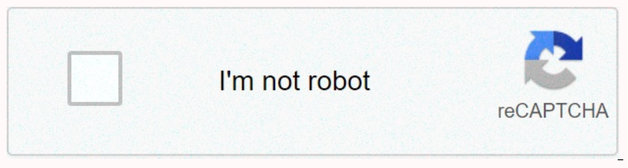

Continue

Cypress reporter example

Hello, everyone. Today I will show you how to create a beautiful test report by performing tests on Cypress.io. As you probably know Cypress is built over Mocha, so most of the mocha features can be used with Cypress ie bdd Mocha syntax, hooks or even reporting section. By default, Mocha provides multiple journalists (more than 9 journalists) that you can select one (or more) of them as your own test report. The configuration of Mocha's journalist in Cypress.io is so simple, just set the value of the journalist's parameter in the cypress.json file. First of all, let me show you some built-in Mocha journalists and how to use it with Cypress.io1. SpecSpec journalist is a default journalist for Mocha. So if you don't set up any journalist for your project, then you'll see this as your test report. The spec reporter displays the test results for TTY with ANSI-color like thatSpec Reporter To configure the reporter spec on Cypress.io, set the reporter parameter as spec2. Dot Matrix If you want to see some progress while your tests are running, you can choose this. Dot Matrix reporter is a series of Dot. characters representing the progress of the tests performed. The failure of the tests is marked with red exclamation marks! , pending tests or skipped tests are marked with a blue comma, to configure the Dot Matrix reporter in Cypress, specify the journalist parameter as dot3. Nyan If you think the two journalists above are very boring, what about it? The New York reporter could be the one you're waiting for. It's a cute cat running on your TTY while your tests are running (=-y-=)There are more built-in mocha journalists you can choose from. Please find more information on this page to create a beautiful Mocha journalist with MochawesomeMochawesome is a custom mocha journalist who creates an offline HTML/CSS report to visualize your test results. Displays the step-by-step test case and stack detection for failed tests. Additionally, it supports adding your own custom context to the report. This means that you can add screenshots to failed tests, for example. It's as awesome as his name. Step 1: Install Mochawesomemocha is used as a dependency library to create a custom reporter.mochawesome creates a beautiful exposure in HTML format.mochawesome-merge Since Cypress v3.0.0, runs each spec separately, which leads to the creation of multiple mochawesome reports (one for each spec) So we need to use this library to merge all the reports and then create it as an HTML file. Step 2: Set mochawesome as a Cypress ReporterConfigure reporter as a mochawesome and you can also add some options for the reporter. The parameters of the journalist's fluctuations depend on the journalist you have specified. Here are some options for Mochawesome reporter.reportDir: To specify the output directory for the HTML report. You can place it anywhere you want.overwrite: To replace existing report files. In this case, since cypress runs each test spec file separately, separately, I have to configure this parameter to false to avoid replacing existing reference files.html: To create html report after testing (spec) run. I plan to create a single HTML report after all my spec files are finished. So I set this parameter to falsejson: To create JSON report after the test (spec) runs. Yes I want this because I want these files (one per spec) to merge all my tests running into a single HTML report. So I set this parameter to truetimestamp: To append a timestamp to a specified format in the report file name. I prefer this option rather than the default (The default is just an operation number like 001, 002) You can find more options for the mochawesome journalist here3. Create your own Cypress Custom Test RunnerNormally, you can run tests on Cypress with a cypress run command for any simple use case. But in this case, I want to do something before and after all my tests. So I need to create my own Custom Test Runner using Cypress Module APIInstall dependencies for Custom Test Runneryargs: To analyze the argument from command-line options. This is a optional.ls: To list all existing reference.rimpaf files: To delete all existing report files before viewing the test. Thus, the report is created after all tests are performed through the generateReport function. Gotcha! This is the HTML mochawesome report. You can see the details of your test cases, such as the test code, the test run time (for each test and all tests) and the Test Filtering menu on the left side. It's pretty cool.4. Attach screenshots to the report on failed testsAcheons previously, you can add your own custom frame to the journalist mochawesome. So I'm going to attach screenshots to the failed tests in my report. In fact, Cypress automatically captures screenshot when a failure occurs during the test run. So I just add screenshot files to the report and that's it. I hear the test:after:run event to attach screenshots to the report when the failure occurs through Cypress Support FileaddContext is an object provided by Mochawesome that allows you to add any custom frame to the report. Not only a screenshot file but also anything you want like series, object, etc. You can read more details about this API here. Then I will update screenshotPlassed parameter in cypress.json to place my screenshots in the list of mochawesome assets. I want to make sure that the My report may be correctly referenced in the screenshot files. I create a new test case that always fails at run time. After that, I run the test again. Finally, the test report is created similar to this. The screenshot is attached to the report nicely. Nice! Woohoo. The screenshot is attached to the report! You can see my full test report here. Finally, you can apply this technique to the continuous integration system. It helps you visualize the test results in a beautiful form. Your team members can clearly read and understand the test report. I hope this article will be useful for your work. Happy test! Thanks to the cool lib: Import E2E test as part of your continuous integration process can be challenging, but it also provides tons of benefits for the quality and scalability of your application. Let's delve into this topic, starting with learning how to create beautiful test reports for your customers with Cypress, using simple examples based on a real project. In this article, which is the 1st part of a larger series on E2E testing and continuous integration, I'm going to go over the reporting process for a simple project using the Cypress tool and a simple project example (warehouse included). Stay tuned for the rest of the articles, in which I'm going to talk about the test configuration with Circle CI, Bitbucket Pipelines and the Slack messaging platform. Reports have always been an important aspect of testing and creating useful reports is something of a trend in QA. Everyone would agree that what customers really want to know is the true state of their system, not how the tests themselves work. With proper reporting, we can tell them if the software is working properly, what errors were discovered, and how they affected the system. We can also determine the duration of each test and the very specific point on the code base that causes a test to fail. Key packages I'll show you now how to set up mochawesome reports. Let's start by installing all the necessary packages. A good tip is not to use mocha: last command. The latest version is usually unstable and can have a negative impact on the reports that are generated. mochawesome:5.2.0, mochawesome: 4.1.0, mochawesome-merge: 2.0.1, mochawesome-report-generator: 4.0.1Formation of exposure to cypress.jsonNow, let's add the configuration of our journalist to the cypress.json file. As before, I avoided journalist: mochawesome due to stability problems. reporter: ./node_modules/mochawesome/src/mochawesome.js, reporterOptions: { reportDir: cypress/reports/separate-reports, overwrite: false, html: false, json: true } Let's explain what is happening here:reportDir – it's the directory to which we're goe to output the results of our tests. The replacement flag – alternates the rule that allows/prohibits the replacement of previous references. The html flag – creates a report when a test is completed. The json flag - creates a json file for each completed test. Scenario configuration in package.jsonIf there was only one file with spec tests in the project, we could just set the html flag to the truth and that would be enough for a fully functional journalist. Unfortunately, if you were to run tests on some files, the report will be created for each of them individually. Since time and convenience are important considerations for us, we want all reports to be included in one file. There is a fairly simple way to solve this problem and get a proper, fully fully Journalist. net exposures:rm -rf cypress/reports, test: npx cypress run, merge-report: npx mochawesome-merge --reportDir cypress/reports/separate-reports cypress/reports/full_report.json, generate-report: npx mochawesome-report-generator --reportDir cypress/reports/reports/full_report.json, after:tests: npm run merge-report; npm runs create-report, cypress: npm runs clean reports; npm execution test; npm runs after:clean:reports tests - removes the directory that contains references before performing new tests (a script can be added to create timestamp reports; with this, there will be no need to remove the directory anymore. otherwise, the report will include data from previously performed tests). tests - starts tests via Cypress. report:merge-report – combines all json reports generated for each test file into one report. report:generate-report – creates a full HTML report from the merged json file. The review of the report Now we have the report, let us take a closer look at the data it provides. Right in the top row, there's some basic information about: No. verified files. No testing. of the tests included in files.No. tests with a positive result No. tests with a negative result. Duration of all tests. Comprehensive information about specific tests is available below. We can check the following: Duration of each test. How long did it take for a test to be completed in a given description. The actual code that was verified. In the upper left corner, there is another menu that allows us to learn a few more things: Filter all tests from one result (passed/failed/pending/omitted). Navigation (useful if lists include multiple tests). The date of execution of each test. Negative testing and filteringBelow, we can see an example of a test that ended with a failed result. Full access to the stacktrace of the error and a cause of the failed result is provided. In this case, the mail was wrong. Below, we can also see an example filtering run to quickly find tests that have failed or pending. And that's basically it. If you want to know even more about the topic, check my repository with the project example. Get more information about the mochawesome custom journalist used in the article, as well as other journalists compatible with Cypress. And if you want to check a report from a real project, go check out another of my Cypress projects on GitHub. Github.