

Fundamentals of Data Science 21CSS202T



Unit III

Django Web Framework: Web development basics and Features of Django, Installing Django and MVC model , HTTP webserver concepts - Use HTTP request and response objects, Create Views, Use URLConf - URL Mapping

Introduction to Django Template System, Load Template Files, Render Templates, Create Forms, Process Form Data and Customize Form Field Validation

Introduction to Django Models, Use Model Fields, populate a Database, CRUD, Use QuerySets for data retrieval, Use jQuery and AJAX with Django to create Dynamic websites

T7: Implement Django framework using python – creating basic Django App

T8: Create a simple View using Django

T9: Implement Django app for real-time applications using MVC model

Django

- Django is a **back-end server side web framework**.
- Django is **free, open source** and written in Python.
- Django makes it **easier to build web pages** using Python.

Django

- Django is a **Python-based web framework** which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks.
- When you're building a website, you always need a similar set of components:
 - a way to handle user authentication (signing up, signing in, signing out)
 - a management panel for your website,
 - Forms
 - a way to upload files, etc.
- Django gives you ready-made components to use.

Why Django?

- It's very easy to switch database in Django framework.
- It has built-in admin interface which makes easy to work with it.
- Django is fully functional framework that requires nothing else.
- It has thousands of additional packages available.
- It is very scalable.

Features of Django

- **Versatility of Django**

Django can build almost any type of website. It can also work with any client-side framework and can deliver content in any format such as HTML, JSON, XML etc. Some sites which can be built using Django are wikis, social networks, new sites etc.

- **Security**

Since Django framework is made for making web development easy, it has been engineered in such a way that it automatically do the right things to protect the website. For example, In the Django framework instead of putting a password in cookies, the hashed password is stored in it so that it can't be fetched easily by hackers.

Features of Django

- **Scalability**

Django web nodes have no stored state, they scale horizontally – just fire up more of them when you need them. Being able to do this is the essence of good scalability. Instagram and Disqus are two Django based products that have millions of active users, this is taken as an example of the scalability of Django.

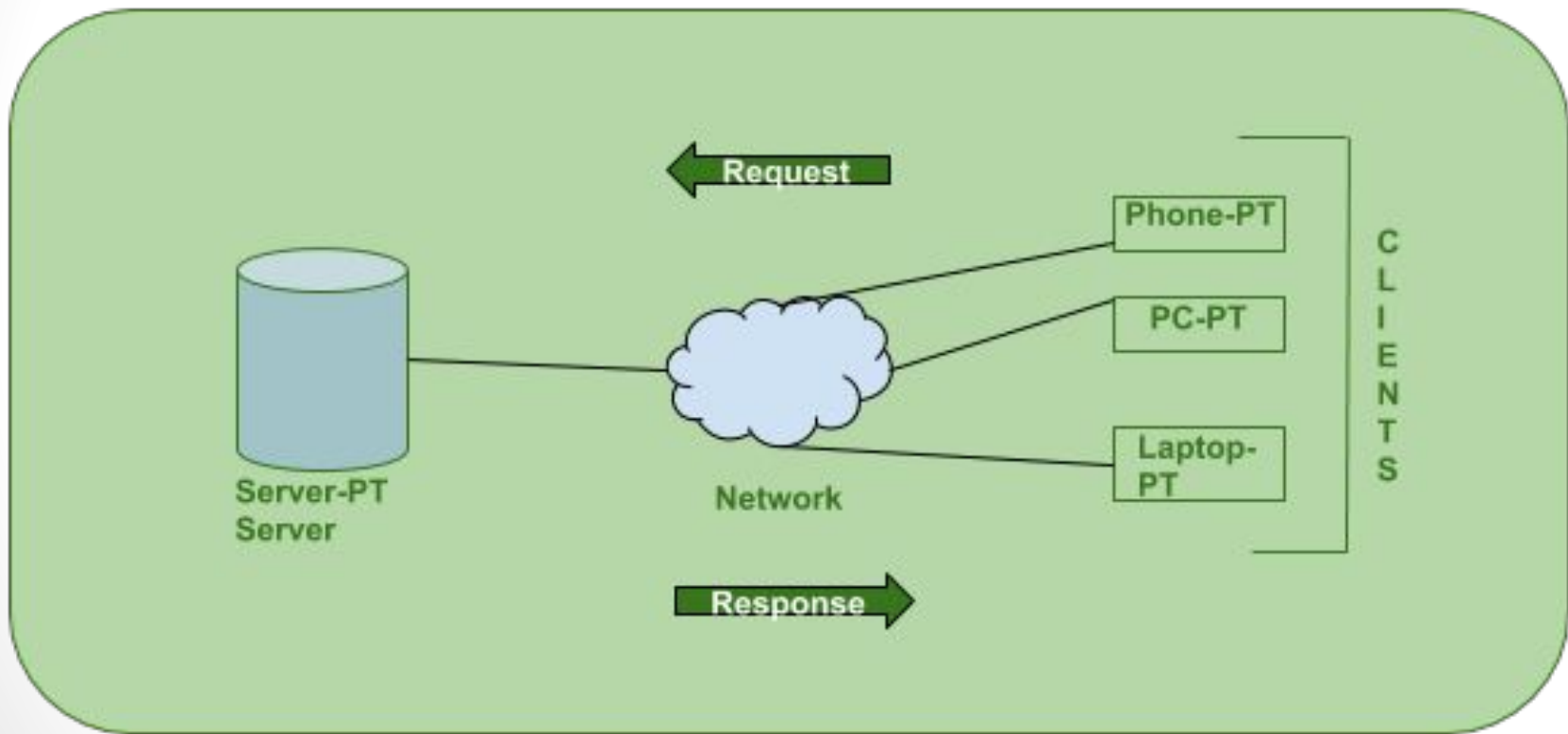
- **Portability**

All the codes of the Django framework are written in Python, which runs on many platforms. Which leads to run Django too in many platforms such as Linux, Windows and Mac OS.

Popularity of Django

- Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic etc. There are more than 5k online sites based on Django framework.
- Sites like [Hot Frameworks](#) assess the popularity of a framework by counting the number of GitHub projects and StackOverflow questions for each platform, here Django is in 6th position.

Web Development Basics



Web Development Basics

- The Client-server model is a **distributed application** structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.
- **Clients do not share any of their resources.**
- Examples of Client-Server Model are Email, World Wide Web, etc.

Web Development Basics

- **How the Client-Server Model works ?**

Client: When we talk the word **Client**, it mean to talk of a person or an organization using a particular service.

Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).

- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

Django Installation

- **Install pip-** Open command prompt and enter following command
 - **python -m pip install -U pip**
- **Install virtual environment-** Enter following command in cmd-
 - **pip install virtualenv**
- **Set Virtual environment-** Setting up the virtual environment will allow you to edit the dependency which generally your system wouldn't allow.

Django Installation

Command Prompt - pip install virtualenv

Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Kalpana>python -m pip install -U pip

Requirement already satisfied: pip in c:\users\kalpana\appdata\local\programs\python\python310\lib\site-packages (22.2.2)

C:\Users\Kalpana>

C:\Users\Kalpana>pip install virtualenv

Collecting virtualenv

Downloading virtualenv-20.16.5-py3-none-any.whl (8.8 MB)

----- 8.8/8.8 MB 5.7 MB/s eta 0:00:00

Collecting distlib<1,>=0.3.5

Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)

----- 468.5/468.5 kB 7.4 MB/s eta 0:00:00

Collecting platformdirs<3,>=2.4

Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)

Collecting filelock<4,>=3.4.1

Downloading filelock-3.8.0-py3-none-any.whl (10 kB)

Installing collected packages: distlib, platformdirs, filelock, virtualenv

Django Installation

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Kalpana>python -m pip install -U pip
Requirement already satisfied: pip in c:\users\kalpana\appdata\local\programs\python\python310\lib\site-packages (22.2.2)

C:\Users\Kalpana>
C:\Users\Kalpana>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.16.5-py3-none-any.whl (8.8 MB)
    ----- 8.8/8.8 MB 5.7 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.5
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
    ----- 468.5/468.5 kB 7.4 MB/s eta 0:00:00
Collecting platformdirs<3,>=2.4
  Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.8.0-py3-none-any.whl (10 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv
Successfully installed distlib-0.3.6 filelock-3.8.0 platformdirs-2.5.2 virtualenv-20.16.5

C:\Users\Kalpana>
```

Django Installation

- Follow these steps to set up a virtual environment

1) Create a virtual environment by giving this command
in

`virtualenv env_site`

2) Change directory to env_site by this command

`cd env_site`

3) Go to Scripts directory inside env_site and activate
virtual environment

`cd Scripts`

`activate`

Django Installation

Command Prompt

```
Requirement already satisfied: pip in c:\users\kalpana\appdata\local\programs\python\python310\lib\site-packages (22.2.2)
C:\Users\Kalpana>
C:\Users\Kalpana>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.16.5-py3-none-any.whl (8.8 MB)
    ----- 8.8/8.8 MB 5.7 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.5
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
    ----- 468.5/468.5 kB 7.4 MB/s eta 0:00:00
Collecting platformdirs<3,>=2.4
  Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.8.0-py3-none-any.whl (10 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv
Successfully installed distlib-0.3.6 filelock-3.8.0 platformdirs-2.5.2 virtualenv-20.16.5

C:\Users\Kalpana>virtualenv env_site
created virtual environment CPython3.10.7.final.0-64 in 72668ms
  creator CPython3Windows(dest=C:\Users\Kalpana\env_site, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\Kalpana\AppData\Local\pypa\virtualenv)
    added seed packages: pip==22.2.2, setuptools==65.3.0, wheel==0.37.1
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\Kalpana>
C:\Users\Kalpana>
C:\Users\Kalpana>
C:\Users\Kalpana>
```


Django Installation

```
Command Prompt
Downloading virtualenv-20.16.5-py3-none-any.whl (8.8 MB)
----- 8.8/8.8 MB 5.7 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.5
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
----- 468.5/468.5 kB 7.4 MB/s eta 0:00:00
Collecting platformdirs<3,>=2.4
  Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.8.0-py3-none-any.whl (10 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv
Successfully installed distlib-0.3.6 filelock-3.8.0 platformdirs-2.5.2 virtualenv-20.16.5

C:\Users\Kalpana>virtualenv env_site
created virtual environment CPython3.10.7.final.0-64 in 72668ms
  creator CPython3Windows(dest=C:\Users\Kalpana\env_site, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\Kalpana\AppData\Local\pypa\virtualenv)
    added seed packages: pip==22.2.2, setuptools==65.3.0, wheel==0.37.1
    activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\Kalpana>cd env_site

C:\Users\Kalpana\env_site>cd Scripts

C:\Users\Kalpana\env_site\Scripts>activate

(env_site) C:\Users\Kalpana\env_site\Scripts>
```

Django Installation

```
Command Prompt - pip install django
C:\Users\Kalpana>cd env_site
C:\Users\Kalpana\env_site>cd Scripts
C:\Users\Kalpana\env_site\Scripts>activate
(env_site) C:\Users\Kalpana\env_site\Scripts>pip install django
Collecting django
  Downloading Django-4.1.2-py3-none-any.whl (8.1 MB)
----- 8.1/8.1 MB 5.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
----- 42.8/42.8 kB 1.0 MB/s eta 0:00:00
Collecting tzdata
  Downloading tzdata-2022.4-py2.py3-none-any.whl (336 kB)
----- 336.7/336.7 kB 3.5 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
```

Django Installation

```
Command Prompt
C:\Users\Kalpana>cd env_site
C:\Users\Kalpana\env_site>cd Scripts
C:\Users\Kalpana\env_site\Scripts>activate
(env_site) C:\Users\Kalpana\env_site\Scripts>pip install django
Collecting django
  Downloading Django-4.1.2-py3-none-any.whl (8.1 MB)
    ----- 8.1/8.1 MB 5.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
    ----- 42.8/42.8 kB 1.0 MB/s eta 0:00:00
Collecting tzdata
  Downloading tzdata-2022.4-py2.py3-none-any.whl (336 kB)
    ----- 336.7/336.7 kB 3.5 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.2 django-4.1.2 sqlparse-0.4.3 tzdata-2022.4
(env_site) C:\Users\Kalpana\env_site\Scripts>
```

Django Installation

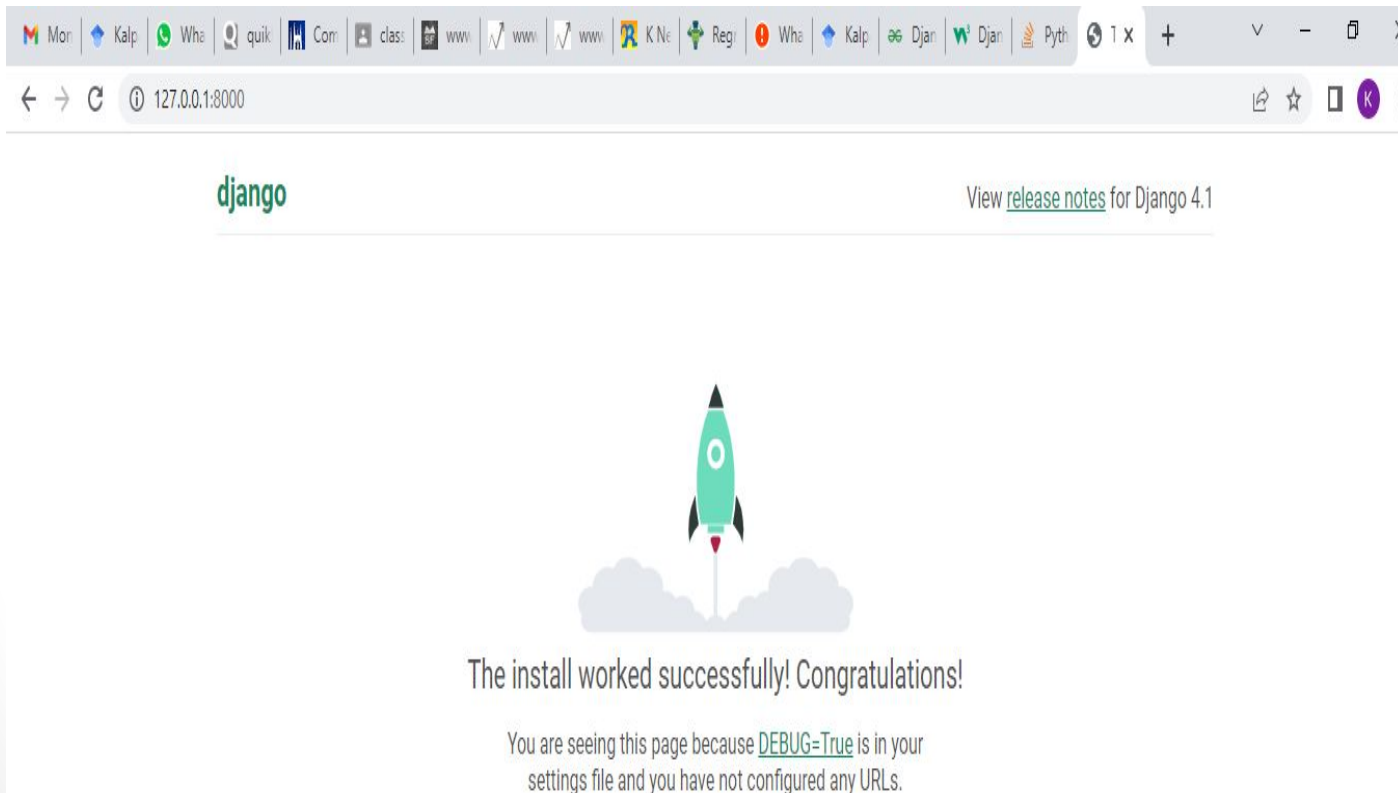
- **Install Django-** Install django by giving following command
`pip install django`
- Return to the env_site directory
`cd ..`
- Start a project by following command
`django-admin startproject fds-demo`
- Change directory to fds-demo
`cd fds-demo`
- **Start the server-** Start the server by typing following command in cmd
`python manage.py runserver`

Django Installation

```
ca. Command Prompt
C:\Users\Kalpana>cd env_site
C:\Users\Kalpana\env_site>cd Scripts
C:\Users\Kalpana\env_site\Scripts>activate
(env_site) C:\Users\Kalpana\env_site\Scripts>pip install django
Collecting django
  Downloading Django-4.1.2-py3-none-any.whl (8.1 MB)
    ----- 8.1/8.1 MB 5.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
    ----- 42.8/42.8 kB 1.0 MB/s eta 0:00:00
Collecting tzdata
  Downloading tzdata-2022.4-py2.py3-none-any.whl (336 kB)
    ----- 336.7/336.7 kB 3.5 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.2 django-4.1.2 sqlparse-0.4.3 tzdata-2022.4
(env_site) C:\Users\Kalpana\env_site\Scripts>cd..
(env_site) C:\Users\Kalpana\env_site>django-admin startproject fds_demo
(env_site) C:\Users\Kalpana\env_site>cd fds_demo
(env_site) C:\Users\Kalpana\env_site\fds_demo>_
```

Django Installation

- To check whether server is running or not go to web browser and enter **http://127.0.0.1:8000/** as url.



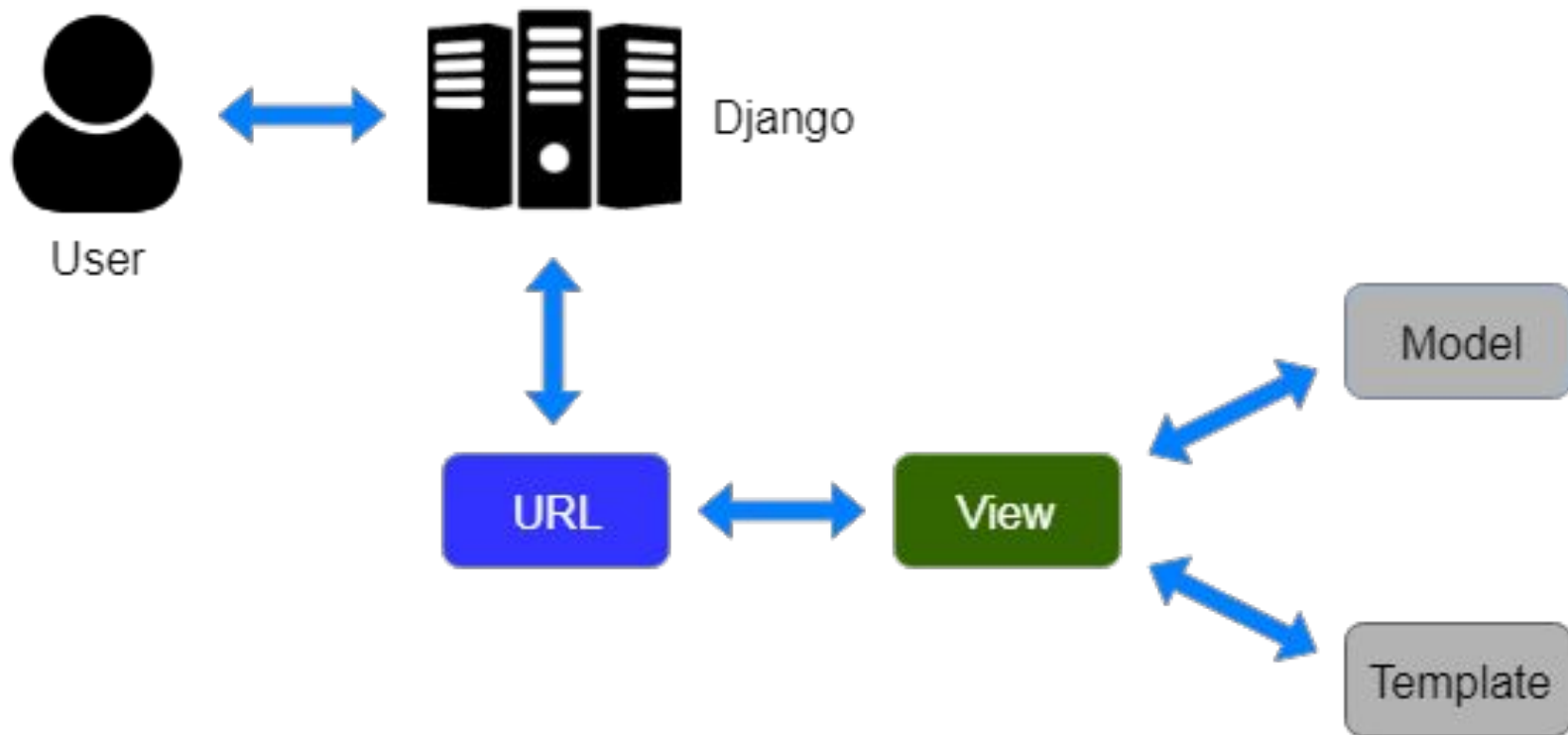
MVC Model

- Django is based on **MVT (Model-View-Template)** architecture.
- MVT is a software design pattern for developing a web application.
- MVT has three parts:
 - **Model**
 - **View**
 - **Template**

MVC Model

- The Model helps to **handle database**. It is a data access layer which handles the data.
- The Template is a presentation layer which handles **User Interface part completely**. The View is used to **execute the business logic** and interact with a model to carry data and renders a template.
- Although Django follows MVC pattern but maintains it's own conventions. So, control is handled by the framework itself.
- There is no separate controller and complete application is based on Model View and Template. That's why it is called MVT application.

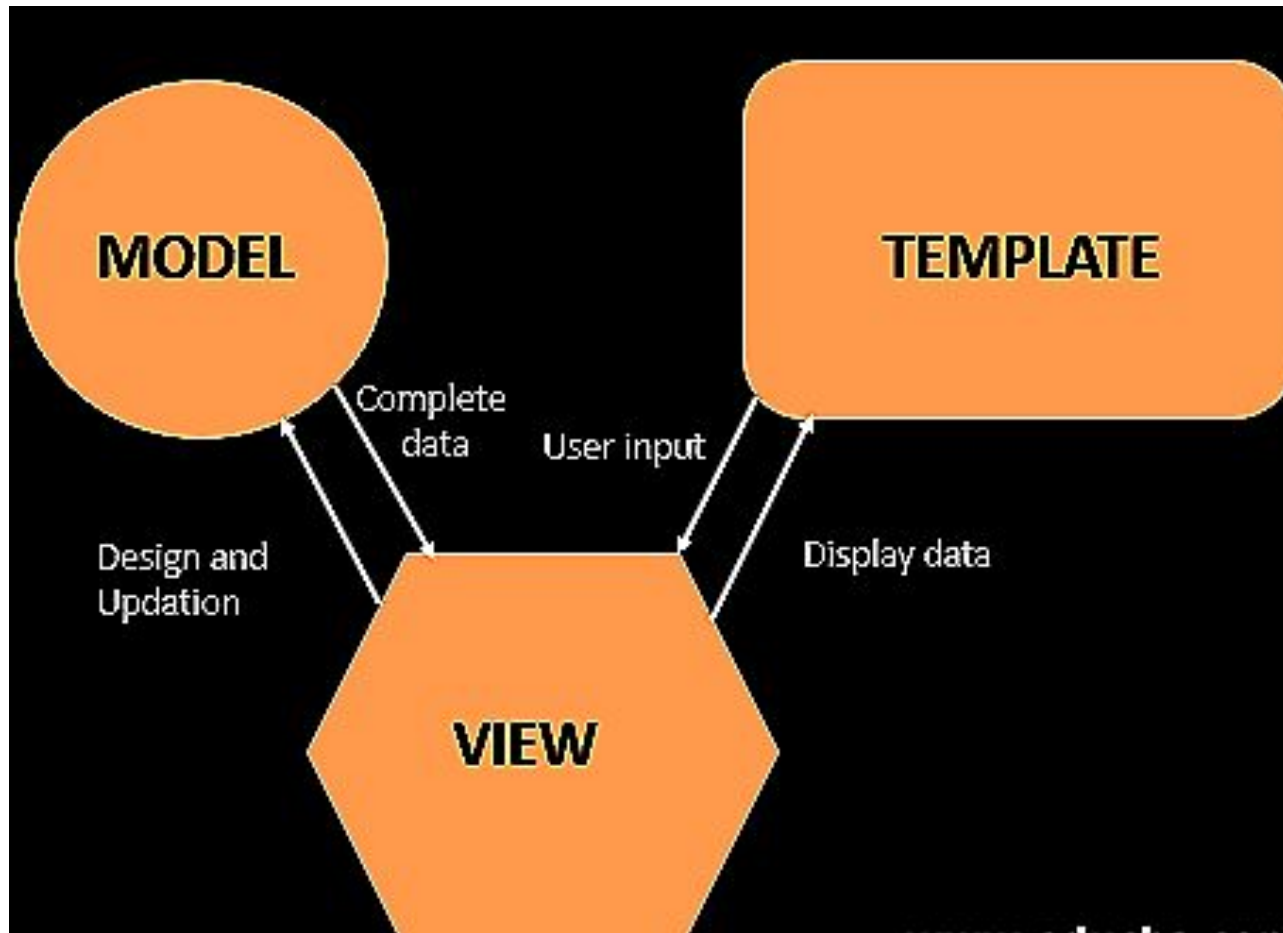
MVC Model



MVC Model

- Here, a user **requests** for a resource to the Django, Django works as a controller and check to the available resource in URL.
- If URL maps, **a view is called** that interact with model and template, it renders a template.
- Django responds back to the user and sends a template as a **response**.

MVC Model



MVC Model

- **Model:** The model is going to act as the **interface of your data**. It is responsible for **maintaining data**. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySQL, Postgres).
- **View:** The View is the **user interface** — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.
- **Template:** A template consists of **static parts of the desired HTML output** as well as some special syntax describing how dynamic content will be inserted.

Project Structure

Ctrl + C – exit from server

Command Prompt - python manage.py runserver

```
(env_site) C:\Users\Kalpana\env_site\Scripts>cd..

(env_site) C:\Users\Kalpana\env_site>cd fds_site
The system cannot find the path specified.

(env_site) C:\Users\Kalpana\env_site>cd fds_sitenew

(env_site) C:\Users\Kalpana\env_site\fds_sitenew>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
October 10, 2022 - 10:04:36
Django version 4.1.2, using settings 'fds_sitenew.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[10/Oct/2022 10:04:54] "GET / HTTP/1.1" 200 10681
[10/Oct/2022 10:04:56] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[10/Oct/2022 10:04:57] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
[10/Oct/2022 10:04:57] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
[10/Oct/2022 10:04:57] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
Not Found: /favicon.ico
[10/Oct/2022 10:04:59] "GET /favicon.ico HTTP/1.1" 404 2115
```

Project Structure

- A Django Project when initialized contains basic files by default such as `manage.py`, `view.py`, etc.
- A simple project structure is enough to create a single-page application.
- Here are the major files and their explanations. Inside the `fds_sitenew` folder (project folder) there will be the following files-
 - **`manage.py`**
 - **`fds_sitenew` (Folder)**

Project Structure

Command Prompt

```
(env_site) C:\Users\Kalpana\env_site\fds_sitenew>dir
Volume in drive C has no label.
Volume Serial Number is 7051-49B0

Directory of C:\Users\Kalpana\env_site\fds_sitenew

10-10-2022  10:04    <DIR>          .
10-10-2022  10:04    <DIR>          ..
10-10-2022  10:04                0 db.sqlite3
10-10-2022  10:04    <DIR>          fds_sitenew
10-10-2022  10:03                689 manage.py
                2 File(s)                689 bytes
                3 Dir(s)  62,153,637,888 bytes free

(env_site) C:\Users\Kalpana\env_site\fds_sitenew>
```

File Explorer: fds_sitenew

File | Home | Share | View

Clipboard: Pin to Quick access, Copy, Paste

Organise: New folder, Properties, Select

Search: Search fds_sitenew

Name	Date modified
fds_sitenew	10-10-2022 10:04
db.sqlite3	10-10-2022 10:04
manage	10-10-2022 10:03

Project Structure

- **manage.py**- This file is used to interact with your project via the command line(start the server, sync the database... etc). For getting the full list of commands that can be executed by manage.py type this code in the command window-

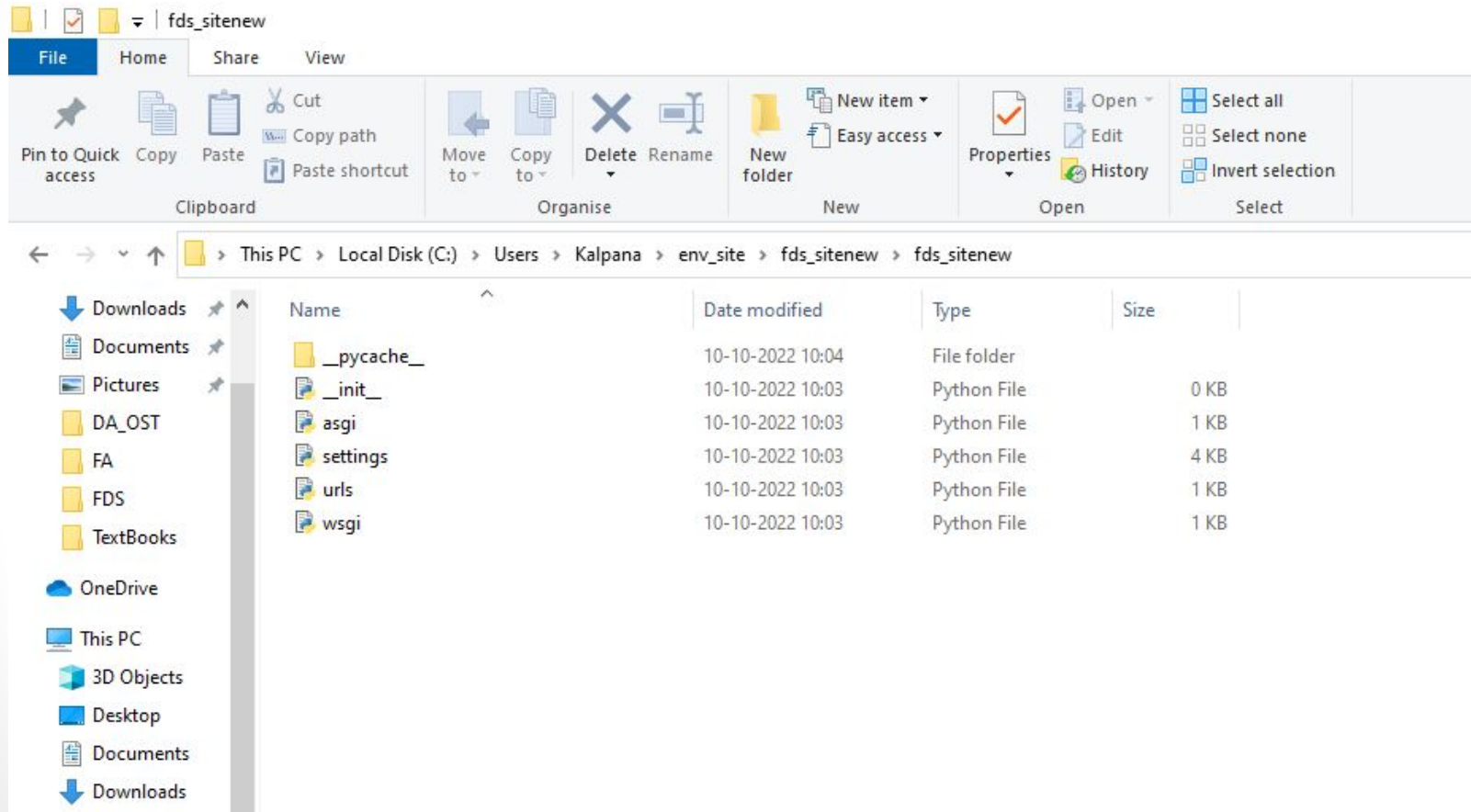
\$ python manage.py help

- **folder (fds_sitenew)** – This folder contains all the packages of your project. Initially, it contains four files –
 - **_init_.py**
 - **settings.py**
 - **url.py**
 - **wsgi.py**

Project Structure

- **_init_.py** – It is a python package. It is invoked when the package or a module in the package is imported. We usually use this to execute package initialization code, for example for the initialization of package-level data.
- **settings.py** – As the name indicates it contains all the website settings. In this file, we register any applications we create, the location of our static files, database configuration details, etc.
- **urls.py** – In this file, we store all links of the project and functions to call.
- **wsgi.py** – This file is used in deploying the project in WSGI. It is used to help your Django application communicate with the webserver.

Project Structure



Project Structure

Command Prompt - python manage.py runserver

Volume in drive C has no label.

Volume Serial Number is 7051-49B0

Directory of C:\Users\Kalpana\env_site\fds_sitenew

```
10-10-2022  10:04    <DIR>        .
10-10-2022  10:04    <DIR>        ..
10-10-2022  10:04                0 db.sqlite3
10-10-2022  10:04    <DIR>        fds_sitenew
10-10-2022  10:03                689 manage.py
                2 File(s)                689 bytes
                3 Dir(s)  62,153,637,888 bytes free
```

(env_site) C:\Users\Kalpana\env_site\fds_sitenew>cd..

(env_site) C:\Users\Kalpana\env_site>cd Scripts

(env_site) C:\Users\Kalpana\env_site\Scripts>activate

(env_site) C:\Users\Kalpana\env_site\Scripts>cd..

(env_site) C:\Users\Kalpana\env_site>cd fds_sitenew

(env_site) C:\Users\Kalpana\env_site\fds_sitenew>python manage.py runserver

Watching for file changes with StatReloader

Performing system checks...

Project Structure

Create a new file `views.py` inside the project folder where `settings.py`, `urls.py` and other files are stored and save the following code

```
# HttpResponseRedirect is used to  
# pass the information  
# back to view  
from django.http import HttpResponseRedirect
```

```
# Defining a function which  
# will receive request and  
# perform task depending  
# upon function definition  
def hello_srm (request) :
```

```
# This will return Hello Geeks  
# string as HttpResponseRedirect  
return HttpResponseRedirect("Hello SRM")
```

Project Structure

- Create a new file `views.py` inside the project folder where `settings.py`, `#urls.py` and other files are stored and save the following code

views.py - C:\Users\Kalpana\env_site\fds_sitenew\fds_sitenew\views.py (3.10.7)

File Edit Format Run Options Window Help

```
# HttpResponse is used to
# pass the information
# back to view
from django.http import HttpResponse

# Defining a function which
# will receive request and
# perform task depending
# upon function definition
def hello_srm (request) :

    # This will return Hello Geeks
    # string as HttpResponse
    return HttpResponse("Hello SRM")
```

Project Structure

- Open **urls.py** inside project folder (projectName) and add your entry-
 - Import **hello_srm** function from views.py file.
from projectName.views import hello_srm

Project Structure

urls.py - C:\Users\Kalpana\env_site\fds_sitenew\fds_sitenew\urls.py (3.10.7)

File Edit Format Run Options Window Help

```
"""fds_sitenew URL Configuration
```

```
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.1/topics/http/urls/
```

```
Examples:
```

```
Function views
```

1. Add an import: `from my_app import views`
2. Add a URL to urlpatterns: `path('', views.home, name='home')`

```
Class-based views
```

1. Add an import: `from other_app.views import Home`
2. Add a URL to urlpatterns: `path('', Home.as_view(), name='home')`

```
Including another URLconf
```

1. Import the include() function: `from django.urls import include, path`
2. Add a URL to urlpatterns: `path('blog/', include('blog.urls'))`

```
"""
```

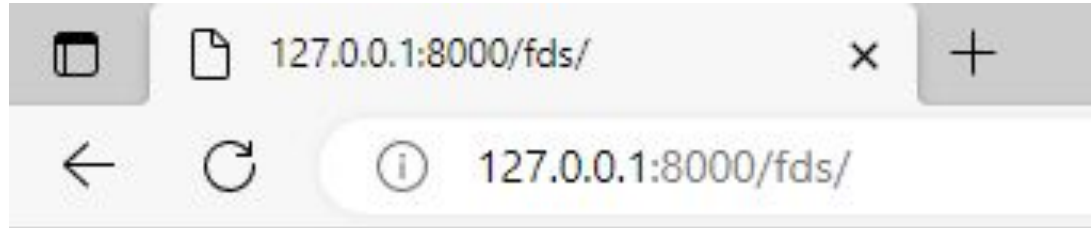
```
from django.contrib import admin
```

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

```
from fds_sitenew.views import hello_srm
```

```
urlpatterns = [
    path('fds/', hello_srm),
]
```

Project Structure



Hello SRM

Middleware

- Middleware is like a middle ground between a request and response.
- It is like a window through which data passes. As in a window, light passes in and out of the house. Similarly, when a request is made it moves through middlewares to views, and data is passed through middleware as a response.

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

HttpRequest and HttpResponse

- The client-server architecture includes two major components **request** and **response**.
- The Django framework uses **client-server architecture** to implement web applications.
- When a client requests for a resource, a **HttpRequest** object is created and correspond view function is called that returns **HttpResponse** object.
- To handle request and response, Django provides HttpRequest and HttpResponse classes.
- Each class has it's own attributes and methods.

HttpRequest and HttpResponse

- Django uses request and response objects to pass state through the system.

When a page is requested, Django creates an HttpRequest object that contains metadata about the request. Then Django loads the appropriate view, passing the HttpRequest as the first argument to the view function. Each view is responsible for returning an HttpResponse object.

- **Django HttpRequest**
- This class is defined in the **django.http** module and used to handle the client request.

HttpRequest and HttpResponse

Attribute	Description
HttpRequest.scheme	A string representing the scheme of the request (HTTP or HTTPS usually).
HttpRequest.body	It returns the raw HTTP request body as a byte string.
HttpRequest.path	It returns the full path to the requested page does not include the scheme or domain.
HttpRequest.path_info	It shows path info portion of the path.
HttpRequest.method	It shows the HTTP method used in the request.
HttpRequest.encoding	It shows the current encoding used to decode form submission data.
HttpRequest.content_type	It shows the MIME type of the request, parsed from the CONTENT_TYPE header.

HttpRequest and HttpResponse

Attribute	Description
HttpRequest.content_params	It returns a dictionary of key/value parameters included in the CONTENT_TYPE header.
HttpRequest.GET	It returns a dictionary-like object containing all given HTTP GET parameters.
HttpRequest.POST	It is a dictionary-like object containing all given HTTP POST parameters.
HttpRequest.COOKIES	It returns all cookies available.
HttpRequest.FILES	It contains all uploaded files.
HttpRequest.META	It shows all available Http headers.
HttpRequest.resolver_match	It contains an instance of ResolverMatch representing the resolved URL.

HttpRequest - Methods

Attribute	Description
<code>HttpRequest.get_host()</code>	It returns the original host of the request.
<code>HttpRequest.get_port()</code>	It returns the originating port of the request.
<code>HttpRequest.get_full_path()</code>	It returns the path, plus an appended query string, if applicable.
<code>HttpRequest.build_absolute_uri(<i>location</i>)</code>	It returns the absolute URI form of location.
<code>HttpRequest.get_signed_cookie(<i>key</i>, <i>default=RAISE_ERROR</i>, <i>salt=""</i>, <i>max_age=None</i>)</code>	It returns a cookie value for a signed cookie, or raises a <code>django.core.signing.BadSignature</code> exception if the signature is no longer valid.
<code>HttpRequest.is_secure()</code>	It returns True if the request is secure; that is, if it was made with HTTPS.
<code>HttpRequest.is_ajax()</code>	It returns True if the request was made via an XMLHttpRequest.

HttpRequest

Django HttpRequest Example

views.py

```
def methodinfo(request):  
    return HttpResponse("Http request is: "+request.method)
```

urls.py

```
path('info',views.methodinfo)
```

HttpRequest

Django HttpRequest Example

views.py

```
def methodinfo(request):  
    return HttpResponse("Http request is: "+request.method)
```

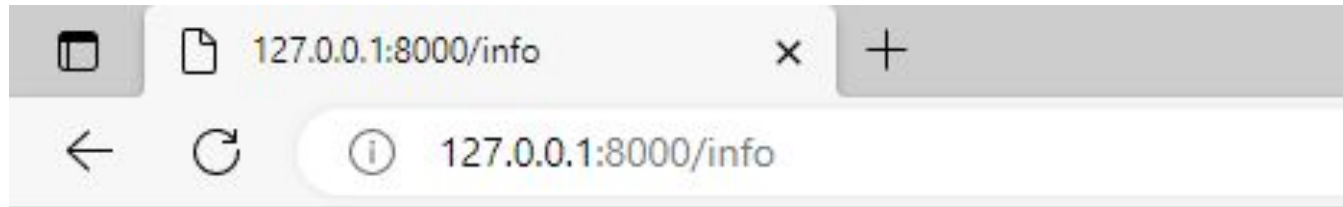
urls.py

```
from newdemo.views import methodinfo
```

```
path('info', methodinfo)
```


HttpRequest

Django HttpRequest Example



Http request is: GET

HttpResponse

- Django uses request and response objects to pass state through the system.
When a page is requested, Django creates an `HttpRequest` object that contains metadata about the request. Then Django loads the appropriate view, passing the `HttpRequest` as the first argument to the view function. Each view is responsible for returning an `HttpResponse` object.
- This class is a part of **`django.http`** module.
- It is responsible for generating response corresponds to the request and back to the client.

HttpResponse

Attribute	Description
HttpResponse.content	A bytestring representing the content, encoded from a string if necessary.
HttpResponse.charset	It is a string denoting the charset in which the response will be encoded.
HttpResponse.status_code	It is an HTTP status code for the response.
HttpResponse.reason_phrase	The HTTP reason phrase for the response.
HttpResponse.streaming	It is false by default.
HttpResponse.closed	It is True if the response has been closed.

HttpResponse

Method	Description
<code>HttpResponse.__init__(content="", content_type=None, status=200, reason=None, charset=None)</code>	It is used to instantiate an HttpResponse object with the given page content and content type.
<code>HttpResponse.__setitem__(header, value)</code>	It is used to set the given header name to the given value.
<code>HttpResponse.__delitem__(header)</code>	It deletes the header with the given name.
<code>HttpResponse.__getitem__(header)</code>	It returns the value for the given header name.
<code>HttpResponse.has_header(header)</code>	It returns either True or False based on a case-insensitive check for a header with the provided name.
<code>HttpResponse.setdefault(header, value)</code>	It is used to set default header.

HttpResponse

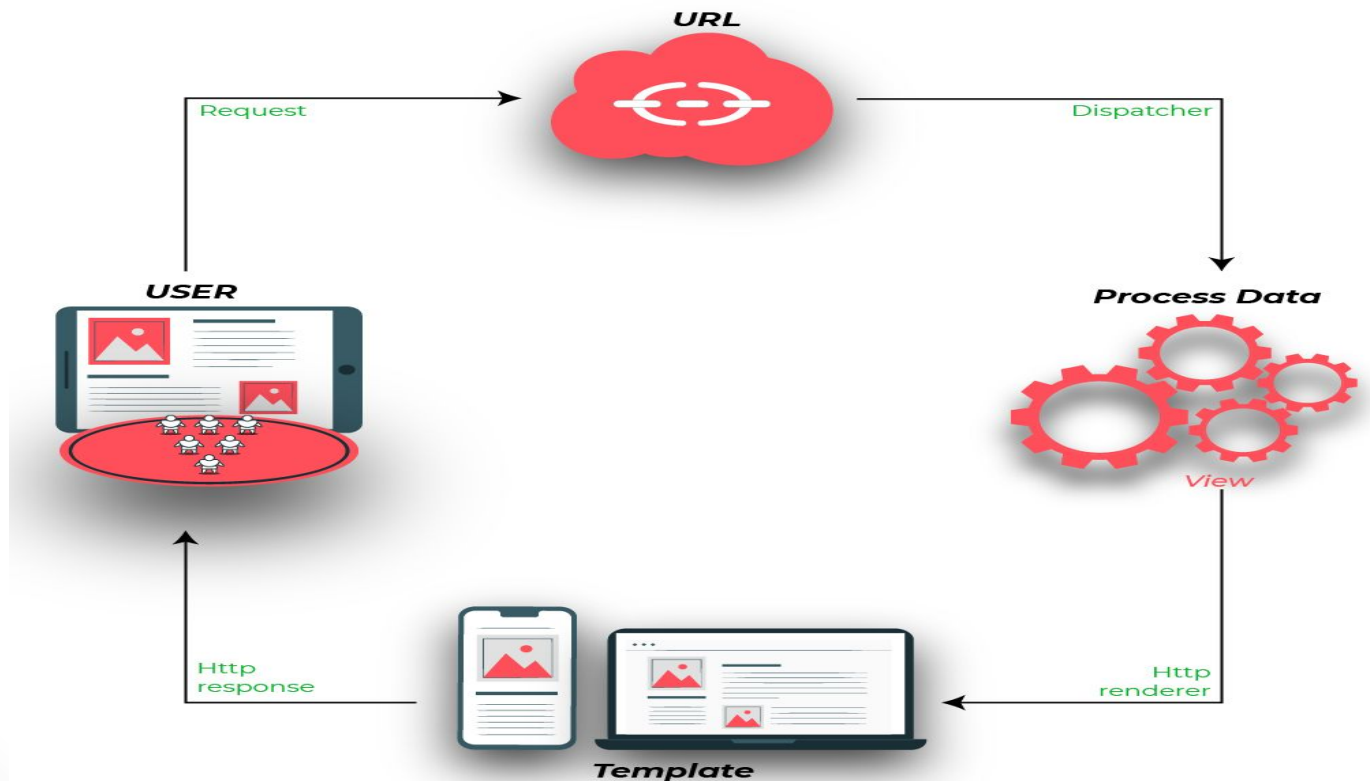
Method	Description
<code>HttpResponse.write(<i>content</i>)</code>	It is used to create response object of file-like object.
<code>HttpResponse.flush()</code>	It is used to flush the response object.
<code>HttpResponse.tell()</code>	This method makes an <code>HttpResponse</code> instance a file-like object.
<code>HttpResponse.getvalue()</code>	It is used to get the value of <code>HttpResponse.content</code> .
<code>HttpResponse.readable()</code>	This method is used to create stream-like object of <code>HttpResponse</code> class.
<code>HttpResponse.seekable()</code>	It is used to make response object seekable.

Create Views

- Django Views are one of the vital participants of MVT Structure of Django.
- As per Django Documentation, **A view function is a Python function that takes a Web request and returns a Web response.**
- This **response** can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser can display.

Create Views

- Django views are part of the user interface — they usually render the HTML/CSS/Javascript in your Template files into what you see in your browser when you render a web page.



Create Views

```
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime
```

```
# create a function
def demo_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```


Create Views

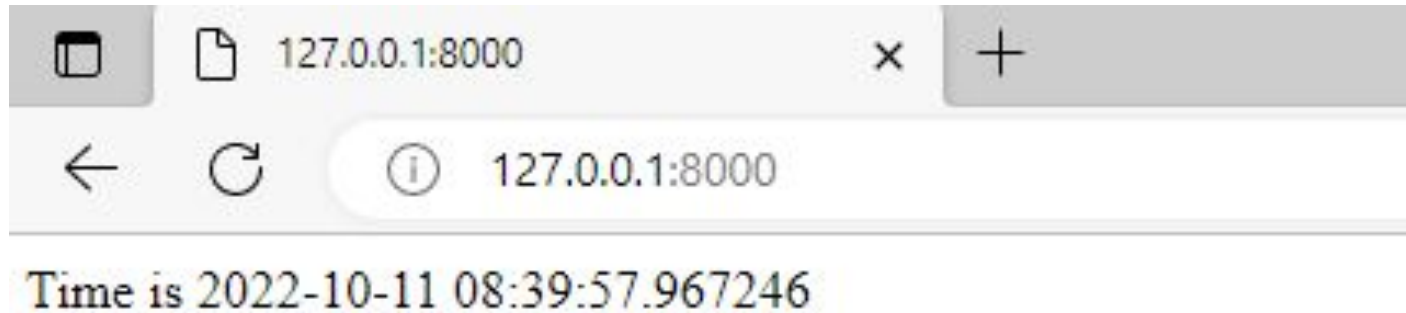
```
from django.urls import path
```

```
# importing views from views..py
```

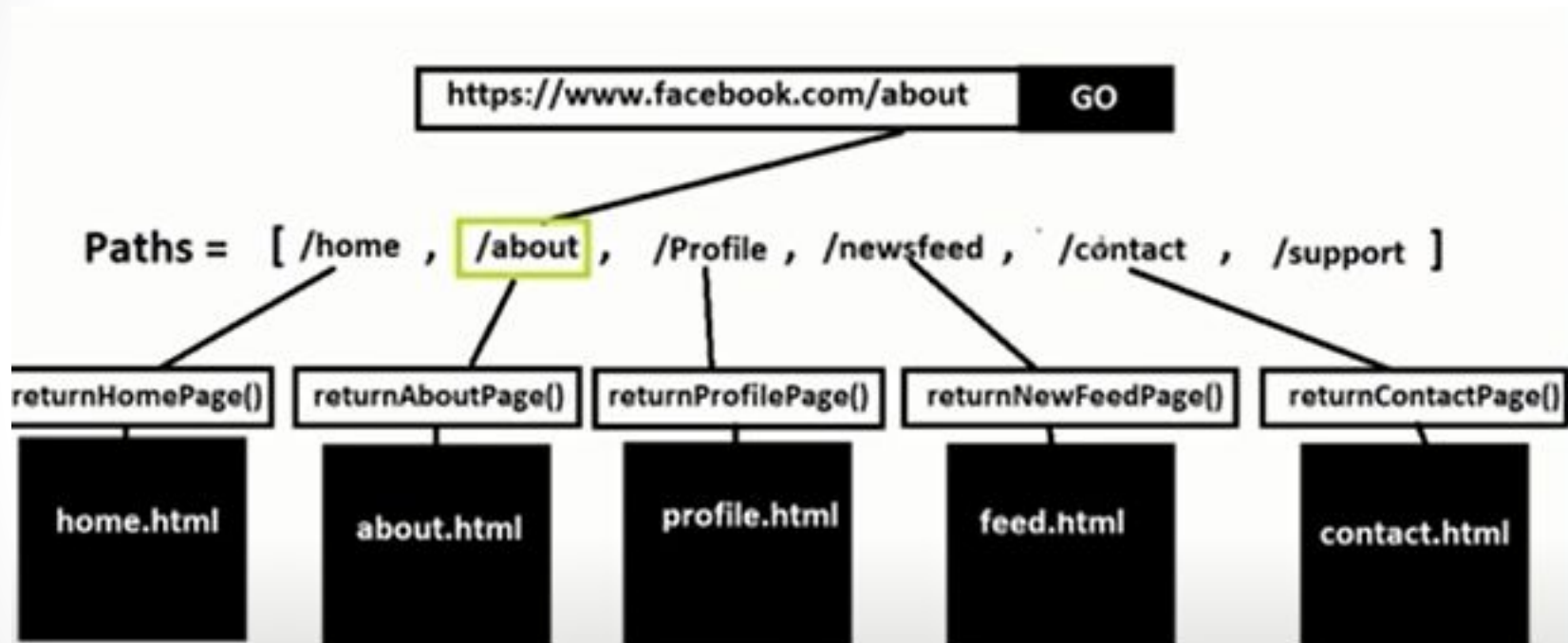
```
from .views import demo_view
```

```
urlpatterns = [  
    path("", demo_view),  
]
```

Create Views



Use URLConf



Use URLConf

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

from django.http import HttpResponseRedirect

def home(request):
    return HttpResponseRedirect("Home page")

def contact(request):
    return HttpResponseRedirect("Contact page")

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", home),
    path('about/', contact),
]
```

Use URLConf

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

from django.http import HttpResponse

def home(request):
    return HttpResponse("Home page")

def contact(request):
    return HttpResponse("Contact page")

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home),
    path('about/', contact),
]
```

Use URLConf



Home page



Contact page

Use URLConf

```
(env_site) C:\Users\Kalpana\env_site\admindemo>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
October 11, 2022 - 08:56:25
Django version 4.1.2, using settings 'admindemo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[11/Oct/2022 08:56:25] "GET / HTTP/1.1" 200 9
[11/Oct/2022 08:58:16] "GET /about/ HTTP/1.1" 200 12
(env_site) C:\Users\Kalpana\env_site\admindemo>
```

URL Mapping

- Django is a web application framework, it gets user requests by URL locator and responds back.
- To handle URL, **django.urls** module is used by the framework.

urls.py

```
from django.contrib import admin
```

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

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


URL Mapping

- Django already has mentioned a URL here for the admin. The path function takes the first argument as a route of string or regex type.
- The view argument is a view function which is used to return a response (template) to the user.
- The **django.urls** module contains various functions, **path(route,view,kwargs,name)** is one of those which is used to map the URL and call the specified view.

URL Mapping

```
from django.shortcuts import render
# Create your views here.
from django.http import HttpResponse, HttpResponseNotFound
from django.views.decorators.http import require_http_methods
@require_http_methods(["GET"])
def hello(request):
    return HttpResponse('<h1>This is Http GET request.</h1>')
```

URL Mapping

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
    path('hello/', views.hello),
]
```

- Now, start the server and enter **127.0.0.1:8000/hello** to the browser. This URL will be mapped into the list of URLs and then call the corresponding function from the views file.
- In this example, hello will be mapped and call hello function from the views file. It is called URL mapping.

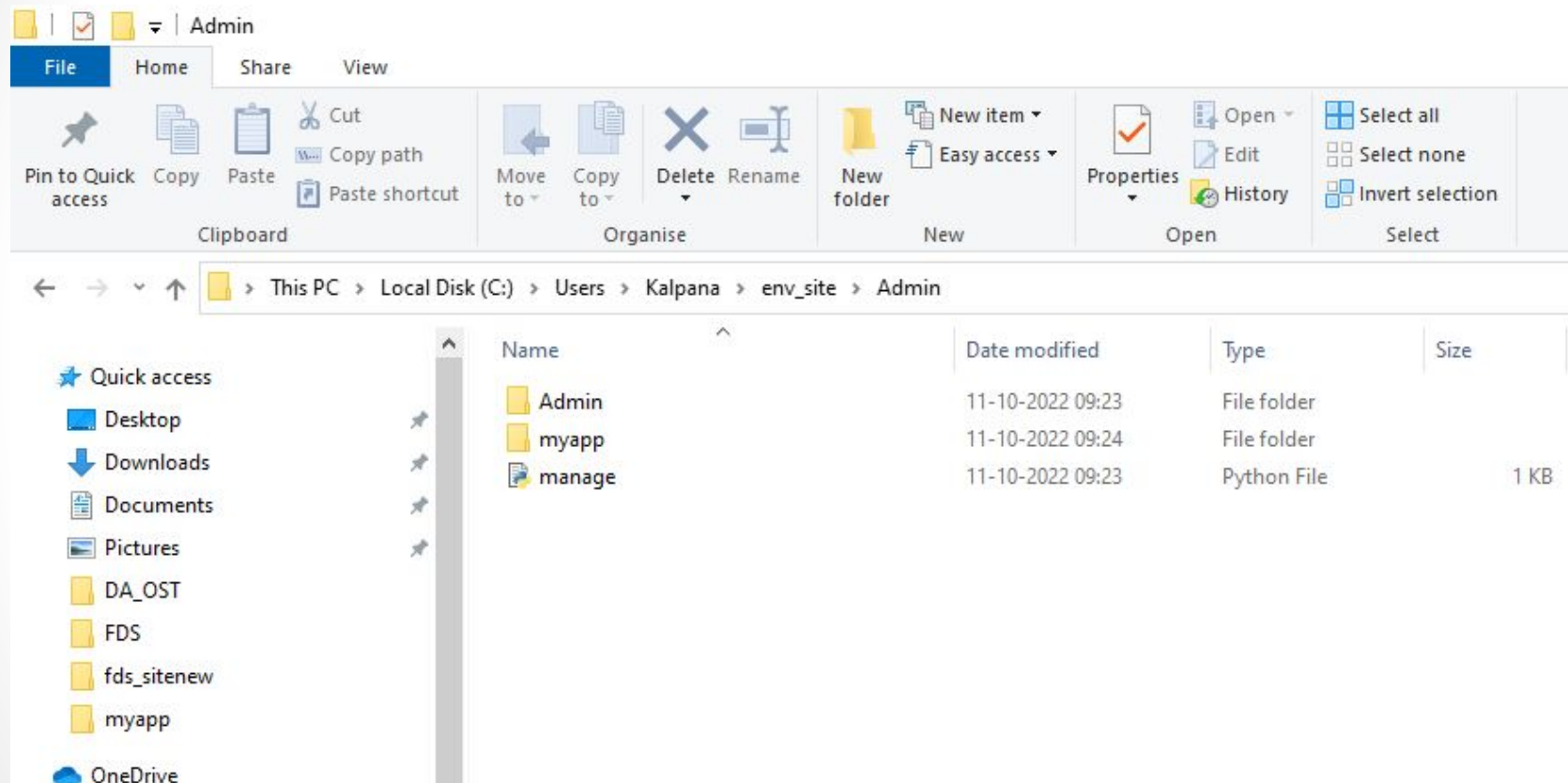
Create an App

- A project is a sum of many applications.
- Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others.
- See it as a module of your project.

Create an Application

- We assume you are in your project folder. In our main “myproject” folder, the same folder then `manage.py` –
- **\$ `python manage.py startapp myapp`**

Create an App



Create an App

- You just created myapp application and like project, Django create a “myapp” folder with the application structure –

```
myapp/  
    __init__.py  
    admin.py  
    models.py  
    tests.py  
    views.py
```

Create an App

- **__init__.py** – Just to make sure python handles this folder as a package.
- **admin.py** – This file helps you make the app modifiable in the admin interface.
- **models.py** – This is where all the application models are stored.
- **tests.py** – This is where your unit tests are.
- **views.py** – This is where your application views are.

Create an App

- **Get the Project to Know About Your Application**
- At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update `INSTALLED_APPS` tuple in the `settings.py` file of your project (add your app name) –

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'myapp',  
)
```


Create an App

```
# Application definition

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

Create an App

- Before launching your server, to access your Admin Interface, you need to initiate the database –
\$ python manage.py migrate
- syncdb will create necessary tables or collections depending on your db type, necessary for the admin interface to run. Even if you don't have a superuser, you will be prompted to create one.
- If you already have a superuser or have forgotten it, you can always create one using the following code –
\$ python manage.py createsuperuser

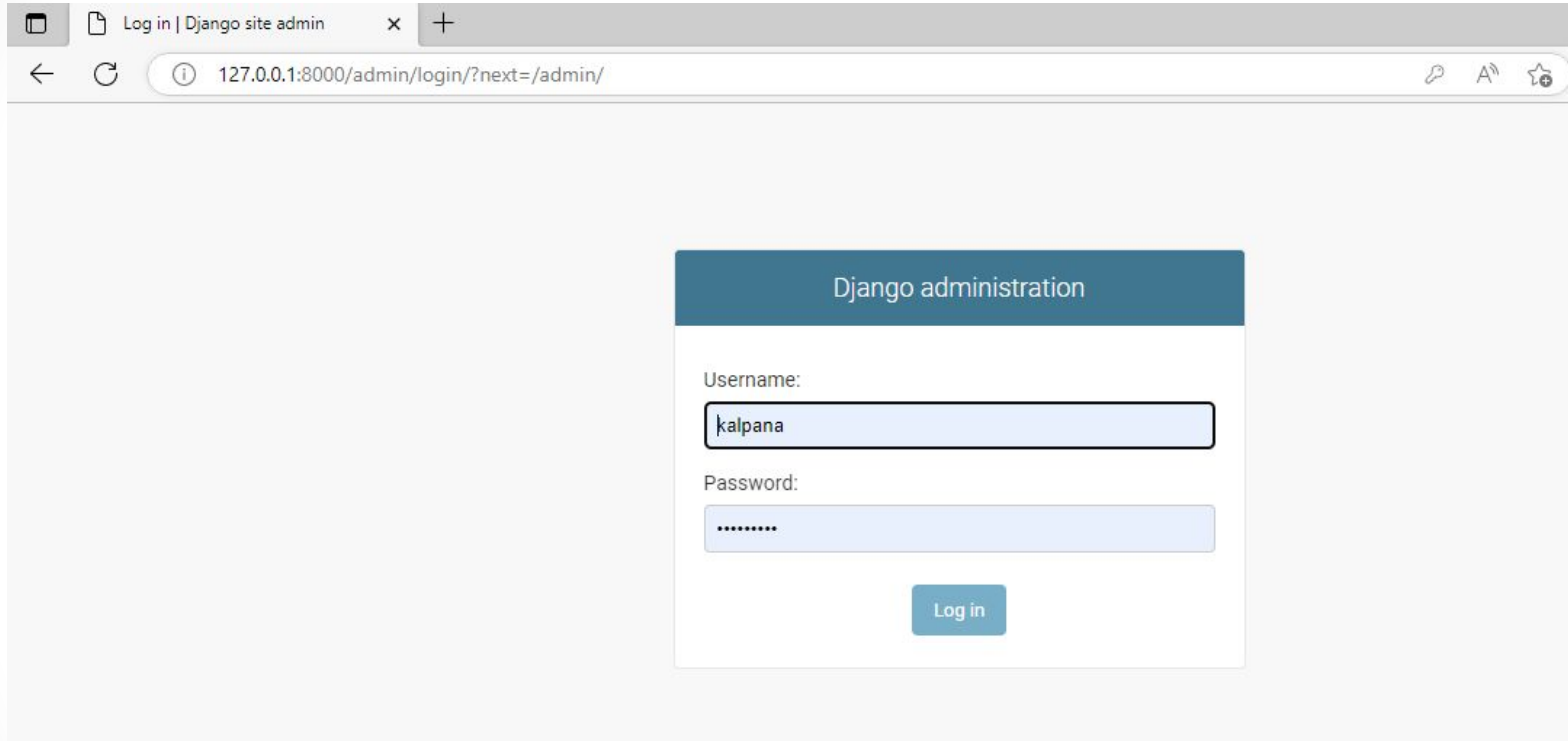
Create an App

- Now just run the server.

\$ python manage.py runserver

And your admin interface is accessible at:
<http://127.0.0.1:8000/admin/>

Create an App



The image shows a web browser window with the title "Log in | Django site admin". The address bar displays the URL "127.0.0.1:8000/admin/login/?next=/admin/". The main content area features a "Django administration" login form. The form has a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". The "Username:" field contains the text "kalpana". The "Password:" field is masked with dots. A blue "Log in" button is positioned below the password field.

Log in | Django site admin

127.0.0.1:8000/admin/login/?next=/admin/

Django administration

Username:

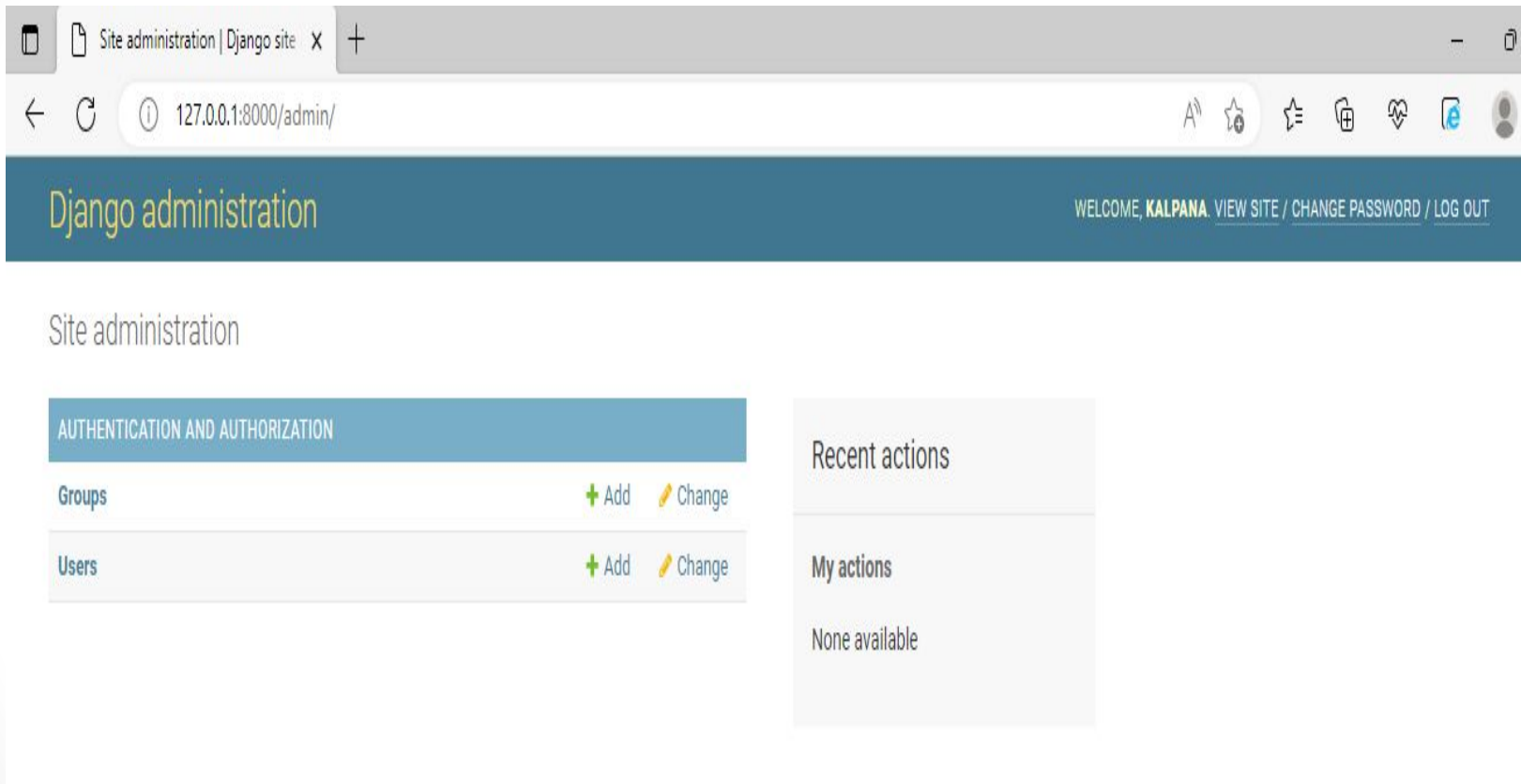
kalpana

Password:

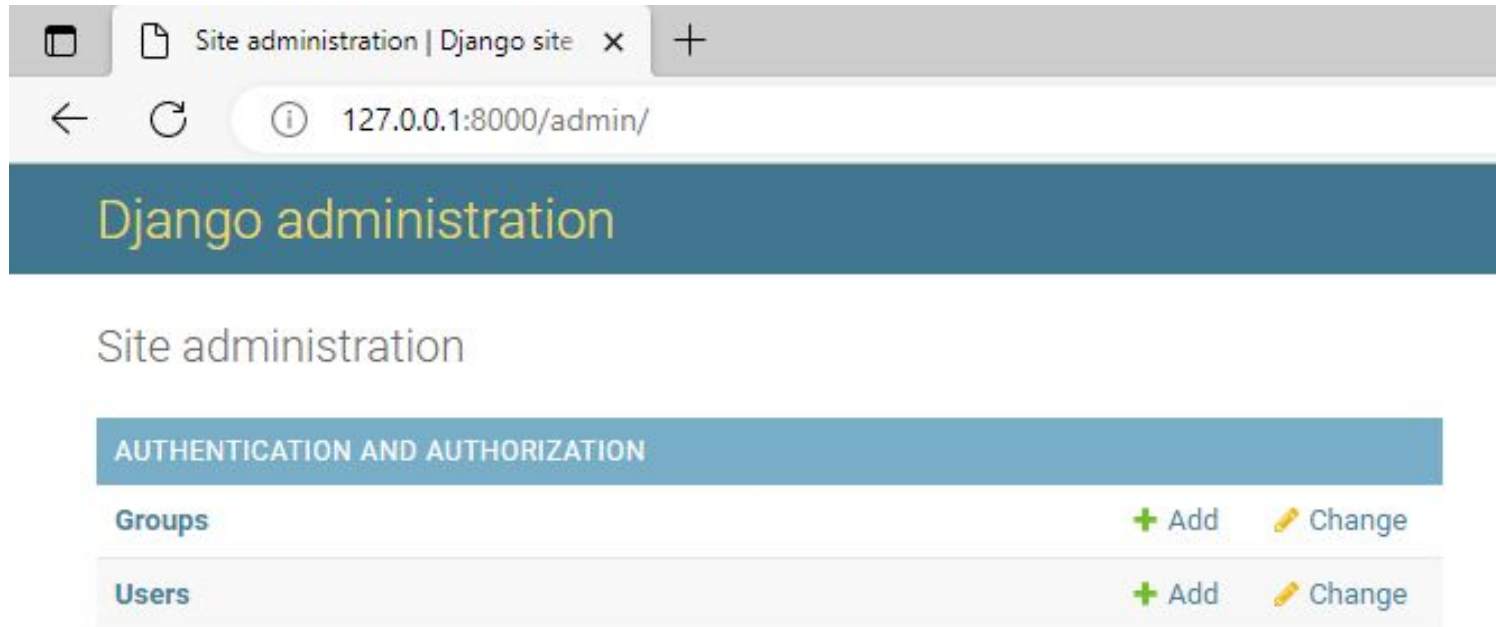
.....

Log in

Create an App



Create an App



Introduction to Django Template System, Load Template Files, Render Templates, Create Forms, Process Form Data and Customize Form Field Validation

Templates

- Django provides a convenient way to generate dynamic HTML pages by using its template system.
- A template consists of **static parts of the desired HTML output** as well as some special syntax describing how dynamic content will be inserted.

Why Templates?

- We are hardcoding HTML code inside our views. At a later date, if we want to modify our HTML it would be very painful to go through each view one by one to modify the page.
- Django comes bundled with a powerful templating system which allows us to create complex HTML pages easily instead of hardcoding them inside views. If we keep hardcoding HTML directly in the view we wouldn't be able to use loops or conditional statements that Django templating system provides inside our HTML
- In the real world, a page consists of many dynamic components. Embedding dynamic content in a large page using `format()` method is very error-prone and tedious.
- At this point, we haven't yet introduced database into the scene because it would create more mess.

Why Templates?

- In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser. We know that HTML is a static markup language, while Python is a dynamic programming language.
- Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.
- Templates not only show static data but also the data from different databases connected to the application through a context dictionary

Templates

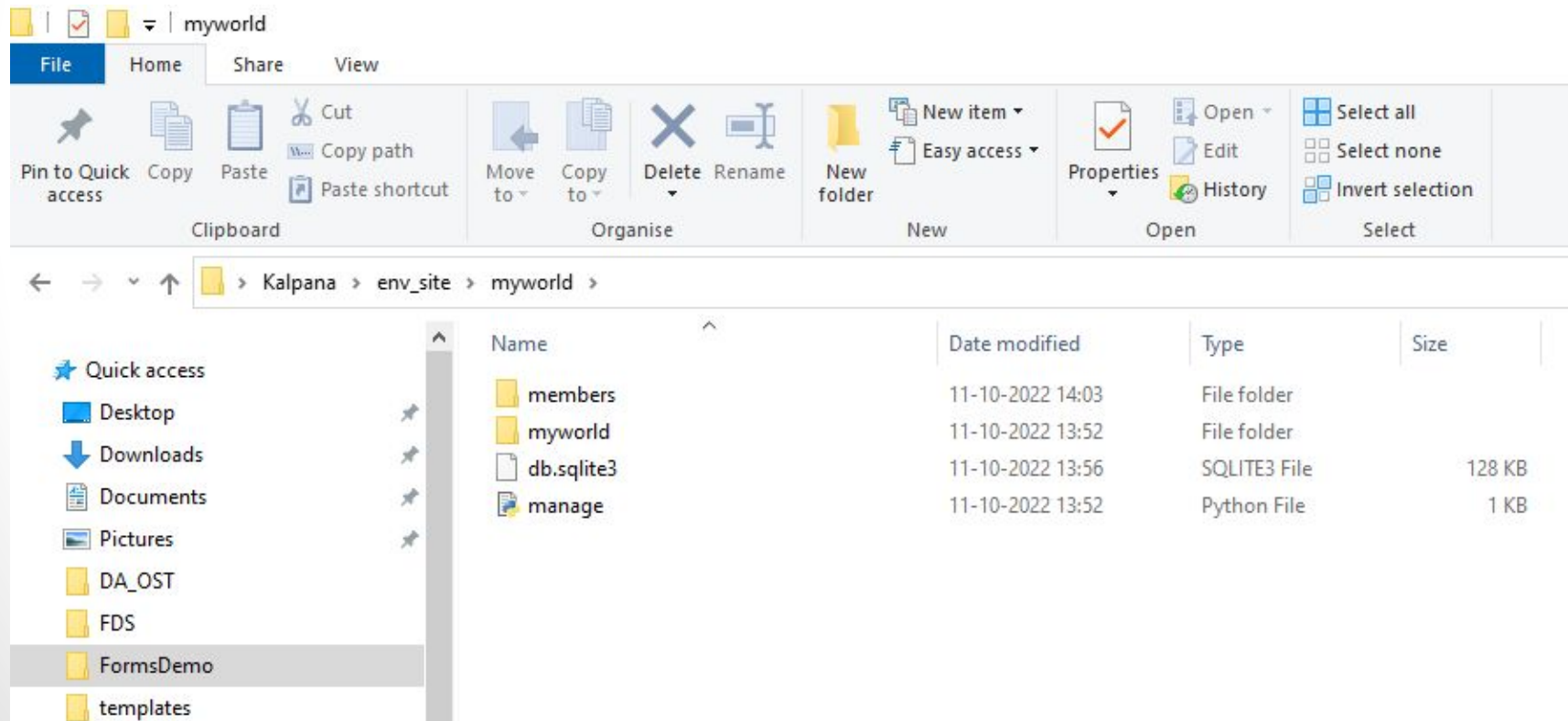
```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    },  
]
```

Simple Template

```
<!DOCTYPE html>
<html> <head> <title>Blog Post</title> </head>
<body>
<h1>{{ post_title }}</h1>
<p>Published by <span>Author : {{ author|title }}</span></p>
<p>{{ post_content }}</p>
<p>Related Posts:</p>
<ul>
{% for item in item_list %}
    <li>{{ item }}</li>
{% endfor %}
</ul>
{% if comments %}
    <p>This post has some comments</p>
{% else %}
    <p>This post has no comments</p>
{% endif %}
</body></html>
```

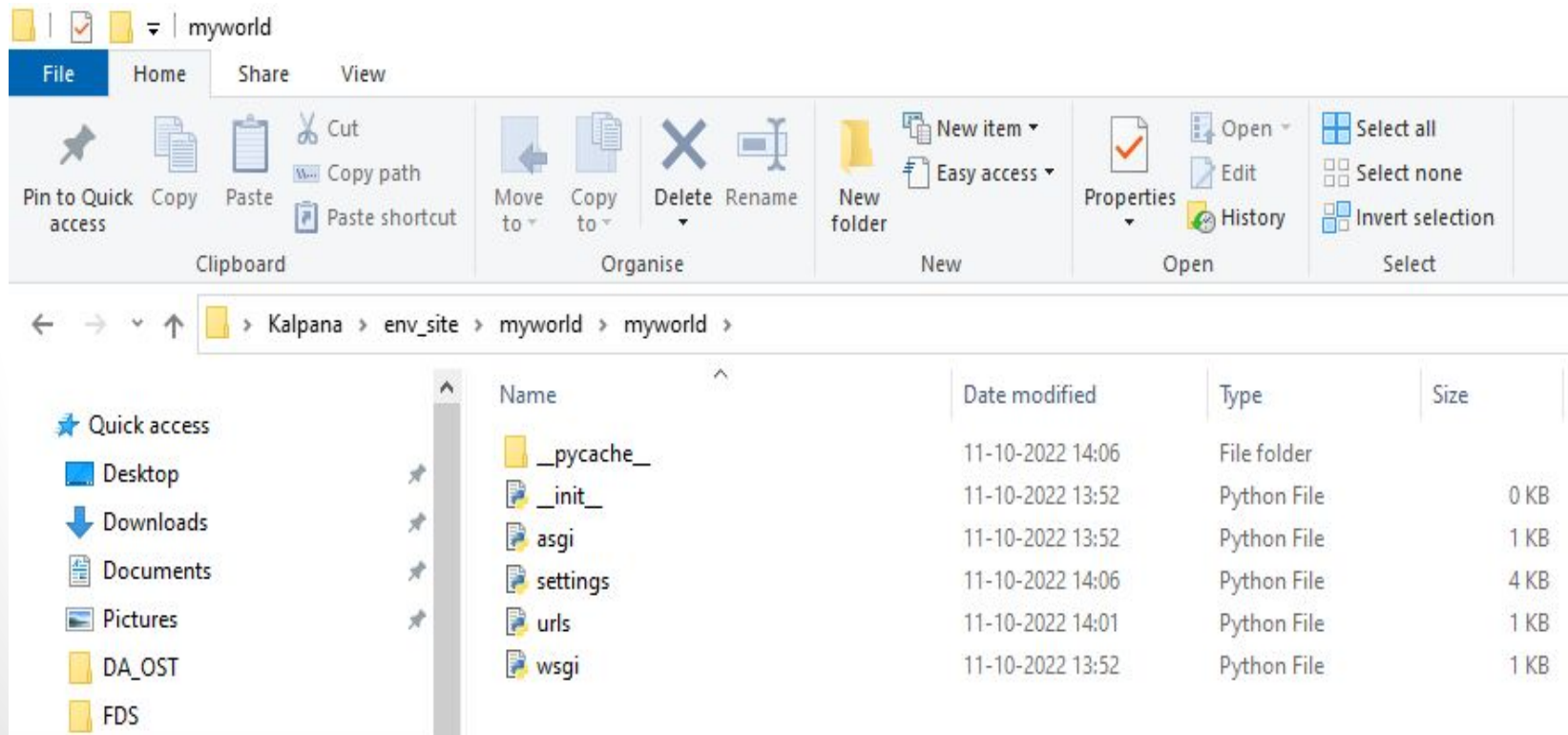
Templates

- Create a **templates** folder inside the **members** folder, and create a HTML file named **myfirst.html**.



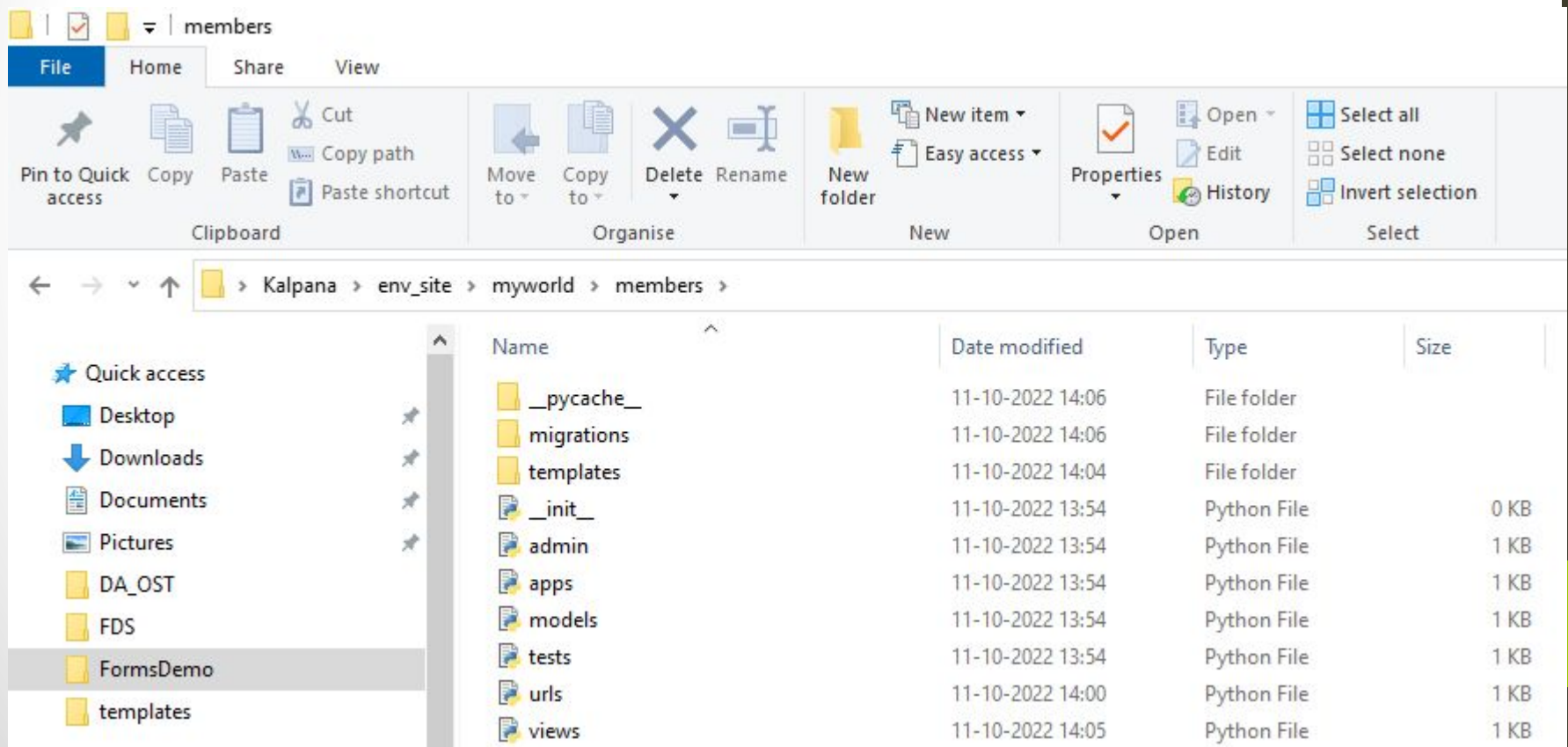
Templates

- Create a **templates** folder inside the **members** folder, and create a HTML file named **myfirst.html**.



Templates

- Create a **templates** folder inside the **members** folder, and create a HTML file named **myfirst.html**.



Templates

members/templates/myfirst.html:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body> <h1>Hello World!</h1>
```

```
<p>Welcome to my first Django project!</p>
```

```
</body>
```

```
</html>
```


Templates

Modify the View

- Open the views.py file and replace the index view with this:

members/views.py:

```
from django.http import HttpResponse
from django.template import loader
def index(request):
    template = loader.get_template('myfirst.html')
    return HttpResponse(template.render())
```

Templates

Change Settings

- To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.
- This is done in the settings.py file in the myworld folder.
- Look up the INSTALLED_APPS[] list and add the members app like this:

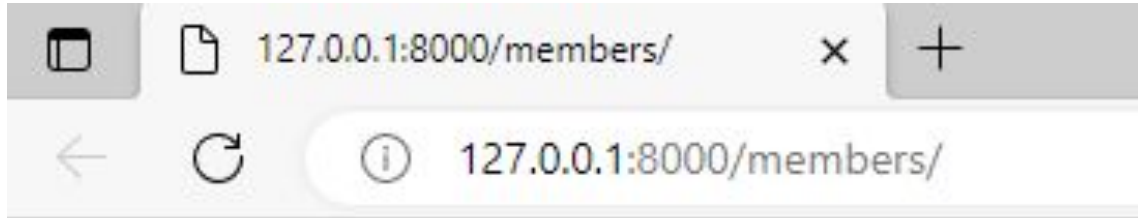
myworld/settings.py:

```
INSTALLED_APPS = [  
'django.contrib.admin',  
'django.contrib.auth',  
'django.contrib.contenttypes',  
'django.contrib.sessions',  
'django.contrib.messages',  
'django.contrib.staticfiles',  
'members.apps.MembersConfig' ]
```

Templates

- Then run this command:
py manage.py migrate
- Start the server by navigating to the /myworld folder and execute this command:
python manage.py runserver
- In the browser window, type 127.0.0.1:8000/members/ in the address bar.

Templates



Hello World!

Welcome to my first Django project!

Loading Templates

#Open views.py under members folder and you will see the following:

```
from django.http import HttpResponse
```

```
import datetime
```

```
def today_is(request):
```

```
    now = datetime.datetime.now()
```

```
    t = template.Template("<html><body>Time is{{now}}</body></html>")
```

```
    c = template.Context({'now': now})
```

```
    html = t.render(c)
```

```
    return HttpResponse(html)
```

Loading Templates

```
def today_is(request):  
    now = datetime.datetime.now()  
    t = template.Template("<html><body>Time is{{now}}</body></html>")  
    c = template.Context({'now': now})  
    html = t.render(c)  
    return HttpResponse(html)
```

Loading Templates

- Start the server and you will get the current date and time
- Create a new file called **datetime.html** inside **newapp/templates/newapp** directory and add the following code to it

Loading Templates

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8"> <title>Current Time</title> </head>
<body>
  {# This is a comment #}
  {# check the existence of now variable in the template using if tag #}
  {% if now %}
    <p>Time is {{ now }}</p>
  {% else %}
    <p>now variable is not available</p>
  {% endif %}
</body>
</html>
```


Loading Templates

#Open views.py under members folder and you will see the following:

```
from django.http import HttpResponse
import datetime
def today_is(request):
    now = datetime.datetime.now()
    t = template.loader.get_template('newapp/datetime.html')
    c = template.Context({'now': now})
    html = t.render(c)
    return HttpResponse(html)
```

Loading Templates



Time is 2022-10-11 08:39:57.967246

Rendering Templates

Most of the time a view does the following task:

- Pull data from the database using models
 - Load the template file and create Template object.
 - Create Context object to supply data to the template.
 - Call `render()` method.
 - Create `HttpResponse()` and send it to the client.
-
- Django provides a function called **`render_to_response()`** to do all things mentioned above from step 2 to 5. It accepts two arguments template name and a dictionary (which will be used to create **Context** object).
 - To use **`render_to_response()`** you must first import it from **`django.shortcuts`** module.

Rendering Templates

```
def today_is(request):  
    now = datetime.datetime.now()  
    return render_to_response('newapp/datetime.html', {'now': now })
```

Rendering Templates

- The **render()** function works similar to **render_to_response()** but it provides some additional variables inside the Django templates (although there are many other subtle differences between **render()** and **render_to_response()** but for now we will not going to discuss them).
- One such variable is **request** which is an object of type **HttpRequest**.
- Recall that every view function accepts **request** object as a first parameter. If you want to access **request** object inside templates you must use **render()** instead of **render_to_response()**.
- At this point, we can't do anything useful with **request** object but just to give you an example, we will try to get the scheme used to access the web page.
- Open **datetime.html** template and make the following changes:

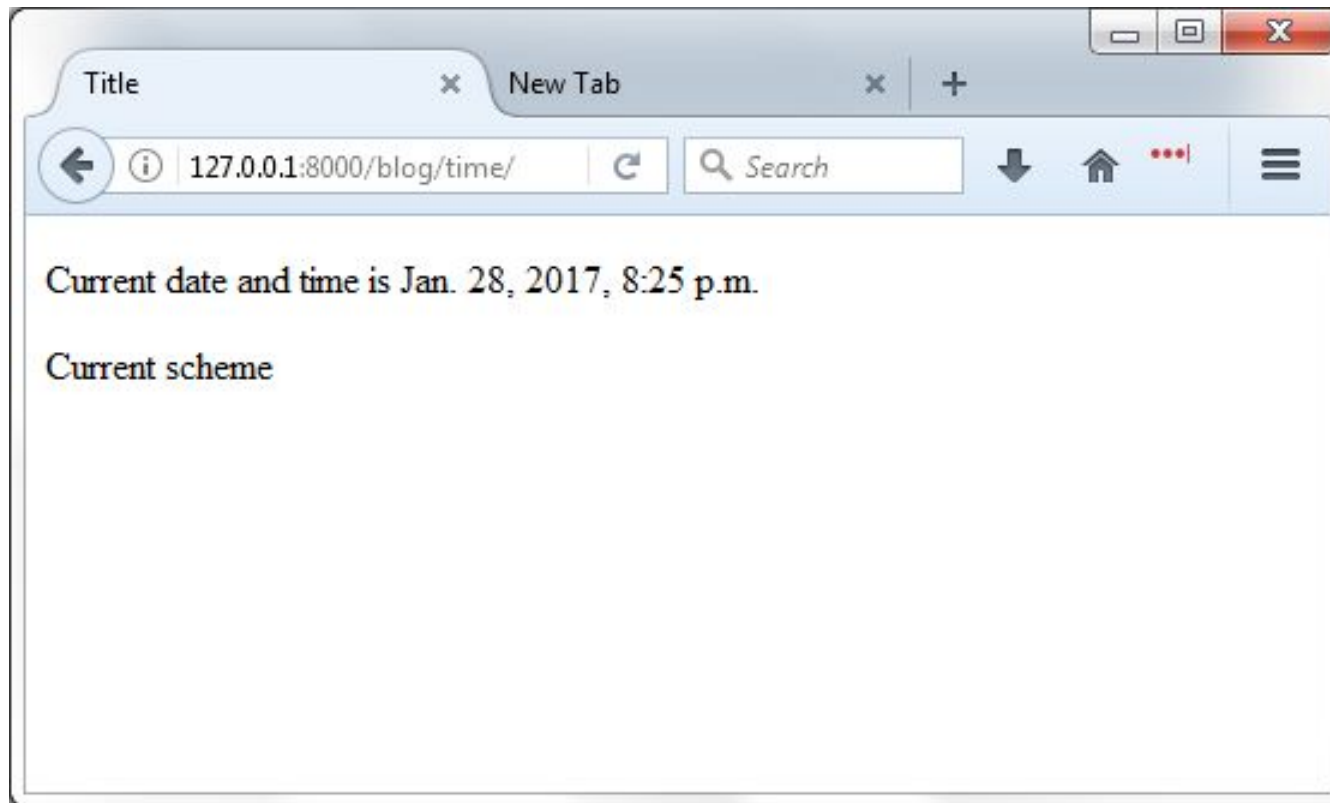
Rendering Templates

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8"> <title>Current Time</title>
</head>
<body>
  {# This is a comment #}
  {# check the existence of now variable in the template using if tag #}
  {% if now %}
    <p>Current date and time is {{ now }}</p>
  {% else %}
    <p>now variable is not available</p>
  {% endif %}
  <p>Current scheme: {{ request.scheme }}</p>
</body> </html>
```

Rendering Templates

- The **request** object has an attribute called **scheme** which returns the scheme of the request.
- In other words, if the page is requested using the **http** then **scheme** is **http**.
- On the other hand, if it is requested using **https** then **scheme** is **https**.

Rendering Templates

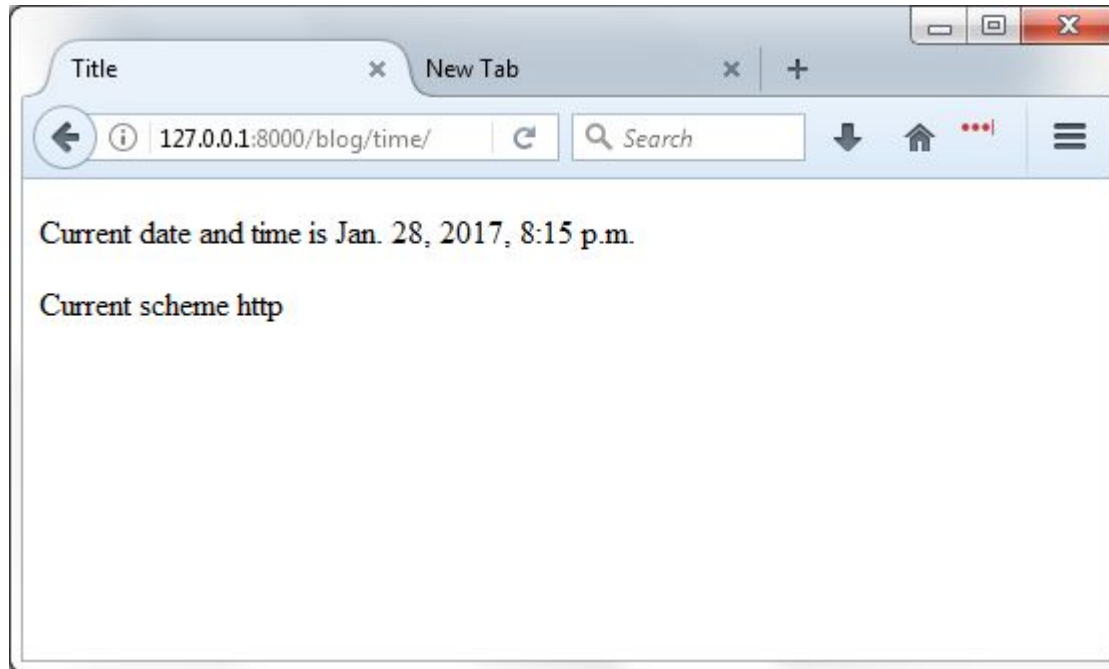


Rendering Templates

- Notice that nothing is printed in the place of **request.scheme** because we are using **render_to_response()**.
- To use **render()** function first import it from **django.shortcuts** module.
- Let's update **today_is()** view function to use **render()** instead of **render_to_response()** function.

```
from django.http import HttpResponse
import datetime from django.shortcuts
import render #...
def today_is(request):
    now = datetime.datetime.now()
    return render(request, 'newapp/datetime.html', {'now': now})
```

Rendering Templates



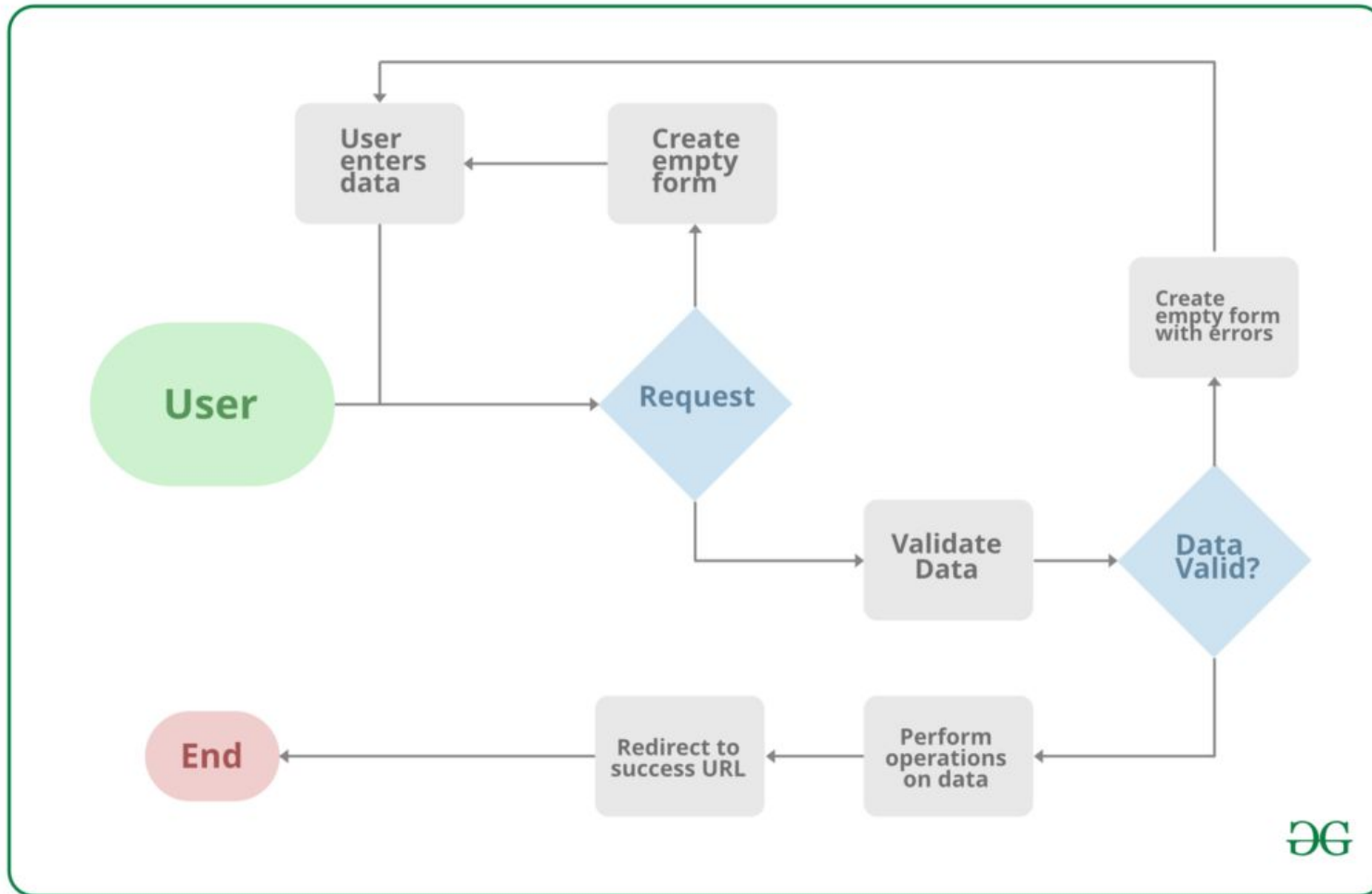
Create Forms

- Django forms are an advanced set of HTML forms that can be created using python and support all features of HTML forms in a pythonic way.
- When one creates a **Form** class, the most important part is defining the fields of the form.
- Each field has custom validation logic, along with a few other hooks.

Create Forms

- Forms are basically used for taking input from the user in some manner and using that information for logical operations on databases. For example, Registering a user by taking input as his name, email, password, etc.
- Django maps the fields defined in Django forms into HTML input fields.
- Django handles three distinct parts of the work involved in forms:
 - **preparing and restructuring data to make it ready for rendering**
 - **creating HTML forms for the data**
 - **receiving and processing submitted forms and data from the client**

Create Forms



Create Forms

- Django forms are an advanced set of HTML forms that can be created using python and support all features of HTML forms in a pythonic way.

```
from django import forms
```

```
# creating a form
```

```
class InputForm(forms.Form):
```

```
    first_name = forms.CharField(max_length = 200)
```

```
    last_name = forms.CharField(max_length = 200)
```

```
    roll_number = forms.IntegerField(  
        help_text = "Enter 6 digit roll number")
```

```
    password = forms.CharField(widget = forms.PasswordInput())
```

Forms

views.py

```
from django.shortcuts import render
```

```
from .forms import InputForm
```

```
# Create your views here.
```

```
def home_view(request):
```

```
    context = {}
```

```
    context['form'] = InputForm()
```

```
    return render(request, "home.html", context)
```

Forms

- **Syntax :**

Field_name = forms.FieldType(attributes)

views.py

```
from django.shortcuts import render  
from .forms import InputForm
```

```
# Create your views here.
```

```
def home_view(request):  
    context = {}  
    context['form'] = InputForm()  
    return render(request, "home.html", context)
```


Forms

templates > home.html

```
<form action = "" method = "post">
    {% csrf_token %}
    {{form }}
    <input type="submit" value=Submit">
</form>
```

- All set to check if the form is working or not let's visit <http://localhost:8000/>
-

Forms

```
<form action = "" method = "post">  
    {% csrf_token %}  
    {{form.as_p }}  
    <input type="submit" value=Submit">  
</form>
```

Forms



A screenshot of a web browser window. The address bar shows 'localhost:8000'. The page content includes a form with the following elements:

- First name:
- Last name:
- Roll number:
- Enter 6 digit roll number Password:
-

Form Validation

```
models.py – (Project folder)
from django.db import models
class User(models.Model):
    # username field
    username = models.CharField(max_length=30, blank=False, null=False)
    # password field
    password = models.CharField(max_length=8, blank=False, null=False)
```

Form Validation

- Now, we have to migrate the models. To make migrations, run the following command.

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Form validation

#forms.py

```
from django.forms import ModelForm
from django import forms
```

```
from validation.models import User
```

```
class UserForm(ModelForm):
```

```
    # meta data for displaying a form
```

```
    class Meta:
```

```
        # model
```

```
        model = User
```

```
        # displaying fields
```

```
        fields = '__all__'
```

```
    # method for cleaning the data
```

```
    def clean(self):
```

```
        super(UserForm, self).clean()
```

```
        # getting username and password from cleaned_data
```

```
        username = self.cleaned_data.get('username')
```

```
        password = self.cleaned_data.get('password')
```

```
        # validating the username and password
```

```
        if len(username) < 5:
```

```
            self._errors['username'] = self.error_class(['A minimum of 5 characters is required'])
```

```
        if len(password) < 8:
```

```
            self._errors['password'] = self.error_class(['Password length should not be < 8 characters'])
```

```
        return self.cleaned_data
```

Form Validation

- Create a templates folder and a template called home.html inside the app. And paste the following code inside the home template.

```
{% load crispy_forms_tags %}
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8" />
<meta name="viewport" content="width=device-width,
initial-scale=1.0" /> <title>Form Validation</title> <link
rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bo
otstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh
0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"
/> </head>
```

Form Validation

```
<body>
<div class="container">
<div class="row">
<div class="col-md-4 col-md-offset-4">
<h2>User</h2>
<form action="" method="post"> {%csrf_token%}
{{ form|crispy }}
<div class="form-group">
<button type="submit" class="btn btn-success"> Add User
</button> </div> </form>
</div>
</div>
</div>
</body> </html>
```


Form Validation

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import UserForm
# Create your views here.
def home_view(request):
    # cheking the request
    if request.method == 'POST':
        # passing the form data to LoginForm
        user_details = UserForm(request.POST)
        # validating the user_details with is_valid() method
        if user_details.is_valid():
            # writing data to the database
            user_details.save()
```

Form Validation

```
# redirect to another page with success message
return HttpResponseRedirect("Data submitted successfully")
else:
    # redirect back to the user page with errors
    return render(request, 'validation/home.html',
{'form':user_details})
else:
    # in case of GET request
    form = UserForm(None)
    return render(request, 'validation/home.html',
{'form':form})
```

Form Validation

```
from django.contrib import admin
from django.urls import path
from validation.views import home_view
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home_view, name='home'),
]
```

Form Validation

User

Username*

Password*

Add User

User

Username*

A minimum of 5 characters is required

Password*

Password length should not be less than 8 characters

Add User