**Django Sessions**

About Django Sessions

[[Okan Yenigün](https://medium.com/@okanyenigun?source=post_page-----70a40aaeaa28--------------------------------)](https://medium.com/@okanyenigun?source=post_page-----70a40aaeaa28--------------------------------)

[[AWS Tip](https://awstip.com/?source=post_page-----70a40aaeaa28--------------------------------)](https://awstip.com/?source=post_page-----70a40aaeaa28--------------------------------)

[Okan Yenigün](https://medium.com/@okanyenigun?source=post_page-----70a40aaeaa28--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F776cbeb7041f&operation=register&redirect=https%3A%2F%2Fawstip.com%2Fdjango-sessions-70a40aaeaa28&user=Okan+Yenig%C3%BCn&userId=776cbeb7041f&source=post_page-776cbeb7041f----70a40aaeaa28---------------------post_header-----------)

Published in

[AWS Tip](https://awstip.com/?source=post_page-----70a40aaeaa28--------------------------------)

·

5 min read

·

4 days ago

116

1



Photo by [Artiom Vallat](https://unsplash.com/@virussinside?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash) on [Unsplash](https://unsplash.com/photos/gray-and-black-asus-laptop-turned-on-D1XjCORamys?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash" \t "_blank)

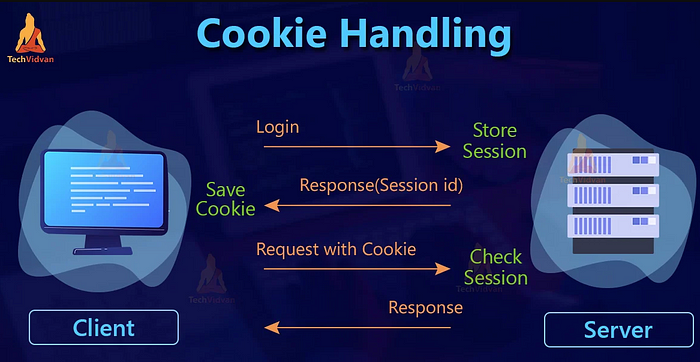
HTTP, which stands for HyperText Transfer Protocol, is the foundational protocol used for communication on the web. It is designed to be stateless, meaning each request from a client (like a web browser) to a server is processed independently, without any memory of previous interactions. After a request is processed, the server does not retain any state (or memory) of that request.

While statelessness simplifies the protocol and makes it more scalable, it poses challenges for creating interactive and personalized web applications. For instance, without maintaining state, a web application wouldn’t be able to easily remember a user’s previous actions, preferences, or authentication status across different web pages.

In typical server-based applications (like those built using Django), a new session is initiated when a user (or more specifically, a user’s browser) first interacts with the application. A session is a way to store data for that particular user’s interaction with the server.

When a session is created, the server sends a small piece of data called a “session cookie” to the user’s browser. This cookie contains a unique identifier called the “session ID”. This ID is what the server uses to recognize and differentiate between different users (or more accurately, different browser sessions).

The session cookie is stored in the user’s browser and sent back to the server with each subsequent request, allowing the server to maintain a consistent session across multiple requests from the same browser.



[Source](https://techvidvan.com/tutorials/django-cookies-handling/#google_vignette)

Middleware components are executed in order during the request and response phases. When a request comes in, the Session Middleware takes the session ID from the client’s cookie (if present) and retrieves the associated session data from the configured session engine (database, file, cache, etc.). It then makes this data available as request.session during the processing of the view.

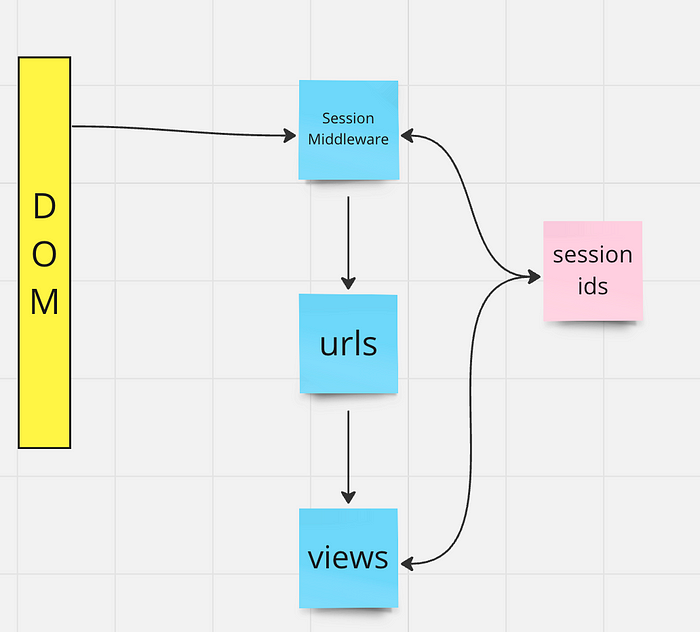


Image by the author.

# Middleware in settings.py  
MIDDLEWARE = [  
...  
django.contrib.sessions.middleware.SessionMiddleware  
...  
]

After migrating, Django creates a table in the database for sessions.

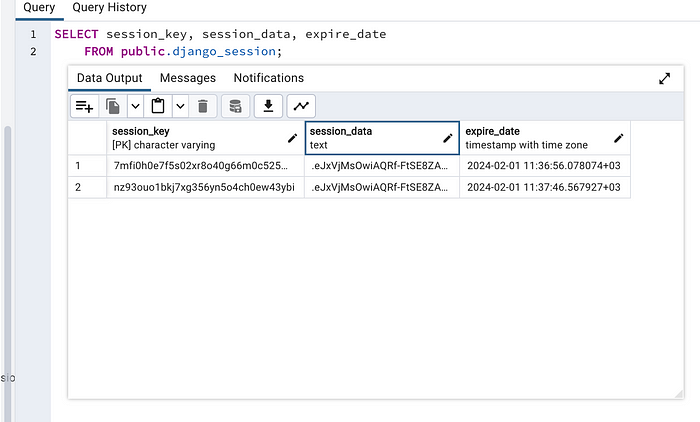


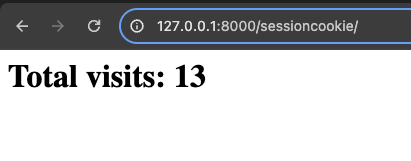
Image by the author.

In Django, every user request to the server is encapsulated in a request object. This object includes an attribute named request.session. It works similarly to a Python dictionary, allowing us to store, retrieve, and manage data using key-value pairs.

For demonstration purposes, I created a new app (*sessioncookie*) in the project. [You can get the project here.](https://github.com/okanyenigun/djangoexample)

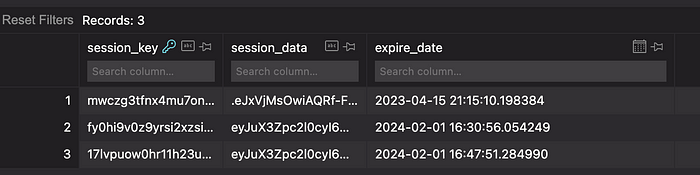
Here, I have created a view that essentially counts the number of times a user visits the URL.

# views.py  
  
from django.http import HttpResponse  
from django.template import Template, Context  
from django.views.decorators.cache import never\_cache  
  
@never\_cache  
def sessionvisit\_view(request):  
 n\_visits = int(request.session.get('n\_visits', 0)) + 1  
 t = Template('<h1>Total visits: {{n\_visits}}</h1>')  
 c = Context({"n\_visits": request.session["n\_visits"]})  
 request.session["n\_visits"] = n\_visits  
 return HttpResponse(t.render(c))



Whenever I refresh the page, the count increases.

In the django\_session table, I can find the session keys and the data.



We can show the session attributes in the web browser as well (using the Set-Cookie header). Thus, the browser stores this data in cookies, and each time it makes a request to the server, this information is transmitted within the cookie header.

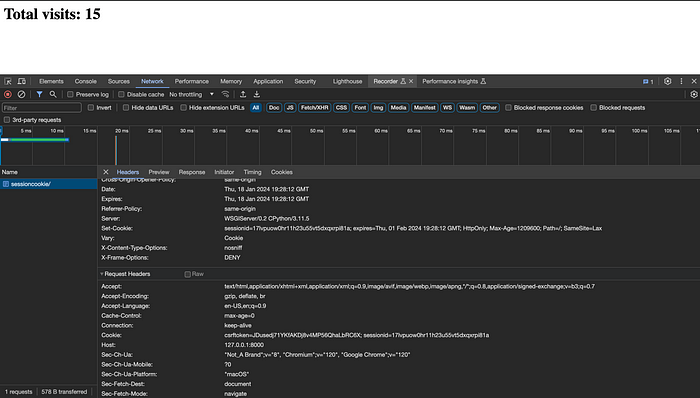


Image by the author.

The session\_data is an encoded string (base64). We can decode it.

import base64  
import json  
  
session\_str = "eyJuX3Zpc2l0cyI6MTR9:1rQVYV:tvm4LNPRtDXalmCcaR3cXeIMgjj3c8BqzwDca0GdK0U"  
  
# Split the string to get the base64 portion  
base64\_data = session\_str.split(":")[0]  
  
# Pad the string if necessary  
padded\_base64\_data = base64\_data + '=' \* (-len(base64\_data) % 4)  
  
# Decode the base64 string  
decoded\_data = base64.b64decode(padded\_base64\_data)  
  
# Assuming the decoded data is JSON serialized  
deserialized\_data = json.loads(decoded\_data)  
  
print(deserialized\_data)  
  
"""  
{'n\_visits': 14}  
"""

Sessions and cookies are both key components in managing state and user data in web applications, but they serve different purposes and operate in distinct ways.

Storage Location

* Cookies: Stored on the client-side, i.e., in the user’s web browser.
* Sessions: Stored on the server-side. The session ID, which is a reference to the server-side data, may be stored in a client-side cookie.

Security

* Cookies: As they are stored on the client-side, cookies are more vulnerable to security threats like tampering and eavesdropping. Sensitive data should not be stored directly in cookies.
* Sessions: Generally more secure as the data is stored on the server. Only the session ID is exchanged with the client, reducing the risk of sensitive data exposure.

Size

* Cookies: Limited in size (typically 4KB). Not suitable for storing large amounts of data.
* Sessions: Can store larger amounts of data as they are not subject to the same size limitations as cookies.

Use Cases

* Cookies: Ideal for storing small pieces of non-sensitive information like user preferences or settings.
* Sessions: Better suited for managing user authentication, storing user-specific data during the web session, and handling sensitive data.

**One Sentence To Remember**: Django sessions provide a mechanism to persist user data across multiple HTTP requests, enabling stateful interactions in an otherwise stateless protocol.