Research Article / Araştırma Makalesi

Model Proposal for Testing Websites in Multiple Browsers: Case of Selenium Test Tool

Çoklu Tarayıcılarda Web Sitesi Testine Yönelik Model Önerisi: Selenium Test Aracı Örneği

Cem Ufuk BAYTAR [1]

ABSTRACT

Automated software testing has critical advantages when compared with manual software testing. The selenium test tool is one of the test tools that specialists use to test web applications or websites automatically. This test tool consists of 4 software components, i.e., Selenium IDE, Selenium RC, Selenium WebDriver, and Selenium Grid. The main purpose of this article is to implement a proposed model for cross-browser website testing by using Selenium WebDriver. Selenium WebDriver is required to manage the actions of a web browser. Drivers are needed to make a bridge between Selenium WebDriver and the relevant web browsers (chrome, edge, firefox). Other components of the model were Python, Unittest as a test framework, and PyCharm. PyCharm was used as an editor to write test scripts. One positive scenario and one negative scenario were applied to the relevant website. The results of automated test scenarios in 3 browsers were reported on the PyCharm screen. As a result, the model was validated because automated test results had been supported by manual test results.

Anahtar Kelimeler: *information system, selenium webdriver, automated software testing, website, unittest*

1 Istanbul Topkapı University, Management Information Systems Department, ufukbaytar@topkapi.edu.tr, ORCID: 0000-0003-0844-8160

ÖZ

Otomatik yazılım testi, manuel yazılım testi ile karşılaştırıldığında kritik avantajlara sahiptir. Selenium test aracı, uzmanların web uygulamalarını ya da web sitelerini otomatik olarak test etmek için kullandığı test araçlarından biridir. Bu test aracı, Selenium IDE, Selenium RC, Selenium WebDriver ve Selenium Grid olmak üzere 4 yazılım bileşeninden oluşur. Bu makalenin temel amacı, Selenium WebDriver'ı kullanarak birden fazla tarayıcıda web sitesi testi için önerilen bir modeli uygulamaktır. Selenium WebDriver, bir web tarayıcısının eylemlerini yönetmek için gereklidir. Selenium WebDriver ile ilgili web tarayıcıları (chrome, edge, firefox) arasında köprü kurmak için sürücülere ihtiyaç vardır. Modelin diğer bileşenleri Python, Unittest ve PyCharm bileşenleridir. PyCharm, test senaryosu komutlarını yazma editörü olarak kullanılmıştır. İlgili web sitesine bir olumlu ve bir olumsuz senaryo uygulanmıştır. 3 tarayıcıdaki otomatik test senaryolarının sonuçları PyCharm ekranında raporlanmıştır. Sonuç olarak, otomatik test sonuçları manuel test sonuçları tarafından desteklendiğinden önerilen modelin doğrulaması gerçekleşmiştir.

Keywords: *bilgi sistemi, selenium webdriver, otomatik yazılım testi, web sitesi, unittest*

1. Introduction

The testing process has an important role in the life cycle of software development to create high-quality software products (Rana & Latif, 2020). Bugs in coding are detected with the prepared test scenarios. In this context, the level of fulfillment of customer demands in an information system project is also measured (Meriç & Özbayoğlu, 2021). To find out software errors, test specialists apply different test techniques. Some examples are regression testing, functional testing, and automated testing (Koruyan & Uzun, 2019, p.55).

Regression testing is used to test any changes in program codes. It aims to provide correctness of software which is developed (Nguyen & Le, 2021). Functional testing tries to make sure that software meets necessary functional requirements.

It may be included in the regression testing process (Simplified, 2019).

Manuel testing process is realized without any tool. It is time-consuming and less reliable because of human errors (Sharma & Angmo, 2014). Automated testing is a test technique to test any application, for example, web applications, by using a software framework or a software tool (Yusifoğlu et al., 2015). According to Takgil & Kara (2016), automated testing has advantages as follows:

- Minimizing human effect on testing
- Finding more bugs in a short time
- Enabling re-use of test scenarios
- Increasing the percentage of code covered by test scenarios
- Being able to apply test cases continuously

People can visit many websites easily by using electronic devices (desktops, laptops, tablets, smartphones) connected to the internet. Users prefer different web browsers to open web pages. Every browser has its features that interpret the source code of a website differently. Browsers display changing technical behaviors among each other. This situation may cause problems related to compatibility of the browser, i.e., missing web elements on the web page, functions not working properly, etc. A website has to work correctly in every browser (MS edge, chrome, firefox) as much as possible to satisfy users. In other words, it has cross-browser compatibility if it works appropriately in every type of browser (Sabaren et al., 2018). That is why automated testing of a website in various browsers plays an important role in ensuring correctness and functionality for a website to work properly and to make customers spend much more time on websites that have high quality.

In this paper, a model has been proposed to be able to test a website in multiple browsers (cross-browsing) by using the selenium test tool. Selenium is an open-source test framework (Garcia et al., 2020). It is used especially to test web applications. It supports all web browsers and some programming languages, such as Java, .Net, Perl, PHP, and Python (Hanna et al., 2018).

• The objectives of this paper are:
• Carrying out the proposed model to show how to implement cross-browser web testing
• Validating the model by comparing manual test results with automated test results.
• Contributing to an understanding of how the Selenium test tool works.

The remaining part of this paper is organized as follows; Section 2 gives information about cross-browser testing, Section 3 describes Selenium, Section 4 defines the research model, and Section 5 includes implementation and validation of the model. Finally, Section 6 summarizes the results and possible future directions for this work.

2. Cross Browser Testing

Cross-browser testing is to verify that web applications or websites work properly for various combinations of web browsers, operating systems, and devices. Although web browsers support standards of the World Wide Web Consortium (W3C), they might render code in different ways. According to Datadog (n.d.), browser-related issues occur due to specified factors as follows:

• Different default settings of a browser, i.e., default font,
• Different user-defined settings, i.e., screen resolution,
• Different settings in hardware functionality cause differences in some parameters, for example, color balancing, and screen resolution.
• Differences in ways of processing web instructions,
• Variations among web standards in clients,
• Using handy technologies, i.e., screen readers,
• Various JavaScript coding,
• Mistakes in CSS,
• Lack of supporting HTML5,
• Incompatibility between browser and layout,
• Wrong frameworks,

One of the methodologies to carry out cross-browser testing is to write scripts. These scripts simulate behaviors of users who visit a website, i.e., creating an account, logging on to a website, clicking a button to send a message, etc. In addition, it is possible to add assertions, for

example, to check whether a web element appears on a page, etc. In this work, the selenium test tool and python as a programming language have been used to create such scripts.

3. Selenium as Automated Test Tool

Selenium is an open-source integrated test product. It is made up of some components, such as Selenium IDE, Selenium RC, Selenium WebDriver, and Selenium Grid. Selenium IDE (Integrated Development Environment) is a plugin for the Firefox web browser. It allows users to record their actions to test a web application by replaying those actions later. Selenium RC (Remote Control) is a server that

builds a bridge between a JavaScript-enabled web browser and test cases. Selenium Grid is used to carry out test scenarios at the same time in different computers to decrease execution time as much as possible. Selenium WebDriver is used to manage the behavior of a website to realize test scripts.

In literature, there have been articles about automated testing with the Selenium tool. Related work is summarized in Table 1.

Table 1. Related Work

No	Subject / Related to	Tool	Author
1	Introduction to regression test and Selenium	Selenium	Umesh et al., 2015
2	Comparison of Watir with Selenium	Watir, Selenium	Gogna, 2014
3	Describing the standard environment for testing with Selenium,	Selenium	Holmes & Kellogg, 2006
4	Developing automated testing framework	Webdriver, TestNG	Gojare et al., 2015
5	Comparison of QTP with Selenium	QTP, Selenium	Jagannatha et al., 2014
6	Descriptive survey on how the community uses Selenium	Selenium	Garcia et al., 2020
7	Proposed automated testing framework	Selenium	Hanna et al., 2018
8	Comparison of some testing tools	Selenium, QTP, Fitnesse, Watir, WinRunner etc.	Sharma & Angmo, 2014
9	Analysis of automated testing tools	Ranorex, Test Complete, Selenium	Kakaraparthy, 2017
10	Study on Selenium IDE	Selenium IDE	Sharma et al., 2017
11	Evaluating WebRTC	Selenium WebDriver	Garcia et al., 2022

Selenium test tool is cross-platform. It supports all major browsers. Table 2 provides comparison of different automated test tools (Sharma & Angmo, 2014).

Table 2. *Comparison of Test Tools*

No	Tools	Language Use	Operating system	Type	Language supported	Browser supports
1	Sele-nium	Java	Cross - platform	Software testing framework for web application	Domain specific language	All major browser
2	Watir	Ruby	Cross -platform	Software testing framework for web application	Java, .NET, c#	Originally only for Internet Explorer
3	HP-QTP	VB Script	Microsoft Windows	Test Automation Tool	VBscript	IE 6,7,8,10, Firefox 3.0 and later
4	Test Complete	Java	Microsoft Windows	Test Automation Tool	VBscript, Jscript, C++, Delphi Script, c# Script.	IE, Firefox, Google chrome
5	FitNesse	Java	Cross - platform	Test Automation Tool	C++, Python, Ruby, Delphi, c# etc	Platform independent
6	HP Load Runner	C	Microsoft windows & Linux	Load Testing Tool	VB, VBscript, java, javaScript, c#	Platform independent
7	TestNG	Java	Window, Linux, MAC	Testing Framework	Java	IE, FireFox, chrome
8	Silktest	4Test Scripting Language	Microsoft Windows	Test Automation Tool	Java, 4Test, VB, c#, VB.net	IE, FireFox
9	Win Runner	C	Microsoft Windows and Linux	Load TestingTool	Test Scripting Language (TSL)	IE, Netscape

Figure 1. Research Model
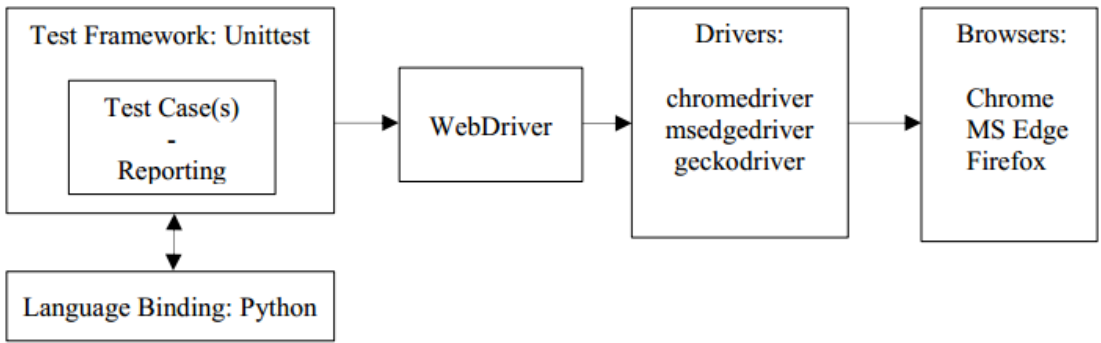
4. Research Model

The research model has been presented as depicted in Figure 1. This model aims to provide an infrastructure to implement cross-browsing testing. In other words, users can carry out the same test scenario(s) in 3 web browsers (chrome, edge, firefox). Selenium WebDriver is an object-oriented programming interface that effectively controls a web browser to run test cases of a web application or a website.

It uses language bindings and code implementations for driving a website. WebDriver is a suggestion of the World Wide Web Consortium (W3C). It communicates with a web browser through a driver, which is an executable module whose task is to manage the behavior of a web browser. Every browser has its driver as shown in Table 3. WebDriver sends a command to a web browser. The browser answers it. WebDriver has not any information about test scenarios that are written for a website. Test frameworks are necessary to execute WebDriver and to run steps in test scenarios. NUnit for .NET, JUnit for Java, RSpec for Ruby, and Unittest for Python are some examples of test frameworks (Garcia et al., 2022; Selenium, n.d.). In this work, Unittest has been used as a test framework.

Unittest was originally inspired by JUnit. It supports some important concepts for test automation in an object-oriented way as explained below (Python, n.d.):

• Test fixture: provides some facilities to realize test scenarios and cleaning actions.
• Test case: represents a testing unit. According to some specific inputs, it controls some responses. A base class, which is called TestCase, is used to create new test cases.
• Test suite: helps test specialists to run the relevant test cases together.

Table 3. Drivers used in the Model

Browser Name	Browser Version	Driver Name	Driver Version
Chrome	103.0.5060.66	chromedriver	103.0.5060.53
MS Edge	103.0.1264.37	msedgedriver	103
Firefox	101.0.1	geckodriver	0.31.0

Table 4. *Tools to Build Test Environment*

Tool Name	Explanation
Python – 3.10.5	https://www.python.org/downloads/
Chromedriver	
Msedgedriver	
Geckodriver	
PyCharm – community edition, 2022.1.2	https://www.jetbrains.com/pycharm/download/
Selenium – 4.2	

• Test runner: is used to managing test executions and to show test results on a graphical interface.

PyCharm editor has been used to write test scripts and to get a report about the automated test result. Necessary tools to build a test environment have been listed as shown in Table 4 (Selenium-python, n.d.).

It should execute the command **"pip install selenium"** in the command prompt of the Windows operating system to install the Selenium package. This package enables to access of Selenium WebDriver. Pip is the installer program of Python.

5. Implementation of the Model

2 test scenarios have been designed. There are similar test cases in the literature (Imtiaz et al., 2021; Hanna et al., 2018; Singla & Kaur, 2014; Jagannatha et al., 2014; Garousi et al, 2021). The first scenario is a positive one, another one is a negative test scenario. Dergipark's website has been tested with these test scenarios. Test scripts were written in PyCharm editor to realize test scenarios. In the positive scenario as shown in Figure 2, firstly a user opens the main page, then the language flag is changed to English. After clicking on the login button, the sign-in form is filled in. The form includes an e-mail address and password. Finally, the page "My Journals" is opened.
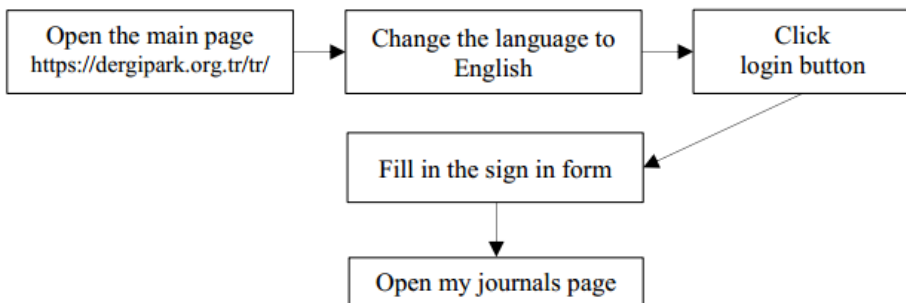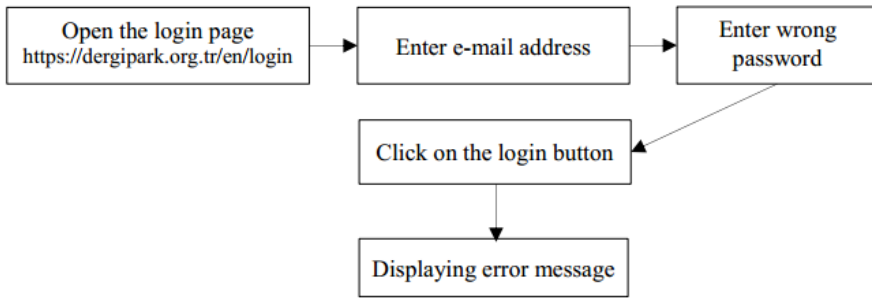
Figure 2. *Positive Test Scenario*

Figure 3. *Negative Test Scenario*

In the negative scenario, as depicted in Figure 3, after opening the login form, a user begins to fill in the login form. After the e-mail address, the related password is entered in the wrong way. The website shows an error message, i.e., "You have entered an invalid e-mail address or password "

Before writing a test case in the PyCharm editor, it is necessary to make some settings and to import some libraries, such as unittest, webdriver etc. Service objects, which are a new change in Selenium 4.0, are created for every browser (chrome, firefox, edge) separately as shown in Table 5, Table 6, and Table 7.

Table 5. *Settings part of Scenarios for Chrome Browser*

#	Test Code
1	import unittest
2	from selenium import webdriver
3	from selenium.webdriver.common.by import By
4	from selenium.webdriver.chrome.service import Service
5	s = Service('C:\driver_\chromedriver.exe') # create a service object

6	driver = webdriver.Chrome(service=s) # create a driver for chrome
7	url = 'https://dergipark.org.tr/tr/' # address of website

The parameter of service objects is the name of the driver file belonging to a browser, i.e., chromedriver, msedgedriver, and geckodriver.

Table 6. Settings part of Scenarios for MS Edge Browser

#	Test Code
1	import unittest
2	from selenium import webdriver
3	from selenium.webdriver.common.by import By
4	from selenium.webdriver.edge.service import Service
5	s = Service('C:\driver_\msedgedriver.exe') # create a service object
6	driver = webdriver.Edge(service=s) # create a driver for edge
7	url = 'https://dergipark.org.tr/tr/' # address of website

After creating service objects, drivers are defined for every browser, i.e., driver = webdriver.Firefox(service=s). Variable url has the name of the website which is under test.

Table 7. Settings part of Scenarios for Firefox Browser

#	Test Code
1	import unittest
2	from selenium import webdriver
3	from selenium.webdriver.common. by import By
4	from selenium.webdriver.firefox. service import Service
5	s = Service('C:\driver_\geckodriver. exe') # create a service object
6	driver = webdriver.Firefox(service=s) # create a driver for firefox
7	url = 'https://dergipark.org.tr/tr/' # address of website

After settings and imports, a positive test case was created as shown in Table 8. Based on the test scenario in Figure 2, the necessary script was written using python language. The full script is shown in Figure 4.

Table 8. Test Case part of the Positive Scenario

#	Test Code
1	class MyFirstTestCase(unittest. TestCase):
2	def test_cross_browsing(self): # create a testcase
3	driver.get(url) # open the main page

4	driver.find_element(By.LINK_TEXT, "English").click() # change language flag (EN)
5	driver.find_element(By.XPATH, '//a[@href="https://dergipark.org.tr/en/login"]').click()
	# open login page
6	driver.find_element(By.NAME, "_username").send_keys("ufukbaytar@gmail.com")
	# email address
7	driver.find_element(By. NAME, "_password").send_keys("xX123456") # password
8	driver.find_element(By.ID, "kt_login_submit").click() # click the login button
9	driver.find_element(By.ID, "item__ojs_journal_user_my"). click()
	# open my journals page
10	if __name__ == '__main__': # the end of testcase
	unittest.main()

A negative test case was created as shown in Table 9. Based on the test scenario in Figure 3, the relevant script was written in python

Figure 4. *Example Code of Positive Scenario*

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.service import Service

s = Service('C:\driver_\geckodriver.exe') # create a service object
driver = webdriver.Firefox(service=s) # create a driver for chrome browser
url = 'https://dergipark.org.tr/tr/' # address of website

class MyFirstTestCase(unittest.TestCase):

    def test_cross_browsing(self): # create a testcase
    driver.get(url)  # open the main page
    driver.find_element(By.LINK_TEXT, "English").click() # change language flag (EN)
    driver.find_element(By.XPATH, '//a[@href="https://dergipark.org.tr/en/login"]').click() # open login page
    driver.find_element(By.NAME, "_username").send_keys("ufukbaytar@gmail.com") # email address
    driver.find_element(By.NAME, "_password").send_keys("?uU9636160") # password
    driver.find_element(By.ID, "kt_login_submit").click() # click the login button
    driver.find_element(By.ID, "item__ojs_journal_user_my").click() # open my journals page

if __name__ == '__main__': # the end of testcase
 unittest.main()
```
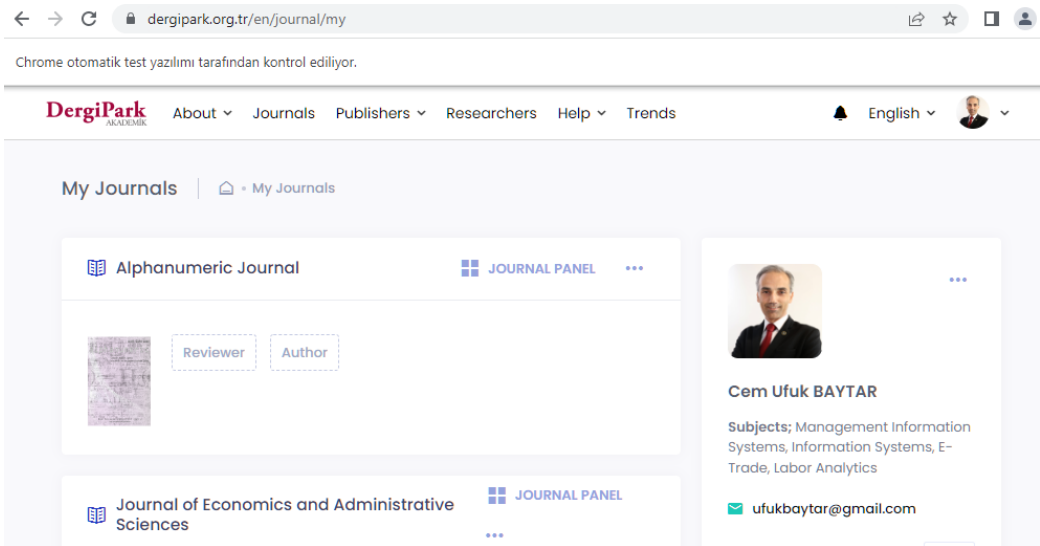
Table 9. Test Case part of negative Scenario

#	Test Code
1	class MyFirstTestCase(unittest.Test-Case):
2	def test_cross_browsing(self): # create a testcase
3	driver.get(url) # open the main page
4	driver.find_element(By.LINK_TEXT, "English").click() # change language flag (EN)
5	driver.find_element(By.XPATH, '//a[@href="https://dergipark.org.tr/en/login"]').click() # open login page
6	driver.find_element(By.NAME, "_username").send_keys("ufukbaytar@gmail.com") # email address
7	l = driver.find_element(By.NAME, "_password")
8	l.send_keys("xY123456") # wrong password
9	v = l.get_attribute("value") # assign the password value to v
10	driver.find_element(By.ID, "kt_login_submit").click() # click the login button
11	print("Test result in the Chrome driver") # informative message
12	self.assertEqual(v, "xX123456") # comparing wrong password with right password
13	if __name__ == '__main__': # the end of testcase unittest.main()

To validate the model as depicted in Figure 1, automated test results (reports) were compared with the results of tests that were done manually.

Figure 5. Manual Test Result for a positive scenario in 3 browsers

As shown in Figure 5, manual tests have been completed successfully for all browsers (chrome, firefox, edge). The page "My Journals" was reached at the end of the scenario. The results of manual tests have been supported by the results of automated tests as depicted in Figure 6. As an example, the result of one of the automated tests was okay as written in line 6. It took 7.055 seconds.

Based on the negative test scenario as depicted in Figure 3, manual tests have given an error message because of the wrong password as shown in Figure 7.

This situation has been supported by the results of automated tests as shown in Figure 8. The test result failed as written in line 5. The reason for failure was an assertion error as shown in line 10. In other words, the password in the test case was

Figure 6. Automated Test Report for a positive scenario in 3 browsers

```
1 C:\Users\Ufuk\PycharmProjects\testpro_1\venv\Scripts\
  python.exe C:/Users/Ufuk/PycharmProjects/testpro_1/
  cross_browser.py
2 .
3 --------------------------------------------------------
  -----------------
4 Ran 1 test in 7.055s
5
6 OK
7
8 Process finished with exit code 0
```

Figure 7. *Manuel Test Result of negative scenario in 3 browsers*
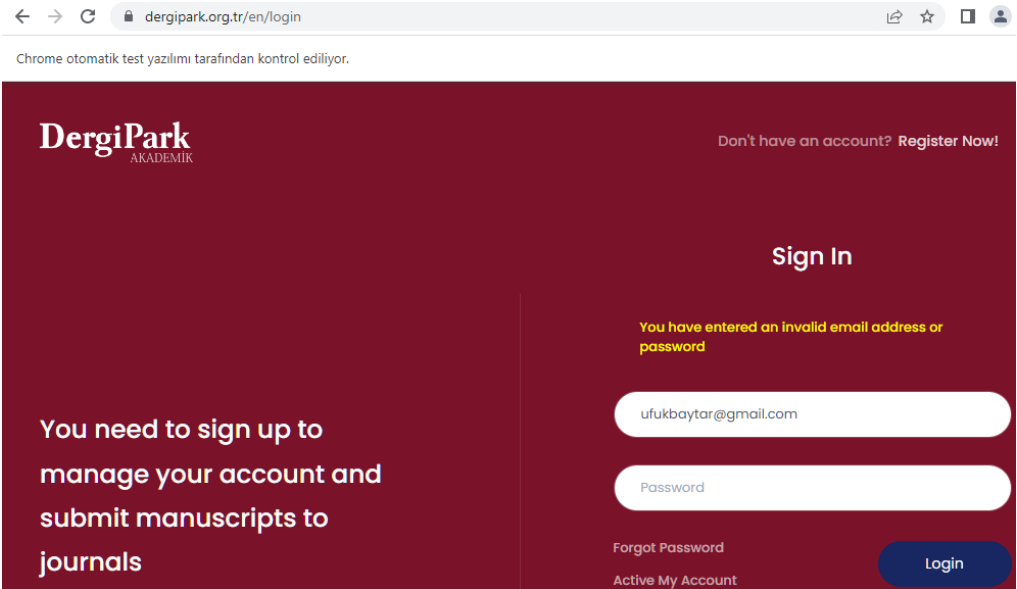
different from the password entered on the login page of the website.

As seen in Table 10, the time for testing manually is at least 2 times longer than automated test time. In other words, time reduction is approximately 61 percent.

These time values show that the usefulness level of the selenium test tool is high. It is so easy for users to run test scenarios because it is enough to press the run button whereas it is time-consuming to take every step manually in the scenario.

Figure 8. *Automated Test Report of negative scenario in 3 browsers*

```
 1 C:\Users\Ufuk\PycharmProjects\testpro_1\venv\Scripts\
   python.exe C:/Users/Ufuk/PycharmProjects/testpro_1/
   cross_browser.py
 2 Test result in the Firefox driver
 3 F
 4 =====================================================
   ================
 5 FAIL: test_cross_browsing (__main__.MyFirstTestCase)
 6 -----------------------------------------------------
   ----------------
 7 Traceback (most recent call last):
 8   File "C:\Users\Ufuk\PycharmProjects\testpro_1\
   cross_browser.py", line 22, in test_cross_browsing
 9     self.assertEqual(v, "xX123456") # comparing wrong
   password with right password
10 AssertionError: 'xY123456' != 'xX123456'
11 - xY123456
12 ?   ^
13 + xX123456
14 ?   ^
15
16
17 -----------------------------------------------------
   ----------------
18 Ran 1 test in 8.553s
19
20 FAILED (failures=1)
21
22 Process finished with exit code 1
```

Table 10. *The assessment of the Selenium Test Tool*

Test Name	Number of test steps	Automated Test Time (s)	Manually test time (s)
Positive Test Scenario	5	7.06	18
Negative Test Scenario	5	8.55	24

6. Conclusion

The testing process is so vital to developing reliable and well-done information systems, i.e., web applications or websites. To make customer engagement powerful, websites should work properly in multiple browsers. That is why cross-device testing for websites plays an important role. For achieving such a purpose, a model has been proposed. To implement the model, some components were needed, i.e., Python, Selenium WebDriver, browsers' drivers, Unittest framework, and PyCharm editor. Test scenarios were applied to the relevant website in 3 different browsers (chrome, edge, and firefox). There were one positive scenario and one negative scenario. All scenarios have been completed successfully. PyCharm editor reported test results on the screen. Automated test results have been supported by manual test results. This situation provided the validation of the proposed model. In the future, some studies may be conducted to measure cross-browser performance metrics by using the Selenium test tool.

Author Contributions

The author contributed to the study fully.

Conflicts of Interest

No conflict of interest was declared by the author.

References

Datadog (n.d.). Cross browser testing overview. Retrieved 15.06.2022 from https:// https://www.datadoghq.com/knowledge-center/cross-browser-testing/

Garcia, B., Gallego, M., Gortázar, F., & Munoz-Organero, M. (2020). A survey of the Selenium ecosystem. *Electronics*, *9*, 1067; doi:10.3390/electronics9071067. Retrieved from: https://www.researchgate.net/publication/342581217_A_Survey_of_the_Selenium_Ecosystem

Garcia. B., Kloos, C.D., Alario-Hoyos, C., & Munoz-Organero, M. (2022). Selenium-Jupiter: A JUnit 5 extension for Selenium WebDriver. *The Journal of Systems & Software 189*(2022) https://dl.acm.org/doi/abs/10.1016/j.jss.2022.111298

Garousi, V., Keleş, A.B., Balaman, Y., Güler, Z.Ö. & Arcuri. A. (2021). Model-based testing in practice: An experience report from the web applications domain. *The Journal of Systems & Software*, *180*(2021). https://doi.org/10.1016/j.jss.2021.111032

Gogna, N. (2014). Study of browser based automated test tools WATIR and Selenium. *International Journal of Information and Education Technology*, *4*(4), 336-339. Retrieved from: https://www.researchgate.net/publication/284440228_Study_of_Browser_Based_Automated_Test_Tools_WATIR_and_Selenium

Gojare, S., Joshi, R., & Gaigaware, G. (2015). Analysis and design of Selenium WebDriver automation testing framework. *Procedia Computer Science 50*(2015) 341-346. DOI: https://doi.org/10.1016/j.procs.2015.04.038

Hanna, M., Aboutabl, A.E., & Mostafa, M.S.M. (2018). Automated software testing framework for web applications. *International Journal of Applied Engineering Research*, *13*(11), 9758-9767. Retrieved from: https://www.ripublication.com/ijaer18/ijaerv13n11_141.pdf

Holmes, A., & Kellogg, M. (2006). Automating functional tests using Selenium. *IEEE*, Proceedings of AGILE 2006 Conference (AGILE'06). Retrieved from: https://www.cs.swarthmore.edu/~bylvisa1/cs97/f13/Papers/25620270.pdf

Imtiaz, J., Iqbal, M.Z., & Khan, M.U. (2021). An automated model-based approach to repair test suites of evolving web applications. *The Journal of Systems & Software*, *171*(2021), 110841

Jagannatha, S., Niranjanamurthy, M., Manushree, SP., & Chaitra, G.S. (2014). Comparative study on automation testing using Selenium testing framework and QTP. *Journal of Computer Science and Information Technology*, *3*(10), 258-267. Retrieved from: https://www.ijcsmc.com/docs/papers/October2014/V3I10201485.pdf

Kakaraparthy, D. (2017). Overview and analysis of automated testing tools: Ranorex, Test Complete, Selenium. *International Research Journal of Engineering and Technology*, *4*(10), 1575-1579. Retrieved from: https://www.irjet.net/archives/V4/i10/IRJET-V4I10290.pdf

Koruyan, K., & Uzun, B. (2019). Yazılım test sürecinde durum raporlamasına genel bakış ve yaklaşımlar. *Yönetim Bilişim Sistemleri Dergisi*, *5*(1), 52-63. Retrieved from: https://dergipark.org.tr/tr/download/article-file/876421

Meriç, Ö., & Özbayoğlu, A. (2021). Yapay öğrenme ile yazılım test eforu tahmini. *Veri Bilimi Dergisi*, *4*(1), 38-44. Retrieved from: https://dergipark.org.tr/tr/download/article-file/1205719

Nguyen, V., & Le, B. (2021). RLTCP: A reinforcement learning approach to prioritizing automated user interface tests. *Information and Software Technology 136*(2021) 106574. DOI: https://doi.org/10.1016/j.infsof.2021.106574

Python. (n.d.). Unit testing framework. Retrieved 06.09.2021 from https://docs.python.org/3/library/unittest.html#module-unittest

Rana, T., & Latif, B. (2020). A preliminary survey on software testing practices in Khyber PakhtunKhwa region of Pakistan. Turkish Journal of Electrical Engineering & Computer Sciences, *28*, 575-589. Retrieved from: https://journals.tubitak.gov.tr/elektrik/issues/elk-20-28-1/elk-28-1-42-1903-6.pdf

Sabaren, L., Mascheroni, M., Greiner, C., & Irrazábal, E. (2018). A systematic literature review in cross-browser testing. *Journal of Computer Science & Technology*, *18*(1), 18-27.

Selenium. (n.d.). WebDriver. Retrieved 07.09.2021 from https://www.selenium.dev/documentation/webdriver

Selenium.dev. (n.d.). Getting started. Retrieved 01.09.2021 from https://www.selenium.dev/documentation/webdriver/getting_started/

Selenium-python (n.d.). Installation. Retrieved 01.09.2021 from https://selenium-python.readthedocs.io/installation.html

Sharma, M., & Angmo, R. (2014). Web based automation testing and tools. *International Journal of Computer Science and Information Technologies*, *5*(1), 908-912. Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?-doi=10.1.1.642.4838&rep=rep1&type=pdf

Sharma, R., Devi, J., & Bhatia, K. (2017). A study on functioning of Selenium automation testing structure. *International Journal of Advanced Research in Computer Science and Software Engineering, 7*(5), 855-862. Retrieved from: https://www.researchgate.net/publication/318930970_A_Study_on_Functioning_of_Selenium_Automation_Testing_Structure

Simplified. (22.04.2019). Automated unit testing of a web application in Python. Retrieved 10.09.2021 from https://medium.com/@mashood.snhu/automated-unit-testing-of-a-web-application-in-python-de426af-da5a3

Singla S., & Kaur, H. (2014). Selenium keyword driven automation-testing framework. *International Journal of Advanced Research in Computer Science and Software Engineering*, *4*(6), 125-129

Umesh, N., Saraswat, A., & Himanshi (2015). Automation testing: An introduction to Selenium. *International Journal of Computer Applications*, *119*(3), 49-51. Retrieved from: https://citeseerx.ist.psu.edu/viewdoc/download?-doi=10.1.1.695.3234&rep=rep1&type=pdf

Takgil, B., & Kara, R. (2016). Android mobil uygulamalar için yazılım testi. *El-Cezerî Fen ve Mühendislik Dergisi*, *3*(2), 324-328. Retrieved from: https://dergipark.org.tr/tr/down-load/article-file/230831

Yusifoğlu, V.G., Amannejad, Y., & Can, A.B. (2015). Software test-code engineering: A systematic mapping. Information and Software Technology *58*(2015) 123 -147. DOI: http://dx.doi.org/10.1016/j.infsof.2014.06.009