Name: Amena Tousiath Unnisa
Project Title: ToDo application using Flask

ABSTRACT

The ToDo application is a web-based task management system designed to streamline the process of organizing and tracking daily tasks. Built using Flask, a lightweight web framework, and SQLAlchemy, an ORM for database interactions, the application offers a simple yet effective solution for users seeking to manage their tasks efficiently. The system features a clean interface that allows users to perform CRUD operations—Create, Read, Update, and Delete tasks—facilitating easy task management.

The project highlights essential web development concepts and best practices, such as routing, form handling, and database integration. It uses an SQLite database to store task information and a well-structured codebase to ensure functionality and maintainability. The ToDo application not only meets its objectives of providing a practical task management tool but also serves as a foundational example for learning web application development with Flask and SQLAlchemy.

Overall, the application provides a solid introduction to web development, demonstrating how modern technologies can be utilized to build functional and user-friendly applications.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

The ToDo Application is a web-based task management tool designed to assist users in organizing and managing their daily tasks efficiently. The application enables users to create, view, update, and delete tasks through a user-friendly web interface. This project leverages the Flask web framework and SQLAlchemy for database interactions to provide a seamless and interactive experience.

1.1 Problem Statement

Managing tasks can be challenging without a systematic approach. Many existing solutions are either too complex or lack essential features for basic task management. There is a need for a straightforward, user-friendly application that allows users to efficiently track and manage their daily tasks.

Challenges

- **Simplicity vs. Functionality**: Balancing the need for a simple user interface with the requirement for robust task management features.
- **Database Management**: Ensuring that the database operations are efficient and handle user data correctly.
- **User Experience**: Designing an interface that is both functional and easy to navigate.

1.2 Project Scope

The project encompasses the development of a web-based ToDo application that includes features for adding, viewing, updating, and deleting tasks. It focuses on providing a basic yet functional task management system.

1.3 Objectives

- Develop a web application that allows users to manage tasks effectively.
- Implement CRUD (Create, Read, Update, Delete) functionality for tasks.
- Utilize Flask and SQLAlchemy to handle backend logic and database operations.
- Ensure a responsive and intuitive user interface using HTML and CSS.

CHAPTER-2

SYSTEM DESIGN

2.1 Architecture Diagram

The system follows a Model-View-Controller (MVC) architecture:

- **Model**: Represents the database schema and data operations handled by SQLAlchemy.
- **View**: HTML templates rendered by Flask to present data to the user.
- **Controller**: Flask routes and handlers that process user requests and interact with the model.

2.2 Component Description

- **Flask App**: The core application logic that handles routing, request processing, and rendering templates.
- **Database**: An SQLite database used for storing task information.
- **Frontend**: HTML, CSS, and JavaScript files for user interaction.

2.3 Database Design

The database consists of a single table `Todo` with the following schema:

- id (INTEGER, PRIMARY KEY): Unique identifier for each task.
- title (TEXT): Title of the task.
- description (TEXT): Description of the task.
- date_created (DATETIME): Timestamp of when the task was created.

2.4 Software Requirements

- **Flask**: A web framework for Python used to build the application.
- **SQLAlchemy**: An ORM (Object-Relational Mapping) tool used for database interactions.
- **SQLite**: A lightweight database used for storing tasks.
- **Python**: The programming language used to develop the application.
- **HTML/CSS**: Technologies used for the frontend interface.
- **JavaScript**: For any dynamic behaviour in the frontend.
- **Text Editor/IDE**: For coding and editing. Examples: VSCode.

2.5 Hardware Requirements

- **Processor**: Any modern processor (e.g., Intel i5 or equivalent).
- **RAM**: At least 4 GB of RAM.
- **Storage**: At least 500 MB of free storage for the development environment.
- **Processor**: Basic server-grade processor.
- **RAM**: Minimum of 1 GB of RAM.
- **Storage**: Sufficient storage to handle the application and database.

CHAPTER-3

IMPLEMENTATION DETAILS

3.1 Code Overview

3.1.1 app.py: This file contains the main application logic. It includes route definitions for handling task operations and interacts with the SQLite database.

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


app = Flask(__name__)
app = Flask(__name__, static_url_path='/static')

app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///D:/env/var/app-
instance/Todo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False
db=SQLAlchemy(app)

#models
#table(database)
class Todo(db.Model):
    sno=db.Column(db.Integer, primary_key=True)
    title=db.Column(db.String(200), nullable=False)
    desc=db.Column(db.String(500),nullable=False)
    date_created=db.Column(db.DateTime,default=datetime.utcnow)

    def __repr__(self)->str:
        return f"{self.sno} - {self.title}"


@app.route('/', methods=['GET', 'POST'])
def hello_world():
    if request.method=='POST':
        title=request.form['title']
        desc=request.form['desc']


        todo=Todo(title=title, desc=desc)
        db.session.add(todo)
        db.session.commit()

    allTodo=Todo.query.all
    return render_template('index.html', allTodo=allTodo)
```

```
@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/show')
def products():
    allTodo=Todo.query.all()
    print(allTodo)
    return 'this is products page'

@app.route('/update/<int:sno>', methods=['GET','POST'])
def update(sno):
    if request.method=='POST':
        title=request.form['title']
        desc=request.form['desc']
        todo=Todo.query.filter_by(sno=sno).first()
        todo.title=title
        todo.desc=desc
        db.session.add(todo)
        db.session.commit()
        return redirect("/")

    todo=Todo.query.filter_by(sno=sno).first()
    return render_template('update.html',todo=todo)

@app.route('/delete/<int:sno>')
def delete(sno):
    todo=Todo.query.filter_by(sno=sno).first()
    db.session.delete(todo)
    db.session.commit()
    return redirect("/")

if __name__=="__main__":
    app.run(debug=True, port=8000)
```

3.1.2 Static Files:

- **CSS**: Located in static/css/style.css, used for styling the application.
- **JavaScript**: Located in static/js/scripts.js, used for interactivity.

3.1.3 Templates:

- **base.html:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>MyTodo</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- Custom CSS for background -->
    <link rel="stylesheet" href="{{
url_for('static',filename='css/style.css')}}">
    <script src="{{ url_for('static',filename='js/test.js') }}"></script>

    <style>
      body {
        background-image: url("{{ url_for('static',
filename='todooo.jpg') }}");
        background-size: cover;
        background-position: center;
        min-height: 100vh;
        margin: 0;
        padding: 0;
      }
    </style>
</head>
<body>

{% block body %}
{% endblock body %}

<!-- Bootstrap Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

- **index.html**: Displays the list of tasks.

```
{% extends 'base.html' %}
{% block body %}

  <nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
      <a class="navbar-brand" href="/">MyTodo</a>
      <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
```

```
        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page"
href="#">Home</a>
            </li>
            <li class="nav-item">

              <a class="nav-link" href="/about">About</a>

            </li>
          </ul>
          <form class="d-flex" role="search">
            <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search">
            <button class="btn btn-outline-primary"
type="submit">Search</button>
          </form>
        </div>
      </div>
    </nav>

  <div class="container my-3">
    <h2>Add a Todo</h2>
    <form action="/" method="POST">
      <div class="mb-3">
        <label for="title" class="form-label">Todo Title</label>
        <input type="text" class="form-control" name="title" id="title"
aria-describedby="emailHelp">
      </div>
      <div class="mb-3">
        <label for="desc" class="form-label">Todo Description</label>
        <input type="text" class="form-control" name="desc" id="desc">
      </div>

      <button type="submit" class="btn btn-primary">Submit</button>
    </form>


  </div>
  <div class="container my-3">
    <h2>Your Todo's</h2>
    {% if allTodo|length==0 %}
    <div class="alert alert-info" role="alert">
      No Todo's found. Add your first TODO now!!
    </div>
    {% else %}
    <table class="table">
      <thead>
```

```
            <tr>
              <th scope="col">SNO</th>
              <th scope="col">Title</th>
              <th scope="col">Description</th>
              <th scope="col">Time</th>
              <th scope="col">Actions</th>
            </tr>
          </thead>
          <tbody>
            {% for todo in allTodo %}
            <tr>
              <th scope="row">{{ loop.index }}</th>
              <td>{{ todo.title }}</td>
              <td>{{ todo.desc }}</td>
              <td>{{ todo.date_created }}</td>
              <td>
                <a href="/update/{{ todo.sno }}" type="button" class="btn
btn-outline-primary btn-sm mx-1">Update</a>
                <a href="/delete/{{ todo.sno }}" type="button" class="btn
btn-outline-primary btn-sm mx-1">Delete</a>
              </td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
        {% endif %}
    </div>
    {% endblock body %}
```

- **update.html**: Allows users to update task details.

```
    {% extends 'base.html' %}

    {% block body %}

      <nav class="navbar navbar-expand-lg bg-body-tertiary">
        <div class="container-fluid">
          <a class="navbar-brand" href="/">MyTodo</a>
          <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse"
id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
              <li class="nav-item">
```

```
            <a class="nav-link active" aria-current="page"
href="#">Home</a>
          </li>
          <li class="nav-item">

            <a class="nav-link" href="/about">About</a>

          </li>
        </ul>
        <form class="d-flex" role="search">
          <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search">
          <button class="btn btn-outline-primary"
type="submit">Search</button>
        </form>
      </div>
    </div>
  </nav>

 <div class="container my-3">
  <h2>Update Todo</h2>
  <form action="/update/{{todo.sno}}" method="POST">
    <div class="mb-3">
      <label for="title" class="form-label">Todo Title</label>
      <input type="text" class="form-control" name="title" id="title"
value="{{todo.title}}" aria-describedby="emailHelp">
    </div>
    <div class="mb-3">
      <label for="desc" class="form-label">Todo Description</label>
      <input type="text" class="form-control" value="{{todo.desc}}"
name="desc" id="desc">
    </div>

    <button type="submit" class="btn btn-primary">Update</button>
  </form>


 </div>
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/js/bootstrap.bundle.min.js"></script>

{% endblock body %}
```

Database Schema

The Todo.db file contains the Todo table with fields for task ID, title, description, and creation date. SQLAlchemy ORM is used to manage interactions with this database.

3.2 Features

Add Task

Users can add new tasks by providing a title and description. The task is then stored in the SQLite database and displayed on the home page.

View Tasks

The home page lists all tasks stored in the database, showing each task's title and creation date. Users can view details by selecting a task.

Update Task

Users can update existing tasks by modifying the title and description. The updated information is saved back to the database.

Delete Task

Users can delete tasks from the list. The task is removed from the database and no longer appears on the home page.

3.3 User Interface Design

Home Page

The home page provides an overview of all tasks. It includes options to add, update, or delete tasks. The design is clean and intuitive, allowing easy navigation.

About Page

The about page contains information about the application, its purpose, and how to use it. It helps users understand the functionality of the application.

Update Page

The update page allows users to modify existing tasks. It features a form pre-filled with the current task details, enabling users to make changes and save them.

CHAPTER-4

TESTING AND VALIDATION

4.1 Testing Methods

1. **Unit Testing:**

   Individual components of the ToDo application were tested to ensure they functioned correctly in isolation. This included testing database interactions to verify that CRUD operations performed as expected and route handlers to ensure they correctly processed requests and returned appropriate responses.

2. **Integration Testing:**

   This testing phase verified that various parts of the application work together seamlessly. For example, it ensured that the integration between the Flask routes and SQLAlchemy ORM was functional, and that the user interface correctly reflected changes made in the database.

3. **User Testing:**

   Usability testing was conducted with a sample group of users to assess the application's effectiveness and user experience. Feedback was collected to identify any usability issues, ensuring that the ToDo application met user needs and provided a smooth, intuitive interface for managing tasks.

4.2 Results

All major features (add, view, update, delete) were tested and confirmed to work as intended. No critical bugs were identified, and the application performed well under normal use conditions.

Below are the following results:

Fig no. 4.2.1 Result1

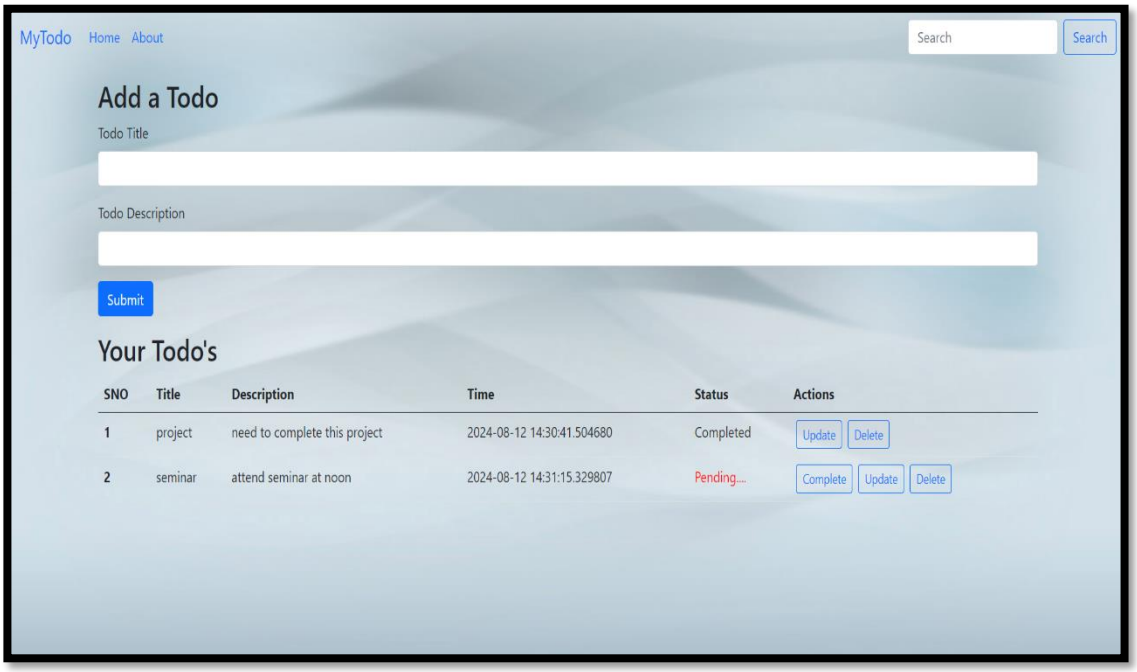

Fig no. 4.2.2 Result2

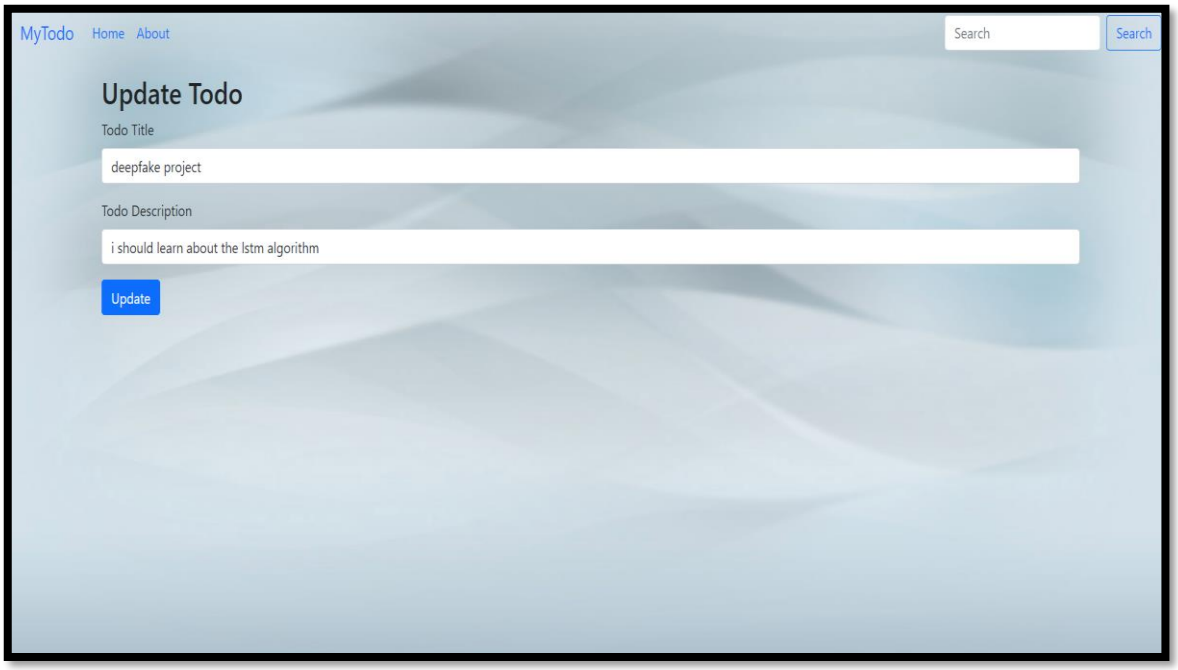

Fig no. 4.2.3 update page

CHAPTER-5
CONCLUSION

The ToDo application exemplifies a robust and user-friendly solution for managing daily tasks. Leveraging Flask for web development and SQLAlchemy for database management, this project integrates core concepts of CRUD operations and web application architecture. The application effectively meets its objectives by allowing users to easily add, view, update, and delete tasks, offering a practical tool for personal productivity. The clean and intuitive interface enhances user experience, while the underlying codebase demonstrates sound software engineering practices.

In summary, the ToDo application is a testament to the potential of Flask and SQLAlchemy in developing functional web applications. The project not only serves as an effective task management tool but also provides a valuable learning experience in web development and database design

CHAPTER-6
REFERENCES

[1] Flask Documentation**: https://flask.palletsprojects.com/en/3.0.x/**

[2] SQLAlchemy Documentation**: https://docs.sqlalchemy.org/en/20/**

[3] Bootstrap**: https://getbootstrap.com/docs/5.3/getting-started/introduction/**