Building the first Web application with views and URLConfs

Chapter 2

Objectives

- In this chapter we'll build a Django app that simply says "Hello, World" on the homepage.
- Understanding the use of views and URLConfs.
- What is **MTV framework?**

What is a Django application?

- Django uses the concept of projects and apps to keep code clean and readable.
- A single Django project contains one or more apps within it that all work together to power a web application.
- Remember that we have created a project inside the django_projects folder with the startproject command for a new Django project in Chapter 1 as follows:

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

- For example, a real-world Django e-commerce site might have one app for user authentication, another app for payments, and a third app to power item listing details. Each focuses on an isolated piece of functionality.

What is a Django application? (cont'd)

- The term **application** describes a Python package that provides some set of features. Applications may be reused in various projects.
- A Django application exists to perform a particular task. You need to create specific applications that are responsible for providing your site with particular kinds of functionality.
- Applications include some combination of models, views, templates, template tags, static files, URLs, etc.
- They are wired into the projects with the **INSTALLED_APPS** setting.

Step 1: Create an app

- From the command line, remember **to change directory (cd)** to the project folder (the folder created with the command django-admin startproject) before creating the app.
- Remember that a single Django project contains one or more apps within it that all work together to power a web application
- Use the startapp command to create an app called "pages" in the project myTestSite (created in Chapter 1).

```
(django) 05:34 ~/django_projects/myTestSite $ python manage.py startapp pages
```

- If you look inside the directory, you'll see Django has created a pages directory with these files:

```
├── pages
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
```

The app files

- admin.py is a configuration file for the built-in Django Admin app
- apps.py is a configuration file for the app itself
- migrations/ keeps track of any changes to our models.py file so our database and models.py stay in sync
- models.py is where we define our database models, which Django automatically translates into database tables
- tests.py is for our app-specific tests
- views.py is where we handle the request/response logic for our web app

Step 2: Edit settings.py (INSTALLED_APPS)

- Even though our new app exists within the Django project, Django doesn’t “know” about it until we explicitly register it.
- In your text editor open the settings.py file and scroll down to INSTALLED_APPS where you’ll see six built-in Django apps already there. Add our new pages app at the bottom.

```
settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'pages', # new
]
```

Views and URLConfs

- Now that you have created an application, to empower it, the contents of the page are produced by a **view function**, and the URL is specified in a **URLconf**.
- In Django, Views determine what content is displayed on a given page while URLConfs determine where that content is going.
- When a user requests a specific page, like the homepage, the URLConf maps that request to the appropriate view function which then returns the correct data.
- Let's get started and write our first view function.

Step 3: Updating the views.py file in our pages app

- Basically, we're saying whenever the view function homePageView is called, return the text "Hello, World!"

```
# pages/views.py
from django.http import HttpResponse
def homePageView(request):
    return HttpResponse('Hello, World!')
```

- To define a function in Python, you use the def statement.
- We've created a function called homePageView that accepts the request object and returns a response with the string Hello, World!.

Your First URLconf

- Our project doesn't yet know about the homePageView yet; we need to tell Django explicitly that we're activating this view at a particular URL.
- To hook a view function to a particular URL with Django, we use a URLconf.
- A URLconf is like a table of contents for your Django-powered web site.
- Basically, it's a mapping between URLs and the view functions that should be called for those URLs.
- It's how you tell Django, *For this URL, call this code, and for that URL, call that code*.
- We will have a project-level urls.py file and an application-level urls.py file to handle the mapping between URLs and the view functions.

Step 4: Configure our project-level urls.py file

- It's common to have multiple apps within a single Django project, so they each need their own route.
- Update the myTestSite/urls.py file as follows:

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

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

Two updates are made to the existing content:
- We've imported include on the second line next to path and then created a new urlpattern for our pages app.
- Now whenever a user visits the homepage at yourusername.pythonanywhere.com/, they will first be routed to the application-level urls.py (pages.urls)

Step 5: Create and edit application-level urls.py file in our pages app

- Now we need to configure our urls. Within the pages app, create a new urls.py file.
- Then update it with the following code:

```
# pages/urls.py
from django.urls import path
from . import views

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

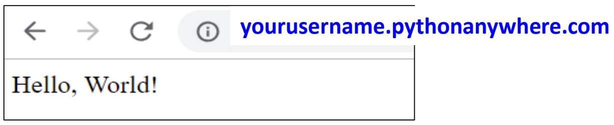
The period reference the current directory, which is our pages app containing both views.py and urls.py.

Our urlpattern has 3 parts:
- A Python regular expression for the empty string ' '
- Specify the view which is called homePageView
- Add an optional url name of 'home'

If the user requests the homepage, represented by the empty string '', then use the view called homePageView.

Step 6: Test if our pages app works

- To confirm everything works as expected, refresh the browser for **yourusername.pythonanywhere.com**, it now displays the text “Hello, world!”


yourusername.pythonanywhere.com

Hello, World!

So What Just Happened?

1. Our browser (client) sent a message to the server requesting it return content located at the root URL (**yourusername.pythonanywhere.com**).
2. Django then looked for a URL pattern that matches the request, by first searching the site level (project-level) urls.py, and then each of the apps for a urls.py file containing a pattern that matches.
3. Django checks the first pattern (admin/) in our site level urls.py, which doesn’t match, so it moves on to the second line in which the empty string (root URL) matches.
4. The matching pattern includes the urls.py from the pages app. Basically this include says “go look in the pages app for a pattern that matches”.
5. Once in the app-level urls.py, the empty string matches again, but this time the request is sent to the homePageView.
6. The homePageView then renders our simple HTML message to a HttpResponse and sends it to the browser.
7. The browser renders the response and we see our page heading.

Every Django application follows this same basic process each time it receives a request from the browser.

https://www.youtube.com/watch?v=RLo9RJhbOrQ

Why the need for project level and application level urls.py?

- When startproject created our website, it created a urls.py file in our project folder (e.g. \myTestSite\urls.py).
- This is a good place for site-wide navigation but is rarely a good place to put URLs relating to individual applications.
- Not only is having all our URLs in one file more complex and less portable, but it can lead to strange behavior if two applications use a view with the same name.
- To solve this problem, we create a new urls.py file for each application.

Wrap up of the steps to create a Django app

Inside your project folder (the folder created with django-admin startproject) with the corresponding virtual environment

- Step 1: Create an app

  ```
  python manage.py startapp pages
  ```
- Step 2: Edit settings.py (INSTALLED_APPS)
- Step 3: Update the views.py file in our pages app
- Step 4: Create and edit application-level urls.py file in our pages app
- Step 5: Configure our project-level urls.py file

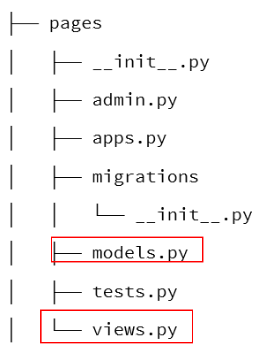
The Big Picture - How Django is Structured

- Django is modeled around a *Model-View-Controller* (MVC) framework. MVC is a software design pattern that aims to separate a web application into three interconnecting parts:
  1. The **model**, which provides the interface with the database containing the application data;
  2. The **view**, which decides what information to present to the user and collects information from the user; and
  3. The **controller**, which manages the business logic for the application and acts as an information broker between the model and the view.
- Django uses slightly different terminology in its implementation of MVC. In Django:
  1. The **model** is functionally the same. Django's Object-Relational Mapping (ORM—more on the ORM later) provides the interface to the application database;
  2. The **template** provides display logic and is the interface between the user and your Django application; and
  3. The **view** manages the bulk of the applications data processing, application logic and messaging.

Django implementation of MVC pattern – MTV framework

Django uses the following components to describe the MVC pattern:

- **models.py** contains a description of the database table as a Python class. Using this class, you can create, retrieve, update, and delete records in your database using simple Python code rather than writing repetitive SQL statements; details to be discussed later.
- **views.py** contains the business logic for the page.

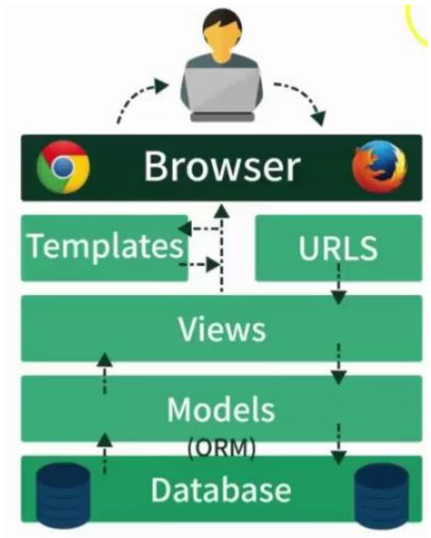


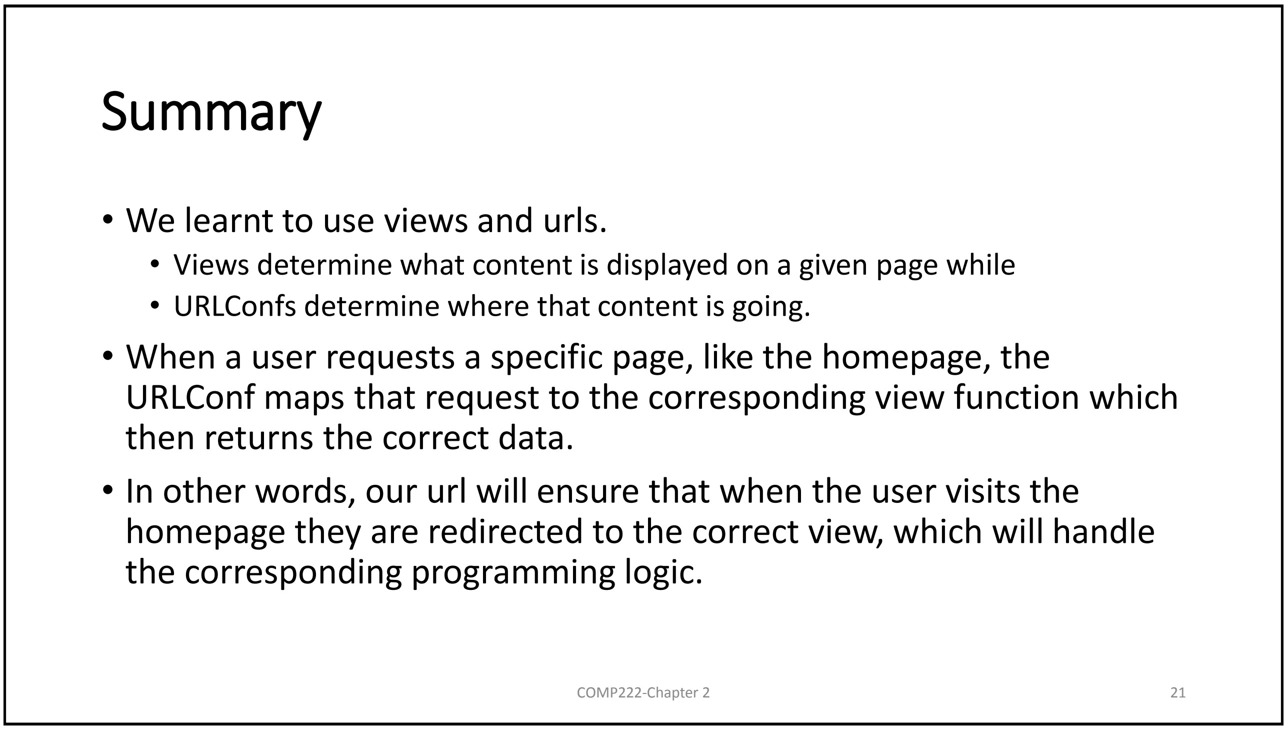
```
├── pages
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
```

MTV framework

- Django follows the MVC pattern closely, however, it does use its own logic in the implementation.
- Django is often referred to as an *MTV framework*.
- In the MTV development pattern,
  - **M stands for "Model,"** the data access layer. This layer contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data.
  - **T stands for "Template,"** the presentation layer. This layer contains presentation related decisions: how something should be displayed on a web page or other type of document.
  - **V stands for "View,"** the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template(s). You can think of it as the bridge between models and templates.

MTV framework





Summary

- We learnt to use views and urls.
  - Views determine what content is displayed on a given page while
  - URLConfs determine where that content is going.
- When a user requests a specific page, like the homepage, the URLConf maps that request to the corresponding view function which then returns the correct data.
- In other words, our url will ensure that when the user visits the homepage they are redirected to the correct view, which will handle the corresponding programming logic.