Python & Django

Table of Contents

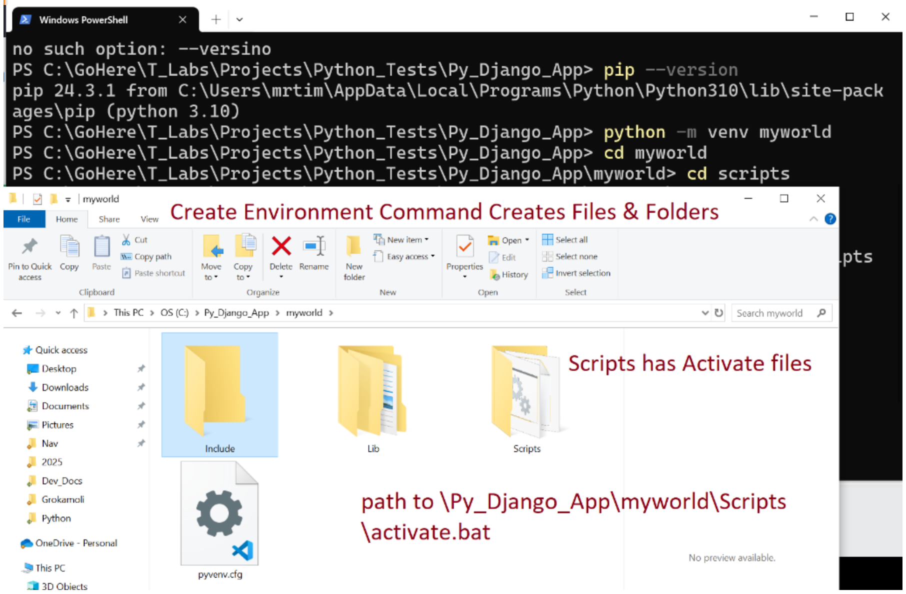
W3 Schools Django App Notes – Virtual Environments........ 3
Workflow in Django 8
👍Let’s make your myworld venv auto-activate in VS Code........ 10
Add myworld to .gitignore 11
A key piece of how Django’s **development server** works. 12
Simple Visual “Map” 13
Fancier “full stack view” including: 14

W3 Schools Django App Notes – Virtual Environments

1. S

```
Hey Chat Happy Labor Day
- In path \Projects\Py_Django_App

If I run
python -m venv myworld
- is myworld the top level folder in the environment

\Py_Django_App
myworld
  Include
  Lib
  Scripts
  .gitignore
  pyvenv.cfg

Or Py_Django_App

– you don’t run it with python.

The activation script is a batch file (activate.bat)
the command line.
path to \Py_Django_App\myworld\Scripts\activate.bat
```

2.



```
no such option: --versino
PS C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App> pip --version
pip 24.3.1 from C:\Users\mrtim\AppData\Local\Programs\Python\Python310\lib\site-pack
ages\pip (python 3.10)
PS C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App> python -m venv myworld
PS C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App> cd myworld
PS C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App\myworld> cd scripts
```

Create Environment Command Creates Files & Folders

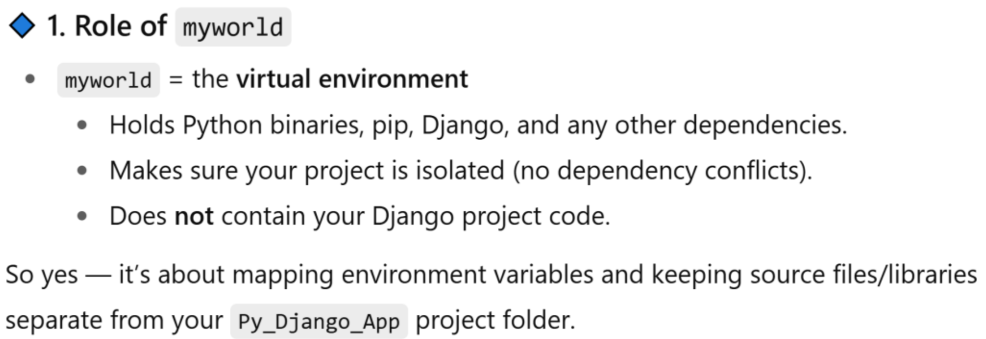
Scripts has Activate files

path to \Py_Django_App\myworld\Scripts \activate.bat

3.
4. Activation = no file changes, just PATH + prompt.
5. Installation = adds packages only inside myworld\Lib\site-packages.

◆ How the flow works

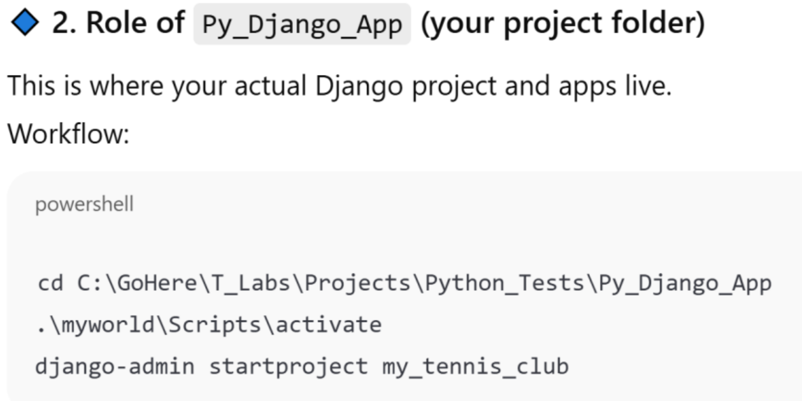
1. **VS Code terminal** → you activate `myworld` so Python/Django commands point to the venv.
2. **manage.py** runs commands like `runserver` or `migrate`.
3. **urls.py** maps URLs to specific app views.
4. **Apps** contain `views.py` for logic, `models.py` for data, and optional `templates/static` for front-end.
5. **Django dev server** listens on `127.0.0.1:8000` and sends responses.
6. **Browser** receives HTML/CSS/JS and displays pages.

6. 

◆ 1. Role of `myworld`

- `myworld` = the **virtual environment**
  - Holds Python binaries, pip, Django, and any other dependencies.
  - Makes sure your project is isolated (no dependency conflicts).
  - Does **not** contain your Django project code.

So yes — it's about mapping environment variables and keeping source files/libraries separate from your `Py_Django_App` project folder.

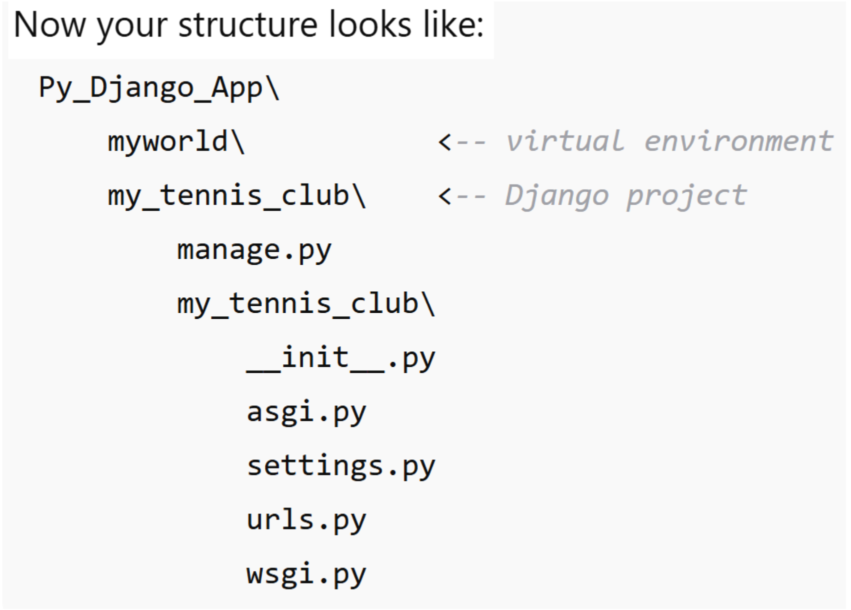
7. 

◆ 2. Role of `Py_Django_App` (your project folder)

This is where your actual Django project and apps live.
Workflow:

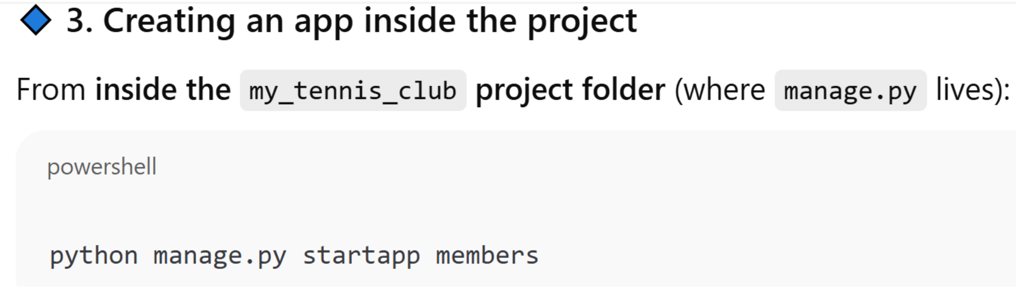
powershell

```
cd C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App
.\myworld\Scripts\activate
django-admin startproject my_tennis_club
```

8. 

Now your structure looks like:

```
Py_Django_App\
    myworld\            <-- virtual environment
    my_tennis_club\     <-- Django project
        manage.py
        my_tennis_club\
            __init__.py
            asgi.py
            settings.py
            urls.py
            wsgi.py
```

9. 

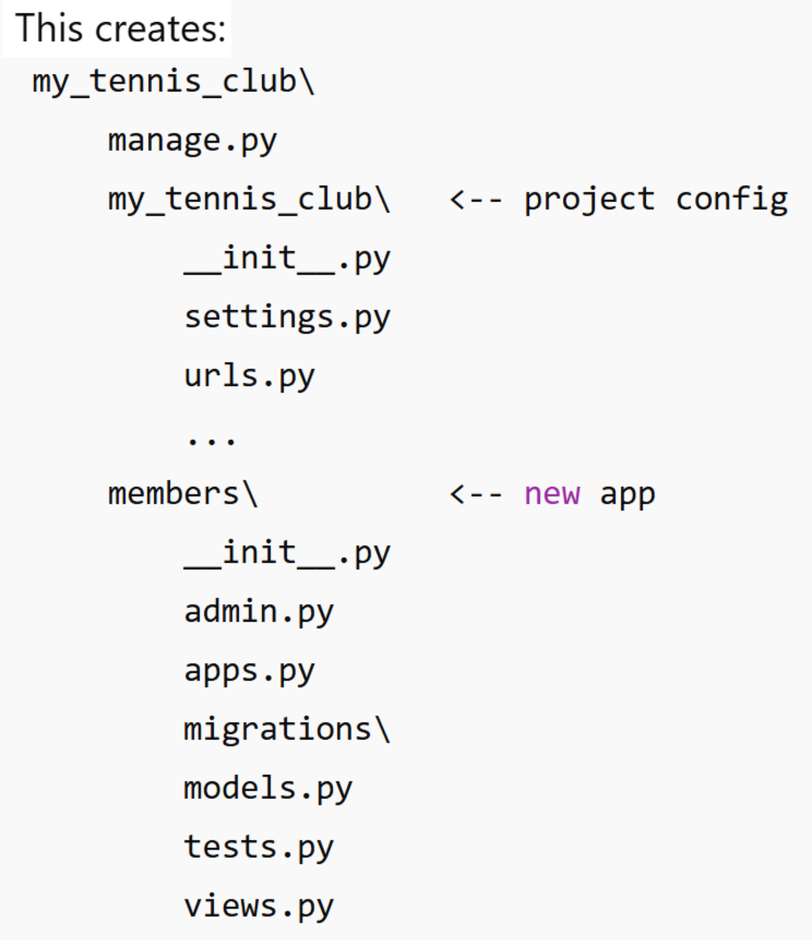
◆ 3. Creating an app inside the project

From **inside the** `my_tennis_club` **project folder** (where `manage.py` lives):

powershell

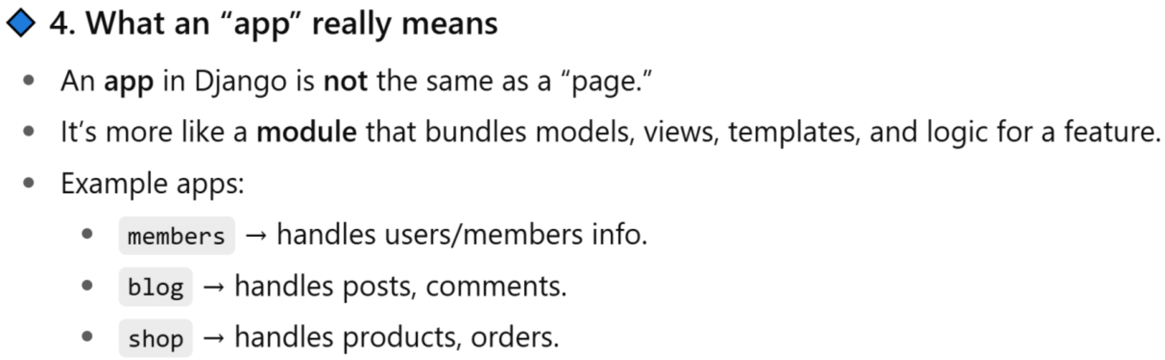
```
python manage.py startapp members
```

10.



This creates:

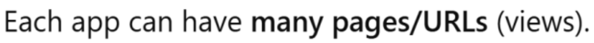
```
my_tennis_club\
    manage.py
    my_tennis_club\   <-- project config
        __init__.py
        settings.py
        urls.py
        ...
    members\          <-- new app
        __init__.py
        admin.py
        apps.py
        migrations\
        models.py
        tests.py
        views.py
```



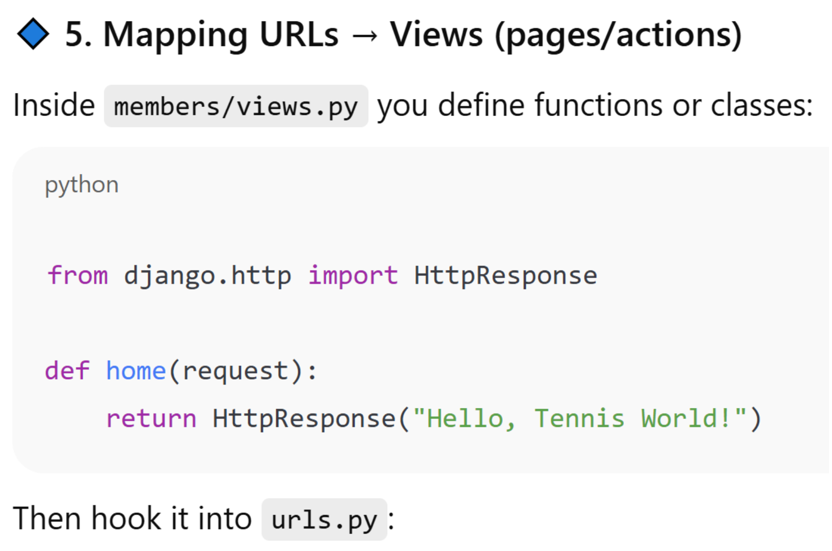
◆ 4. What an “app” really means

- An **app** in Django is **not** the same as a “page.”
- It’s more like a **module** that bundles models, views, templates, and logic for a feature.
- Example apps:
  - `members` → handles users/members info.
  - `blog` → handles posts, comments.
  - `shop` → handles products, orders.

11.



Each app can have **many pages/URLs** (views).

12. 

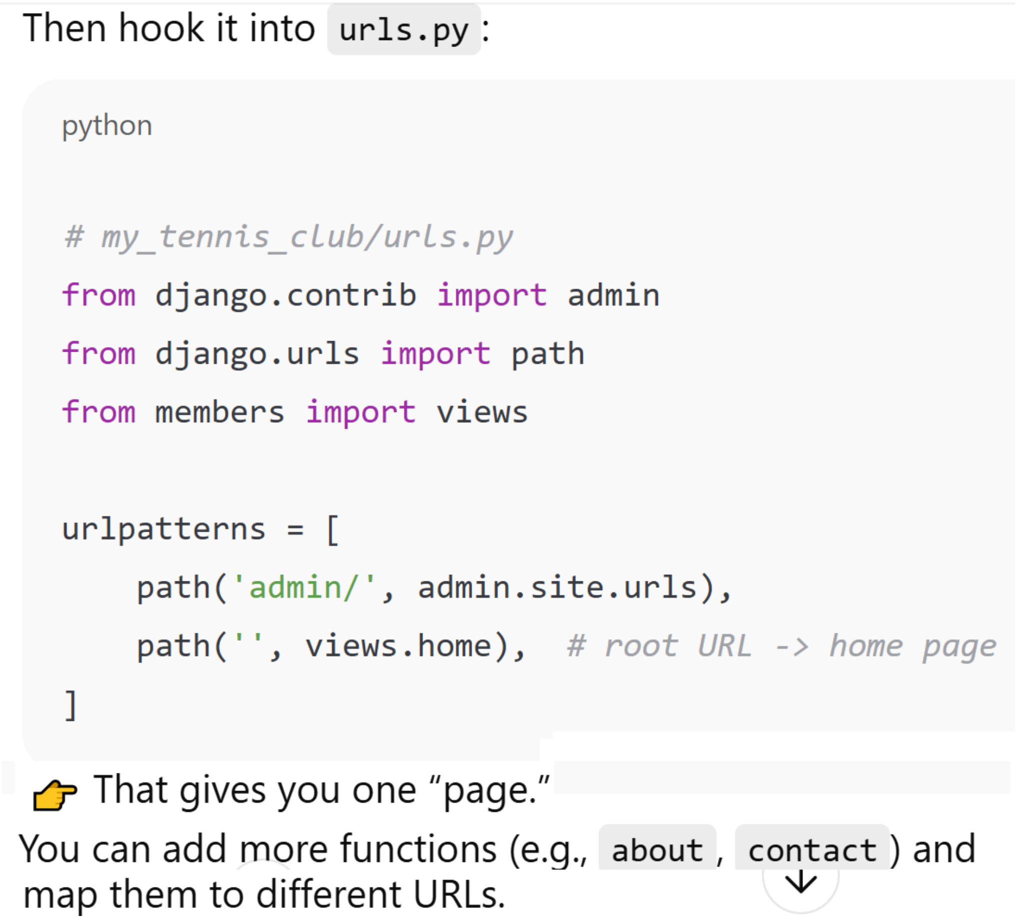
🔷 5. Mapping URLs → Views (pages/actions)

Inside `members/views.py` you define functions or classes:

python

```python
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, Tennis World!")
```

Then hook it into `urls.py`:

13. 

Then hook it into `urls.py`:

python

```python
# my_tennis_club/urls.py
from django.contrib import admin
from django.urls import path
from members import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),  # root URL -> home page
]
```

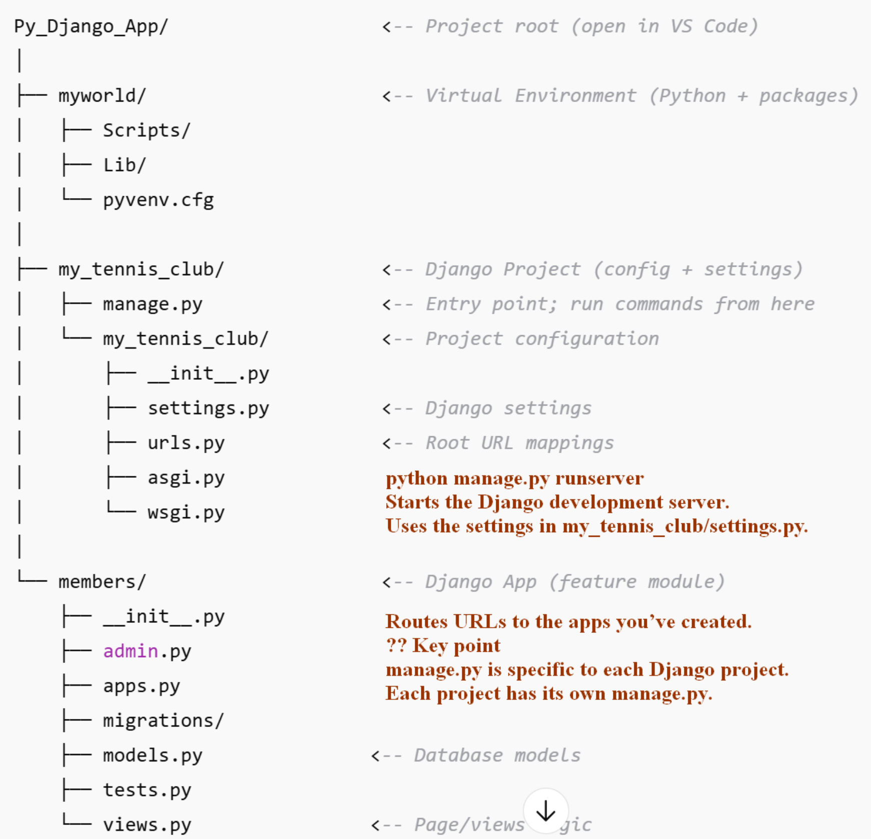
👉 That gives you one "page."

You can add more functions (e.g., `about`, `contact`) and map them to different URLs.

✅ So:

- `myworld` = Python/Django environment (hidden support layer).
- `my_tennis_club` = Django project container (settings + global config).
- `members` = Django app (a feature module, can have many pages/actions).

14.



```
Py_Django_App/                  <-- Project root (open in VS Code)
│
├── myworld/                    <-- Virtual Environment (Python + packages)
│   ├── Scripts/
│   ├── Lib/
│   └── pyvenv.cfg
│
├── my_tennis_club/             <-- Django Project (config + settings)
│   ├── manage.py               <-- Entry point; run commands from here
│   └── my_tennis_club/         <-- Project configuration
│       ├── __init__.py
│       ├── settings.py         <-- Django settings
│       ├── urls.py             <-- Root URL mappings
│       ├── asgi.py
│       └── wsgi.py
│
└── members/                    <-- Django App (feature module)
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations/
    ├── models.py              <-- Database models
    ├── tests.py
    └── views.py               <-- Page/views   gic
```

python manage.py runserver
Starts the Django development server.
Uses the settings in my_tennis_club/settings.py.

Routes URLs to the apps you’ve created.
?? Key point
manage.py is specific to each Django project.
Each project has its own manage.py.

15.

🔷 How `python manage.py runserver` fits in

1. You **activate** `myworld` → Python + Django commands point to the venv.
2. You **run** `manage.py` inside `my_tennis_club` → it reads `settings.py`.
3. Django starts the **development server**, using:
   - URLs from `my_tennis_club/urls.py`
   - Apps (like `members`) to handle pages/views

Flow of a request:

```
Browser → Server (runserver) → urls.py → app views → templates → response
```

- Apps can have multiple URLs (pages) inside a single app folder.
- `manage.py` orchestrates all project-level commands (runserver, migrate, createsuperuser, etc.).

16.

Workflow in Django

◆ 1. Workflow in Django

1. **Create your project** (if not already):

```powershell
django-admin startproject my_tennis_club
```

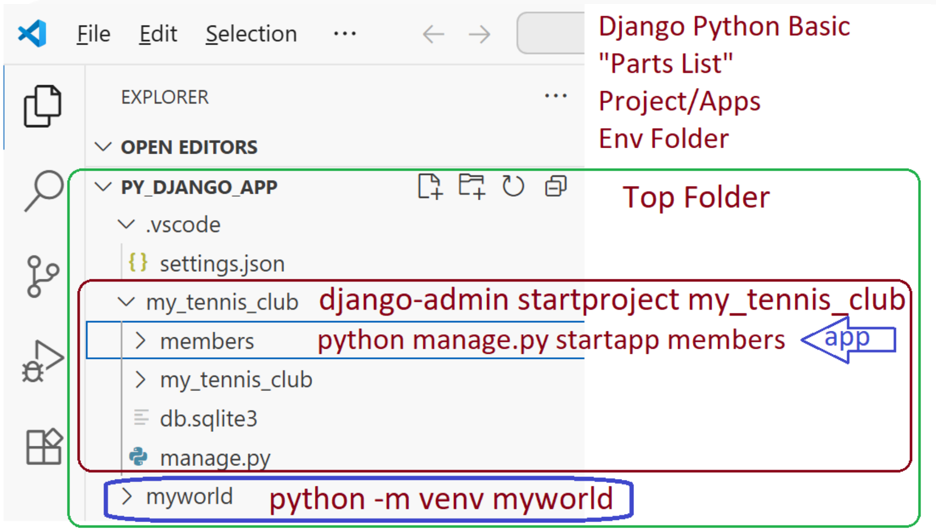
2. **Add apps** as needed:

```powershell
cd my_tennis_club
python manage.py startapp members
```

3. **Edit your code** in VS Code:

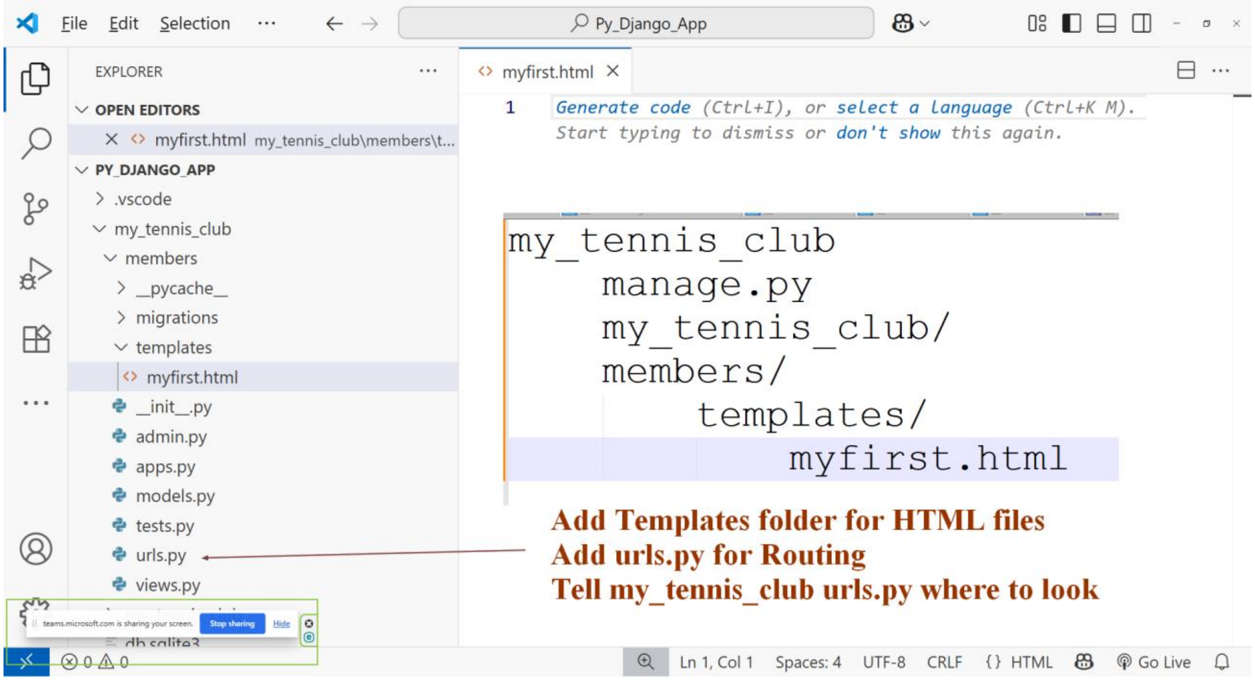
- `views.py` → change logic or return different content
- `templates/` → HTML changes
- `urls.py` → change URL routing

4. **Run the server**:



Django Python Basic
"Parts List"
Project/Apps
Env Folder

```
django-admin startproject my_tennis_club
python manage.py startapp members
python -m venv myworld
```



Generate code (Ctrl+I), or select a language (Ctrl+K M). Start typing to dismiss or don't show this again.

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
            myfirst.html
```

Add Templates folder for HTML files
Add urls.py for Routing
Tell my_tennis_club urls.py where to look

👍Let’s make your myworld venv auto-activate in VS Code

🔷 Step 1: Open your project in VS Code

Open this folder as your **workspace**:
C:\GoHere\T_Labs\Projects\Python_Tests\Py_Django_App

🔷 Step 2: Tell VS Code which Python to use

1. Press **Ctrl+Shift+P** → search for **Python: Select Interpreter**.
2. Pick:
3. .\myworld\Scripts\python.exe

(that’s your venv’s Python interpreter).
This tells VS Code to always use that venv for this project.

🔷 Step 3: Create a settings.json file

In your project, VS Code will create a hidden .vscode folder. Inside it, add or edit:
Py_Django_App/.vscode/settings.json

```
{
  "python.defaultInterpreterPath": "C:\\GoHere\\T_Labs\\Projects\\Python_Tests\\Py_Django_App\\myworld\\Scripts\\python.exe",
  "python.terminal.activateEnvironment": true
}
```

- python.defaultInterpreterPath → points to your venv.
- python.terminal.activateEnvironment → makes new terminals automatically activate it.

🔷 Step 4: (Optional) Auto-activate in Debugging

If you ever use **Run/Debug** in VS Code, add a launch.json file:
Py_Django_App/.vscode/launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Django",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}\\my_tennis_club\\manage.py",
      "args": ["runserver"],
      "django": true,
      "justMyCode": true,
      "console": "integratedTerminal"
    }
  ]
}
```

Now pressing **F5** will launch your Django server with the venv active.

✅ Result:

- Every time you open Py_Django_App in VS Code → your venv myworld will be used.
- New terminals auto-activate (myworld).
- Debugger knows about Django.

Add myworld to .gitignore

Never accidentally push the whole venv to GitHub - Include a requirements.txt file to list dependencies.

◆ Step 1: Create or edit .gitignore

In your project root (Py_Django_App), create a file named .gitignore (if it doesn't exist) and add:

```
# Python virtual environment
myworld/
*.pyc
__pycache__/
db.sqlite3
.env
```

Explanation:

- myworld/ → ignores your venv folder
- *.pyc and __pycache__/ → ignore Python cache files
- db.sqlite3 → ignore local SQLite database (if using it)
- .env → ignore environment variable files (if you store secrets there)

◆ Step 2: Check if venv is already tracked

If you previously added myworld to Git, run:

```
git rm -r --cached myworld
```

- --cached → removes it from Git tracking **without deleting the folder locally**.

Then commit:

```
git add .gitignore
git commit -m "Ignore virtual environment"
```

◆ Step 3: Add requirements.txt

Generate a list of dependencies so anyone else can recreate your environment:

```
python -m pip freeze > requirements.txt
```

- This creates a requirements.txt file with all installed packages (like Django).
- Others can run:

```
python -m venv myworld
.\myworld\Scripts\activate
pip install -r requirements.txt
```

and get the exact same environment.

✅ Result:

- myworld is ignored by Git.
- Your code and requirements.txt are safe to push.

If you want, I can **mock up a full .gitignore for a typical Django + venv project** so you can just copy it into your project. Do you want me to do that?

A key piece of how Django’s **development server** works.

◆ Default URL: http://127.0.0.1:8000/

1. **127.0.0.1** → “localhost” (your own computer).
    - This is the loopback IP address.
    - It tells your browser to connect to your **local machine** instead of the internet.
2. **8000** → default port used by Django’s development server.
    - You can think of a port as a “channel” for network communication.
    - Django chooses **8000** by default unless you tell it otherwise.

◆ How it’s set

When you run:

```
python manage.py runserver
```

It is actually running:

```
django.core.management.commands.runserver.Command.handle()
```

Inside, if you don’t pass any arguments:

```
python manage.py runserver [addrport]
```

- Django defaults to 127.0.0.1:8000.
- The server listens on that IP and port.

◆ How to change it

1. **Change the port**:

```
python manage.py runserver 8080
```

→ runs on http://127.0.0.1:8080/

2. **Change the IP and port** (e.g., allow other devices on your network):

```
python manage.py runserver 0.0.0.0:8000
```

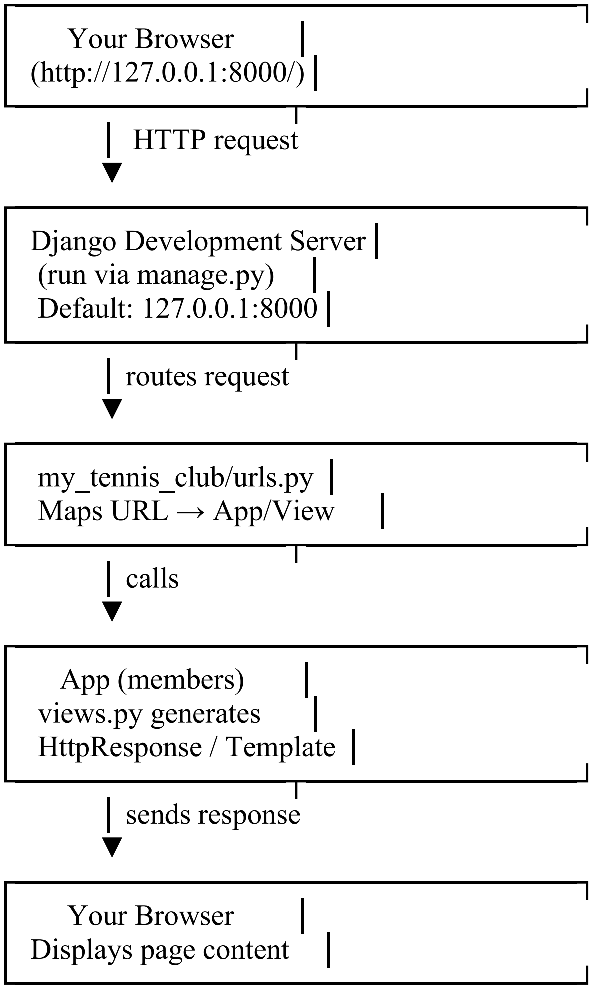
- 0.0.0.0 = listen on all network interfaces.
- You can then access it from another computer on the same network using your PC’s IP.

◆ Summary

Component	**Default**	**How to change**
IP address	127.0.0.1	runserver [IP]
Port	8000	runserver [IP:PORT]
Browser URL	http://127.0.0.1:8000/	change port/IP above

Simple Visual “Map”

Showing how Django’s development server maps http://127.0.0.1:8000/ to your project and browser.



◆ **Notes**

- 127.0.0.1 = your own PC (loopback).
- 8000 = default port for Django’s dev server.
- urls.py maps the incoming request to the **correct app/view**.
- The app’s views.py generates the response (HTML, JSON, etc.).

Fancier "full stack view" including:

- myworld venv
- Django project (my_tennis_club)
- Apps (members)
- Templates/static files
- Browser

That way you can see **everything from VS Code → server → browser**.

Perfect! Here's a "full stack" view of your Django setup, showing **VS Code → virtual environment → project → apps → templates/static → browser** and how requests flow through everything.

```
VS Code / Terminal
┌─────────────────────────────────────┐
│ Open project folder:                │
│ C:\GoHere\T_Labs\Projects\          │
│ Python_Tests\Py_Django_App          │
│                                     │
│ Terminal activates venv:            │
│ .\myworld\Scripts\Activate.ps1      │
└─────────────────┬───────────────────┘
      │
      ▼
Virtual Environment (myworld)
┌─────────────────────────────────────┐
│ Python interpreter & packages       │
│ - Django                            │
│ - Other libs                        │
└─────────────────┬───────────────────┘
      │ python / pip commands
      ▼
Django Project (my_tennis_club)
┌─────────────────────────────────────┐
│ manage.py (entry point)             │
│ my_tennis_club/                     │
│  - settings.py (config)             │
│  - urls.py (routes URLs)            │
│  - wsgi/asgi.py                     │
└─────────────────┬───────────────────┘
      │ routes HTTP requests
      ▼
Django Apps (e.g., members)
┌─────────────────────────────────────┐
│ views.py → functions/classes        │
│ models.py → database tables         │
│ templates/ → HTML pages             │
│ static/ → CSS/JS/images             │
└─────────────────┬───────────────────┘
      │ generates response
      ▼
Django Development Server
┌─────────────────────────────────────┐
```

```
│ runserver → listens on      │
│ 127.0.0.1:8000 (default)    │
└─────────────────────────────┴───────────────────┘
             │ HTTP response
             ▼
Browser
┌─────────────────────────────────────────────────┐
│ http://127.0.0.1:8000/      │
│ Renders pages from views      │
│ CSS/JS from static folders    │
└─────────────────────────────────────────────────┘
```

◆ **How the flow works**

1. **VS Code terminal** → you activate myworld so Python/Django commands point to the venv.
2. **manage.py** runs commands like runserver or migrate.
3. **urls.py** maps URLs to specific app views.
4. **Apps** contain views.py for logic, models.py for data, and optional templates/static for front-end.
5. **Django dev server** listens on 127.0.0.1:8000 and sends responses.
6. **Browser** receives HTML/CSS/JS and displays pages.