Gopalan College of Engineering and Management

Module-1   LECTURE NOTES

Module-1: MVC based web designing

Web framework, MVC Design Pattern, Django Evolution, Views, Mapping URL to Views, Working of Django URL Confs and Loose Coupling, Errors in Django, Wild Card patterns in URLS.

1.1. Web framework:

A web framework (also called web development framework) is a software platform that can be used for developing websites, web APIs (Application Programming Interface), and other web resources. It includes a library of prewritten code, components, snippets, and application templates that developers can use to create applications, products, or services to be deployed on the web.

The two types of web frameworks are:

Client-side framework – works in front end and used for managing user interface

Server-side framework – works in the background and is used for smooth functioning of the website

Types of Web Frameworks - Frontend and Backend

Web frameworks can be broadly classified as frontend and backend frameworks that ensure the effective development of the web application.

Frontend web development deals with the user or client side of the application. Any website or application has end users who need the website or application to be easy to use and meet their needs. Frontend framework takes care of this aspect of web development. With frontend web development framework, it is possible to develop applications and websites that have GUI and are very friendly for users.

Backend web development manages the server side of the application i.e., the functionality of the website or application. The aim of the backend framework is to ensure that there are no conflicts and glitches in the use of the website or application.

For frontend and backend web development, specific skills and knowledge are required and professionals who possess these skills are in great demand. In fact, a career in web development has become a viable option for those who know the frontend and backend languages and understand web development in detail.

What Are Web Frameworks Used For?

Web frameworks can be consistently applied across any website to manage uniformity in page elements like tables, texts, forms, and buttons and tasks like default browser settings, file structures, layout templates, among others. Frameworks are necessary as they:

Help to increase web traffic

Ensure quick and easy web development and maintenance

Include standardized codes and conventions

Systemize programming process and reduce errors and bugs

Efficiently handles background tasks like data binding and configuration

1.2. MVC Design Pattern:

• 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.

• The views.py file contains the business logic for the page, in the latest_books() function. This function is called a view.

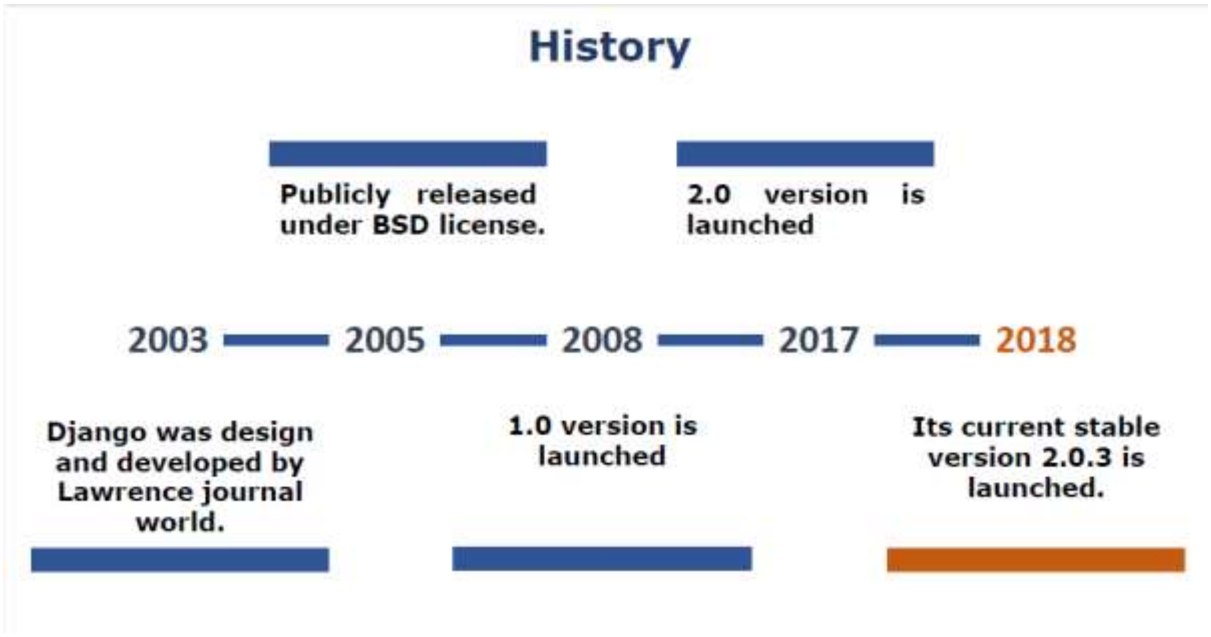
• The urls.py file specifies which view is called for a given URL pattern. In this case, the URL /latest/ will be handled by the latest_books() function.

• latest_books.html is an HTML template that describes the design of the page.

Taken together, these pieces loosely follow the Model-View-Controller (MVC) design pattern. Simply put, 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.

1.3. Django Evolution:


History
Publicly released under BSD license.
2.0 version is launched
2003
2005
2008
2017
2018
Django was design and developed by Lawrence journal world.
1.0 version is launched
Its current stable version 2.0.3 is launched.

Django's evolution traces back to 2003 when Adrian Holovaty and Simon Willison, then developers at the Lawrence Journal-World newspaper in Kansas, USA, created a framework to streamline their web development tasks. Originally intended to meet the specific needs of their newsroom, the framework aimed to simplify the creation of database-driven websites. This project eventually evolved into Django.

In July 2005, Django was released as an open-source project, with its first official version, 0.90, following shortly after in September of the same year. The release garnered attention from the developer community due to its focus on rapid development, clean design, and adherence to the DRY (Don't Repeat Yourself) principle.

As Django gained traction, its community expanded, contributing to its development and refining its features. Version 1.0 was a significant milestone, released in September 2008, marking Django's stability and maturity as a framework. With this release, Django solidified its position as one of the leading web frameworks for Python.

Subsequent versions introduced enhancements and new features, catering to the evolving needs of web developers. Version 1.2, released in May 2010, introduced support for multiple database connections and improved internationalization capabilities. Version 1.3, released in March 2011, brought features like class-based views and improved CSRF protection.

Django's popularity continued to grow, fueled by its robustness, scalability, and extensive documentation. Version 1.4, released in March 2012, introduced support for timezone-aware datetime objects and customizable user models. Version 1.6, released in November 2013, introduced a built-in migrations framework, making database schema changes more manageable.

In subsequent years, Django maintained a steady pace of development, with each release introducing enhancements, optimizations, and security updates. Notable versions include 1.8, which introduced support for Django's LTS (Long-Term Support) releases, and 2.0, which dropped support for Python 2.x in favor of Python 3.x, reflecting the framework's commitment to modernization and best practices.

Today, Django stands as a mature and feature-rich web framework, powering a multitude of websites, from small-scale projects to large-scale applications. Its vibrant community, comprehensive documentation, and commitment to backward compatibility ensure its continued relevance and adoption in the ever-changing landscape of web development.

1.4. Views:

A view function, or view for short, 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 . . . 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. There's no other requirement–no "magic," so to speak. For the sake of putting the code somewhere, the convention is to put views in a file called views.py, placed in your project or application directory.

A simple view

Here's a view that returns the current date and time, as an HTML document:

```python
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)
```

First, we import the class HttpResponse from the django.http module, along with Python's datetime library.

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.

Note that the name of the view function doesn’t matter; it doesn’t have to be named in a certain way in order for Django to recognize it. We’re calling it current_datetime here, because that name clearly indicates what it does.

The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

1.5. Mapping URL to Views:

URLconf serves as a roadmap for Django-powered websites, linking URL patterns to corresponding view functions.

It essentially instructs Django on which code to execute for specific URLs.

View functions must be accessible on the Python path to be utilized.

Upon initiating a project with django-admin.py startproject, a default URLconf file named urls.py is generated automatically.

The default urls.py file includes a urlpatterns variable, which defines the URL-to-view function mapping.

The urlpatterns variable is expected to be present in the project's ROOT_URLCONF module.

Initially, all mappings in the URLconf are commented out, leaving the Django application as a blank canvas.

An empty URLconf triggers Django to display the "It worked!" page, signaling the initiation of a new project.

Let’s edit this file to expose our current_datetime view:

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

from mysite.views import current_datetime

urlpatterns = patterns('',

(r'^time/$', current_datetime),

)
```

We made two changes here. First, we imported the current_datetime view from its module (mysite/views.py, which translates into mysite.views in Python import syntax). Next, we added the line (r'^time/$', current_datetime),. This line is referred to as a URLpattern—it's a Python tuple in which the first element is a simple regular expression and the second element is the view function to use for that pattern.

1.6. Working of Django URL Confs and Loose Coupling:

URLconfs and Django embody the principle of loose coupling, a fundamental philosophy in software development emphasizing interchangeable pieces. In Django, URL definitions and the corresponding view functions are loosely coupled, residing in separate locations. This separation allows for flexibility; altering one piece does not significantly impact the other. This stands in contrast to other web development platforms where URLs are tightly coupled to the program's structure. For instance, in PHP applications, the URL is dictated by the filesystem structure, while early versions of CherryPy bound URLs to method names. While these approaches may seem convenient initially, they become unwieldy over time. In Django, changing the URL or modifying the view function can be done independently. For example, relocating a function from '/time/' to '/currenttime/' only requires a URLconf adjustment, leaving the underlying function intact. Similarly, altering the function's logic does not affect its URL binding. Moreover, exposing functionality through multiple URLs is easily managed via URLconf edits, without touching the view code. This exemplifies the practical application of loose coupling, a concept further explored throughout this book.

1.7. Errors in Django:

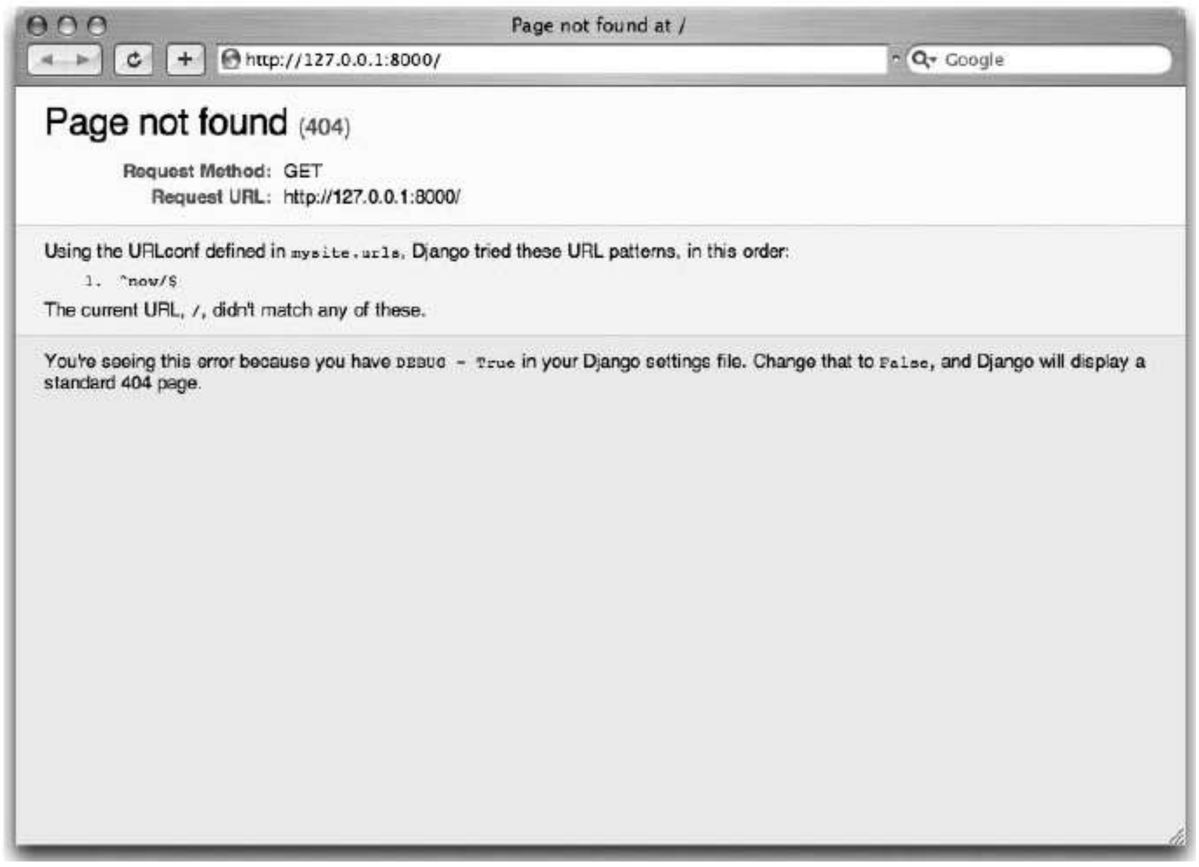
In Django development, encountering errors is not uncommon, but understanding and effectively managing them are crucial for smooth application development. One common type of error is the "Page not found (404)" error, indicating that the requested URL does not exist in the project's URL patterns. For instance, if a user tries to access a non-existent page like "example.com/nonexistent-page/", Django will raise a 404 error.

Another frequent error is the "TemplateDoesNotExist" error, typically occurring when Django cannot locate a specified template. This might happen if the template file is misspelled, or if the template directory is not properly configured in the settings. For example, if a view attempts to render a template named "missing_template.html" but it does not exist in the template directory, Django will raise a TemplateDoesNotExist error.

Furthermore, database-related errors are common, such as the "OperationalError" or "ProgrammingError" exceptions. These errors often arise due to issues like incorrect database configuration, SQL syntax errors, or database connection problems. For instance, if there's a typo in the database configuration settings or if the database server is down, Django might raise an OperationalError.

Handling errors effectively involves thorough debugging techniques, including examining Django's error messages, reviewing the code for potential issues, and utilizing Django's built-in debugging tools like the Django Debug Toolbar. Additionally, implementing proper error handling mechanisms within the application, such as custom error pages and logging, can enhance user experience and simplify troubleshooting processes.

404 Errors


Page not found at /

http://127.0.0.1:8000/

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/

Using the URLconf defined in `mysite.urls`, Django tried these URL patterns, in this order:

```
1. ^now/$
```

The current URL, `/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

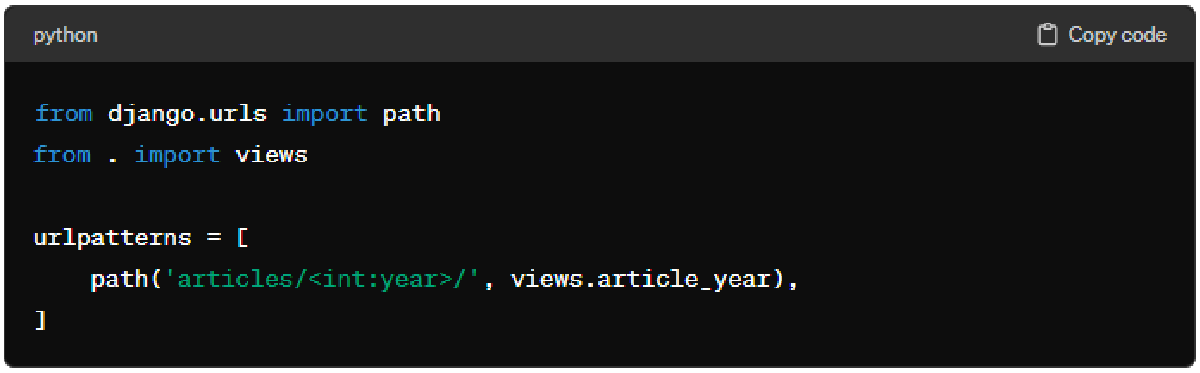
In our URLconf, we've currently defined a single URLpattern handling requests to '/time/'. But what occurs when a different URL is requested? To explore this, run the Django development server and try accessing various pages like '/hello/', '/does-not-exist/', or even the site root ('/'). You'll encounter a "Page not found" message, denoted by a pastel-colored display (see Figure 3-2). Django generates this message when the requested URL isn't defined in the URLconf.

This page serves more than a simple 404 error; it provides insight into the URLconf used and each pattern within it. Armed with this information, you can diagnose why the requested URL resulted in a 404 error. However, such details are sensitive and intended solely for web developers. In a live production environment, exposing this information to the public would be undesirable. Hence, the "Page not found" page is only visible when your Django project is in debug mode. We'll discuss deactivating debug mode later; for now, understand that every new Django project begins in debug mode, with a different response presented if debug mode is disabled.

1.8. Wild Card patterns in URLS:

Wildcard patterns in URLs for Python Django provide a dynamic and flexible approach to capturing segments of a URL. Utilizing angle brackets <> within URL patterns, developers can define placeholders for dynamic data. Commonly used wildcards such as <int>, <slug>, <uuid>, and <path> offer versatility in matching specific types of data within URLs. These patterns enable Django to handle dynamic URL routing, which is particularly valuable for building RESTful APIs or managing user-generated content. Wildcard patterns capture dynamic segments of a URL and pass them as arguments to the associated view function. This mechanism allows views to process parameters and generate responses tailored to the captured data. For instance, a URL pattern like path('articles/<int:year>/', views.article_year) captures an integer representing the year and passes it to the article_year view function. Consequently, URLs such as /articles/2023/ can be dynamically matched, with the year parameter being passed to the view function for further processing. Overall, wildcard patterns enhance the flexibility and robustness of Django web applications by enabling dynamic URL handling and parameter passing.

Wildcard patterns in URLs for Python Django offer a powerful tool for handling dynamic routing scenarios. For example, consider a blog application where articles are categorized by year:

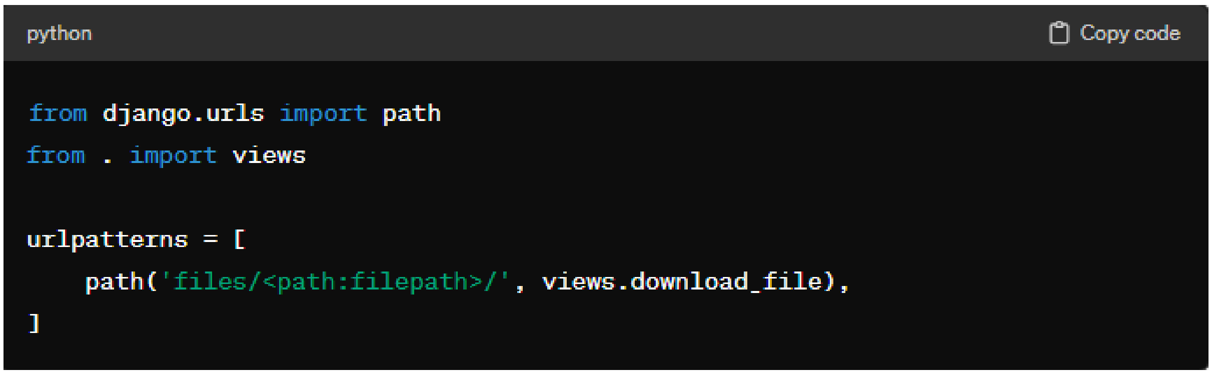


```python
from django.urls import path
from . import views

urlpatterns = [
    path('articles/<int:year>/', views.article_year),
]
```

In this example, <int:year> captures an integer representing the year and passes it to the article_year view function. Thus, URLs like /articles/2023/ will match this pattern, and the year 2023 will be passed as an argument to the view function for further processing.

Wildcard patterns also facilitate handling various file paths, as demonstrated in the following example:



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

urlpatterns = [
    path('files/<path:filepath>/', views.download_file),
]
```

In this case, <path:filepath> captures a complete URL path, allowing for flexible handling of file paths. URLs like /files/documents/report.pdf will match this pattern, with the entire path documents/report.pdf passed as an argument to the download_file view function.