Django Cookie

A cookie is a small piece of information which is stored in the client browser.

It is used to store user's data in a file permanently (or for the specified time).

In Django, cookies are small pieces of data that are stored on the client-side (usually in the user's browser).

Cookies can be used for various purposes, such as keeping a user logged in, storing preferences, or tracking user behavior.

Setting a Cookie in Django

To set a cookie, you can use the HttpResponse object and set the

set\_cookie() method.

You can add cookies to the response before returning it to the user.

from django.http import HttpResponse

def set\_cookie(request):

response = HttpResponse("Cookie has been set!")

# Set a cookie with the key 'username' and value 'Jpriya'

response.set\_cookie('username', 'priya', max\_age=3600) # expires in 1 hour

return response

'username': The key for the cookie.

'priya': The value for the cookie.

max\_age=3600:

Optional Parameters:

max\_age: The maximum age of the cookie in seconds.

expires: You can set an exact expiration date and time for the cookie.

path: Defines the URL path for which the cookie is valid.

domain: Specifies the domain for which the cookie is valid.

secure: If set to True, the cookie will only be sent over HTTPS.

httponly: If set to True, the cookie can only be accessed via HTTP requests (

Getting a Cookie in Django

To read a cookie, you can access it via the request.COOKIES dictionary.

If the cookie exists, it will return its value; otherwise, it will return None

This sets the cookie's expiration time in seconds (optional). In this example, the cookie will expire after one hour.

from django.http import HttpResponse

def get\_cookie(request):

username = request.COOKIES.get('username', 'Guest') # Default to 'Guest' if cookie doesn't exist

return HttpResponse(f"Hello, {username}!")

Deleting a Cookie in Django

To delete a cookie, you can use the delete\_cookie() method on the response object. You can set the cookie's expiration date to a past time to instruct the browser to delete it.

from django.http import HttpResponse

def delete\_cookie(request):

response = HttpResponse("Cookie has been deleted!")

response.delete\_cookie('username')

return response

Cookie has its expiry date and time and removes automatically when

gets expire.

Django provides built-in methods to set and fetch cookie.

The set\_cookie() method is used to set a cookie and get() method is

used to get the cookie.

A dictionary containing all cookies,keys and values are strings

1)set\_cookie()-

set\_cookie() is used to set/create/sent cookies

syntax:

HttpResponse.set\_cookie(key,value='',max\_age=None,expires=None,path='',domain=None

,secure=False,httponly=False,samesite=None)

key-This is the name of the cookie

value=this sets the value of the cookie.This value is stored on the clients computer

name and value are required to set cookie

when you omit the optional cookie field the browser files

them in automatically with reasonable defaults

max\_age-

it should be a number of seconds,or None if the cookie

should last only as long as the clients browser session

if expires is not specified it will be calculated

ex:

set\_cookie('name','sonam',max\_age=60\*60\*24\*10)//10days

expires-it describe the time when cookie will be expires

it should either be a string in the format

('Wdy',DD-MON-YY HH:MM:SS GMT") or a datetime.datetime.object in UTC

if expires is a datetime objectthe max\_age will be calculated

eg:

set\_cookie('name','priya',expires=atetime.utcnow()+timedelta(days=2))

domain-

use domain if you want to set a cross domain cookie

domain='example.com' will set a cookie that is readable by the

domains www.new.com

otherwise a cookie will only be readable by the domain that set it

httponly-

Use httponly-True

if you want to prevent client side js from having access to the

cookie

HttpOnly is a flag included in a Set-Cookie HTTP responce header

samesite-

use samesite=Strict

reading/accessing cookie

HttpRequest.COOKIES-a DICTIONARY CONTAINING ALL COOKIES

KEYS AND VALUES ARE STRINGS

syntax:

request.COOKIES['key']

request.COOKIES['name']

syntax:

request.COOKIES.get('key','default')

request.COOKIES.get('name')

example:

def setcookie(request):

response=render(request,'setcookie.html')

response.set\_cookie('name','priya')

return response

write url:

path('set/',views.setcookie)

write code in setcookie.html

<h4>cookie set</h4>

def getcookie(request):

name=request.COOKIES['name']

return render(request,'getcookie.html',{'name':name})

getcookie.html

<h4>Get cookie</h4>

{{name}}

urls.py

path('get/',views.getcookie)

if cookie not found then give error

so we have to write

def getcookie(request):

name=request.COOKIES.get('name')

return render(request,'getcookie.html',{'name':name})

**Steps to Use Sessions in Django**

1. **Ensure Session Middleware is Enabled**:
   * By default, Django includes session middleware. Check that it’s in your MIDDLEWARE settings.

MIDDLEWARE = [ 'django.middleware.security.SecurityMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware', # other middleware ]

**Session Settings in settings.py**:

* You can configure how session data is stored by modifying the SESSION\_ENGINE setting.

SESSION\_ENGINE = 'django.contrib.sessions.backends.db' # Store sessions in the database (default). # Other session backends: 'django.contrib.sessions.backends.cache', 'django.contrib.sessions.backends.file', 'django.contrib.sessions.backends.cached\_db'

**Using Sessions in Views**:

* You can store and retrieve session data within your views using request.session

def set\_session(request):

request.session['username'] = 'priya'

return HttpResponse("Session data stored.")

**Retrieving data from the session**:

def get\_session(request):

username = request.session.get('username', 'Guest')

# Default value is 'Guest' if 'username' is not set

return HttpResponse(f"Hello, {username}!")

**Deleting session data**:

def delete\_session(request):

if 'username' in request.session:

del request.session['username']

return HttpResponse("Session data deleted.")

**Session Expiry**:

* You can set the session to expire after a certain period using the SESSION\_COOKIE\_AGE setting.

SESSION\_COOKIE\_AGE = 3600 # Session will expire after 1 hour.

from django.shortcuts import render

from django.http import HttpResponse

def set\_session(request):

request.session['user\_id'] = 42

request.session['user\_name'] = ‘priya’

return HttpResponse("Session data has been set.")

def get\_session(request):

user\_id = request.session.get('user\_id', None)

user\_name = request.session.get('user\_name', 'Guest')

if user\_id:

return HttpResponse(f"User ID: {user\_id}, User Name: {user\_name}")

else:

return HttpResponse("No session data found.")

def delete\_session(request):

try:

del request.session['user\_id']

del request.session['user\_name']

except KeyError:

pass

return HttpResponse("Session data has been deleted.")

In Django, **sessions** are a way to store data between requests for a particular user. They allow you to store information (such as user data or preferences) on the server side and associate it with a session ID, which is typically stored in a cookie on the user's browser. Sessions are useful for maintaining stateful interactions in web applications.

### How Django Sessions Work

1. **Session Storage**: By default, Django stores session data in the database, but it also supports other backends like cached sessions or file-based sessions.
2. **Session ID**: Django generates a unique session ID and stores it in a cookie on the client’s browser. This session ID is sent with each subsequent request and is used to retrieve the corresponding session data from the backend.
3. **Session Data**: Session data is stored as a dictionary-like object, and you can add, modify, or delete data in the session using request.session.