

Programação de Aplicação Web

Prof. Diego Cirilo

Aula 06: Function-based Views

Function-based Views

- Cada função representa uma *view*;
- As funções são chamadas a partir das regras de `urls.py` ;
- Cada *view* recebe um argumento `request` , que traz os dados da requisição;
- Dentro da função fazemos o processamento/tratamento/aquisição/armazenamento/etc dos dados;
- No fim a função *renderiza* e retorna uma página web, através de *templates*.

Views

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html")
```

Argumentos

- Como é uma função, a *view* pode receber argumentos;
- Os argumentos extras vêm depois do `request` ;
- Ex.: `def minha_view(request, usuario)` ;
- Como passar esses argumentos?

Argumentos na URL

- Usamos `<tipo:variavel>` para definir argumentos na URL;
 - `tipo` é o tipo de dado (`str` , `int` , `slug` , `uuid` , `path`);
 - `variavel` é exatamente o argumento que será passado para a função;

Exemplo

```
# url.py
...
path("fotos/<int:id_foto>/", views.fotos, name="fotos"),
...

# views.py
...
def fotos(request, id_foto):
    print(f"O id da foto é {id_foto}")
...
```

URLs no template

- Para gerar as URLs com os argumentos nos templates fazemos:

```
<a href="{% url 'fotos' id_foto %}">Fotos</a>
```

- Se houver mais de um argumento:

```
{% url 'fotos' id_foto id_usuario %}
```

- É possível também explicitar de qual app é a URL, para evitar conflitos:

```
{% url 'meuapp:fotos' id_foto id_usuario %}
```

O request

- O objeto `request` possui vários atributos úteis;
 - `request.method` : o método HTTP (`GET` , `POST` , etc.);
 - `request.GET` : um *QueryDict* com todos os valores da *querystring*;
 - `request.POST` : um *QueryDict* com os parâmetros do POST;
 - `request.FILES` : idem para arquivos enviados em um formulário;
 - `request.user` : o usuário que fez a requisição;
- [Referência](#).

Exemplo

- views.py

```
...  
def minha_view(request):  
    if request.method == "GET":  
        print("Essa página foi acessada com uma requisição GET")  
...
```

QueryString

- *Querystrings* são expressões adicionadas a URL;
- Não interferem nas rotas;
- Podem trazer informações para filtragem de dados, paginação, etc;
- Seguem o padrão:
 - `url-original?`
`chave=valor&outrachave=outrovalor`
- São utilizadas para requisições que não alteram o estado do sistema;
- Permitem compartilhar uma visualização específica do sistema.

Querystring

- Podem ser acessadas na view com `request.GET` ;
- O valor retornado é um objeto do tipo *QueryDict*;
- Funciona como um dict, com algumas diferenças;
- Os valores podem ser acessados com:
 - `query_dict["chave"]` ;
 - `query_dict.get("chave")` ;
- A vantagem do `.get()` é que não dá erro se a chave não existir.

Exemplo

- URL:

```
http://127.0.0.1:8000/usuarios?ordenar=idade&cidade=SPP
```

- No views.py:

```
...  
def usuarios(request):  
    query_dict = request.GET  
    print(f"0 usuarios serão ordenados por {query_dict['ordenar']}")  
    print(f"0 usuarios são da cidade {query_dict.get('cidade')}")  
    ...
```

Redirecionamento

- Além de usar o `return render(request, 'template.html')` é possível redirecionar para outra URL;
- Usamos o `redirect(nome-da-url)`
- Ex.:

```
def minha_view(request):  
    if not request.GET.get("status"):  
        print("Não veio o status na querystring!")  
        return redirect('nomedoapp:nomedaview') # o nomedoapp é opcional!
```

Dúvidas?

