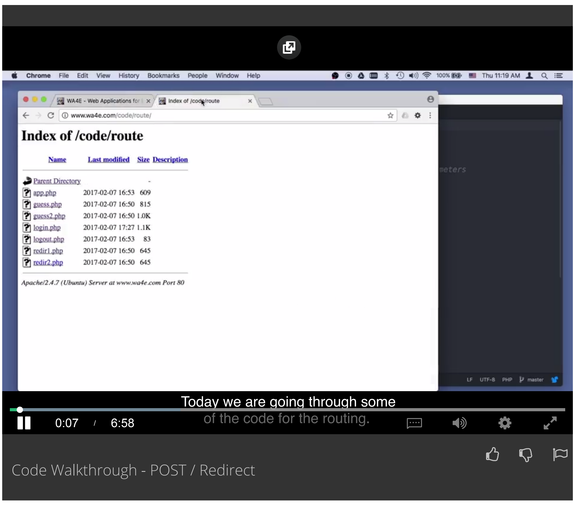
Lecture Content

Assignments

Bonus Materials



Code Walkthrough - POST / Redirect

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

Interactive Transcript

English

0:00
Hello, and welcome to Web Applications for Everybody. Today we are going through some of the code for the routing. We've done some redirecting, and now I want to play with this code, guess.php. And so this guess.php code. This is the code that I used to introduce model view controller. So the model part here is the silent data processing bit, and the view is the part below it with a few little bits and pieces being injected. Kind of an HTML template with some dynamic bits in it. And then the context sort of falls from this top part to this bottom part. But I want to show you a flaw in this. So if I do 41, and it's too low, that's fine. And I do 42, and it's just right.

0:43
And I'm really happy. The problem is, is that I'm sitting on a POST. The last thing I sent was a POST. And if I hit Refresh, because POSTs are considered by the browser to be expecting to modify data. If you were going to decrement your savings account, you would do it in a POST. The browser doesn't want to resend the POST request for you, just without your knowledge. Now, this is not coming from my application. This is coming from my browser, that's keeping me from doing something stupid, right? It says, you might be decrementing your bank account balance, or transferring $100 twice, or something, right? So it has run that I have to do this. We as application developers, we have lost control of the user experience at this point. And so that's pretty tacky, and we're not very happy about that. And so, there is a way to fix that, okay? And that is to never generate output on POST. And you can say POST, you can Google POST Redirect GET. And you will see some Wikipedia pages, which I love so much, And I even use this in my lecture. It basically says, the problem with the POST, and then at 200 it comes back. And then you hit Refresh, and it sends the POST again to generate the page, and that's the dangerous moment, right? So what we want to do, is we want to do a POST, and then we do the work, and then we Redirect back to ourselves with a GET request. And then we put the actual page out on the GET request. And then if you Refresh it, it's doing the GET request over and over, and so that's all cool. The problem is, is what if we want to put a message out on this screen, and we're generating the message here in this POST code? Success, or guessed too low, or whatever. What we're going to do, is we're going to use the session [COUGH] to copy the data from this moment to that moment, okay? We're going to look at the session to copy from this to this. So we have to use the session to get the message. because otherwise, what we were doing, is we're just putting the too low message out right here. But we've gotta do a POST redirect. So we're going to know the too low message here, and we're going to send it in the session to the next one. And it's going to pull it out, and then print it, okay? So, this is the bad one. It falls through, produces output as a result of the POST. Here is the good one. Now, one of first things we see in the good one, is we have to use the session, because we're going to pass data.

3:07
Pretty much almost everything, except this little bit right here, is the same. All the HTML entities, and all that stuff is the same.

3:16
And, let's go to guess2.php. Okay, now the problem is, is that this data up here in the model part has the old guess, and the message. And it wants to communicate it, but it's going to come through here, and then come through again. So the POST data is gone, the GET data is gone, but the session data is not. So one of the things we do, is get the context. And that is the data we pass from the model to the controller, get the context into the session. So if we have a POST of guess, we calculate the guess, and then we stick the old guess into the session with the key guess. because remember, you can write this. [COUGH] You can put stuff into the session, and it just stores it for later. And then we do the old logic, and we say a $_SESSION['message'] is the high, low, great job. And then we redirect back to ourselves. And that causes the browser to immediately grab and get a new copy of us. So it comes down, but this is a GET request. So this was a POST request, but we say don't do it, right? And we're right here in this picture.

4:25
We're sending back the answer to the POST request, which is, do a GET request. Which immediately turns into a GET request to the same page, right? So it's going to come back in, POST is no longer defined, but session is defined. So we look down here, we go like okay, well let's grab the old guess. If there's a guess in the session, we'll grab that. And if there's a message in the session, we'll grab that. Oops, that should say false right there. I don't quite know why it doesn't say false right there. So we'll save that. So if ($message !== false) we print it out, and then we print the old guest. But we pulled them out of the session, because we're in a second request response cycle. The second request response cycle. So we pull the data out of the session to produce that page. So let's do it. So let's do a guess of 41. Now, let's do View > Developer > Console. Okay, so look at the Network. So we're going to do a POST to guess2, and there you go. A POST to guess2, and we send the guess in, 41. And we got back a response of 302, which says it's not really a page, it's something else. And the something else sits here, and the Location says go back to guess2. So we're redirecting back to the exact same script. And so the browser immediately grabs that script, and it sends a GET request to it. And that GET request comes down here, grabs the old guess out of $_SESSION, grabs the old message out of $_SESSION, and then it renders this. And so the actual output is here, to the GET request. If we look at the output of the POST request, there is nothing, and that's because up here. After we did the redirect, we returned and got the heck out of there, so we didn't follow through. We kind of caused us to come back by saying, come back! Come back to the same script again, please. And away it goes. And so, there you go. That's sort of a POST Redirect implementation. And the pattern will be, when you have POST data, you eventually have to put things in the session, do a redirect, and return. And so we'll see more complex code, but this pattern of set the session up, do a redirect, return, will be done over, and over, and over again. So that we can follow the best practice rules of POST Redirect GET. So, I hope you found this helpful. [MUSIC]

Downloads

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

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