http://doi.org/10.35784/iapgos.5374

received: 24.07.2023 | revised: 18.08.2023 | accepted: 09.09.2023 | available online: 20.12.2023

BROWSERSPOT – A MULTIFUNCTIONAL TOOL FOR TESTING THE FRONT-END OF WEBSITES AND WEB APPLICATIONS

Szymon Binek[1,2], Jakub Góral[3]
[1]ClickRay Sp. z o.o., Cracow, Poland, [2]Kozminski University, Warsaw, Poland, [2]University of Economics, Cracow, Poland

Abstract. *The article presents the multifunctional BrowserSpot tool, which serves as an automated environment for testing websites and web applications for Android and iOS systems. It highlights and describes the individual stages of research and development work, the issues with solutions currently available on the market, as well as the project's results. The article also discusses the reasons for undertaking work on the tool, its functionalities, and the methods of its usage.*

Keywords: automation testing, bug tracking, smart test automation, responsive website

BROWSERSPOT – MULTIFUNKCJONALNE NARZĘDZIE DO TESTOWANIA FRONT-ENDU STRON INTERNETOWYCH ORAZ APLIKACJI SIECIOWYCH

Streszczenie. *W artykule zaprezentowano multifunkcyjne narzędzie BrowserSpot stanowiące zautomatyzowane środowisko do testowania stron internetowych oraz aplikacji webowych dla systemów Android i iOS. Wyróżnione i opisane zostały poszczególne etapy prac badawczo rozwojowych, problemy aktualnych rozwiązań dostępnych na rynku, a także rezultaty projektu. Przedstawiono również powody podjęcia się prac nad narzędziem, funkcjonalności narzędzia, oraz sposoby jego użytkowania.*

Słowa kluczowe: automatyzacja testowania, śledzenie błędów, inteligentna automatyzacja testów, responsywna strona internetowa

Introduction

This publication is the result of an R&D project carried out under a grant from the European Union between years 2017- 2019. The aim of our R&D project was to introduce a new "Software as a Service" (commonly known as SaaS) solution to the market. More specifically, it could be considered as a Testing as a Service platform [1]. It's main purpose is to automate the process of validation of website pages in the context of proper visibility on different devices and resolution types and generate automated reports regarding the results of the performed tests. We recognized the lack of modern automatic testing systems [3] that could work on testing different devices, process large amounts of data and also be a cloud based solution. The solution would also be much more user friendly thanks to the simple UX and UI design, codeless workflow and a drag&drop interface. The result of the R&D project was a fully working cloud based application, ready to be launched on the market for customers, as well as for corporate clients.

1. Literature and market review

During the start of our R&D project, there were already some solutions that offered automated tests of website pages but they either required technical knowledge to program and execute requests or they were not cloud-based. Since then, it has changed and we see more simulated solutions being present on the market. One of the industry's problems during the start of our R&D project was the lack of growth in noticeable increases in the efficiency and effectiveness of software testing. Most testing systems were designed on the basis of systems developed in the early 21st century. A big problem for the industry was the ever-increasing amount of data, making manual testing ineffective, and the increasing complexity of applications requiring more and more testing effort [4]. It was therefore necessary to redefine the work of testers, implementing new methods and practices.

The industry's problems were illustrated by the so-called Test Gap or the coverage testing gap (Fig. 1). It reveals areas that should have been tested, but could not be analysed due to time constraints, human resource limitations or high frequency of builds. An additional obstacle was that the testing gap is constantly widening, and the complexity of testing increases exponentially as new features are released. More code is produced than testers can test, leading to a gap in test coverage [2].

The main difference between the available solutions and our R&D project was that all of our tests would be performed on our cloud therefore we would not use the computing resources of our user's computers. In short, compared to the available solutions on the market, the user had to connect to a virtual machine and stay with the internet connection until the test finished. If the test failed for some reason, the automated test was stopped. A broken internet connection also caused the test to stop. Our solution, on the other hand, works based fully in the cloud, where the test scenario is created by the user using a drag&drop interface, and once finished, the test is sent to the cloud, where the algorithm distributes the test on the available devices. This also produces much better efficiency as compared to the available solution, which requires a constant connection to the internet and to the virtual machine according to the scheme one user-one virtual machine. The aim of the R&D project was to develop and prepare for market deployment a solution to address the challenges in test automation identified above, for which the name BrowserSpot was adopted.

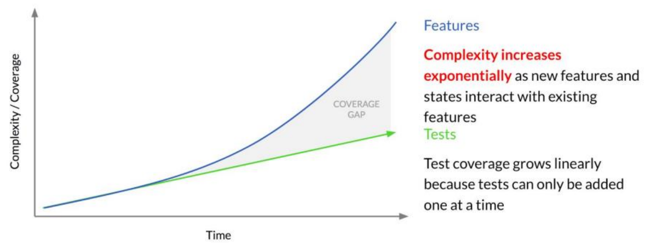


Fig. 1. The coverage testing gap theory [2]

2. Research and development

This chapter presents conducted research and development during the project. It was divided into eight stages that represent the goals and aims of individual stages of the project. Each stage was a combination of tasks related to both the research and development stages of this project.

First stage of the project focused on creating a code structure for the Selenium server, connecting it to browsers and devices. Key developments included algorithms for desktop and mobile browser operations, along with emulators for various platforms. The stage also aimed to verify technology functionality and browser compatibility with the Selenium server.

The aim of second stage was to research the possibility of effective separation of virtual environment technologies (created in stage 1) into physical machines and whether it will be possible to seamlessly communicate and exchange data

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Utwór dostępny jest na licencji Creative Commons Uznanie autorstwa – Na tych samych warunkach 4.0 Miedzynarodowe.

between the server and these machines in ranges of the hypotheses described in stage 1. The research will lead to knowledge of the technical possibilities of the server in conjunction with physical devices. A local environment will be created in which the technical capabilities of the server will be tested.

During the third stage of the project, research was carried out in terms of the introduction of cloud computing to the application. We also developed the user interface needed to run a SaaS solution at this stage. The conducted research gave an unequivocal answer to whether SaaS's cloud computing is a more optimal arrangement for the BrowserSpot solution. A report was produced to showcase the performance of the cloud base solution.

The fourth stage aimed to enhance the module's technical capabilities for improved site performance. This involved creating a component for page display performance, enabling the verification of page links, assessing rendering quality, error detection, and generating a detailed list of page element loading information. Key tasks included recording data to a database, enabling data export to the frontend, developing test generation capabilities, designing frontend components, implementing the project via API, testing module functionality, and creating and verifying necessary databases.

The fifth stage focused on expanding the capabilities of the language correctness testing module within BrowserSpot/component. This involved connecting it to external databases for research on grammatical correctness, stylistic accuracy, and content adaptation on websites. Activities included data handling, frontend integration, algorithm development, design, API implementation, testing, and database development.

The sixth stage aimed to enhance the module's capabilities for comparing elements on rendered web pages across different browsers. It involved developing a component to identify errors in individual page elements and generate descriptive reports about differences in their display. The module extracted data from the server and processed it to detect errors related to specific elements on web pages. Key tasks included algorithm development, test generation for error detection, element comparison, error presentation, integration with the frontend, and technology verification.

The seventh stage involved conducting industrial research to acquire knowledge and skills in data conversion, communication, and presentation. A component was developed to retrieve errors in various website elements and generate descriptive reports on differences in element display across different browsers. Research included developing a communication algorithm and protocol for handling website rendering data and presenting it in the user interface.

The technical solutions obtained would be part of a prototype for the BrowserSpot service. Goals for this stage included generating comprehensive reports combining correctness checks for page display, linguistic correctness, and individual element examination, as well as creating partial reports as notifications for user activities related to verification.

Activities included developing algorithms for data retrieval and export to the frontend, conducting tests on data processing and report generation, PDF report generation, frontend component design, API implementation, integration, and various tests to ensure module and tool correctness. A database algorithm was also developed for module operation.

The eighth stage focused on integrating all technology elements and conducting real-world testing for the BrowserSpot service pilot. The goal was to complete a prototype of the BrowserSpot tool and test it in real conditions, including demonstrations among a selected group of users.

Prior to the demonstrations, activities included integrating technological components, addressing errors in information generation and browser operations, improving the API, finalizing the design of components and the prototype, making frontend adjustments, and verifying the tool's operation in terms of error generation, API functionality, frontend performance, and overall workload.

The prototype demonstrations aimed to verify assumptions regarding performance, user support, and concurrent testing, the correctness of error generation, report generation in various software components, interface functionality, and identifying and resolving any performance bottlenecks in the tool.

3. Results

The aim of this chapter is to present results of each individual stage of the project in order to give an insight on which aims and goals have been properly achieved, and which were more problematic.

During the first stage of the project, a virtual machine was created as the runtime environment and the core of the entire project. A virtual machine is an isolated environment running on a host computer, and simulating the operation of a physical device with a separate operating system, freely chosen independently of the host computer system. The main component running in such a prepared environment was an engine based on Selenium technology. Selenium is a framework that automates tasks related to functional testing of web applications. It allows you to control web browsers from the code level and define sets of actions to execute within them. Selenium itself includes a set of tools - among them, WebDriver was used in the project. Its integration with Java was used. After the creation of a working prototype of the BrowserSpot tool engine with such features as: configuration and readiness of the virtual environment for further development, server startup, creation of an algorithm to unify support for different browsers, the ability to remotely operate browsers (Google Chrome, Mozilla Firefox, Opera, Safari, Internet Explorer) (Fig. 2) on the virtual machine, the ability to remotely operate browsers on mobile devices emulated within the virtual machine, logging of the course of action of browsers, the end of this stage have been reached.

After the first stage of work, the system used virtual machines supported by standard desktop operating systems and a set of mobile device emulators. The operation of the latter offered an approximation of the behavior of the devices, and thus the ability to test the operation of sites in their environments, but did not correspond 1:1 to real use cases due to a number of drawbacks that the use of emulators entails. So at this stage, as much of the emulators as possible were replaced with physical devices. This required designing proprietary mechanisms for integrating physical devices into the tool engine. The algorithm had to provide reliable two-way communication with the device. A solution was implemented to support physical devices from the desktop computer and display the screen content of the mobile device on the computer screen - the ability to perform real-time operations - latency was reduced to a minimum. A similar solution was applied to desktop systems. New functions were designed and implemented, such as downloading screenshots containing web pages rendered by different browsers, downloading codes of tested sites, comparing how sites are displayed by different environments. Implementation of new functionalities and development of existing ones allowed the creation of a database. The database was to be used to store all data necessary for the operation of the site and arising during its use. At this stage of the project's advancement, these included user data, logs, screenshot data, and test history. Due to the proliferation of system functions and the connection of new devices to the system, the need to implement load balancing methods was recognized. To this end, a load balancer was used, which is a solution for distributing tasks among devices in the infrastructure. To conclude the second stage of the project, a working prototype was developed with the following features: use of physical devices alongside emulators, ability to remotely operate physical devices in real time, downloading screenshots, comparing page renders, implemented and secured database, use of load balancer.

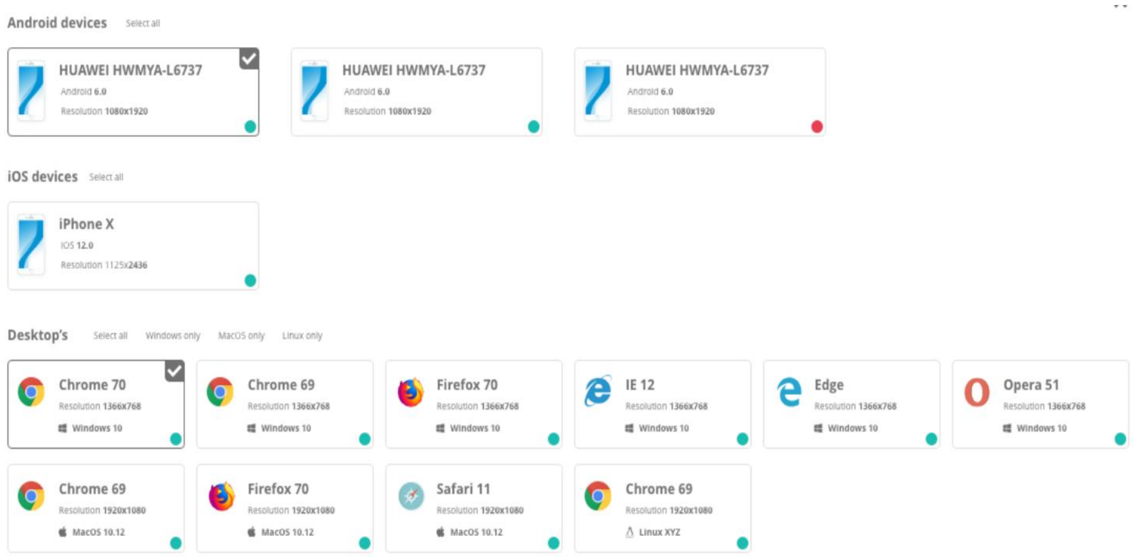


Fig. 2. Ability to run tests on multiple devices, browsers and systems using

Table 1. Traditional and SaaS model comparison

Traditional model (on premise)	SaaS model
• The user has to maintain the entire system manually in order to conduct tests (different browser versions, etc.). • The performance of the software (speed of conducting tests and task execution) is limited by the user's computer computational power. • Blocking other user activities while conducting tests (tests do not work in the background).	• The user receives the whole package, including the software and infrastructure (devices in the cloud with various browser versions, different resolutions, without incurring device maintenance costs). • The model can be scaled by enriching the cloud with new devices; a single task is distributed and executed simultaneously on multiple computers, and traffic can be balanced using a load balancer. • The user's computer is not burdened as the application natively launches browsers, and tests are executed in the background.

During the third stage, a performance comparison module has been developed between an application running in a traditional model (when the user installs the software on his computer) and software delivered as a service (SaaS model). The analysis was carried out from the perspective of user usability.

Performance calculations were carried out for a number of test cases. Estimated software performance (speed of testing and task execution) for an example test case – to test a page of 57 subpages, on 3 browsers. Assumptions used for calculations: average time to test 1 sub-page using one computer – 35 s (averaged value, the exact time depends on the parameters of the device such as the amount of RAM, processor speed, etc.). The results of our SaaS model showed a drastic improvement in performance. Depending on the complexity of the webpage and the components of a compared traditional model computer, we have noticed a performance increase of up to 70% using our SaaS model solution.

Given these results, the focus was on moving the service to the cloud and making it available in a SaaS model – Software as a Service. This is a cloud computing model in which a service is made available to users remotely over the Internet, and all computing is done on the server side. Until now the service operated locally, requiring installation and proper configuration by the user, or rather, the developer. The key advantages of moving the service to the SaaS model include no need for the user to install the software which results in a lower entry threshold for potential customers, no specific hardware requirements, easier distribution of the solution, easier maintenance of the service in the long term, increased scalability of the system. To enable the service to operate in this model, it was necessary to create a user interface. This part of the work included: establishing the image of the application, designing layouts, implementing layouts, designing and implementing mechanisms for controlling remote devices, integrating the front-end with the server's API, testing the interface, starting work on the ongoing maintenance of the service.

The completion of stages form 4 to 6 have been reached after the successful implementation and testing of: the performance testing module, the language testing module and the render comparison module. The automatic site quality (performance) testing module allows you to verify a website for performance and correct application of SEO practices, and suggests possible improvements. The test is carried out completely automatically – the user only enters the website address, and assigns a given test to: the client, the project and a milestone he/she has set. Following the performance test, a report is automatically generated that contains a range of data, including, in particular: information about the quantitative share of the server's individual http response codes (200, 301,404), information about the amount of data downloaded from the server along with detailed addresses from which these data were downloaded, information about the amount of data downloaded from the server taking into account the type of data (html, css, js, etc.), basic summary information about response and page load times, suggestions for site improvements (YSLOW) taking into account Javascript code optimization, file compression of the number of HTTP requests, etc.

The second functionality introduced during these stages was a module for testing the linguistic correctness of the site. Running on the basis of the LanguageTool tool, the algorithm checks all the text visible to the user and checks it for: grammar, spelling, phraseology, punctuation, syntax. After the test, a report is generated with suggestions for improvements. The third functionality launched at this stage of work was a module for comparing renders of different browsers. The tool is based on previously developed functionalities – downloading screenshots, html code and a tool looking for differences. When run, the module returns a result containing previews of the page rendered by each of the selected browsers. Each of them can be freely viewed and manually compared. The main element of the tool, however, is an automatic comparison of rendered pages. The tool allows you to: select two renders and juxtapose them against each other, synchronously scroll through the rendered pages for visual comparison, automatically find and list differences, set a tolerance for differences in element sizes, review and locate differences, search for elements that are missing, search for elements that appeared in the code, but you are not sure about their display on the user's side. In addition, we have developed component responsible for retrieving detected errors in the scope of individual elements of the studied website and generating reports automatically presenting information in a descriptive manner on the differences in the display of individual elements of the rendered page on different browsers.

During the seventh stage of the project, a component responsible for generating reports for the end user was developed. Functionalities related to storing and sharing historical and current reports have been implemented. All reports are stored in a secured database. The repository contains reports from 3 types of tests: render tests, performance and SEO tests, and language tests. Each of the reports is available in the BrowserSpot tool itself under the "Reports" tab, as well as it is possible to download them in the form of a formatted PDF document for presentation outside the service or in paper form.

The eighth stage consisted of the merge of technology components developed in earlier stages, a series of tests, and a prototype of the BrowserSpot tool that was made available to a closed group of users for UX testing. As a result of the work carried out as part of the project, a complete IT product was created with the following features and functionalities: backend based on technologies: Java and Selenium, virtualization of devices in an isolated environment, encrypted database storing application and user data, support and functions for remote control of selected physical devices making up the device testing lab, control of network traffic with a load balancer, making the service available in the SaaS model, mechanisms for handling user accounts and workspaces, rendering web pages on emulated devices, virtual machines and physical devices, downloading rendered pages, downloading source code of pages, automatic comparison of renders between browsers, automatic analysis of text displayed by browsers for correctness and suggestions for improvements, automatic creation of reports on performance and meeting SEO standards organization of the above tests by client, ability to plan tests for the future and create a schedule, archiving of historical tests, creation of an API for the front-end interface, creation of a front-end interface, tutorials, access to documentation, modifiable interface of the management panel, support for automatic performance testing functions, SEO, language, renders, support for the report archive view, support for the schedule, insight into statistics. The described functionalities went through a cycle of tests and corrections to eliminate possible errors in both the application design and its implementation. Subsequently, UX testing was carried out in accordance with the following methodological assumptions. The results of the survey were collected using a dedicated questionnaire. The data source was a task test with a framework scenario - each participant was tasked with using BrowserSpot in a real - current or historical - case. The requirements for the task were formulated as follows: the task is related to the testing of any service, the object of the task is a web application with an average level of operational complexity, the task involves a single application or its component. The group of respondents was selected from among programmers, QA staff (software testers) and IT project managers. The surveys were conducted asynchronously, without a moderator. There were 3 areas of problem analysis: quality of solutions provided, redundancy or functional deficiencies, bugs and overall service performance. The study participants were provided with a manual and basic user materials, and no guidance or advice was given during the test. The study was anonymous, the only data of the participants provided during the course of the study was their position and seniority. The results obtained were used to improve the BrowserSpot service before its full rollout to the market.

4. Selected applications

4.1. Translation of the execution code

The code that is used to take a screenshot of the website has to be translated based on the device, software version and used application. The tool checks the selected parameters and does the translation in the background, without user input in order to ensure its ease of use. The translated code is then executed on the website and the results are compared and verified. Here are the examples of the translated code snippets for the use in the chrome browser.

The function to scroll to the top of the page and wait 2 seconds for the page to fully load:

```
    {
this.driver.executeScript("document.querySelector('body').scrollTop=" +
(0) + ";document.querySelector('html').scrollTop=" + (0) + "");
        try {
            Thread.sleep(2000);
    }
```

The function to take a screenshot of the page:

```
    {
TakesScreenshot takesScreenshot = (TakesScreenshot) new
Augmenter().augment(this.driver);
byte[] image = takesScreenshot.getScreenshotAs(OutputType.BYTES);
    }
```

4.2. Environment scalability

The tools environment has been developed in a way that allows an easy expansion if necessary. New devices, software, and versions can be added and quickly configured on the backend of the tool, but also the new systems and machines that handle the operation functions of the tool. That way the need for more computation power can be solved by simply adding a new server or computer and download the appropriate software that will automatically set itself up and keep updated with the rest of the environment.

4.3. Tasks load balancer

The tool has been designed with task load balancing in mind. The way this problem has been solved is by running a load check before sending a task request to the specific machine. If the check finds a machine that is currently under load it forwards the request to the next one and the process begins again. The simplified way is illustrated on the following chart (Fig 3).

4.4. Physical devices

The tests conducted using the tool are requested to be performed on a series of physical devices that are connected to the system. Usually these types of tests are performed on an emulator on virtual machine that simulates the selected device. The decision to use physical devices has been made mainly due to the numerous problems with emulators but also for the performance and accuracy of the tests and results.

5. Conclusions

As a result of the conducted R&D project, a comprehensive testing tool has been developed, allowing for significant automation and standardization of testing activities, particularly in small and medium projects. The results of individual project stages have verified the assumptions defined at the beginning of the R&D project. The results have been implemented in the market. The tool is offered under the commercial name BrowserSpot in a SaaS/TaaS model. The tool addresses key challenges faced by the automated testing industry, primarily related to technical infrastructure and high costs of maintaining device laboratories and human resources. The developed tool helps alleviate staffing issues and is accessible to individuals who do not possess extensive programming skills. After the completion of the R&D project, the tool was further developed to better align with user needs and expectations, such as enriching it with a function for creating codeless automated tests using a graphical user interface and drag-and-drop functionality and progress tracking dashboard (Fig. 4).

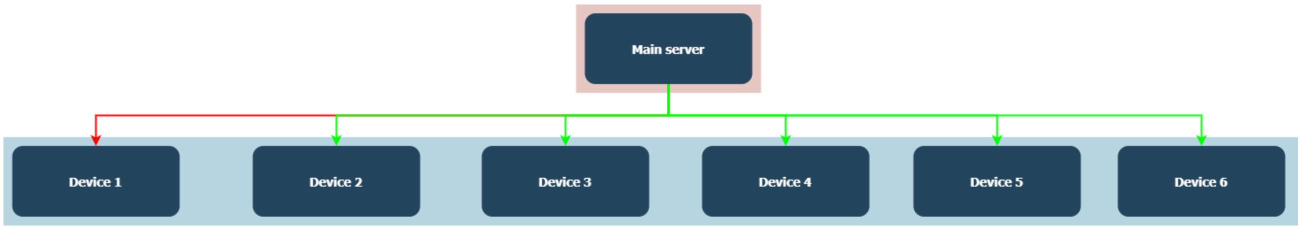
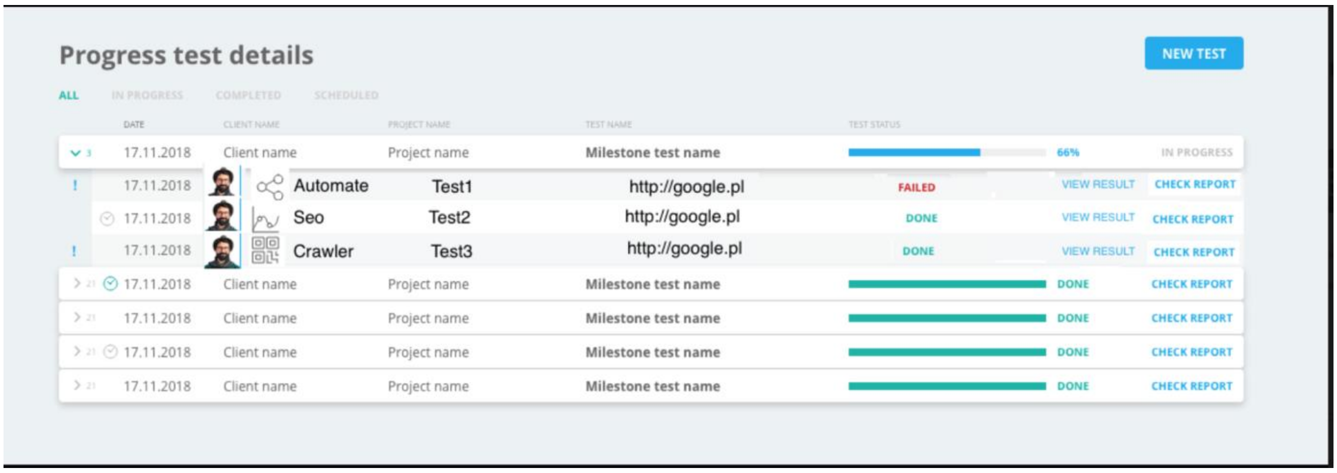


Fig. 3. Load balancing method flowchart



	DATE	CLIENT NAME	PROJECT NAME	TEST NAME	TEST STATUS		
3	17.11.2018	Client name	Project name	Milestone test name		66%	IN PROGRESS
!	17.11.2018	Automate	Test1	http://google.pl	FAILED	VIEW RESULT	CHECK REPORT
	17.11.2018	Seo	Test2	http://google.pl	DONE	VIEW RESULT	CHECK REPORT
!	17.11.2018	Crawler	Test3	http://google.pl	DONE	VIEW RESULT	CHECK REPORT
21	17.11.2018	Client name	Project name	Milestone test name		DONE	CHECK REPORT
21	17.11.2018	Client name	Project name	Milestone test name		DONE	CHECK REPORT
21	17.11.2018	Client name	Project name	Milestone test name		DONE	CHECK REPORT
21	17.11.2018	Client name	Project name	Milestone test name		DONE	CHECK REPORT

Fig. 4. The function of progress tracking dashboard

Acknowledgments

The R&D project, the results of which were presented in the article, was co-financed using a grant by the European Union. Project was titled „Developing a unique rendering mechanism to launch a proto-type BrowserSpot service". Research project no. RPMP.01.02.01-12-0487/16 co-financed from the funds of Regional Operational Programme for the Malopolska Region 2014-2020, Priority Axis Knowledge Economy, Measure 1.2 Research and innovation in Enterprises, Sub-measure 1.2.1 R&D projects of Enterprises co-financed by the European Regional Development Fund.

References

[1] Ali A. et al.: Automated Parallel GUI testing as a service for mobile applications. Journal of Software: Evolution and Process 30(10), 2018 [http://doi.org/10.1002/smr.1963].
[2] Arbon J.: AI for Software Testing. Pacific NW Software Quality Conference, Portland 2017.
[3] Moreira R. M. et al.: Pattern-based GUI testing: Bridging the gap between design and Quality Assurance. Software Testing, Verification and Reliability 27(3), 2017, e1629 [http://doi.org/10.1002/stvr.1629].
[4] World Quality Report. Capgemini, 2019 [http://www.capgemini.com/news/press-releases/world-quality-report/] (accessed 29 June 2023).

M.Sc. Szymon Binek
e-mail: s.binek@clickray.eu

He is the main originator and the co-founder of ClickRay. He is a specialist in online and HubSpot API development services and a truly wide range of digital ventures and experiments for clients. He supervised a research and development project co-financed by European funds, which resulted in the BrowserSpot platform.
Research interests: Artificial Intelligence and Machine Learning, Software Test Automation.

http://orcid.org/0009-0001-7936-8056

B.Sc. Jakub Góral
e-mail: goral.jakub99@gmail.com

In 2022 he received a Bachelor of Science degree (B.Sc.) at the University of Economics in Cracow in the field of Business Management with a specialization in Small Business Management. Later on he continued his studies and is now in the process of getting a Master's degree in International Business Management.
Research interests: technology, Industry 4.0, Artificial Intelligence, Data science.

http://orcid.org/0009-0003-9634-4915