



Django Setup & Deployment Guidebook 2024

Setting up a basic Django Project and Deploying to Heroku

This document is for Django v4 projects and contains the terminal commands for the dependencies required for your Bootcamp Capstone project.

As you follow the steps in this guidebook, you will not learn anything you haven't already been taught from the Code Institute LMS or from your assigned SME.

Think of this as a quick refresher course, before you begin your Project.

Pre-Setup Notes:

It is recommended when you are still learning this content that you **type out each line of code**, rather than copying and pasting. This will help you learn!

Have your Heroku MFA device at hand before you begin. Signing in to GitHub, Gitpod, and Cloudinary beforehand will also speed up your progress.

Key:

proj_name = The name of your project. (This is the name of the directory where your settings.py file is located)

app_name = Any app within your Django project (Blog, about, comments etc.)

Part 1: Install Django and run the server to test.

Here we will install the Django Framework and generate a requirements.txt file that lists all installed Python packages and their versions (this file is crucial for recreating the environment elsewhere). We will then start a new Django project in the current directory, ensuring that we give our project a suitable name. We will add the hostname to ALLOWED_HOSTS in settings.py which is a security measure to prevent HTTP [Host header attacks](#).

In the Terminal:

	Step	Code
	Install Django:	pip3 install Django~=4.2.1
	Create requirements file	pip3 freeze --local > requirements.txt
	Create Project (proj_name)	django-admin startproject proj_name . (Don't forget the . at the end!)
	Apply pre-built Django account migrations	python3 manage.py migrate
	Run Server to Test	python3 manage.py runserver
	<p>You will see a yellow error screen, don't worry! Your server is running properly. This error is telling you that, for security reasons, Django doesn't recognise the hostname - the server name your project is running on.</p> <p>Select and copy the hostname after "Invalid HTTP_HOST header". In this example, that is '8000-nielmc-django-project-0kylrta3cs.us2.codeanyapp.com' - you can include the quotes.</p>	<div><p>DisallowedHost at /</p><p>Invalid HTTP_HOST header: '8000-nielmc-django-project-0kylrta3cs.us2.codeanyapp.com'. You may need to add '8000-nielmc-django-project-0kylrta3cs.us2.codeanyapp.com' to ALLOWED_HOSTS.</p></div> <p>Paste the hostname between the square brackets of ALLOWED_HOSTS. For the above example, this would look like ALLOWED_HOSTS = ['8000-nielmc-django-project-0kylrta3cs.us2.codeanyapp.com']</p>
	Immediately below the ALLOWED_HOSTS variable, add the following line of code to allow your IDE and Heroku to pass CSRF verification	CSRF_TRUSTED_ORIGINS = ['https://*.codeinstitute-ide.net', 'https://*.herokuapp.com']

Part 2: Creating an app in the Django Project

Here we will create a new app within the Django project. Think of apps as components of a Django project that handle specific functionalities. We will also register our app in `INSTALLED_APPS` enabling Django to include it in the project. Make sure we give the app an appropriate (lowercase) name like `blog` or `todo`.

	Step	Code
	Create App (app_name)	<code>python3 manage.py startapp app_name</code>
	Add to 'INSTALLED_APPS' in settings.py	<pre>INSTALLED_APPS = [... 'app_name',]</pre>
	Save file	

Part 3: Setting Up Heroku

In the Heroku dashboard we will set up a new app and add some configuration variables. We will also install gunicorn, a Python WSGI HTTP server, which Heroku uses to serve the Django application. (Think of this as a middleman between your Django app and the web) We then add a Procfile that defines the process to run the application on Heroku, specifying that Gunicorn should serve the Django project. We will then add Heroku to our allowed hosts as we did with our local server. Linking our GitHub profile directly to Heroku helps us deploy more easily.

The DISABLE_COLLECTSTATIC config var that we add below is useful during the initial setup before static files are configured properly. You will need to remove it later when you have static files in your project.

In Heroku:

	Step	Code
	Navigate to your Heroku dashboard	Heroku Dashboard
	Create new Heroku app	<ul style="list-style-type: none">- Choose a unique app name- Select a region close to you
	Add Config Var in app settings	<ul style="list-style-type: none">- Navigate to Settings tab and scroll down to Config Vars- Click “Reveal Config Vars”- Add new key DISABLE_COLLECTSTATIC value 1

In the Terminal/IDE:

	Step	Code
	Install web server gunicorn and freeze requirements	<pre>pip3 install gunicorn~=20.1 pip3 freeze --local > requirements.txt</pre>
	Create a Procfile	Create new file “Procfile” in the root directory Note: This file has no file extension and the P must be capitalised!
	Declare the process in Procfile	Inside the Procfile, add the following line of code: web: gunicorn proj_name .wsgi
	Add deployed app to ALLOWED_HOSTS	In settings.py add “.herokuapp.com” to the ALLOWED_HOSTS list

In Heroku:

	Connect to repository	<ul style="list-style-type: none">- In Heroku app, navigate to Deploy tab- Search for your Github repo and select it
	Check for Add-ons and Dynos	Inside the app's Resources tabs, ensure you're using Eco Dynos and delete any Postgres DB Add-ons

Part 4: Creating a Database

To create your Postgres Database for your project:

Use The [CI Database Maker](#)

This will provide you with a database url that you can utilise both in your project and in Heroku.

In these steps, we will utilise Code Institute's database creation tool to set up a PostgreSQL database and send the connection details to your email.

With the CI DB Maker:

	Step	Code
	Open the CI DB Maker	https://dbs.ci-dbs.net/
	Follow the 3 Step instructions	Input your email address and Click "Submit"
	Read Important Notes!	Be aware that there is a limit to the amount of concurrent databases you can create with this tool!
	Get Database URL	You will have received an email with your new PostgreSQL database URL. Copy the URL.

Part 5: Connecting to your Database

Here we will install the necessary packages required to connect Django to a PostgreSQL database. **dj-database-url** allows you to easily configure your database settings in Django by parsing a database URL. **psycopg** is a database adapter for Python, enabling your Django application to interact with a PostgreSQL database. We create an `env.py` to store our environment variables securely, preventing them from being exposed in the source code. By adding `env.py` to our `.gitignore` we ensure the `env.py` file is not tracked by version control for security reasons. Importing the `os` module is necessary to manage our `DATABASE_URL` and `SECRET_KEY` environment variables in Django. With these established, we can then add them to our environment variables on Heroku. We then make the necessary changes to our `settings.py` file to point Django towards the correct Database and Secret Key.

In the Terminal/IDE:

	Step	Code
	Install Database Packages	<code>pip3 install dj-database-url~=0.5 psycopg</code> Freeze your requirements using the freeze command shown in Part 3!
	Create <code>env.py</code> file	In the root directory, create a new file " <code>env.py</code> "

In `env.py`

	Step	Code
	Add <code>env.py</code> to <code>.gitignore</code>	Open the <code>.gitignore</code> file and add " <code>env.py</code> " on a new line (without quotation marks) (This is already added if you've used the CI template)
	Import <code>os</code> library	At the top of the <code>env.py</code> file add this line of code: <code>import os</code>
	Set environment variables	In <code>env.py</code> , add the following: <code>os.environ["DATABASE_URL"] = "Paste in PostgreSQL database URL"</code>
	Add in secret key	In <code>env.py</code> add the following: <code>os.environ["SECRET_KEY"] = "Make up your own randomSecretKey"</code>

In Heroku:

	Step	Code	
	Add Secret Key to Config Vars	SECRET_KEY, "randomSecretKey"	
	Add a Config Var called DATABASE_URL	DATABASE_URL, "yourDBUrlgoeshere"	Note: The value should be the PostgreSQL database url you copied in the previous step

In settings.py

	Step	Code
	Update env.py with the following code (Note: code in bold is new)	from pathlib import Path import os import dj_database_url if os.path.isfile("env.py"): import env
	Remove the insecure secret key and replace - <i>links to the SECRET_KEY variable on Heroku</i> (Note: code in bold is new)	SECRET_KEY = os.environ.get('SECRET_KEY')
	Comment out the old DataBases Section	# DATABASES = { # 'default': { # 'ENGINE': 'django.db.backends.sqlite3', # 'NAME': BASE_DIR / 'db.sqlite3', # } # }
	Add new DATABASES Section <i>- links to the DATABASE_URL variable on Heroku</i>	DATABASES = { 'default': dj_database_url.parse(os.environ.get("DATABASE_URL")) }

Part 6: Migrating your Database

The following section creates all of the database tables from the models within your app when we run the migrate command. If we find errors in the field names, field type or existing attributes we must run the makemigrations command first. Use caution when changing models if you know there are existing records in the database, as this can lead to errors.

	Step	Code
	Save all files and Make Migrations	python3 manage.py migrate

Creating a Super User

Creating a super user in Django is an important step that creates an admin user. A Django admin user is a special type of user which has access to the backend interface and extra privileges.

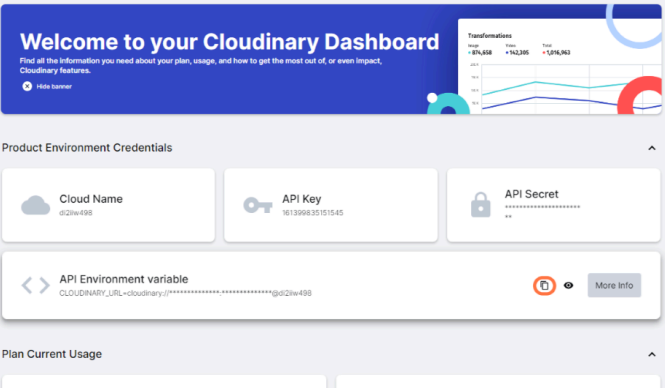
	Step	Code
	Create Super User	python3 manage.py createsuperuser

Part 7: Get our static and media files stored on Cloudinary:

Here we will install the cloudinary and urllib packages. Cloudinary makes it easier for us to upload images and optimize them. urllib is a standard Python library for working with URLs. It provides various modules for fetching data across the web, handling URL encoding and decoding, and managing HTTP requests. Additionally we will create directories for our static files and templates and wire them up in Django settings, then finally we will install whitenoise to efficiently serve our static files. Whitenoise integrates seamlessly with Django, requiring minimal configuration and simplifies the setup for serving static files, eliminating the need for external services.

	Step	Code
	Install Cloudinary	<pre>pip3 install dj3-cloudinary-storage~=0.0.6 pip3 install urllib3~=1.26.15</pre> Freeze your requirements using the freeze command shown in Part 3!

In Cloudinary.com: (Note: must be logged in)

	Step	Code
	Copy your CLOUDINARY_URL e.g. API Environment Variable.	<p>From Cloudinary Dashboard</p> 

In env.py:

	Step	Code
	Add Cloudinary URL to env.py - <i>be sure to paste in the correct section of the link</i>	<pre>os.environ["CLOUDINARY_URL"] = "cloudinary://*****"</pre>

In Heroku:

	Step	Code
	Add Cloudinary URL to Heroku Config Vars - <i>be sure to paste in the correct section of the link</i>	<p>Add to Settings tab in Config Vars e.g. CLOUDINARY_URL, cloudinary://*****</p>

In settings.py:

	Step	Code
	Add Cloundinary Libraries to installed apps	<pre>INSTALLED_APPS = [..., 'django.contrib.staticfiles', 'cloudinary_storage', 'cloudinary', ...,]</pre> <p>(note: order is important)</p>
	Setup Static Files	<pre>STATIC_URL = 'static/' STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'),] STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')</pre>
	Link file to the templates directory in Heroku <i>Place under the BASE_DIR line</i>	<pre>TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')</pre>
	Change the templates directory to TEMPLATES_DIR <i>Place within the TEMPLATES array</i>	<pre>TEMPLATES = [{ ..., 'DIRS': [TEMPLATES_DIR], ..., }, },]</pre>

In the IDE file explorer or terminal:

	Step	Code
	Create 3 new folders on top level directory	media, static, templates

* **Note:** Save all files

	Step	Code
	Install WhiteNoise	pip3 install whitenoise~=5.3.0 Freeze your requirements using the freeze command shown in Part 3!
	Wire up WhiteNoise to Django's MIDDLEWARE in the settings.py file.	'whitenoise.middleware.WhiteNoiseMiddleware', Note: The 'whitenoise' middleware must be placed directly after the Django SecurityMiddleware

In order to test if your project is working properly, and your static files are being served, use the code provided to create a basic view to render a template, add a html template to your templates folder and wire up your urls. Create and link a custom stylesheet to your template before deployment.

	Step	Code
	In project_name urls.py	from django.contrib import admin from django.urls import path, include urlpatterns = [path('admin/', admin.site.urls), path("", include('my_app.urls'), name='home'),]
	In app_name urls.py (create urls.py).	from . import views from django.urls import path urlpatterns = [path("", views.HomePage.as_view(), name='home'),]
	In app_name views.py	from django.shortcuts import render from django.views.generic import TemplateView class HomePage(TemplateView): """ Displays home page" """ template_name = 'index.html'
	In templates folder index.html	{% load static %}

		<pre> <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Django-Setup</title> <link rel="stylesheet" href="{% static 'css/style.css' %}" > </head> <body> <h1>Hello Django From Heroku</h1> </body> </html> </pre>
	In static/css folder style.css	<pre> h1 { color: red; } </pre>

In the Terminal:

	Step	Code
	Add, Commit and Push	<pre> git add . git commit -m "Deployment Commit" git push </pre>

In Heroku:

	Step	Code
	Deploy Content manually through heroku/	E.g Github as deployment method, on main branch