Apostila 6: Autenticação, Usuários e Permissões

1. Introdução: O Modelo User

O Django já vem com um sistema de autenticação completo. Ele inclui "baterias inclusas" na forma de:

- Um Model de User (com campos para username, password, email, first_name, last_name).
- Formulários de criação de usuário (UserCreationForm) e de login (AuthenticationForm).
- Views prontas para Login e Logout.
- Um sistema de "Sessão" (o "crachá") que mantém o usuário logado.

Nosso primeiro passo é conectar o nosso Model de Tarefa a este Model de User.

2. Módulo 1: Associando Tarefas a Usuários (O ForeignKey)

Vamos dizer ao banco de dados que "toda Tarefa pertence a um Usuário". Faremos isso com uma Chave Estrangeira (ForeignKey).

Passo 1: Edite core/models.py

Precisamos importar o Model de User e adicioná-lo à nossa Tarefa.

Python

```
# core/models.py

from django.db import models

# 1. Importe o Model de User padrão do Django

from django.contrib.auth.models import User


class Tarefa(models.Model):

    # 2. ADICIONE ESTE NOVO CAMPO

    # 'on_delete=models.CASCADE' diz ao banco:

    # "Se o usuário for deletado, delete todas as tarefas dele também."

    user = models.ForeignKey(User, on_delete=models.CASCADE)


    titulo = models.CharField(max_length=200)

    concluida = models.BooleanField(default=False)
```

```
    criada_em = models.DateTimeField(auto_now_add=True)


    def __str__(self):
        return self.titulo
```

3. Módulo 2: A Migração (Recomeçando o Banco)

ATENÇÃO! LEIA COM CUIDADO!

Acabamos de fazer uma mudança estrutural no banco. Dissemos que o campo user é obrigatório. Mas o nosso banco de dados (db.sqlite3) já tem tarefas criadas... e elas não têm um usuário!

Teríamos um conflito. Para um tutorial, a forma mais fácil e limpa de resolver isso é recomeçando o banco de dados do zero.

Passo 1: Delete o Banco de Dados Antigo

1. Pare o servidor (Ctrl+C).
2. Na pasta raiz do seu projeto, delete o arquivo db.sqlite3.
3. Dentro da pasta core/migrations/, delete TODOS os arquivos, exceto o __init__.py.

Passo 2: Crie as Novas Migrações

Bash

```
# 1. Crie o novo "plano de obra" (ex: 0001_initial.py)
python manage.py makemigrations core
# 2. Execute o plano e crie o novo 'db.sqlite3'
python manage.py migrate
```

Passo 3: Crie um novo Superusuário Como o banco é novo, nossos usuários antigos se foram. Precisamos criar um novo.

Bash

```
python manage.py createsuperuser
```

(Siga os passos e crie seu usuário admin novamente).

4. Módulo 3: As Views de Login e Logout (As "Baterias Inclusas")

O Django nos dá Views prontas (baseadas em classes) para login e logout. Só precisamos criar as URLs e os templates .

Passo 1: Crie o Template de Login

Crie um novo arquivo templates/login.html.

HTML

```
{% extends 'base.html' %}

{% block title %}Login{% endblock %}


{% block content %}

<h2>Login</h2>


{% if form.errors %}

    <p style="color: red;">Seu usuário ou senha estão incorretos. Tente novamente.</p>

{% endif %}


<form method="POST" action="{% url 'login' %}">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Entrar</button>

</form>


<p><a href="{% url 'register' %}">Não tem uma conta? Cadastre-se</a></p>

{% endblock %}
```

Note que já incluímos um link para a página register, que criaremos a seguir.

Passo 2: Crie as URLs de Login e Logout

Vamos adicionar as URLs de autenticação no arquivo de URLs principal do projeto.

Abra meuprojeto/urls.py:

Python

```
# meuprojeto/urls.py
from django.contrib import admin
from django.urls import path, include
# 1. Importe as views de autenticação
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('core.urls')), # Nossas URLs do app 'core'

    # 2. ADICIONE ESTAS URLs

    # URL de Login
    # Ela usa a View pronta 'LoginView' e diz a ela para usar nosso template
    path('login/',
         auth_views.LoginView.as_view(template_name='login.html'),
         name='login'),

    # URL de Logout
    # Ela usa a View 'LogoutView'. 'next_page' diz para onde ir após o logout.

    path('logout/',
         auth_views.LogoutView.as_view(next_page='login'),
```

```
        name='logout'),
]
```

5. Módulo 4: A View de Cadastro (register)

O Django tem um formulário pronto para criar usuários (UserCreationForm), mas, diferentemente do login, ele não tem uma view pronta. Vamos criar uma.

Passo 1: Crie o Template de Cadastro Crie o arquivo templates/register.html:

```
HTML
{% extends 'base.html' %}
{% block title %}Cadastro{% endblock %}

{% block content %}
<h2>Cadastre-se</h2>

<form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Criar Conta</button>
</form>
{% endblock %}
```

Passo 2: Crie a View register (VERSÃO CORRIGIDA). Abra core/views.py e adicione a lógica de cadastro:

```
Python
# core/views.py
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.forms import UserCreationForm # 1. Importe o form
from django.contrib.auth import login # 2. Importe a função 'login'
# (Importe também seus Models e Forms de Tarefa)
from .models import Tarefa
from .forms import TarefaForm
```

```
... (suas views 'home', 'concluir_tarefa', 'deletar_tarefa') ...


# 3. ADICIONE A NOVA VIEW DE CADASTRO (LÓGICA CORRIGIDA)
def register(request):
    # Se a requisição for POST, o usuário enviou o formulário
    if request.method == 'POST':
        # Cria uma instância do formulário com os dados enviados
        form = UserCreationForm(request.POST)
        # Verifica se o formulário é válido (ex: senhas batem, username não existe)
        if form.is_valid():
            user = form.save() # Salva o novo usuário no banco
            login(request, user) # Faz o login automático do usuário
            return redirect('home') # Redireciona para a home
    # Se a requisição for GET, o usuário apenas visitou a página
    else:
        form = UserCreationForm() # Cria um formulário de cadastro vazio

    # Prepara o contexto e renderiza o template
    context = {'form': form}
    return render(request, 'register.html', context)
```

Passo 3: Crie a URL de Cadastro

Abra core/urls.py (o do app):

```
Python
# core/urls.py
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('', views.home, name='home'),
    path('tarefa/<int:pk>/concluir/', views.concluir_tarefa, name='concluir_tarefa'),
    path('tarefa/<int:pk>/deletar/', views.deletar_tarefa, name='deletar_tarefa'),

    # ADICIONE A URL DE CADASTRO
    path('register/', views.register, name='register'),
]
```

6. Módulo 5: Protegendo a "Home" e Filtrando os Dados

Agora vem a parte mais importante. Vamos "trancar" nossa página inicial e modificar a lógica dela.

Passo 1: Protegendo a View home

Queremos que apenas usuários logados vejam a home. O Django tem um "decorador" (decorator) para isso. Abra core/views.py:

Python

```
# core/views.py
...
from django.contrib.auth import login
from django.contrib.auth.decorators import login_required # 1. Importe o decorador

# 2. ADICIONE O DECORADOR ACIMA DA VIEW 'home'
@login_required
def home(request):
    ...
```

É só isso! Agora, se um usuário anônimo tentar acessar a home (/), o Django irá automaticamente redirecioná-lo para a página de login.

Para isso funcionar, adicione esta linha no final do seu meuprojeto/settings.py:
LOGIN_URL = 'login'

Passo 2: Filtrando Tarefas e Salvando com Usuário

Altere a view home em core/views.py:

Python

```
# core/views.py
@login_required
def home(request):
    if request.method == 'POST':
        form = TarefaForm(request.POST)
        if form.is_valid():
            #
            # MUDANÇA 1: Salvando com o usuário
            #
            # 'commit=False' cría o objeto na memória, mas não salva no banco.
            tarefa = form.save(commit=False)
            # Atribui o usuário logado (request.user) ao campo 'user' da tarefa
            tarefa.user = request.user
            # Agora sim, salva o objeto completo no banco
            tarefa.save()
            return redirect('home')
    else:
        form = TarefaForm()

    #
    # MUDANÇA 2: Filtrando a lista de tarefas
```

```
    #
    # Antes: Tarefa.objects.all()
    # Agora: Filtre apenas onde o campo 'user' é igual ao 'request.user'
    todas_as_tarefas = Tarefa.objects.filter(user=request.user).order_by('-criada_em')

    context = {
        'nome_usuario': request.user.username, # Use o nome do usuário logado!
        'tecnologias': ['Autenticação', 'ForeignKey', 'Login'],
        'tarefas': todas_as_tarefas,
        'form': form,
    }
    return render(request, 'home.html', context)
```

Passo 3: (Segurança) Proteja as views de concluir e deletar Precisamos garantir que o usuário só possa deletar ou concluir as suas próprias tarefas. Modifique concluir_tarefa e deletar_tarefa em core/views.py:

Python

```
# core/views.py

# 1. Adicione o decorador @login_required em ambas
@login_required
def concluir_tarefa(request, pk):
    # 2. Modifique o 'get_object_or_404'
    # Busque a Tarefa pela 'pk' E ONDE o 'user' é o 'request.user'
    tarefa = get_object_or_404(Tarefa, pk=pk, user=request.user)
    if request.method == 'POST':
        tarefa.concluida = True
        tarefa.save() # Não se esqueça de salvar!
```

```
    return redirect('home')


@login_required
def deletar_tarefa(request, pk):
    # 3. Faça o mesmo filtro de segurança aqui
    tarefa = get_object_or_404(Tarefa, pk=pk, user=request.user)
    if request.method == 'POST':
        tarefa.delete()
    return redirect('home')
```

7. Módulo 6: Links Condicionais no base.html

Por fim, vamos mostrar "Login/Cadastro" para usuários deslogados e "Olá, Usuário / Logout" para usuários logados. Abra templates/base.html e edite a área do <header> (ou <nav>):

HTML

```
{% load static %}
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>{% block title %} Meu Site Django {% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
  </head>
  <body>
    <header>
      <nav>
        <a href="{% url 'home' %}">Home</a> |

        {% if user.is_authenticated %}
```

```
        <span>Olá, {{ user.username }}!</span> |

        <form action="{% url 'logout' %}" method="POST" style="display: inline;">
            {% csrf_token %}
            <button type="submit" style="background:none; border:none;
padding:0; color:blue; text-decoration:underline; cursor:pointer;">
                Sair (Logout)
            </button>
        </form>

    {% else %}
        <a href="{% url 'login' %}">Entrar (Login)</a> |
        <a href="{% url 'register' %}">Cadastre-se</a>
    {% endif %}
    </nav>
  </header>

  <main>
    {% block content %}
    {% endblock %}
  </main>

  <footer>
    <p>© 2025 - Meu Site Incrível</p>
  </footer>
 </body>
</html>
```

8. Revisão da Aula

Parabéns! Você transformou um app global em um app pessoal e seguro! O que fizemos:

1. ForeignKey: Ligamos o Model de Tarefa ao Model de User.

2. Banco de Dados: Recriamos o banco (db.sqlite3) para aceitar a nova estrutura. 3. Views (Auth): Usamos as LoginView e LogoutView prontas do Django.

4. View (Custom): Criamos uma view register (agora corrigida) usando o UserCreationForm.

5. @login_required: Protegemos nossas views de usuários anônimos.

6. request.user: Usamos esta variável mágica para:

○ Filtrar dados: Tarefa.objects.filter(user=request.user)

○ Salvar dados: tarefa.user = request.user

○ Garantir segurança: get_object_or_404(Tarefa, pk=pk, user=request.user)

7. Templates: Usamos {% if user.is_authenticated %} para mostrar conteúdo condicional.