Learning Objectives: Setting Up Django

Learners will be able to…

- **Create a Django project**
- **Define the Model, View, Template paradigm**
- **Create and add a Django app to a Django project**
- **Create a URL pattern that returns a specific view**

info

Make Sure You Know

You are comfortable entering simple commands in the terminal and using Conda for package management. You are also familiar with Python, including classes.

Limitations

This Django project is a static website and does not make use of models and the database.

Django Setup

What is Django?

Django is a full-featured web framework that simplifies the process of making web applications in Python. Note that Django is described as a framework and not a module or package. A framework is a collection of tools to aid you in web development. For example, you would create a model of data in Python. Django then converts the model to be used in a database. As a developer you do not need to worry about the conversion process. Django does this automatically.

Django uses the **Model-View-Template (MVT)** paradigm. The **Model** manages data, the **view** describes the subset of data sent to the user, and the **template** displays the data for the user. Underlying all of this is a URL pattern matching system. If a user goes to `www.yourpage.com/about`, Django will take the `/about` URL pattern and match it to a view, which is presented to the user.

This might seem confusing at first. Django is a complex piece of software. Over time, however, you will see how all of these pieces interact with one another. But first, we need to install Django.

Installing Django

Start by creating the `django` virtual environment.

```
conda create --name django -y
```

Then activate the newly created environment.

```
conda activate django
```

Install Django as well as Black. Black is a Python code formatter. The project describes itself as being uncompromising, which means it is going to make changes to the code that might seem unusual. For instance, Black defaults to double quotes over single quotes. You can read more about the Black code style here. Black is very popular in the Django community, so we will be using this tool to format our code. Both packages are installed from the Conda Forge channel to get their latest versions.

```
conda install -c conda-forge django black -y
```

Starting Our Django Project

Now that we have a virtual environment and Django installed, it is time to create our project. **Important** the terminal to the left is currently inside the `beginner_project` directory that we cloned from GitHub. Run the following command to create our Django project. We are going to name it `beginner_project`.

```
django-admin startproject beginner_project .
```

Use the `tree` command to print a graphical representation of our current directory.

```
tree
```

What was once an empty directory is now contains another directory `beginner_project`, which itself contains five files. We also have a `manage.py` file. This is the benefit of using a framework. It knows that you need these directories and files for a proper Django project, so it creates them for you.

```
.
├── beginner_project
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Each file in the `beginner_project` directory serves a specific purpose:

- `asgi.py` - this is an optional file used with an asynchronous server gateway interface.
- `__init__.py`- this indicates the directory is part of a Python package.
- `settings.py` - this file controls the settings for our Django project.
- `urls.py` - this file tells Django which page to load when a user visits our site.
- `wsgi.py` - the web server gateway interface is used when we host our site on a production server.

The `manage.py` file is not technically part of the Django project since it is outside the `beginner_project` directory. However, this file is very important. We will use it when we need to make migrations to the database, run our development server, etc.

If we activated the `django` virtual environment at the beginning of a page, we want to get into the habit of deactivating it at the end of a page.

```
conda deactivate
```

Codio-Specific Settings

Running Django On Codio

info

Django and Codio

The changes on this page are done to allow Django to run on the Codio platform. By default, Django has strict security features that keep Codio from embedding Django.

The code samples below **only** apply for Codio. You **would not** make these changes when running Django outside of Codio.

Before setting up Django to run on Codio, you first need to import the `os` module.

```
import os
```

The two biggest obstacles in getting Django to run inside Codio are recognizing the unique host name for each Codio project and cookies. The changes below will tell Django the exact host name of the Codio project and alter how Django handles cookies. Make sure your settings match the ones below.

```
ALLOWED_HOSTS = ['*']
X_FRAME_OPTIONS = 'ALLOW-FROM ' +
        os.environ.get('CODIO_HOSTNAME') + '-8000.codio.io'
CSRF_COOKIE_SAMESITE = None
CSRF_TRUSTED_ORIGINS = ['https://' +
        os.environ.get('CODIO_HOSTNAME') + '-8000.codio.io']
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SAMESITE = 'None'
SESSION_COOKIE_SAMESITE = 'None'
```

Scroll down a bit and look for the definition of the variable `MIDDLEWARE`. Comment out the two lines that refer to `csrf`. CSRF stands for cross-site request forgery, which is a way for a malicious actor to force a user to perform an unintended action. Django normally has CSRF protections in place, but we need to disable them. There are serious security implications to doing so, however your project on Codio is not publicly available on the internet. Comment out the two lines in the `MIDDLEWARE` setting as shown below.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
#     'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
#     'django.middleware.clickjacking.XFrameOptionsMiddleware',

]
```

Starting the Server

Once you finish making the changes to the `settings.py` file, we want to enforce the Black style guide. Start by opening the terminal with the link below. Then activate the `django` virtual environment.

```
conda activate django
```

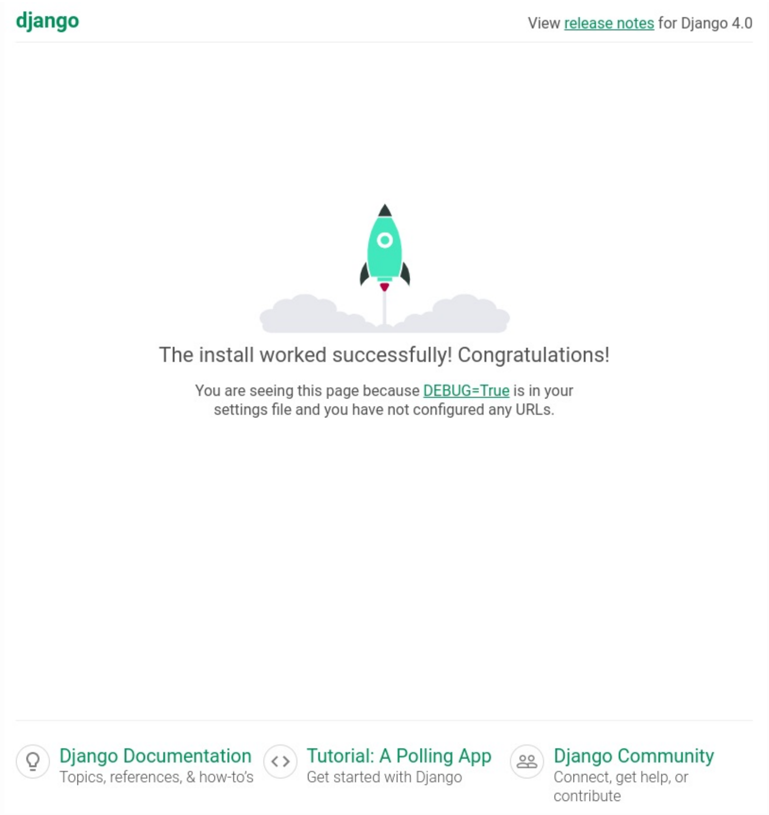
Now run Black on the `settings.py` file. If you were to click on the tab for the settings file, you would see the newly formatted settings.

```
black beginner_project/settings.py
```

Use the `manage.py` file to run the development server. We are going to specify `0.0.0.0` for the address of localhost and port 8000.

```
python manage.py runserver 0.0.0.0:8000
```

Finally, open the preview of Django running on the development server with the link below. You should see the default success message from Django. We see this message because all we did was start a Django project and change a few settings. There is no website to show, so we get the success message instead.


django

View release notes for Django 4.0

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Image depicts the success message from Django that the install is working.

Reminder: these changes only apply to working with Django on Codio. **Do not** make these changes to a project you plan on making available on the internet.

Django Apps

Creating a Django App

We have the skeleton of a Django project. A project is comprised of several Django apps. Each app should perform a specific function of the overall project. Combine all the apps together, and you have a Django project.

Our first Django project is going to be a simple webpage that focuses on templates and views. The idea is to clarify the relationship between URL patterns and information displayed in a template. The models and associated database will come in a later project.

Start by activating the `django` virtual environment.

```
conda activate django
```

To add an app to a Django project, use the `manage.py` file with the `startapp` argument. In addition, give this new app a name.

info

Project Content

This first project is going to be about programming languages. This is not a terribly exciting topic for all of you. I would encourage you to substitute the content of this project with something that appeals to you. Perhaps you have a hobby, a favorite musician or video game, etc. Picking a topic that interests you will make this project more engaging and meaningful.

```
python manage.py startapp languages
```

If you enter the `tree` command in the terminal, you will see the new structure of our Django project. In addition to the `beginner_project` directory, we now have the `languages` directory. This new directory is our app.

```
.
├── beginner_project
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   ├── settings.cpython-310.pyc
│   │   ├── urls.cpython-310.pyc
│   │   └── wsgi.cpython-310.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── db.sqlite3
├── languages
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── manage.py
```

Django automatically creates any files needed by the app. Each file has a specific purpose:

- `admin.py` - handles configuration of the built-in Django Admin feature
- `apps.py` - handles configuration for the app itself
- `migrations` - this directory keeps track of changes to models so the database stays in sync
- `models.py` - handles the definition of database models
- `tests.py` - handles any tests specific to the app
- `views.py` - handles any HTTP requests and responses for the app

Adding the App

Django makes it really easy to add an app to a project. The problem, however, is that the project “doesn’t know” about the newly created app. We need to add our `languages` app to the list of installed apps for the Django project. Open `settings.py` (the project settings) with the link below. Then add our languages app to the end of the list.

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "languages.apps.LanguagesConfig",
]
```

`LanguagesConfig` is a class in the `apps.py`file inside the `languages` directory (our new Django app). Django created this file and its contents during the app creation process. We do not need to make any changes to this file. We just need to make our Django project aware of it.

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black beginner_project/settings.py
```

Deactivate the virtual environment.

```
conda deactivate
```

Hello World

Dynamic and Static Webpages

Activate the `django` virtual environment.

```
conda activate django
```

With regards to Django, there are two kinds of webpages — dynamic and static. A **dynamic** website has content stored in a database. Users can alter the data in a database so the content displayed on the website can vary. A **static** website does not rely on a database. All of the information is hardcoded in the file and does not change.

Django uses the MVT paradigm, so a dynamic website uses all of the four components: models, views, templates and URL pattern matching. A static site only uses views, templates, and URL pattern matching. This project is a static website. We will discuss how to create dynamic webpages in another module.

Let’s start by creating the first view for our website. Open up the `views.py` file and change the code to look like the sample below. When a user requests the homepage view, Django will send an HTTP response with the string `'Hello, World!'`.

```
from django.http import HttpResponse

def homePageView(request):
  return HttpResponse('Hello, World!')
```

▼ **Did you notice?**

Django is not sending a template (an HTML document) as the response. It is just sending a string. This is not common practice in web development. We will upgrade our project to use templates in a bit.

There are two types of views in Django — function-based views and class-based views. Our `homePageView` is a function-based view. These views are easy to write and understand. We will convert this project to class-based views later on.

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/views.py
```

URL Patterns

We may have a view for our homepage, but we need a way to connect a URL to its corresponding view. This is done through URL pattern matching. URL patterns are determined in two places — at the project level and at the app level. When Django created our `languages` app, it did not create a `urls.py` file. Open the terminal. Then create `urls.py` inside the `languages` directory.

```
touch languages/urls.py
```

Once you created the new file, open it using the link below. The create the URL pattern for our `homePageView`.

```
from django.urls import path
from .views import homePageView

urlpatterns = [
  path('', homePageView, name='home'),
]
```

URL patterns are composed of three parts. The first is a regular expression, the second is a reference to the page view, and the third component is an optional name for the URL pattern. In our case, we have an empty string for the regular expression. That means whenever a user goes to our main website, they will be directed to our `homePageView`.

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/urls.py
```

We also need to update the URL patterns for the Django project. Open the `urls.py` file located in the `beginner_project` directory. Update the contents of the file to include our `languages` app.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
  path('admin/', admin.site.urls),
  path('', include('languages.urls')),
]
```

When a user requests our website, Django, at the project level, will look up the URL pattern. This, in turn, goes to the `urls.py` file inside the `languages` app and finally returns the `homePageView`.

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black beginner_project/urls.py
```

Run the development server and then open our website. You should see the text `Hello, World!`.

```
python manage.py runserver 0.0.0.0:8000
```

In order to deactivate the virtual environment, we need to first stop the development server. Open the terminal, then press `Ctrl + C` on the keyboard. This should stop the dev server. Then enter the command to deactivate the `django` environment.

```
conda deactivate
```