Django Tutorial Part 7: Sessions framework

Created by 이 윤 준 (yoonjoon.lee@gmail.com)

June, 2019

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

Overview

The LocalLibrary website we created 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.

We may wish to provide individual users with a customized experience, based on their previous use of the site, preferences, etc. For example, we 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 they want displayed on each page).

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

What are sessions?

All communication between web browsers and servers is via the HTTP protocol, 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 for keeping track of the "state" between the site and a particular browser. Sessions allow us 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. We can configure Django to store the session data in other places, but the default location is a good and relatively secure option.

Enabling sessions

Sessions were enabled automatically when we created the skeleton website.

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

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

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

Using sessions

We can access the `session` attribute in the view from the `request` parameter. This session attribute represents the specific connection to the current user (or to be more precise, the connection to the current browser).

The session attribute is a dictionary-like object that we can read and write as many times as we like in our view, modifying it as wished. We 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, we'll just use the standard "dictionary" API to get and set values.

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

```
# Get a session value by its key (e.g. 'my_car'),
# raising a KeyError if the key is not present
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.

Saving session data

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 we're updating some data using its session key, then we 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 we're updating some information within session data, then Django will not recognise that we'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). In this case we will need to explicitly mark the session as having been modified.

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

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

Simple example — getting visit counts

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.

```
def index(request):
    ...
    # The 'all()' is implied by default.
    num_authors = Author.objects.count()

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

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. The `num_visits` variable is then passed to the template in our context variable.

Add the line seen at the bottom of the 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_
  <li><strong>Authors:</strong> {{ num_authors }}</li>
</ul>

<p>You have visited this page {
  { num_visits }}
  {% if num_visits == 1 %} time
  {% else %} times
  {% endif %}.
</p>
```

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