Flask Assignment

1. Create a Flask app that displays “Hello, World!” on the homepage

```
pip install Flask
```

Now, create a file named `app.py` and add the following code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(debug=True)
```

Run the app using:

```
python app.py
```

2. Build a Flask app with static HTML pages and navigate between them

Create a folder structure like this:

```
/project
    /templates
        index.html
        about.html
    app.py
```

app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

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

if __name__ == '__main__':
    app.run(debug=True)
```

templates/index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Home Page</h1>
    <a href="/about">About</a>
</body>
</html>
```

templates/about.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>About</title>
</head>
<body>
    <h1>About Page</h1>
    <a href="/">Home</a>
</body>
</html>
```

3. Develop a Flask app that uses URL parameters to display dynamic content

Update app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/user/<name>')
def user(name):
    return f'Hello, {name}!'

if __name__ == '__main__':
    app.run(debug=True)
```

4. Create a Flask app with a form that accepts user input and displays it

Update app.py:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('form.html')

@app.route('/greet', methods=['POST'])
def greet():
    name = request.form['name']
    return f'Hello, {name}!'

if __name__ == '__main__':
    app.run(debug=True)
```

templates/form.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Form</title>
</head>
<body>
    <form action="/greet" method="post">
        <label for="name">Enter your name:</label>
        <input type="text" id="name" name="name">
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

5. Implement user sessions in a Flask app to store and display user-specific data

Update app.py:

```
from flask import Flask, render_template, request, session, redirect, url_for

app = Flask(__name__)
app.secret_key = 'your_secret_key'

@app.route('/')
def home():
    if 'username' in session:
        return f'Logged in as {session["username"]}'
    return render_template('login.html')

@app.route('/login', methods=['POST'])
def login():
    session['username'] = request.form['username']
    return redirect(url_for('home'))

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('home'))

if __name__ == '__main__':
    app.run(debug=True)
```

templates/login.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <form action="/login" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

6. Build a Flask app that allows users to upload files and display them on the website

```
pip install Flask
```

Create a folder structure like this:

```
/project
    /uploads
    /templates
        upload.html
    app.py
```

app.py:

```
import os
from flask import Flask, request, redirect, url_for, render_template
from werkzeug.utils import secure_filename

UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/')
def upload_form():
    return render_template('upload.html')

@app.route('/', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return 'No file part'
    file = request.files['file']
    if file.filename == '':
        return 'No selected file'
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        return redirect(url_for('uploaded_file', filename=filename))
    return 'File not allowed'

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return f'File uploaded successfully: {filename}'

if __name__ == '__main__':
    app.run(debug=True)
```

templates/upload.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Upload File</title>
</head>
<body>
    <h1>Upload a File</h1>
    <form method="post" enctype="multipart/form-data">
        <input type="file" name="file">
```

```
        <input type="submit" value="Upload">
    </form>
</body>
</html>
```

7. Integrate a SQLite database with Flask to perform CRUD operations on a list of items

```
pip install Flask-SQLAlchemy
```

Create a folder structure like this:

```
/project
    /templates
        index.html
    app.py
```

app.py:

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

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

class Item(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)

@app.route('/')
def index():
    items = Item.query.all()
    return render_template('index.html', items=items)

@app.route('/add', methods=['POST'])
def add_item():
    name = request.form['name']
    new_item = Item(name=name)
    db.session.add(new_item)
    db.session.commit()
    return redirect(url_for('index'))

@app.route('/delete/<int:id>')
def delete_item(id):
    item = Item.query.get(id)
    db.session.delete(item)
    db.session.commit()
    return redirect(url_for('index'))

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

templates/index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Items</title>
</head>
<body>
    <h1>Items</h1>
    <form action="/add" method="post">
        <input type="text" name="name" placeholder="Item name">
        <input type="submit" value="Add">
    </form>
    <ul>
        {% for item in items %}
            <li>{{ item.name }} <a href="/delete/{{ item.id }}">Delete</a></li>
        {% endfor %}
```

```
    </ul>
</body>
</html>
```

8. Implement user authentication and registration in a Flask app using Flask-Login

```
pip install Flask-Login Flask-SQLAlchemy
```

Create a folder structure like this:

```
/project
    /templates
        login.html
        register.html
        home.html
    app.py
```

app.py:

```
from flask import Flask, render_template, redirect, url_for, request
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SECRET_KEY'] = 'your_secret_key'
db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and user.password == password:
            login_user(user)
            return redirect(url_for('home'))
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        new_user = User(username=username, password=password)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/home')
@login_required
def home():
    return f'Hello, {current_user.username}!'

@app.route('/logout')
@login_required
def logout():
    logout_user()
```

```
    return redirect(url_for('login'))

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

templates/login.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password">
        <input type="submit" value="Login">
    </form>
    <a href="/register">Register</a>
</body>
</html>
```

templates/register.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
</head>
<body>
    <h1>Register</h1>
    <form method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password">
        <input type="submit" value="Register">
    </form>
    <a href="/login">Login</a>
</body>
</html>
```

templates/home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome, {{ current_user.username }}!</h1>
    <a href="/logout">Logout</a>
</body>
</html>
```

9. Create a RESTful API using Flask to perform CRUD operations on resources like books or movies

```
pip install Flask Flask-RESTful Flask-SQLAlchemy
```

Create a folder structure like this:

```
/project
    app.py
```

app.py:

```
from flask import Flask, request
from flask_restful import Resource, Api
from flask_sqlalchemy import SQLAlchemy

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

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(80), nullable=False)
    author = db.Column(db.String(80), nullable=False)

class BookResource(Resource):
    def get(self, book_id):
        book = Book.query.get(book_id)
        if book:
            return {'id': book.id, 'title': book.title, 'author': book.author}
        return {'message': 'Book not found'}, 404

    def put(self, book_id):
        data = request.get_json()
        book = Book.query.get(book_id)
        if book:
            book.title = data['title']
            book.author = data['author']
            db.session.commit()
            return {'message': 'Book updated'}
        return {'message': 'Book not found'}, 404

    def delete(self, book_id):
        book = Book.query.get(book_id)
        if book:
            db.session.delete(book)
            db.session.commit()
            return {'message': 'Book deleted'}
        return {'message': 'Book not found'}, 404

class BookListResource(Resource):
    def get(self):
        books = Book.query.all()
        return [{'id': book.id, 'title': book.title, '
```

10. Design a Flask app with proper error handling for 404 and 500 errors

```
/project
    /templates
        404.html
        500.html
    app.py
```

app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return 'Welcome to the Home Page!'

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

if __name__ == '__main__':
    app.run(debug=True)
```

templates/404.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>404 Not Found</title>
</head>
<body>
    <h1>404 Not Found</h1>
    <p>The page you are looking for does not exist.</p>
    <a href="/">Go to Home</a>
</body>
</html>
```

templates/500.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>500 Internal Server Error</title>
</head>
<body>
    <h1>500 Internal Server Error</h1>
    <p>Something went wrong on our end.</p>
    <a href="/">Go to Home</a>
</body>
</html>
```

11. Create a real-time chat application using Flask-SocketIO

```
pip install Flask-SocketIO
```

Create a folder structure like this:

```
/project
    /templates
        chat.html
    app.py
```

app.py:

```
from flask import Flask, render_template
from flask_socketio import SocketIO, send

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
socketio = SocketIO(app)

@app.route('/')
def home():
    return render_template('chat.html')

@socketio.on('message')
def handle_message(msg):
    send(msg, broadcast=True)

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

templates/chat.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Chat</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.0/socket.io.js"></script>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const socket = io();
            const form = document.getElementById('form');
            const input = document.getElementById('input');
            const messages = document.getElementById('messages');

            form.addEventListener('submit', (e) => {
                e.preventDefault();
                if (input.value) {
                    socket.send(input.value);
                    input.value = '';
                }
            });

            socket.on('message', (msg) => {
                const item = document.createElement('li');
                item.textContent = msg;
                messages.appendChild(item);
            });
        });
    </script>
</head>
<body>
    <ul id="messages"></ul>
    <form id="form" action="">
        <input id="input" autocomplete="off"><button>Send</button>
    </form>
</body>
</html>
```

12. Build a Flask app that updates data in real-time using WebSocket connections

eate a folder structure like this:

```
/project
    /templates
        counter.html
    app.py
```

app.py:

```
from flask import Flask, render_template
from flask_socketio import SocketIO

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
socketio = SocketIO(app)

counter = 0

@app.route('/')
def home():
    return render_template('counter.html')

@socketio.on('increment')
def handle_increment():
    global counter
    counter += 1
    socketio.emit('update', counter)

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

templates/counter.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Counter</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.0/socket.io.js"></script>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const socket = io();
            const button = document.getElementById('button');
            const counter = document.getElementById('counter');

            button.addEventListener('click', () => {
                socket.emit('increment');
            });

            socket.on('update', (count) => {
                counter.textContent = count;
            });
        });
    </script>
</head>
<body>
    <h1>Counter: <span id="counter">0</span></h1>
    <button id="button">Increment</button>
</body>
</html>
```

13. Implement notifications in a Flask app using websockets to notify users of updates

We'll extend the previous example to include notifications.

Create a folder structure like this:

```
/project
    /templates
        notifications.html
    app.py
```

app.py:

```
from flask import Flask, render_template
from flask_socketio import SocketIO

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
socketio = SocketIO(app)

@app.route('/')
def home():
    return render_template('notifications.html')

@socketio.on('notify')
def handle_notify(message):
    socketio.emit('notification', message)

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

templates/notifications.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Notifications</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.0/socket.io.js"></script>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const socket = io();
            const button = document.getElementById('button');
            const notifications = document.getElementById('notifications');

            button.addEventListener('click', () => {
                const message = 'New notification at ' + new Date().toLocaleTimeString();
                socket.emit('notify', message);
            });

            socket.on('notification', (message) => {
                const item = document.createElement('li');
                item.textContent = message;
                notifications.appendChild(item);
            });
        });
    </script>
</head>
<body>
    <h1>Notifications</h1>
    <button id="button">Send Notification</button>
    <ul id="notifications"></ul>
</body>
</html>
```