djangoku

A straightforward guide to deploying Django projects on Heroku

Table of Contents

DJANGOKU

Introduction

Every chapter closes with a checklist. That way you can quickly reference important steps, without the noise.

What you will learn

What you need to start

Since this book tackles a rather specific use case, I assume that you have basic knowledge in Python, Django and GIT. That said the setup you will encounter in the next chapters is kept as basic as possible, so even beginners can follow along.

This book will be based on the version stated in the list below. However, it is possible to follow the steps with different versions – either older or newer. The general steps will be the same. But there might be differences in details.

On a personal note: When I started learning programming, I often abandonded tutorials and books, because *my* code didn't work as the one in the book. I want you to succeed and finish this book. If your code does not work as epxected or you encounter a roadblock, don't hesitate to contact me and I will try to help out.

A word about operating systems

The screenshots you will see in this book are taken on a Mac. But you can follow this book on Windows and Linux aswell.

Prerequisites

- Python 3.8+
- Git
- PostgreSQL 11+
- GitLab OR GitHub Account
- Heroku Account

Python and Git

Please make sure you have Python and Git installed on your system. The easiest way is to google e.g. `install python 3 on mac` and `install git on mac` and follow the best results along. Chances are that Python and Git are already available on your system. While the git version is not that important, your Python version should be greater than version 3.

PostgreSQL

Generally speaking you can work with any database that Django and Heroku support. (Note: SQLite is not supported). This book is based on PostgreSQL, because you don't need any additional elements on Heroku and it is easy to setup on Django. Since we are only setting up the database and not do fancy interactions with it, the PostgreSQL version you are using is not significant. While it is alway a good idea to stay recent when starting new projects, you may stick to a version you already have on your system. To see if your version works with Heroku, see this list.

GitLab or GitHub account

In chapter X I will provide two variants of working with a remote repository. You only need one account. So you can work either with GitLab or GitHub.

Heroku account and Heroku CLI

Last but not least, you obviously need an Heroku account to deploy your Django app on Heroku. The good thing about Heroku is, that it is free (how we use it) and scalable (if needed).

Please make also sure that you have installed the Heroku Commandline Interface (CLI).

Checklist

Objective: All prerequisites are fulfilled.

✔ Python 3+ is installed

Terminal (any window)

```
python --version
```

→ Outputs python version 3+

> Depending on how you installed Python 3, you have to use `python3 --version` or even `python3.8 --version` instead.

✔ Git is installed

Terminal (any window)

```
git --version
```

→ Outputs any git version

✔ PostgreSQL is installed

Terminal (any window)

```
postgres --version
```

→ Outputs any PostgreSQL version. Ideally 11+

✔ GitLab or GitHub account created

✔ Heroku account created

✔ Heroku CLI installed

Terminal (any window)

```
heroku --version
```

→ Outputs any Heroku CLI version

Starting the project

Now that all the prerequisites are fullfilled it is time to start the *real* work. We will create our project folder, a local Git repository and a virtual environment. Then we will add Django to the mix.

Beginning with the basics

Git will keep track of our file versioning and will be our gateway to distribute the files to GitLab and Heroku later. By using a virtual environment we have an isolated sandbox and don't get confused with Python versions and modules that may be installed on the operating system already. Both Git and a virtual environment are mandatory for a streamlined Heroku workflow. But it is generally a good practice to start any Python projects that way.

In this book the project will be called *"djangoku_dev"* and you will see that the root path to the project is `~/Dropbox/GitMac/djangoku/djangoku_dev`. Obviously you will have your custom root path and maybe a more suitable project name –– Keep in mind that you have to adjust some commands accordingly.

Create the project folder

Create a project folder and navigate with the terminal to it:

Terminal (any window)

```
mkdir ~/Dropbox/GitMac/djangoku/djangoku_dev
cd ~/Dropbox/GitMac/djangoku/djangoku_dev
```

> If you are on the Mac, there is an easy way to do this: Create the project folder in the Finder, open a new Terminal window, write `cd `, drag the folder from the Finder into the Terminal window and hit *Enter*

Initiate a new Git repository

Initiate a new Git repository:

Terminal (same window)

```
git init
```

Create the virtual environment

> If you are confused about virtual environments (as I still am sometimes), check out this great article on Real Python to get an overview.

In this book the virtual environment will be called *virtualenv*. You can name your virtual environment any way you want and adjust the commands accordingly.

Create a new virtualenvironment in your project folder and activate it:

Terminal (same window)

```
python3 -m venv virtualenv
source virtualenv/bin/activate
```

You can see that you have activated the virtual environment if the project foldername is wrapped in parenthesis e.g. *(djangoku_dev)*

This virtual environment folder is needed only locally. GitLab and Heroku will create their own environments later. Therefore it is important to not track the virtual environment folder with Git.

Create a *.gitignore* file and add *virtualenv* to it:

Terminal (same window)

```
echo virtualenv > .gitignore
```

Adding your first commit

Now that we have created the base layer for our project, we should do our first commit.

Stage all changed files – that's just the *.gitignore* file as we ignored everything else – and commit them:

Terminal (same window)

```
git add .
git commit –m "Start Djangoku 🌱"
```

Django. Finally!

After all this preparation and groundwork it is finally time to add Django.

Install Django

Make sure the virtualenvironment is still activated by checking if the project folder is wrapped in parenthesis.

Install the Django module:

Terminal (same window)

```
pip install django
```

Start the Django project

Now that the Django module was installed, we can start the actual Django project. Note that the command has a `.` at the end – this will make sure we create the Django project in the current directory and not create a new one.

In this book the Django project will be called *djangoku*. You can call your django project differently and adjust the commands accordingly.

Start the Django project:

Terminal (same window)

```
django-admin.py startproject djangoku .
```

If you look into your root project folder you should see that a file called *manage.py* and a *djangoku* folder were added.

Run Django:

Terminal (same window)

```
python manage.py runserver
```

After you have run the command, you should see an address like this: `http://127.0.0.1:8000/`. Open this in your browser and see if Django is running successfully.

Ignore the SQLite database

Django creates an SQLite database automatically. We will not need this file and delete it later, because we will work with a PostgreSQL database. However, we can ignore this for now. And by ignoring, I mean: .gitignoring it. Note that the command has two `>`, because we want to append a line to the *.gitignore* file.

> There are other Django projects, where a SQLite database is good enough. Even then you should always ignore *.sqlite3* files.

Terminal (same window)

```
echo *.sqlite3 >> .gitignore
```

Commit

Track the new files in Git:

Terminal (same window)

```
git add .
git commit –m "Add Django project 🤠"
```

Checklist

Objective: The project is tracked via Git and Django is running in an activated virtual environment.

✔ Project is tracked via Git

Terminal (same window)

```
git log
```

→ Outputs the last commit messages

✔ Virtual environment is activated

Terminal (same window)
→ The root project folder is wrapped in parenthesis in the Terminal.

✔ Virtual environment uses the correct Python

Terminal (same window)

```
which python
```

→ Outputs a path that leads into the virtual environment folder.

✔ Django is installed

Terminal (same window)

```
python –m django ––version
```

→ Outputs Django version.

✔ Django works

Browser (any window)
→ Visit the URL that Django showed on startup and see if it shows the Django success message.