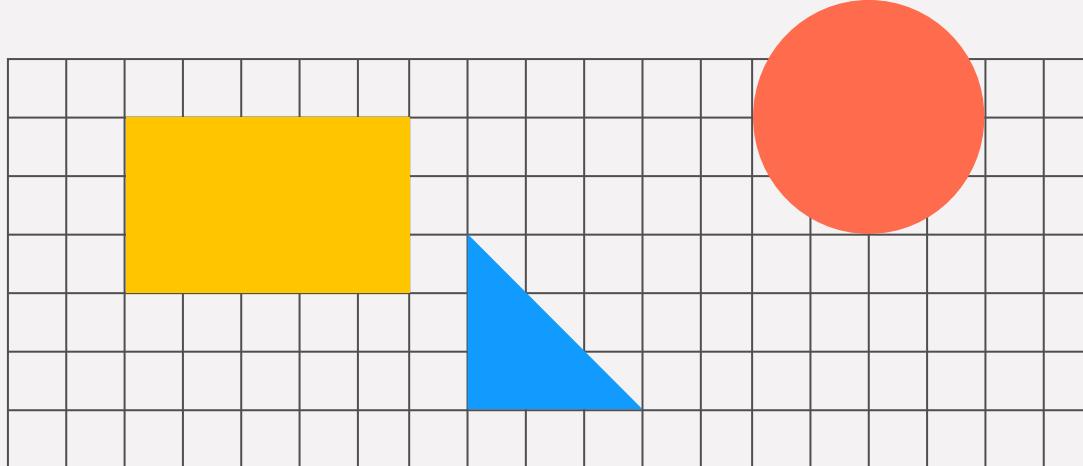
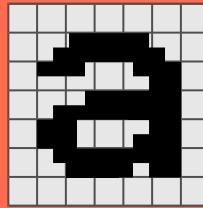


Django at a Glance

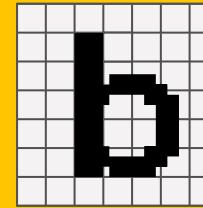
Ali Abrishami



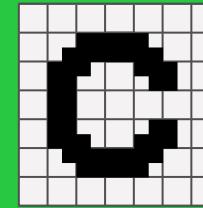
In This Lecture You Will...



Learn what
Django is and
what does it solve



Learn how to
initiate a Django
project



Start making your
first website!

Here Comes Django

Introduction

- Developed in a fast-paced newsroom environment.
- Designed to make common web development tasks **fast** and **easy**.
- Comes with an object-relational mapper in which you describe your database layout in Python code.



django



Django was created by two people named Adrian Holovaty and Simon Willison back in 2003. It grew from a practical need: World Online, a newspaper web operation responsible for building intensive web applications on journalism deadlines.

The Django web framework was named after Django Reinhardt, the legendary jazz guitarist, as a tribute to his creativity, innovation, and ability to craft brilliance despite constraints!

Fun Fact!

Fun Fact!

Django's Philosophy

DRY

Quick Development

Loose coupling

Consistency

Explicit Is Better Than
Implicit

Less Code

Real-world Applications That Use Django



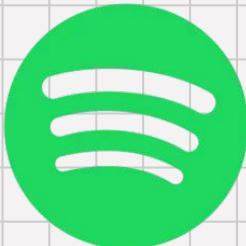
YouTube



Dropbox



Instagram



Spotify®



Pinterest

Setting up Django

Introduction

- Installing Django is quite simple.
- Create a virtual environment.
- install `django` with `pip`
- Get to work!

VEnv

- A virtual environment is needed to isolate your current development workspace from other parts of your system (especially your other python packages and projects).
- it's typically named `venv`, `.venv`, or `env`.
- Python provides a package to create virtual environments!

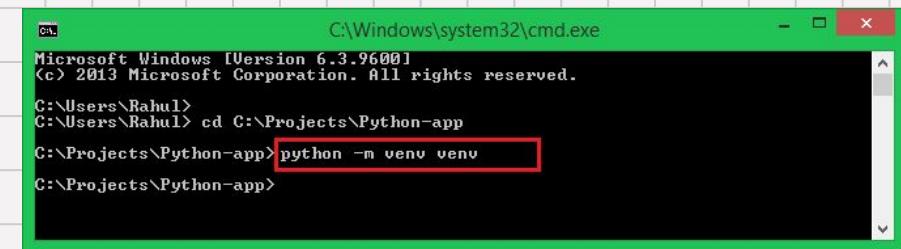
Setting up a VEnv

MacOS / Linux



```
[manish@manish-laptop tmp.TIWmmn7ij3]$ python -m venv .venv
[manish@manish-laptop tmp.TIWmmn7ij3]$ source .venv/bin/activate
(.venv) [manish@manish-laptop tmp.TIWmmn7ij3]$
```

Windows



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Rahul>
C:\Users\Rahul> cd C:\Projects\Python-app
C:\Projects\Python-app> python -m venv venv
C:\Projects\Python-app>
```

```
PS C:\Users\Gaurav Gahlot\Desktop\project-36> .\venv\Scripts\activate
(venv) PS C:\Users\Gaurav Gahlot\Desktop\project-36> python --version
Python 3.6.6
```

Fun Fact!

You can also do this using `virtualenv`!

Install The Django Itself

When in a virtual environment, execute:

```
floaterm(1/1) └── mani pip install django
Collecting django
  Using cached django-5.2.3-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref≥3.8.1 (from django)
  Using cached asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse≥0.3.1 (from django)
  Using cached sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Using cached django-5.2.3-py3-none-any.whl (8.3 MB)
Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
Using cached sqlparse-0.5.3-py3-none-any.whl (44 kB)
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.2.3 sqlparse-0.5.3
```

Best Practice: It's best to keep the modules you need in a separate text file `requirements.txt`.

Create Your First Ever Django Project!

In order to create a project named `firstblog`, you must execute the following command:

```
floaterm(1/1)
  mani  django-admin startproject firstblog
```

FirstBlog

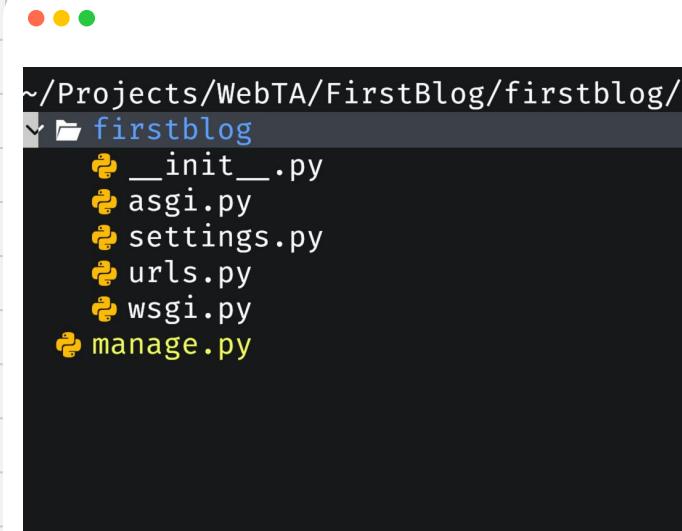


What is `django-admin`?

- CLI tool for managing Django projects.
- Create apps, run servers, and apply migrations.
- Works outside the project code (before setup).
- Automates common setup and admin tasks (more on this later!).

Django's Project Structure

- This is the top-level directory of your Django project. It acts as a **container** for your application(s) and project-level configurations.
- As you can see, there are two main parts in the root directory:
 - `firstblog` (a package named after our project)
 - `manage.py`



A screenshot of a Mac OS X desktop showing a file browser window. The window title is ~/Projects/WebTA/FirstBlog/firstblog/. The contents of the folder are:

- __init__.py
- asgi.py
- settings.py
- urls.py
- wsgi.py
- manage.py

The Project Package



ASGI/WSGI

Interface specifications for Python web servers and applications



Settings

Contains all the configuration for your Django project.



URLs

Acts as the central URL dispatcher for your Django project.

The manage.py file

- Project-specific Django CLI tool.
- Wraps django-admin with project settings.
- Used for migrations, running server, shell, etc.
- Must be run from your project root.

```
floaterm(1/1) mani python manage.py help
Type 'manage.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

[auth]
    changepassword
    createsuperuser

[contenttypes]
    remove_stale_contenttypes

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    makemigrations
    migrate
    optimizemigration
    sendtestemail
    shell
```

Django runserver

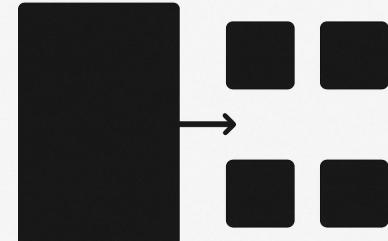


- When executed in `manage.py`, it runs a built-in development web server.
- `runserver` watches your files and automatically reloads your project if something changes.
- It serves your project at a local address for the sake of testing.
- Remember not to use `runserver` in production!

Breaking Our Big App... Into More Apps

In Django, a project is just the container — the real building blocks are apps.

- In Django, apps are the building blocks of your project (the big app).
- Split features into isolated, manageable parts that can be used across different projects.
- Grow your project without turning it into a mess.
- Keep related code (models, views, URLs) together.



Class Activity Time!

What apps do we need for a blog?

Create The First App!

```
floaterm(1/1)---  
  mani  python manage.py startapp blog
```

firstblog

```
  mani
```

firstblog

Our first app is **blog**

It will handle the blog-related functionality in our project.

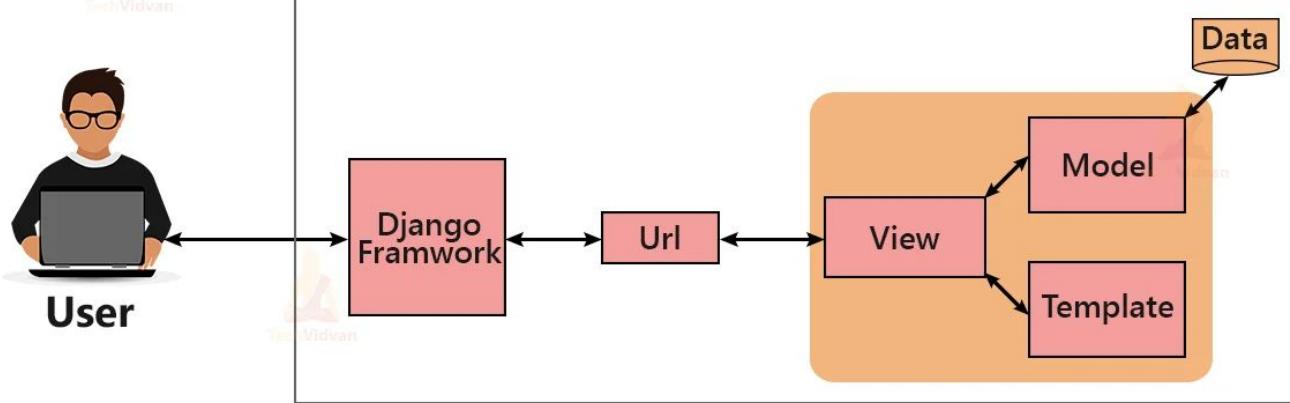
Django's App Structure

- **models**: defines the data structure of your application using Django's ORM (more on that later).
- **views**: contains the logic for handling HTTP requests and returning responses.
- **tests**: where you write unit tests and integration tests for your application's code.
- **admin**: register your models with Django's automatic admin interface.
- **apps**: defines the application configuration for the Django app.

```
~/Projects/WebTA/Fi
└── blog
    ├── migrations
    │   ├── __init__.py
    │   ├── admin.py
    │   ├── apps.py
    │   ├── models.py
    │   ├── tests.py
    │   └── views.py
    └── firstblog
        └── manage.py
```

Django Software Arch.

Control Flow Of MVT



Source: Aritra Pain -
LinkedIn

Implement The Model



firstblog/blog/models.py

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    text = models.TextField()
    created_date = models.DateTimeField(auto_now=True)
    published_date = models.DateTimeField(auto_now_add=True)
```

Django's ORM



- Normally, you have to write SQL to interact with the database.
- Django comes with a powerful feature called the Object-Relational Mapper (**ORM** for short).
- It abstracts SQL into Python code. Which means you interact with the database using Python classes and methods instead of raw queries.
- Each model is a Python class where attributes map to table columns.
- TL;DR: Create, read, update, and delete records without writing a single line of SQL.

Class Activity Time!

Why is it **safer** to use ORM?

Can we migrate from MySQL to SQLite? What do we have to do?

Can it be more efficient to write SQL yourself?

Common Model Fields



Field	Usage
CharField	Short text (e.g. name, text, title)
TextField	Long text (e.g. content, description, biography)
IntegerField	Whole numbers
FloatField	Decimal numbers with less precision
DecimalField	Decimal numbers with more precision
BooleanField	True or false values

Tweak The Settings



- For Django to recognize and utilize your app, you have to add your app to the `INSTALLED_APPS` list.
- You have to specify the database that your application is going to utilize in order for it to work properly. This can be done through changing the `DATABASES` list.
- SQLite works for now. We will learn how to use PostgreSQL, MariaDB, MySQL, etc., later.

Migrations



- Migrations **track changes** to your models over time (just like a VCS).
- They generate **Python files** that describe these changes.
- Run `makemigrations` to **create** migration files.
- Run `migrate` to **apply** them to the database.
- They handle **schema changes** like adding or removing fields.
- Each app has its **own migration history**.
- You must make **new migrations** whenever you **change** a model.

Make Migrations, Migrate!

```
floaterm(1/1)
  ✓ mani python manage.py makemigrations
    Migrations for 'blog':
      blog/migrations/0001_initial.py
        + Create model Post

  ✓ mani python manage.py migrate
    Operations to perform:
      Apply all migrations: admin, auth, blog, contenttypes, sessions
    Running migrations:
      Applying contenttypes.0001_initial ... OK
      Applying auth.0001_initial ... OK
      Applying admin.0001_initial ... OK
      Applying admin.0002_logentry_remove_auto_add ... OK
      Applying admin.0003_logentry_add_action_flag_choices ... OK
      Applying contenttypes.0002_remove_content_type_name ... OK
      Applying auth.0002_alter_permission_name_max_length ... OK
      Applying auth.0003_alter_user_email_max_length ... OK
      Applying auth.0004_alter_user_username_opts ... OK
      Applying auth.0005_alter_user_last_login_null ... OK
```

firstblog

firstblog

Create a Post (Shell)

```
floaterm(1/1)-
mani python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.7 (main, Aug 15 2025, 12:34:02) [GCC 15.2.1 20250813] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> Post.objects.create(title="Shell", text="i created this in django shell")
<Post: Post object (2)>
>>> █
```

firstblog

Views



- A view processes an HTTP request and returns an HTTP response.
- It connects URL patterns to application logic.
- Can use templates to render HTML, or return raw data (e.g., JSON) directly.
- Comes in two main styles: Function-Based Views and Class-Based Views.
- Should delegate heavy logic to services/models instead of doing everything itself.

Sample Views Code



firstblog/blog/views.py

```
from django.shortcuts import get_object_or_404, render

from .models import Post


def post_list(request):
    posts = Post.objects.order_by("-published_date")
    return render(request, "blog/post_list.html", {"posts": posts})


def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, "blog/post_detail.html", {"post": post})
```

Templates



- Templates define the presentation layer, structure, and layout separate from business logic.
- Use Django Template Language (DTL) with tags, filters, and variables to inject dynamic data.
- Typically rendered via views using `render(request, template, context)`.
- Support inheritance (`base.html`) to avoid duplication and enforce consistent layout.

Sample Template I



Projects/WebTA/FirstBlog/firstblog/blog/templates/blog/post_list.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Posts</title>
</head>
<body>
    <h1>Posts</h1>

    {% for post in posts %}
        <div>
            <h2><a href="{{ post.pk }}/">{{ post.title }}</a></h2>
            <small>By {{ post.author }} - {{ post.published_date }}</small>
            <p>{{ post.text|truncatewords:20 }}</p>
        </div>
    {% empty %}
        <p>No posts yet.</p>
    {% endfor %}

</body>
</html>
```

Sample Template II



Projects/WebTA/FirstBlog/firstblog/blog/templates/blog/post_detail.html

```
<html>
  <head>
    <title>{{ post.title }}</title>
  </head>
  <body>
    <h1>{{ post.title }}</h1>
    <small>By {{ post.author }} - {{ post.published_date }}</small>

    <p>{{ post.text }}</p>

    <a href="/">Back</a>
  </body>
</html>
```

URLs



- URL configs map incoming paths to view functions or class-based views using urlpatterns.
- Defined per app (`urls.py`) and composed hierarchically via `include()` for modularity.
- Patterns use path converters (`<int:id>`, `<slug:slug>`) to extract params from the URL.
- The router resolves from top to bottom, order matters; ambiguous patterns are a design flaw.
- URLs should describe resources cleanly; avoid encoding logic or state into the path itself.

Include in Project's URLs



firstblog/firstblog/urls.py

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

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

Assign URLs



firstblog/blog/urls.py

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

```
urlpatterns = [  
    path("", views.post_list, name="post_list"),  
    path("<int:pk>/", views.post_detail, name="post_detail"),  
]
```

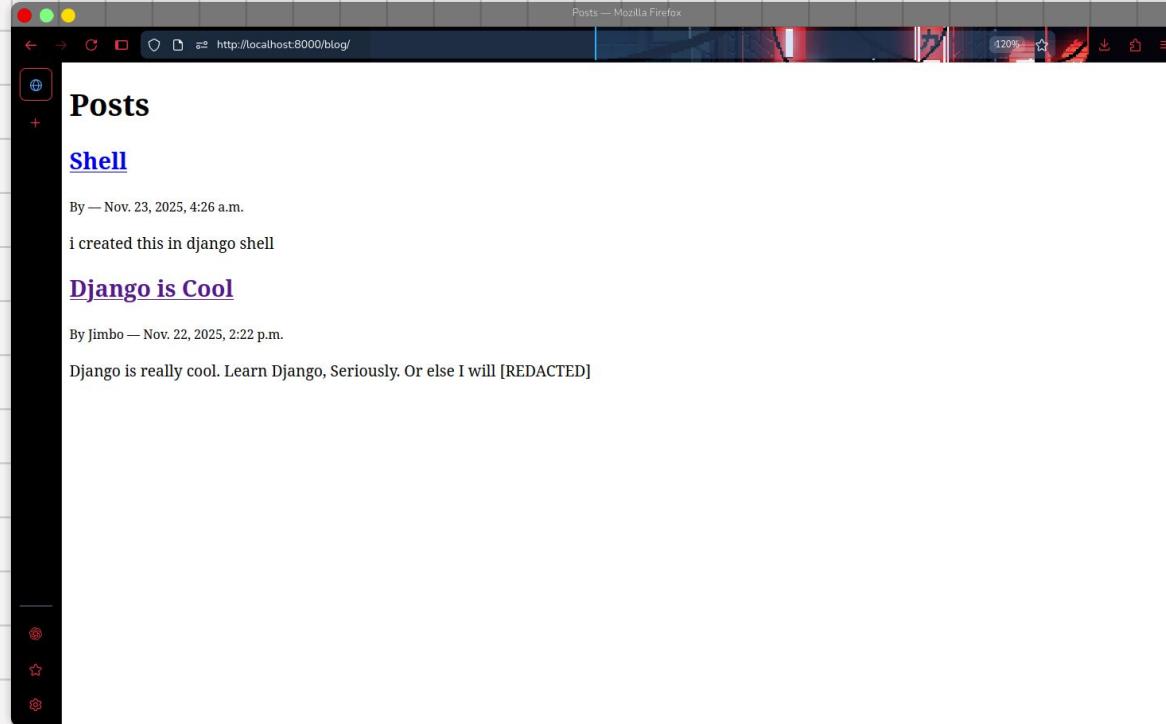
Run Luke, Run!

```
floaterm(1/1)
man python manage.py runserver
Watching for file changes with StatReloader
Performing system checks ...

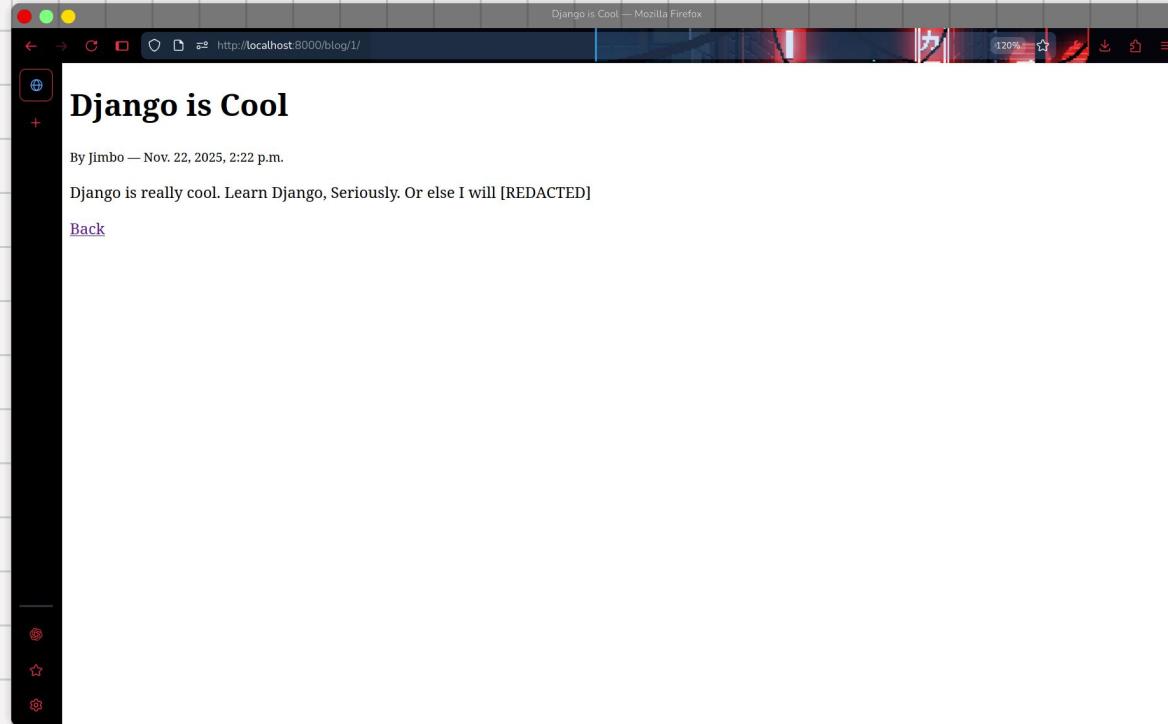
System check identified no issues (0 silenced).
November 22, 2025 - 14:15:34
Django version 5.2.1, using settings 'firstblog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting.
Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

And The Results!



And The Results! (cont.)



Django Admin



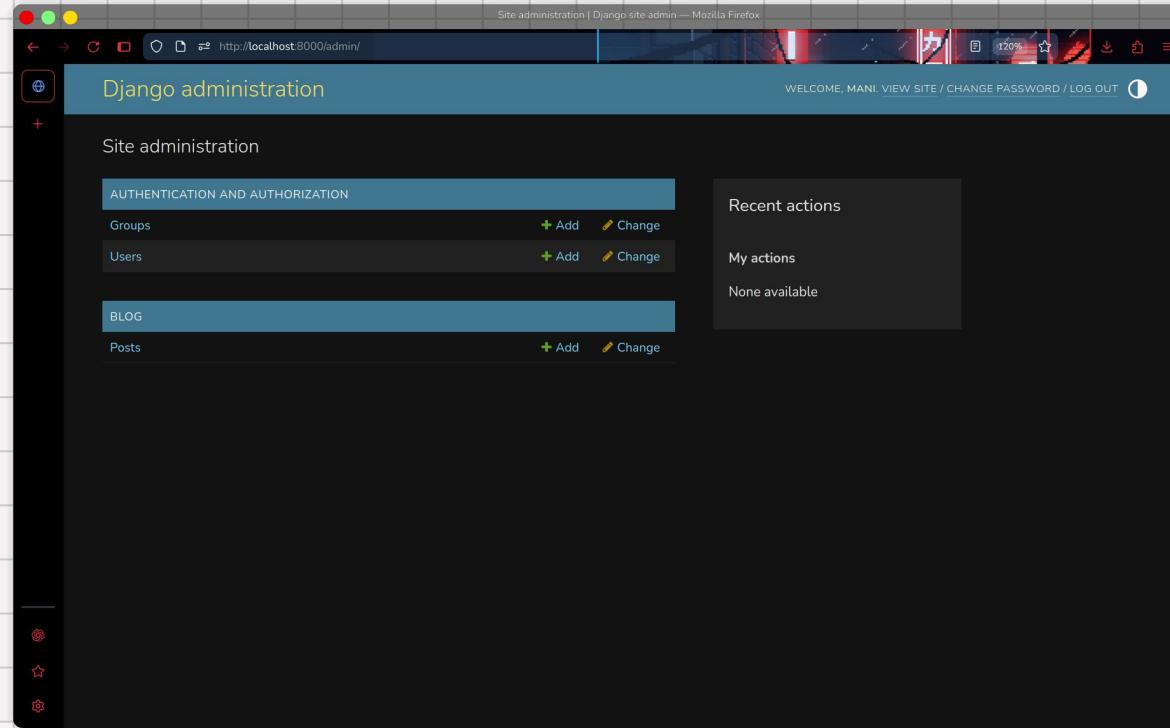
- Built-in admin interface generated from models, allowing CRUD operations without writing views or templates.
- Highly configurable via ModelAdmin: list filters, search fields, custom actions, fieldsets, readonly fields.
- Authentication and permissions integrated with Django's auth system; access intended for staff, not public users.
- Not a substitute for a frontend or a user-facing dashboard; meant for internal ops, debugging, and data inspection.
- Let's register our `Post` model by writing `admin.site.register(Post)` in blog's `admin.py`.

Create an Admin User

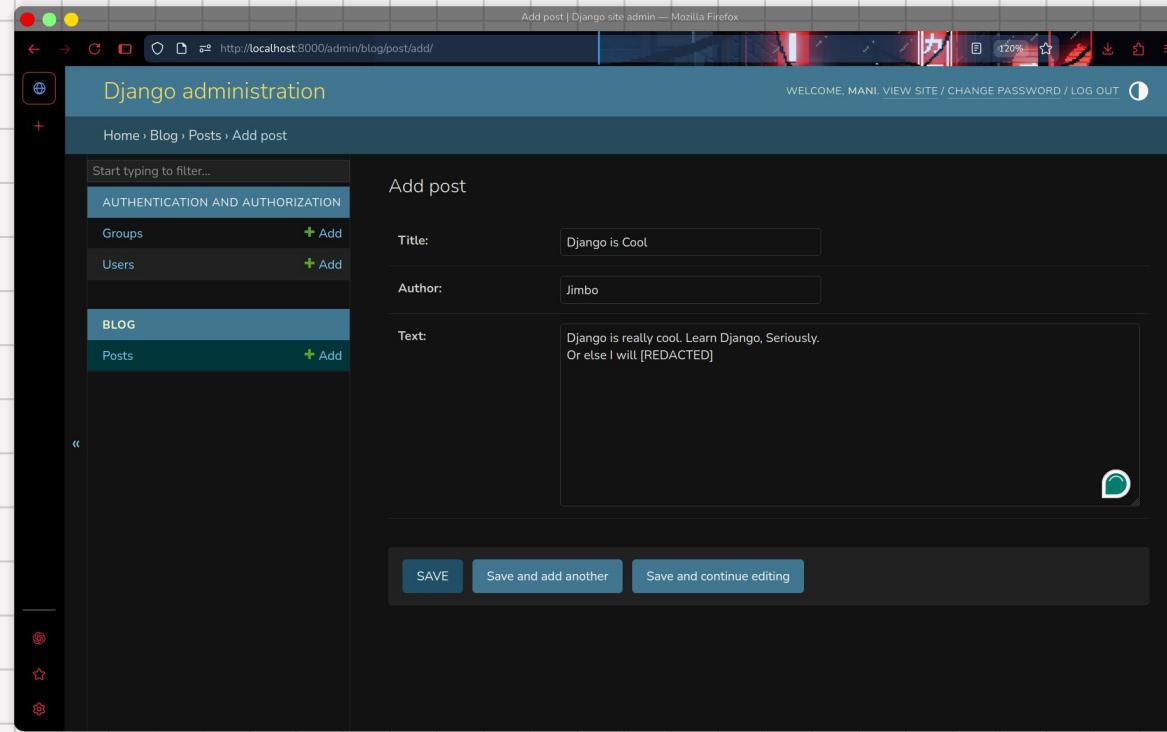
```
floaterm(1/1) mani python manage.py createsuperuser
Username (leave blank to use 'mani'): mani
Email address: mani.ebra@gmail.com
Password:
Password (again):
Superuser created successfully.
```

firstblog

Admin Panel



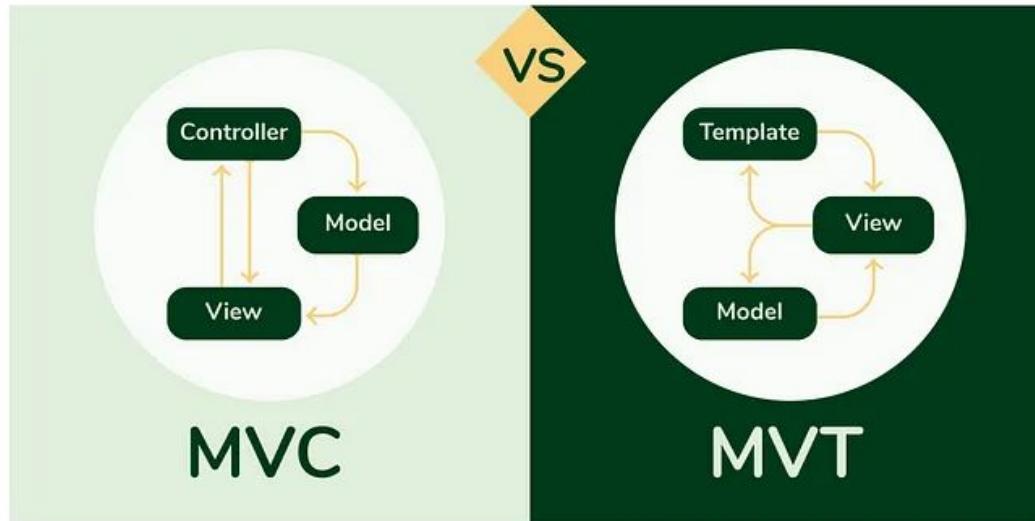
Create a Post



The Software Architecture

- Django uses an MVC-like architecture named **MVT**.
- Models define the **data schema** and **business logic**, interacting directly with the database.
- Views act like the **controller** in MVC. They process requests and return responses (often using templates).
- Templates handle the **presentation** layer. They render dynamic HTML using context data.

The Software Architecture (cont.)



Read this if you're interested:



Source: Greatness Marshal - Medium