Flask Demo Notes

Written by Lauren Gee

These instructions are assuming people are using Windows.

If you are using Linux, the steps are pretty similar. Linux users don't need WSL, and your default terminal will be the Linux terminal, so things are even simpler if you are using Linux.

Things I have installed:

- Windows Subsystem for Linux
- Git for Windows
- vscode extensions:
    - Remote - WSL
    - Python

For this Flask Demo, I will be largely following this tutorial:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Setting things up

1. Create a new private repo on GitHub

- Give it a name
- Make sure it is private (unless you want it to be public)
- Add a readme file
- Add a gitignore if you want (choose Python on the dropdown menu)

2. Open vscode with no folder open

If there is a folder open, go to File->Close Folder

3. Go to Source Control then clone repo from GitHub

Click the "Clone Repository" button on the Source Control panel. This will open up a little text field for you to enter something into.

Go to GitHub on your web browser, go to the repo you made earlier. Click on the green "Code" button, which will make a little drop down panel appear. Copy the https address that appears, paste it into the text field in vscode. When you paste it, it should say "Clone from URL" under the text field.

Click on "Clone from URL" and it should open a window for navigating your folders. Navigate to a folder where you want to put that repo. It will create a folder inside that folder, and that is the folder for your repo.

vscode will download any files that are already in GitHub's version of the repo. My repo is new, so I just have a readme file and a .gitignore

Once the repo is successfully cloned, you can open it and look inside. Click the "Open" button on the popup that appears.

Optional Step: Activate the Remote - WSL extension

In order to use it, click on the green >< icon in the bottom left corner of your vscode window. I chose the "reopen folder in WSL" option from the list. It took a little time to open. Now, when you open the terminal, WSL is the default terminal.

After that, I wanted to get vscode to recognise the Python interpreter I have in WSL. I already had the Python extension for normal vscode. When I looked up Python on the extensions panel, there was a button for installing a WSL-specific version of the Python extension.

Using the Remote - WSL extension and Python extension together makes things really convenient. The WSL terminal becomes the default terminal. Also, vscode can automatically activate the virtual environment if you set it up right.

4. Open the WSL terminal inside VSCode

If you have the Remote - WSL extension running, when you open a terminal it will be a WSL terminal by default. If you don't have Remote - WSL running, the default terminal will be powershell.

If you have powershell open, you can click the little down-arrow next to the plus sign, and select something like "Ubuntu-20.04 (WSL)". (If you don't have WSL installed, you should probably install it now.)

Once you've opened the right kind of terminal, it should look something like this:
`something@something:/mnt/c/path/to/my/folder$`

5. Update your WSL

Since I haven't updated my software for a while, I will update using these commands in the WSL terminal:

```
sudo apt-get update
sudo apt-get upgrade
```

(You need to run both of these. You will probably be prompted for your WSL password. You will have set a WSL password when you installed WSL. If you forgot your WSL password, you'll have to reset it.)

6. Make sure you have Python 3 installed

Try to run python to see if python is installed.

Run this command:

`python`

Or:

`python3`

At least one of "**`python`**" and "**`python3`**" should work if you have it installed.
If you don't have it installed, try installing it with this:

`sudo apt-get install python3`

Once you have python running, close it by either entering this into the python prompt:

`exit()`

Or with the keyboard shortcut ctrl->D.

(If you accidentally do ctrl->Z by mistake, it puts the program in the background, but doesn't actually close it. To bring it back into the foreground enter this command into the terminal:

`fg`

Then you can close the program in the normal way.)

To test that it's working well, I'll make a simple python script and run it.

`test.py`

```
print("hello world")
```

(Note that **`test.py`** is the name/path of the file, and **`print("hello world")`** is the content of the file. I'll be writing code snippets in this format for the rest of the document.)

Let's run this script in the WSL terminal using this command:

`python3 myfile.py`

When I run it, this output appears in the terminal:

```
hello world
```

By the way, if you need to run the same command multiple times, you can use the up and down arrows on your keyboard to bring up previous commands you have run. This is much quicker than manually re-typing commands.

7. Install pip

Run this command to install pip:

`sudo apt-get install python3-pip`

8. Install venv

Run this command to install venv:

```
sudo apt-get install python3-venv
```

9. Create a virtual environment with venv

(I'm assuming you haven't changed folder in the terminal, and you are still in the main folder of your repo.)

Run this command in the terminal to make a new virtual environment:

```
python3 -m venv venv
```

The files for that virtual environment will be created in a subfolder called 'venv' within your folder. This 'venv' folder should now be visible in vscode. (click the refresh button if you can't see it)

10. Activate the virtual environment

You will need to activate the virtual environment using this command:

```
source venv/bin/activate
```

However, if you have the Remote - WSL and Python extensions set up correctly, and you set your default python interpreter to the virtual environment you just made, vscode will automatically activate the virtual environment for you.

How to do this:
When you have a python file open in vscode's editor (such as **`test.py`**), at the bottom right corner of the vscode window, next to Python there should be a number like 3.8.10, or something similar. If you click on that, you can change which interpreter vscode uses by default. I chose the one that says **`./venv/bin/python`**. If that doesn't come up, click the refresh button at the top right corner of the little popup.

(In some cases, vscode may automatically choose your virtual environment as the default python interpreter without you having to specify it.)

If you set up your Python extension correctly, the terminal will automatically activate your virtual environment whenever you open a terminal in vscode. Otherwise, you'll have to activate your virtual environment every time you open a new terminal.

To check if your virtual environment is activated, look at your terminal window. If it is activated, the terminal prompt will look something like this:

```
(venv) something@something:/mnt/c/path/to/my/folder$
```

If there is a (venv) in brackets at the start of the line, that means the virtual environment is activated.

11. Install flask in your virtual environment

Now that your virtual environment is activated, whenever you install things with pip, they will be installed only in that virtual environment. Virtual environments are good for when you have multiple python projects on one computer, and they each use different packages. If you have a separate virtual environment for each project, your python packages won't conflict between projects.

Make sure your virtual environment is activated. Your terminal prompt should now look something like this:
`(venv) something@something:/mnt/c/path/to/my/folder$`

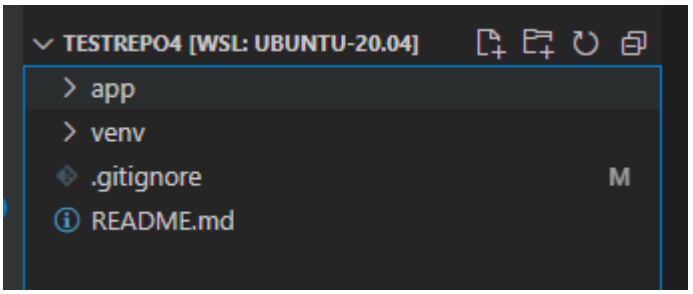
Now, lets install the flask using this command:

```
pip install flask
```

Making a simple Flask app

Feel free to delete **`test.py`** now, we don't need it anymore.

12. Make an 'app' folder in our working folder



13. Make '__init__.py' in the 'app' folder

```
app/__init__.py
```

```
from flask import Flask

app = Flask(__name__)

from app import routes
```

---

14. Make 'routes.py' in the 'app' folder

app/routes.py

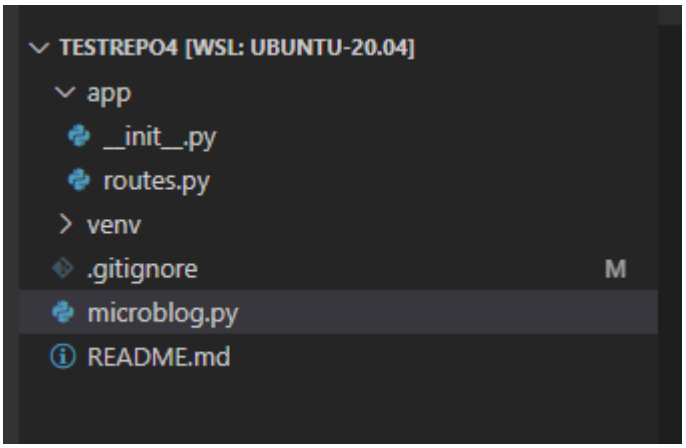
```
from app import app

@app.route('/')
@app.route('/index')
def index():
     return "Hello, World!"
```

15. Make 'microblog.py' in the main folder

(you can name this whatever you like, in the original tutorial it is called microblog)

microblog.py

```
from app import app
```



16. Set your FLASK_APP environment variable

Run this command in the WSL terminal:

`export FLASK_APP=microblog.py`

17. Run your flask app

You can run your flask app using this command in the WSL terminal:

`flask run`

Take a look at your website in a web browser.

You can open the website by doing ctrl->click on the address that appears in your terminal window when you run flask.

You can close the flask app by doing ctrl->C in the WSL terminal.

18. Install python-dotenv so that you can more conveniently set your environment variables

You can install python-dotenv using this command:

```
pip install python-dotenv
```

You'll need to make a .flaskenv file to hold your flask-related environment variables.

.flaskenv

```
FLASK_APP=microblog.py
FLASK_ENV=development
```

Setting **`FLASK_ENV=development`** in the environment variables is also a good idea. This enables some cool debugging features, and also the page will automatically reload whenever you make changes to the code. (Don't use development mode when you actually deploy your website. Development mode is just for when you're developing it.)

Quickest way to run Flask

Now that you have everything set up nicely, there aren't that many steps to run your flask app.

Assuming you've been following the steps up to this point, and you don't have vscode open, these are the steps to run your flask app:

1. Open vscode (check that you have the right folder open in vscode)
2. Open a WSL terminal
3. Run this command in the WSL terminal:

```
flask run
```

The various packages and extensions we have installed have made it so that we don't have to manually activate the virtual environment, and we don't have to manually set our environment variables.

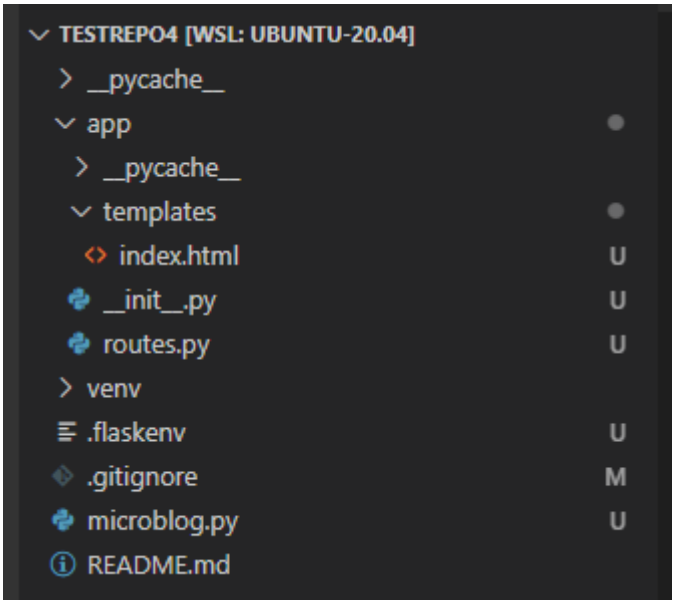
Also, remember you can close your flask app by doing ctrl->C in the terminal.

Adding proper HTML

Let's serve an actual HTML page instead of just "Hello, World!"

19. Make a 'templates' folder inside our 'app' folder

20. Make 'index.html' inside the templates folder



21. Type ! then press Enter to make generate a quick HTML file

Feel free to modify this HTML file, or make your own from scratch

22. Add some content to the HTML page

Put some content in the body of your website, so that when you run the website you'll be able to see if it's working.

Here's the basic HTML page I created:

app/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      <title>Microblog</title>
</head>
<body>
      <h1>Hello!</h1>
</body>
</html>
```

23. Edit 'app/routes.py', use render_template to load our new HTML page

app/routes.py

```
from flask import render_template
from app import app

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

24. Check the website in your browser to see if it is working correctly

Do this after every step! It's a good idea to test your code frequently, as this makes debugging easier.

25. Add some dynamic content to your HTML

app/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
     <title>{{ title }} - Microblog</title>
</head>
<body>
     <h1>Hi, {{ user.username }}!</h1>
</body>
</html>
```

app/routes.py

```
from flask import render_template
from app import app

@app.route('/')
@app.route('/index')
def index():
     user = {'username': 'Miguel'}
     return render_template("index.html", title="Home", user=user)
```

26. Add the if/else and for loop stuff that was in the tutorial

app/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      {% if title %}
      <title>{{ title }} - Microblog</title>
      {% else %}
      <title>Microblog</title>
      {% endif %}
</head>
<body>
      <h1>Hi, {{ user.username }}!</h1>
      {% for post in posts %}
      <div><p>{{ post.author.username }} says: <b>{{ post.body
}}</b></p></div>
      {% endfor %}
</body>
</html>
```

app/routes.py

```
from flask import render_template
from app import app

@app.route('/')
@app.route('/index')
def index():
      user = {'username': 'Miguel'}
      posts = [
      {
            'author': {'username': 'John'},
            'body': 'Beautiful day in Portland!'
      },
      {
            'author': {'username': 'Susan'},
            'body': 'The Avengers movie was so cool!'
      }
      ]
      return render_template("index.html", title="Home", user=user,
posts=posts)
```

27. Add the template inheritance stuff that was in the tutorial

Make a new file 'base.html' in app/templates, and modify index.html

---

app/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      {% if title %}
      <title>{{ title }} - Microblog</title>
      {% else %}
      <title>Microblog</title>
      {% endif %}
</head>
<body>
      <div>Microblog: <a href="/index">Home</a></div>
      <hr>
      {% block content %}{% endblock %}
</body>
</html>
```

---

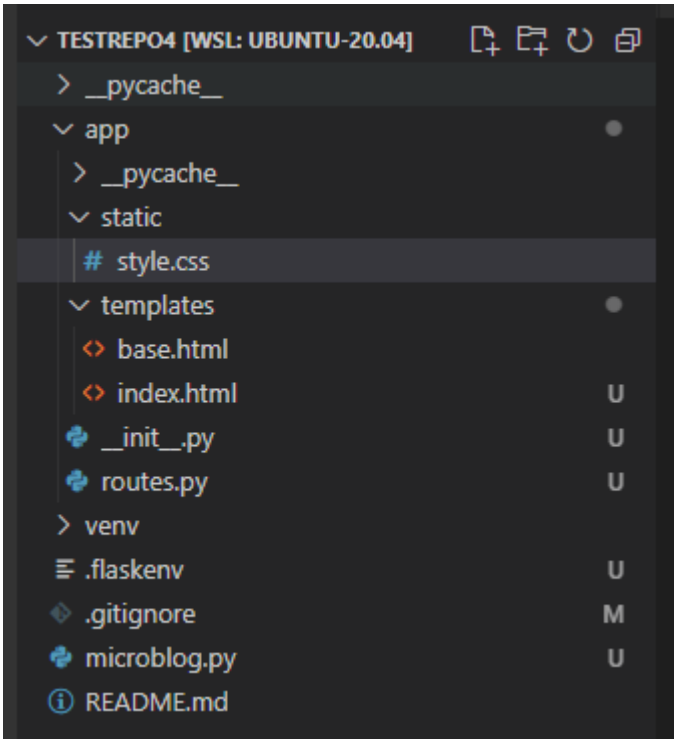
app/templates/index.html

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

{% block content %}
      <h1>Hi, {{ user.username }}!</h1>
      {% for post in posts %}
      <div><p>{{ post.author.username }} says: <b>{{ post.body
}}</b></p></div>
      {% endfor %}
{% endblock %}
```

Adding CSS

Now let's add some static CSS to our website. The tutorial does have a section on how to add bootstrap, we'll just use a static CSS file to keep things simple.

28. Make a 'static' folder in our 'app' folder



29. Make 'style.css' in our 'static' folder

```
    app/static/style.css
```

```
div.header {
     background-color: palegreen;
     padding: 5px;
}
```

---

30. Add the stylesheet to your HTML

app/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      {% if title %}
      <title>{{ title }} - Microblog</title>
      {% else %}
      <title>Microblog</title>
      {% endif %}
      <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body>
      <div class="header">Microblog: <a
href="/index">Home</a></div>
      <hr>
      {% block content %}{% endblock %}
</body>
</html>
```

31. You can play around with the CSS and HTML to make it look good

You'll probably just need to refresh the page on your browser to see new changes.

Committing your Changes to GitHub

32. Set up your user.name and user.email on your local git installation

First, figure out if you want to keep your email address private or not. If you ever want to commit code to a public repo, it is a good idea to keep your email address private. With GitHub open in your web browser, go to your settings, and look at your email settings. I have "Keep my email addresses private" and "Block command line pushes that expose my email" both turned on. That page should also provide a special dummy email address for you, it will look something like this:
`12345678+username@users.noreply.github.com`

You can use that email address as your **`user.email`** in git, if you want to keep your email address private. If you're not worried about keeping your email address private, then you can instead use the email address you use to log into your GitHub account.

If you are using the Remote - WSL extension, go to the WSL terminal, and use these commands:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

(you can put whatever you want for **`user.name`**, I think)
(Remove the **`--global`** part if you have multiple accounts you use with git, and you only want to set up your user.name and user.email for one repo at a time)

If you are not using the Remote - WSL extension, vscode will be using Git for Windows instead.
Option 1: Open git bash inside of vscode, and use the same commands I said earlier:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

Option 2: In your file explorer, navigate to your repo folder, then right click and select "Git GUI here" (if you click "Git Bash here" that's another way of opening the git bash)
In Git GUI go to Edit->Options and you can set your user.name and user.email there.

33. In vscode, go to the source control panel

Click the refresh button to make sure all your new files / changed files appear in the list

34. Stage your changes

Hover over "Changes" and a plus sign + will appear. Click that plus sign to add all changes to our next commit.

There should now be quite a few items under "Staged Changes" and no items under "Changes".

35. Make a commit

Click the tick ✓ symbol at the top of the source control panel to make a commit.

A text box will appear, type a message to describe what change you made / what code you are trying to commit. It's a good idea to always write meaningful messages, so that future you or your collaborators will be able to figure out what you did. Don't just mash random keys on your keyboard.

36. Do a "git pull"

Click on the three dots icon ... then click "pull" to download any changes your collaborators might have made.

(If your collaborators made changes, you'll have to merge them with your changes. However, in this little demonstration, there shouldn't be any changes to merge. In the project you will have to collaborate with other people on a shared git repo, there are resources available to learn how to do that.)

37. Do a "git push"

Click on the three dots icon ... then click "push" to upload your changes to GitHub

38. Check GitHub

Go on GitHub, go to your repo, your new changes should be there now.

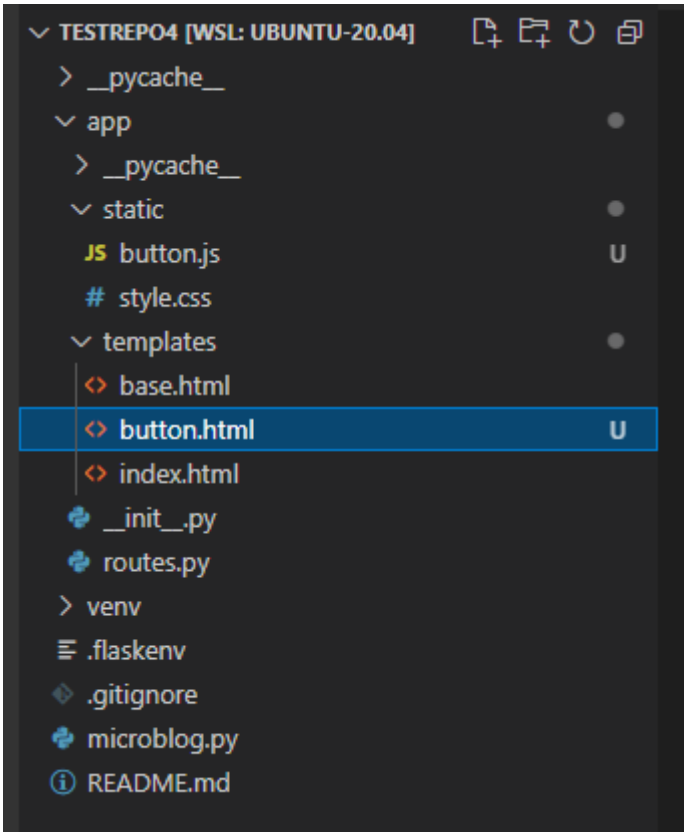
You can also view your commit history to see the history of changes there are in the repo.

Adding a Second Page to our Website

Now that we have commited our changes, let's go back to the explorer panel and add something new to our website. Then we'll commit that too.

It's a good idea to break your work up into smaller steps, and have a commit for each step. It's good to have small commits focused on a single subject, with each commit having a one-line description. It's easier to understand. Also, if you're saving your work to GitHub several times per day, if you get a hardware failure you won't lose a whole day's worth of work (or several days).

Let's add a new page to our website. A page with a button on it. We will use a js script to make the button do something too.



39. Make 'button.html' in our templates folder

```
    app/templates/button.html
```

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

{% block content %}
     <h1>Hello</h1>

     <div>
     <button id="myButton" type="button"
onclick="toggleText()">Button!</button>
     <p id="textBox"></p>
     </div>
     <script src="{{ url_for('static', filename='button.js')
}}"></script>
{% endblock %}
```

40. Make 'button.js' in our 'static' folder.

```
    app/static/button.js
```

```
var toggle = false;
function toggleText() {
    toggle = !toggle;
    let text;
    if (toggle) {
        text = "You pressed the button!";
    } else {
        text = "";
    }
    document.getElementById("textBox").innerHTML = text;
}

document.addEventListener("keypress", function(event) {
    if (event.key === "Enter") {
        event.preventDefault();
        document.getElementById("myButton").click();
    }
});
```

To make our website more accessible to people with vision impairments, I made it so that the button can also be activated by pressing the Enter key.

41. Add the Button page to our routes

app/routes.py

```
from flask import render_template
from app import app

@app.route('/')
@app.route('/index')
def index():
     user = {'username': 'Miguel'}
     posts = [
     {
          'author': {'username': 'John'},
          'body': 'Beautiful day in Portland!'
     },
     {
          'author': {'username': 'Susan'},
          'body': 'The Avengers movie was so cool!'
     }
     ]
     return render_template("index.html", title="Home", user=user,
posts=posts)

@app.route('/button')
def button():
     return render_template("button.html", title="Button!")
```

42. Now we need to update our header in 'base.html' to include a link to our Button page.

app/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      {% if title %}
      <title>{{ title }} - Microblog</title>
      {% else %}
      <title>Microblog</title>
      {% endif %}
      <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body>
      <div class="header">
      Microblog:
      <a href="/index">Home</a>
      <a href="/button">Button!</a>
      </div>
      <hr>
      {% block content %}{% endblock %}
</body>
</html>
```

43. Check that everything is working correctly

44. Commit your changes to GitHub (see steps 33 to 38)