DJANGO MIDDLEWARE

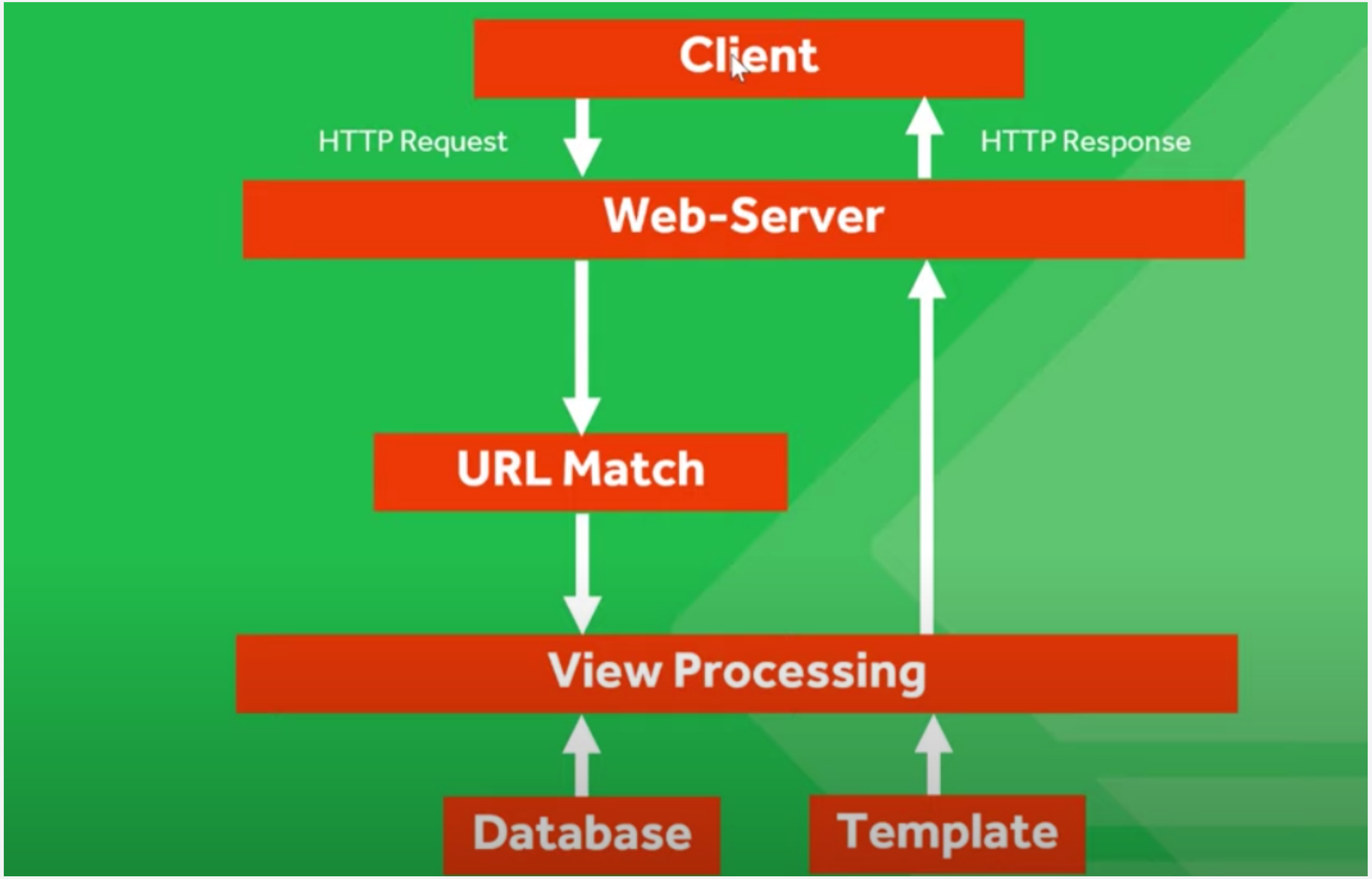
CONTENT

MIDDLEWARE

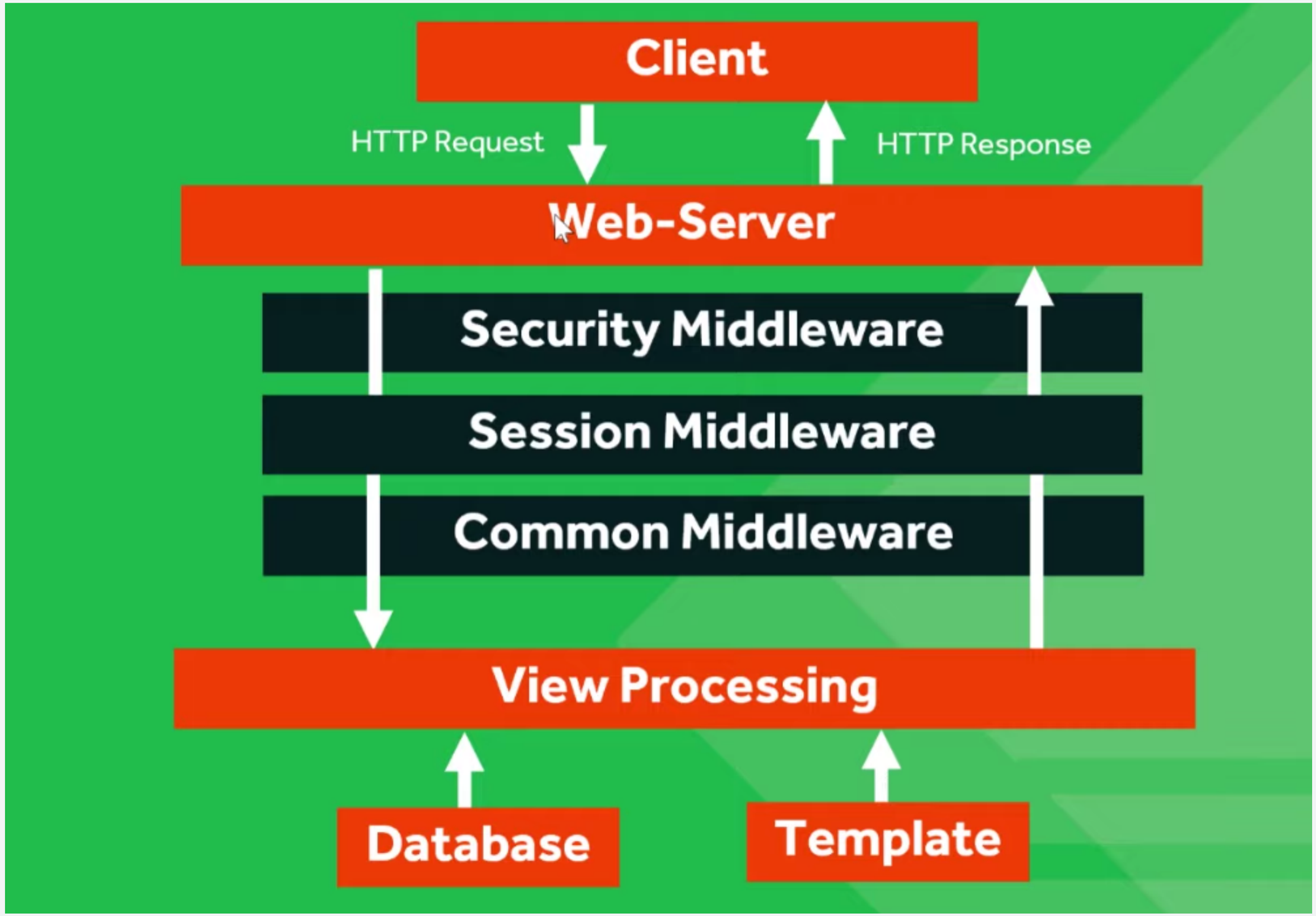
Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Middleware lies in between the web server and the django view function.

Work Flow


Client
HTTP Request
HTTP Response
Web-Server
URL Match
View Processing
Database
Template

Middleware comes into picture



USE CASES

Objective 1

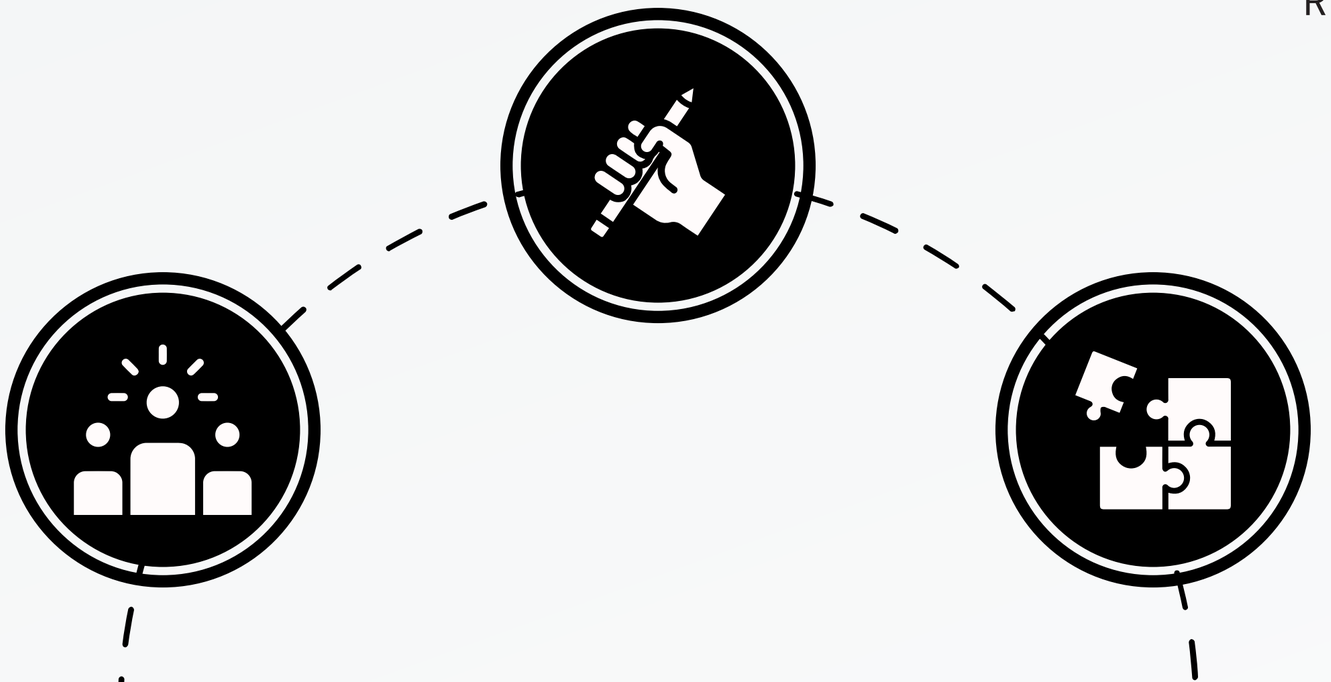
Filtering the Requests and Response across the entire application.

Objective 2

Injecting data or modifying the content in the Request and Response cycle.

Objective 3

Performing the transaction logs for the Request and Response cycle.



IMPLEMENTATION

- By default django has pre configured some of the middlewares in the settings.py file.
- We can create our custom middleware by class based or function based middleware.
- Order of the middleware must be preserved.
- Configure the custom middleware in settings.py
  - **'your_app.custom_middleware_file.CustomMiddleware_class'**
- Incoming request will pass through top to bottom order.
- Outgoing response will pass through bottom to top order.

Order of 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',
    'DemoProject.custom_middleware.CustomMiddleware',
]
```

Middleware hooks

- *perform _view*
- *perform_request*
- *perform_exception*
- *perform_response*
- *perform_template_response*

Class based middleware

```python
class CustomMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        self.request_count = 0
        self.exception_count = 0
        self.template_msg = {"message": "Data loaded Successfully"}

    def __call__(self, request):
        # Code block that is executed in each request before the view is called

        response = self.get_response(request)

        # Code block that is executed in each request after the view is called
        if hasattr(self, 'process_response'):
            response = self.process_response(request, response)
        return response

    # This code is executed just before the view is called
    2 usages (2 dynamic)
    def process_view(self, request, view_func, view_args, view_kwargs):
        """This is executed before a call to view"""
        self.request_count += 1
        print(f"Total request received {self.request_count}")

    # This code is executed if an exception is raised
    2 usages (2 dynamic)
    def process_exception(self, request, exception):
        self.exception_count += 1
        print(f"Exception Count: {self.exception_count}")

    def process_response(self, request, response):
        print("Process Response Called")
        return response
```

Function based middleware

```python
def custom_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.
        print("Called before the view function")

        response = get_response(request)

        # Code to be executed for each request/response after
        # the view is called.
        print("Called after the view function")

        return response

    return middleware
```

THANK YOU