Lab Exercise for Python

1. Create a basic Flask application with a route that returns “Hello, World!” when accessed via the browser.

Step 1: Set Up Your Environment

1.1 Install Python

Ensure Python is installed on your system. You can download it from the official website:

During installation, make sure to check the box that says **"Add Python to PATH"**.

1.2 Install Flask

Flask is a lightweight Python web framework. To install it, open your terminal or command prompt and run:

```
pip install flask
```

📝 Step 2: Create the Flask Application

2.1 Create a New Python File

Create a new file named `app.py` in your chosen directory.

2.2 Write the Code

Open `app.py` in your text editor 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)
```

This code creates a basic Flask application with a single route (`/`) that returns "Hello, World!" when accessed.

□ Step 3: Run the Application

3.1 Open the Command Line

- **Windows**: Press `Win + R`, type `cmd`, and press Enter.
- **macOS/Linux**: Open the Terminal application.

3.2 Navigate to the Script Directory

Use the `cd` command to change to the directory where your `app.py` file is located. For example:

```
cd path/to/your/script
```

3.3 Run the Application

Execute the script by typing:

```
python app.py
```

If you have multiple versions of Python installed, you might need to use

```
python3 app.py
```

This will start the Flask development server.DigitalOcean

✅ Step 4: Access the Application

Once the server is running, open your web browser and navigate to:

```
http://127.0.0.1:5000/
```

You should see the message **"Hello, World!"** displayed in your browser.

2. Develop a web form using HTML and Flask to capture user input (e.g., name, email) and display the input on another page

Step 2: Create the Flask Application

2.1 Create a New Directory for Your Project

Open your terminal or command prompt and run:

```
mkdir flask_form_example
cd flask_form_example
```

2.2 Create the Python Script

Create a new file named `app.py` in the `flask_form_example` directory and add the following code:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        return render_template('result.html', name=name, email=email)
    return render_template('form.html')

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

This script defines two routes:Medium+7Stack Overflow+7Stack Overflow+7

- `/` (GET and POST): Displays the form and handles form submissions.
- `/result` (GET): Displays the submitted data.Stack Overflow+2Medium+2Stack Overflow+2

2.3 Create the HTML Templates

Inside the `flask_form_example` directory, create a folder named `templates`. Inside the `templates` folder, create two HTML files:

form.html

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

```
<body>
    <h1>Contact Form</h1>
    <form method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Submitted Data</title>
</head>
<body>
    <h1>Submitted Data</h1>
    <p><strong>Name:</strong> {{ name }}</p>
    <p><strong>Email:</strong> {{ email }}</p>
    <a href="/">Go Back</a>
</body>
</html>
```

These templates define the form and the page that displays the submitted data.

3. Write a Flask application that connects to a MySQL database using MySQL-connector or SQL Alchemy. Create a table called students with columns id, name, age, and grade. Insert records and fetch data from the database to display on a webpage.

MySQL database using Flask-SQLAlchemy. This application will:

- Connect to a MySQL database.
- Create a `students` table with columns: `id`, `name`, `age`, and `grade`.
- Insert records into the table.
- Fetch and display the data on a webpage.

🛠 Step 1: Set Up Your Environment

1.1 Install Python and MySQL

Ensure Python and MySQL are installed on your system.

- **Python**: Download and install from python.org.
- **MySQL**: Download and install from mysql.com.

1.2 Install Required Python Packages

Open your terminal or command prompt and run:

```
pip install flask flask_sqlalchemy mysql-connector-python
```

📝 Step 2: Create the Flask Application

2.1 Create the Project Directory

```
mkdir flask_mysql_app
cd flask_mysql_app
```

2.2 Create the Flask Application Script

Create a file named `app.py` and add the following code:

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/student_db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    grade = db.Column(db.String(2), nullable=False)

    def __repr__(self):
        return f'<Student {self.name}>'

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

if __name__ == '__main__':
    with app.app_context():
```

```
        db.create_all()
    app.run(debug=True)
```

Replace `'root:password@localhost/student_db'` with your actual MySQL credentials and database name.

2.3 Create the HTML Template

Create a folder named `templates` and inside it, create a file named `index.html` with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Student Records</title>
</head>
<body>
    <h1>Student Records</h1>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Age</th>
            <th>Grade</th>
        </tr>
        {% for student in students %}
        <tr>
            <td>{{ student.id }}</td>
            <td>{{ student.name }}</td>
            <td>{{ student.age }}</td>
            <td>{{ student.grade }}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

□ Step 3: Run the Application

3.1 Start the Flask Development Server

In your terminal or command prompt, navigate to the project directory and run:

```
python app.py
```

3.2 Access the Application

Open your web browser and go to:

```
http://127.0.0.1:5000/
```

You should see a table displaying the student records.

📁 Step 4: Insert Sample Data into the Database

To insert sample data into your `students` table, you can use the following Python script:

```
import mysql.connector

db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    database="student_db"
)

cursor = db.cursor()

cursor.execute("""
    CREATE TABLE IF NOT EXISTS students (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100),
        age INT,
        grade VARCHAR(2)
    )
""")

cursor.executemany("""
    INSERT INTO students (name, age, grade)
    VALUES (%s, %s, %s)
""", [
    ('Alice', 20, 'A'),
    ('Bob', 22, 'B'),
    ('Charlie', 23, 'C')
])

db.commit()
cursor.close()
db.close()
```

Replace `"root"`, `"password"`, and `"student_db"` with your actual MySQL credentials and database name.

4. Implement HTML Dropdown to display dynamic options, Dynamic HTML tables, product list and product details using the flask.

Step 1: Set Up Your Environment

1.1 Install Required Packages

Ensure you have Python installed, then install Flask and MySQL connector:

```
pip install flask flask_sqlalchemy mysql-connector-python
```

1.2 Set Up MySQL Database

Create a MySQL database named `product_db` and a table named `products`:

```
CREATE DATABASE product_db;

USE product_db;

CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    description TEXT,
    price DECIMAL(10, 2)
);

INSERT INTO products (name, description, price) VALUES
('Product A', 'Description of Product A', 10.99),
('Product B', 'Description of Product B', 20.99),
('Product C', 'Description of Product C', 30.99);
```

📝 Step 2: Create the Flask Application

2.1 Create the Flask Application Script

Create a file named `app.py` and add the following code:

```
from flask import Flask, render_template, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/product_db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text, nullable=False)
    price = db.Column(db.Numeric(10, 2), nullable=False)

@app.route('/')
def index():
```

```
    products = Product.query.all()
    return render_template('index.html', products=products)

@app.route('/product/<int:product_id>')
def product_detail(product_id):
    product = Product.query.get_or_404(product_id)
    return render_template('product_detail.html', product=product)

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

Replace `'root:password@localhost/product_db'` with your actual MySQL credentials.

2.2 Create the HTML Templates

Create a folder named `templates` and inside it, create two HTML files:

index.html

```
html
CopyEdit
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product List</title>
</head>
<body>
    <h1>Product List</h1>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Price</th>
            <th>Details</th>
        </tr>
        {% for product in products %}
        <tr>
            <td>{{ product.id }}</td>
            <td>{{ product.name }}</td>
            <td>{{ product.price }}</td>
            <td><a href="{{ url_for('product_detail', product_id=product.id)
}}">View Details</a></td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

product_detail.html

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

```
    <title>{{ product.name }}</title>
</head>
<body>
    <h1>{{ product.name }}</h1>
    <p><strong>Description:</strong> {{ product.description }}</p>
    <p><strong>Price:</strong> ${{ product.price }}</p>
    <a href="{{ url_for('index') }}">Back to Product List</a>
</body>
</html>
```

Step 3: Run the Application

3.1 Start the Flask Development Server

In your terminal, navigate to the project directory and run:

```
python app.py
```

3.2 Access the Application

Open your web browser and go to:

```
http://127.0.0.1:5000/
```

You should see the product list. Clicking on "View Details" will show more information about each product.

6. Create a simple MySQL database and connect it to a Flask application. Develop a form that inserts user data into the database.

Step 1: Set Up Your Environment

1.1 Install Required Packages

Ensure Python is installed, then install Flask and MySQL connector:

```
pip install flask flask_mysqldb
```

1.2 Set Up MySQL Database

Create a MySQL database named `user_db` and a table named `users`:

```
CREATE DATABASE user_db;

USE user_db;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

📝 Step 2: Create the Flask Application

2.1 Create the Flask Application Script

Create a file named `app.py` and add the following code:

```
from flask import Flask, render_template, request
from flask_mysqldb import MySQL

app = Flask(__name__)

app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'password'
app.config['MYSQL_DB'] = 'user_db'

mysql = MySQL(app)

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

@app.route('/add_user', methods=['POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        cursor = mysql.connection.cursor()
        cursor.execute('''INSERT INTO users (name, email) VALUES (%s, %s)''',
(name, email))
        mysql.connection.commit()
        cursor.close()
        return 'User added successfully!'
    return render_template('index.html')

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

Replace `'password'` with your actual MySQL password.

2.2 Create the HTML Form

Create a folder named `templates` and inside it, create a file named `index.html` with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add User</title>
</head>
<body>
    <h1>Add User</h1>
    <form method="POST" action="/add_user">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

□ Step 3: Run the Application

3.1 Start the Flask Development Server

In your terminal, navigate to the project directory and run:

```
python app.py
```

3.2 Access the Application

Open your web browser and go to:

```
http://127.0.0.1:5000/
```

You should see the form to add a user. After submitting the form, the user data will be inserted into the `users` table in your MySQL database.

7. Create a Flask app that allows users to add, update, and delete records from a database using a web interface.

Step 1: Set Up Your Environment

1.1 Install Required Packages

Ensure Python is installed, then install Flask and MySQL connector:

```
pip install flask flask_sqlalchemy mysql-connector-python
```

1.2 Set Up MySQL Database

Create a MySQL database named `user_db` and a table named `users`:

```
CREATE DATABASE user_db;

USE user_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

📝 Step 2: Create the Flask Application

2.1 Create the Flask Application Script

Create a file named `app.py` and add the following code:

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

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/user_db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)

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

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

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

```
def edit_user(id):
    user = User.query.get_or_404(id)
    if request.method == 'POST':
        user.name = request.form['name']
        user.email = request.form['email']
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('edit_user.html', user=user)

@app.route('/delete/<int:id>', methods=['POST'])
def delete_user(id):
    user = User.query.get_or_404(id)
    db.session.delete(user)
    db.session.commit()
    return redirect(url_for('index'))

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

Replace `'root:password@localhost/user_db'` with your actual MySQL credentials.

2.2 Create the HTML Templates

Create a folder named `templates` and inside it, create the following HTML files:

index.html

```
html
CopyEdit
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User List</title>
</head>
<body>
    <h1>User List</h1>
    <a href="{{ url_for('add_user') }}">Add New User</a>
    <ul>
        {% for user in users %}
        <li>
            {{ user.name }} ({{ user.email }})
            <a href="{{ url_for('edit_user', id=user.id) }}">Edit</a>
            <form action="{{ url_for('delete_user', id=user.id) }}"
method="POST" style="display:inline;">
                <button type="submit">Delete</button>
            </form>
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

add_user.html

```
html
CopyEdit
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add User</title>
</head>
<body>
    <h1>Add User</h1>
    <form method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
        <button type="submit">Add User</button>
    </form>
    <a href="{{ url_for('index') }}">Back to User List</a>
</body>
</html>
```

edit_user.html

```
html
CopyEdit
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Edit User</title>
</head>
<body>
    <h1>Edit User</h1>
    <form method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" value="{{ user.name }}"
required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" value="{{ user.email }}"
required><br><br>
        <button type="submit">Update User</button>
    </form>
    <a href="{{ url_for('index') }}">Back to User List</a>
</body>
</html>
```

□ Step 3: Run the Application

3.1 Start the Flask Development Server

In your terminal, navigate to the project directory and run:

```
python app.py
```

3.2 Access the Application

Open your web browser and go to:

```
http://127.0.0.1:5000/
```

You should see the user list. You can add, edit, and delete users through the web interface.

8. Use Bootstrap with Flask to style the CRUD application developed in above task. Create a form with proper layout and design. Use Bootstrap classes for buttons, tables, and forms.

Step 1: Set Up Your Environment

Ensure you have the following installed:

- Python 3.x
- Flask
- MySQL Server
- Bootstrap (via CDN)

Install Flask and MySQL connector:

```
pip install flask flask_sqlalchemy mysql-connector-python
```

📝 Step 2: Flask Application with Bootstrap Integration

2.1 Flask Application Script (`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'] =
'mysql+mysqlconnector://root:password@localhost/user_db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)

@app.route('/')
def index():
    users = User.query.all()
    return render_template('index.html', users=users)
```

```
@app.route('/add', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('add_user.html')

@app.route('/edit/<int:id>', methods=['GET', 'POST'])
def edit_user(id):
    user = User.query.get_or_404(id)
    if request.method == 'POST':
        user.name = request.form['name']
        user.email = request.form['email']
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('edit_user.html', user=user)

@app.route('/delete/<int:id>', methods=['POST'])
def delete_user(id):
    user = User.query.get_or_404(id)
    db.session.delete(user)
    db.session.commit()
    return redirect(url_for('index'))

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

Replace `'root:password@localhost/user_db'` with your actual MySQL credentials.

2.2 HTML Templates with Bootstrap Styling

templates/layout.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}User Management{% endblock %}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css
" rel="stylesheet">
</head>
<body>
    <div class="container my-4">
        {% block content %}
        {% endblock %}
    </div>
```

```
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.mi
n.js"></script>
</body>
</html>
```

templates/index.html

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

{% block title %}User List{% endblock %}

{% block content %}
<h1 class="mb-4">User List</h1>
<a href="{{ url_for('add_user') }}" class="btn btn-primary mb-3">Add New
User</a>
<table class="table table-striped">
    <thead>
        <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for user in users %}
        <tr>
            <td>{{ user.name }}</td>
            <td>{{ user.email }}</td>
            <td>
                <a href="{{ url_for('edit_user', id=user.id) }}" class="btn
btn-warning btn-sm">Edit</a>
                <form action="{{ url_for('delete_user', id=user.id) }}"
method="POST" style="display:inline;">
                    <button type="submit" class="btn btn-danger btn-
sm">Delete</button>
                </form>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
{% endblock %}
```

templates/add_user.html

```
html
CopyEdit
{% extends 'layout.html' %}

{% block title %}Add User{% endblock %}

{% block content %}
<h1 class="mb-4">Add User</h1>
<form method="POST">
    <div class="mb-3">
        <label for="name" class="form-label">Name</label>
```

```
        <input type="text" id="name" name="name" class="form-control"
required>
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" id="email" name="email" class="form-control"
required>
    </div>
    <button type="submit" class="btn btn-success">Add User</button>
    <a href="{{ url_for('index') }}" class="btn btn-secondary">Back</a>
</form>
{% endblock %}
```

templates/edit_user.html

```
html
CopyEdit
{% extends 'layout.html' %}

{% block title %}Edit User{% endblock %}

{% block content %}
<h1 class="mb-4">Edit User</h1>
<form method="POST">
    <div class="mb-3">
        <label for="name" class="form-label">Name</label>
        <input type="text" id="name" name="name" class="form-control"
value="{{ user.name }}" required>
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" id="email" name="email" class="form-control"
value="{{ user.email }}" required>
    </div>
    <button type="submit" class="btn btn-warning">Update User</button>
    <a href="{{ url_for('index') }}" class="btn btn-secondary">Back</a>
</form>
{% endblock %}
```

□ Step 3: Run the Application

1. **Start the Flask Development Server**:

   ```
   python app.py
   ```

2. **Access the Application**:

   Open your web browser and navigate to:

   ```
   http://127.0.0.1:5000/
   ```

   You should see the styled user list with options to add, edit, and delete users.

9. Implement input validation for the form created in the previous exercise.

Step 1: Install Required Packages

Ensure you have the necessary packages installed:

```
pip install flask flask_sqlalchemy flask-wtf wtforms
```

Step 2: Update Flask Application with WTForms

2.1 Define the Form with Validation

Create a new file named `forms.py` and define the form with validation rules:

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired, Email, Length

class UserForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired(), Length(max=100)])
    email = StringField('Email', validators=[DataRequired(), Email(),
Length(max=100)])
    submit = SubmitField('Submit')
```

2.2 Update Flask Application Script (`app.py`)

Modify your `app.py` to use the `UserForm`:

```
from flask import Flask, render_template, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from forms import UserForm

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/user_db'
app.config['SECRET_KEY'] = 'your_secret_key'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)

@app.route('/add', methods=['GET', 'POST'])
def add_user():
    form = UserForm()
    if form.validate_on_submit():
```

```
        new_user = User(name=form.name.data, email=form.email.data)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('add_user.html', form=form)

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

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

Step 3: Update HTML Templates to Display Validation Errors

3.1 Update `add_user.html`

Modify `add_user.html` to display form validation errors:

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

{% block title %}Add User{% endblock %}

{% block content %}
<h1 class="mb-4">Add User</h1>
<form method="POST">
    {{ form.hidden_tag() }}
    <div class="mb-3">
        <label for="name" class="form-label">Name</label>
        {{ form.name(class="form-control") }}
        {% if form.name.errors %}
            <div class="invalid-feedback">
                {% for error in form.name.errors %}
                    <p>{{ error }}</p>
                {% endfor %}
            </div>
        {% endif %}
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        {{ form.email(class="form-control") }}
        {% if form.email.errors %}
            <div class="invalid-feedback">
                {% for error in form.email.errors %}
                    <p>{{ error }}</p>
                {% endfor %}
            </div>
        {% endif %}
    </div>
    <button type="submit" class="btn btn-success">Add User</button>
```

```
    <a href="{{ url_for('index') }}" class="btn btn-secondary">Back</a>
</form>
{% endblock %}
```

3.2 Update `index.html`

Ensure `index.html` is set up to display the list of users:

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

{% block title %}User List{% endblock %}

{% block content %}
<h1 class="mb-4">User List</h1>
<a href="{{ url_for('add_user') }}" class="btn btn-primary mb-3">Add New
User</a>
<table class="table table-striped">
    <thead>
        <tr>
            <th>Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        {% for user in users %}
        <tr>
            <td>{{ user.name }}</td>
            <td>{{ user.email }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Step 4: Run the Application

1. **Start the Flask Development Server**:

   ```
   python app.py
   ```

2. **Access the Application**:

   Open your web browser and navigate to:

   ```
   http://127.0.0.1:5000/
   ```

   You should see the user list. Click on "Add New User" to access the form with validation.

10. Develop a login page using Flask that includes session management and secure password storage using hashing

Step 1: Install Required Packages

Install the necessary Python packages:

```
pip install flask flask_sqlalchemy flask_login flask_bcrypt flask_wtf
```

Step 2: Configure Flask Application

Create a Flask application with the following structure:

```
/your_project
    /templates
        login.html
        dashboard.html
    /static
        /css
            styles.css
    app.py
    models.py
    forms.py
```

Step 3: Define the User Model

In `models.py`, define the `User` model with hashed password storage:

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(256), nullable=False)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

Step 4: Create Forms for Login and Registration

In `forms.py`, create forms using Flask-WTF:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

class RegisterForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Register')
```

Step 5: Set Up Flask Application

In `app.py`, configure the Flask application:

```
from flask import Flask, render_template, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, login_user, login_required,
logout_user, current_user
from flask_bcrypt import Bcrypt
from models import db, User
from forms import LoginForm, RegisterForm

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'

db.init_app(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'

@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 current_user.is_authenticated:
        return redirect(url_for('dashboard'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and user.check_password(form.password.data):
            login_user(user)
```

```
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Login failed. Check your username and/or password.',
'danger')
    return render_template('login.html', form=form)

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

@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('You have been logged out.', 'info')
    return redirect(url_for('login'))

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

Step 6: Create HTML Templates

templates/login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
</head>
<body>
    <h2>Login</h2>
    <form method="POST">
        {{ form.hidden_tag() }}
        <div>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
                <span>{{ error }}</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
                <span>{{ error }}</span><br>
            {% endfor %}
        </div>
        <div>
            {{ form.submit() }}
```

```
        </div>
    </form>
    <p>Don't have an account? <a href="{{ url_for('register') }}">Register
here</a></p>
</body>
</html>
```

templates/dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
</head>
<body>
    <h2>Welcome, {{ current_user.username }}!</h2>
    <p>This is your dashboard.</p>
    <a href="{{ url_for('logout') }}">Logout</a>
</body>
</html>
```

Step 7: Create a Simple CSS File

In `static/css/styles.css`, add basic styling:

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

h2 {
    color: #333;
}

form {
    margin-bottom: 20px;
}

div {
    margin-bottom: 10px;
}

span {
    color: red;
}
```

Step 8: Run the Application

1. Initialize the database:

   ```
   from app import db
   db.create_all()
   ```

2. Run the Flask application:

   ```
   python app.py
   ```

3. Access the application in your browser at `http://127.0.0.1:5000/login`.

Security Considerations

- **Password Hashing**: Use `generate_password_hash` and `check_password_hash` from `werkzeug.security` to securely hash and verify passwords. This ensures that passwords are not stored in plain text and are protected against common attacks.
- **Session Management**: Utilize Flask-Login's session management features to handle user sessions securely. This includes managing user authentication states and protecting routes that require login.
- **Form Validation**: Implement form validation using Flask-WTF to ensure that user inputs are properly validated before processing. This helps prevent malicious inputs and enhances security.

11. Develop a REST API using Flask that performs CRUD operations on a student's table in the MySQL database.

Step 1: Install Required Packages

Ensure you have the necessary Python packages installed:

```
pip install flask flask_sqlalchemy flask_marshmallow mysql-connector-python
```

Step 2: Set Up MySQL Database

Create a MySQL database and a table named `students`:

```
CREATE DATABASE school;

USE school;
```

```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    grade VARCHAR(10) NOT NULL
);
```

Step 3: Define Flask Application and Models

Create a file named `app.py` and define the Flask application, database connection, and models:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/school'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
ma = Marshmallow(app)

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    grade = db.Column(db.String(10), nullable=False)

    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

class StudentSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = Student

db.create_all()
```

Step 4: Implement CRUD Operations

Define routes for CRUD operations:

```
@app.route('/students', methods=['POST'])
def add_student():
    name = request.json['name']
    age = request.json['age']
    grade = request.json['grade']
    new_student = Student(name, age, grade)
    db.session.add(new_student)
    db.session.commit()
```

```
    return student_schema.jsonify(new_student), 201

@app.route('/students', methods=['GET'])
def get_students():
    students = Student.query.all()
    return students_schema.jsonify(students)

@app.route('/students/<int:id>', methods=['GET'])
def get_student(id):
    student = Student.query.get_or_404(id)
    return student_schema.jsonify(student)

@app.route('/students/<int:id>', methods=['PUT'])
def update_student(id):
    student = Student.query.get_or_404(id)
    student.name = request.json['name']
    student.age = request.json['age']
    student.grade = request.json['grade']
    db.session.commit()
    return student_schema.jsonify(student)

@app.route('/students/<int:id>', methods=['DELETE'])
def delete_student(id):
    student = Student.query.get_or_404(id)
    db.session.delete(student)
    db.session.commit()
    return '', 204
```

Step 5: Run the Application

To run the application, execute the following command:

```
python app.py
```

The Flask development server will start, and you can access the API at http://127.0.0.1:5000.

12. Implement API endpoints for creating, retrieving, updating, and deleting student records

Step 1: Install Required Packages

Ensure you have the necessary Python packages installed:

```
pip install flask flask_sqlalchemy flask_marshmallow mysql-connector-python
```

Step 2: Set Up MySQL Database

Create a MySQL database and a table named `students`:

```
CREATE DATABASE school;

USE school;

CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    grade VARCHAR(10) NOT NULL
);
```

Step 3: Define Flask Application and Models

Create a file named `app.py` and define the Flask application, database connection, and models:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:password@localhost/school'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
ma = Marshmallow(app)

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    grade = db.Column(db.String(10), nullable=False)

    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

class StudentSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = Student

db.create_all()
```

Step 4: Implement CRUD Operations

Define routes for CRUD operations:

```
@app.route('/students', methods=['POST'])
def add_student():
    name = request.json['name']
    age = request.json['age']
    grade = request.json['grade']
    new_student = Student(name, age, grade)
    db.session.add(new_student)
    db.session.commit()
    return student_schema.jsonify(new_student), 201

@app.route('/students', methods=['GET'])
def get_students():
    students = Student.query.all()
    return students_schema.jsonify(students)

@app.route('/students/<int:id>', methods=['GET'])
def get_student(id):
    student = Student.query.get_or_404(id)
    return student_schema.jsonify(student)

@app.route('/students/<int:id>', methods=['PUT'])
def update_student(id):
    student = Student.query.get_or_404(id)
    student.name = request.json['name']
    student.age = request.json['age']
    student.grade = request.json['grade']
    db.session.commit()
    return student_schema.jsonify(student)

@app.route('/students/<int:id>', methods=['DELETE'])
def delete_student(id):
    student = Student.query.get_or_404(id)
    db.session.delete(student)
    db.session.commit()
    return '', 204
```

Step 5: Run the Application

To run the application, execute the following command:

```
python app.py
```

The Flask development server will start, and you can access the API at http://127.0.0.1:5000.

13. Use Python's requests module to fetch data from an open API (e.g., weather or currency exchange API) and display the results.

Example Code:

```
import requests

# Replace with your OpenWeatherMap API key
api_key = 'YOUR_API_KEY'
city = 'London'  # Change this to the city you want to get the weather for
url =
f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&uni
ts=metric'

# Send GET request to fetch data
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    data = response.json()  # Convert the response to JSON format

    # Extract relevant information from the JSON data
    city_name = data['name']
    weather_description = data['weather'][0]['description']
    temperature = data['main']['temp']
    humidity = data['main']['humidity']
    wind_speed = data['wind']['speed']

    # Print the weather data
    print(f"Weather in {city_name}:")
    print(f"Description: {weather_description}")
    print(f"Temperature: {temperature}°C")
    print(f"Humidity: {humidity}%")
    print(f"Wind Speed: {wind_speed} m/s")
else:
    print(f"Failed to retrieve data. HTTP Status code:
{response.status_code}")
```

Explanation:

1. **API URL**: We build the URL by replacing `city` with the desired city name, and `api_key` with your actual OpenWeatherMap API key.
2. **Making the Request**: We use the `requests.get()` method to send a GET request to the API.
3. **Response Handling**: The response is checked for a successful status code (200). If successful, we parse the JSON data to extract the weather details such as description, temperature, humidity, and wind speed.
4. **Displaying Data**: Finally, the relevant weather data is printed to the console.

Steps to Get Your API Key:

1. Go to OpenWeatherMap and sign up for a free account.
2. After signing in, navigate to the "API keys" section and create a new API key.
3. Replace `'YOUR_API_KEY'` in the code with the API key you get from OpenWeatherMap.