Modern Web Automation with Cypress.Io

Jyolsna Thekkan Othayoth[1*], Syahid Anuar[2]

[1,2,]Razak Faculty of Technology and Informatics, Universiti Teknologi Malaysia,54100 Kuala Lumpur
[1]thekkan1998@graduate.utm.my,[2]syahid.anuar@utm.my

Article history

Received:
20 Sept 2022

Received in revised form:
30 Nov 2022

Accepted:
1 Dec 2022

Published online:
15 Dec 2022

*Corresponding author
suriani.kl@utm.my

Abstract

Over the past few decades, software test automation has seen significant changes. Less manual intervention and great test result accuracy are guaranteed by the automation tools. Web automation has been essential to delivering continuous delivery and guaranteeing excellent quality of the product in each iteration by automating repetitive testing as Agile software development has grown in popularity. Despite the fact that test automation is essential, there are some shortcomings with popular web automation tools, such as wait time problems and erratic test results, a lengthy process for creating tests, difficulties setting up the test environment, problems with page and element load, insufficient test result reporting, a lack of built-in commands for automating tests, the absence of features for image testing, etc. In this case, the development and client organizations, as well as the project team, would benefit more from a solution that is more dependable, quick, and supports development and testing simultaneously. A more solid and powerful web automation tool is required to successfully deliver within an agile CI/CD pipeline and to effectively handle the features of dynamic web applications. An emerging web test automation tool called Cypress.io is becoming more popular among the industry experts. Cypress has been highlighted as a modern web automation tool capable of addressing the new issues posed by today's industry trends. In this project, a test automation framework using Cypress.io is implemented for a work-in-progress application, and the tool's effectiveness is assessed.

Keywords: *Software testing, Test Automation, Web testing, Automated testing in Agile, Cypress.io*

1. Introduction

Agile Software Development is the most preferred choice in the global software development world. It is an iterative method wherein each iteration is a self-contained approach that consists of requirement analysis, design, coding, and testing [1]. Quality delivery with Continuous integration, is an integral part of successful projects in Agile; with continuous integration code is being deployed to the production environment continuously or whenever there is a business/project need [2]. A web automation framework that supports end-to-end automation and continuous delivery is integral to any Agile projects. Test automation involves, tools to automate the tests and ensure less manual intervention [3]. It helps to free up the test resources for other effective and efficient use of them rather than executing the repeated time-consuming tests [4]. Web automation faced longer development and execution time that results into delayed product releases [5]; Testing contributes a

** Corresponding author. thekkan1998@graduate.utm.my*

high portion of overall software life cycle, with the iterative nature of the Agile, the traditional Software testing is not a right fit for ensuring the quality product, Agile enforces the need for a wide and stable test automation approach to bring the necessary agility in the project and helps it to respond to rapid changes and iterative releases [6]. During the initial years of test automation, capture and playback tools were popular, the tester would capture a set of action on the applications and the tool records those actions and the tester can play back the recorded action for the future testing [7]. In the following years, different test automation approaches have been evolved namely,

Linear Scripting, Structured scripting, Data driven, Keyword Driven, Process Driven, Model Based etc. The automation approach was decided based on the System under test and the Project characteristics [8]. Most of the popular web automation tools have been identified as not effective for implementing continuous integration. With the above challenges and the rapid Agile needs [9]; the industry experts are on the verge of analyzing and exploring new Web automation tools to meet the testing and quality needs of the modern web applications [10]. Cypress.io is an evolving tool which is gaining more popularity in the field of test automation for web applications. Cypress is a JavaScript-based end-to-end testing framework; unlike other web testing tools, Cypress does not use Selenium drivers at all [11]. It is built on NodeJS architecture. It is built on top of Mocha (JavaScript test framework). It runs in the same loop as the web application and captures snapshots at the time of test execution [12]. Cypress is a collection of libraries that are packed together. It connects directly with the Application Under Test (AUT) from the browser. The browser is used to run all Cypress test scripts. For example, to click on a certain button, Cypress does not use a specific driver to convey the command to the browser. Instead, it sends the click command to the button using DOM events. As a result, test findings are executed significantly more quickly. This allows the team to develop tests that are faster, easier, and more. Cypress uses automatic waiting, whereas it knows that application is busy and waits on its own. For Cypress to achieve parallel execution one will have to use docker images and it support headless chrome. Test efficiency refers to the cost-effectiveness in relation to an organization's resources or in this case the total effort involved in building the automation script. The most efficient test is one that can achieve an acceptable software quality level with the least amount of work [13]. Compared to other web automation tools, Cypress creates fewer lines of code. This suggests that writing automation scripts in Cypress requires less effort, resulting in a considerable gain in test efficiency. In case of other automation tools, programs must be instantiated with importing required libraries and web pages, whereas Cypress handles most of the automation script directly on the browser [14]. This project aims to analyze Cypress.io and to implement a test automation framework using cypress.io within a development project that follows Agile CI process. The advantages and efficacy of the tool will be discussed towards the end of the project.

2. Methodology

2.1 Introduction

The objective of this project is to automate the E2E test cases of a web application named 'Smart TV', which is under development, targeting stable, efficient and flake free automated tests that complies with the standard guidelines recommended by ISTQB for effective and efficient test automation. The application under test is built in JavaScript, and tests written in JavaScript using Cypress enable for better DEV-QE collaboration, which is an important factor in the CI process [15]. This chapter explains the development of the whole project, that focuses on design, implementation, execution, results, and reports. The Agile cycle for this project will be the same as for the Smart TV development project. The set of UI tests that should be automated will be identified based on the end-to-end functionality that requires regression towards the end of every sprint. Then these tests will be automated using Cypress.io and an evaluation will be performed to understand the efficacy of the tool.

2.2 Project Plan

a) The test project will follow the same Agile development cycle of the application under test.

b) The end-to-end UI tests that require sprint vise regression tests will be automated.

c) The software quality engineers dedicated for the development project Smart TV, will be responsible for automating the identified E2E tests with cypress.io, together with their other responsibilities.

2.2 Smart TV Project Structure

There are 3 Software Quality Engineers, dedicated for the Smart TV Project including one senior QE, and all three of them are responsible for the test automation with Cypress.io. The team can seek occasional advises from the Test automation Architect who is basically serving as a test automation consultant across multiple projects. The quality team within the Smart TV project reports to the Software Quality Manager who oversee the quality team deliverables across multiple projects.

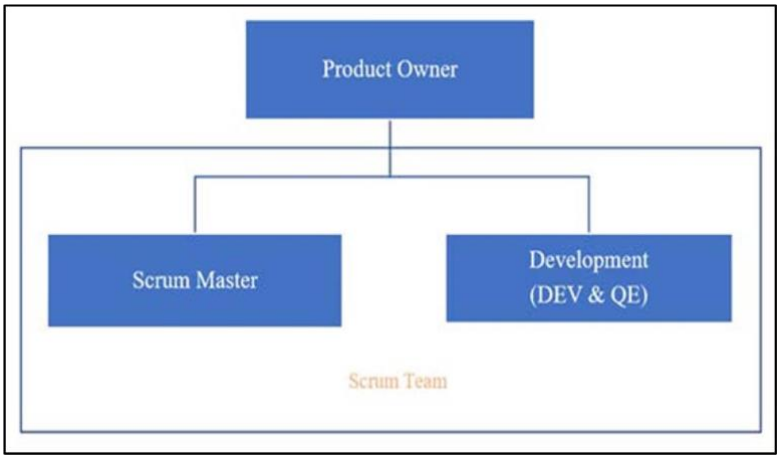

Figure 1. Scrum Team Structure for the Smart TV Project

The Smart TV application follows an Agile development model with sprints spanning 2 weeks (10 working days) per sprint. As in the Figure 1, the scrum team for the 'Smart TV' project comprises of Product Owner who is responsible for preparing the requirements and pass to the scrum team, Scrum
Master who is basically an agile coach makes sure that the team can function and deliver within the 'agile processes and practices' and helps to address any bottlenecks that impact the team delivery and efficiency. The scrum team comprises of 6 Developers and 3 QEs.

2.4. Identifying tests to be automated

Test cases to be automated as part of the Cypress.io implementation have been identified based on three parameters basically,

a. Tests for critical features of the application.
b. Tests covering end to end business flows.
c. Tests that require mandatory regression tests for each release.

2.5. Generic Test Automation Architecture

The test framework with Cypress.io follows the generic Test Automation Architecture (TAA) recommended by ISTQB as in the Figure 2.

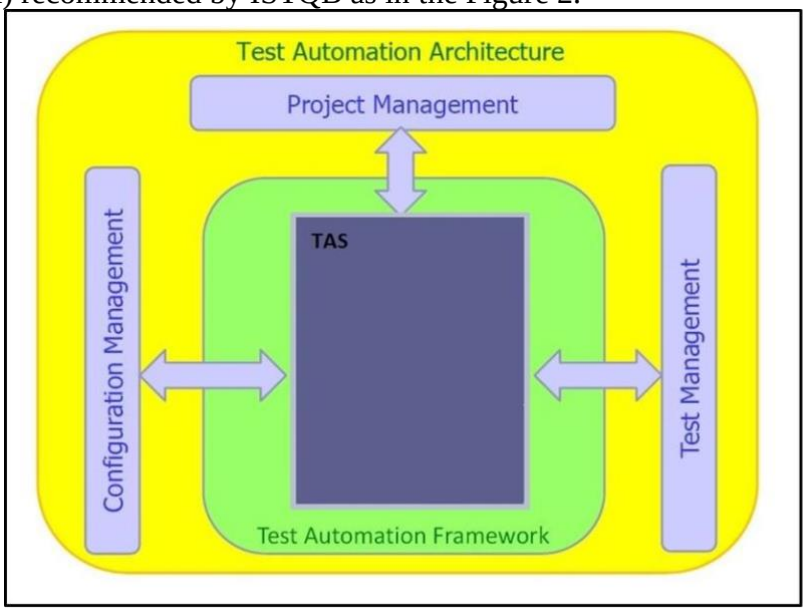

Figure 2. Generic Test Automation Architecture recommended by ISTQB

As in the Figure 2, the Test Automation System (TAS) interacts with the Project Management System, which is Jira in Smart TV project and Configuration management which is GitHub and Test Management which is TestRail in the current project.

2.6. Test Automation framework with cypress

Figure 3 depicts the test automation framework with Cypress architecture, Cypress comprises of all required libraries within it such as assertion libraries and mocking mechanism. This makes installing and using Cypress easy. Cypress directly communicates with the AUT through the browser unlike other automation tools.

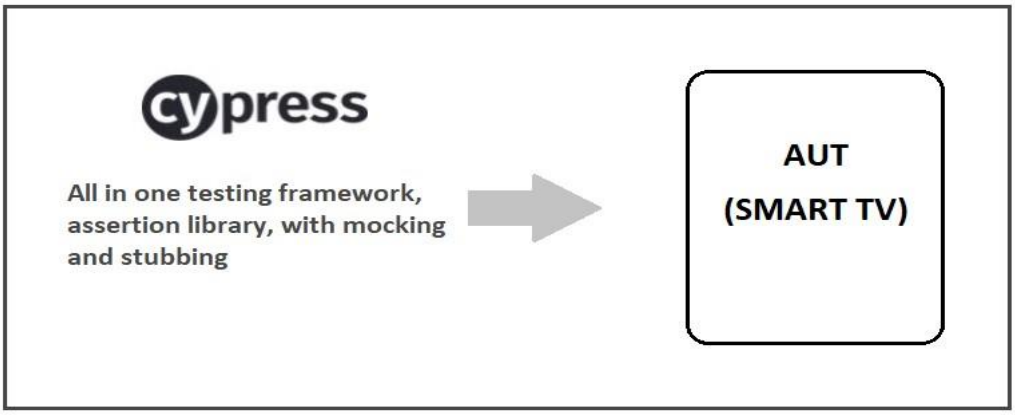

Fig. 3. **Test Automation framework with Cypress.io**

Cypress uses JavaScript as the programming language. Test automation scripts are built using the Page Object Model (POM). As a good test automation practices to improve maintainability and to maintain the modular structure of the project, page objects and tests are separated from each other [14].

2.7. Tools Required for Test Framework Development

Table 1 lists out the tools used for Test Framework Development and the purpose/role of each tool.

Table 1. Tools used for Test Framework Development

SI. No.	Tool	Purpose
1	Cypress.io	Web test automation
2	Visual Studio Code 1.62 IDE	Integrated Development Environment
3	GitHub	Source code management
4	Jenkins	Automation tool used for CI purposes
5	Node JS	JavaScript runtime environment
6	Mocha chai	Mocha is a JavaScript test framework and Chai is TDD assertion library

7	Node Server	The Node Server sets up an HTTP server that listens for server ports, also respond to clients
8	Run-Server	Static file server used on a specific port
9	Local storage manager	Used to modify local and session storage data
10	Allure	Test reporting extension
11	Jira	Agile Project Management tool
12	TestRail	TestRail is an end-to-end test management tool
13	Microsoft Office 2019	Documentation

2.8. Project Setup

Once the above tools have been installed and the setup has been completed, a test folder, as in the Figure 4(a) will be created under the root folder (Smart-TV) of UI Project for the AUT (Smart TV application). As highlighted in Figure 4(a) Page Object Model has been adopted and the tests are separated from the page.

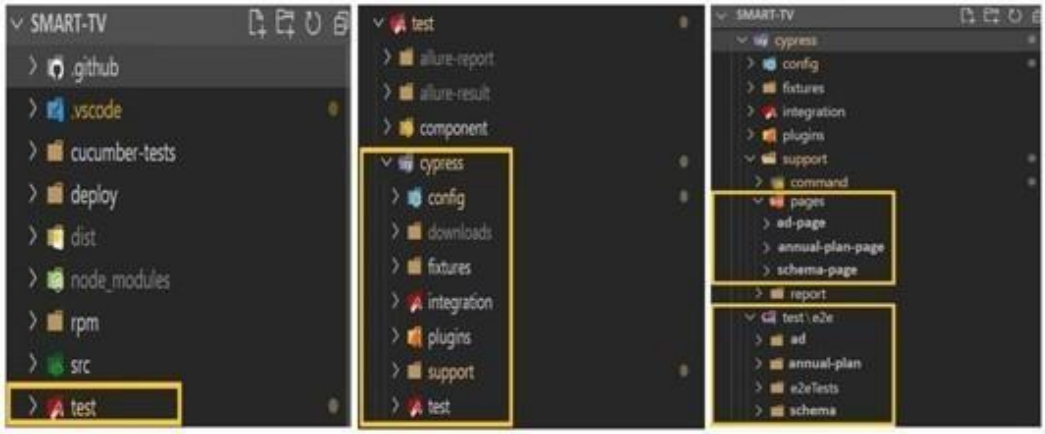

Figure 4. (a, b, c) Smart-TV Project - Folder Structure

Within the test folder there is another folder named Cypress that contains another 2 folders test and support Figure 4(b). All class implementations are done within pages folder inside the support folder and all tests are implemented in tests folder Figure 4(c). This way pages and tests are separated from each other.

2.9. Code Snippet

In this project Cypress tests are implemented using Mocha chai framework. Mocha is a JavaScript test framework. It is used to plan and perform tests, and it supports asynchronous testing as well. Chai is a test-driven development (TDD) assertion library for NodeJS and the browser. There are two main function calls needed when

writing a test using Mocha chai framework: describe () and it (). In Mocha, describe () is merely a technique to arrange tests. It () is used to run a single test case. Figure 5 depicts the code snippet for implementing tests and pages. In the Figure 5 (a) the test is implemented using describe () and it () functions, Figure 5 (b) depicts the class implementation of corresponding page.

```
import { AdDashboardPage, AdViewPages } from 'Pages';
import { InfoManager } from '../../../fixtures/info-manager';

const infoManager = new InfoManager();

describe('Ad Status', () => {
    beforeEach(() => {
        cy.loginAsPlanner();
        AdDashboardPage.visit();
    });
    const outline_00 = [
        {
            "Ad_status": "Completed"

        }
    ];
    outline_00.forEach(({ Ad_status }) => {
        it(`Ads in ${Ad_status} status do not have the Delete Ad button visible`, () => {
            Report.When('I open a Ad in ${Ad_status} status');
            AdViewPages.setMock(infoManager.getAdViewMockByStatus(Ad_status));
            cy.get(AdDashboardPage.BtnAnnualPlan).type(infoManager.getAdDetailMockByStatus(Ad_status)
            cy.contains(infoManager.getAdDetailMockByStatus(Ad_status).name).click();
            AdViewPages.waitForDataPreload();
                        Report.Then("I check the Ad status");
            cy.get('[data-automation="bottom-delete-Ad"]').should('not.exist');
        });
    });
});
```

```
class AdDashboard {
    public url = 'dashboard';

    BtnCreate = '.ad-global-create-modern__button-icon > .icon';
    BtnSTVAd = '[href="#/ad"]';
    LinkNavigation = '[automation="toggle"]'
    IconSearch = '[automation="ad-table"] > tbody > tr';
    BtnFinalDelete = '[automation="popup_apply_button"] > .ad-button';
    TxtTotal = ':nth-child(1) > .radio-button > .radio-button__el';

    commonAd: any;

    public visit() {
        Report.When('I visit STV Landing Page');
        cy.visit(this.url);
        cy.get(this.LinkNavigation).first().click();
        return this;
    }

    public clickCreateAdBtn() {
        cy.get(this.BtnSTVAd).click();
        return this;
    }
}

export const CampaignDashboardPage = new AdDashboard();
```

Figure 5. (a, b) Smart-TV Code Snippet Sample

Once tests are implemented using the page object model, open the cypress test window by executing the command ‘npx cypress open’ on the terminal. Upon successful completion of executing the command Cypress window opens as in Figure 6(a). Meanwhile, POM is a high-level abstraction that isolates web pages from test cases to increase code reuse. It minimizes the coupling between test cases and web pages, allowing them to be independent of one another and more easily reused in other sections of the code. Furthermore, POM implementation makes it easy to write test cases.

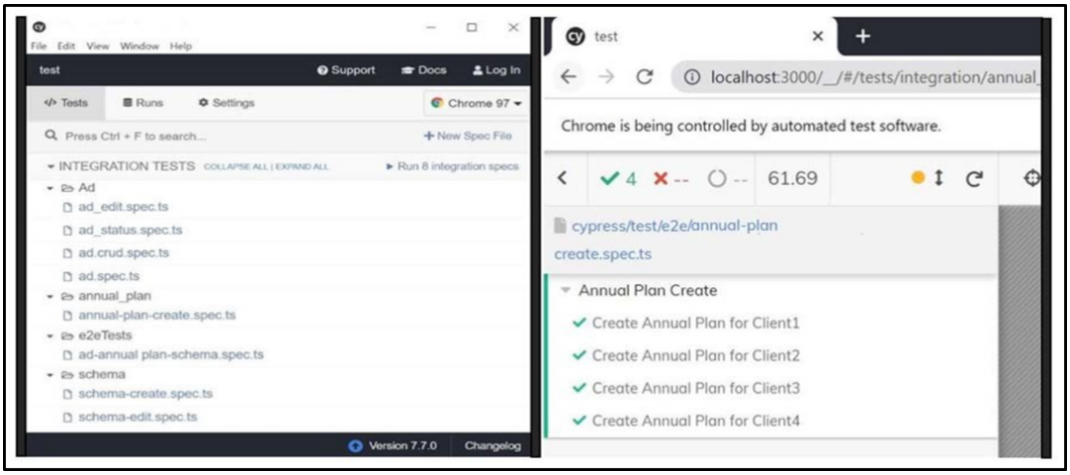

Figure 6. (a, b) Cypress Test Window

The test that needs to be executed can be triggered by double clicking on the test in the newly opened Cypress window. Test will start to execute and upon completion the test result will be shown as the output as in the Figure 6(b). In case of failure the execution window will provide details of the failure. The QE can also debug backtracking to the failed test. In addition to the debugging capabilities, Cypress also provides a feature for inspecting the elements on the webpage.

2.10. The implementation of CI process integrating the cypress.io test framework

The Continuous integration process flow with the Cypress implementation has been depicted in Figure 7. The steps in CI process with Cypress:

a) Developer creates a feature branch from the master branch.
b) Completes development of new feature and write unit/integration tests.
c) Creates a PR with WIP tag.
d) QE creates build in Jenkins and run regression tests locally.
e) QE writes test scripts and merge the changes to the same branch.
f) Run E2E tests in Jenkins.
g) Reviewer approves the PR, and the PR will be merged to the master.

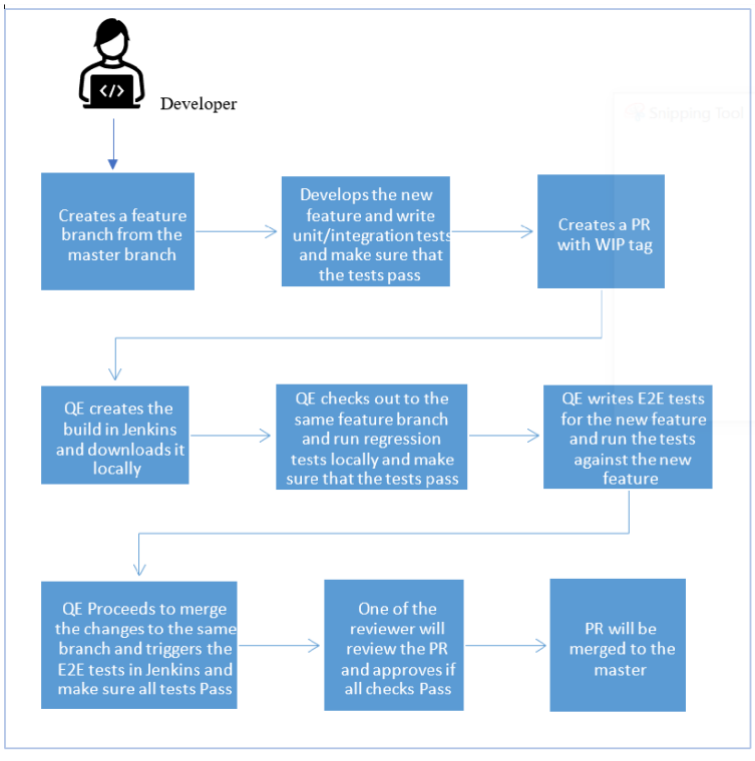

Figure 7. Continuous integration process flow with the Cypress implementation

The development and testing take place in the same cycle within the CI process. The frontend developer creates a branch locally in his/her machine and completes the development of the feature and verify that the unit and integration tests pass; Next the developer proceeds to push the changes to GitHub remote repository and creates a pull request (PR). In the PR, the developer would provide a comment WIP indicating that the feature is under work in progress and not ready to merge to the master. Next the developer will move the feature in the Project management platform Jira, from 'In progress' to 'Ready for Review' notifying QE team tagging PR as 'Ready for Review'. Figure 8 depicts a pull request with changes approved and ready to be merged to the Master.

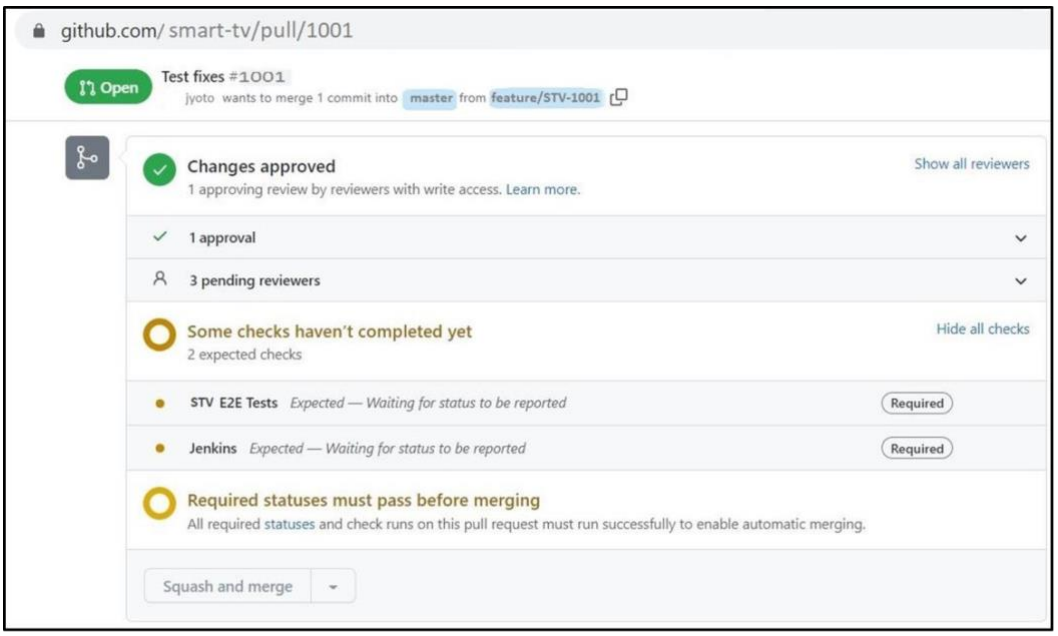

Figure 8. GitHub pull request

QE will create the build in Jenkins and download it. QE will also checkout to the same branch and run regression tests locally. QE develops the test script for the new feature in the same branch, The QE will test the newly developed feature by executing the tests. Once QE completes building tests and verification for the new feature, the changes will be merged to the same branch. QE will trigger the end-to-end test execution in Jenkins by providing a comment e2e test in the PR. This is achieved through GitHub and Jenkins integration. Once all the end-to-end test pass in Jenkins, QE will update the PR with a comment 'Ready for merge'. One of the reviewers within the project team will review the PR and approves it for merging if the changes meet all criteria and tests are passed. Then Dev/ QE will Merge the PR to the master. Now the developer can move the story in the Jira board from 'In Review' lane to 'In QA' lane tagging the build number in the comment. Figure 9 depicts the Jenkins job scheduled to run E2E tests.

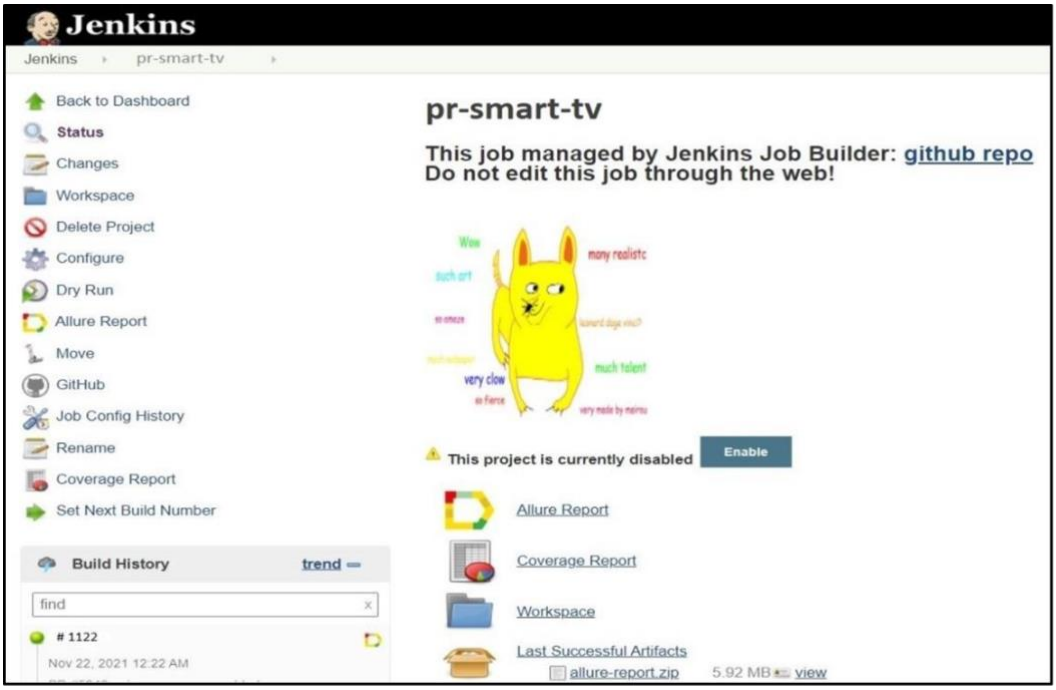

Figure 9. The Jenkins job scheduled to run E2E tests

The QE will execute the regression tests triggering a Jenkins job as shown in the Figure 9, and make sure that all the E2E tests passed on QA environment before the story gets moved to the Done column. In this approach the development phase and testing phases are integrated together. This ensure both Development and Test code goes to the master at the same time; also, it helps to ensure that each feature is tested before merging to master. Also, it helps to make sure that tests are maintained for every single change in the development code. The corresponding Jira process flow has been depicted in Figure 10.

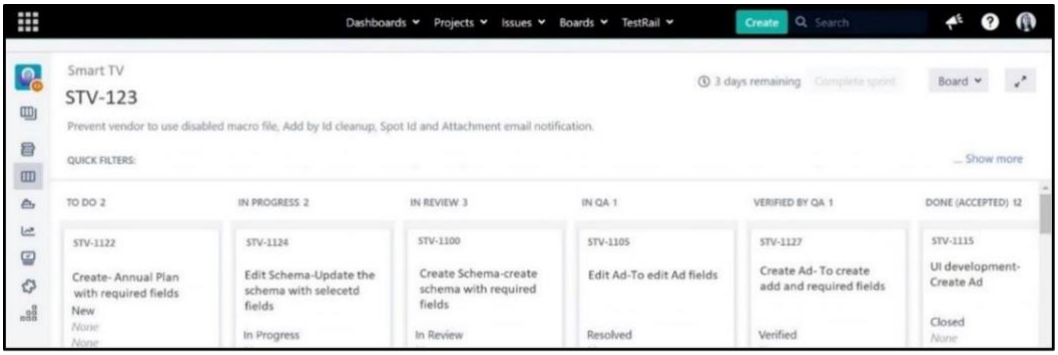

Figure 10. Jira workflow for a new feature

The Jira process contains six stages, starting from ‘New’, ‘In Progress’, ‘In Review’, ‘In QA’, ‘Verified by QA’, and ‘Done’ (Accepted).

2.11. Test Results and Reporting

Tests results reporting is achieved through integrating Allure extension to the test automation framework, Once the Jenkins job has been completed with the test execution Allure reports will be available in Jenkins. The test reports can be configured to share in email or other communication channels such as 'Slack', as per the stakeholder's preferences. Figure 11 depicts Allure Reporting for the Smart TV test execution which is integrated to Jenkins.

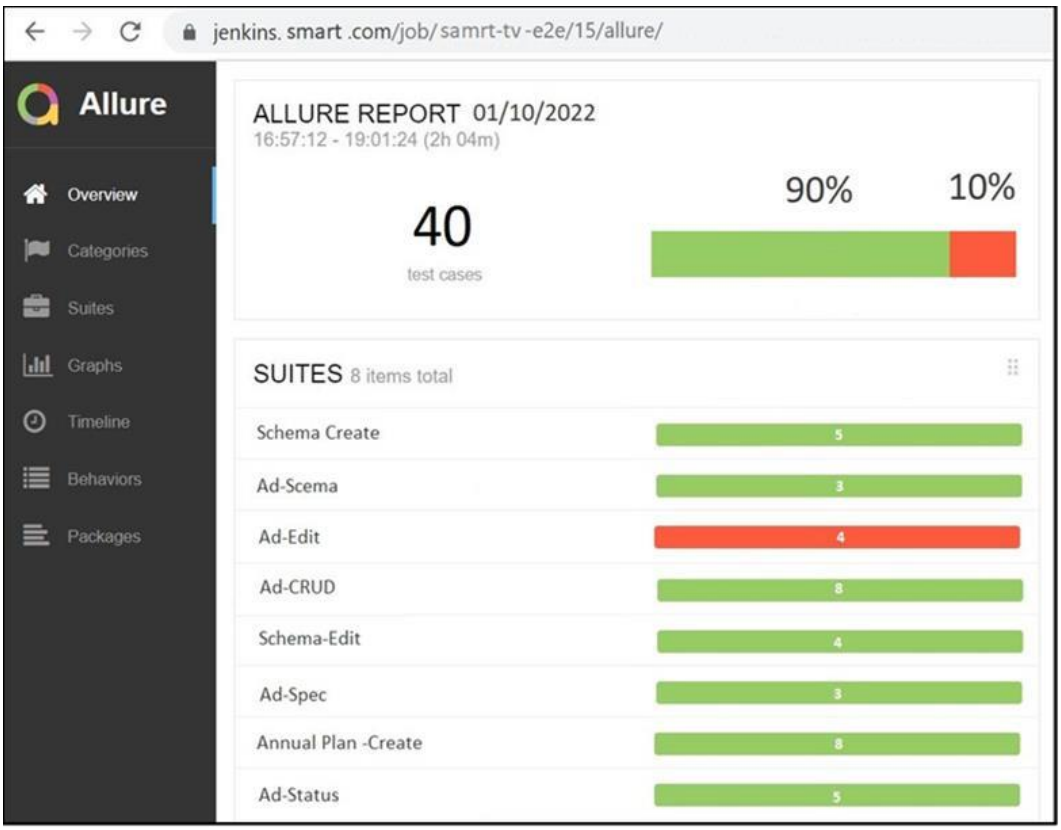

Figure 11. Allure Reporting for the Smart TV test execution

The above Figure 11 illustrates the test reporting with Allure for AUT test execution. In the above figure there are 40 test cases within 8 test runs. The 8 Test runs are: Schema Create, Ad-Schema, AdEdit, Ad-CRUD, Schema-Edit, Ad-Spec, Annual Plan-Create, Ad-Status. All 8 Test runs together contains 40 test cases out of which 36 tests (96%) passed in the current test execution. The specific test cases can be verified by double clicking on the test run.

3. Results

3.1. Automated End to End tests for Smart TV

There are 40 test cases identified and automated as part of this project. Tests are identified based on parameters detailed in section 2.4. Figure 12 illustrates the E2E automated tests for Smart TV.

Jira ID	TestRail ID	Title	Automation Status
STV-1001	C12101	Verify Create Ad Basic Flow and Poll result with-New audience	Automated
STV-1002	C12102	Verify Create Ad Basic Flow and Poll result with-old audience	Automated
STV-1003	C12103	Verify Create Ad Basic Flow and Poll result with cost allocation	Automated
STV-1004	C12104	Verify Create Ad Basic Flow and Poll result with-New audience and with cost allocation	Automated
STV-1005	C12105	Verify Create Ad Basic Flow and Poll result with-old audience and with cost allocation	Automated
STV-1006	C12106	Delete Ad that exist in the Smart TV system	Automated
STV-1007	C12107	The Ad is displayed on the Ad page	Automated
STV-1008	C12108	Verify Update Ad Basic Flow for Existing Audience	Automated
STV-1009	C12109	Verify Update Ad Basic Flow for New Audience-without anual plan	Automated
STV-1010	C12110	Verify Update Ad Basic Flow for Existing Audience-with anual plan	Automated
STV-1011	C12111	Verify Update Ad Basic Flow for New Audience-with anual plan	Automated
STV-1012	C12112	Verify Create Ad Basic Flow and DD-Schema create	Automated
STV-1013	C12113	Verify Create Ad Basic Flow and HH-Schema create	Automated
STV-1014	C12114	Verify Create Ad Basic Flow and OT-Schema create	Automated
STV-1015	C12115	Verify Create Ad Basic Flow and Custom-Schema create	Automated
STV-1016	C12116	Total Count is displayed on the Ad page	Automated
STV-1017	C12117	Field level validation on the create Ad page	
STV-1018	C12118	Ad Page- Duplicate name validation	Automated
STV-1019	C12119	Verify Ad Status- Draft	Automated
STV-1020	C12120	Ads in 'Completed' status do not have the Delete Ad button visible	Automated
STV-1021	C12121	Create-DD-Schemas-Audience Buy Custom SSP Without anualplan and Existing Audience-Save As Draft	Automated
STV-1022	C12122	Create-DD Schemas-Audience Buy Custom-Without anualplan and New Audience-Save As Draft	Automated
STV-1023	C12123	Create Schema-Audience Buy with Custom-Schemas and With anualplan-Save	Automated
STV-1024	C12124	Create-DD Schemas-Audience Buy Custom-Without anualplan and Existing Audience-Submit	Automated
STV-1025	C12125	Create-DD Schemas-Audience Buy Custom-Without anualplan and New Audience-Submit	Automated
STV-1026	C12126	Create Schema-Audience Buy with Custom-Schemas and With anualplan-Save-Poll Result	Automated
STV-1027	C12127	Create HH-Schema-Existing Ad-Draft	Automated
STV-1028	C12128	Create OT-Schema-Existing Ad - Ready for Review	Automated
STV-1029	C12129	Create HH-Schema-Existing Ad-In progress	Automated
STV-1030	C12130	Create HH-Schema-Existing Ad-Draft	Automated
STV-1031	C12131	Edit-Schemas-Audience Buy Custom-Without annual plan and Existing Audience	Automated
STV-1032	C12132	Edit-Schemas-Audience Buy Custom-Without annual plan and New Audience	Automated
STV-1033	C12133	Edit-Schemas-Audience Buy Custom-With annual plan and Existing Audience	Automated
STV-1034	C12134	Create Annual Plan with required fields	Automated
STV-1035	C12135	Annual Plan Override	Automated
STV-1036	C12136	Create Annual Plan- New Ad -Draft	Automated
STV-1037	C12137	Create Annual Plan-Existing Ad-Draft	Automated
STV-1038	C12138	Create Annual Plan- New Ad-In progress	Automated
STV-1039	C12139	Create Annual Plan-Existing Ad-In progress	Automated
STV-1040	C12140	Delete Annual Plan that exists in the Smart TV system	Automated

Figure 12. E2E automated tests for Smart TV

In Figure 12, there are 4 columns, Jira ID, TestRail Id, Title, and Automation Status. Requirements are stored in project management tool Jira and listed in the form of Jira Ids, each requirement has a unique Jira ID. Test cases are captured and listed in the form of Jira Ids, each test case has a unique TestRail ID. Title indicates the TestRail title of each test case. Automation Status used to indicate whether the test has been automated or not.

3.2. Requirements Traceability Matrix

Generally, the requirements are tracked using the Traceability Matrix which is also called Requirements Traceability Matrix (RTM). The requirements are matched against test cases, with RTM the test coverage and traceability are achieved.
In this project, the RTM is not maintained explicitly, however the requirement traceability is achieved integrating Project Management tool Jira and TestRail. Every User story/requirement in Jira will have a corresponding test case in TestRail. Requirements Traceability is achieved here within project and test management tool integrations. This ensures that all requirements are covered as tests and all tests are automated as part of the test automation project.

4. Conclusions

Cypress is an E2E web automation solution that is more dependable, faster, and allows concurrent development and testing. Until Cypress.io era, end-to-end testing was not easy, but with Cypress setting up, writing, running, and debugging tests are easier compared to existing tools, implementing the CI process also found to be successful with Cypress.io. Most testing tools run outside of the browser and execute remote commands over the network, but Cypress runs alongside the AUT in the same run loop. Cypress is powered by a Node.js server. Cypress and the Node.js process are in constant communication, synchronization, and task execution mode. The ability to respond to the application's events in real time is enabled by having access to both sections (front and back).
Cypress can also read and change web traffic on the fly at the network layer. This allows Cypress to change not only what comes in and out of the browser, but also code that could interfere with its ability to automate it. Cypress finally has complete control over the automation process, putting it in the unique position of being able to comprehend everything that happens inside and outside of the browser. As a result, Cypress can produce more consistent results than any other web testing tool. Cypress can also tap into the operating system for automation activities because it is installed locally on the DEV-QE workstation. This allows capturing screenshots, capture movies, and do general file system and network tasks. Cypress.io can be evaluated with below parameters:

a) Setting up tests: Setting up Cypress is easier compared to existing automation tools, no dependencies to be installed or configured, no servers or driver bindings.

b) Writing tests: Cypress tests are simple to read and understand for any third person, which makes it easier when new team members are added to the team.
c) Running tests: Cypress renders content as quickly as browser allows. One can watch tests run in real time as they work on the application.
d) Debugging failures: Debugging is easy in Cypress; the error messages are detailed and precise. Also, can back track to the failures upon clicking on it. Error messages are more readable compared to other tools.
e) Time travel: Cypress takes screenshots while the tests are running. To observe exactly what happened at each stage and for further inquiry, the test analyst can hover over the in the Command Log.
f) Realtime reloads: Cypress reloads the test automatically whenever it is changed. One may observe commands run in real time in the application.
g) Consistent results: Test results are stable, fast and flake free.
h) Debuggability: The test analyst does not need to do any guessing for the failures, debugging is possible with developer tools too.
i) Automatic waiting: There is no need for the test analyst to add additional waits in the code, Cypress automatically waits for commands before executing the next step.
j) Screenshots and videos: Screenshots and videos are available throughout the test, which makes it easier to track back in the event of a failure.

It is identified that most of the features and specialties mentioned in the Cypress.io official portal is very much useful when it comes to reality. Cypress.io has been identified as a promising web automation tool with the implementation of new framework.

References

[1] Vahid Garousi, Alper Buğra Keleş, Yunus Balaman and Zeynep Özdemir Güler Testinium A.Ş. (2020) 'Test automation with the Gauge framework: Experience and best practices', Part of the Lecture Notes in Computer Science book series (LNCS, volume 12250).
[2] Abdul Rauf EM and E. Madhusudhana Reddy. (2015) 'Software Test Automation: An algorithm for solving system management automation problems', Research and Development Centre, Bharathiyar University, Coimbatore641014, Tamil Nadu, India.
[3] Rosnisa Abdull Razak and Fairul Rizal Fahrurazi. (2011) 'Agile Testing with Selenium', 2011 Malaysian Conference in Software Engineering.
[4] Kai Presler-Marshall, Eric Horton, Sarah Heckman, Kathryn T. Stolee. (2019) 'Wait Wait. No, Tell Me. Analyzing Selenium Configuration Effects on Test Flakiness', 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), Montreal, QC, Canada.
[5] Alok Mishra and Ziadoon Otaiwi. (2020) 'DevOps and software quality: A systematic mapping', Faculty of Logistics, Molde University College-Specialized University in Logistics, Molde, Norway.
[6] Mubarak Albarka Umar and Chen Zhanfang. (2019) 'A Study of Automated Software Testing: Automation Tools and Frameworks', 1,2 School of Computer Science and Technology, Changchun University of Science and Technology.
[7] Fatini Mobaraya and Shahid Ali. (2019) 'Technical analysis of Selenium and Cypress as Functional Automation Framework for Modern Web Application Testing', Department of Information Technology, AGI Institute, Auckland, New Zealand.
[8] Satish Gojarea, Rahul Joshib and Dhanashree Gaigaware. (2015) 'Analysis and Design of Selenium WebDriver Automation Testing Framework', 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15).
[9] Maurizio Leotta1, Diego Clerissi1, Filippo Ricca1 and Cristiano Spadaro. (2013) 'Repairing Selenium Test Cases: An Industrial Case Study about Web Page Element Localization', 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg.

[10] Eliane Figueiredo Collins and Vicente Ferreira de Lucena Jr. (2012) 'Software Test Automation Practices in Agile Development Environment: An Industry Experience Report', 2012 *7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland*.
[11] Abdullah, N. N. B., Honiden, S., Sharp, H., Nuseibeh, B., & Not- kin, D. (2011) 'Communication Patterns of Agile Requirements Engineering', Proceedings of the 1st Workshop on Agile Requirements Engineering (pp. 1–4).
[12] Alexey Ieshin, Marina Gerenko and Vadim Dmitriev. (2009) 'Test Automation: Flexible Way', 2009 5th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR). Moscow, Russia.
[13] Megan Sumrell. (2007) 'From Waterfall to Agile – How does a QA Team Transition?', Agile 2007 (AGILE 2007), Washington, DC, USA
[14] Sandeep Sivanandan and Yogeesha C. B. (2014) 'Agile Development Cycle: Approach to Design an Effective Model Based Testing with Behavior Driven Automation Framework', 20th Annual International Conference on Advanced Computing and Communications (ADCOM).
[15] Michel Nass, Emil Alégroth and Robert Feldt. (2020) 'Why many challenges with GUI test automation (will) remain', H. SERL, Blekinge Institute of Technology, Sweden, P. SERL, Blekinge Institute of Technology, Sweden, Chalmers University of Technology, Sweden.