Unit_03 - Django Frameworks

Introduction

- Django is a back-end server side web framework and it is free, open source and written in Python.
- A web framework is a software that supports the development of dynamic websites, applications, and services. It provides a set of tools and functionalities that solves many common problems associated with web development, such as security features, database access, sessions, template processing, URL routing, internationalization, localization, and much more.

Django Architecture

- Django web framework follows the MVT (Model View Template) architecture.
- This architectural pattern is designed for easy and rapid web development.

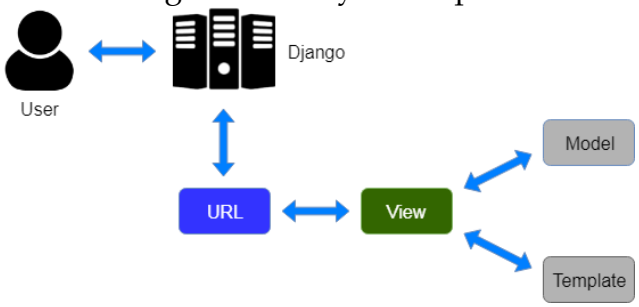


Fig: MVT based control flow

Model

- The model provides data from the database.
- In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.
- The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.
- Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

View

- A request handler that returns the relevant template and content - based on the request from the user.
- A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

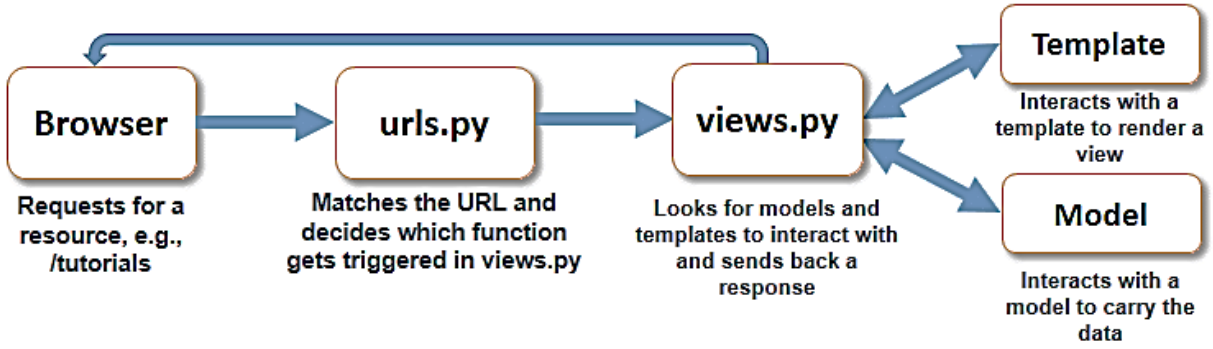
Template

- A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

- The Template is a presentation layer which handles User Interface part completely.
- A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Communication of components in MVT Architecture:

- First, a user requests for a resource. Django considers the request as a URL and matches it with the existing URL paths in the urls.py file.
- This process of matching the user-requested URL to the one in urls.py is known as URL mapping. Once the URL matches, Django carries out the further process.
- Once the URL is mapped, Django jumps to the views.py folder and calls a view.
- The triggered view looks for models and templates to interact with, and then it returns the response back to the user. Here, the model deals with the data associated with the user request.
- On the other hand, the template deals with the HTML and the static files, such as CSS files, JS files, images, etc., that are required to complete the view.



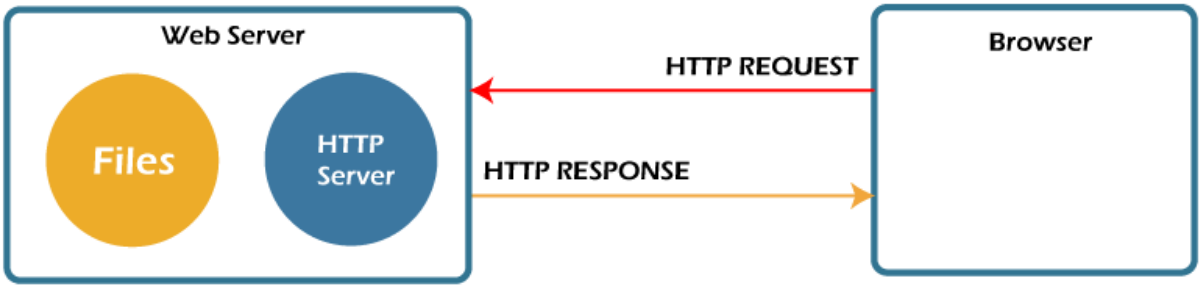
Features of Django

- **Rapid Development:** Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.
- **Secure:** Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.
- **Scalable:** Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.
- **Fully loaded:** Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.
- **Versatile:** Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

- **Open Source:** Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.
- **Vast and Supported Community:** Django is a one of the most popular web framework. It has widely supportive community and channels to share and connect.

Web Server

- A web server is software and hardware that uses HTTP (Hypertext-Transfer-Protocol) and other protocols to respond to client requests made over the World Wide Web. The main job of a web server is to display website content through storing, processing and delivering webpages to users.
- However, web servers can serve static as well as dynamic contents. Web Servers also assists in emailing services and storing files. Therefore it also uses SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) protocols to support the respective services. Web servers are mainly used in web hosting or hosting the website's data and running web-based applications.
- The hardware of the web servers are connected to the Internet that manages the data exchange facility within different connected devices. In contrast, the software of web server software is responsible for controlling how a user accesses delivered files. Typically, web server management is an ideal example of the client/server model. Therefore, it is compulsory for all computers that host websites (whether with state or dynamic web page content) to have web server software.
- At the most basic level, whenever a browser needs a file that is hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct (hardware) web server, the (software) HTTP server accepts the request, finds the requested document, and sends it back to the browser, also through HTTP. (If the server doesn't find the requested document, it returns a 404 response instead.)



To publish a website, you need either a static or a dynamic web server.

- A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files as-is to your browser.
- A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database. We call it "dynamic" because the

application server updates the hosted files before sending content to your browser via the HTTP server.

How do web servers work?

The term web server can denote server hardware or server software, or in most cases, both hardware and software might be working together.

1. ***On the hardware side***, a web server is defined as a computer that stores software and another website raw data, such as HTML files, images, text documents, and JavaScript files. The hardware of the web servers are connected to the web and supports the data exchange with different devices connected to the Internet.
2. ***On the software side***, a web server includes server software accessed through website domain names. It controls how web users access the web files and ensures the supply of website content to the end-user. The web server contains several components, including an HTTP server.

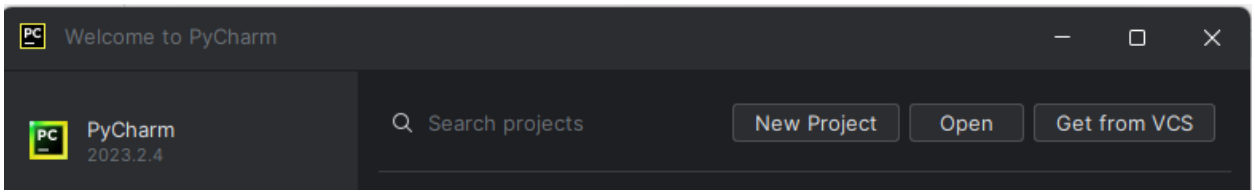
Whenever any web browser, such as Google Chrome, Microsoft Edge or Firefox, requests for a web page hosted on a web server, the browser will process the request forward with the help of HTTP. At the server end, when it receives the request, the HTTP server will accept the request and immediately start looking for the requested data and forwards it back to the web browser via HTTP.

Let's discover the step-by-step process of what happens whenever a web browser approaches the web server and requests a web file or file. Follow the below steps:

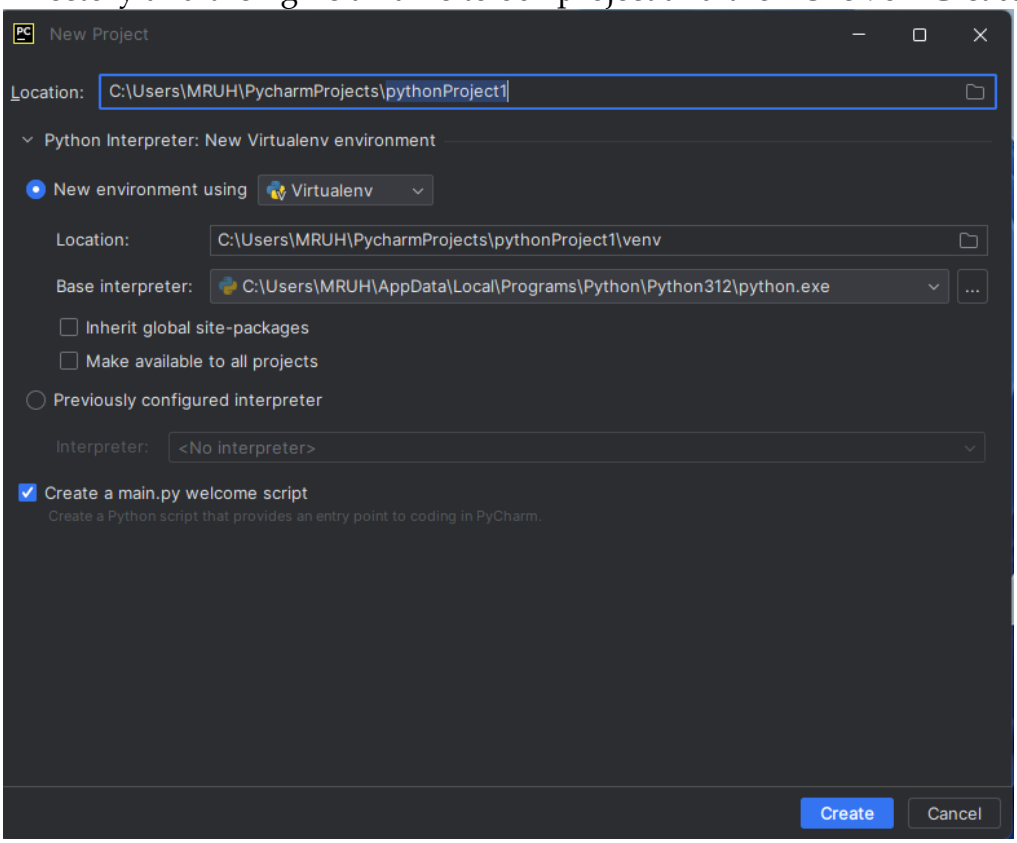
1. First, any web user is required to **type the URL of the web page in the address bar** of your web browser.
2. With the help of the URL, your **web browser will fetch the IP address of your domain** name either by converting the URL via DNS (Domain Name System) or by looking for the IP in cache memory. The IP address will direct your browser to the web server.
3. After making the connection, the **web browser will request for the web page from the web server** with the help of an HTTP request.
4. As soon as the web server receives this request, it immediately **responds by sending back the requested page** or file to the web browser HTTP.
5. If the web page requested by the **browser does not exist or if there occurs some error in the process**, the web server will return an error message.
6. If there occurs no error, the browser will successfully display the webpage.

Installing & Configuring Django Components

- Django is a Python web framework, thus requiring Python to be installed on machine.
- We can download python from the following website: https://www.python.org/
- Download the executable installer and run it. Check the boxes next to “Install launcher for all users (recommended)” then click “Install Now”.
- Open PyCharm and Click on **New Project**.



- Select Directory and then give a name to our project and then Click on **Create**.



- Open the terminal in pycharm and type the following command to install Django

  `pip install django`

  This command fetches the django package from the Python Package Index (PyPI) using pip. After the installation has completed, you can pin your dependencies to make sure that you're keeping track of which Django version you installed:
- Type the following Command in pycharm Terminal to check the Django version.

  `python -m django –version`

- Start a project by using following command-

  `django-admin startproject Myproject`

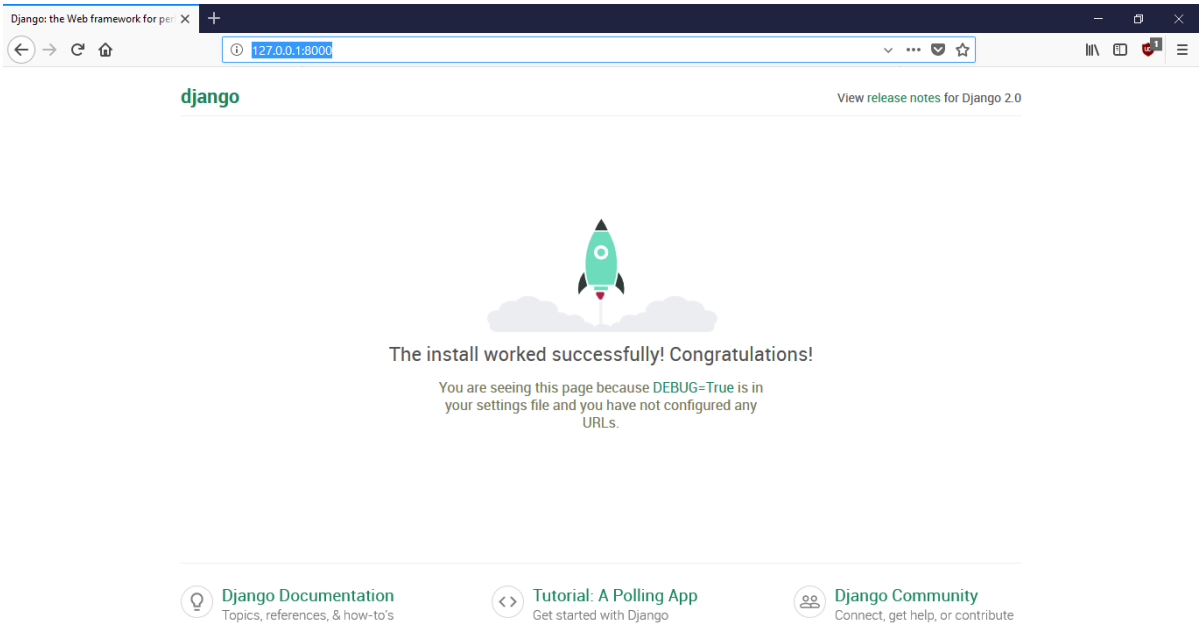
- Change directory to Myproject.

  `cd Myproject`

- Start the server by typing the command

  `python manage.py runserver`

  By clicking the link http://127.0.0.1:8000/ we can see the result.



Django Hello World Application

Step 1: Open PyCharm and Click on **Create New Project**.
Step 2: Select Directory and then give a name to our project and then Click on **Create**.
Step 3: Then Check if Django is installed or not in your Computer.
Type the following Command in pycharm Terminal located at Bottom Left.:

```
python -m django --version
```

If it is Already installed then we can see the Django version installed in our Computer.

Step 4: Start a project by using following command-

```
django-admin startproject Myproject
```

Step 5: Change directory to Myproject.

```
cd Myproject
```

Step 6: Create the application by using following command

```
Python manage.py startapp myapp
```

Create views.py and write the following code:

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

```
def myview(request):
   return HttpResponse("Hello world!")
```

Create the urls.py and write the following code

```
from django.urls import path

from .views import myview

urlpatterns = [
   path("",myview, name="home"),
]
```

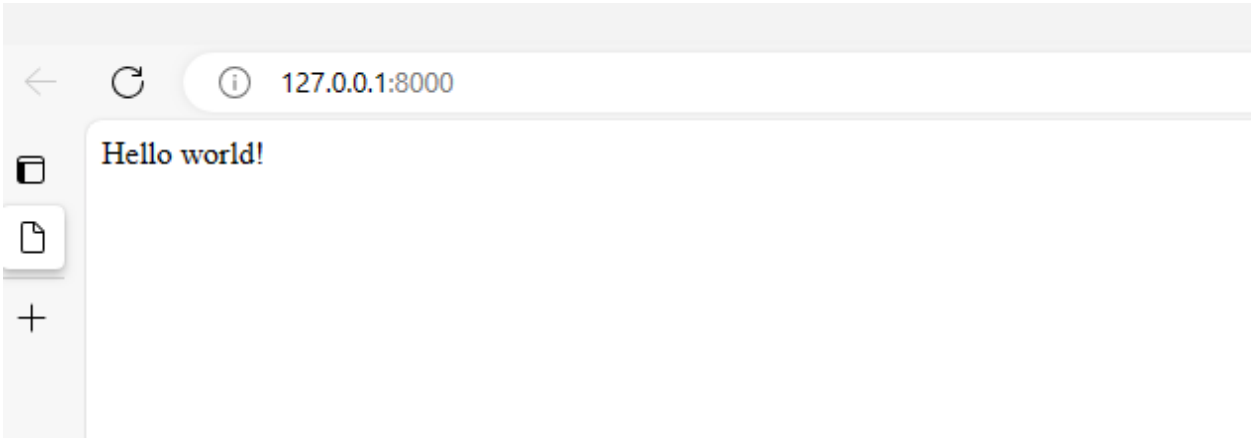
Update the Myproject urls.py with the following code,

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

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

Step 7: Start the server by typing following command in **cmd–**

```
python manage.py runserver
```

By clicking the link http://127.0.0.1:8000/ we can see the result.



Creating views and Mapping URLs in Django

A view function, or "view" for short, is simply 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, etc.

views.py (in application directory)

```
from django.http import HttpResponse

def myview(request):
   text = """<h1>welcome to my app !</h1>"""
   return HttpResponse(text)
```

- First, we import the class HttpResponse from the django.http module
- Next, we define a function called **myview**. This is the view function. Each view function takes an HttpRequest object as its first parameter, which is typically named request.
- The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

Create a file urls.py

To access that view via a URL. Django has his own way for URL mapping and it's done by editing your project url.py file.

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

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

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.

Then create urls.py in application directory and map views

```
from django.urls import path

# importing views from views.py
from .views import myview

urlpatterns = [
    path('', myview),
]
```

- When a user makes a request for a page on your web app, Django controller takes over to look for the corresponding view via the url.py file, and then return the HTML response or a 404 not found error, if not found.
- In url.py, the most important thing is the "urlpatterns" tuple. It's where you define the mapping between URLs and views.

url mapping is composed of three elements −

- The pattern − A regexp matching the URL you want to be resolved and map. Everything that can work with the python 're' module is eligible for the pattern.
- The python path to the view − Same as when you are importing a module.
- The name − In order to perform URL reversing, you'll need to use named URL patterns as done in the examples above. Once done, just start the server to access your view via http://127.0.0.1/8000.

Template

- Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates.
- A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.
- A Django project can be configured with one or several template engines.
- Django ships built-in backends for its own template system, creatively called the Django template language (DTL), and for the popular alternative Jinja2.
- Django defines a standard API for loading and rendering templates regardless of the backend.
- Loading consists of finding the template for a given identifier and preprocessing it, usually compiling it to an in-memory representation.
- Rendering means interpolating the template with context data and returning the resulting string.

Create a template folder inside the project directory, and create a HTML file named myfirst.html.
Open the HTML file and insert the following:

```
<!DOCTYPE html>
<html>
<body>

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

Open the ***views.py*** file and replace the view with the following updates

```
from django.http import HttpResponse
from django.template import loader

def myview(request):
  template = loader.get_template('myfirst.html')
  return HttpResponse(template.render())
```

Update the Settings.py

Open the settings.py in project directory.

Look up the INSTALLED_APPS[ ] list and add the our app.

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

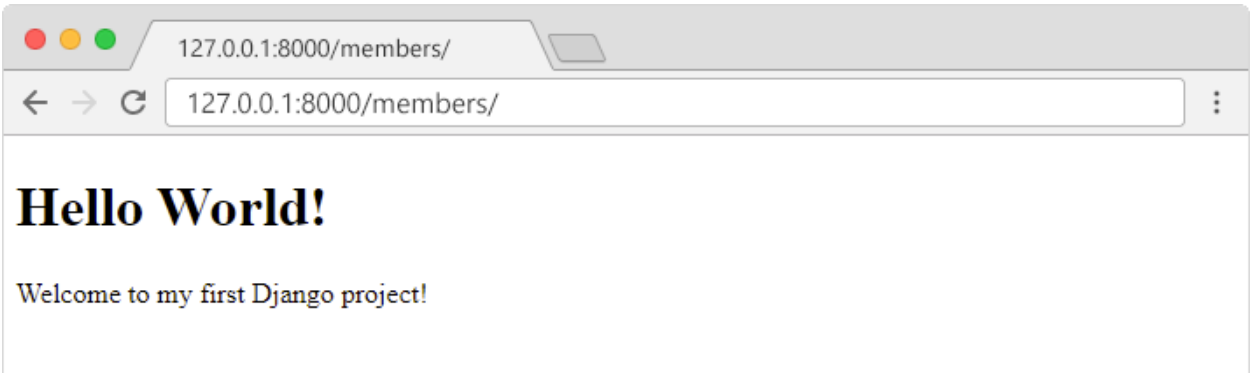
Find the DIRS[ ] and update with the following code,

```
'DIRS': [os.path.join(BASE_DIR, 'template')],
```

Start the server by typing following command in **cmd–**

```
python manage.py runserver
```

By clicking the link http://127.0.0.1:8000/ we can see the result.



Rendering dynamic content in templates

- Django has Template System for Display Content on Web.
- Using Views and Django Template we can display data in the template.
- There are lots of Django Template Tags and Filter for displaying data dynamically.
- We can pass the Model data to view for rendering it in the template.

Here the context dictionary will be key- value pairs that are passed to the template being rendered as variables. Each key becomes the variable name in the template, and you can access the value in the template by referencing the key name. The snippet below populates the context variable with some dynamic data.

views.py

```
from django.shortcuts import render

def myview(request):
    cname = 'Python'
    duration = '4 Months'
    seats = 5
    course_details = {'nm': cname, 'du': duration, 'st':
seats}
    return render(request,'course.html',course_details)
```

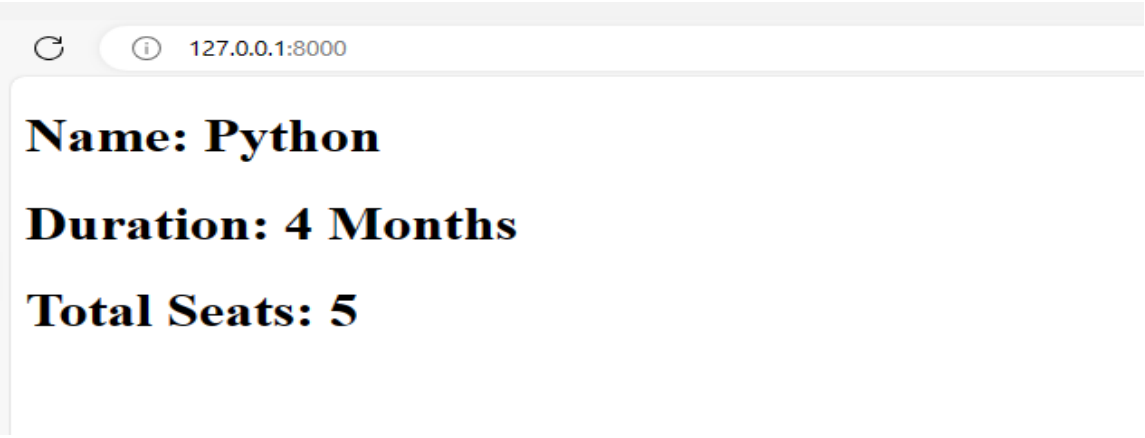
- Variables in Django Templates are referenced using the special syntax of two curly braces, followed by the variable name, followed by two additional curly braces.
- It looks like this:
  {{ variable }}
- This is not standard HTML but is picked up by Django as it parses and evaluates the template file. When the template engine finds a variable, it evaluates that variable and replaces it with the value contained in the context.

- Variable names can be any combination of alphanumeric characters and the underscore ("_") but may not start with an underscore, and may not be a number. You also cannot have spaces or punctuation characters in variable names.

Create a file course.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewpoint" content="width=device-
width,initial-scale=1.0">
    <title>Django</title>
</head>
<body>
    <h1>Name: {{nm}}</h1>
    <h1>Duration: {{du}}</h1>
    <h1>Total Seats: {{st}}</h1>
</body>
</html>
```

The result is that we now have dynamic data getting loaded into the template for each page view.



Template inheritance

Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override. Let's look at template inheritance by starting with an example:

Create a View Function

First, create a view function that renders the templates. In this case, you will render the base.html template. Import the render method from Django shortcuts. Then create a view function named index that returns and renders the index template.

Views.py

```
from django.shortcuts import render

from django.http import HttpResponse
from django.template import loader


def myview(request):
   template = loader.get_template('child.html')
   return HttpResponse(template.render())
```

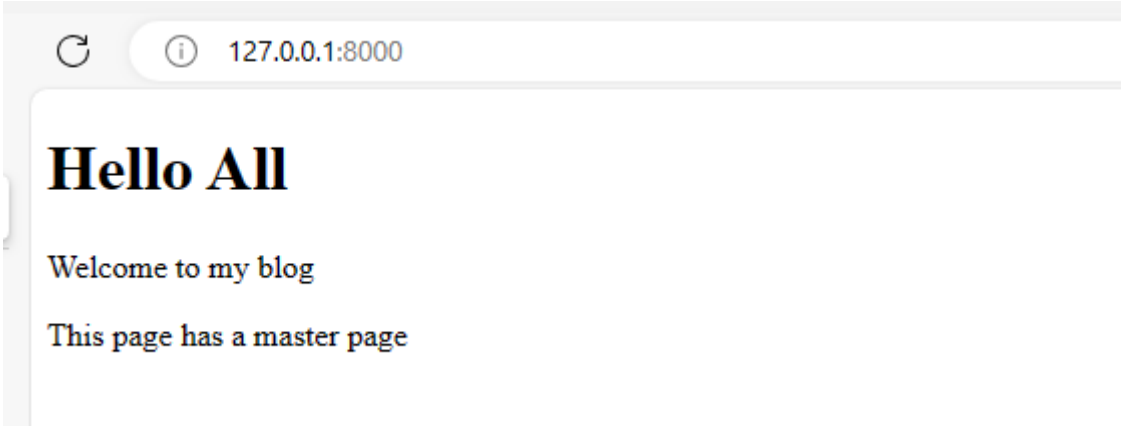
base.html

```
<html>
<!DOCTYPE html>
<html>
<body>

<h1>Hello All</h1>
<p>Welcome to my blog</p>

{% block mymessage %}
{% endblock %}

</body>
</html>
```

- This template, which we'll call **base.html**, defines an HTML skeleton document that you might use for a two-column page. It's the job of "child" templates to fill the empty blocks with content.
- In this example, the **block** tag defines three blocks that child templates can fill in. All the **block** tag does is to tell the template engine that a child template may override those portions of the template.
- A child template might look like this:

```
{% extends 'base.html' %}

{% block mymessage %}
  <p>This page has a master page</p>
{% endblock %}
```

- The "**extends"** tag is the key here. It tells the template engine that this template "extends" another template. When the template system evaluates this template, first it locates the parent – in this case, "base.html".
- At that point, the template engine will notice the three **block** tags in **base.html** and replace those blocks with the contents of the child template. Depending on the value of **blog_entries**, the output might look like:



DRY (don't repeat yourself) principle

- The DRY (don't repeat yourself) principle is a best practice in software development that recommends software engineers to do something once, and only once.
- According to the DRY principle, every discrete chunk of knowledge should have one, unambiguous, authoritative representation within a system. The goal of the DRY principle is to lower technical debt by eliminating redundancies in process and logic whenever possible.

- ***Redundancies in process***

  To prevent redundancies in processes (actions required to achieve a result), followers of the DRY principle seek to ensure that there is only one way to complete a particular process. Automating the steps wherever possible also reduces redundancy, as well as the number of actions required to complete a task.
- ***Redundancies in logic***

  To prevent redundancies in logic (code), followers of the DRY principle use abstraction to minimize repetition. Abstraction is the process of removing characteristics until only the most essential characteristics remain.

- An important goal of the DRY principle is to improve the maintainability of code during all phases of its lifecycle. When the DRY principle is followed, for example, a software developer should be able to change code in one place, and have the change automatically applied to every instance of the code in question.

- In Django, DRY stands for "Don't Repeat Yourself," which is a software development principle that promotes code reuse and reduces duplication. The DRY principle emphasizes the importance of writing modular, reusable code and avoiding repetition wherever possible.
- Django provides several mechanisms to help developers follow this principle:
  - **Django's URL dispatcher**: It allows you to define URL patterns and map them to corresponding views. By using named URLs and including URL patterns from other URL configurations, you can avoid hardcoding URLs in multiple places, promoting code reuse.
  - **Template system:** Django's template language supports template inheritance, allowing you to define base templates that contain common elements shared across multiple pages. Child templates can then extend the base template and provide specific content for each page, reducing the need for duplicate HTML code.
  - **Model inheritance:** Django's object-relational mapping (ORM) supports model inheritance, enabling you to define common fields and behaviors in a base model and derive specialized models from it. This approach helps avoid duplicating code and allows for code reuse and extensibility.
  - **Middleware:** Django's middleware framework lets you define reusable components that process requests and responses. Middleware can be used to handle common tasks such as authentication, session management, and caching, eliminating the need to repeat the same logic in multiple views.

Review Questions

1. Explain Django framework with architecture.
2. List and explain the features of Django framework.
3. Define and explain the working process of webservers.
4. Write the procedure to install and configure the Django components.
5. Create the hello world application using Django. Explain with relevant example.
6. Write the procedure to create views.
7. Write short note on Django URL mapping.
8. Explain in details about Django templates with example.
9. Discuss about rendering dynamic content in templates.
10. Outline the DRY principles in Django.
11. Explain in detail about DRY template inheritance in Django.