05

Resolvendo um POST no servidor

Transcrição

Por algum motivo, não estamos conseguindo enviar um POST para o servidor. Mas por que isso está acontecendo? Analisando o nosso código, tudo parece estar correto.

No Flask, quando temos uma rota e não a configuramos com mais informações, por padrão, ela aceita somente o método GET, e o nosso HTML está enviando um POST.

Para configurarmos a rota `/criar`, passaremos uma lista de métodos que ela deve aceitar — no caso, `[post,]` (a vírgula no final é um padrão quando escrevemos listas no Python).

```
@app.route('/criar', methods=['POST',])
```

Vamos reiniciar a aplicação e acessar a URL http://127.0.0.1:5000/novo (http://127.0.0.1:5000/novo). Dessa vez, devemos conseguir criar um novo jogo na lista:

Jogos

Nome	Categoria	Console
Super Mario	Acao	SNES
Pokemon Gold	RPG	GBA
Mortal Kombat	Luta	SNES
Rayman	Indie	Xbox

Funcionou! Agora temos um crud quase completo, com uma lista de jogos e a opção de criar novos jogos. Uma coisa que vem nos atrapalhando é a necessidade de sempre reiniciar a aplicação, o que é bem simples de resolver.

Na aplicação **jogoteca**, temos a função `app.run()`. Nós precisamos reiniciar a aplicação várias vezes porque essa função não está em um contexto de desenvolvimento, no qual ela conseguiria recarregar automaticamente.

Passaremos esse contexto de desenvolvimento com uma variável `debug`, que receberá o valor `true`.

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

Com essa configuração, o próximo recarregamento da aplicação deve retornar a seguinte mensagem:

- Restarting with stat
- Debugger is active!
- Debugger PIN: 812-210-734
- Running on http://127.0.0.1:5000/ (http://127.0.0.1:5000/) (Press CTRL+C to quit)

Ou seja, a aplicação foi reiniciada e o contexto `Debugger` está ativo. Isso significa que, se fizermos alguma alteração no nosso código Python e salvarmos ela com "Ctrl + S" no Windows (ou "Command + S" no Mac), o `Debugger` enxergará que essa alteração foi feita e, automaticamente, reiniciará a aplicação.

Como exemplo, em vez de `ola()`, vamos definir a nossa primeira função como `index()`, e mudar a rota inicial para apenas `/`, de modo que a primeira página acessada seja http://127.0.0.1:5000/ (http://127.0.0.1:5000/):

```
@app.route('/')
def index():
    return render_template('lista.html', titulo='Jogos',
                           jogos=lista
```

Agora, iremos trabalhar com outro problema. Depois que adicionamos um novo jogo à lista por meio do formulário no navegador, se atualizarmos a página http://127.0.0.1:5000/criar (http://127.0.0.1:5000/criar), receberemos uma mensagem perguntando se realmente desejamos repetir o envio do formulário:

> Confirmar reenvio do formulário
>
> A página que você está procurando usou as informações inseridas. Voltar à essa página poderá fazer com que todas as ações realizadas antes sejam repetidas. Quer continuar?

E não é isso que gostaríamos de fazer, certo? Somente queremos ver a lista novamente. Porém, quando fizemos a alteração para renderizar o *template* de `lista.html`, nós mantivemos a mesma rota `/criar`, que ainda tem os dados do formulário anterior. Além disso, a rota `/criar` nem mesmo recebe o método GET, somente POST. Resolveremos esse problema na próxima aula. Até lá!