging in to our email account

server = smtplib.SMTP('smtp.gmail.com', 587)

server.ehlo()

server.starttls()

server.login(yourEmail, yourPassword)

# Sender's and Receiver's email address

msg = EmailMessage()

msg.set\_content("First Name : "+str(name)

+"\nEmail : "+str(email)

+"\nSubject : "+str(subject)

+"\nMessage : "+str(message))

msg['To'] = email

msg['From'] = yourEmail

msg['Subject'] = subject

# Send the message via our own SMTP server.

try:

# sending an email

server.send\_message(msg)

print("Send")

except:

print("Fail to Send")

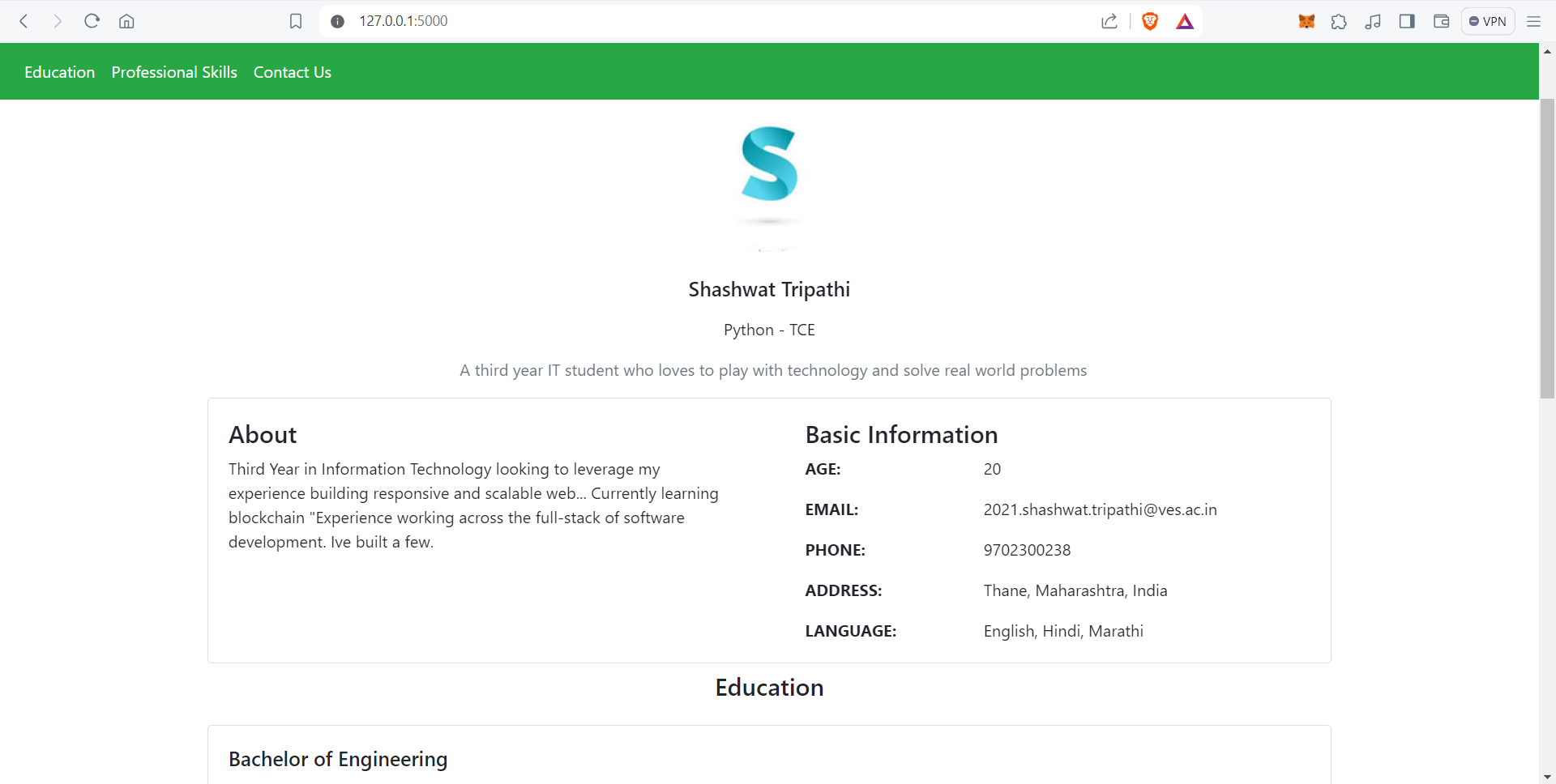
pass

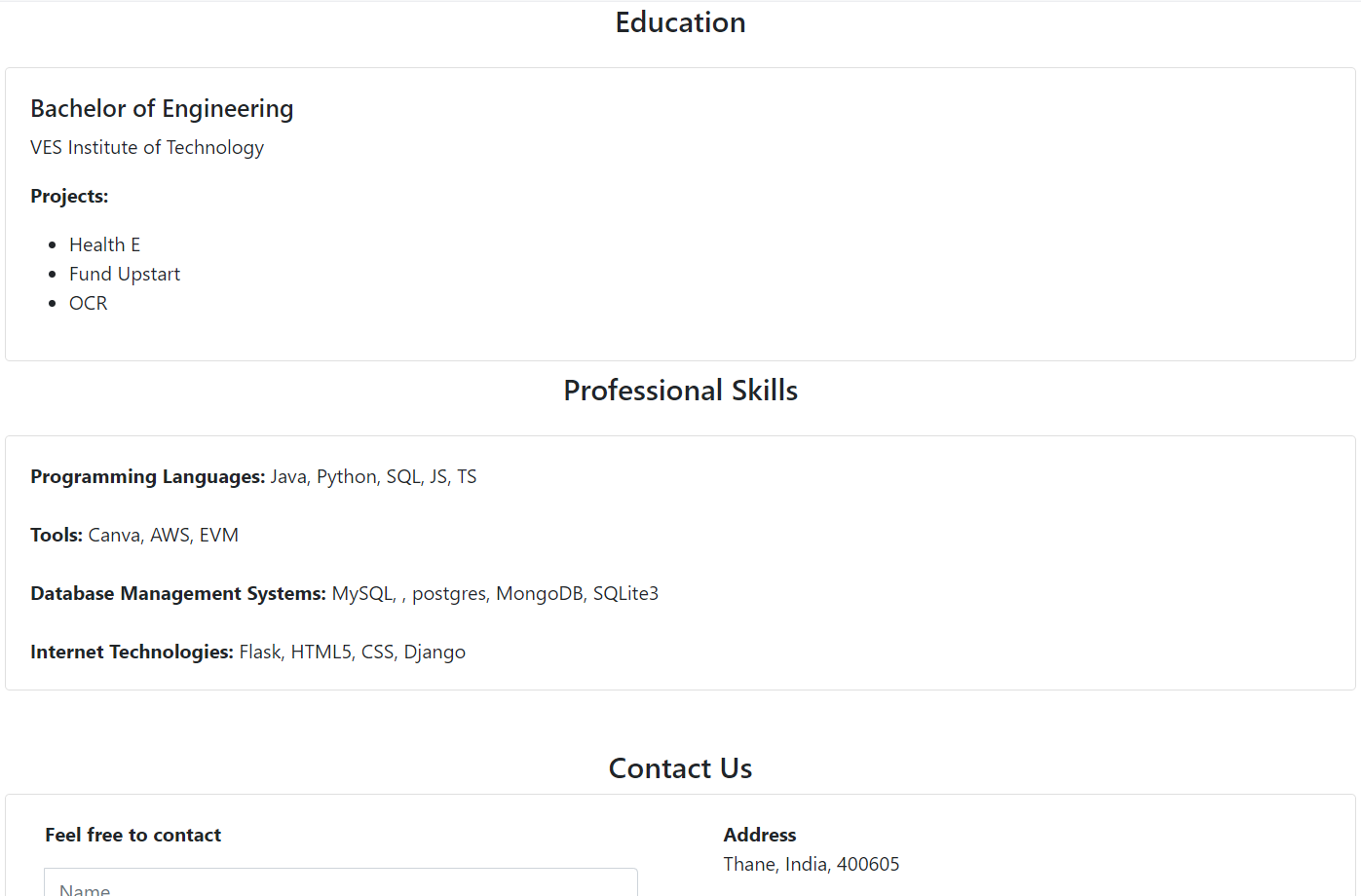
return redirect('/')

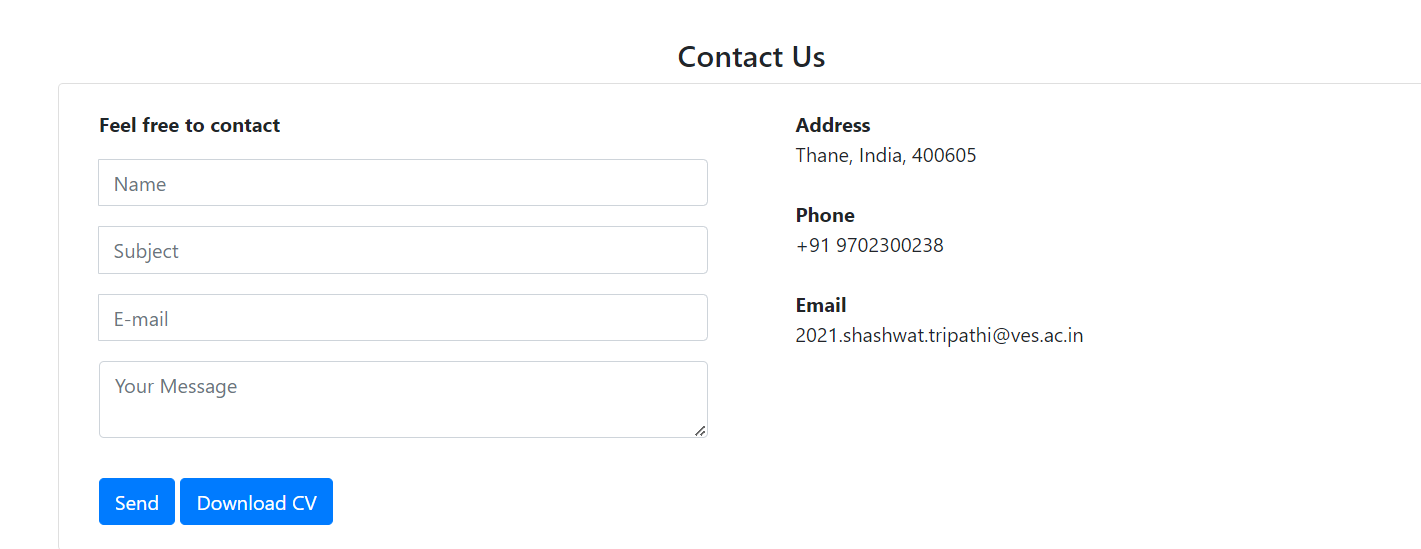
if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**OUTPUT:**







**Conclusion:** Thus we have studied and implemented flask in this experiment.