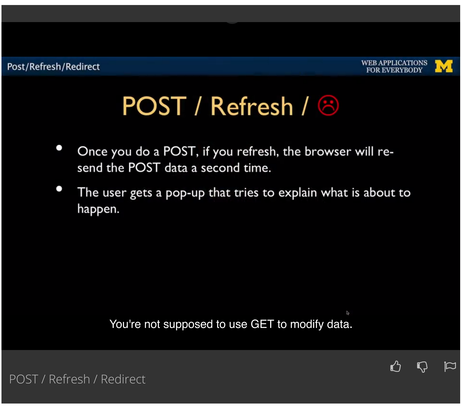
Lecture Content

Assignments

Bonus Materials



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

Interactive Transcript

English

0:08
Now that you understand the mechanics of redirect and how it generally works mechanically, now we're going to start talking about why we might use it. One of the things we talked a bunch about is, when you use GET and when you use POST. You're not supposed to use GET to modify data. You're supposed to use POST to modify data. The problem is, if you go and you POST a screen and then you get it back in another screen and POST. And, literally, every bit of code I've shown you that does POST are doing this wrong up until now. I'm sorry. Now I'm going to show you the right way to do it. And you are going to know about Redirect. You need to do POST, what's called POST redirect. The problem is, if you do a POST to a screen, you hit the Submit button and then you hit the Refresh button, it's like, do you want me to send that POST again? And what if you made a $100 payment? And you hit the POST again and you make a second $100 payment. And, you hit refresh and the browser goes like, time out. I don't want to do this because I know that POSTs are dangerous and it could be that you are giving money away or whatever. And so, they get a little pop up. And that pop up is often kind of badly worded. And, it's not something that you control. This is something the browser is doing to avoid double posting the same set of form data. Okay? And it just means that we don't want to do a POST and we don't ever want to return HTML for the POST. So if the pattern here is, you have a form, it's method equals POST, you press the Submit, that does a POST and it gives me back a page. And then I get this page, after I hit refresh on the page that came back, it says this is the browser talking, the page you're looking for had information entered and you might cause action to be repeated. It's trying to explain to you that you don't want to do this. Everyone puts Cancel, but then, what are we supposed to do next? And that's because, we had HTML returned on a POST. Meaning that you pretty much are supposed to do a redirect after a POST and return no HTML. And then, POST redirect back a GET, that actually returns the HTML. We end up with a POST-Redirect-GET, is the pattern. Let's talk about how we're going to avoid these double POSTs. Like I said, POSTs are adding and modifying, GETs are looking at stuff. You don't want to do the same POST. It's really a bad user experience because you aren't controlling it as the developer. It's just bad in general. It's really very tacky. And literally, every POST code I've shown you in this class so far, has been wrong, But, not anymore. The simple rule as I've said, is you're never supposed to generate HTML content when you receive a POST. And you're supposed to redirect. Usually, we redirect right back to the same script. You do POST redirect. And, here's a sort of a picture of how it works. You're in a form, that's a POST, you Submit it. It does something. Updates the database and then it sends back a screen. But, this is a POST. The response to the POST, you hit refresh. The user refreshes the page and then it asks the questions, like, are you sure you want to do this? Because, if you're going to really do it and you say yes, then it's going to send that same data twice and then do whatever it did in the database twice. That's not the way we're supposed to do it. What we're supposed to do is we fill out the form, we do a POST, the data gets updated, once, then, you send a redirect and then silently, blink, blink, blink. It does a GET back to the same page. And then, the confirmation page is the result of the GET. This is not a POST, that's a GET. Now, if you hit refresh, and you can do this as many times as you want, there is, it's a GET request not to a POST requests. This POST data is not being done twice, this is only done once. Here it's being done twice, three, four. Here, it's only done once. And this is the POST-Redirect-GET pattern. POST-Redirect-GET pattern. Just so, that the refresh ultimately works. Awesome. Okay. So here is bad code. I am so ashamed that I ever showed you this code. But, I had to show you this code because you didn't know what a Redirect was, until like 20 minutes ago. Remember we have the model in the controller. I mean the model in the view. The Controller sort of, we're going to learn more about controllers, like orchestrating it all. And, we have this thing called the context, remember? A context falls through. We put stuff into the $guess and $message. We have sort of no logic down here. We start the code and we look at $message and $guess properly using HTML entities, because we're nice people. That's a sort of model view controller but, it's not a model view controller that properly does the POST redirect. Okay. And so we're using just a falling through of this message. Okay. But it's bad, because if you hit refresh on this, who knows if it will actually do it, depending on the browser. But you may get that error message. Okay. Here we have to come up with a way to pass this context information from the model to the view, because remember, that $_GET and $_POST are gone in between requests. In comes a POST request, you've got $_POST. A later GET request is not going to have that data. We have to save it. Well, where are we going to save it? Sessions to the rescue. We start our session, which basically, sets a cookie. We talked about all that before. Set's a cookie, has a little bit of data on the server. A little data for our little session. And then, when we do all our stuff, the thing that we want to pass down to the context, we're going to put in the session. We're going to have the SESSION['guess']. We have guess=42. And then, we're going to do our tests and we're going to have a message= blah blah blah. In the session, remember that you can write to the session. We're writing from the session. So, we've put the context, the information that we want to pass down to the view. We've put it in session and then we Redirect right back to ourselves. Right back to ourselves. And then, we return and go out. We turn and this one is done. That request response cycle is done. Then what happens is, a GET request comes in. The second request response cycle starts. Now, that's a GET request. There is no POST data. The immediate one before that had POST data. But, we start the session. And look what we have in session? That pulls all that stuff into session. It goes straight down to here, because there is no POST data and the session is ready for us to use down in the view. Two request response cycles. POST-Redirect-GET. Then, in we come. This is a GET request now. See the first GET request or our GET request that happened right after a POST. We check to see is there something in the session? If it is, that's what we're going to put out as the message. The guess, old guess and the message, the same way. We're going to put a false, a blank and then, we're just going to do our view. At this point we just do the view. We just pulled this stuff out of session. We use session momentarily to hold the old guess and the old message to pass from a POST to session to Redirect to GET. And then, it pulled it back out of the session. That's what we just did. POST-Redirect-GET. And, we had data in the POST that we want it to pass to the GET but this is a second request response cycle and that's a first request response cycle. Oh yeah. That's pretty darn cool. Pretty darn cool. We are going to use this like all day long. We're going to even name it the flash pattern, where you put a flash message in session. But we have to delete that first. And again, I warned you that maybe your browser won't show both of these but you can see how we'll do a Submit. The response comes back 302. And that says, go immediately retrieve a guess, but we have passed the session data between these two things. And, that's where these two things come from. We've processed the POST. We put our data in a session, then we send the Redirect. And then, we read the data from the session on the success of GET. And then, this is a real 200 with the real HTML that makes this page. This is finally getting fun. This is finally getting fun. You're learning like, awesome POST Redirect stuff. And then of course, if you press Refresh, you don't get any bad error message. And it's a GET request and it gives you 200. And, it gives you back the stuff. Why? Because it pulled it out of session conveniently. It was still there. Those two key values are sitting in session and still there. And here I have my favorite picture in the whole world. We have a POST, we have the form. We do a POST. The POST comes in and then it takes the data and puts it in the session. And then, it immediately sends a Redirect. And then, the browser does another GET request. And then, it pulls the data out of the session. And then, it produces the HTML with the old guess and the message. And, it sends that back to the browser, which passes the response and puts it into the DOM and then you see it. And then, if you press refresh, it sends a GET request in. Which then, pulls the same data out of session, produces the same HTML and then the response comes back to you. And that's POST-Redirect-GET-Refresh. Now you got it. So, now that we got this all figured out, we're going to do login and logout using session. Understand that just having a session is not the same as being logged in or logged out.

Downloads

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

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