# Program No: 1

## Problem statement:

static HTML page.

## Aim:

A program that creates a simple web server and serves a

To solve this problem statement.

## Problem Description:

Develop a Python program using Flask to create a

simple web server serving a static HTML page. The objective is to use the flask command (flask --app <YourAppName> run) instead of running the script directly. Ensure proper project organization, write a basic HTML file, and confirm functionality by accessing http://127.0.0.1:5000/ in a web browser.

## Algorithm:

**Step 1:** start

**Step 2:** Install Flask

**Step 3:** create a Project Structure

**Step 4:** Write a HTML file and save as a index.html file

**Step 5:** Write a Flask file and save as a app.py file

**Step 6:** Run that Flask file.

**Step 7:** Stop

## HTML FILE:

**Index.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>Static HTML Page</title>

</head>

<body>

<h1>Hello, this is a static HTML page served by Flask!</h1>



</body>

</html>

## FLASK FILE:

**app.py**

from flask import Flask, render\_template app = Flask( name )

# Define a route for the root URL "/" @app.route("/")

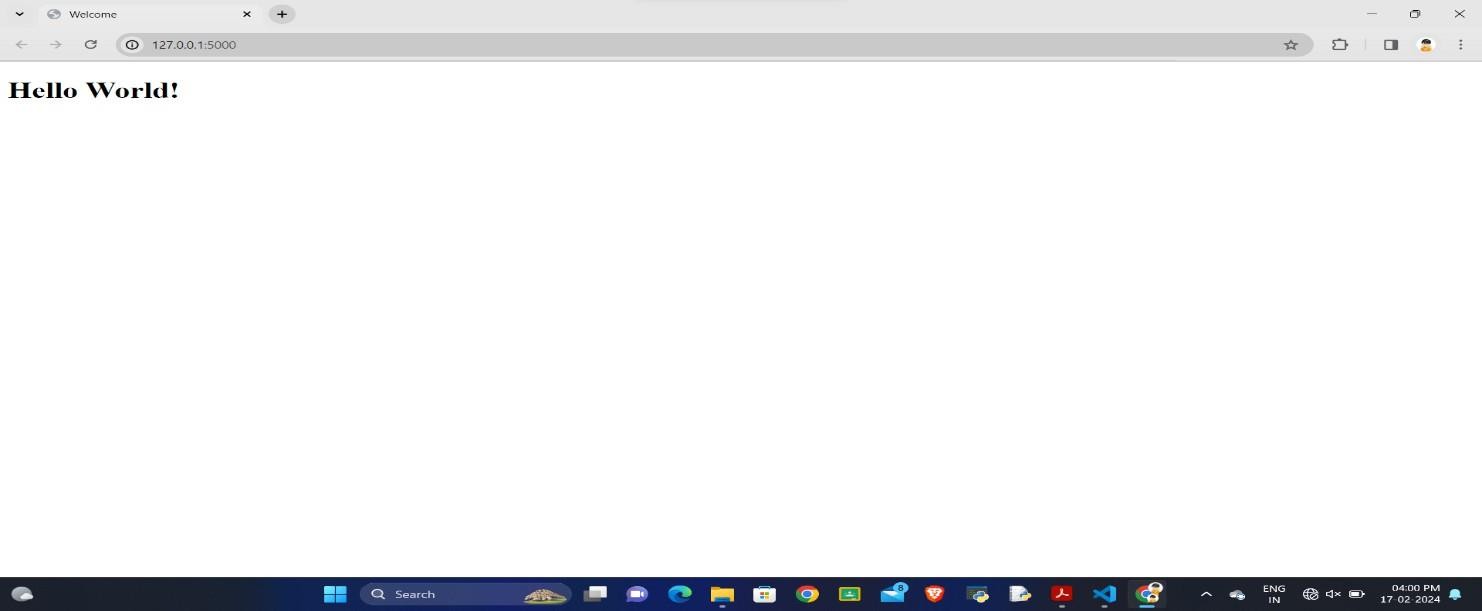
def home():

# Render the static HTML page located in the "templates" folder return render\_template("index.html")

# Run the app if this script is the main program if name == " main ":

app.run(debug=True)

## Output:



**Conclusion:**

Here We have successfully creates a simple web server and serves a static HTML page.



# Program No: 2

## Problem statement:

A program that creates a web application that allows

users to register and login.

## Aim:

To solve this problem statement.

## Problem Description:

The web application will be built using a combination of

front-end and back-end technologies. The front-end will be responsible for creating the user interface, while the back-end will handle user registration, login authentication, and data storage.

## Algorithm:

**Step 1:** start

**Step 2:** create a Project Structure

**Step 3:** Write a HTML file and save as a index.html ,login.html & register.html file

**Step 4:** Write a Flask file and save as a app.py file

**Step 5:** Run that Flask file.

**Step 6:** Stop

## HTML FILE:

**Index.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>Static HTML Page</title>

</head>

<style>



@import url[("h](https://fonts.googleapis.com/css2?family=Poppins%3Awght%40500&display=sw)t[tp](https://fonts.googleapis.com/css2?family=Poppins%3Awght%40500&display=sw)s[://fonts.googleapis.com/css2?family=Poppins:wght@500&display=sw](https://fonts.googleapis.com/css2?family=Poppins%3Awght%40500&display=sw) ap");

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

height: 100vh; width: 100%; display: flex;

justify-content: center; align-items: center; flex-direction: column; background: #ff5a5f;

}

h1 {

font-family: "Poppins", sans-serif; color: #fff;

margin: 30px 50px; font-size: 3rem;

}

input {

padding: 10px 20px; border: 3px solid #fff; border-radius: 10px;

background: rgb(16, 208, 16); font-size: 1.5rem;

color: white;

font-family: "Poppins", sans-serif; font-weight: 300;

transition: .3s; &:hover{ background: #fff; color: #000; cursor: pointer;

}

}



</style>

<body>

<h1>Hello, this is a static HTML page served by Flask!</h1>

<form action="{{ url\_for('register') }}">

<input type="submit" value="Register" />

</form>

</body>

</html>

## login.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>User Login</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

height: 100vh; width: 100%; display: flex;

align-items: center; justify-content: center; flex-direction: column;

background: rgb(9, 9, 121); background: linear-gradient( 30deg,

rgba(9, 9, 121, 1) 0%,

rgba(2, 0, 36, 1) 29%,

rgba(0, 212, 255, 1) 100%

);

}



.container { display: flex;

align-items: center;

justify-content: space-evenly; flex-direction: column; width: 600px;

border-radius: 20px; height: 500px; background: #ffffff5a; backdrop-filter: blur(20px); & h1 {

font-family: Arial, Helvetica, sans-serif; color: #fff;

margin: 30px 0;

}

& li {

list-style: none;

}

& form { & label {

color: white;

font-family: Arial, Helvetica, sans-serif; font-size: 1.4rem;

margin: 10px 20px;

}

& .log\_button { color: #fff; background: red; border: none; outline: none; padding: 5px 10px; border-radius: 10px; font-size: 1.2rem; transition: 0.3s;

transform: translateX(130px); &:hover {

background: #fff; color: #000; cursor: pointer;



}

}

& .password{ padding: 10px 20px; border-radius: 20px; outline: none; border: none;

}

& .username{ padding: 10px 20px; border-radius: 20px; outline: none; border: none;

}

& input {

margin: 10px 20px;

}

}

}

.error { color: red;

}

.success { color: green;

}

.default { color: black;

}

</style>

</head>

<body>

<div class="container">

<h1>User Login</h1>

{% with messages = get\_flashed\_messages() %} {% if messages %}

<ul>

{% for message in messages %}

<li

class="{% if 'error' in message %}error{% elif 'success' in message

%}success{% else %}default{% endif %}"



>

{{ message }}

</li>

{% endfor %}

</ul>

{% endif %} {% endwith %}

<form method="post" action="{{ url\_for('login') }}">

<label for="username" class="username\_label">Username:</label>

<input type="text" name="username" class="username" required />

<br />

<label for="password" class="password\_label">Password:</label>

<input type="password" name="password" class="password" required />

<br />

<input type="submit" class="log\_button" value="Log in" />

</form>

<p>

Don't have an account?

<a href="{{ url\_for('register') }}">Register here</a>.

</p>

</div>

</body>

</html>

## register.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>User Registration</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {



height: 100vh; width: 100%; display: flex;

align-items: center; justify-content: center; flex-direction: column;

background: rgb(9, 9, 121); background: linear-gradient( 30deg,

rgba(9, 9, 121, 1) 0%,

rgba(2, 0, 36, 1) 29%,

rgba(0, 212, 255, 1) 100%

);

}

.container { display: flex;

align-items: center;

justify-content: space-evenly; flex-direction: column; width: 600px;

border-radius: 20px; height: 500px; background: #ffffff5a; backdrop-filter: blur(20px); & h1 {

font-family: Arial, Helvetica, sans-serif; color: #fff;

margin: 30px 0;

}

& li {

list-style: none;

}

& form { & label {

color: white;

font-family: Arial, Helvetica, sans-serif; font-size: 1.4rem;

margin: 10px 20px;

}



& .register\_button { color: #fff; background: red; border: none; outline: none; padding: 5px 10px; border-radius: 10px; font-size: 1.2rem; transition: 0.3s;

transform: translateX(130px); &:hover {

background: #fff; color: #000; cursor: pointer;

}

}

& .password { padding: 10px 20px; border-radius: 20px; outline: none; border: none;

}

& .username { padding: 10px 20px; border-radius: 20px; outline: none; border: none;

}

& input {

margin: 10px 20px;

}

}

}

.error { color: red;

}

.success { color: green;

}



.default { color: black;

}

</style>

</head>

<body>

<div class="container">

<h1>User Registration</h1>

{% with messages = get\_flashed\_messages() %} {% if messages %}

<ul>

{% for message in messages %}

<li

class="{% if 'error' in message %}error{% elif 'success' in message

%}success{% else %}default{% endif %}"

>

{{ message }}

</li>

{% endfor %}

</ul>

{% endif %} {% endwith %}

<form method="post" action="{{ url\_for('register') }}">

<label for="username" class="username\_label">Username:</label>

<input type="text" name="username" class="username" required />

<br />

<label for="password" class="password\_label">Password:</label>

<input type="password" name="password" class="password" required />

<br />

<input type="submit" class="register\_button" value="Register" />

</form>

<p>

Already have an account?

<a href="{{ url\_for('login') }}">Log in here</a>.

</p>

</div>

</body>

</html>



## FLASK FILE:

**app.py**

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

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash import secrets

# print(secrets.token\_hex(16)) app = Flask( name )

app.secret\_key = secrets.token\_hex(16) app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///users.db' # SQLite database, change for other databases

db = SQLAlchemy(app)

# Define the User model class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(50), unique=True, nullable=False) password = db.Column(db.String(256), nullable=False)

# Ensure the creation of all tables inside the application context with app.app\_context():

db.create\_all()

@app.route("/") def home():

# Render the static HTML page located in the "templates" folder return render\_template("index.html")

# Define a route for the registration page @app.route('/register', methods=['GET', 'POST']) def register():

if request.method == 'POST': username = request.form['username']



password = request.form['password']

# Check if the username is already taken

if User.query.filter\_by(username=username).first(): flash('Username already taken. Please choose another.', 'error')

else:

# Hash the password before storing it

hashed\_password = generate\_password\_hash(password, method='pbkdf2:sha256')

# Create a new user instance

new\_user = User(username=username, password=hashed\_password)

# Add the user to the database db.session.add(new\_user) db.session.commit()

flash('Registration successful. You can now log in.', 'success') return redirect(url\_for('login'))

return render\_template('register.html') # Define a route for the login page

@app.route('/login', methods=['GET', 'POST']) def login():

if request.method == 'POST': username = request.form['username'] password = request.form['password']

# Query the user from the database

user = User.query.filter\_by(username=username).first()

# Check if the username exists and the password is correct

if user and check\_password\_hash(user.password, password): session['username'] = username

flash('Login successful!', 'success') return redirect(url\_for('dashboard'))

else:

flash('Invalid username or password. Please try again.', 'error')



return render\_template('login.html')

# Define a route for the dashboard (a protected route accessible only after login) @app.route('/dashboard')

def dashboard():

if 'username' in session:

return f'Welcome to the dashboard, {session["username"]}!' else:

flash('Please log in to access the dashboard.', 'info') return redirect(url\_for('login'))

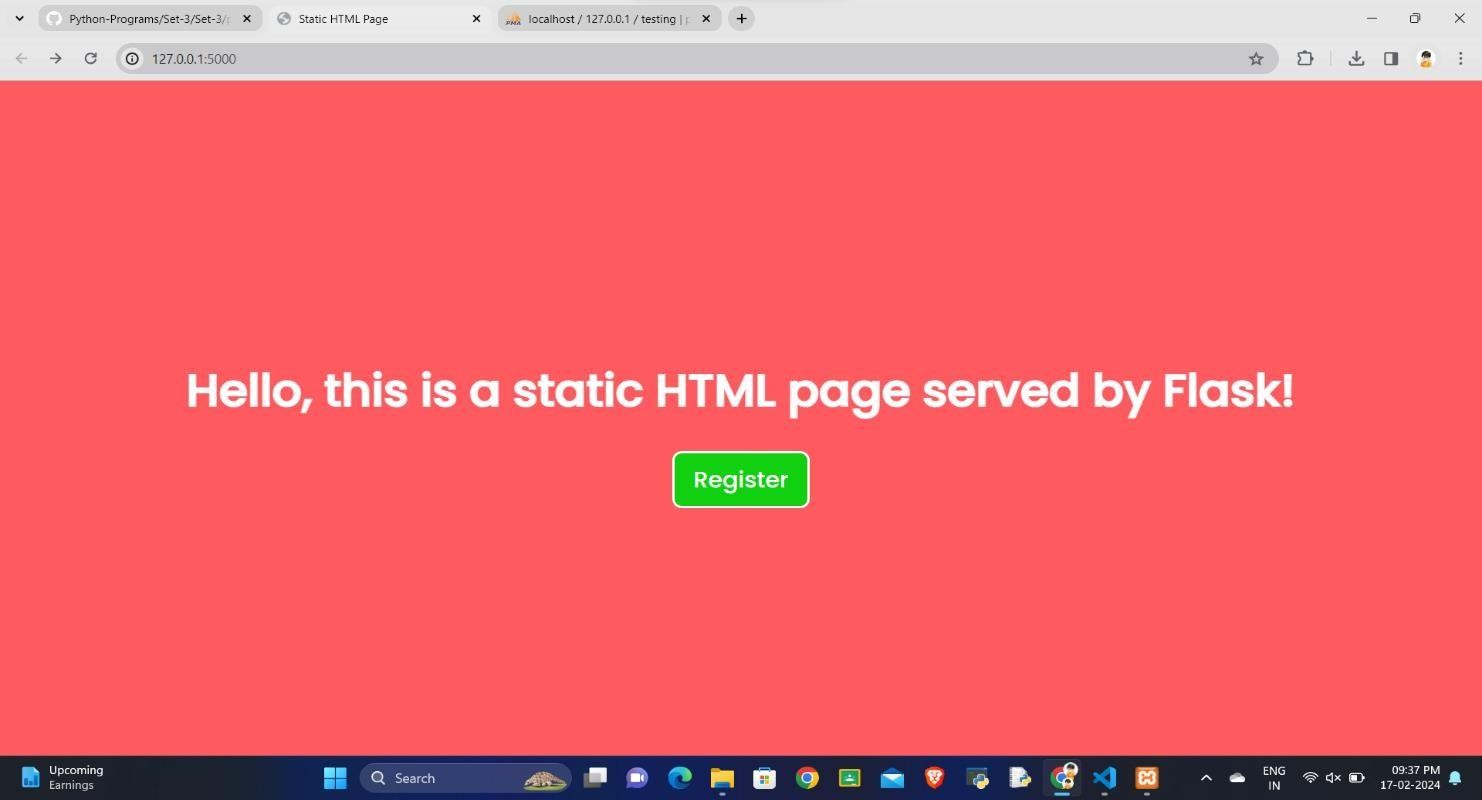
# Logout route @app.route('/logout') def logout():

session.pop('username', None) flash('You have been logged out.', 'info') return redirect(url\_for('login'))

if name == ' main ': app.run(debug=True)

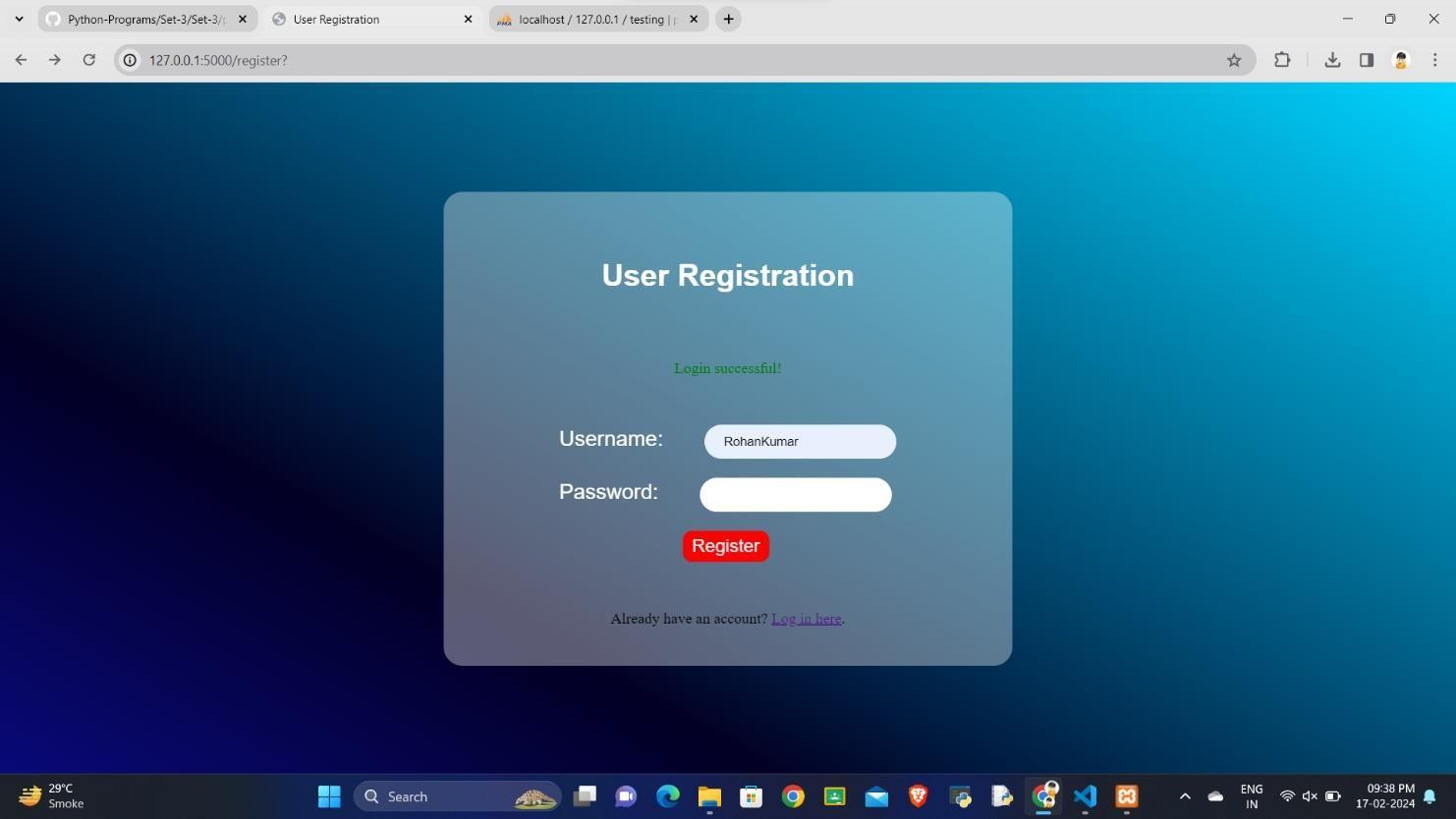
## Output:

**#static html page:**

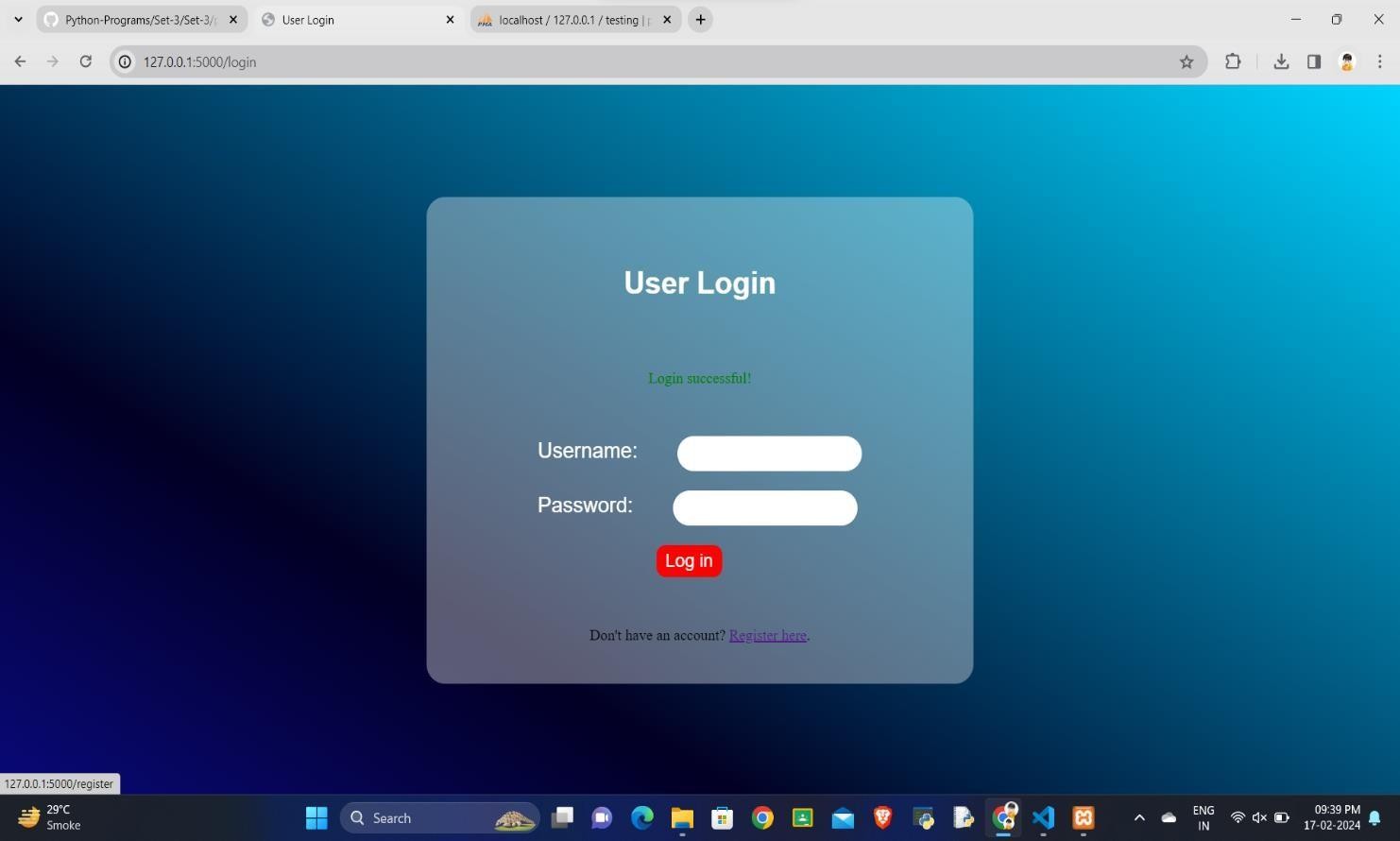




## #Registation page

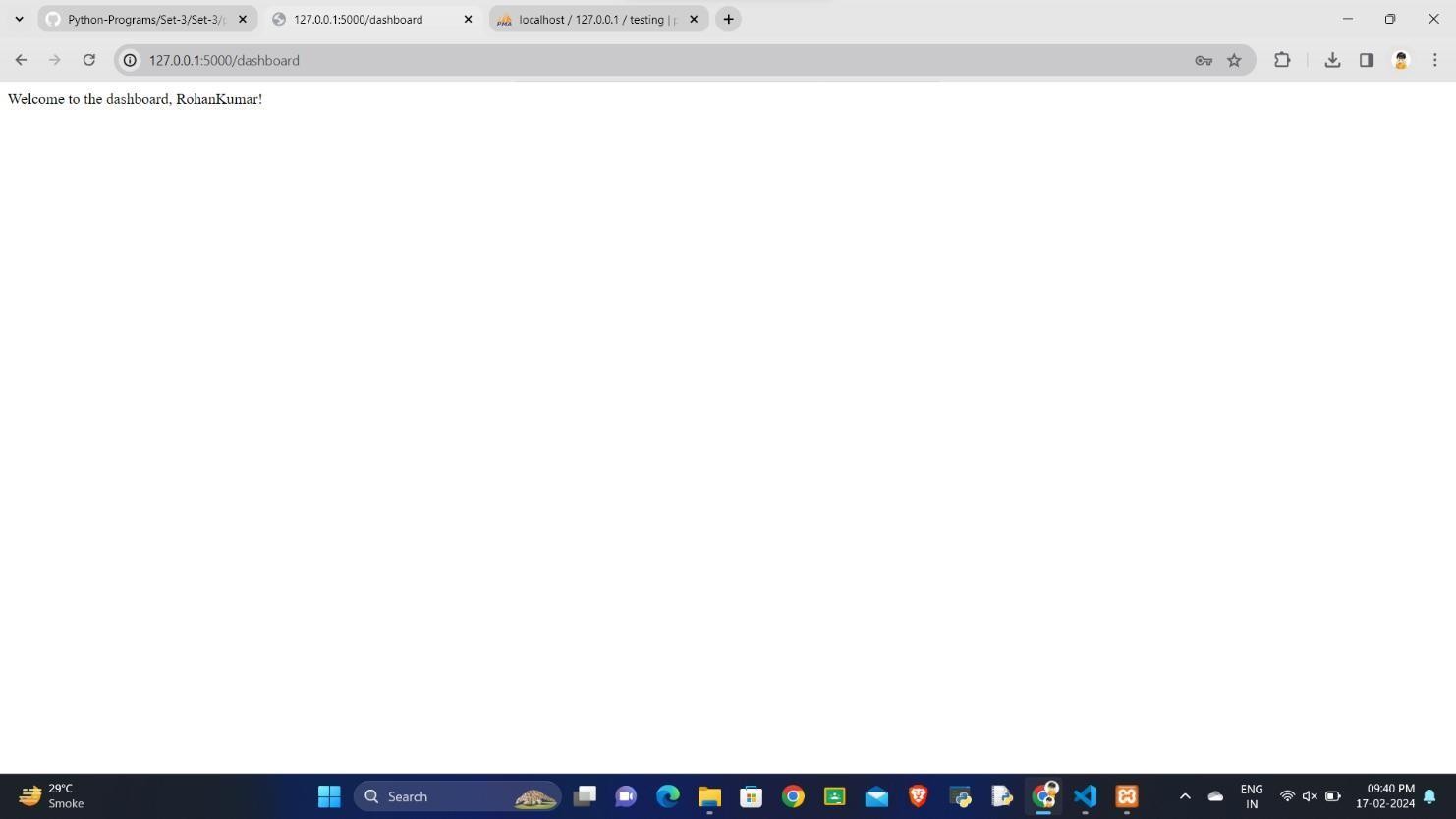


**#Login page**





## #Dashboard



**Conclusion:**

Here We have successfully creates a web application that

allows users to register and login.



# Program No:3

## Problem statement:

A program that creates a web application that allows

users to upload and download files.

## Aim:

To solve this problem statement.

## Problem Description:

Develop a Flask-based web application featuring:

## File Upload:

* Users can upload files through a web form.
* Prevent form submission without selecting a file.
* Save uploaded files to a server directory (e.g., uploads).

## File Display:

* Display a dynamic list of uploaded files on the main page.
* Each file entry includes a "Download" button.

## File Download:

* Enable users to download files by clicking the corresponding "Download" button.

## User Interface:

* Design a clean and user-friendly interface using HTML templates.
* Separate application code (app.py) and HTML templates.

## Algorithm:

**Step 1:** start

**Step 2:** create a Project Structure

**Step 3:** Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.

**Step 6:** Stop



## HTML FILE

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>File Upload and Download</title>

</head>

<body>

<h1>File Upload and Download</h1>

<form action="/upload" method="post" enctype="multipart/form-data">

<label for="file">Choose a file:</label>

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

<br>

<input type="submit" value="Upload">

</form>

<h2>Uploaded Files</h2>

{% for filename in filenames %}

<div>

<span>{{ filename }}</span>

<a href="{{ url\_for('download\_file', filename=filename) }}" download>

<button>Download</button>

</a>

</div>

{% endfor %}

</body>

</html>

## FLSK FILE

**app.py**

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

import os

app = Flask( name )



UPLOAD\_FOLDER = 'uploads' app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True) @app.route('/')

def index():

filenames = os.listdir(app.config['UPLOAD\_FOLDER']) return render\_template('index.html', filenames=filenames)

@app.route('/upload', methods=['POST']) def upload\_file():

if 'file' not in request.files: return "No file part"

file = request.files['file'] if file.filename == '':

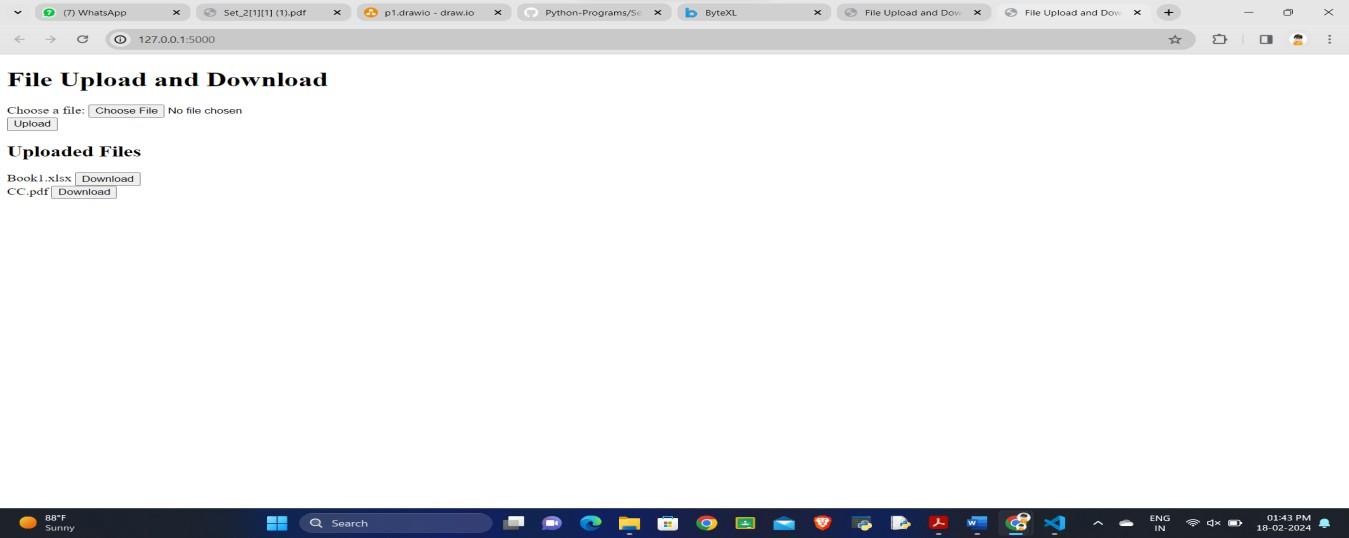
return "No selected file" file.save(os.path.join(app.config['UPLOAD\_FOLDER'], file.filename)) return redirect(url\_for('index'))

@app.route('/download/<filename>') def download\_file(filename):

return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename)

if name == ' main ': app.run(debug=True)

## Output:



**Conclusion:**

Here We have successfully creates a web application that

allows users to upload and download files.



# Program No:4

## Problem statement:

A program that creates a web application that displays data from a database in a tabular format.

## Aim:

To solve this problem statement.

## Problem Description:

The program utilizes Flask, SQLAlchemy, and Pandas

to create a web application. It defines a simple SQLAlchemy model (Person) to represent data with attributes like name and age. Sample data is inserted into an SQLite database. The main route (/) queries the database, converts the data to a Pandas DataFrame, and renders it as an HTML table using a Flask template (index.html).

## Algorithm:

**Step 1:** start

**Step 2:** create a Project Structure

**Step 3:** Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.

**Step 6:** Stop

## HTML FILE

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Data Display</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">



<h1>Data Display</h1>

<!-- Render the HTML table -->

{{ table\_html | safe }}

</div>

</body>

</html>

## FLSK FILE

**app.py**

from flask import Flask, render\_template import pandas as pd

app = Flask( name )

sample\_data = {'name': ['John', 'Alice', 'Bob'], 'age': [25, 30, 22]}

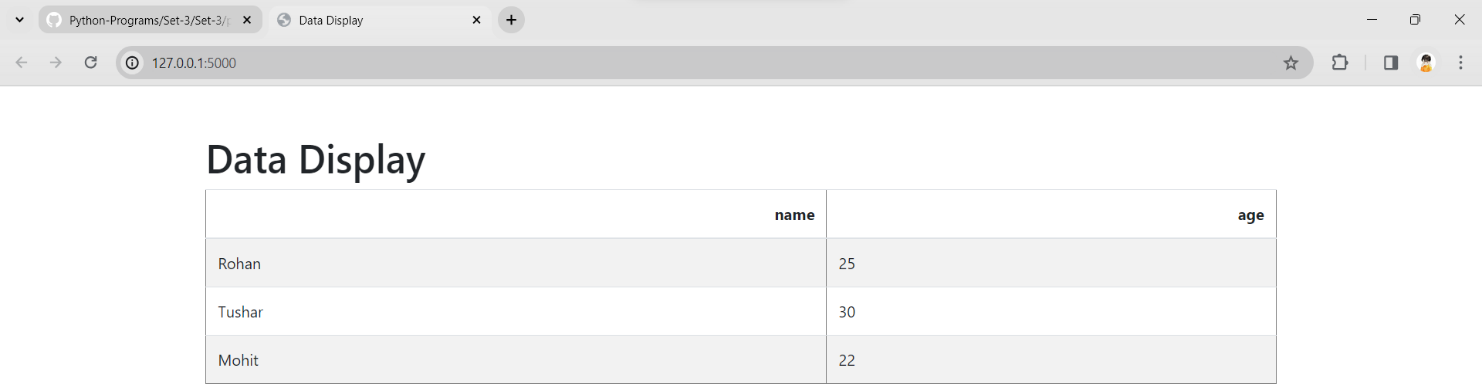
df = pd.DataFrame(sample\_data) @app.route('/')

def display\_data():

table\_html = df.to\_html(classes='table table-striped', index=False) return render\_template('index.html', table\_html=table\_html)

if name == ' main ': app.run(debug=True)

## Output:



**Conclusion:**

Here We have successfully creates a web application that

displays data from a database in a tabular format.



# Program No:5

## Problem statement:

A program that creates a web application that accepts

user input and sends it to a server-side script for processing.

## Aim:

To solve this problem statement.

## Problem Description:

You are tasked with creating a web application using

Flask that enables users to input data on the main page through a form. The entered data should be sent to the server, processed by a server-side script, and the result displayed on the same page. The provided code includes a basic structure for achieving this, where user input is obtained from a form, and a simple processing logic is applied.

## Algorithm:

**Step 1:** start

**Step 2:** create a Project Structure

**Step 3:** Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.

**Step 6:** Stop

## HTML FILE

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>User Input</title>

</head>

<style>

\* {

margin: 0;



padding: 0;

box-sizing: border-box;

}

body {

height: 100vh; width: 100%;

background: #a2d2ff; display: flex;

align-items: center; justify-content: center; flex-direction: column;

}

.container { display: flex;

align-items: center;

justify-content: space-evenly; flex-direction: column; width: 500px;

height: 600px; border-radius: 20px;

background: #ffffff5a; backdrop-filter: blur(20px); & h1{

font-family: Arial, Helvetica, sans-serif; color: #3a86ff;

font-size: 2rem;

}

& label{

color: #3a86ff;

font-family: Arial, Helvetica, sans-serif; font-size: 1.2rem;

padding: 10px; margin: 10px 20px;

}

& .enter{

padding: 10px 20px; border: none; outline: none; border-radius: 20px;



}

& .submit{

padding: 10px 20px; color: #fff; background: #2a9d8f; outline: none;

border: none; border-radius: 10px; transition: .3s;

transform: translateX(150px); margin: 30px;

&:hover{ color: #000;

cursor: pointer; background: #fff;

}

}

& h2{

font-family: Arial, Helvetica, sans-serif; color: #3a86ff;

font-size: 2rem;

}

}

</style>

<body>

<div class="container">

<h1>User Input Form</h1>

<form method="post" action="/">

<label for="user\_input">Enter something:</label>

<input type="text" class="enter" name="user\_input" id="user\_input" required />

<br />

<input class="submit" type="submit" value="Submit" />

</form>

{% if result %}

<div>

<h2>Result:</h2>

<p>{{ result }}</p>



</div>

{% endif %}

</div>

</body>

</html>

## FLSK FILE

**app.py**

from flask import Flask, render\_template, request

app = Flask( name )

@app.route('/', methods=['GET', 'POST']) def index():

result = None

if request.method == 'POST':

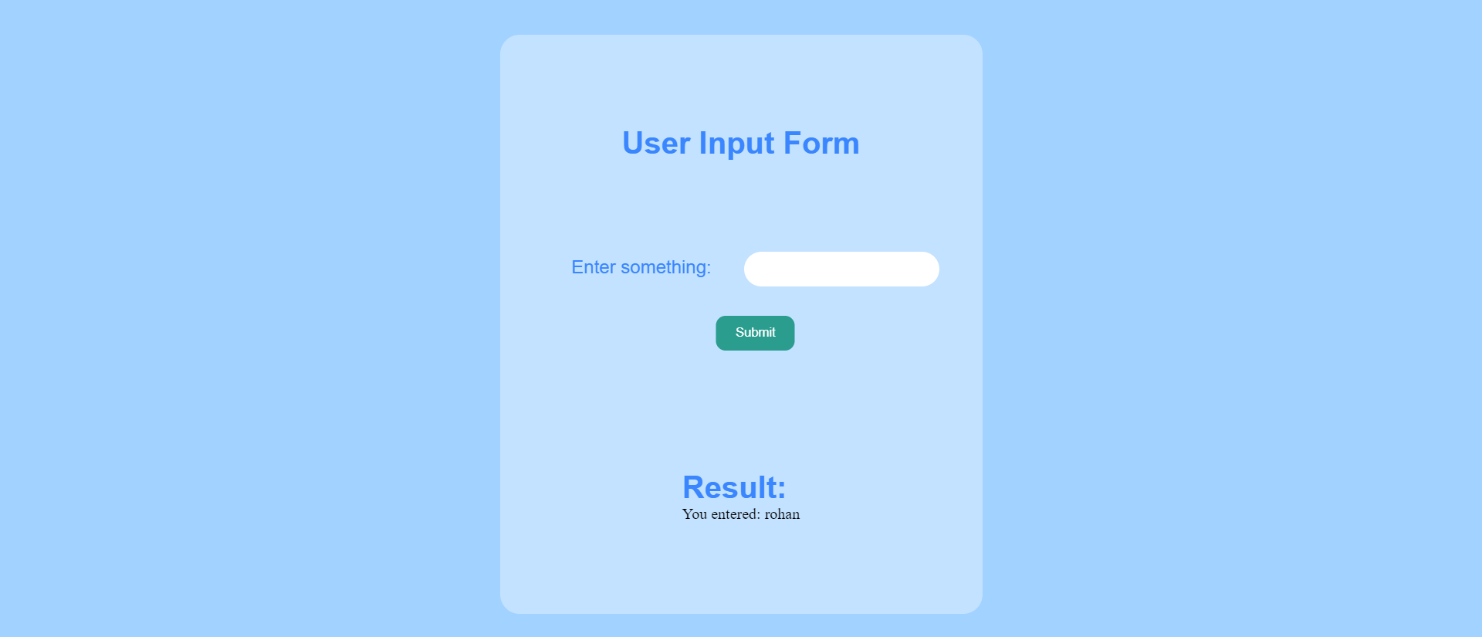
# Get user input from the form

user\_input = request.form.get('user\_input') result = f"You entered: {user\_input}"

return render\_template('index.html', result=result)

if name == ' main ': app.run(debug=True)

## Output:



**Conclusion:**

Here We have successfully creates a web application that

accepts user input and sends it to a server-side script for processing.



