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.
- **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

Version	Release Date[34]	End of mainstream support	End of extended support	Notes[35]
3.2 LTS[57]	6 Apr 2021	7 Dec 2021	April 2024	Tracking many to many relationships, added support for Python 3.11
4.0[58]	7 Dec 2021	3 Aug 2022	April 2023	Support for `pytz` is now deprecated and will be removed in Django 5.0.
4.1[59]	3 Aug 2022	April 2023	December 2023	Asynchronous ORM interface, `CSRF_COOKIE_MASKED` setting, outputting a form, like `{{ form }}`
4.2 LTS[60]	3 Apr 2023	December 2023	April 2026	Psycopg 3 support, `ENGINE` as `django.db.backends.postgresql` supports both libraries.
5.0	4 Dec 2023	August 2024	April 2025	

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

https://en.wikipedia.org/wiki/Django_(web_framework)

4

Django

- After nearly 14 years of growth, Django continues to grow in popularity.
- Djangosites (https://djangosites.org/)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.
- 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/

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, P21XXXXX as the account name so that your blog's URL will take the form, P22XXXXX.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'.
- 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)

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 (4.2) of Django, but still want to maintain your Django 3.2 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.

Versions of Python

Version	Latest micro version	Release date	End of full support	End of security fixes
3.8	*3.8.18*[62]	2019-10-14[62]	2021-05-03[b][62]	2024-10[62]
3.9	*3.9.18*[63]	2020-10-05[63]	2022-05-17[b][63]	2025-10[63][64]
3.10	*3.10.13*[65]	2021-10-04[65]	2023-04-05[b][65]	2026-10[65]
3.11	*3.11.7*[66]	2022-10-24[66]	2024-04-01[66]	2027-10[66]
3.12	*3.12.1*[67]	2023-10-02[67]	2025-05[67]	2028-10[67]
3.13	*3.13.0a2*[68] [*needs update*]	2024-10-01[68]	2026-05[68]	2029-10[68]
Legend:		Old version	Older version, still maintained	**Latest version**	Latest preview version	Future release	
Italics indicates the latest micro version of currently supported versions as of 2023-12-10.				

https://en.wikipedia.org/wiki/History_of_Python

11

Creating virtual environment

- In the Bash console, run the command to create a virtual environment called django with a particular version of python.

  17:12 ~ $ mkvirtualenv django --python=/usr/bin/python3.8

  (django)17:13 ~ $

- Note the change in the prompt after creating virtual environment.
- You should now see parentheses on your bash console with the name of virtual environment activated.
  (django)17:13 ~ $
- If the virtual environment name django is missing, activate it with

  $ workon django
- To delete a virtual environment,

  (django)17:13 ~ $ rmvirtualenv django

12

What is pip?

- Next, run the command to install a particular version of django. Confirm that the virtual environment django is activated.

```
(django)17:13 ~ $ pip install django==3.2
```

- PIP (Python Install Package): the standard package manager that allows you to install and manage additional packages that are not part of the Python standard library.

13

Creating a Django project

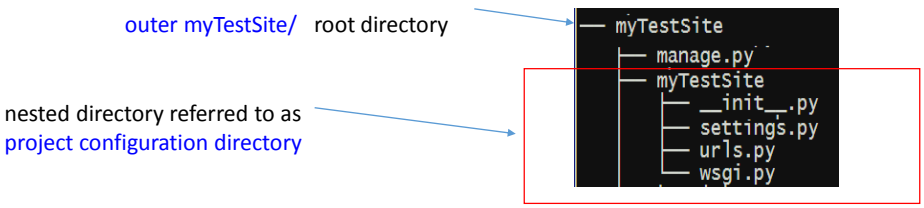
- **Create** a folder to hold all your Django project files.
  - You DON'T have to do this step in the future if this folder already exists
  - This step is mainly for better organization of files, not mandatory
  - We make a directory with "mkdir" and then "cd" to change directory to it.
- Change directory to the corresponding folder and create a new Django project called myTestSite with the following command.

```
(django)17:18 ~/django_projects $ django-admin startproject myTestSite
```

This is simply a folder planned to store all of our Django projects

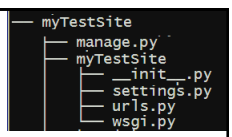
Django project structure

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



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

Django project structure (cont'd)



- The outer myTestSite/ root directory is a container for your project.
  - manage.py, a command-line utility that lets you interact with your Django project.
  - The inner myTestSite/ directory is the Python package for your project. It's the name you will use to import anything inside it (for example, myTestSite.urls).
    - myTestSite/__init__.py, an empty file that tells Python that this directory should be considered a Python package.
    - myTestSite/settings.py, settings and configuration for this Django project.
    - myTestSite/urls.py, the URL declarations for this Django project.
    - myTestSite/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.

17

/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.

18

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.

20

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.

21

Summary

Wrap up of what we have achieved using PythonAnywhere!

1. Create a virtual environment and install Django in it.

```
17:10 ~ $ mkvirtualenv django --python=/usr/bin/python3.8
(django)17:12 ~ $ pip install django==3.2
```

2. While in the virtual environment created (as seen from the prefix with the parenthesis enclosing the name of virtualenv), create a directory (e.g. django_projects) to hold all your Django project files, if such a folder does NOT already exist.

```
(django)17:18 ~ $ mkdir django_projects
(django)17:18 ~ $ cd django_projects
```

3. Inside the django_projects directory (as seen in the prefix of the command), create a project with the command `django-admin startproject yourProjectNameHere`

```
(django)17:18 ~/django_projects $ django-admin startproject myTestSite
```

4. You can use the "`ls`" command to list the content of the files created for you as a result.

22