Computer science → Backend → Django → Templates

Rendering templates

10% completed

... Check to skip Go to practice

9 minutes read

When you construct a site, you design it to be optimized. For example, you don't want to rewrite the HTML code when adding new data to the site. Django provides rendering templates to solve this problem.

The site receives a request, does the heavy lifting, and passes the data further to prepare HTML pages. For adjusting the HTML templates, the processing requests may stay the same. In this topic, you'll learn how to bind these two parts together.

Rendering

Jack Torrance started working on his new book called *Shining*. He wants to publish this book online, with the table of contents on the main page and each chapter on a separate page. He sends the first draft of his story:

```
book = {
    'Chapter 1': 'All work and no play makes Jack a dull boy..
    'Chapter 2': 'All work and no play makes Jack a dull boy..
    'Chapter 3': 'All work and no play makes Jack a dull boy..
    'Chapter 4': 'All work and no play makes Jack a dull boy..
}
```

So, you start making the site. We created a project named `booksite` with the `book` app inside. Then, we created the *templates* and *book* directory. Now, we have this tree of our project:

```
booksite
├── book
│   ├── templates
│   │   └── book
│   ├── urls.py
│   └── views.py
└── booksite
    ├── urls.py
    └── settings.py
```

This novel seems a bit strange, but anyway, let's create an HTML template for the main page with the contents of the book and add it to the *book/templates/book/contents.html* file:

```
<h2> Shining </h2>
<ul>
  {% for chapter in book %}
  <li>
    <a href="/chapter/{{ forloop.counter }}">Chapter {{ forloo
  </li>
  {% endfor %}
</ul>
```

Each unordered list item links to a chapter page, so users can comfortably read each chapter on a separate HTML page.

To return a user to the contents page, we need to implement an HTTP handler with the `get` method to the *views.py* file:

```python
from django.views import View
from django.shortcuts import render

# book variable mentioned above can be declared here

class MainPageView(View):
    def get(self, request, *args, **kwargs):
        return render(request, 'book/contents.html', context={
```

We call the `render` function and pass the template path and the `context`

18

Explore

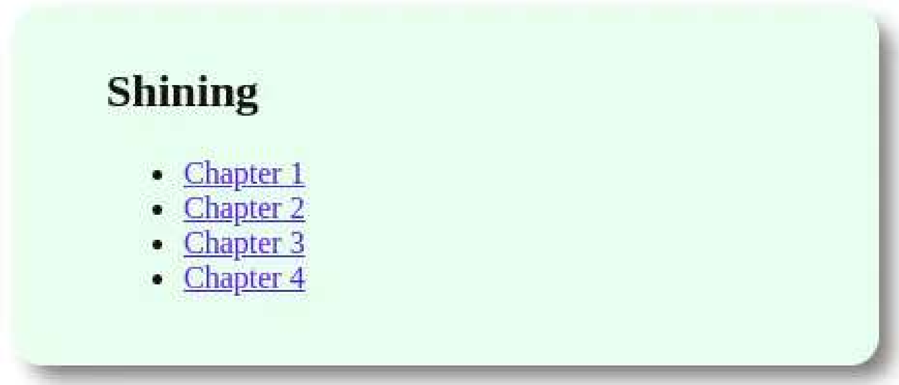
Study plan

Map

Problem of the day

Repeat what you've learned

Find topic


Shining

- Chapter 1
- Chapter 2
- Chapter 3
- Chapter 4

TemplateView class

In the example above, we've defined the HTTP handler with the `get` method. This method is idle, as it delegates all the work to the `render` function. Is there a more straightforward way to write this in the code?

You can use one of the **built-in Django classes** — `TemplateView`. `TemplateView` renders a template with the context containing parameters captured in the URL. This class can be used as an HTTP handler or inherited to create a new one.

In the following example, we will explore using the `TemplateView` by creating a single chapter page. First, a new template should be created in the *book/templates/book/chapter.html*:

```
<h2> Chapter {{ n_chapter }} </h2>
<ul>
  {{ content }}
</ul>
```

In the next step, we are going to make an HTTP handler by creating the new class `ChapterView` in the *views.py* file that inherits from `TemplateView`. To specify the HTML template path, define the class `template_name` property.

The pattern for defining the template path in `template_name` is the same as the one we used in the previous section.

```
from django.views.generic.base import TemplateView


```

```python
class ChapterView(TemplateView):
    template_name = 'book/chapter.html'
```

✓✓

The HTTP handler defined in the example above will automatically render the specified HTML template by providing context containing the chapter number defined in the URL. The context works as a dictionary, as in the previous section. However, the `content` parameter required for rendering the template is not present in the URL path.

To extend the context generated by `TemplateView` in the background, define the `get_context_data` method inside the `ChapterView` class:

```python
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        n_chapter = str(context['n_chapter'])
        context['content'] = books['Chapter ' + n_chapter]
        return context
```

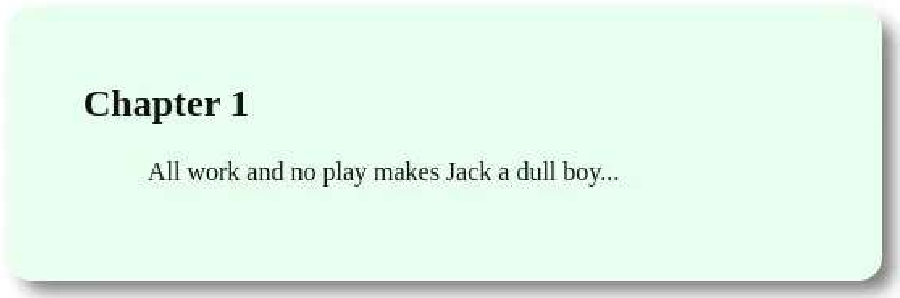
The `super().get_context_data()` method retrieves the context created by `TemplateView`. In the next step, the context is extended with the `content` parameter, and this context will be used for rendering the template.

Defining a correct URL router pattern is the last thing to make the HTTP handler work. `ChapterView` expects to receive the variable `n_chapter` as the `**kwargs` argument of the `get_context_data` method, so the URL path should contain a path variable named `n_chapter` of the *int* type. URL should be placed in the `urlpatterns` inside the *urls.py* file:

```python
path('chapter/<int:n_chapter>', book.views.ChapterView.as_view
```

In the code piece above, the `n_chapter` variable will be found by a regular expression. This means that Django looks for a number, and after that, we can get it from the `**kwargs`.

The rendered single-chapter page should look like this:



Chapter 1

All work and no play makes Jack a dull boy...

You can see the whole code below:

```
from django.views.generic.base import TemplateView

class ChapterView(TemplateView):  # create an HTTP handler
    template_name = 'book/chapter.html'  # specify a HTML temp

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        # call get_context_data of the parent to add data to t
        n_chapter = str(context['n_chapter'])
        context['content'] = books['Chapter ' + n_chapter]
        return context  # return the context in which the new
```

The final tree of our project now looks like this:

```
booksite
├── book
```

```
│   ├── templates
│   │   └── book
│   │       ├── contents.html
│   │       └── chapter.html
│   ├── urls.py
│   └── views.py
└── booksite
    ├── urls.py
    └── settings.py
```

Conclusion

Rendering templates separate the request's processing and the content's final representation. In this topic, we've taken a look at two ways of rendering templates:

- The `render` function, and
- The `TemplateView` built-in Django class.

Both of these ways work. However, the second method suits static pages, so it should be used in the following cases:

- View simple static pages, such as *About the Author* or *About the Company*;
- Display pages with a simple context;
- Show pages with all their content described in the HTML template;

Start practicing

6 learners liked this piece of theory. 6 didn't like it. **What about you?**

Related topics

✓ Django template language
✓ Method overriding
✓ Kwargs
✓ Intro to views in Django

Comments (3) Useful links (0) Show discussion