**MIDDLEWARE IN DJANGO**

Middleware is a framework or a plugin or a hook.

So when the request comes in the middle ware will take the request make some changes as required and it will send it to the view.

Similarly it can work on the templates, once the view process everything it returns the response, it can change the response as required and then send that response back to the view.

List of middleware, which Django uses in the project by default:

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',  
]

'django.middleware.security.SecurityMiddleware' :

Responsible for https or SSL support and other security features.

'django.contrib.sessions.middleware.SessionMiddleware':

Help us to do all the Session Management. Ie taking the incoming request make sure there is a session still active and when the response goes back to the web browser. This is one which adds the session cookie.

'django.middleware.common.CommonMiddleware':

Used to work with the URLS with ‘/’ at the end and dissolve it.

'django.middleware.csrf.CsrfViewMiddleware':

It checks if every post request or any form is submitted. It also looks for the CSRF token, if not will throw error.

'django.contrib.auth.middleware.AuthenticationMiddleware':

This is responsible for authenticating the end user request. It also makes sure that there is a user object available for the view, for the template.

'django.contrib.messages.middleware.MessageMiddleware':

'django.middleware.clickjacking.XFrameOptionsMiddleware':

**CUSTOM MIDDLE WARE**

WE will create a class and will add the below two methods :

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

\_\_call\_\_(self,response)

These two methos init and call are mandatory in every little value.

process\_view(self, request, view\_func, view\_args, view\_kwargs) : Invoked just before our view is called. Here we have direct access to the view function, arguments that are passed to the view function.

If we want to encrypt, decrypt, zip, unzip logics can be added to these lifecycle method.

process\_exception(self, request, exception) : We can handle the exceptions that are raised within our application in this method.

process\_template\_response(self, request, response): This method is invoked just before the process response is sent back to the client and it has access to request as well as the response.

The above three highlighted are optional and we can implement them as per our requirement.

Once all the middle ware are done, Django will handover the request to the view.

\_\_call\_\_(self,response) : The incoming method is given to this method. With this method we can work around with the request and on the way back the response can be accessed within this call method before this response goes back to the client.

**Middleware Code**

class MiddlewareLifeCycle:  
 *#INVOKED ONLY ONCE FOR THE ENTIRE APPLICATION* def \_\_init\_\_(self,get\_response):  
 print('init method')  
 self.get\_response = get\_response  
  
 *# INVOKED FOR EVERY INCOMING REQUEST* def \_\_call\_\_(self,request):  
 print("Before the view is executed !!!!")  
 response = self.get\_response(request)  
 print('After the view is executed !!!!!!')  
 return response

**Middleware added in settings.py file**

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',  
 'cookieApp.middleware.MiddlewareLifeCycle'  
]

**Output**

System check identified no issues (0 silenced).

init method

May 22, 2024 - 22:23:31

Django version 5.0.4, using settings 'cookiesProject.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

Before the view is executed !!!!

After the view is executed !!!!!!

[22/May/2024 22:23:35] "GET /index/ HTTP/1.1" 200 249

We have added an exception in the view (Index), which shows the below error message:

def index(request):  
 raise Exception("Exception in the Index Page")  
 return render(request, 'cookieApp/index.html')

A screenshot of a computer

Description automatically generated

The user is not aware of the entire stack trace, which we don’t need to provide the user.

Just added this in the middleware.py file and added this in the setting.py

class ExceptionHandlingMiddleWare:  
 *# INVOKED ONLY ONCE FOR THE ENTIRE APPLICATION* def \_\_init\_\_(self, get\_response):  
 self.get\_response = get\_response  
  
 *# INVOKED FOR EVERY INCOMING REQUEST* def \_\_call\_\_(self, request):  
 return self.get\_response(request)  
  
 *# EXCEPTIONS HADNDLED BY THE APPLICATION, DISPLAY A USER FRIENDLY MESSAGE* def process\_exception(self, request, exception):  
 return HttpResponse("<b> We are currently working on this issues !!! Will shortly be back. !!! </b>")

'cookieApp.middleware.MiddlewareLifeCycle',  
'cookieApp.middleware.ExceptionHandlingMiddleWare',

A screenshot of a computer

Description automatically generated

Now the entire message, is on the index page, this page is only shown.

By adding these two lines:

*# EXCEPTIONS HADNDLED BY THE APPLICATION, DISPLAY A USER FRIENDLY MESSAGE*def process\_exception(self, request, exception):  
 print(exception.\_\_class\_\_.\_\_name\_\_)  
 print(exception)  
 return HttpResponse("<b> We are currently working on this issues !!! Will shortly be back. !!! </b>")

**The output will be:**

Before the view is executed !!!!

Exception

Exception in the Index Page

After the view is executed !!!!!!

[22/May/2024 22:44:39] "GET /index/ HTTP/1.1" 200 78