Classroom Exercise: Nginx and Jinja2 with Python

In this updated exercise, students will:

1. Set up **Nginx** as a web server with multiple `location` blocks for various routes.
2. Serve static HTML files using Nginx and dynamic HTML content with **Jinja2** in a Python HTTP server.
3. Use **Jinja2** features such as conditionals, loops, and filters.
4. Extend Flask routes to serve multiple templates with dynamic data.

Part 1: Installing and Configuring Nginx

Step 1: Install Nginx

1. Log in to your server via SSH.
2. Install Nginx using the package manager of your distribution:
   - For **Ubuntu** or **Debian**:

     ```
     bash
     Copy code
     sudo apt update
     sudo apt install nginx
     ```

   - For **CentOS** or **RHEL**:

     ```
     bash
     Copy code
     sudo yum install nginx
     ```

   - For **Arch Linux**:

     ```
     bash
     Copy code
     sudo pacman -S nginx
     ```

3. Start and enable Nginx:

   ```
   bash
   Copy code
   sudo systemctl start nginx
   sudo systemctl enable nginx
   ```

4. Verify Nginx is running by visiting your server's IP address in the browser. You should see the default Nginx welcome page.

Step 2: Configuring Nginx with Multiple `location` Blocks

1. Create a new directory to store your static HTML files:

   ```
   bash
   Copy code
   sudo mkdir -p /var/www/html/your_website
   ```

2. Create the `index.html` file to serve as a homepage:

   ```
   bash
   Copy code
   cd /var/www/html/your_website
   sudo nano index.html
   ```

   Add this content:

   ```
   html
   Copy code
   <html>
   <head><title>Welcome to My Website</title></head>
   <body>
       <h1>Welcome to My Website</h1>
       <p>This is the homepage.</p>
   </body>
   </html>
   ```

3. Open the Nginx configuration file to set up multiple routes:

   ```
   bash
   Copy code
   sudo nano /etc/nginx/sites-available/default
   ```

4. Modify the configuration to include multiple `location` blocks:

   ```
   nginx
   Copy code
   server {
       listen 80;
       server_name your_server_ip;

       root /var/www/html/your_website;
       index index.html;

       location / {
           try_files $uri $uri/ =404;
       }

       location /about {
           alias /var/www/html/your_website/about.html;
       }

       location /contact {
   ```

```
            alias /var/www/html/your_website/contact.html;
        }

        location /flaskapp {
            proxy_pass http://127.0.0.1:5000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
```

5. Create the additional HTML files:
    - `about.html`:

      ```
      html
      Copy code
      <html>
      <head><title>About Us</title></head>
      <body>
          <h1>About Us</h1>
          <p>Learn more about our website.</p>
      </body>
      </html>
      ```

    - `contact.html`:

      ```
      html
      Copy code
      <html>
      <head><title>Contact Us</title></head>
      <body>
          <h1>Contact Us</h1>
          <p>Contact us at contact@example.com.</p>
      </body>
      </html>
      ```

6. Restart Nginx to apply changes:

   ```
   bash
   Copy code
   sudo systemctl restart nginx
   ```

7. Now visit:
    - `http://your_server_ip/` for the homepage.
    - `http://your_server_ip/about` for the About page.
    - `http://your_server_ip/contact` for the Contact page.

Part 2: Setting Up a Python HTTP Server with Jinja2

Step 3: Install Flask and Jinja2

1. Install the necessary Python packages:

   ```
   bash
   Copy code
   pip install Flask Jinja2
   ```

Step 4: Create the Python Server with Multiple Routes

1. Create a Python script (`app.py`) for the Flask server.
2. Add multiple routes and templates to practice various **Jinja2** features.

```
python
Copy code
from flask import Flask, render_template
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Student, Course  # Assuming these models are already set up

app = Flask(__name__)

# Database setup
engine = create_engine('sqlite:///students.db')
Session = sessionmaker(bind=engine)
session = Session()

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

# Students list with courses (Using Jinja2 for-loops)
@app.route('/students')
def students():
    students = session.query(Student).all()
    return render_template('students.html', students=students)

# Single student profile with conditionals and filters
@app.route('/student/<int:student_id>')
def student_profile(student_id):
    student = session.query(Student).get(student_id)
    return render_template('student_profile.html', student=student)

# Dynamic content with Jinja2 (form input simulation)
@app.route('/courses')
def courses():
    courses = session.query(Course).all()
    return render_template('courses.html', courses=courses)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Part 3: Creating Jinja2 Templates

1. Template for Home Page (`home.html`)

```
html
Copy code
<!DOCTYPE html>
<html>
<head><title>Home</title></head>
<body>
    <h1>Welcome to the Student Management System</h1>
    <ul>
        <li><a href="/students">List of Students</a></li>
        <li><a href="/courses">List of Courses</a></li>
    </ul>
</body>
</html>
```

---

2. Template for Students List (`students.html`)

- Using **for-loops** to iterate over students.

```
html
Copy code
<!DOCTYPE html>
<html>
<head><title>Students List</title></head>
<body>
    <h1>List of Students</h1>
    <table border="1">
        <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Grade</th>
            <th>Courses</th>
        </tr>
        {% for student in students %}
        <tr>
            <td><a href="/student/{{ student.id }}">{{ student.name }}</a></td>
            <td>{{ student.age }}</td>
            <td>{{ student.grade }}</td>
            <td>
                <ul>
                    {% for course in student.courses %}
                    <li>{{ course.course_name }}</li>
                    {% endfor %}
                </ul>
            </td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

3. Template for Single Student Profile (`student_profile.html`)

- Using **conditionals** and **filters**.

```
html
Copy code
<!DOCTYPE html>
<html>
<head><title>{{ student.name }}'s Profile</title></head>
<body>
    <h1>{{ student.name }}'s Profile</h1>
    <p>Age: {{ student.age }}</p>
    <p>Grade: {{ student.grade }}</p>

    <h2>Courses:</h2>
    <ul>
        {% for course in student.courses %}
        <li>{{ course.course_name | title }}</li>  <!-- Using filter to
capitalize the course names -->
        {% endfor %}
    </ul>

    {% if student.grade == 'A' %}
    <p><strong>Excellent performance!</strong></p>
    {% elif student.grade == 'B' %}
    <p><strong>Good job, but there's room for improvement.</strong></p>
    {% else %}
    <p><strong>Needs to work harder!</strong></p>
    {% endif %}
</body>
</html>
```

4. Template for Courses List (`courses.html`)

- Using **for-loops** to display the courses.

```
html
Copy code
<!DOCTYPE html>
<html>
<head><title>Courses List</title></head>
<body>
    <h1>Available Courses</h1>
    <ul>
        {% for course in courses %}
        <li>{{ course.course_name }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Part 4: Configure Nginx to Proxy to Flask

Step 5: Configure Nginx to Forward Requests to Flask

1. Open the Nginx config file:

   ```
   bash
   Copy code
   sudo nano /etc/nginx/sites-available/default
   ```

2. Modify the `location /flaskapp` block to point to the Flask app:

   ```
   nginx
   Copy code
   location /flaskapp {
       proxy_pass http://127.0.0.1:5000;
       proxy_set_header Host $host;
       proxy_set_header X-Real-IP $remote_addr;
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
       proxy_set_header X-Forwarded-Proto $scheme;
   }
   ```

3. Restart Nginx:

   ```
   bash
   Copy code
   sudo systemctl restart nginx
   ```

4. Now, visiting `http://your_server_ip/flaskapp` should show the dynamically generated content from Flask.