Popebu CMS Documentation

Release 1.0.0

Paula Grangeiro

out 03, 2017

Sumário

Popebu é um CMS opensource baseado no Django.

Sinta-se a vontade para fazer um fork ou relatar um incidente no repositório do projeto.

CAPÍTULO 1

Requisitos

- Python 2.7+
- Pip
- Mysql
- mysql-connector-python
- Virtualenv (opcional)

CAPÍTULO 2

Instalação

1. Depois de baixar o código do projeto e descompactá-lo em uma pasta, é necessário instalar as bibliotecas de dependência do projeto. Caso tenha optado por utilizar o virtualenv, crie e ative o seu ambiente antes da instalação das bibliotecas.

```
$ pip install -r popebu/conf/requirements/requirements.txt
```

2. Depois de instalar as bibliotecas, inicie o servidor da aplicação. Caso esteja desenvolvendo um projeto local, o servidor de desenvolvimento do Django pode ser utilizado.

```
$ python manage.py runserver
```

3. Em um primeiro acesso, é exibida a página de setup do Popebu onde você deve informar alguns dados básicos para a configuração inicial do projeto.
4. Depois de concluir a configuração do projeto com sucesso, reinicie o servidor da aplicação.

CAPÍTULO 3

Documentação

Iniciando

A área administrativa do Popebu está disponível em <url>/admin

Logando

Para um primeiro acesso, utilize o usuário criado na configuração do projeto.

Criando usuários

Para criar novos usuários acesse o menu Auth > Usuários.

As permissões do usuário estão divididas em três grupos:

- Administrador: Possui acesso à todas as áres do sistema.
- Redator: Possui acesso somente as áreas relativas à postagens.
- Segurança: Possui acesso somente as áreas de autorização do sistema.

Ao concluir com sucesso o cadastro do usuário, uma senha temporária será enviada para o email informado no cadastro.

Postando conteúdo

Para criar novas postagens acesse Administração > Postagens.

Widgets

Os widgets são as partes que compõem o template.

- **Templates: Blocos e Includes** O sistema de template do Django permite o máximo reaproveitamento de código através de blocos e includes.
  - **Blocos:** Observe esses dois arquivos:

    pai.html

    ```
    <html>
        <head>
            {% block head %}
            <title>Ola, Mundo!</title>
            {% endblock %}
        </head>
        <body>
            {% block content %}{% endblock %}
        </body>
    </html>
    ```

    filho.html

    ```
    {% extends "pai.html" %}

    {% block content %}
        <p>Olá, mundo!</p>
    {% endblock %}
    ```

    O template denominado pai possui todo o html básico do template, enquanto o filho possui somente parte de um html dentro de uma tag **{% block content %}{% endblock %}**. O que ocorre aqui é que quando o template filho.html for renderizado para browser ele vai possui todo o html do template pai.html, exceto a parte do bloco content, que na verdade será a parte do html descrita no template filho. Isso acontece porque o template filho extende (observe a primeira linha do arquivo filho.html) do template pai, ou seja, herda do template pai.
  - **Include** Os includes servem para renderizar html diretamente para algum template. Blocos podem conter includes, mas includes não podem conter blocos. Includes funcionando adicionando a tag **{% include "arquivo.html" %}** diretamente no template.

  Para mais informações, consulte a documentação do Django sobre templates.
- **Views e templatetags** As views e as templatetags são as responsáveis por gerar os dados que serão renderizados para os templates.
  - **Views** Views são utilizadas para realizar consultas e retornar o resultado para um template específico. Observe os exemplos abaixo:

    views.py

    ```
    from django.shortcuts import render_to_response
    from administracao.models import Postagem
    from django.template import RequestContext

    def index(request):
        result = Postagem.objects.filter(rascunho=False)
        return render_to_response(
            'blog/index.html',
            {'result': result},
            context_instance=RequestContext(request))
    ```

    index.html

```
{% extends "blog/base.html" %}

{% block content %}
    <div>
        {% for item in result %}
        <div>
            <h2><a href="{{ item.get_absolute_url }}">{{ item.titulo }}
↪</a></h2>
            <div>{{ item.conteudo }}</div>
        </div>
        {% endfor %}
    </div>
{% endblock %}
```

O método index presente no arquivo views.py faz a consulta de todas as postagens cadastradas e retorna os dados para o arquivo index.html

– **Templatetags** São utilizadas para renderizar consultas para parte de um template. Observe abaixo:

sidebar_tags.py

```
from django import template
from auth.models import UserProfile

register = template.Library()

@register.inclusion_tag('usuarios.html')
def get_usuarios(context):
    result = UserProfile.objects.filter(is_active=True)

    return {'result': result}
```

usuarios.html

```
<div class="users">
{% for item in result %}
    <h4>{{ item.first_name }}</h4>
    <p>{{ item.biografia }}</p>
{% endfor %}
</div>
```

O método get_usuarios presente no arquivo sidebar_tags.py consulta os perfis de usuários cadastrados e retorna os dados para o arquivo usuarios.html. Observe que este arquivo não possui nenhum bloco, nem extende de nenhum template. Para que este aquivo seja renderizado, é necessário que em algum template (pai ou filho) carregue o arquivo sidebar_tags e faça a chamada do método get_usuarios, confome abaixo:

pai.html

```
<html>
    <head>
        {% block head %}
        <title>Ola, Mundo!</title>
        {% endblock %}
    </head>
    <body>
        {% block content %}{% endblock %}
        {% load sidebar_tags %}
        <div>
```

```
            {% get_usuarios %}
        </div>
    </body>
</html>
```

Sempre que criar um novo arquivo de templatetags, lembre-se de reiniciar o servidor da aplicação. Para maiores informações, leia a documentação do Django sobre views e templatetags.