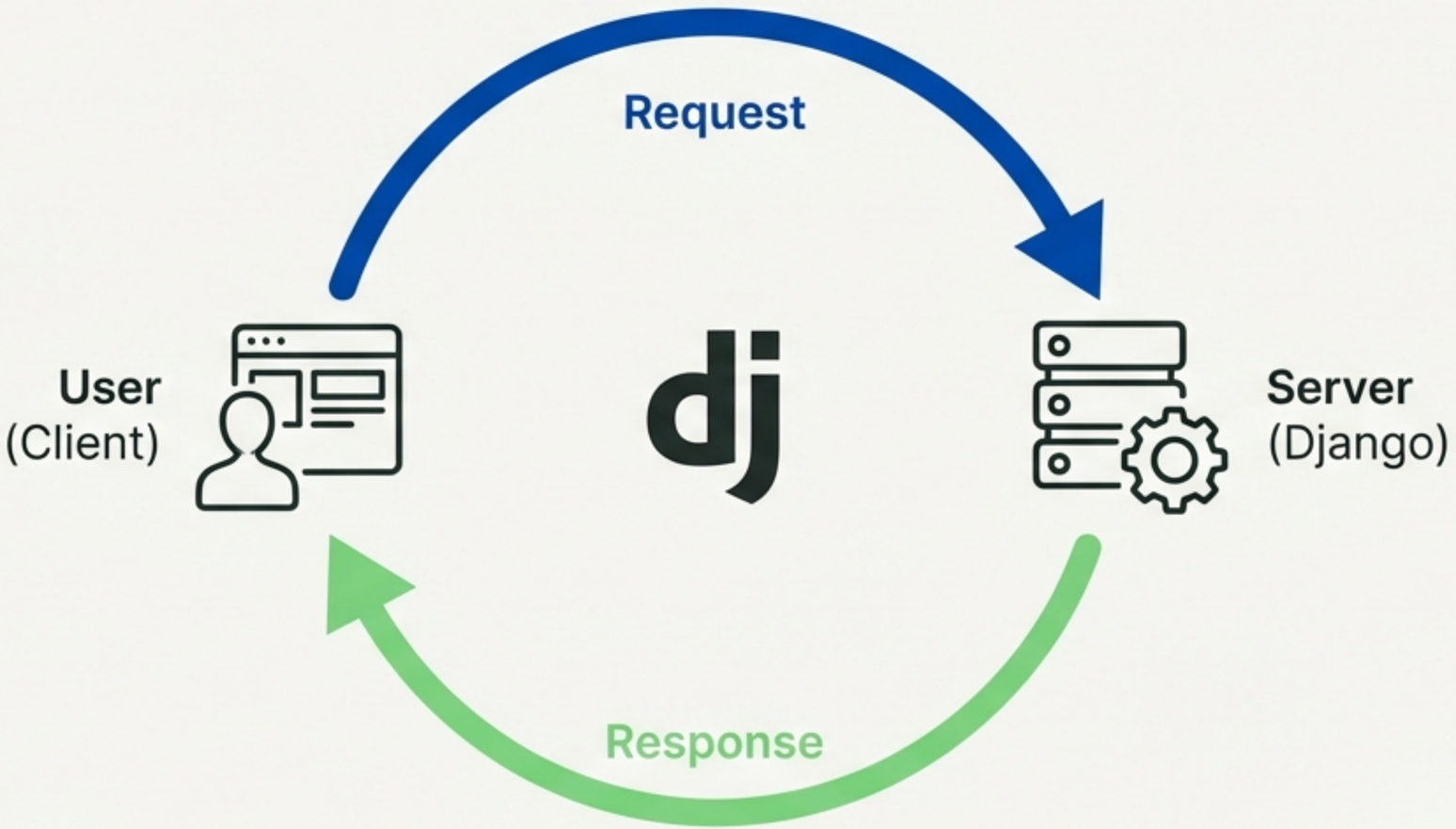
Django Architecture: From Request to Response

Mastering the Lifecycle, Routing, and the Strategic Choice Between Function and Class-Based Views

innovativeskills

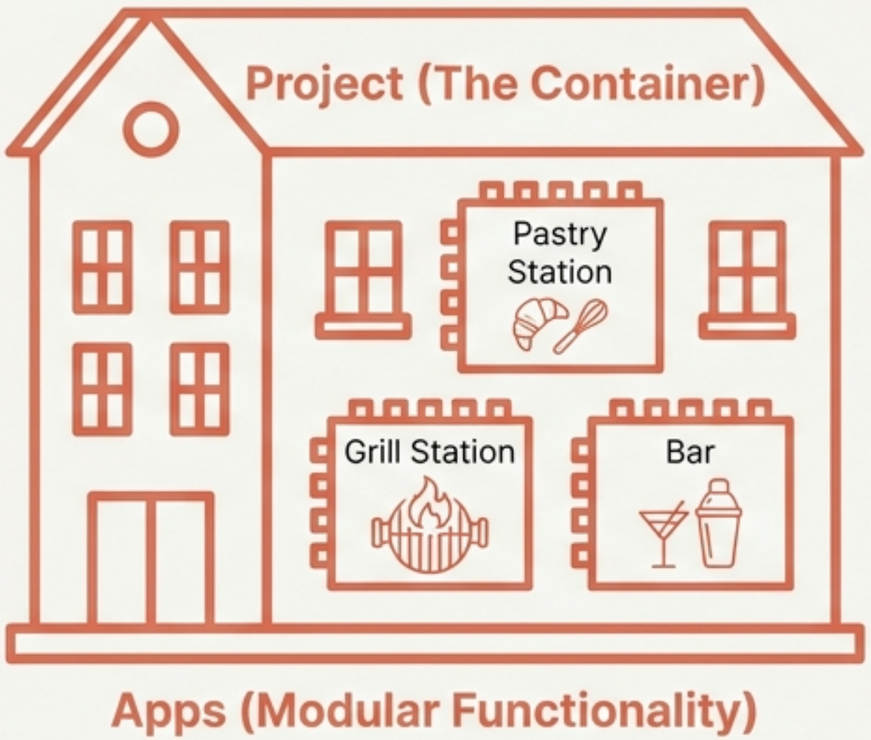


A comprehensive guide to the 'Batteries Included' framework.

Authored by Abdullah Al Mahmud

The Foundation: Projects vs. Apps

The Architecture



Django follows a “Batteries Included” philosophy. You configure the main building (settings.py), then build specific stations (apps) to handle the work.

The Directory Structure

```
myproject_root/
├── manage.py ← Command Center
└── myproject/ ← Project Configuration
    ├── settings.py ← Control Panel
    ├── urls.py ← Table of Contents
    └── wsgi.py ← Entry Door
```

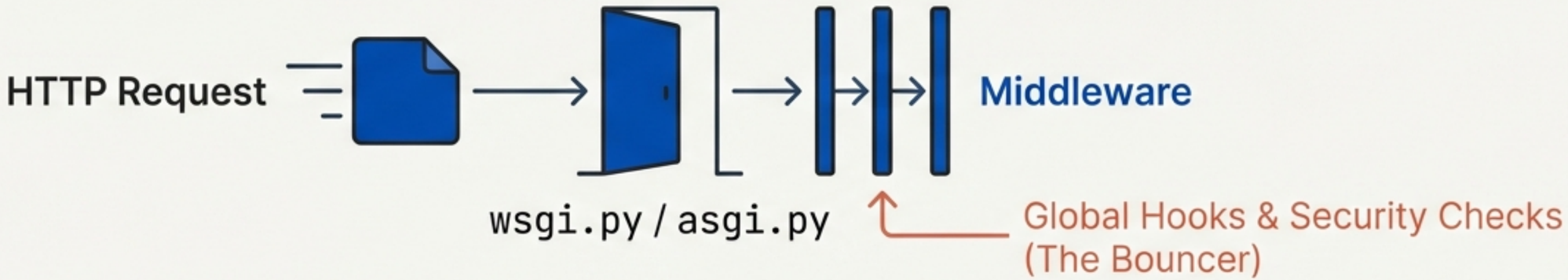
Authored by Abdullah Al Mahmud

The Lifecycle Map

The web is a conversation. We will trace the life of a single HTTP request through these five stages.

Step 1: The Incoming Request

The Entry Point



The HttpRequest Object

HttpRequest

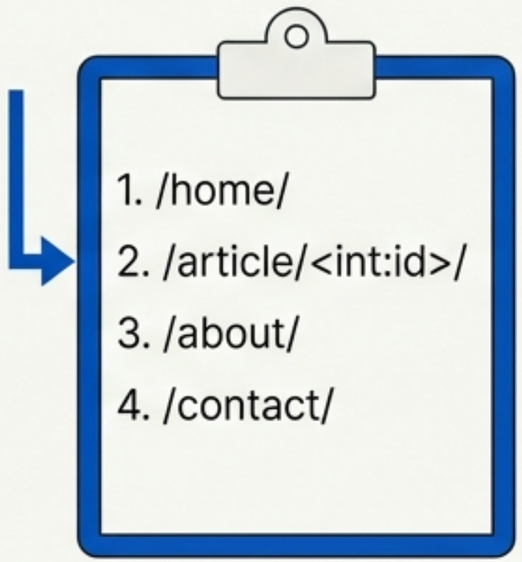
- `request.method: 'GET' or 'POST'`
- `request.user: <User: Alice>`
- `request.path: '/articles/10/'`
- `request.GET: { }`

Django automatically creates this metadata object when traffic hits the server. It contains everything we need to know about the user's intent.

Authored by Abdullah Al Mahmud

Step 2: Routing (The URL Dispatcher)

The Table of Contents



Django reads urlpatterns from top to bottom and stops at the first match.

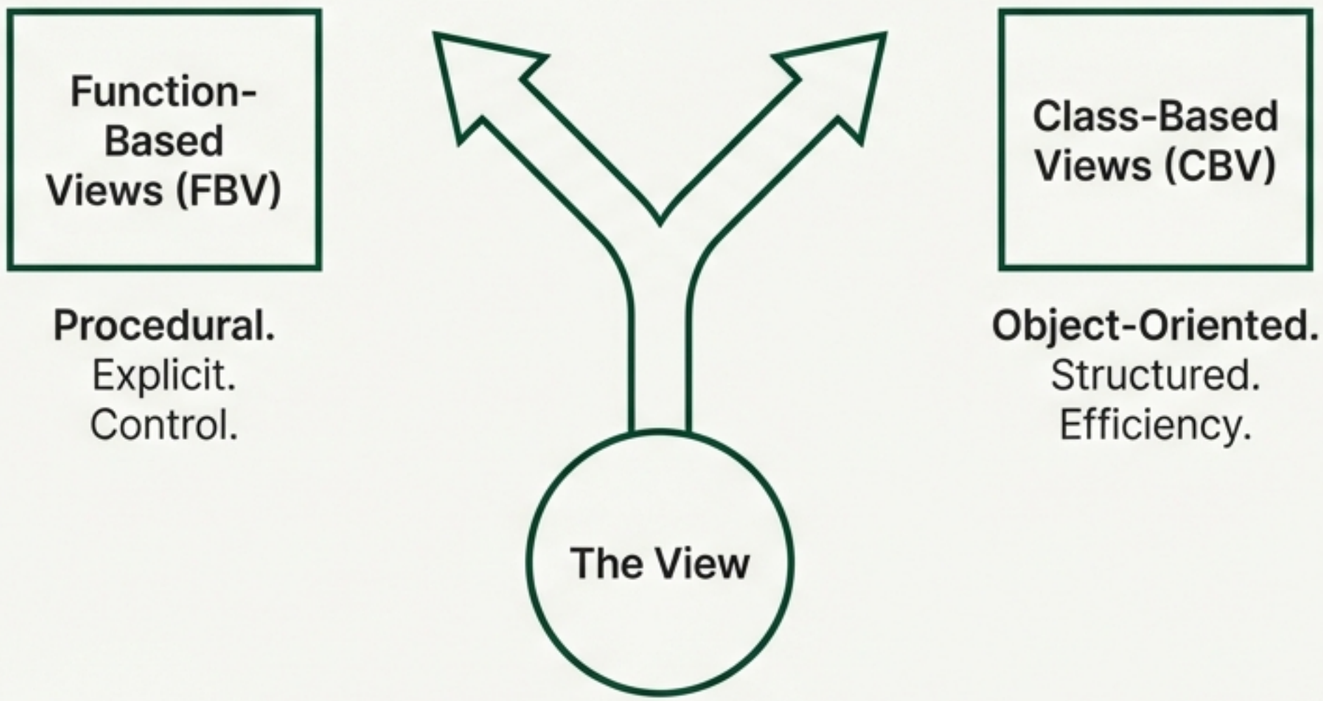
```
urlpatterns = [
    path('home/', views.home_view),
    path('article/<int:id>/', views.article_detail),
]
```

Path Converter. Captures the number from the URL (e.g., /article/42/) and passes '42' to the view as an argument.

Authored by Abdullah Al Mahmud

Step 3: The View (The Brains)

As logic grows, developers face a choice in architectural style.



Procedural.
Explicit.
Control.

Object-Oriented.
Structured.
Efficiency.

The Chef.
Accepts request, executes logic (DB queries), returns response.

Approach A: Function-Based Views (FBV)

The Explicit Approach

- **Simple** Python functions.
- **Total Control:** You measure and cut every piece yourself.
- **Manual Logic:** Must write if-statements for flow control.
- **Best for:** Learning, Debugging, Custom Logic.

```
def my_view(request):
    if request.method == 'POST':
        # Handle data submission
        return HttpResponse('Saved!')
    else:
        # Handle page load
        return render(request, 'page.html')
```

Manual Flow Control

Authored by Abdullah Al Mahmud

Approach B: Class-Based Views (CBV)

The Object-Oriented Approach

- Inherits from django.views.View.
- **Reusability**: Snap-together functionality (Mixins).
- **Implicit Logic**: Django automatically routes GET/POST to methods.
- **Abstraction**: Write less code for standard patterns.

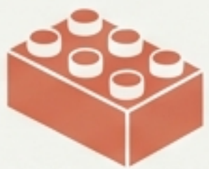
```
class MyView(View):
    def get(self, request):
        # Handle page load
        return render(request, 'page.html')

    def post(self, request):
        # Handle data submission
        return HttpResponse('Saved!')
```

Automatic Method Routing

Authored by Abdullah Al Mahmud

Head-to-Head: FBV vs. CBV

Feature	Function-Based (FBV)	Class-Based (CBV)
Reusability	Low. Requires helper functions.	High. Uses Inheritance & Mixins.
Logic Control	Explicit. You write every “if” statement.	Implicit. Logic abstracted in parent classes.
Best Use Case	Custom, complex flows.	Standard CRUD operations.
Analogy	Raw Lumber 	LEGO Set 

Decision Framework: When to Use Which?

Use Function-Based Views (FBV) when:

1. You are learning the fundamentals.
2. You need to debug complex logic flow top-to-bottom.
3. The logic is non-standard (e.g., emailing users, updating APIs, and querying legacy DBs simultaneously).

Use Class-Based Views (CBV) when:

1. You are building standard CRUD apps (Blogs, Inventories).
2. You need to DRY (Don’t Repeat Yourself) across multiple views.
3. You want to write significantly less code.

Authored by Abdullah Al Mahmud

Step 4: The Outgoing Response

Plating the Meal

You don’t hand the customer a raw steak. It **must** be **plated** and **presented** correctly.

HttpResponse Object

Status Code: 200 OK
Headers: Content-Type: text/html
Body: <html>...</html>

The View returns this object. The **server** translates it into the **HTTP protocol** the browser understands. →

Authored by Abdullah Al Mahmud