Cookies & Sessions

Springboard

Download Demo Code

Goals

- Define what it means for HTTP to be stateless
- Compare different strategies for persisting state across requests
- Explain what a cookie is, and how client-server cookie communication works
- Compare cookies and sessions
- Implement session functionality with Flask

Motivation

Saving "State"

HTTP is what's called a "stateless" protocol.

On its own, it remembers nothing.

It's like a goldfish. Every time it circles around, what it sees is brand new.

Some Ways To Save State

- Passing info in a query param / POST form hidden field
  - **/step-zero?fav-color=blue → /step-one?fav-color=blue →** …
- Keeping info in URL path
  - **/fav-color/blue/step-zero → /fav-color/blue/step-one →** …
- Using JS localStorage API
  - Nice, but only JS can access this — you can't get data on server
  - Useful for single-page applications or heavily AJAX-driven apps
- Using cookies / sessions

Cookies

Flask's ***session*** is powered by cookies; let's start there

Cookies Save "State"

Cookies are a way to store small bits of info on client (browser)

What is a Cookie?

Cookies are **name/string-value pair** stored by the **client** (browser).

The server tells client to store these.

The client sends cookies to the server with each request.

Site	Cookie Name	Value
rithmschool.com	number_visits	"10"
rithmschool.com	customer_type	"Enterprise"
localhost:5000	favorite_food	"taco"

Cookies, A Conversation

- *Browser*: I'd like to get the resource ***/upcoming-events***.
- *Server*: Here's some HTML. Also, please remember this piece of information: ***favorite_food*** is `"taco"`.
- *Browser* (stores this somewhere on the computer)
- *Browser*: I'd like to get the resource ***/event-detail***. Also, you told me to remind you that ***favorite_food*** is `"taco"`.
- *Server*: Here's the HTML for that.
- *Browser*: I'd like to get the resource ***/calendar.jpg***. Also, you told me to remind you that ***favorite_food*** is `"taco"`.
- …

Seeing Cookies in Chrome

Dev Tools → Application → Storage → Cookies

Settings Cookies in Flask

demo/app.py

```
@app.route("/handle-form-cookie")
def handle_form():
    """Return form response; include cookie for browser."""

    fav_color = request.args["fav_color"]

    # Get HTML to send back. Normally, we'd return this, but
    # we need to do in pieces, so we can add a cookie first
    html = render_template("response-cookie.html", fav_color=fav_color)

    # In order to set a cookie from Flask, we need to deal
    # with the response a bit more directly than usual.
    # First, let's make a response obj from that HTML
    resp = make_response(html)

    # Let's add a cookie to our response. (There are lots of
    # other options here--see the Flask docs for how to set
    # cookie expiration, domain it should apply to, or path)
    resp.set_cookie("fav_color", fav_color)

    return resp
```

Reading Cookies in Flask

demo/app.py

```
@app.route("/later-cookie")
def later():
    """An example page that can use that cookie."""

    fav_color = request.cookies.get("fav_color", "<unset>")

    return render_template("later-cookie.html", fav_color=fav_color)
```

Cookie Options

- **Expiration**: how long should the browser remember this?
  - Can be set to a time; default is "as long as web browser is running" *(session cookie)*
- **Domain**: which domains should this cookie be sent to?
  - Send only to **books.site.com** or everything at **site.com**?
- **HttpOnly** - HTTP-only cookies aren't accessible via any kind of JavaScript
  - Useful for cookies that contain server-side information and don't need to be available to JavaScript.

Site	Cookie	Expiration	Domain	Value
www.rithmschool.com	number_visits	(browser)	*.rithmschool.com	"10"
shop.rithmschool.com	customer_type	2015-12-31	shop.rithmschool.com	"Enterprise"
localhost:5000	favorite_color	(browser)	localhost:5000	"blue"

Comparison of Types of Browser Storage

- LocalStorage
  - Stores data with no expiration date, and gets cleared only through JavaScript, or clearing the Browser cache
  - Domain specific
  - Storage limit is much larger than a cookie.
- SessionStorage
  - Stores data only for until the browser or tab is closed.
  - Storage limit is much larger than a cookie.
- Cookie
  - Cookies can be made secure by setting the httpOnly flag as true for that cookie. This prevents client-side access to that cookie
  - Sent from the browser to the server for every request to the same domain
  - Set usually from server-side. Can we read by a server

A Visual Display

LocalStorage	5MB/10MB storage It's not session based, need to be deleted via JS or manually Client side reading only Less older browsers support
SessionStorage	5MB storage It's session based and working per window or tab Client side reading only Less older browsers support
Cookie	4KB storage Expiry depends on the setting and working per window or tab Server and client side reading More older browsers support

- Credit

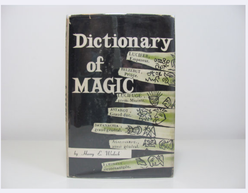
Sessions

Cookies Can Be Tricky

- Cookies are just strings
- Cookies are limited by how much information you can store
- Cookies are a bit low-level in how you use them

Sessions

Flask sessions are a "magic dictionary"



- Contain info for the current browser
- Preserve type (lists stay lists, etc)
- Are "signed", so users can't modify data

Using Session in Flask

- Import ***session*** from ***flask***
- Set a ***secret_key***

```
from flask import Flask, session

app = Flask(__name__)
app.config["SECRET_KEY"] = "SHHHHHHHHH SEEKRIT"
```

Now, in routes, you can treat ***session*** as a dictionary:

```
@app.route('/some-route')
def some_route():
    """Set fav_number in session."""

    session['fav_number'] = 42
    return "OK, I put that in the session."
```

To get things out, treat it like a dictionary:

```
from flask import session

@app.route('/my-route')
def my_route():
    """Return information using fav_number from session."""

    return f"Favorite number is {session['fav_number']}"
```

It will stay the same kind of data (in this example, an integer)

You also have direct access to ***session*** automatically in Jinja templates:

```
Your favorite number is {{ session['fav_number'] }}
```

How Do Sessions Work?

- Different web frameworks handle this differently
- In Flask, the sessions are stored in the browser as a cookie
  - `session = "eyJjYXJ0IjlDIsMlwyLOJdfQ.CP6ryAJEMSZdE"`
  - They're "serialized" and "signed"
  - So users could see, but can't change their actual session data—only Flask can

Advanced details: Flask by default uses the ***Werkzeug*** provided "secure cookie" as session system. It works by serializing the session data, compressing it and base64 encoding it.

Are "Sessions" Related to "Session Cookies"?

Not directly, no.

They both just use the term "session" but to mean something different.

By default: Flask sessions use browser-lifetime cookies ("session cookies"). So a Flask session lasts as long as your browser window.

Yes, you can change that (read the Flask docs!)

This distinction isn't too important right now, but the terminology sometimes comes up in interviews, so be sure to review this material!

Server-Side Sessions

- Some web frameworks store session data on the server instead
  - Often, in a relational database
  - Send a cookie with "session key", which tells server how to get the real data
  - Useful when you have lots of session data, or for complex setups
  - Flask can do this with the add-on Flask-Session

Which Should I Use? Cookies or Sessions?

Generally, sessions.

It's important to know how cookies work, but if your framework provides sessions (as Flask does), they're easier to work with.