

**MODULE-1:****MVC BASED WEB DESIGNING****INTRODUCTION****□ DEFINITION OF FULL STACK WEB DEVELOPMENT**

The process of planning, building, testing, and launching an entire web application from beginning to end is known as full stack development. Full stack development is the combination of both the front end (client side) and back end (server side) portions of a web application.

**□ CLIENT SIDE (FRONT END)**

The front end is also known as the client side; it is a part of the website that the user sees and interacts with. It consists of a user interface (UI) of the website, and it runs on the user's local system, instead of where it is hosted.

**□ SERVER SIDE (BACK END)**

The back end is also known as the server side; it is responsible for all the logic and functions of any website. It includes things like database routing and API creation. The back end ensures that the server is up all the time and also manages the incoming traffic to the website.

Full stack developers require different skills and tools to develop the front end and back end of any web application. Let's get into the role of a full stack developer.

**□ WHO IS A FULL STACK DEVELOPER?**

A full stack developer is an engineer who is proficient in working with both server-side and client-side programming software. They manage everything from front end development languages, back-end development methodologies, server handling, and Application Programming Interface (API) designing to working with version control systems.

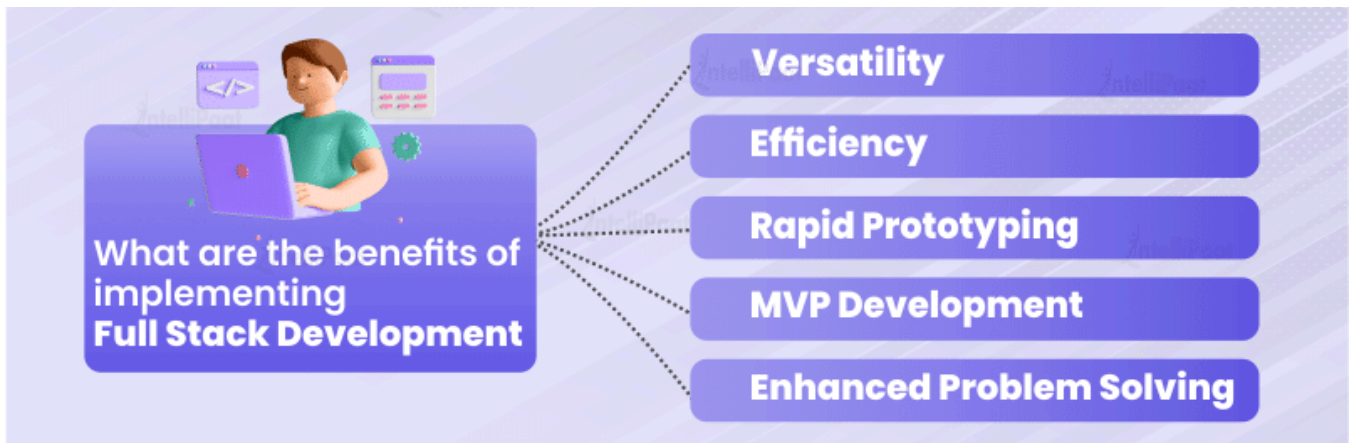
**□ WHY WOULD YOU NEED A FULL STACK DEVELOPER?**

Following are the multiple reasons why organizations require a Full Stack Developer:

- Full Stack developers ensure to keep the end-to-end web application running without any hiccups.

- Full stack developers can fit into multiple roles in the application development process, thereby greatly reducing the costs and time required to solve problems.
- Full Stack developers can actively debug applications alongside development and also extend help in testing and actively developing contingency protocols for the application.

#### □ ADVANTAGES OF FULL STACK DEVELOPMENT



##### 1. Versatility and Efficiency

One primary benefit of having full stack development is its versatility. A full stack developer is proficient in multiple programming languages and frameworks, which helps them handle multiple aspects of a project. A single developer can manage the front end as well as the back end, making the development process seamless and efficient.

##### 2. Rapid Prototyping and MVP Development

Prototyping and the minimum viable product (MVP) are easy to achieve when we use full stack development. Full stack developers have a wide range of skills that enable them to rapidly produce end-to-end working web applications.

##### 3. Enhanced Problem Solving

Full stack developers have a broad understanding of the entire tech stack used to build the whole application, which makes it easy to find and resolve bugs.

#### □ FULL STACK DEVELOPMENT TECHNOLOGIES

- A full stack developer needs to know certain tools and technologies based on the application they are creating. Mainly, the Full Stack Development Technologies are divided into two parts: one for front end creation and the other for the back end.
- The front end majorly focuses on the design part of the website. There are some very popular front end programming languages to build the UI for websites.
  - HTML (Hyper Text Markup Language): It is used to create the structure or skeleton of any website. By using HTML, you can add data and visual content to your website.

- o CSS (Cascading Style Sheets): This is used for styling the HTML components to make them more attractive and create a more appealing layout by adding colors to them.
- o JavaScript: Javascript can be used as a front end and back end programming language. In the front end, we use it to make dynamic and interactive components such as a responsive navbar that turns into a hamburger menu when the screen size changes.

## TECHNOLOGIES TO DEVELOP BACK END



- The back end handles the data transfer between the front end and the database or server; it connects the webpage to the main server and establishes the connection between them. The most commonly used back end development languages are PHP, Java, Python, and Node.js.
- The back end can be further divided into 3 sub-layers:
  - o **API Layer:** It is responsible for connecting the back end to the front end.
  - o **Storage Layer:** It stores all the data coming from the front end; it also manages access to data based on the conditions provided by the developer.
  - o **Logic Layer:** This layer consists of all the logic for the website. It is the most important part because it ensures the working of any website.

## □ SKILLS REQUIRED TO BECOME A FULL STACK DEVELOPER

- A full stack developer requires a specialty in computer science and should have a high level of proficiency in both back end and front end programming languages and frameworks. They are skilled in PHP, Django, Node.js, Express.js, JavaScript, and HTML.
- Full stack developers are also knowledgeable about a wide range of DBMS (database management systems), including MongoDB, PostgreSQL, and MySQL.

## □ WHAT ARE THE POPULAR FRAMEWORKS FOR FULL STACK DEVELOPMENT?



There are more than 100 full stack frameworks available out there, but you don't need to learn all of them to become a full stack developer.

Here are the top 5 full stack frameworks:

- **Node.js and Express.js (Javascript frameworks)**

Node.js and Express are the two most popular frameworks for full stack development. According to W3Tech, around 3.1% of websites are developed using Node.js, which is around 6.3 million websites.

Node.js is a runtime Javascript framework that runs on the server side and Express.js works on top of node.js, providing robust features for building websites. It provides simplicity, flexibility, and cross-platform functionality, and it also provides great community support.

- **Ruby on Rails**

Ruby on Rails offers many features, like continuous updates and a vast open-source library. Apart from that, it is very easy to learn and implement. Some of the most popular websites are built using Ruby on Rails, which includes Github, Airbnb, Shopify, and many more. It also offers security, and fast processing, and the architecture is based on a model view controller, making it one of the best full stack development frameworks.

- **Django**

One of the greatest frameworks for full stack web development is Django, which is used by companies like Google, Instagram, YouTube, and NASA. It enables programmers to rapidly construct online apps without having to worry about tasks like creating HTML templates or database management. A template engine for producing HTML views, an object-relational mapper (ORM) for communicating with databases, and a multitude of tools and libraries for common tasks are all included in Django.

- **Spring Boot**

Spring boot is one of the most flexible and compatible Java frameworks. It helps in developing a production-ready application, which means that you can directly deploy the application without worrying about bugs and errors.

Another benefit of using spring-boot is that it offers an extensive range of integrations and customizations, so you can make it work exactly how you want. Spring Boot is a great option, whether you're searching for an easy approach to building web applications that are ready for production or need a very configurable framework.

- **Laravel**

Laravel is a thoroughly documented PHP web application framework that offers a clean, intelligent syntax that helps you create web applications quickly and easily. Larvel comes with lots of functionalities, including ORM (Object Relational Mapping), routing, and authentication. Larvel also offers a huge collection of libraries and built-in methods that are useful in maintaining and developing web applications.

### **WHAT IS WEB FRAMEWORK?**

A web framework is a software tool that provides a way to build and run web applications. As a result, you don't need to write code on your own and waste time looking for possible miscalculations and bugs.

In the early days of web development, all applications were hand-coded, and only the developer of a certain app could change or deploy it. Web frameworks introduced a simple way out of this trap. Since 1995, all the hassle connected with changing an application's structure has been put in order because of the appearance of a general performance. And that's when web-specific languages appeared. Their variety is now working well for both static and dynamic web pages.

A Web framework provides a programming infrastructure for your applications, so that you can focus on writing clean, maintainable code without having to reinvent the wheel. In a nutshell, that's what Django does.



### **WEB FRAMEWORKS: FEATURES AND ARCHITECTURE**

#### **ARCHITECTURE**

The architecture of almost all most popular web development frameworks is based on the decomposition of several separate layers (applications, modules, etc), which means that you can extend functionality according to your requirements and integrate your changes with framework code, or use third-party applications designed by external vendors. This flexibility is another key benefit of frameworks. There are a lot of open-source communities and commercial organizations that produce applications or extensions for popular frameworks e.g., Django REST Framework, ng-bootstrap, etc.).

The MVC – that is, a Model, View and Controller – are the three things each web framework is made of. It is considered to be a basic structure, but there can be several contrasts among them.

Sample Python CGI script that displays the ten most recently published books from a database.

```
#!/usr/bin/env python
import MySQLdb

print "Content-Type: text/html\n"

print "<html><head><title>Books</title></head>" print "<body>"
print "<h1>Books</h1>" print "<ul>"

connection = MySQLdb.connect(user='me', passwd='letmein', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC LIMIT 10")

for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]
print "</ul>"
print "</body></html>"

connection.close()
```

Despite its simplicity, this approach has a number of problems and annoyances.

- What happens when multiple parts of your application need to connect to the database?
- Should a developer *really* have to worry about printing the “Content-Type” line and remembering to close the database connection?
- What happens when this code is reused in multiple environments, each with a separate database and password? At this point, some environment-specific configuration becomes essential.
- What happens when a Web designer who has no experience coding Python wishes to redesign the page?

### **MVT DESIGN PATTERN (MODEL VIEW TEMPLATE)(Django follows the MVT model)**

- Model - The data you want to present, usually data from a database.
- View - A request handler that returns the relevant template and content - based on the request from the user.
- Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data

Here's how you might write the previous CGI code using Django. The first thing to note is that we split it over three Python files (models.py, views.py, urls.py) and an HTML template (latest\_books.html):

#### **# models.py (the database tables)**

```
from django.db import models
```

```
class Book(models.Model):
```

```
    name = models.CharField(max_length=50)
```

```
    pub_date = models.DateField()
```

#### **# views.py (the business logic)**

```
from django.shortcuts import render_to_response
from models import Book

def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html', {'book_list': book_list})
```

#### # urls.py (the URL configuration)

```
from django.conf.urls.defaults import * import
views

urlpatterns = patterns("", (r'^latest/$', views.latest_books),
)
```

#### # latest\_books.html (the template)

```
<html>
<head><title>Books</title></head>
<body>
<h1>Books</h1>
<ul>
{% for book in book_list %}
<li>{{ book.name }}</li>
{% endfor %}
</ul>
</body></html>
```

## MODEL

- The model provides data from the database. The model contains all the data and business logic layers, its rules and functions.
- 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.
- The models are usually located in a file called **models.py**. The models.py file contains a description of the database table, as a Python class. This is called a model. Using this class, you can create, retrieve, update, and delete records in your database using simple Python code rather than writing repetitive SQL statements.

## VIEWS



- The view, on the other hand, is responsible for all visual representations of data, like diagrams, charts etc. 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.
- The views are usually located in a file called **views.py**. The views.py file contains the business logic for the page.
- The urls.py file specifies which view is called for a given URL pattern.

### TEMPLATE

- A template is a file where you describe how the result should be represent or describes the design of the page
  - Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.
  - Django uses standard HTML to describe the layout
  - The templates of an application is located in a folder named **templates**.
- 
- MVC defines a way of developing software so that the code for defining and accessing data (the model) is separate from request routing logic (the controller), which in turn is separate from the user interface (the view).
  - A key advantage of such an approach is that components are loosely coupled. That is, each distinct piece of a Django-powered Web application has a single key purpose and can be changed independently without affecting the other pieces.
  - For example, a developer can change the URL for a given part of the application without affecting the underlying implementation. A designer can change a page's HTML without having to touch the Python code that renders it. A database administrator can rename a database table and specify the change in a single place, rather than having to search and replace through a dozen files.

### FEATURES

- **WebCaching**  
Web caching simply helps store different documents and avoids annoying phenomenon of the server overload. Users can use it in various systems if several conditions are met. It also works on the server side. For example, you may notice cached content links on the SERP (Search Engine Results Page) of a search engine like Google.
- **Scaffolding**  
This is another important technique to know and use, which is supported by some MVC frameworks. Typical parts of application or the entire project structure (in case of initialization) can be generated by the framework automatically. This approach increases the speed of the development cycle and standardizes the codebase.
- **Web template system**



A web template system is a set of different methodologies and software implemented to construct and deploy web pages. Template engines are used to process web templates. They are a tool for web publishing in a framework.

- **Security**

Online security has plenty of criteria for identifying and permitting or rejecting access to different functions in a web framework. It also helps recognize the profiles that use the application to avoid clickjacking. As a result, the framework itself is authentic and authorized.

- **URL Mapping**

If you want to simplify the indexing of your website by search engines while creating a clear and eye-catching site name, this web frameworks' feature is custom-made for it. URL Mapping can also facilitate access to your sites' URLs.

- **Applications**

Numerous types of web applications are supported by web frameworks. The most common and best frameworks for app development support the construction of blogs, forums, general-purpose websites, content management systems, etc.

### **DJANGO'S HISTORY**

- Django was design and developed by Lawrence journal world in 2003 and publicly released under BSD license in July 2005. Currently, DSF (Django Software Foundation) maintains its development and release cycle.
- Django was released on 21, July 2005.

Version	Date	Description
0.90	16 Nov 2005	
0.91	11 Jan 2006	magic removal
0.96	23 Mar 2007	newforms, testing tools
1.0	3 Sep 2008	API stability, decoupled admin, unicode
1.1	29 Jul 2009	Aggregates, transaction based tests
1.2	17 May 2010	Multiple db connections, CSRF, model validation
1.3	23 Mar 2011	Timezones, in browser testing, app templates.

Version	Date	Description
1.5	26 Feb 2013	Python 3 Support, configurable user model
1.6	6 Nov 2013	Dedicated to Malcolm Tredinnick, db transaction management, connection pooling.
1.7	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	1 Apr 2015	Native support for multiple template engines. <i>Supported until at least April 2018</i>
1.9	1 Dec 2015	Automatic password validation. New styling for admin interface.
1.10	1 Aug 2016	Full text search for PostgreSQL. New-style middleware.
1.11 LTS	1.11 LTS	Last version to support Python 2.7. <i>Supported until at least April 2020</i>
2.0	Dec 2017	First Python 3-only release, Simplified URL routing syntax, Mobile friendly admin.

**Lab Program 1 :****INSTALLATION OF DJANGO****STEP 1: PYTHON INSTALLATION**

- Installing Python Django is written in 100% pure Python code, so you'll need to install Python on your system
- To check if your system has Python installed, run this command in the **command prompt**:

**python --version**

**STEP 2: PIP INSTALLATION**

- To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.
- For windows >py -3 -m ensurepip  
Other OS > python -m ensurepip --upgrade
- To check if your system has PIP installed, run this command in the **command prompt**:

**pip --version**

**STEP 3: SETTING VIRTUAL ENVIRONMENT**

- It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is venv, which is included in Python.
- The name of the virtual environment is your choice, in this tutorial we will call it myworld.
- Type the following in the command prompt, remember to navigate to where you want to create your project:
- Windows:

**py -m venv myworld**

- Unix/MacOS:

**python -m venv myworld**

- Then you have to activate the environment, by typing this command:

- o Windows:

**myworld\Scripts\activate.bat**

- o Unix/MacOS:

**source myworld/bin/activate**

- Once the environment is activated, you will see this result in the command prompt:
- Windows: **(myworld) C:\Users\Your Name>**
- Unix/MacOS: **(myworld) ... \$**

**STEP 4: INSTALL DJANGO**

- Now, that we have created a virtual environment, we are ready to install Django.
- Note: Remember to install Django while you are in the virtual environment!
- Django is installed using pip, with this command:
- Windows:

**(myworld) C:\Users\Your Name>py -m pip install Django**

- Unix/MacOS:

**(myworld) ... \$ python -m pip install Django**

- Which will give a result that looks like this:

```
Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
    | 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
    | 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\Your Name\myworld\Scripts\python.exe -m pip install --upgrade
pip' command.
```

## STEP 5: CHECK DJANGO VERSION

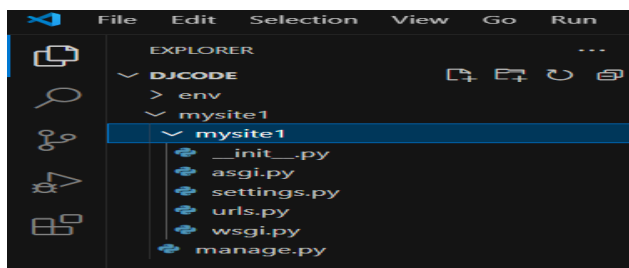
**django-admin --version**

## STEP 6: DJANGO CREATE PROJECT

- A project is a collection of settings for an instance of Django, including database configuration, Django-specific options, and application-specific settings.
- Create a directory by name djcode: **mkdir djcode**
- Once you have come up with a suitable name for your Django project, navigate to where in the file system you want to store the code (in the virtual environment), I will navigate to the folder, and run this command in the command prompt:

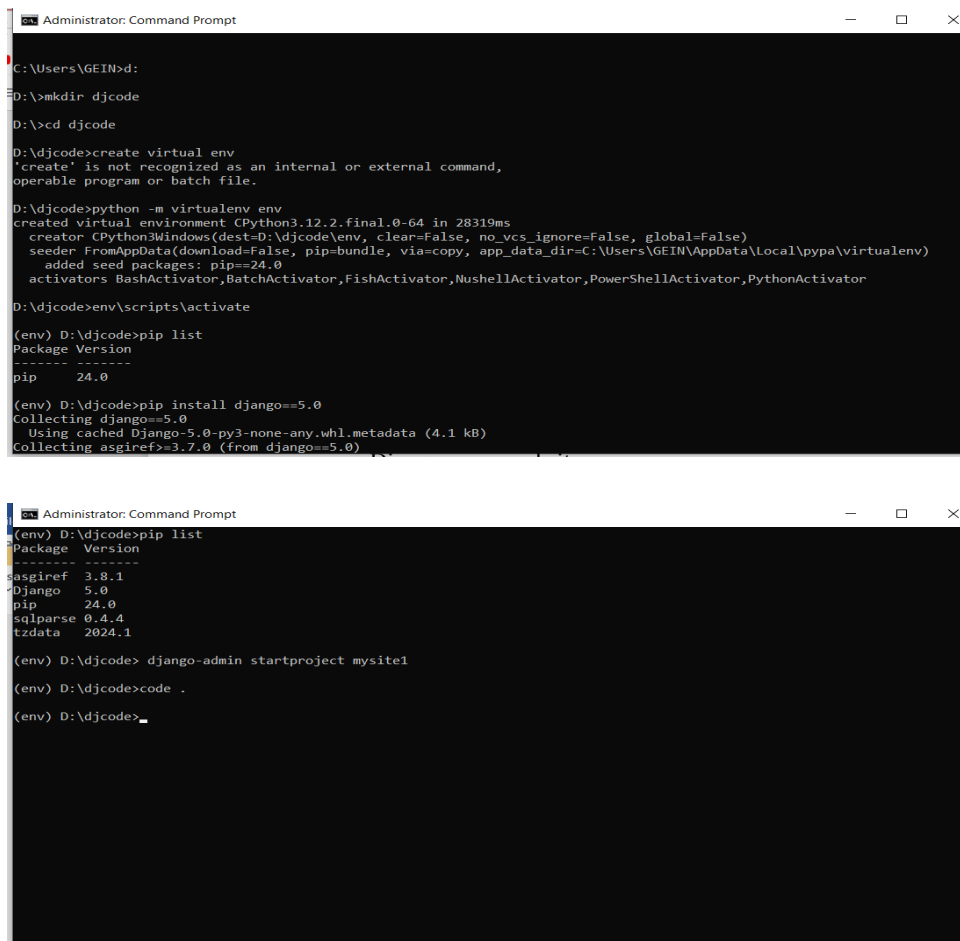
**django-admin startproject mysite**

- Django creates a **mysite** folder on my computer, with this content:



These files are as follows:

- **\_\_init\_\_.py**: A file required for Python; treat the directory as a package (i.e., a group of modules).
- **manage.py**: A command-line utility that lets you interact with this Django project in various ways
- **settings.py**: Settings/configuration for this Django project
- **urls.py**: The URL declarations for this Django project; a “table of contents” of your Django-powered sit



```
Administrator: Command Prompt

C:\Users\GEIN>d:
D:\>mkdir djcode
D:\>cd djcode

D:\djcode>create virtual env
'create' is not recognized as an internal or external command,
operable program or batch file.

D:\djcode>python -m virtualenv env
created virtual environment CPython3.12.2.final.0-64 in 28319ms
creator CPythonWindows(dest=D:\djcode\env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=C:\Users\GEIN\AppData\Local\pypa\virtualenv)
added seed packages: pip==24.0
activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator

D:\djcode>env\scripts\activate

(env) D:\djcode>pip list
Package Version
-----
pip      24.0

(env) D:\djcode>pip install django==5.0
Collecting django==5.0
  Using cached Django-5.0-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref>=3.7.0 (from django==5.0)
  Using cached asgiref-3.8.1-py3-none-any.whl.metadata (3.8 kB)
Collecting sqlparse>=0.4.4 (from django==5.0)
  Using cached sqlparse-0.4.4-py3-none-any.whl.metadata (3.8 kB)
Collecting tzdata>=2024.1 (from django==5.0)
  Using cached tzdata-2024.1-py3-none-any.whl.metadata (3.8 kB)
Installing collected packages: asgiref, sqlparse, tzdata, django
Successfully installed asgiref-3.8.1 django-5.0 sqlparse-0.4.4 tzdata-2024.1

(env) D:\djcode>django-admin startproject mysite1

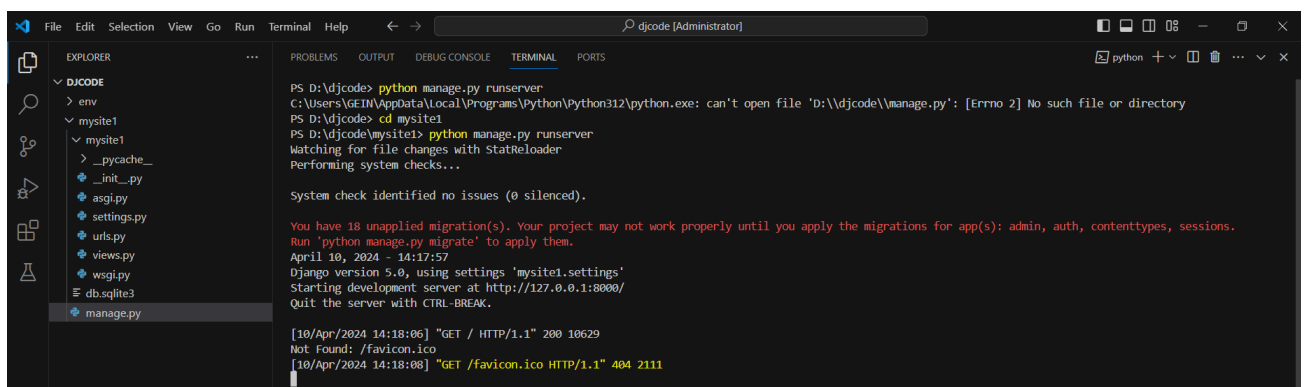
(env) D:\djcode>code .
(env) D:\djcode>
```

## STEP 7: RUN THE DJANGO PROJECT

- Now that you have a Django project, you can run it, and see what it looks like in a browser. In cmd prompt run “**code .**” this will take to VS Code current project. Open terminal in VS Code
- Navigate to the /mysite folder and execute this command in the command prompt:

**py manage.py runserver**

- Which will produce this result:



```
VS Code Explorer:
- DJCODE
  - env
  - mysite1
    - __pycache__
    - __init__.py
    - asgi.py
    - settings.py
    - urls.py
    - views.py
    - wsgi.py
    - db.sqlite3
    - manage.py

Terminal Output:
PS D:\djcode> python manage.py runserver
C:\Users\GEIN\AppData\Local\Programs\Python\Python312\python.exe: can't open file 'D:\djcode\manage.py': [Errno 2] No such file or directory
PS D:\djcode> cd mysite1
PS D:\djcode\mysite1> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

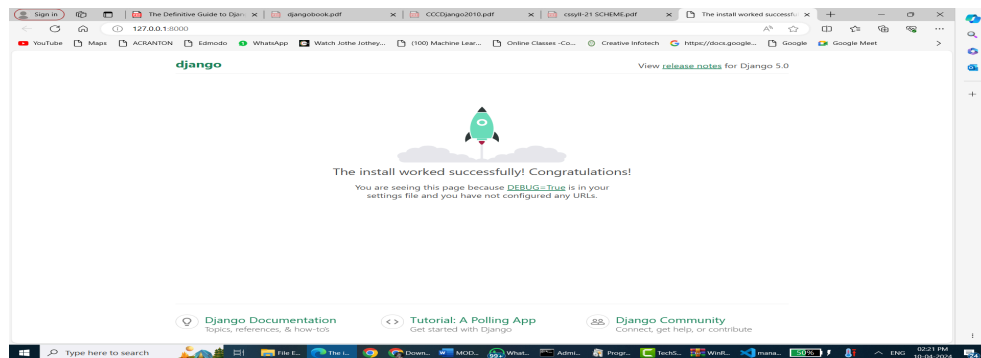
System check identified no issues (0 silenced).

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.
April 10, 2024 - 14:17:57
Django version 5.0, using settings 'mysite1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[10/Apr/2024 14:18:06] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[10/Apr/2024 14:18:08] "GET /favicon.ico HTTP/1.1" 404 2111
```

- Open a new browser window and type <http://127.0.0.1:8000/> in the address bar.

- The result:



## First Django-Powered Page: Hello World

Publish a simple “Hello world” Web page. With Django, you specify these two things as:

- The contents of the page are produced by a *view function*, and
- URL is specified in a *URLconf*.

First, let’s write the “Hello world” view function.

### First View

Within the mysite directory that `django-admin startproject` made in the last chapter,

1. Create an python application named hello
2. Create an urls.py under the hello app and type the content

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

urlpatterns = [
    path= ('',views.Hello,name='hello')
]
```

3. In views.py of app type the content # import the class HttpResponse, which lives in the django.http module

**from django.http import HttpResponse**

# define a function called hello. Each view function takes at least one parameter, called request.  
# This is an object that contains information about the current Web request that has triggered this #view, and it’s an instance of the class django.http.HttpRequest.

**def Hello(request):**

# function returns an HttpResponse object that has been instantiated with the text "Hello world".  
**return HttpResponse("Hello world")**

- 4.
- 5.

### First URLconf

If you run `python manage.py runserver` again, you may not get required output mysite project doesn’t know about the hello view; you need to tell Django explicitly that you’re activating this view at a particular URL. hence Django, use a URLconf.

A *URLconf* is like a table of contents for a Django-powered Web site. Basically, it's a mapping between URLs and the view functions that should be called for those URLs. It's how you tell Django, "For this URL, call this code, and for that URL, call that code."

When you executed `django-admin.py startproject` in the previous chapter, the script created a URLconf for you automatically: the file `urls.py`. By default, it looks something like this:

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
    # to INSTALLED_APPS to enable admin documentation:
    # (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    # (r'^admin/', include(admin.site.urls)),
)
```

This default URLconf includes some commonly used Django features commented out, so activating those features is as easy as uncommenting the appropriate lines. If you ignore the commented-out code, here's the essence of a URLconf:

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
)
```

Let's step through this code one line at a time:

- The first line imports all objects from the `django.conf.urls.defaults` module, which is Django's URLconf infrastructure. This includes a function called `patterns`.
- The second line calls the function `patterns` and saves the result into a variable called `urlpatterns`. The `patterns` function gets passed only a single argument: the empty string. (The string can be used to supply a common prefix for view functions, which we'll cover in Chapter 8.)

The main thing to note is the variable `urlpatterns`, which Django expects to find in the URLconf module. This variable defines the mapping between URLs and the code that handles those URLs. By default, the URLconf is empty—the Django application is a blank slate.

To add a URL and view to the URLconf, just add a Python tuple mapping a URL pattern to the view function. Here's how to hook in the `hello` view:

```
from django.conf.urls.defaults import *

from mysite.views import hello

urlpatterns = patterns('', (r'^hello/$', hello),
)
```

Two changes were made:

- First, the `hello` view was imported from its module: `mysite/views.py`, which translates into `mysite.views` in Python import syntax. (This assumes that `mysite/views.py` is on the Python path; see the sidebar for details.)
- Next, the line `(r'^hello/$', hello)` was added to `urlpatterns`. This line is referred to as a *URL pattern*. It's a Python tuple in which the first element is a pattern-matching string (a regular expression; more on this in a bit) and the second element is the view function to use for that pattern.

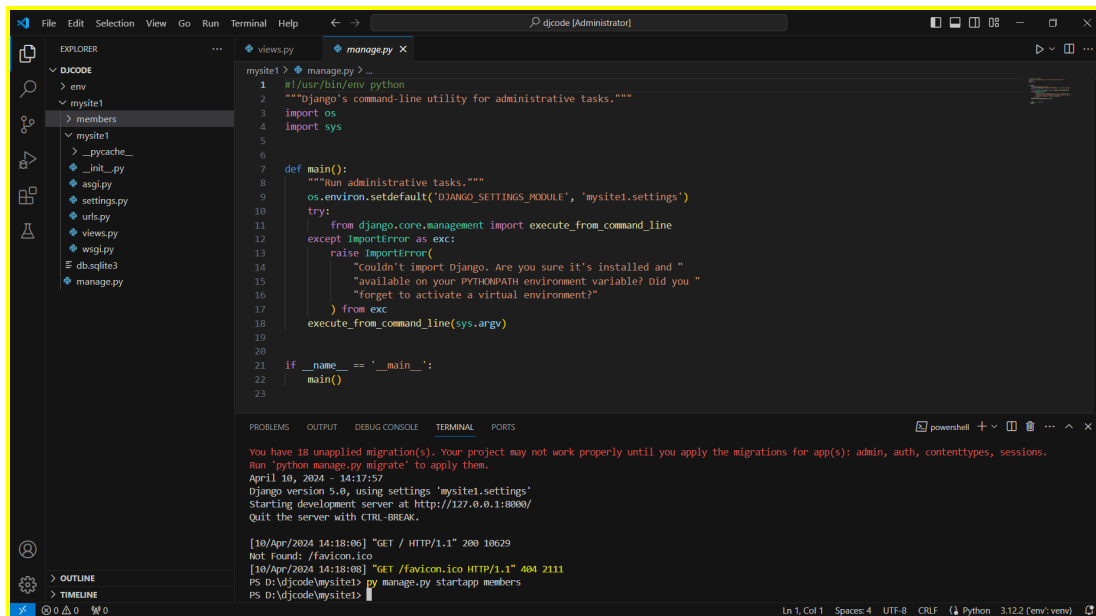
In a nutshell, Django was told that any request to the URL `/hello/` should be handled by the `hello` view function.



**STEP 8: DJANGO CREATE APP**

- An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.
- Start by navigating to the selected location where you want to store the app, in my case the **mysite1** folder, and run the command below.
- If the server is still running, and you are not able to write commands, press [CTRL] [BREAK], or [CTRL] [C] to stop the server and you should be back in the virtual environment.
- Creation of app should be done in VS Code terminal only

**py manage.py startapp members**

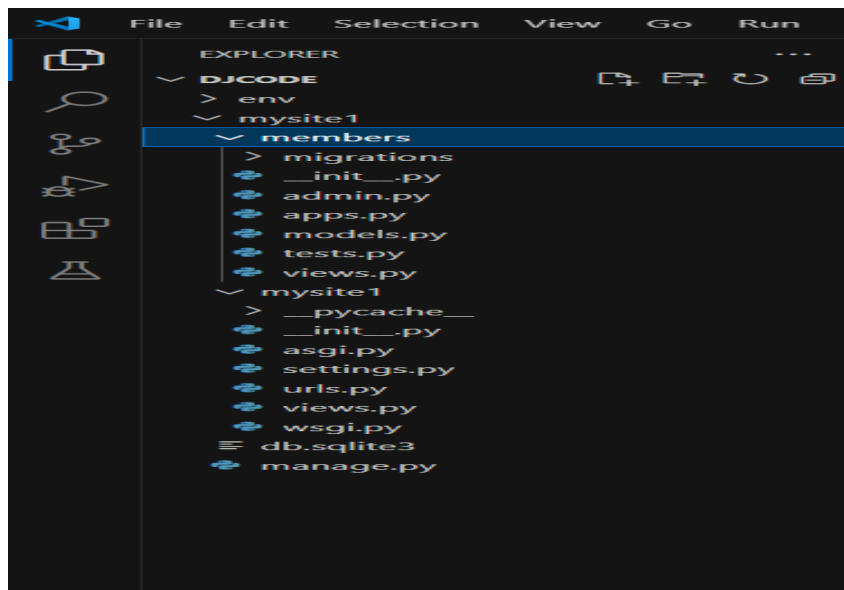


The screenshot shows the VS Code interface with the Explorer view on the left and the Terminal view at the bottom. The Explorer view shows the project structure: DJCODE > env > mysite1 > members. The members folder contains files: \_\_init\_\_.py, asgi.py, settings.py, urls.py, views.py, wsgi.py, db.sqlite3, and manage.py. The Terminal view shows the output of the command 'py manage.py startapp members'. The output indicates that 18 unapplied migrations exist and provides instructions to run 'python manage.py migrate' to apply them. It also shows the Django version (5.0) and the settings used ('mysite1.settings'). The terminal output is as follows:

```
PS D:\djcode\mysite1> py manage.py startapp members
PS D:\djcode\mysite1>

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.
April 10, 2024 - 14:17:57
Django version 5.0, using settings 'mysite1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[10/Apr/2024 14:18:06] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[10/Apr/2024 14:18:08] "GET /favicon.ico HTTP/1.1" 404 2111
PS D:\djcode\mysite1> py manage.py startapp members
PS D:\djcode\mysite1>
```



## **DANJO VIEWS**

- 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 . . . or anything, really.
- The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it's on your Python path.
- A web page that uses Django is full of views with different tasks and missions.
- Views are usually put in a file called `views.py` located on your app's folder.

## **STEPS TO CREATE VIEWS:**

- First create a file called **views.py** in the mysite directory. (right click on the folder mysite in VS CODE – new file – give file name as `views.py`)
- view that returns the current date and time, as an HTML document:

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

**views.py**

- First, we import the class **HttpResponse**, which lives in the **django.http** module.
- The **datetime module** contains several functions and classes for dealing with dates and times, including a function that returns the current time.
- Next, we define a function called **current\_datetime**. This is the view function. Each view function takes an **HttpRequest object** as its first parameter, which is typically named `request`.
- The first line of code within the function calculates the current **date/time** as a **datetime.datetime** object, and stores that as the local variable `now`.
- The second line of code within the function constructs an **HTML response** using Python's format-string capability. The `%s` within the string is a placeholder, and the percent sign after the string means "Replace the `%s` with the value of the variable `now`."
- Finally, the view returns an **HttpResponse object** that contains the generated response. Each view function is responsible for returning an `HttpResponse` object.

## **MAPPING URLS TO VIEWS**

- A URLconf is like a table of contents for your Django-powered Web site. Basically, it's a mapping between URL patterns and the view functions that should be called for those URL patterns. It's how you tell Django, "For this URL, call this code, and for that URL, call that code."
- When we executed **django-admin.py startproject**. The script created a URLconf for you automatically: the file **urls.py**. Let's edit that file:

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

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

**urls.py**

Or

```
from django.contrib import admin
from django.conf.urls import include
from django.urls import path, re_path
from mysite1.views import current_datetime

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path('^time/$', current_datetime)
    #path('home/', views.current_datetime)
]
```

**urls.py**

When a user requests a page from your Django-powered site, this is the algorithm the system follows to determine which Python code to execute:

1. Django determines the root **URLconf** module to use. Ordinarily, this is the value of the **ROOT\_URLCONF** setting, but if the incoming **HttpRequest** object has a **urlconf** attribute (set by middleware), its value will be used in place of the **ROOT\_URLCONF** setting.
2. Django loads that Python module and looks for the variable **urlpatterns**. This should be a sequence of **django.urls.path()** and/or **django.urls.re\_path()** instances.
3. Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL, matching against **path\_info**.
4. Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function (or a class-based view). The view gets passed the following arguments:
  - o An instance of **HttpRequest**.
  - o If the matched URL pattern contained no named groups, then the matches from the regular expression are provided as positional arguments.

- o The keyword arguments are made up of any named parts matched by the path expression that are provided, overridden by any arguments specified in the optional **kwargs** argument to **django.urls.path()** or **django.urls.re\_path()**.
5. If no URL pattern matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.

## PATH CONVERTERS

The following path converters are available by default:

- **str** - Matches any non-empty string, excluding the path separator, '/'. This is the default if a converter isn't included in the expression.
- **int** - Matches zero or any positive integer. Returns an **int**.
- **slug** - Matches any slug string consisting of ASCII letters or numbers, plus the hyphen and underscore characters. For example, **building-your-1st-django-site**.
- **uuid** - Matches a formatted UUID. To prevent multiple URLs from mapping to the same page, dashes must be included and letters must be lowercase. For example, **075194d3-6885-417e-a8a8-6c931e272f00**. Returns a **UUID** instance.
- **path** - Matches any non-empty string, including the path separator, '/'. This allows you to match against a complete URL path rather than a segment of a URL path as with **str**.
- If the paths and converters syntax isn't sufficient for defining your URL patterns, you can also use regular expressions. To do so, use **re\_path()** instead of **path()**.

## REGULAR EXPRESSIONS

- Regular expressions (or regexes) are a compact way of specifying patterns in text. While Django URLconfs allow arbitrary regexes for powerful URL-matching capability, you'll probably use only a few regex patterns in practice.

Symbol	Matches
.	Any character
\d	Any digit
[A-Z]	Any character, A–Z (uppercase)
[a-z]	Any character, a–z (lowercase)
[A-Za-z]	Any character, a–z (case insensitive)
+	One or more of the previous character (e.g., \d+ matches one or more digit)
[^/]+	All characters until a forward slash (excluding the slash itself)
?	Zero or more of the previous character (e.g., \d* matches zero or more digits)
{1,3}	Between one and three (inclusive) of the previous expression

## HOW DJANGO PROCESSES A REQUEST

- The command `python manage.py runserver` imports a file called `settings.py` from the same directory. This file contains all sorts of optional configuration for this particular Django instance.

- The `ROOT_URLCONF` setting tells Django which Python module should be used as the URLconf for this Web site.
- The view function is responsible for returning an `HttpResponse` object.

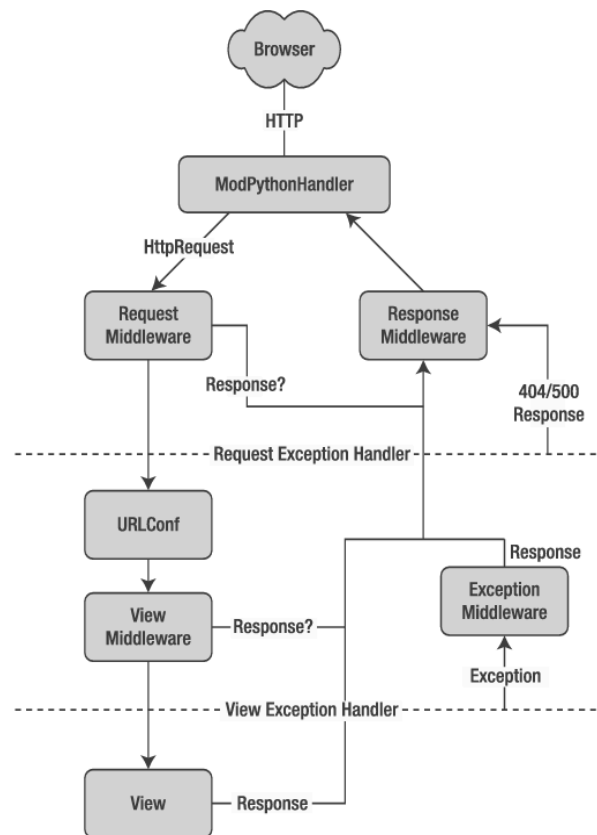


Figure 3-1. The complete flow of a Django request and response

- When an HTTP request comes in from the browser, a server-specific handler constructs the `HttpRequest` passed to later components and handles the flow of the response processing.
- The handler then calls any available Request or View middleware. These types of middleware are useful for augmenting incoming `HttpRequest` objects as well as providing special handling for specific types of requests. If either returns an `HttpResponse`, processing bypasses the view.
- If a view function raises an exception, control passes to the exception middleware. If this middleware does not return an `HttpResponse`, the exception is reraised.
- Even then, all is not lost. Django includes default views that create a friendly 404 and 500 response. Finally, response middleware is good for postprocessing an `HttpResponse` just before it's sent to the browser or doing cleanup of request-specific resources.

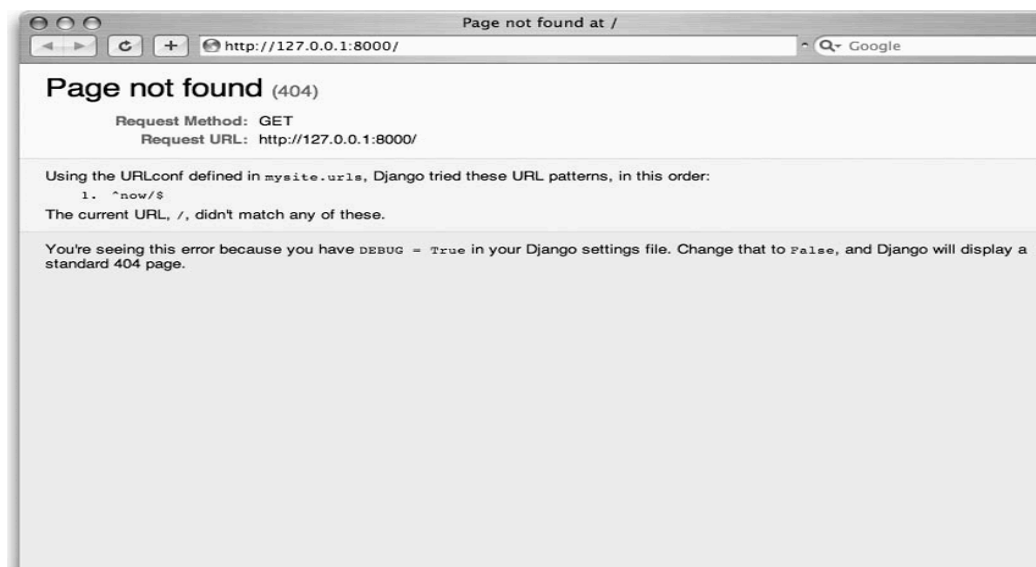
### URLconfs AND LOOSE COUPLING

- Loose coupling is a software-development approach that values the importance of making pieces interchangeable. If two pieces of code are loosely coupled, then changes made to one of the pieces will have little or no effect on the other.

- In a Django Web application, the URL definitions and the view functions are loosely coupled; that is, the decision of what the URL should be for a given function, and the implementation of the function itself, reside in two separate places. This lets a developer switch out one piece with out affecting the other.
- For example, consider the view function we wrote earlier, which displays the current date and time. If we wanted to change the URL for the application— say, move it from `/time/` to `/currenttime/`—we could make a quick change to the URLconf, without having to worry about the underlying implementation of the function. Similarly, if we wanted to change the view function—altering its logic somehow—we could do that without affecting the URL to which the function is bound.

### 404 Errors

- when a different URL is requested, Django displays “Page not found” message in debug mode because you requested a URL that’s not defined in your URLconf.
- The utility of this page goes beyond the basic 404 error message; it also tells you precisely which URLconf Django used and every pattern in that URLconf.



### Wildcard URL patterns

- view example, the contents of the page—the current date/time—were dynamic, but the URL (`/time/`) was static. In most dynamic Web applications, though, a URL contains parameters that influence the output of the page.
- Let’s create a second view that displays the current date and time offset by a certain number of hours.
- URLconf like this:

```
urlpatterns = patterns("",
    (r'^time/$', current_datetime),
```

```

        (r'^time/plus/1/$', one_hour_ahead),
        (r'^time/plus/2/$', two_hours_ahead),
        (r'^time/plus/3/$', three_hours_ahead),
        (r'^time/plus/4/$', four_hours_ahead),
    )

```

- let's put a wildcard in the URLpattern, a URLpattern is a regular expression; hence, we can use the regular expression pattern `\d+` to match one or more digits:

```

from django.contrib import admin
from django.conf.urls import include
from django.urls import path, re_path

```

```

from mysite1.views import current_datetime, hours_head
urlpatterns = [
    path('admin/', admin.site.urls),
    re_path('^time/$', current_datetime),
    re_path(r'^time/plus/(\d{1,2})/$', hours_head),
    #path('home/', views.current_datetime)
]

```

**urls.py**

```

from django.http import HttpResponse

```

```

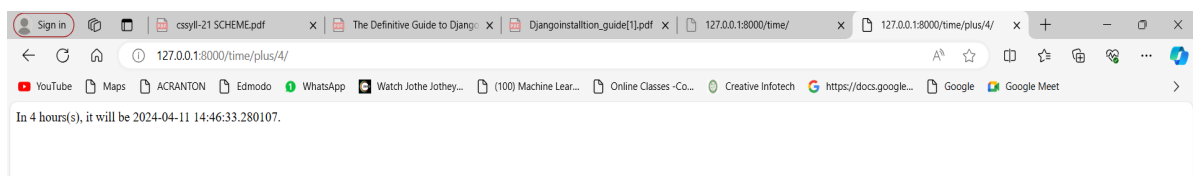
import datetime
def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)

def hours_head(request, offset):
    offset=int(offset)
    dt=datetime.datetime.now()+datetime.timedelta(hours=offset)
    html="<html><body>In %s hours(s), it will be %s.</body></html>"%(offset,dt)
    return HttpResponse(html)

```

**views.py**

Output:



- request** is an **HttpRequest** object, just as in **current\_datetime**.
- each view always takes an **HttpRequest** object as its first parameter.
- offset** is the string captured by the parentheses in the URL pattern. For example, if the requested URL were `/time/plus/3/`, then **offset** would be the string `'3'`. If the requested URL were `/time/plus/21/`, then **offset** would be the string `'21'`. Note that captured strings will always be strings, not integers, even if the string is composed of only digits, such as `'21'`



- function **int()** on **offset**. This converts the string value to an integer. Python will raise a `ValueError` exception if you call `int()` on a value that cannot be converted to an integer, such as the string 'foo'. **(\d{1,2})**—captures only digits.
- construct the HTML output of this view function uses Python's format-string capability with two values, two `%s` symbols in the string and a tuple of values to insert: **(offset, dt)**
- return an **HttpResponse** of the HTML
- start the Django development server (if it's not already running), and visit  
    `http://127.0.0.1:8000/time/plus/3/` to verify it works. Then try  
    `http://127.0.0.1:8000/time/plus/5/`.  
    <http://127.0.0.1:8000/time/plus/24/>.  
    `http://127.0.0.1:8000/time/plus/100/`
- pattern in your `URLconf` only accepts one- or two-digit numbers; Django should display a "Page not found" error in this case, just as we saw in the "404 Errors" section earlier. The URL `http://127.0.0.1:8000/time/ plus/` (with no hour designation) should also throw a 404.
- comment out the **offset = int(offset)** line in the **hours\_ ahead** view: **TypeError** message displayed at the very top: **"unsupported type for timedelta hours component: str"**. **datetime.timedelta** function expects the **hours** parameter to be an **integer**, and we commented out the bit of code that converted offset to an integer. That caused **datetime.timedelta** to raise the **TypeError**.