Introduction to Web Applications Development with Python Flask

Adapted from: https://scotch.io/tutorials/getting-started-with-flask-a-python-microframework#toc-file-structure

The Flask MicroFramework

- Flask is a microframework for Python based on
  - Werkzeug (a WSGI - Web Server Gateway Interface toolkit)
  - Jinja 2 (a template Engine)
- The “micro” in microframework means Flask aims to keep the core simple but extensible.
  - Flask won’t make many decisions for you, such as what database to use.
  - Those decisions that it does make, such as what templating engine to use, are easy to change.

Growing with Flask

- Once you have Flask up and running, you’ll find a variety of extensions available in the community to integrate your project for production.

- The Flask core team reviews extensions and ensures approved extensions do not break with future releases.

Flask Features

- It's easy to set up
- It's supported by an active community
- It's well documented
- It's very simple and minimalistic, and doesn't include anything you won't use
- At the same time, it's flexible enough that you can add extensions if you need more functionality

Installation

You may already have Python installed on your system. You can check by running the python command in your terminal. If it's installed, you should see the following output:

```
$ python3
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
```

Installation

Install pip

```
$ sudo yum install epel-release
$ sudo yum install python3-pip
$ pip3 –version
```

Installation

Install Flask

```
$ sudo pip3 install Flask
$ pip3 freeze | grep Flask
```

Create a Project

Next, let's create a directory for our app. This is where all our files will go:

```
$ mkdir my-project
$ cd my-project
```

Hello World Flask

Create the following file, hello_world.py, in your favourite text editor:

```
# hello_world.py

from flask import Flask
app = Flask(__name__)


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

We begin by importing the Flask class, and creating an instance of it.

Hello World Flask

Create the following file, hello_world.py, in your favourite text editor:

```
# hello_world.py

from flask import Flask
app = Flask(__name__)


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

We use the __name__ argument to indicate the app's module or package, so that Flask knows where to find other files such as templates.

Hello World Flask

Create the following file, hello_world.py, in your favourite text editor:

```
# hello_world.py

from flask import Flask
app = Flask(__name__)


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

Then we have a simple function that will display the string Hello World! The preceeding decorator simply tells Flask which path to display the result of the function. In this case, we have specified the route /, which is the home URL.

Run the Hello World Application

```
$ export FLASK_APP=hello_world.py
$python3 -m flask run --host=0.0.0.0
 * Serving Flask app "run.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

The first command tells the system which app to run.

The next one starts the server. The parameter –host=0.0.0.0 is to allow access from outside the server

Enter the specified URL (http://0.0.0.0:5000/) in your browser.

Flask Projects Directory Structure

These are a few of the common directories in a Flask project:

/app: This is a directory within my-project . We'll put all our code in here

/app/templates: This is where our HTML files will go.

/app/static: This is where static files such as CSS (Cascading Style Sheets) and JavaScript files as well as images usually go.

Flask Projects Directory Structure

Create directories:

```
mkdir app app/templates
```

Your project directory should now look like this:

```
├── my-project
    ├── app
    |    ├── templates
    ├── hello_world.py
```

File Structure

Most Flask apps have the following basic file structure:

run.py: This is the application's entry point. We'll run this file to start the Flask server and launch our application.

config.py: This file contains the configuration variables for your app, such as database details.

app/__init__.py: This file intializes a Python module. Without it, Python will not recognize the app directory as a module.

File Structure

app/views.py: This file contains all the routes for our application. This will tell Flask what to display on which path.

app/models.py: This is where the models are defined. A model is a representation of a database table in code.

File Structure

Go ahead and create these files, and delete hello_world.py

```
$ touch run.py config.py
$ cd app
$ touch __init__.py views.py
$ rm hello_world.py
```

File Structure

New directory Structure

```
├── my-project
    ├── app
    |    ├── __init__.py
    |    ├── templates
    |    └── views.py
   ├── config.py
    └── run.py
```

Configuration

The config.py file should contain one variable per line, like so:

```
# config.py

# Enable Flask's debugging features. Should be False in
production
DEBUG = True
```

Initialization

Next, we have to initialize our app with all our configurations. This is done in the app/__init__.py file. Note that if we set instance_relative_config to True, we can use app.config.from_object('config') to load the config.py file.

```
# app/__init__.py

from flask import Flask

# Initialize the app
app = Flask(__name__, instance_relative_config=True)

# Load the views
from app import views

# Load the config file
app.config.from_object('config')
```

Runfile

All we have to do now is configure our run.py file so we can start the Flask server.

```
# run.py

from app import app

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

Test

To use the command flask run like we did before, we would need to set the FLASK_APP environment variable to run.py, like so:

```
$ export FLASK_APP=run.py
$ python3 -m flask run --host=0.0.0.0
```

We'll get a 404 page because we haven't written any views for our app. We'll be fixing that shortly.

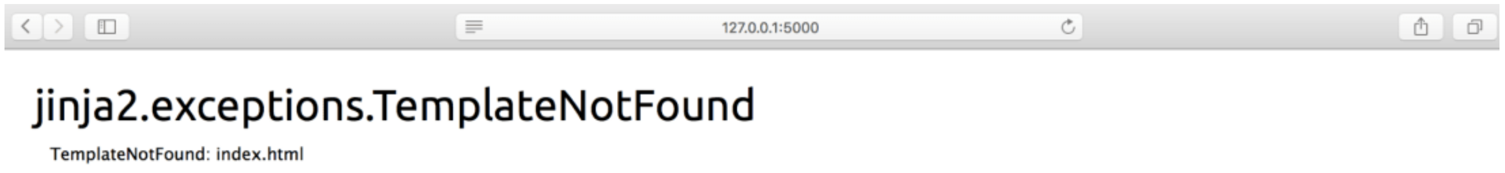
Views

```
# views.py

from flask import render_template

from app import app

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


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

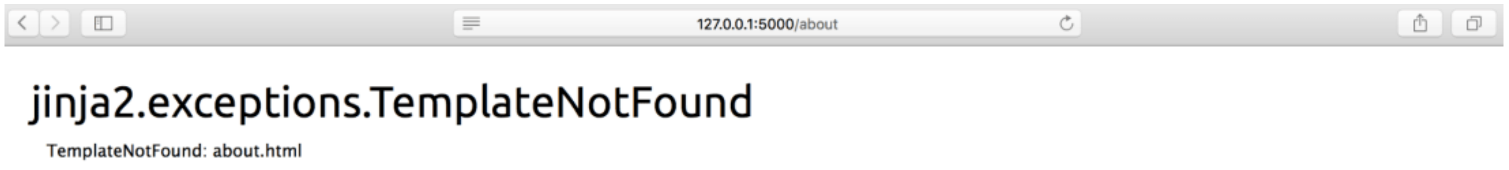
Flask provides a method, render_template, which we can use to specifiy which HTML file should be loaded in a particular view.

Views

The index.html and about.html files don't exist yet, so Flask will give us a Template Not Found error when we navigate to these paths. Go ahead; run the app and see:


127.0.0.1:5000

```
jinja2.exceptions.TemplateNotFound

TemplateNotFound: index.html
```


127.0.0.1:5000/about

```
jinja2.exceptions.TemplateNotFound

TemplateNotFound: about.html
```

Templates

- Flask allows us to use a variety of template languages, but Jinja2 is by far the most popular one.
- Jinja provides syntax that allows us to add some functionality to our HTML files, like if-else blocks and for loops, and also use variables inside our templates.
- Jinja also lets us implement template inheritance, which means we can have a base template that other templates inherit from.

Templates

Let's begin by creating the following three HTML files:

```
$ cd app/templates
$ touch base.html index.html about.html
```

Templates – base.html

We'll start with the base.html file, using a slightly modified version of this example Bootstrap template:

```
<!-- base.html -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>{% block title %}{% endblock %}</title>
    <!-- Bootstrap core CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
    <!-- Custom styles for this template -->
    <link href="https://getbootstrap.com/examples/jumbotron-narrow/jumbotron-narrow.css" rel="stylesheet">
  </head>
  <body>
    <div class="container">
      <div class="header clearfix">
        <nav>
          <ul class="nav nav-pills pull-right">
            <li role="presentation"><a href="/">Home</a></li>
            <li role="presentation"><a href="/about">About</a></li>
            <li role="presentation"><a href="http://flask.pocoo.org" target="_blank">More About Flask</a></li>
          </ul>
        </nav>
      </div>
      {% block body %}
      {% endblock %}
      <footer class="footer">
        <p>© 2016 Your Name Here</p>
      </footer>
    </div> <!-- /container -->
  </body>
</html>
```

Templates – index.html

We'll use {% block %} and {% endblock %} tags in the templates that inherit from the base template:

```
<!-- index.html-->

{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block body %}
<div class="jumbotron">
  <h1>Flask Is Awesome</h1>
  <p class="lead">And I'm glad to be learning so much about it!</p>
</div>
{% endblock %}
```

Templates – about.html

We'll use {% block %} and {% endblock %} tags in the templates that inherit from the base template:

```
<!-- about.html-->

{% extends "base.html" %}
{% block title %}About{% endblock %}
{% block body %}
<div class="jumbotron">
  <h1>The About Page</h1>
  <p class="lead">You can learn more about my website here.</p>
</div>
{% endblock %}
```

Templates – about.html

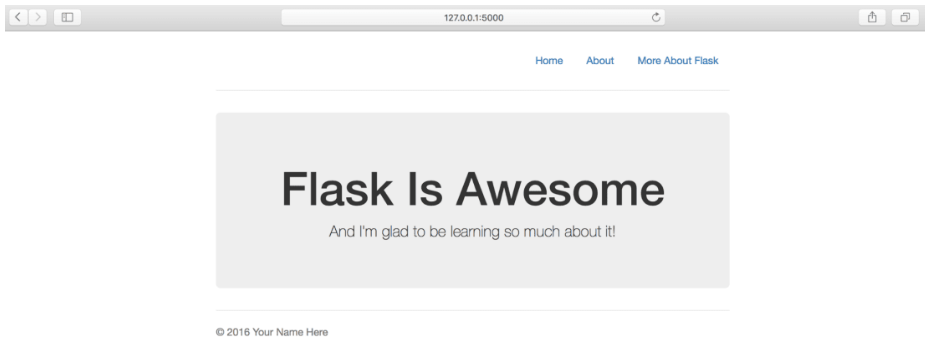
We'll use {% block %} and {% endblock %} tags in the templates that inherit fro

```
<!-- about.html-->

{% extends "base.h
{% block title %}Ab
{% block body %}
<div class="jumbo
  <h1>The About P
  <p class="lead">You can learn m        out my website here.</p>
</div>
{% endblock %}
```

We use {% extends %} the tag to inherit from the base template. We insert the dynamic content inside the {% block %} tags. Everything else is loaded right from the base template, so we don't have to re-write things that are common to all pages, such as the navigation bar and the footer.

Result

Let's refresh our browser and see what we have now:



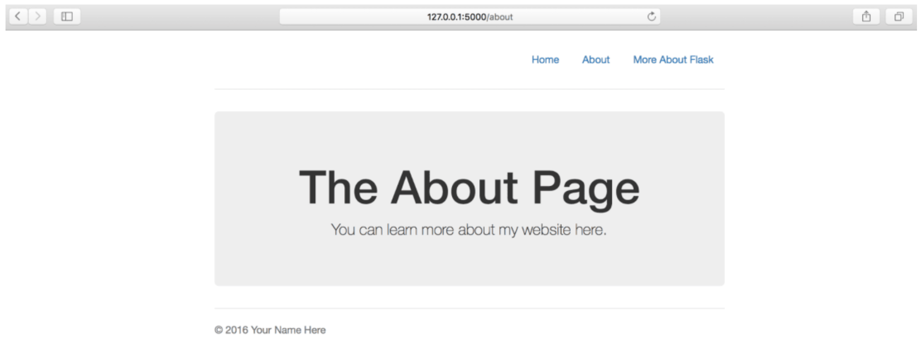
127.0.0.1:5000

Home About More About Flask

Flask Is Awesome

And I'm glad to be learning so much about it!

© 2016 Your Name Here



127.0.0.1:5000/about

Home About More About Flask

The About Page

You can learn more about my website here.

© 2016 Your Name Here

Deploying site in Apache + Centos

Install wsgi modulo

```
# yum install python3-mod_wsgi
```

Verificar instalación

```
apachectl -M | grep wsgi
```

Verificar que el selinux este deshabilitado, en caso contrario deshabilítelo en /etc/selinux/config

```
$ sestatus
SELinux status:                 disabled
```

Deploying site in Apache + Centos

Creating a .wsgi file:

Copy my-project diectory to /var/www

In /var/www/my-project create a file named application.wsgi

```
#!/usr/bin/python
import sys
sys.path.insert(0,"/var/www/my-project/")
from app import app as application
```

Deploying site in Apache + Centos

Configuring Apache:

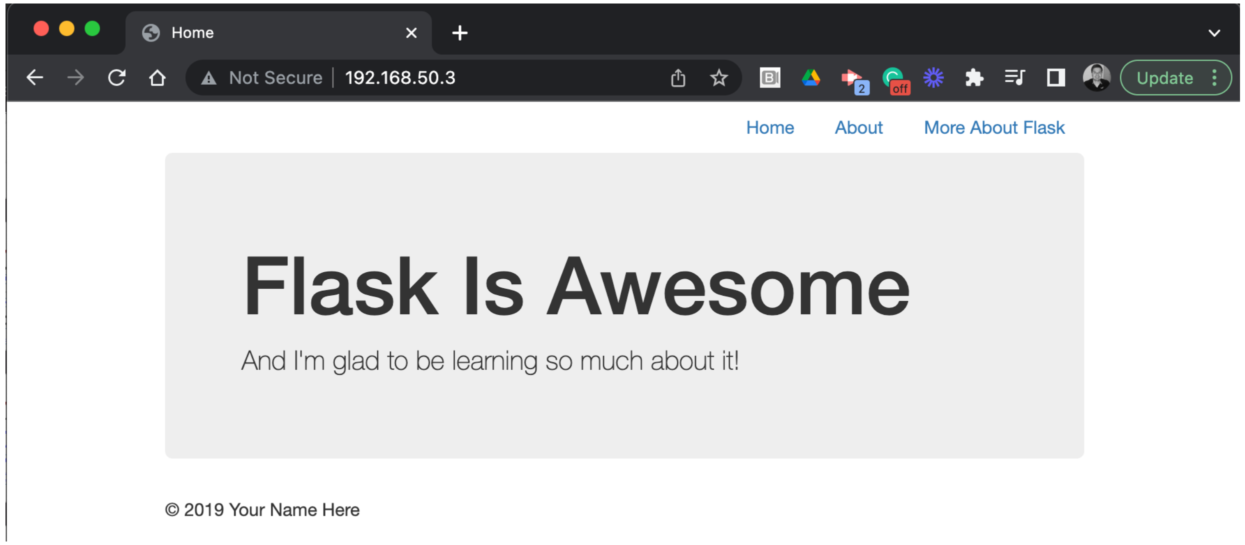
In /etc/httpd/conf/httpd.conf configure a virtual host:

```
WSGIScriptAlias / /var/www/my-project/application.wsgi

<VirtualHost *>
   ServerName www.servicios.com
   <Directory /var/www/my-project/>
      Order deny,allow
      Allow from all
   </Directory>
</VirtualHost>
```

Result

Restart httpd and test:

```
# service httpd restart
```


Home

Not Secure | 192.168.50.3

Home About More About Flask

Flask Is Awesome

And I'm glad to be learning so much about it!

© 2019 Your Name Here

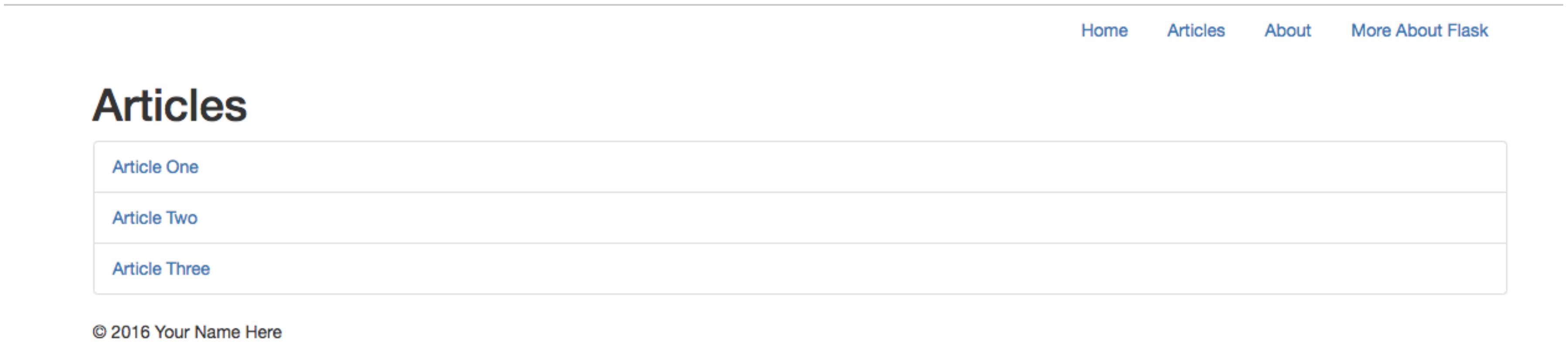
En caso de error, verifique los logs de apache:

```
#cat /var/log/httpd/error_log
```

Practice

1. Deploy the site in Apache

2. Add a view that shows a list of Articles



Articles

Article One

Article Two

Article Three

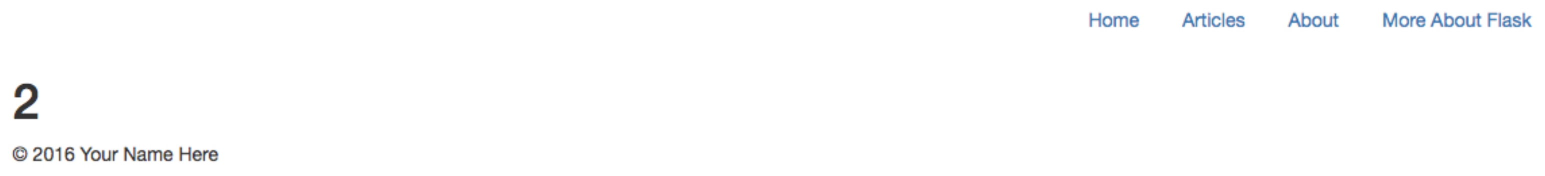
© 2016 Your Name Here

The list of articles must be loaded from the data.py file located in classroom.

Practice

3. Create a view that shows the article id once it is selected from the article lists.

For example, if the user click on the article 2 hyperlink, it must show:



2

© 2016 Your Name Here

References

- Flask: http://flask.pocoo.org/
- Getting Started With Flask: https://scotch.io/tutorials/getting-started-with-flask-a-python-microframework#toc-file-structure
- Apache Configuration: http://flask.pocoo.org/docs/1.0/deploying/mod_wsgi/
- FLASK HELLO WORLD APP WITH APACHE WSGI: https://www.bogotobogo.com/python/Flask/Python_Flask_HelloWorld_App_with_Apache_WSGI_Ubuntu14.php
- Use mod_wsgi to Run Python as a Web Application on CentOS 7: https://www.1and1.com/cloud-community/learn/application/python/use-mod-wsgi-to-run-python-as-a-web-application-on-centos-7/