**CS 021 –Computer Programming I: Python**

**Use Cases – Web Development**

Overview

The following procedure outlines the process of creating dynamic web applications with Python. We will be using a package called Django – a widely used, open source web framework – to develop our web application and highlight the power and breadth of scope that Python encompasses as a language. We will start with the basics, like file structure, strengths and weaknesses and testing after which we will dive into the intricacies of the framework as well as some more advanced web dev. techniques specific to Django.

*Note: it is normally best practice to install a virtual environment via ‘virtualenvwrapper’ when developing professional web applications with Django, however this detail has been emitted for brevity*

**Procedure**

*Note: this tutorial is for Windows, if you are on a Mac, simply add the ‘sudo’ markup before each command*

1. To start, we need to install Django via ‘pip’, Python’s package manager. Open up a command prompt window and navigate to your desktop. Once on the desktop, type the command below into command prompt, press enter and wait for the installation to complete.
   1. Commands
      1. ‘**pip install Django**’
2. Once this is finished, let’s make sure that the installation was successful. Run the following command in command prompt and hit enter. Compare the output in your command prompt to the output below, if a version number greater than 2.0.0 was not returned, return to step one.
   1. Commands
      1. ‘**python -m django --version**’



1. Now that we know Django has been installed correctly, let’s create our first project. Run the following command to do just that.
   1. Command
      1. ‘**django-admin startproject <project-name>**
   2. We will use project name ‘**mysite**’ for this example

Note: You’ll need to avoid naming projects after built-in Python or Django components. In particular, this means you should avoid using names like django (which will conflict with Django itself) or test (which conflicts with a built-in Python package).

1. What did we just do?
   1. Initialized a Django project with the following file structure:

mysite/

manage.py

mysite/

\_\_init\_\_.py

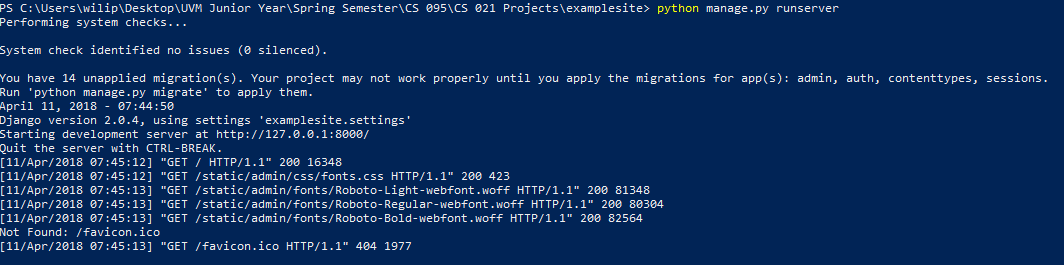
settings.py

urls.py

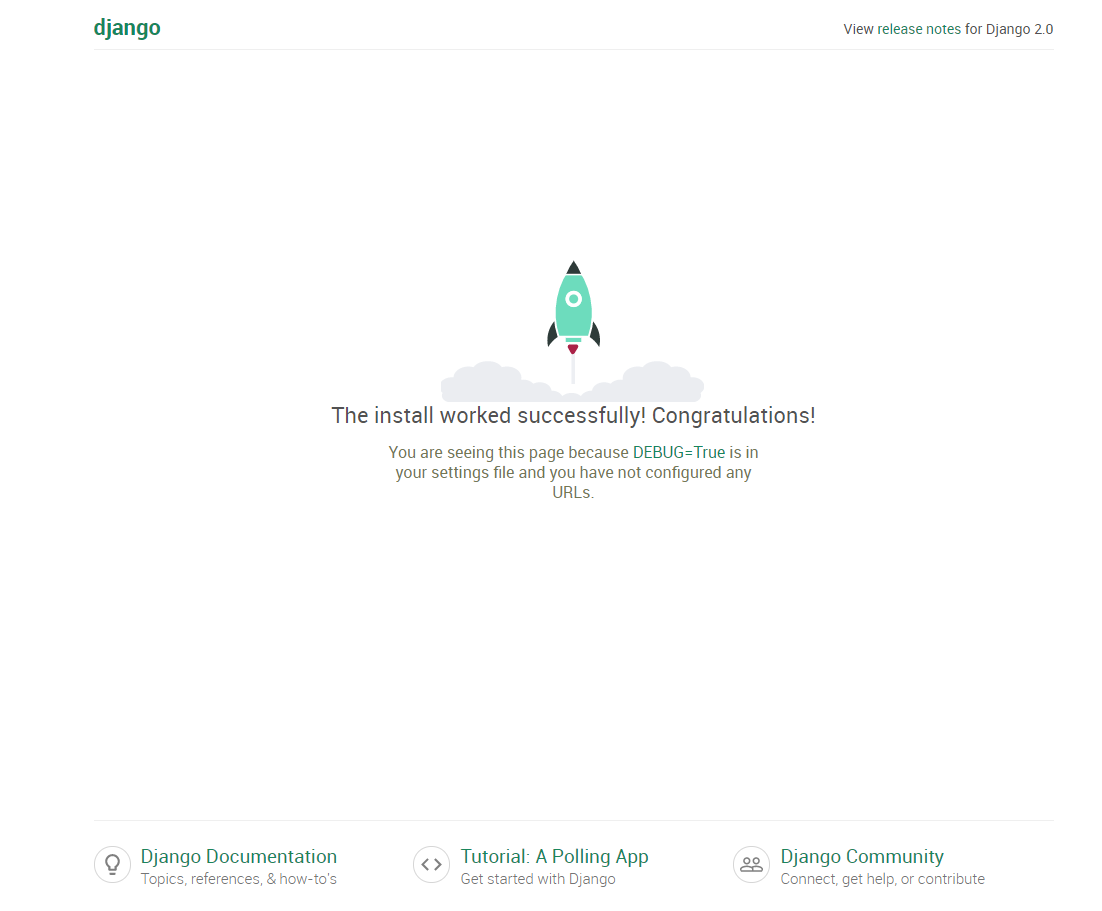
wsgi.py

* 1. What do these files do?
* The outer mysite/ root directory is just a container for your project. Its name doesn’t matter to Django; you can rename it to anything you like.
* manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin and manage.py.
* The inner mysite/ directory is the actual Python package for your project. Its name is the Python package name you’ll need to use to import anything inside it (e.g. mysite.urls).
* mysite/\_\_init\_\_.py: An empty file that tells Python that this directory should be considered a Python package. If you’re a Python beginner, read more about packages in the official Python docs.
* mysite/settings.py: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
* mysite/urls.py: The URL declarations for this Django project; a “table of contents” of your Django-powered site. You can read more about URLs in URL dispatcher.
* mysite/wsgi.py: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

1. To make sure the project was created correctly, run the command below, open a web browser and navigate to the URL that is output in the console, in this case, it is ‘**http://127.0.0.1:8000**’ but yours could vary.
   1. Command
      1. ‘**python manage.py runserver**’



* 1. You should see this page in your browser if everything worked correctly.



1. Now that we know our project was created correctly, we need to create an application inside of the project directory. Run the following command to initialize a new project.
   1. Command
      1. ‘**python manage.py startapp polls**’
   2. File structure

myapp/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

1. Once this is finished, we can start writing our app! To start, we need to open the **‘views.py’** file in a IDE (Integrated Development Environment) or text editor. For this example, we will be using IDLE, the editor that comes bundled with Python. Go ahead and do that now.
2. Now, let’s start by creating and defining our initial entry point for our application. Add the following code to your **‘views.py’** file.

**from** **django.http** **import** HttpResponse

**def** index(request):

return render(request,'index.html')

1. After this, we need to create a file called **‘urls.py’** in the /myapp directory. This will tell our application that when someone initially navigates to our site, render the index view. Create a **‘urls.py’** file in the /myapp directory and add the following code to it.

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('', views.index, name='index'),

]

1. When this is complete, we need to edit the ‘urls.py’ file in the **/examplesite (**myapp’s parent) directory to reflect the code below. The include() function allows referencing other URLconfs. Whenever Django encounters include(), it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLconf for further processing.

Source: <https://docs.djangoproject.com/en/2.0/intro/tutorial01/>

Note: This ‘urls.py’ file is different from that in step 9. Ensure you are editing the correct file inside of **/examplesite** and not **/myapp**.

**from** **django.contrib** **import** admin

**from** **django.urls** **import** include, path

urlpatterns = [

path('myapp/', views.index, include(‘myapp.urls’),

path('admin/', admin.site.urls),

]

1. Then, create a directory called **templates** inside of **/myapp**, cd into **/templates**, then create **index.html** file. This will be rendered by the **index** function that we created in the step eight. Add the following code to your **index.html** file.

<!DOCTYPE html>

<html lang="en">

<head>

{% block title %}<title>Example Site</title>{% endblock %}

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>

<body>

<nav class="navbar navbar-default">

<div class="container-fluid">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">My Company</a>

</div>

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<form class="navbar-form navbar-left">

<div class="form-group">

<input type="text" class="form-control" placeholder="Search">

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>

<ul class="nav navbar-nav navbar-right">

<li><a href="#">Link</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Dropdown <span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a href="#">Action</a></li>

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

</ul>

</li>

</ul>

</div>

</div>

</nav>

<div class="jumbotron">

<h1>Hello, world!</h1>

<p>...</p>

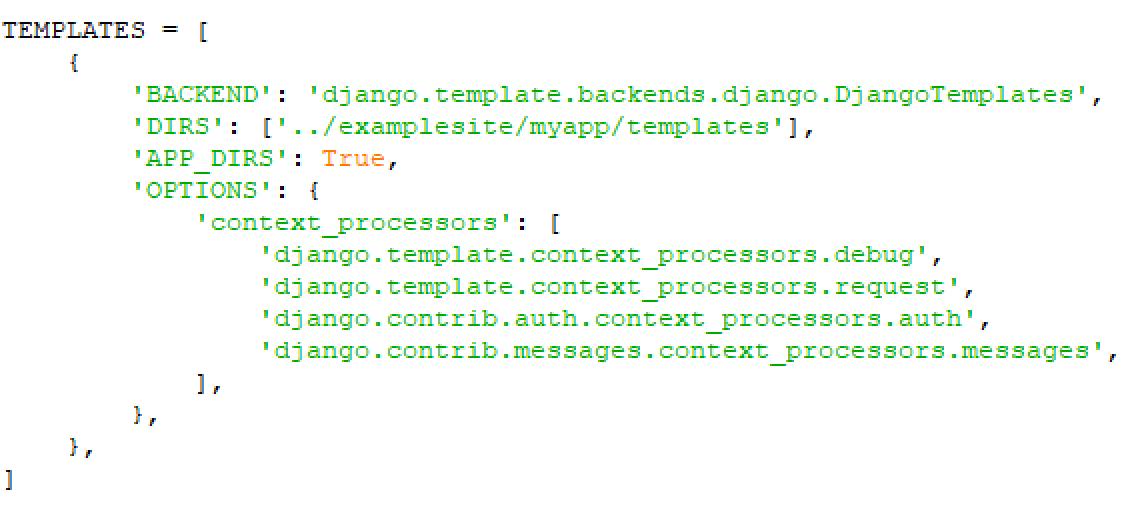
<p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a></p>

</div>

</body>

</html>

1. Before we can run it, we have to configure our Django instance a little bit. We have to tell Django where to look for out frontend files (HTML files) when we call **render()** in **views.py.** In **/examplesite/examplesite** open **settings.py** and add ‘**../examplesite/myapp/templates/’** to the DIRS list. That’s it, we’re good to go now!



1. Now, let’s restart our development server to make sure everything is working correctly. Run **‘python manage.py runserver’** inside of the **/examplesite** directory. Once it has started up, navigate to the output URL, in this case ‘**http://127.0.0.1:8000’** but this time add the **‘/myapp/’** endpoint, forming ‘**http://127.0.0.1:8000/myapp/’**. That’s it! Now you know the basics of web development with Python and Django.

