Programação de Sistemas para Internet

Prof. Diego Cirilo

Aula 05: Apps Django

Apps no Django

- Os códigos do sistema em si, fazem parte de *aplicações* Django;
- Os projetos podem conter apenas uma ou mais aplicações;
- Dividir o seu projeto em aplicações pode permitir o reuso no futuro;

Criando um *app* no Django

- Utilizamos o *script* `manage.py`

```
python manage.py startapp meuapp
```

Um *app* no Django

- O *app* é criado com a seguinte estrutura:

```
meuapp
├── admin.py
├── apps.py
├── __init__.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
└── views.py
```

Estrutura de um app

- `admin.py` - Configuração da interface padrão de administração do Django
- `apps.py` - Configurações do *app*
- `migrations` - Pasta onde aparecerão as *migrations*, que são as definições/alterações na estrutura do BD.
- `models.py` - Onde serão descritos os `models`, que definem a estrutura de dados do sistema.
- `tests.py` - Testes de software.
- `views.py` - Descrição das *views*.

Ativando um *app*

- Não basta criar o *app*
- O arquivo `settings.py` contém a lista dos *apps* ativos
- Para ativar, basta adicionar o nome do *app* na lista.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'meuapp',
]
```

URLs

- *Uniform Resource Locator*
- Também conhecido como *rotas*.
- Os recursos que o usuário procura na URL do navegador são *filtrados* no `urls.py`.
- Os filtros direcionam cada URL solicitada para uma *view* que será responsável por processar a requisição.
- É possível organizar as URLs por *apps*.

`urls.py`

- Usamos o `include` para redirecionar as requisições para seu *app* responsável:

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

urlpatterns = [
    path("", include("tarefas.urls")),
    path("admin/", admin.site.urls),
]
```

Criando `urls.py` no *app*

- Dentro na pasta do *app* deve ser criado um arquivo `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

Views

- Provavelmente o arquivo mais "movimentado" no projeto.
- Recebem e processam as requisições, retornando um conteúdo baseado nos *templates*.
- Podem ser escritas como funções (*Funcion-based View* - FBVs) ou como classes (*Class-based View* - CBVs).
- Iniciaremos com FBVs.
- As requisições são direcionadas para cada *view* a partir da regra da rota (`urls.py`).

FBV

- As funções são chamadas a partir das regras de `urls.py`.
- Cada função 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 retorna uma página web, através de *templates*.

Views

```
from django.shortcuts import render


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

Templates

- As *views* podem retornar páginas estáticas, mas qual seria a vantagem?
- Templates permitem a substituição dinâmica de dados em uma página HTML
- Isso é possível através de *tags*.
- A estrutura é de um HTML normal, com *tags* extras.

Templates

- O Django por padrão procura os templates na pasta `templates` em cada *app*
- É possível configurar outra pasta no arquivo `settings.py`

Tarefa

- Configure o ambiente (venv, git)
- Inicialize um projeto (`config`) e um *app* chamado `exemplo`
- Configure as *urls* do projeto (em `config`) e as *urls* do *app*.
- As *urls* do *app* devem apontar para uma *view*: `index` .
- Crie a *views* `index` e o *template* `index.html` . O conteúdo do *template* pode ser uma página simples.
- Verifique o funcionamento do sistema.
- Faça os commits quando julgar necessário, e envie para o Github.

Dúvidas?