**# SBSAutomationTests:**

1. **In this project, I have reported results in Allure HTML report which will be opened by default in the browser when execution is completed and also saved in temporary folder. There is currently an issue going on with the Allure Command line tool (allure generate command creating blank report) and hence HTML report is not being generated at designated path from the generated xml files of the report. But I manage to save HTML report in temporary path folder and open by default in the browser via 'allure serve' command. Click 'Ctrl+C' in the terminal after HTML report is generated. I confirmed on Stack Overflow that this issue is global at this time**. Xml files of the report are saved in allure-results folder. HTML report is having graphs, details, dashboard, timings and logs generated at various points via 'allureReporter.addStep("message")' method

2. I have created test suite with the name 'sbsAutomationTestSuite' containing both files (sbsAPITests.js and sbsFrontendTests.js) in wdio.conf.js file. Then I have provided its entry in 'script' section of package.json file. Hence, Test suite can be run with command 'npm run automationTestSuite'. Both test cases are run in parallel in Chrome. Run config can be changed via wdio.conf.js file

3. Mocha tests are under test->specs->sbsAPITests.js (API test case) and test->specs->sbsFrontendTests.js (frontend tests) files

4. Screenshots in the HTML report will be provided if test case gets failed. In order to achieve this, I have updated afterTest and onComplete hooks in wdio.conf.js file. Make sure that you have correct execution policy to access script.p1 like powershell files may be from Node. If not then run command 'Set-ExecutionPolicy RemoteSigned' after opening Windows Powershell as an administrator

5. Pages (.js files) of front-end application are created in test->frontendPageObjects folder. Pages of ‘Get API’ are created in test->apiPages folder. Pages have class which contains getter fields (elements) and methods. Methods and fields of pages are called from tests via creating objects of page classes

6. Install all the packages present in package.json file (Installing node.js is pre-requisite)

7. Test data for front-end tests is in class that resides in test->TestData->frontendTestData.js file. Test data for API tests is in class that resides in test->TestData->apiTestData.js file. Test data is called from tests via creating objects of test data classes

8. Front end Audio test not only validates what is there in the PDF but also validates progression of audio at various intervals

9. Screenshot of the generated HTML report:

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated