



LAB 2 :

Web Application

HTML and Jinja Templating

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

Objective:

In this lab we will learn how to create web pages that give structure and layout to your web application using HTML codes and JINJA templating.

Requirement:

1. Complete Lab 1
2. Install jinja2 templating library in your python environment using the python **pip** tool.

Steps:

1. Create a directory in your project directory to place the HTML template files.
 - a. *hint: templates file must be placed in "/static/templates"*
2. Create the base template file and give it a name as "base.html".
 - a. *hint: base.html must be a valid standard html file with minimum html, head and body tags.*
3. Write HTML code to structure your base.html
 - a. Define header block
 - b. Define content block
 - c. Define footer block
4. Write HTML code for index.html
 - a. write HTML Jinja code to inherit main.html
 - i. *hint: use "extend" to inherit main.html into your index.html*
 - b. Write HTML Jinja code to display content for index.html
 - c. Write an html code to welcome visitors to your web page.
 - d. Edit flask @app.route for "/" to return the template as response.
 - i. *hint: return render_template("index.html")*
 - e. Re-launch flask app and use an internet browser to access to index.html, i.e. point the browser the the web application root end-point "/"
 - f. Try to change the content of your index.html, and notice the changes.
5. Create another html file in the template directory and name it "dashboard.html"
 - a. Copy index.html and rename it to "dashboard.html".
 - b. Use Jinja code to dynamically parse data to the web application to be rendered in your dashboard.html
 - i. *hint: use double curly brackets "{{ }}" as a placeholder to display data in your Jinja block content.*
 - ii. *hint: create a flask route to accept dynamic queries from the user and use the value received via client request to be re-assigned to a variable to be used with the "render_template" function.*

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

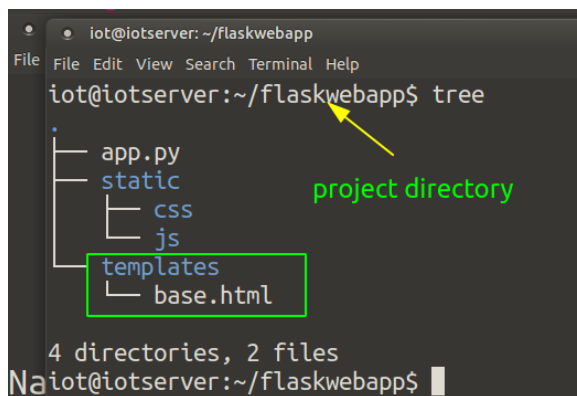
- iii. Change variable values in your flask code and pass it as a parameter for the “render_template” function along with “dashboard.html”.
- iv. Reload the web app and notice the changes.

Step 1:

Create a directory structure to place your flask project and it should follow below tree structure:

Name your project directory as you like:

example “/flaskwebapp” and create the “templates” directory as shown in directory tree structure below:



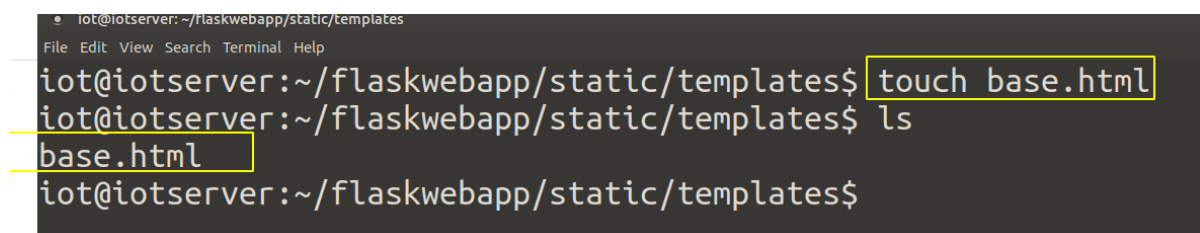
```
iot@iotserver: ~/flaskwebapp
File Edit View Search Terminal Help
iot@iotserver:~/flaskwebapp$ tree
.
├── app.py
├── static
│   ├── css
│   └── js
└── templates
    └── base.html

4 directories, 2 files
iot@iotserver:~/flaskwebapp$
```

A terminal window showing the directory structure of a Flask project. The command `tree` is executed, displaying a tree view of the files and directories. The `templates` directory is highlighted with a green box, and the `base.html` file inside it is also highlighted. A yellow arrow points to the `tree` command.

Step 2:

Create the `base.html` in the `templates` directory:



```
iot@iotserver: ~/flaskwebapp/static/templates
File Edit View Search Terminal Help
iot@iotserver:~/flaskwebapp/static/templates$ touch base.html
iot@iotserver:~/flaskwebapp/static/templates$ ls
base.html
iot@iotserver:~/flaskwebapp/static/templates$
```

A terminal window showing the creation of the `base.html` file. The command `touch base.html` is executed, and the output of the `ls` command is shown, confirming the file's existence.

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

Step 3:

Now edit “base.html” with your preferred text editor or an IDE software, and enter the following content:

“base.html” content:

```
<!DOCTYPE html>
<html>

<head>
    <title>Flask Template Example</title>
</head>

<body>
    {% block header %}
    <h1>Flask Template Example</h1>
    {% endblock %}

    {% block content %}
    {% endblock %}

</body>

</html>
```

Test you base template by editing you app.py flask application:

Simple flask application code:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def myapp():
    return render_template("base.html")

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

*Note that you must import **render_template** function from the flask module.

run your flask application:

```
iot@iotserver:~/flaskwebapp$ python3 app.py
* Serving Flask app "app" (lazy loading)
```

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

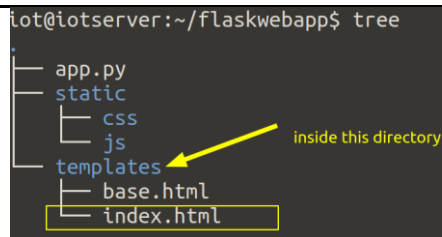
- * Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
- * Debug mode: on
- * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
- * Restarting with stat
- * Debugger is active!
- * Debugger PIN: 252-203-194

Step 4:

Create a new html file called “index.html”

```
iot@iotserver:~/flaskwebapp/static$ touch index.html
```

```
iot@iotserver:~/flaskwebapp$ tree
.
├── app.py
├── static
│   ├── css
│   ├── js
│   └── templates
│       ├── base.html
│       └── index.html
└──
```



Edit the file using your preferred text editor or IDE.

“index.html” content:

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

{% block header %}
<h1>Welcome to My Flask Main Page !</h1>
{% endblock %}

{% block content %}
<br>
<h1>Hello {{visitor}} !</h1>
{% endblock %}
```

Notice the directive `{% extends "base.html" %}`, this will cause the jinja template engine to inherit “base.html” code and combine it within your index.html code.

Notice the place holder “`{{visitor}}`”, this will tell jinja to render any value that the variable visitor contains into the page at the `<h1>` tag location.

edit your flask application code “app.py”:

```
from flask import Flask, render_template
```

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

```
app = Flask(__name__)

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

@app.route('/hello/<visitor>')
def hello(visitor):
    return render_template("index.html", visitor=visitor)

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

Notice that the *new endpoint* **“/hello”** is added to flask application code and the endpoint can accept additional input query from the user and parsed the value to variable **“visitor”** to be used by python and jinja template engine.

Save the file, flask will automatically reload the code.

Open your internet browser and point to the new endpoint **“/hello/<put your name here>”**



Notice the changes on the page....

The page is now showing your “index.html” which contains the greeting “Hello {{visitor}} !”, along with it is the original greeting “Welcome to My Main Page !” which is coded in the “main.html”.

This shows that your jinja template engine has combined the main.html and index.html into one page and response to the request.

Notice also, the new endpoint can accept another input in the form of query parameters by extending the endpoint /hello with parameter input “/yourname”.

The input is rendered after the word “Hello” in your index.html.

Step 5:

Create a Dashboard page:

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

In the templates directory copy your index.html then paste and rename it to “dashboard.html”

```
iot@iotserver:~/flaskwebapp/templates$ cp index.html dashboard.html
iot@iotserver:~/flaskwebapp/templates$ ls
base.html  dashboard.html  index.html
```

Then edit the dashboard.html with your preferred text editor or IDE.

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

{% block header %}
<h1>Welcome To IoT Restaurant !</h1>
{% endblock %}

{% block content %}
<br>
<div>
<h2>Restaurant Table Layout will be done here ! <h2>
</div>
{% endblock %}
```

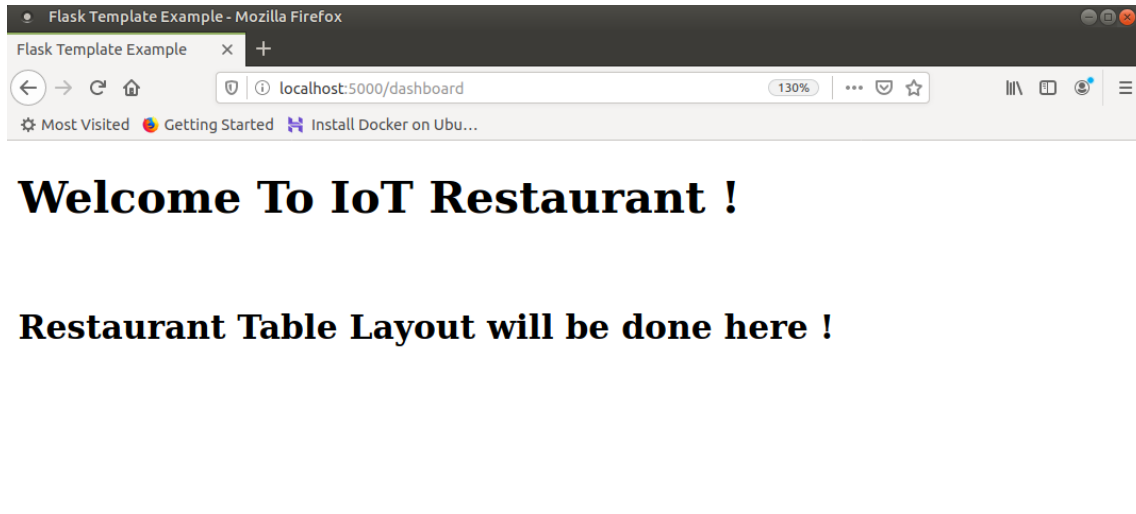
Create another endpoint to access the dashboard.html via the internet browser.

add the code snippet below to your flask application code.

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

LAB 2: HTML AND JINJA TEMPLATING FOR WEB APPLICATION

You should get a result similar to the picture below.



That's it... you have learnt from this lab the jinja templating engine...! To make your page look more modern and familiar to users, you need to use "CSS" codes so that the page will be laid out correctly and responsively to various screen sizes.