Tutorial Django Parte 7: Sessões

Esse tutorial estende nosso site LocalLibrary, adicionando um contador de visitas baseado em sessões à página inicial. Esse é um exemplo relativamente simples, mas, capaz de mostrar como você pode usar a estrutura de sessão do framework para providenciar um comportamento persistente para usuários anônimos em seu próprio site.

Pré-requisitos:	Completar todos os tópicos anteriores do tutorial, incluindo Django Tutorial Part 6: Generic list and detail views
Objetivo:	Entender como as sessões são usadas.

Visão Geral

O site LocalLibrary que criamos nos tutoriais anteriores permite que os usuarios busquem por livros e autores no catálogo. Enquanto o conteúdo é dinamicamente gerado a partir da base de dados, todos os usuários terão acessos às mesmas páginas e às mesmas informações quando acessarem o site.

Em uma biblioteca "real", você pode querer fornecer uma experiência personalizada para cada usuário, com base no uso anterior do site, nas preferências, etc. Por exemplo, você pode ocultar mensagens de aviso que o usuário reconheceu anteriormente na próxima visita deles ao site ou armazenar e respeitar suas preferências (por exemplo, o número de resultados de pesquisa que eles querem exibir em cada página).

A estrutura da sessão permite implementar esse tipo de comportamento, permitindo que você armazene e recupere dados arbitrários baseados em cada visitante do site.

O que são sessões?

Toda a comunicação entre os navegadores web e os servidores é feita via protocolo HTTP, qual é *stateless* (sem estados). O fato do protocolo ser stateless significa que as mensagenns entre o

cliente e o servidor são completamente independentes uma da outra — não há uma noção de "sequência" ou comportamento diferente baseado nas mensagens anteriores. Como resultado, se você quiser ter um site que monitore os relacionamentos contínuos com um cliente, é necessário implementá-lo por conta própria.

Sessões são o mecanismo usado pelo Django (e muitos outros na Internet) para monitorar o "estado" entre o site e um navegador web em particular. Sessões permitem que você armazene dados arbitrários por navegador web, e têm esse dado disponível no site sempre que o navegador conectar. Dados de itens individuais associados com a sessão são referenciados por uma "chave", que é usada para armazenar e recuperar os dados.

O Django usa um cookie contendo um *identificador* especial de sessão para identificar cada navegador e associar com o site. Os dados da sessão atual são armazenados na base de dados do site por padrão (é mais seguro do que armazenar os dados em cookie, onde é mais vulnerável aos usuários perigisos). Você pode configurar o Django para armazenar os dados da sessão em outros lugares (cache, arquivos, cookies "seguros"), mas o local padrão é uma opção boa e relativamente "segura".

Habilitando as Sessões

As sessões foram ativadas automaticamente quando criamos o esqueleto do site (no tutorial 2).

A configuração e feita nas seções `INSTALLED_APPS` e `MIDDLEWARE` do arquivo (**locallibrary/locallibrary/settings.py**), exibidas a seguir:

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    ....

MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',
    ....
```

Usando Sessões

Você pode acessar o atributo `session` na view a partir do parâmetro `request` (um `HttpRequest` passado como primeiro argumento na view). Esse atributo de sessão representa

a conexão atual específica com um usuário (ou, para ser mais preciso, a conexão com o navegador atual, conforme identificado pelo id da sessão no cookie do navegador para este site).

O atributo `session` é como um objeto dicionário que você pode ler e escrever quantas vezes você quiser na sua view, modificando-o como desejar. Você pode fazer todas as operações normais de um dicionário, incluindo limpar todos os dados, testar se uma chave está presente, loop em torno dos dados, etc. Na maior parte do tempo, você usará apenas a API padrão "dictionary" para obter e setar valores.

O fragmento de código abaixo mostra como você pode obter, setar e deletar qualquer dado com com a chave "`my_car`", associada com a sessão atual (navegador).

> **Nota**: Uma das coisas boas sobre o Django é que você não precisa pensar sobre os mecanismos que vinculam a sessão atual à requisição em sua view. Se nós usarmos os fragmentos abaixo em nossa view, saberemos que as informações sobre `my_car` estão associadas apenas com o navegador que enviou a requisição atual.

```
# Pega um valor de sessão baseado na sua chave (ex.:'my_car'), disparand
my_car = request.session['my_car']

# Pega o valor da sessão, seta o valor padrão ('mini') se a chave não es
my_car = request.session.get('my_car', 'mini')

# Seta o valor da sessão
request.session['my_car'] = 'mini'

# Deleta o valor da sessão
del request.session['my_car']
```

A API também oferece um número de outros métodos que são muito usados para gerenciar a os cookies da sessão associada. Por exemplo, há metodos para testar se cookies são suportados no navegador do cliente, para setar e checar a data de validade do cookie, e para limpar sessões expiradas do armazenamento de dados. Você pode encontrar sobre a API completa em How to use sessions (documentação do Django).

Salvando os dados da sessão

Por padrão, o Django só salva na base de dados da sessão e envia o cookie da sessão para o cliente quando a sessão é *modificada* (atribuida) ou *deletada*. Se você está atualizando alguns

dados utilizando sua chave de sessão, como mostrado na seção anterior, então você não precisa se preocupar com isso! Por exemplo:

```
# This is detected as an update to the session, so session data is saved
request.session['my_car'] = 'mini'
```

Se você está atualizando algumas informações *dentro* dos dados da sessão, então o Django não reconhecerá que você fez uma alteração nos dados da sessão e não salvará os dados (por exemplo, se você alterasse os dados de `"wheels"` dentro dos dados do seu `"my_car"`, como mostrado abaixo). Nesse caso você precisará marcar explicitamente a sessão como tendo sido modificada.

```
# Session object not directly modified, only data within the session. Se
request.session['my_car']['wheels'] = 'alloy'

# Set session as modified to force data updates/cookie to be saved.
request.session.modified = True
```

> **Nota**: Você pode mudar o comportamento do site para atualizar a base de dados/enviar cookie em qualquer requisição adicionando `SESSION_SAVE_EVERY_REQUEST = True` nas configurações (**locallibrary/locallibrary/settings.py**) do seu projeto.

Exemplo simples - obtendo a contagem de visitas

Como um exemplo simples do mundo real, atualizaremos nossa biblioteca para informar ao usuário atual quantas vezes ele visitou o site *LocalLibrary*.

Abra **/locallibrary/catalog/views.py**, e faça as alterações mostradas em negrito abaixo.

```
def index(request):
    ...
```

```
    num_authors = Author.objects.count()  # The 'all()' is implied by de

    # Number of visits to this view, as counted in the session variable.
    num_visits = request.session.get('num_visits', 0)
    request.session['num_visits'] = num_visits + 1

    context = {
        'num_books': num_books,
        'num_instances': num_instances,
        'num_instances_available': num_instances_available,
        'num_authors': num_authors,
        'num_visits': num_visits,
    }

    # Render the HTML template index.html with the data in the context v
    return render(request, 'index.html', context=context)
```

Aqui primeiro obtemos o valor da *session key* `'num_visits'`, setando o valor para 0 se não tiver sido definido anteiormente. Cada vez que uma requisição é recebida, nós então incrementamos o valor e armazenamos novamente na sessão (para a próxima vez que o usuário visitar a página). A variável `num_visits` é então passada para o *template* na nossa varável *context*.

> **Nota**: Também podemos testar se os cookies são suportados no navegador (veja Como usar sessões para exemplos) ou projetar nossa UI (interface do usuário) para que não se importe se os *cookies* são ou não suportados.

Adicione a linha vista na parte inferior do bloco a seguir ao seu *template* HTML principal (**/locallibrary/catalog/templates/index.html**) na parte inferior da sessão *"Dynamic content"*, para exibir a variável *context*:

```
<h2>Dynamic content</h2>

<p>The library has the following record counts:</p>
<ul>
  <li><strong>Books:</strong> {{ num_books }}</li>
  <li><strong>Copies:</strong> {{ num_instances }}</li>
  <li><strong>Copies available:</strong> {{ num_instances_available }}</
  <li><strong>Authors:</strong> {{ num_authors }}</li>
```

```
</ul>

<p>You have visited this page {{ num_visits }}{% if num_visits == 1 %} t
```

Salve suas alterações e reinicie o servidor de teste. Sempre que você atualiza a página, o número deve ser atualizado.

Resumo

Agora você sabe como é fácil utilizar sessões para melhorar sua interação com usuários anônimos.

Em nosso próximo artigo nós iremos explicar a estrutura de autenticação e autorização (permissão), e mostrar como oferecer suporte a contas de usuário.

Veja também

- Como usar sessões (Django docs)

Neste módulo

- Introdução ao Django
- Configurando um ambiente de desenvolvimento Django
- Tutorial Django: Website de uma Biblioteca Local
- Tutorial Django Parte 2: Criando a base do website
- Tutorial Django Parte 3: Usando *models*
- Tutorial Django Parte 4: Django admin site
- Tutorial Django Parte 5: Criando nossa página principal
- Tutorial Django Parte 6: Lista genérica e *detail views*
- Tutorial Django Parte 7: Framework de Sessões
- Tutorial Django Parte 8: Autenticação de Usuário e permissões
- Tutorial Django Parte 9: Trabalhando com formulários
- Tutorial Django Parte 10: Testando uma aplicação web Django
- Tutorial Django Parte 11: Implantando Django em produção
- Segurança de aplicações web Django
- DIY Django mini blog

Last modified: 16 de jul. de 2020, by MDN contributors

Change your language

Português (do Brasil) Change language