
🌱 Starting a Django App — Complete Explanation

🧩 Step 1: Create a New Django App

Run this command inside your Django project directory:

```
django-admin startapp sample_app
```

This creates a new folder named `sample_app` with this structure:

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

🧠 Step 2: Understand the MVT Architecture

Django follows the **MVT pattern** — **Model, View, Template**.

Component	Role	File
Model (M)	Defines the structure of your database tables (what data is stored and how).	`models.py`

View (V)	Contains the logic that handles requests and responses.	`views.py`
Template (T)	Defines how the data is displayed to the user (HTML pages).	Inside a `templates/` folder

Flow:

1. User sends a request (e.g., visiting a URL).
2. Django routes it to the corresponding **View**.
3. View fetches data from the **Model** (if needed).
4. View renders the data into a **Template** and returns a **Response** to the browser.

Understanding the Structure:

🧩 1. `__init__.py`

- Marks this folder as a **Python package**.
- It's usually empty.
- It tells Python: "you can import from this folder."

Example:

```
from sample_app import views
```

works only because `__init__.py` exists.

🧠 2. `admin.py`

- Used to **register your models** with the Django **Admin interface**.
- After registering, you can manage (add/edit/delete) data from the `/admin` panel in your browser.

Example:

```
from django.contrib import admin
from .models import Student

admin.site.register(Student)
```

Now the `Student` model will appear in the Django admin dashboard.

⚙️ 3. `apps.py`

- Contains the **configuration class** for your app.
- Django automatically creates it when you run `startapp`.
- It helps Django recognize your app and manage settings related to it.

Example (auto-generated):

```
from django.apps import AppConfig

class SampleAppConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'sample_app'
```

This is referenced automatically when you add `'sample_app'` in `INSTALLED_APPS` in `settings.py`.

📦 4. `migrations/` folder

- Keeps track of changes you make to your **models** (database schema).
- Each migration file is like a “version control” file for your database.

Example files:

```
migrations/
    __init__.py
    0001_initial.py
    0002_add_field.py
```

Generated using:

```
python manage.py makemigrations
python manage.py migrate
```

🧱 5. `models.py`

- Defines your **database structure** — what tables and fields exist.
- Each model is a Python class that Django turns into a database table.

Example:

```
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    email = models.EmailField()

    def __str__(self):
        return self.name
```

After creating this model, you’d run migrations to apply it to your database.

🌐 6. `views.py`

- Contains the **logic** for handling user requests and returning responses.
- Each view is linked to a URL and can return:
    - Plain text (`HttpResponse`)
    - HTML templates (`render`)
    - JSON data (`JsonResponse`)

Example:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Welcome to the Home Page!")
```

🧪 7. `tests.py`

- Used for **unit testing** — to ensure your app works as expected.
- Django uses Python's built-in `unittest` framework.

Example:

```
from django.test import TestCase
from .models import Student

class StudentModelTest(TestCase):
    def test_create_student(self):
        student = Student.objects.create(name="Ajay", age=22)
        self.assertEqual(student.name, "Ajay")
```

You can run tests using:

```
python manage.py test
```

🔧 (Optional, when you create them)

You might later add:

- **`urls.py`** — to define your app's routes.
- **`templates/` folder** — to store HTML files.
- **`static/` folder** — to store CSS, JS, images, etc.

Example:

```
sample_app/
│
├── templates/
│   └── sample_app/
│       └── home.html
├── static/
│   └── sample_app/
│       └── style.css
```

⚙️ Step 3: Views in Django

Views are Python functions (or classes) that take an HTTP request and return an HTTP response.

There are 2 main types of views:

1. **Function-Based Views (FBV)**
2. **Class-Based Views (CBV)**

Let's focus on **Function-Based Views**, since they are simpler to understand first.

🔹 Function-Based View — Flow

1. Request comes in through the URL.
2. Django finds the correct function in `views.py`.
3. That function returns a response (HTML, text, or JSON).

🔸 Example: `views.py`

```
from django.http import HttpResponse, JsonResponse
from django.shortcuts import render

# Simple text response
def home(request):
    return HttpResponse("Welcome to my website!")

# Rendering HTML template
def about(request):
    return render(request, 'about.html', {'title': 'About Us'})

# Sending JSON response
def api_data(request):
    data = {"message": "This is JSON data", "status": "success"}
    return JsonResponse(data)
```

🧩 Explanation:

- `HttpResponse()` → Sends plain text or HTML directly.
- `render(request, 'template.html', context)` → Loads an HTML template and fills it with data.
- `JsonResponse()` → Sends structured data (JSON), often used for APIs.

🗺️ Step 4: Connect Views to URLs

Every Django project has a **main `urls.py`** (inside the project folder).
You can define all your routes there, or you can add an `urls.py` inside your app.

Option1️⃣— (Simpler) Define routes in project-level `urls.py`

myproject/urls.py

```
from django.contrib import admin
from django.urls import path
from sample_app import views   # Import views directly from app

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('api/', views.api_data, name='api_data'),
]
```

✅ No need for an `urls.py` inside your app.
This works fine for small projects.

Option2️⃣— (Recommended for larger projects)

Create an `urls.py` file inside your app.

sample_app/urls.py

```
from django.urls import path
from . import views   # ✅ Correct way
# import views ❌ will not work because it's not a Python package
import

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
```

```
    path('api/', views.api_data, name='api_data'),
]
```

Then, include it inside the project's main `urls.py`:

myproject/urls.py

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

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

⚙️ Step 5: Link the App to Project Settings

You must tell Django that your new app exists by adding it to **`INSTALLED_APPS`** in `settings.py`.

myproject/settings.py

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

    # Add your app here
    'sample_app',
]
```

🧱 Step 6: Templates (HTML Files)

You can place your HTML files anywhere, but Django needs to know where to find them.

✅ Recommended Setup:

Create a folder named `templates` at the project root.

```
myproject/
├── templates/
│   └── about.html
├── sample_app/
│   └── views.py
└── manage.py
```

Then, update the template settings in `settings.py`:

```
import os

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  # ✅ Add this
        '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',
            ],
        },
    },
]
```

Example Template: `templates/about.html`

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>{{ title }}</title>
</head>
<body>
    <h1>This is the About Page</h1>
    <p>Welcome to our website!</p>
</body>
</html>
```

In `views.py`:

```
def about(request):
    return render(request, 'about.html', {'title': 'About Us'})
```

🧪 Step 7: Test Using Postman

Postman is a tool used to **test APIs** built with Django (especially when you use `JsonResponse` or Django REST Framework).

How to use it:

Run your server:

```
python manage.py runserver
```

1.
2. Open Postman.

Make a request to:

```
http://127.0.0.1:8000/api/
```

3.

You'll receive:

```
{
  "message": "This is JSON data",
  "status": "success"
}
```

4.

This helps you verify your API responses without using a browser.