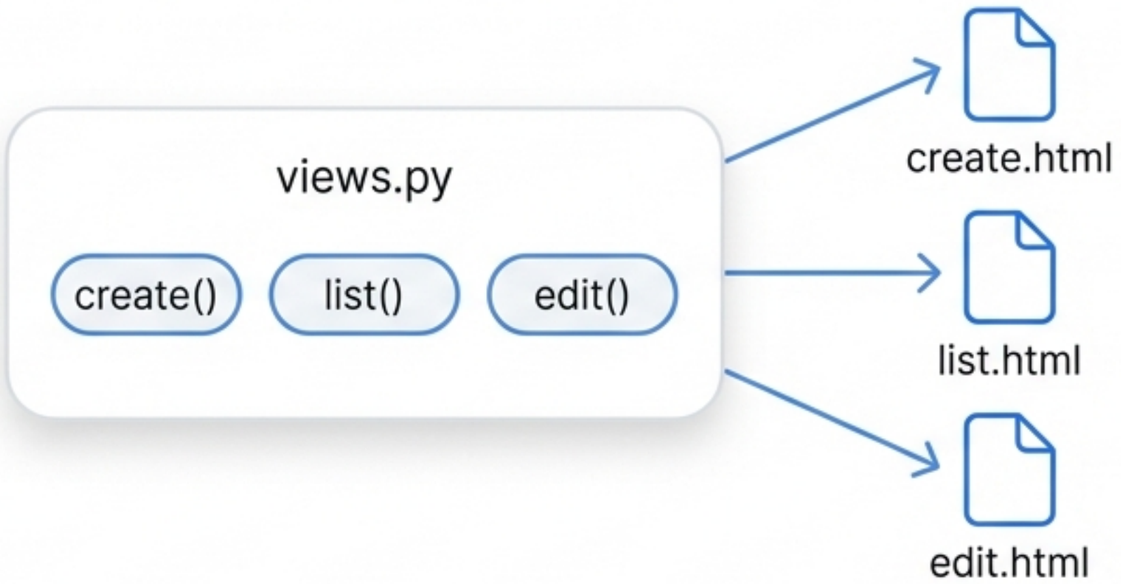
From Repetition to Refinement: Mastering Django Templates

A step-by-step guide to building clean, maintainable web applications.

The Starting Point: A Simple But Flawed App

We begin with a basic ‘Movie DB’ project with three core functions: creating, listing, and editing movie entries.




```python
# views.py
def create(request):
    return render(request, 'create.html')

def list(request):
    return render(request, 'list.html')

def edit(request):
    return render(request, 'edit.html')
```

The Problem: Widespread Code Duplication

Each HTML file is completely independent. This means the navigation menu and basic HTML structure (<html>, <head>, <body>) are copied and pasted across all three files.

A simple change to the menu requires editing three separate files. This is inefficient and error-prone.

create.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Movie DB</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <nav>
        <a href="/create">Create</a>
        <a href="/list">List</a>
        <a href="/edit">Edit</a>
    </nav>
    <h1>Create Movie</h1>
    <!-- Create movie form goes here -->
</body>
</html>
```

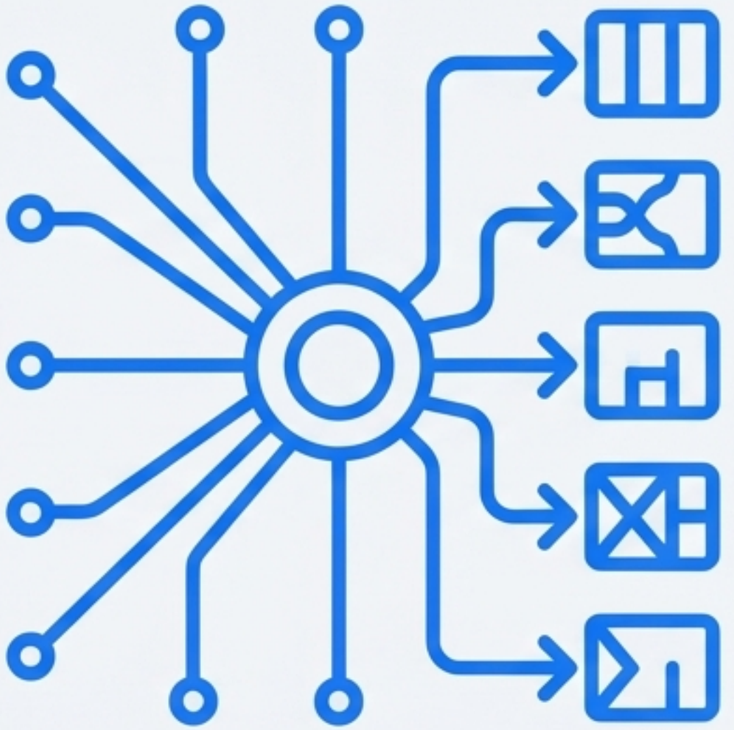
list.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Movie DB</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <nav>
        <a href="/create">Create</a>
        <a href="/list">List</a>
        <a href="/edit">Edit</a>
    </nav>
    <h1>Movie List</h1>
    <!-- Movie list content goes here -->
</body>
</html>
```

Step 1: Centralizing URLs for Maintainability

Instead of loosely connecting views to templates, Django provides a robust URL dispatcher. This lets us create a single, clear map of our application's URLs.

Your URL structure becomes clean, predictable, and easy to manage from one central location: `urls.py`.

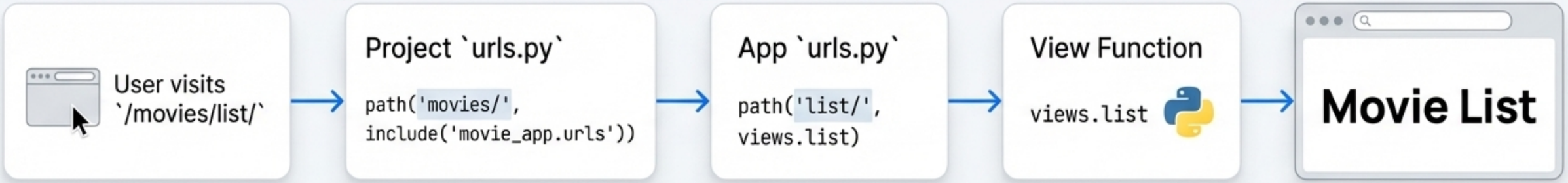


Implementing URL Patterns in `urls.py`

```python
# movie_app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    # Maps the URL '/create/' to the create() function in views.py
    path('create/', views.create, name='create'),

    # Maps the URL '/list/' to the list() function
    path('list/', views.list, name='list'),

    # Maps the URL '/edit/' to the edit() function
    path('edit/', views.edit, name='edit'),
]
```

- We import the `path` function and our `views`.
- The `urlpatterns` list holds all the URL patterns for this app.
- Each `path()` function links a URL route (e.g., `list/`) to a specific view function (e.g., `views.list`).

How a Request Flows Through Django's URL System

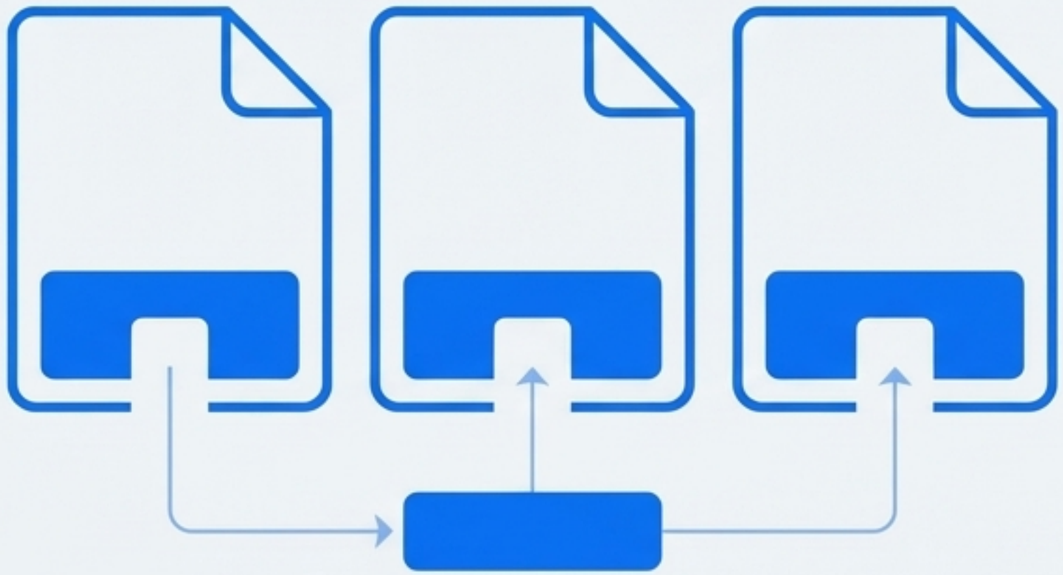


Step 2: Tackling Template Repetition with `{% include %}`

Our navigation menu is still hardcoded in every single template.

We can extract the repetitive HTML into its own file and use Django's `include` tag to pull it into any template that needs it.

Think of it as a reusable component or a building block for your UI.



Before & After: Implementing the `include` Tag

Before

`create.html` (Original)

```
...
<body>
    <nav>
        <a href="/list/">Movie List</a>
        <a href="/create/">Add Movie</a>
    </nav>
    <h1>Create a new movie entry</h1>
    ...
</body>
```

After

`menu.html` (New File)

```
<nav>
    <a href="/list/">Movie List</a>
    <a href="/create/">Add Movie</a>
</nav>
```

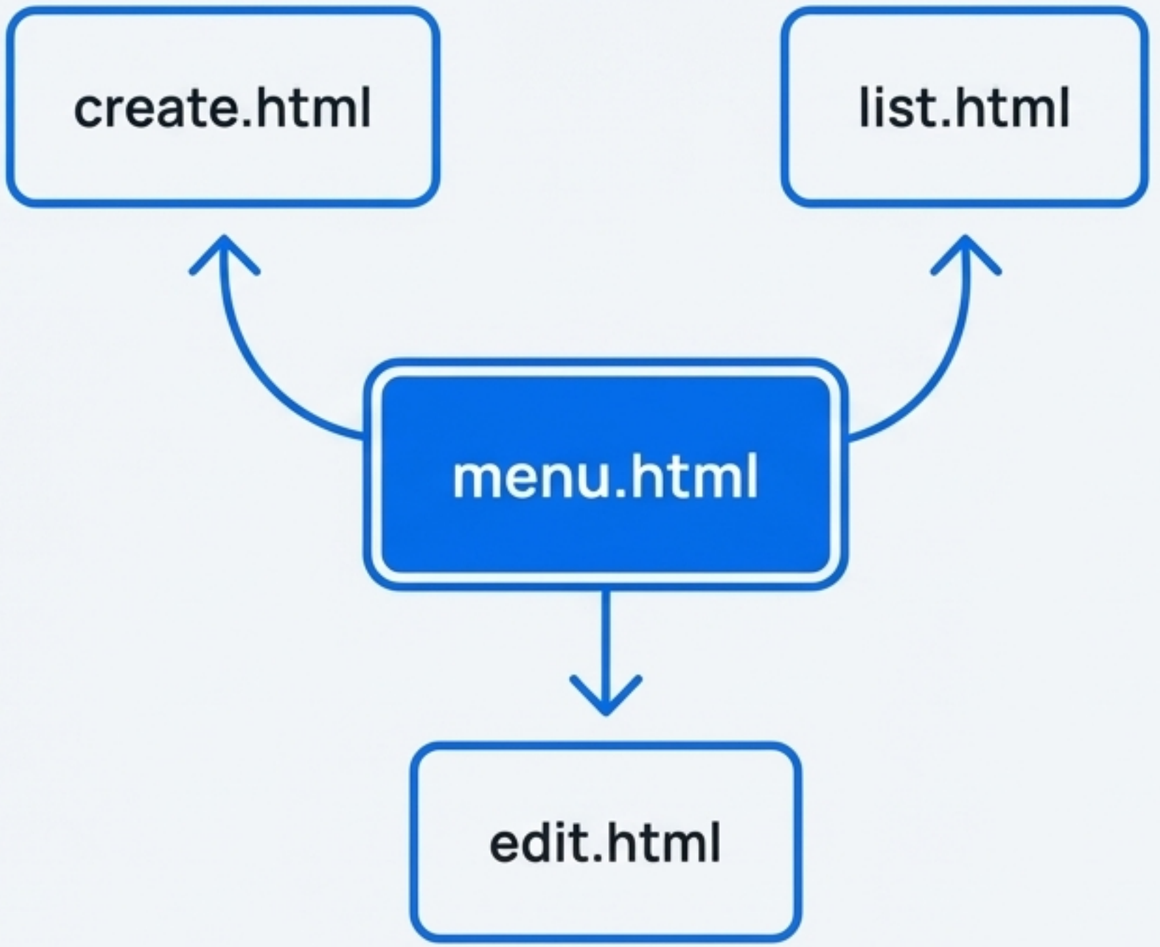
`create.html` (Refactored)

```
...
<body>
    {% include 'menu.html' %}
    <h1>Create a new movie entry</h1>
    ...
</body>
```

The Power of a Single Source of Truth

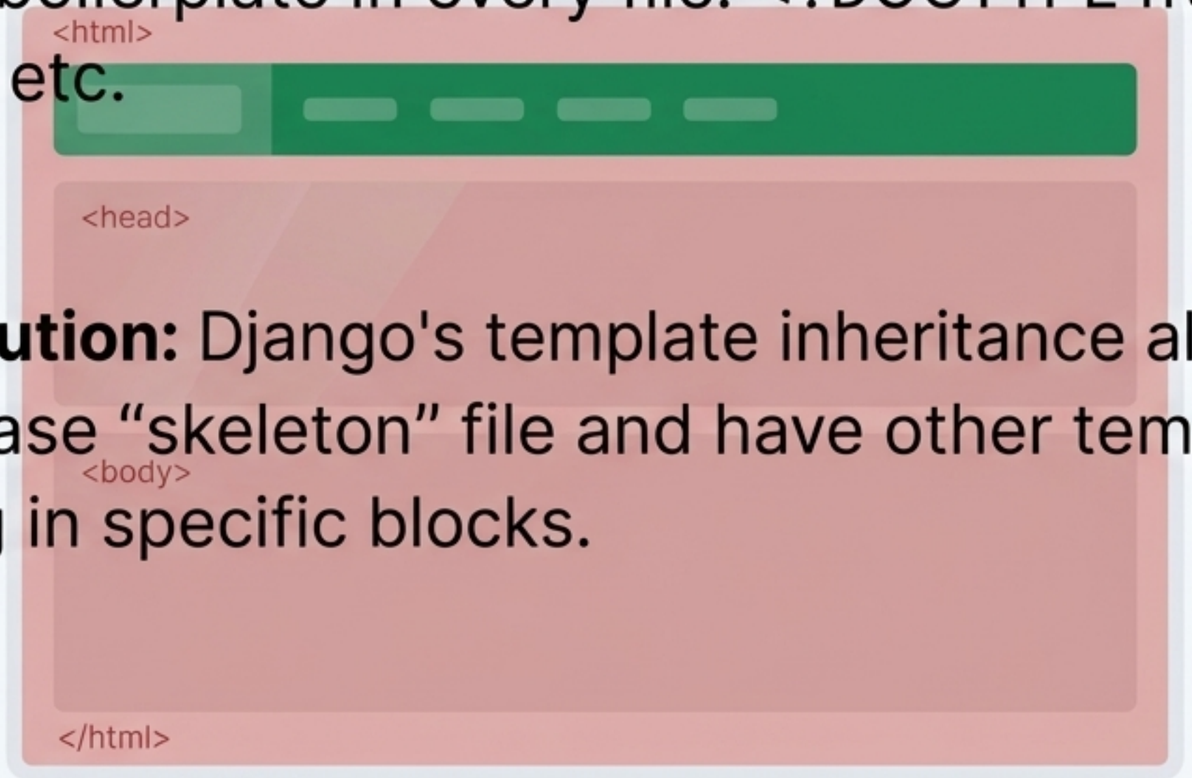
We have now achieved the “Don’t Repeat Yourself’ (DRY) principle for our navigation.

- **Maintainability**: To update the navigation, we now only need to edit one file: `menu.html`.
- **Consistency**: Every page that includes the menu is guaranteed to be identical.
- **Readability**: Our main templates (`create.html`, etc.) are now cleaner and focused on their specific content.



The Final Frontier: Full Template Inheritance

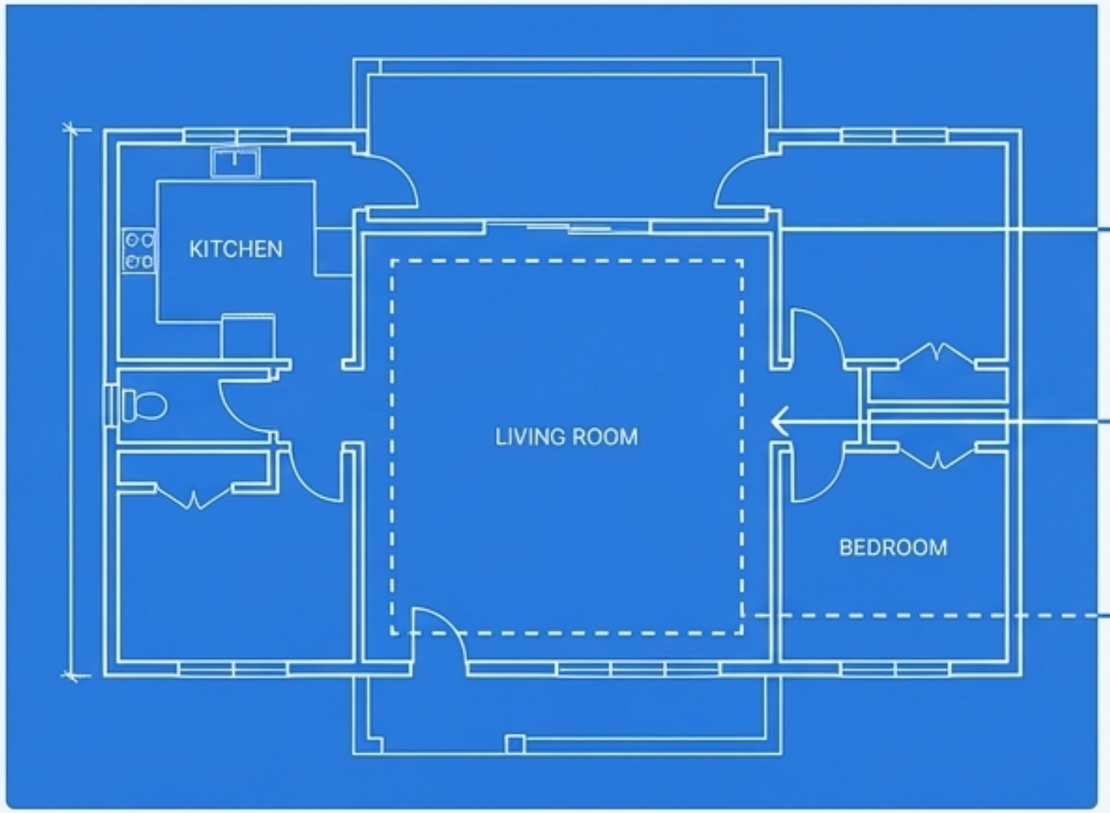
The Lingering Problem: While the menu is fixed, we are still repeating the entire HTML boilerplate in every file: `<!DOCTYPE html>`, `<html>`, <head>, <body>, etc.



The Ultimate Solution: Django's template inheritance allows us to define a single base “skeleton” file and have other templates “extend” it, filling in specific blocks.

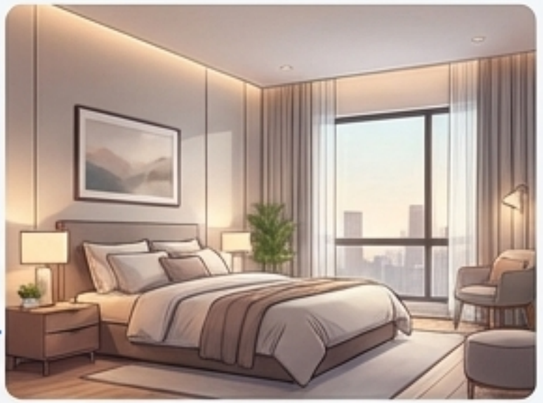
The Blueprint and The Rooms: Understanding Inheritance

`layout.html` (The Blueprint)



Defines the shared structure: the foundation, walls, and roof (<html>, <body>, header, footer). It leaves empty spaces for rooms ({% block %}).

`create.html`, `list.html` (The Finished Rooms)



These files **inherit** the structure from the blueprint and **simply fill in the empty spaces** with their own unique content (furniture).

Building the Blueprint: `layout.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Movie DB</title>
</head>
<body>
    {% include 'menu.html' %}

    <main>
        {# This block is a placeholder for content from child templates. #}
        {% block content %}
        {% endblock %}
    </main>

</body>
</html>
```

This is the 'empty space'. Child templates will insert their unique content here.

Before & After: Extending the Base Template

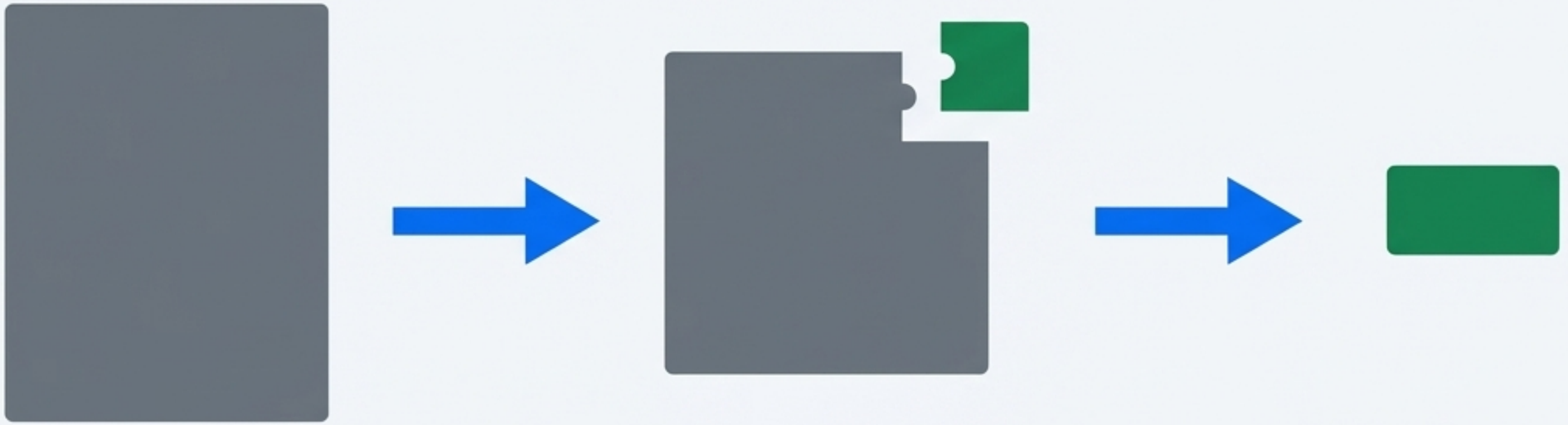
`create.html` (with `include`)

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    {% include 'menu.html' %}
    <h1>Create a new movie entry</h1>
    ...
  </body>
</html>
```

`create.html` (with `extends`)

```
{% extends 'layout.html' %}

{% block content %}
    <h1>Create a new movie entry</h1>
    <!-- Form fields go here... -->
{% endblock %}
```

The Journey of Refinement: A Visual Summary



Standalone

30+ Lines of Repetitive Code

With `include`

Menu is reusable, but boilerplate remains.

With `extends`

5 Lines of Focused Content. Clean, DRY, and Maintainable.

We've built a scalable, maintainable, and elegant front-end structure.

Key Principles for Your Django Projects

Centralize Logic with `urls.py`

Keep your URL routing clean and predictable. A single map is easier to read and maintain than scattered links.

Componentize UI with `{% include %}`

For small, reusable elements like navbars, footers, or sidebars, `include` is your best tool. Don't repeat UI components.

Inherit Structure with `{% extends %}`

This is the foundation of powerful Django templating. Use a base template to enforce site-wide consistency and eliminate boilerplate code.