

Session and Cookies

Chapter 9

1

Objectives

- What are Cookies?
- Shortcomings of using Cookies
- Django Cookies and Cookies Handling
- Using Django Sessions

2



What are Cookies?

- *Cookies, technically called HTTP Cookies are small text files which are created and maintained by your browser on the particular request of Web-Server.*
- They are stored locally by your browser.
- HTTP is a stateless protocol. When any request is sent to the server, the server has no idea if the user is new or has visited the site before.
- Suppose, you are logging in any website, that website will respond the browser with some cookies which will have some unique identification of user generated by the server and some more details according to the context of the website.
- Cookies made these implementations possible with ease which were previously not possible over HTTP implementation.
- Google AdSense and **Google Analytics** can also track you using the cookies they generate. Different websites use cookies differently according to their needs.

3

How Cookies work?

- Whenever a request to a website is made, the webserver returns the content of the requested page.
- In addition, one or more cookies may also be sent to the client, which are in turn stored in a persistent browser cache.
- When a user requests a new page from the same web server, any cookies that are matched to that server are sent with the request.
- The server can then interpret the cookies as part of the request's context and generate a response to suit.
- This process is repeated every time a new request is made by the browser.
- The browser repeats the process until the cookie expires or the session is closed and the cookie is deleted by the browser itself.

4

Cookies, Cookies Everywhere!

- As an example, you may login to a site with a particular username and password.
- When you have been authenticated, a cookie may be returned to your browser containing your username, indicating that you are now logged into the site.
- At every request, this information is passed back to the server where your login information is used to render the appropriate page - perhaps including your username in particular places on the page.
- Your session cannot last forever, however - cookies have to expire at some point in time - they cannot be of infinite length.
- A web application containing sensitive information may expire after only a few minutes of inactivity.
- A different web application with trivial information may expire half an hour after the last interaction - or even weeks into the future.

5

Cookies: potential security holes

- The passing of information in the form of cookies can open up potential security holes in your web application's design.
- Passing a user's credit card number on an e-commerce site as a cookie for example would be highly unwise.
- The cookie could be taken by a malicious program. From there, hackers would have his or her credit card number - all because your web application's design is fundamentally flawed.

6

Potentially sensitive nature of cookies

- Because of the potentially sensitive nature of cookies, lawmakers have taken a particularly keen interest in them.
- In particular, EU lawmakers in 2011 introduced an EU-wide 'cookie law', where all hosted sites within the EU should present a cookie warning message when a user visits the site for the first time.



7

Basic Considerations of using Cookies

When using cookies within your Django application, there's a few things you should consider:

- First, consider what type of cookies your web application requires. Does the information you wish to store need to persist over a series of user browser sessions, or can it be safely disregarded upon the end of one session?
- Think carefully about the information stored using cookies. Remember, storing information in cookies means that the information will be stored on client's computers, too. This is a potentially huge security risk: you simply don't know how compromised a user's computer will be. Consider server-side alternatives if potentially sensitive information is involved.
- As a follow-up to the previous bullet point, remember that users may set their browser's security settings to a high level which could potentially block your cookies. As your cookies could be blocked, your site may function incorrectly. You must cater for this scenario - you have no control over the client browser's setup.

8

Workflow for client-side cookies

If client-side cookies are the right approach for you, then work through the following steps:

1. You can retrieve the value of a cookie with `request.COOKIES['cookie_name']`.
 - cookies are returned as strings, regardless of what they contain. You might need to cast to the correct type.
 - However, if the cookie doesn't exist, it will throw an error.
 - To test if a cookie 'last_visit' exists: `if 'last_visit' in request.COOKIES`
2. An alternative is to use `request.COOKIES.get('key', 'default')` which returns the value if the key exists, otherwise 'default' - you can put anything you like in place of 'default'.
E.g. `visits = int(request.COOKIES.get('visits', '1'))`
3. To update the cookie, pass the value you wish to save to the response you generate.
`response.set_cookie('cookie_name', value)` is the function you call, where two parameters are supplied: the name of the cookie, and the value you wish to set it to.
4. To delete a cookie, use `response.delete_cookie('cookie_name')`

9

Testing Cookie Functionality

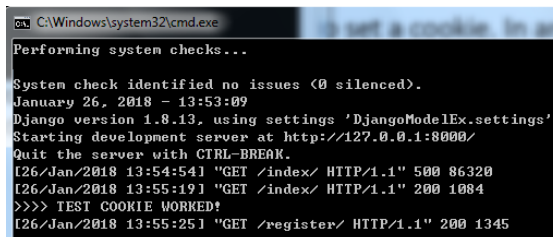
- To test out if client browser accepts cookies, make use of `set_test_cookie()`, `test_cookie_worked()` and `delete_test_cookie()`.
- In one view, you will need to set a cookie with `set_test_cookie()`.
- In another, you'll need to test that the cookie exists with `test_cookie_worked()`.
- Two different views are required for testing cookies because you need to wait to see if the client has accepted the cookie from the server.

10

Testing Cookie Functionality (cont'd)

- In index() view function, `request.session.set_test_cookie()`
- In register() view function,

```
if request.session.test_cookie_worked():
    print(">>> TEST COOKIE WORKED!")
    request.session.delete_test_cookie()
```
- With these in place, navigate to homepage. Once the homepage is loaded, navigate to the registration page.
- When the registration page is loaded, you should see **>>> TEST COOKIE WORKED!** appear in your Django development server's console.



```
C:\Windows\system32\cmd.exe
Performing system checks...

System check identified no issues (0 silenced).
January 26, 2018 - 13:53:09
Django version 1.8.13, using settings 'DjangoModelEx.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[26/Jan/2018 13:54:54] "GET /index/ HTTP/1.1" 500 86320
[26/Jan/2018 13:55:19] "GET /index/ HTTP/1.1" 200 1084
>>> TEST COOKIE WORKED!
[26/Jan/2018 13:55:25] "GET /register/ HTTP/1.1" 200 1345
```

11

Sessions

- HTTP is a stateless protocol, where every request made is always new to the server.
- The request on the server is always treated as if the user is visiting the site for the first time.
- This poses some problems like you can't implement user login and authentication features. These problems were actually solved by Cookies.
- However, since cookies are stored locally, and those cookies which are not sent over HTTPS can be easily caught by attackers. Therefore, it can be dangerous for both the site and the user to store essential data in cookies and returning the same again and again in plain text.

12

Using Django sessions

- Django provides a session framework to store arbitrary data for each visitor.
- Session data is stored on the server side, and cookies contain the session ID unless you use the cookie-based session engine.
- The default session engine stores session data in the database, but you can choose between different session engines. When we migrate the application, we can see the `django_session` table in the database.
- The session middleware manages the sending and receiving of cookies.
- To use sessions, you have to make sure that the MIDDLEWARE setting of your project contains `'django.contrib.sessions.middleware.SessionMiddleware'`, which is added by default to the MIDDLEWARE setting when you create a new project using the `startproject` command.

13

Middleware

- Middleware is a framework of hooks into Django's request/response processing.



/home/COMP222/django_projects/blog_project/blog_project/settings.py

```

43
44 MIDDLEWARE = [
45     'django.middleware.security.SecurityMiddleware',
46     'django.contrib.sessions.middleware.SessionMiddleware',
47     'django.middleware.common.CommonMiddleware',
48     'django.middleware.csrf.CsrfViewMiddleware',
49     'django.contrib.auth.middleware.AuthenticationMiddleware',
50     'django.contrib.messages.middleware.MessageMiddleware',
51     'django.middleware.clickjacking.XFrameOptionsMiddleware',
52 ]

```

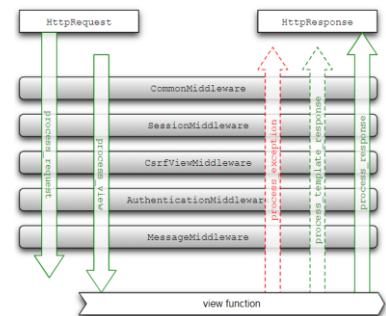
Here is the built-in Middleware classes the `django-admin startproject` command sets up

- Each middleware component is responsible for doing some specific function.
 - For example, the middleware component, `AuthenticationMiddleware`, associates users with requests using sessions. It adds the user attribute, representing the currently-logged-in user, to every incoming `HttpRequest` object.
 - **CSRF protection middleware** adds protection against Cross Site Request Forgeries by adding hidden form fields to POST forms and checking requests for the correct value.
- Django's built-in middleware reference – <https://docs.djangoproject.com/en/3.1/ref/middleware/>

14

Middleware: How it works

- The Middleware classes are called twice during the request/response life cycle.
- During the **request** cycle, the Middleware classes are executed top-down, meaning it will first execute SecurityMiddleware, then SessionMiddleware all the way until XFrameOptionsMiddleware. For each of the Middlewares it will execute the **process_request()** and **process_view()** methods.
- At this point, Django will do all the work on your view function. After the work is done (e.g. querying the database, paginating results, processing information, etc), it will return a **response** for the client.
- During the **response** cycle, the Middleware classes are executed bottom-up, meaning it will first execute XFrameOptionsMiddleware, then MessageMiddleware all the way until SecurityMiddleware. For each of the Middlewares, it will execute the **process_exception()**, **process_template_response()** and **process_response()** methods.

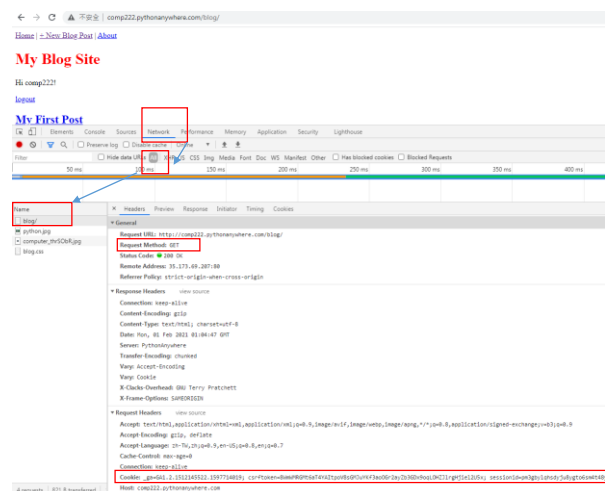


<https://simpleisbetterthancx.com/tutorial/2016/07/18/how-to-create-a-custom-django-middleware.html>

15

Viewing HTTP headers

- In Chrome, right click to select "Inspect" from the menu to bring up the Chrome developer tools.
- Choose the "Network" tab.
- Initially, it is possible the page data is not present/up to date. Hit Ctrl+R to record the reload.
- Observe the page information appears in the listing. Make sure "All" is selected next to the "Hide data URLs" checkbox.



16

Session framework

- The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis.
- It stores data on the server side and abstracts the sending and receiving of cookies.
- Cookies contain a session ID – not the data itself (unless you're using the cookie based backend).
- To enable session functionality, do the following:
 - Edit the MIDDLEWARE setting and make sure it contains 'django.contrib.sessions.middleware.SessionMiddleware'. The default settings.py created by django-admin startproject has SessionMiddleware activated.
- If you don't want to use sessions, you might as well remove the SessionMiddleware line from MIDDLEWARE and 'django.contrib.sessions' from your INSTALLED_APPS. It'll save you a small bit of overhead.

17

Configuring the session engine

- By default, Django stores sessions in your database (using the model `django.contrib.sessions.models.Session`).
- Though this is convenient, in some setups it's faster to store session data elsewhere, so Django can be configured to store session data on your filesystem or in your cache.
- Django offers the following options for storing session data:
 - **Database sessions:** Session data is stored in the database. This is the default session engine.
 - **File-based sessions:** Session data is stored in the filesystem.
 - **Cached sessions:** Session data is stored in a cache backend. You can specify cache backends using the CACHES setting. Storing session data in a cache system provides the best performance.
 - **Cached database sessions:** Session data is stored in a write-through cache and database. Reads-only use the database if the data is not already in the cache.
 - **Cookie-based sessions:** Session data is stored in the cookies that are sent to the browser.

18

Accessing the current session

- You can access the current session using `request.session`, treating it like a Python dictionary to store and retrieve session data.
- You can set a variable in the session like this:
`request.session['foo'] = 'bar'`
- Retrieve a session key as follows: `request.session.get('foo')`
- Delete a key you previously stored in the session as follows:
`del request.session['foo']`

19

Important session-related settings

You can customize sessions with specific settings. Here are some of the important session-related settings:

- **SESSION_COOKIE_AGE**: The duration of session cookies in seconds. The default value is 1209600 (two weeks).
- **SESSION_COOKIE_DOMAIN**: The domain used for session cookies. Set this to `mydomain.com` to enable cross-domain cookies or use `None` for a standard domain cookie.
- **SESSION_COOKIE_SECURE**: A boolean indicating that the cookie should only be sent if the connection is an HTTPS connection.
- **SESSION_EXPIRE_AT_BROWSER_CLOSE**: A boolean indicating that the session has to expire when the browser is closed.
- **SESSION_SAVE_EVERY_REQUEST**: A boolean that, if `True`, will save the session to the database on every request. The session expiration is also updated each time it's saved.
- You can see all the session settings and their default values at <https://docs.djangoproject.com/en/3.0/ref/settings/#sessions>.

20

Session expiration

- You can choose to use browser-length sessions or persistent sessions using the `SESSION_EXPIRE_AT_BROWSER_CLOSE` setting.
- This is set to False by default, forcing the session duration to the value stored in the `SESSION_COOKIE_AGE` setting.
- If you set `SESSION_EXPIRE_AT_BROWSER_CLOSE` to True, the session will expire when the user closes the browser, and the `SESSION_COOKIE_AGE` setting will not have any effect.
- You can use the `set_expiry()` method of `request.session` to overwrite the duration of the current session.

21

Secure Cookies

- An HTTP Cookie is used to store information on a client's computer such as authentication credentials. This is necessary because the HTTP protocol is stateless by design: there's no way to tell if a user is authenticated other than including an identifier in the HTTP Header!
- Django uses sessions and cookies for this, as do most websites. But cookies can and should be forced over HTTPS as well via the `SESSION_COOKIE_SECURE` config.
- It defaults to False so we must set it to True in production. We can also do the same for CSRF cookies using `CSRF_COOKIE_SECURE`.

22

Admin Hardening

- So far it may seem as though the advice is to rely on Django defaults, use HTTPS, add `csrf_token` tags on forms, and set a permissions structure.
- All true. But one step Django does not take on our behalf is hardening the Django admin.
- Consider that every Django website sets the admin, by default, to the `/admin` URL. This is a prime suspect for any hacker trying to access a Django site.
- Therefore an easy step is to simply change the admin URL to literally anything else! E.g. `path('anything-but-admin/', admin.site.urls)`,
- A fun 3rd party package `django-admin-honeypot` will generate a fake admin log in screen and email site admins the IP address of anyone trying to attack your site at `/admin`. These IP addresses can then be added to a blocked address list for the site.
- It's also possible via `django-two-factor-auth` to add two-factor authentication to your admin for an even further layer of protection.

23