CREATING A SIMPLE TO-DO LIST WEB APP WITH CRUD OPERATIONS IN FLASK (PYTHON)

Executive Summary

This case study explains how to build a basic full-stack web application—a to-do list—using Flask in Python. The focus is on explaining the backend logic for Create, Read, Update, and Delete (CRUD) operations, along with route definitions, form handling, and database integration using SQLite. This project is ideal for students learning web development, MVC architecture, and backend workflows in Python.

---

1. Introduction

Web applications often rely on basic data operations such as creating and displaying lists, editing data, and deleting it. A to-do list is a classic beginner project that teaches core concepts of routing, HTML templates, HTTP methods (GET/POST), and database connectivity. This study builds a minimalist task manager using Flask with step-by-step explanation of each part of the logic.

---

2. Problem Definition

Build a Flask web app with the following functionality:

- Add a new task
- Display all tasks
- Update an existing task
- Delete a task

Tasks must be saved in a persistent database using SQLite.

---

3. Environment Setup

- **Language**: Python 3.x

- **Framework**: Flask
- **Database**: SQLite
- **Templates**: Jinja2 with HTML
- **File Structure**:

```
pgsql
CopyEdit
project/
│ app.py
│ tasks.db
│ templates/
│   └ index.html
```

---

4. Backend Code and Logic Explanation

Basic Flask Setup and Database Model

```
python
CopyEdit
from flask import Flask, render_template, request, redirect
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tasks.db'
db = SQLAlchemy(app)

class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.String(200), nullable=False)
```

Explanation:

- **Flask app setup** with SQLite database
- Task model defines table structure with id and content

Create and Read (Display Tasks)

python

CopyEdit

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        task_content = request.form['content']
        new_task = Task(content=task_content)
        db.session.add(new_task)
        db.session.commit()
        return redirect('/')

    tasks = Task.query.all()
    return render_template('index.html', tasks=tasks)
```

Logic:

- POST: Adds new task
- GET: Fetches all tasks and renders them with Jinja in HTML

Update Task

python

CopyEdit

```
@app.route('/update/<int:id>', methods=['GET', 'POST'])
def update(id):
```

```
    task = Task.query.get_or_404(id)
    if request.method == 'POST':
        task.content = request.form['content']
        db.session.commit()
        return redirect('/')
    return render_template('update.html', task=task)
```

Delete Task

python

CopyEdit

```
@app.route('/delete/<int:id>')
def delete(id):
    task = Task.query.get_or_404(id)
    db.session.delete(task)
    db.session.commit()
    return redirect('/')
```

---

5. HTML Template (index.html)

html

CopyEdit

```
<!DOCTYPE html>
<html>
<head><title>To-Do List</title></head>
<body>
    <h1>Task List</h1>
    <form method="POST">
```

```
    <input type="text" name="content" placeholder="Enter task" required>
    <button type="submit">Add</button>
  </form>
  <ul>
  {% for task in tasks %}
    <li>{{ task.content }}
    <a href="/update/{{ task.id }}">Edit</a>
    <a href="/delete/{{ task.id }}">Delete</a></li>
  {% endfor %}
  </ul>
</body>
</html>
```

---

6. Evaluation and Student Learning Points

Concepts Strengthened:

- Route handling (@app.route)
- Working with HTTP methods
- Connecting to and querying a database
- Using Jinja2 templating
- Structuring backend projects

Common Beginner Mistakes Solved:

- Not committing after DB operations
- Forgetting to create or migrate DB before querying
- Mixing GET and POST logic improperly

---

7. Suggested Enhancements

- Add due dates or priorities to tasks
- Add user login system (Flask-Login)
- Use AJAX to make app more dynamic
- Host on Heroku or Render for deployment practice

---

8. Conclusion

This project is an ideal starting point for web development students. It teaches not only CRUD operations in Flask but also the logic flow of MVC architecture, form handling, and database interaction in a clean and testable way.

---

9. References

- Flask Documentation (flask.palletsprojects.com)
- SQLAlchemy ORM Guide
- Mozilla Web Docs: HTML Forms
- Corey Schafer Flask Series (YouTube)