Aula 03

1. Cross-Site Request Forgery (CSRF)

O Cross-Site Request Forgery ou CSRF é uma vulnerabilidade na segurança da web que possibilita que alguma ameaça se passe por clientes comuns. Assim, ela pode se disfarçar como o servidor e passar informações através do método POST.

O CSRF Token consiste em uma série de caracteres aleatórios, gerados a cada formulário a ser preenchido pelo usuário que é enviado pelo servidor. Após o recebimento pelo usuário, o token é checado novamente. O servidor só aceita o POST caso o CSRF Token se provar igual ao enviado inicialmente.

Diferentemente do session token e dos cookies, o CSRF Token não pode ser utilizado por um hacker mal-intencionado. Dessa forma, a existência do CSRF Token é crucial em todo formulário da web, para que o envio de formulários não seja forjado por terceiros.

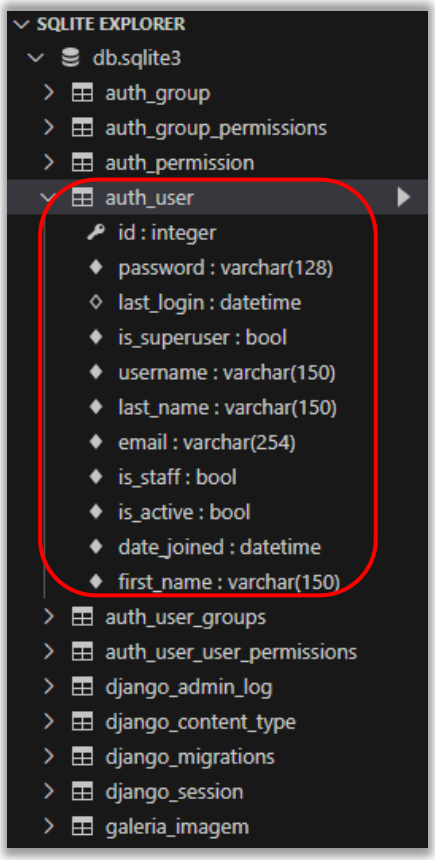
2. Usuário

2.1. Registro do Usuário

Agora que temos os formulários de login e cadastro prontos, já podemos realizar o registro do usuário junto ao banco de dados.

Até o momento, para realizarmos alguma interação com o banco de dados, temos que criar uma classe no arquivo **models.py** e realizar todo o processo já visto.

Acontece que isso não faz sentido, nesse caso. Pois, qual é a necessidade de criar uma tabela de controle de usuário se ela já existe. Se olharmos o banco de dados **db.sqlite3**, podemos ver que existe uma tabela chamada **auth_user**. Como o nome sugere, essa tabela é usada para registrar e autenticar os usuários.



Se observamos ela, toda a estrutura já está pronta. Então não tem o porquê de reinventar a roda. Se realizarmos uma busca na tabela, veremos o registro do nosso usuário admin lá, com a senha criptografada, é claro.

Voltando à tela de cadastro, quando usamos esse tipo de formulário, é comum usar a própria página para validar. Então, passaremos a própria página no campo action do formulário HTML.

Veja como está:

C:\silver-enigma\usuario\templates\usuario\paginas\cadastro.html

```
16
15      <form action="{% url 'usuario:cadastro' %}" method="POST">
14          {% csrf_token %}
```

Partindo dessa ideia, a validação vai se dar na função que faz a requisição da página.

Veja como ela vai ficar:

C:\silver-enigma\usuario\views.py

```
1  from django.shortcuts import render, redirect
2  from django.contrib.auth.models import User
3  from usuario.forms import CadastroForms, LoginForms
4
5  def view_login(request):
6      formulario = LoginForms()
7      return render(request, 'usuario/paginas/login.html', context={'formulario':formulario})
8
9  def view_cadastro(request):
10     formulario = CadastroForms()
11
12     if request.method == 'POST':
13         formulario = CadastroForms(request.POST)
14
15         if formulario.is_valid():
16             if formulario['senha_1'].value() != formulario['senha_2'].value():
17                 return redirect('usuario:cadastro')
18
19             nome = formulario['nome_cadastro'].value()
20             email = formulario['email_cadastro'].value()
21             senha = formulario['senha_1'].value()
22
23             if User.objects.filter(username=nome).exists():
24                 return redirect('usuario:cadastro')
25
26             novo_usuario = User.objects.create_user(
27                 username = nome,
28                 email = email,
29                 password = senha
30             )
31             novo_usuario.save()
32             return redirect('usuario:login')
33
34     return render(request, 'usuario/paginas/cadastro.html', context={'formulario':formulario})
```

Veja a grande atualização feita na função.

1) está sendo realizada mais duas importações novas;
   a) a primeira é a função **redirect**, que será usada para redirecionar para uma página específica;
   b) a segunda é a classe **User**, que será usada para o cadastro do novo usuário;
2) está sendo feito um teste para verificar se a requisição da página foi feita usando o método POST;
3) se for, vai receber o formulário preenchido;
4) então, é feita uma verificação se o formulário enviado está válido;
5) depois vai testar se a senha do primeiro campo está igual a senha do segundo campo;

6) se a senha for diferente, vai ser feito o redirecionamento à página cadastro.html com o formulário em branco;
7) se for a mesma, vai pegar os resultados enviados pelo formulário e salvar em variáveis;
   a) como a senha é a mesma para os campos senha_1 e senha_2, não tem diferença de qual pegar;
8) depois, é feito um teste para ver se o usuário que está se querendo criar ainda não existe na tabela (se existir, é enviado novamente para a página de cadastro.html);
9) se o usuário ainda não existir, é criado um objeto com os dados do formulário;
10) por fim, é salvo no banco de dados;
11) se verificar o banco de dados, o usuário cadastrado estará lá;

Repare que o nome dos campos retornados do formulário são os mesmos que foram usados nos atributos da classe **CadastroForms**.

2.2. Login do Usuário

Uma vez que o usuário se cadastra, é então encaminhado para a tela de login. Mas ainda não há qualquer lógica para realizar o login.

Veja como ficará a função:

C:\silver-enigma\usuario\views.py

```python
from django.shortcuts import render, redirect
from django.contrib import auth
from django.contrib.auth.models import User
from usuario.forms import CadastroForms, LoginForms

def view_login(request):
    formulario = LoginForms()

    if request.method == 'POST':
        formulario = LoginForms(request.POST)

        if formulario.is_valid():
            nome = formulario['nome_login'].value()
            senha = formulario['senha'].value()

            usuario = auth.authenticate(
                request,
                username = nome,
                password = senha
            )

            if usuario is not None:
                auth.login(request, usuario)
                return redirect('galeria:index')
            else:
                return redirect('usuario:login')

    return render(request, 'usuario/paginas/login.html', context={'formulario':formulario})

def view_cadastro(request):
    formulario = CadastroForms()
```

Agora, veja como está funcionando:
1) primeiro está sendo feita a importação do auth;
2) depois, é feita a verificação do POST no request e se o formulário é válido;
3) após, é feita a captura dos valores do formulário;
4) usando o auth, é criado um objeto a partir da variável request e dos dados enviados pelo formulário;
5) é verificado se a chamada no auth.authenticate não gera um valor nulo;
6) depois, é chamada a função login do auth para realizar o login do usuário;
7) e retornado à página index da galeria;
8) se acontecer algum erro, é redirecionado para a página de login novamente;

2.3. Logout do Usuário

Agora que já realizamos o login, temos que ser capazes de realizar o logout também.

Para isso, vamos criar uma função em **/usuario/views.py** para realizar essa tarefa. Veja como ela vai ficar:

```
C:\silver-enigma\usuario\views.py
56
57 def logout(request):
58     auth.logout(request)
59     return redirect('usuario:login')
60
```

Repare que ela é bem simples. Como a funcionalidade será apenas o logout do usuário, não há necessidade de criar uma página dedicada a isso. Embora tenhamos que cadastras ela no arquivo **/usuario/urls.py** para que o link seja visível.

Veja como ficará:

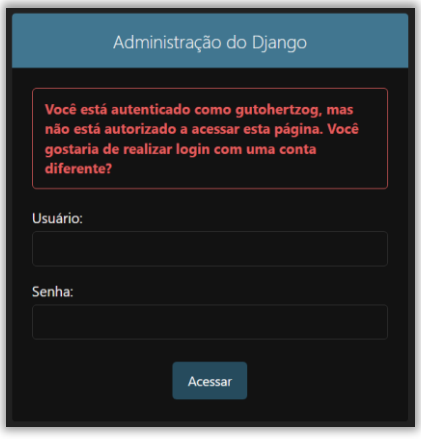
```
C:\silver-enigma\usuario\urls.py
1 from django.urls import path
2 from usuario.views import logout, view_login, view_cadastro
3
4 app_name = 'usuario'
5
6 urlpatterns = [
7     path('login/', view_login, name='login'),
8     path('cadastro/', view_cadastro, name='cadastro'),
9     path('logout/', logout, name='logout'),
10 ]
```

Ali, estamos apenas criando a url para a chamada da função de logout.

Agora, por fim, tem que ainda adicionar o link no menu. Veja abaixo:

```
C:/silver-enigma/meus_templates/global/parciais/menu.html
1  <nav>
2      <ul>
3          <li><a href="{% url 'admin:index' %}">ADMIN</a></li>
4          <li><a href="{% url 'galeria:index' %}">GALERIA</a></li>
5          <li><a href="{% url 'galeria:imagens' %}">IMAGENS</a></li>
6          <li><a href="{% url 'usuario:login' %}">LOGIN</a></li>
7          <li><a href="{% url 'usuario:cadastro' %}">CADASTRO</a></li>
8          <li><a href="{% url 'usuario:logout' %}">LOGOUT</a></li>
9      </ul>
10 </nav>
11 <div>
12     <form action="{% url 'galeria:busca' %}">
13         <input type="text" name="buscando" placeholder="o que quer buscar?">
14         <button type="submit">buscar</button>
15     </form>
16 </div>
```

Veja o que acontece se tentar entrar na página administrativa quando logado:



3. Mensagens

Repare que temos trocentas coisas acontecendo agora com o usuário, mas nenhuma forma de notificação. Para isso, o Django tem uma biblioteca específica para ajudar.

Vamos usá-la no arquivo **/usuario/views.py**. Veja como vai ficar:

C:\silver-enigma\usuario\views.py

```python
from django.shortcuts import render, redirect
from django.contrib import auth
from django.contrib import messages
from django.contrib.auth.models import User
from usuario.forms import CadastroForms, LoginForms

def view_login(request):
    formulario = LoginForms()

    if request.method == 'POST':
        formulario = LoginForms(request.POST)

        if formulario.is_valid():
            nome = formulario['nome_login'].value()
            senha = formulario['senha'].value()

            usuario = auth.authenticate(
                request,
                username = nome,
                password = senha
            )

            if usuario is not None:
                auth.login(request, usuario)
                messages.success(request, f'usuário {nome} logado com sucesso!')
                return redirect('galeria:index')
            else:
                messages.error(request, 'Erro ao tentar logar!')
                return redirect('usuario:login')

    return render(request, 'usuario/paginas/login.html', context={'formulario':formulario})

def view_cadastro(request):
```

C:\silver-enigma\usuario\views.py

```python
def view_cadastro(request):
    formulario = CadastroForms()

    if request.method == 'POST':
        formulario = CadastroForms(request.POST)

        if formulario.is_valid():
            if formulario['senha_1'].value() != formulario['senha_2'].value():
                messages.error(request, 'As senhas não são iguais!')
                return redirect('usuario:cadastro')

            nome = formulario['nome_cadastro'].value()
            email = formulario['email_cadastro'].value()
            senha = formulario['senha_1'].value()

            if User.objects.filter(username=nome).exists():
                messages.error(request, 'O usuário já existe!')
                return redirect('usuario:cadastro')

            novo_usuario = User.objects.create_user(
                username = nome,
                email = email,
                password = senha
            )
            novo_usuario.save()
            messages.success(request, 'Cadastro realizado com sucesso!')
            return redirect('usuario:login')

    return render(request, 'usuario/paginas/cadastro.html', context={'formulario':formulario})

def logout(request):
    auth.logout(request)
    messages.success(request, 'Logout efetuado com sucesso!')
    return redirect('usuario:login')
```

Como pode ser visto acima, é realizado a importação do **messages**. Através dela, são chamadas as funções **success** ou **error** de acordo com a necessidade. Veja que elas são chamadas sempre antes da função **redirect** e depois de ser feito algum teste, como usuário existente, senha incorreta etc.

Agora, basta atualizar os arquivos HTMLs que são usados. Veja abaixo:

C:\silver-enigma\usuario\templates\usuario\paginas\base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>usuario</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link href="{% static 'usuario/css/estilos.css' %}" rel="stylesheet">
    </head>
    <body>
        {% for mensagem in messages %}
        <div><p>{{ mensagem }}</p></div>
        {% endfor %}
        {% include 'global/parciais/menu.html' %}
        {% block conteudo %} {% endblock conteudo %}
        {% include 'usuario/parciais/footer.html' %}
    </body>
</html>
```

Como todos os arquivos são herdados desses arquivos base (dos aplicativos **usuario** e **galeria**), vai ser melhor adicionar a chamada da mensagem neles (como esse arquivo está sendo usado o mesmo layout para ambos os aplicativos, ele poderia ser colocado na pasta de páginas globais e herdado por ambos os aplicativos).

4. Atividades

1. Aplique o que foi visto na aula nos seus projetos.