04

Guardando dados na sessão

Transcrição

Antes de melhorarmos a usabilidade da nossa autenticação, precisamos entender um pouco o funcionamento desse processo.

Nós queremos, por exemplo, que um usuário não logado seja redirecionado a partir da página `/novo` para a tela `/login`. Nela, ele preencherá os dados `usuario` e `senha` e mandará um request para ser autenticado pelo servidor. O servidor irá redirecioná-lo para a página `/novo` ou para a página `/` (a lista de jogos), permitindo o acesso.

Na verdade, esse processo de redirecionamento não é tão simples, pois ele faz dois ciclos de requisição. Por exemplo, quando um usuário manda as informações erradas, o servidor faz um redirecionamento. Isso faz com que o navegador mande outra requisição para `/login` e receba uma resposta que o envia de volta à tela `/login`.

Mas como fazemos para guardar uma informação — por exemplo, um nome de usuário — que se mantenha armazenada durante todos esses ciclos de requisição? A única informação que tínhamos guardada era o objeto `request`. Porém, ele não permite o acesso a dados de mais de um ciclo de requisição.

Para resolvermos esse problema, o Flask nos disponibiliza um objeto chamado `session` (que precisa ser importado do pacote Flask), que armazena dados por mais de um ciclo de requisição. Além disso, cada usuário tem uma sessão. Para testarmos esse objeto, vamos simplesmente exibir o nome do nosso usuário na sessão.

Faremos isso na função `autenticar`, antes de efetuar o redirecionamento de sucesso para `/`. O objeto `session` tem, dentro dele, uma estrutura que é um dicionário. Nele, podemos incluir valores. Nesse caso, incluiremos um valor com a chave `usuario_logado`. O nome do usuário está no formulário, então usaremos `request.form['usuario']`.

```
@app.route('/autenticar', methods=['POST',])
def autenticar():
    if 'mestra' == request.form['senha']:
        session ['usuario_logado'] = request.form['usuario']
        return redirect('/')
    else :
        return redirect ('/login')
```

O próximo passo é exibir uma mensagem dizendo se esse usuário está logado ou não, e os dados dessa mensagem devem estar na sessão. Na sessão padrão do Flask, esses dados são guardados em um cookie, que é uma informação que fica guardada no navegador. Dessa forma, conseguiremos guardar uma mensagem para ser mostrada depois que for feito o redirecionamento.

Queremos exibir uma mensagem quando o usuário faz login com sucesso e também quando esse login não é bem sucedido. No Flask, temos uma função chamada `flash()` (de *flash message*) que mostra uma mensagem rapidamente. Essa função deve ser importada do pacote Flask, e recebe um texto que deverá ser exibido.

No caso do login bem sucedido, passaremos o nome do usuário concatenado com a mensagem "logou com sucesso!". Na outra situação, a mensagem será "Não logado, tente de novo!".

```
@app.route('/autenticar', methods=['POST',])
def autenticar():
    if 'mestra' == request.form['senha']:
        session ['usuario_logado'] = request.form['usuario']
        flash(request.form['usuario'] + ' logou com sucesso!')
        return redirect('/')
    else :
        flash('Não logado, tente de novo!')
        return redirect ('/login')
```

Ainda falta definirmos onde essa mensagem será exibida. Por exemplo, queremos mostrar a mensagem de erro na tela de login. Para isso, teremos que usar o seguinte código:

```
{% with messages = get_flashed_messages() %}
    {% if messages %}
        <ul id="messages" class="list-unstyled">
        {% for message in messages %}
            <li class="alert alert-success">{{ message }}</li>
        {% endfor %}
        </ul>
    {% endif %}
{% endwith %}
```

O *statement* `with` do Python funciona basicamente para podermos usar as mensagens internamente. A função `if messages` verifica se existe alguma mensagem na lista de mensagens. Se sim, será exibida uma `ul` com uma `li` (uma impressão) para cada mensagem dentro de `messages`. No caso, temos uma única mensagem.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Jogoteca</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.css') }}">
  </head>
  <body>
    <div class="container">
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <ul id="messages" class="list-unstyled">
                {% for message in messages %}
                    <li class="alert alert-success">{{ message }}</li>
                {% endfor %}
                </ul>
            {% endif %}
        {% endwith %}
        <h1>Faça seu login</h1>
        <form method="POST" action="/autenticar">
            <p><label>Nome de usuário:</label> <input class="form-control" type="text" name="usu
            <p><label>Senha:</label> <input class="form-control" type="password" name="senha" re
            <p><button class="btn btn-primary" type="submit">Entrar</button></p>
        </form>
    </div>
```

```
</body>
</html>
```

Agora, precisamos repetir essa construção na tela `template.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Jogoteca</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='bootstrap.css') }}">
  </head>
  <body>
    <div class="container">
            {% with messages = get_flashed_messages() %}
                {% if messages %}
                    <ul id="messages" class="list-unstyled">
                    {% for message in messages %}
                        <li class="alert alert-success">{{ message }}</li>
                    {% endfor %}
                    </ul>
                {% endif %}
            {% endwith %}
        <div class="page-header">
            <h1>{{ titulo }}</h1>
        </div>

        {% block conteudo %}{% endblock %}

    </div>
    </body>
</html>
```

Dessa forma, caso tenhamos uma mensagem para ser exibida, isso aconterá em todas as telas. Com essas alterações implementadas, tentaremos entrar com um usuário qualquer e a senha `mestra` na nossa tela de login. Isso resultará em um erro:

> **builtins.RuntimeError**
>
> RuntimeError: The session is unavailable because no secret key was set. Set the secret_key on the application to something unique and secret.

Para trabalhar com sessões e utilizar os dados armazenados nos cookies, o Flask precisa criptografar esses dados de alguma forma. O Flask faz uma assinatura que encripta esses dados e impede que o usuário altere e os reenvie modificados. Somente o próprio Flask consegue alterar os dados da sessão, a partir de uma `secret key` que é estabelecida.

> Essa assinatura não impede que o usuário leia o que está lá dentro, ou seja, não coloque informações sigilosas dentro da sessão!

A configuração de uma `secret key` é feita de maneira bem simples na nossa aplicação. Basta utilizarmos `app.secret_key`, passando uma string com o valor que queremos definir para essa chave, por exemplo, `alura`:

```
from flask import Flask, render_template, request, redirect, session, flash

app = Flask(__name__)
app.secret_key = 'alura'
```

Com a `secret key` implementada, quando fizermos login com a senha `mestra`, receberemos uma mensagem em verde no topo da tela de lista:

luan logou com sucesso!

Se atualizarmos a página, essa mensagem não aparecerá mais, porque só faz sentido que ela apareça uma vez após a requisição. Se errarmos a senha, a outra mensagem que configuramos irá aparecer:

Não logado, tente de novo!

Nossa autenticação está funcionando corretamente, mas a página `/novo` continua aberta para usuários não-logados. Além disso, ainda não implementamos a função de *deslogar*. Adiante, trabalharemos as soluções para esses problemas.