Backend with Flask (Project)

Some Project ideas that can be developed using Flask-python

1. To-Do List App

- Use Flask to create a simple web app.
- Store tasks in an SQLite or SQL Server database.
- Implement CRUD operations (add, update, delete tasks).

2. User Authentication System

- Create a user signup and login system.
- Use SQLAlchemy or PyODBC to store user credentials.
- Implement session-based authentication.

3. Student Management System

- Store student details (name, age, courses).
- Implement API endpoints to manage student records.
- Use Postman to test API calls.

4. Expense Tracker

- Track daily expenses and categorize them.
- Store transactions in a database.
- Display summaries using Flask templates.

5. Simple Blog App

- Allow users to create, read, update, and delete blog posts.
- Store blog data in a database.
- Add a simple front-end using Flask templates.

6. Book Catalog

- Store book details (title, author, genre, published year).
- Implement search and filter options.
- Provide API endpoints for CRUD operations.

7. Weather App

- Fetch and display weather data using an API (e.g., OpenWeatherMap).
- Allow users to search weather by city name.
- Store search history in a database.

8. Contact Manager

- Store and manage contacts (name, phone, email).
- Implement a simple search feature.
- Use a database to persist contacts.

9. URL Shortener

- Generate short URLs for long links.
- Store mappings in a database.
- Redirect users when they visit the short URL.

10. Job Application Tracker

- Allow users to add and update job applications.
- Store job details (company, role, status).
- Show statistics on job applications.

11. Online Polling System

- Create and manage polls.
- Allow users to vote and view results.
- Store poll data in a database.

12. Inventory Management System

- Track stock levels for products.
- Implement user roles (admin vs. viewer).
- Provide REST API endpoints.

Example of a To do list:

This simple to-do list app allows users to:

- View all tasks on the homepage.
- Add a new task using a form.
- Mark tasks as completed by clicking on them.
- Delete tasks when no longer needed.

1. app.py

```
from flask import Flask, render_template, request, redirect, url
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///todo.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Model for the To-Do List
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.String(200), nullable=False)
    completed = db.Column(db.Boolean, default=False)

# Create the database tables before the first request
```

```
def create_tables():
    db.create_all()

# Home Route - Show the index page
@app.route('/')
def index():
    return render_template('index.html')

# Get all tasks (Used by AJAX to auto-refresh tasks)
@app.route('/tasks', methods=['GET'])
def get_tasks():
    tasks = Task.query.all()
    tasks_list = [{'id': task.id, 'content': task.content, 'comp
    return jsonify(tasks_list)

# Add a Task
@app.route('/add', methods=['POST'])
def add_task():
    task_content = request.json.get('content')
    if task_content:
        new_task = Task(content=task_content)
        db.session.add(new_task)
        db.session.commit()
        return jsonify({'message': 'Task added successfully!'}),
    return jsonify({'error': 'Invalid task content'}), 400

# Mark Task as Completed
@app.route('/complete/<int:task_id>', methods=['POST'])
def complete_task(task_id):
    task = Task.query.get(task_id)
    if task:
        task.completed = True
        db.session.commit()
        return jsonify({'message': 'Task marked as completed!'}
    return jsonify({'error': 'Task not found'}), 404
```

```
# Delete a Task
@app.route('/delete/<int:task_id>', methods=['DELETE'])
def delete_task(task_id):
    task = Task.query.get(task_id)
    if task:
        db.session.delete(task)
        db.session.commit()
        return jsonify({'message': 'Task deleted successfully!'}
    return jsonify({'error': 'Task not found'}), 404

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

1. Setting Up Flask and the Database

1.1 Importing Required Libraries

The required libraries are imported to handle web requests, render templates, retrieve user inputs, and interact with the database.

1.2 Initializing Flask App and Database

The Flask application is initialized, and the SQLite database is configured. SQLAlchemy is used as the ORM (Object-Relational Mapping) tool to interact with the database.

1.3 Creating the Task Model

A `Task` model is defined with three attributes:

- `id` (a unique identifier for each task)
- `content` (stores the task description)
- `completed` (a boolean field to track task status)

1.4 Creating the Database Table

A function ensures that the database and its required table are created before handling any requests.

---

2. Defining Routes (API Endpoints)

2.1 Index Route

When a user visits the home page, the application loads the `index.html` template.

2.2 Fetching All Tasks

A route fetches all tasks from the database and returns them in JSON format. This allows the frontend to dynamically update the displayed tasks.

2.3 Adding a Task

A route processes user input, creates a new task, saves it to the database, and returns a success or error message.

2.4 Marking a Task as Completed

A route updates the `completed` status of a task in the database when a user marks it as done.

2.5 Deleting a Task

A route allows users to remove a task from the database permanently.

---

2. Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-s
    <title>To-Do List</title>
    <style>
```

```
        body { font-family: Arial, sans-serif; text-align: cente
        ul { list-style: none; padding: 0; }
        li { padding: 10px; background: #f4f4f4; margin: 5px; di
        .completed { text-decoration: line-through; color: gray
        button { margin-left: 10px; cursor: pointer; }
    </style>
</head>
<body>
    <h2>To-Do List</h2>
    <input type="text" id="taskInput" placeholder="Enter a task"
    <button onclick="addTask()">Add Task</button>

    <ul id="taskList"></ul>

    <script>
        function fetchTasks() {
            fetch('/tasks')
            .then(response => response.json())
            .then(tasks => {
                const taskList = document.getElementById('taskLi
                taskList.innerHTML = '';
                tasks.forEach(task => {
                    const li = document.createElement('li');
                    li.className = task.completed ? 'completed'
                    li.innerHTML = `
                        ${task.content}
                        <div>
                            <button onclick="completeTask(${tasl
                            <button onclick="deleteTask(${task.i
                        </div>
                    `;
                    taskList.appendChild(li);
                });
            });
        }
```

```
        function addTask() {
            const taskInput = document.getElementById('taskInput
            const taskContent = taskInput.value.trim();
            if (taskContent) {
                fetch('/add', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/jso
                    body: JSON.stringify({ content: taskContent
                })
                .then(response => response.json())
                .then(() => {
                    taskInput.value = '';
                    fetchTasks();
                });
            }
        }

        function completeTask(taskId) {
            fetch(`/complete/${taskId}`, { method: 'POST' })
            .then(() => fetchTasks());
        }

        function deleteTask(taskId) {
            fetch(`/delete/${taskId}`, { method: 'DELETE' })
            .then(() => fetchTasks());
        }


        setInterval(fetchTasks, 2000);
        fetchTasks();
    </script>
</body>
</html>
```

3. Frontend (`index.html`)

The frontend consists of an HTML page that interacts with the Flask API using JavaScript.

3.1 Basic HTML Structure

The HTML page includes metadata, title, and styling.

3.2 Task Input and Buttons

An input field is provided for users to enter tasks, along with a button to add them.

3.3 Task List Display

A list element dynamically displays tasks.

3.4 JavaScript for Dynamic Updates

JavaScript fetches the list of tasks from the API, updates the UI, and ensures completed tasks are visually marked.

3.5 Adding a Task

A function sends the task input to the Flask backend, updates the database, and refreshes the task list.

3.6 Completing a Task

A function marks a task as completed by sending a request to the backend.

3.7 Deleting a Task

A function removes a task by sending a request to the backend.

3.8 Auto-Refreshing the Page

The task list is updated automatically every 2 seconds using JavaScript to keep the displayed tasks in sync with the database.

---

4. Running the App

1. **Create the database** by importing the `db` object from the app and running the database creation command.
2. **Run the Flask app** to start the server.
3. **Open the browser** and navigate to the local URL where the app is running.