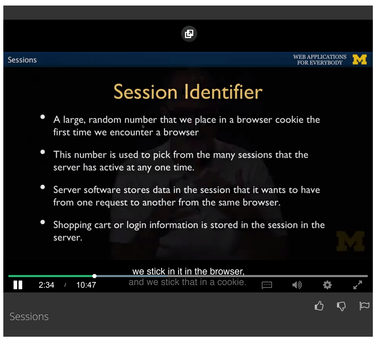
Lecture Materials

Assignment

Bonus Materials



Have a question? Discuss this lecture in the week forums.

Interactive Transcript

English

0:08
Okay, so now we're going to talk about PHP sessions. Now they're related to cookies, but you really have to separate them. So cookies are a browser concept and an HTTP concept. So, just to review, the server sends a cookie, and then it's stored in here. And then the browser sends it back. X equals four. We saw the set cookie request in the cookie header, okay? Now, sessions are data that we store in the server. So, it wouldn't store the current account balance or even the current logged in user at least unencrypted in a cookie, because this is part of the browser, and you can change it, right? You can actually change it. You can delete it. You can throw them away. And I'll show you how that works in some of the demonstration videos where you throw these things away. But what we can use is this cookie to effectively unlock a session. And a session is a little bit of data stored generally in the web server. Although you can store it externally in all kinds of servers depending, for more advanced things. And it can even be stored in the database, but for now, we'll just take it the simple way. It's not in the database. It's a little tiny file that lives on the server. And so just like when this request is coming in, we have Get Post.

1:20
And the GET and the POST come from forms and parameters. We have cookie.

1:26
Once we figure everything out, this data here is going to be session.

1:33
Now, what's cool about session is this is a two-way connection. Cookies are sort of a two-way. We can use set cookie to send it as well. But session is data that's really easy. And it's another super global. So now we're going to talk about, given what we know about cookies, how we implement and use sessions in PHP. So, when we meet a new browser, we check to see if there is a session cookie. It's still a cookie, but a session cookie. And if there is no session cookie, we make up a large random number, some unique mark, and then we send it as a cookie, and then we create a session with that same identifier. And then we get the next request, we see, there's a session cookie here. And then we reconnect to that session, and away we go. And so the PHP takes care of a lot of the session stuff for us, like a web framework. And so the session identifier is security by obscurity. We pick a large random number, turn it into hex characters, and we stick in it in the browser, and we stick that in a cookie. Now, if you somehow compromise your session identifier and lose the session identifier, then someone could fake the session identifier because they could change the cookie in their browser and take over your session. But because these are such large random numbers, we generally don't worry too much about that. And they only live for a little while, so when you close your browser that session is gone and it's gone on the server as well. So we tend to store not too much data in the session, but if you take a look and you took a typical application, especially a php application and you go to it for the very first time, you'll see this cookie. Now you can change the name of this cookie in the configuration of your PHP. By default the cookie is named php session ID. And that is coming back from the browser on the first page, long before you press the login.on anything. And so there's a session. And the login is different. We'll talk about login later. The session is just a place to store data, that is a two-way place to store data.

3:34
And so now, this is space, this is not time. Now what we end up with is a browser that has a session cookie with some large random number. A different browser dot, dot, dot more browsers, different users, different cookie, da da da da different cookie. And inside the server we have a little storage that corresponds to each of those things. So that when we see an request coming in, we can reassociate using the cookie. We can then look up among all the sessions, and pick the right session, and hand them in to PHP as $_session, okay? So the cookie chooses the session for us.

4:16
So, PHP has good support for sessions built in. We basically establish or create a new PHP session by calling session_start before any output. And you'll see a lot of things we'll talk about upcoming, that you've got to do them before any output. This is another word for in the model, so it's in the model code, not the view code.

4:38
If the user has cookies, we can then use the session variable to store from one request to another. So this is the two-way variable, you can take stuff out of it and you can put stuff back in. And so now we have a bit of stuff that you probably wish, why can't I keep that post data. Well a post data manages and is recreated on every request response cycle, but the session is not, it's reassociated. So you have a cookie that reassociates the session, and the default place that we store these things is on disk. So now we have a time situation where once you got the session established, the cookie comes in, it pulls the session in to the session variable. If in your data you change it, then that data is written back up to the disk and then this response comes out. Now, sometime later, we do another thing that creates a GET request with some GET parameters. And then we pull the session data in, and then we can make changes to the session data, and then we send the response. Time passes, another post comes in. It makes post data, so the post is created each time. Different post data, we pull the session data out. And then if we make changes we put the session data back, send the request, time passes, you get the idea. Comes in, got this a post request, we get new post data. But then we pull the session data into session, we make changes in here, get sent back, and the response happens. So this is sort of it's like global variable across a series of requests response cycles. And it is a key value array. You get to pick the keys. You get to do whatever. Now, one thing we don't do put gigantic stuff in session. We use it sort of like little bread crumbs for our application like logging in and logging out of something we're going to store in this session area.

6:21
You can find out where your server is configured by PHPInfo. You can find out where it's storing the sessions at, and you can even go look at the sessions. And there you go, you'll say, look, here we go. This is a little file and it's keyed by the session ID. So when the request comes in, it goes and find this little file. And if you were to look at this, they're not guaranteed to be viewable but this happens to be viewable. It's kind of a series of key value pairs in here and says it's an integer. This is sort of a magic format that we're not supposed to look at. But it's the kind of stuff that's being stored into and out of the session variable is stored on disk.

7:00
Now some people debug their applications by looking at these things, and I've done that from time to time rather than printing it out. So the way this works is somewhere very early in your application before any output has come out, you say session start. And session_start does one of two things. If the cookie doesn't exist, it creates the session, an empty session, and sets the cookie. If the cookie does exist, it reassociates the old session. Whichever the case is, once you're done $_session can be used. So you can't use session until you call session_start. The GET and POST and the cookies are set before the first line executes. The session is only set after you call session_start.

7:45
You can wipe out all the keys with session destroy, and then you simply store stuff in session by using assignment statements. So here's a little bit of code that you can run and play with. And so here we go. Comes in and we're going to call session_start, right? And you can't have any output there, but this is kind of in our model view controller. This is sort of the split here. What we're doing sort of the stuff that you can't see is done here. But we're looking at session data, so remember the session data is coming in and going out of this application. So if there is a key of pizza in session, we print that session is empty. And then we say session sub pizza is 0, so we're storing it in the session. And then we follow through and we print out session. And then the next time it comes in session starts. Well pizza is now equal to 0. That's in the session. It's outside. And once we say session_start, this is pulled into our $_session. So this becomes false because there is a pizza key in there. And so then it says if $_SESSION['pizza'] is less than 3, we're going to take the old value, the 0, add 1 to it and store a 1 in it. And we're going to put a 1 in here. And then it finishes and does the thing. But now time passes and there is a 1 sitting in here.

9:04
It comes back, pizza equals 1. So comes in, it's set, it's less than 3, we add it, it becomes 2, runs, comes in again, it's 4, etc, etc, etc. And you get the idea. The point is is that in this statement right here, you are reading and writing from the session which is outside this and it's not in the browser. The session is in the server. Somehow the server is responsible for it. The cookie unlocks the session, the cookie reassociates the session, but the cookie is not the session. So that's X equals 4 key value pairs stored in the server. Okay, and it's a super global named it $_SESSION). It's established. So if you were to print session out here, it doesn't exist but then once you type this then it does exist. So the session_start reads the stuff off disk. because you can have some of your PHP that doesn't use the session wrong and you don't have to call the session_start if you're not going to use the $_SESSION variable, you don't do that. So here's what happens when you click it. Creates the session ID the first time, and then sets it to 0. And then you keep doing it, it goes 1, 2, 3, and then it calls the reset and then it empties it out. That's this code here is if this gets to be 3 or larger, it calls session destroy and then restarts the session, and then runs and does the thing. And so that all session destroy, it doesn't change the session ID. It's the same session ID, but it does delete all the keys inside of it and so it empties the session out. So the next thing I want to talk about is how sessions can work without cookies. because this I think a really cool feature of PHP.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?