Pragati Sharma: Accessibility Automation Testing

[00:00:00] Hi, everyone! Thanks for joining in for this session. So in this session, we are going to talk all about accessibility. But before getting started, let me just give you a brief introduction about myself. Hi, everybody! I'm Pragati Sharma and I hail from Delhi, which is the capital of India. I've been working in the corporate industry for the past 4.5 years, no a little more than that as a quality engineer/quality consultant. Currently I'm working with Thoughtworks as a senior quality consultant. So in my entire tenure, I have had the chance and the opportunity to work on different facets and different types of testing and automation. So I've done the web automation. I have done web services automation be it REST APIs, GraphQL, and then nonfactional automation as well that comprises of performance and (unintelligible) accessibility. So I'll be very honest, so when I started working on accessability, I clearly understood that it is an underdog because we do not talk much about it. But it is so very important for us as engineers, as a development team, for the company, for the organization, and for the clients and customers. So I guess it's the right time for us to pay heed and attention to accessibility testing as well, and this is what this session is all about.

[00:01:24] So let me first get you well versed with what is accessability and why do we need it. So starting off with device, so whenever we talk about something being accessible, it simply means that it is easily understandable and easily obtainable. So if I say that I want to make a platform which is accessible, it would simply mean that everybody around the world should be able to access it with or without disabilities. Also, 20 percent of the world population is disabled, so that makes a huge part of the entire user base and we definitely need to be cognizant about their needs as well. Let me just give you a very relatable example of how accessibility is not only targeted to a specific group, but on a holistic level it increases the user experience by mighty fold.

[00:02:16] So you would have seen wherever we have staircases, we have a ramp adjacent to that. So the primary motive of that ramp is for people who travel while wheelchairs. But whenever you go to subways, you would see that more than 50 percent of the population is using ramp instead of the staircases going up and going down. So why is that happening? So the ramp which was actually made to solve an accessibility pain point essentially increased the user experience and people are more happier using the ramp instead of the stairs because using stairs is a lot of effort for them.

[00:02:56] So in a similar fashion, let's jump onto the other aspect of it, which is why do we need accessibility? Again for this I would give you a real world example. So there is a very famous streaming company. I would not be taking the name. But in 2016, a heavy lawsuit was actually filed against this company because it did not have the closed caption. So this lawsuit was filed by the National Association of Deaf because their community and the people who could not hear would actually not able to absorb the content on that streaming website. And because of this, the company had to actually incurred a cost of $750 000. And they were made to sign a contract which said that within two years they need to make hundred percent of the content with closed captioning by incorporating closed caption into

their content. So in this example, Title III of ADA, which is Americans with Disabilities Act was violated because of which this entire lawsuit happened at the first place.

[00:04:03] So this was something which is very specific to America. But then different nations have different federal laws. Supposedly UK, it has its own federal law. It goes by the name of Discriminations Disability Act. Then in India, we have another law, which is the Rights of Persons with Disabilities Act. So now, wherever you are, wherever you're deploying your product, you need to make sure that you are actually abiding to the laws of that particular nationality. And this is what makes accessibility all the more important for us to abide to. As quality analysts, we need to make sure that our products are accessible by one and all and at the same time we need to make sure that we are able to give a faster feedback for any kind of accessibility violation so that the team is able to handle that in a timely fashion. And at the end, it's very important for us to automate this entire process so that the overall regression becomes way more easy for us. So let's dive right in.

[00:05:07] Let's try and understand what the problem statement is. So now what we have in hand is a web application and we need to perform functional automation for that and we need to club that with some accessibility automation. For this, we would be using Cypress as our web automation framework. And Cypress actually provides us a handful of advantages, like one of them being that it runs in the same loop as your application, unlike other testing tools which sit outside your browser, which means that you would definitely be encountering a lot less synchronization issues while you use Cypress. Another thing to notice is that it provides you the native access to the DOM elements. So to all the document object models like different objects, window objects, document objects, HTML, devs all of them would be easily accessible using Cypress. Also, it shows you the hidden shadow elements or shadow DOM elements. They would also be accessible using Cypress. Then Cypress can be actually fall off as a one stop solution for all your automation needs because it also has its own assertions. Now coming all of the other thing, which is the Accessibility Engine called Axe. The beauty of Axe is that you can actually integrate that with any number of different testing tools. Name it and you have it so it can beautifully get integrated with Cypress, Selenium, TestCafe, RestAssured. So it simply means that whatever automation suite you have in place, just on the top of it, add Axe at the accessibility engine. Also, whatever modern day browsers we usually use for testing purposes like Chrome, Firefox, IE, Safari, so all of them can be tested using Axe.

[00:06:58] Now moving ahead. So the application that we would be using is ToDo App. So this is basically an app which gives you the flexibility to create your to do list. So now if I want to say that sanitize your hands. Wash your hands. That's my to do list. And as soon as you complete that, you can strike these things off. So now this is what we are going to automate today. Let me refresh this. For this, I mentioned, we would be using Axe. Let me give you a manual demo of what we are trying to achieve here. For that, we would be needing a plugin called axe-coconut. So this is a Chrome extension, you can simply go and add this. After that in this code web application that you want to test for accessibility issues. And here go to axe-coconut click on analyze and you'll figure out the various accessability vulnerabilities that your application has. Now this application has four. Let me walk you through a few of them and then we can get started with the hands on automation.

[00:08:04] So this one shows that there is a lack of sufficient color contrast. Now think of a person who has a poor vision. For that person this background and foreground seems to

get merged and he or she is not able to read that out properly. So now this is an issue that needs to be solved. For that you'll have to change the color of your foregrounds so that there is sufficient contrast. So that it doesn't become incomprehensible. Now if you see that there are certain texts. So now this is the category of color which is getting violated and these are different guidelines. So now these are the text standards and these are different version and good standards as well. So now you'll have to go ahead and ask your product owners as to what have you complied to? And the testing that we do should be according to that because you'd have to prioritize your issues according to which standard are you complying with?

[00:09:08] Let me talk about some other issue. Now, this issue says that every element should have a lang attribute so lang is the language attribute. And this is a category that is mentioned as well. Another thing that we should understand is why is this issue important? This issue is important because people who cannot see who are visually impaired, so they would not be able to see it, but they would be able to hear. For that they use tools called screen readers. There are various screen readers like in Mac there is an inbuilt screen reader called VoiceOver. Then there are other screen readers like NVDA, JAWS. So you can actually make use of those screen readers and you can navigate through your application and figure out what every element sees by figuring it out. Now, if you do not provide a language attribute the screen reader would never come to know whether things written on this application are an English, German, French, or what is the language? So it becomes all the more important for us to provide this attribute. And in case you want to know more about any issue, you can simply go on learn more and it would give you entire details of how to fix the problem, why does this problem even matter? I'll go back and you'll see that we have impacts (unintelligible). This is a serious issue. This one is a critical issue. The other one is a moderate issue. Then again, we have a serious issue. So this would help you to prioritize your issues in a way.

[00:10:34] Now, we'll try and replicate whatever we have seen here into our automation framework. We'll be using IntelliJ in order for us to make automation suite from scratch so that let's create a new project, an empty project. So select empty project, and I write this as a test-guild. Let's open that in a new window. So this is all we have in an empty project. Now I want to create a package.json, so I'll do npm init. And it would create a package.json to me, so you can simply press return because none of these are like mandatory. So now if you check here, you'll have package.json. Now within this you'll get to see all the dependancy that you've added. Since we're using Cypresss so Cypress is a Node dependency we'll be adding Cypress here. So npm install Cypress.

[00:11:41] After installing Cypress, we would be doing a Cypress open. So instead of npm we would be using npx which is our package runner command. So the best part about Cypress is that it would give you a template like structure, which would be having four different folders. Now, if I will walk you through those four different folders, let me take you onto the slides. So, here, you would be able to see that we have four different folders. So we have fixtures, integrations, plugins and support. So fixtures would have all the data validation files wherein we can have some hard coded data that we can use for assertions. Integration is the area where we would be having our test file. So here is where you put on your test suites and test cases. Then we have plug ins so plug ins are required whenever you want to load your project or whenever the project gets open in case conflict changes are there. So it would be able to come with that support. Support is another folder in

Cypress which would have all your custom commands. So in case you make some commands which are not (uintelligible), you want to make some overall commands you can use support folder. And in case you want to have like project wide hooks, you can put them in support folders as well.

[00:12:59] Let's go back to our empty project. Let's see where it has reached. So after adding Cypress as a dependency, you can see it has gotten added here. Now, like I said, we have to do an npx cypress open and it would create a template like structure for us.

[00:13:19] Okay, now let me take you back. Now, here there is a Cypress folder within which we have four different subfolder. So within integration, I'll create a new file with the name ToDo.spec.js. And here we add our test cases. Now, we'll use the describe, so what are we doing is we are actually testing ToDo application.

[00:13:52] Okay so now this becomes the test suite. So now what we're going to do is we are going to add a test case. What we are basically doing is testing the page load. So I add an ID and I'll call the test Test Page Load. And within this, we can actually visit that URL that we saw. So we'll visit our URL. So we visited this URL so I'll fetch it from here you and I'll put it in my test case. As a part of the assertion since I talked about we have inbuilt assertions here so we can actually check that the URL that we visited is correct or not. So here is a should operator, and we can use the keyword called equals. And then we can just compare it with the URL, right? This is what we have done, let's try and test if this works. For running it, we'll use package runner cypress open again. Now we have ToDo spec here and let's try and run it. And the assertion has got pass. Awesome.

[00:15:21] Let's go back and now the next thing that we need to do is to add accessability on the top of it. So now for accessability, I'll have to add certain dependencies. I'll be adding the dependency called cypress-axe. So now this is a dependency that would allow the Cypress Node server to access the axe APS and then internally the axe runner is actually using another dependency called axe-core, so I'll have to install that as well. So it's axe-core. Now, let's check if the dependencies have been added. So now we have axe-core and cypress-axe as well.

[00:16:08] Now, what we want to do is we need to use cypress-axe in the entire project. Now for something to be used in the entire project we need to project wide hooks. And that goes under rest support folder. So in this support folder, I'll simply do require, what do I require is cypress-axe. That's it. Now I'll go to my specification file and here I'll have to use teo commands. Let me tell you more about those two commands - cy.injectAxe. It would extract their DOM. So you'll have to do this before every test case of yours and before every test case we'll have to extract all the document object model so that we can access all the elements, whether it is a button or dev, an input, anything. Then the other command is checkA11y. So checkA11y is the short form of check accessability. So as soon as you run this query, it would run on accessability check and it would give you the vulnerabilities. Now let's use that in our code. So here I'll have to put before each hook. So beforeEach...what do we have to do? We'll have to inject the Axe, so injectAxe is what we'll do. And then I'll write another test case where we're checking accessability for page load. Now we'll check this. And here I'll use cy.checkA11y. That's it.

[00:18:00] Let's see what happens now. I'll run it again npx cypress open. Now let's run the ToDo spec. So this was successful because assertion got pass the other test case says check accessibility for page load. And now we have two accessibility violations. Now if you would have noticed we had four accessibility violations when we did this using our Chrome extension. But here we have just two. So now this is an ongoing issue that you might feel when you use cypress-axe that some of the elements are not getting exploited using cypress-axe.

[00:18:43] Another solution to this is another dependency called better-cypress-axe. So let's see if that solves the problem. I'll do an npm install better-cypress-axe. The only change I have to make is in the support folder. As soon as I do this, better-cypress-axe is ready to be used and then let's run our file again and let's see if we are able to find all the four violations. Test page load is successful, and now we have four accessability violations.

[00:19:20] Now we have understood all of this. The other thing is that how can we customize the usage attacks? Now, this is the project that we made but I have a similar project already created, which is more exhaustive. It has more test cases. So I'll take you to that. So it has ToDo spec. So this is all that we did. So we tested the page load. We did the accessibility violation check. But now there is another option wherein you can actually customize your accessibility test cases like in case you want to filter out using particular standards, like we talked about different tags so you can have certain tags specified here according to your product needs and product demands, and you can only find out those issues which are violating particular standards like these. So in case you are in the United States, Section508 is very essential there. So then you can just find out what are all the violations around Section508 by using this run only command. So this is a kind of custom parameterization that you can do for check accessibility function.

[00:20:34] Similarly, in case you want to find out issues with specific impact areas. Now, in case you want to prioritize that in this print, I'll just be picking up critical issues. Then we'll pick up severe and then moderate. So in case you want to go with that ideology, you can use your test cases in a similar fashion. And then as you go along doing your functional scenarios, you're opening another page, you're opening a frame, you're clicking on a button. After performing every functional scenario, you need to perform an accessibility on the top of it. And this is what my test suit is doing. So let me just run this for you so that you understand this ToDo spec. And just focus on the right frame, what's happening here, because we are actually performing a lot of functional things as well.

[00:21:28] So what we are doing is we are adding certain items to the list and then we are striking them off after getting them completed. So, like you can see here, we are doing accessability check after performing every functional action. After adding, we are again doing an accessibility check and then after checking out the item, we are again doing the accessibility check so as to see what violations we are getting. And if you see here, it gives you a number of instances also. So there could be just one issue, but it could be on four different elements. So now this takes your count to four plus three, which is seven plus one eight nine. So it has nine issues in total, all four different types. So that is how we actually figure out and categorize our different accessibility violations.

[00:22:26] So all in all, this is what we do in order to create an automation framework from scratch, and you can always create a robust framework. You can add as many test cases as you want. So like this one was for the start of this, but as you go along your journey, you will be able to figure out how and where to add more accessibility cases for you to figure out what could be the possible gaps that might lead to origination of different accessibility violations.

[00:23:03] So we have made to the end of the session. Let me just wrap this up by giving you some key takeaways. So today we learned how to make an automation suite on Cypress from scratch and how to integrate that with accessibility engine, which is Axe, and you can primarily use Axe with any of your existing automation test suites as well. And I promise you that it will work seamlessly and in no time you'll be able to figure out where are the accessibility gaps. And that will definitely give the team a better head start in terms of making that application robust and accessible. So I'm pretty hopeful that you gain some knowledge out of this session. In case you have any questions, queries or feedbacks regarding that feel free to shoot that up. Thank you, guys.