Theory: Processing requests

31 minutes 0 / 5 problems solved

Skip this topic Start practicing

1453 users solved this topic. Latest completion was 32 minutes ago.

Any web application exists to provide information to clients and receive it from them. A client sends a request to the server and it answers with some response. Usually, the communication between these two sides is processed by the HTTP protocol using `GET`, `POST` and some other types of requests.

In this topic, we shall focus on processing `GET` requests in Django as they are most frequently used.

§1. How to process GET requests

My friend Willy Wonka heard about Django, and now he wants to use it to advertise the sorts of candy his factory makes. To help him reach the goal, we will create a simple response with candies in it.

We assume that the name of the Django project is *"factory"* and the name of the application is *"candies"*.

Let's look at the assortment:

```
candies = {
    "Fudge":  {
        "color": "beige",
        "price": "priceless",
        "available": 100,
    },
    "Chocolate shock": {
        "color": "brown",
        "price": "precious",
        "available": 50,
    },
    "Marshmallow" : {
        "color": "pink",
        "price": "all the money in the world",
        "available": 200,
    },
}
```

On the main page, Willy wants to put only the assortment list and nothing else. So when you go to his site, you send a **GET** request to his service. GET is a method used to receive data from the server.

Django has classes that take over routine work with HTTP requests, so the only part you should implement by yourself is the response. The response is an instance of the inheritors of Django `HttpResponseBase` class, like `HttpResponse`. We add this piece of code to the *candies/views.py* module:

```
from django.http import HttpResponse
from django.views import View

class MainPageView(View):
    def get(self, request, *args, **kwargs):
        html =  "\n".join(f"<div>{candy}</div>" for candy in candies)
        return HttpResponse(html)
```

First, we make a new class and inherit it from the `View`. To provide a method to handle the GET request, we simply define a method with the name *"get"*. This is a general rule in Django, so if you want to make a POST handler, you define a method with the name *"post"*, et cetera.

Current topic:

Stage 1 Processing requests

Topic depends on:

- ✗ Stage 1 HTTP messages
- ✗ Stage 1 Regexps basics
- ✗ Stage 1 Operations with dictionary
- ✓ Stage 1 Split and join 8★
- ✓ Stage 1 Exceptions 3★
- ✓ Stage 1 Methods 3★
- ✗ Stage 1 Launching web server

Topic is required for:

- Stage 4 Submitting data
- Rendering templates

Table of contents:

So, back to our sweet matters: passing HTML as a string to the `HttpResponse` class, we return a simple HTML page with a list of candies. Django application does the rest of the work by itself to send the data to the client. It's that simple to make a response.

§2. Not Found Pages

For each customer who wants to know more about a particular candy, we make another page and class to process that request. And if the customer asks us about a nonexistent candy, we've got to report that we couldn't find it. The corresponding HTTP status code is 404, but we don't see any codes in the previous example. So how does it even work?

As we said, Django does a lot of work under the hood. The HttpResponse set status code 200 in the answer for you, which means that the communication was OK. We cannot change this code in the HttpResponse class, but we can use the Exception class `Http404`, which will signal to a user that they're trying to GET a page that doesn't exist.

Let's make a handler for a custom candy page in the same module:

```python
from django.http import HttpResponse, Http404
from django.views import View

class CandyView(View):
    def get(self, request, candy_name, *args, **kwargs):
        if candy_name not in candies:
            raise Http404

        candy_info = "".join(

            f"<tr><td>{key}:</td><td>{value}</td></tr>"

            for key, value in candies[candy_name].items()

        )

        return HttpResponse(f"<table><tbody>{candy_info}</tbody></table>")
```

As you see, the third parameter in our GET method is cryptic "*candy_name*". We will learn how to pass this positional parameter at the next step; for now, just assume that we get the name of the candy from a user's request. If we have this sort of candy, we display all the information about it. But if we don't have one, we `raise Http404` Exception because we can't find it in our stock.

Pay attention that we return an instance of HttpResponse and raise Http404 since Http404 is an exception class

§3. URL Routing

We create handlers for requests, but how does Django choose the appropriate one? For this purpose, we define URL routes in *factory/urls.py* module:

```python
from django.urls import path, re_path
from candies.views import MainPageView, CandyView

urlpatterns = [
    path("candies/", MainPageView.as_view()),
    re_path("candies/(?P<candy_name>[^/]*)/?", CandyView.as_view()),
]
```

For example, if Willy's site has the hostname *www.willywonka.com*, then the assortment page will be available at the address *www.willywonka.com/candies,* and information specifically about fudge will be found at *www.willywonka.com/candies/Fudge*.

In the second path link, we see a regular expression `(?P<candy_name>[^/]*)`. This expression extracts the variable `candy_name` from the link and passes it to the handler. That's how we get our `candy_name` in the previous example.

To bind a link with an appropriate handler, we can call `path` or `re_path` functions and add the result to the `urlpatterns` list. The first argument of each function receives a string that describes a pattern for a link that comes after the hostname. It can be a simple string in `path` function and a regular expression in case of `re_path`. The second argument is a handler that will process a request.

The order of the links in `urlpatterns` is kept when Django searches for the required handler. If you paste regular expression `"candies/(?P<candy_name>[^/]*)/?"` before `"candies/"`, you will never reach the second one since `"candies/"` is a subset of the first regular expression. So you should paste the superset expression latter on the list.

The path "" (empty string) is a superset for all routes. If you paste it first in the URL patterns list with call to `re_path`, all the other handlers will become unreachable.

§4. Conclusion

To process a request in the Django application, you should define a handler and implement all methods that it can process. If you inherit this class from the `View`, you should match the desired HTTP verb with the same method in your class. Specifically, if you want to process the GET request, you should define the "*get*" method and return an instance of the Django `HttpResponse` class.

After you do that, you can bind this handler to the link it belongs to. Just add it to the list of `urlpatterns` at the *urls.py* module.

Report a typo

88 users liked this theory. 32 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (11) Hints (0) Useful links (1) Show discussion