

Exploration — HTTP Cookies and HTTP Sessions

Introduction



The HTTP protocol is **stateless**. What this means is that when two requests are sent one after the other by the same user of a web app, HTTP does not maintain any information that links these two requests as coming from the same user. If we think about how we use various web applications this will seem problematic. For example, when shopping online we need to interact with different pages on the shopping website in order to place an order. Placing an order requires that the different requests we send be linked together so that, for example, when we select an item to buy on one page, it shows up in our basket when we go to another page to confirm our purchase and pay for it. However, the HTTP protocol has no built-in capability to link these requests as coming from the same user of the shopping website. Web apps overcome the statelessness of the HTTP protocol by using HTTP cookies and HTTP sessions. This allows web apps to link different requests as coming from the same user. In this exploration, we explore the concepts underlying HTTP cookies and HTTP sessions and explain the API provided by Express to use them in our web apps.

HTTP Cookies

An HTTP cookie (also called browser cookie, web cookie, or simply cookie) is a small piece of data that is created by a web server and sent to the user's web browser for storage. The browser can send this HTTP cookie in later requests to this web server thus identifying who this request is coming from. By linking together multiple requests from the same user, HTTP cookies allow web apps to overcome the stateless nature of the HTTP protocol.

Headers for Cookies

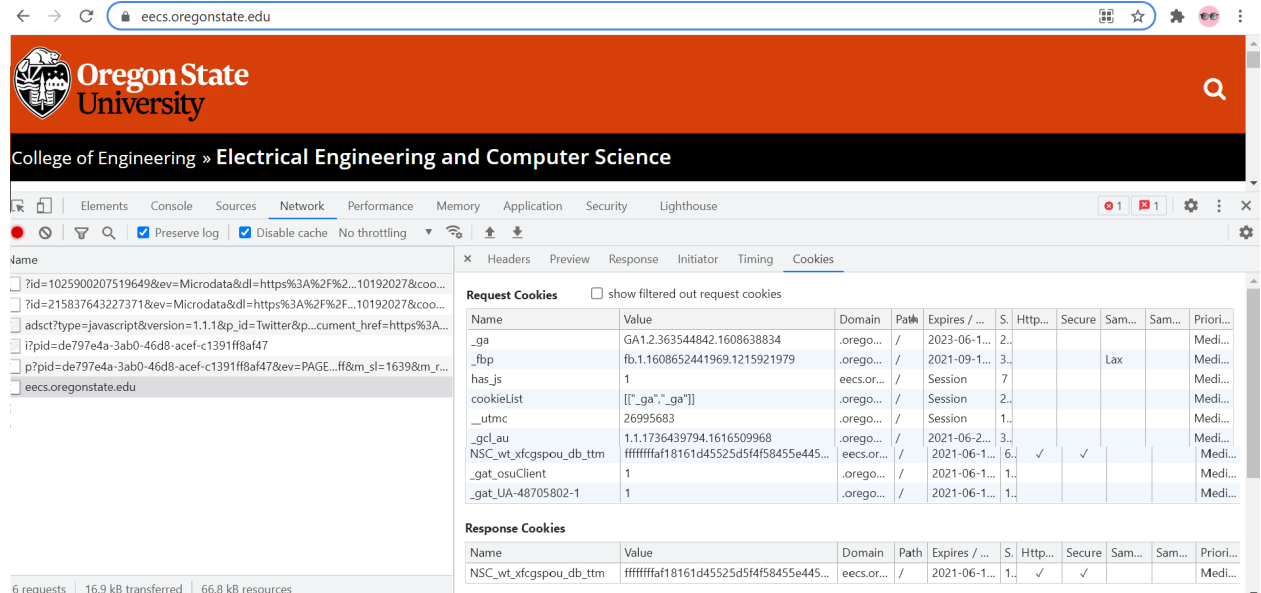
Cookies are name-value pairs.

- When the server wants the browser/client to store a cookie, the server sends the cookie in the response header `Set-Cookie` containing the name-value pair for the cookie.
- When a browser/client wants to send cookies to a server, it sends them in the request header `Cookie` as name-value pairs. A request can contain multiple `Cookie` headers, one for each cookie the browser/client is sending to this server.

Examining Cookies

To examine the cookies being sent in HTTP requests and HTTP responses, we can use the “Developer Tools” in our browser. One way to examine the cookies is by viewing the request and response for the headers `Cookie` and `Set-Cookie`. Browsers, such as Chrome, also provide tabs for viewing just the cookies. For example, in Chrome:

- Open the “Developer Tools” using “Control-Shift-I” or by using the browser menu
- Go to the tab “Network”, click on a request and then in the detailed panel, click on “Cookies.”
- Both the request and the response cookies will be shown.



Request and response cookies when visiting the URL `https://eecs.oregonstate.edu/` displayed by the Chrome web browser’s Developer Tools. Note that the cookies displayed to you might be different based your user preferences and prior history of visiting this website.

Cookies and Express

To use cookies in our Express app, we need to include the [cookie-parser middleware](http://expressjs.com/en/resources/middleware/cookie-parser.html) (<http://expressjs.com/en/resources/middleware/cookie-parser.html>). We need to first install the package by using `npm install cookie-parser`, import the package and use it, e.g., by using the following statements:

```
import cookieParser from 'cookie-parser';
app.use(cookieParser());
```

We can then set, delete and get cookies as follows.

Setting a cookie

Call the method [cookie on the response object](http://expressjs.com/en/4x/api.html#res.cookie) (<http://expressjs.com/en/4x/api.html#res.cookie>).

- E.g., `res.cookie('language', 'English')` will set a cookie with name `language` and value `English` on the response object.

This method can be provided an optional 3rd parameter called `options` which is JSON object that can set certain properties on the cookie. Detailed information can be found in the API docs. We highlight a few important options:

- `maxAge`: This option specifies the time in milliseconds the client should keep the cookie before deleting it. If this value is not specified, the cookie will be deleted when the browser is closed.
- `path`: The path this cookie applies to.
- `secure`: The value `true` specifies that the cookie will only be sent over a secure, i.e., HTTPS, connection.
- `signed`: The value `true` specifies that cookie should be signed, as explained later in this section.

Deleting a cookie

Call the method [**`clearCookie` on the response object**](http://expressjs.com/en/4x/api.html#res.clearCookie) (<http://expressjs.com/en/4x/api.html#res.clearCookie>).

- E.g., `res.clearCookie('language')` will delete the cookie named `language`.

Getting the value of a cookie

We can get the value of a cookie sent from the browser/client from the property [**`cookies`**](http://expressjs.com/en/4x/api.html#req.cookies) (<http://expressjs.com/en/4x/api.html#req.cookies>) of the request object. This property is of type object and a cookie, if sent in the request, will be available with a property with the name of that cookie.

- E.g., `req.cookies.language` will give the value of the cookie named `language` if that cookie was sent by the browser/client. Otherwise, we will get the value `undefined`.

Signed Cookies

Cookies are stored on the machine where the browser is installed. The user has access to the cookies stored on their machine and can change their values. To *prevent* tampering of a cookie's value by the user, we can use **signed cookies**. If the value of a signed cookie has been tampered with, the `cookie-parser` will recognize this and the server will reject the value of the cookie. To use signed cookies, we need to provide a secret to the cookie parser middleware. The cookie parser middleware will then use this secret to sign the cookie.

```
import cookieParser from 'cookie-parser';
app.use(cookieParser('s0me4rAnDom$trIngCangohere'));
```

Setting A Signed Cookie

Once the cookie parser has been set up with a secret, we can create signed cookies by passing the option `{signed: true}` when setting the cookie:

```
res.cookie('favorite_icecream', 'mintChocolateChip', {signed: true });
```

Getting A Signed Cookie

We obtain signed cookies, by using the property `signedCookies` on the request object:

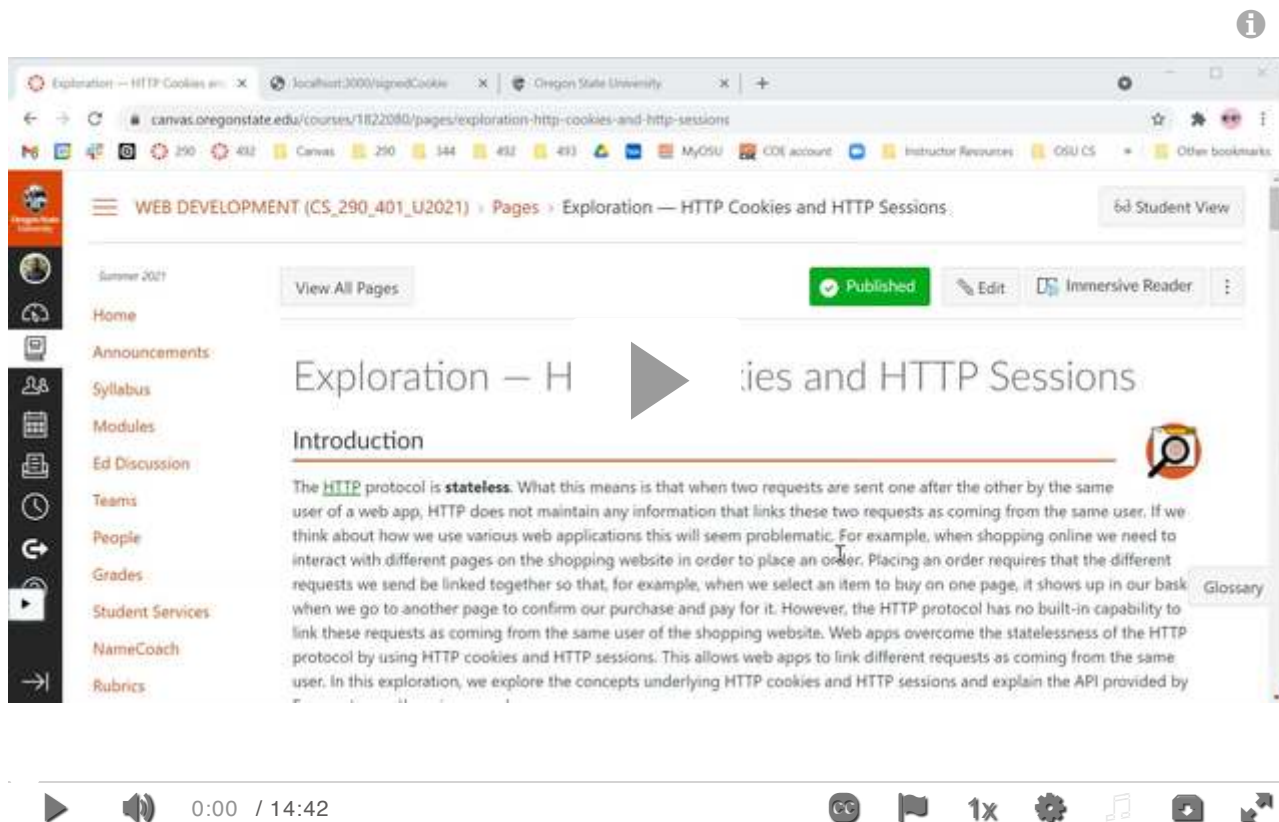
```
const favoriteIcecream = req.signedCookies.favorite_icecream;
```

Example: Setting and Getting Cookies (optional)

To run the example download the file [cookies-example.zip](https://canvas.oregonstate.edu/courses/1879154/files/93831973?wrap=1)

(<https://canvas.oregonstate.edu/courses/1879154/files/93831973?wrap=1>) ↓

(https://canvas.oregonstate.edu/courses/1879154/files/93831973/download?download_frd=1) , unzip it in a new directory, and run the commands `npm install` and `npm start`.



In the following example, the Home Page served to the user asks them to pick a preferred language and submit a form. The submission of the form is handled by the route `POST /` and its handler sets a cookie with name `language` and value of the language chosen by the user. When the user visits the route `GET /greetings`, the route handler gets the value of the preferred language

from the cookie `language` and displays the greeting in the chosen language. The relevant parts of `server.mjs` file are shown below

```
import express from 'express';
import cookieParser from 'cookie-parser';

const COOKIE_SECRET = 's0me4rAnDom$stringCangohere';

// Install cookie-pasrser middleware
app.use(cookieParser(COOKIE_SECRET))

...

/**
 * Actual web apps should use locale to determine the language preference.
 */
app.post('/', (req, res) => {
  // Get user's language preference
  let language = req.body.language;

  // Set cookie
  res.cookie('LANGUAGE', language);

  // Send link to greeting page
  res.send('<a href="/greeting">Click</a> to get your greeting');
});

/**
 * Actual web apps should use locale to determine the language preference.
 */
app.get('/greeting', (req, res) => {
  // Find the preferred language from the cookie and
  // display the greeting in that language
  const greeting = req.cookies.LANGUAGE === 'spanish'
    ? 'Hola Mundo!'
    : 'Hello World!';
  res.send(greeting);
});

/**
 * Send back info based on whether the request includes a
 * signed cookie with name: favorite_icecream
 * If the cookie isn't found on the request, add it to the response.
 */
app.get('/signedCookie', (req, res) => {
  let found = false;
  if (req.signedCookies.favorite_icecream !== undefined) {
    found = true;
  } else {
    // Add a signed cookie to the response
    res.cookie('favorite_icecream', 'mintchocolateChip', { signed: true });
  }
  const message = found ?
    `Your favorite icecream is ${req.signedCookies.favorite_icecream}`
    : `signedCookie was not found. We are setting it.`
  res.send(message)
});
```

HTTP Sessions

HTTP sessions are another abstraction to store state. When large amounts of data need to be stored, one way is by using many cookies. But users don't typically like apps to store large amounts

of their data. Additionally, a user can delete cookies, which means their data can be lost. Users don't like having their data stored, but they don't like having their preferences deleted either.

Most websites store user information, including preferences, on the *server* side, possibly in a database. When a user logs in, the website creates a **session object** on the server and **sends a cookie with a unique identifier** to the user. When subsequent requests contain a cookie with this unique session ID, the website can link requests from this user and can store data relevant to this user's interaction with the website in the session object.

HTTP Sessions and Express

To use sessions in our Express apps, we can use [express-session middleware](https://www.npmjs.com/package/express-session) (<https://www.npmjs.com/package/express-session>). HTTP sessions are built on top of cookies, so in addition to installing `express-session` middleware, we also need to install `cookie-parser`. Additionally, in our program we must include `express-session` middleware after including the `cookie-parser` middleware, as shown below:

```
// Sessions use cookie, so include the cookie parser middleware before the express session middleware
app.use(cookieParser('some cookie secret'))

app.use(expressSession({
  secret: 'some cookie secret',
  cookie: { maxAge: 3600000 }
}))
```

Configuring an HTTP Session in Express

The `express-session` middleware takes a configuration object as a parameter. The details can be found in the [docs of the npm package](https://www.npmjs.com/package/express-session) (<https://www.npmjs.com/package/express-session>). We highlight a few of the options:

- `secret`: The string to be used to sign the cookie created for the session ID.
- `cookie`: Options for the cookie to be created for the session ID.
- `key`: The name of the cookie that will store the value of the unique session identifier. If this is not specified, the default name is `connect.sid`.

Getting and Setting Session Values

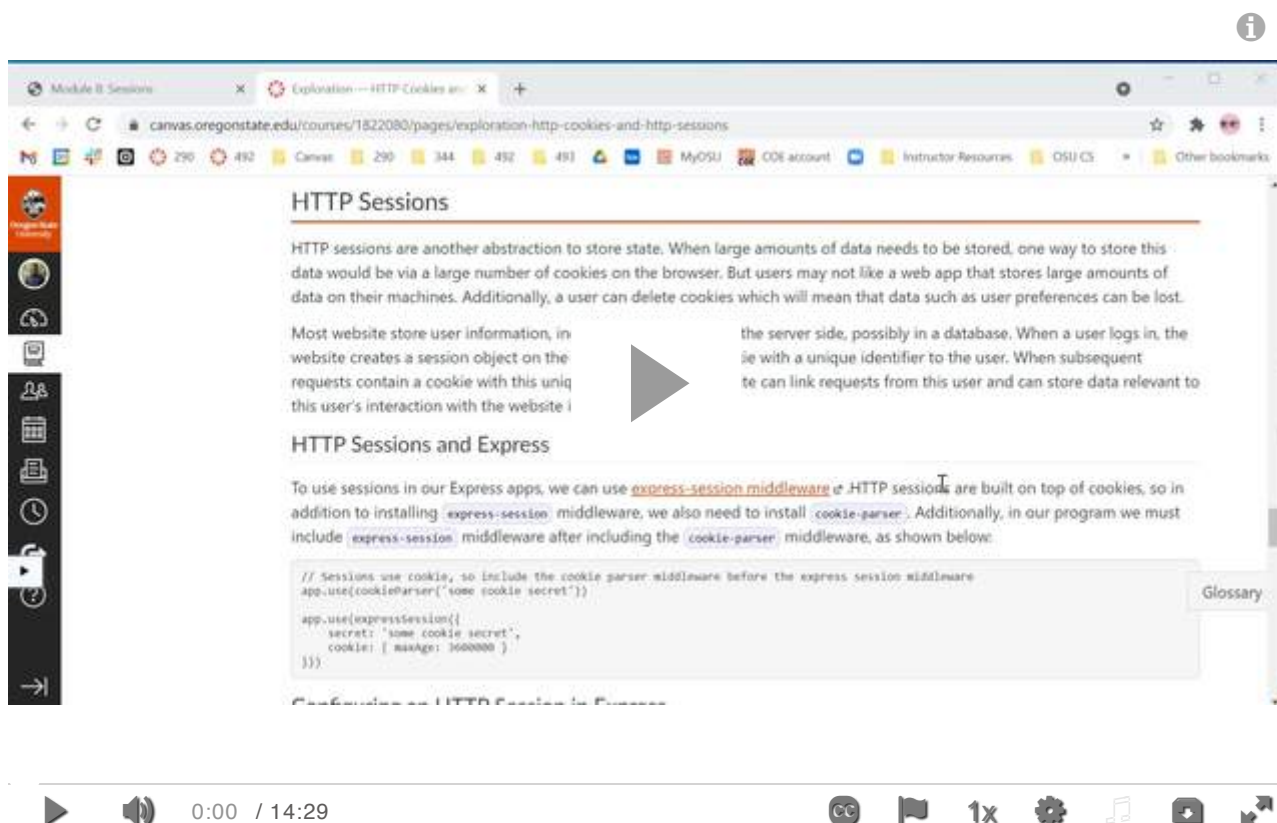
If we set up HTTP sessions in our Express app, then the request object has a property `session` of type object. We can set properties on the `session` object to store data in the session and access these properties to get data from the session, as shown below:

```
req.session.language = 'English';

const languagePreference = req.session.language;
```


Example: Storing and Getting Data from Session (optional)

To run the example download the file [sessions-example.zip](https://canvas.oregonstate.edu/courses/1879154/files/93831972?wrap=1) (<https://canvas.oregonstate.edu/courses/1879154/files/93831972?wrap=1>) , unzip it in a new directory, and run the commands `npm install` and `npm start`.



The screenshot shows a video player interface. The video content is a webpage titled "HTTP Sessions". The text on the page explains that HTTP sessions are an abstraction for storing state, often using cookies. It also discusses using sessions in Express.js, mentioning the "express-session" and "cookie-parser" middleware. A code snippet is shown for configuring these middleware functions.

The following example is a variation on the cookie example we discussed earlier in this exploration. In this example, we store the language preference sent in the HTTP request as a property on the HTTP session in the route handler for the endpoint `POST /`. The route handler for `GET /greeting` gets the value of the language preference from the session and displays the greeting in the preferred language of the user.

```
import express from 'express';
import cookieParser from 'cookie-parser';
import expressSession from 'express-session';

...

const COOKIE_SECRET = 's0me4rAnDom$tringCangohere';

// Sessions use cookie, so include the cookie parser middleware before the express session middleware
app.use(cookieParser(COOKIE_SECRET))

/*
```

```
* We are setting the age of the cookie to 60*60*1000 milliseconds or 1 hour
*/
app.use(expressSession({
  resave: false,
  saveUninitialized: false,
  secret: COOKIE_SECRET,
  cookie: { maxAge: 3600000 }
}));

app.post('/', (req, res) => {
  // Set language preference on the session
  req.session.language = req.body.language;
  // Send link to greeting page
  res.send('<a href="/greeting">Click</a> to get your greeting');
})

app.get('/greeting', (req, res) => {
  // Find the preferred language from the session and
  // display the greeting in that language
  const greeting = req.session.language === 'spanish'
    ? 'Hola Mundo!'
    : 'Hello World!';
  res.send(greeting)
});
```

Configuring Session Storage

By default, Express stores session objects in memory. This is called **memory store** and can be problematic for two reasons:

1. When our server goes down, the session data is lost even if the server restart very quickly and is again ready to process HTTP requests.
2. When a web app needs to handle a very large volume of HTTP requests, scalability is typically achieved by adding multiple servers and sending the HTTP requests to all these servers. In such a scenario, one request from a user may get sent to one server and the next request may get sent to another server. But the session data will be available on one server and not on the other.

Due to these reasons, real-world production Express servers store session information in a database, instead of in memory. This way any of the multiple Express server set up to process HTTP requests can get the session information of a user by querying the session ID cookie. To do this we can configure `express-session` middleware to store session data in any of a long list of supported databases, including MongoDB, which are [listed here](https://www.npmjs.com/package/express-session#compatible-session-stores) [_\(https://www.npmjs.com/package/express-session#compatible-session-stores\)_](https://www.npmjs.com/package/express-session#compatible-session-stores). For details and examples on how to use MongoDB to store Express sessions, see the documentation for [the connect-mongo package](https://www.npmjs.com/package/connect-mongo) [_\(https://www.npmjs.com/package/connect-mongo\)_](https://www.npmjs.com/package/connect-mongo).

Summary

In this exploration, we studied cookies and sessions. Cookies enable us to overcome the limitations of HTTP protocol being stateless. Use of cookies enables use of sessions which are a major feature used in server-side web development. Note that HTML5 has additional support for storing

data on the client-side using **localStorage** [_\(https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage\)_](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage) and **sessionStorage** [_\(https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage\)_](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage). But we are not going to discuss these topics in this course.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- In its overview of HTTP, MDN describes how the use of cookies means that even though HTTP is **stateless, it is not session-less** [_\(https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_is_stateless_but_not_sessionless\)_](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_is_stateless_but_not_sessionless).
- Cookies and their usage is discussed on **MDN here** [_\(https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies\)_](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies).
- Documentation of the npm package **cookie-parser** [_\(https://www.npmjs.com/package/cookie-parser\)_](https://www.npmjs.com/package/cookie-parser) and **express-session** [_\(https://www.npmjs.com/package/express-session\)_](https://www.npmjs.com/package/express-session).