**DJANGO WEEK 6**

**22. How to task in Django?**

—> A Django app to run new background tasks from either admin or cron and inspect task history from admin.

—> To implement asynchronous tasks in django, you can use Celery.

**Features :**

- Create async tasks either programmatically or from admin
- Monitor async tasks from admin
- log all tasks in the database for later inspection
- Optionally save task-specific logs in a TextField and/or in a FileField

**pip install django-task**

**23. How to add task in celery?**

—> Follow the Steps to add tasks in Celery :

1. Install Celery : pip install ‘celery[redis]’
2. Create a Django Project : python -m django startproject (name)
3. Configure Celery in Django : Create celery.py

```
from __future__ import absoulte_import, unicode_literals
import os
From celery import Celery
from django.conf import settings
os.environ.setdefault(‘DJANGO_SETTINGS_MODULE’,’your_pro.settings’)
app = Celery(‘your_pro’)
app.config_from_object(‘django.conf:settings’, namespace = ‘CELERY’)
app.autodiscover_tasks(lambda:settings.INSTALLED_APPS)
```

4. Configure Celery Broker (Redis) : settings.py

```
CELERY_BROKER_URL = ‘redis://localhost:6379/0’’
```

5. Create Celery Tasks : tasks.py

```
from celery import shared_task
@shared_task
 def my_task(arg1, arg2):
      result = arg1 + arg2
      return result
```

6. Use Celery Tasks : You can now use celery tasks in your Django views, models or any other parts of your app. To call a task import it

```
from .tasks import my_task
Result = my_task.delay(3, 5)
```

7. Start Celery Worker : To run celery and process tasks you need to start a Celery worker process.

```
Celery -A your_pro worker --loglevel=info
```

8. redis-server
9. redis-cli ping : if it returns PONG then you’re ready to go

**DJANGO WEEK 6**

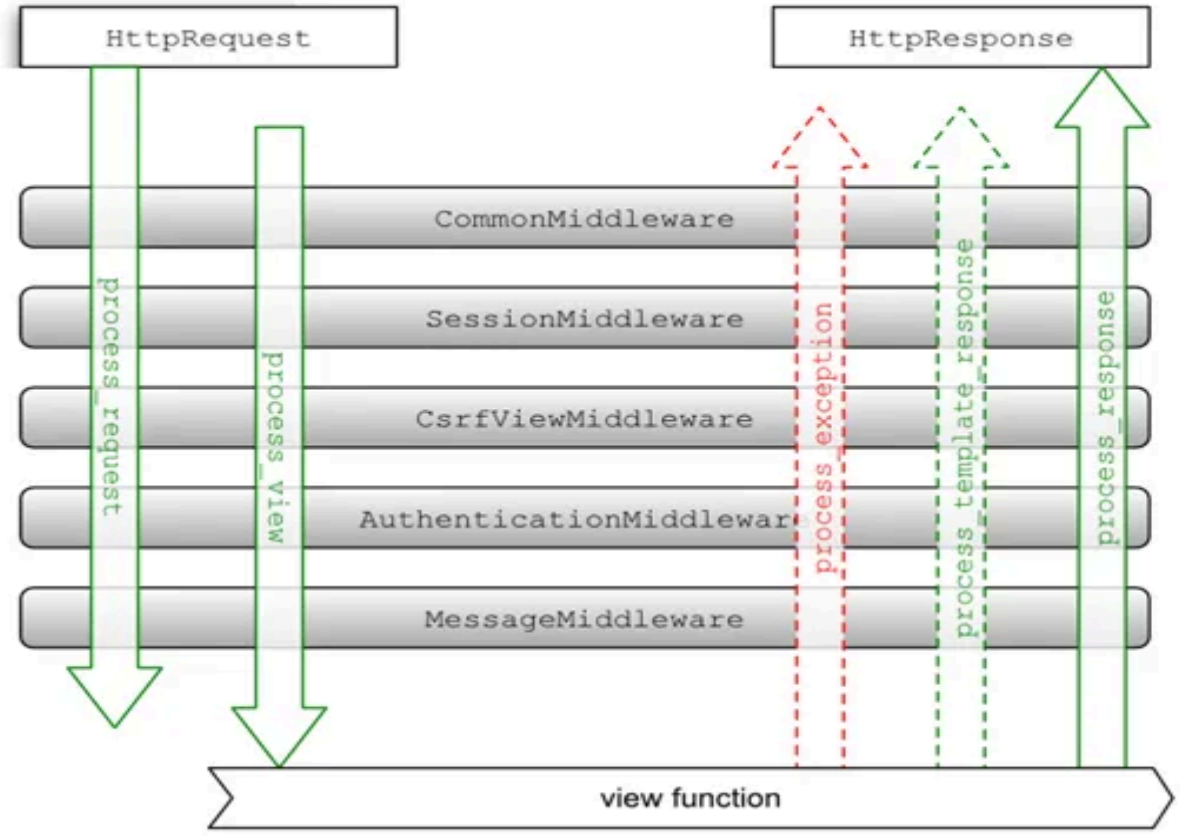
**24. Middleware :**

—> Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Dajngo's input or output. Each middleware component is responsible for doing some specific function.

**How does Middleware works :**

—> When a user makes request from your application, a WSGI handler is instantiated, which handles the following things :

- Imports project's settings.py file and Django exception classes.
- Loads all the middleware classes which are written in MMIDDLEWARE tuple located in settings.py file.
- Builds list of methods which handle processing of request, view, response & exception.
- Loops through the request methods in order.
- Resolves the requested URL
- Loops through each of the view processing methods.
- Calls the view function.
- Processes exception method (if any)
- Loops through each of the response methods in the reverse order from request middleware.
- Builds a return value and makes a call to the callback function.



# DJANGO WEEK 6

**25. Types of Middlewares :**

—> There are two types of Middleware in Django :

1. Built-in Middleware
2. Custom Middleware

**1. Built-in Middleware : They are provided by default in Dajngo when you create your project.**

—> These are the default middlewares that come with Django :

- Cache Middleware
- Common Middleware
- GZip Middleware
- Message Middleware
- Security Middleware
- Session Middleware
- Site Middleware
- Authentication Middleware
- CSRF Protection Middleware

**Authentication Middleware : It adds the user attribute, representing the logged-in user, to every incoming request object. If the user is not logged in then it will be set as an AnonymousUser Object.**

**Session Middleware : It helps you to store arbitrary data on a per-user basis like username, email, etc on the server side. On every request and response, a session_id is attached to the cookies that you can check in your browser through the console. Every time a request is made, session middleware gets the stored session data for the particular session_id and attaches it to the request object.**

**Message Middleware : It is used to display some notification message aka flash message to the user after form submission. You can store the message to the request object in your view and then show the message on your front end.**

**CSRF Middleware : It prevents Cross-Site Request Forgery attacks by adding hidden fields like CSRF_TOKEN to the POST forms and later validating for the correct value.**

**26. Custom Middleware :**

**2. Custom Middleware : You can write your own middleware which can be used throughout your project.**

—> Create a Python Package(a folder with __init__.py inside) named as middleware.

—> Create a file named custom_middleware.py (or anything which you like) and a regular Python function/class in it.

—> You can write middleware as a function or a class whose instances are callable.

**DJANGO WEEK 6**

**Function Based Middleware :**

```
def simple_middleware(get_response):
    # One-time configuration and initialization.
    def middleware(request):
    # Code to be executed for each request before the view (and later middleware) are called.
        response = get_response(request)
    # Code to be executed for each request/response after the view is called
        return response
    return middleware
```

**Class Based Middleware :**

```
class ExampleMiddleware:
    def __init__(self, request):
        self.get_response = get_response
    def __call__(self, request):
        # Code that is executed in each request before the view is called
        response = self.get_response(request)
        # Code that is executed in each request after the view is called
        return response
    def process_view(request, view_func, view_args, view_kwargs):
        # This code is executed if an exception is raised
    def process_template_response(request, response):
        # This code is executed if the response contains a render() method
        return response
```

—> Now the final step will be to add your custom middleware in MIDDLEWARE List in settings.py file.

```
MIDDLEWARE = [
            'your_app.middleware_directory.custom_middleware_file.CustomMiddleware_class',
        ]
```

# DJANGO WEEK 6

## 27. GITHUB

—> Add git before this following commands :

| | |
|---|---|
| **clone** | Clone a repository into a new directory |
| **init** | Create an empty Git repository or reinitialize an existing one work on the current change (see also: git help everyday) |
| **add** | Add file contents to the index |
| **mv** | Move or rename a file, a directory, or a symlink |
| **restore** | Restore working tree files |
| **rm** | Remove files from the working tree and from the index |

**Examine the history and state (see also: git help revisions)**

| | |
|---|---|
| **bisect** | Use binary search to find the commit that introduced a bug |
| **diff** | Show changes between commits, commit and working tree, etc |
| **grep** | Print lines matching a pattern |
| **log** | Show commit logs |
| **show** | Show various types of objects |
| **status** | Show the working tree status |

**grow, mark and tweak your common history**

| | |
|---|---|
| **branch** | List, create, or delete branches |
| **commit** | Record changes to the repository |
| **merge** | Join two or more development histories together |
| **rebase** | Reapply commits on top of another base tip |
| **reset** | Reset current HEAD to the specified state |
| **switch** | Switch branches |
| **tag** | Create, list, delete or verify a tag object signed with GPG |

**collaborate (see also: git help workflows)**

| | |
|---|---|
| **fetch** | Download objects and refs from another repository |
| **pull** | Fetch from and integrate with another repository or a local branch |
| **push** | Update remote refs along with associated objects |

## 28. ORM

—> An ORM (Object-Relational Mapping) is a technique in Python that allows developers to interact with databases using Python objects and methods, instead of writing SQL statements directly.

—> This approach enables a more natural and readable way to interact with databases while also providing a higher level of abstraction, resulting in better code organization and maintainability.

—> Django comes with a built-in ORM called Django ORM, which provides a high-level interface to work with databases. Dajngo ORM provides a set of classes and functions that map python objects to database tables, making it easy to perform database operations using Python code.

—> With this example, you can use the ORM to create, retrieve, update and delete records in the database using the python code.