Controle de Sessão: Autenticação e Autorização

Disciplina: Desenvolvimento Web

Prof. Dr. Rafael Will M. de Araujo

Conteúdo

Recebendo valores em uma aplicação Flask

- Até o momento vimos como exibir (renderizar) páginas no Flask através de *templates*.
- Uma das formas de tornar o nosso sistema ainda mais interativo é recebendo dados do usuário. Há vários casos em que isso é desejável:
  - Fazer o cadastro de algum cliente, produto, etc;
  - Receber algum *feedback* ou mensagem dos usuários pelo próprio sistema;
  - Realizar consultas no sistema;
  - Autenticar usuários através de um login e senha;
  - etc.

Recebendo valores em uma aplicação Flask

- Até o momento vimos como exibir (renderizar) páginas no Flask através de *templates*.
- Uma das formas de tornar o nosso sistema ainda mais interativo é recebendo dados do usuário. Há vários casos em que isso é desejável:
  - Fazer o cadastro de algum cliente, produto, etc;
  - Receber algum *feedback* ou mensagem dos usuários pelo próprio sistema;
  - Realizar consultas no sistema;
  - Autenticar usuários através de um login e senha;
  - etc.
- Note que os *path parameters* não são adequados para essa tarefa, pois seu objetivo é facilitar a criação de rotas dentro de uma aplicação Flask;

Recebendo valores em uma aplicação Flask

- Até o momento vimos como exibir (renderizar) páginas no Flask através de *templates*.
- Uma das formas de tornar o nosso sistema ainda mais interativo é recebendo dados do usuário. Há vários casos em que isso é desejável:
  - Fazer o cadastro de algum cliente, produto, etc;
  - Receber algum *feedback* ou mensagem dos usuários pelo próprio sistema;
  - Realizar consultas no sistema;
  - Autenticar usuários através de um login e senha;
  - etc.
- Note que os *path parameters* não são adequados para essa tarefa, pois seu objetivo é facilitar a criação de rotas dentro de uma aplicação Flask;
- Na aula de formulários HTML nós aprendemos como construir formulários e como os dados inseridos pelos usuários podem ser enviados ao servidor;
- Agora vamos aprender como receber esses dados no lado do servidor.

Recebendo parâmetros via GET

- Para receber os valores contidos na *query string* (parte da URL que que atribui valores a parâmetros especificados), utilizamos o objeto `request.args`:
  - ▷ Ele é um dicionário, no qual podemos consultar uma chave através do método: `request.args.get('nome_da_chave')`;
  - ▷ Se a chave não existir, será devolvido o valor *None*;
  - ▷ A chave será um parâmetro da *query string*, ou seja, um *name* enviado pelo formulário.
- Para isso, você precisa importar o objeto `request` a partir do módulo *flask*.

Aplicação em Flask

```
1  from flask import Flask, render_template, request

2  app = Flask(__name__)

3  @app.route('/pagina')
4  def recebe_parametros():
5    nome = request.args.get('nome')
6    idade = request.args.get('idade')
7    return f'Nome e idade recebidos: {nome}, {idade} anos'

8  app.run(debug=True)
```

- Teste o código com a seguinte URL: `http://127.0.0.1:5000/pagina?nome=Rafael&idade=123`

Recebendo parâmetros via GET: exemplo 1

Aplicação em Flask

```
1  from flask import Flask, render_template, request
2  app = Flask(__name__)
3  @app.route('/pagina')
4  def recebe_parametros():
5    nome = request.args.get('nome')
6    idade = request.args.get('idade')
7    return f'Nome e idade recebidos: {nome}, {idade} anos'
8  @app.route('/formulario')
9  def formulario():
10   return render_template('formulario.html')
11 app.run(debug=True)
```

Arquivo */templates/formulario.html*

```
1  <form method="GET" action="/pagina">
2    <div>
3      <label for="tx-nome">Nome: </label>
4      <input type="text" name="nome" id="tx-nome">
5    </div>
6    <div>
7      <label for="tx-idade">Idade: </label>
8      <input type="number" name="idade" id="tx-idade">
9    </div>
10   <div>
11     <input type="submit" value="Enviar">
12   </div>
13 </form>
```

- Note como utilizamos o atributo `action` do formulário.

Recebendo parâmetros via GET: exemplo 2

Aplicação em Flask

```
1  from flask import Flask, render_template, request

2  app = Flask(__name__)

3  @app.route('/formulario')
4  def formulario():
5    nome = request.args.get('nome')
6    idade = request.args.get('idade')
7    if nome is None or idade is None:
8      return render_template('formulario.html')
9    else:
10     return f'Nome e idade recebidos: {nome}, {idade} anos'

11 app.run(debug=True)
```

Arquivo */templates/formulario.html*

```
1  <form method="GET">
2    <div>
3      <label for="tx-nome">Nome: </label>
4      <input type="text" name="nome" id="tx-nome">
5    </div>
6    <div>
7      <label for="tx-idade">Idade: </label>
8      <input type="number" name="idade" id="tx-idade">
9    </div>
10   <div>
11     <input type="submit" value="Enviar">
12   </div>
13 </form>
```

- Se não utilizarmos o atributo `action` do formulário, ele será enviado para a mesma URL.

Recebendo parâmetros via POST

- Para receber os valores enviados via POST, utilizamos o objeto `request.form`:
  - Assim como o `request.args`, o `request.form` também é um dicionário;
  - Podemos consultar uma chave através dos métodos:
    - `request.form.get('nome_da_chave')`, para valores simples;
    - `request.form.getlist('nome_da_chave')`, este último para campos com múltiplos valores (como *checkboxes* de um mesmo grupo, por exemplo). (Este método também funciona no objeto `request.args`).
  - Uma chave será um parâmetro enviado no cabeçalho da requisição, isto é, um *name* enviado pelo formulário.

Recebendo parâmetros via POST

- Para os próximos dois exemplos, considere o seguinte template:

Arquivo *templates/formulario.html*

```
1  <form method="POST">
2    <div>
3      <label for="tx-nome">Nome: </label>
4      <input type="text" name="nome" id="tx-nome">
5    </div>
6    <div>
7      <label for="tx-idade">Idade: </label>
8      <input type="number" name="idade" id="tx-idade">
9    </div>
10   <div>
11     Atividades favoritas:<br>
12     <input type="checkbox" name="atividades" value="Comer" id="cb-comer">
13     <label for="cb-comer">Comer</label><br>
14     <input type="checkbox" name="atividades" value="Dormir" id="cb-dormir">
15     <label for="cb-dormir">Dormir</label><br>
16     <input type="checkbox" name="atividades" value="Programar" id="cb-programar">
17     <label for="cb-programar">Programar</label><br>
18   </div>
19   <div>
20     <input type="submit" value="Enviar">
21   </div>
22 </form>
```

Recebendo parâmetros via POST: exemplo 1

Aplicação em Flask

```
1  from flask import Flask, render_template, request

2  app = Flask(__name__)

3  @app.route('/formulario')
4  def formulario():
5    nome = request.form.get('nome')
6    idade = request.form.get('idade')
7    atividades = request.form.getlist('atividades')
8    if nome is None or idade is None:
9      return render_template('formulario.html')
10   else:
11     return f'Nome, idade e atividades recebidas: {nome}, {idade} anos, {atividades}'

12 app.run(debug=True)
```

- Qual o problema com o código acima?

Recebendo parâmetros via POST: exemplo 1

Aplicação em Flask

```
1  from flask import Flask, render_template, request

2  app = Flask(__name__)

3  @app.route('/formulario')
4  def formulario():
5    nome = request.form.get('nome')
6    idade = request.form.get('idade')
7    atividades = request.form.getlist('atividades')
8    if nome is None or idade is None:
9      return render_template('formulario.html')
10   else:
11     return f'Nome, idade e atividades recebidas: {nome}, {idade} anos, {atividades}'

12 app.run(debug=True)
```

- Qual o problema com o código acima?
- A rota não aceita requisições via POST!

Recebendo parâmetros via POST: exemplo 2

- Solução: adicionar o parâmetro nomeado `methods` ao método `app.route()` com a lista dos métodos permitidos.
  - ▷ No nosso exemplo, a rota */formulario* aceita tanto requisições via GET (para exibir o formulário), como também via POST (para mostrar os dados recebidos no servidor a partir do formulário).

Aplicação em Flask

```
1  from flask import Flask, render_template, request

2  app = Flask(__name__)

3  @app.route('/formulario', methods=['GET', 'POST'])
4  def formulario():
5    nome = request.form.get('nome')
6    idade = request.form.get('idade')
7    atividades = request.form.getlist('atividades')
8    if nome is None or idade is None:
9      return render_template('formulario.html')
10   else:
11     return f'Nome, idade e atividades recebidas: {nome}, {idade} anos, {atividades}'

12 app.run(debug=True)
```

Conteúdo

Fluxo de Autenticação

- Existem diversos mecanismos de autenticação de usuários (sejam pessoas ou outros computadores): certificados digitais, biometria facial ou dos dedos, *captchas*, etc. Mas o mais comum encontrado nos sites da internet é o **sistema de usuário e senha**.
- Dependendo do mecanismo de autenticação utilizado, o fluxo do sistema de autenticação pode variar um pouco, mas a estrutura é muito similar entre eles.

Fluxo de Autenticação

- Existem diversos mecanismos de autenticação de usuários (sejam pessoas ou outros computadores): certificados digitais, biometria facial ou dos dedos, *captchas*, etc. Mas o mais comum encontrado nos sites da internet é o **sistema de usuário e senha**.
- Dependendo do mecanismo de autenticação utilizado, o fluxo do sistema de autenticação pode variar um pouco, mas a estrutura é muito similar entre eles.

Envio dos dados de autenticação

- Essa fase é a que envolve a interação direta do usuário. Em nossos exemplos (nos próximos slides), o usuário deverá digitar seu nome de usuário, sua senha e submeter o formulário ao servidor. Em alguns casos, temos a utilização de um *captcha*, um mecanismo de identificação de robôs na navegação dos sites.
- Algumas outras técnicas permitem que o nosso site delegue essa função para outros provedores de identidade. Esse é o caso do OAuth2, que permite que a nossa aplicação delegue para um outro serviço ou aplicação (como o Google, Facebook, etc) a verificação de autenticidade do usuário. Nesse caso, pulamos direto para a etapa 4, pois a 1, 2 e 3 seriam feitos por esse serviço.

Conteúdo

Exemplo 1: página de login

- Vamos criar uma página de login simples, que solicita o nome de usuário e senha de uma pessoa.
- Note que já deixamos a *template tag* `{{ erro }}` para exibir possíveis mensagens de erro durante o processo de autenticação.

Arquivo *templates/login.html*

```
1  <form method="POST">
2    <div>
3      <label for="tx-usuario">Usuário:</label><br>
4      <input type="text" name="usuario" id="tx-usuario">
5    </div>
6    <div>
7      <label for="tx-senha">Senha:</label><br>
8      <input type="password" name="senha" id="tx-senha">
9    </div>
10   <div>
11     <p>{{ erro }}</p>
12   </div>
13   <div>
14     <input type="submit" value="Entrar">
15   </div>
16 </form>
```

Exemplo 1: criando a sessão do usuário

- A seguir, vamos construir um *Controller* que recebe dados de login de um usuário via POST e o autentica.
  - ▷ Por enquanto, a nossa autenticação é bem simples: o usuário pode ser "rafael" com senha "1234", ou então "maria" com senha "1111";
  - ▷ Se o usuário for autenticado com sucesso, ele será **redirecionado** para a rota `/area_logada`.
- Caso seja feita uma requisição via GET, será exibida a página *login.html*;

Controller que cria a sessão do usuário (arquivo *app.py*)

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

2  app = Flask(__name__)
3  app.secret_key = 'CHAVE-MUITO-SECRETA'

4  @app.route('/login', methods=['GET', 'POST'])
5  def login():
6    msg_erro = ''
7    if request.method == 'POST':  # verifica se a requisição é via POST
8      usuario = request.form.get('usuario')
9      senha = request.form.get('senha')
10     if usuario == 'rafael' and senha == '1234':  # verifica as credenciais do usuário
11       session['usuario'] = 'rafael'
12       return redirect('/area_logada')   # redireciona para a rota /area_logada
13     elif usuario == 'maria' and senha == '1111':
14       session['usuario'] = 'maria'
15       return redirect('/area_logada')
16     else:
17       msg_erro = 'Usuário ou senha inválidos, tente novamente'
18   return render_template('login.html', erro=msg_erro)   # se não for POST, renderize a página de login
```

Exemplo 1: o cookie de sessão

- Veja que para "saber" que um usuário está autenticado, guardamos o seu nome de usuário no `session` (objeto de sessão) do Flask. Esse objeto se comporta como um dicionário Python, e permite guardar os dados referentes a essa sessão de usuário.
- O Flask pega os dados presentes no `session` e serializa (traduz para *string*) no *cookie*, usando criptografia, e o envia ao cliente. Toda requisição que esse *cookie* estiver presente, ele será desserializado (traduzido para objeto) e o objeto `session` fica povoado.
- Note também que foi necessário criar uma chave secreta na aplicação Flask. Isso é necessário porque o Flask criptografa os dados de sessão do usuário, uma vez que essa informação será armazenada no computador do usuário através de um *cookie*.
- Os *cookies* são arquivos especiais acessados pelo navegador. Em geral, eles possuem prazo de validade (expiram), ou seja, ao final desse prazo eles são apagados. No Flask, por padrão, o *cookie* de sessão é válido enquanto o navegador estiver aberto.
  - ▷ No Chrome, os *cookies* podem ser visualizados nas "Ferramentas do desenvolvedor" ⇒ "Aplicação" ⇒ Menu "Cookies".

Exemplo 1: a área logada

- Uma vez que a sessão do usuário exista, será possível visualizar a área logada do usuário (rota `/area_logada`).

continuação do arquivo *app.py*

```
1  @app.route('/area_logada')
2  def area_logada():
3    if 'usuario' in session:  # verifica se o usuário está logado
4      nome_pessoa = ''
5      if session['usuario'] == 'rafael':   # verifica quem é o usuário
6        nome_pessoa = 'Rafael Will'
7      elif session['usuario'] == 'maria':
8        nome_pessoa = 'Maria dos Santos'
9      return render_template('area_logada.html', nome=nome_pessoa)
10   else:
11     return redirect('/login')
12
13 @app.route('/logout')
14 def sair():
15   session.clear()  # limpa a sessão
16   return redirect('/login')
```

Arquivo */templates/area_logada.html*

```
1  <p><a href="/logout">Sair</a> da área logada</p>
2  <h1>Bem-vindo à área logada, {{ nome }}!</h1>
```

- Experimente acessar a URL `http://127.0.0.1:5000/area_logada` quando não estiver logado e veja o que acontece.

Exemplo 2: autenticação utilizando um *Model*

- Vamos aprimorar o exemplo anterior, utilizando uma classe que simula o acesso a um Banco de Dados.
- Para isso, vamos criar uma pasta chamada *models*, e dentro dela vamos criar um arquivo chamado *usuario.py* que contém 2 classes:
  - **Aluno**, que serve para facilitar o armazenamento e representação de dados de alunos;
  - **BancoDeDados**, que cria um "banco de dados" artificial através de uma lista e disponibiliza alguns métodos para consultas.
- Apesar da classe **BancoDeDados** no exemplo a seguir criar um BD artificial, o acesso a um banco de dados real não seria muito diferente. Ao invés de percorrer a lista criada dentro dos métodos, teríamos feito uma conexão a algum SGBD, e enviado consultas via linguagem SQL. No final do processo, o resultado da consulta viria no formato de um objeto ou de uma lista de objetos.

Exemplo 2: autenticação utilizando um *Model*

Arquivo */models/usuario.py*

```
1  class Aluno:
2    # Classe que representa dados simplificados de alunos
3    def __init__(self, usuario, senha, nome, curso, data_inicio, notas):
4      self.usuario = usuario
5      self.senha = senha
6      self.nome = nome
7      self.curso = curso
8      self.data_inicio = data_inicio
9      self.notas = notas

10 class BancoDeDados:
11   # Classe que simula operações sobre um banco de dados
12   def __init__(self):
13     # cria um "banco de dados" artificial
14     self.__bd = [
15       Aluno('rafael', '1234', 'Rafael Will', 'Ciência da Computação', '01/02/2021', [6.5, 6.0, 9.8]),
16       Aluno('maria', '1111', 'Maria dos Santos', 'Análise de Sistemas', '13/08/2020', [10.0, 5.5, 8.9]),
17       Aluno('jose', '9876', 'Jose Silva', 'Redes de Computadores', '20/06/2021', [7.5, 7.8, 9.5]),
18       Aluno('ana', '5678', 'Ana Beatriz', 'Administração', '15/01/2019', [6.3, 8.8, 7.2])
19     ]
20
21   def existe_aluno(self, usuario, senha):
22     # checa se um aluno existe pelo seu usuário e senha
23     for aluno in self.__bd:
24       if aluno.usuario == usuario and aluno.senha == senha:
25         return True
26     return False
27
28   def obter_dados(self, usuario):
29     # obtém dados de um aluno através do seu usuário, se esse existir
30     for aluno in self.__bd:
31       if aluno.usuario == usuario:
32         return aluno
33     return None
```

Exemplo 2: autenticação utilizando um *Model*

- Agora podemos importar o nosso model e utilizar os seus métodos para verificar se algum usuário existe e obter os seus dados;
- Note que a lógica verificar os usuários e obter seus dados não está mais nos *Controllers*. Independente do número de usuários cadastrados na aplicação, os *Controllers* não precisam mais serem alterados!

Aplicação do Flask: arquivo *app.py*

```
1  from flask import Flask, render_template, request, session, redirect
2  from models.usuario import BancoDeDados

3  app = Flask(__name__)
4  app.secret_key = 'CHAVE-MUITO-SECRETA'

5  @app.route('/login', methods=['GET', 'POST'])
6  def login():
7    msg_erro = ''
8    if request.method == 'POST':
9      usuario = request.form.get('usuario')
10     senha = request.form.get('senha')
11     bd = BancoDeDados()
12     if bd.existe_aluno(usuario, senha):
13       session['usuario'] = usuario
14       return redirect('/area_logada')
15     else:
16       msg_erro = 'Usuário ou senha inválidos, tente novamente'
17   return render_template('login.html', erro=msg_erro)

18 @app.route('/area_logada')
19 def area_logada():
20   if 'usuario' in session:
21     bd = BancoDeDados()
22     dados_aluno = bd.obter_dados(session['usuario'])
23     return render_template('area_logada.html', aluno=dados_aluno)
24   else:
25     return redirect('/login')
```

Exemplo 2: autenticação utilizando um *Model*

continuação do arquivo *app.py*

```
1  @app.route('/notas')
2  def notas():
3    if 'usuario' in session:
4      bd = BancoDeDados()
5      dados_aluno = bd.obter_dados(session['usuario'])
6      return render_template('notas.html', aluno=dados_aluno)
7    else:
8      return redirect('/login')

9  @app.route('/logout')
10 def sair():
11   session.clear()
12   return redirect('/login')

13 app.run(debug=True)
```

- Note também que é muito mais simples obter os dados dos usuários como objetos da classe **Aluno**. Assim, independente da quantidade de atributos, os nossos *Controllers* não necessitarão de alterações.

Exemplo 2: autenticação utilizando um *Model*

- O código das duas *Views* utilizadas:
  - ▷ A tela de login permanece a mesma do Exemplo 1.

Arquivo */templates/area_logada.html*

```
1 <p><a href="/logout">Sair</a> da área logada | <a href="/notas">Visualizar as minhas notas</a></p>
2 <h1>Bem-vindo à área logada, {{ aluno.nome }}!</h1>
3 <p>Seu curso: {{ aluno.curso }}</p>
4 <p>Início do curso curso: {{ aluno.data_inicio }}</p>
```

Arquivo */templates/notas.html*

```
1 <p><a href="/logout">Sair</a> da área logada | <a href="/area_logada">Voltar para a área logada</a></p>
2 <h1>Bem-vindo à área logada, {{ aluno.nome }}!</h1>
3 <p>Suas notas:</p>
4 <ul>
5   {% for nota in aluno.notas %}
6     <li>{{ nota }}</li>
7   {% endfor %}
8 </ul>
```

Exemplo 2: autenticação utilizando um *Model*

- Uma observação importante é que a nossa aplicação armazena a senha em "texto claro" (*plain text*), o que não é recomendado por questões de segurança. Por motivos didáticos, para manter o foco no processo de autenticação e a aplicação simples, não realizamos esta etapa **importantíssima**.
- Uma forma de não armazenar a senha em *plain text* é utilizar algoritmos de *hashing* seguros, como o SHA-256 ou SHA-512 bits.

Exemplo de como obter o *hash* de uma string em Python

```
1  import hashlib
2  s = 'Um texto qualquer'
3  hash = hashlib.sha256(s.encode('UTF-8')).hexdigest()
4  print(hash)
```

Exemplo 3: modificando o tempo de expiração do cookie de sessão

- As sessões do Flask expiram assim que você fecha o navegador. Uma forma de contornar isso é criando uma sessão permanente:

Criação de uma sessão permanente (com tempo de expiração)

```
1  from datetime import timedelta
2
2  @app.before_request
3  def before_request():
4      session.permanent = True
5      app.permanent_session_lifetime = timedelta(minutes=1, seconds=30)
```

- Adicione o seguinte *Controller* aos exemplos vistos anteriormente.