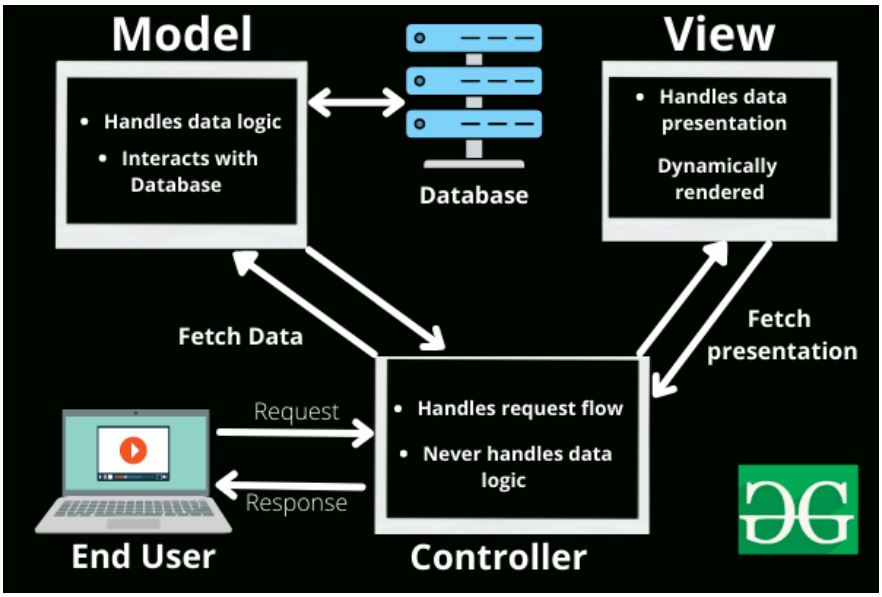
Module-1: MVC based Web Designing

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.

Module 1

The MVC Design Pattern



Components of MVC

The MVC framework includes the following 3 components:

* Controller
* Model
* View

Controller:

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn't have to worry about handling data logic, it just tells the model what to do. It processes all the business logic and incoming requests, manipulates data using the Model component, and interact with the View to render the final output.

View:

The View component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but

these data aren’t taken directly but through the controller. It only interacts with the controller.

Model:

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller’s request because the controller can’t interact with the database by itself. The model interacts with the database and gives the required data back to the controller.

 # models.py (the database tables)

```
from django.db import models

class Book(models.Model):

name = models.CharField(maxlength=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>
```

• **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.

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)
```

• First, we import the class HttpResponse, which lives in the django.http module.

• Then we import the datetime module from Python's standard library, the set of useful

modules that comes with Python. 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. 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, but it could just as well be named super_duper_awesome_current_time or something equally revolting. Django doesn't care. The next section explains how Django finds this function.

• 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." (Yes, the HTML is invalid, but we're trying to keep the example simple and short.)

• Finally, the view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object. (There are exceptions, but we'll get to those later.)

Mapping URLs to Views

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

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

urlpatterns = patterns('',

 (r'^mysite/', include('mysite.apps.foo.urls.foo')),

(r'^admin/', include('django.contrib.admin.urls')),)
```

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

• The first line imports all objects from the django.conf.urls.defaults module, including 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 rest of the lines are commented out. (The string can be used to supply a common prefix for view functions, but we'll skip this advanced usage for now.)

Regular Expressions

How Django Processes a Request: Complete Details

* 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.
* Bugs slip by even the best programmers, but exception middleware can help squash them.
* 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.