Django Tutorial Part 7: Sessions framework

Overview: Django

This tutorial extends our LocalLibrary website, adding a session-based visit-counter to the home page. This is a relatively simple example, but it does show how you can use the session framework to provide persistent behaviour for anonymous users in your own sites.

Prerequisites:	Complete all previous tutorial topics, including Django Tutorial Part 6: Generic list and detail views
Objective:	To understand how sessions are used.

Overview

The LocalLibrary website we created in the previous tutorials allows users to browse books and authors in the catalog. While the content is dynamically generated from the database, every user will essentially have access to the same pages and types of information when they use the site.

In a "real" library you may wish to provide individual users with a customized experience, based on their previous use of the site, preferences, etc. For example, you could hide warning messages that the user has previously acknowledged next time they visit the site, or store and respect their preferences (e.g. the number of search results that they want to be displayed on each page).

The session framework lets you implement this sort of behaviour, allowing you to store and retrieve arbitrary data on a per-site-visitor basis.

What are sessions?

All communication between web browsers and servers is via HTTP, which is *stateless*. The fact that the protocol is stateless means that messages between the client and server are completely independent of each other— there is no notion of "sequence" or behaviour based on previous messages. As a result, if you want to have a site that keeps track of the ongoing relationships with a client, you need to implement that yourself.

Sessions are the mechanism used by Django (and most of the Internet) for keeping track of the "state" between the site and a particular browser. Sessions allow you to store arbitrary data per browser, and have this data available to the site whenever the browser connects. Individual data items associated with the session are then referenced by a "key", which is used both to store and retrieve the data.

Django uses a cookie containing a special *session id* to identify each browser and its associated session with the site. The actual session *data* is stored in the site database by default (this is more secure than storing the data in a cookie, where they are more vulnerable to malicious users). You can configure Django to store the session data in other places (cache, files, "secure" cookies), but the default location is a good and relatively secure option.

Enabling sessions

Sessions were enabled automatically when we created the skeleton website (in tutorial 2).

The configuration is set up in the `INSTALLED_APPS` and `MIDDLEWARE` sections of the project file (**locallibrary/locallibrary/settings.py**), as shown below:

```
INSTALLED_APPS = [
    ...
```

```
    'django.contrib.sessions',
    ....

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

Using sessions

You can access the `session` attribute in the view from the `request` parameter (an `HttpRequest` passed in as the first argument to the view). This session attribute represents the specific connection to the current user (or to be more precise, the connection to the current *browser*, as identified by the session id in the browser's cookie for this site).

The `session` attribute is a dictionary-like object that you can read and write as many times as you like in your view, modifying it as wished. You can do all the normal dictionary operations, including clearing all data, testing if a key is present, looping through data, etc. Most of the time though, you'll just use the standard "dictionary" API to get and set values.

The code fragments below show how you can get, set, and delete some data with the key "`my_car`", associated with the current session (browser).

> **Note**: One of the great things about Django is that you don't need to think about the mechanisms that tie the session to your current request in your view. If we were to use the fragments below in our view, we'd know that the information about `my_car` is associated only with the browser that sent the current request.

```
    # Get a session value by its key (e.g. 'my_car'), raising a KeyError if t
    my_car = request.session['my_car']

    # Get a session value, setting a default if it is not present ('mini')
    my_car = request.session.get('my_car', 'mini')

    # Set a session value
```

```
request.session['my_car'] = 'mini'

# Delete a session value
del request.session['my_car']
```

The API also offers a number of other methods that are mostly used to manage the associated session cookie. For example, there are methods to test that cookies are supported in the client browser, to set and check cookie expiry dates, and to clear expired sessions from the data store. You can find out about the full API in How to use sessions (Django docs).

Saving session data

By default, Django only saves to the session database and sends the session cookie to the client when the session has been *modified* (assigned) or *deleted*. If you're updating some data using its session key as shown in the previous section, then you don't need to worry about this! For example:

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

If you're updating some information *within* session data, then Django will not recognise that you've made a change to the session and save the data (for example, if you were to change "`wheels`" data inside your "`my_car`" data, as shown below). In this case you will need to explicitly mark the session as having been modified.

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

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

> **Note**: You can change the behavior so the site will update the database/send cookie on every request by adding `SESSION_SAVE_EVERY_REQUEST = True` into your project settings (**locallibrary/locallibrary/settings.py**).

Simple example — getting visit counts

As a simple real-world example we'll update our library to tell the current user how many times they have visited the *LocalLibrary* home page.

Open **/locallibrary/catalog/views.py**, and make the changes shown in bold below.

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

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

    # 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 variabl
    return render(request, 'index.html', context=context)
```

Here we first get the value of the `'num_visits'` session key, setting the value to 0 if it has not previously been set. Each time a request is received, we then increment the value and store it

back in the session (for the next time the user visits the page). The `num_visits` variable is then passed to the template in our context variable.

> **Note**: We might also test whether cookies are even supported in the browser here (see How to use sessions for examples) or design our UI so that it doesn't matter whether or not cookies are supported.

Add the line seen at the bottom of the following block to your main HTML template (**/locallibrary/catalog/templates/index.html**) at the bottom of the "Dynamic content" section to display the context variable:

```
<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>
  <li><strong>Authors:</strong> {{ num_authors }}</li>
</ul>

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

Save your changes and restart the test server. Every time you refresh the page, the number should update.

Summary

You now know how easy it is to use sessions to improve your interaction with *anonymous* users.

In our next articles we'll explain the authentication and authorization (permission) framework, and show you how to support user accounts.

See also

- How to use sessions (Django docs)

Overview: Django

In this module

Last modified: Nov 12, 2020, by MDN contributors

Related Topics

Complete beginners start here!

- Getting started with the Web

HTML — Structuring the Web

- Introduction to HTML
- Multimedia and embedding
- HTML tables

CSS — Styling the Web

- CSS first steps
- CSS building blocks
- Styling text
- CSS layout

JavaScript — Dynamic client-side scripting

- JavaScript first steps
- JavaScript building blocks
- Introducing JavaScript objects
- Asynchronous JavaScript
- Client-side web APIs

Web forms — Working with user data

- Core forms learning pathway
- Advanced forms articles

Further resources