sendcloud

SENDCLOUD's CRISPY SUCCOSTASH APPLICATION

TEST APPROACH
&
AUTOMATION SOLUTION

INTRODUCTION 3
TEST STRATEGY 4
SOLUTION 4
SCOPE 4
TEST COVERAGE 5
ROLES & RESPONSABILITIES 6
AUTOMATION SOLUTION 7
DESIGN 8
EXECUTION 10
REQUIREMENTS 10
URL CONFIGURATION 10
1) INTERACTIVELY EXECUTION 11
2) NON-INTERACTIVELY EXECUTION 13
3) REPORTING 14
4) VISUAL VALIDATION TESTS 16

INTRODUCTION

This document contains the description of the test approach taken for the ***Crispy Succotash*** application as well as the details about the automation solution along with instructions on how to run it, its design and reporting.

TEST STRATEGY

Given main goal of solution is to focus in automated test cases implementation and time is limited, this is a brief test strategy with the approach, scope and other definitions to consider in the test process.

SOLUTION

Crispy-Succotash provided solution is a web application which has been developed to cover the following requirements:

- ➔ User account creation and login
- ➔ Support to follow multiple feeds with at least support for the following ones:
  - http://www.nu.nl/rss/Algemeen
  - https://feeds.feedburner.com/tweakers/mixed
- ➔ Ability to add a bookmark/favorite to a feed item.
- ➔ Ability to add Markdown based comments to the feed items.

SCOPE

Given time available for implementation is limited, test suite will include only the main flows/cases of the mentioned requirements related use cases. Some additional test cases might be added with data driven approach in some additional test cases.

All test cases will be automated.

Tests will be 95% functional tests and 1 test case sample of visual validation will be added.

TEST COVERAGE

Functionality	Tests Included
USER CREATION	• *Validate default sign up page* • *Validate Positive User sign up* • *Validate sign up with empty fields* • *Validate sign up with different passwords* • *Validate sign up with short passwords* -> data driven test cases (8 in total)
LOG IN	• *Validate default log in page* • *Validate positive user log in*
FEEDS	• *Validate default new feed page* • *Validate submit empty feed* • *Validate My Feeds when empty* • *Validate All Feeds when added* • *Validate specific feed added* • *Validate empty bookmark feeds page* • *Validate bookmarked feed* • *Validate New Feed creation* • *Validate Feed Entry when Logged Out* • *Validate Feed Entry when Logged In* • *Validate add comment in Feed Entry*
VISUAL VALIDATION	• *Validate application aspect for logged out user.*

ROLES & RESPONSABILITIES

Role	Assigned to	Responsibilities
Software QA Automation Engineer	Matías Cárdenas	➔ Write test strategy ➔ Design and implement test automation solution ➔ Document test automation solution ➔ Upload solution and provide instructions on how to execute it

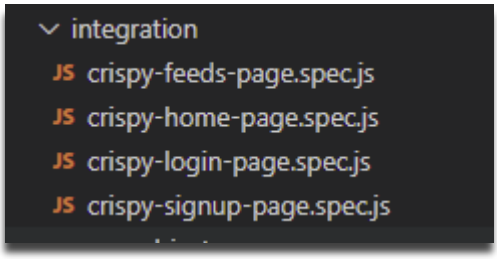
AUTOMATION SOLUTION

Automation solution has been developed in order to have more tests executed, make them faster and more efficient.

Solution has been built using ***Cypress***, which allow us to create faster, easier and more reliable tests.

DESIGN

- Tests are divided in 4 main parts: **Home, Sign Up, Log In and Feeds** pages.



HOME TESTS
(crispy-home-page.spec.js)

- *Validate default home page*
- *Validate Log In opens*
- *Validate All Feeds opens*
- *Validate Sign Up opens*

SIGN UP TESTS
(crispy-signup-page.spec.js)

- *Validate default sign up page*
- *Validate Positive User sign up*
- *Validate sign up with empty fields*
- *Validate sign up with different passwords*
- *Validate sign up with short passwords ->* data driven test cases (8 in total)

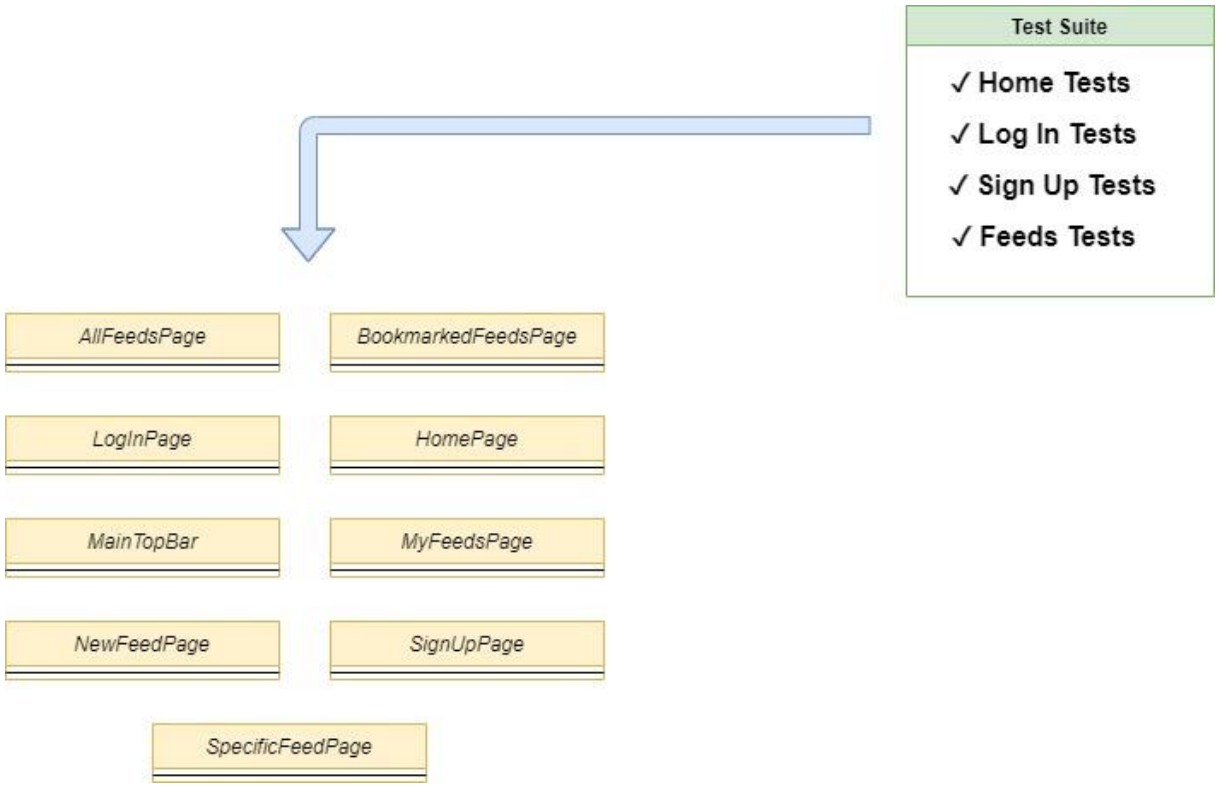
LOG IN TESTS
(crispy-login-page.spec.js)

- *Validate default log in page*
- *Validate positive user log in*

FEEDS TESTS
(crispy-feeds-page.spec.js)

- *Validate default new feed page*
- *Validate submit empty feed*
- *Validate My Feeds when empty*
- *Validate All Feeds when added*
- *Validate specific feed added*
- *Validate empty bookmark feeds page*
- *Validate bookmarked feed*
- *Validate New Feed creation*
- *Validate Feed Entry when Logged Out*
- *Validate Feed Entry when Logged In*
- *Validate add comment in Feed Entry*

- Also ***PAGE OBJECT*** pattern has been implemented as a neat, reusable and efficient way of instantiating web pages objects and performing manipulations on them.



EXECUTION

REQUIREMENTS

- NodeJS and NPM
- Cypress
- A Browser installed (Chrome, Firefox)
- Crispy Succostash application should be running:

  https://gitlab.com/sendcloud-public/crispy-succotash

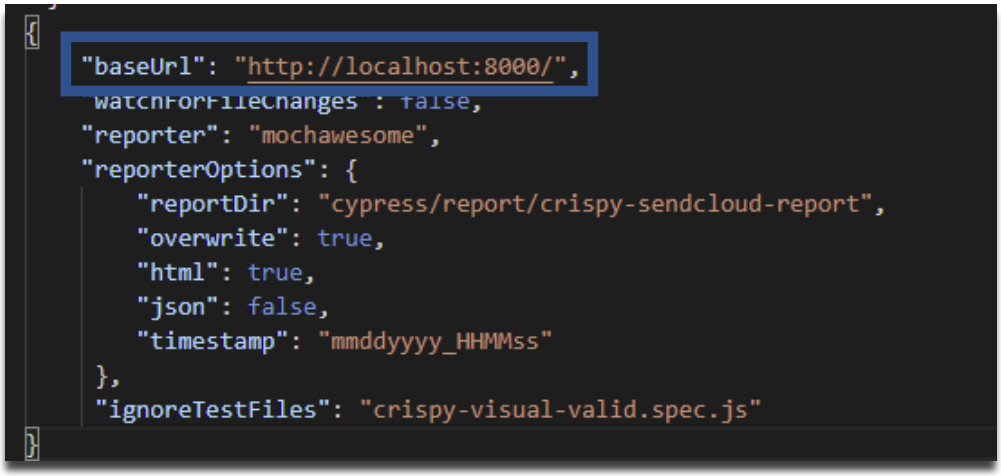
- Get the solution:

  - ***git clone https://github.com/maticardenas/sendcloud-test-automation.git***

URL CONFIGURATION

If you are running Crispy Succostash application by default, it will run in port 8000 in your localhost. Solution is currently configured to run against that, so nothing is required to be done.

However, this test solution should work whichever the port or url the app is running in (as long as it is accessible), if that is the case, the url can be updated in the ***cypress.json*** file of the project and the test solution will you use it to run the whole suite:



1) INTERACTIVELY EXECUTION

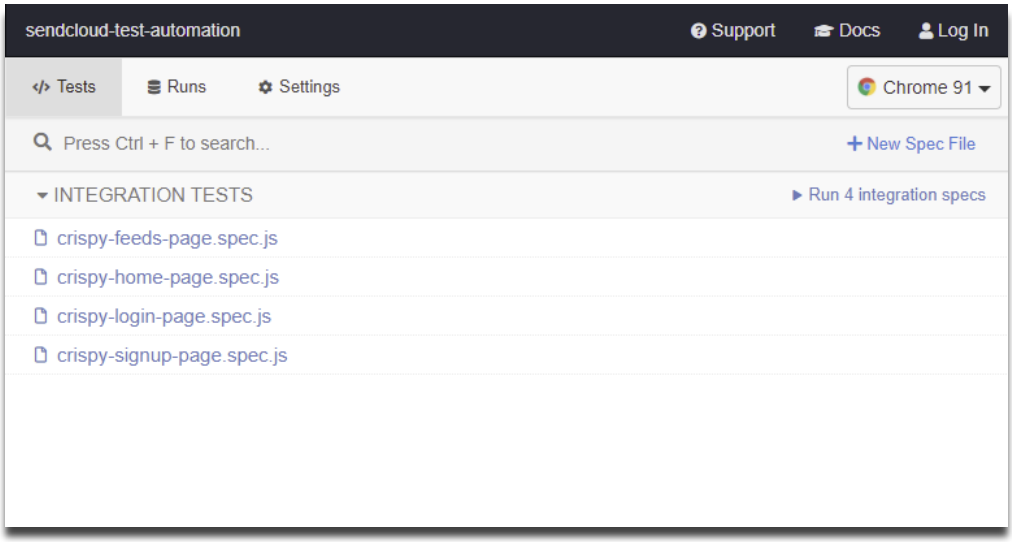
Scripts shortcuts have been added at ***package.json*** file, for executing all tests from npm.

All tests can be run **INTERACTIVELY** running this command:

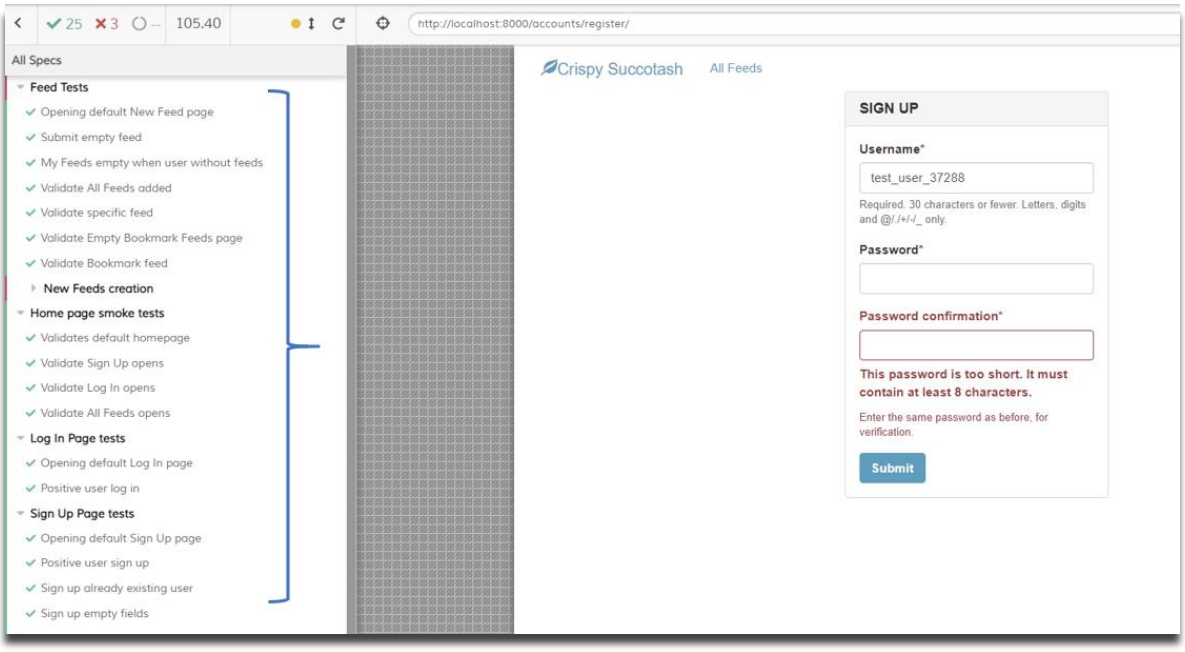
- ***npm run cypress***

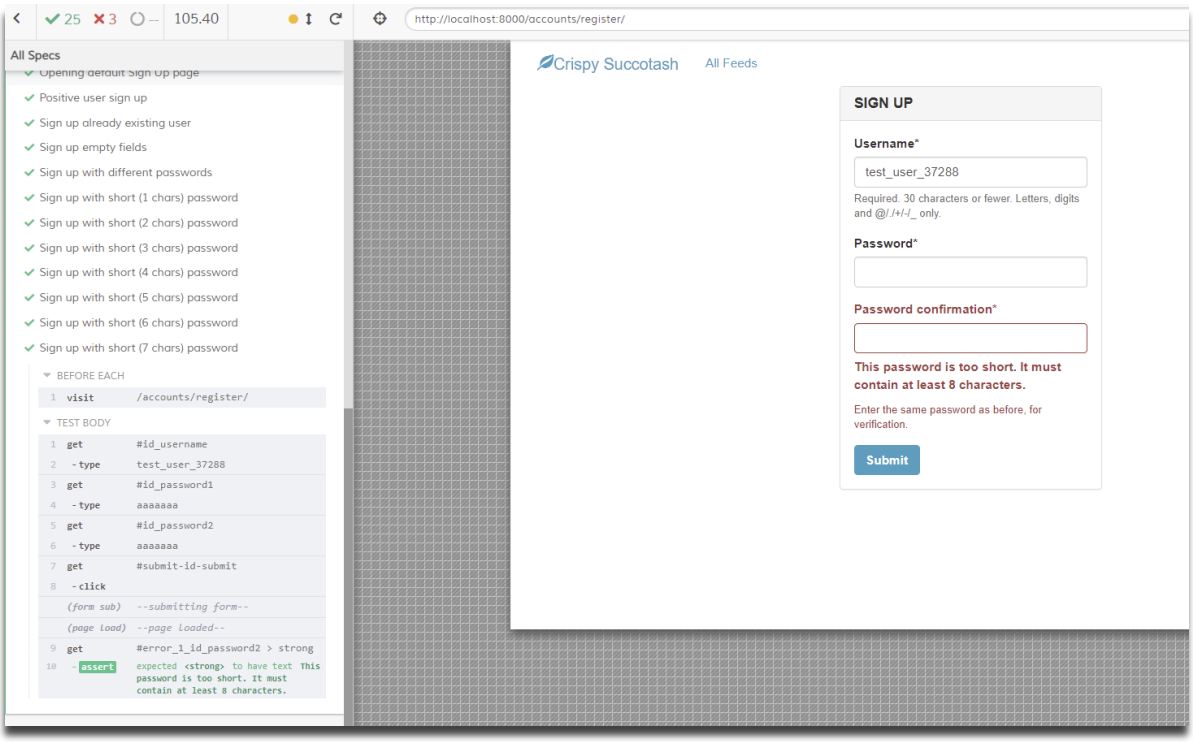
This command executes the Test Runner from cypress, from where is possible to execute all of the tests or the desired ones, in interactive mode.

O



Once executed, tests results as well as its steps can be checked next to the web page being displayed while performing the execution.



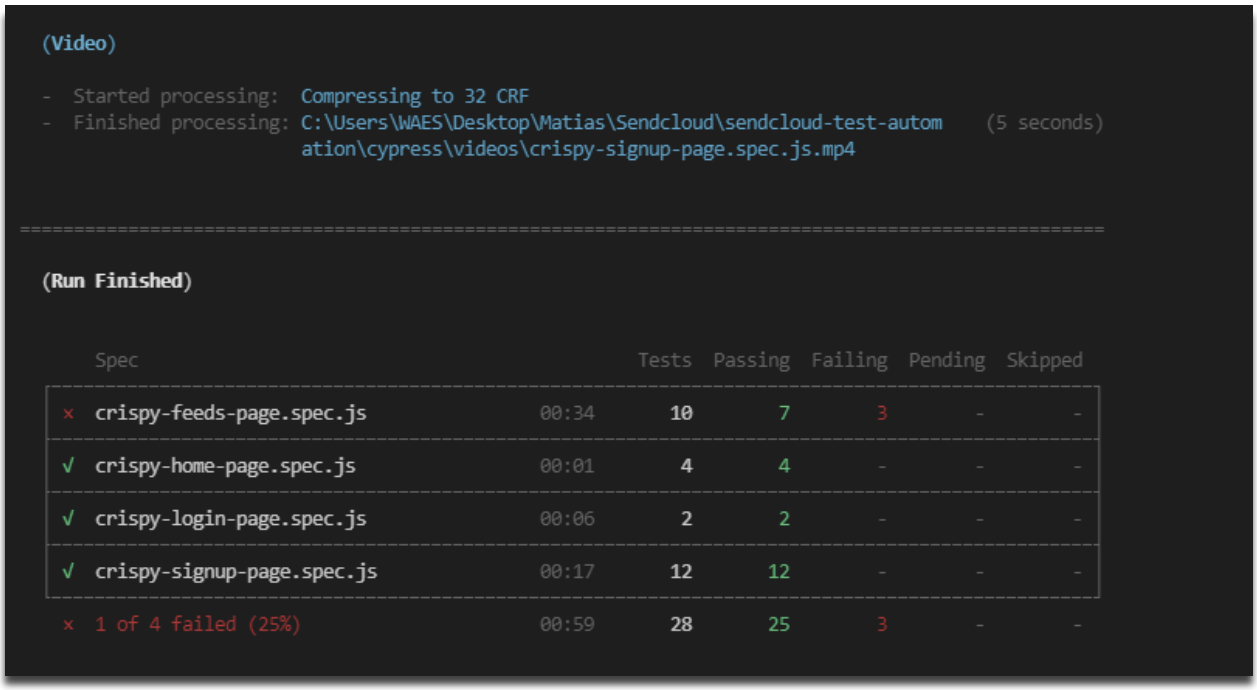


2) NON-INTERACTIVELY EXECUTION

All tests can be run **NON-INTERACTIVELY** running this command:

- ***npm run test***

This command will execute all tests in non-interactive mode, displaying results in the console without opening the interactive runner. It will generate though the videos of the execution at */videos/* folder of the repository.

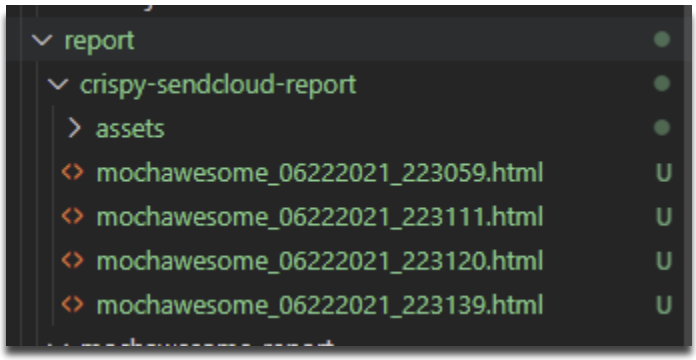


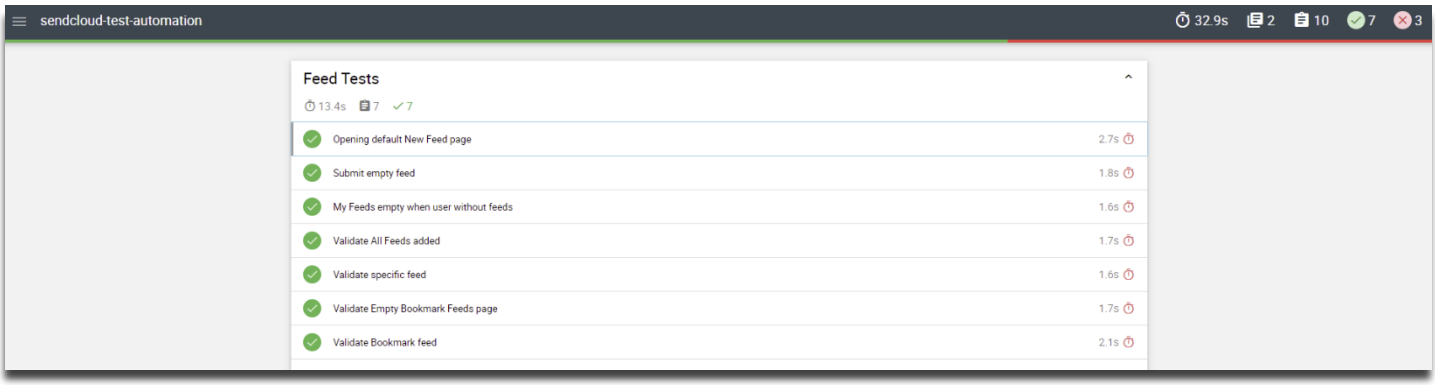
For executing each test suite individually the following commands can be used:

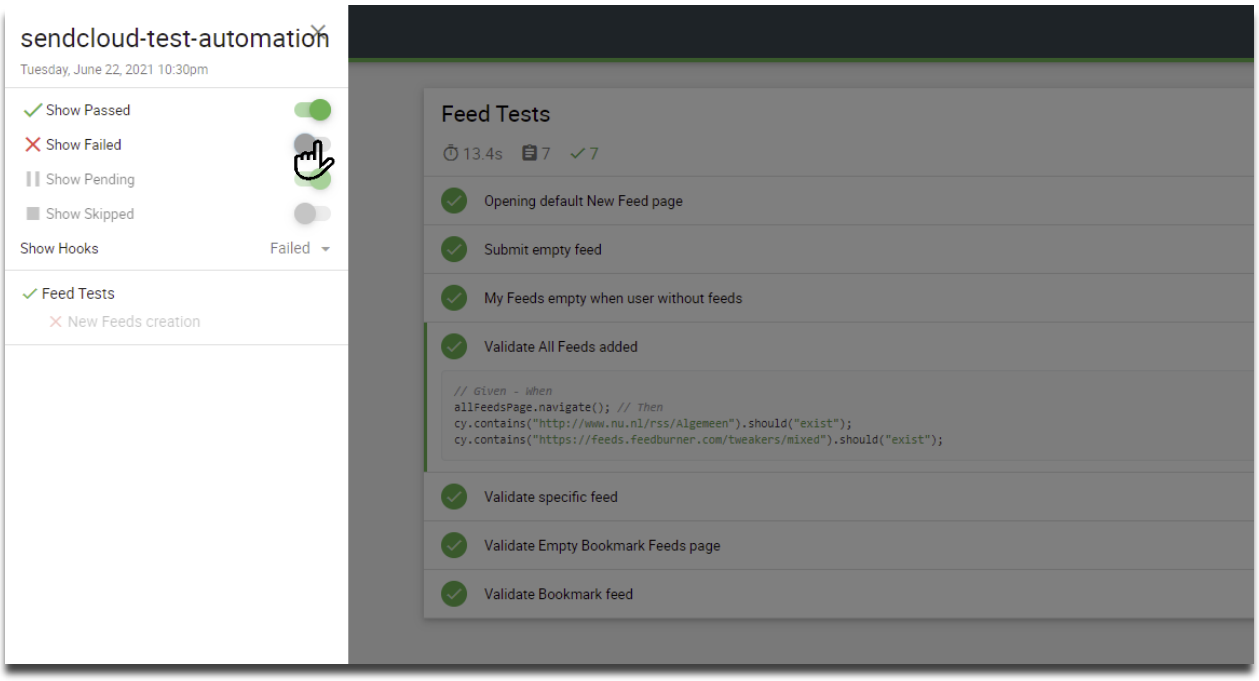
- ***npm run home-tests***
- ***npm run login-tests***
- ***npm run signup-tests***
- ***npm run feeds-tests***

3) REPORTING

Making use of ***mochawesome*** framework, at the end of the execution a report for the tests is generated at */report/crispy-sendcloud-report* directory:







<u>NOTE:</u> MOCHAWESOME REPORTS ARE ONLY GENERATED WHEN RUNNING THE TESTS IN NON-INTERACTIVELY MODE.

4) VISUAL VALIDATION TESTS

All previous mentioned and implemented tests are functional tests, there is also added a visual validation test, to validate the aspect of the application.

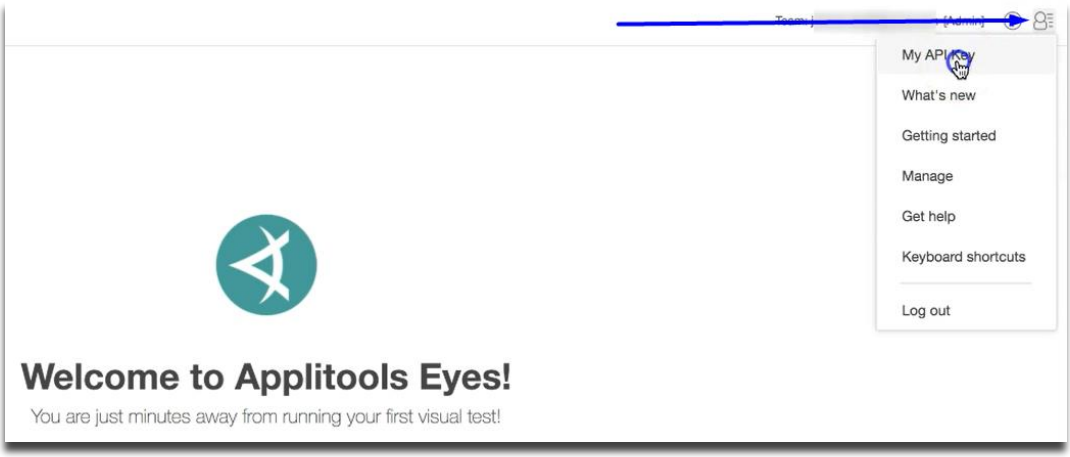
Test added as sample is for **logged out navigation** (crispy-visual-valid.spec.js) and with the same way much more several tests can be added to tests different aspects in different scenarios of the application. (no more were added given limited time to work on it).

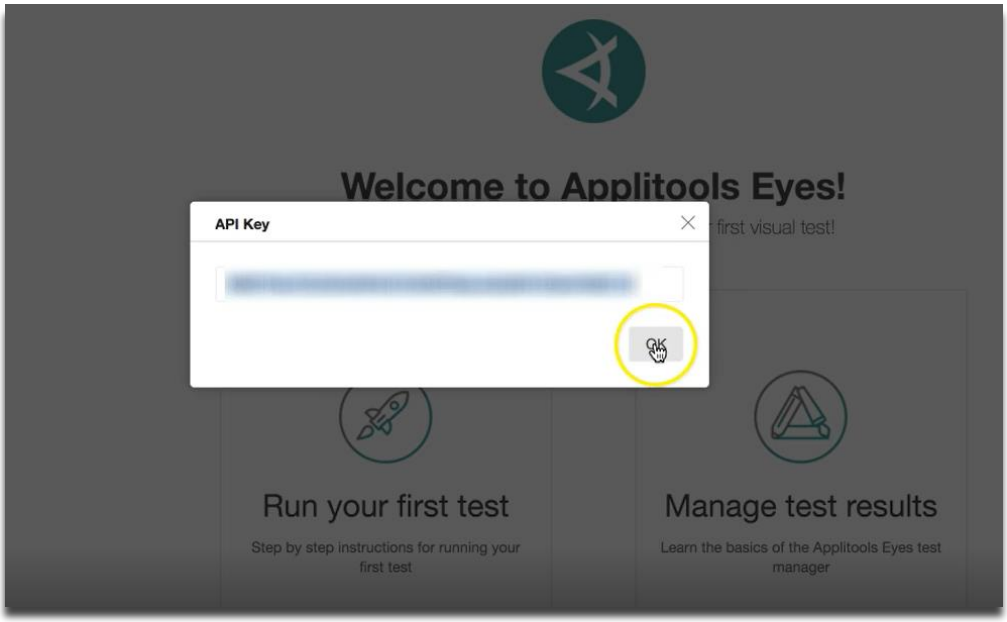
For running the visual validation tests **applitools / eyes-cypress** dependency is required. Also to see it's required to signup an account at applitools.com (it is possible to sign up with github account) and retrieve the API key from there.

REQUIREMENTS

- Install dependency:
  - **`npm install @applitools/eyes-cypress`**
- Link to cypress:
  - **`npx eyes-setup`**
- Setup API Key:

  1) Retrieve API key linked to your account from applitools.com





2) Set API Key as environment variable Linux terminal / Git bash / CMD (where you are executing the tests) :

Linux/Git bash:

- **`export APPLITOOLS_API_KEY=<API_KEY_VALUE>`**

Windows:

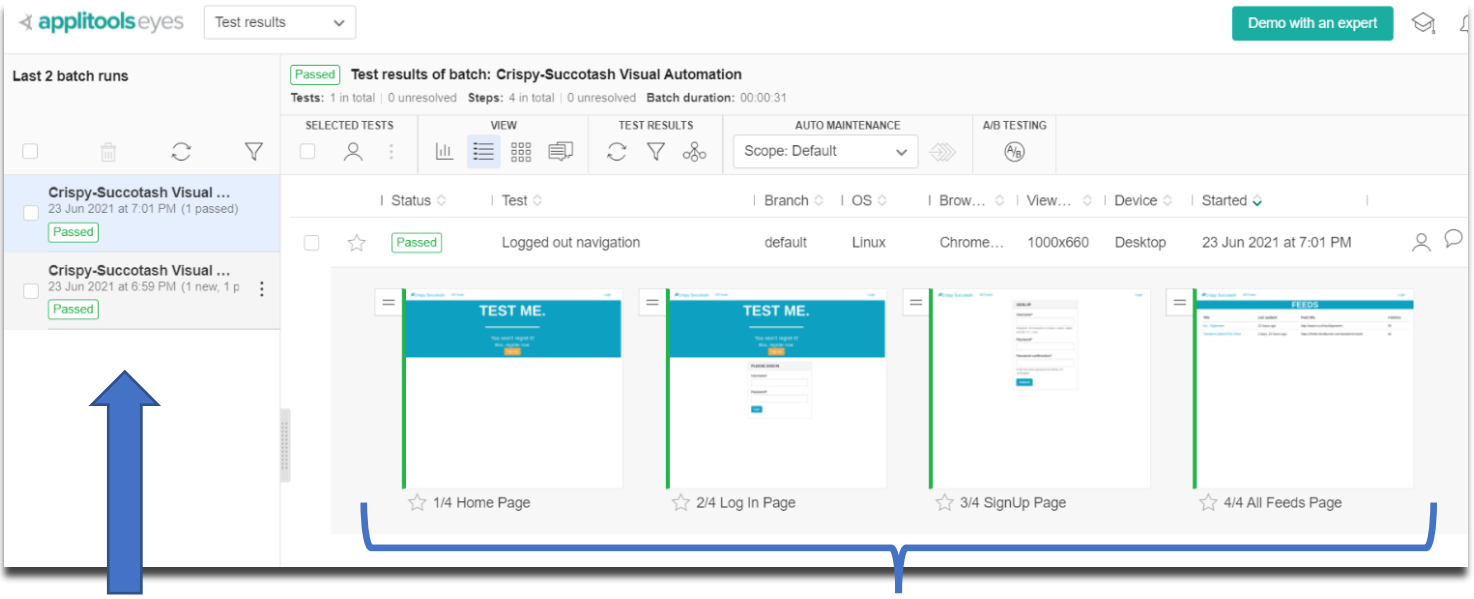
- **`set APPLITOOLS_API_KEY=<API_KEY_VALUE>`**

EXECUTION

Given the requirements step, this test is disabled by default in the suite. In order to enable it, it is just needed to remove the ignore line at *cypress.json* file:

```
{
    "baseUrl": "http://localhost:8000/",
    "watchForFileChanges": false,
    "reporter": "mochawesome",
    "reporterOptions": {
        "reportDir": "cypress/report/crispy-sendcloud-report",
        "overwrite": true,
        "html": true,
        "json": false,
        "timestamp": "mmddyyyy_HHMMss"
    },
    "ignoreTestFiles": "crispy-visual-valid.spec.js"
}
```

Test can be normally executed interactively or non-interactively running:

- **`npm run visual-validation`**

After finishing, at APPLITOOLS account there will be a record of the test executed with the snapshots taken for the visual validation: