Digital Career Institute

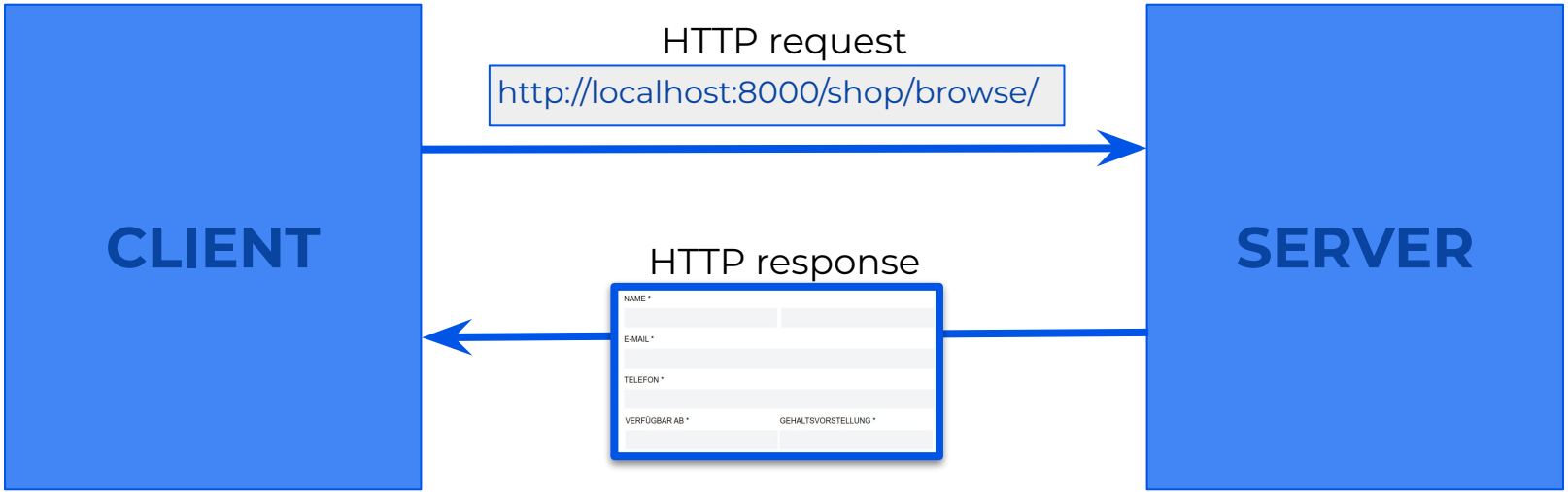
Python Course - Django Advanced Features

Digital Career Institute

Web Sessions

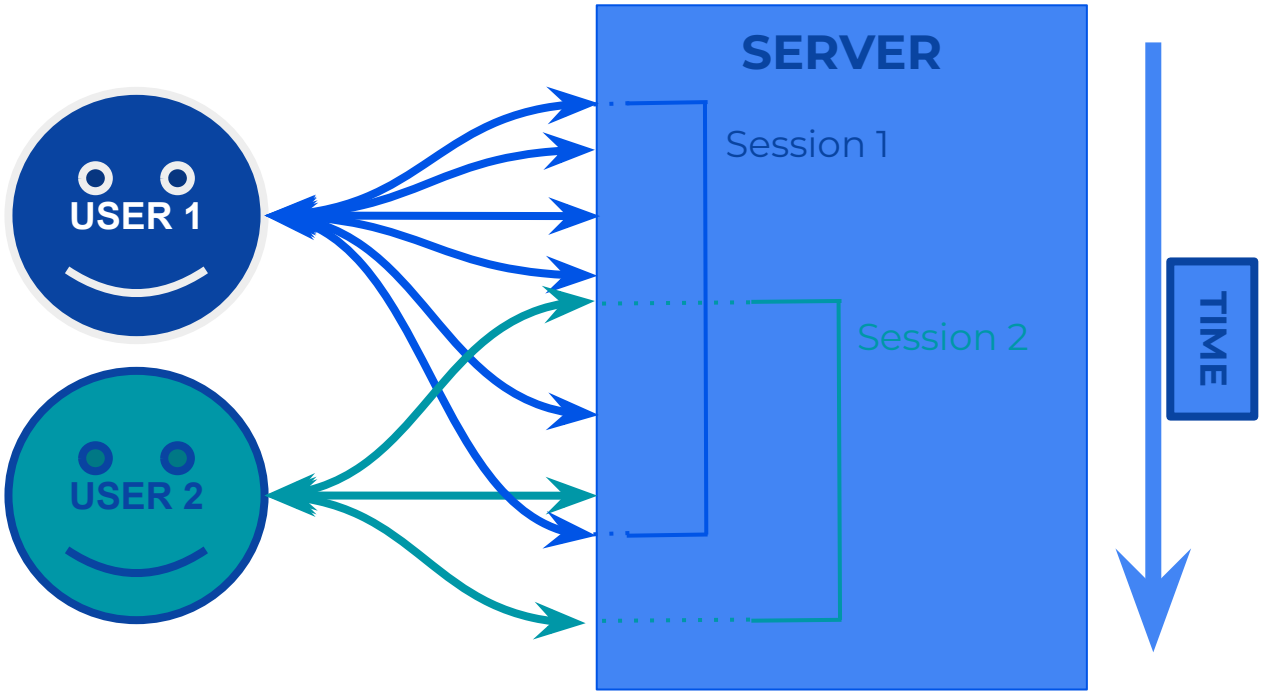
HTTP requests

HTTP requests are **stateless**.



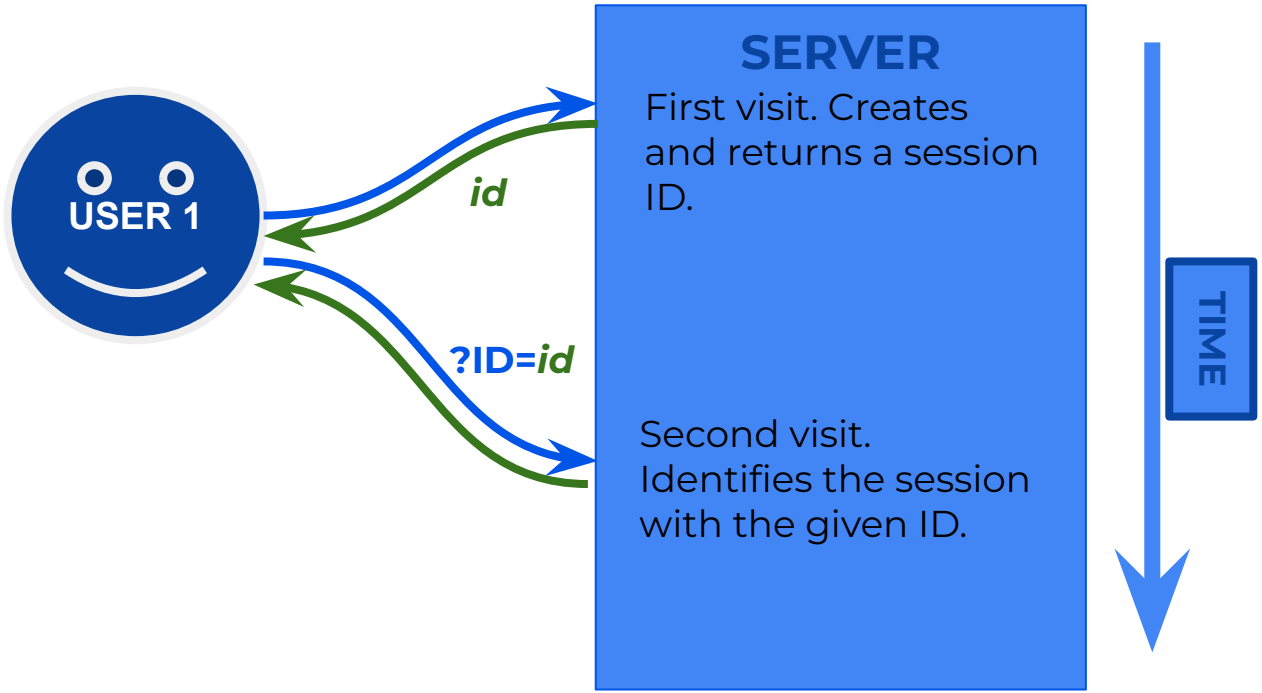
They don't change state.
An HTTP request knows nothing about any previous HTTP request ... or, does it?

What are Sessions

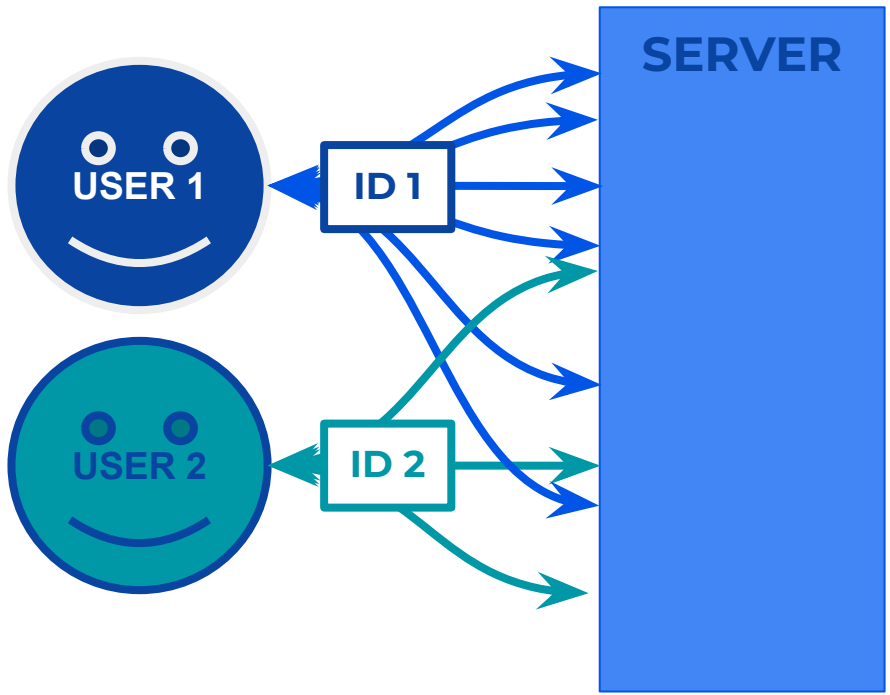


What are Sessions

How Sessions Work: Session ID

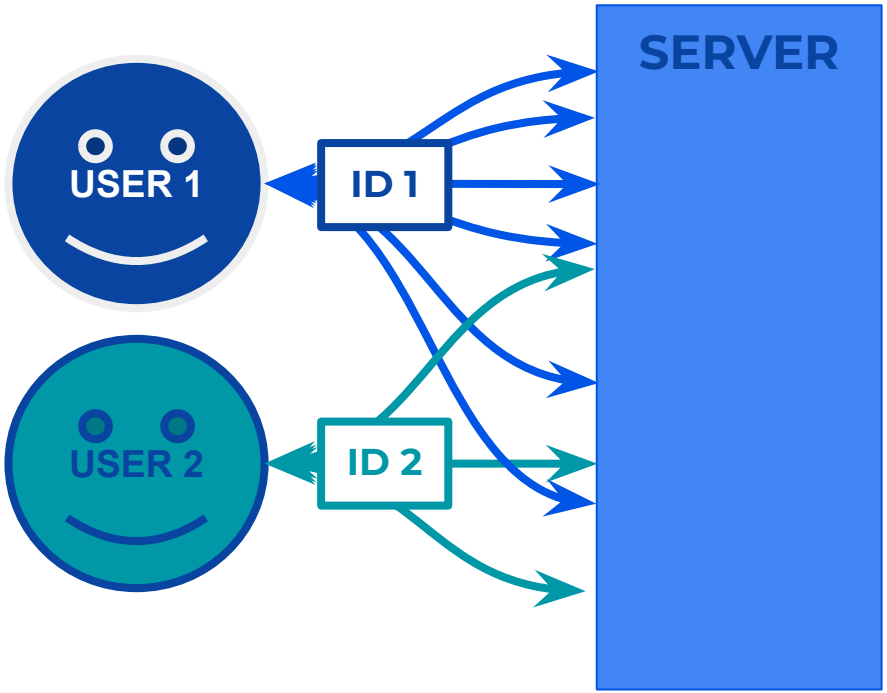


Why are Sessions Useful: Session Data



- User 1 is logged in.
- User 2 is not.
- User 1 has the notifications off.
- User 1 has already applied for an offer.
- User 2 has visited the site for the first time ever, a special message will be shown.
- User 1 is not interested in electronics. The site will show only offers about clothing.
- The article X has been modified by User 1.
- ...

When to use Sessions



We need them

- To implement a user system.
- To keep track of user actions.
- To tailor the content.

We don't need them

- To publish a static website.
- To publish a one-way website.

Where is the Session ID Stored?

Where are the Session Data Stored?

Cookies

Cookies are small fragments of text stored in the browser.

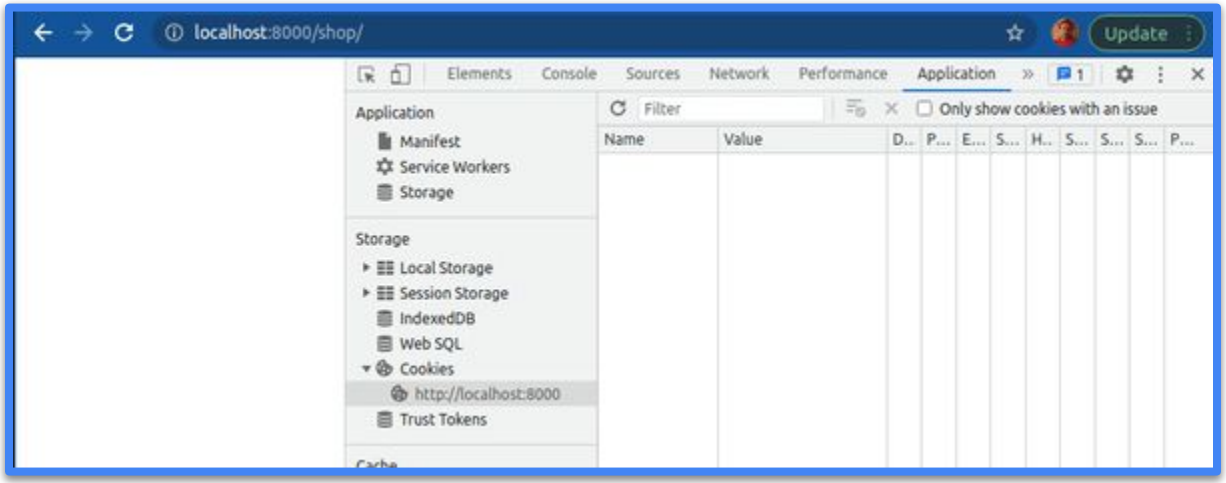
They stay in the browser after the request has been resolved.

They can only be accessed by pages on the domain they were created.

They have an expiration date.

Setting Session Data with Django

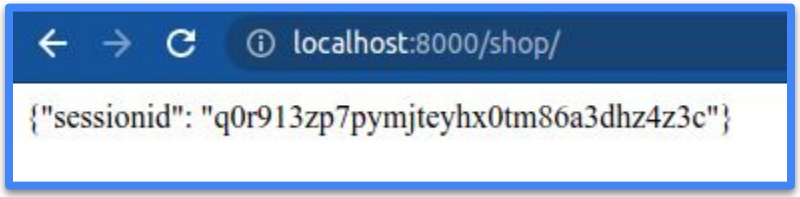
When we set session data, Django automatically stores the session ID in a cookie named **`sessionid`**.



Getting the Session ID

shop/views.py

```
def home(request):
    if request.session.get("is_first_visit", True):
        request.session["is_first_visit"] = False
    message = json.dumps(request.COOKIES)
    return HttpResponse(message)
```



The cookies can be accessed using the **`request.COOKIES`** dictionary.

Getting the Session ID

shop/views.py

```
def home(request):
    if request.session.get("is_first_visit", True):
        request.session["is_first_visit"] = False
    message = request.session.session_key
    return HttpResponse(message)
```

The session id can also be accessed using the **`session_key`** property of the **`request.session`** object.

Session Expiration with Django

Django's session expires when the `sessionid` cookie expires.

By default, this is **two weeks** after it has been set. The default value can be changed by adding a constant in the `hello/settings.py` file named `SESSION_COOKIE_AGE` with the expiry time in seconds.

shop/views.py

```
def home(request):
    if request.session.get("is_first_visit", True):
        message = "Welcome newcomer!"
        request.session["is_first_visit"] = False
        if today_is_crazy_monday():
            request.session.set_expiry(60)
```

The session expiration can also be defined programmatically with the `set_expiry` method of the `request.session` object.

Session Expiration with Django

Django's `request.session` also provides methods to get the current expiry details.

shop/views.py

```
def home(request):
    if request.session.get("is_first_visit", True):
        request.session["is_first_visit"] = False
        print("Expires on",
              request.session.get_expiry_date())
        print("Expires in",
              request.session.get_expiry_age(),
              "seconds")
```

```
Expires on 2021-10-03 12:47:57.571251+00:00
Expires in 1209600 seconds
```

Ending the Session with Django

The session can be programmatically ended by using the **`flush`** method.

shop/views.py

```
def home(request):
    request.session.flush()
    if request.session.get("is_first_visit", True):
        request.session["is_first_visit"] = False
```

Old expired sessions that may still reside in the browser's storage can be purged with the **`clear_expired`** method.

Django's Session Storage Options

Django can store the session data using any of these methods:

Django's Default Storage

To use the default method we must prepare first the **database**.

Django comes ready to be used with a self-contained database (SQLite) that doesn't require any further configuration or software.

```
(env) $ python manage.py migrate
```

The `migrate` command will prepare the database with the structure required for our web application.

Session File Storage

To store the session data in files, the **`SESSION_ENGINE`** constant must be defined as **`django.contrib.sessions.backends.file`** in the **`settings.py`** file.

hello/settings.py

```
...
SESSION_ENGINE = "django.contrib.sessions.backends.file"
SESSION_FILE_PATH = "/path/to/directory/"
...
```

The location where these files will be stored defaults to the default temporary directory (usually **`/tmp`**), but this can be changed with **`SESSION_FILE_PATH`**.

Session Cookie Storage

To store the session data in cookies, the `SESSION_ENGINE` constant must be defined as `django.contrib.sessions.backends.cookies` in the `settings.py` file.

hello/settings.py

```
...
SESSION_ENGINE = "django.contrib.sessions.backends.cookies"
...
```

The data will be stored on the user's browser and this means higher vulnerability to malicious attacks.

Storing a lot of data in cookies may also have an impact on the website's speed.

Session Security

Session's biggest threat are the cookies stored in the browser.

The risk can never be taken down to 0%, but some things can be done to increase security:

- Store the session data on the server.
- Encrypt the cookie by setting `SESSION_COOKIE_SECURE = True` in `settings.py`.
- If possible, do not allow websites in subdomains to set cookies for all the domain.
- Do not reuse session ids on user authentication. When the user authenticates, a new session id should always be generated.

Login Example

shop/views.py

```
def home(request):
    """The shop home view."""
    username = request.session.get("user_id", None)
    if not username:
        return HttpResponseRedirect("login")
    else:
        return HttpResponse(f"Welcome, {username}!")
```

If the user is not authenticated the application will redirect them to the **`login`** view.

If the user is authenticated it will greet them with their name.

Login Example

shop/views.py

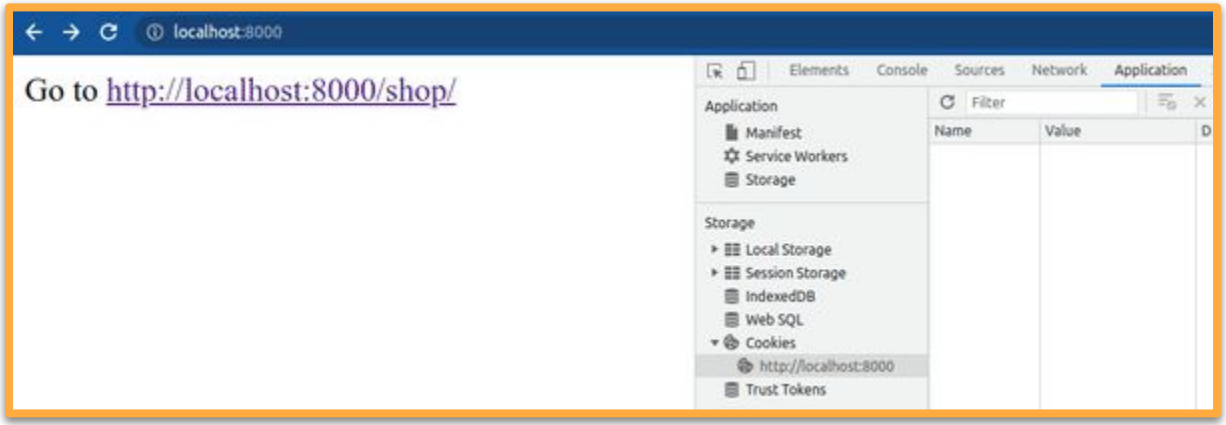
```
class Login(FormView):
    template_name = "login.html"
    form_class = LoginForm
    success_url = reverse_lazy("shop")

    def get(self, request):
        if self.request.session.get("user_id", None):
            # The user is already authenticated
            # No need to proceed with the login
            return HttpResponseRedirect(self.success_url)
        else:
            # The user is not authenticated
            # We proceed with the login
            return super().get(request)
...
```

Login Example

shop/views.py

```
class Login(FormView):
    ...
    def form_valid(self, form):
        username = form.data.get("username", None)
        password = form.data.get("password", None)
        if username == "admin" and password == "admin":
            # Flush the session to generate a new sessionid
            self.request.session.flush()
            # Set the session data
            self.request.session["user_id"] = username
            # Go back to the shop
            return HttpResponseRedirect(self.success_url)
        else:
            # Reload the form
            return super().form_valid(request)
```

Login Example



Logout Example

shop/views.py

```
def Logout(request):
    # We remove all the data and the session id
    request.session.flush()
    # We redirect to the home page
    return HttpResponseRedirect(reverse("home"))
```

OR

shop/views.py

```
class Logout(View):

    def dispatch(self, request):
        self.request.session.flush()
        return HttpResponseRedirect(reverse("home"))
```

We learned ...

- That HTTP requests are stateless and what this means.
- What are sessions and how do they work.
- What are cookies and why are they needed in web applications.
- How Django implements session management and how can we use it to store data.
- One way to implement a login feature with Django.

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute