Web Service (Api) Test cases

Test Strategy

Revision History

Date	Version	Author	Description
13.07.2021	1	Abdul Wahab	Test Strategy for web service

Table of Contents

1. Scope

Project Goals

The scope of this testing strategy/documents is to perform full round of api testing for the webservice that generate user on the base of country code. The goal is to check the api functionality understand the api response and write manual test suite and perform manual testing. Some of test case are automates using cypress.io framework, Cucmber.js with BBD format. In the case of automation all the test cases run in docker and proper test report is generated. All the testing activates perform mention in the documents.

Test Scope:

• Manual Test Cases Document
• Automated Test cases Using cypress
• Execute Test cases in Docker container
• Test Reports

• **Project Milestone :** One week Sprint
• **Company** : Sumup
• **Department**: Hardware tribe
• **Team lead** : Ugo Briasco

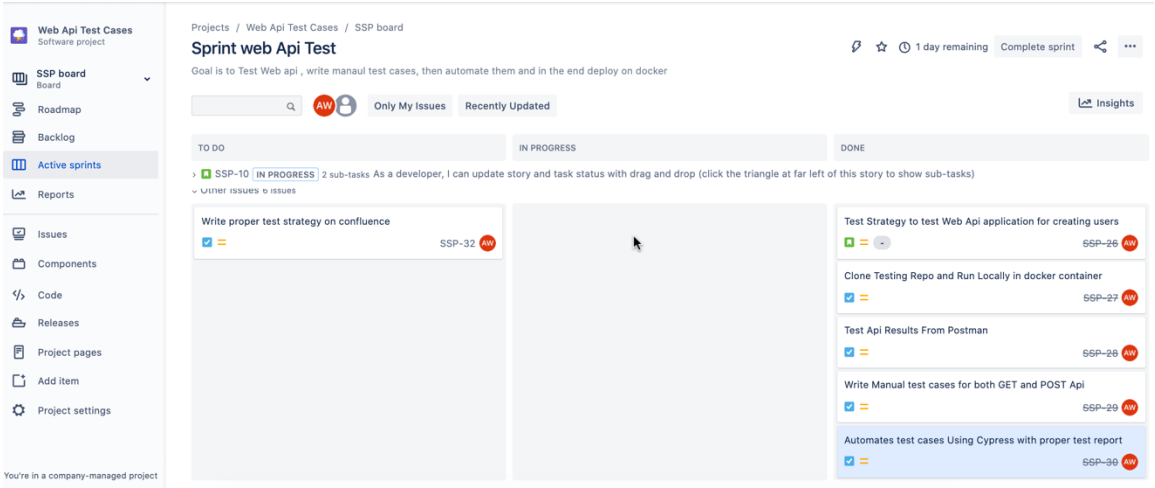
2. Test Approach

According to given scenario our task is to test webservice which has Rest API for generating user on the base of different country code. The first step for testing Rest Api is to understand the high-level Setup Requirement in order to run project to any machine. After running project on my local machine, then we have check the folder structure and routes in the project. According to mention scenario we have to test RestApi, for that we use POSTMAN api testing tools for functional testing that's ensure that api functionality fine. When we hit our api and we are getting proper GET and POST Json response. From Api 'POST request response we will try understand the json body fields and all its requirements.

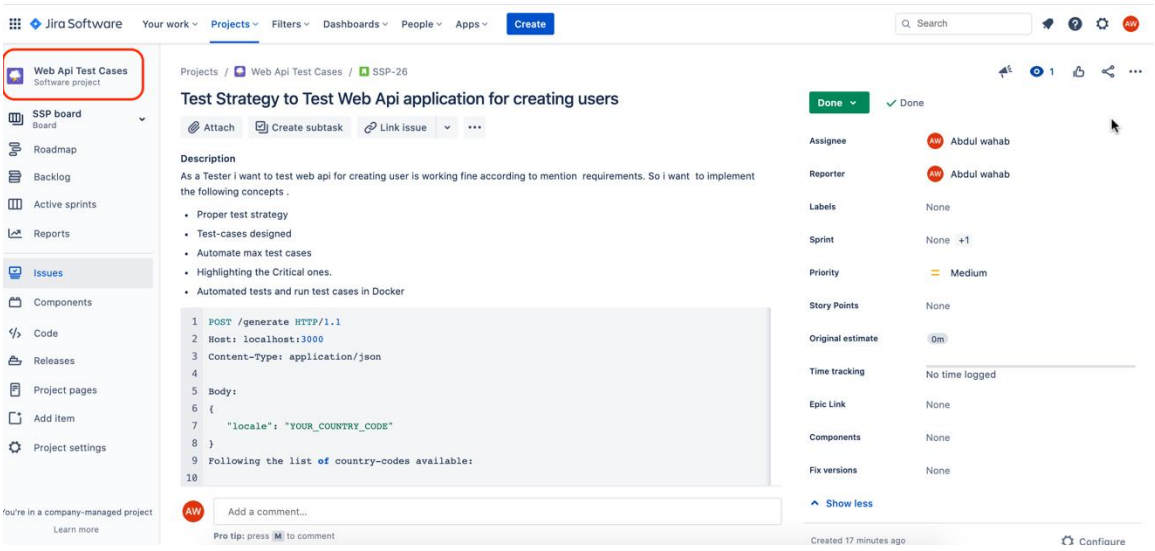
After analysing he functionality of Api then we will define different levels of testing in order to test our web service properly to ensure that api implementation is working correctly as expected no bugs!. After getting good understating of Api response we will make a plan to define Test scenario categories for example some basic Positive tests (happy paths), Negative test cases, Load and Performance Testing, Security, Authorization, and Permission tests cases (If required).Now our next task is to create proper text documents in which we write all test cases .After writing all manual test cases now we will decide which types of test cases we can automate them. According to test cases and its requirements we decide automation tool so that we can automates 90 % test cases. After automate test cases , we will run test case in docker container and in the end we generate test reports. Which test cases executed, passed, failed. and add new defects. In the end highlighting the critical ones test and sign off from for this web service api feature.

3. Test Environment

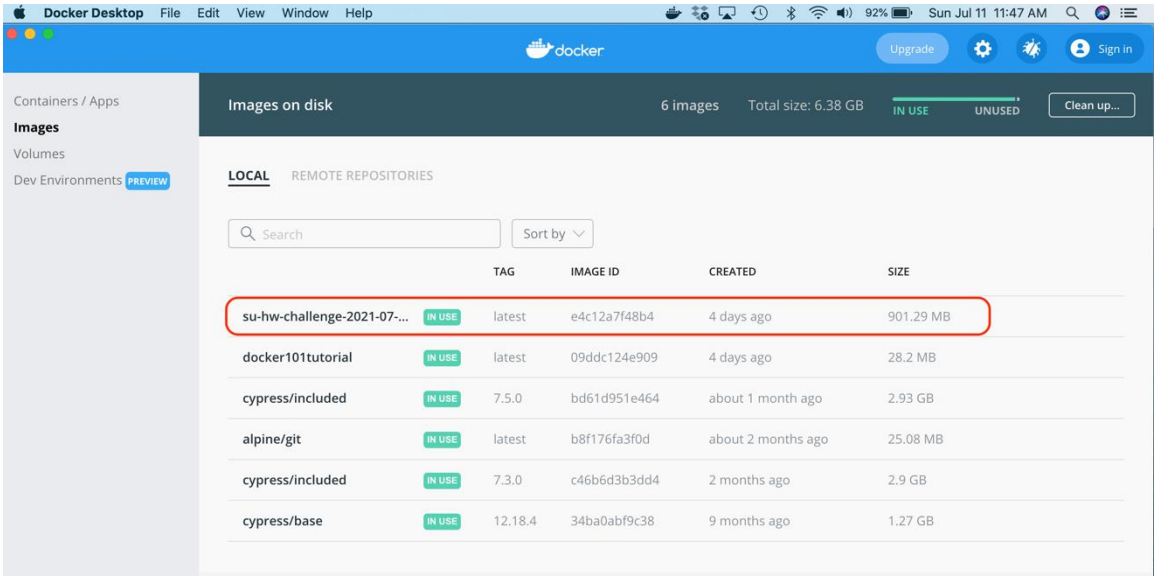
As Tester I created one week sprint to implement the all testing functionality. For this I used **Atlassian Jira** and create project. Add our main story then attached all the task with our main story .

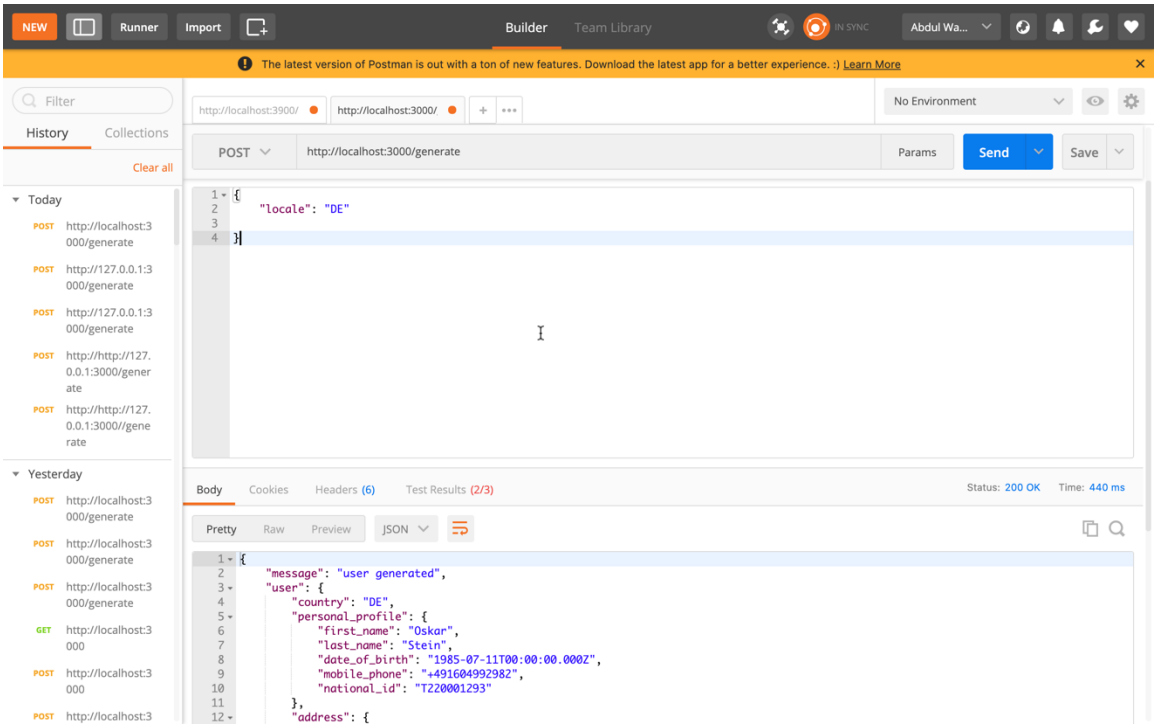


Our main story for testing with attached testing task.



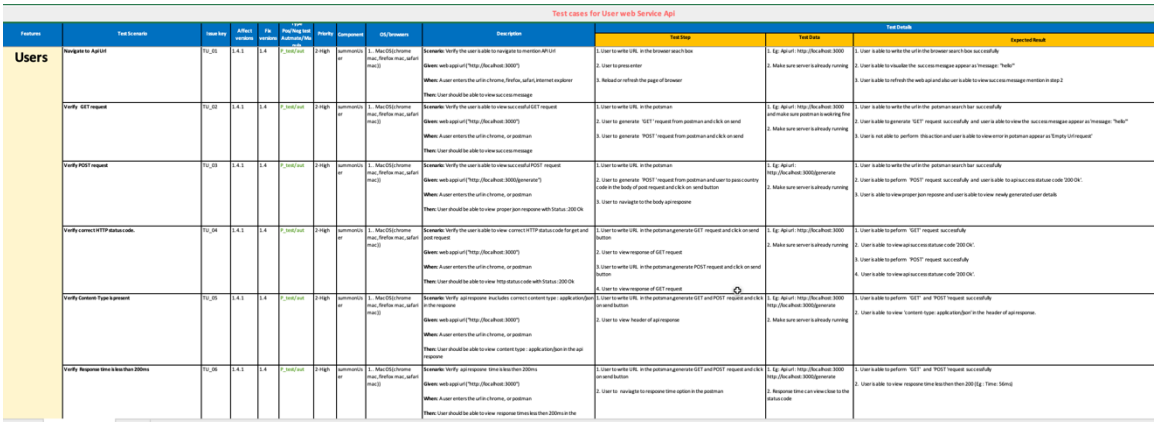
Now we clone our GitHub project repo (su-hw-challenge), then we install dependencies and run the server locally .we also install docker desktop for mac and check that our container su-hw-challenge is running properly .





After running our server and docker container now we will check the GET and POST response of our Api.We used POTSMAN api testing tool and view the api response. According to Postman our api shows status code 200 Ok ,Its mean that our api working fine.

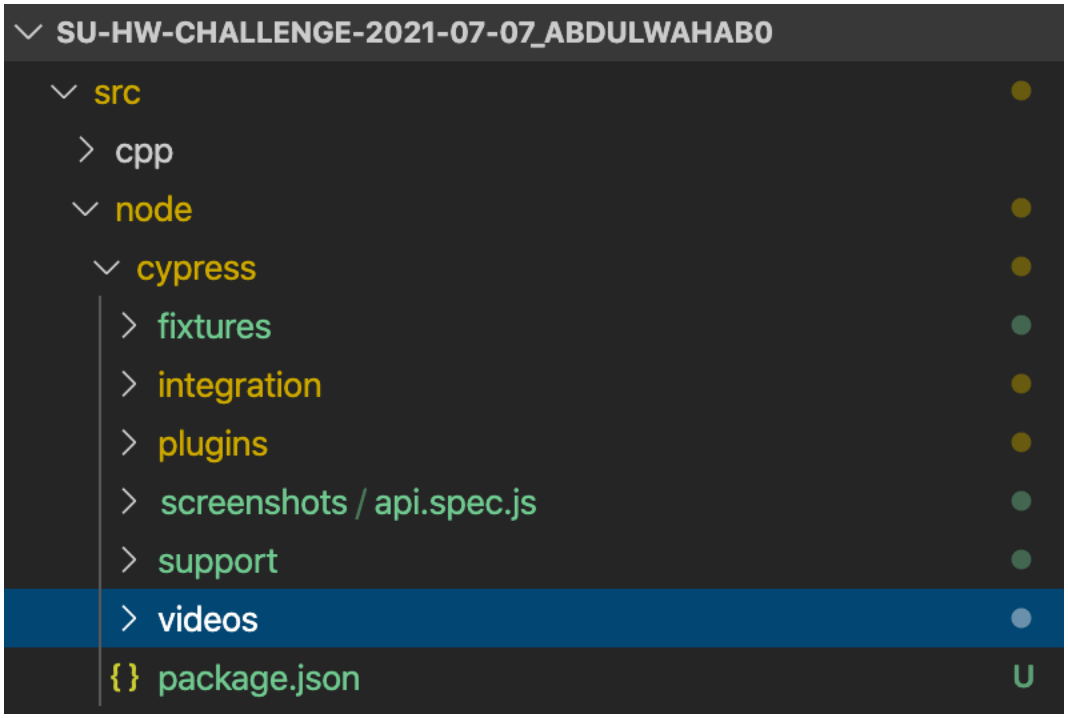
After deeply analysing the api responses then we will write manual test . In order to write manual test cases we design test template using Microsoft excel. In the manual testing suite we define test scenario ,issue key test description ,test steps, actual and expected results .



Test cases for User web Service Api

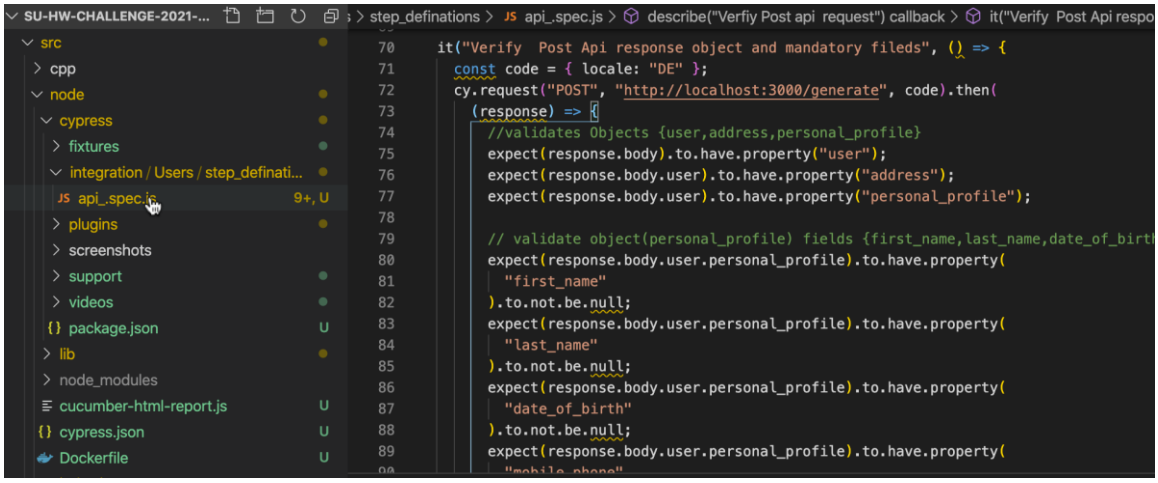
Features	Test Scenario	Issue key	Affect version	Fix version	Type: Bug/Bug test Automate/Manual	Priority	Component	OS/browsers	Description	Test Details		
										Test Step	Test Data	Expected Result
Users	Navigate to Api Url	TU_01	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify the user is able to navigate to mention API Url **Given:** web app url ("http://localhost:3000") **When:** A user enters the urlin chrome, firefox, safari, internet explorer **Then:** User should be able to view success message	1. User to write URL in the browser search box 2. User to press enter 3. Reload or refresh the page of browser	1. Eg: Api url : http://localhost:3000 2. Make sure server is already running	1. User is able to write the url in the browser search box successfully 2. User is able to visualize the success message appear as 'message: "hello" 3. User is able to refresh the web api and also user is able to view success message mention in step 2
	Verify GET request	TU_02	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify the user is able to view successful GET request **Given:** web app url ("http://localhost:3000") **When:** A user enters the urlin chrome, or postman **Then:** User should be able to view success message	1. User to write URL in the potsman 2. User to generate 'GET ' request from postman and click on send 3. User to generate 'POST ' request from postman and click on send	1. Eg: Api url : http://localhost:3000 and make sure postman is working fine 2. Make sure server is already running	1. User is able to write the url in the potsman search bar successfully 2. User is able to generate 'GET' request successfully and user is able to view the success message appear as 'message: "hello" 3. User is not able to perform this action and user is able to view error in potsman appear as 'Empty Url request'
	Verify POST request	TU_03	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify the user is able to view successful POST request **Given:** web app url ("http://localhost:3000/generate") **When:** A user enters the urlin chrome, or postman **Then:** User should be able to view proper json response with Status :200 Ok	1. User to write URL in the potsman 2. User to generate 'POST ' request from postman and user to pass country code in the body of post request and click on send button 3. User to navigate to the body api response	1. Eg: Api url : http://localhost:3000/generate 2. Make sure server is already running	1. User is able to write the url in the potsman search bar successfully 2. User is able to perform 'POST' request successfully and user is able to api success statuse code '200 OK'. 3. User is able to view proper json response and user is able to view newly generated user details
	Verify correct HTTP status code.	TU_04	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify the user is able to view correct HTTP status code for get and post request **Given:** web app url ("http://localhost:3000") **When:** A user enters the urlin chrome, or postman **Then:** User should be able to view http status code with Status :200 Ok	1. User to write URL in the potsman generate GET request and click on send button 2. User to view response of GET request 3. User to write URL in the potsman generate POST request and click on send button 4. User to view response of GET request	1. Eg: Api url : http://localhost:3000 2. Make sure server is already running	1. User is able to perform 'GET' request successfully 2. User is able to view api success statuse code '200 OK'. 3. User is able to perform 'POST' request successfully 4. User is able to view api success statuse code '200 OK'.
	Verify Content-Type is present	TU_05	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify api response includes correct content type : application/json in the response **Given:** web app url ("http://localhost:3000") **When:** A user enters the urlin chrome, or postman **Then:** User should be able to view content type : application/json in the api response	1. User to write URL in the potsman generate GET and POST request and click on send button 2. User to view header of api response	1. Eg: Api url : http://localhost:3000 http://localhost:3000/generate 2. Make sure server is already running	1. User is able to perform 'GET' and 'POST' request successfully 2. User is able to view 'content-type: application/json' in the header of api response.
	Verify Response time is less than 200ms	TU_06	1.4.1	1.4	P_test/aut	2-High	summonUs er	1. Mac OS(chrome mac, firefox mac, safari mac))	**Scenario:** Verify api response time is less than 200ms **Given:** web app url ("http://localhost:3000") **When:** A user enters the urlin chrome, or postman **Then:** User should be able to view response times less then 200ms in the	1. User to write URL in the potsman generate GET and POST request and click on send button 2. User to navigate to response time option in the postman	1. Eg: Api url : http://localhost:3000 http://localhost:3000/generate 2. Response time can view close to the status code	1. User is able to perform 'GET' and 'POST' request successfully 2. User is able to view response time less then then 200 (Eg : Time : 56ms)

Now we will automate our test cases, in order to automates our test cases we will cypress.io automation tool. We install cypress.io in our current directory according to their mention instruction manual .

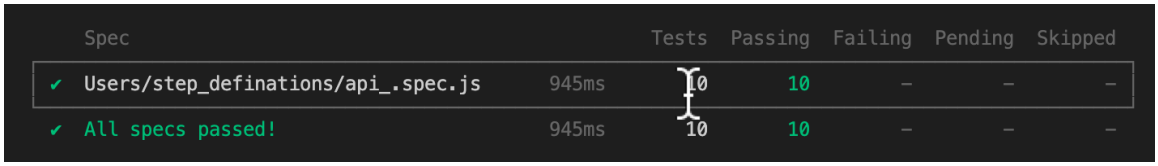


Now we will write all test cases in step definition file (with describe and it block).

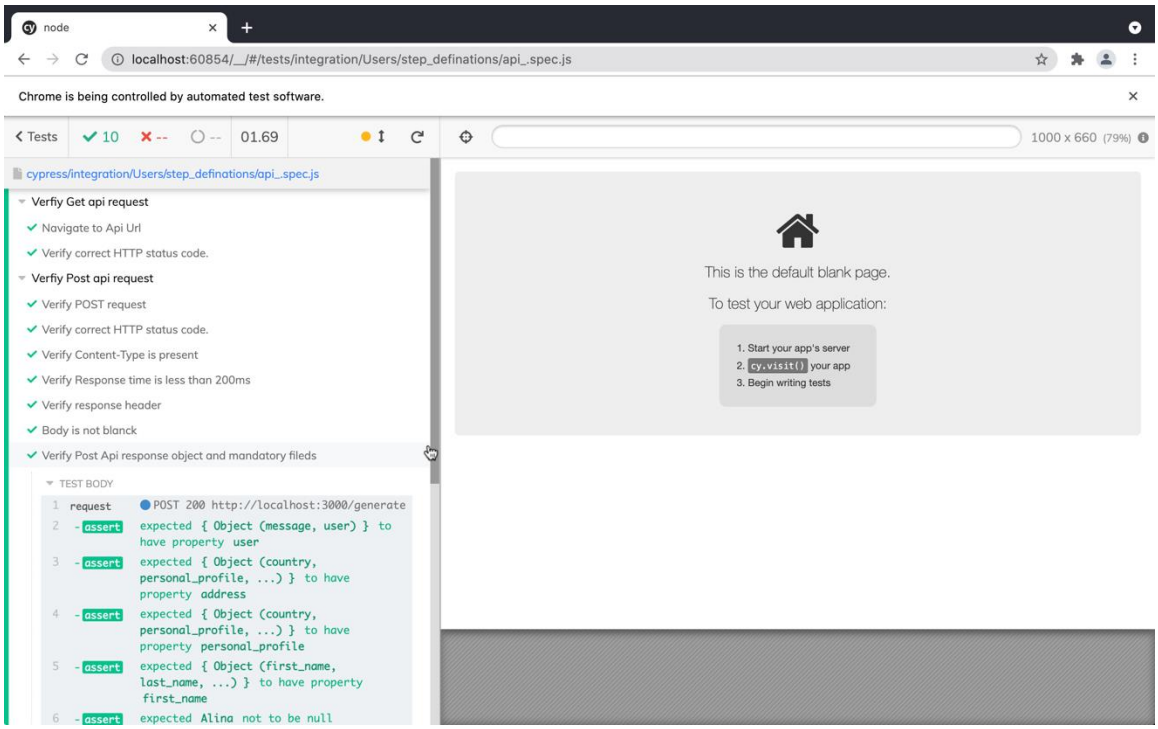
We will apply different cypress assertions and automate all test results.



Now we run all these test cases in headless mode using this command cypress run --headless --spec cypress/integration/Users/step_definations/**/* --browser electron



We can also view test results using cypress runner .



Run cypress test cases in Docker container :

Now we will run or our automated test cases in docker container. In order to rum the test cases first we need docker image for cypress. After downloading the cypress images for GitHub we will run following command to run all test cases in

docker container .

docker run --network=host -v $PWD:/e2e -w /e2e cypress/included:7.5.0

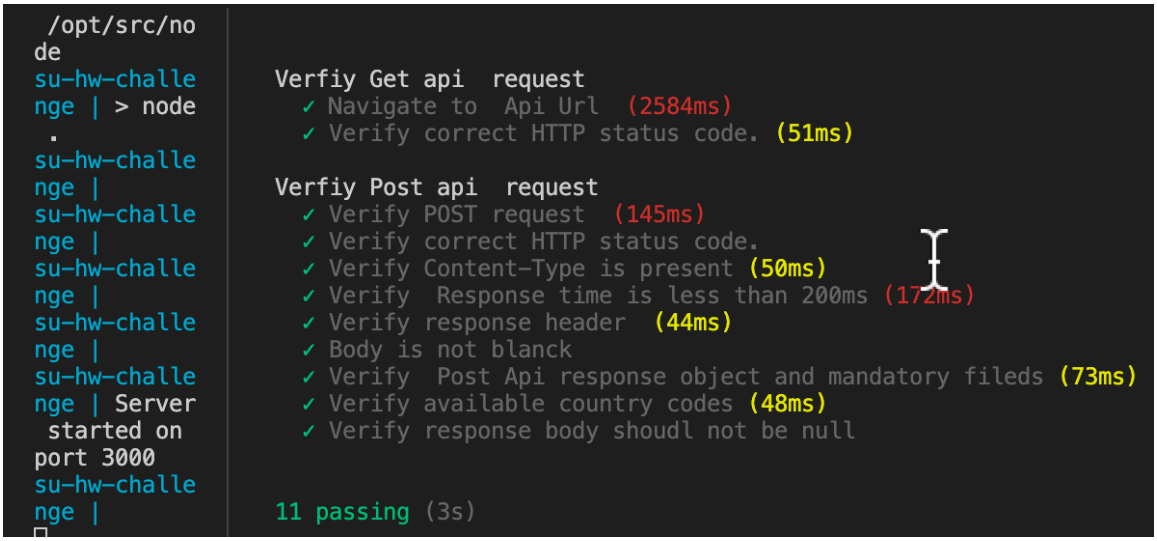
Explanation of the "docker run" command line arguments

- -it = interactive terminal

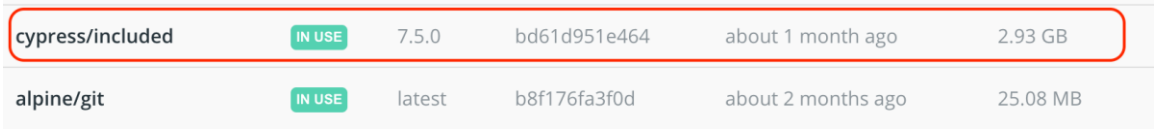
-v $PWD:/e2e = map current folder to /e2e inside the container

-w /e2e = set working directy to /e2e

After running the above command the final out in the docker cypress container .



Cypress Image in Docker Container



Test Report using cypress MochaAwsome:

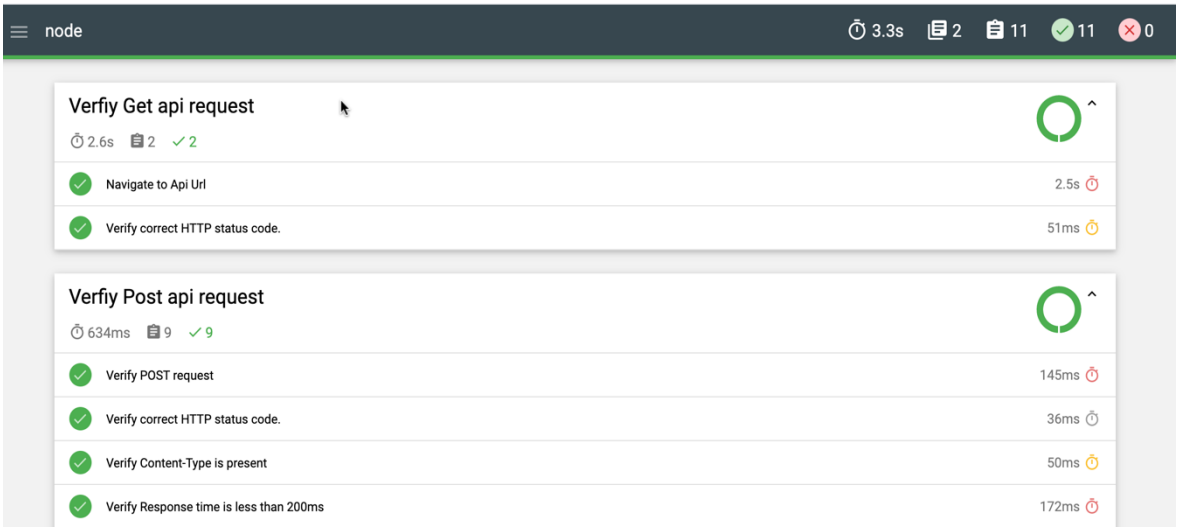
Now we can generate report of our test cases installing moachawsome package.

npm install mocha --save-dev

npm install mochawesome-merge --save-dev

npm install mochawesome-report-generator --save-dev

After adding reporter settings in cypress.json our final report



4. Testing Tools

Basically our main focus is to validate web service that generating the user .

So this webservice is using different Restfully Api .In order to test api

response

We user following tools, technologies and programming languages.

• Postman
• Cypress.io Framework (To Automates test cases)
• Mocha.js, chai.js and Cucumber.js with BBD (Behavioural Driven Development) approach
• Docker Container
• JavaScript
• Jira
• Mocha Awesome Reports
• VS code
• Microsoft Excel (For writing manual test cases)

5. Risk Analysis and Critical Test case

After running all test cases we just notice that some test are critical and its create some risk for web api .

- When we pass county code in the post body request , it may b a chance on different country code the information of user are same .
- When we pass wrong input in the body of post body request(eg: locale: 'DER') then the api http status code shows 200 and in body of response its shows message 'user generated', but in user object we have information of user .It kind of confusing .Devloper should apply proper validation check it the country code is empty , wrong input and more than 3 character then api fails or message show user is not generated .It give more clear pictures as user perspective

6. Test Summary

We perform manual testing and automated testing for our web api application here is the overall summary of our testing results ,

1. **Passed Test case**

   Almost 98 % test cases passes and giving expected output

• Navigate to Api URL
• Verify GET request
• Verify POST request
• Verify correct HTTP status code.
• Verify Content-Type is present
• Verify Response time is less than 200ms
• Verify response header
• Verify response payload
• Verify and check valid JSON body with Key and value pair
• Verify first name and last name should not be null
• Verify first name and last name should not be have special character and length
• Verify mobile number shoud not be null and validate mobile number format
• Verify available country codes
• Verify casesensivity of country code in post request

2. Fail Test cases

Some of test cases fails already mentioned in Risk analysis section

- Verify mobile number format(its fails when we have to exact mobile number length)

3. Area not tested

Some of important area of web api is not tested. For example **load or Performance testing** is not implemented mean that when a lot of user git our post api then the api works fine and give proper results in less time. Second scenario is that we are not to perform security testing,

Penetration testing , Fuzz testing and Authentication On All Endpoints.