🎓 Build a To-Do App Using Flask (with UI)

📌 Problem Statement

You must build a **complete To-Do Web Application** using **Flask**.
The application should:

- Allow users to **create, update, delete, and view tasks**
- Store tasks in a **SQLite database**
- Include a **UI of your choice**
  (Bootstrap, Tailwind, plain HTML, your own styling — anything is allowed)

The goal is to demonstrate your ability to combine:

- Python backend
- Flask routing
- HTML templates
- CRUD operations
- Database integration
- Frontend + API interaction

🎯 Objectives

- Build a full-stack Flask application
- Use SQLite with SQLAlchemy or direct SQL
- Design clean HTML pages
- Perform CRUD operations
- Pass data between backend and templates
- Write forms & handle POST requests
- Update tasks dynamically
- Delete tasks with a click

🧩 Features Required

Your application **must include** the following functionalities:

1️⃣ Add a New Task

A form on the UI should allow adding:

- Task title
- Optional description

2️⃣ Show All Tasks

Display them in a list/table:

- Title
- Description
- Status (Pending / Completed)
- Created date

3️⃣ Mark a Task as Completed

A button/icon:

- Should change status from Pending → Completed
- Should update database
- UI must reflect the change

4️⃣ Update/Edit a Task

Student should be able to edit:

- Title
- Description

Use:

- Separate edit page **or** popup **or** in-place editing.

5️⃣ Delete a Task

Add a delete button next to each task.

6️⃣ UI Requirements

Students may choose **any UI design**, such as:

- Bootstrap
- Tailwind
- Material UI
- Pure HTML/CSS
- A ready-made theme

But UI must include:

- Navigation bar
- Clear table/list display
- Buttons for actions
- Clean structure

Database Schema (SQLite)

Table: `tasks`

Column	Type	Description
id	Integer (PK)	Auto-increment
title	Text	Name of task
description	Text	Optional
status	Text	"Pending" or "Completed"
created_at	DateTime	Timestamp

Technical Requirements

1. Use Flask
2. Use SQLite
3. Use `flask_sqlalchemy` or `sqlite3`
4. Create templates using Flask's Jinja2 template engine
5. Use GET/POST methods
6. Organize code clearly (`static` , `templates` , `app.py`)

🧪 Test Cases (Students must test these)

✓ Test Case 1 — Add Task

Input:

- Title: "Buy groceries"
- Description: "Milk, eggs, bread"

Expected:

- Task appears in task list
- Status = Pending

✓ Test Case 2 — Complete Task

Action:

- Click "Complete"

Expected:

- Status changes to Completed
- UI reflects change

✓ Test Case 3 — Edit Task

Change:

- Title: “Buy groceries for the week”

Expected:

- Updated title appears correctly

✓ Test Case 4 — Delete Task

Action:

- Click “Delete”

Expected:

- Task removed from list
- Not present in DB
