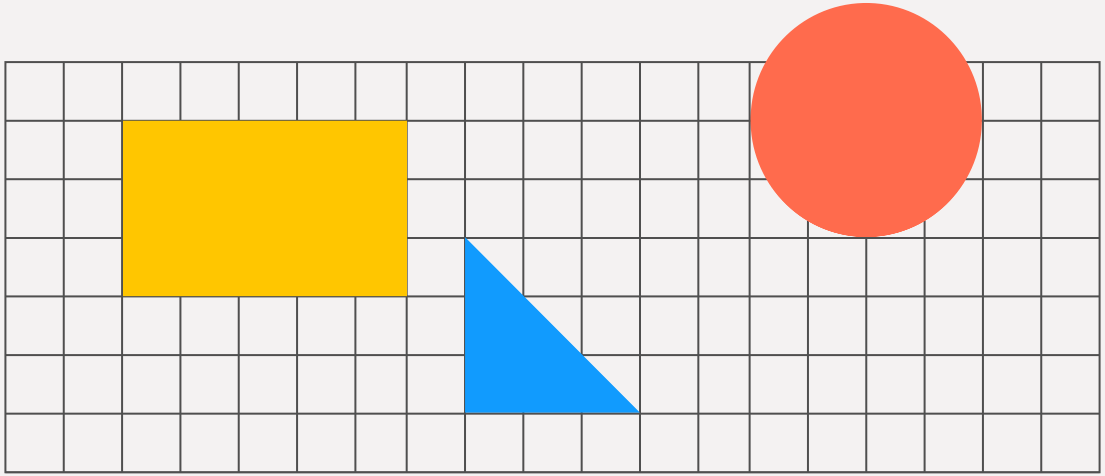
Django Views & Templates

Ali Abrishami



In this Lecture You will…

a

Get to know Django Templates

b

Explore Django's Admin Panel

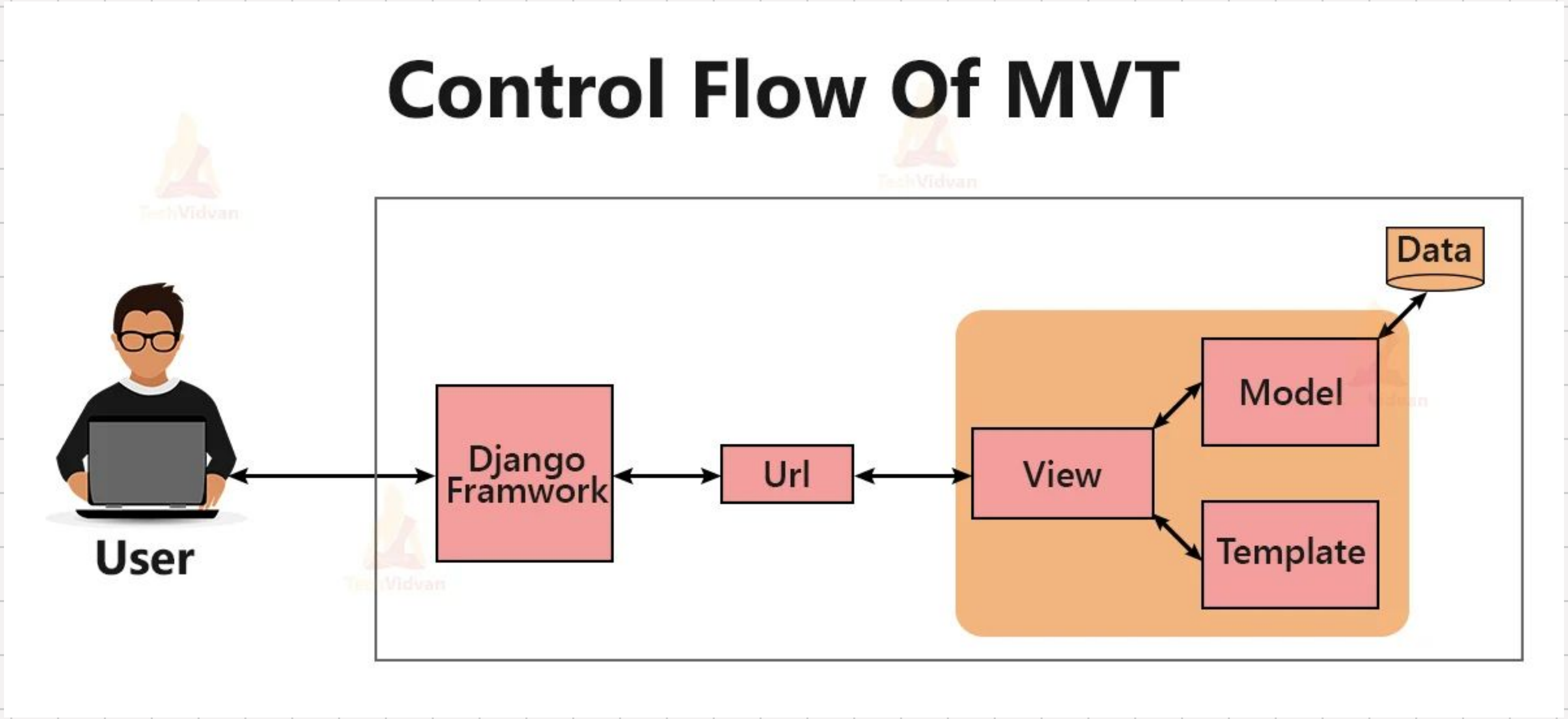
c

Learn Django development best practices

What is a View?

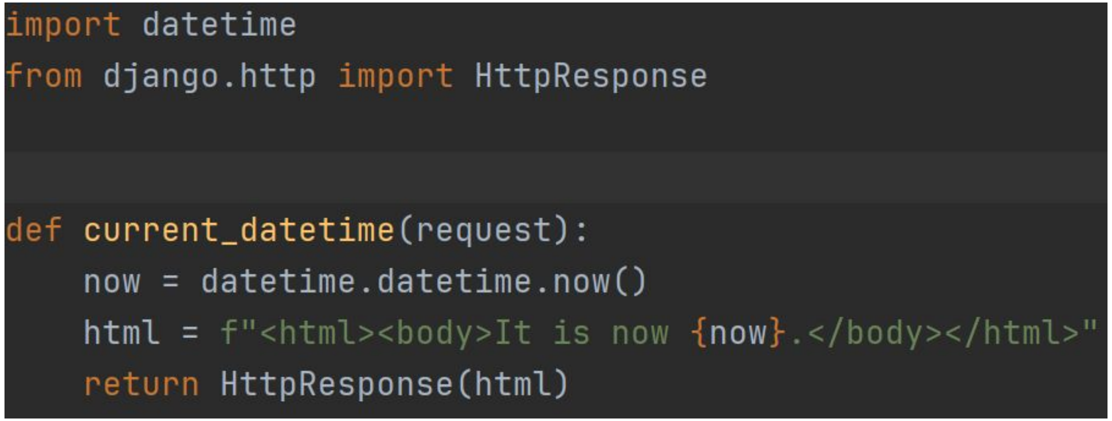
- A view is a function or class that receives a web **request** and returns a web **response**.
- A view is Python code that:
  - Takes an HTTP request from the user,
  - Processes data or interacts with the database,
  - Returns an HTTP response (HTML, JSON, redirect, file, etc.).
- Two types of views:
  - Function-Based Views
  - Class-Based Views

Url, View, Template



Function-Based Views

- A Function-Based View (FBV) in Django is a view written as a simple Python function.
- It receives an HTTP request (`HttpRequest`) and returns an HTTP response (`HttpResponse`).



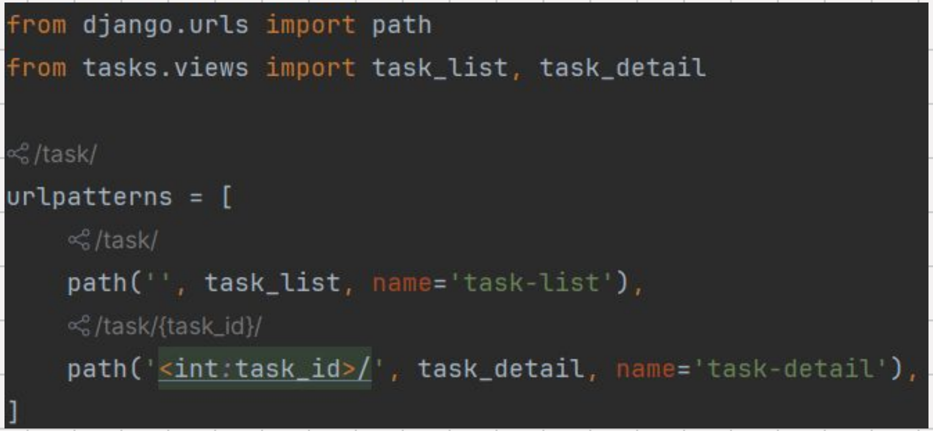
```python
import datetime
from django.http import HttpResponse


def current_datetime(request):
    now = datetime.datetime.now()
    html = f"<html><body>It is now {now}.</body></html>"
    return HttpResponse(html)
```

URLconfs

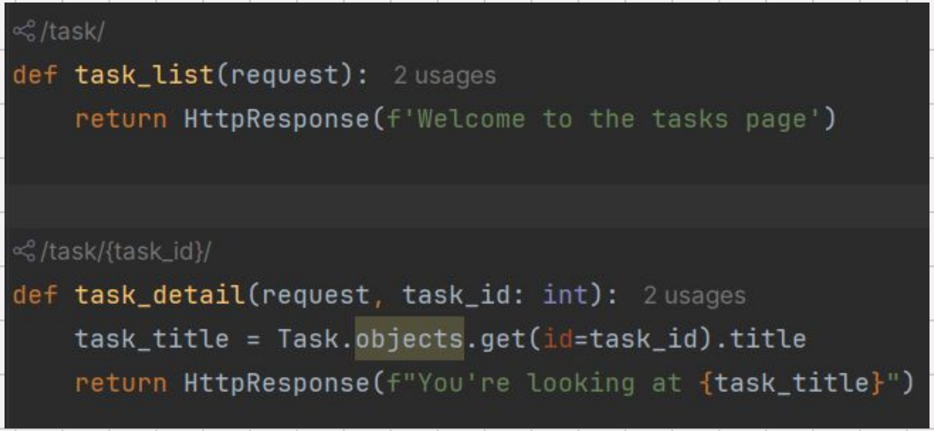
- Django will choose a view by examining the URL that's requested.
- To get from a URL to a view, Django uses what are known as '**URLconfs**'.
- A URLconf maps URL patterns to views.
- URL patterns can be named so you can refer to them in templates or redirects:
  - <a href="{% url 'task_detail' pk=task.id %}">
  - return redirect('task_detail', pk=task.id)

Create tasks URLs



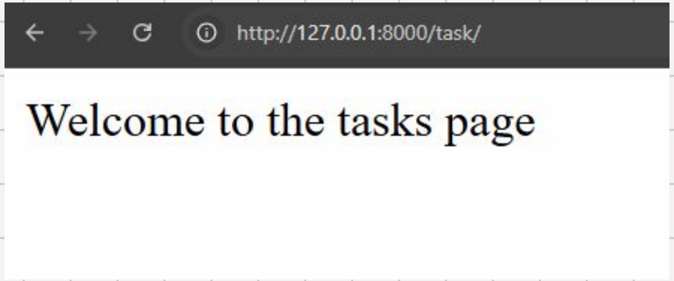
```
from django.urls import path
from tasks.views import task_list, task_detail

/task/
urlpatterns = [
    /task/
    path('', task_list, name='task-list'),
    /task/{task_id}/
    path('<int:task_id>/', task_detail, name='task-detail'),
]
```



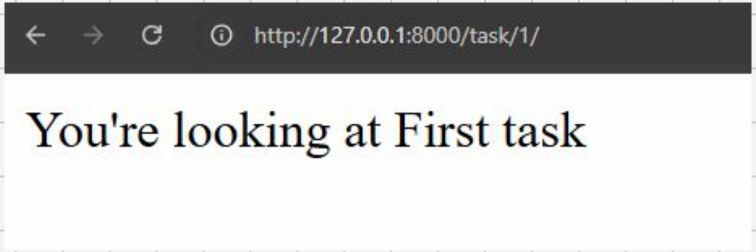
```
/task/
def task_list(request):  2 usages
    return HttpResponse(f'Welcome to the tasks page')


/task/{task_id}/
def task_detail(request, task_id: int):  2 usages
    task_title = Task.objects.get(id=task_id).title
    return HttpResponse(f"You're looking at {task_title}")
```



http://127.0.0.1:8000/task/

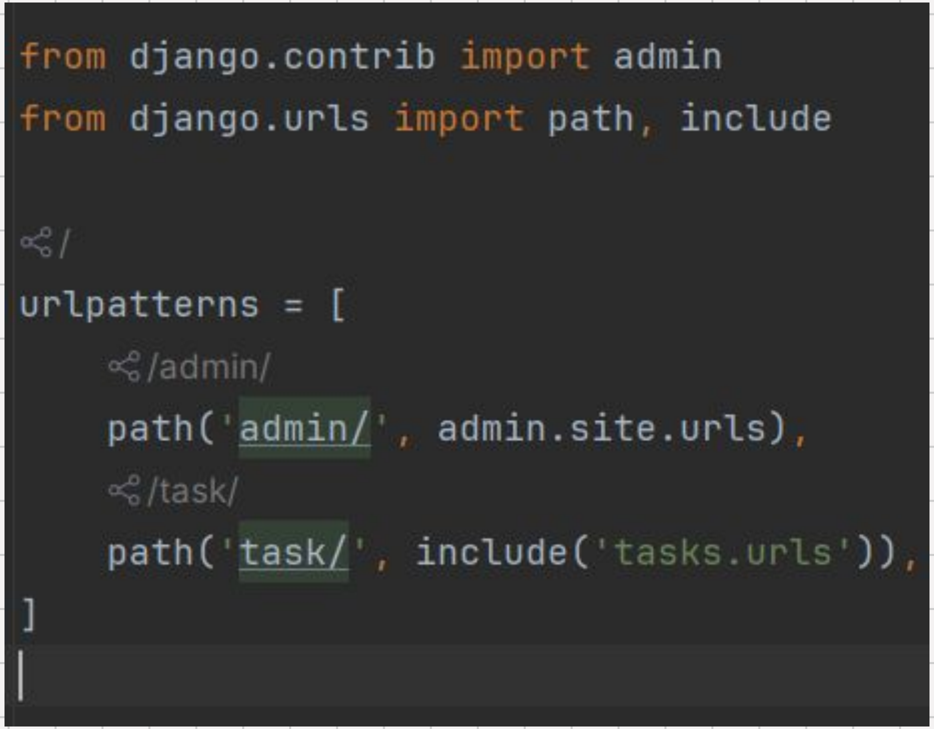
Welcome to the tasks page



http://127.0.0.1:8000/task/1/

You're looking at First task

Include tasks URLs



```
from django.contrib import admin
from django.urls import path, include


/
urlpatterns = [
    /admin/
    path('admin/', admin.site.urls),
    /task/
    path('task/', include('tasks.urls')),
]
```

Catching URL Parameters

- URL patterns can capture variables:
  - path('task/<int:id>/', views.task_detail)
- Types include:
  - `<int:pk>`
  - `<slug:slug>`
  - `<str:username>`
  - `<uuid:id>`

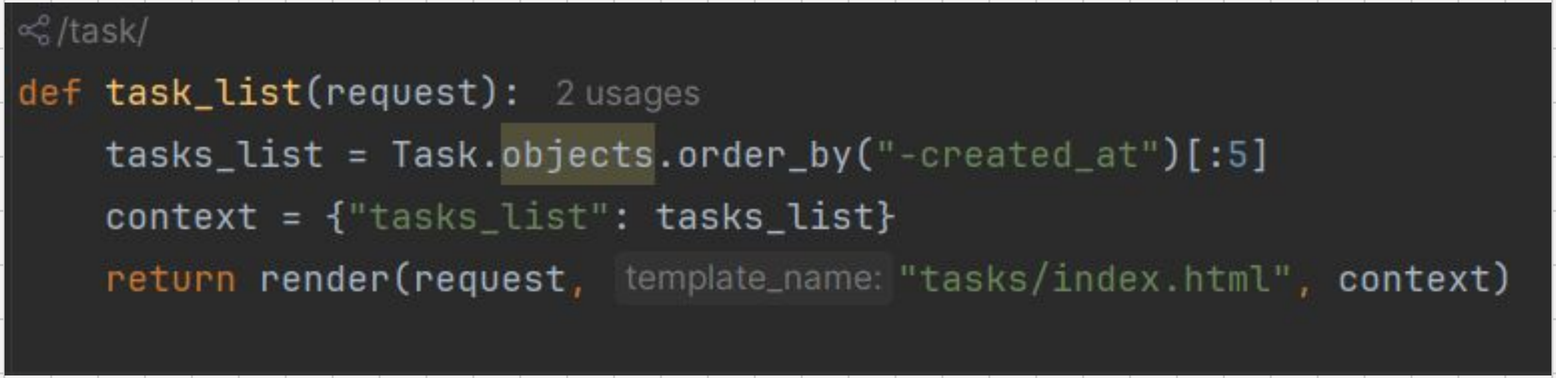
That’s Not Enough!

- No proper website contains text and text only!
- You need more.
- How to render HTML views with our Django app?

Templates System

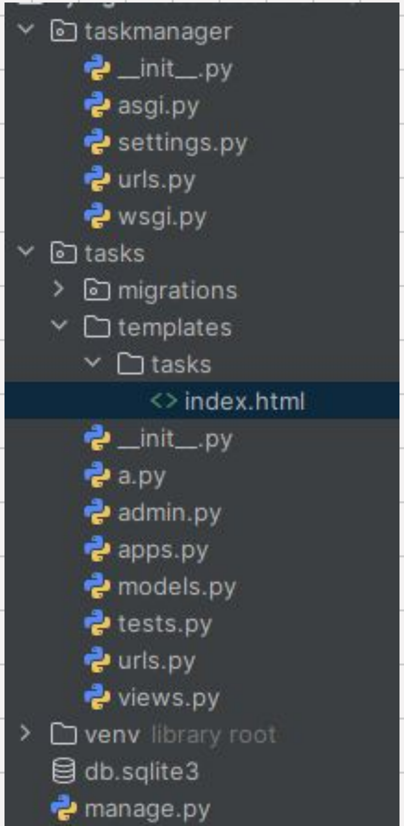
- Being a web framework, Django needs a convenient way to generate HTML **dynamically**.
- The most common approach relies on templates.
- A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.
- A Django template is a text document or a Python string marked-up using the Django template language. Some constructs are recognized and interpreted by the template engine. The main ones are **variables** and **tags**.
- By convention DjangoTemplates looks for a “templates” subdirectory in each of the INSTALLED_APPS.

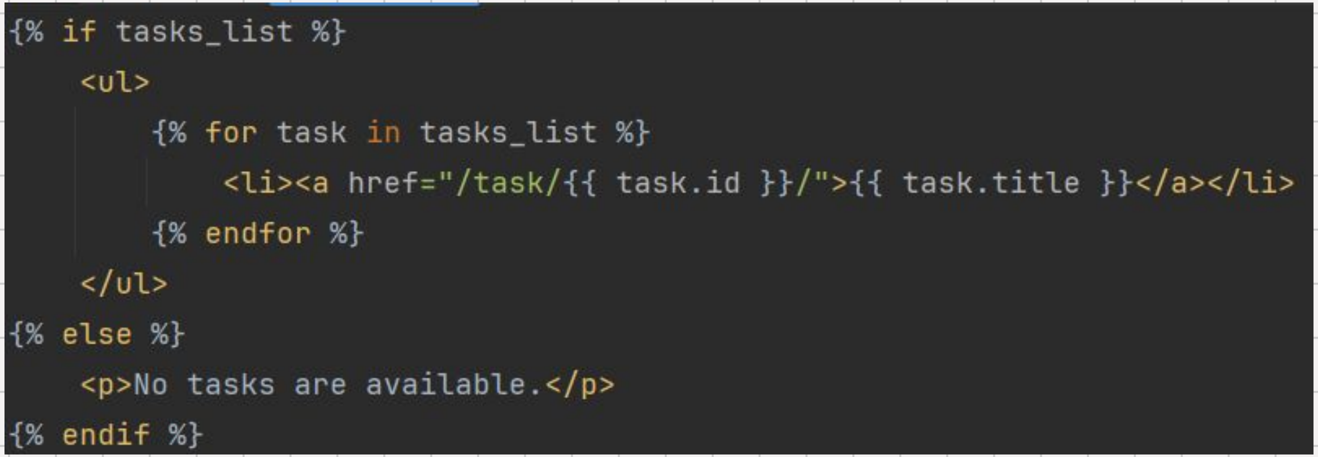
Update The View



```
/task/
def task_list(request):  2 usages
    tasks_list = Task.objects.order_by("-created_at")[:5]
    context = {"tasks_list": tasks_list}
    return render(request, template_name: "tasks/index.html", context)
```

Create A Template


```
taskmanager
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
tasks
    migrations
    templates
        tasks
            index.html
    __init__.py
    a.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
venv library root
db.sqlite3
manage.py
```

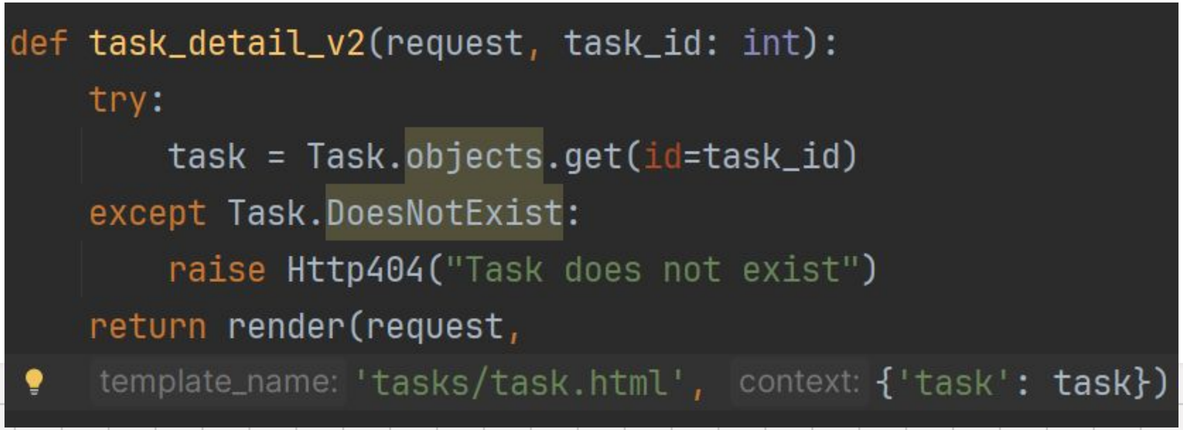

```
{% if tasks_list %}
    <ul>
        {% for task in tasks_list %}
            <li><a href="/task/{{ task.id }}/">{{ task.title }}</a></li>
        {% endfor %}
    </ul>
{% else %}
    <p>No tasks are available.</p>
{% endif %}
```

Some Basic Views

- Choosing your views is an important step!
- `task_list`: show all tasks (or user's tasks)
- `task_detail`: show a single task
- `task_create`: add a new task
- `task_update`: edit an existing task
- `task_delete`: delete a task

Handling 404s & Errors

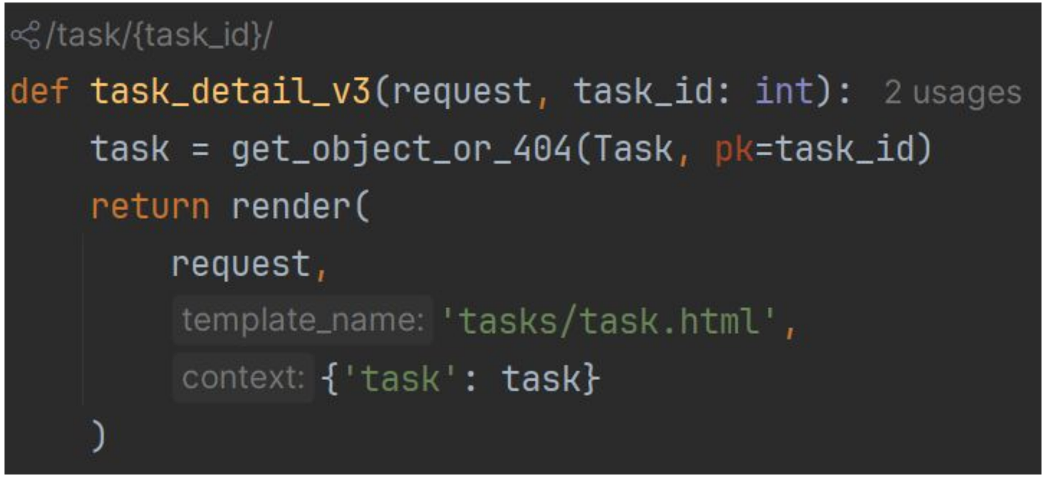
- Each view is responsible for doing one of two things:
    - Returning an `HttpResponse` object containing the content for the requested page,
    - Or raising an exception such as `Http404`.



```python
def task_detail_v2(request, task_id: int):
    try:
        task = Task.objects.get(id=task_id)
    except Task.DoesNotExist:
        raise Http404("Task does not exist")
    return render(request,
                  template_name: 'tasks/task.html', context: {'task': task})
```

A Better Way!

- You can use `get_object_or_404` to raise the `Http404` exception when the object doesn't exist instead of handling `DoesNotExist` and raising `Http404` by yourself.

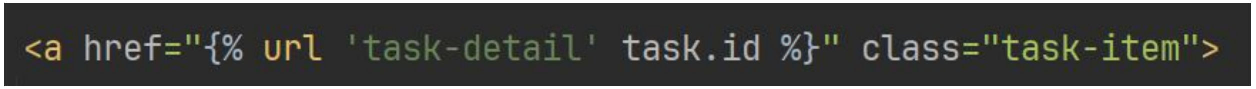


```
/task/{task_id}/
def task_detail_v3(request, task_id: int):  2 usages
    task = get_object_or_404(Task, pk=task_id)
    return render(
        request,
        template_name: 'tasks/task.html',
        context: {'task': task}
    )
```

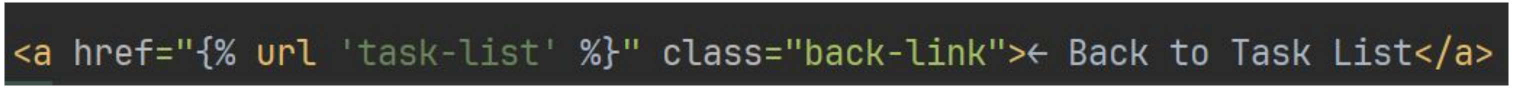
Removing hardcoded URLs in templates

- `{% url %}` Returns an absolute path reference (a URL without the domain name) matching a given view and optional parameters.

- This is a way to output links without violating the DRY principle by having to hardcode URLs in your templates.



```
<a href="{% url 'task-detail' task.id %}" class="task-item">
```



```
<a href="{% url 'task-list' %}" class="back-link">← Back to Task List</a>
```

HttpRequest

- The `HttpRequest` object represents everything the client (browser) sends to your Django app.
- Django automatically creates this object and passes it as the first argument to your view.
- What does `HttpRequest` contain?

  1. URL and path information
     - request.path — the URL path (e.g., /tasks/3/)
     - request.method — HTTP method: GET, POST, etc.

  2. Query parameters
     - request.GET — data sent in the URL
     - request.POST — data submitted via forms

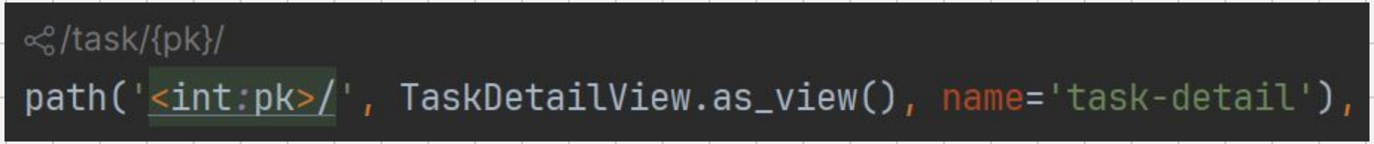
HttpResponse

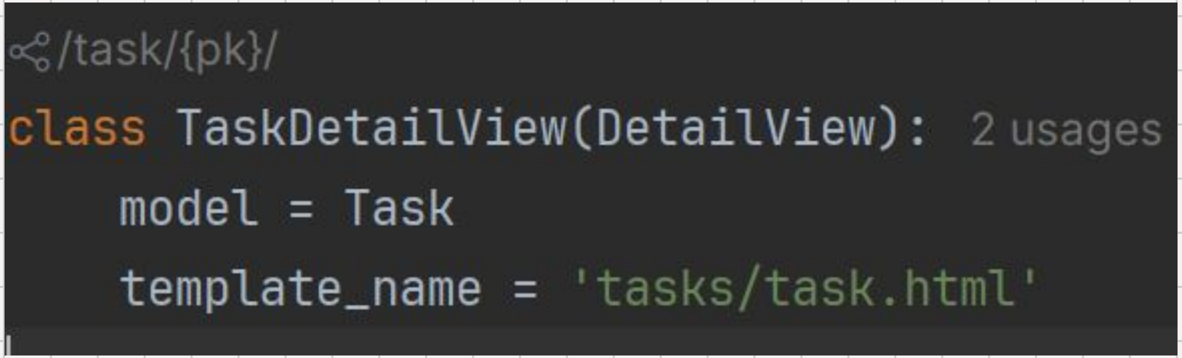
- A `HttpResponse` object is what your view sends back to the browser.
- Your view must return either:
  - `HttpResponse`
  - OR a subclass (e.g. `JsonResponse`, `HttpResponseRedirect`)
  - OR a helper like `render()`

Class-Based Views

- Class-Based Views (CBVs) allow you to write views using Python classes instead of functions.
- They provide **structure**, **reusability**, and built-in features that **reduce boilerplate code**.
- Why Use Class-Based Views?
  - Less code for common patterns (CRUD)
  - Reusable via inheritance
  - More organized than function-based views

Class-Based Views Example


```
/task/{pk}/
path('<int:pk>/', TaskDetailView.as_view(), name='task-detail'),
```


```
/task/{pk}/
class TaskDetailView(DetailView):  2 usages
    model = Task
    template_name = 'tasks/task.html'
```

Important CBVs

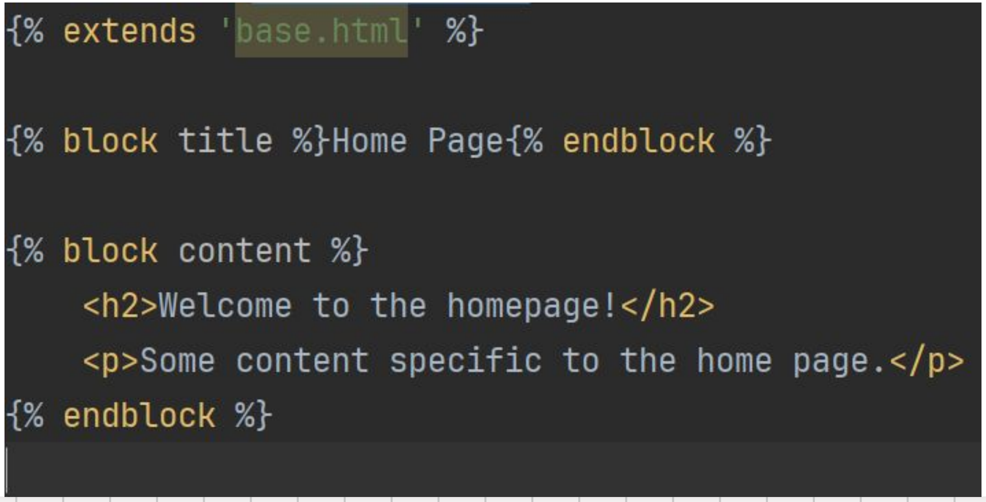
Category	Important Views
Base	`View`
Display	`TemplateView`, `RedirectView`
Object display	`ListView`, `DetailView`
CRUD	`CreateView`, `UpdateView`, `DeleteView`
Forms	`FormView`
Mixins	`SingleObjectMixin`, `MultipleObjectMixin`, `ModelFormMixin`
Auth	`LoginView`, `LogoutView`, Password CBVs
APIs (DRF)	`APIView`, Generic API Views

Common Template Tags

- `{% for %} ... {% endfor %}`
- `{% if %} ... {% endif %}`
- `{% block %}` and `{% extends %}`
- `{% url 'name' %}`
- `{% include 'file.html' %}`

Template Inheritance

- Django template inheritance is a way to structure your HTML templates so that you can reuse common elements (like headers, footers, navigation, etc.) across multiple pages
- Blocks (`content`, `title`, etc.)
- DRY template reuse



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

{% block title %}Home Page{% endblock %}

{% block content %}
    <h2>Welcome to the homepage!</h2>
    <p>Some content specific to the home page.</p>
{% endblock %}
```

Static Files

- In Django, static files refer to files like CSS, JavaScript, images, and other assets that don't change dynamically but are used to style and enhance the front-end of your application. These files are typically placed in a dedicated directory called static and can be referenced in your templates.
- Static files are usually placed inside a static/ directory in your app or project.
- `{% load static %}`