Klover Multidisciplinary Journal of Engineering
Volume 10 Issue 2, April-June 2022
ISSN: 2995-4118
Impact Factor: 6.40
http://kloverjournals.org/journals/index.php/Engineering

INNOVATIVE TESTING SOLUTIONS: REDOC-ENHANCED CUSTOM FRAMEWORK FOR WEB APPS

Aleksandar Ivanovski
Faculty of Contemporary Sciences and Technologies, South East European University, Tetovo, North Macedonia

Abstract
Software testing has become increasingly crucial in the IT industry, leading companies to invest significant time, effort, and resources in this field. The reporting of test results and the achievement of high success rates are essential for the acceptance of many applications, particularly in industries where failure tolerance is minimal. Consequently, companies employ a combination of manual and automated testing approaches using various software testing tools and frameworks to meet their specific testing needs. However, existing tools and frameworks often lack comprehensive features, necessitating the development of customized solutions by combining multiple frameworks at considerable costs.
This research aims to address this challenge by proposing a new customized framework based on Selenium, a widely adopted and powerful software testing framework. The architecture and information flow of this customized framework are presented in detail, showcasing its potential for enhancing testing capabilities in applications composed of multiple modules. To demonstrate the effectiveness of the proposed framework, a real case study is conducted, and the implementation results are thoroughly analyzed.
The customized framework offers an automation-centric approach to software testing, maximizing efficiency and effectiveness. By leveraging the strengths of Selenium and integrating tailored features from various frameworks, it provides a comprehensive solution for diverse testing requirements. The results of the case study indicate significant improvements in testing capabilities, underscoring the advantages of the proposed customization approach

Keywords: software testing, customized framework, automation, Selenium, software testing tools, information flow, case study, testing capabilities, modular applications.

1 Introduction

1.1 Applying the styles to an existing paper

Software testing is gaining more importance nowadays and the most of the IT companies are investing a lot of time, effort and resources in this field. Moreover, the testing reporting and percentage of success is mandatory in the final steps of many applications acceptance. Especially for some industries where the accepted margin of the failure is close to zero, there is the need to use a set of software testing tools and frameworks for different modules or applications. As we have shown in our previous researches, none of the existing software tools and frameworks offer all the features and testing needs for all the companies. For this reason, most of them have a mixed of testing, manual and automated [1, 2].
The goal is to have as much as it is possible the automation of the software testing. Consequently, various testing tools and frameworks are reviewed to have the best choice [3]. For many of them, the best solution would be to have a customized framework that gets the most needed features from several

frameworks and optimizes the outcome of the testing. This comes with high costs of investment in time, money and resources.

In this research, we will try to introduce a new customized framework that is based on the most powerful and used software-testing framework, Selenium. We will show in detail the architecture and the information flow of this customization [4]. At the end of it, we will also show a real case study with all the results of its implementation. This will show how this customized framework improves our testing capabilities for this kind of applications that are composed by many modules. Writing a new document with this template.

2 Customized framework

The aim of our customized automation framework is:

- creating and running tests and test suites with full control over test concurrency and producing extremely thorough yet understandable test reports and
- providing technological extensions for testing systems of any type.

2.1 Main architecture components

In the below figure it is shown the architecture overview of the framework, Figure 1.

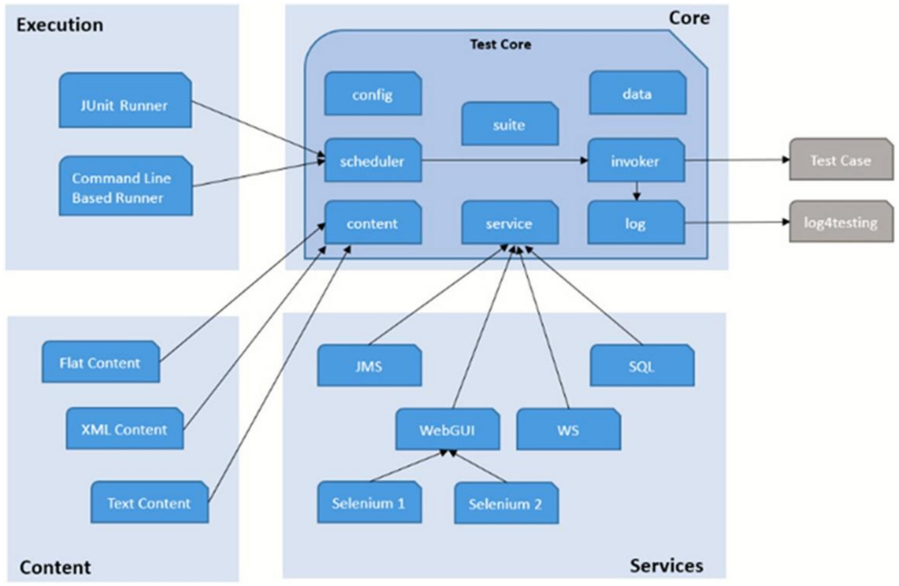


Fig. 1. Architecture overview

The following are the primary component types found in the framework:

A platform core that includes everything needed for creating, running and reporting tests. [5] One of the core classes of the framework contains main methods that are needed to perform user actions on an WebElement(DOM element) such as click(), getText(), selectCheckbox(), deselectCheckbox(), moveMouseOverElement(), rightClickOnElement(). Every general driver method is located in this class.

The Core components of the architecture of the customised framework are:

- *config*: Parses the configuration files and provides the framework with their parameters.

- *Scheduler*: Parses test suites and concurrency data and organizes it into a tree structure. Executes the tests in the tree while adhering to the specified concurrency requirements.
- *Content*: Provides the test classes with broad data/document-based validation functionality.
- *Service*: Manages the life cycle of services and declares service provider interfaces for platform extensions.
- *logfortesting*: Keeps track of test execution and logs it.
- Invoker: Provides an implementation for default test class mechanism and defines a service provider interface for executing tests.
- *Data*: Defines what a service is.

Execution extensions that allow running tests from the command line, as well as the JUnit unit testing framework.

Services which contain service provider interfaces for interacting with a system under test and give mechanisms to execute tests from the command line and the JUnit unit testing framework [6] (e.g. a web GUI, database, JMS).

2.2 The schedules class

The basic entry point for parsing a given Suite or Test class and constructing the described tree structure is SuiteParser. The class org.test.scheduler.RunnerTree is used to group all pending tests into a tree structure. The tree is made up of multiple types of nodes (represented by child classes of the org.test.scheduler.node.RunnerNode parent class):

- **Parent nodes:** Child nodes are tree nodes that combine other nodes. A test suite is represented by a parent node, which combines test cases and/or test suites. The class org.test.scheduler.node.RunnerGroup represents them.
- **Leaf nodes** are tree nodes that do not have any children. Each leaf node represents an atomic test case. The class org.test.scheduler.node.RunnerLeaf represents them.

The root node of the tree is mapped to the base test suite class. With RunnerNode as the component, RunnerNode as the leaf, and RunnerGroup as the composite, the tree node classes form a Composite Pattern. Because a test class method might be called multiple times, Java class elements are assigned to nodes in the following way:

Child node test class -> child node test method -> child node test invocation (or parameter set) -> leaf node.

2.3 Other settings

Java's Executor mechanism: RunnerTree schedules and executes Runnables that invoke the test cases using Java's ExecutorService mechanism. The essential concept is that an abstract service definition (ExecutorService) is used to asynchronously execute tasks (Runnable or Callable) and that an object handle (Future) is used to query execution status, obtain execution results, wait for termination, or cancel execution.

Some of the main plugins used from this customised framework are:

Test rail Connector: Allows framework test to send testrail results via the testrail connection module (using Rest API), Figure 2. The utility class that creates the TestPlanEntry is called first, followed by the

Listener that updates the results case by case, and finally a utility class that cleans up the un-tested cases from the testPlanEntry [7].

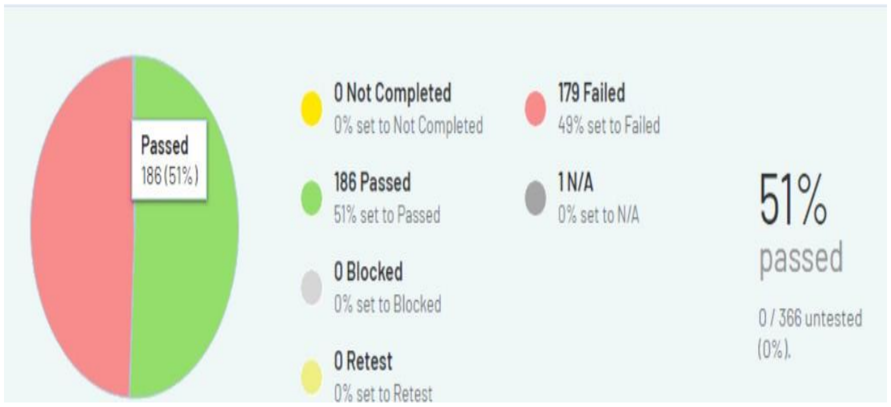


Fig. 2. Jenkins results displayed in test rail

Test Cloud Manager: The Cloud Manager acts as a "Load Balancer" for the Selenium hosts that are needed to run all the test cases. It is a comprehensive application made up of various parts (and supports extensions).

DatabaseTest, Database service is one of the most significant topics in database testing. This Database Service allows creating, updating, and querying databases.

Jenkins-plugin: Jenkins Plugin provides useful data for tasks that run framework-based tests. Each task must perform a unique Post-Build phase provided by this Plugin to obtain the framework data from the executed run in order to use the charts. Following that, both the run and the job have an additional button on the left sidebar. The statistics for the run concentrate on the current and maybe past runs, while the statistics for the job concentrate on the change in results over time. Jenkins Plugin offers sophisticated statistics and graphics for test executions [8]. This plugin helps analysing how test cases perform over multiple releases, allowing to rapidly spot potentially unstable test cases (or instabilities in Application under test). It also helps identifying trends by comparing results with previous builds, Figure 3.

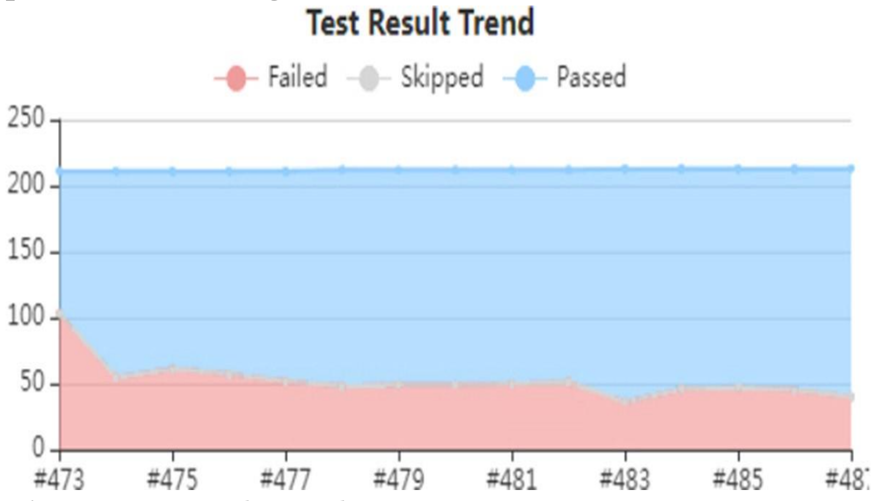
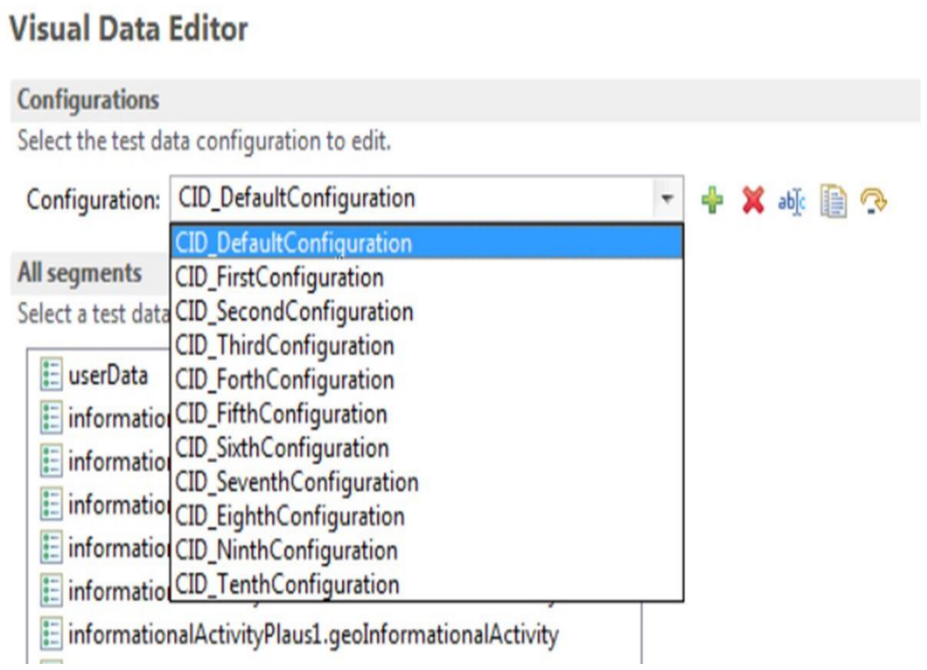


Fig. 3. Test result trend

Visual Data Editor (VDE): For test data, framework presently supports two formats: Excel files in a customized format and XML files in framework's own testdata xml format. To edit the latter, the framework ecosystem provides the Visual Data Editor as an Eclipse plugin. Because XML files are difficult to edit by hand, this plugin provides an editor for.testdata.xml files in Eclipse, Figure 4.


Visual Data Editor

Configurations
Select the test data configuration to edit.

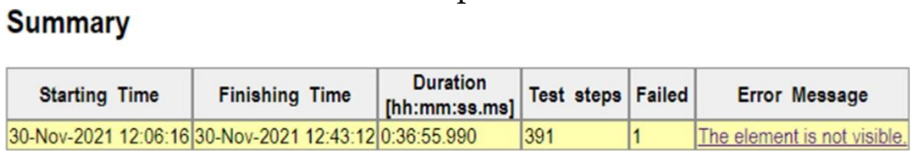
Configuration: CID_DefaultConfiguration

CID_DefaultConfiguration
CID_FirstConfiguration
CID_SecondConfiguration
CID_ThirdConfiguration
CID_ForthConfiguration
CID_FifthConfiguration
CID_SixthConfiguration
CID_SeventhConfiguration
CID_EighthConfiguration
CID_NinthConfiguration
CID_TenthConfiguration

All segments
Select a test data

userData
informatio
informatio
informatio
informatio
informatio
informationalActivityPlaus1.geoInformationalActivity

Fig. 4. The screen of the VDE in eclipse
Logfortesting is a component that offers the technical basis for test execution functional logging. It is used to create HTML or XML reports that are well structured and easy to read [9, 10], Figure 5.

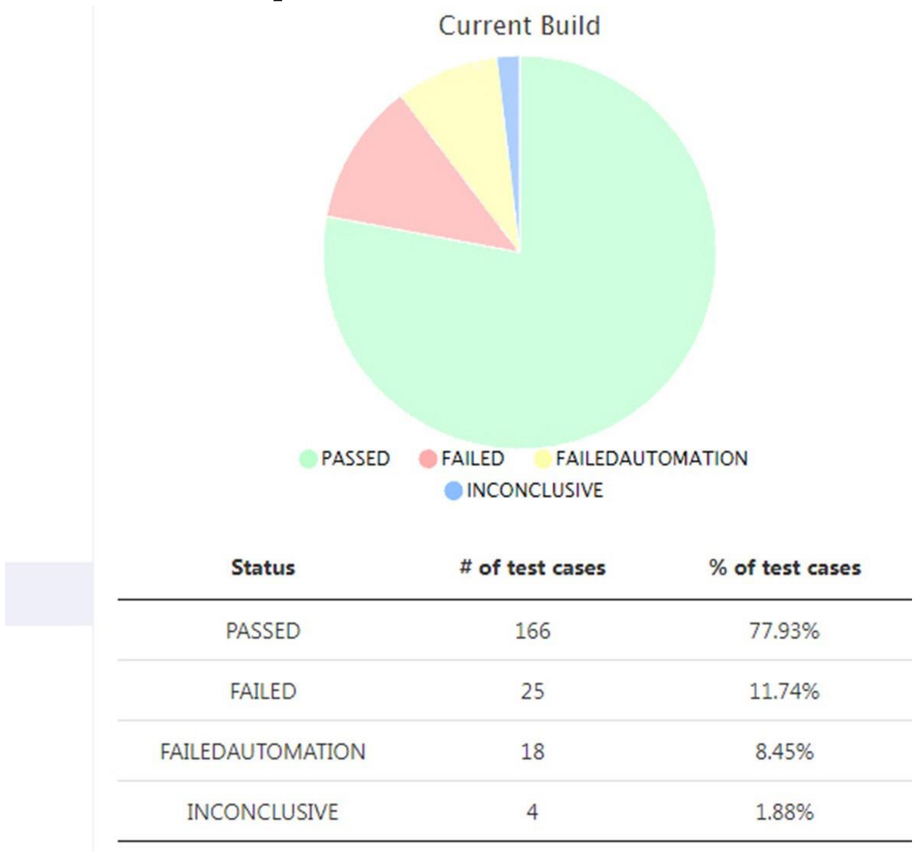

Summary

Starting Time	Finishing Time	Duration [hh:mm:ss.ms]	Test steps	Failed	Error Message
30-Nov-2021 12:06:16	30-Nov-2021 12:43:12	0:36:55.990	391	1	The element is not visible.

Fig. 5. A summary report of the test framework
There are several statuses for the executed test cases. Full list is below [11, 12], Figure 6:

- Pending, a test is scheduled to run, but it has not yet begun.
- Running, a test is presently in progress.
- Passed, a test was run, and no issues were found.
- Ignored, a test was disregarded. This can be due to a variety of factors for different sorts of test objects: This indicates to TestSuites and TestCases that a test data set has been designated to be ignored by the user, that the test has been run, but that the result code has not been propagated to the suite. It indicates that both a prior step was incorrect, and hence this TestStep was skipped. If a test step was incorrect but was set up to be retried despite an initial failure.
- Failed, a test failed due to a functional error e.g. incorrect value.

- Failed Performance, this implies that the SUT is stuck or takes an inordinate amount of time to reply to a user action.
- Failed Access, A test failed owing to a failed system access. The system is most likely unavailable, or the access setting is incorrect.
- Failed Automation, A test failed owing to an error that a Test Developer might resolve. Technical errors, such as a changing technical identifier, are the most common.
- Inconclusive, a test failed due to an error that has to be examined by a Test Automation Framework Developer.



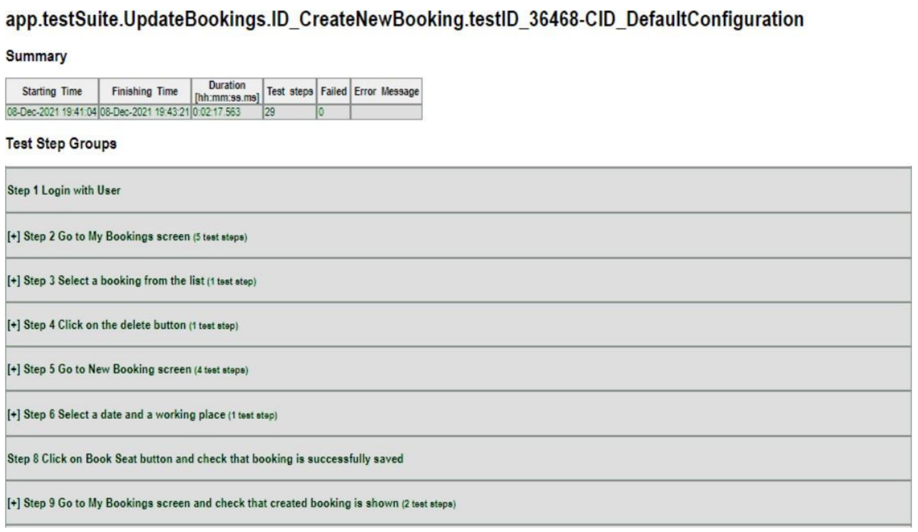
Status	# of test cases	% of test cases
PASSED	166	77.93%
FAILED	25	11.74%
FAILEDAUTOMATION	18	8.45%
INCONCLUSIVE	4	1.88%

Fig. 6. Types of failures of the test cases

3 Our experiments

A test case is used to get a better understanding of how the framework operates. The test case consists of eleven different configurations translated in eleven different test data to go through test steps. Aim of this scenario is to assure that the user can book a seat on a certain date and check that the booking is successfully saved and displayed in My Bookings screen. Scenario steps are displayed in the following print screen (part of the generated report after test case finishes running).
At first, starting and finishing time of test case execution is displayed. Then, steps in test case implementation are organized to form testGroups. Each testGroup gives complete information of the TAS and SUT at that step, Figure 7.

Klover Multidisciplinary Journal of Engineering
Volume 10 Issue 2, April-June 2022
ISSN: 2995-4118
Impact Factor: 6.40
http://kloverjournals.org/journals/index.php/Engineering


app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_DefaultConfiguration

Summary

Starting Time	Finishing Time	Duration [hh:mm:ss.ms]	Test steps	Failed	Error Message
08-Dec-2021 19:41:04	08-Dec-2021 19:43:21	0:02:17.563	29	0	

Test Step Groups

Step 1 Login with User

[+] Step 2 Go to My Bookings screen (5 test steps)

[+] Step 3 Select a booking from the list (1 test step)

[+] Step 4 Click on the delete button (1 test step)

[+] Step 5 Go to New Booking screen (4 test steps)

[+] Step 6 Select a date and a working place (1 test step)

Step 8 Click on Book Seat button and check that booking is successfully saved

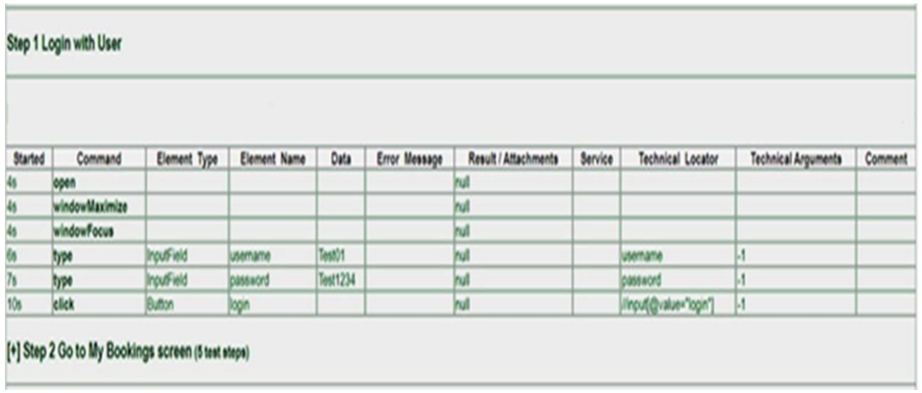
[+] Step 9 Go to My Bookings screen and check that created booking is shown (2 test steps)

Fig. 7. Report of the test steps

Often, certain functionality must be invoked before (set-up) or after (teardown) test execution. This can be achieved by putting relevant code into methods. For this reason, each test case class that is added in the framework has to extend a base class: BaseTestCaseClass.

This class contains all the methods that are needed to be executed before a test starts running and after it finishes execution (e.g., installation and configuration issues with test environment: database setup and initial load, services start/stop). Beside these methods, other Jenkins jobs are configured to start automatically before test Suite begins running [13] (e.g., database clear up).

Login method is same for all test cases, Figure 8. It can take different test data (username and password value) based on the functionality it will test. Following print screen is part of the generated report, login method is displayed with full information how it operates. Firstly, it goes to the provided URL (application domain), then maximizes the chrome driver window. Afterwards, it interacts with the SUT (System under Test), inserts values in username and password input fields. Technical Locator column indicates how xpath, classname or CSS locator locates the WebElements. Assertions are made for each operation conducted on the application to ensure that it performs properly.


Step 1 Login with User

Started	Command	Element Type	Element Name	Data	Error Message	Result / Attachments	Service	Technical Locator	Technical Arguments	Comment
4s	open					null				
4s	windowMaximize					null				
4s	windowFocus					null				
6s	type	InputField	username	Test01		null		username	-1	
7s	type	InputField	password	Test1234		null		password	-1	
10s	click	Button	login			null		//input[@value="login"]	-1	

[+] Step 2 Go to My Bookings screen (5 test steps)

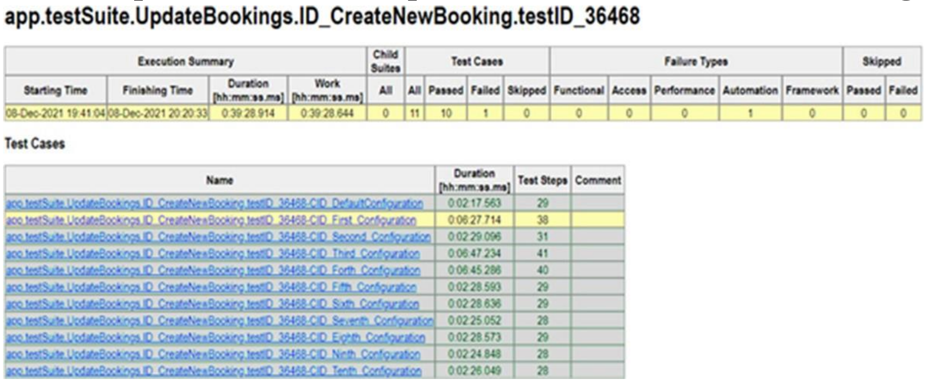
Fig. 8. Login method

The test logs give detailed information about the execution steps, actions and responses of a test case and/or test suite. However, the logs alone cannot provide a good overview of the overall execution result. For this, it is necessary to have in place reporting functionality. After each execution of the test suite or test case, a concise report must be created and published [14].

Whenever a test case encounters a failure, the framework makes sure that all information needed to analyse the problem is available/stored, as well as any information regarding the continuation of testing, if applicable. Screenshots and other visual captures are saved during test execution for further use during failure analysis.

Below it is a part of the report that is generated automatically when a test case finishes execution. This report is customized for the framework because Selenium itself does not have reporting tools, extra configuration is needed to be done to produce such detailed reports and log files. For each step, there is full information regarding WebElements, user actions, test data, assertion of that step.

In case of second run, test case has failed, and its execution time is significantly higher than the other runs. A maximum execution time has been specified in the framework core, a corresponding timer is started that will terminate (and fail) the test case if it has not finished in the configured time. Besides that, for each action performed on the SUT there is a taskCompletionTimeout which checks whether an action in performed which a predefined time in milliseconds, Figure 9.



app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468

Execution Summary				Child Suites	Test Cases				Failure Types					Skipped	
Starting Time	Finishing Time	Duration [hh:mm:ss.ms]	Work [hh:mm:ss.ms]	All	All	Passed	Failed	Skipped	Functional	Access	Performance	Automation	Framework	Passed	Failed
08-Dec-2021 19:41:04	08-Dec-2021 20:20:33	0:39:28.914	0:39:28.644	0	11	10	1	0	0	0	0	1	0	0	0

Test Cases

Name	Duration [hh:mm:ss.ms]	Test Steps	Comment
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_DefaultConfiguration	0:02:17.563	29	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_First_Configuration	0:06:27.714	38	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Second_Configuration	0:02:29.096	31	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Third_Configuration	0:06:47.234	41	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Forth_Configuration	0:06:45.286	40	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Fifth_Configuration	0:02:28.593	29	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Sixth_Configuration	0:02:28.636	29	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Seventh_Configuration	0:02:25.052	28	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Eighth_Configuration	0:02:28.573	29	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Ninth_Configuration	0:02:24.848	28	
app.testSuite.UpdateBookings.ID_CreateNewBooking.testID_36468-CID_Tenth_Configuration	0:02:26.049	28	

Fig. 9. Test execution results

4 Conclusion

There are many different types of automated testing frameworks, so choosing the proper one for your needs is important. Using one that is well structured, can improve the team's productivity by boosting test correctness, maximizing test coverage, and minimizing costs and maintenance—eventually providing a higher return on investment. We believe that our customized framework complies with the key concepts that support easy development, evolution, and maintenance of a Test Automation Solution.

Each component of the framework has a single responsibility [15], it is in charge of a one task, e.g.: setting up the driver, logging results, generating detailed execution reports, creating and reading data files, executing teardown methods. All these functionalities are secluded in platform core, which leads to another good principle being applied: tests (scenarios) are separated from automation framework (as already mentioned in case study, test data and test cases are in different packages from framework core). [16] In conclusion, although some of the principles must be followed, it is important that the automation framework serves the business needs and is customized according to the application. That is why this kind of customized framework is wanted; otherwise, one test automation solution would work for all kinds of applications. Further analysis and discussion will be done for the presented customized framework with more real case studies.

5 References

E. Pelivani and B. Cico, “A Comparative Study of Automation Testing Tools for Web Applications,” 2021 10th Mediterranean Conference on Embedded Computing (MECO), 2021, pp. 1–6, https://doi.org/10.1109/MECO52532.2021.9460242

E. Pelivani, A. Besimi, B. Cico, “An Empirical Study of User Interface Testing Tools”, International Journal on Information Technologies & Security, № 1 (vol. 14), 2022.

Anwar, Nahid & Kar, Susmita. (2019). Review Paper on Various Software Testing Techniques & Strategies. Global Journal of Computer Science and Technology. 43–49, https://doi.org/10.34257/GJCSTCVOL19IS2PG43

Muhammad Abid Jamil; Muhammad Arif; Normi Sham Awang Abubakar; Akhlaq Ahmad, “Software Testing Techniques: A Literature Review,” 2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M), 2016, pp. 177–182, https://doi.org/10.1109/ICT4M.2016.045

Raju Ranjan, “A Review Paper on Software Testing”, International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, vol.6, issue 1, pp. 25–32, January 2019, Available: http://www.jetir.org/papers/JETIREQ06004.pdf

Okezie, Adaugo & Odun-Ayo, Isaac & Bogle, S. (2019). A Critical Analysis of Software Testing Tools. Journal of Physics: Conference Series. 1378. 042030, https://doi. org/10.1088/1742-6596/1378/4/042030

Mojtaba. Shahin, Muhammad Ali Babar, Liming Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” in IEEE Access, vol. 5, 2017, pp. 3909–3943, https://doi.org/10.1109/ACCESS.2017.2685629

Ateşoğulları, Dilara & Mishra, Alok. White Box Test Tools: A Comparative View. International Journal of Information and Computer Security. Vol. 11, 2019, pp 79–90.

Satish Gojare, Rahul Joshi, Dhanashree Gaigaware, Analysis and Design of Selenium WebDriver Automation Testing Framework, Procedia Computer Science, vol. 50, 2015, pp. 341–346, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.04.038

Khushboo Sawant; Reetu Tiwari; Swapnil Vyas; Pawan Sharma; Aryan Anand; Shivani Soni, "Implementation of Selenium Automation & Report Generation Using Selenium Web Driver & ATF," 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1–6, https://doi. org/10.1109/ICAECT49130.2021.9392455

Nidhika Uppal Vinay Chopra, "Design and Implementation in Selenium IDE with WebDriver", International Journal of Computer Applications (0975 – 8887), puter Applications, vol. 46, no. 12, pp. 8–11, ISSN:0975 – 8887, May 2012.

Devi, Jyoti & Bhatia, Kirti & Sharma, Rohini. (2017). A Study on Functioning of Selenium Automation Testing Structure. International Journal of Advanced Research in Computer Science and Software Engineering. vol. 7, pp. 855–862, https://doi.org/10.23956/ijarcsse/ V7I5/0204

Ali Raza, Sergey Oplavin, Agiletestware Pangolin Connector for TestRail, https://plugins. jenkins.io/, last update 2020.

Ravinder Singh, Virender Singh, Selenium WebDriver Achitecture, https://www.toolsqa. com/selenium-webdriver/selenium-webdriver-architecture/, November 22, 2021.

Selenium community, Selenium overview https://www.selenium.dev/, last modified December 7, 2021.

Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker. Advanced Level Syllabus Test Automation Engineer, 2016.