Theory: Rendering templates

34 minutes 0 / 5 problems solved Skip this topic Start practicing

846 users solved this topic. Latest completion was 15 minutes ago.

Making a site, we want to isolate the processing of requests from the final representation to the client, so that we can update each part independently. We receive a request, do the hard work, and pass the data further to prepare HTML pages. And if we want to rework our HTML templates, the processing requests may stay the same. One thing is still missing: how to bind these two parts together?

§1. Render

A not-so-famous writer Jack Torrance started working on his new book. 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 us the first draft of his story, so we start making a site for this book:

```
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...',
}
```

This novel seems a little bit strange, but what do we know about modern literature, so we create an HTML template for the main page with the contents of the book:

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

Each item of an unordered list is a link to a chapter page, so the users can comfortably read each chapter on a separate HTML page. We name our application "*book*" and save this template to the "*book/templates/book/contents.html*" file.

To return the user to the contents page, we need to implement an HTTP handler with the `get` method:

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


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

All that we do is call the `render` function and pass to it the template path and a `context` dictionary with the book. All the variables from the `context` will be available in the template now, and we can access them by the key from a dictionary.

By default, Django is looking for templates in all of your "*<application_name>/templates*" directories, so we do not include "*book/templates*" in the template path, because Django will do it for us.

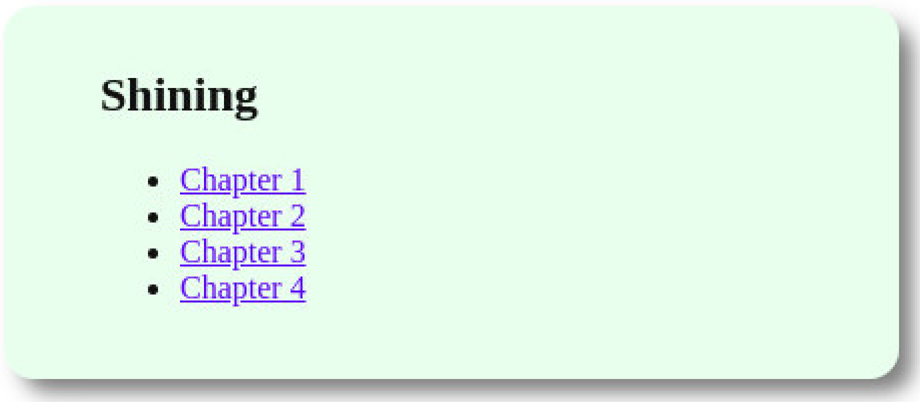
Current topic:

Rendering templates

Topic depends on:

✓ Kwargs
✗ Method overriding
✗ Django template language (Stage 2)
✗ Processing requests (Stage 1)

Table of contents:



Shining

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

To serve this handle on the route "*/contents*", add `path('contents', book.views.MainPageView.as_view())` to the `urlpatterns` list in the "*<project_name>/urls.py*" module.

§2. TemplateView

In the example above, we define the `get` method. This method seems idle because it delegates all the work to the `render` function. How can we omit this duplication of responsibilities?

This time we have a new template in the "*book/templates/book/chapter.html*" file to render a single chapter :

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

Django contains many other classes with predefined code for different use cases. We define a new handler to serve the chapter content and inherit it from `django.views.generic.base.TemplateView`, because we have a template and some data to pass to it.

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


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

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        n_chapter = kwargs['n_chapter']

        context['n_chapter'] = n_chapter

        context['content'] = book['Chapter ' + n_chapter]

        return context
```

Most often, we create one template per page, so we save it as the `template_name` attribute of our class. We don't need to define the `get` method: we fulfill the context and return it from the `get_context_data` call. It looks like we are working directly with a template and don't think about the routines of processing the request.

§3. Routing pattern

The last thing to do to make our handler work is define a correct pattern for the router. We expect to receive the variable `n_chapter` in the `**kwargs` argument of the method, so we define our URL path with the help of a regular expression:

```
re_path('chapter/(?P<n_chapter>\d+)', book.views.ChapterView.as_view())
```

We again add it to the `urlpatterns`. The part `n_chapter` should match the variable name we expect to receive in the `**kwargs` argument of the `get_context_data` method. Get the result page:

Chapter 1

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

Here, our work is finished for now. We'll eagerly expect more chapters from the writer. Meanwhile, all theory and no practice makes Jack a dull boy: let's play with some practical tasks.

Report a typo

43 users liked this theory. **50** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (19) Hints (0) Useful links (0) Show discussion