Introduction to Django and Getting Started

Chapter 1

1

Objectives

- What is Django?
- Initial setup with PythonAnywhere
- What is virtual environment?
- Understanding Django's project structure

2

What is Django?

- The world of Python web frameworks is full of choices. Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, Falcon, and many more are competing for developer mindshare.
- **Django** is a free and open source web framework, written in Python that encourages rapid development.
- Django is pronounced JANG-oh. The “D” is silent.
- Basically, it follows the MVC (Model-view-controller) pattern, with its own modification to be called the MTV framework (Model-Template-View)
- Django provides all basic features that are part of a generic web application: authentication, security and data management.
- Includes ORM that supports many databases – Postgresql, MySQL, Oracle, SQLite.

3

What is Django? (cont’d)

- Named after famous Guitarist “Django Reinhardt”
- Developed by Adrian Holovaty and Simon Willison
- Open sourced in 2005
- First Version released September 3, 2008

2.2 LTS[58]	1 Apr 2019	Security release. *Supported until at least April 2022*
3.0[59]	2 Dec 2019	ASGI support
3.1[60]	4 Aug 2020	Asynchronous views and middleware
3.2 LTS[61]	Apr 2021	*Extended Support until at April 2024*
4.0[61]	Dec 2021	*Extended Support until at April 2023*

Old version | Older version, still maintained | **Latest version** | Latest preview version | Future release

4

Django

- After nearly 14 years of growth, Django continues to grow in popularity.
- Djangosites lists over 5000 sites using Django, and that is only for sites that register with Djangosites. It would be impossible to guess how many pages Django serves every day.
- For a list of websites powered by Django, can visit https://djangosites.org/ .
- Take a look at some of the popular websites powered by Django: https://djangostars.com/blog/10-popular-sites-made-on-django/

5

Packages, Packages and More Packages!

- Many of Django's large international community of developers give back to the community by releasing their projects as open-source packages.
- You will find the largest repository of these projects on the Django Packages site https://djangopackages.org/
- A quick tour of popular packages includes:
  - **Cookiecutter**. Quick and easy setup of Django project and app structures for more advanced applications
  - **Django REST Framework**. Implements a REST API in Django
  - **Django allauth**. Facebook, GitHub, Google and Twitter authentication for your Django apps
  - **Debug toolbar**. Display debug information as your project is running
  - **Django Celery**. Provides Celery integration for Django
  - **Oscar**, **Django Shop** and **Cartridge**. E-commerce frameworks for Django (Cartridge is an extension for Mezzanine CMS)

6

Common tasks supported by Django

- Django supports the common tasks in web development:
  - user authentication
  - templates, routes, and views
  - admin interface
  - robust security
  - support for multiple database backends
  - and much much more

PythonAnywhere

- We will be using PythonAnywhere, a PaaS (Plaform as a Service) for Python web applications.
- PythonAnywhere is a tool for us to host, run and code Python in the cloud.
- You can register a free beginner's account, the name of which will be used for your blog's URL in the form yourusername.pythonanywhere.com.
- **Please use your student ID, P19XXXXX as the account name so that your blog's URL will take the form, P19XXXXX.pythonanywhere.com.**
- Refer to the details of Lab 1 on the steps to setup PythonAnywhere to have a django site live and on the Internet, **yourusername.pythonanywhere.com** from a browser.

The Bash Console (PythonAnywhere)

- The Bash console is a *textual* way to interact with the system, just as the 'desktop', in conjunction with your mouse, is the *graphical* way to interact your system.
- Some common commands:
  - cd (change down a directory)
  - cd .. (change up a directory)
  - ls (list files in your current directory)
  - mkdir (make directory)
  - zip -r myzipfile my_folder_name (to create a zip file)

What is virtual environment?

- You might be running several Python applications that require a different version to run. For example, you want to switch to the new version of Django, but still want to maintain your Django 1.11 project.
- The solution is to use virtual environments.
- Virtual environments (virtualenv or venv) allow multiple installations of Python and their relevant packages to exist together in harmony.
- In the Bash console, run the command to create a virtual environment called django2 with a particular version of python.

```
17:12 ~ $ mkvirtualenv django2 --python=/usr/bin/python3.8
(django2)17:13 ~ $
```

- Note the change in the prompt after the successful creation of virtual environment. Next, you can run the following command to install a particular version of django

```
(django2)17:13 ~ $ pip install django==2.2.17
```

What is pip?

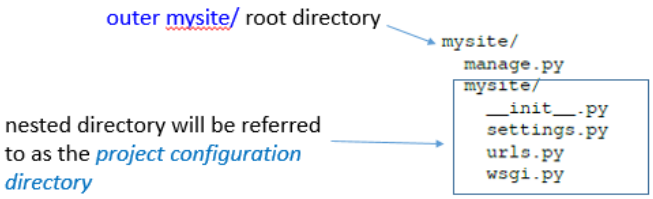
- PIP (Python Install Package): the standard package manager for Python.
- Pip allows you to install and manage additional packages that are not part of the Python standard library.
- Pip provides a simple, clean means of adding (or removing) high-quality third party code libraries to your Django project.
- Django itself is installed with pip, which is why we have to begin by installing pip.

Creating a Django project

- You should now see parentheses on your bash console with the name of virtual environment activated. For example, something like
  (django2)17:18 ~ $
- If the virtual environment name django2 is missing, you have to run the following command to activate it.
  $ workon django2
- Change directory to the corresponding folder and create a new Django project called mysite with the following command.
  (django2)17:18 ~/django_projects $ django-admin startproject mysite

Django project structure

- If you just run django-admin startproject mysite then by default Django will create the following directory structure.



- See how it creates a new directory mysite and then within it a manage.py file and a mysite directory.

Django project structure (cont'd)

```
mysite/
  manage.py
  mysite/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

- The outer mysite/ root directory is a container for your project.
  - manage.py, a command-line utility that lets you interact with your Django project.
  - The inner mysite/ directory is the Python package for your project. It's the name you will use to import anything inside it (for example, mysite.urls).
    - mysite/__init__.py, an empty file that tells Python that this directory should be considered a Python package.
    - mysite/settings.py, settings and configuration for this Django project.
    - mysite/urls.py, the URL declarations for this Django project.
    - mysite/wsgi.py, an entry-point for WSGI-compatible web servers to serve your project -- This is *not* the one you need to change to set things up on PythonAnywhere -- the system here ignores that file.

We will update these 2 files later on.

Django Settings

- The settings.py file contains the configuration information for your Django project.
- When you ran startproject, Django created several common settings with default values for you.
- There are numerous settings available —core settings for database configuration, caching, email, file uploads and globalization, and a range of additional settings for authentication, messaging, sessions and static file handling.

15

/home/IP2021/django_projects/myTestSite/myTestSite/settings.py

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '[redacted]'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

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',
]

ROOT_URLCONF = 'myTestSite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

If true: Displays full stack of errors on request pages for quick analysis.
If false: Displays custom error pages without any stack details to limit security threats.
Leave it to be true while debugging.

- The purpose of ALLOWED_HOSTS is to validate a request's HTTP Host header.
- If ALLOWED_HOSTS is empty, Django refuses to serve requests and instead responds with HTTP 400 bad request pages, since it can't validate incoming HTTP Host headers.
- **Define ALLOWED_HOSTS=['yourusername.pythonanywhere.com']**, which would only accept requests with an HTTP Host yourusername.pythonanywhere.com.
- In a similar fashion, if you want to accept any HTTP host -- effectively bypassing the verification -- you would define ALLOWED_HOSTS=['*'] which indicates a wild-card.
- Use ALLOWED_HOSTS to only accept requests from trusted hosts.

We have just created a project. The next step is to create the applications in this project and we will then add the names of the applications here.

16

Never deploy a site into production with DEBUG turned on

Some notes on wsgi.py

- The Web Server Gateway Interface (**WSGI**, pronounced whiskey or WIZ-ghee) is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language.
- Django works using the "WSGI protocol", which PythonAnywhere supports. This file's job is to tell PythonAnywhere where our web app lives and what the Django settings file's name is.
- As well as WSGI, Django also supports deploying on ASGI (Asynchronous Server Gateway Interface), the emerging Python standard for asynchronous web servers and applications. However, ASGI not yet supported by Pythonanywhere.

Django Applications

- You might have noticed there is no real program code in your project so far—you have a settings file with configuration information, an almost empty URLs file and a command-line utility that launches a website which doesn't really do anything.
- This is because to create a functioning Django web application, you need to create Django applications.
- A Django application (or app for short) is where the work is done. This will be covered in the next chapter.
- A Django project is the collection of apps and configuration settings that make up a Django web application.

19