**Video file :64**

**Code file :56**

**Middleware in Django 5**

# Middleware in Django

### 🔹 What is Middleware?

* Middleware is a **layer between Django’s request/response cycle**.
* It is a **hook system** that lets you process requests **before** they reach the view and responses **before** they go back to the client.
* Think of it as a **pipeline**: request enters → passes through middleware → view → response passes back through middleware → client.

### 🔹 Why We Use Middleware?

* To apply **global logic** across the entire project without writing it in every view.
* Helps in **modularizing cross-cutting concerns** (security, sessions, logging, authentication, etc.).

### 🔹 Real-World Use Cases of Middleware

1. **Authentication** → Check if user is logged in before accessing certain views.
2. **Security** → Block suspicious IPs, prevent attacks, add security headers.
3. **Logging & Monitoring** → Log request time, errors, and performance.
4. **Session Management** → Attach session data to requests.
5. **Request/Response Processing** → Modify headers, compress response, transform data.
6. **Error Handling** → Catch exceptions globally and return custom error pages.

### 🔹 How Middleware Works (Request/Response Flow)

1. **Request Phase** → Request comes from client, middleware can:
   * Modify request
   * Block request
   * Add extra data
2. **View Phase** → Request reaches the view.
3. **Response Phase** → Response goes back through middleware, can:
   * Modify response headers
   * Add cookies
   * Compress or log response

### 🔹 Built-in Middleware Examples

* AuthenticationMiddleware → Associates users with requests.
* SessionMiddleware → Manages sessions.
* CommonMiddleware → Adds common HTTP improvements.
* CsrfViewMiddleware → Protects against CSRF attacks.
* SecurityMiddleware → Adds extra security features (HTTPS, HSTS).

### 📖 Summary (For Notes)

* **Middleware = Request/Response hooks** in Django.
* **Why**: To handle global tasks (security, auth, logging, sessions).
* **Where used**: Authentication, error handling, performance monitoring, response transformation.
* **Flow**: Request → Middleware → View → Response → Middleware → Client.

OUR OWN MATERIAL:

Two types of middleware :

1.Function based middleware

2.Class based middleware

1.Function based middleware :

App/middlewares.py:

def my\_fun\_middleware(get\_response):

    print("One time Initialiazation")  # her e we write only that code that can implements only one time

    def my\_func(request):

        print("This is before view")

        response = get\_response(request)

        print("This is after view")

        return response

    return my\_func

views.py:

from django.shortcuts import render

from django.template.response import TemplateResponse

def home(request):

  print("i am home view")

  return render(request, 'blog/home.html')

def my\_math(request):

 print("I am my\_math View")

 a = 10/0

 return render(request, 'blog/math.html', {'a': a})

def user\_info(request):

 print("I am user\_info View")

 context = {'name':'Rahul'}

 return TemplateResponse(request, 'blog/user.html', context)

app/urls.py:

from django.urls import path

from blog.views import home, my\_math, user\_info

urlpatterns = [

    path('', home, name="home"),

    path('math/', my\_math, name="my\_math"),

    path('user/', user\_info, name="user\_info"),

]

After that,

we can activate our middle go into the settings and add in the middleware:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'blog.middlewares.my\_fun\_middleware',  # add our custom middleware

]

Some times we need that we cannot run the views

**We can send response only from the middleware:**

In Some situations, we use that like we want to make the underconstruction page

Middlewares.py:

from django.shortcuts import HttpResponse

def my\_fun\_middleware(get\_response):

    print("One time Initialiazation")  # her e we write only that code that can implements only one time

    def my\_func(request):

        print("This is before view")

        response = HttpResponse("Response from my\_fun middle-ware")

        print("This is after view")

        return response

    return my\_func

another example:

from django.shortcuts import HttpResponse,render

def my\_fun\_middleware(get\_response):

    print("One time Initialiazation")

    def my\_func(request):

        print("This is before view")

        response = render(request,'blog/ui.html') # now here we direct move to html

        print("This is after view")

        return response

    return my\_func

templates/blog/ui.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>This site is under construction</h1>

</body>

</html>

2.Class based middleware

Middlewares.py:

# this can excute accourding to middleware like 1st middleware execute first and than go on

class MyClMiddleware:

    def \_\_init\_\_(self,get\_response):

        self.get\_response = get\_response

        print("one time intialization")

    def \_\_call\_\_(self,request):

        print("this is before view")

        response = self.get\_response(request)

        print("this is after view")

        return response

Settings.py:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'blog.middlewares.MyClMiddleware', # her write the class name

]

multipleMiddlewares:

middlewares.py:

#Mulitiple middleware work

class MyMiddleware1:

 def \_\_init\_\_(self, get\_response):

  self.get\_response = get\_response

  print("One Time MyMiddleware1 Initialization")

 def \_\_call\_\_(self, request):

  print("This is MyMiddleware1 before view")

  response = self.get\_response(request)

  print("This is MyMiddleware1 after view")

  return response

class MyMiddleware2:

 def \_\_init\_\_(self, get\_response):

  self.get\_response = get\_response

  print("One Time MyMiddleware2 Initialization")

 def \_\_call\_\_(self, request):

  print("This is MyMiddleware2 before view")

  response = self.get\_response(request)

  print("This is MyMiddleware2 after view")

  return response

class MyMiddleware3:

 def \_\_init\_\_(self, get\_response):

  self.get\_response = get\_response

  print("One Time MyMiddleware3 Initialization")

 def \_\_call\_\_(self, request):

  print("This is MyMiddleware3 before view")

  response = self.get\_response(request)

  print("This is MyMiddleware3 after view")

  return response

settings.py:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    #multiple middlwares

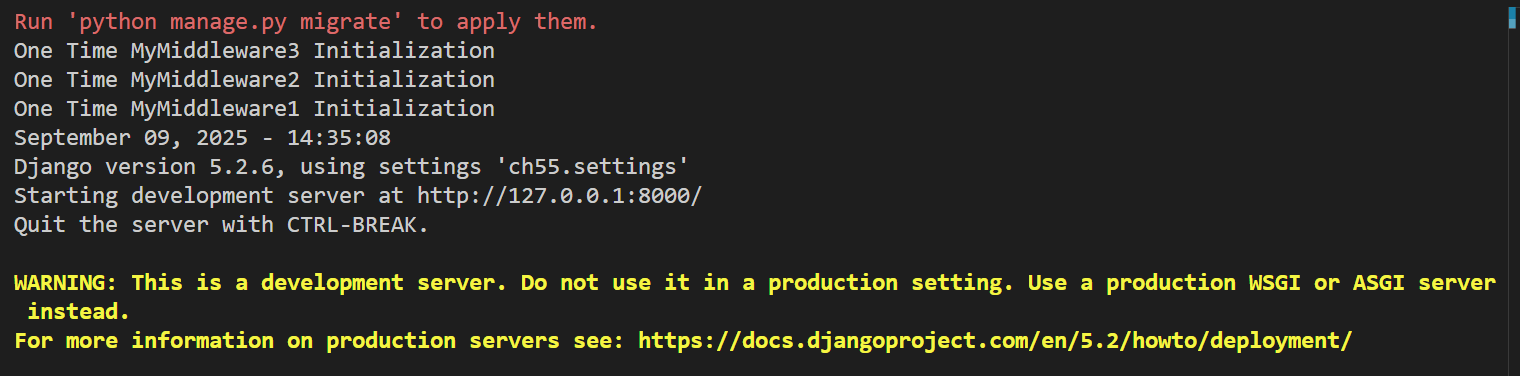
    'blog.middlewares.MyMiddleware1',

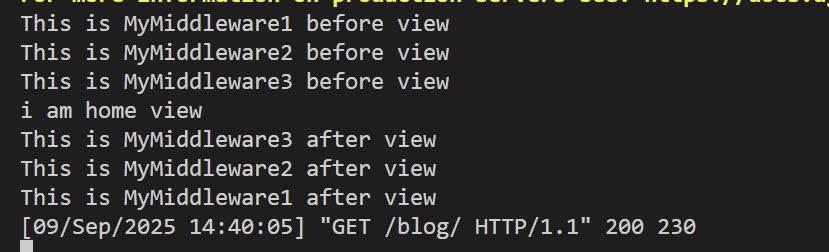
    'blog.middlewares.MyMiddleware2',

    'blog.middlewares.MyMiddleware3',

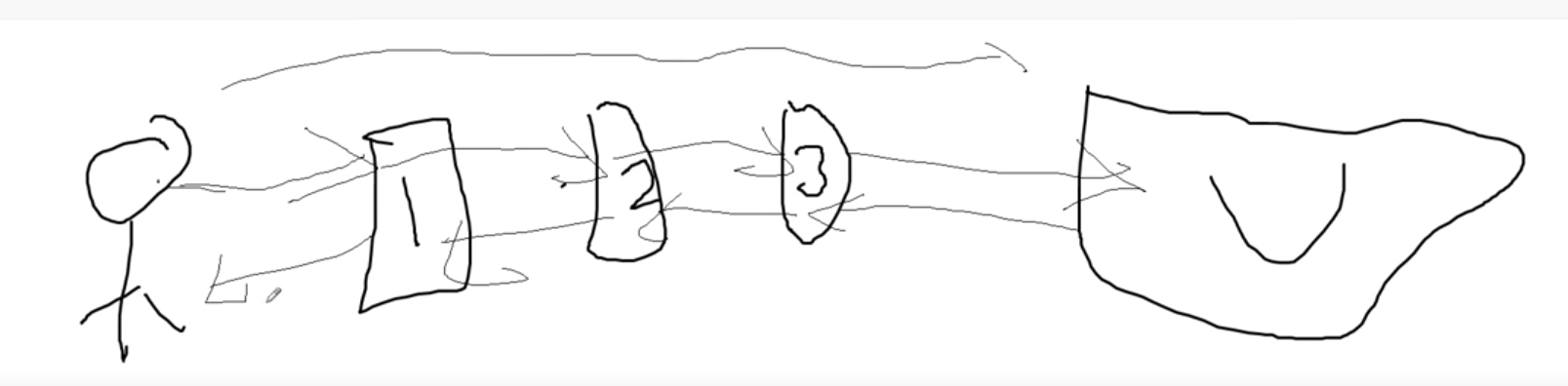
]

Mulitple middlware output:





Analogy behind that:



NOW MIDDLEWARE GIVE 3 HOOKS

HERE WE WILL SEE THESE HOOKS AND WHEN WE USE THEM

MyprocessMiddleware

MIDDLEWARES.PY:

#hooks

class MyprocessMiddleware:

    def \_\_init\_\_(self,get\_response): HERE WE WRITE OUR LOGICE BEFOARE ANDAFTER

        self.get\_response = get\_response

    def \_\_call\_\_(self, request):

        response = self.get\_response(request)

        return response

    def process\_view(request,\*args,\*\*kwargs):

        print("This is process view - Before view")

        return None

SETTINGS.PY:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    #HOOK

    'blog.middlewares.MyprocessMiddleware',

]

HOOKS NO 2:

MyExceptionMiddleWare

This hook we use in case od exception

Middlewares.py:

from django.shortcuts import HttpResponse

class MyExceptionMiddleWare:

    def \_\_init\_\_(self,get\_response):

        self.get\_response = get\_response

    def \_\_call\_\_(self, request):

        response = self.get\_response(request)

        return response

    def process\_exception(self,request,exception):

        print("Exception Occured")

        msg = exception

        class\_name = exception.\_\_class\_\_.\_\_name\_\_

        print(class\_name)

        print(msg)

        return HttpResponse(msg)

settings.py:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    #HOOK

    'blog.middlewares.MyExceptionMiddleWare',

]

Views.py:

def my\_math(request):

 print("I am my\_math View")

 a = 10/0

 return render(request, 'blog/math.html', {'a': a})

here we made exception if we hit that it shows exception

Output:



HOOKS NO 2:

MyTemplateResponseMiddleware

This hook is use doing manipulate our data before giving the response

Middlwares.py:

class MyTemplateResponseMiddleware:

  def \_\_init\_\_(self, get\_response):

    self.get\_response = get\_response

  def \_\_call\_\_(self, request):

    response = self.get\_response(request)

    return response

  def process\_template\_response(self, request, response):

    print("Process Template Response From Middleware")

    response.context\_data['name'] = 'Sonam'

    return response

settings.py:

MIDDLEWARE = [

    #HOOK

    'blog.middlewares.MyTemplateResponseMiddleware',

]

Views.py:

def user\_info(request):

 print("I am user\_info View")

 context = {'name':'Rahul'}

 return TemplateResponse(request, 'blog/user.html', context)

here we pass data in context and then before execution we can manipulate the data using hook templateresponse middleware