Session

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.

By default, Django stores sessions in your database.

As it stores sessions in database so it is mandatory to run makemigrations and migrate to use session. It will create required tables.

The Django sessions framework is entirely, and solely, cookie-based.

django.contrib.sessions.middleware.SessionMiddleware

django.contrib.sessions

Session

database-backed sessions - If you want to use a database-backed session, you need to add 'django.contrib.sessions' to your INSTALLED_APPS setting.

Once you have configured your installation, run manage.py migrate to install the single database table that stores session data.

file-based sessions - To use file-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.file".

You might also want to set the SESSION_FILE_PATH setting (which defaults to output from tempfile.gettempdir(), most likely /tmp) to control where Django stores session files. Be sure to check that your Web server has permissions to read and write to this location.

cookie-based sessions - To use cookies-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.signed_cookies". The session data will be stored using Django’s tools for cryptographic signing and the SECRET_KEY setting.

Session

cached sessions - For better performance, you may want to use a cache-based session backend. To store session data using Django’s cache system, you’ll first need to make sure you’ve configured your cache.

Using sessions in views

When SessionMiddleware is activated, each HttpRequest object, the first argument to any Django view function will have a session attribute, which is a dictionary-like object.

You can read it and write to request.session at any point in your view. You can edit it multiple times.

Set Item

request.session['key'] = 'value’

Get Item

returned_value = request.session['key']

returned_value = request.session.get('key', default=None)

Using sessions in views

Delete Item

del request.session['key’]

This raises KeyError if the given key isn’t already in the session.

Contains

'key' in request.session

Session Methods

keys() method returns a view object that displays a list of all the keys in the dictionary.

Syntax:- dict.keys()

items() method returns the list with all dictionary keys with values.

Syntax:- dict.items()

clear() function is used to erase all the elements of list. After this operation, list becomes empty.

Syntax:- dict.clear()

setdefault() method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary.

Syntax: dict.setdefault(key, default_value)

Session Methods

flush() – It deletes the current session data from the session and deletes the session cookie. This is used if you want to ensure that the previous session data can’t be accessed again from the user’s browser (for example, the django.contrib.auth.logout() function calls it).

Session Methods

get_session_cookie_age() – It returns the age of session cookies, in seconds. Defaults to SESSION_COOKIE_AGE.

set_expiry(value) – It sets the expiration time for the session. You can pass a number of different values:

If value is an integer, the session will expire after that many seconds of inactivity. For example, calling request.session.set_expiry(300) would make the session expire in 5 minutes.

If value is a datetime or timedelta object, the session will expire at that specific date/time. Note that datetime and timedelta values are only serializable if you are using the PickleSerializer.

If value is 0, the user's session cookie will expire when the user's Web browser is closed.

If value is None, the session reverts to using the global session expiry policy.

Reading a session is not considered activity for expiration purposes. Session expiration is computed from the last time the session was modified.

Session Methods

get_expiry_age() – It returns the number of seconds until this session expires. For sessions with no custom expiration (or those set to expire at browser close), this will equal SESSION_COOKIE_AGE.

This function accepts two optional keyword arguments:

modification: last modification of the session, as a datetime object. Defaults to the current time.

expiry: expiry information for the session, as a datetime object, an int (in seconds), or None. Defaults to the value stored in the session by set_expiry(), if there is one, or None.

get_expiry_date() – It returns the date this session will expire. For sessions with no custom expiration (or those set to expire at browser close), this will equal the date SESSION_COOKIE_AGE seconds from now.

This function accepts the same keyword arguments as get_expiry_age().

Session Methods

get_expire_at_browser_close() – It returns either True or False, depending on whether the user's session cookie will expire when the user's Web browser is closed.

clear_expired() – It removes expired sessions from the session store. This class method is called by clearsessions.

cycle_key() – It creates a new session key while retaining the current session data. django.contrib.auth.login() calls this method to mitigate against session fixation.

Session Methods

set_test_cookie() – It sets a test cookie to determine whether the user's browser supports cookies. Due to the way cookies work, you won't be able to test this until the user's next page request.

test_cookie_worked() – It returns either True or False, depending on whether the user's browser accepted the test cookie. Due to the way cookies work, you'll have to call set_test_cookie() on a previous, separate page request.

delete_test_cookie() – It deletes the test cookie. Use this to clean up after yourself.

Session Settings

SESSION_CACHE_ALIAS - If you're using cache-based session storage, this selects the cache to use. Default: 'default'

SESSION_COOKIE_AGE - The age of session cookies, in seconds. Default: 1209600 (2 weeks, in seconds)

SESSION_COOKIE_DOMAIN - The domain to use for session cookies. Set this to a string such as "example.com" for cross-domain cookies, or use None for a standard domain cookie.

Be cautious when updating this setting on a production site. If you update this setting to enable cross-domain cookies on a site that previously used standard domain cookies, existing user cookies will be set to the old domain. This may result in them being unable to log in as long as these cookies persist. Default: None

Session Settings

SESSION_COOKIE_HTTPONLY - Whether to use HttpOnly flag on the session cookie. If this is set to True, client-side JavaScript will not be able to access the session cookie.

HttpOnly is a flag included in a Set-Cookie HTTP response header. It's part of the RFC 6265 standard for cookies and can be a useful way to mitigate the risk of a client-side script accessing the protected cookie data.

This makes it less trivial for an attacker to escalate a cross-site scripting vulnerability into full hijacking of a user's session. There aren't many good reasons for turning this off. Your code shouldn't read session cookies from JavaScript. Default: True

SESSION_COOKIE_NAME - The name of the cookie to use for sessions. This can be whatever you want (as long as it's different from the other cookie names in your application). Default: 'sessionid'

Session Settings

SESSION_COOKIE_PATH - The path set on the session cookie. This should either match the URL path of your Django installation or be parent of that path.

This is useful if you have multiple Django instances running under the same hostname. They can use different cookie paths, and each instance will only see its own session cookie. Default: '/'

Session Settings

SESSION_COOKIE_SAMESITE - The value of the SameSite flag on the session cookie. This flag prevents the cookie from being sent in cross-site requests thus preventing CSRF attacks and making some methods of stealing session cookie impossible.

Possible values for the setting are:

'Strict': prevents the cookie from being sent by the browser to the target site in all cross-site browsing context, even when following a regular link.

For example, for a GitHub-like website this would mean that if a logged-in user follows a link to a private GitHub project posted on a corporate discussion forum or email, GitHub will not receive the session cookie and the user won't be able to access the project. A bank website, however, most likely doesn't want to allow any transactional pages to be linked from external sites so the 'Strict' flag would be appropriate.

'Lax' (default): provides a balance between security and usability for websites that want to maintain user's logged-in session after the user arrives from an external link.

In the GitHub scenario, the session cookie would be allowed when following a regular link from an external website and be blocked in CSRF-prone request methods (e.g. POST).

'None' (string): the session cookie will be sent with all same-site and cross-site requests.

False: disables the flag.

Session Settings

SESSION_COOKIE_SECURE - Whether to use a secure cookie for the session cookie. If this is set to True, the cookie will be marked as “secure”, which means browsers may ensure that the cookie is only sent under an HTTPS connection.

Leaving this setting off isn’t a good idea because an attacker could capture an unencrypted session cookie with a packet sniffer and use the cookie to hijack the user’s session. Default: False

SESSION_ENGINE – Controls where Django stores session data. Included engines are:

'django.contrib.sessions.backends.db'

'django.contrib.sessions.backends.file'

'django.contrib.sessions.backends.cache'

'django.contrib.sessions.backends.cached_db'

'django.contrib.sessions.backends.signed_cookies'

Default: 'django.contrib.sessions.backends.db'

Session Settings

SESSION_EXPIRE_AT_BROWSER_CLOSE - Whether to expire the session when the user closes their browser. Default: False

SESSION_FILE_PATH - If you're using file-based session storage, this sets the directory in which Django will store session data. When the default value (None) is used, Django will use the standard temporary directory for the system. Default: None

SESSION_SAVE_EVERY_REQUEST - Whether to save the session data on every request. If this is False (default), then the session data will only be saved if it has been modified that is, if any of its dictionary values have been assigned or deleted. Empty sessions won't be created, even if this setting is active. Default: False

Session Settings

SESSION_SERIALIZER - Full import path of a serializer class to use for serializing session data. Included serializers are:

'django.contrib.sessions.serializers.PickleSerializer'

'django.contrib.sessions.serializers.JSONSerializer'

Default: 'django.contrib.sessions.serializers.JSONSerializer’

Time Zone Link : https://en.wikipedia.org/wiki/List_of_tz_database_time_zones