Middlewares:

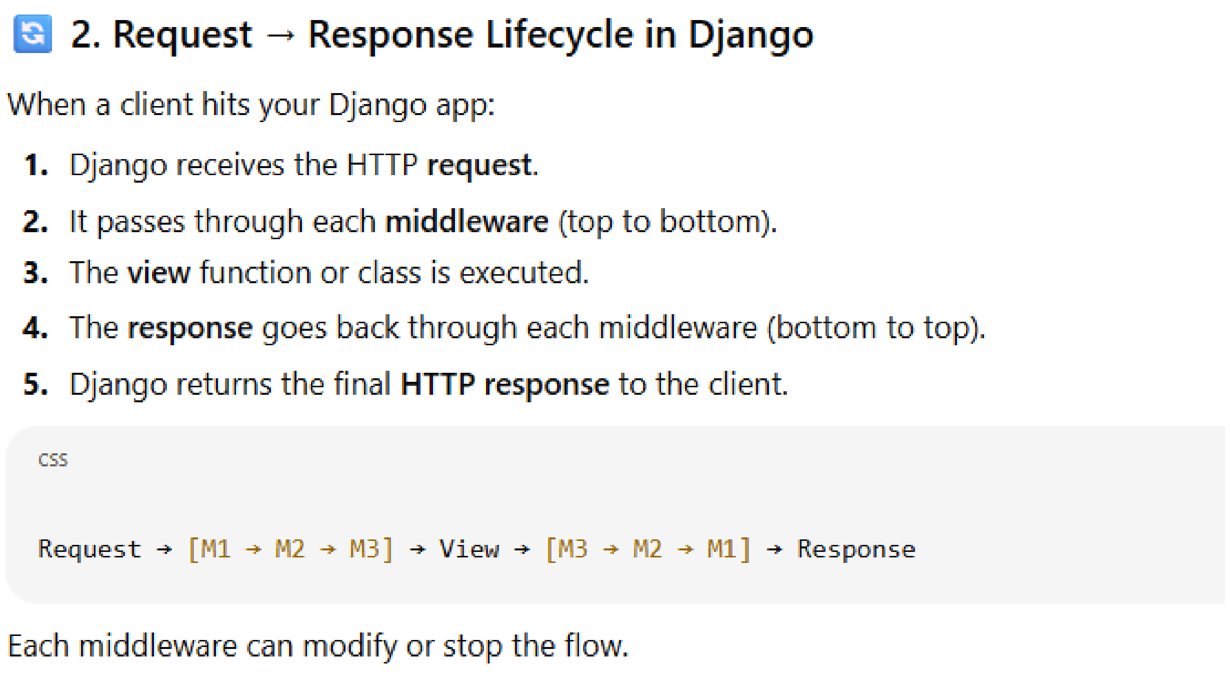
⚙️ 1. What is Middleware?

Middleware is a layer of logic that sits *between the request and response cycle* in Django.

Whenever a request comes in or a response goes out, middleware can intercept it and perform actions like:

- Processing, validating, or modifying requests/responses
- Handling authentication or security
- Managing sessions or CSRF protection
- Logging, throttling, caching, etc.

Think of middleware as a pipeline or a filter chain.



🔄 2. Request → Response Lifecycle in Django

When a client hits your Django app:

1. Django receives the HTTP **request**.
2. It passes through each **middleware** (top to bottom).
3. The **view** function or class is executed.
4. The **response** goes back through each middleware (bottom to top).
5. Django returns the final **HTTP response** to the client.

```css
Request → [M1 → M2 → M3] → View → [M3 → M2 → M1] → Response
```

Each middleware can modify or stop the flow.

🧠 3. When Are Middlewares Used?

You use middleware when you want to apply **common functionality globally** across all requests/responses, like:

Use Case	Description
Authentication	Identify logged-in users from cookies or tokens
Security	CSRF, XSS, Clickjacking protection
Session management	Handle user sessions
Performance	Add caching or request timing
Logging	Log request details globally
API Monitoring	Track request/response count, errors, etc.
Maintenance mode	Temporarily disable site access

🧩 4. Built-in Django Middlewares (Common Ones)

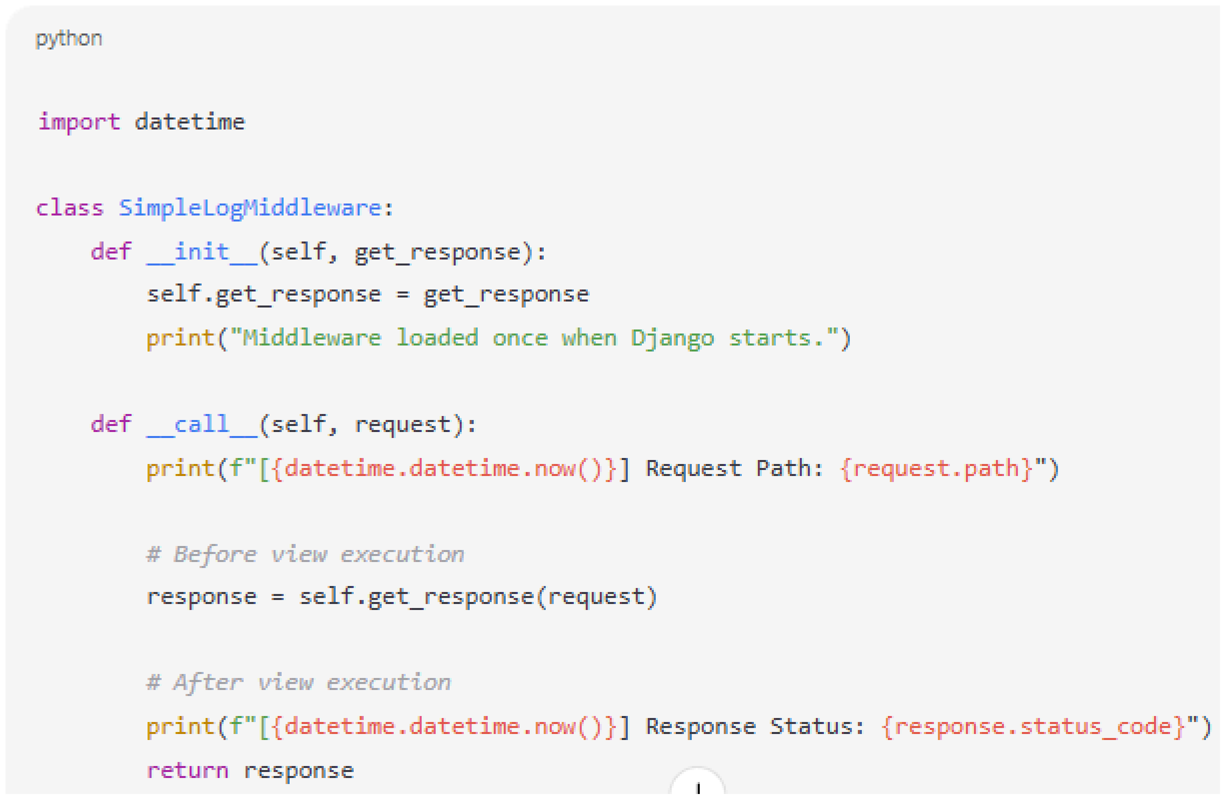
Middleware	Purpose
`AuthenticationMiddleware`	Associates users with requests
`SessionMiddleware`	Manages session data
`CsrfViewMiddleware`	Adds CSRF protection
`CommonMiddleware`	Adds standard headers, URL redirects
`SecurityMiddleware`	Adds HTTPS and security headers
`MessageMiddleware`	Handles messages between requests
`LocaleMiddleware`	Handles localization/internationalization
`CacheMiddleware`	Adds caching at middleware level

All these are defined in your `settings.py`:

Simple Custom Middleware Example

Let’s make a **request logger** middleware.

`middlewares.py`



python

```python
import datetime

class SimpleLogMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        print("Middleware loaded once when Django starts.")

    def __call__(self, request):
        print(f"[{datetime.datetime.now()}] Request Path: {request.path}")

        # Before view execution
        response = self.get_response(request)

        # After view execution
        print(f"[{datetime.datetime.now()}] Response Status: {response.status_code}")
        return response
```

🧪 7. Example 2 — Block Requests Based on IP

python

```python
from django.http import HttpResponseForbidden

class BlockIPMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        self.blocked_ips = ['192.168.1.10', '127.0.0.5']

    def __call__(self, request):
        ip = request.META.get('REMOTE_ADDR')
        if ip in self.blocked_ips:
            return HttpResponseForbidden("Access Denied")

        return self.get_response(request)
```

Add in `settings.py`

python

```python
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'yourapp.middlewares.SimpleLogMiddleware',   # 👈 add this
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
]
```

Now every request and response is logged automatically to the console.

⏱️ 8. Example 3 — Measure Request Time

python

```python
import time
from django.http import JsonResponse

class TimerMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        start = time.time()
        response = self.get_response(request)
        end = time.time()
        print(f"Time Taken: {end - start:.4f} seconds")
        return response
```

JWT Middleware (Instead of Decorator)

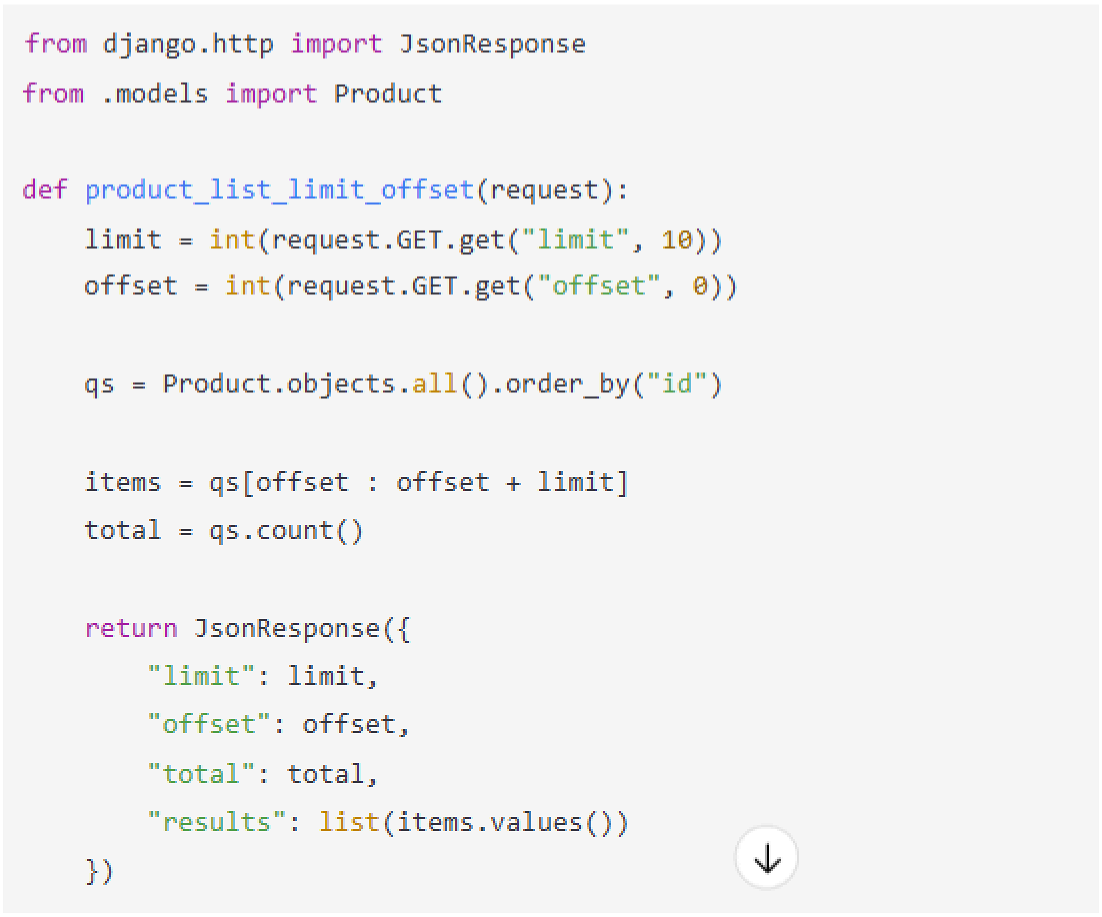
```python
class JWTAuthenticationMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        token = request.headers.get('Authorization')
        if token:
            try:
                data = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
                request.user = User.objects.get(id=data['user_id'])
            except Exception:
                request.user = None
        else:
            request.user = None

        response = self.get_response(request)
        return response
```

```python
from django.http import JsonResponse
from .models import Product

def product_list_limit_offset(request):
    limit = int(request.GET.get("limit", 10))
    offset = int(request.GET.get("offset", 0))

    qs = Product.objects.all().order_by("id")

    items = qs[offset : offset + limit]
    total = qs.count()

    return JsonResponse({
        "limit": limit,
        "offset": offset,
        "total": total,
        "results": list(items.values())
    })
```

⚖️ 12. Middleware vs Decorator vs Mixin

Feature	**Middleware**	**Decorator**	**Mixin**
Scope	Global (affects all views)	Per-view	Per-class-based-view
Works with	Entire project	Function-based view	Class-based view
When runs	Before/after view	Before/after specific view	Before dispatch of class
Example	`AuthenticationMiddleware`	`@login_required`	`LoginRequiredMixin`
Use case	Logging, security, sessions	Auth per view	Auth per CBV

Pagination:



```python
from django.http import JsonResponse
from .models import Product

def product_list_limit_offset(request):
    limit = int(request.GET.get("limit", 10))
    offset = int(request.GET.get("offset", 0))

    qs = Product.objects.all().order_by("id")

    items = qs[offset : offset + limit]
    total = qs.count()

    return JsonResponse({
        "limit": limit,
        "offset": offset,
        "total": total,
        "results": list(items.values())
    })
```