Django Sessions

COMP 8347
Slides prepared by Dr. Arunita Jaekel arunita@uwindsor.ca

Django Sessions

- Topics
 - Sessions Introduction
 - Sessions in Views
 - Session Objects
 - Setting Cookies
 - Saving Sessions
 - Additional Operations

Enabling Sessions

- Edit the MIDDLEWARE_CLASSES setting
 - It should contain'django.contrib.sessions.middleware.SessionMiddleware'.
 - The default settings.py created by djangoadmin.py startproject has SessionMiddleware activated.
- Database Backed Sessions: by default Django stores session data in the database.
 - Add 'django.contrib.sessions' to INSTALLED_APPS.
 - Django creates a single database table that stores session data.

Alternative Configurations

- Use SESSION_ENGINE setting for alternative configurations:
 - Using cached sessions
 - Store session data using Django's cache system
 - Using file-based sessions.
 - Store session data using the computers file system.
 - Web server should have permissions to read and write to this location
 - Using cookie-based sessions.
 - Store session data using Django's tools for cryptographic signing and the SECRET_KEY setting.

Session Object

- Session Object: A dictionary-like object, which is an attribute of a HttpRequest object.
 - When SessionMiddleware is activated,
 each HttpRequest object has a session attribute.
 - HttpRequest is the first argument to any Django view function.
 - By default, Django only saves to the session database when the session has been modified
 - if any of its dictionary values have been assigned or deleted
- You can read and write to request.session at any point in your view.
 - You can edit it multiple times.

Session Objects

- Session objects use standard dict methods.
 - Can use usual dictionary access methods
 - Example:
 - fav_color = request.session['fav_color'] # __getitem__
 - request.session['fav_color'] = 'blue' # __setitem__
 - del request.session['fav_color'] # ___delitem___
 - 'fav_color' in request.session # __contains__
 - Additional dict methods that can be used:
 - keys(), items(), setdefault(), clear()

More Methods

- flush():
 - Delete the current session data from the session and regenerate the session key value that is sent back to the user in the cookie.
 - Used to ensure that the previous session data can't be accessed again from the user's browser
- set_test_cookie():
 - Sets a test cookie to determine whether the user's browser supports cookies.
- test_cookie_worked():
 - Returns either True or False, depending on whether the user's browser accepted the test cookie.
- delete_test_cookie():
 - Deletes the test cookie. Use this to clean up after yourself.



More Methods

- set_expiry(value):
 - -Sets the expiration time for the session.
 - value is integer: session expires after that many seconds of inactivity.
 - value is datetime object: the session expires at specified date/time.
 - value is 0: session cookie expires when the Web browser is closed.
 - value is None: session uses the global session expiry policy.
- get_expiry_age():
 - Returns the number of seconds until this session expires.
- clear_expired():
 - Removes expired sessions from the session store.
- cycle_key():
 - Creates a new session key while retaining current session data.



Example

```
# This simplistic view sets a has_commented
# variable to True after a user posts a comment. It
# doesn't let a user post a comment more than once.
def post_comment(request, new_comment):
 if request.session.get('has_commented', False):
   return HttpResponse("You've already commented.")
 c = comments.Comment(comment=new_comment)
 c.save()
  request.session['has_commented'] = True
 return HttpResponse('Thanks for your comment!')
```

Testing Cookies

- Call set_test_cookie()
 method
 of request.session in a
 view.
- 2. Call test_cookie_worked() in a <u>subsequent</u> view NOT in the same view.
 - you can't actually tell whether a browser accepted it until the browser's <u>next</u> request.
- 3. It's good practice to use delete_test_cookie() to clean up afterwards.
 - Do this after you've verified that the test cookie worked.

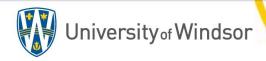
```
def login(request):
    if request.method == 'POST':
         if request.session.test_cookie_worked():
               request.session.delete_test_cookie()
               return HttpResponse("You're logged
               in.")
         else:
               return HttpResponse("Please
                                 enable cookies
                                 and try again.")
    request.session.set_test_cookie()
    return render('foo/login_form.html')
```

Sessions Outside of Views

- An API is available to manipulate session data outside of a view.
 - The SessionStore object can be imported directly from the appropriate backend.
 - For django.contrib.sessions.models each session is a normal Django model.
 - Can be accessed using normal Django db API.
- >>> from django.contrib.sessions.models import Session
- >>> s = Session.objects.get(pk='2b1189a188b44ad18c35e113ac6ceead')
- >>> s.expire_date
- datetime.datetime(2015, 8, 20, 13, 35, 12)

Session Expiration

- SESSION_EXPIRE_AT_BROWSER_CLOSE:
 - Controls if session framework uses browser-length sessions or persistent sessions.
 - Global default setting for session framework
 - get_expire_at_browser_close(): True if session cookie expires when user's browser is closed.
 - Can be overwritten at a per-session level by explicitly calling the set_expiry() method of request.session.
 - By default, it is set to False
 - This means session cookies will be stored in users' browsers for as long as SESSION_COOKIE_AGE.
 - Use this if you don't want people to have to log in every time they open a browser
 - If it is set to True
 - Cookies expire as soon as user closes their browser.
 - Use this if you want people to have to <u>log in every time</u> they open a browser



Clearing SessionStore

- When a user logs in, Django adds a row to the django_session db table.
 - Django updates this row each time the session data changes.
 - If the user logs out manually, Django deletes the row.
 - If the user does not log out, the row never gets deleted.
- As users create new sessions on your website, session data can accumulate in your session store.
 - If you're using the database backend, the django_session db table will grow.
 - If you're using the file backend, your temporary directory will contain an increasing no. of files.
- Django does not provide automatic purging of expired sessions.
 - It is your job to purge expired sessions on a regular basis.
 - Django provides a clean-up management command for this purpose: clearsessions.
 - It is recommended to call this command on a regular basis.