



Assignment of bachelor's thesis

Title:	ANALYSIS OF OPEN-SOURCE TOOLS FOR AUTOMATED TESTING OF WEB APPLICATIONS
Student:	Samson Ošlakov
Supervisor:	Ing. Martin Komárek
Study program:	Informatics
Branch / specialization:	Information Systems and Management
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Analyze and describe open-source tools used primarily for automated testing of the front-end of web applications.

1. Analyze the current state of the tools, i.e., evaluate how can the tool be used in different types of testing, how easy it is to integrate it with other testing tools for testing web applications and what technical skills should the user of the tool have.
2. Compare and economically evaluate the benefits of implementing automated tests with the selected tools.
3. Design a test plan with test scenarios comprising UI and API tests for the selected demo application.
4. Implement automated tests for the given test scenarios with the tools that were analyzed.
5. Evaluate the benefits and risks of using automation testing tools on a web application from an economic perspective.

Bachelor's thesis

**ANALYSIS OF
OPEN-SOURCE TOOLS
FOR AUTOMATED
TESTING OF WEB
APPLICATIONS**

Samson Ošlakov

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Martin Komárek
May 11, 2022

Czech Technical University in Prague
Faculty of Information Technology

© 2022 Samson Ošlakov. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Ošlakov Samson. *ANALYSIS OF OPEN-SOURCE TOOLS FOR AUTOMATED TESTING OF WEB APPLICATIONS*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Contents

Acknowledgments	viii
Declaration	ix
Abstract	x
Introduction	1
1 Aim of the thesis	3
1.1 Research part	3
1.2 Practical part	3
1.3 Structure of the thesis	3
2 Concepts	5
2.1 End-to-end testing	5
2.2 Functional testing	5
2.3 Behavior-driven development	5
2.4 Test-driven development	5
3 Analysis of test automation tools	7
3.1 Cucumber	7
3.2 Karate	8
3.2.1 Description of features	9
3.2.2 Required technical knowledge	12
3.2.3 Set up	12
3.2.4 Integration with other tools	12
3.2.5 Community and support	13
3.2.6 Disadvantages	13
3.3 Selenium	13
3.3.1 Description of features	13
3.3.2 Required technical knowledge	17
3.3.3 Set up	17
3.3.4 Integration with other tools	18
3.3.5 Community and support	18
3.3.6 Disadvantages	18
3.4 Appium	18
3.4.1 Description of features	19
3.4.2 Required technical knowledge	21
3.4.3 Set up	21
3.4.4 Integration with other tools	21
3.4.5 Community and support	22
3.4.6 Disadvantages	22
3.5 SikuliX	22
3.5.1 Description of features	22

3.5.2	Required technical knowledge	24
3.5.3	Set up	24
3.5.4	Integration with other tools	24
3.5.5	Community and support	24
3.5.6	Disadvantages	25
3.6	Cypress	25
3.6.1	Mocha	25
3.6.2	Chai	25
3.6.3	Description of features	26
3.6.4	Required technical knowledge	28
3.6.5	Set up	29
3.6.6	Integration with other tools	29
3.6.7	Community and support	29
3.6.8	Disadvantages	30
3.7	Puppeteer	30
3.7.1	Description of features	30
3.7.2	Required technical knowledge	32
3.7.3	Set up	32
3.7.4	Integration with other tools	32
3.7.5	Community and support	32
3.7.6	Disadvantages	32
3.8	Playwright	32
3.8.1	Description of features	33
3.8.2	Required technical knowledge	35
3.8.3	Set up	35
3.8.4	Integration with other tools	35
3.8.5	Community and support	35
3.8.6	Disadvantages	35
3.9	TestCafe	35
3.9.1	Description of features	35
3.9.2	Required technical knowledge	37
3.9.3	Set up	37
3.9.4	Integration with other tools	37
3.9.5	Community and support	38
3.9.6	Disadvantages	38
3.10	Nightwatch.js	38
3.10.1	Description of features	38
3.10.2	Required technical knowledge	40
3.10.3	Set up	40
3.10.4	Integration with other tools	40
3.10.5	Community and support	40
3.10.6	Disadvantages	41
3.11	TestProject	41
3.11.1	Description of features	41
3.11.2	Required technical knowledge	43
3.11.3	Set up	43
3.11.4	Integration with other tools	43
3.11.5	Community and support	43
3.11.6	Disadvantages	43
3.12	WebdriverIO	43
3.12.1	Description of features	44
3.12.2	Required technical knowledge	45

3.12.3	Set up	45
3.12.4	Integration with other tools	45
3.12.5	Community and support	45
3.12.6	Disadvantages	46
3.13	Summary	46
4	Quantitative analysis	47
4.1	Browser support	48
4.2	Data formats	48
4.3	Language support	49
4.4	Dev platforms	49
4.5	Reporting	49
4.6	Selectors	49
4.7	Web UI	50
4.8	Community	50
4.9	Integration	51
4.10	Summary	52
5	Test implementation	55
5.1	Introduction of the demo application	55
5.1.1	Functionality	56
5.2	Test plan	58
5.2.1	Scenario 1	58
5.2.2	Scenario 2	59
5.2.3	Scenario 3	59
5.3	Implementation of the test plan	60
5.3.1	Selenium	60
5.3.2	Appium	63
5.3.3	Karate	65
5.3.4	SikuliX	68
5.3.5	Cypress	69
5.3.6	Puppeteer	73
5.3.7	Playwright	75
5.3.8	TestCafe	77
5.3.9	Nightwatch.js	78
5.3.10	TestProject	79
5.3.11	WebdriverIO	82
5.4	Summary	83
6	Benefits and risks of automated testing	85
6.1	Benefits	85
6.2	Risks and limitations	85
6.3	Other factors	86
	Conclusion	87
	Content of the attached media	97

List of Figures

3.1	Image showing how the Cucumber framework operates with tests written in Java [6]	8
3.2	Example of a default HTML page generated by Karate	9
3.3	Image showing how the Selenium WebDriver testing process works [21]	14
3.4	Overview of how Selenium WebDriver worked before version 4 [22]	14
3.5	Overview of how WebDriver works since version 4 [22]	15
3.6	Image of Selenium IDE	16
5.1	Diagram of Seat Reservation Demo [120]	56
5.2	Front end of Seat Reservation Demo	57
5.3	Image showing an emulated Android device	64
5.4	Image showing a running Appium Server	64
5.5	Image showing Scenario 1 implemented using SikuliX	69
5.6	Image showing the menu of Cypress	70
5.7	Image showing the development mode in Cypress	71
5.8	Image showing the menu of TestProject	80
5.9	Image showing the recording capability of TestProject	81
5.10	Image showing a report of a test executed in TestProject	82

List of Tables

4.1	Table with supported browsers	48
4.2	Table with supported data formats	49
4.3	Table with programming languages	49
4.4	Table with reporting parameters	49
4.5	Table with supported selectors	50
4.6	Table with Web UI parameters	50
4.7	Table with community parameters	51
4.8	Table with supported integrations	51
4.9	Complete table with all of the parameters.	53

List of code listings

1	Example of a test scenario written in Gherkin [1]	8
2	Example of an API test written in the Karate framework	10
3	Example of the workflow using SikuliX commands [2]	22
4	Scenario 1 in the Gherkin syntax	59
5	Scenario 2 in the Gherkin syntax	59
6	Scenario 3 in the Gherkin syntax	60
7	Example of Selenium and JUnit configuration of the tests	61
8	Example showing part of the Selenium implementation of Scenario 1	62
9	Example of Java Cucumber test runner	62
10	Example of Java Cucumber steps	63
11	Example of the Appium test configuration in JUnit	65
12	Example of test generated using the Karate extension for VSC	66
13	Example of the Scenario 1 implemented in Karate	67
14	Example of an API test for POST request in Karate	68
15	Example showing part of the implementation of Scenario 1 in Cypress	72
16	Example showing the test for a POST request to the demo app in Cypress	73
17	Example showing part of the implementation of Scenario 1 using Puppeteer	74
18	Example of an API test for POST request in Puppeteer	75
19	Example showing part of the implementation of Scenario 1 in Playwright	76
20	Example of an API test for POST request in Playwright	77
21	Example showing part of the implementation of Scenario 1 in TestCafe	78
22	Example showing part of the implementation of Scenario 1 in Nightwatch.js	79
23	Example showing part of the implementation of Scenario 1 in WebdriverIO	83

I would like to immensely thank Mr Ing. Martin Komárek for his guidance, useful advice and for providing the demo application. I would also like to thank APPLIFTING S.R.O. for their willingness to cooperate on a similar topic. Finally, I would like to thank my family and friends for their support both in life and in my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular, that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2022

.....

Abstrakt

Tato bakalářská práce popisuje a porovnává open-source nástroje pro automatizované testování webových aplikací. Práce se zaměřuje především na nástroje používané při automatizovaném end-to-end a funkčním testování webových aplikací s grafickým uživatelským rozhraním.

Práce analyzuje 12 nástrojů pro automatizované testování webových aplikací a diskutuje jejich kvality a nedostatky. Nad rámec zadání je provedena kvantitativní analýza nástrojů s využitím vážených parametrů pro kategorizaci nástrojů pomocí objektivních a subjektivních metrik, na základě, kterých byly nástroje hodnoceny. Znamky parametrů byly sečteny do celkového součtu a nástroj s nejvyšším počtem bodů byl považován za nejvíce doporučený k použití. Výsledkem je, že čtenář má k dispozici tabulku v Google sheets, kde bude moci pomocí úpravy vah u jednotlivých zkoumaných parametrů určit, který nástroj by byl nejlepší volbou ve zvoleném případě použití.

V praktické části je navržen a realizován testovací plán pro demo aplikaci s využitím analyzovaných nástrojů. Jednotlivé implementace jsou porovnány. Nakonec jsou diskutovány přínosy a nevýhody automatizace testování z ekonomického hlediska.

Získané poznatky mohou být velkým přínosem pro vývojáře nebo jiné odpovědné osoby při výběru vhodných nástrojů pro automatizované testování webových aplikací na různých projektech.

Klíčová slova testování softwaru, automatizace testování softwaru, open-source, testování webových aplikací, analýza testovacích nástrojů, automatizace E2E testování, automatizace funkčního testování

Abstract

This bachelor thesis describes and compares open-source tools for automated testing of web applications. The thesis primarily focuses on the tools used in the automated end-to-end and functional testing of web applications with a graphical user interface.

The thesis analyses 12 tools for automated testing of web applications, discussing their qualities and shortcomings. Beyond the terms of assignment, a quantitative analysis of the tools using weighed parameters to categorise tools using objective and subjective metrics on which were the tools graded. Grades of the parameters were summed into total, and the tool with the highest amount of points was considered the most recommended for usage. As a result, the reader has access to a Google sheet table where he will be able to use the adjusted weights of the individual parameters examined to determine which tool would be the best choice in the selected use case.

In the practical part, a test plan for a demo application is designed and implemented using the analysed tools. The implementations are compared. Finally, the thesis discusses the benefits and liabilities of test automation from an economic point of view.

The findings can greatly benefit developers or other responsible individuals in selecting the right tools for automated testing of web applications on different projects.

Keywords software testing, software testing automation, open-source, web application testing, analysis of testing tools, E2E testing automation, functional testing automation

List of abbreviations

- AI** Artificial Intelligence. 80
- API** Application programming interface. 1, 8, 9, 11, 12, 14, 17–20, 27, 29–34, 37–39, 41–43, 45, 47, 51, 56, 58, 60, 61, 63, 67, 72–77, 83, 84, 88
- app** Application. 18, 19, 27
- BDD** Behavior-Driven Development. 5, 7, 8, 25, 37, 45, 83
- CD** Continuous delivery. 1, 13, 18, 24, 26, 48, 51
- CI** Continuous integration. 1, 9, 10, 13, 17, 18, 21, 24, 26, 27, 29, 32–35, 43, 48, 51, 75
- CLI** Command Line Interface. 28, 36, 37, 77, 79, 82
- CSS** Cascading Style Sheets. 10, 28, 34, 35, 37, 39–41, 44, 45, 73, 78
- CSV** Comma-separated values. 9, 26, 42
- DDT** Data-Driven Testing. 9, 35, 36, 39, 42, 44, 47
- DevOps** Set of practices that combines software development (Dev) and IT operations (Ops). 1, 86
- DOM** Document Object Model. 27, 31, 33, 36–40, 44, 45, 70, 71
- E2E** End-to-end. 1, 3, 5, 7, 27, 35, 37–39, 41, 43–45, 47, 77, 84
- GUI** Graphical user interface. 1, 3, 19, 22, 24
- HTML** Hypertext Markup Language. vi, 8–12, 28, 31, 35, 37, 40, 45, 75
- HTTP** Hypertext Transfer Protocol. 11, 18, 19, 56, 58, 74
- IDE** Integrated Development Environment. 12, 15, 17, 19, 21, 23–25, 29, 32, 33, 35, 37, 40, 45, 68, 69, 78, 80, 83, 84
- JS** JavaScript. 10, 12, 13, 22, 26, 28, 30–37, 39–41, 44, 45, 56, 60, 66, 71, 73, 77, 84
- JSON** JavaScript Object Notation. 9–11, 14, 17–19, 26, 36, 37, 39, 44, 67, 74
- npm** Node.js package manager. 19, 29, 38
- QA** Quality assurance. 1, 5
- ROI** Return on investment. 1

SDK Software development kit. 18, 42, 80

SOAP Simple Object Access Protocol. 11

TDD Test-Driven Development. 5, 6, 26, 45

UI User interface. 1, 8, 9, 42

VSC Visual Studio Code. vii, 29, 35, 45, 66, 75

W3C World Wide Web Consortium. 10, 14, 38, 43, 82

XML Extensible Markup Language. 9, 11, 26, 38

XPath XML Path Language. 10, 20, 27, 31, 34, 36, 39, 41, 44, 73, 77, 78

YAML YAML Ain't Markup Language. 9, 26

Introduction

It is widely known that testing is a crucial phase of the software development life cycle.

Manual tests are still widely used among many software companies. It is a common practice for software companies to employ a full-time Quality assurance (QA) team. This team is responsible for preventing or at least minimising bugs in the software products. A standard part of their work is creating a collection of “test plans” or step by step checklists that assert whether a feature of a software project behaves as expected. The team would then manually execute the test plans by clicking through the application or interacting with the software and Application programming interface (API) with the appropriate tools every time a new update or change was introduced into the system. They would then return the results of the test plans to the engineering team for review and any further development to address issues. [3]

This process was slow, expensive, and error-prone because the testers from the QA team often needed to learn how the system works and behaves in certain circumstances. That takes some time, and even after the testers get to know the system well, they are prone to make typos or omit steps in the test script or miss some bugs that could later gravely affect the user experience.

That is why companies started automating their test plans. Automated tests are performed by a machine that executes a test script written using various test automation frameworks using multiple tools. These tests can vary in complexity, from testing a single method in a class to making sure that performing a sequence of complex actions in the User interface (UI) leads to the same results. It is much more robust and reliable than manual tests, but the quality of automated tests depends on how well were the test scripts written. Automation brings huge gains for team efficiency and Return on investment (ROI) of QA teams. However, it also poses a significant challenge in finding the right people who can write and maintain these tests. Major changes in the code of the application can often break the automated tests, so the test scripts need to be updated. [3]

Automated testing is a crucial component of Continuous integration (CI) and Continuous delivery (CD). Most modern agile and DevOps software projects now include automated testing in the project’s conception. It is a great way to scale the QA process as new features are added to an application. Automated testing puts ownership responsibilities in the hands of the engineering team. The test scripts are developed alongside regular project road-map feature development and are automatically executed by CI/CD tools. Automated testing enables the QA team to focus on more sensitive tasks like exploratory testing.

There are a lot of tools available in the test automation market. Choosing the right ones for the job can be difficult. Sometimes it can be just one tool to cover the testing needs of the project, but most of the time, it is a combination of wisely picked tools.

I have noticed that there have not been many comprehensive resources available that compare these tools so that the developer can have a better time choosing the tools best suitable for his project. This is the reason why I chose this assignment.

This problem is addressed as the thesis analyses different tools for automated testing of web applications. Because the domain of automation testing is vast, the thesis will mainly focus on the tools used for automating functional and End-to-end (E2E) tests for web applications through the Graphical user interface (GUI) of web applications.

The main tools under the scope of the analysis are free and open-source, but some of the tools offer premium services for a fee. My main criteria for choosing the tools were popularity and usefulness.

The primary sources are the official websites and documentation written for the tools. Other sources include technical articles, forums and university publications.



Chapter 1

Aim of the thesis

This thesis aims to analyse tools for automated testing of web applications that mainly have a GUI. The main focus will be testing the web applications on desktops across operating systems. However, the thesis will also discuss testing web applications from mobile devices and using cloud services.

The analysed tools are currently still maintained, so hopefully, the content of this thesis will not get outdated soon, but that is unlikely due to the technological progress being so fast-paced that new open-source tools for automation are likely to emerge.

After reading this thesis, the readers should be able to tell which tools are suitable for their E2E testing automation needs by the terms they set using specified parameters. In this thesis, one can also find recommendations on what to test in web applications and which part should be automated, including the economic implications of test automation.

1.1 Research part

The first objective is to qualitatively analyse the selected tools used for automating functional and E2E tests.

The second goal is to give an insight into the benefits and liabilities of automation testing from an economic perspective.

1.2 Practical part

In the practical part of the thesis, the first objective is to give insight into the results and methods behind the quantitative analysis of the tools.

The second objective of the practical part is to analyse the tools using tests from a test plan created for a demo application.

1.3 Structure of the thesis

The thesis starts by giving an introduction to some concepts.


Then comes the analysis of tools for creating automated functional and E2E tests for web applications. The chapter describes the tools and discusses their qualities and shortcomings.

After discussion, the thesis comes with an evaluation of the quantitative analysis that grades parts of the tools on their parameters.

Next comes the comparison of the using the tests examples. The chapter starts with a description of the demo application for which were the tests written. Then it describes the test plan for the application. The tools are then compared using their implementation of the test plan.

In the next section, the thesis delves into the benefits and liabilities of automation testing from an economic perspective.

At the end of my thesis, I evaluate the whole study.



Chapter 2

Concepts

This chapter introduces some of the concepts mentioned in the text.

2.1 End-to-end testing

End-to-end testing is a software testing method that involves testing an application's workflow from beginning to end. This method aims to replicate real user scenarios from the end user's experience so that the system can be validated for integration and data integrity. [4]

2.2 Functional testing

Functional testing is a process through which the QA team determines if a piece of software is acting under pre-determined requirements. It uses black-box testing techniques, in which the tester does not know the internal system logic. Functional testing is only concerned with validating if a system works as intended. [5]

2.3 Behavior-driven development

Behavior-Driven Development (BDD) is a type of development process used by software teams. The members map business requirements to their software implementation, using specific examples that make developers and business analysts better understand the problems they are trying to solve.

One of the main goals of BDD is to get the team talking to each other about the system's use cases. Ideally, during these conversations, somebody should specify the use case scripts, which try to explain what should happen when a user performs one or other action, so they can be used as documentation to implement the corresponding functionalities. Besides, the use case scripts can also be automated as tests to verify that the system's behaviour is as expected.

Thus, a team follows BDD principles when it goes through an iterative process to define the system's features using the three stages: discussion, specification and test writing.. [6]

2.4 Test-driven development

Test-Driven Development (TDD) is a software development practice in which unit and acceptance test cases are incrementally developed before writing the production code and directing the design

of the target software.

Lately, TDD has gained substantial awareness among practitioners and researchers, beyond its initial context in Extreme Programming, with the promise of several benefits to the software development process.

Advocates of TDD proclaim that TDD improves code quality, application quality and developer productivity compared with traditionally testing strategies. [7]

Analysis of test automation tools

The following tools have been chosen for their popularity and usefulness in writing scripts for automated testing of web applications.

In the following sections, the open-source tools to analyse in this bachelor thesis are introduced. These tools are primarily used for automating functional and E2E tests.

All of the following tools are cross-platform, which means they can be used on Windows, Linux and macOS. Most tools support browsers from the list below and can operate on browsers in the headless mode except for SikuliX.

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Google Chrome is currently the most popular browser on the internet, with more than half of the browser market share belonging to Chrome. The current browser share can be viewed using a link.¹

3.1 Cucumber

Cucumber is a wildly popular tool to support *BDD*. [6] It provides two major features:

Gherkin syntax Helps with defining scenarios, features, and use cases. It follows a specific Given-When-Then pattern like shown in the example 1. It can be described as a business readable behaviour description language.

An open-source framework Translates the Gherkin steps to executable code that can be run as tests. It is available in multiple languages. The list of supported languages can be found on the installation website.²

Thus, Cucumber covers two stages of BDD: writing the use cases and automating the tests. They can be found referenced in the Cucumber documentation as Formulation and Automation. [8]

¹<https://gs.statcounter.com/browser-market-share>

²<https://cucumber.io/docs/installation/>


```

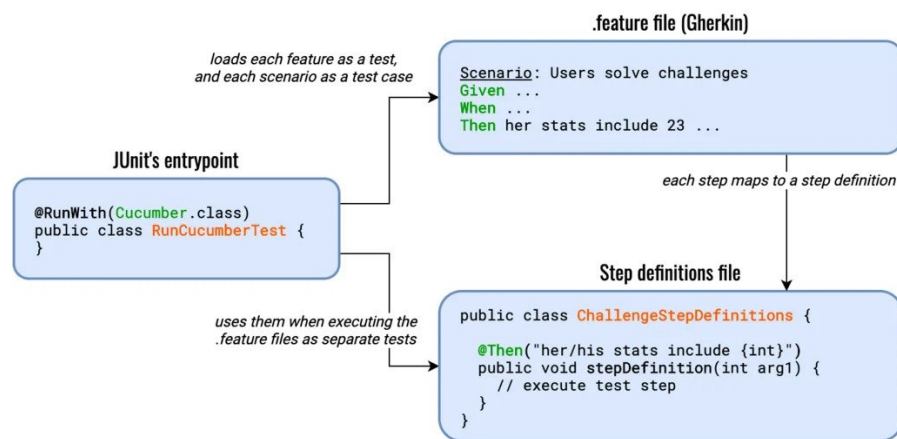
Scenario: Breaker guesses a word
  Given the Maker has chosen a word
  When the Breaker makes a guess
  Then the Maker is asked to score

```

■ **Code listing 1** Example of a test scenario written in Gherkin [1]

The Cucumber framework uses “feature files” for describing tests using Gherkin. The feature files are stored in files with a “.feature” filename extension, hence the name feature file. The test steps in the feature file are then mapped onto functions written in the chosen programming language. The functions that describe the logic of a test are called “glue” and are stored in “step definition” files. [1]

Image shown in the figure 3.1 explains how the Cucumber framework operates on a test written in Java using the JUnit library.



■ **Figure 3.1** Image showing how the Cucumber framework operates with tests written in Java [6]

It was first released in 2008 and is now under SmartBear Software company. The “Open” version is still under active development by the team and the open-source community. Cucumber has around 10 000 stars on GitHub [9] and around 10 000 questions on Stack Overflow. [10]

Cucumber Open is under the MIT License³ so it can be used commercially.

Cucumber also offers a premium version called “CucumberStudio”⁴ that offers more useful features for BDD.

Cucumber can be used with all of the other tools subject to this analysis.

3.2 Karate

“Karate is the only open-source tool to combine API test-automation, mocks, performance-testing and even UI automation into a single, unified framework. The behaviour-driven development syntax popularised by Cucumber is language-neutral and easy for even non-programmers. Assertions and HTML reports are built-in, and you can run tests in parallel for speed.” [11] Karate was added to the list because it is an all-in-one framework that includes environment

³<https://github.com/cucumber/cucumber-jvm/blob/main/LICENCE>

⁴<https://cucumber.io/tools/cucumberstudio/features/>

switching and CI integration. Karate was first created for API testing, but now it can do more than that.

3.2.1 Description of features

Karate is implemented in Java. It can be used with Maven or Gradle or using the executable. Java API is also available for those who prefer to integrate Karate into their Java code. It works with JUnit 4 and JUnit 5. [12] The tests are stored in “.feature” files, similarly to feature files in the Cucumber framework.

Mixing API and UI test automation within the same test script and re-using code already written by calling other feature files are possible in Karate.

3.2.1.1 Browser support

In addition to the browsers mentioned at the beginning of the chapter, it works with:

- Internet Explorer
- Safari

3.2.1.2 Test recording tool

Karate does not offer a test recording tool.

3.2.1.3 Supported data formats

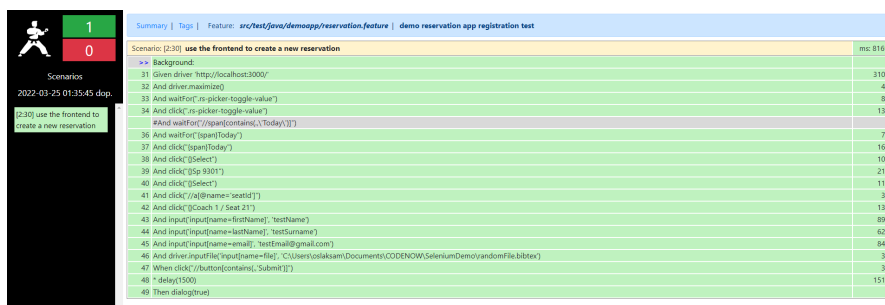
Karate has built-in support for JSON, Comma-separated values (CSV), XML and YAML files and conversion between some of them. Therefore there is no need to depend on external frameworks. [13] This comes in handy for DDT or simply for tests where some external data is needed.

3.2.1.4 Reporting

Karate offers a great default HTML report that looks good and tells all of the essential details about the test. An example can be found in the figure 3.2.

Karate has an option for the JUnit XML output from the parallel runner that can determine the status of the build as well as generate reports that can be integrated with most reporting and CI tools. [14]

It has examples for the set-up of reporting and an example for third-party libraries that can be easily used to generate excellent looking reports from the JSON output of the parallel runner. An example of a commonly used library with Karate is the “cucumber-reporting” open-source library. [13]



Scenario	Summary	Feature	ms
Scenario: (2.30) use the frontend to create a new reservation	Scenario: (2.30) use the frontend to create a new reservation	src/test/feature/demoapp/reservation.feature demo reservation app registration test	ms: 8169
1	>> Background:		
31	Given driver http://localhost:3000/		3107
32	And driver.maximize()		45
33	And waitFor('#picker-toggle-value')		83
34	And click('#picker-toggle-value')		135
35	And waitFor('input[contains,\"Today\"]')		
36	And waitFor('#spanToday')		72
37	And click('#spanToday')		162
38	And click('#select')		100
39	And click('#input:radio')		211
40	And click('#select')		119
41	And click('@[name=seats]')		36
42	And click('@coach 1 / Seat 21')		138
43	And input(input[name=firstName], 'testName')		897
44	And input(input[name=lastName], 'testSurname')		624
45	And input(input[name=email], 'testEmail@gmail.com')		844
46	And driver.uploadFile('input[name=file]', 'C:/Users/lostakami/Documents/CODENOW/Selenium/Demo/randomFile.txt')		37
47	When click('#button[contains,Submit]')		32
48	* delay(1500)		1517
49	Then dialog(true)		7

■ Figure 3.2 Example of a default HTML page generated by Karate

Karate has adequate traceability for the errors that might occur during tests. Detailed wire-protocol logs can be enabled in line with the test steps in the HTML report. [12]

3.2.1.5 Language support

The syntax for the Karate tests is similar to Gherkin but mixed with specific keywords and characters for doing certain operations as shown in the example 2 Karate offers dynamic variables and functions and can even pass variables to JSON.

It offers JavaScript and Java interoperability. You can chain commands and use JavaScript using the `script()` function. [14] Adding third-party Java libraries and creating own Java or JavaScript functions is also possible with Karate. For users not comfortable with Java, Karate offers a cross-platform stand-alone executable. [12]

```
Scenario: Try to create a new reservation with an empty body and fail.
  * def req =
    """
    {}
    """

  Given url apiUrlPost
  And request req
  When method post
  Then status 500
```

■ **Code listing 2** Example of an API test written in the Karate framework

3.2.1.6 Parallel testing

Parallel testing is frankly easy to configure. Unlike Selenium, choosing between execution on a single node, Cloud CI environment or Docker without needing a “controller node” or configuring a “grid” is possible. There is even an option to run tests in parallel across different machines. Karate can aggregate the results and present them as one. [14]

3.2.1.7 UI automation and selectors

In this section, the words “locator” and “selector” are used interchangeably as they refer to the same concept when finding a web element on the page. Karate supports fundamental user interactions with the web page, such as refresh, click, scroll, drag, etc. It also supports iFrame, browser tabs, alerts, and uploading files which are all very important for simulating the user.

W3C WebDriver support is built-in and can also utilise remote/grid providers. Chrome automation is made better using the Chrome DevTools Protocol, similar to Puppeteer. Android and iOS mobile testing is supported via Appium. It even supports cross-platform desktop automation that can be mixed into web automation flows if needed. [14]

Karate has support for standard CSS and XPath selectors. There are also the wildcard and “friendly” locators that select without inspecting the HTML-page source, CSS, or internal XPath structure. Karate also offers a `locateAll()` function that filters elements using a filter in the form of a JavaScript (JS) function. [14]

Karate includes a straightforward `retry` and `wait` strategy. It likewise allows handling common combinations like: “submit() + click()” by chaining them, which is common in many JS testing frameworks.

Karate contains broad assertion abilities. For easy troubleshooting, the failed tests report which web element and path are not as anticipated. Karate's match assertion command and other UI assertions core capabilities can use regular expressions (regex).

Mimicking standard user input types like keyboard key combinations and mouse actions is possible in Karate.

Karate allows intercepting HTTP requests made by the browser and uses Karate mocks to stub/modify server responses and replace HTML content. [14]

3.2.1.8 API testing

API testing was the first main feature that Karate supported. It is carefully designed for HTTP(S), JSON, GraphQL and XML requests and data types. [12]

Comprehensive support for different types of requests:

- SOAP/XML requests
- HTTPS/SSL without needing certificates, keystores or truststores
- HTTP proxy server support
- URL encoded HTML form data
- Multi-part file upload, including multipart/mixed and multipart/related
- Browser-like cookie handling
- Full control over HTTP headers, path and query parameters
- Re-try until a condition is met
- WebSocket support

As was mentioned in the previous section, the assertion capabilities are also applied to API testing.

3.2.1.9 Performance testing

Karate integrates performance testing using a Gatling adapter.

“Gatling is a powerful open-source load testing solution. Gatling is designed for continuous load testing and integrates with your development pipeline. Gatling includes a web recorder and colourful reports.” [15]

Karate can be used to see how well does an application handle stress by re-using Karate API tests as performance tests executed by Gatling. Gatling only needs to be used for defining a load model. Karate can handle everything else.

Karate assertions are integrated with the Gatling report. The line numbers where assertions failed can be seen in the report.

Asserting that server responses are as behaving as expected under load is much harder in Gatling and other performance testing tools than with Karate. Another benefit is that API invocation sequences representing end-user workflows are much easier to express in Karate. Distributing the load test over multiple hardware nodes or Docker containers is also possible. [16]

3.2.1.10 Environment configuration

Karate is controlled by a configuration file “karate-config.js” where multiple environments can be easily configured by changing the values of some variables or adding functionality that is only meant to be applied to the selected environments. [12] This can be useful when running Karate using Docker and Kubernetes.

3.2.1.11 Screenshots and video recordings of tests

Karate can transform the HTML of the website to PDF and capture the whole web page as an image utilising the Chrome Java API. There is furthermore an option to embed video recordings of tests into the HTML report from a Docker container. [14]

3.2.1.12 Multifunctionality

From all of the functionalities mentioned afore, it can be assessed that Karate is a unique multi-purpose testing framework which has a goal to keep things simple and bundled together.

All of this is helpful in case using multiple tools for testing automation is not favourable, as it can often be demanding to teach each team member to use multiple tools.

3.2.1.13 Documentation and test examples

Karate maintains thorough documentation and offers many different test examples and usages, which is a correct approach because it uses its syntax. Without detailed documentation and numerous examples, this tool would not be as valuable as it is today.

The documentation can be found on the documentation website.⁵ Examples and another a part of the documentation can also be found on GitHub.⁶

3.2.2 Required technical knowledge

The syntax of Karate is clean and straightforward. It is well suited for people new to programming or test automation.

The users do not have to know complicated programming concepts such as “callbacks”, “async and await” and “promises” that are common in JS. [14]

3.2.3 Set up

To set up the Karate framework, one needs a computer with macOS or Windows or Linux and Java Development Kit (JDK) or Java SE version 8 or later to execute the Karate standalone.

In case of the need to develop and debug the tests, an Integrated Development Environment (IDE) should be installed. Then it would be best to have a project management tool for Java, either Maven or Gradle.

3.2.4 Integration with other tools

Karate can be run in a Docker container and run well on the cloud.

Karate has experimental support for Playwright, a framework we will talk about later because Playwright has even more cross-browser and headless options that can connect to a server or Docker container using the Playwright wire protocol.

IDE support and syntax-colouring options for Cucumber and Karate. Karate has an extension for Visual Studio Code. It step-debugging and even back-stepping to edit and re-play steps. [17] The extension can even generate API tests from OpenAPI specification.⁷ If the specification is written correctly, the generated tests will work perfectly.

Karate offers asynchronous support that allows seamless integration to handle custom events or listen to message queues.

⁵<https://karatelabs.github.io/karate/>

⁶<https://github.com/karatelabs/karate/>

⁷<https://swagger.io/resources/open-api/>

Karate integrates well with CI/CD tools because of the standalone version. It also has Spring Boot support.

Karate is integrable with GitLab and GitHub actions.

3.2.5 Community and support

Peter Thomas developed karate in 2017 [13]. It is relatively new multiple new features are expected to be added. Karate is licensed under the MIT License.⁸

It is managed by Karate Labs Incorporated.⁹ It is still being maintained mostly by the author. It has around 3 200 questions on Stack Overflow [18] many of them answered by the creator of Karate, around 6 000 stars and 1 600 used by statistics on GitHub. [12] The author is actively responding to GitHub issues and Stack Overflow questions.

3.2.6 Disadvantages

One of the disadvantages is that the main contributor to Karate is the author himself. The progress can be slow if one person mostly maintains the project.

The fact that it uses its syntax can be offsetting for someone because the interoperability with Java and JS can be problematic and bothersome.

Since the tool is relatively new and not used by many people, encountering problems that have not yet been solved or at least asked on Stack Overflow is possible, so fixing them or asking how to fix them is a personal responsibility.

3.3 Selenium

Selenium is probably the most widely-known and used suite of tools for testing web applications. [19] It is often used for regression testing. The suite consists of:

Selenium IDE Is used for recording tests, executing, debugging, and generating scripts.

Selenium WebDriver Drives a browser natively, as a user would locally or on a remote machine and marks a big step forward in browser automation. Selenium WebDriver refers to both the language bindings and the individual browser controlling code implementations. [20] It is designed as a simple, object-oriented and more concise programming interface.

Selenium Remote Control Is an old version of WebDriver that had to be used with an old version of Selenium Grid.

Selenium Grid Works as a central point for managing tests on different devices and running tests in parallel.

3.3.1 Description of features

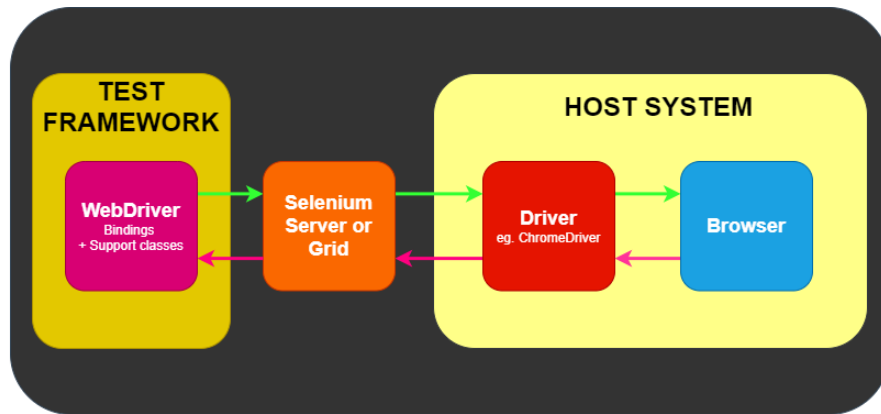
The primary purpose of the WebDriver is to communicate with the browser. It is not a testing tool as it cant be used to compare things, assert, pass or fail, print reports and other standard features of test tools. To fill the spot, different frameworks come into play.

Running WebDriver in tests will require a test framework that matches the language bindings like JUnit for Java and other similar frameworks for different languages.

The test framework is responsible for executing commands to the WebDriver and related steps in tests. [21] A more visual example of the explanation can be found in the figure 3.3.

⁸<https://github.com/karatelabs/karate/blob/master/LICENSE>

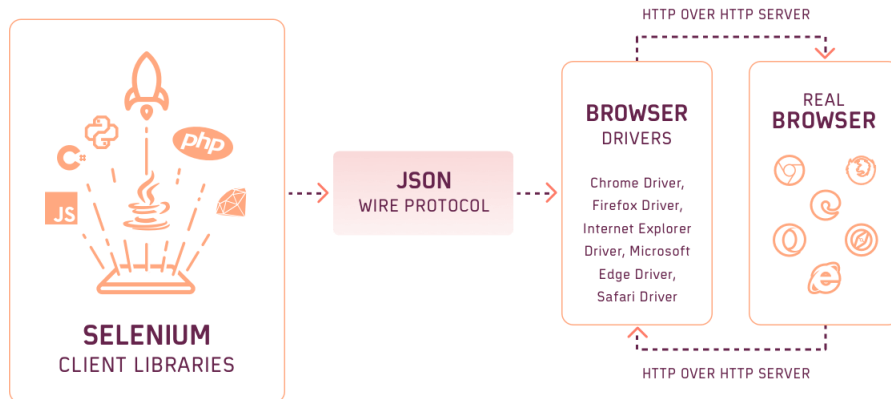
⁹<https://www.karatelabs.io/>



■ **Figure 3.3** Image showing how the Selenium WebDriver testing process works [21]

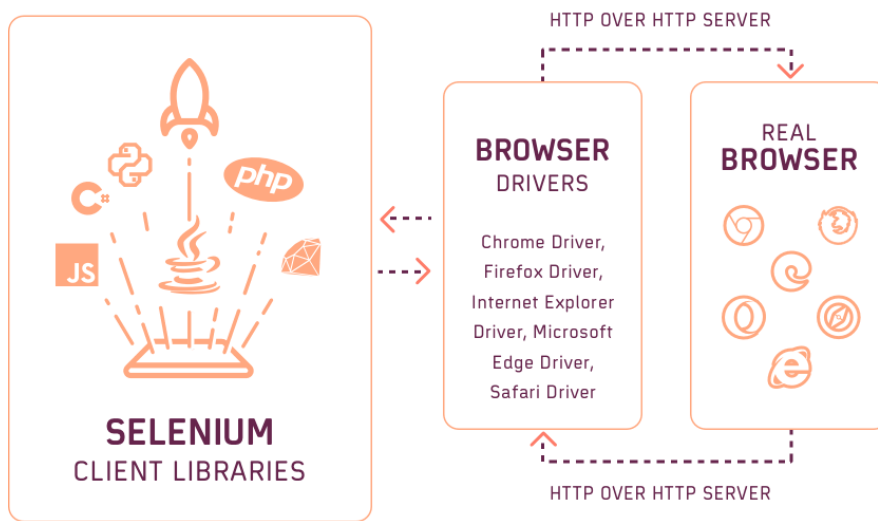
3.3.1.1 The JSON Wire Protocol

In the versions before Selenium 4, the JSON Wire Protocol was used for transmitting commands to the web browser, as shown in the figure 3.4. As the Selenium Client libraries use the JSON Wire Protocol, and the web browser uses the W3C protocol, API calls, encoding and decoding were involved in the entire process, making the communication process slower. [22]



■ **Figure 3.4** Overview of how Selenium WebDriver worked before version 4 [22]

With Selenium 4, the JSON Wire Protocol is no longer supported, as can be seen in the figure 3.5. For backward compatibility, Selenium provides Java bindings and Selenium Server. With a focus on the W3C protocol, Java bindings will persist in being backwards compatible so that tests working on Selenium 3 do not break on Selenium 4. [22]



■ **Figure 3.5** Overview of how WebDriver works since version 4 [22]

3.3.1.2 Browser support

Selenium additionally supports:

- Internet Explorer
- Safari
- Opera

3.3.1.3 Test recording tool

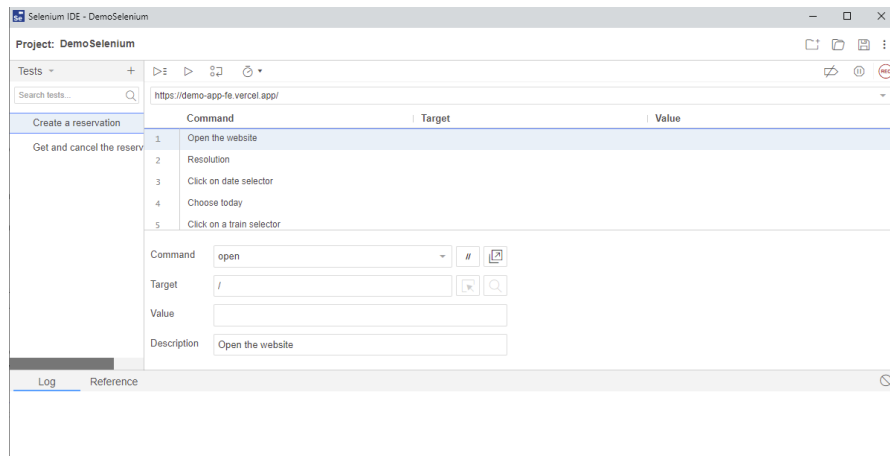
Selenium IDE is the tool used for developing Selenium test cases. It is an easy-to-use extension for Chrome and Firefox and is typically the most efficient method to create test cases.

It can record the users' movements in the browser for different test cases using existing Selenium commands, with parameters determined by the specific element's context, which saves much time and is also an excellent way of comprehending Selenium scripting syntax. [23]

It allows the creation of projects and individual tests inside the project. The recorded or manually created tests can be run and debugged using Selenium IDE.

It offers several options for locating each web element, with the possibility to change the input values for input elements. Steps can be added manually from the available list of commands and even create comments for each test step. Exporting tests into code generated in the language Selenium IDE offers is a great way to save time and speed up the automation process. Even the comments will be exported into the code.

A screenshot of Selenium IDE can be found in the figure 3.6.



■ **Figure 3.6** Image of Selenium IDE

3.3.1.4 Supported data formats

Data formats and their transformations are not part of Selenium's functionality.

Appropriate third-party libraries for the chosen implementation languages need to be used.

3.3.1.5 Reporting

Reporting relies on the testing framework and third-party reporting libraries. Since Selenium is prevalent numerous reporting libraries support it.

3.3.1.6 Language support

Selenium is implemented in many officially¹⁰ and unofficially¹¹ supported programming languages.

3.3.1.7 Parallel testing

Selenium Grid is used to execute WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances. It seeks to provide an easy method to execute tests in parallel on multiple devices. It allows the configuration of different browser versions and browser configurations centrally instead of configuring each test.

It solves a subset of typical delegation and distribution issues, but it will not manage the infrastructure and might not fit some distinct requirements.

It offers a central entry point for all tests and the ability to scale and load balance the nodes. [24]

3.3.1.8 UI automation and selectors

Selenium was the first tool to set an example of how to automate the way users navigate the browsers — the methods of the WebDriver cover the primary commands for the browser like tab switching and file uploading.

Selenium 4 makes it easy to navigate shadow DOMs, iFrames, Alerts and other bothersome web elements. It offers multiple types of waits and listeners for events and event handlers.

¹⁰<https://www.selenium.dev/downloads/>

¹¹<https://www.selenium.dev/ecosystem/>

The execution speed was greatly improved by getting rid of the JSON Wire Protocol. The methods are named and parameterised nicely, so the scripting process is genuinely intuitive.

3.3.1.9 Multifunctionality

The functionality of Selenium revolves around the WebDriver and support for it. Other tools and frameworks need to be used in order for it to function correctly. The performance or API testing is not part of Selenium, so other tools need to be used.

3.3.1.10 Environment configuration

Environment configuration is possible, and there are many examples of how to do it online though it is not an easy process. It entails configuring the drivers for each browser, configuring different pathways and taking care of many other things.

Some tools or services make it easier, but people who want to configure it themselves should know what they are doing.

3.3.1.11 Screenshots and video recordings of tests

Selenium allows taking screenshots, but it requires a lot of external commands to save them. Video recording is possible only with additional libraries that also need to be configured.

3.3.1.12 Documentation and test examples

Selenium maintains good documentation¹² with examples in multiple languages because it is a very known and widely used tool.

Over the years, many tutorials and articles have been published, so the pool of learning material is enormous. Many examples, tutorials and repositories can be found in the publications. One can learn how to configure Selenium to work well, learn how the methods work, and integrate it with other tools properly.

3.3.2 Required technical knowledge

Selenium is a set of intuitive tools. A considerable amount of time is required to learn to use it well. In order to utilise Selenium, it needs to be integrated with many other tools like a testing framework, reporting library, CI configuration and Selenium Grid. . .

When someone intends to use it, they should have experience or good IT skills.

3.3.3 Set up

The setup process differs depending on the programming language chosen for the test scripts.

Nevertheless, one should install an IDE and the prerequisites for the language chosen for the implementation. To use Selenium, one has to install the web drivers for each browser and configure the pathway for them.

Selenium Grid needs to be configured for parallel testing, and nodes need to be connected to it. The process is not trivial.

¹²<https://www.selenium.dev/documentation/>

3.3.4 Integration with other tools

Numerous frameworks and libraries support Selenium, and many testing frameworks are built on top of Selenium. Many versatile Selenium plugins and extensions were developed for the IDEs.

Selenium tests can run in Docker. Selenium has examples of integration with GitHub and GitLab. It is favoured in testing cloud platforms. Many examples of configuring CI/CD pipelines for Selenium can be found.

3.3.5 Community and support

Selenium was initially developed in 2004 by Jason Hugging and is currently managed by the Software Freedom Conservancy. [25]

It is licensed under Apache License 2.0.¹³

It has around 95 000 questions on Stack Overflow, [26] so many problems that users can face while using Selenium might already be solved there. Selenium gained around 30 000 stars and 141 000 used by statistics on GitHub.[27]

Users seeking support can use the bug tracker, user group, chat rooms on multiple chatting platforms, contact sponsors and Slack.

3.3.6 Disadvantages

The test scripts can take a significant amount of time to write and maintain. To debug the code or get to the problematic part, one needs to rerun the test.

The JSON Wired Protocol caused problems in the tests because it added an extra layer of HTTP communication.

It needs to be used alongside many other tools and frameworks to test effectively.

3.4 Appium

Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS and Android mobile platforms and the Windows desktop platform.

The native apps are applications written using the iOS, Android, or Windows SDK. Mobile web apps are web apps accessed using a mobile browser. [28]

The subsequent four principles summarise the principles of Appium:

1. “You shouldn’t have to recompile an app or modify it in any way to automate it.”
2. “You shouldn’t be locked into a specific language or framework to write and run tests.”
3. “A mobile automation framework shouldn’t reinvent the wheel when it comes to automation APIs. (it uses Selenium WebDriver)”
4. “A mobile automation framework should be open source, in spirit and practice as well as in name!” [28]

To uphold the first principle by using the automation frameworks developed by the creators of each platform (Apple, Google, Microsoft), implying that one does not have to compile any additional code with an app.

The second principle is upheld by wrapping the frameworks into one API, the WebDriver API. As discussed with Selenium, one can use any client implementation and test frameworks they choose because Appium and WebDriver clients are automation libraries and not test frameworks.

¹³<https://github.com/SeleniumHQ/selenium/blob/trunk/LICENSE>

The third principle relies on WebDriver because it became a standard for automating web browsers. Creating a new standard for mobile browsers would be a waste of time. Instead, the protocol was extended with extra API methods useful for mobile automation.

The last one speaks for itself because Appium is open source. [28]

3.4.1 Description of features

Appium lets its users write tests on iOS, Android and Windows using the same API, enabling code reuse between iOS, Android, and Windows test suites.

Appium can be configured to operate on an actual mobile device or an emulated mobile device.

The core of Appium is the server that exposes a REST API and receives connections from a client. The server executes the commands sent in by the client on a mobile device and returns an HTTP response that represents the command execution result. This type of architecture allows executing tests on different machines so one can configure it to run in a cloud. The server is written in Node.js. It can be built and installed from a source or installed directly from npm.

The automation is performed in the context of a session. Clients initiate a session with a server in ways specific to each library, but they all send a POST request to the server with a JSON object called the “desired capabilities”. The object is used to configure the environment where it will run the test. Upon the request’s arrival, the server will start up the automation session and respond with a session ID used for sending further commands.

Appium Desktop is a GUI wrapper around the Appium server that can be downloaded for any platform. It comes bundled with everything required to run the Appium server. It also comes with an Inspector, which enables users to check out an app’s hierarchy, a feature useful for writing tests. [28]

3.4.1.1 Browser support

Appium also supports the same browsers as Selenium does because it tries not to reinvent the wheel and uses It for browser testing.

Most importantly, Appium supports Safari on iOS and Chrome or the built-in app for browsing on Android.

3.4.1.2 Test recording tool

Appium does not offer a test recording tool, but one can use Selenium IDE for mobile browser testing because it will use the same calls as the Selenium WebDriver.

3.4.1.3 Supported data formats

Data formats and their transformations are not part of Appium’s functionality.

Appropriate third-party libraries for the chosen implementation languages need to be used.

3.4.1.4 Reporting

Reporting relies on the testing framework and third-party reporting libraries. Many reporting libraries support Appium.

3.4.1.5 Language support

The client libraries are implemented in various programming languages. The list of supported languages can be found on the website¹⁴ with supported clients.

¹⁴<https://appium.io/docs/en/about-appium/appium-clients/index.html>

Beware when using Appium. One needs to use these client libraries instead of the regular WebDriver client. [28]

3.4.1.6 Parallel testing

Appium allows users to automate multiple Android sessions on a single device on a single server instance. State that it is unattainable to have more than one session running on the same device.

Since Xcode9, Appium supports parallel *RealDevice* and *Simulator* testing for iOS.

All it takes is starting the Appium server on any open port and correctly configuring the *desired capabilities*. [29]

3.4.1.7 UI automation and selectors

What was discussed about Selenium WebDriver accounts for Appium's mobile browser testing with some mobile commands added to the API.

As far as the elements of native and hybrid apps go, it takes the abilities of the integrated frameworks that run under its hood.

The strategies for selecting can be seen in the following list:

- Accessibility ID
- Class name
- ID
- Name
- XPath
- Image
- Android UiAutomator
- Android View Tag
- Android Data Matcher
- iOS UIAutomation [30]

Details about the selectors can be found on the Appium elements page.¹⁵

3.4.1.8 Multifunctionality

The functionality of Appium revolves around the automation of mobile applications UI and support for it.

Other tools and frameworks need to be used in order for Appium to function correctly, including emulators, test frameworks, and reporting libraries.

The performance or API testing is not part of Appium, so other tools need to be used.

¹⁵<https://appium.io/docs/en/commands/element/find-elements/>

3.4.1.9 Environment configuration

Appium is very focused on environment configuration, and there are many examples of how to do it locally or in the cloud. It is generally not an easy process.

It entails configuring the drivers for each browser and configuring different pathways. It requires configuring the devices or emulators and their desired capabilities for them, including changing values for the Appium server. . .

Some tools or cloud services make it easier. However, users who take matters into their own hands definitively need to know a lot about the tools and processes if they want to configure and run their tests successfully.

3.4.1.10 Screenshots and video recordings of tests

Appium allows taking screenshots [31] and recording videos. [32] Both options are configurable, and both require some extra configuration and commands to be stored somewhere.

3.4.1.11 Documentation and test examples

Appium has useful documentation¹⁶ with examples in numerous languages.

Extensive material is online explaining how to configure the devices and projects to ensure that everything works correctly.

3.4.2 Required technical knowledge

Appium is a heavy configuration framework it requires basic knowledge about the iOS or Android system and the structure of its applications. When doing mobile browser testing, one also needs to learn Selenium.

When someone intends to use it, they should have experience creating or testing mobile applications and using mobile device emulators.

3.4.3 Set up

The setup process depends on the programming language and the mobile device chosen for the test scripts.

Regardless, one should install an IDE and the prerequisites for the language chosen for the implementation — getting an existing iOS or Android device or installing and configuring an emulator, choosing the proper testing framework and configuring test scenarios. To do mobile web testing, one has to install the web drivers for each browser and configure the pathway for them.

Appium server needs to be installed and configured to run the tests.

3.4.4 Integration with other tools

Multiple frameworks and libraries support Appium, and many testing frameworks are built on top of Appium and Selenium. Multiple Appium plugins and extensions were developed for the IDEs.

Appium tests can run in Docker. Appium is well integrated with GitHub and GitLab. It is often used in testing on cloud platforms. It is well integrated with CI, but configuration for specific environments can be tricky.

¹⁶<https://appium.io/docs/en/about-appium/api/>

3.4.5 Community and support

Appium was initially developed in 2011 by Dan Cuellar and is currently managed by the JS Foundation. [33]

It is licensed under Apache License 2.0.¹⁷

It has around 8 000 questions on Stack Overflow, [34] which is considerable. Appium gained around 15 000 stars and 2 600 used by statistics on GitHub. [35]

Users seeking support can use the Appium discuss forum.¹⁸

3.4.6 Disadvantages

The tests can take a significant amount of time to write and maintain due to the complexity of the configuration and maintenance of devices and emulators.

To debug the code or get to the problematic part, one needs to rerun the test, which can be very slow when using emulators.

It needs to be used alongside many other tools and frameworks to test effectively.

3.5 SikuliX

“SikuliX automates anything you see on the screen of your desktop computer running Windows, Mac or some Linux/Unix. It uses image recognition powered by **OpenCV** to identify GUI components. This is handy in cases when there is no easy access to a GUI’s internals or the source code of the application or web page you want to act on.” [36]

3.5.1 Description of features

SikuliX mainly features functions for visual testing. It has commands that work well for automating repetitive tasks in daily apps, video games, administration, etc.

SikuliX uses the OpenCV package for locating an image on the screen. It is based on OpenCV’s method *matchTemplate()*, which expects an even-sized or larger image (base), where the given image (target) should be searched. [2] In the example 3, an example workflow using SikuliX commands is shown.

```
openApp(someApp) # we use an application someApp
click(imageButton.png) # we click some button
wait(imageExpected.png)
# we wait for the app to react and show the expected result on the screen
type(\some text"); type(Key.ENTER) # we fill in some text and press ENTER
wait(imageExpected1) # again we wait for some expected reaction or result
click(...) # we click ...
```

■ **Code listing 3** Example of the workflow using SikuliX commands [2]

3.5.1.1 Browser support

SikuliX locates elements using images, and browsers have a GUI, so for every browser on a desktop, SikuliX scripts can be written. However, it is not valid if the browser is run in headless mode, then it does not show the GUI.

¹⁷<https://github.com/appium/appium/blob/master/LICENSE>

¹⁸<https://discuss.appium.io/>

3.5.1.2 Test recording tool

SikuliX IDE can set up and maintain visual workflows. [2] More will be discussed in the test implementation part of the thesis.

3.5.1.3 Supported data formats

The primary data formats supported are images. External data must be added using other libraries. That can be quickly done when SikuliX is used as a Java library. Otherwise, it is much more challenging.

3.5.1.4 Reporting

Logs are used in SikuliX IDE or when it is used as a library. Further reporting needs to be configured using other libraries.

3.5.1.5 Language support

- Python 2.7 (supported by Jython, runs in SikuliX IDE)
- RobotFramework text-scripts
- Ruby 1.9 and 2.0 (supported by JRuby)
- JavaScript (supported by the Java Scripting Engine)

SikuliX can be used in Java and with any Java aware programming/scripting language. [36]

3.5.1.6 Parallel testing

SikuliX does not support parallel testing.

3.5.1.7 UI automation and selectors

As already mentioned, SikuliX uses images as selectors for elements. It also features practical functions for automating user actions:

- Click on an element
- Right click on an element
- Find an element
- Double click on an element
- Check whether an element is present on the screen
- Type a string into a text box
- Wheeling on a particular image
- Drag and drop an image/element
- Roll hover on a particular image
- Paste a copied string [37]

3.5.1.8 Multifunctionality

It simply automates the GUI using image recognition to find elements and by simulating simple user commands.

3.5.1.9 Environment configuration

SikuliX is not very configurable. A script written in SikuliX IDE can be executed in the IDE or be exported and run. The screen should be ready in some predefined state when running the script, so the script executes correctly.

When used as a library in Java, it can be reasonably configured. For example, configuring a virtual screen is possible so it can run in the cloud.

3.5.1.10 Screenshots and video recordings of tests

Screenshots can be captured when used as a library. Additional commands need to be run to save it, similar to Selenium. For videos, additional libraries need to be used.

3.5.1.11 Documentation and test examples

SikuliX has good documentation, but it is all over the place. Multiple versions of the documentation exist on different sites. The newest documentation can be found using the link.¹⁹

The documentation lacks solid examples.

3.5.2 Required technical knowledge

Excluding SikuliX's basic commands, users do not need programming or scripting knowledge, but making more advanced scripts requires some knowledge of one of the supported languages.

3.5.3 Set up

SikuliX needs an actual screen running the application under test or at least an equivalent virtual solution. SikuliX is only available on devices running Windows, macOS or Linux. Moreover, it requires Java version 8 or higher. Then it is up to the choice whether to use SikuliX IDE or use it as a Java library.

3.5.4 Integration with other tools

SikuliX is mainly integrable with Java, which means that it can be integrated with multiple testing frameworks. CI/CD pipelines can be configured for tests using SikuliX, but the test must then be connected to a properly configured real or virtual screen.

Tests using SikuliX can run in a Docker container or be deployed in the cloud, but configuring a virtual screen might not be trivial.

3.5.5 Community and support

Sikuli was started in 2009 as an open-source research project at the User Interface Design Group at MIT by Tsung-Hsiang Chang and Tom Yeh. Both left the project in 2012. Raimund Hocke decided to take over development and support and rename it SikuliX. [36]

It is licensed under the MIT License.²⁰

¹⁹<https://sikulix.github.io/docs>

²⁰<https://github.com/RaiMan/SikuliX1/blob/master/LICENSE>

It has around 900 questions on Stack Overflow, [38] which is not much. SikuliX gained around 1 800 stars and 2 400 used by statistics on GitHub. [39]

Users seeking support can use the SikuliX launchpad.²¹

3.5.6 Disadvantages

Scripts written using SikuliX IDE cannot be exported to code, only to runnable *JAR* files.

SikuliX requires an actual screen or a virtual one. Script using reference images captured for a screen with a specific resolution often does not work on another screen or another resolution.

It can be integrated into a sensible advanced test plan only by using Java alongside many other tools and frameworks to test effectively.

Similar to Karate, the project is maintained primarily by one person.

3.6 Cypress

Cypress is a great front end testing tool built for the modern web. It consists of a free, open-source, locally installed Test Runner and a Dashboard Service for recording tests. [40] Cypress is a JavaScript testing framework based on *Chai* library and *Mocha* framework.

3.6.1 Mocha

Cypress has adopted BDD syntax from Mocha to fit perfectly with both integration and unit testing. Mocha offers outstanding *async* support. Cypress has extended Mocha and fixed unpleasant functionalities. These fixes are all fully transparent. All of the tests in Cypress sit on the fundamental saddle Mocha provides, specifically:

- `describe()`
- `context()`
- `it()`
- `before()`
- `beforeEach()`
- `afterEach()`
- `after()`
- `.only()`
- `.skip()` [41]

3.6.2 Chai

Chai provides Cypress with the ability to write assertions quickly. It offers readable assertions with exceptional error messages. Cypress extends this, fixes several typical catches, and envelops Chai's DSL using *subjects* and the `.should()` command. [42]

²¹<https://answers.launchpad.net/sikuli>

3.6.3 Description of features

- Time Travel – using the snapshots of the test run
- Debugging – errors, stack traces, Chrome Devtools
- Automatic Waiting – no need for sleep and waits
- Spies, Stubs, and Clocks – verification of the behaviour of functions, server responses or timers
- Network Traffic Control
- Consistent Results – unlike Selenium
- Screenshots and Videos
- Cross-browser Testing
- Integration with CI/CD tools
- Integration with GitHub and Slack [40]

3.6.3.1 Browser support

In addition to the browsers mentioned at the beginning of the chapter, Cypress supports **Electron** and **Brave**.

3.6.3.2 Test recording tool

Cypress provides a tool for recording tests only in the premium version named *Cypress Studio*.²²

3.6.3.3 Supported data formats

Cypress has built-in support for JSON, YAML and text files for a good TDD setup. Other external data files (CSV, XML) can be added easily using other JS libraries.

3.6.3.4 Reporting

Cypress uses the spec reporter to output information to *STDOUT* in the default configuration.

Reporters built for Mocha can be used with Cypress. *Teamcity* and *JUnit* are the two most common third party reporters for Mocha are built into Cypress.

Creating custom reporters or using any third party reporter is supported in Cypress.

There is also a possibility of merging reports across specs and using multiple reporters. [43]

The Cypress Dashboard is a service that provides access to recorded test results. Typically when running tests from a CI provider. The Dashboard provides insight into what transpired when the tests ran. It is possible to add more additional record keys. The Dashboard is free to use for **non-commercial open-source** public projects. [44]

3.6.3.5 Language support

Cypress supports JS and TypeScript.

²²<https://docs.cypress.io/guides/core-concepts/cypress-studio>

3.6.3.6 Parallel testing

Running tests parallelly across many virtual machines can save time and money when running tests in CI.

While parallel tests can likewise technically run on a single machine, it is not recommended since the machine would demand significant resources to run tests efficiently.

The parallelization system is file-based. In order to use parallelization, tests will need to be split across individual files.

The multiple machines need to be set up in the used CI tool. [45] Furthermore, parallel and cross-browser test execution can be set up together.

3.6.3.7 UI automation and selectors

Cypress has a `cy.get()` command to select elements from the DOM. The command can be chained with other command calls to get a specific element that passes the filters set by the commands like `cy.get('.nav').contains('Hello')`. This can be combined and makes selecting elements more intuitive. Cypress also supports standard XPath and css selectors. Selecting elements relative to an element is also possible in Cypress.

Usages of the `get` command:

- `cy.get(selector)`
- `cy.get(alias)`
- `cy.get(selector, options)`
- `cy.get(alias, options)` [46]

The `cy.contains()` can be used to select elements too. It can select elements that match a regular expression, text, selector or even contain a number for example `cy.contains(2)` will select the first element that contains number 2.

Cypress can work with the shadow DOM and iFrames, though working with iFrames is not very intuitive. Cypress can use multiple tabs. Asserts are very intuitive.

It features retries and auto waits, but they are not ideal and sometimes do not react well to some UI changes. This can be fixed by using manual waits.

Cypress can sometimes select multiple elements from a dynamic element, which can sometimes be problematic.

Cypress offers useful callback functions to create *aliases*, *debugging functions* and *closures* that enable to keep references around to refer to work done in previous commands. [47]

Sharing context between tests is possible using `cy.fixture()`. The call is used for loading a fixed set of data located in a file.

Cypress features multiple events and ways of handling them. Like listening for uncaught exceptions and preventing Cypress from failing the test. Listening for `alert` or `confirm` calls and changing the confirm behaviour. This is primarily used to handle alerts in browses. Listening for `window:before:load` events and modifying the window before any app code is executed between page transitions. Listen for `command:retry` events to comprehend why Cypress is internally retrying for debugging intentions. [48]

File uploading can be done only after installing a plugin.

3.6.3.8 Multifunctionality

Cypress enables writing E2E, integration and unit tests. [40]

Cypress supports UI and API testing and can test some performance aspects but probably does not cover all of the demands for performance testing.

Cypress can be considered a multifunctional tool for testing modern web applications.

3.6.3.9 Environment configuration

Environment variables in Cypress are helpful when:

- Values should be different across developer devices.
- Values should be different across multiple environments: (*dev, staging, qa, prod*)
- Values change frequently and are highly dynamic.

Cypress also has **OS-level** variables. Environment variables can be created and modified easily in 6 different ways:

1. Configuration file (`cypress.json`)
2. `cypress.env.json` (new file)
3. `CYPRESS_*` (OS-level)
4. `-env` (CLI)
5. Plugins
6. Test configuration [49]

3.6.3.10 Screenshots and video recordings of tests

Cypress can take screenshots manually using the `cy.screenshot()` command, and it captures screenshots automatically on a failed test run. However, Screenshots on failure are not automatically taken during `cypress open`. Capturing screenshots on a test failure can be turned off entirely.

Screenshots are stored in `cypress/screenshots` by default. Cypress removes any existing screenshots before `cypress run`. This feature can be turned off. [50]

Cypress records a video per spec file when running tests during `cypress run`. Videos are not automatically recorded during `cypress open`. Video recording can be turned off.

After the `cypress run` completes, the video is automatically compressed to make the file size smaller. It compresses to a **32 CRF** by default, but this is configurable.

Using the `-record` flag for running tests, videos are processed, compressed, and uploaded to the **Dashboard Service** after every spec file runs. To only process videos if tests fail can be set in the configuration.

Videos are stored in `cypress/videos` by default. Cypress removes any existing videos before a `cypress run`. This feature can also be turned off. [50]

3.6.3.11 Documentation and test examples

Authors of the documentation believe that their documentation should be approachable, enabling readers to understand the what fully and why. [40]

The authors achieved their goal well. This documentation discusses all of the aspects of the tool in detail. It has numerous links for easier orientation and offers a vast amount of examples. Example projects were created to showcase most of the features in Cypress.

It might have the best documentation of the analysed tools.

3.6.4 Required technical knowledge

Users should have basic knowledge of JS, programming concepts, HTML, CSS and an overview of how websites function to use Cypress.

3.6.5 Set up

Cypress is a desktop application that needs to be installed.

The application supports these operating systems:

- macOS 10.9 and above (64-bit only)
- Linux Ubuntu 12.04 and above, Fedora 21 and Debian 8 (64-bit only)
- Windows 7 and above (64-bit only)

Node.js is 12 or 14 and above is required for installing Cypress using *npm*. Alternatively, it can be directly downloaded. [51]

3.6.6 Integration with other tools

Many third-party IDE extensions and plugins help integrate IDE with Cypress.

For VSC:

- Cypress Fixture – IntelliSense
- Cypress Helper
- Cypress Snippets
- Open Cypress
- Test Utils
- Intelligent Code Completion [52]

For IntelliJ:

- Cypress Support
- Cypress Support Pro [52]

Free extensions for test recording a then script generation exist, but they are much worse than the tool in the premium version.

Integrations with Jira are allowed in the premium version, but it is still possible to code the integration using Jira API and JUnit reporter. [53]

Cypress has excellent integration with Slack for notifications about test results. [54]

It has numerous different guides and examples of integrations with various CI tools. Including official guides for GitHub actions and GitLab CI, CircleCI, Bitbucket pipelines, AWS Codebuild. Cypress tests are often available in testing clouds. [55]

Cypress probably offers the most significant amount of integrations from the analysed tools.

3.6.7 Community and support

The first commit on Cypress happened in 2014. [56] It is an open-source tool managed by Cypress.io.

It is licensed under the MIT License.²³

It has around 6 000 questions on Stack Overflow, [57] which is a considerable number. Cypress gained around 38 000 stars and 409 000 used by statistics on GitHub. [58] Making it a popular tool with a big community.

Users seeking support can use the Stack Overflow, Git issues or Gitter.²⁴

²³<https://github.com/cypress-io/cypress/blob/develop/LICENSE>

²⁴<https://gitter.im/cypress-io/cypress>

3.6.8 Disadvantages

One disadvantage could be that Cypress is mainly for JS.

Some of the capabilities of Cypress in the premium version are available for free in other tools. §

File uploading can be done only after installing a plugin.

3.7 Puppeteer

Puppeteer is a Node.js library which provides a high-level API to manage Chrome or Chromium over the DevTools Protocol. Puppeteer operates in the headless mode by default but can be configured to operate full (non-headless) Chrome or Chromium. [59]

3.7.1 Description of features

Aims of Puppeteer:

1. Create a slim, canonical library that emphasises the powers of the DevTools Protocol.
2. Supply a reference implementation for similar testing libraries. Eventually, these other frameworks could adopt Puppeteer as their foundation.
3. Increase the adoption of headless/automated browser testing.
4. Aid testing new DevTools Protocol features!
5. Discover more about the pain points of automated browser testing and help seal those cracks. [59]

Chromium principles adapted by Puppeteer:

1. Speed: Puppeteer has nearly zero performance overhead over an automated page.
2. Security: Puppeteer operates off-process to Chromium, making it safe to automate potentially malicious pages.
3. Stability: Puppeteer should not be flaky and should not leak memory.
4. Simplicity: Puppeteer provides a high-level API that's easy to use, understand, and debug. [59]

Puppeteer operates over Google Chrome directly without requiring additional tools like drivers. Setting up automated scripts is faster and simpler.

3.7.1.1 Browser support

Puppeteer supports chromium-based browsers, but experimental Firefox support is available.

3.7.1.2 Test recording tool

Free extensions are available on GitHub and Chrome Web Store. They are mostly not very advanced but they work.

3.7.1.3 Supported data formats

Other JS libraries need to be used for supporting external data because Puppeteer is not built for this.

3.7.1.4 Reporting

Puppeteer does not support reporting. It only outputs logs and error traces. Integrating Puppeteer into testing requires using it with a JS testing framework.

3.7.1.5 Language support

Puppeteer is only available in JS.

3.7.1.6 Parallel testing

It is possible to run Puppeteer scripts in parallel. Primarily by using third party libraries.

3.7.1.7 UI automation and selectors

Puppeteer supports the following type of selectors:

- Type selector
- Class selector
- ID selector
- Attribute selector
- XPath selector [60]

Puppeteer also uses **Selectors API**. Navigation in Puppeteer is done using the **page** object. Things like opening a new page, going back and forward in the page navigation history and reloading a page.

By default, the viewport is 800x600px and can be set to a different value.

Emulating a device can be done by setting the user agent to a specific device and setting the viewport accordingly.

Puppeteer can get the HTML source of a page and can handle multiple tabs, iFrames and shadow DOM. However, it is a bit challenging. Using *eval()* it is possible to get an object from DOM and perform actions on it. Puppeteer allows using JS functions in the page context. Mouse and keyboard operations are available in Puppeteer. Puppeteer can focus on elements and type values into them. Generating a PDF from a page can also be done using Puppeteer. Setting the content of a page is possible too.

Waiting for events and handling them is also part of Puppeteer. It is often used on alerts. Puppeteer extends the **EventEmitter** object from Node.js. [61]

Puppeteer has no asserting capabilities. They have to be added using other libraries.

3.7.1.8 Multifunctionality

Puppeteer can be used in API testing, but it was not created. It can intercept the GET request for the website, change it to another request, and add the body and awaiting the script constructed is wired. Other tools should be used for the purpose.

3.7.1.9 Environment configuration

Puppeteer is a library, so it is up to the user how he configures the tool in different environments. It does not come with a particular configuration file.

3.7.1.10 Screenshots and video recordings of tests

Puppeteer can take a screenshot of the page, saving it to the filename selected using the path as a parameter. Video recording in Puppeteer is only possible using third party libraries.

3.7.1.11 Documentation and test examples

Puppeteer has technical documentation²⁵ that is not too user-friendly but provides good insights for developers. The documentation has some examples, but they are not abundant.

3.7.2 Required technical knowledge

To use Puppeteer, a user should have a solid knowledge of JS and the web because it will require different configurations and integrations to use as a proper testing tool.

3.7.3 Set up

To set up the development process in Puppeteer requires a desktop with Windows, macOS or Linux. The newest version of **Node.js** and an IDE.

3.7.4 Integration with other tools

Puppeteer can be naturally integrated into a lot of different JS testing frameworks. Some extensions for IDE make the development of Puppeteer scripts a bit easier, but there are not many of them. It can be integrated with multiple different CI tools. It can run in a Docker container. Few testing clouds support Puppeteer.

3.7.5 Community and support

Puppeteer was released in 2018. It is an open-source tool managed by Google.

It is licensed under the Apache License 2.0.²⁶

It has around 6 500 questions on Stack Overflow, [62] which is a considerable number. Puppeteer gained around 76 000 stars and 197 700 used by statistics on GitHub. [59] It is a popular tool with a large community.

There are many ways to get help on Puppeteer, but most problems are discussed on Stack Overflow and GitHub issues.

3.7.6 Disadvantages

Puppeteer has no asserting capabilities.

Requires a lot of work and integration with other tools to be used it as a proper testing tool.

Puppeteer is not very beginner-friendly.

3.8 Playwright

Playwright is a framework for automated web testing. It allows testing Chromium, Firefox and WebKit with a single API. Playwright enables cross-browser web automation that is ever-green, capable, reliable and fast.[63]

²⁵<https://pptr.dev/>

²⁶<https://github.com/puppeteer/puppeteer/blob/main/LICENSE>

3.8.1 Description of features

Cross-browser Playwright supports all modern rendering engines, including Chromium, WebKit, and Firefox.

Cross-platform Test on Windows, Linux, and macOS, locally or on CI, headless or not.

Cross-language] Use the Playwright API in TypeScript, JS, Python, .NET, and Java.

Test Mobile Web Native mobile emulation of Google Chrome for Android and Mobile Safari. The same rendering engine works on the Desktop and in the Cloud.

Auto-wait Playwright waits for elements to be actionable prior to performing actions. It also features a rich set of introspection events. Combining the two eliminates the need for artificial timeouts – the primary cause of breakable tests.

Web-first assertions Playwright assertions are explicitly created for the dynamic web. Checks are automatically retried until the necessary conditions are met.

Tracing Configure test retry strategy, capture execution trace, videos, and screenshots to eliminate flakes.

Trusted events Hover elements, interact with dynamic controls, and produce trusted events. Playwright uses real browser input pipeline indistinguishable from the actual user.

Test frames, pierce Shadow DOM Playwright selectors pierce shadow DOM and allow entering frames seamlessly.

Multiple everything Test scenarios that run multiple tabs, multiple origins and multiple users. Create scenarios with different contexts for different users and run them against a server, all in one test.

Browser contexts Browsers run web content belonging to different origins in different processes. Playwright is aligned with the architecture of the modern browser and runs tests out-of-process, making Playwright free of the typical in-process test runner limitations.

Log in once Playwright can save the authentication state of the context and reuse it in all the tests.

Codegen Playwright can generate tests by recording user actions and saving them in any language.

Playwright inspector Inspect page, generate selectors, step through the test execution, see click points, and explore execution logs.

Trace Viewer Capture all the information to investigate the test failure. Playwright trace contains test execution screencast, live DOM snapshots, action explorer, test source, etc. [63]

3.8.1.1 Test recording tool

Codegen is great when used with **Playwright inspector**. The tool is free but not quite as convenient as Selenium IDE.

3.8.1.2 Supported data formats

Third-party libraries are recommended to be installed to operate with external data.

3.8.1.3 Reporting

By default, Playwright generates an excellent HTML web page report with all necessary information about the executed tests, including everything about tracing described in the features. Debugging tests are made much easier with the default reporter and other valuable tools Playwright provides. Multiple reporters can be used, and it can aggregate results from the sharded executions into one. The reporter can be easily changed and configured in the configuration file.

3.8.1.4 Parallel testing

Playwright Test executes tests in parallel. It runs several worker instances that operate at the same time. Tests in a single file are run in the order in the same worker process but can be configured to run in a single file in parallel. The whole project can be configured to have all tests run in parallel. Limits for workers and failed tests can be set. Test executions can be sharded to run different parts of the test execution on multiple machines. Parallelism can be disabled by limiting the number of workers to one. [64]

3.8.1.5 UI automation and selectors

Playwright uses Puppeteer as one of the essential layers of the program.

Playwright supports standard XPath and CSS selectors. It also features a text selector, filter by another locator, inside CSS selector, n-th element selector, role selectors, id, name and supports some combinations. [65]

Playwright Test uses **expect** library for test assertions. This library provides many matching functions like *toEqual*, *toContain*, *toMatch*, *toMatchSnapshot* and more. [66]

It supports mobile testing using mobile browsers and has experimental features for Android.

Alerts are handled using events similar to Puppeteer. File uploading can be done by default but is not configured intuitively.

The syntax is usually intuitive and logical.

Tests are executed swiftly.

3.8.1.6 Multifunctionality

Playwright is a reliable multifunctional tool for testing modern web apps. It supports API testing, and the tests in Playwright look very similar to the ones in Cypress. It can also be used in performance testing a bit but not in most aspects.

3.8.1.7 Environment configuration

Playwright provides options to configure the default browser, context, page fixtures, screenshots and videos. For example, there are options for *headless*, *viewport* and *ignoreHTTPSErrors*.

There are many testing options like *timeout* or *testDir* that configure how tests are collected and executed. Options can be set globally in the configuration file, most locally in a test file.

Each project can be configured separately and run a different set of tests with different options. Options can be set to configure which tests should the project run and how should they be set up and teardown. This is useful for configuring tests to run in different environments. [67]

3.8.1.8 Documentation and test examples

Playwright has beautiful example-rich and user-friendly documentation. Examples are available in Typescript and JS. The documentation is well connected using links and contains setup examples in different CI tools.

3.8.2 Required technical knowledge

Users of Playwright should have basic knowledge of JS, HTML, CSS and programming concepts.

3.8.3 Set up

Playwright requires a desktop with Windows, macOS or Linux. The newest version of **Node.js** and an IDE.

3.8.4 Integration with other tools

Playwright can be integrated with **Slack**, but the integration is not official.

A pre-built **Docker** image is available, which can be used directly or as a reference to edit existing Docker definitions. **CI** set-up is available in the documentation for GitHub actions, GitLab, AWS pipelines, CircleCI, Jenkins and Bitbucket Pipelines. [68]

Mocking can be integrated into Playwright.

Useful extensions are primarily available for **VSC** like a test runner and a few for **IntelliJ** tools.

3.8.5 Community and support

Playwright was released in 2020. [69] A big part of the team previously working on the Puppeteer currently work on Playwright. Playwright is under the management of **Microsoft**.

Playwright is licensed under the Apache License 2.0.²⁷

It has around 800 questions on Stack Overflow, [70] which is not a lot because it is new. Playwright gained around 35 000 stars and 11 000 used by statistics on GitHub. [71] It is becoming a more popular tool and is growing an extensive community.

Users seeking help can visit GitHub issues, Stack Overflow or might try their luck on Microsoft support.

3.8.6 Disadvantages

It is relatively new and does not offer official integrations for Jira, Slack or Microsoft teams. Playwright does not support DDT by default.

3.9 TestCafe

TestCafe is a JS framework running on **Node.js**, and it aims to simplify E2E testing.

3.9.1 Description of features

TestCafe additionally supports TypeScript. It uses the browsers already installed on the system and does not require WebDriver. It can do cross-browser testing without manual timeouts and using cumbersome boilerplate expressions. Resulting in less time tracking down annoying issues and more time doing what matters most, which is writing code in a framework with very intuitive syntax. [72]

²⁷<https://github.com/microsoft/playwright/blob/main/LICENSE>

3.9.1.1 Browser support

In addition to browsers supported by all analysed tools, TestCafe supports:

- Internet Explorer
- Safari
- Opera

3.9.1.2 Test recording tool

The test recording tool is only available in the premium version.

3.9.1.3 Supported data formats

TestCafe does not have built-in support for data formats to be used for DDT, excluding JSON, but since it is a JS tool, many other libraries can help.

3.9.1.4 Reporting

TestCafe has following built-in reporters:

- spec
- list
- minimal
- xUnit
- JSON

The default reporting setting is set to **spec**. It primitively reports to the CLI, so another reporter should probably be configured instead. A custom reporter can be created to fulfil one's needs. The community has developed custom reporters for Slack, NUnit, TeamCity and Tesults. [73]

3.9.1.5 Parallel testing

TestCafe has a concurrent mode to run multiple browser instances simultaneously. It can be enabled using the concurrency configuration file property. With concurrency enabled, tests are run in parallel, decreasing the execution time of the test suite, but requiring more resources from the test environment. The headless mode of browsers can be used to speed up execution time. [74]

3.9.1.6 UI automation and selectors

In TestCafe, it is possible to create custom selectors and import them to tests, an interesting feature that seems to complicate things redundantly. By default, it does not support XPath, but the implementation can be found and imported from GitHub.

Using the **Selector** object allows filtering elements in DOM by chaining different methods and using matches by index, text, attribute, visibility or relativity to a selected element like a parent, child, next sibling, or previous sibling. The selector object also allows shadow root navigation. DOM node state can also be accessed. TestCafe also supports framework-specific selectors (React, Angular, etc.). [75]

TestCafe supports basic user action methods, including file uploading, iFrames and handling dialogues.

Smart assertions that can do retries. For asserting the BDD inspired *expect* method is used. *Regex* can be used for matching text. Page element properties can be accessed for assertion. [76]

TestCafe has a speed variable that can be changed in the configuration to speed up or slow down the pace of tests. TestCafe features request mocking and server-side caching to speed up some parts of the tests.

Roles can be used for user authentication. It takes data from *cookies*, *sessionStorage* and *localStorage*. [74]

3.9.1.7 Multifunctionality

The functionality of TestCafe revolves around user E2E testing. It does not support API testing and probably should not be used for performance testing.

3.9.1.8 Environment configuration

Launch options can be specified from CLI.

A configuration file can be either in JSON or JS. JS is the better option because it is dynamic.

The configuration file allows managing many settings like browsers, reporters, screenshots, videos, debugging, concurrency, timeouts, variables, etc.

The *export* command can be used to create an environment variable, and it can be accessed using the *process.env* property. [77]

3.9.1.9 Screenshots and video recordings of tests

TestCafe allows taking screenshots of the tested webpage and recording videos of test runs. Options can be configured, and screenshots can also be taken arbitrarily using *takeScreenshot()* method. [78]

3.9.1.10 Documentation and test examples

The documentation of TestCafe is clear and well optimized for orientation. It features good examples of how to use methods or configure some parameters. Implemented tests can be found on the official GitHub.

3.9.2 Required technical knowledge

Users of TestCafe should have solid knowledge of JS, HTML, CSS, DOM structure, programming concepts and should definitely read the documentation.

3.9.3 Set up

To set up TestCafe, one should possess a desktop with Windows, macOS or Linux, and have the newest version of **Node.js**, IDE and some time to read the documentation, import commands from GitHub, and configure the framework to work well.

3.9.4 Integration with other tools

Thankfully for community members, TestCafe reports can be integrated with **Slack**. [73]

The documentation features numerous guides on configuring TestCafe to run using different CI tools like GitHub actions, GitLab, Jenkins, etc. [79]

A few extensions for TestCafe are also available for IDEs.

TestCafe it can be run on BrowserStack, SauceLabs and LambdaTest testing clouds.

3.9.5 Community and support

TestCafe was released in 2016. [80]

TestCafe is under the management of **Developer Express Incorporated**.

It is licensed under the MIT License.²⁸

It has around 1 600 questions on Stack Overflow, [81] which sounds OK.

TestCafe gained around 9 300 stars and 10 200 used by statistics on GitHub. [82] TestCafe has a good community of users and is not unknown, but it can be probably said that this tool will grow but not much.

More focus might be put on their premium version than the open-sourced one, but the team is reportedly committed to the open-source community and is actively upgrading TestCafe.

Users seeking help can visit GitHub issues, Stack Overflow.

3.9.6 Disadvantages

TestCafe does not create the configuration file upon installing from *npm*.

Documentation needs to be read extensively before starting development.

The selector object is complicated initially, and importing selectors from GitHub is not ideal.

3.10 Nightwatch.js

NightwatchJS is an integrated, easy to use, E2E testing solution for browser-based applications and websites and is running on Node.js. It uses the W3C WebDriver API to perform commands and assertions on DOM elements.

NightwatchJS has a clean and simple syntax so that tests can be written swiftly. It comes with a built-in test runner, it has support for page objects within the framework, and it can be easily extended to create items like custom reporters.[83]

3.10.1 Description of features

NightwatchJS is an integrated, easy to use, E2E testing solution for browser-based applications and websites and is running on Node.js. It uses the W3C WebDriver API to perform commands and assertions on DOM elements.

NightwatchJS has a clean and simple syntax so that tests can be written swiftly. It comes with a built-in test runner, it has support for page objects within the framework, and it can be easily extended to create items like custom reporters.[83]

Cloud Testing Support Works with BrowserStack out of the box and other clouds can be easily added.

WebDriver Service Manages Selenium or WebDriver services (ChromeDriver, GeckoDriver, Edge, Safari) automatically in a separate child process.

Continuous Integration JUnit XML reporting is built-in for integrating tests in the build process with systems such as Teamcity, Jenkins, CircleCI etc.

Plugin API Flexible command and assertion framework makes it easy to implement custom plugins and extend the built-in commands and assertions APIs. [83]

²⁸<https://github.com/DevExpress/testcafe/blob/master/LICENSE>

3.10.1.1 Test recording tool

Free extensions are available on Chrome Web Store or GitHub for generating NightwatchJS scripts, but they are not customisable, but they will record some of the actions, and that should suffice.

3.10.1.2 Supported data formats

Similar to JS frameworks mentioned before excluding Cypress, NightwatchJS does not have built-in support for data formats to be used for DDT, excluding JSON. However, since it is a Node.js tool, additional libraries can solve the problem.

3.10.1.3 Reporting

By default, NightwatchJS reports into the console, which can be changed using third party or custom reporters.

3.10.1.4 Parallel testing

Parallel testing can be configured in the test runner settings.

3.10.1.5 UI automation and selectors

NightwatchJS allows locating web elements using the *element()* object for CSS and XPath selectors or particular Nightwatch selector objects. However, methods that switch between the two need to be called in the browser context, making it strange. NightwatchJS allows usage of Selenium locators and instances of **WebElement** objects, which makes it easy for Selenium users to select. [84]

Since **WebElement** instances are available in NightwatchJS supports its methods.

NightwatchJS features standard user actions and offers advanced methods like *keyDown()* and *keyUp()* for the keyboard and *press()* and *release()* for the mouse. Alerts, iFrames and file uploading are handled intuitively. [85]

Ensure API allows waiting until a specific condition is met regarding the state of DOM. [86]

The built-in extendable **assert/verify** library is available as two namespaces containing the same methods which perform assertions on elements:

.assert Upon failing an assertion, the test terminates, skipping all other assertions.

.verify Upon failing an assertion, the test logs the failure and resumes with further assertions.

Negate assertions are available for both versions of calls to negate the assertion logic. For example, *browser.assert.not.visible('.visible')* should fail because the element is visible even though we wanted it not to be visible.

Assertions can automatically retry, and timeout can be configured. [87]

3.10.1.6 Multifunctionality

NightwatchJS is a good E2E testing tool, and it is not made for API and performance testing.

3.10.1.7 Environment configuration

The *nightwatch.json* file allows managing many settings like test suites, sources, environments, browsers, reporters, screenshots, concurrency, timeouts, variables, etc.

Multiple environments of test settings can be defined so that they can overwrite specific values per environment. A “default” environment is demanded. An environment inherits all the base settings and settings defined under the “default” environment and can overwrite settings as required. [88]

3.10.1.8 Screenshots and video recordings of tests

Configurable screenshots are built-in and can also be taken arbitrarily. Recording videos can be done using third party libraries.

3.10.1.9 Documentation and test examples

The documentation contains detailed information for developers and many usage examples. However, it is not orientation friendly and well displayed.

3.10.2 Required technical knowledge

Users should be on a similar level recommended for Playwright, they should have basic knowledge of JS, HTML, CSS, DOM structure, programming concepts.

3.10.3 Set up

The set-up process is identical to TestCafe and Playwright.

3.10.4 Integration with other tools

NightwatchJS can be unofficially integrated with Slack. [89]

It can run in a **Docker** container [90].

It integrates with CI tools, but guides or examples are not available in the documentation.

A small number of extensions for IDEs exists, but it is still better than nothing.

3.10.5 Community and support

NightwatchJS was created in 2014 by Andrei Rusu. [83]

NightwatchJS is under the management of **BrowserStack**.

It is licensed under the MIT License.²⁹

It has around 1 600 questions on Stack Overflow, [91] which is roughly the same as TestCafe.

NightwatchJS gained around 11 100 stars and 133 000 used by statistics on GitHub. [92]

NightwatchJS is highly configurable, which opens a window for the skilful users from the community to create and share their modifications and plugins. It has the potential to grow.

Users seeking help can visit GitHub issues, Stack Overflow and Discord. The team is committed to the open-source community and actively upgrades NightwatchJS and adds exciting features like Selenium support.

²⁹<https://github.com/nightwatchjs/nightwatch/blob/main/LICENSE.md>

3.10.6 Disadvantages

It is only available in JS.

Switching between XPath and CSS selectors should be removed, and the selectors should be united.

The documentation could be made better for exploration.

3.11 TestProject

TestProject is a great free E2E automation platform for web, mobile, and API testing that is built on top of **Selenium** and **Appium**.

Most testing can be done using TestProject from the browser on the platform, but there is also an offline version available if storing data in the cloud is a problem.

3.11.1 Description of features

Easy to get started with The TestProject recorder is a powerful and easy to use record and playback tool that helps anyone get started with making tests with a minimal learning curve

Full team collaboration Test automation works best when the whole team can work together on the same platform. TestProject makes sharing tests between team members straightforward and simple

Extensibility There is a library of shared add-ons available to help extend the default capabilities of TestProject. Teams of people can also create their add-ons to simplify the work.

Integration into existing workflows TestProject has an API that can run within existing continuous integration workflows. It also has a developer SDK that allows creating or importing existing tests that users might have into the platform.

Cross Browser and Cross Platform Creating and running mobile tests (Android and iOS) is possible. TestProject can be installed on any Windows, macOS and Linux. It only takes a quick install to access testing on all connected platforms and browsers instantly.

Reliable Technology TestProject uses reliable and proven technology like Selenium and Appium for a robust and well-understood way of interacting with web pages.

Free! The Free plan that TestProject offers is unparalleled in test automation in terms of features and capabilities. TestProject is a powerful and fully-featured product that anyone can use for free. [93]

3.11.1.1 Browser support

In addition to browsers supported by all analysed tools, TestProject supports **Safari** and **Internet Explorer**. It supports running in **headless** mode. It can connect to browsers installed on the device where an agent is running and to **SauceLabs**.

3.11.1.2 Test recording tool

The test recording tool must be the best free test recording tool currently available. More is covered in the implementation part of the thesis.

3.11.1.3 Supported data formats

DDT is supported in TestProject using parameters. External data can be uploaded in CSV format. [94]

3.11.1.4 Reporting

Reporting is done by default, and the reports look astonishingly professional. The reporting menu is described in detail with images in the implementation part of the thesis.

3.11.1.5 Language support

TestProject has SDK available in **Python, Java and C#**.

3.11.1.6 Parallel testing

A parallel task, likewise known as an *Agent Worker*, can be a test/job execution, recording, or OpenSDK development session. Running tests and jobs in parallel demands extra resources and capabilities from the machine they are utilising. Appropriate settings must be configured for the **Agent**, so it does not starve the CPU and memory. There are options to run browsers or tests in parallel or both simultaneously. [95]

3.11.1.7 UI automation and selectors

Web automation in TestProject runs Selenium under the hood, so it offers all of Selenium's commands and selectors. The commands can be accessed using OpenSDK or better in the test creation menu in the browser. Simply by finding the command in the menu and changing its parameters, inputs and output from the UI without writing code. By chaining further commands step by step, a test is created.

The commands include asserting, file uploading, iFrame switching, alert handling, etc. Furthermore, when the usual commands are not enough using multiple add-ons is possible in a few clicks.

This could also be a great way to utilise the work of complete beginners while giving them a platform to comprehend some of the abstract concepts.

The tests written in the browser can be exported into code written in chosen language running on OpenSDK, and tests written using OpenSDK can be imported to the platform.

More is covered in the implementation part of the thesis.

3.11.1.8 Multifunctionality

TestProject is a multifunctional tool as it can be used in API testing using cleverly designed add-ons with commands in which asserting the response and saving the output can be configured. It can also test mobile applications and allows offline and online development.

3.11.1.9 Environment configuration

Configuration of test and project parameters can be done in TestProject. Tests can be organised in different folders, and the values of parameters can be changed for different environments.

3.11.1.10 Screenshots and video recordings of tests

TestProject can be configured to take screenshots on different occasions or by simply executing the command for taking a screenshot. Videos can be currently recorded by integrating with BrowserStack, which costs money, or by coding the functionality using OpenSDK.

3.11.1.11 Documentation and test examples

The documentation of TestProject is written well and offers plenty of visual examples. Articles discussing aspects or containing tutorials for TestProject are available on their website.

3.11.2 Required technical knowledge

The users should know how to use a computer and be aware of the website structure. A total beginner in testing and programming can handle it with some guidance.

3.11.3 Set up

Firstly, an account must be created for TestProject. Then a TestProject Agent must be downloaded and installed on the selected device along with the browsers. After the installation, the Agent should be configured and registered to the created account with an API key. The setup is ready, and tests can be created and executed. [96]

3.11.4 Integration with other tools

TestProject can be officially integrated with Jenkins, TeamCity, CircleCI, GitHub and GitLab CI tools.

It can also be integrated with Slack, and email notifications can be set up too. [97]

TestProject Agent can run in a Docker container [98] and even on Kubernetes. [99]

Finally, it can be integrated with BrowserStack and SauceLabs cloud testing services. [97] The integration guide can be found in the documentation.

3.11.5 Community and support

TestProject was publicly launched in 2018.

TestProject is currently under the management of **Tricentis**. [100]

It is offered as a free forever service, and the OpenSDK is licensed under the Apache License 2.0.³⁰

Users seeking help can visit their forum or contact the team directly from the project using the help-seeking window. The support is said to be responsive.

The team and the open-source community actively upgrade TestProject and add new exciting add-ons.

3.11.6 Disadvantages

Using TestProject in the cloud mode can be a problem when under maintenance. The tests are inaccessible.

Importing tests to the platform can be challenging.

3.12 WebdriverIO

WebdriverIO is a modern Node.js E2E test automation framework that allows running tests based on the W3C WebDriver protocol or Chrome DevTools Protocol and Appium. [101] WebdriverIO offers third-party integrations that make testing and debugging a lot more efficient. [102]

³⁰<https://github.com/testproject-io/java-opensdk/blob/master/LICENSE>

3.12.1 Description of features

WebdriverIO Framework can be configured to Page Object Model. It does not restrict origins. Testers can automate them unconditionally. It is highly extendable so that users can customize the framework as they please. [103]

3.12.1.1 Browser support

WebdriverIO additionally supports Internet Explorer and Safari. [104]

3.12.1.2 Test recording tool

Katalan Recorder is an extension for Chromium browsers that allows test recording and code generating. An extension for WebdriverIO syntax export can be added to it.

3.12.1.3 Supported data formats

Similar to JS frameworks mentioned before excluding Cypress, WebdriverIO does not have built-in support for data formats to be used for DDT, excluding JSON.

3.12.1.4 Reporting

WebdriverIO reports to the console by default but comes with a vast list of reporters that can be easily configured instead, like *JUnit*, *Video*, *HTML*, *JSON* and *Mochawesome Reporters*. The list is far from over. [103]

3.12.1.5 Language support

Scripts are mainly written in JS, but TypeScript support can be configured. [103]

3.12.1.6 Parallel testing

Users can configure WebdriverIO to launch multiple instances and execute tests parallelly by changing the settings of max instances and grouping tests into suites. [105]

3.12.1.7 UI automation and selectors

WebdriverIO supports Jasmine and Mocha testing frameworks and the Cucumber framework. [106]

By default, WebdriverIO has a built-in assertion library that uses the `expect()` function. It allows powerful assertions on various aspects of the browser or elements within the web page. It extends **Jests Matchers** functionality with additional E2E testing optimized matches.[107]

It has no interactive or development mode. Debugging can be done using available debug methods and installing third-party integrations that simplify debugging. [103]

Scenarios using an iFrame can be automated using simple web driver commands. WebdriverIO supports switching to and from multiple windows and tabs. It tries to shorten the length of scripts by using the **\$** and **\$\$** characters for selectors compared to other analysed JS tools. It features XPath and CSS selectors and many other valuable selectors that use *link text*, *name*, *tag*, *id* and *deep selectors* for shadow DOM. It also has numerous selectors for mobile devices. [108]

It can intuitively handle alerts and upload files. WebdriverIO can use mocks and spies and add custom commands. [103]

3.12.1.8 Multifunctionality

The primary purpose of WebdriverIO is E2E testing which is why it cannot be used in API testing. It is integrable with **PerformanceTotal**. [109] **Google Lighthouse** can also be integrated using the **Devtools service**. [110] Both integrations enable measuring performance during testing, but that does not make WebdriverIO a performance testing tool because it lacks functionalities.

3.12.1.9 Environment configuration

The configuration file is located in *wdio.conf.js*. Similarly to a configuration file used by Nightwatch.js, it is used to configure *test suites, outputs, reporting, parallel execution, drivers, timeouts, hooks*, etc. [111] Importantly, test suites and environmental variables can be configured there as well.

3.12.1.10 Screenshots and video recordings of tests

Screenshot saving can be configured and also invoked using the WebDriver API. Video recording can be set up using a video reporter. [112]

3.12.1.11 Documentation and test examples

The documentation is user friendly and well-linked. It provides appropriate examples and guides for integrations and services.

3.12.2 Required technical knowledge

Users should be on a similar level recommended for Playwright and NightwatchJS, they should have elementary knowledge of JS, HTML, CSS, DOM structure, programming concepts.

3.12.3 Set up

The setup is analogous to other JS frameworks. It requires a desktop with a suitable operating system and having **Node.js** at least *v12.16.1* or higher installed. [113]

3.12.4 Integration with other tools

WebdriverIO provides support for multiple BDD/TDD test frameworks and will run tests locally or in the cloud using **Sauce Labs, BrowserStack, TestingBot, LambdaTest, Cross-BrowserTesting and Perfecto**. Regarding CI tools WebdriverIO supports GitHub actions, Jenkins, Bamboo and Travis CI. [101]

WebdriverIO can run in a Docker container. [114]

It supports numerous services, including Slack [115] and Microsoft Teams. [116]

Autocomplete extensions were developed for VSC and IntelliJ IDE. [117]

3.12.5 Community and support

The first releases of WebdriverIO could be seen in 2013. [102] WebdriverIO is under the management of **JS Foundation**.

It is licensed under the MIT License.³¹

³¹<https://github.com/webdriverio/webdriverio/blob/main/LICENSE-MIT>

It has around 1 500 questions on Stack Overflow, [118] which is a bit less than TestCafe and NightwatchJS.

WebdriverIO gained around 7 500 stars and 40 700 used by statistics on GitHub. [102]
Help can be found on Stack Overflow, Slack, Gitter and GitHub discussions. [119]

3.12.6 Disadvantages

The syntax of WebdriverIO can be confusing for beginners.

No reliable test recording tools are available.

It needs to be initially configured a lot to be valuable.

3.13 Summary

In this chapter, the tools were analysed in-depth on their capabilities and shortcomings.

Cucumber has an interesting approach to testing with its *feature* files and function mappings, but it is definitively not for everyone. Each team should decide if it is worth it for them. I would not recommend it, as writing the functions matching specific steps is constraining and bothersome.

Selenium and Appium have presented themselves as valuable tools that drive other tools and frameworks. Selenium was used in Nightwatch.js, TestProject and WebdriverIO, and Appium was used in Karate, TestProject and WebdriverIO.

Karate has shown that it is a jack of all trades that simplifies syntax and tries to keep all testing within one framework.

SikuliX emerged as a helpful accessory that can be integrated into a custom Java testing framework with other tools.

TestProject is an unbelievably beneficial beginner-friendly tool that probably beats most of the tools in terms of the quality of functionalities in its default settings. It appeals to me the most from all of the analysed tools. Though its strength is its weakness, having everything available on the cloud is great, but it is a free service, so security and availability can be concerning. However, when set up in the offline mode, a part of its functionality and simplicity is traded for configuration and data management.

Regarding JS frameworks, Cypress and Playwright are the most developer-friendly, advanced, and popular. If having a lot of different integrations and services readily available is an essential factor for selection, then WebdriverIO would be a great candidate.

Nightwatch.js has exciting capabilities, but I would not choose it over Cypress and Playwright due to the development experience that can be had with those tools.

TestCafe did not impress me with a missing XPath selector and other parts of its functionalities. Coding selectors by making operations with DOM can be a memorable exploratory experience, but not when the goal is to write a test, then not so much.

Concerning fact can be that Cypress and TestCafe offer premium versions of their products that have functionalities that other open-source frameworks offer for free, like test recording and professionally looking reporting. The developers might focus on the premium aspect more than the open-source one, and the tool might start to deteriorate in the eyes of the community.

Quantitative analysis

For the quantitative analysis part of the thesis, a decision was made to create a Google sheet table to compare the tools using weighted parameters. These parameters will try to categorize a tool using objective and subjective metrics on which should be the tools graded.

The grades are then summed into total, and the tool with the highest points is considered the most recommended (with the selected weights). The explanation for grading of each parameter and the entire table can be found using a link.¹

Explanation for chosen parameters:

Browser support In E2E, testing a web application's browser support is crucial. More browsers supported means more points for the tool.

Test recording tool It is very convenient for the developer of the test scripts if the tool allows him to record his test as if he was the actual user of the app and just recorded scenarios. It is a massive bonus if the tool offers a test recording capability. The better the recording tool is, the more points it gets.

Required technical knowledge If the tool is hard to use, configure or has a steep learning curve, it gets fewer points.

Supported data formats Many tests use data for input and output, which is especially important in DDT. The more formats they support, the better points they get.

Reporting Reporting is paramount for the testing process. The reports need to be in a suitable format and preferably exportable.

Language support This parameter is essential if the codebase should be based in a specific language or if having an option to write tests in many languages is advantageous.

Parallel testing This parameter is crucial for the rapid execution of the tests.

Web UI automation This is an integral part of the UI aspect of the test automation. It grades the tools on how they handle certain parts of web elements and other aspects of UI testing.

API testing If the tool includes API testing, it gets more points. They get fewer points if API tests can be run in the same script but using different libraries.

Performance testing If the tool includes performance testing, it gets more points. They get fewer points if performance tests can be run in the same script but using different libraries.

¹<https://docs.google.com/spreadsheets/d/1mzyTKND-6bCwpxybEJpuiv5uQWvMIPGpBKQxkoqyzf8/edit#gid=699299515>

Development platform variability Development platforms are Windows, macOS, and Linux. Point per supported platform.

Environment configuration Meaningful category when it is essential to run tests in different environments like VM, Cloud, Desktop, Mobile, Emulator, or it is essential to configure different parameters.

Set up difficulty The harder it is to install the prerequisites and configure the run and development environment, the fewer points it gets.

Maintenance How hard is it to maintain the tests, configurations, and devices. The harder it is, the fewer points it gets.

Integration with other useful dev tools This is a vital category because, in a modern environment, the tools should be integrable with many other tools like CI/CD or communication tools.

Community/Support This parameter rates the tool by downloads, community, age and company size.

Screenshots and Video recordings of tests This parameter is useful for reporting and debugging.

Multifunctionality This parameter is vital if it is crucial to have one tool for everything and minimize the need to use many different tools for each part of the testing.

Documentation and test examples This is important, especially if a user needs to get started with the tool or needs to find out how the tool works. The more convenient the documentation and vast the examples are, the more points for the tool.

Selectors This is also an important parameter that touches on Web UI automation. The more selectors the tools support or if it has any other exciting capabilities with selecting web elements, the more points it gets.

4.1 Browser support

Table 4.1 shows points given to tools if they support the browser.

One point was given if it supported the browser, and 0 points were given if the tool did not support it.

Browser	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Chrome	1	1	1	1	1	1	1	1	1	1	1	1
Firefox	1	1	1	1	1	1	1	1	1	1	1	1
Edge	1	1	1	1	1	1	1	1	1	1	1	1
Internet explorer	1	1	0	1	0	0	0	1	1	1	1	1
Safari	1	1	0	1	1	1	0	1	1	1	1	1
Headless mode	1	1	1	0	1	1	1	1	1	1	1	1
Opera	0	1	0	1	1	1	0	1	0	0	0	1
Sum	6	7	4	6	6	6	4	7	6	6	6	

■ **Table 4.1** Table with supported browsers

4.2 Data formats

In the table 4.2 one can see points given to tools according to how they support data types.

Two points were given if the tool had built-in support for the data type, One point was given if the tool could be integrated with other libraries that support the data type. 0 points were given if the tool did not support the data type.

Data type	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
CSV	2	1	1	1	1	1	1	1	1	2	1	1
JSON	2	1	2	1	1	2	2	2	2	2	2	1
XML	2	1	1	1	1	1	1	1	1	2	1	1
YAML	2	1	2	1	1	1	1	1	1	2	1	1
Text files	1	1	2	1	1	1	1	1	1	2	1	1
Sum	9	5	8	5	5	6	6	6	6	10	6	

■ **Table 4.2** Table with supported data formats

4.3 Language support

Table 4.3 displays points given to tools if they support the programming language meaning that it is possible to implement test scripts in them, or it offers interoperability or bindings.

One point per supported language and 0 points were given otherwise.

Language	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Java	1	1	0	1	1	1	0	0	0	1	0	1
Javascript	1	1	1	0	1	1	1	1	1	0	1	1
Python	0	1	0	1	1	1	0	0	0	1	0	1
Ruby	0	1	0	1	1	0	0	0	0	0	0	1
PHP	0	1	0	0	1	0	0	0	0	0	0	1
C#	0	1	0	0	1	1	0	0	0	1	0	1
Other	1	1	1	0	1	1	1	1	0	1	0	1
Sum	3	7	2	3	7	5	2	2	1	4	1	

■ **Table 4.3** Table with programming languages

4.4 Dev platforms

All tools got 3 points per supported platform (macOS, Windows, Linux).

4.5 Reporting

Table 4.4 shows tools rated on their reporting capabilities like logging and showing test steps having an excellent report with graphs, and having the ability to export the report.

The first two parameters were graded 0 or 1 depending on whether the tool supported the feature or not. The other parameters were rated on an integer scale from 0 to 2, depending on the quality of supportability.

Reporting type	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Logs	1	1	1	1	1	1	1	1	1	1	1	1
Test steps	1	1	1	0	1	1	0	1	0	1	1	1
Graphs	1	1	1	1	1	1	1	1	1	2	2	1
Report format	2	1	1	1	1	2	1	1	1	2	2	1
Exportable report	1	1	1	1	1	1	1	1	1	2	1	1
Sum	6	5	5	4	5	6	4	5	4	8	7	

■ **Table 4.4** Table with reporting parameters

4.6 Selectors

The tools are rated on their selector capabilities in the figure 4.5.

One point is given to the tool for supporting the following type of selectors: CSS selectors, XPath selectors, Images (using image recognition), Selecting relative to some elements, Mobile app selectors and Desktop app selectors. If the selector is not supported, 0 points are given.

The “custom useful selectors” parameter is rated from 0 to 2 points based on the subjective experience of the benefit and sophistication of the custom selectors.

Selector	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
CSS selectors	1	1	1	0	1	1	1	1	1	1	1	1
XPath selectors	1	1	1	0	1	1	1	1	1	1	1	1
Custom useful selectors	1	1	2	1	1	2	1	1	1	2	2	1
Images	1	0	0	1	0	0	0	0	0	0	1	1
Relative to some elements	1	1	1	1	1	1	0	0	1	1	0	1
Mobile app selectors	1	0	0	0	1	0	0	0	0	1	1	1
Desktop app selectors	1	0	0	0	1	0	0	0	0	0	0	1
Sum	7	4	5	3	6	5	3	3	4	6	6	

■ **Table 4.5** Table with supported selectors

4.7 Web UI

This section contains a rating of the tools for handling some of the main aspects of browser testing and some general testing capabilities. The results can be found in the table 4.6.

Most of the parameters are rated mixed on objective and subjective criteria on their intuitiveness of implementation or presence. Details about grading can be found in the Google sheet.

Capability	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Waits	2	2	2	1	2	3	2	2	2	3	3	1
Events	1	2	2	1	2	2	2	2	1	2	2	1
Alerts	2	2	2	2	2	1	1	1	2	2	2	1
Asserting	2	1	2	1	1	2	1	2	2	2	2	1
Mobile support	2	0	1	0	2	1	1	1	2	2	2	1
Execution speed	3	2	2	1	1	3	3	2	2	2	2	1
iFrame	2	2	1	2	2	2	2	2	2	2	2	1
Shadow root	1	2	2	2	2	2	1	2	1	1	2	1
Multiple tabs	1	1	1	1	1	1	1	0	1	1	1	1
File upload	2	2	1	0	2	2	2	2	2	2	2	1
Syntax	2	2	2	2	2	2	1	1	1	2	2	1
Data driven	2	1	2	1	1	1	1	1	1	2	1	1
Debugging	1	1	1	1	1	1	0	1	0	1	1	2
Sum	24	21	22	16	22	24	18	20	19	25	21	

■ **Table 4.6** Table with Web UI parameters

4.8 Community

In this section, the tools were rated using the following criteria:

Stack Overflow questions Because it is an important place for developers to ask questions. More questions answered or asked means more problems addressed.

GitHub stars and used by statistics Reflect popularity and community.

Downloads Enumerate the popularity of the tool.

Responsible authority Is a criterium to categorize the size of the team managing the tool. It makes a huge difference whether a tool is managed by one man or a team from a company like Microsoft or Google.

Support This parameter rated the tools by the platforms that the responsible authorities to answer questions from other developers using the tools.

Age Is a crucial factor. The older the tool is, the more people got into contact with it. That includes written articles about the tools, tests written with the tools, YouTube or other tutorials published about the tool, etc.

These parameters were sorted into different groups selected by closest intervals of size, age, etc., and given points according to the group they were put in.

The results can be found in the table 4.7.

Statistic	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Stack overflow questions	3200	95000	6000	900	8000	800	6500	1600	1600	13	1500	1
GitHub stars	6000	30000	38000	1700	15000	35000	76000	9300	11100	200	7500	1
GitHub used by	1600	140000	409000	2400	2600	11000	197700	10200	133000	-	40700	1
NPM downloads	-	3000000	4000000	-	-	2000000	3000000	250000	187000	-	1000000	1
Other downloads	-	130000000	-	400000	2000000	-	-	-	-	102000	-	1
Responsible authority	Karate Labs Inc.	Software Freedom Conservancy	Cypress.io	Sikuli Lab	OpenJS Foundation	Microsoft	Google	Developer Express Inc.	BrowserStack	Tricentis	OpenJS Foundation	1
Age	2017	2004	2014	2012	2011	2020	2018	2016	2014	2017	2013	1
Support	Stack overflow, Github issues	Bug tracker, user group, chat rooms (multiple platforms), sponsors, Slack	Gitter, Stack overflow, Github issues	Questions and Bugs on their website	Appium discuss	Microsoft support, Github issues	Github issues, Stack overflow	Github issues, Stack overflow	Github, Discord, Stack overflow	Help articles, Contact us	Slack, Github discussions, Gitter	1
Stack overflow questions in points	2	4	3	1	3	1	3	2	2	1	2	1
GitHub stars in points	1	3	3	1	2	3	4	1	2	1	1	1
GitHub used by in points	1	2	4	1	1	2	3	1	2	1	1	1
Downloads in points	1	3	2	1	2	2	2	1	1	1	2	1
Responsible authority in points	1	3	2	1	3	4	4	2	2	3	3	1
Age in points	1	4	2	3	3	1	1	2	2	1	3	1
Support in points	2	2	2	1	1	2	2	2	2	1	2	1
Sum	9	21	18	9	15	15	19	11	13	9	14	

Table 4.7 Table with community parameters

4.9 Integration

Integration with other tools is a valuable perk for an automation tool. It involves posting a notification about a completed test run to a Slack channel or Microsoft Teams. Automatically create a Jira test execution report after the test run is finished. Running in a Docker container and integrating well with popular CI/CD tools like GitLab and Jenkins.

This is why the tools are rated on how well they integrate with other devices in this section.

The tools were rated for the existence of integration. It was either non-existent (or not found), so it got 0 points, or it existed, but it required usage of third-party libraries to get 1 point, or the creators of the tool officially supported the integration, then it got 2 points.

The important cloud and CI/CD tools like AWS, Azure pipelines, and GitLab got the point per found integration. Other CI tools were counted into a CI category, and the tool got 1 point per supported CI tool.

All tools can be used in testing that can be reported to Jira using its API, but some offer a more advanced integration, so they got an additional point.

All tools are usable with the Cucumber framework.

All tools can be run in a Docker container though SikuliX needs a virtual display configuration to function.

For someone testing, clouds are an essential service to satisfy ones testing needs. Nevertheless, for the default configuration of the table, the weights for these parameters were lowered because it does not go well with the open-source theme.

The results can be seen in the table 4.8.

Tool	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Slack	0	0	2	0	0	1	1	1	1	2	1	1
Microsoft teams	0	0	1	0	0	0	0	0	0	2	1	1
GitHub actions	1	1	2	1	2	2	1	2	2	2	2	2
GitLab	1	2	2	1	2	2	2	2	2	2	2	2
CI (Circle CI, Jenkins, Bitbucket...)	3	6	12	6	6	6	6	6	1	4	2	0.1
Testing clouds	0	8	7	1	15	9	6	3	5	2	6	0.1
Cucumber	1	1	1	1	1	1	1	1	1	1	1	2
Jira	0	1	1	0	1	1	1	1	0	0	1	1
Docker	1	1	1	1	1	1	1	1	1	1	1	3
Mocking	2	1	2	1	1	2	1	2	1	1	2	1
AWS	1	1	1	1	1	1	1	1	0	1	1	1
IDE plugins or extensions	2	2	1	1	1	2	1	2	1	2	2	1
Azure pipelines	1	1	1	0	1	1	1	1	1	1	1	1
Sum	15.3	18.4	23.9	12.7	28.1	22.2	18.2	21.9	17.6	22.6	22.8	

Table 4.8 Table with supported integrations

4.10 Summary

The values found in **Sum** row of tables from previous sections are used as values in the rows corresponding to the names of previous sections. This decision was made to lower the number of rows in the final table and split relative parameters into relevant tables. The final table 4.9 contains all parameters in one table.

In the row labelled **Total** one can see the sum of all points multiplied by their weights. The tool with the most points is TestProject and can be seen as a winner and the best tool in the bunch. However, that is not necessarily the truth.

The table is designed with weights in order for individuals to emphasise what they value or want from their tool more. Better integration, mobile support, multifunctionality? The emphasis can be changed by adjusting the weight. The result might end up entirely differently. Moreover, some of the parameters are purely subjective, so the grades given to them by the author might differ from grades given by someone else. There is also a possibility that the grading system for some of the parameters is not good or the parameter itself is useless. Alternatively, there might be a possibility that there are not enough grades or parameters. This is why an editable version of the Google sheet is available.² Anyone interested can modify it according to their own needs and rate it by the parameters suitable to their needs.

Regarding the results using the normalised version where the max score is 100, it is apparent that tools like Cypress, TestProject and Playwright are presentably not far from each other in the score. All of them are popular tools and could be an excellent choice for the right team.

TestProject is likely the winner because it is free and can work in the cloud or offline. It can be managed from a browser, and the default capabilities of the tool are excellent. Writing cross-browser tests without code only using UI is great because beginners can do so it can cut costs. The add-ons add great functionality in a few clicks. The reports look great and if having it in the cloud is bothersome, running it on other servers is possible while adding personal integrations and configurations.

In other bulk of close results, we can see Karate, Selenium, Cypress, Appium and WebdriverIO. Each could have gotten its most significant share of points from different parameters.

Nightwatch.js and TestCafe seem not on the same level of JS testing frameworks as Playwright and Cypress, as this analysis showed.

Puppeteer proved to be a helpful tool mainly when combined with other tools. SikuliX might come off as a loser here, but it is still an excellent tool for automating UI without a need to know the structure of the application.

The Google sheet made available even has a quality list where one can see tools asses according to some of their qualities. Using a link,³ one can go to the Google Drive folder with editable unsorted material used to create this thesis.

²https://docs.google.com/spreadsheets/d/1C_r5Bwvej2QhFhL00Q4hrdZTy8iNv0dbnnCovvWrzqw/edit#gid=699299515

³https://drive.google.com/drive/folders/1CWG_o-luwJAgwFsJxKpsEDqEXV5z4hDL?usp=sharing

Capability	Karate	Selenium	Cypress	Sikuli and SikuliX	Appium	Playwright	Puppeteer	TestCafe	Nightwatch.js	TestProject	WebdriverIO	Weight
Browser support	6.00	6.00	4.00	6.00	6.00	6.00	4.00	7.00	6.00	6.00	6.00	1.00
Test recording tool	0.00	3.00	1.00	1.00	2.00	3.00	1.00	1.00	1.00	3.00	1.00	2.00
Required technical knowledge	2.00	1.00	2.00	3.00	1.00	2.00	1.00	2.00	1.00	3.00	2.00	2.00
Supported data formats	9.00	5.00	8.00	5.00	5.00	6.00	6.00	6.00	6.00	10.00	6.00	1.50
Reporting	6.00	5.00	5.00	4.00	5.00	6.00	4.00	5.00	4.00	8.00	7.00	2.00
Language support	3.00	7.00	2.00	3.00	7.00	5.00	2.00	2.00	1.00	4.00	1.00	1.00
Parallel testing	2.00	1.00	1.00	0.00	1.00	2.00	1.00	1.00	1.00	2.00	2.00	2.00
Web UI automation	24.00	21.00	22.00	16.00	22.00	24.00	18.00	20.00	19.00	25.00	21.00	1.50
API testing	3.00	1.00	3.00	1.00	1.00	3.00	2.00	1.00	1.00	3.00	1.00	1.00
Performance testing	2.00	0.00	1.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	1.00	1.00
Development platform variability	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	1.00
Environment configuration	2.00	2.00	2.00	1.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	1.00
Set up difficulty	0.00	0.00	1.00	0.00	0.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00
Maintenance	1.00	1.00	2.00	0.00	0.00	2.00	0.00	1.00	1.00	2.00	2.00	1.00
Integration with other useful dev tools	15.30	18.40	23.90	12.70	20.10	22.20	18.20	21.90	17.60	22.60	22.80	1.00
Community/Support	9.00	21.00	18.00	9.00	15.00	15.00	19.00	11.00	13.00	9.00	14.00	1.50
Screenshots and Video recordings of tests	1.00	1.00	2.00	1.00	2.00	3.00	2.00	2.00	2.00	2.00	3.00	1.00
Multifunctionality	2.00	0.00	1.00	0.00	0.00	1.00	1.00	1.00	0.00	1.00	0.00	3.00
Documentation and test examples	3.00	4.00	4.00	2.00	4.00	3.00	2.00	3.00	2.00	3.00	3.00	2.00
Selectors	7.00	4.00	5.00	3.00	7.00	5.00	3.00	3.00	4.00	6.00	6.00	1.50
Total	141.80	143.90	152.40	97.20	140.60	158.20	114.60	128.90	114.60	161.60	143.30	
Normalised	87.75	89.05	94.31	60.15	87.00	97.90	70.92	79.76	70.92	100.00	88.68	

■ **Table 4.9** Complete table with all of the parameters.

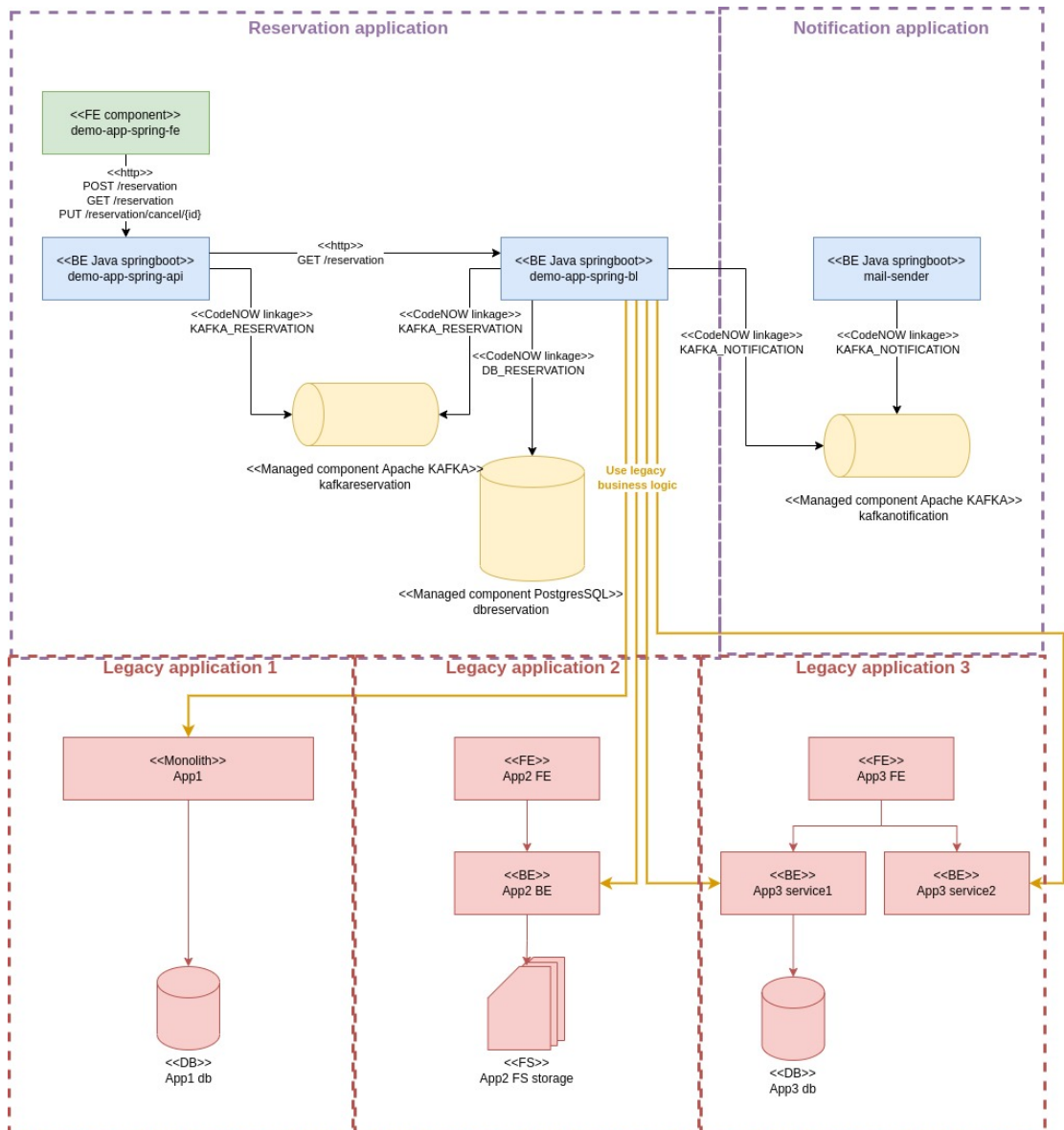
Test implementation

In this chapter, the analysed tools will be put under practical use.

The introduced demo application will be tested using tools analysed in the previous chapters. The implementations will be commented on and compared.

5.1 Introduction of the demo application

The demo application is called Seat Reservation Demo and was provided for testing purposes by the supervisor. In the figure 5.1 one can see an architectural diagram of the application.



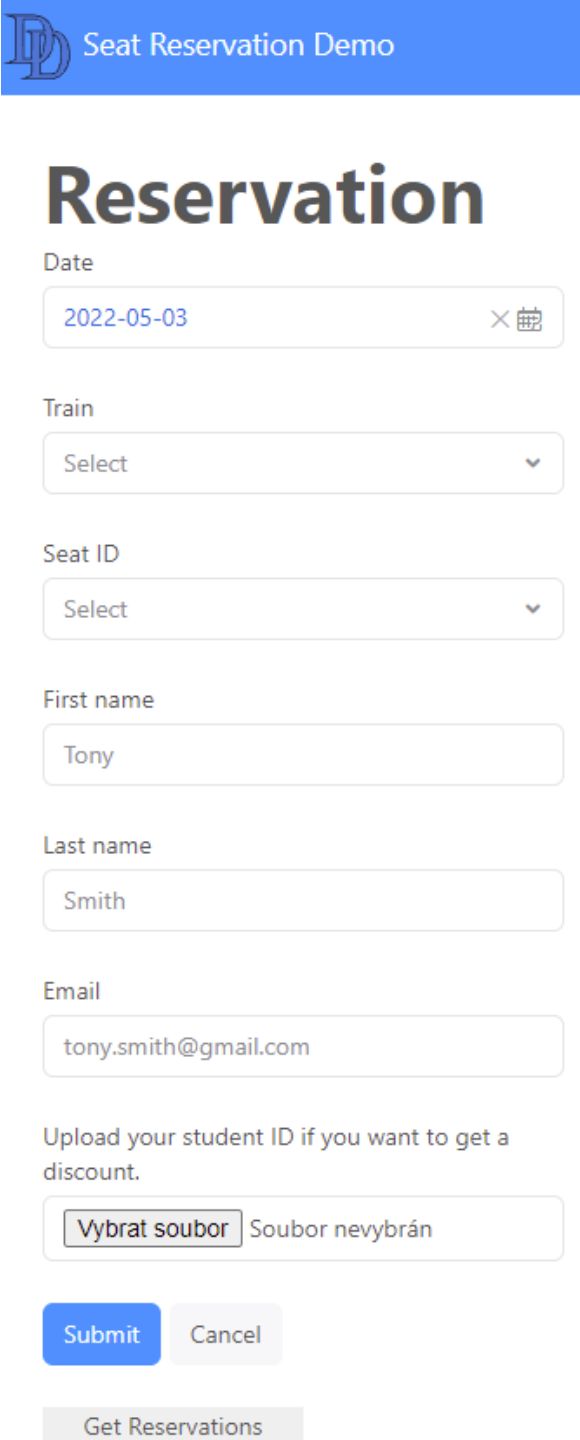
■ **Figure 5.1** Diagram of Seat Reservation Demo [120]

5.1.1 Functionality

All requests from the frontend component are sent to the API component. API component then sends the requests further to the backend component via Kafka or HTTP. The backend component can send a notification request via Kafka to the mail sender application. [120]

The front end was changed from the original to provide a file upload functionality for testing purposes and is hosted on a free *Vercel* website,¹ and can be seen in the figure 5.2. Vercel is a free deployment platform for JS applications. The front end is connected to the backend hosted on the Stratox Cloud Native servers.

¹<https://demo-app-fe.vercel.app/>



The image shows the front end of a 'Seat Reservation Demo' application. At the top, there is a blue header with a logo and the text 'Seat Reservation Demo'. Below this is a large heading 'Reservation'. The form consists of several input fields: a date field with '2022-05-03', two dropdown menus for 'Train' and 'Seat ID', text input fields for 'First name' (Tony) and 'Last name' (Smith), and an email field (tony.smith@gmail.com). There is also a file upload section with a button 'Vybrat soubor' and the text 'Soubor nevybrán'. At the bottom, there are 'Submit' and 'Cancel' buttons, and a 'Get Reservations' button.

Seat Reservation Demo

Reservation

Date

2022-05-03

Train

Select

Seat ID

Select

First name

Tony

Last name

Smith

Email

tony.smith@gmail.com

Upload your student ID if you want to get a discount.

Vybrat soubor Soubor nevybrán

Submit Cancel

Get Reservations

■ **Figure 5.2** Front end of Seat Reservation Demo

5.1.1.1 Create Reservation

After filling and submitting a reservation form in the frontend component, a request is sent to the API component, which sends the request to the backend component via Kafka. The backend component sets the reservation status to *ACCEPTED*, saves the reservation into a database and sends a notification to the mail sender via Kafka. The mail sender sends an email notification. [120]

5.1.1.2 View of Reservations

After clicking on the button *Get Reservations*, a request is sent to the API component, which sends the request to the backend component via HTTP. The backend component finds the last 50 created reservations and sends the reservations back to the API component and API back to the frontend. The front end generates a table with all the reservations received from the backend. When a reservation has the status *ACCEPTED*, a button *Cancel Reservation* is shown in the *STATUS* column. [120]

5.1.1.3 Cancel Reservation

After clicking the *Cancel Reservation button*, a request with an id of the reservation selected to be cancelled is sent to the API component, which sends the request to the backend component via Kafka. The backend component finds the reservation, sets its status to *CANCELED* and sends a notification request to the mail sender via Kafka. The mail sender sends an email notification. [120]

5.2 Test plan

Part of the thesis assignment is to create a test plan and implement it using the analysed tools.

In some tools that offer API testing the following requests will be tested:

POST Request for creating a reservation

GET Request for getting the last 50 created reservations

PUT Request for canceling a reservation

The API tests will send either a valid or invalid request to the backend of the demo application and assert that the response status and message are correct.

The functionalities of the demo application can be tested using the following test plan comprised of three scenarios. The scenarios are described using the Gherkin syntax.

5.2.1 Scenario 1

Scenario 1 is shown in the example 4, and it describes a user visiting the front end of the application, filling in all of the details, and sending the reservation. After which, the user should be notified with an alert acknowledging the success of creating the reservation.

```
Scenario: Create a new reservation using the front end of the demo application.
  Given the user opens a browser
  When the user enters the demo application
  Then the user should see the reservation details
  When the user enters all of the necessary information
  And clicks the Submit button
  Then the user should see an alert with a message:
  Reservation created successfully
  And the user closes the browser
```

■ **Code listing 4** Scenario 1 in the Gherkin syntax

5.2.2 Scenario 2

Scenario 2 is shown in the example 5, and it describes a user visiting the front end of the application after creating the reservation in Scenario 1. The user views the created reservations and cancels the first one shown in the reservations table. After dealing with alerts presented by the browser, the user should see that the status of the reservation has changed.

```
Scenario: Get a list of reservation and cancel the latest reservation
that you created in the Scenario 1.
  Given the user opens a browser
  When the user enters the demo application
  And the user already created a reservation
  When the user clicks the Get Reservations button
  Then the user should see an alert with a message:
  Get reservations view is successful
  When the user clicks the OK button in the alert view
  Then the user should see the table with the latest 50 reservations
  When the user clicks the Cancel Reservation button located
  in the first row of the reservation table
  Then the user should see an alert with a message:
  Reservation canceled successfully.
  When the user clicks the OK button in the alert view
  Then the user should see an alert with a message:
  Get reservations view is successful
  When the user clicks the OK button in the alert view
  Then the status of the first reservation
  in the reservation table should be CANCELED
  And the user closes the browser
```

■ **Code listing 5** Scenario 2 in the Gherkin syntax

5.2.3 Scenario 3

This scenario describes a user visiting the front end of the application, filling in all the details correctly except for the email address by not adding the @ character. The user will not be able to create a reservation after submitting the reservation and will be hit with a temporary message box from the email input informing him that the email is not in the correct format. The scenario can be seen in the example 6.

```
Scenario: Try to create a new reservation but enter the invalid email address.  
  Given the user opens a browser  
  When the user enters the demo application  
  Then the user should see the reservation details  
  When enters the email address without the @ symbol  
  And clicks the Submit button  
  Then the user should see not see an alert with a message:  
  Reservation created successfully  
  And the user closes the browser
```

■ **Code listing 6** Scenario 3 in the Gherkin syntax

5.3 Implementation of the test plan

In this section, the implementations written using the analysed tools are shown, discussed and compared. The code written for the can be found on GitHub² and will be a part of the attached media.

README.md is available in the folders of the implementation, instructing users on how to install the prerequisites and run the tests. To run the tests, Node.js and Java should be installed.

A third party library **Faker**, will be used for Java³ and JS⁴ for generating data that will be used for inputting data to the front end and API requests.

5.3.1 Selenium

In the file located in the folder with code implementation *SeleniumDemo/src/test/java/com/seleniumdemo/DemoAppTest.java* one can see an implementation of the test scenarios using Java and JUnit library with Selenium WebDriver commands.

As shown in the example 7 before each test, objects are initialised, and parameters can be set to change the properties of the web drivers. It is possible to initialise different drivers for different browsers used during the testing. For this implementation, the Chrome driver was chosen. At the end of each test, the driver is shut down.

²<https://github.com/oslaksam/CodenowDemoAppTestExamples>

³<https://github.com/DiUS/java-faker>

⁴<https://github.com/faker-js/fakerprojects>

```
private WebDriver driver;
private Map<String, Object> vars;
private JavascriptExecutor js;
private Faker faker;
private final String URL = "https://demo-app-fe.vercel.app/";

@BeforeEach
public void setUp() {
    System.setProperty("webdriver.chrome.driver",
        "src/test/resources/chromedriver.exe");
    driver = new ChromeDriver();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<String, Object>();
    faker = new Faker();
}

@AfterEach
public void tearDown() {
    driver.quit();
}
```

■ **Code listing 7** Example of Selenium and JUnit configuration of the tests

Most of the code was generated using the Selenium IDE, which records the scenarios. The project in which scenarios are recorded can be found in *SeleniumDemo/DemoSelenium.side* and can be imported into Selenium IDE.

As shown in the example 8, the syntax of the WebDriver API is straightforward, and commands are understandable, handling alerts is intuitive. Selenium IDE generated the comments in the code. All implemented scenarios can be found in the above mentioned Java code file. To run the tests, the project can be loaded into an IDE, favourably IntelliJ IDEA and run from there or can be compiled using Maven (*mvn clean install*). In the created *target* directory, there will be an executable *JAR* that will run the tests.

5.3.1.1 Cucumber

To show how an implementation of the Cucumber framework could look like, it was decided to use the Selenium implementation. The scenarios written in Gherkin syntax can be found in *SeleniumDemo/src/test/resources/features/demoapp.feature*.

The difference between the implementation with and without the Cucumber framework is that in the Cucumber framework, there is a different runner to use. It is specified where the feature file is and where the glue is. This runner is located in *SeleniumDemo/src/test/java/com/cucumberselenium/DemoAppCucumberTest.java*. How the runner looks like in code can be seen in the example 9.

```

@Test
@Order(1)
public void createReservation() {
    // Generated by Selenium IDE
    // Test name: Create a reservation
    // Step # | name | target | value
    // 1 | open | / |
    // Open the website
    driver.get(URL);
    // 2 | setWindowSize | max |
    // Resolution
    driver.manage().window().maximize();
    // 3 | click | css=.rs-picker-toggle-value |
    // Click on date selector
    driver.findElement(By.cssSelector(".rs-picker-toggle-value")).click();
    ...
    // 10 | type | name=firstName | Sony
    // Type in the name
    driver.findElement(By.name("firstName")).sendKeys(faker.name().firstName());
    ...
    // 15 | type | name=file | C:\fakepath\file.random
    // Type in the path to the selected file
    driver.findElement(By.name("file")).sendKeys("C:\\file.random");
    // 16 | click | xpath=//button[contains(.,'Submit')] |
    // Click on the submit button
    driver.findElement(By.xpath("//button[contains(.,'Submit')]")).click();
    // 17 | assertAlert | Reservation created successfully |
    // Wait for an alert and assert that the reservation was created successfully
    Alert alert = new WebDriverWait(driver, Duration.ofSeconds(3))
        .until(driver -> driver.switchTo().alert());
    assertEquals(alert.getText(), "Reservation created successfully");
    alert.accept();
}

```

■ **Code listing 8** Example showing part of the Selenium implementation of Scenario 1

```

@RunWith(Cucumber.class)
@CucumberOptions(
    features = {"classpath:features/demoapp.feature"},
    glue = {"com.cucumberselenium"},
    plugin = {"pretty", "html:target/cucumber-reports"})
public class DemoAppCucumberTest {
}

```

■ **Code listing 9** Example of Java Cucumber test runner

The glue is the code used in implementation, but it is redistributed into methods that correspond to steps in the feature file. The glue can be found in *SeleniumDemo/src/test/java/com/cucumberselenium/ReservationSteps.java*.

As shown in the example 10, Cucumber makes it a bit more complicated to write tests as the mapping required for the scenario steps. It is up to the team that decides to use the Cucumber framework if it seems beneficial to them or not. It is possible to generate the functions mapped

to feature file steps using an IDE with some Cucumber plugins, but the rest of the code must be written by someone.

```
@And("clicks the Submit button")
public void clicksTheSubmitButton() {
    // 16 | click | xpath=//button[contains(.,'Submit')] |
    // Click on the submit button
    driver.findElement(By.xpath("//button[contains(.,'Submit')]")).click();
}

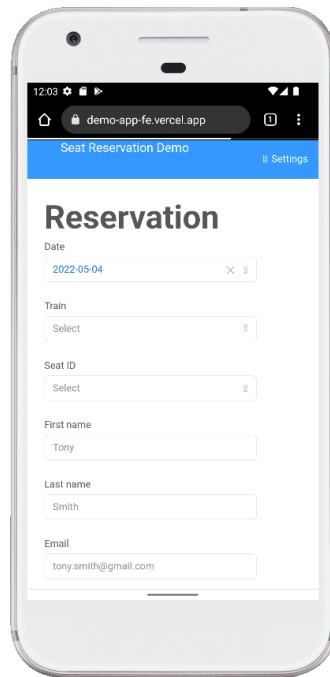
@Then("the user should see an alert with a message: Reservation created successfully")
public void theUserShouldSeeAnAlertWithAMessageReservationCreatedSuccessfully() {
    // 17 | assertAlert | Reservation created successfully |
    // Wait for an alert and assert that the reservation was created successfully
    alert = new WebDriverWait(driver, Duration.ofSeconds(3))
        .until(driver -> driver.switchTo().alert());
    assertEquals(alert.getText(), "Reservation created successfully");
}
```

■ **Code listing 10** Example of Java Cucumber steps

5.3.2 Appium

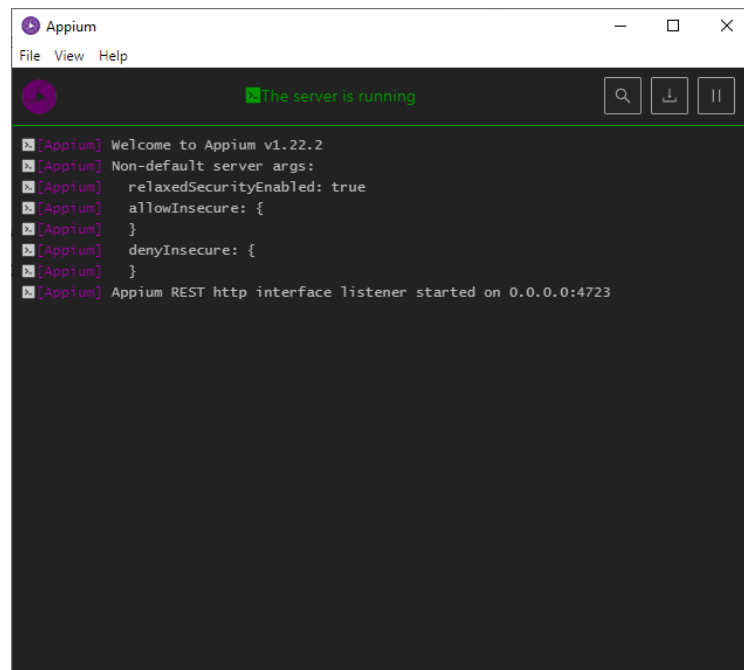
The implementation for Appium was almost the same as the implementation for Selenium, as it used the WebDriver API. The significant difference was in the configuration of the device. For this example, an emulated Android Device was used.

The Android Studio needed to be installed for its Virtual device management capabilities. The virtual device had to be selected configured in terms of virtual size and the Android version and emulator API. A *Google Pixel* phone with android version code name *Tiramisu* was used for this implementation. The emulated phone can be seen in the figure 5.3.



■ **Figure 5.3** Image showing an emulated Android device

To get the tests running, Appium Server needs to be configured and run. For this implementation, Appium Server GUI for Windows was chosen, and an example of a running server can be seen in the figure 5.4.



■ **Figure 5.4** Image showing a running Appium Server

The difference in code can be seen in the example 11. The difference is in the driver configuration. Instead of using a Selenium WebDriver, the Appiums AndroidDriver is used. The pathway to the Chrome driver is set in the capabilities, and the Chrome driver will be used to execute the tests in the Chrome installed on the emulated device.

```
private AndroidDriver driver;

@BeforeEach
public void before() throws MalformedURLException {
    DesiredCapabilities caps = new DesiredCapabilities();
    //caps.setCapability("VERSION", "10.0");
    caps.setCapability("deviceName", "emulator-5554");
    caps.setCapability("platformName", "Android");
    caps.setCapability("browserName", "chrome");
    //Replace withou your path to chromedriver.exe
    caps.setCapability("chromedriverExecutable", "C:\\chromedriver.exe");
    driver = new AndroidDriver(new URL("http://0.0.0.0:4723/wd/hub"), caps);
}
```

■ **Code listing 11** Example of the Appium test configuration in JUnit

5.3.3 Karate

It was previously stated that the syntax of Karate is based on the Gherkin syntax from Cucumber. There are IDE extensions available that can help with syntax highlighting, and there is also an excellent Karate extension for Visual Studio Code that allows code generation.⁵

For this example, the OpenAPI specification of the demo application was used and converted to a YAML file available in *KarateDemo/src/test/java/demoapp/openapi.yaml* the generated tests are working well if the specification is well defined. The example data in OpenAPI is often used as testing data. The generated test and data can be seen in the folder *KarateDemo/src/test/java/demoapp*. Test data is stored in the *test-data* folder and the generated tests are in the *cancelReservationUsingPUT.feature*, *createReservationUsingPOST.feature* and *getViewOfReservationsUsingGET.feature* files. The generated code example can be seen in the example 12.

⁵<https://marketplace.visualstudio.com/items?itemName=KarateIDE.karate-ide>

```

@operationId=createReservationUsingPOST
Scenario Outline: Test createReservationUsingPOST for <status> status code

* def args = read(<testDataFile>)
* def result = call read('createReservationUsingPOST.feature@operation') args
* match result.responseStatus == <status>
Examples:
| status | testDataFile |
| 200    | 'test-data/createReservationUsingPOST_200.yml' |
| 200    | 'test-data/createReservationUsingPOST_201.yml' |
| 401    | 'test-data/createReservationUsingPOST_401.yml' |
| 403    | 'test-data/createReservationUsingPOST_403.yml' |
| 404    | 'test-data/createReservationUsingPOST_404.yml' |

@operationId=createReservationUsingPOST
Scenario: explore createReservationUsingPOST inline
You can use this test to quickly explore your API.
* def payload =
  """
  {
    "statusCode": 200,
    "headers": {},
    "params": {},
    "body": {
      "date": "2019-08-24T14:15:22Z",
      "email": "string",
      "firstName": "string",
      "lastName": "string",
      "seatId": "string",
      "trainId": "string"
    },
    "matchResponse": true
  }
  """
* call read('createReservationUsingPOST.feature@operation') payload

```

■ **Code listing 12** Example of test generated using the Karate extension for VSC

Tests are written in *KarateDemo/src/test/java/demoapp/reservation.feature* and the environment is configured in *KarateDemo/src/test/java/demoapp/karate-config.js*. The environment configuration can be made easily by editing the config files and adding different conditions and variables to tests using the JS logic of the file.

The integration with Cucumber is generally pretty nice. The tests are written in feature files, and there is an option to call other feature files within feature files. That gives users the ability to reuse code.

For example, the user has to log into an application to do certain things. The logic for logging in can be written in one feature file and called whenever presented with a scenario where the user needs to be logged in to do other things.

Scenario 1 is implemented in the example 13. One can notice that a Java library *Faker* is loaded into Karate, and a new object is created for generating fake data that are passed into variables *fName*, *lName* and *mailId*. This feature is a part of interoperability with Java. In

Karate UI testing, it is possible to combine the Gherkin keywords to one's liking. The code is concise and simple, and helpful locators like `{ }Select` are useful. Managing alerts and asserting is also uncomplicated and convenient.

Feature: Demo reservation app reservation tests

Background:

```
# this section is optional !
# steps here are executed before each Scenario in this file
# variables defined here will be 'global' to all scenarios
# and will be reinitialised before every scenario
* def Faker = Java.type('com.github.javafaker.Faker') #loading a Java library
* def fakerObj = new Faker()
* def fName = fakerObj.name().firstName()
* def lName = fakerObj.name().lastName()
* def mailId = fName+'.'+lName+'@test.com'
* def feUrl = 'https://demo-app-fe.vercel.app/'
```

Scenario: Use the front end to create a new reservation

```
# replace the executable with your own
* configure driver = { type: 'chrome', showDriverLog: true,
executable: 'C:\\chrome.exe' }
Given driver feUrl
And driver.maximize()
When waitFor(".rs-picker-toggle-value")
Then click(".rs-picker-toggle-value")
When waitFor("{span}Today")
Then click("{span}Today")
When waitFor("{}Select")
Then click("{}Select")
When waitFor("{}Sp 9301")
Then click("{}Sp 9301")
When waitFor("{}Select")
Then click("{}Select")
When waitFor("//a[@name='seatId']")
Then click("//a[@name='seatId']")
When waitFor("{}Coach 1 / Seat 21")
Then click("{}Coach 1 / Seat 21")
And input('input[name=firstName]', fName )
And input('input[name=lastName]', lName)
And input('input[name=email]', mailId)
# replace the file with your own file
And driver.inputFile('input[name=file]', 'C:\\random.file')
When click("//button[contains(.,'Submit')]")
And delay(1200)
Then match driver.dialogText == 'Reservation created successfully'
And dialog(true)
```

■ **Code listing 13** Example of the Scenario 1 implemented in Karate

The first main functionality of Karate was API testing and it can be done in a few lines of code, as shown in the example 14 or in the example 2. Karate offers advanced asserting abilities for requests and responses. Karate offers advanced asserting abilities for requests and responses. Using the interoperability with Java, it is even possible to pass a variable to a JSON as shown in the example 14.

```
Scenario: Send an API request for a new reservation
* def req =
  """
  {
    "date": "2022-03-21T02:24:59.815Z",
    "email": "#(mailId)",
    "firstName": "#(fName)",
    "lastName": "#(lName)",
    "seatId": "1-21",
    "trainId": "ICE-575"
  }
  """

Given url apiUrlPost
And request req
When method post
Then status 200
```

■ **Code listing 14** Example of an API test for POST request in Karate

5.3.4 SikuliX

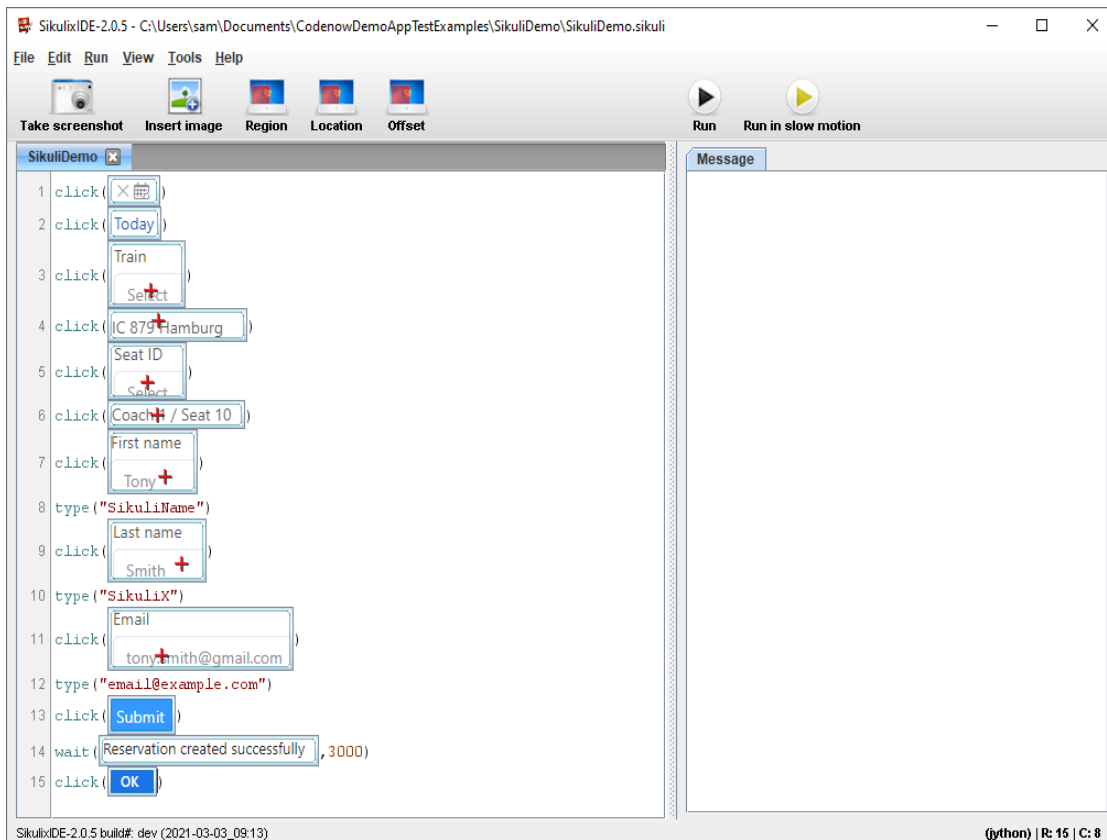
SikuliX tests can be implemented using SikuliX IDE, which helps writing simple Jython scripts for SikuliX, or it is often used as a library for Java.

The *SikuliDemo/SikuliDemo.sikuli* folder can be opened as a project in SikuliXIDE, and it is an implementation of the Scenario 1. Other folders that end with *.sikuli* in the *SikuliDemo* folder are implementations of the remaining scenarios. The scripts were written for the Google Chrome browser.

The SikuliX IDE offers the ability to insert images and configure the region, location and offset to describe where exactly should SikuliX click on an image that matches with a predefined image in the script when executing the scripts. The IDE also offers the ability to take screenshots. When the picture is inserted, it can be used as an object that can be passed to the script functions.

SikuliX can run the scripts and export them as runnable *JAR* files. When running the scripts, it is required to have an environment ready or fail because it will not be able to find the required GUI elements. It is possible to configure a virtual desktop for the cloud and other environments.

Implementation of Scenario 1 is shown in the figure 5.5. Before running the script, it is required to open the front end of the demo application in the browser.

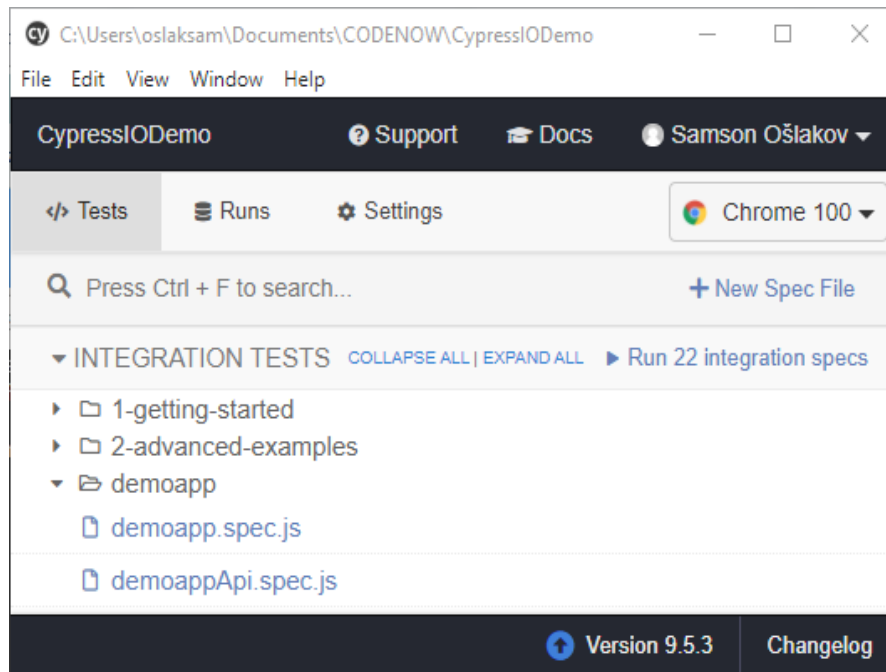


■ **Figure 5.5** Image showing Scenario 1 implemented using SikuliX

The better way to use SikuliX is using it in a Java program combined with other tools like JUnit, Selenium or Appium to automate the GUI using image recognition.

5.3.5 Cypress

Implementing tests in Cypress is remarkably developer-friendly. Upon opening Cypress for the first time, numerous examples can be seen to navigate through the functions of Cypress. There are also numerous extensions for Cypress available for IDEs. In the figure 5.6 one can see the main menu after running Cypress. The tests can be found in the *CypressIODemo/cypress/integration/demoapp* folder.



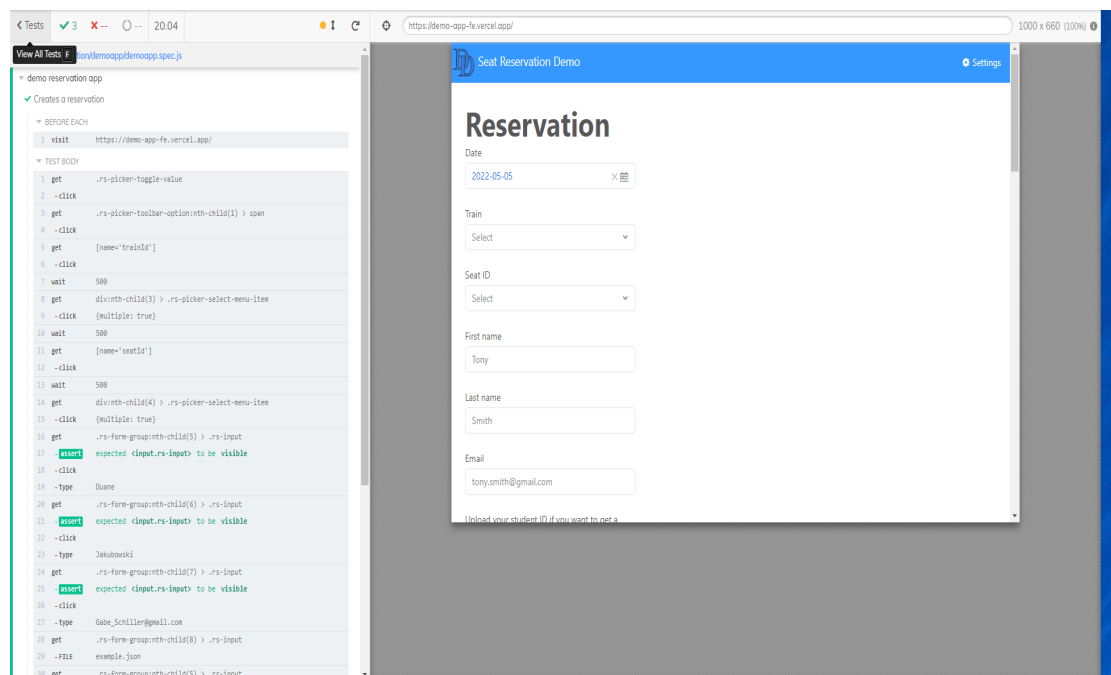
■ **Figure 5.6** Image showing the menu of Cypress

A test recording tool is available in the premium version of Cypress, but free test recording extensions are available, though they are not as sophisticated as the premium tool. For this implementation, a free extension called *Cypress Recorder*⁶ was used.

When creating or opening a spec file, Cypress launches a selected browser and starts executing the commands in the test file. It works in developer mode. When the test code is changed, Cypress starts executing the code in the spec file. This feature can swiftly notify the developer whether his code works properly or breaks somewhere.

After all of the commands in a test are executed, it is marked as passed. If an error happens along the way, the test is marked as failed. The developer can see the cause of the error in the browser. The test steps are displayed on the left side of the window, and it is possible to go step by step before and after every action because Cypress can snapshot the DOM of the website, making debugging easy and by default. In the step where the test fails, a detailed description can be found explaining the error. Image in the figure 5.7 shows an example of the development mode in Cypress.

⁶<https://chrome.google.com/webstore/detail/cypress-recorder/glcapdcacdfkokcmicllhcjigeodacab>



■ **Figure 5.7** Image showing the development mode in Cypress

Cypress can be seamlessly integrated with CI tools. An account can be created on the company website⁷ allowing users to connect their projects to a dashboard with details about the test execution and other reporting capabilities. The dashboard also offers parallelization and load balancing, but the free version has only three users and 500 test results per project.

The menu of Cypress after running it In the folder, *CypressIODemo/cypress/support* one can find a global configuration file for tests in *index.js* and *commands.js* for storing custom utility functions. Users write functions to be reused across different tests by importing them into the code file.

File uploading and downloading is not a functionality built-in by default, but they are supported using plugins.

Mocha and Chai keywords can be seen in the example 15, they are used in the structure and assertions of the test. Flow of the implementation is straightforward and understandable though it requires basic knowledge of JS. The way the alerts are handled is acceptable. The auto wait function is not ideal, so it had to be supported using the *cy.wait(500)* function, so the DOM has time to change its state before the following command is executed. Without the wait, the test often failed.

⁷<https://dashboard.cypress.io/signup>


```

describe("demo reservation app", () => {
  beforeEach(() => {
    cy.visit("https://demo-app-fe.vercel.app/");
  });
  it("Creates a reservation", () => {
    const name = faker.name.firstName();
    ...
    cy.get(".rs-picker-toggle-value").click();
    cy.get(".rs-picker-toolbar-option:nth-child(1) > span").click();
    cy.get("[name='trainId']").click();
    let val = Math.floor(Math.random() * (4 - 1 + 1) + 1);
    cy.wait(500);
    cy.get("div:nth-child(" + val + ") > .rs-picker-select-menu-item").click({
      multiple: true,
    });
    ...
    cy.get(".rs-form-group:nth-child(5) > .rs-input")
      .should("be.visible")
      .click()
      .type(name);
    ...
    const filepath = "example.json";
    cy.get(".rs-form-group:nth-child(8) > .rs-input").attachFile(filepath);
    ...
    cy.get(".rs-btn-primary").should("be.visible").click();
    cy.wait(1001);
    cy.on("window:confirm", (text) => {
      expect(text).to.contains("Reservation created successfully");
    });
  });
});

```

■ **Code listing 15** Example showing part of the implementation of Scenario 1 in Cypress

It is also possible to write API tests in Cypress. Writing them and asserting them is not complicated, as can be seen in the example 16. The variables created by the *Faker* library can be set to a body of the request, which is convenient.

```
context("POST /reservation", () => {
it("Should create a reservation", () => {
  cy.request({
    method: "POST",
    url: "https://demo-app-spring-api-aca-demo.stxcn-aca.stxcn.codenow.com/reservation",
    body: {
      date: "2022-03-21T02:24:59.815Z",
      email: faker.internet.email(),
      firstName: faker.name.firstName(),
      lastName: faker.name.lastName(),
      seatId: "1-21",
      trainId: "ICE-575",
    },
  }).should((response) => {
    expect(response.status).to.eq(200);
    expect(response.body).to.eq("Reservation accepted.");
    cy.log(JSON.stringify(response.body));
  });
});
```

■ **Code listing 16** Example showing the test for a POST request to the demo app in Cypress

5.3.6 Puppeteer

Implementing tests using Puppeteer was an unpleasant experience. The tests are located in the *PuppeteerDemo* folder.

Puppeteer is a Node.js library which provides an API to control Chrome over the DevTools Protocol. The information shows that Puppeteer is not a testing framework, so it should be integrated with a testing framework like Mocha to use it properly for testing purposes.

For this implementation, it was decided that the test will be marked as passed when running the JS test file with Puppeteer code will not throw an error, which means that all of the commands in the script will be executed successfully. Alternatively, when running the file returns an error, the test will be classified as a failure. The *expect* function from the *Chai* asserting library was imported to run asserts.

Most of the code was recorded using an extension from Google Webstore called Headless recorder.⁸

Part of the implementation of Scenario 1 can be seen in the example 17. In default settings, the scripts are run in headless mode. The headless mode was turned off to ensure that the commands were executed.

CSS selectors are handled intuitively in Puppeteer but selecting and interacting with elements using XPath is unintuitive. File uploading and input typing are part of the API as they should. Alert handling is handled as an event as it can be problematic to have one listener for multiple alerts.

⁸<https://chrome.google.com/webstore/detail/headless-recorder/djeegiggleadkkbgopoonhjimgghda>

```

(async () => {
  const browser = await puppeteer.launch({
    headless: false,
  });
  const page = await browser.newPage();
  await page.goto(url);
  await page.setViewport({ width: 1920, height: 1080 });
  await page.waitForSelector(
    ".rs-form > .rs-form-group > .rs-picker-date > .rs-btn > .rs-picker-toggle-value"
  );
  await page.click(
    ".rs-form > .rs-form-group > .rs-picker-date > .rs-btn > .rs-picker-toggle-value"
  );
  ...
  await page.waitForXPath("//a[@name='seatId']");
  let elHandle = await page.$x("//a[@name='seatId']");
  await elHandle[0].click();
  ...
  await page.click(
    ".rs-container > .rs-content > .rs-form > .rs-form-group:nth-child(5) > .rs-input"
  );
  await page.focus(
    ".rs-container > .rs-content > .rs-form > .rs-form-group:nth-child(5) > .rs-input"
  );
  await page.keyboard.type(faker.name.firstName());
  ...
  const elementHandle = await page.$(
    ".rs-container > .rs-content > .rs-form > .rs-form-group:nth-child(8) > .rs-input"
  );
  await elementHandle.uploadFile(
    "C:\\random.file"
  );
  page.on("dialog", async (dialog) => {
    console.log(dialog.message());
    await expect(dialog.message()).to.equal("Reservation created successfully");
    await dialog.accept();
  });
  await page.click(
    ".rs-content > .rs-form > .rs-form-group > .rs-btn-toolbar > .rs-btn-primary"
  );
  await browser.close();
})();

```

■ **Code listing 17** Example showing part of the implementation of Scenario 1 using Puppeteer

API testing is also possible using Puppeteer though it was probably not designed. The way it works is when the command `page.goto(url)` is executed, an *HTTP GET* request is sent to a target server but because the `page.setRequestInterception(true)` command was executed, and an event listener for requests was added. The *GET* request was intercepted and handled by the event listener, which can be modified to be another type of request. As shown in the example 18, the *GET* request was changed into a *POST* request with a JSON body added to it. The *POST* request is then sent to the server. The response from the server can then be further handled.

```
(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  const POST_JSON = {
    date: "2022-03-21T02:24:59.815Z",
    email: faker.internet.email(),
    firstName: faker.name.firstName(),
    lastName: faker.name.lastName(),
    seatId: "1-21",
    trainId: "ICE-575",
  };
  await page.setRequestInterception(true);
  page.on("request", (request) => {
    let data = {
      method: "POST",
      postData: JSON.stringify(POST_JSON),
    };
    // Request modified... finish sending!
    request.continue(data);
  });
  const res = await page.goto(TARGET_URL);
  const content = await res.text();
  console.log(content);
  await expect(content).to.include("Reservation accepted.");
  await browser.close();
})();
```

■ **Code listing 18** Example of an API test for POST request in Puppeteer

5.3.7 Playwright

Playwright is a testing framework created mainly by the same people who worked on Puppeteer, so there is a significant possibility that similar concepts occur in Playwright as it utilises Puppeteer. Being a testing framework, Playwright has a configuration file where it is possible to configure devices, drivers, outputs, reports, timeouts, CI, workers and other test-related parameters.

For this implementation, the configuration file is located in *PlaywrightDemo/playwright.config.js*, and the tests are located in this folder *PlaywrightDemo/tests*.

Setting up Playwright was the least problematic task in comparison with other *js* tools. It performs well and runs tests in parallel out of the box.

The implementation of the scenarios was generated using the test generator functionality. It is even possible to preserve an authenticated state while recording tests using cookies. There is even geolocation, language and timezone emulation available. Test generator can generate code in multiple programming languages supported by the Playwright API. However, the tool is not as advanced as Selenium IDE because it allows choosing between multiple selectors and offers numerous additional features.

It offers good extensions for VSC for executing and debugging.

By default, reports are generated onto an HTML page. The reports are detailed and contain screenshots of the application before and after each step in the test to make debugging easier.

The code shown in the example 19, shows the similarity between the Puppeteer and Playwright because they are both run in “async” mode, and both use the page object. Playwright uses some useful custom selectors like *placeholder* and *role*. Uploading files is done very unintuitively, using events. Alert handling is done in the same spirit as Puppeteer and Cypress, using

event listeners and handlers.

```
test.beforeEach(async ({ page }) => {
  await page.goto("https://demo-app-fe.vercel.app/");
  await expect(page).toHaveTitle(/React App/);
});
test("Create a reservation", async ({ page }) => {
  ...
  // Click a[role="combobox"]:has-text("YYYY-MM-DD")
  await page.locator('a[role="combobox"]:has-text("YYYY-MM-DD)').click();
  // Click a[role="button"]:has-text("Today")
  await page.locator('a[role="button"]:has-text("Today)').click();
  ...
  // Click [placeholder="Tony"]
  await page.locator('[placeholder="Tony"]').click();
  // Fill [placeholder="Tony"]
  await page.locator('[placeholder="Tony"]').fill(firstName);
  ...
  const [fileChooser] = await Promise.all([
    // It is important to call waitForEvent before click to set up waiting.
    page.waitForEvent("filechooser"),
    // Opens the file chooser.
    page
      .locator(
        ".rs-container > .rs-content > .rs-form > .rs-form-group:nth-child(8) > .rs-input"
      )
      .click(),
  ]);
  await fileChooser.setFiles(
    "C:\\random.file"
  );
  page.once("dialog", (dialog) => {
    console.log(`Dialog message: ${dialog.message()}`);
    expect(dialog.message()).toBe("Reservation created successfully");
    dialog.accept().catch(() => {});
  });
  // Click text=Submit
  await page.locator("text=Submit").click();
  await page.close({ runBeforeUnload: true });
});
```

■ **Code listing 19** Example showing part of the implementation of Scenario 1 in Playwright

Playwright can also be used for API testing. In the example 20, a test for *POST* request is shown. In the first part, a correct body is sent, and the response is asserted to have an OK status. Then a request with an empty body is sent, following the assertion that the result was not OK. The implementation looks similar to Cypress.

```
test("Create a new reservation", async ({ request }) => {
  const newIssue = await request.post(`/reservation`, {
    data: {
      date: "2022-03-21T02:24:59.815Z",
      email: faker.internet.email(),
      firstName: faker.name.firstName(),
      lastName: faker.name.lastName(),
      seatId: "1-21",
      trainId: "ICE-575",
    },
  });
  expect(newIssue.ok()).toBeTruthy();
  expect(await newIssue.text()).toEqual("Reservation accepted.");
  const badIssue = await request.post(`/reservation`, {
    data: {},
  });
  expect(badIssue.ok()).not.toBeTruthy();
});
```

■ **Code listing 20** Example of an API test for POST request in Playwright

5.3.8 TestCafe

TestCafe is a JS E2E testing framework. TestCafe claims that it has a concise syntax without cumbersome boilerplate expressions and manual timeouts. TestCafe does not require WebDriver and uses browsers installed on the device.

The premium version offers a test recording tool and other useful features, but the license is expensive.

In the example 21, part of the implementation of Scenario 1 can be found. Tests implementing the scenarios are located in *TestCafeDemo/demoapp.js*. TestCafe was disappointing because to use XPath selectors, a custom function found on GitHub of TestCafe had to be imported into the test for XPath selectors to work. The same goes for other valuable functionalities that could have been integrated into TestCafe but instead are available as JS files on GitHub.

Handling alerts is done by configuring a native dialogue handler. The dialogue handler should return *true* when the alert is OK. Otherwise, it should return *false*. The designed solution is not ideal, but it works. The file upload method is designed well.

Some selectors have practical methods like *withAttribute()*.

TestCafe uses a CLI reporter by default.

TestCafe allows API mocking, but it is not a suitable tool for API testing.

```

fixture`Demo app test`.page`https://demo-app-fe.vercel.app/`;
test("Reservation create test", async (t) => {
  ...
  await t.setNativeDialogHandler((type, text, url) => {
    if (text === "Reservation created successfully") return true;
    return false;
  });
  ...
  const select = await XPathSelector("//a[.='Select']");
  await t
    .click(select)
    .click("div:nth-child(" + randomValue + ") > .rs-picker-select-menu-item");
  ...
  await t
    .click(Selector("input").withAttribute("name", "firstName"))
    .typeText(Selector("input").withAttribute("name", "firstName"), firstName)
    .click(Selector("input").withAttribute("name", "lastName"))
    .typeText(Selector("input").withAttribute("name", "lastName"), lastName)
    .click(Selector("input").withAttribute("name", "email"))
    .typeText(Selector("input").withAttribute("name", "email"), randomEmail)
    .setFilesToUpload(Selector("input").withAttribute("name", "file"), [
      "./random.file",
    ]);
  const submit = await XPathSelector("//button[contains(.,'Submit')]");
  await t.click(submit);
  await t.wait(1001);
});

```

■ **Code listing 21** Example showing part of the implementation of Scenario 1 in TestCafe

5.3.9 Nightwatch.js

Implementations of the scenarios in Nightwish.js are stored in the *NightwatchJsDemo/tests* folder. The configuration file is located in *NightwatchJsDemo/nightwatch.conf.js*. It can configure test sources, reporting, drivers, devices, connection to BrowserStack, connection to Selenium Server, plugins, custom commands, environment configuration, etc.

A test recording extension from GitHub⁹ was used to generate some of the code in the script. Due to poorly generated selectors, a large part of the code needed to be rewritten.

However, the tool is not as advanced as Selenium IDE because it allows choosing between multiple selectors and offers numerous additional features.

In the example 22, part of the implementation of Scenario 1 is shown. TestCafe and Nightwatch.js have similar chaining properties. It is possible to write a test only using methods invoked on the isolated occurrence of the driving object in the test code.

Nightwatch.js supports XPath and CSS selectors but it is required to call methods *useXPath()* and *useCss()* to switch between them which is burdensome. Nevertheless, it is possible to use other selectors provided by Nightwatch.js.

At first it was tricky to understand what parameter to pass into the *waitForElementVisible()* and *setValue()* methods instead of XPath and CSS selectors but after reading the documentation it was no longer a problem.

File uploading and alert handling work are similar to Selenium, which is intuitive. It is also

⁹<https://github.com/vvscodex/js--nightwatch-recorder>

convenient that the `await` keyword is not used, unlike in Playwright and Puppeteer. Nightwatch.js uses a CLI reporter by default.

```
const DEFAULT_TIMEOUT = 800;
module.exports = {
  "Create a reservation": function (browser) {
    browser
      .url("https://demo-app-fe.vercel.app/")
      .waitForElementVisible(".rs-picker-toggle-value", DEFAULT_TIMEOUT)
      .click(".rs-picker-toggle-value")
      ...
    let randomValue = Math.floor(Math.random() * (4 - 1 + 1) + 1);
    browser
      .waitForElementVisible(
        "div:nth-child(" + randomValue + ") > .rs-picker-select-menu-item",
        DEFAULT_TIMEOUT
      )
      .click("div:nth-child(" + randomValue + ") > .rs-picker-select-menu-item")
      .pause(300)
      .useXpath()
      .waitForElementVisible("//a[.='Select']", DEFAULT_TIMEOUT)
      .click("//a[.='Select']");
    ...
    browser
      .useCss()
      ...
      .waitForElementVisible("form input[name='firstName']", DEFAULT_TIMEOUT)
      .click("form input[name='firstName']")
      .waitForElementVisible("input[name='firstName']", DEFAULT_TIMEOUT)
      .setValue("input[name='firstName']", faker.name.firstName())
      ...
      .waitForElementVisible("form input[name='file']", DEFAULT_TIMEOUT)
      .setValue("form input[name='file']", __dirname + "\\README.md")
      .waitForElementVisible("form .rs-btn.rs-btn-primary", DEFAULT_TIMEOUT)
      .click("form .rs-btn.rs-btn-primary")
      .pause(900)
      .getAlertText((results) => {
        console.log(results.value);
        browser.assert.equal(results.value, "Reservation created successfully");
      })
      .acceptAlert();
  },
};
```

■ **Code listing 22** Example showing part of the implementation of Scenario 1 in Nightwatch.js

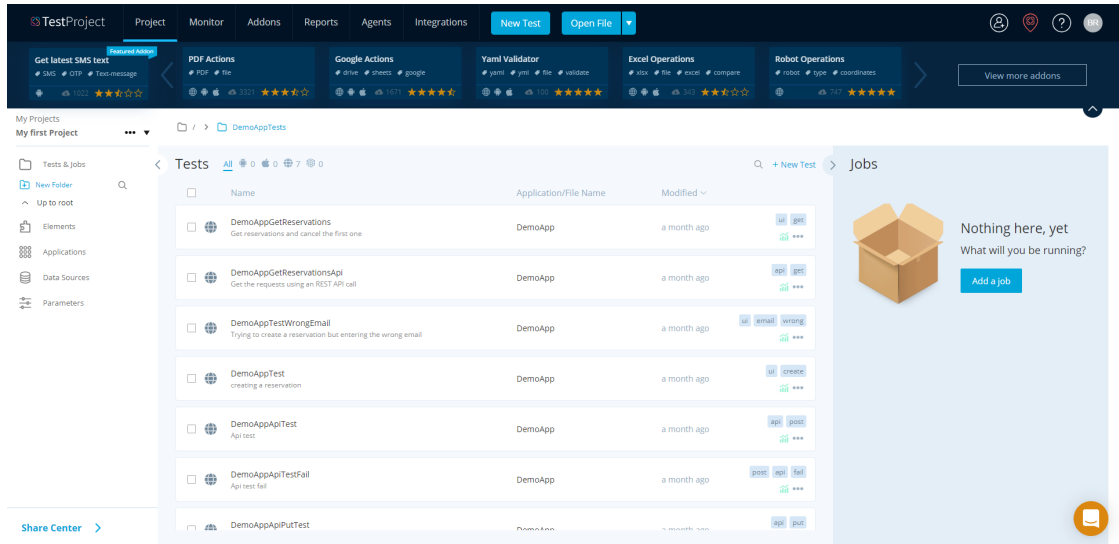
5.3.10 TestProject

Implementing the scenarios in TestProject was a great experience. Tests can be developed and run from a browser. An account is needed to use TestProject.

TestProject Agent needs to be installed on the device to execute the tests. After installing the agent, it needs to be linked with the project. The agent is responsible for test execution and

communication with TestProject. The agent uses browsers installed on the device, so when a user decides to run the tests, he is presented with a list of browsers available for running the tests on. The agent can run on Windows, macOS and Linux or as a Docker container.

In the figure 5.8, a menu of the project for testing the demo application is shown. UI elements on the left side of the menu allow users to switch between projects, create folders, and move tests between the folders. Next, they allow users to add a new application that they want to test. Adding data sources is possible if external data is needed for the tests. Parameters can be configured for a project, and for individual tests, they essentially work as variables.



■ Figure 5.8 Image showing the menu of TestProject

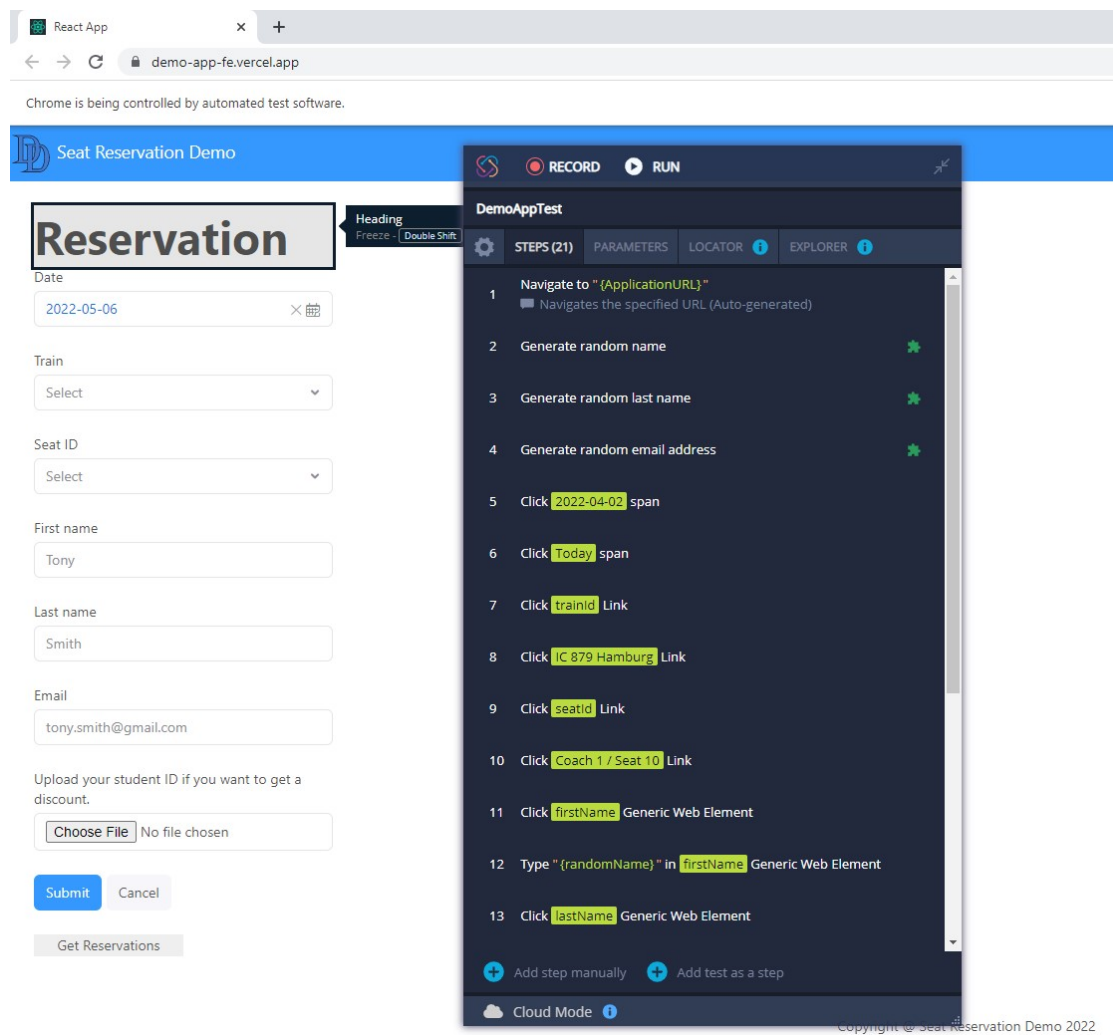
When creating a project, a user is presented with three options: mobile, web and code. Mobile tests use Appium and AI. Selenium and AI are used in web tests.

These tests can be developed in the browser by combining the regular functionality of TestProject and add-ons. Instead of coding, a developer defines the test by searching and applying commands step by step using the TestProject platform on his browser. The commands can be further configured.

If the default commands are insufficient, many add-ons can be added using a click. The add-ons often solve common problems or do practical operations that simplify a tester’s life.

Code tests give the user the ability to use TestProjects SDK to develop tests in their way. Creating useful add-ons for TestProject is very welcomed by the community.

TestProject offers a fantastic tool for recording tests. In the figure 5.9 is a screenshot of the demo application launched in recording mode by the agent. The test editing tool is also on the screenshot. After the user interacts with the app, a command is added to the test script. Recording can be stopped to configure the commands in the test script and can be resumed. While configuring the commands, selectors of web elements can be chosen from the list similar to Selenium IDE. However, the tool used by TestProject is more sophisticated.

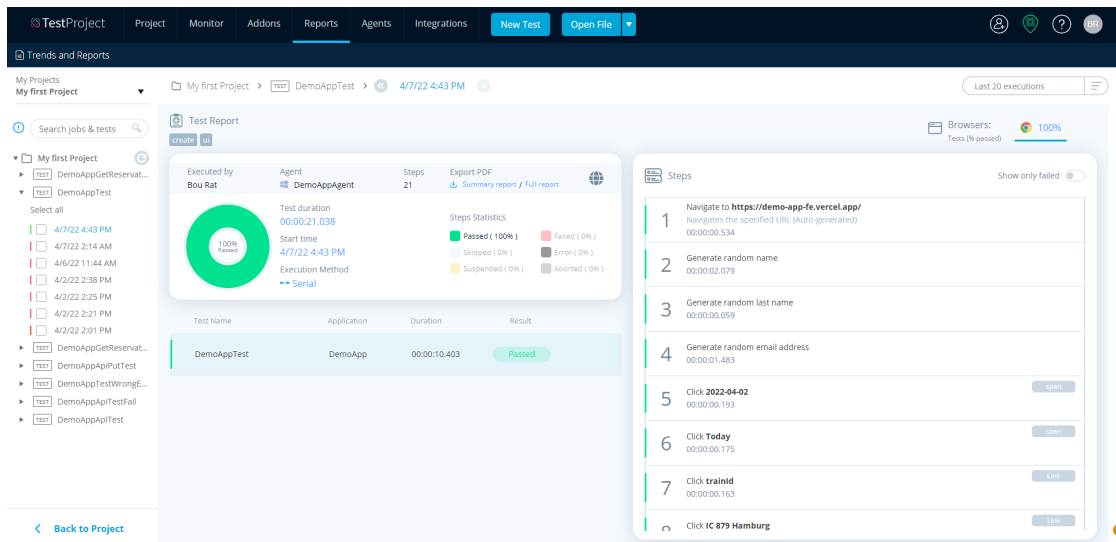


■ **Figure 5.9** Image showing the recording capability of TestProject

Test execution can be started manually or using a job. Jobs can be configured to execute tests at the specified time, serially or parallelly. Webhooks, email notifications, browsers and desired capabilities are configured in the job specification. It is possible to see the progress of the execution during the execution.

After a test is executed, the report can be seen in the reports section. Reports contain useful statistics about test runs. When a user selects a finished test execution, he is presented with information about the step by step execution from each browser. By clicking on a step, the user might find additional helpful information. The test execution can be exported to pdf as a full or summary report.

A report of test execution can be seen in the figure 5.10.



■ **Figure 5.10** Image showing a report of a test executed in TestProject

By default, TestProject is run in the cloud hybrid mode. That means that data and tests of the project are stored in the cloud. It could be problematic when TestProject is under maintenance because the users cannot access their projects. For this reason, it is possible to use TestProject offline and store files and tests locally. However, it requires some effort to configure but, in the end, is still worth the effort.

5.3.11 WebdriverIO

WebdriverIO is a progressive automation framework that offers a lot of different integrations and third-party services and libraries. WebdriverIO can run on the W3C WebDriver protocol or a solution based on the Chrome DevTools protocol.

WebdriverIO does not offer a test recording tool, but Katalon Recorder¹⁰ was used with an extension for exporting test recordings into WebdriverIO scripts.¹¹

The code for the implementation can be found in *WebdriverIODemo/test/specs/demoapp.js*.

The configuration file is located in *WebdriverIODemo/wdio.conf.js*.

The *await* keyword is present in the code, similar to Playwright and Puppeteer. WebdriverIO uses a CLI reporter by default.

In the example 23, a part of the implementation of Scenario 1 is shown. The *link text* and *name* selectors can be seen in the example. File uploading and Alert handling are made intuitively.

¹⁰<https://chrome.google.com/webstore/detail/katalon-recorder-selenium/ljdbomomgdgljniojadhoplhkpialdid>

¹¹<https://chrome.google.com/webstore/detail/webdriverio-exporter-for/kccnpljpbjgkjbjoncfpbmkobhacfekko>

```

describe("Demo application", () => {
  it("Should create a reservation.", async () => {
    await browser.url("https://demo-app-fe.vercel.app/");
    await expect(browser).toHaveUrl("https://demo-app-fe.vercel.app/");
    await expect(browser).toHaveTitle("React App");
    await $("//div/a/span").click();
    ...
    await $("=IC 879 Hamburg").click();
    ...
    await $(' [name="firstName"]').click();
    await $(' [name="firstName"]').setValue(faker.name.firstName());
    ...
    await $(' [name="file"]').setValue("C:\\\\random.file");
    await $("//button[@type='submit']").click();
    await browser.pause(3000);
    const msg = await browser.getAlertText();
    await console.log(msg);
    await browser.acceptAlert();
    await expect(msg).toBe("Reservation created successfully");
  });
});

```

■ **Code listing 23** Example showing part of the implementation of Scenario 1 in WebdriverIO

5.4 Summary

In conclusion, the demo app and the test plan for it were described in detail.

The three scenarios described earlier in this chapter were successfully automated for every analysed tool. Furthermore, API tests were developed using Cypress, Karate, Puppeteer, Playwright and TestProject. In total **51** tests were implemented for this project (3 scenarios, 3 types of API requests) to summarise the outputs.

Selenium is usually the first option people think of for automating browsers. It requires downloading drivers for the browsers and configuring them in code to run correctly. Users should be skilled enough to understand the configuration, testing and basic programming. The WebDriver is designed well, and the syntax is intuitive.

Cucumber was combined with the done Selenium code. Commands were mapped onto the test steps functions generated from the *feature* file into Java. Sometimes it was hard to find a piece of code that could be mapped to a line from the *emphfeature* file, so it was left empty. Cucumber and BDD are not often the best approach to testing and development. It highly varies on the team and the approach on BDD.

Appium works well, and it is often used in mobile application testing. Configuring and managing actual and emulated mobile devices can be challenging. Drivers and the Appium Server also need to be configured to run. Users should have a comparable technical level, as conveyed in the suggestion for Selenium users. Its API is designed well.

Karate is a great beginner-friendly multipurpose tool to suit most of the web application automated testing needs. Great for small projects and in cases where minimising the number of tools is optimal.

SikuliX is a decent tool when it is used as a Java library. SikuliX IDE is a bit interesting, but it cannot effectively export the recorded projects into code that could be used in other applications. That significantly limits environment configuration and access to crucial functionalities other tools provide, concluding that it is most useful as a Java library.

The process of writing tests in Cypress is enjoyable. The IDE snapshots and the development mode make it easy to fix bugs while writing the code. It is widespread and a good choice for many JS projects. It requires basic JS knowledge that can be also said about the rest of the analysed JS tools.

Puppeteer is just a library and needs to be combined with many other tools to be usable. The natural thing to do is to use Playwright as it is based on Puppeteer and functions as an E2E testing framework.

Working with Playwright was enjoyable, and the code gen functionality works nicely. The design of some of the aspects of the API was subjectively unintuitive. Nonetheless, the tool is still a great choice due to its ease of configuration, functionalities and integrations.

TestCafe is not a good choice when compared to other JS testing frameworks. TestCafe uses its way to manipulate the browsers instead of using the standard W3C WebDriver. Important functionalities are either added by importing JS files from GitHub or buying a premium version.

Nightwatch.js is probably best used for integrating with the BrowserStack cloud testing service. Some API aspects are not as satisfactory as in other JS testing frameworks.

TestProject is an outstanding cross-platform E2E automation testing tool. The default configuration is much better than in other analysed tools. Even a non-programmer can quickly start automating web applications. It can be operated effectively from the browser. It works as a hybrid cloud but offers the ability to work offline.

WebdriverIO is a likeable JS automation framework. The API is good, and it offers a lot of different integrations.

Benefits and risks of automated testing

Academic papers discussing automation software testing from economic and utility perspectives are the primary resources to evaluate the risks and benefits of automated testing.

6.1 Benefits

The results from a research paper “Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey” published in 2012 showed that the main benefits of test automation are:

- Reusability
- Repeatability
- Effort saved in test executions [121]

The results also indicated that automation improves test coverage, even in cases where excessive regression testing was not needed.

Other benefits are:

- Improved product quality
- Reduced testing time
- Reduction in cost
- Effort saved in test executions [122]

Increased fault detection was considered as a valid benefit by 58% of the respondents. One of them noted that the tester facilitates high defect detection and that one can get high defect detection with both manual and automated testing depending on how they are used. [121]

6.2 Risks and limitations

Regarding risks and limitations, the main factors in the study:

High initial cost In designing the test cases, buying a test automation tool, and training the staff.

Maintenance Was also perceived as problematic.

Unsuitable for specific needs 45% of the practitioners responded that test automation tools in 2012 offered a poor fit for their needs. [121]

In 2022 buying a test automation tool might no longer be a problem because the market offers many free and open-source tools for automation testing. The current capabilities of the tools are vastly improved compared to the tools in 2012.

80% of respondents did not agree that automated testing can replace manual testing. [121]

Other limitations of test automation:

- Process of test automation needs time to mature
- False expectations
- Inappropriate test automation strategy
- Lack of skilled people [122]

6.3 Other factors

Some of the respondents said that further research should be focused on more prevailing parameters regarding choosing tools for automated testing like:

1. Have an easy learning curve. This would mitigate the high initial investment required for test automation.
2. Utilization of test cases that are highly maintainable and robust. The maintenance burden of automated testing is likely to increase. Techniques and tools developed for software maintenance and evolution can be applied for test code would be valuable, like tools that automatically “fix” test cases when the production system changes.
3. Efficient creation of test cases. Test recording tools already allow the efficient production of test code, but the generated code can be unmaintainable and fragile.
4. Ease of configuration and integration with DevOps components
5. Support an incremental delivery of test automation. To lower the risk. [121]

Topics mentioned in the list, excluding the fifth point, were discussed in the analysis part of the thesis, which indicates that the thesis is not irrelevant.

Conclusion

In conclusion, the aims set at the beginning of this thesis were achieved.

Twelve tools used in automated testing of web applications were qualitatively analysed in detail and compared.

List of analysed tools:

- Karate
- Selenium
- Cypress
- SikuliX
- Appium
- Playwright
- Puppeteer
- TestCafe
- Nightwatch.js
- TestProject
- WebdriverIO

Beyond the terms of assignment, a qualitative analysis was also done. The method behind qualitative analysis was to use weighed parameters to categorise tools using objective and subjective metrics on which were the tools graded. Grades of the parameters were summed into total, and the tool with the highest amount of points was considered the most recommended for usage.

According to the results of the quantitative analysis, the most suitable tool to pick is the TestProject, as it is free and can work in the cloud or offline. It can be managed from a browser, and the default capabilities of the tool are excellent. Writing cross-browser tests without code only using UI is great because beginners can do so it can cut costs.

The quantitative analysis results are not final because the weights and the grading setup contain a mix of subjective and objective parametrised criteria. So anyone who wants to prefer some parameter of the tools over the others can change the weights or grades to get a recommended tool that better suits one's needs. The Google sheet is made publicly available for anyone to copy it, change values, and add or delete parameters and tools. The links can be found at the end of chapter 4.

A test plan was designed to test the demo application provided by the supervisor. The functionality of the demo application and the test plan are described in chapter 5.

The test plan was implemented using the analysed tools. In total **51** tests were implemented for this project (3 scenarios, 3 types of API requests). API tests were developed using Cypress, Karate, Puppeteer, Playwright and TestProject. The development of the UI test involved all of the tools. The tools were also compared in their implementations of the tests.

The benefits and liabilities of test automation were also discussed from an economic standpoint.

While writing this thesis, I gained much practical knowledge of testing tools and their capabilities.

The results of this thesis can be a great benefit for developers or other responsible individuals in selecting the right tools for the testing automation needs for a project.

Further work could optimise the quantitative analysis's parameters and grading scale. Also, adding new tools would be beneficial. Other research could compare the tools set up in a professional testing environment with different integrations. The tools could also be compared on other parameters like performance, etc.

Bibliography

1. SMARTBEAR SOFTWARE. *Introduction - Cucumber Documentation* [online]. 2019 [visited on 2022-04-24]. Available from: <https://cucumber.io/docs/guides/overview/>.
2. HOCKE, Raimund. *Introduction — SikuliX* [online]. 2021 [visited on 2022-05-08]. Available from: <https://sikulix.github.io/docs/>.
3. ATlassian. *Automated software testing for continuous delivery* [online]. 2022 [visited on 2022-04-20]. Available from: <https://www.atlassian.com/continuous-delivery/software-testing/automated-testing>.
4. KATALON. *What is End-to-End (E2E) Testing? — All You Need to Know* [online]. 2021 [visited on 2022-04-24]. Available from: <https://katalon.com/resources-center/blog/end-to-end-e2e-testing>.
5. BOSE, Shreya. *Functional Testing : Definition, Types and Examples — BrowserStack* [online]. BrowserStack, 2021 [visited on 2022-04-24]. Available from: <https://www.browserstack.com/guide/functional-testing>.
6. THEPRACTICALDEVELOPER. *Introduction to Microservice End-to-End tests with Cucumber* [online]. 2020 [visited on 2022-04-25]. Available from: <https://thepracticaldeveloper.com/cucumber-guide-1-intro-bdd-gherkin/>.
7. LI, Angela. *Understanding the Efficacy of Test Driven Development* [online]. Auckland, 2009 [visited on 2022-05-11]. Available from: <https://core.ac.uk/download/pdf/56361543.pdf>. MSc thesis. Auckland University of Technology, School of Computing and Mathematical Sciences. Supervised by Jim BUCHAN.
8. SMARTBEAR SOFTWARE. *Behaviour-Driven Development - Cucumber Documentation* [online]. 2019 [visited on 2022-04-24]. Available from: <https://cucumber.io/docs/bdd/>.
9. CUCUMBER. *Cucumber* [online]. 2022 [visited on 2022-04-26]. Available from: <https://github.com/cucumber>.
10. STACK OVERFLOW. *Newest 'cucumber' Questions* [online]. 2022 [visited on 2022-04-26]. Available from: <https://stackoverflow.com/questions/tagged/cucumber?tab=Newest>.
11. THOMAS, Peter. *Karate* [online]. Karate Labs, 2019 [visited on 2022-04-24]. Available from: <https://karatelabs.github.io/karate/>.
12. KARATELABS. *karatelabs/karate: Test Automation Made Simple* [online]. 2022 [visited on 2022-04-26]. Available from: <https://github.com/karatelabs/karate>.
13. SOFTWARE TESTING HELP. *Karate Framework Tutorial: Automated API Testing with Karate* [online]. 2022 [visited on 2022-04-26]. Available from: <https://www.softwaretestinghelp.com/api-testing-with-karate-framework/>.

14. KARATELABS. *Karate UI* [online]. 2022 [visited on 2022-04-26]. Available from: <https://karatelabs.github.io/karate/karate-core/>.
15. GATLING. *Open Source Load Testing - Gatling* [online]. 2022 [visited on 2022-04-27]. Available from: <https://gatling.io/open-source/>.
16. KARATELABS. *karate/karate-gatling at Master — karatelabs/karate* [online]. 2021 [visited on 2022-04-27]. Available from: <https://github.com/karatelabs/karate/tree/master/karate-gatling>.
17. KARATELABS. *IDE Support — KarateLabs/Karate Wiki* [online]. 2021 [visited on 2022-04-26]. Available from: <https://github.com/karatelabs/karate/wiki/IDE-Support>.
18. STACK OVERFLOW. *Newest 'karate' Questions* [online]. 2022 [visited on 2022-04-26]. Available from: <https://stackoverflow.com/questions/tagged/karate>.
19. BERGA, Mariana. *Top 7 Automation Testing Tools (2021)* [online]. Blog — Imaginary Cloud, 2021 [visited on 2022-04-24]. Available from: <https://www.imaginarycloud.com/blog/top-automation-testing-tools/>.
20. SOFTWARE FREEDOM CONSERVANCY. *WebDriver* [online]. 2021 [visited on 2022-04-28]. Available from: <https://www.selenium.dev/documentation/webdriver/>.
21. SOFTWARE FREEDOM CONSERVANCY. *Selenium components* [online]. 2021 [visited on 2022-04-28]. Available from: <https://www.selenium.dev/documentation/overview/components/>.
22. SHETH, Himanshu. *Selenium 4 New Features and Improvements — What's New in Selenium 4* [online]. LambdaTest, 2020 [visited on 2022-04-28]. Available from: <https://www.lambdatest.com/blog/selenium4-w3c-webdriver-protocol/>.
23. SOFTWARE FREEDOM CONSERVANCY. *Selenium components* [online]. 2021 [visited on 2022-04-28]. Available from: <https://www.selenium.dev/documentation/overview/components/>.
24. SOFTWARE FREEDOM CONSERVANCY. *Selenium Grid 4* [online]. 2021 [visited on 2022-04-28]. Available from: <https://www.selenium.dev/documentation/grid/>.
25. SOFTWARE FREEDOM CONSERVANCY. *History* [online]. 2012 [visited on 2022-04-29]. Available from: <https://www.selenium.dev/history/>.
26. STACK OVERFLOW. *Newest 'selenium' Questions* [online]. 2022 [visited on 2022-04-29]. Available from: <https://stackoverflow.com/questions/tagged/selenium>.
27. SELENIUMHQ. *Selenium* [online]. 2022 [visited on 2022-04-29]. Available from: <https://github.com/SeleniumHQ>.
28. JS FOUNDATION. *Introduction - Appium* [online]. 2022 [visited on 2022-04-24]. Available from: <https://appium.io/docs/en/about-appium/intro/>.
29. JS FOUNDATION. *Setup for Parallel Testing - Appium* [online]. 2022 [visited on 2022-05-01]. Available from: <https://appium.io/docs/en/advanced-concepts/parallel-tests/>.
30. JS FOUNDATION. *Find Elements - Appium* [online]. 2022 [visited on 2022-05-01]. Available from: <https://appium.io/docs/en/commands/element/find-elements/>.
31. JS FOUNDATION. *Screenshot - Appium* [online]. 2022 [visited on 2022-05-01]. Available from: <http://appium.io/docs/en/commands/session/screenshot/>.
32. JS FOUNDATION. *Start Screen Recording - Appium* [online]. 2022 [visited on 2022-05-01]. Available from: <https://appium.io/docs/en/commands/device/recording-screen/start-recording-screen/>.
33. JS FOUNDATION. *Appium: Mobile App Automation Made Awesome*. [Online]. 2022 [visited on 2022-05-02]. Available from: <https://appium.io/history.html?lang=en>.

34. STACK OVERFLOW. *Newest 'appium' Questions* [online]. 2022 [visited on 2022-05-02]. Available from: <https://stackoverflow.com/questions/tagged/appium>.
35. JS FOUNDATION. *appium/appium: Automation for iOS, Android, and Windows Apps*. [Online]. 2022 [visited on 2022-05-02]. Available from: <https://github.com/appium/appium>.
36. HOCKE, Raimund. *RaiMan's SikuliX* [online]. 2012 [visited on 2022-04-24]. Available from: <http://sikulix.com/>.
37. ESWARI, Anitha. *Introduction to Sikuli (GUI Automation Tool) - Sikuli Tutorial Part 1* [online]. 2014 [visited on 2022-05-08]. Available from: <https://www.softwaretestinghelp.com/sikuli-tutorial-part-1/>.
38. STACK OVERFLOW. *Newest 'sikuli' Questions* [online]. 2022 [visited on 2022-05-09]. Available from: <https://stackoverflow.com/questions/tagged/sikuli>.
39. HOCKE, Raimund. *RaiMan/SikuliX1: SikuliX Version 2.0.0+ (2019+)* [online]. 2022 [visited on 2022-05-09]. Available from: <https://github.com/RaiMan/SikuliX1>.
40. CYPRESS.IO. *Why Cypress? — Cypress Documentation* [online]. 2022 [visited on 2022-04-24]. Available from: <https://docs.cypress.io/guides/overview/why-cypress#Features>.
41. CYPRESS.IO. *Bundled Tools — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/references/bundled-tools#Mocha>.
42. CYPRESS.IO. *Bundled Tools — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/references/bundled-tools#Chai>.
43. CYPRESS.IO. *Reporters — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/tooling/reporters>.
44. CYPRESS.IO. *Dashboard — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/dashboard/introduction#Features>.
45. CYPRESS.IO. *Parallelization — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/guides/parallelization>.
46. CYPRESS.IO. *Get — Cypress Documentation* [online]. 2022 [visited on 2022-05-10]. Available from: <https://docs.cypress.io/api/commands/get>.
47. CYPRESS.IO. *Variables and Aliases — Cypress Documentation* [online]. 2022 [visited on 2022-05-10]. Available from: <https://docs.cypress.io/guides/core-concepts/variables-and-aliases>.
48. CYPRESS.IO. *Catalog of Events — Cypress Documentation* [online]. 2022 [visited on 2022-05-10]. Available from: <https://docs.cypress.io/api/events/catalog-of-events>.
49. CYPRESS.IO. *Environment Variables — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/guides/environment-variables>.
50. CYPRESS.IO. *Screenshots and Videos — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/guides/screenshots-and-videos>.
51. CYPRESS.IO. *Installing Cypress — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/getting-started/installing-cypress>.

52. CYPRESS.IO. *IDE Integration — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/tooling/IDE-integration>.
53. CYPRESS.IO. *Jira Integration — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/dashboard/jira-integration>.
54. CYPRESS.IO. *Slack Integration — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/dashboard/slack-integration>.
55. CYPRESS.IO. *CI Provider Examples — Cypress Documentation* [online]. 2022 [visited on 2022-05-09]. Available from: <https://docs.cypress.io/guides/continuous-integration/ci-provider-examples#Guides>.
56. MANN, Brian. *Cypress Is Now Public Beta* [online]. 2017 [visited on 2022-05-09]. Available from: <https://www.cypress.io/blog/2017/10/10/cypress-is-now-public-beta/>.
57. STACK OVERFLOW. *Newest 'cypress' Questions* [online]. 2022 [visited on 2022-05-09]. Available from: <https://stackoverflow.com/questions/tagged/cypress>.
58. CYPRESS.IO. *cypress-io/cypress: Fast, Easy and Reliable Testing for Anything That Runs in a browser*. [Online]. 2022 [visited on 2022-05-09]. Available from: <https://github.com/cypress-io/cypress>.
59. PUPPETEER. *puppeteer/puppeteer: Headless Chrome Node.js API* [online]. 2022 [visited on 2022-04-24]. Available from: <https://github.com/puppeteer/puppeteer>.
60. MANJUNATHA, Manu. *Selectors in Puppeteer - tools4testing* [online]. 2019 [visited on 2022-05-10]. Available from: <https://www.tools4testing.com/contents/puppeteer/selectors-in-puppeteer>.
61. COPEES, Flavio. *Introduction to Puppeteer* [online]. flaviocopes.com, 2019 [visited on 2022-05-10]. Available from: <https://flaviocopes.com/puppeteer/>.
62. OVERFLOW, Stack. *Newest 'puppeteer' Questions* [online]. 2022 [visited on 2022-05-10]. Available from: <https://stackoverflow.com/questions/tagged/puppeteer>.
63. MICROSOFT CORPORATION. *Fast and Reliable End-to-end Testing for Modern Web Apps — Playwright* [online]. 2022 [visited on 2022-04-24]. Available from: <https://playwright.dev/>.
64. MICROSOFT CORPORATION. *Parallelism and Sharding — Playwright* [online]. 2022 [visited on 2022-05-10]. Available from: <https://playwright.dev/docs/test-parallel>.
65. MICROSOFT CORPORATION. *Selectors — Playwright* [online]. 2022 [visited on 2022-05-10]. Available from: <https://playwright.dev/docs/selectors>.
66. MICROSOFT CORPORATION. *Assertions — Playwright* [online]. 2022 [visited on 2022-05-10]. Available from: <https://playwright.dev/docs/test-assertions>.
67. MICROSOFT CORPORATION. *Configuration — Playwright*. Playwright, [n.d.].
68. MICROSOFT CORPORATION. *Continuous Integration — Playwright* [online]. Playwright, 2022 [visited on 2022-05-10]. Available from: <https://playwright.dev/docs/ci>.
69. HEGDE, Ganesh. *Playwright Framework: Tutorial on Getting Started — BrowserStack* [online]. 2022 [visited on 2022-05-10]. Available from: <https://www.browserstack.com/guide/playwright-tutorial>.
70. STACK OVERFLOW. *Newest 'playwright' Questions* [online]. 2022 [visited on 2022-05-10]. Available from: <https://stackoverflow.com/questions/tagged/playwright>.
71. MICROSOFT CORPORATION. *microsoft/playwright: Playwright Is a Framework for Web Testing and Automation. It Allows Testing Chromium, Firefox and WebKit with a Single API*. [Online]. 2022 [visited on 2022-05-10]. Available from: <https://github.com/microsoft/playwright>.

72. DEVELOPER EXPRESS INCORPORATED. *Crossbrowser E2E Testing Framework — TestCafe* [online]. 2012 [visited on 2022-04-24]. Available from: <https://testcafe.io/>.
73. DEVELOPER EXPRESS INCORPORATED. *Reporters — Concepts — Guides — Docs* [online]. 2022 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402825/guides/concepts/reporters>.
74. DEVELOPER EXPRESS INCORPORATED. *Speed up Test Execution — Advanced Guides — Guides — Docs* [online]. 2022 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402963/guides/advanced-guides/speed-up-test-execution>.
75. DEVELOPER EXPRESS INCORPORATED. *Interact with the Page — Basic Guides — Guides — Docs* [online]. 2022 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402833/guides/basic-guides/interact-with-the-page>.
76. DEVELOPER EXPRESS INCORPORATED. *Assert — Basic Guides — Guides — Docs* [online]. 2022 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402837/guides/basic-guides/assert>.
77. DEVELOPER EXPRESS INCORPORATED. *Configuration File — Reference — Docs* [online]. 2015 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402638/reference/configuration-file>.
78. DEVELOPER EXPRESS INCORPORATED. *Screenshots and Videos — Advanced Guides — Guides — Docs* [online]. 2019 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402840/guides/advanced-guides/screenshots-and-videos>.
79. INCORPORATED, Developer Express. *Continuous Integration — Guides — Docs* [online]. 2022 [visited on 2022-05-10]. Available from: <https://testcafe.io/documentation/402809/guides/continuous-integration>.
80. DEVELOPER EXPRESS INCORPORATED. *Introducing TestCafe Testing Framework — Framework — Release Notes* [online]. 2016 [visited on 2022-05-10]. Available from: <https://testcafe.io/402864/release-notes/framework/2016-10-17-introducing-testcafe-open-source-testing-framework>.
81. STACK OVERFLOW. *Newest 'testcafe' Questions* [online]. 2022 [visited on 2022-05-10]. Available from: <https://stackoverflow.com/questions/tagged/testcafe>.
82. DEVELOPER EXPRESS INCORPORATED. *DevExpress/testcafe: a Node.js Tool to Automate end-to-end Web testing*. [Online]. 2022 [visited on 2022-05-10]. Available from: <https://github.com/DevExpress/testcafe>.
83. BROWSERSTACK. *Nightwatch.js — Node.js Powered End-to-End Testing Framework* [online]. 2022 [visited on 2022-04-24]. Available from: <https://nightwatchjs.org/>.
84. BROWSERSTACK. *API Reference — Nightwatch.js* [online]. 2022 [visited on 2022-05-10]. Available from: <https://nightwatchjs.org/api/element/>.
85. BROWSERSTACK. *API Reference — Nightwatch.js* [online]. 2022 [visited on 2022-05-10]. Available from: <https://nightwatchjs.org/api/useractions/>.
86. BROWSERSTACK. *API Reference — Nightwatch.js* [online]. 2022 [visited on 2022-05-10]. Available from: <https://nightwatchjs.org/api/ensure/#ensure-api>.
87. BROWSERSTACK. *API Reference — Nightwatch.js* [online]. 2022 [visited on 2022-05-10]. Available from: <https://nightwatchjs.org/api/assert/>.
88. BROWSERSTACK. *nightwatch-docs/test-environments.md at Main — nightwatchjs/nightwatch-docs* [online]. 2021 [visited on 2022-05-10]. Available from: <https://github.com/nightwatchjs/nightwatch-docs/blob/main/guide/running-tests/test-environments.md>.

89. NGS. *ngs/nightwatch-slack-reporter: a @nightwatchjs Reporter That Notifies Results to Slack* [online]. 2016 [visited on 2022-05-10]. Available from: <https://github.com/ngs/nightwatch-slack-reporter>.
90. TSCHAN, Sebastian. *Docker Hub* [online]. 2022 [visited on 2022-05-10]. Available from: <https://hub.docker.com/r/blueimp/nightwatch>.
91. STACK OVERFLOW. *Newest 'nightwatch.js' Questions* [online]. 2022 [visited on 2022-05-10]. Available from: <https://stackoverflow.com/questions/tagged/nightwatch.js>.
92. BROWSERSTACK. *nightwatchjs/nightwatch: End-to-end Testing Framework Written in Node.js and Using the W3C WebDriver API* [online]. 2022 [visited on 2022-05-10]. Available from: <https://github.com/nightwatchjs/nightwatch/>.
93. TRICENTIS. *Why Should You Use TestProject - TestProject Documentation* [online]. 2022 [visited on 2022-04-24]. Available from: <https://docs.testproject.io/why-testproject/why-should-you-use-testproject>.
94. TRICENTIS. *Using Data Driven Tests and Jobs in TestProject - TestProject Documentation* [online]. 2021 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/using-the-smart-test-recorder/using-data-driven-jobs-in-testproject>.
95. TRICENTIS. *Parallel Execution - TestProject Documentation* [online]. 2022 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/schedule-and-run-tests/parallel-execution>.
96. TRICENTIS. *Installation and Setup - TestProject Documentation* [online]. 2022 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/getting-started/installation-and-setup>.
97. TRICENTIS. *Integrations - TestProject Documentation* [online]. 2022 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/testproject-integrations/>.
98. TRICENTIS. *TestProject Agent in Docker - TestProject Documentation* [online]. 2022 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/testproject-agents/testproject-agent-in-docker>.
99. TRICENTIS. *TestProject Agent on Kubernetes - TestProject Documentation* [online]. 2022 [visited on 2022-05-11]. Available from: <https://docs.testproject.io/testproject-agents/testproject-agent-on-kubernetes>.
100. TRICENTIS. *About us — TestProject* [online]. 2022 [visited on 2022-05-11]. Available from: <https://testproject.io/about/>.
101. TIWARI, Garima. *WebDriverIO Tutorial for Selenium Automation — BrowserStack* [online]. 2021 [visited on 2022-04-24]. Available from: <https://www.browserstack.com/guide/webdriverio-tutorial-for-selenium-automation>.
102. WEBDRIVERIO. *webdriverio/webdriverio: Next-gen Browser and Mobile Automation Test Framework for Node.js* [online]. 2022 [visited on 2022-04-24]. Available from: <https://github.com/webdriverio/webdriverio>.
103. HEGDE, Ganesh. *Cypress Vs WebDriverIO: Key Differences — BrowserStack* [online]. 2021 [visited on 2022-05-11]. Available from: <https://www.browserstack.com/guide/cypress-vs-webdriverio>.
104. JS FOUNDATION. *Automation Protocols — WebDriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/automationProtocols/>.
105. JS FOUNDATION. *Organizing Test Suite — WebDriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/organizingsuites/>.
106. JS FOUNDATION. *Frameworks — WebDriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/frameworks>.

107. JS FOUNDATION. *Assertion — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/assertion>.
108. JS FOUNDATION. *Selectors — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/selectors>.
109. JS FOUNDATION. *PerformanceTotal Service — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/wdio-performancetotal-service/>.
110. JS FOUNDATION. *Devtools Service — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/devtools-service>.
111. JS FOUNDATION. *Testrunner Configuration — WebdriverIO* [online]. 2017 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/configurationfile>.
112. JS FOUNDATION. *Video Reporter — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/wdio-video-reporter/>.
113. JS FOUNDATION. *Getting Started — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/gettingstarted/>.
114. JS FOUNDATION. *Docker — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/docker>.
115. JS FOUNDATION. *Slack Service — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/wdio-slack-service>.
116. JS FOUNDATION. *Microsoft Teams Service — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/wdio-ms-teams-service>.
117. JS FOUNDATION. *Autocompletion — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/docs/autocompletion>.
118. STACK OVERFLOW. *Newest 'webdriver-io' Questions* [online]. 2022 [visited on 2022-05-11]. Available from: <https://stackoverflow.com/questions/tagged/webdriver-io>.
119. JS FOUNDATION. *Need Help? — WebdriverIO* [online]. 2022 [visited on 2022-05-11]. Available from: <https://webdriver.io/community/support/>.
120. STRATOX CLOUD NATIVE. *Ticket Reservation Demo App - Java Spring Boot - Stratox Cloud Native - Tetra* [online]. 2022 [visited on 2022-05-03]. Available from: <https://shorturl.at/fhvLW>.
121. RAFI, Dudekula; MOSES, Katam; PETERSEN, Kai; MÄNTYLÄ, Mika. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: 2012, pp. 36–42. ISBN 978-1-4673-1821-1. Available from DOI: 10.1109/IWAST.2012.6228988.
122. LINDHOLM, David. *Economics of Test Automation: Test case selection for automation*. Linköping, 2019. Available also from: <http://www.diva-portal.org/smash/get/diva2:1294193/FULLTEXT01.pdf>. MSc thesis. Linköping University, Department of Computer and Information Science. Supervised by Azeem AHMAD.

Content of the attached media

	readme.txt	brief description of the contents of the media
	src	
	impl	implementation source code
	thesis	source form of the work in \LaTeX format
	text	text of thesis
	thesis.pdf	text of thesis in PDF format