Oct 30th, 9:00 AM - 10:30 AM

Software Automated Testing using BDD Approach with Cucumber Framework

Arbesë Musliu
University for Business and Technology - UBT, am46107@ubt-uni.net

Xhelal Jashari
University for Business and Technology - UBT, xhelal.jashari@ubt-uni.net

Software Automated Testing using BDD Approach with Cucumber Framework

Arbesë Musliu[1], Xhelal Jashari[2]

[1,2]UBT, Prishtina, Kosovo
am46107@ubt-uni.net[1], xhelal.jashari@ubt-uni.net[2]

Abstract. Software testing is becoming essentially needed nowadays, but at the same time is becoming more complex and taking more resources. Software development using agile methodology helps us deliver a considered number of versions in a short amount of time. However, using these methodologies means that we have to repeatedly test each version by testing all functionalities, including the updated ones. This research paper will show the importance of Automation Testing using the Cucumber framework and behavior driven development (BDD) approach to test new features and maintain the previous ones. We will show the implementation of the Cucumber framework using Selenium WebDriver with Java programming language using BDD scenarios. This paper provides real-life examples of how we can test and automate software testing in best practices. At the same time, the following process relates to test plan, previous and actual test cases, test analyses, test environment, test execution, and test reporting. By developing those examples, automation tools are necessary process and low cost to maintain the developed software and offer good quality products in the software applications industry.

Keywords: Software Testing, Automation Tools, Cucumber, Selenium, WebDriver, BDD, Quality Assurance

1. Introduction

Requirements changing, need to go faster to market, delivering product in short amount of time are some of the main factors when it comes to Software Development. Testing is an essential part of any software development life cycle model [9]. Being able to deliver a bug free software is very hard when you work on agile methodology. Even after you have tested the software in all procedures and processes inside your company it is hard to maintain the software due to requirements change and new features added from client requests.

This is especially true for regression testing due to its normal growth because today's testing of a new feature will become future's regression test. To handle with increasing demand on frequent testing, usually with small amount of resources, the test automation is an important strategy to deal with this challenge. But, what basically is Software Automation Testing? Software automation testing is Software testing but now you will be using tools and software so you can test a Software. Let's dive a bit more into it and make it even simple. Automation testing is writing test cases on scripts that you can execute in order to test software products. Nowadays you can use a lot of tools and frameworks to create and run scripts.

The transition from manual testing to automation testing it's another thing that software companies that work in an agile environment are facing as an immediate and important factor to grow and deliver quality at the same time. We have explored and will give you a deep understanding of BDD Cucumber Framework as a very good practice of transition from manual testing to automation testing.

An automated test framework may be loosely defined as a set of abstract concepts, processes, procedures and environment in which automated tests will be designed, created and implemented. In addition, it includes the physical structures used for test creation and implementation, as well as the logical interactions among those components [2].

One thing that we should state is that automation testing cannot be done without manual testing, but automation testing comes in play when you have to maintain big software's, instead of repeating manually all the test cases when you add a new feature you can create a regression suite and add there all the test cases so whenever you need you just can execute regression suite.

Another main thing for nowadays companies are having great communication and understanding between technical and non-technical people within a company. That's when BDD Cucumber Framework stands up from the rest, with the usage of the Gherkin language we are able to write test scenarios that can be understood also by non-technical stakeholders like Product Owners, Business Analysts, Project Managers, etc.
Besides, when automation testing is needed in this paper we will also show the situation when we do not need the automation testing such as small projects, need of a lot of resources and the cost is high and there is no return in investment.

2. Literature Review

What is Software Testing?
Software Testing means the process of defining Test Stories (or Test Scenarios), each containing Test Cases, and executing them with the aim of detecting unexpected behavior.[9] As per ANSI/IEEE 1059, Testing in Software Engineering is a process of evaluating a software product to find whether the current software product meets the required conditions or not.
Here are the advantages of using software testing:
Cost-Effective: It's one among of the most important advantages of software testing. Testing any IT project on time helps you to avoid wasting your money for the long run. Just in case if the bugs caught within the earlier stage of software testing, it will have less cost fixing that bug.
Security: It is the foremost vulnerable and sensitive advantage of software testing. People are in need for trusted products. It helps in removing risks and problems earlier.
Product quality: It is a basic requirement of any software product. Testing makes sure a quality product is delivered to customers.
Customer Satisfaction: The aim of any product is to bring satisfaction to their customers. UI/UX Testing ensures the most satisfying user experience.

What is and what are the types of Automated Testing
Automated software testing is the ability to have a software tool or suite of software tools test your applications directly without human intervention. Generally test automation involves the testing tool send data to the application being tested and then compare the results with those that were expected when the test was created [6].
The test automation can be applied on both black box and white box testing types. The process of testing the executable program without referring the source code is called black box testing. White box testing deals with executing special test cases, in specific branches and paths in source code. Numerous types of test automations are available under the black box and white box testing labels. Few types of test are not feasible to perform manually [2].

Functional Test
The functional tests have many responsibilities, as it is kind of black box testing and main aim of these tests is to make sure that the system meet the customers' requirements. Furthermore, the functional tests are performed to ensure that the goals and metrics have been met, such as the systems performance metrics. The functional test, are often replacements of manual black box testing. The test cases are executed that produces the documentations in the form of graphs and results summaries and the tools provide the ability to produce statistical results on demand. Various kinds of tests fall under the umbrella of functional test [2].

Regression Testing
Regression Testing is a sort of testing that is performed to ensure that a code change within software does not crash the existing functionality of the product. This is to make sure that the software works fine with new functionality, bug fixes or any changes to the existing feature. Previously executed test cases are re-executed so we can verify the impact of the change.

Unit Testing
The Unit testing part of a testing methodology is the testing of individual software modules or components that make up an application or system. These tests are usually written by the developers of the module and in a test-driven-development methodology (such as Agile, Scrum or XP) they are actually written before the module is created as part of the specification. Each module function is tested by

a specific unit test fixture written in the same programming language as the module [7].

User Interface (UI) Testing

In an ideal world, the presentation layer would be very simple and with sufficient unit tests and other code-level tests (e.g. API testing if there are external application program interfaces (APIs)) you would have complete code coverage by just testing the business and data layers.[5] Unfortunately, reality is never quite that simple and you often will need to test the Graphic User Interface (GUI) to cover all of the functionality and have complete test coverage. That is where GUI testing comes in [7].

API Testing

API testing involves testing the application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer.[6] API testing is critical for automating testing because APIs now serve as the primary interface to application logic and because GUI tests are difficult to maintain with the short release cycles and frequent changes commonly used with Agile software development and DevOps.[8]

Performance, Load, Stress Testing

There are several different types of performance testing in most testing methodologies, for example: performance testing is measuring how a system behaves under an increasing load (both numbers of users and data volumes), load testing is verifying that the system can operate at the required response times when subjected to its expected load, and stress testing is finding the failure point(s) in the system when the tested load exceeds that which it can support.

Manual versus Automated Testing

In manual testing, the tester assumes the role of a user executing the system under test (SUT) to verify its behavior and ¬any observable defects. In automated testing, developers develop test code scripts (for example, using the Selenium framework) that execute without human intervention to test the SUT's behavior. If planned and implemented properly, automated testing can yield various benefits over manual testing, such as repeatability and reduced test costs (and thus effort). However, if not implemented properly, automated testing will increase costs and effort and could even be less effective than manual testing.

Selenium

Selenium is possibly the most popular open-source test automation framework for Web applications[2]. It originated in 2000s and evolved over a decade. Selenium has been an automation framework of choice for Web automation testers, especially for those who possess advanced programming and scripting skills. Selenium is a core framework for other open-source test automation tools such as Katalon Studio, Watir, Protractor, and Robot Framework [2].

Selenium supports multiple system environments such as Windows, Mac, Linux, and browsers such as Chrome, Firefox, IE, and Headless browsers. The scripts can be written in various programming languages such as Java, Groovy, Python, C#, PHP, Ruby, and Perl.

While testers have flexibility with Selenium, and they can write complex and advanced test scripts to meet various levels of complexity, it requires advanced programming skills and effort to build automation frameworks and libraries for specific testing needs.

Selenium WebDriver is the greatest change in Selenium recently. Selenium-WebDriver makes direct calls to the browser using browser's native support for automation.

To install Selenium means to set up a project in a development so a program can be written using Selenium. Setting of the driver depends on the chosen programming language and the development environment.

The easiest way to set up a Selenium 2.0 Java project is to use Maven. Maven will download the Java bindings - the Selenium 2.0 Java client library and all its dependencies. This will create the project, using a maven pom.xml (project configuration) file. Once these steps are executed, the maven project is ready to be imported into the preferred IDE, IntelliJ IDEA or Eclipse [2].

Behavior Driven Development

The key to success in BDD lays with the execution of the acceptance tests which describe in an easily understood and easily definable manner a scenario which consists in defining the context, the executed actions and the expected response.

BDD (Behavior Driven Development) – is a growing methodology which relies on the principles of Agile. It was developed by Dan North [1] as an answer to the principles promoted by the TDD (Test Driven Development) approach. BDD uses a level of abstraction which allows for the use of the framework by nontechnical people. The level of abstraction consists in a DSL (Domain Specific Language) called the Gherkin language.

In BDD, the scenarios of the acceptance tests are explicitly defined by the following syntax [4]:

Given <defining an initial context>
When <executing certain actions or steps>
Then <defining the expected results>

The requirements of the software applications or the behavior of the application are defined using the former structure, where the keywords, Given, When, Then are defined as Steps. Each set of steps is parsed and executed by a specific BDD Framework which transforms the business requirements into technical details, code classes and test methods.

This scenario format is proving very useful in aligning knowledge and understanding the common principles among programmers, testers and the final client. Moreover, these steps will serve as documentation for the developed application.

What is Cucumber?

Cucumber is a tool that supports Behaviour-Driven Development(BDD). If you're new to Behaviour-Driven Development read our BDD introduction first.

Cucumber reads executable specifications written in plain text and validates that the software does what those specifications say. The specifications consists of multiple examples, or scenarios [13]. For example:

```
Scenario: Breaker guesses a word
  Given the Maker has chosen a word
  When the Breaker makes a guess
  Then the Maker is asked to score
```

Figure 1Scenario example taken from Cucumber.io[13]

Each scenario is a list of steps for Cucumber to work through. Cucumber verifies that the software conforms to the specification and generates a report indicating ✓ success or ✗ failure for each scenario [13].

In order for Cucumber to understand the scenarios, they must follow some basic syntax rules, called Gherkin [13].

What is Gherkin?

Gherkin is a set of grammar rules that makes plain text structured enough for Cucumber to understand. The scenario above is written in Gherkin [13]. Gherkin serves multiple purposes [13]:

Unambiguous executable specification;
Automated testing using Cucumber;
Document how the system actually behaves;
Single source of Truth;

The Cucumber grammar exists in different flavours for many spoken languages so that your team can use the keywords in your own language [13]. Gherkin documents are stored in *.feature* text files and are typically versioned in source control alongside the software.[13]

What are Step Definitions?

Step definitions connect Gherkin steps to programming code. A step definition carries out the action that should be performed by the step. So step definitions hard-wire the specification to the implementation.[13]

Step definitions can be written in many programming languages. Here is an example using JavaScript:

```
When("{maker} starts a game", function(maker) {
  maker.startGameWithWord({ word: "whale" })
})
```

What is Scrum?

Scrum is a framework that helps teams work together. Much like a rugby team (where it gets its name) training for the big game, scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve [14]. Others see Scrum as a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value [15]. Scrum is best suited in the case where a cross functional team is working in a product development setting where there is a non-trivial amount of work that lends itself to being split into more than one 2 – 4 week iteration [16].

3. Problem Declaration

The aim of all companies is that all of the software releases go to production without a glitch, without defects, where all stakeholders are informed of the outcomes and current status.

Today we have too many software releases in a short amount of time and we can see that while companies are rushing to deliver they also are dropping software failures. If we ignore this problem at some point resources will need to increase in order to handle the not tracked issues or bugs, and we may miss critical client deadlines which have consequences in lost revenue, penalties, lost business, and further damage to our software quality. Another big issue for big companies nowadays is maintaining the quality of the existing products while they add new features to that product.

We will try to give a solution to those problems by using the BDD Cucumber framework that has a very good approach to handling functional tests in an automated way.

4. Methodology

In this chapter we present the approach of the work and the methods used in this paper that aims to solve the problems stated above. We based our methodology on literature browsing, web application development and testing:

Literature Review. - The sources used are mainly secondary data sources taken from the literature and earlier research in the faculty, relevant books, scientific journals, and as well articles by authors who already have a high academic baggage.

Case Study.- We developed a Testing framework from scratch in order to test and compare results. Our Framework consists of BDD Framework with a set of technologies like Java Programming language, Selenium WebDriver, Cucumber, etc.

5. Our Case study and Results

1.First we create a WebDriver class which is called as Selenium WebDriver and is an interface that defines a set of methods. However implementation is provided for by the browser specific classes. The WebDriver main functionality is to control the browser. It even helps us to select the HTML page elements and perform operations on them such as click, filling a form fields etc. The methods appeared below contains all the necessary methods for different browsers specific classes, which from those methods we do not need to download the WebDriver file of a specific browser but we automatically already have the latest version of those versions by using those methods. If we want to execute test cases in Google chrome, we use ChromeDriver and so on and so far for the other types of browser. There's more to indicate for this class, we actually can run test cases in headless which mean that a headless browser is a term used to define browser simulation programs that do not have a GUI. These programs execute like any other browser but do not display any UI. In headless browsers, when Selenium tests run, they execute in the background.

2.Cucumber is a Testing Framework which allows us to create an structure. In our case I create in java folder some main folders as:

GeneralUtils – which includes config files with methods that are going to be used in steps definitions

Page objects – which includes page elements, page object with methods that will used in page elements classes

Selenium WebDriver – includes the webdriver classes

Resources which contains:

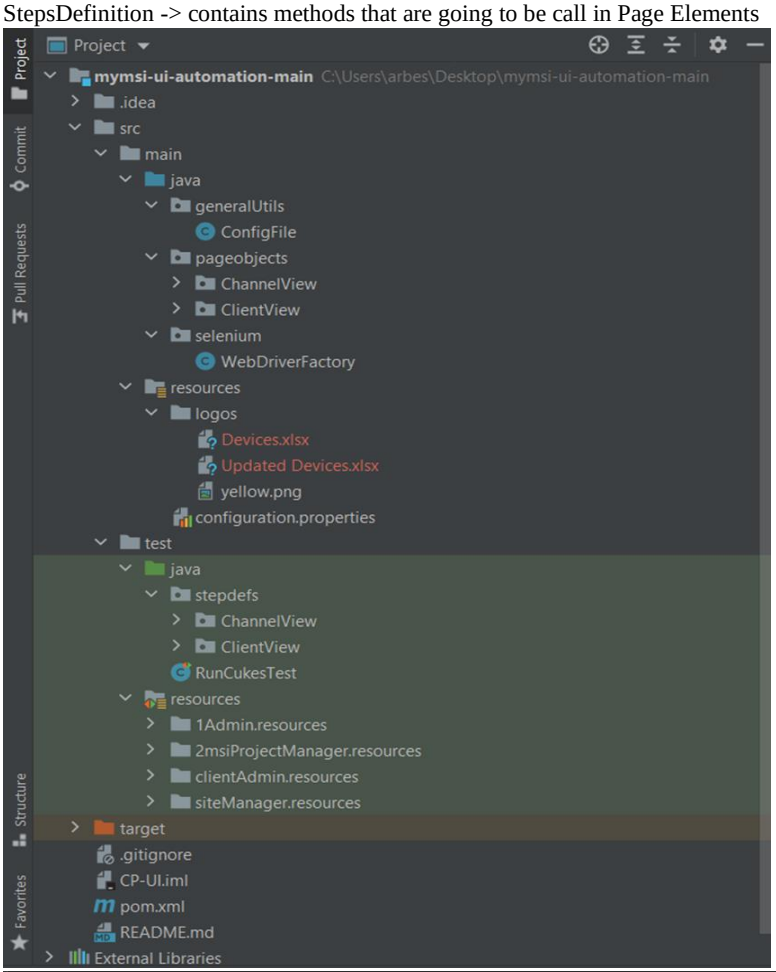
Files – includes, photos, files, excel template etc.,

Configuration properties – includes url, credentials, browser type

Until In test folder are some other main folders as:

Features -> contains bdd scenarios (Test cases that gonna execute)

StepsDefinition -> contains methods that are going to be call in Page Elements




```java
public enum AdminPanelPageElements implements PageObjects{

    users(PageObjectUtils.LocatorType.XPATH, id: "//button//span[contains(text(), 'Users')]"),
    OnASpecificUser(PageObjectUtils.LocatorType.XPATH, id: "//td[contains(text(), 'Arbese Musliu')]"),
    addcategory(PageObjectUtils.LocatorType.XPATH, id: "//button[contains(text(), 'Add Category')]"),
    addcategoryandtpyes(PageObjectUtils.LocatorType.XPATH, id: "//button[contains(text(), 'Add Category & Type')]"),
    categoryDropD(PageObjectUtils.LocatorType.XPATH, id: "//mat-select[@name='category']"),
    categoryField(PageObjectUtils.LocatorType.XPATH, id: "//input[@formcontrolname='newCategory']"),
    categoryName(PageObjectUtils.LocatorType.XPATH, id: "//span[contains(text(), 'New Category')]"),
    typeDropd(PageObjectUtils.LocatorType.XPATH, id: "//input[@name='type']"),
    categoryNameDD(PageObjectUtils.LocatorType.XPATH, id: "//span[contains(text(), 'Temperature Controls')]"),
    userProjects(PageObjectUtils.LocatorType.XPATH, id: "//button//span[contains(text(), 'Projects')]");



    public PageObjectUtils.LocatorType locator;
    public String id;

    AdminPanelPageElements(PageObjectUtils.LocatorType locator, String id) {
        this.locator = locator;
        this.id = id;
    }

    public void initializeMap() {}

    public WebElement getElement() { return PageObjectUtils.locateElement(driver, locator, id); }


    public By getBy() { return PageObjectUtils.locateBy(locator, id); }
```

```java
package pageobjects.ChannelView;

import ...
public class PageObjectUtils {

    public static void isElementVisible(WebDriver driver, By by, int seconds) {

        WebDriverWait wait = new WebDriverWait(driver, seconds);
        wait.until(ExpectedConditions.visibilityOfElementLocated(by));
    }

    public static void elementIsNotVisible(WebDriver driver, By by, int seconds) {

        WebDriverWait wait = new WebDriverWait(driver, seconds);
        wait.until(ExpectedConditions.invisibilityOfElementLocated(by));
    }

    public static void isElementClickable(WebDriver driver, By by, int seconds) {

        WebDriverWait wait = new WebDriverWait(driver, seconds);
        wait.until(ExpectedConditions.elementToBeClickable(by));
    }

    public static void waitForNumberOfElements(WebDriver driver, By by, int numberOfElements, int seconds) {

        WebDriverWait wait = new WebDriverWait(driver, seconds);
        wait.until(ExpectedConditions.numberOfElementsToBe(by,numberOfElements));
    }
```

```java
package pageobjects.ChannelView;

import ...
public interface PageObjects {
    WebDriver driver = WebDriverFactory.getInstance().getWebDriver();

    void initializeMap();
}
```

3.Writting test Cases. - After creating the project structure, we start to write test cases that are going to be automated. Test cases are written in Gherkin Language. We create every feature for every class, every feature contains multiple scenarios. Scenarios is one of the core Gherkin structures. Consider a case, where we need to execute a test scenario more than once. Suppose, we need to make sure that the login functionality is working for all types of subscription holders. That requires execution of login functionality scenario multiple times. Copy paste the same steps in order to just re-execute the code, does not seem to be a smart idea. For this, Gherkin provides one more structure, which is scenario outline.

File - C:\Users\arbes\Desktop\mymsi-ui-automation-main\src\test\resources\1Admin.resources\ICDevices.feature

```
 1 Feature: Test IC Module
 2
 3 Background: Steps That execute before every scenario
 4 Given User Open Browser and goes to specify
   ChannelUrl
 5 When User Enter KodeAdmin as username and Password as
    password and click Login
 6
 7
 8   @SmokeTest @IC @Devices
 9   Scenario: Apply Fitler when there's no device
10   When Click on selection apps icon
11   And Click on Mymsi
12   Then Check if user is at MyMsi app
13   And User clicks on the added Project
14   Then check if user is at project details
15   And msi admin clicks on Integration Checklist
16   Then Check if user is at IC page
17   And msi admin clicks on added IC
18   And check if user is at IC details
19   And user clicks on Devices tab
20   Then check if user is at Devices page
21   And user clicks on Filters
22   And Apply filter by manufacturer kode
23   Then check if the message is appeared
24   And Apply filter by Gateway gateway
25   Then check if the message is appeared
26   And Apply filter by protocol
27   And user clicks on Filter button
28   Then check if uno devices is displayed
29
30   @SmokeTest @IC @Devices
31   Scenario: Check if export button is disabled when
   there's no devies
32     When Click on selection apps icon
33     And Click on Mymsi
34     Then Check if user is at MyMsi app
35     And User clicks on the added Project
36     Then check if user is at project details
37     And msi admin clicks on Integration Checklist
38     Then Check if user is at IC page
```


4.After creating features, we **create steps definitions** for that features, so for every step in scenarios we create method.

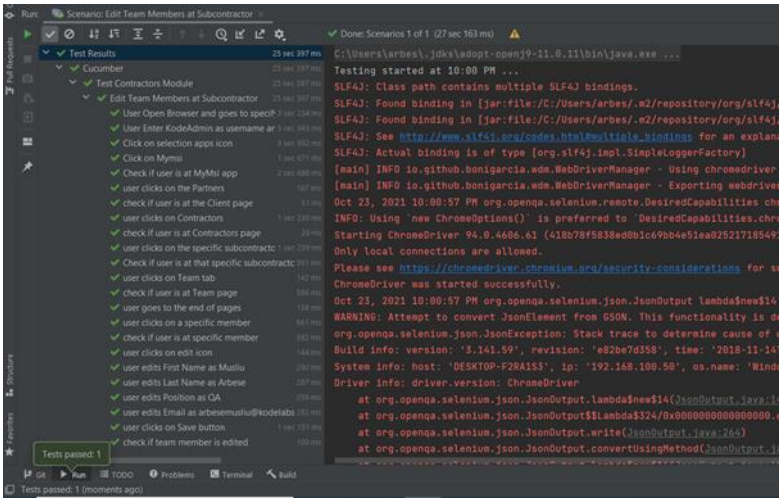
File - C:\Users\arbes\Desktop\mymsi-ui-automation-main\src\test\java\stepdefs\ChannelView\ICSteps.java

```java
package stepdefs.ChannelView;

import io.cucumber.java.en.And;
import io.cucumber.java.en.Then;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import pageobjects.ChannelView.*;

import java.util.List;

public class ICSteps extends CommonStepObjects {

    @Then("^msi admin clicks on choose device button$")
    public void msi_admin_clicks_on_choose_device_button() throws Throwable {
        PageObjectUtils.isElementClickable(driver, ICPageElements.choose_device_button.getBy(), 10);
        ICPageElements.choose_device_button.getElement().click();
    }

    @And("^msi admin clicks on Integration Checklist$")
    public void msi_admin_clicks_on_integration_checklist() throws Throwable {
        PageObjectUtils.isElementClickable(driver, ICPageElements.on_integration_checklist.getBy(), 10);
        ICPageElements.on_integration_checklist.getElement().click();
    }

    @And("^msi admin clicks on add IC button$")
    public void msi_admin_clicks_on_add_ic_button() throws Throwable {
        PageObjectUtils.isElementClickable(driver, ICPageElements.on_add_ic_button.getBy(), 10);
```

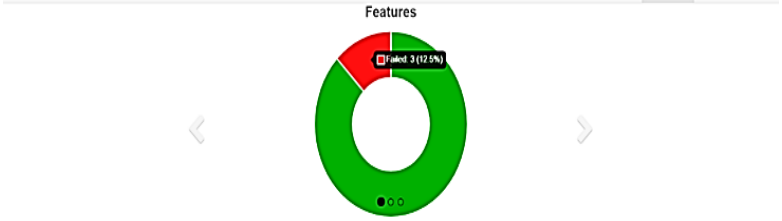

5.After that, we call all features in **Cukes Runner Class**, so we can execute them all together.

```java
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.AfterClass;
import org.junit.runner.RunWith;
import stepdefs.ChannelView.CommonStepObjects;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty","json:target/cucumber.json"},
        features = {"src/test/resources"},
        glue = "stepdefs"
)

public class RunCukesTest {

    @AfterClass
    public static void TearDown() { CommonStepObjects.closeDriver(); }
}
```

6.Results after running tests, we can see the results that will show each test case if it pass or failed. Also it will show time needed for execution of each test case and overall execution of tests



7.Jenkins Plugin.- We can use a Jenkins Plugin that works with Cucumber so we can generate different reports. Those reports can be configured to run on specified time. For example it's a good practice to run all the test in the morning and when you go to work you can check if your app is up and running.



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Apps	18	0	0	0	0	18	2	0	2	9.404	Passed
Client Apps	11	0	0	0	0	11	2	0	2	3.514	Passed
Client Test	29	3	9	0	0	41	1	3	4	42.978	Failed
Clients	245	3	9	0	0	257	12	3	15	1:43.313	Failed
Edit Profile	19	0	0	0	0	19	2	0	2	6.405	Passed
Homepage	6	0	0	0	0	6	2	0	2	8.712	Passed
Login Tests	13	0	0	0	0	13	3	0	3	9.441	Passed
Release Log	51	0	0	0	0	51	3	0	3	13.783	Passed
Sites Apps	25	0	0	0	0	25	2	0	2	7.549	Passed
Test Apps Module	20	0	0	0	0	20	2	0	2	5.919	Passed

6. Conclusion

References

[1] D. Gafurov, A. E. Hurum and M. Markman, "Achieving Test Automation with Testers without Coding Skills: An Industrial Report," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 749-756, doi: 10.1145/3238147.3240463.

[2] Maheen, Khutaija & Hameed, Kauser & Asif, Amna. (2019). Software Test Automation.

[3] R. Broer Bahaweres et al., "Behavior-driven development (BDD) Cucumber Katalon for Automation GUI testing case CURA and Swag Labs," 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), 2020, pp. 8792,doi:10.1109/ICIMCIS51567.2020.9354325.

[4] A. Contan, L. Miclea and C. Dehelean, "Automated testing framework development based on social interaction and communication principles," 2017 14th International Conference on Engineering of Modern Electric Systems (EMES), 2017, pp. 136-139, doi: 10.1109/EMES.2017.7980399.

[5] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, pp. 77-81, doi: 10.1109/ICECDS.2017.8389562.

[6] Y. Labiche, "Test Automation - Automation of What?," 2018 IEEE

International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2018, pp. 116-117, doi: 10.1109/ICSTW.2018.00037.
[7] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," in IEEE Software, vol. 34, no. 2, pp. 90-96, Mar.-Apr. 2017, doi: 10.1109/MS.2017.34.
[8] C. Klammer and R. Ramler, "A Journey from Manual Testing to Automated Test Generation in an Industry Project," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 591-592, doi: 10.1109/QRS-C.2017.108.
[9] Fehlmann, Thomas & Kranich, Eberhard. (2020). A Framework for Automated Testing. 10.1007/978-3-030-56441-4_20.
[10] Curreri, Matthew. "Automation Test Blockchain: 2019 IEEE Automation Test Conference." 2019 IEEE AUTOTESTCON (2019)
[11] Axelrod, Arnon. (2018). Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. 10.1007/978-1-4842-3832-5.
[12] Chaubal A. Pinakin, Mastering Behavior-Driven Development Using Cucumber: Practice and Implement Page Object Design Pattern, Test Suites in Cucumber, POM TestNG Integration, Cucumber Reports, and work with Selenium Grid (English Edition). India, BPB Publications, 2021, ISBN:9789391030476, 9391030475.
[13] Cucumber.io. 2021. Introduction - Cucumber Documentation. [online] Available at:<https://cucumber.io/docs/guides/overview/> [Accessed 23 October 2021].
[14] Atlassian. 2021. Scrum - what it is, how it works, and why it's awesome. [online] Available at: <https://www.atlassian.com/agile/scrum> [Accessed 23 October 2021].
[15] Scrum.org, 2021, What is Scrum?, [online] Available at: <https://www.scrum.org/resources/what-is-scrum> [Accessed 23 October 2021].
[16] Agile Alliance |. 2021. What is Scrum?. [online] Available at: <https://www.agilealliance.org/glossary/scrum/> [Accessed 23 October 2021].