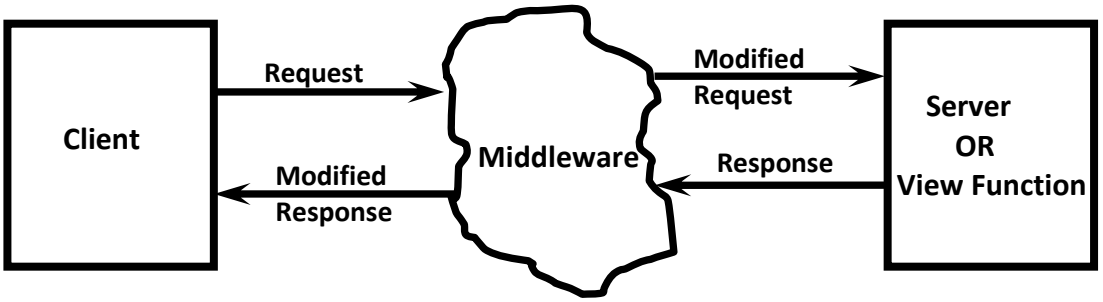
Chapter-13

Working with Django Middleware

Middleware:

Middleware is a framework of hooks into Django's request/response processing.
It is a ligth, low level 'plugin' system for globally altering Django's input or output.

If we want to perform any activity at the time of pre processing of the request or post processing of the request then we should go for middleware.



Client → Request → Middleware → modifiedrequest
Server → Response → Middleware → modifiedresponse

Django contains several inbuilt middlewares which are configured inside settings.py

```
1) MIDDLEWARE = [
2)     'django.middleware.security.SecurityMiddleware',
3)     'django.contrib.sessions.middleware.SessionMiddleware',
4)     'django.middleware.common.CommonMiddleware',
5)     'django.middleware.csrf.CsrfViewMiddleware',
6)     'django.contrib.auth.middleware.AuthenticationMiddleware',
7)     'django.contrib.messages.middleware.MessageMiddleware',
8)     'django.middleware.clickjacking.XFrameOptionsMiddleware',
9) ]
```

All these middlewares will be executed before and after processing of every request.

1) **SecurityMiddleware provides security enhancements like SSL Redirects(like from http request to https request) etc.**
2) **SessionMiddleware enables session support.**
3) **CommonMiddleware provides a common set of features like adding slash at the end of the URL.**
4) **CsrfViewMiddleware is responsible to verify whether POST request has csrf_token or not.**
5) **AuthenticationMiddleware is responsible to add user attribute to the request object.**

If we comment this middleware in settings.py then we cannot access user attribute in our view function.If we are trying to access you will get error.

print(request.user)

AttributeError at /first/
'WSGIRequest' object has no attribute 'user'

Note: Middlewares are applicable for every incoming request and for every outgoing response.

Middleware Structure:

Based on our requirement we can define our own customized middlewares.
Every customized middleware is a python class and it is the child class of object,contains 2 mandatory methods and 3 optional methods.

class LoginMiddleware(object):

```
  def __init__(self,get_response):
    #one time configuration and initialization on start-up,get_response is a reference to previous middleware response
    self.get_response=get_response
```

def __call__(self,request):

```
    #This code will be executed before the view(and other middleware) is called
    response=self.get_response(request) #It triggers next phase
    #This code will be executed after the view(and other middleware) is called
    return response # to finish middleware sequence
```

def process_view(self,request,view_func,view_args,view_kwargs):

```
    # Logic will be executed before a call to View
    # Gives access to the view itself and arguments
```

def process_exception(self,request,exception):

```
    #Logic will be executed if an exception/error occurs in the view
```

def process_template_response(self,request,response):

```
    #Logic is executed after view is called.
    It is required to alter the response itself to perform additional logic on it like modifying context or template.
```

Demo Application for Custom Middleware Execution Flow:

middleware.py:(inside application folder)

```
1)  class ExecutionFlowMiddleware(object):
2)      def __init__(self,get_response):
3)          self.get_response=get_response
4)
5)      def __call__(self,request):
6)          print('This line added at pre-processing of request')
7)          response=self.get_response(request)
8)          print('This line added at post-processing of request')
9)          return response
```

settings.py

```
1)  MIDDLEWARE = [
2)      ....,
3)      'testapp.middleware.ExecutionFlowMiddleware'
4)  ]
```

views.py

```
1)  from django.http import HttpResponse
2)
3)  # Create your views here.
4)  def welcome_view(request):
5)      print('This line added by view function')
6)      return HttpResponse('<h1>Custom Middleware Demo</h1>')
```

Results:

If we send a request in the server console we can see:

This line added at pre-processing of request
This line added by view function
This line added at post-processing of request

Before and After processing every request middleware will be executed.

Execution Process for a Single Middleware Class:

1) **__init__() method will be called only once at the time of server start-up.**
2) **__call__() method will be called for every request.**
3) **If we declare process_view() method then it will be called**

4) Inside __call__() method whenever we are using self.get_response(request) then view function starts its execution
5) If we declare process_exception() method, then it will be executed if any exception/error occurs inside view function.
6) View Method Finishes.
7) If we declare process_template_response() then it will be executed whenever view returns TemplateResponse.

Middleware application to show information saying application under maintenance:

middleware.py

```
1) from django.http import HttpResponse
2) class AppMaintenanceMiddleware(object):
3)     def __init__(self,get_response):
4)         self.get_response=get_response
5)
6)     def __call__(self,request):
7)         return HttpResponse('<h1>Currently Application under maintenance...
   plz try after 2 days!!!')
```

settings.py

```
1) MIDDLEWARE = [
2)     ...
3)     'testapp.middleware.AppMaintenanceMiddleware'
4) ]
```

views.py

```
1) from django.http import HttpResponse
2)
3) # Create your views here.
4) def home_page_view(request):
5)     return HttpResponse('<h1>Hello This is from home page view</h1>')
```

Middleware application to show meaningful response if view function raises any error:

In this case we have to use process_exception() method which will be executed if view function raising any exception/error.

middleware.py

```
1)  from django.http import HttpResponse
2)  class ErrorMessageMiddleware(object):
3)      def __init__(self,get_response):
4)          self.get_response=get_response
5)
6)      def __call__(self,request):
7)          return self.get_response(request)
8)
9)      def process_exception(self,request,exception):
10)         return HttpResponse('<h1>Currently we are facing some technical problems
        plz try after some time!!!</h1>')
```

settings.py

```
1)  MIDDLEWARE = [
2)      ...
3)      'testapp.middleware.ErrorMessageMiddleware'
4)  ]
```

views.py

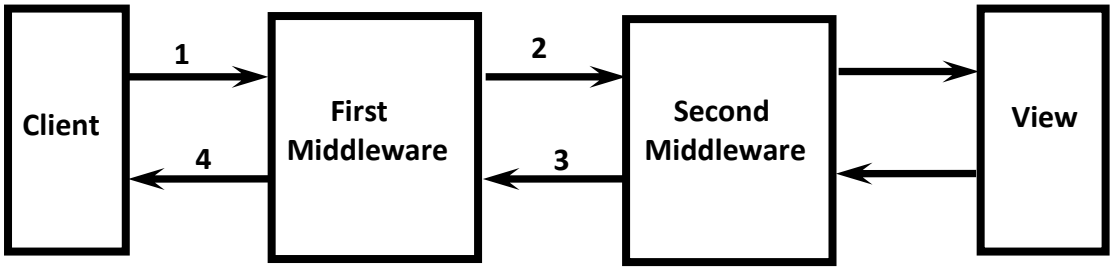
```
1)  from django.http import HttpResponse
2)
3)  # Create your views here.
4)  def home_page_view(request):
5)      print(10/0)
6)      return HttpResponse('<h1>Hello This is from home page view</h1>')
```

How to display raised exception information:

```
def process_exception(self,request,exception):
  return HttpResponse('<h1>Currently we are facing some technical problems plz try after
some time!!!</h1><h2>Raised Exception:{}</h2><h2>Exception
Message:{}</h2>'.format(exception.__class__.__name__,exception))
```

Configuration of multiple middleware classes:

We can configure any number of middlewares and all these middlewares will be executed according to order declared inside settings



middleware.py

```
1)  class FirstMiddleware(object):
2)      def __init__(self,get_response):
3)          self.get_response=get_response
4)
5)      def __call__(self,request):
6)          print('This line printed by FirstMiddleware at pre-processing of request')
7)          response=self.get_response(request)
8)          print('This line printed by FirstMiddleware at post-processing of request')
9)          return response
10) class SecondMiddleware(object):
11)     def __init__(self,get_response):
12)         self.get_response=get_response
13)
14)     def __call__(self,request):
15)         print('This line printed by SecondMiddleware at pre-processing of request')
16)         response=self.get_response(request)
17)         print('This line printed by SecondMiddleware at post-processing of request')
18)         return response
```

settings.py

```
1)  MIDDLEWARE = [
2)      ....,
3)      'testapp.middleware.FirstMiddleware',
4)      'testapp.middleware.SecondMiddleware'
5)  ]
```

views.py

```
1)  def home_page_view(request):
2)      print('This line printed by view function')
3)      return HttpResponse('<h1>Hello This is from home page view</h1>')
```

In the Server Console:

This line printed by FirstMiddleware at pre-processing of request
This line printed by SecondMiddleware at pre-processing of request
This line printed by view function
This line printed by SecondMiddleware at post-processing of request
This line printed by FirstMiddleware at post-processing of request

Note: If we change the order of middlewares inside settings.py then the output at server console is:

This line printed by SecondMiddleware at pre-processing of request
This line printed by FirstMiddleware at pre-processing of request
This line printed by view function
This line printed by FirstMiddleware at post-processing of request
This line printed by SecondMiddleware at post-processing of request