| **Experiment No. – 9** | | | | |
| --- | --- | --- | --- | --- |
| **Date of Performance:** | **16/4/25** | | | |
| **Date of Submission:** | **23/4/25** | | | |
| Program Execution/  formation/  correction/  ethical practices  (06) | Timely  Submission  (01) | Viva  (03) | Experiment  Total (10) | Sign with Date |
|  |  |  |  |  |

**9.1 Aim :** Design a Web Page using Flask.

**9.2 Course Outcome :** Develop Rich Internet Application using proper choice of Framework.

**9.3 Learning Objectives:**

* Understand the architecture of web applications.
* Learn to use Flask for web development.
* Implement routing, templates, and backend logic in Flask.
* Perform CRUD (Create, Read, Update, Delete) operations using a database.
* Integrate a frontend with backend APIs to build a complete RIA.
* Understand CORS and API communication with frontend frameworks.

**9.4 Requirement:**

### **Software Requirements:**

* Python 3.x
* Flask
* Flask-CORS
* Flask-SQLAlchemy
* SQLite
* Node.js & npm (for frontend)
* React (optional frontend framework)
* A computer with minimum 4GB RAM
* Internet connection for package installations
* Text Editor or IDE (VS Code, PyCharm, etc.)

**9.5 Related Theory:**

### **What is Flask?**

Flask is a lightweight, open-source Python web framework used to build web applications. It follows the WSGI (Web Server Gateway Interface) standard and provides essential tools and features to create dynamic websites. Flask is known for its simplicity and flexibility, making it ideal for small to medium applications.

### **Rich Internet Applications (RIA):**

RIAs are web applications that have features and functionalities similar to desktop applications. They provide rich, interactive user experiences by using advanced client-side technologies like JavaScript frameworks (React, Angular, etc.), and communicate asynchronously with the server.

### **Key Concepts Used in This Project:**

#### **1. Flask Routing**

Flask provides decorators like @app.route() to bind URLs to Python functions, allowing dynamic content to be served based on the route.

#### **2. Templates (Jinja2)**

Flask uses Jinja2 templating engine to embed Python code within HTML files, allowing dynamic rendering of content.

#### **3. Database Integration**

Using SQLAlchemy with Flask allows for Object Relational Mapping (ORM) which simplifies working with databases using Python classes.

#### **4. RESTful APIs**

Flask can be used to build RESTful APIs, enabling the frontend and backend to communicate through HTTP requests like GET, POST, PUT, and DELETE.

#### **5. CORS (Cross-Origin Resource Sharing)**

Using Flask-CORS, we allow the frontend (usually on a different port or domain) to communicate with the Flask backend securely.

#### **6. Frontend Integration**

React (or plain HTML/CSS/JS) can be used to build a user-friendly interface that communicates with the Flask backend for data management.

**Implementation of Code:**

**Steps:**

**Create a Python virtual environment**

mkdir flask-react-task-manager

cd flask-react-task-manager

python -m venv venv

source venv/bin/activate # On Windows use: venv\Scripts\activate

**Install Flask and other required packages**

pip install Flask Flask-CORS flask\_sqlalchemy

**Create app.py file**

*from* flask *import* Flask, request, jsonify

*from* flask\_cors *import* CORS

*from* flask\_sqlalchemy *import* SQLAlchemy

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

CORS(app)

app.config['SQLALCHEMY\_DATABASE\_URI'] *=* 'sqlite:///tasks.db'

db *=* SQLAlchemy(app)

*class* Task(db.Model):

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

title *=* db.Column(db.String(200), nullable*=*False)

completed *=* db.Column(db.Boolean, default*=*False)

*@app.route*('/tasks', methods*=*['GET'])

*def* *get\_tasks*():

tasks *=* Task.query.all()

*return* jsonify([{'id': t.id, 'title': t.title, 'completed': t.completed} *for* t *in* tasks])

*@app.route*('/tasks', methods*=*['POST'])

*def* *create\_task*():

data *=* request.get\_json()

task *=* Task(title*=*data['title'])

db.session.add(task)

db.session.commit()

*return* jsonify({'id': task.id, 'title': task.title, 'completed': task.completed})

*@app.route*('/tasks/<int:id>', methods*=*['PUT'])

*def* *update\_task*(id):

data *=* request.get\_json()

task *=* Task.query.get\_or\_404(id)

task.title *=* data['title']

task.completed *=* data['completed']

db.session.commit()

*return* jsonify({'message': 'Task updated'})

*@app.route*('/tasks/<int:id>', methods*=*['DELETE'])

*def* *delete\_task*(id):

task *=* Task.query.get\_or\_404(id)

db.session.delete(task)

db.session.commit()

*return* jsonify({'message': 'Task deleted'})

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

*with* app.app\_context():

db.create\_all()

app.run(debug*=*True)

**Run the Flask server**

python app.py

**Set Up React Frontend:**

**Open a new terminal and navigate to the project folder**

cd flask-react-task-manager

npx create-react-app task-manager

cd task-manager

**Install Axios (for API calls)**

npm install axios

**Replace the contents of src/App.js with:**

***import* *React*, { *useEffect*, *useState* } *from* 'react';**

***import* *axios* *from* 'axios';**

***function* *App*() {**

***const [tasks, setTasks] = useState([])*;**

***const [title, setTitle] = useState(*''*)*;**

***const fetchTasks = async () => {***

***const res = await axios.get(*'http://localhost:5000/tasks'*);***

***setTasks(res.data);***

***}*;**

***useEffect*(() *=>* { *fetchTasks*(); }, []);**

***const addTask = async () => {***

***if (!*title*) return;***

***await axios.post(*'http://localhost:5000/tasks'*, {* title *});***

***setTitle(*''*);***

***fetchTasks();***

***}*;**

***const toggleComplete = async (*task*) => {***

***await axios.put(`http://localhost:5000/tasks/${task.id}`, {***

***title: task.title,***

***completed: !task.completed***

***});***

***fetchTasks();***

***}*;**

***const deleteTask = async (*id*) => {***

***await axios.delete(`http://localhost:5000/tasks/${*id*}`);***

***fetchTasks();***

***}*;**

***return* (**

**<div *style=*{{ padding: 20 }}>**

**<h1>Task Manager</h1>**

**<input**

***value=*{title}**

***onChange=*{(e) *=>* *setTitle*(*e*.*target*.*value*)}**

***placeholder=*"New Task"**

**/>**

**<button *onClick=*{addTask}>Add</button>**

**<ul>**

**{*tasks*.*map*(task *=>* (**

**<li *key=*{*task*.*id*}>**

**<span**

***onClick=*{() *=>* *toggleComplete*(task)}**

***style=*{{**

**textDecoration: *task*.*completed* *?* 'line-through' *:* 'none',**

**cursor: 'pointer'**

**}}>**

**{*task*.*title*}**

**</span>**

**<button *onClick=*{() *=>* *deleteTask*(*task*.*id*)} *style=*{{ marginLeft: 10 }}>Delete</button>**

**</li>**

**))}**

**</ul>**

**</div>**

**);**

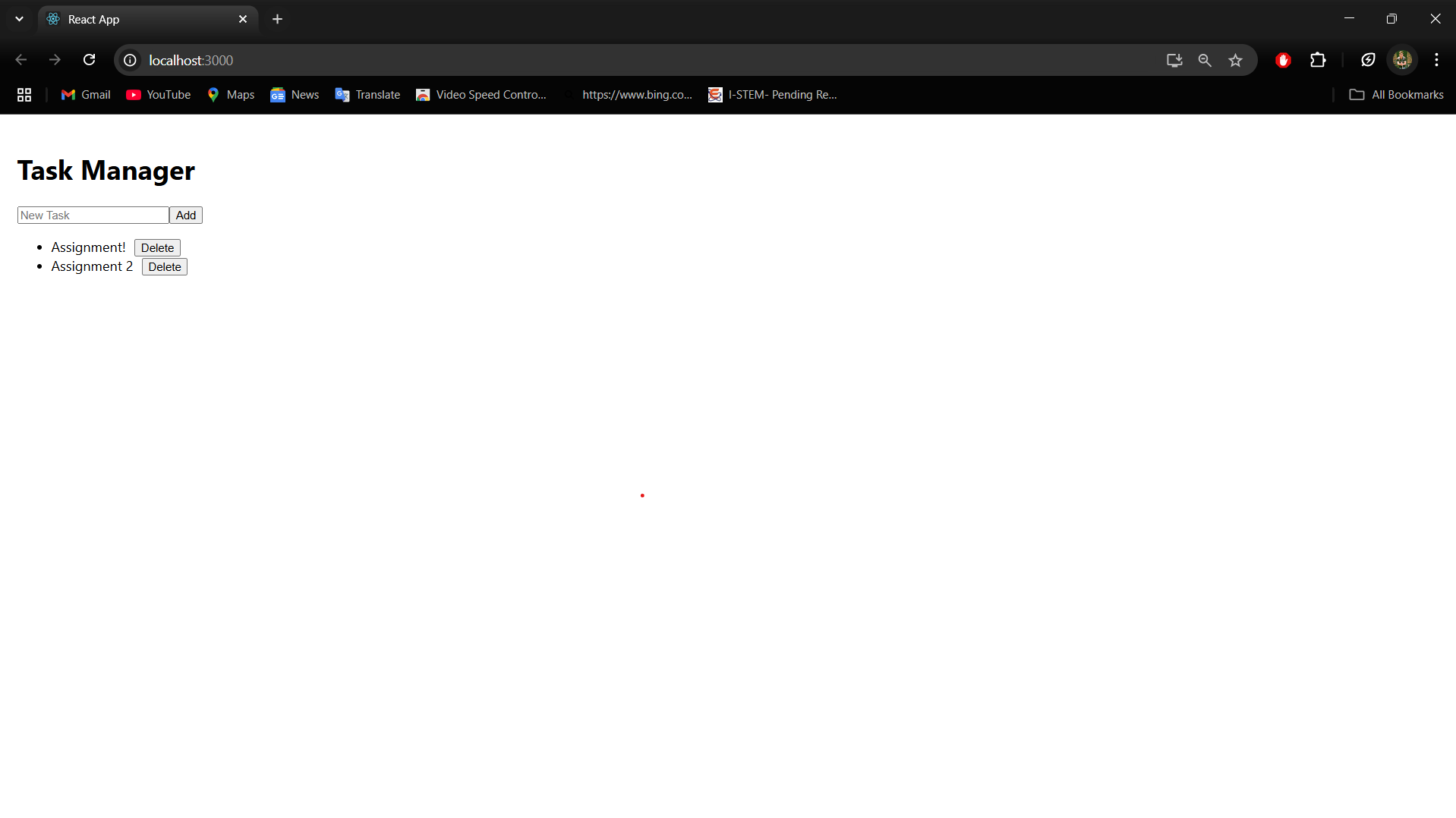
**}**

***export* *default* App;**

**Start React App**

npm start

**Output:**

****

## **9.6Program/Output:**

**app.py**

from flask import Flask, render\_template

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

@app.route("/")

def home():

return render\_template("index.html")

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

app.run(debug=True)

**templates/index.html**

<!DOCTYPE html>

<html>

<head>

<title>My Flask Web Page</title>

</head>

<body>

<h1>Welcome to My Flask App!</h1>

<p>This is a simple web page rendered using Flask.</p>

</body>

</html>

Output:

## 

## **9.7 Conclusion:**

In this project, a dynamic and interactive web page was developed using the Flask framework. The backend was built to handle routing, data processing, and API communication. A rich interface was achieved by integrating the backend with a frontend interface (React), allowing real-time interaction and user control. This project demonstrates the practical application of Rich Internet Application principles using Flask, and reinforces the understanding of full-stack development concepts.

Sure! Below are **viva-style Q&A (like 4.9 Questions)** based on your **Flask + RIA web app project** (Section 9). These questions are conceptual and practical — perfect for exams or submission.

## **9.7 Questions:**

### **1. What is Flask and why is it used in web development?**

**Flask** is a lightweight Python web framework used to build web applications. It is used for its simplicity, flexibility, and support for routing, templating, and integration with databases.

### **2. What does @app.route() do in Flask?**

@app.route() is a decorator in Flask that binds a specific URL to a function, enabling routing and dynamic URL handling.

### **3. What is the purpose of db.create\_all() in a Flask app using SQLAlchemy?**

db.create\_all() creates all the tables defined in your SQLAlchemy models in the connected database.

### **4. How does Flask support the development of Rich Internet Applications?**

Flask provides RESTful APIs and dynamic routing which, when connected with frontend frameworks like React or Vue, enables real-time, interactive applications that avoid full-page reloads.

### **5. What is the difference between Flask and Django?**

Flask is a micro-framework that provides basic tools and flexibility, while Django is a full-stack framework with built-in admin, ORM, and more features out of the box.

### **6. What is CORS and why is Flask-CORS used?**

CORS (Cross-Origin Resource Sharing) allows a web page to make requests to a different domain. Flask-CORS is used to enable frontend apps (like React) to access Flask APIs running on another port or domain.

### **7. What is a REST API and how is it used in this project?**

A REST API uses HTTP methods (GET, POST, PUT, DELETE) to communicate between client and server. In this project, Flask routes are used to expose APIs for task management.

### **8. How is a database connected in Flask using SQLAlchemy?**

Flask is configured with a SQLALCHEMY\_DATABASE\_URI, and SQLAlchemy provides ORM features to interact with the database using Python classes.

### **9. What are templates in Flask and how are they used?**

Templates in Flask (using Jinja2) are HTML files that can include dynamic content using Python logic, enabling dynamic web page generation.

### **10. Why is React used with Flask in this project?**

React is used to build a dynamic, component-based frontend. It communicates with Flask via API calls to fetch and update data, creating a Rich Internet Application experience.