







Next



Unit 3 / Lesson 4 / Assignment 1

### Structuring a Flask project

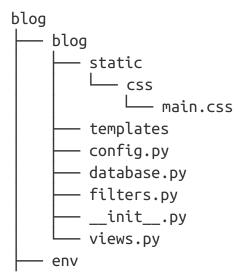
(S) Estimated Time: 1 hour

In the next few assignments you are going to learn how to structure and build a larger project using Flask and SQLAlchemy by building a simple blogging engine.

In this first assignment you'll set up the basic structure of the app and perform various configuration tasks. You will then use this structure to build up your code in the following assignments.

#### First steps

First, you'll set up your directory structure. Create a new folder called *blog* and create the following structure:



Root directory
Module directory
Static files directory
Stylesheets directory
Main stylesheet
Templates directory
App configuration options
Database setup and SQLAlchemy models
Custom Jinja filters
App setup and configuration
Views
Virtual environment

manage.py
requirements.txt

App management file Install requirements

Next, follow a basic setup procedure:

- 1. Move into the root directory cd blog
- 2. Set up the virtualenv python3 -m venv env
- 3. Activate the virtualenv source env/bin/activate
- 5. Install your dependencies pip install -r requirements.txt
- 6. Create a database createdb blogful

Now let's get to the point where you can fire up the development server. Add the following code to your \_\_init\_\_.py file.

```
from flask import Flask
app = Flask(__name__)
from . import views
from . import filters
```

Here you import the Flask object and create your app in the usual way. After you've created the app you import the (currently non-existent) views and Jinja filters. Notice how these imports come after you create the app rather than at the top of the file as usual. This is because the *views.py* and *filters.py* files will both make use of the app object. This means that they need importing after the object has been created. Note the relative import notation; this is a short-hand for import blog.views and import

blog.filters, and will import additional modules from the current package only.

Next you need to call the app.run method to start your development server. You are going to do this in the *manage.py* file, which will contain a series of commands intended to help you out as you develop the application. You could specify a command-line interface for this file using the argparse module, but in order to speed this work up a bit you are going to introduce a new module called Flask-Script.

Flask-Script is designed to allow you to easily specify tasks to help you manage your application. So let's see how you define a task. In the *manage.py* file add the following code:

```
import os
from flask_script import Manager

from blog import app

manager = Manager(app)

@manager.command
def run():
    port = int(os.environ.get('PORT', 8080))
    app.run(host='0.0.0.0', port=port)

if __name__ == "__main__":
    manager.run()
```

First you import the Manager object, and create an instance of it. Then you add a command to the manager by decorating a function using the manager.command decorator. The name of the function corresponds with the name of the argument which you give the manage script. For example to call the run function you'll say python manage.py run.

Inside the run function you try to retrieve a port number from the environment, falling back to port 8080 if it is unavailable. You then run the development server, telling it to listen on that port. A number of hosts use the PORT environment variable to tell the app which port it should be listening on, so it is generally good practice to comply with this.

Finally in the main block you run the manager. Try using this script to launch the development server by saying python manage.py run. You should see the server launched as usual listening on port 8080.

#### **Configuring your app**

Now that you have your app running take a few more steps so that you can manage the configuration of the app. In the *config.py* file add the following class:

```
import os
class DevelopmentConfig(object):
    SQLALCHEMY_DATABASE_URI = "postgresql://ubuntu:thinkful@localhost:
    DEBUG = True
```

You use this class to contain the configuration variables which control the Flask app. You tell Flask to use its debug mode to help you track down any errors in your application, and set the location of your database. Note that you might find that PostgreSQL isn't running; if you see errors referring to port 5432 and "connection refused", execute sudo service postgresql start . Section 2.1.3 has further information about this, if you aren't sure.

Next, load the configuration in the \_\_init\_\_.py file:

```
import os
from flask import Flask
```

```
app = Flask(__name__)
config_path = os.environ.get("CONFIG_PATH", "blog.config.DevelopmentColapp.config.from_object(config_path)

from . import views
from . import filters
```

Again, you're trying to get an environment variable which will set the path to your configuration object. If the variable is not set then you default to your development configuration. This provides a way to switch between configurations easily in different situations; for example in the next lesson you'll use this to switch over to a different configuration for testing the application. You then use the app.config.from\_object method to configure the app using the object specified.

#### Setting up the database

Now that the app is configured let's look at setting up a connection to the database where you'll be storing blog entries. Add the following setup code to *database.py*.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

from . import app

engine = create_engine(app.config["SQLALCHEMY_DATABASE_URI"])
Base = declarative_base()
Session = sessionmaker(bind=engine)
session = Session()
```

This should look very familiar; it's the basic boilerplate code for working with a database using SQLAlchemy. You create the engine which will talk to the database at the database URI which you specified in the config. You then create a declarative base, and start a new session.

That's pretty much everything which you need for the project configured. In the next assignment you'll use the structure you've built here to start to construct the blog engine.

#### How do you add a command to a Flask-Script Manager object?

You write a function which is decorated with the @manager.command decorator.

# What does the Flask.config.from\_object method do? When would we want to change the argument which we pass into the method?

It takes a string containing the path to a file, dictionary or class, and uses the variables in the corresponding object to populate the config object. You can change the argument to use different configurations for different environments, for example you may have a configuration for development, one for testing and one for production.

## Why is it useful for web apps to listen on a port specified by an environment variable?

It allows us to deploy on servers which don't have a fixed outgoing port or deploy on multiple servers each using a different outgoing port without having to change our code.

☆ ☆ ☆ ☆ . Report a typo or other issue
 ✓ Mark as completed

**Previous** 

Next