08

Múltiplos usuários

Transcrição

Nós definimos a senha `mestra` como padrão, mas se essa senha vazasse, seria a mesma coisa que não ter autenticação nenhuma. Para resolvermos esse problema, tentaremos criar vários usuários, cada um com sua própria senha.

Para guardarmos informações de um conceito — nesse caso, de usuários —, podemos criar uma classe para representar objetos desse tipo.

O que precisamos ter na classe `Usuario` ? Apesar de importante, vamos ignorar por enquanto o encapsulamento das nossas variáveis, para sermos mais práticos e focarmos no aprendizado do Flask.

Quando criamos um usuário, algumas informações importantes são o `id` (o nome de usuário, que é único), um `nome` (o nome próprio do usuário) e uma `senha` que fica armazenada. Vamos criar os objetos referentes a cada uma dessas informações.

```
class Usuario:
    def __init__(self, id, nome, senha):
        self.id = id
        self.nome = nome
        self.senha = senha
```

Com essa classe criada, já conseguiremos ilustrar alguns usuários no nosso código, da mesma forma que criamos alguns jogos padrões. Por exemplo, o `usuario1` . Para isso, vamos instanciar o tipo `Usuario()` , passando um `id` ("Luan"), um `nome` ("Luan Marques") e a `senha` ("1234").

```
usuario1 = Usuario('luan', 'Luan Marques', '1234')
```

Esse usuário está com a senha aberta, mas resolveremos isso no futuro. Como cada usuário deve ter sua própria senha, vamos criar outros usuários para realizar os nossos testes:

```
usuario1 = Usuario('luan', 'Luan Marques', '1234')
usuario2 = Usuario('nico', 'Nico Steppat', '7a1')
usuario3 = Usuario('flavio', 'Flávio', 'javascript')
```

Com esses usuários, vamos começar a implementar a ideia do objeto `Usuario` no nosso sistema. Por enquanto, na nossa autenticação, estamos comparando a `senha` passada pelo usuário com a `senha` que temos na aplicação, sem verificar se ela é de algum usuário.

Inicialmente, precisaremos pegar as informações de usuário, que acabamos de criar. Uma maneira de procurarmos esses usuários é por meio de um dicionário (um conjunto de dados), que chamaremos de `usuarios` . Nele, cada usuário terá uma chave (por exemplo, `usuario1.id`), e um dado que o representa, no caso, o próprio objeto do usuário.

```
usuarios = { usuario1.id: usuario1,
             usuario2.id: usuario2,
```

```
                usuario3.id: usuario3 }
```

Na função `autenticar()`, recebemos a `senha` e um `usuario`, que é um nome de usuário. Antes de qualquer coisa, é possível verificarmos se esse usuário existe dentro do nosso dicionário `usuarios`, ou seja, `if request.form['usuario'] in usuarios`.

Se essa condição for satisfeita, passaremos uma chave para nosso dicionário (`usuarios[request.form['usuario']]`) de modo a retornar o valor dessa chave (que é o usuário com seu `id` específico).

Com o usuário, podemos pegar a senha utilizando `.senha`. Em seguida, precisamos verificar se ela é igual à senha que está sendo passada no formulário de login (`if usuarios[request.form['usuario']].senha == request.form['senha']`). Com essa comparação, saberemos se a senha passada para determinado usuário está correta.

Se a senha estiver correta, continuaremos com os passos que definimos antes: o usuário receberá uma mensagem de que logou com sucesso e será redirecionado para a página de origem.

Para limparmos um pouco nosso código, podemos simplesmente guardar o nome de usuário que está em `usuarios[request.form['usuario']]` em uma variável `usuario`. Além disso, na mensagem que é exibida na tela, é interessante colocarmos o nome próprio do usuário que está acessando o sistema. Assim, teremos:

```
@app.route('/autenticar', methods=['POST',])
def autenticar():
    if request.form['usuario'] in usuarios:
        usuario = usuarios[request.form['usuario']]
        if usuario.senha == request.form['senha']:
            session['usuario_logado'] = usuario.id
            flash(usuario.nome + ' logou com sucesso!')
            proxima_pagina = request.form['proxima']
            return redirect(proxima_pagina)

    else :
        flash('Não logado, tente de novo!')
        return redirect(url_for('login'))
```

Com tudo isso implementado, podemos testar o acesso na página `/login`. Se entrarmos, por exemplo, com o usuário `Luan` e a senha `1234`, seremos redirecionados para a página de lista (`/`) e receberemos a mensagem "Luan Marques logou com sucesso!". A mesma coisa acontecerá com os outros usuários e suas respectivas senhas.

Agora nosso login está funcionando bem melhor, certo? Nós conseguimos organizar a lógica de criação de jogos e de usuários por meio de objetos. Porém, sempre que criarmos um novo jogo, esses dados só estão sendo inseridos no servidor. Ou seja, sempre que reiniciarmos o servidor, só aparecerão os jogos inicializados nele (os três que criamos no início do curso).

Precisamos persistir esses dados de alguma forma. Mas isso nós veremos só na parte 2 (https://cursos.alura.com.br/course/flask-upload-persistencia-javascript-jquery) do curso. Até lá!