

Publicado em 11 de Setembro de 2020 às 01:00

Última atualização em 3 de Novembro de 2020 às 12:39

Tutorial Django - Criando um modelo de usuário personalizado (Custom User Model)

🐍 python, django



Acompanhe esse conteúdo pelo YouTube

E aí, beleza?

Hoje você vai aprender a criar um modelo de usuário personalizado para os seus projetos Django.

Com isso, em vez de usar o modelo de usuário padrão do Django (*django.contrib.auth.models.User*), você vai usar o seu próprio modelo, que estende a classe *django.contrib.auth.models.AbstractUser*.

Se quiser ir direto para o código no GitHub, clique aqui.

Setup inicial

Ao iniciar um projeto é sempre uma boa ideia criar um ambiente virtual. No Windows eu inicio meus projetos dessa forma:

```
# Cria um ambiente virtual
python -m venv venv
# Ativa o ambiente virtual
.\venv\Scripts\activate
# Instala o Django no ambiente virtual
pip install django
# Inicia o projeto (repare o ponto no fim do comando)
# Esse ponto é usado para criar o projeto na pasta atual
django-admin startproject tutorialcustomuser .
```

Recomendo também já iniciar um repositório Git e criar o arquivo .gitignore.

Para garantir que está tudo funcionando como esperado, execute o comando para iniciar o servidor local:

```
python manage.py runserver
```

E acesse o endereço <http://127.0.0.1:8000/>. Você deverá ver uma tela indicando que a instalação foi um sucesso.

Não vamos rodar o comando migrate agora.

Modelo de usuário personalizado

A documentação do Django recomenda criar um modelo de usuário personalizado ao iniciar um novo projeto, e é isso que vamos fazer agora.

Vamos criar um app chamado users:

```
python manage.py startapp users
```

E agora vamos editar o arquivo users/models.py:

```
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

Com esse código temos um modelo de usuário igual ao modelo do Django.

Podemos acrescentar campos à vontade, por exemplo, para adicionar um campo com uma descrição do usuário basta digitar:

```
from django.contrib.auth.models import AbstractUser
from django.db import models

class User(AbstractUser):
    bio = models.TextField(blank=True)
```

No arquivo settings.py precisamos fazer duas modificações. A primeira é adicionar o app users na lista de apps:

```
INSTALLED_APPS = [
    # django apps
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # local apps
    "users.apps.UsersConfig",
]
```

E a segunda é acrescentar a variável AUTH_USER_MODEL, que indica qual modelo de usuário o projeto vai usar:

```
# User Model

AUTH_USER_MODEL = "users.User"
```

E no terminal:

```
python manage.py makemigrations users
python manage.py migrate
```

Agora vamos criar um superuser, digite esse comando e preencha as informações solicitadas:

```
python manage.py createsuperuser
```

Só que agora temos um pequeno “problema”: Ao acessar a interface de admin do Django (<http://127.0.0.1:8000/admin/>), não encontramos a página de usuários.

Precisamos de mais um pouco de código.

No arquivo users/admin.py, digite:

```
from django.contrib import admin

from .models import User

admin.site.register(User)
```

Agora atualize a interface de admin e veja que a página de usuários está presente

Agora atualize a interface de admin e veja que a página de usuários está presente.

Mas temos outro problema. Se você navegar por essa página, entrar nos detalhes de um usuário ou tentar criar um usuário novo, você verá que a página não está se comportando como esperado. Por exemplo, a senha não deveria ser editável, e a lista de usuários deveria exibir mais campos e opções de filtro.

Para corrigir isso rapidamente, edite o arquivo `users/admin.py`:

```
from django.contrib import admin
from django.contrib.auth import admin as auth_admin

from .models import User

admin.site.register(User, auth_admin.UserAdmin)
```

Atualize a interface de admin e você verá tudo funcionando corretamente.

Isso que fizemos agora funciona muito bem, e até poderíamos encerrar o artigo, mas tem mais uma coisa que podemos fazer: estender a classe `UserAdmin`.

Essa classe é responsável por deixar a interface de admin da forma como vemos agora. Nela são definidos os campos que serão exibidos na tela, os forms que serão usados, e vários outros atributos e métodos. Recomendo dar uma olhada no código que define essa classe.

O mínimo que precisamos fazer para estender a `UserAdmin` é isso aqui:

Crie o arquivo `users/forms.py` e digite:

```
from django.contrib.auth import forms

from .models import User

class UserChangeForm(forms.UserChangeForm):
    class Meta(forms.UserChangeForm.Meta):
        model = User

class UserCreationForm(forms.UserCreationForm):
    class Meta(forms.UserCreationForm.Meta):
        model = User
```

E no arquivo `users/admin.py`:

```
from django.contrib import admin
from django.contrib.auth import admin as auth_admin

from .forms import UserChangeForm, UserCreationForm
from .models import User

@admin.register(User)
class UserAdmin(auth_admin.UserAdmin):
    form = UserChangeForm
    add_form = UserCreationForm
    model = User
```

E é isso.

Repare que deixamos as classes prontas para serem estendidas, mas não fizemos nada diferente do padrão.

O modelo de usuário está igual ao modelo padrão (a única diferença é o campo “bio”), os forms também são os mesmos e a interface de admin é exatamente igual a interface de um projeto que usa o modelo padrão.

Mas agora, com o código desse jeito, estender essas classes é muito fácil. O código está “no esquema”. Agora é só mexer nele de acordo com as suas necessidades.

Um exemplo simples: podemos editar o arquivo `users/admin.py` dessa forma:

```
from django.contrib import admin
from django.contrib.auth import admin as auth_admin

from .forms import UserChangeForm, UserCreationForm
from .models import User

@admin.register(User)
class UserAdmin(auth_admin.UserAdmin):
    form = UserChangeForm
    add_form = UserCreationForm
    model = User
    fieldsets = auth_admin.UserAdmin.fieldsets + (
        ("Campos Personalizados", {"fields": ("bio",)}),
    )
```

E agora se você atualizar a página de detalhes de um usuário você verá lá embaixo o campo bio.

E de onde eu tirei esse nome fieldsets? Do código que define a classe UserAdmin.

Pronto. Agora o seu projeto tem um modelo de usuário personalizado!

Se você quiser se aprofundar mais nesse tópico, clique aqui e leia a documentação do Django.

Qualquer dúvida, sugestão ou correção é só deixar um comentário.

Um abraço, falou!


Compartilhe

Gostou deste artigo? Compartilhe para esse conteúdo chegar a mais pessoas!



Não perca nenhum conteúdo!

Se você quiser receber um e-mail sempre que eu postar um artigo novo, basta se inscrever aqui:

 Insira seu e-mail

Enviar

Artigos que podem ser do seu interesse

11 de Setembro de 2020

Tutorial Django - Cadastro e login de usuários apenas com e-mail e senha
Aprenda a fazer o cadastro e login de usuários apenas com e-mail e senha.

🔗 python, django

15 de Novembro de 2020

Python + VS Code + Black: Como formatar o seu código Python no VS Code automaticamente usando Black

Um passo a passo ensinando a configurar o VS Code para formatar código Python automaticamente.

🔗 python

20 de Maio de 2020

Funções com valores padrão mutáveis: Qual é o problema?

Através de um exemplo simples entendemos por que não devemos definir funções com valores padrão mutáveis.

🔗 python

20 de Abril de 2020


Resenha - Think Python (Pense em Python)

Meus comentários sobre esse livro.

🔗 python, resenhas

0 Comentários

fabioruicci

 Disqus' Privacy Policy

1


Entrar

 Recomendar

 Tweet


 Compartilhar

Ordenar por Mais votados




Iniciar a discussão...


FAZER LOGIN COM


OU REGISTRE-SE NO DISQUS 

Nome

Seja o primeiro a comentar.

 Inscreva-se

 Adicione o Disqus no seu siteAdicionar DisqusAdicionar

 Do Not Sell My Data

© Fabio Ruicci 2020
contato@fabioruicci.com.br

