Django by Example

Tuesday, February 25, 2025 2:14 PM

any Django project, you will always work with models, views, templa
u will learn how they fit together.

architecture

ow Django processes requests and how the request/response cycle is
Django components – URLs, views, models, and templates:

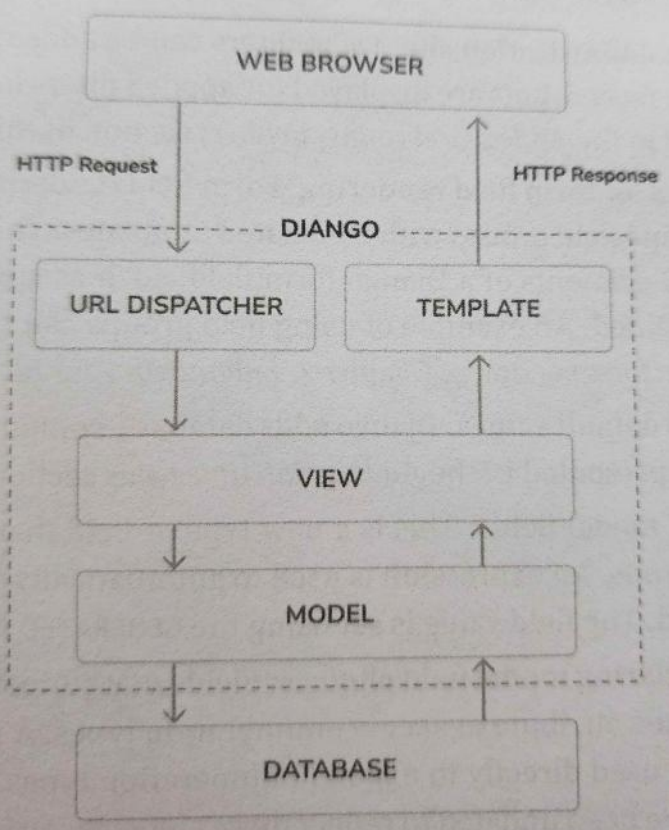


Figure 1.1: The Django architecture

1. **Web browser requests a page by its URL and web server passes the HTTP request to Django**
   - Used WSGI or ASGI to bridge between web server and django
   - Middleware processes defined in settings.py
2. **Django runs through its configured URL patterns and stops at the first one that matches requested URL**
   - Determines which views function or class to use
   - path('', views.post_list, name='post_list')
3. **Django executes the view that corresponds to the matched URL pattern**
   - The matched views function is called
4. **The view potentially uses data models to retrieve information from the database**
   - Views can process data, interact with models, perform any logic
   - Views queries the SQL database

- Renders html templates
- Uses the model to fetch, update, or delete data from the database

5. **Data models provide data definitions and behaviours. They are used to query the database**
6. **The view renders a template (usually HTML) to display the data and returns it with an HTTP response**

- Creates a HttpResponse or subclass of it.
- Passes through middleware in reverse order
- Sent back through the WSGI or ASGI

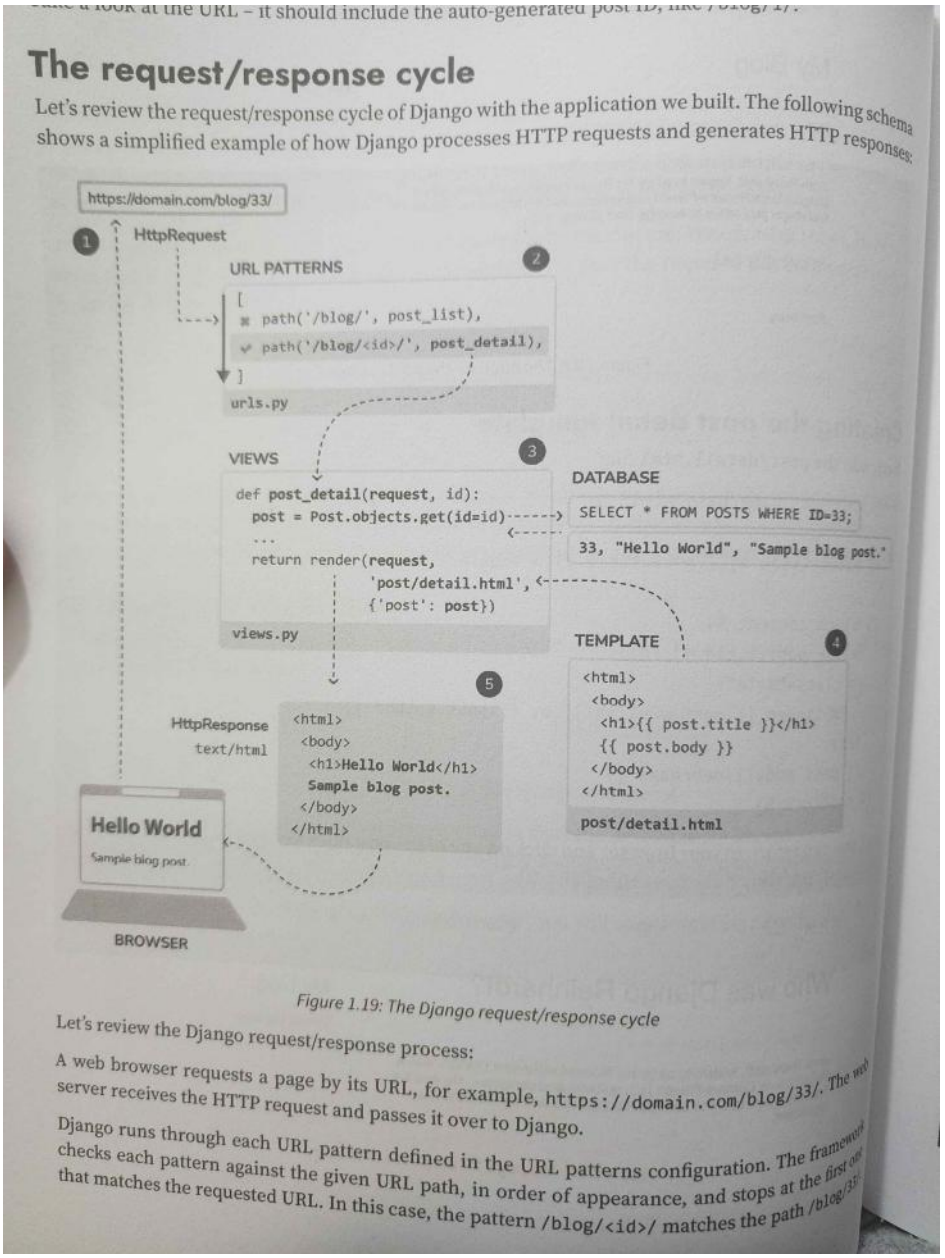


Figure 1.19: The Django request/response cycle

Let's review the Django request/response process:

A web browser requests a page by its URL, for example, `https://domain.com/blog/33/`. The web server receives the HTTP request and passes it over to Django.

Django runs through each URL pattern defined in the URL patterns configuration. The framework checks each pattern against the given URL path, in order of appearance, and stops at the first one that matches the requested URL. In this case, the pattern `/blog/<id>/` matches the path `/blog/33/`

Vocabulary

Thursday, February 27, 2025 11:51 PM

WSGI: Web Server Gateway Interface
ASGI: Asynchronous Server Gateway Interface

CLI and Shell Commands

Friday, February 28, 2025 12:43 PM

To create the file structure for a new django project named mysite
Django-admin startproject mysite

To create the file structure for a new django application named blog
Python manage.py startapp blog

To apply all database migrations
Python manage.py migrate

To create migrations for the models of the blog application
Python manage.py makemigrations blog

To view the SQL statements that will be executed with the first migration of the blog application
Python manage.py sqlmigrate blog 0001

To run the django development server:
Python manage.py runserver

To run the development server specifying host/port and settings file:
Python manage.py runserver 127.0.0.1:8001 --settings=mysite.setings

To run Django shell
Python manage.py shell

To create a suepruser using the django authentication framework:
Python manage.py createsuperuser

To access variables and methods of classes first import the .py

Can then access the variables through for example

```
class Post(models.Model):
    class Status(models.TextChoices):
        DRAFT = 'DF', "Draft"
        PUBLISHED = 'PB', "Published"

    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(
        max_length=2,
        choices=Status,
```

```
    default=Status.DRAFT
)
```

Post.Status.choices
[('DF', 'Draft'), ('PB', 'Published')]
Post.Status.labels
['Draft', 'Published']
Post.Status.values
['DF', 'PB']
Post.Status.names
['DRAFT', 'PUBLISHED']

Migrating models.py to the SQL server

Python manage.py makemigrations (app name)

- Scans for changes in your models
- Create a migrations for SQL POST model
- You should see a migrations folder appear

This does not create the tables in SQL
Output should look like this.

```
CREATE TABLE "blogapp_post" (
"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(250) NOT NULL,
"slug" varchar(250) NOT NULL,
 "body" text NOT NULL,
"publish" datetime NOT NULL,
 "created" datetime NOT NULL,
 "updated" datetime NOT NULL,
"status" varchar(2) NOT NULL,
 "author_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED);
```

Python manage.py migrate
Takes the generated migration files and applies them to your database

Creating a Superuser

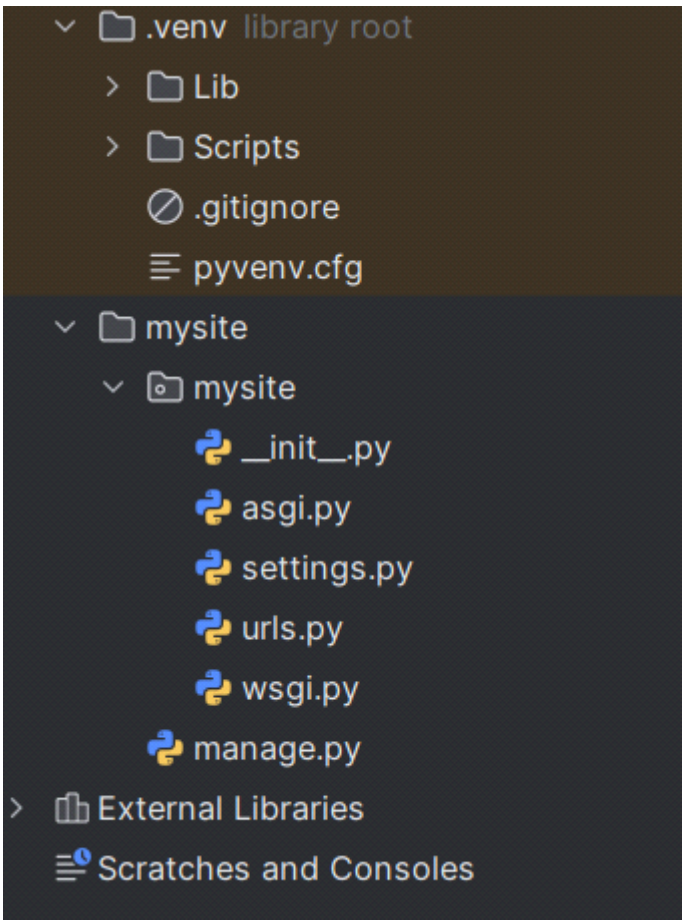
Python manage.py createsuperuser
Asks for a:

- Username
- Email
- Password

Start a Django Project

Thursday, February 27, 2025 10:43 PM

1. pip install django
2. django-admin startproject mysite



Manage.py: Command line utility used to interact with your projects. Wont usually need to edit this file.

Mysite/: this is a python package for your project which consists of the following files.

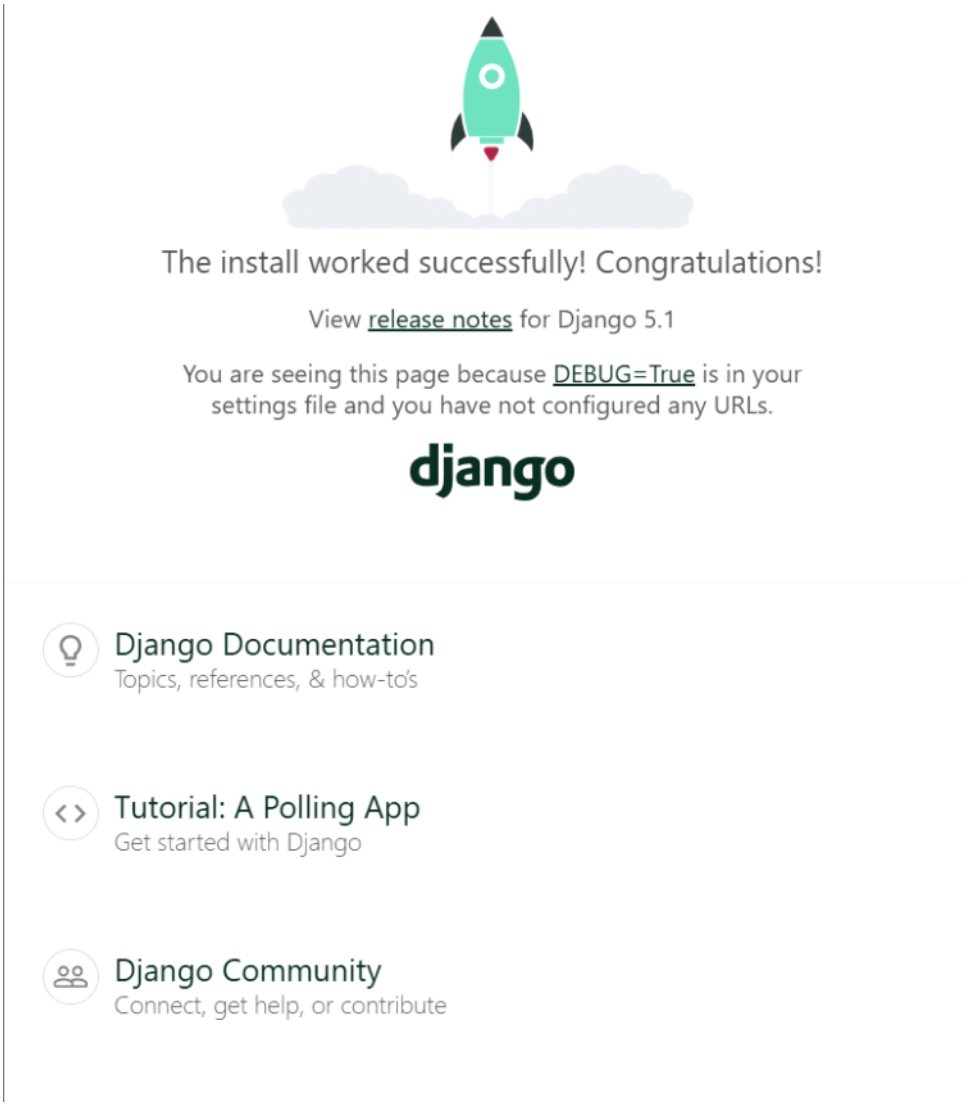
- __init__.py: an empty file that tells python to treat the mysite directory as a python module
- asgi.py: This is the configuration to run your project as an ASGI application with ASGI compatible web servers ASGI is the emerging python standard for asynchronous web servers and applications
- Settings.py: This indicates settings and configuration for your projects and contains initial default settings
- Urls.py: This is the place where your URL patterns live. Each URL defined here is mapped to a view
- Wsgi.py: This is the configuration to run your project as a web server gateway interface {WSGI) application with WSGI compatible web servers

To complete project setup need to create tables associated with the models of the default django applications included in the INSTALLED_APPS setting.

3. Cd mysite
4. Python manage.py migrate

Finally run the server

5. Python manage.py runserver



Create the app folder

1. Django-admin startapp (app_name)

2. Add app to installed_apps in settings.py
3. Open urls.py under project folder
4. Add app path() to urls

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

5. Under app views.py import render

```
def index(request):
    return render(request, 'portfolio_app/index.html')
```

6. Create a urls.py in app folder
7. Add urlpatterns list

```
from django.urls import path
from . import views

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

8. Create HTML templates folder

Applying initial database migrations

Thursday, February 27, 2025 11:53 PM

Django apps require a database to store data

Settings.py contains database configurations for project in the DATABASES setting. Default is a sqlite3 database

Settings.py also includes a list names INSTALLED_APPS that contains common Django applications that are added to your project by default

Django apps contain data models that are mapped to database table

Urls.py

Monday, March 3, 2025 6:52 PM

path()

Path is used for url routing. It lets you map url patterns to views to view functions or classes in your application

Parameters:

Route: A string that defines a url pattern

View: The function or class that should be called if the pattern matches

Kwargs(optional): any addition keyword arguments to pass to the view

Name(optional): A unique name for the URL pattern, which is useful for URL revising

Views.py

Sunday, March 2, 2025 9:29 PM

A django view is a python function that recieves a web request and returns a web response. All the logic to return the desired response goes inside the view

Takes a http request in as default. Then renders the request with the .html page. Can also take data from models and apply to the html page. For example a comment from a Comment class inside models. It Takes the comment and sends it into the html file where it accepts comments

{% for comment in comments %}

```
def post_list(request):
    post_list = Post.published.all()
    #Pagination with 3 posts per page
    paginator = Paginator(post_list, 3)
    page_number = request.Get.get('page', 1)
    posts = paginator.page(page_number)

    return render(
        request,
        'blog/post/list.html',
        {'posts': posts}
    )


def post_detail(request, year, month, day, post):
    post = get_object_or_404(
        Post,
        status=Post.Status.PUBLISHED,
        slug=post,
        publish__year=year,
        publish__month=month,
        publish__day=day
    )
    return render(
        request,
        'blog/post/detail.html',
        {'post': post}
    )
```

Benefits of class based views

- Organize code relates to HTTP methods such as GET, POST, or PUT, in separate methods instead of using conditional branching
- Use multiple inheritance to create reusable view classes

Paginator:

from django.core.paginator import Paginator

Adds pages to applications.


```
#Pagination with 3 posts per page
paginator = Paginator(post_list, 3)
page_number = request.Get.get('page', 1)
posts = paginator.page(page_number)

return render(
    request,
    'blog/post/list.html',
    {'posts': posts}
)
```

1. Instantiate the Paginator class with the number of objects to return per page
2. Retrieve the page GET HTTP parameter and store it in the page_number variable.
3. Obtain the objects for the desired page by calling the page() method of Paginator. Returns a Page object that we store in the posts variable
4. Pass the posts object into the HTML template

Models.py

Friday, February 28, 2025 12:16 PM

In order to add models to the administration site go to admin.py notes

Whenever data inside the model Class is changed you must make migrations:

- **python manage.py makemigrations (appname)**
  - Output will show a .py file name that is saved inside migrations folder of app
- **python manage.py migrate**
  - Output will show running migrations:
  - Applying (appname).(migration)…OK

This is where objects in the database are made

For example

```
class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

This is a blog post object storing title, slug, body, publish date, created date, and updated date.

Using class Meta: allows for metadata of the models

```
class Meta:
    # Using - means posts will appear in reverse order
    ordering = ['publish']
    indexes = [models.Index(fields=['-publish'])]
```

Creating Model Managers

```
class PublishedManager(models.Manager):
    def get_query(self):
        return(
            super().get_queryset().filter(status=Post.Status.PUBLISHED)
        )


class Post(models.Model):

    objects = models.Manager()
    published = PublishedManager()
```

The first manager declared in a model becomes the default manager. If none is declared django automatically creates one

get_queryset() returns the QuerySet that will be executed. The .filter creates a custom QuerySet for published

Reverse Method:

- reverse(Viewname, args)
- Builds the URL dynamically using the URL name defined in the URL patterns of urls.py
- Remember that the blog namespace is defined in the main urls.py when including the URL patterns from blog.urls. The post_detail URL is defined in the apps urls.py
- 'blog:post_detail; can be used globally to refer to the post detail URL.
- Post_detail requires an ID which is the id of the Post object [self.id]

Settings.py

Friday, February 28, 2025 12:24 PM

Here is where we add installed apps to the site

DEBUG - boolean that returns debug mode of the project. If set to true we allow others to see sensitive information. Never let a project go live with debug on

ALLOWERD-HOSTS - Not applied while debug is on or testing. Once you move site to production and debug is off you will have to add our domain host and allow it to serve your django site

INSTALLED_APPS - This is where we add applications that are active for the site.

Django.contrib.admin: Administration site
Django.contrib.auth: authentication framework
Django.contrib.contenttypes: handles content types
Django.contrib.sessions: session framework
Django.contrib.messages: messaging framework
Django.contrib.staticfiles: manages statics such as CSS files, JS files, and images

MIDDLEWARE - list that contains middleware to be executed

ROOT_URLCONF - Indicates the python module where the root URL patterns of application are defined

DATABASES - dictionary that contains settings for all databases used in the project. There always must be a default. Default uses SQLite3

LANGUAGE_CODE - defines the default language for the django site

USE_TZ - tells django to activate.deactivate timezone support. Setting is automatically true

Admin.py

Saturday, March 1, 2025 11:46 AM

from .models import (class name)

admin.site.register(class name)

Customize the admin page by creating a class that inherits from **admin.ModelAdmin**

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'slug', 'author', 'publish', 'status']
    list_filter = ['status', 'created', 'publish', 'author']
    search_fields = ['title', 'body']
    prepopulated_fields = {'slug': ('title',)}
    raw_id_fields = ['author']
    date_hierarchy = 'publish'
    ordering = ['status', 'publish']
```

list_display - fields displayed
list_filter - allows searching and filtering of posts
search_fields - creates a search bar

Creating Objects

Saturday, March 1, 2025 12:06 PM