

QSpiders Global

PYTHON FULL STACK



INSTRUCTOR: RAJAT NAROJI
Subject: DJANGO-FRAMEWORK

Overview:

Software:

- A software is a set of programs or instructions that tells a computer to perform some specific task, from simple calculations to complex operations.

Types of Softwares:

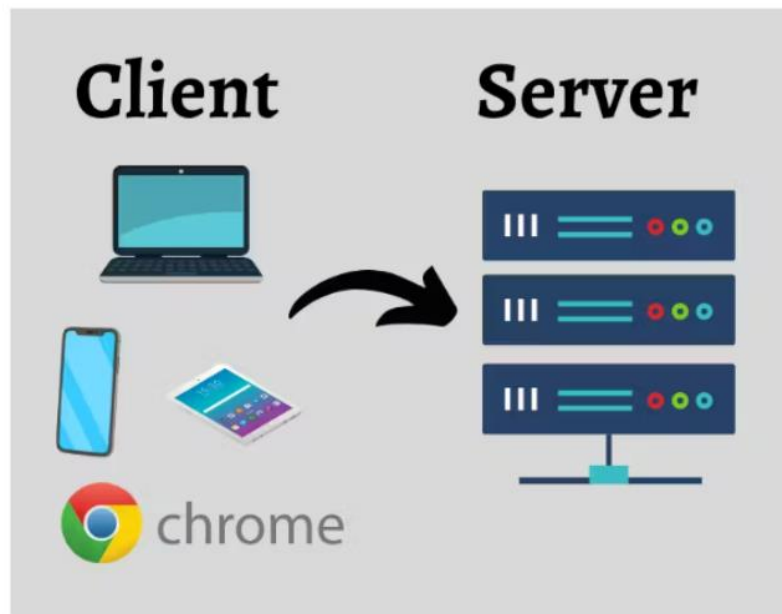
Standalone Software

These are the softwares which do not need any database or internet connectivity



Client - Server

- These are the type of application which we need Internet connectivity to interact with.
- This is where Client sends a request and the Server responds to that particular request



Mobile Apps

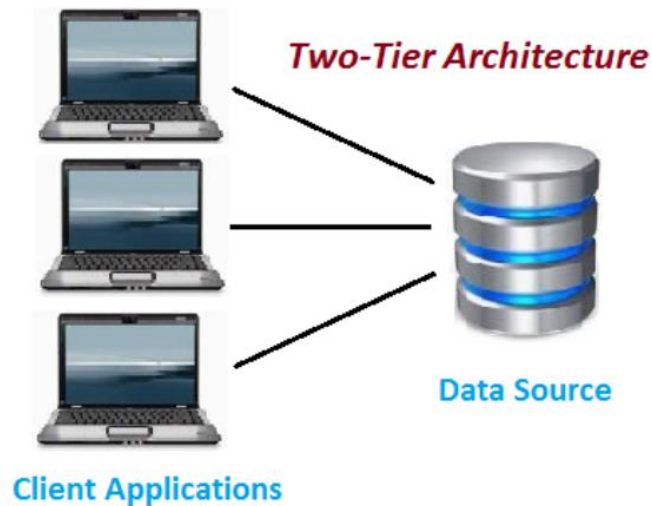
- Software applications designed to run on mobile devices like smartphones and tablets, providing specific functionalities or services to users on the go.



Different Architectures in Client-Server:

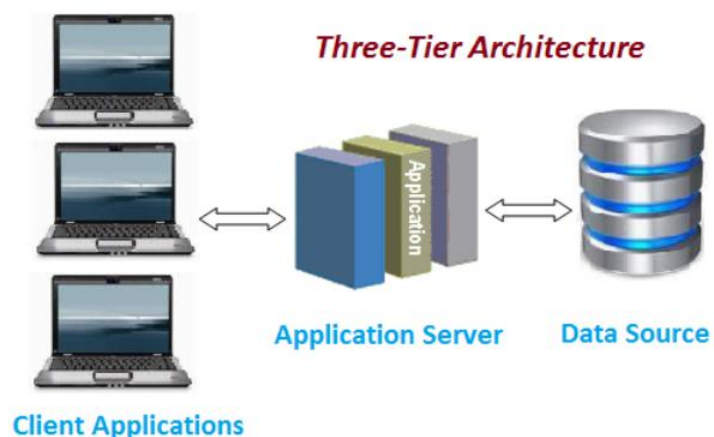
2-Tier Architecture

- Has Two layers: Client & the Server.
- Client connects directly to the database.
- Application logic can be placed in the server database, the client, or both.



3-Tier Architecture

- Has Three layers: the client layer, business layer and data layer
- Application logic is placed in the middle layer, separate from client and the server



Framework

- It is a set of conceptual structure and guidelines that is used to build something useful.

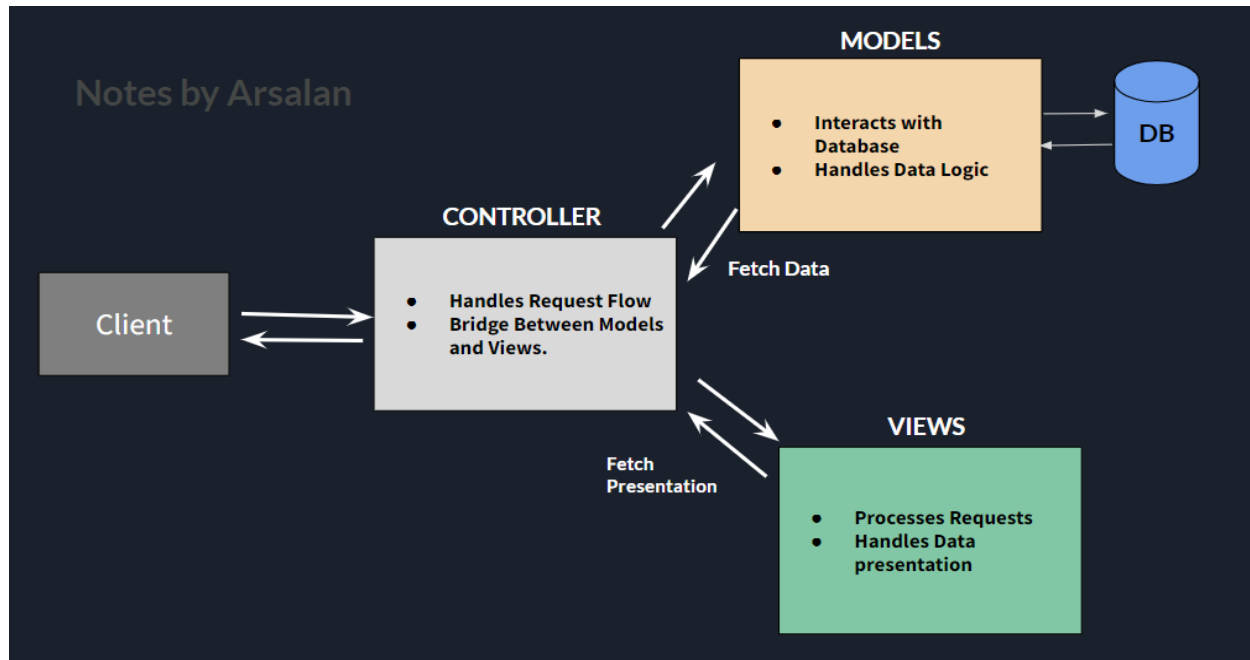
Example :-

- *Consider a brick maker who initially makes bricks by hand. This involves shaping, measuring, and cutting the bricks to the required size. This process, while effective, is time-consuming and requires a lot of manual effort.*
- *Now, imagine we provide the brick maker with a mold or container. By using this mold, the brick maker can work faster and more productively. The mold ensures that each brick is uniform in size and shape, reducing the effort needed for shaping and measuring.*
- *In this analogy, the mold or container can be considered a framework. Just as the mold streamlines and enhances the brick-making process, a framework in software development provides a structured and efficient way to build applications. It offers pre-defined tools and components that developers can use to speed up their work and ensure consistency.*

Web Framework

- Web framework is a framework which helps us build web applications.
- It provides us with tools and libraries to simplify common web development operations. This can include web services, API's and other resources.
- It helps with a variety of tasks, from templating and database access to session management and code reuse.
- More than 80% of all the web app frameworks rely on MVC (Model View Controller) architecture.

MVC (Model, View, controller):

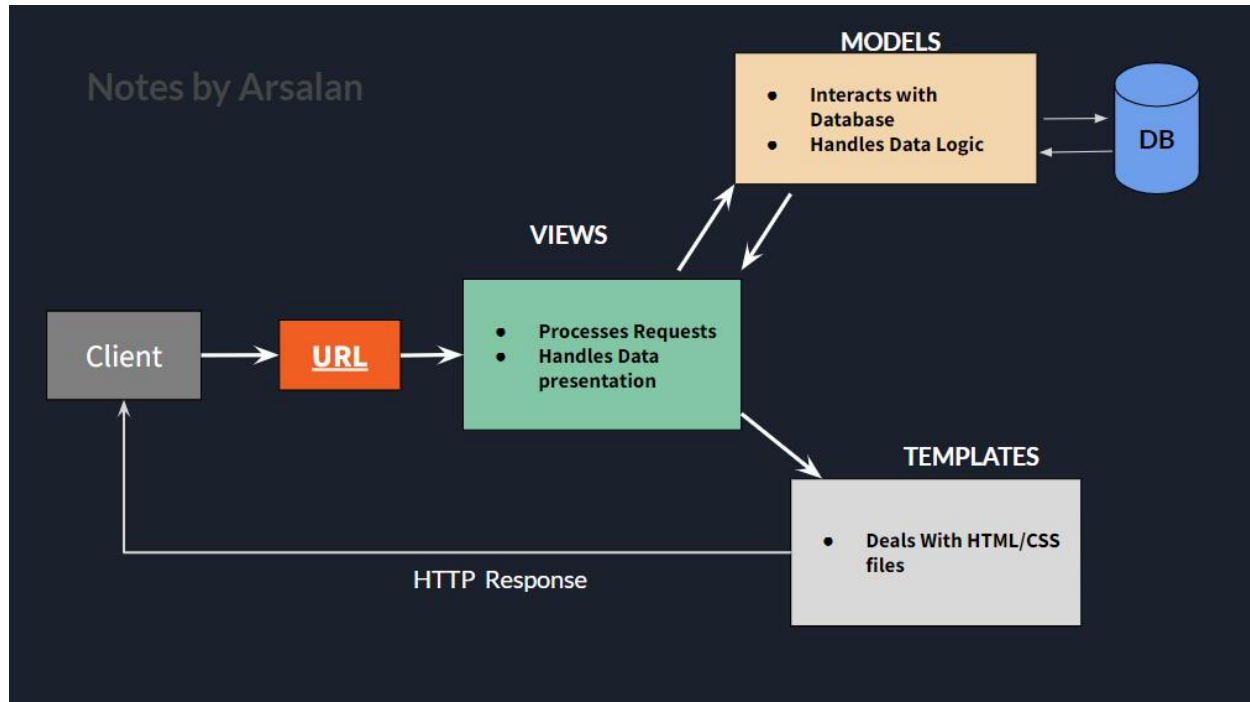


Working of MVC Architecture:

- **Model:** This is the component that is responsible for interacting and handling the database. Basically it is the data access layer that handles the data.
- **View:** It contains our entire Business or application Logic. It is responsible for taking requests, processing them, and to decide whether to do some logical operations or to render the templates based on the requests.
- **Controller:** It handles the request and response flow and serves as an intermediary between *Views & Models*.
- When it comes to MVC architecture when a user sends a request it first comes to the **controller**. The **controller** then sends this request to the **views** to process.
- Once the **view** receives the request it starts processing it and based on the request it will generate a response and give it back to the **controller**. After receiving the response the controller will send it to the client.
- If in case the user is trying to access the database while processing the request, then the view is going to make a request to fetch the data and send it to the **controller**, and the same will be sent to **models** to process the data request.
- After processing it the **model** is going to fetch the data from the database and return it back to the **views** through the **controller**.
- And after receiving the data **view** is going to generate the response which will be sent to the **client** by the **controller**.

\

MVT (Model, View, Templates):



Working of MVT Architecture:

- **Model:** This is the component that is responsible for interacting and handling the database. Basically it is the data access layer that handles the data.
- **View:** It is responsible for taking requests, processing them, and to decide whether to do some logical operations or to render the templates based on the requests. It acts like a mediator between templates and models.
- **Templates:** It represents how data should be shown to the application user. This is where users can read or write the data. Basically it is our user interface which consists of HTML, CSS, Javascript files mixed with (DTL) Django Template Language.

What is Django?

- It is a free, open source Python based, High-level web framework.
- It follows the Model View Template (MVT) architecture.
- It was created by Adrian Holovaty and Simon Wilson.
- **Advantages:** Open source, fast, secure, scalable, Authentication, Varsatile and provides Development Web Server and SQLite database by default.
- Website Built using Django: youtube, Instagram, Pinterest, dropbox, etc...

Django Requirements:

- Python 3.0 or higher
- PIP
- Text/code editor: VScode, PyCharm, Sublime, Notepad++, etc..
- Web browser.

System Environment V/S Virtual Environment:

System Environment:

- **Definition:** The default environment in which the operating system runs and manages all installed software and dependencies.
- **Dependencies:** All installed packages and libraries are globally available to all projects and users on the system.
- **Management:** Managing dependencies can become complex as different projects may require different versions of the same packages.
- **Risk:** There's a risk of version conflicts and dependency issues, affecting other projects or even system stability.

Virtual Environment:

- **Definition:** An isolated environment created specifically for a project to manage its dependencies independently from the system environment.
- **Dependencies:** Packages and libraries installed in a virtual environment are only available to that environment, ensuring isolation.
- **Management:** Each project can have its own dependencies and versions, making dependency management easier and more flexible.
- **Risk:** Reduced risk of version conflicts and dependency issues, as changes in one virtual environment do not affect others or the system environment.

Instal Django & Create Project

i) Set up a virtual environment:

```
python -m venv <folder_name>
```

ii) Activate the virtual environment:

```
venv\scripts\activate
```

iii) Install Django:

```
pip install django
```

iv) Create/Start project

```
django-admin startproject <project_name>
```

Eg: django-admin startproject school

To verify that Django is installed, type `python` from your shell. Then at the Python prompt, try to import Django:

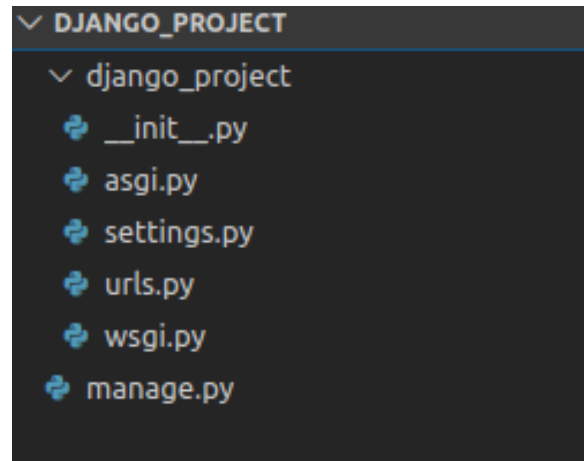
```
>>> import django
```

```
>>> print(django.get_version())
```

Django Project:

A Django project is a collection of settings and configurations for a particular web application. It acts as a container for various applications that make up the web application.

Django Project Directory Structure:



The outer **DJANGO_PROJECT** / root directory is a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.

The inner **django_project** is called Project Configuration Directory. It contains all the project configuration files. As listed below:

__init__.py: Any folder containing this file is considered as a python package.

wsgi.py: WSGI (Web Server Gateway Interface) describes how a web server communicates with the web application, and how web applications can be chained together to process a request. It provides a standard for synchronous Python apps.

asgi.py: ASGI (Asynchronous Server Gateway Interface) is a spiritual successor of WSGI, intended to provide standard interface between async-capable Python web servers, frameworks and applications. Basically it provides standards for both asynchronous and synchronous apps

settings.py: This file contains all the data about project settings.

Eg: Database config, Template, Installed apps, Validators.

urls.py: This file contains all the routing or urls associated with the application.

manage.py: It is created automatically along with the creation of every project. It is Django's command-line utility. It also sets the DJANGO_SETTINGS_MODULE environment variable so that it points to your project's settings.py file. Generally when we work on a simple project it's easier to work with manage.py instead of django-admin.

Application & its Creation:

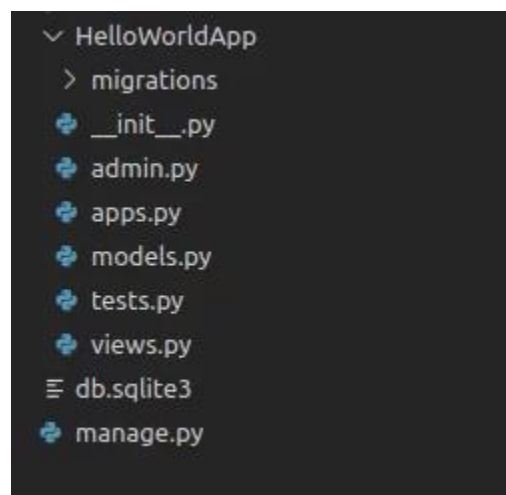
In Django, an app is a self-contained module that provides specific functionality to a web project. It is designed to be reusable and can be plugged into multiple projects. Each app focuses on a single aspect of the application and can include models, views, templates, static files, and other components.

Create Application:

- Once we create our project we first **cd** into the project/root directory
- And then we can create our apps as shown below

```
python manage.py startapp <app_name>
```

Django App Directory Structure:



__init__.py: Any folder containing this file is considered as a python package.

Migrations: This folder contains `__init__.py` file which means it's a python package. It also contains all files which are created after running some commands which are called migrations commands.

admin.py: This file is used to register sql tables so we could perform CRUD operation from the admin panel or application. Admin Application is provided by Django to Perform CRUD operation.

apps.py: This file is used to configure apps.

models.py: This file is used to create our own model classes which will be later converted into database tables by Django for our application.

test.py: This file is used to create tests.

views.py: This file is used to create views. We write all the business logic related code in this file.

Project vs App:

What's the difference between a project and an app?

- An app is a Web application that does something – e.g., a Weblog system, a database of public records or a simple library app, or payment gateway, or cart section, etc.. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps.

How to Run Server:

runserver: This command is used to run built-in server of Django

STEPS:

- Activate your virtual environment.
- Go to your project directory
- Run command ***python manage.py runserver***

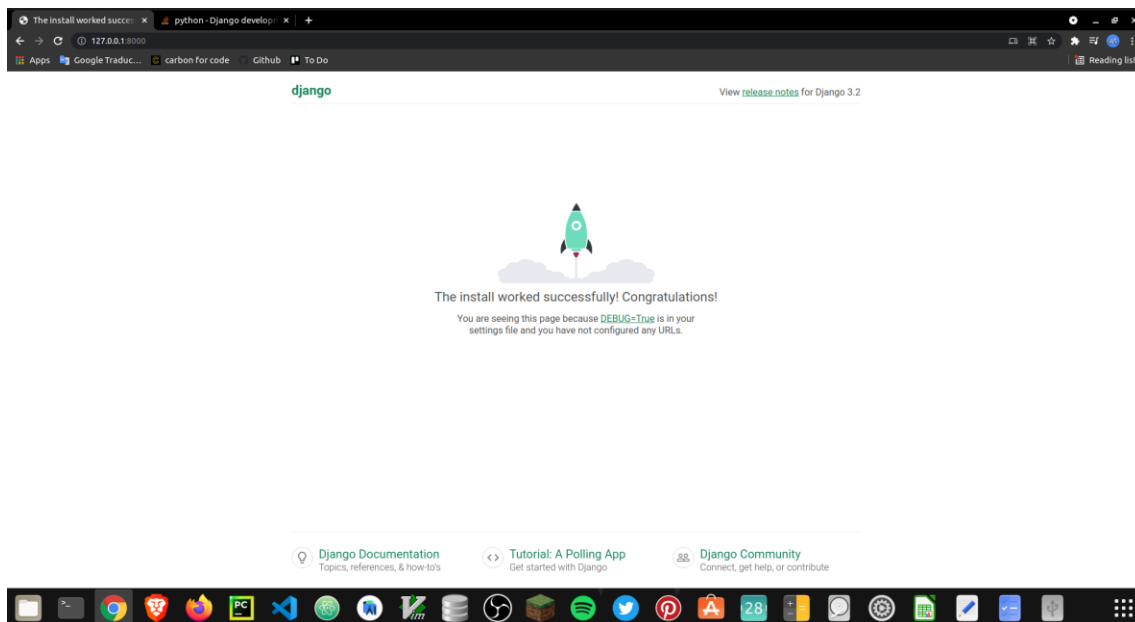
Once we run our server it runs on the port **8000** by default at <http://127.0.0.1:8000> or <http://localhost:8000>

If you want to run your server on any other port then you can do so by directly specifying the port you want to run your server on: **python manage.py runserver 5500**

Stopping the server: **ctrl + c** is used to Stop the server

***note:** sometimes when you make changes in your project and if the changes are not being reflected in your output, you may have to try restarting the server.*

Once we run our server we can now copy the url and paste it on to our browser and if we get the below output the you have successfully installed and configured your Django Project



DJANGO SETTINGS:

A Django settings.py file contains all the configurations of your Django Project. Let's learn how settings work and which settings are available.

The Basics:

- A settings file is just a python module with module level variables.
- Eg:

```
ALLOWED_HOSTS = ["www.example.com"]
DEBUG = False
DEFAULT_FROM_EMAIL = "webmaster@example.com"
```

Since settings is a python module following applies:

- It doesn't allow for python syntax errors.
- It can assign settings dynamically using normal python syntax.
- It can import values from other settings files.

Django provides us with sensible default values in our settings.

Creating your own settings:

We can define our own settings however we want for our django apps, one must follow these guidelines.

- Setting names must be all uppercase.
- Don't reinvent an already-existing setting.

Registering Apps in settings.py file:

- Once we make our apps before writing our views, we must register the app in our settings.py file.
- It needs to be done to consider it as an app of our particular.
- If we don't configure our app then it will be a non-existing part in our project.
- To do so we configure our INSTALLED_APPS variable as shown below

```
INSTALLED_APPS = [
    ...
    'myapp',
]
```

- Once we do this we are good to go now we are ready to write our first view.

Writing our first app:

views.py:

A view is a function or method that takes an HTTP request as an argument, imports the relevant models, and finds out what data to send to the template and returns the final result. These views are usually located in the file called **views.py**.

- To write our first view, go to the app directory and open views.py file.
- You will see some default statements which you have to ignore.
- We don't have to worry about any default python statements in Django.
- Now we can define a function and write our first view as done below.

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

- This is the simplest view possible in Django.
- This view is to basically print the text written inside our HttpResponse() function.
- Now to call this view we need to map it to a URL and to do so we need URL configuration.
- To create URL configuration in your app directory create a file name **urls.py**
- Now your app directory should look like this:

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

Now in your **urls.py** file include the following code:

```
from django.urls import path  
  
from . import views  
  
urlpatterns = [  
    path("", views.index, name="index"),  
]
```

The next step is to point the root URLConf at your **app.urls** module

- Go to your **project.urls** module.
- Add an import for **django.urls.include**.
- Now insert the **include()** in the **urlpatterns** list, so you have:

```
from django.contrib import admin  
from django.urls import include, path  
  
urlpatterns = [  
    path("polls/", include("polls.urls")),  
    path("admin/", admin.site.urls),  
]
```

The **include()** function allows referencing to another URLConf. Once Django encounters **include()** it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLConf for further processing. **include()** should be used when we want to include other URL patterns.

The idea behind **include()** is to make it easy to plug-and-play URLs.

The **path()** function is given four arguments, two required: **route** and **view**, and two optional: **kwargs** and **name**.

path() argument: **route**:

- **Route** is a string that contains a URL pattern. When processing requests, Django starts at the first pattern in urlpatterns and makes its way down the list, comparing the requested URL against each pattern until it finds one that matches.
- Patterns don't search for GET or POST parameters, or the domain name. For example : **https://www.example.com/myapp/**, the URLconf will look for **myapp/**.

path() argument : **view**

- When Django finds a matching pattern, it calls the specified view function with an **HttpRequest** object as the first argument and any "captured" values from the route as keyword arguments.

path() argument: **kwargs**

- Any keyword arguments can be passed in a dictionary to the target view.

path() argument: **name**

- Naming your URL lets you refer to it unambiguously from elsewhere in Django, especially from within templates.
- This powerful feature allows you to make global changes to the URL patterns of your project while only touching a single file.
- It is just like providing an alias name or pet name for your entire url for a specific route.