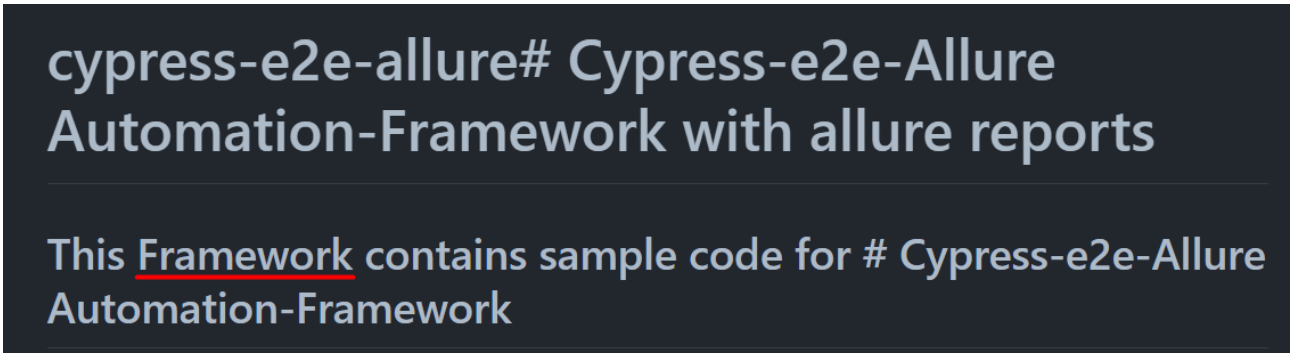
Review on the “cypressAllure” repository

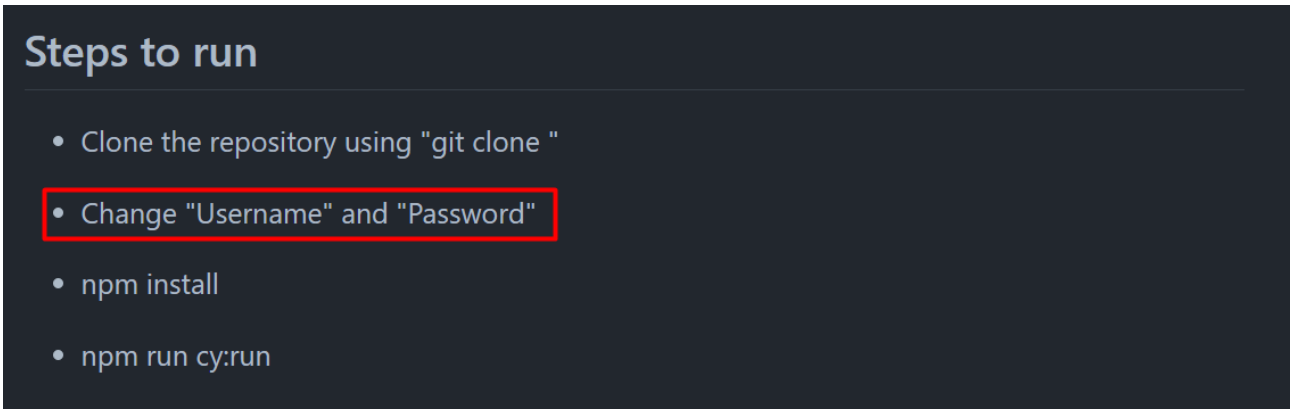
Repository author: AZANIR

Repository link: https://github.com/AZANIR/cypressAllure

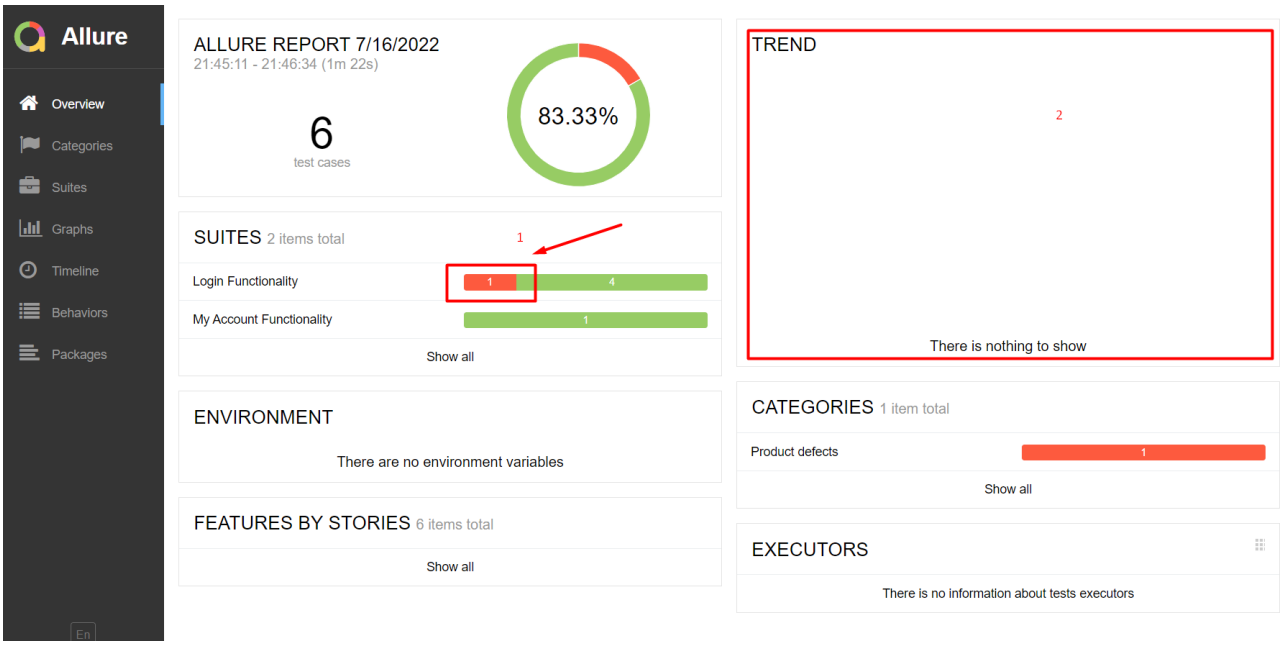
1. The repository named as “Framework”.




2. Steps to run has unclear command and are incomplete. Also use «```» characters to submit commands as code is a best practice.



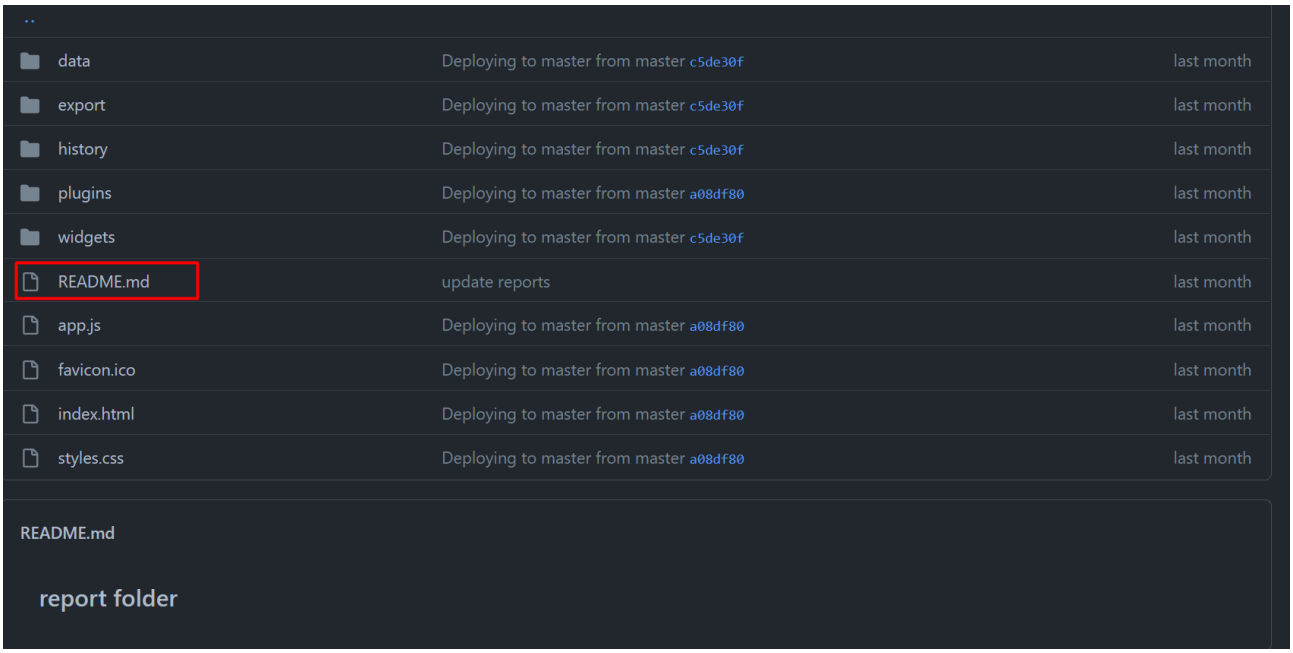

3. (1) - Some tests failed. (2) - The history of allure reports isn’t displayed (but it isn’t necessarily).



```
Login Functionality
  √ login with valid credentials (20318ms)
  √ login with valid credentials read data from fixture (49427ms)
  √ login with invalid email credentials read data from fixture (27614ms)
  1) login with invalid password credentials read data from fixture
  2) login with wrong email format credentials read data from fixture


3 passing (2m)
2 failing
```

4. The repository has some redundant files (README.md in the “docs” folder, commands.js in the “support” folder)




```
1  // ***********************************************
2  // This example commands.js shows you how to
3  // create various custom commands and overwrite
4  // existing commands.
5  //
6  // For more comprehensive examples of custom
7  // commands please read more here:
8  // https://on.cypress.io/custom-commands
9  // ***********************************************
10 //
11 //
12 // -- This is a parent command --
13 // Cypress.Commands.add('login', (email, password) => { ... })
14 //
15 //
16 // -- This is a child command --
17 // Cypress.Commands.add('drag', { prevSubject: 'element'}, (subject, options) => { ... })
18 //
19 //
20 // -- This is a dual command --
21 // Cypress.Commands.add('dismiss', { prevSubject: 'optional'}, (subject, options) => { ... })
22 //
23 //
24 // -- This will overwrite an existing command --
25 // Cypress.Commands.overwrite('visit', (originalFn, url, options) => { ... })
```

5. There is no general page for commands. It would be better to use a two-level page object pattern.

6. Selectors should be declared as constants before the class declaration so that they are called only when needed and easily replaced if the selectors break on the site.

```
/// <reference types="cypress" />

class LoginPage {
    get signinLink() { return cy.get('.login') }
    get emailAddressTxt() { return cy.get('#email') }
    get passwordTxt() { return cy.get('#passwd') }
    get signinBtn() { return cy.get('#SubmitLogin') }
    get alertBox() { return cy.get('p:contains("error")')}
    get alertMessage() { return cy.get('.alert-danger > ol > li') }

    public launchApplication() {
        cy.visit('/')
    }

    public login(emailId: string, password: string) {
        this.signinLink.click()
        this.emailAddressTxt.type(emailId)
        this.passwordTxt.type(password)
        this.signinBtn.click()
    }
```

7. It is better to use “require” for import classes in small projects.

```
import { loginPage } from '../pages/loginPage'
import { myAccountPage } from '../pages/myAccountPage'
```

8. (1) – There is “beforeEach” for only one “it” block.

(2) – There are some commented commands and redundant comments

```
import { loginPage } from "../pages/loginPage";

describe('My Account Functionality', () => {
    beforeEach(() => {
        cy.visit('https://google.com');
        //loginPage.launchApplication()
    })
    it('Sample Test', () => {
        console.log("This is a sample test")
    })
})
```

```
// ***********************************************************
// This example support/index.js is processed and
// loaded automatically before your test files.
//
// This is a great place to put global configuration and
// behavior that modifies Cypress.
//
// You can change the location of this file or turn off
// automatically serving support files with the
// 'supportFile' configuration option.
//
// You can read more here:
// https://on.cypress.io/configuration
// ***********************************************************

// Import commands.js using ES2015 syntax:
import './commands'
import '@shelex/cypress-allure-plugin'

// Alternatively you can use CommonJS syntax:
// require('./commands')
```

9. “it” blocks better to named with word “Should”. For example: “Should login with valid credentials”.

```
it('login with valid credentials', function () {
    loginPage.login("testautomation@cypresstest.com", "Test@1234")
    myAccountPage.validateSuccessfulLogin()
    myAccountPage.logout()
    myAccountPage.validateSuccessfulLogout()
})
```

10. Assertions should be in specs files. This is makes tests more understandable.

```
/// <reference types="cypress" />

class LoginPage {
    get signinLink() { return cy.get('.login') }
    get emailAddressTxt() { return cy.get('#email') }
    get passwordTxt() { return cy.get('#passwd') }
    get signinBtn() { return cy.get('#SubmitLogin') }
    get alertBox() { return cy.get('p:contains("error")')}
    get alertMessage() { return cy.get('.alert-danger > ol > li') }

    public launchApplication() {
        cy.visit('/')
    }

    public login(emailId: string, password: string) {
        this.signinLink.click()
        this.emailAddressTxt.type(emailId)
        this.passwordTxt.type(password)
        this.signinBtn.click()
    }

    public validateLoginError(errorMessage: string) {
        this.alertBox.should('be.visible')
        this.alertMessage.should('have.text', errorMessage)

    }
}
export const loginPage: LoginPage = new LoginPage()
```

11. Messages and test data for inputs should be declared as constants in the top of spec.

```
    it('login with invalid email credentials read data from fixture', function () {
        loginPage.login(this.data.invalid_credentials.invalid_email.emailId,
            this.data.invalid_credentials.invalid_email.password)
        loginPage.validateLoginError('Authentication failed.')
    })
    it('login with invalid password credentials read data from fixture', function () {
        loginPage.login(this.data.invalid_credentials.invalid_password.emailId,
            this.data.invalid_credentials.invalid_password.password)
        loginPage.validateLoginError('Authentication failed.')
    })
    it('login with wrong email format credentials read data from fixture', function () {
        loginPage.login(this.data.invalid_credentials.wrong_email_format.emailId, this.data.invalid_credentials.wrong_email_format.password)
        loginPage.validateLoginError('Invalid email addressssss.')
    })
```

```
    it('login with valid credentials', function () {
        loginPage.login("testautomation@cypresstest.com", "Test@1234")
        myAccountPage.validateSuccessfulLogin()
        myAccountPage.logout()
        myAccountPage.validateSuccessfulLogout()
    })
```

12. There is camel case in most repository, but in fixtures there is some snake case.

```
{
    "valid_credentials": {
        "emailId": "testautomation@cypresstest.com",
        "password": "Test@1234"
    },
    "invalid_credentials": {
        "invalid_email": {
            "emailId": "invalidUser@cypresstest.com",
            "password": "Test@1234"
        },
        "invalid_password": {
            "emailId": "testautomation@cypresstest.com",
            "password": "test@12345"
        },
        "wrong_email_format": {
            "emailId": "testautomationresstest.com",
            "password": "test@12345"
        }
    }
}
```