Programação de Sistemas para Internet

Prof. Diego Cirilo

Aula 10: Autenticação

Autenticação e Autorização

- É necessário limitar o acesso a operações nos sistemas web;
- Usuários devem apresentar credenciais (*login* e senha) para *provar* sua identidade para o sistema - Autenticação;
- Cada usuário tem um conjunto de operações permitidas - Autorização;
- O Django já possui essas funcionalidades.

Django User Model

- O Django já possui um Model padrão para User;
- Já conta com vários atributos:
  - `username`, `first_name`, `last_name`, `email`, `password`, `groups`, `user_permissions`, `is_staff`, `is_active`, `is_superuser`, `last_login` e `date_joined`;

Custom User

- Nem sempre o User do Django atende as nossas necessidades;
- Há duas possibilidades:
  - Criar nossa própria classe *User*, herdando de *AbstractUser* ou *AbstractBaseUser*;
  - Criar uma nova classe com os dados extras, deixando *User* apenas para autenticação;
- Qual a melhor?

Custom User

- *AbstractUser*: É basicamente o *User* do Django, porém como classe abstrata.
- *AbstractBaseUser*: É classe base, sem a maioria dos atributos da classe *User*. É útil quando não temos interesse nesses atributos padrão.
- A não ser que você tenha um bom motivo, recomendo herdar de *AbstractUser*;
- Referência.

Exemplos

- Classe *User* customizada (`models.py`):

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

class User(AbstractUser): # ou AbstractBaseUser
  cpf = models.CharField(max_length=11, unique=True) # exemplo
```

- Devemos adicionar a configuração (`settings.py`)

```
AUTH_USER_MODEL = "nomedoapp.User"
```

Exemplos

- Classe de *Perfil*, que adiciona campos extras sem alterar *User*;
- Exemplo (`models.py`):

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

class Perfil(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE)
  cpf = models.CharField(max_length=11, unique=True) # exemplo
```

Exemplos

- No caso do *Perfil*, podemos acessar os dados extras com:

```
usuario = User.objects.get(id=2) # pega o user com id 2
cpf_do_user = usuario.perfil.cpf
```

Custom User

- O Django por padrão usa o `username` como chave de login;
- É comum usarmos `email` ou mesmo `cpf` ao invés de `username`;
- Na customização podemos configurar isso;
- É necessário fazer várias alterações;
- Porém podemos reutilizar o código em outros projetos.

Exemplo

```
class Usuario(AbstractUser):
    # Para usar como login, é necessário ser único
    email = models.EmailField(max_length=255, unique=True)

    # Define qual o campo é o nome de usuário
    USERNAME_FIELD = "email"
    # Necessário para createsuperuser continuar funcionando
    REQUIRED_FIELDS = ["username"]
```

Views padrão de autenticação

- O Django possui um sistema completo de autenticação pronto;
- Views de Login/Logout/Alterar Senha/etc.;
- Views de cadastro não incluídas;
- Templates também não incluídos;
- Como tudo no Django, é possível customizar.

URLs

- Para utilizar as views padrão, devemos adicionar ao `config/urls.py` (ou diretamente no app):

```
urlpatterns = [
    ...
    path("accounts/", include("django.contrib.auth.urls")),
    ...
]
```

- Os *names* nas URLs (para usar com a tag `url` nos templates) são: `login`, `logout`, `password_change`, `password_change_done`, `password_reset`, `password_reset_done`, `password_reset_confirm`, `password_reset_complete`.

URLs

- Para customizar as URLs e outros parâmetros das *views* podemos adicionar um a um no `urls.py`:

```
from django.contrib.auth import views as auth_views

urlpatterns = [
    ...
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('password_change/', auth_views.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', auth_views.PasswordChangeDoneView.as_view(), name='password_change_done'),
    path('password_reset/', auth_views.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset/done/', auth_views.PasswordResetDoneView.as_view(), name='password_reset_done'),
    path('reset/<uidb64>/<token>/', auth_views.PasswordResetConfirmView.as_view(), name='password_reset_confirm'),
    path('reset/done/', auth_views.PasswordResetCompleteView.as_view(), name='password_reset_complete'),
    ...
]
```

Configurações Úteis

```
AUTH_USER_MODEL = "usuarios.User"

LOGIN_URL = "login"
LOGOUT_REDIRECT_URL = "index"
LOGIN_REDIRECT_URL = "index"
```

Templates

- Os *templates* devem ser colocados em `templates/registration` por padrão;
- Os nomes são respectivamente: `login.html`, `logged_out.html`, `password_change_form.html`, `password_change_done.html`, `password_reset_form.html`, `password_reset_done.html`, `password_reset_confirm.html`, `password_reset_complete`.
- Também é possível customizar com:

```
path("change-password/",
    auth_views.PasswordChangeView.as_view(template_name="change-password.html"),
),
```

- Dica: é possível ver os templates que Django Admin usa aqui.

Forms

- Apesar de ser necessário criar os *templates* os forms já estão disponíveis;
- Basta usar `{{ form }}`;
- Assim como os forms comuns é possível customizar ou construir o form diretamente;
- Esses forms podem ser utilizados diretamente em outras partes do sistema, como o *PasswordChangeForm*;
- Outro form importante é o *UserCreationForm* que pode ser utilizado na página de registro.

Forms

- Os forms podem ser customizados herdando do form original;
- Por exemplo, para remover o campo `username` quando não for necessário.

Recuperação de Senha

- O Django já gera o email com o link para recuperação de senha;
- Assim como os outros, precisa ter os templates em `registration`;
- Para que o email seja enviado, é necessário ter um servidor de emails e configurar o `EMAIL_BACKEND` no `settings.py`;
- Backend para testes que apenas imprime o email no terminal:
- `EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"`

Limitando acesso a usuários logados

- Quando usamos o *decorator* `@login_required` em uma view, apenas usuários logados terão acesso ao recurso;
- Caso o usuário não esteja logado, ele será redirecionado para `LOGIN_URL`, definida no `settings.py`

Referências

- https://docs.djangoproject.com/en/5.1/topics/auth/

Dúvidas?