

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Assessment of Simple Web Applications in a Code Playground

Luís Miguel Maia da Costa



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. José Paulo Leal

Second Supervisor: Prof. Ricardo Queirós

July 27, 2023



# **Assessment of Simple Web Applications in a Code Playground**

**Luís Miguel Maia da Costa**

Mestrado em Engenharia Informática e Computação

July 27, 2023

# Resumo

Cursos introdutórios de programação requerem geralmente que os alunos resolvam vários exercícios em várias linguagens de programação para testar as suas competências. A avaliação automática é útil uma vez que existem várias formas de alcançar o mesmo resultado, e os professores demoram menos tempo a avaliar os seus alunos.

Ao longo dos anos, várias ferramentas de avaliação, utilizando uma abordagem black-box, foram criadas para analisar automaticamente o código em diferentes linguagens de programação. O modelo black-box emprega uma série de casos de teste para avaliar o software, comparando os resultados do programa do aluno com o resultado esperado atribuído pelo professor. Embora seja fácil de implementar com qualquer linguagem de programação, esta abordagem representa um desafio mais significativo para as aplicações web. Estas combinam várias linguagens (geralmente HTML, CSS, e Javascript), devem aceitar pequenas variações visuais em vez de esperar uma duplicação exacta da interface de referência, e processam eventos numa sequência aleatória em vez de uma ordem pré-definida do fluxo de dados, ao contrário dos exercícios de programação baseados em texto. Esta dissertação propõe um pacote de npm para facilitar a integração em ambientes virtuais de aprendizagem para avaliar automaticamente aplicações introdutórias de programação web. Há três aspectos significativos considerados para esta ferramenta. Um algoritmo de correspondência de interface web para avaliação de interface, comparando interfaces de estudantes com interfaces de referência, permitindo pequenos desvios um do outro. Testes unitários para avaliação funcional sem necessidade de identificação dos elementos da interface do utilizador, e um gestor de feedback incremental para ajudar o aluno a ultrapassar dificuldades enquanto resolve exercícios.

Para a sua validação, foi criada uma plataforma de demonstração para avaliar soluções fornecidas por dois grupos diferentes de alunos num cenário real. Para comparar o impacto do gestor de feedback no progresso da aprendizagem dos alunos, um dos grupos tem acesso ao gestor de feedback incremental, enquanto o outro não.

# Abstract

Introductory programming courses usually require students to solve various exercises in multiple languages to test their skills. Automated assessment comes in handy since there are several ways of achieving the same result, and teachers take less time to assess their students.

Over the years, several assessment tools using a black-box approach have been created to analyze code automatically in different programming languages. The black-box model employs a series of test cases to evaluate the software, comparing the student program results with the expected output assigned by the teacher. Although it is easy to implement with any programming language, this approach poses a more significant challenge for web applications. These combine several languages (usually HTML, CSS, and Javascript), must accept minor visual variations rather than expecting an exact duplicate of the reference interface, and process events in a random sequence rather than a data stream predefined order, contrary to text-based programming exercises. This dissertation proposes an npm package to ease the integration into virtual learning environments to assess introductory web programming applications automatically. There are three significant aspects considered for this tool. A web interface matching algorithm for interface evaluation by comparing student interfaces with reference interfaces, allowing minor deviations from each other. Unit testing for functional assessment without requiring user interface elements identification. And an incremental feedback manager to help the student overcome difficulties while solving exercises.

For its validation, a demo code playground was created to evaluate the solution provided by two different groups of students in a real scenario. To compare the feedback manager's impact on the student's learning progress, one of the groups has access to the feedback manager, while the other doesn't.

**Keywords:** Automated Assessment; User Interface Testing; Web Application Evaluation; Feedback Manager

**ACM Classification** — Applied computing → Education → Interactive learning environments; Information systems → World Wide Web → Web interfaces

# Agradecimientos

I would like to begin by expressing my gratitude to my supervisor, Professor José Paulo Leal, for all the help, availability, and tips throughout this work.

A special thanks go to Professor Ricardo Queirós for his assistance in organizing the validation process, by making available all the necessary resources and space for conducting it.

I would also like to extend my appreciation to my parents and my girlfriend for their constant support, encouragement, and belief in me. Their love and motivation have been a constant help during the most challenging moments.

Lastly, I would like to acknowledge all the participants who took part in the validation process for their help in gathering valuable insights and feedback.

Luís Costa

*“Compare yourself to who you were yesterday,  
not to who someone else is today.”*

Jordan B. Peterson

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem . . . . .	1
1.2	Objectives . . . . .	2
1.3	Document Structure . . . . .	2
<b>2</b>	<b>State of the Art</b>	<b>4</b>
2.1	Dynamic Assessment . . . . .	6
2.2	Static Assessment . . . . .	6
2.3	Interface Assessment . . . . .	6
2.4	Functional Assessment . . . . .	7
2.5	Incremental Feedback . . . . .	9
<b>3</b>	<b>Tool Design</b>	<b>11</b>
3.1	Proposal . . . . .	11
3.1.1	Use Cases . . . . .	12
3.1.2	Activity Diagrams . . . . .	12
3.2	Architecture . . . . .	15
3.2.1	Evaluator . . . . .	16
3.3	Integration . . . . .	17
<b>4</b>	<b>Implementation</b>	<b>18</b>
4.1	Exercise Manager . . . . .	18
4.2	Request Validation . . . . .	19
4.3	Syntax Validation . . . . .	20
4.4	HTML and CSS Processing . . . . .	22
4.4.1	Browser Emulation . . . . .	22
4.4.2	Element Extraction . . . . .	23
4.4.3	Element Relationships . . . . .	24
4.5	Functional Testing . . . . .	25
4.6	Feedback Manager . . . . .	26
4.7	Webpal API . . . . .	26
<b>5</b>	<b>Proof of Concept</b>	<b>28</b>
5.1	Main Page . . . . .	28
5.1.1	Feedback Log . . . . .	29
5.2	Updated Page . . . . .	30
5.2.1	Feedback Log . . . . .	30
5.3	Sidebar . . . . .	31



5.3.1	Exercise List . . . . .	31
5.4	Backoffice . . . . .	35
5.5	Log Files . . . . .	35
5.5.1	Log File Parameters . . . . .	36
<b>6</b>	<b>Validation</b>	<b>38</b>
6.1	Experiment . . . . .	38
6.1.1	Definition of Instruments . . . . .	40
6.2	Results . . . . .	41
6.2.1	Hypothesis Test . . . . .	42
6.2.2	Correlation of Data . . . . .	44
6.3	Discussion of Results . . . . .	46
<b>7</b>	<b>Conclusion and Future Work</b>	<b>48</b>
	<b>References</b>	<b>50</b>
<b>A</b>	<b>Installation Guide</b>	<b>53</b>
A.1	Requirements . . . . .	53
A.2	Installation . . . . .	53
A.3	Importing . . . . .	54
A.4	Troubleshoot . . . . .	54
<b>B</b>	<b>User Guide</b>	<b>55</b>
B.1	Importing Exercise Related Files . . . . .	55
B.2	Executing Webpal API Functions . . . . .	56
<b>C</b>	<b>Stress Test</b>	<b>57</b>
<b>D</b>	<b>Extracted Data</b>	<b>59</b>
D.1	Auxiliar Tables for Hypothesis Tests . . . . .	59
D.2	Hypothesis Testing Code . . . . .	61
<b>E</b>	<b>Questionnaire Structure and Data</b>	<b>64</b>
E.1	Questionnaire Structure . . . . .	64
E.2	Individual Responses - Tables . . . . .	66
E.3	Graphs . . . . .	67
<b>F</b>	<b>Correlation Matrix Generation</b>	<b>70</b>

# List of Figures

1.1	Publications on automatic evaluation with trend line [26]	2
3.1	Webpal Use Cases	12
3.2	Create Exercise Activity Diagram	13
3.3	Delete Exercise Activity Diagram	14
3.4	Get Exercise Activity Diagram	14
3.5	Evaluate Attempt Activity Diagram	15
3.6	Architecture Component Diagram	16
5.1	Webpal Main Page	29
5.2	Webpal Feedback Log	29
5.3	Webpal Updated Page	30
5.4	Webpal Feedback Log Updated	30
5.5	Webpal Sidebar	31
5.6	Webpal Backoffice	36
6.1	Gender Distribution - Control Group	39
6.2	Gender Distribution - Experimental Group	39
E.1	Form - 1st Part	64
E.2	Form - 2nd Part	65
E.3	Form - 3rd Part	65
E.4	Were the instructions for using the Webpal demo clear and easy to understand? - Control Group	67
E.5	Were the instructions for using the Webpal demo clear and easy to understand? - Experimental Group	67
E.6	How would you rate the difficulty of the exercises? - Control Group	68
E.7	How would you rate the difficulty of the exercises? - Experimental Group	68
E.8	Was the feedback provided by Webpal helpful for improving your code? - Control Group	68
E.9	Was the feedback provided by Webpal helpful for improving your code? - Experimental Group	69
E.10	Did you enjoy using Webpal for the coding exercises? - Control Group	69
E.11	Did you enjoy using Webpal for the coding exercises? - Experimental Group	69

# List of Tables

6.1	Response to the key questions - Average . . . . .	41
6.2	Response to the questions of the experiment . . . . .	43
6.3	Correlation Matrix (Part 1) - Control Group . . . . .	44
6.4	Correlation Matrix (Part 2) - Control Group . . . . .	44
6.5	Correlation Matrix (Part 1) - Experimental Group . . . . .	45
6.6	Correlation Matrix (Part 2) - Experimental Group . . . . .	45
D.1	Number of exercises started by each student - Experimental Group . . . . .	59
D.2	Number of exercises started by each student - Control Group . . . . .	59
D.3	Number of exercises completed by each student - Experimental Group . . . . .	59
D.4	Number of exercises completed by each student - Control Group . . . . .	60
D.5	Number of attempts by each student - Experimental Group . . . . .	60
D.6	Number of attempts by each student - Control Group . . . . .	60
D.7	Time spent by each student in exercise "Sum Two Numbers" - Experimental Group	61
D.8	Time spent by each student in exercise "Sum Two Numbers" - Control Group . .	61
E.1	Responses from the control group . . . . .	66
E.2	Responses from the experimental group . . . . .	66

# Listings

4.1	JSON File Example . . . . .	19
4.2	Functional Tests Example . . . . .	20
4.3	HTML Syntax Validation Example . . . . .	21
4.4	CSS Syntax Validation Example . . . . .	21
4.5	JS Syntax Validation Example . . . . .	22
A.1	Start Project . . . . .	53
A.2	Webpal Installation . . . . .	53
A.3	Importing Webpal . . . . .	54
A.4	Importing Webpal - ES 6 . . . . .	54
A.5	Installing Dependencies in Linux Server . . . . .	54
B.1	Importing Webpal . . . . .	55
B.2	Read Files Content . . . . .	55
B.3	Create Exercise Function . . . . .	56
C.1	Stress Test Source Code . . . . .	57
D.1	Hypothesis Test Source Code . . . . .	61
F.1	Correlation Matrix Source Code . . . . .	70

# Abbreviations and Symbols

VLE	Virtual Learning Environment
GUI	Graphical User Interface
CS	Computer Science
LMS	Learning Management System
JS	Javascript
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
DOM	Document Object Model

# Chapter 1

## Introduction

The internet has become an essential tool to provide resources for research and learning, allowing teachers and students to share and access information. With the growth of the internet in the early-mid 1990s, Virtual Learning Environments/Code Playgrounds appeared to withstand the education area, facilitating ways of sharing knowledge, assessing, and giving feedback to students [25]. These environments can also be understood as e-learning spaces that provide tools for remote education [6]. This type of platform was of great use during the recent COVID-19 outbreak improving classroom activities as well. E-learning involves using the internet and other technologies to create learning materials, instruct students, and manage courses within an organization [3].

One of the significant advantages of e-learning is its flexibility and self-paced nature, which allows students to manage their time better.

Since it saves time and money, it increases access to higher education and can reveal a considerable help for slow learners as they can take as much time as necessary to read and participate in activities [33].

The introduction of automated assessment tools into virtual learning environments has opened up new possibilities in computer science education, particularly in programming courses. These tools offer means of evaluating student solutions to programming exercises, presenting numerous advantages and opportunities for educators and learners.

Figure 1.1 shows the increase in publications of automatic evaluation by year and its trend line, displaying the rise in interest in automated assessment.

### 1.1 Problem

It can be very challenging to manually assess students' programming skills since there are several ways to accomplish the same goal. Additionally, because there are usually so many programming assignments needed to practice programming, their assessment burdens the teachers [2].

Web applications typically don't rely solely on one programming language, applying different programming and markup languages to achieve the desired result - a web application coded in HTML, CSS, and JavaScript. Also, in a web application, events are executed in random order

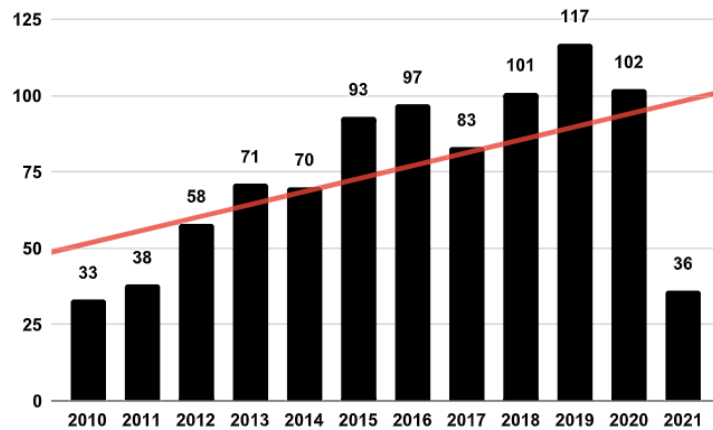


Figure 1.1: Publications on automatic evaluation with trend line [26]

instead of processing data following a predetermined format. Because of these unique features, black-box methodologies cannot be used to analyze Web interfaces and GUIs [20, 2].

The majority of programming languages evaluation tools employ the black-box model, which uses a series of test cases to evaluate the student's attempt. This model compares the student input/program with the expected output assigned by the teacher. Although it is easy to implement and compatible with any programming language, this approach evaluates the outcomes of the execution rather than the execution itself [20]. Also, this model expects a determined format for the data stream, which doesn't happen on web technologies, making it an unviable method.

## 1.2 Objectives

The objective of this work revolves around the development of a tool aimed at automating the assessment of simple web applications. This tool possesses the following capabilities:

- Evaluation of interfaces constructed with HTML and CSS, allowing minor deviations.
- Assessment of code functionality, particularly in JavaScript, without requiring user interface elements identification.
- Generation of constructive and non-repetitive feedback to assist learners, without explicitly providing the solution.

Overall, it is expected that this tool will serve as a source of motivation for students studying web programming languages, due to its rapid evaluation and feedback generation.

## 1.3 Document Structure

This document is structured into 7 chapters. Chapter 1 serves as the introduction, providing context for this dissertation. It explains the concepts of virtual learning environments and code playgrounds, identifies the problem at hand, and outlines the objectives to be achieved. Chapter 2 dives

into the state-of-the-art, exploring relevant literature and previous implementations and prototypes related to interface evaluation, functionality assessment, and feedback generation. In Chapter 3, the proposed solution is presented, detailing the design and architecture of a tool aimed at addressing the identified problem. Chapter 4 focuses on the development process, providing insights into the tool's development, the technologies employed, and the communication between code playgrounds and the tool. Chapter 5 provides a detailed description of a proof of concept created specifically to test Webpal functionalities and integration in a replicated code playground. Chapter 6 provides an in-depth explanation of the evaluation of the tool's effectiveness. It includes details of the conducted experiment and the methodology used. Finally, Chapter 7 offers an overview of the work accomplished and outlines future steps and potential areas of improvement. In addition, the document includes appendices that provide supplementary information on specific aspects of the project. These appendices cover topics such as data analysis, installation guidelines, usage instructions, and the source code used for various analyses. It is worth noting that Chapters 2, 3, and 4 expand upon the content published in the paper titled *Automated Assessment of Simple Web Applications*[5] which was presented at ICPEC 2023.



## Chapter 2

# State of the Art

In this dissertation, a tool designed to evaluate simple web applications built with HTML, CSS, and JavaScript is presented. To the author's knowledge, there don't appear to be any existing tools that offer automated assessment in this specific context. Hence, this analysis concentrates on tools carrying features that are relevant to this project, with a focus on interface and functional evaluation, and incremental feedback management. It's worth mentioning that the existing tools referenced in the literature on interface assessment seem to have emerged in the early 2000s, focusing mainly on tools developed in Java.

Most teachers would agree that it's difficult to automate the assessment of every aspect of programming code, which still holds nowadays. However, from their perspective, the tools have improved significantly when compared with the oldest ones [26]. Douce, Livingstone, and Orwell [8] reviewed various automated assessment tools and identified three generations of automated assessment systems:

- The first-generation systems were the first attempts to automate testing.
- The second generation was composed of a command line and GUI.
- The third generation happened when the tools became available on the web and could be accessed directly by the students.

The authors concluded by pointing out some advantages and disadvantages of these systems. On the positive side, since assessing programming assignments is slow and difficult, these tools can relieve some work on the teachers, allowing the teacher's time to be spent on other aspects like clarifying CS and programming concepts [8]. Also, computers are less likely to fail if configured correctly, while humans are faulty. On the other hand, there are many restrictions on what can be assessed automatically, and the feedback provided by these tools needs to be revised for educational reasons [26].

In 2010, Ihantola et al. [15] presented a literature review of the current systems for the automatic assessment of programming assignments from 2006 to 2010. The authors recommended that

new automated assignment systems should explain clearly how the tool functions and that the security of these systems should get more consideration. They also suggested that automated assessment tools should be open-source to avoid scattering, emphasized the need to integrate automated assessment into virtual learning environments, and identified the assessment of web applications as an area for future research.

Traditional programming exercises are essential for students to learn the basics of programming but are not everything since software projects involve many phases in their pipeline. In a recent literature review, Paiva et al.[26] identified some skill domains different from traditional programming that can be fundamental for CS students:

- **Visual programming** — introduces programming effectively at various levels [19]. Visual programming will be helpful most of the time since it will increase student engagement and retention of information [24];
- **Alternative programming assignments** — comprehend and debug other people’s code, for example;
- **System administration** — instead of looking at one file or folder as in the case of the typical automated assessment, evaluating a system administration task may require inspecting the complete operating system;
- **Formal languages and automata** — a core component of CS. Give feedback on conversions from natural language, for example;
- **Software modeling** — models can describe and express the most crucial aspects of complicated systems by abstracting some details [24];
- **Software testing** — learners could exhibit improper programming techniques [4]. With this approach, students are challenged to create tests that verify a program’s correctness, and their success is evaluated afterward;
- **Database** — An everyday chore for developers is querying and editing databases, which is also the cause of significant performance and security issues. Therefore, it’s crucial to give students plenty of practice utilizing them;
- **Web development** — it usually makes part of CS courses, and there are many online courses available, so there’s much interest in it. In addition to verifying the source code, evaluating a web project includes testing the browser’s user interface, simulating actions on the content, and examining how those actions affect the browser’s state. Assignments may make use of a variety of services, such as web servers and databases, which may be running concurrently;
- **Parallelism and concurrency** — since more than one process is running simultaneously, it’s hard to assess manually;

- **Mobile development** — due to their frequently intricate user interfaces and sophisticated functionality, such as the ability to receive data from sensors and connect to external services, mobile apps are difficult to create and, because of these reasons, their evaluation is commonly done manually;
- **Computer graphics** — since the result (picture or animation) frequently has several correct alternatives, automated evaluation is presented with significant difficulty.

To automatically assess assignments, it's necessary to define the technique(s) to delineate what to consider, how to create feedback, and security considerations.

## 2.1 Dynamic Assessment

Dynamic assessment can be understood as assessing a program by executing it. It's essential to have a secure environment (sandbox) that doesn't interfere with the host's computer integrity. On that matter, Reek proposed a system called TRY that creates a wrapper directory that the application sees as the file system's root, preventing access to other directories [30, 26]. With this type of assessment, it's possible to check the program functionality by comparing the result with some test data sets, efficiency by monitoring execution-related activities such as execution time and memory usage, and testing skills by asking students to provide test data sets along with their programming assignment, and evaluating the quality of the test data as well [2].

## 2.2 Static Assessment

Static assessment, in opposition to dynamic, means assessing programming code without executing it. Adequate software code doesn't just guarantee proper execution, there are still other critical elements to consider, such as coding style and code smells, to check if the code contains the correct syntax. Software metrics can be computed statically, such as the number of lines of code or checking the number of variables and operators in a program, giving insights to the teacher about the structure of the code. It's also conceivable to assess the design of a program by comparing it with a determined interface or structure and by checking for plagiarism in the source code [2, 1, 29, 36]. Jackson and Usher developed a software called ASSYST that enables teachers to choose weights for each test (and component), consult a full report of the assessment through a GUI, and evaluate a variety of static measurements, including CPU time, code complexity, and code style [16, 26].

## 2.3 Interface Assessment

Several techniques for testing GUIs include code structure comparison, web interface matching algorithms, computer vision, or phishing detection methods [28].

In 2006, Gray and Higgins [14] proposed an approach to grade GUIs, by analyzing the **hierarchical relationship** between interface components and the use of this approach, they were able

to retrieve data from the graphical components rather than relationship estimations. This method takes advantage of object-oriented programming languages due to dynamic class loading, which contains information about the interface's object structure. Then, a file explains what needs to be performed and on what object to test and grade assignments.

Štěpánek and Šimková [37] wrote a report describing three algorithms to compare the inner structure of HTML trees. The first algorithm counts the **number of occurrences of each element in a tree**, producing a set of key-value pairs representing the element name and the count of element occurrence, respectively. The second algorithm gets the **sequence of nested elements for each node** and calculates a ratio that represents a percentage of reference paths that can be found in the compared interface. The third algorithm finds the possible **largest subtree in both interfaces** and calculates a ratio between them. The authors concluded that the algorithms could be beneficial in detecting duplicate interfaces (phishing) and for education.

Thackston [35] stated in 2020 that there are only a few tools to automate the grading of CS assignments. Therefore, the author proposed a method to automate assessment using **XPath queries** that first receives an HTML file as a reference, parse it into a DOM, and renders it as a tree. Then, the teachers can choose the node(s) to assess by selecting them on the tree. The author concluded that XPath has the potential for automatic evaluation, however, it presents a lack of flexibility, incomplete coverage, and difficulty grading when the assignment is partially correct.

Primo and Leal [20] proposed a **web interface matching algorithm** that relies on element mapping to identify and compare elements from different interfaces. This method obtains original properties from the DOM API and derived properties reflecting spatial relations between elements. A comparison between the algorithm's evaluation and that of a panel of experts revealed a connection with 99% certainty. The authors suggested some refinements to the algorithm and planned its extension to evaluate interfaces visually and functionally and create an incremental feedback manager.

## 2.4 Functional Assessment

Javascript is a flexible and expressive language, but being dynamic and event-driven makes it hard to analyze and test [22].

English [10] was one of the first to describe a system, called JEWEL, for automated assessment. JEWEL is a Java library consisting of packages that teach students how to create Java GUI-based apps. It features an **event loop that handles a stream of characters**, similar to text-based software produced by controls. The evaluation of the application is done in parallel with the program's execution through threads. However, this system has several flaws that make it feasible to evaluate the application's functionality but not the quality of the user interface.

Feng and McAllister [11] developed a grader to **assess multi-window Java applications automatically**. The authors emphasize that their tool differs from the others because students can: build applications with various windows; choose types of objects of their preference to satisfy the needs of the assignment; choose the object's position and layout; and choose labels for the objects.

The automated evaluation is based on a **test plan**; however, this plan only performs the functional assessment.

An application was proposed by Sztipanovits, Qian, and Fu [34] to assess web applications. One of the main requirements of this implementation is the **possibility of automatically evaluating multiple submissions with various interfaces**. An excel document defines the inputs and test cases, and a list of URLs hosting each submission must be provided with the interfaces to assess. This solution ignores the web design or spatial relations between the HTML elements, evaluating only the functionality of the submissions.

The js-assess online playground [17] is a tool that automatically assesses Javascript programming exercises. It is a serverless tool that runs inside a Javascript engine and combines an online editor with libraries to evaluate functionality, style, programming errors, and software metrics. Due to the lack of a server, feedback cannot be stored, making it useful only for self-studying. This tool has a collection of assignments available and is implemented in pure HTML and Javascript, facilitating the integration of js-assess into any web page. The authors highlighted the importance of researching how Javascript should be taught, the libraries that should be used, and where Javascript should be introduced to students.

The SymJS [21] is a framework for automated assessment of Javascript web applications. It comprises three main components: the front end obtains Web pages, parses them, and stores data locally. The middle end creates, schedules, and manages event sequences. And the back end contains a **symbolic virtual machine**. The back-end interprets Rhino [7] Icode, forks states, and manages the states.

Mirshokraie [23] proposed an automated technique to generate test cases for individual Javascript functions and DOM event sequences. This report focuses on answering two questions: **how can test cases be generated efficiently for JS web applications**, and **how can the effectiveness of the tests be assessed**. The former consists of a three-step approach: dynamically explore the application using a method to deduce a test model. After that, generate test cases for JS functions and DOM event sequences, and finally, automatically create test assertions using mutation testing. The latter uses a technique to analyze the application statically and dynamically. For that, the author proposes a set of specific mutation operators that identifies common JS programming errors to lessen the scope of the mutation process so that only necessary code for the application is analyzed through variable usage frequency, dynamic invariants, etc. The study results showed that the generated tests covered an average of 68.4%, demonstrating that this method is not particularly interesting in generating tests for web applications.

The WebWolf [32] is a framework developed for automatically evaluating introductory web programming exercises. It can **load web pages, find and inspect elements, click links, and perform unit testing**. This tool lets teachers create JUnit-like Java programs to test websites. Test methods are written to simulate user inputs on the page, and assertions are made to compare the student's result with the expected output. WebWolf requires knowledge of Selenium's WebElement and the specification of ids on the web elements for the assignments to be testable, which is not desirable. The authors concluded that this framework significantly reduced the grading time

compared to manual grading and saw it as a promise for automated assessment.

Mesbah [22] described the recent advances in analysis and testing techniques for JS web applications. The report explored empirical studies to understand **how JS applications are being used nowadays by web applications and testing techniques from industrial tools to more advanced methods**. Test oracles automate the examination of program correctness in testing scenarios, test adequacy evaluates the quality of a given web application (code coverage, mutation testing, etc.), failure handling, and programmer support that describes various approaches to facilitate the developer's work.

Ask-Elle [13] is a tool designed for assessing programming exercises related to Haskell, a functional programming language. It has the ability to verify the correctness of incomplete programs and provide students with helpful hints. This work aimed to make a contribution by mixing automated feedback with a specified set of programming exercises, solutions, and properties, all curated by teachers. The testing approach undertaken employs two key strategies: model tracing testing and property-based testing. The model tracing programming strategy was developed from a set of model solutions for an exercise. It's used to monitor student progress and generate feedback. This approach can be conceptualized as a series of fundamental steps that reframe model tracing as a parsing problem. Property-based testing comes into play when none of the model solutions can be identified. The properties of an exercise return a permutation of the input list (explicitly defined in the configuration file), and these are tested on the student's program. The authors assert that teachers require control over their learning environments, and as such, these environments need to be adaptable. In this matter, the tool was designed to provide flexibility, allowing teachers to integrate new exercises and modify or specialize feedback as required. The authors employed the *QuickCheck* library, a resource that allows for the specification and testing of properties, along with defining custom random input generators. Haskell's default compiler, *GHC*, was also used to test these properties.

Indirectly related to functional assessment, Peveler et al. [27] developed a method that **uses Docker to create containers for each student assignment submission** that builds their code with any dependencies and then runs it. Creating various containers simultaneously allows concurrent assessment and for each network to be isolated from the others. For the automated evaluation, the authors used the Selenium framework that interacts with the student's page and executes tests. This solution also allows teachers to assess the assignment manually by running the container locally.

## 2.5 Incremental Feedback

There isn't a lot of information on systems implementing incremental feedback managers.

ProtoAPOGEE [12] is an automated grading system to guide students and elevate faculty productivity. It **guides failed unit testing through informative and iterative feedback**. This tool consists of three modules. Project specification to specify requirements, grading policies, and test cases. Automatic grading runs a browser in the background and evaluates the student

project with the previously established test cases. A grading report viewer displays a grading summary of all the requirements set by the professor. This tool requires the teacher to understand the Ruby programming language to write scripts. At the time, one difficulty was providing detailed feedback to students that needed a teaching member to write long descriptive messages about the errors. ProtoAPOGEE generates an animation demo for each failed test case, providing a step-by-step guide on how the student failed a specific requirement. Professors can also prepare textual hints to help the students during the assignment. This tool can provide feedback at two levels: a summary report and detailed feedback for each requirement and test case.

As referenced in the previous subchapter, Ask-Elle [13] is a tutor designed for Haskell programming exercises. One of its standout features is the ability to adapt feedback. This allows an educator to personalize the feedback received by students through the annotation of model solutions. To achieve this, a unique type of source code comment known as *pragma* is utilized. Feedback messages are structured hierarchically, anchored on the abstract syntax tree of the model solution. This unique arrangement allows educators to fine-tune the tool, thus enhancing the granularity and level of detail in the feedback provided.

Kazerouni et al. [18] prepared an article to assess the expense of conducting **mutation analysis** on introductory programming projects so that it can be implemented to give students quick incremental feedback about their tests. This work was not further studied in this context since the authors concluded that this method is **inadequate for small programs developed for introductory courses** due to its cost.

## Chapter 3

# Tool Design

This chapter provides the specification of a new tool called Webpal, an acronym for Web Programming Assessment for Learning. The initial segment provides an overview of the tool (3.1), encapsulating a description of its expected functionality, core components, as well as features that should be implemented. This is followed by the introduction of a set of use cases (3.1.1) that the tool is designed to comply with, accompanied by the respective activity diagrams. Subsequently, we dive into the specifics of the tool's architecture (3.2), along with an exploration of how Webpal will integrate (3.3) with code playground environments. These different elements together create a full picture of what Webpal aims to be.

### 3.1 Proposal

This subsection presents the proposal for Webpal, an automated assessment tool designed for automated assessment and feedback generation on simple web programming assignments. Webpal aims to simplify the process of assessing web programming exercises within virtual learning environments and code playgrounds. This tool wants to enhance the learning experience, provide timely guidance, and relieve the burden of manual assessment for educators.

The core components of Webpal revolve around exercise management, validation, assessment, and feedback generation. These components work together to ensure a reliable assessment process:

- **Exercise Management:** Webpal provides a flexible exercise management system, allowing teachers to create and delete exercises, retrieve exercise data, test data, and assignment details. The exercise data contains the problem statement, expected solutions, and functional evaluation criteria.
- **Validation:** A validation process to ensure the correctness and integrity of exercise submissions. It employs a predefined exercise schema to validate the structure and properties of exercise data. Additionally, Webpal performs syntax validation to identify and flag potential syntax errors in student submissions.



- **Assessment:** The assessment component of Webpal compares the submitted student attempts against the expected solutions, runs the provided test data, and generates feedback. The evaluation process takes into account both correctness and adherence to best practices.
- **Feedback Generation:** Another key aspect of Webpal is its ability to generate incremental and non-repetitive feedback for students. Webpal analyzes the specific errors in the student's code and generates targeted feedback based on these findings. By leveraging the evaluation results and comparing them to previous submissions, Webpal ensures that students receive feedback that addresses their individual learning needs, encourages improvement, and avoids repetitive comments.

### 3.1.1 Use Cases

This subsection outlines the primary use cases for Webpal. VLE/Code Playground is identified as the sole actor.

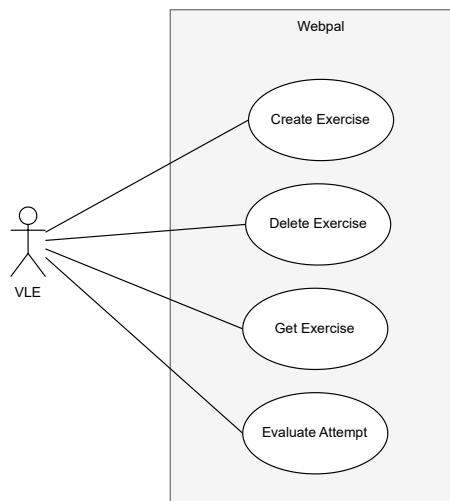


Figure 3.1: Webpal Use Cases

As depicted in Figure 3.1, there are four main use cases: Creating an Exercise, Deleting an Exercise, Retrieving an Exercise, and Evaluating an Attempt. Each of these use cases is further detailed through an associated activity diagram, which will be presented and explained in the following subsection.

### 3.1.2 Activity Diagrams

Each activity diagram features two swim lanes, representing the VLE/Code Playground and Webpal. All actions are initiated by the VLE, and upon successful completion, the final state is also on the VLE swim lane.

### 3.1.2.1 Create Exercise

As illustrated in Figure 3.2, the creation of an exercise begins when the VLE transmits the exercise data to Webpal. This data includes the solution code, a set of functional tests, and a description of the assignment. The solution code must adhere to a predefined JSON schema. Webpal initiates the process by verifying that the code conforms to this schema and subsequently performing syntax validation to catch any potential errors within the assignment code. Provided there are no issues during these stages, the exercise is saved to Webpal, and an exercise ID is returned to the VLE.

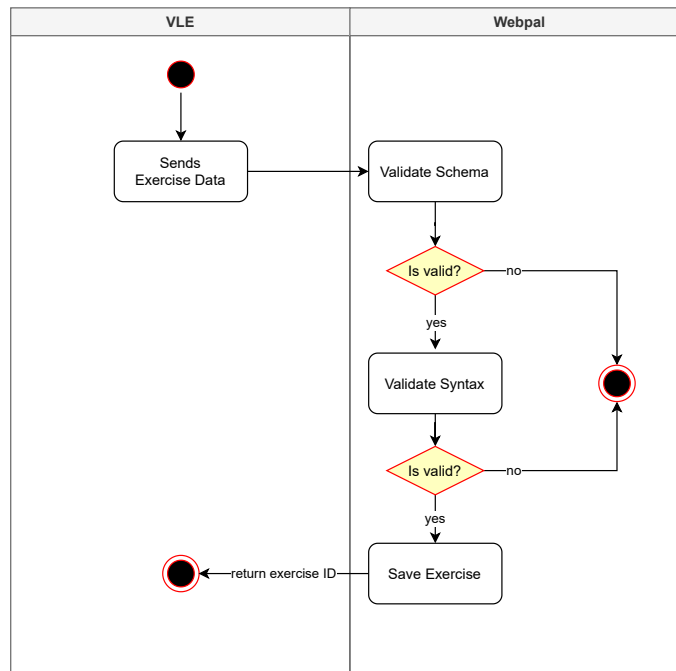


Figure 3.2: Create Exercise Activity Diagram

### 3.1.2.2 Delete Exercise

As shown in Figure 3.3, the deletion of an exercise commences with the VLE transmitting the exercise ID to Webpal. Webpal verifies the existence of the exercise in its storage. If it exists, the exercise is removed, and a confirmation message indicating successful deletion is returned to the VLE.

### 3.1.2.3 Get Exercise

As presented in Figure 3.4, to retrieve an exercise, the VLE sends the exercise ID to Webpal. Webpal checks its storage for the existence of the exercise. If found, Webpal retrieves the exercise data and returns it to the VLE.

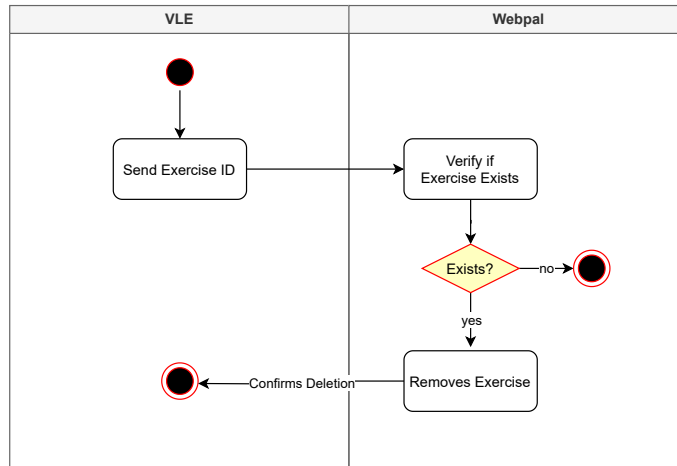


Figure 3.3: Delete Exercise Activity Diagram

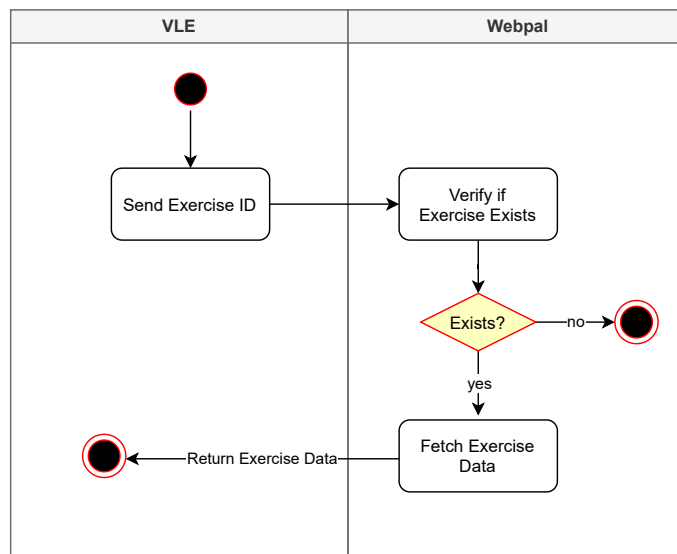


Figure 3.4: Get Exercise Activity Diagram

### 3.1.2.4 Evaluate Attempt

Figure 3.5 delineates the process of evaluating an attempt, which represents the core functionality of Webpal and is slightly more complex than the previous use cases. The process starts with the VLE sending the attempt data to Webpal, which includes the exercise ID, the code attempt, and an array of feedback strings generated from previous attempts. Webpal first checks its storage for the exercise's existence. If found, Webpal validates the schema and syntax of the attempt. If the syntax is not validated, feedback regarding the syntax errors is generated. In the absence of syntax errors, Webpal proceeds to evaluate the functionality of the code, including the analysis and comparison of interfaces. Following this step, feedback is generated and returned to the VLE.

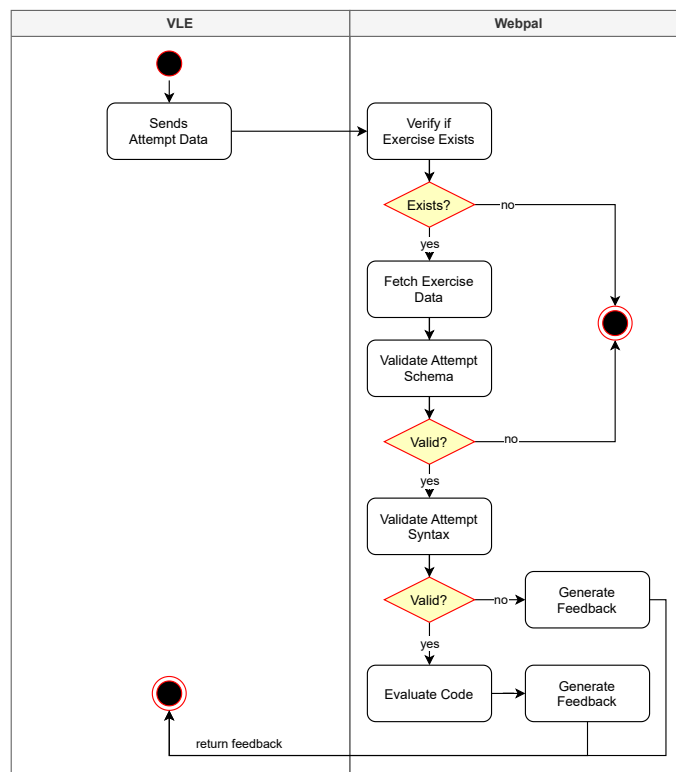


Figure 3.5: Evaluate Attempt Activity Diagram

## 3.2 Architecture

This section presents the architecture of Webpal through a component diagram. The section begins with a presentation of the UML diagram, followed by a detailed discussion of each component.

As illustrated in Figure 3.6, Code Playgrounds and VLEs comprise two sub-components:

- **Client** — This component serves as the interface where educators and students can interact with the code playground. The Client operates independently of Webpal, communicating solely with internal VLE components.

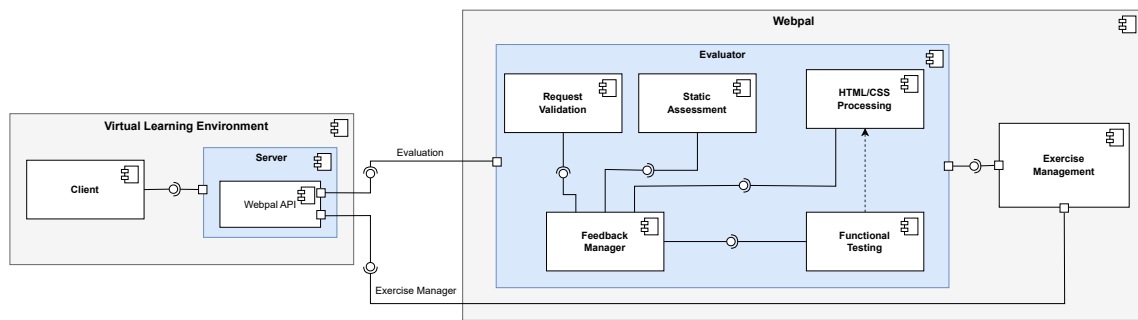


Figure 3.6: Architecture Component Diagram

- **Server** — Acting as an intermediate between the Client and Webpal, this component implements Webpal and employs the Webpal API for interactions. The Webpal API enables the Server to execute exercise and evaluation instructions within Webpal.

The Webpal component also contains two sub-components that manage evaluations and exercises:

- **Evaluator** — This component embodies the main function of this tool. Triggered by the Webpal API on the Server, the Evaluator carries out request validation, static assessment, interface processing, functional testing, and feedback generation through the feedback manager. All sub-components within the Evaluator serve the feedback generation process, supplying feedback messages to the Feedback Manager, which then selects the most suitable feedback to return to the code playground.
- **Exercise Management** — This component is responsible for the creation, storage, deletion, and retrieval of web programming assignments created by the code playground.

### 3.2.1 Evaluator

This subsection delves deeper into the functionalities and roles of the evaluator's sub-components within the Webpal architecture.

- **Request Validation** — Exercise and student attempt data are required to adhere to a structured JSON format. Validation against a JSON schema ensures data standardization and facilitates interactions.
- **Static Assessment** — This preliminary step in the Webpal evaluation process involves an analysis of the code to identify potential syntax errors, such as unclosed HTML and CSS tags, undeclared JavaScript variables, and so on, before proceeding to execute the code and functional tests.

- **HTML/CSS Processing** — This sub-component is responsible for the extraction and analysis of HTML and CSS elements from both an exercise and an attempt. It verifies relationships between these elements, which are later used to map the elements when conducting functional tests.
- **Functional Testing** — At this stage, unique IDs from the attempt elements are mapped to the reference ones, and then functional tests are performed on the functional code of the attempt.
- **Feedback Manager** — Acting as the central hub of communication, the Feedback Manager receives information from all other sub-components. It is responsible for processing this information and determining which feedback is most suitable to return to the VLE.

### 3.3 Integration

A primary goal of the Webpal tool is to provide a simple experience for users and to ensure ease of integration with a variety of code playgrounds and virtual learning environments. To support this, Webpal is designed as a Node Package Manager (*npm*) package. *npm* is a widely adopted package manager for the JavaScript runtime environment Node.js, which allows developers to install and manage software dependencies. Leveraging *npm* allows Webpal to be easily included in any JavaScript-based project.

The decision to implement Webpal as an *npm* package has various benefits:

- **Portability:** As an *npm* package, Webpal can be installed and utilized in virtually any environment where Node.js can run. This makes it accessible to a wide range of code playgrounds and virtual learning environments, regardless of their underlying technology stack.
- **Ease of Integration:** *npm* facilitates the integration of packages into projects by managing dependencies and versions. This means that integrating Webpal into a project is as simple as running a single command in the terminal.
- **Version Control:** *npm* provides version control for packages, ensuring that users can choose to use a specific version of Webpal, or always stay updated with the latest version. This flexibility allows developers to select the version of Webpal that best fits their needs.

## Chapter 4

# Implementation

This chapter showcases the implementation of Webpal, according to the design outlined in the previous chapter. Utilizing Node.js as the development platform, Webpal was designed to function as an *npm* package. The chapter details the implementation of crucial features such as:

- The works of the exercise management component
- How are the attempt and reference files structure validated;
- What's the methodology used for the syntax validation;
- How are HTML and CSS files processed and assessed;
- How functional assessment is executed;
- How the feedback manager works;
- Communication process between code playground and Webpal

### 4.1 Exercise Manager

The Exercise Manager is responsible for managing all the exercises in the system. This class is responsible for the creation of new exercises, reading existing ones, updating the status of an exercise, and deleting an exercise when necessary. Each exercise is stored locally, in Webpal, to facilitate the integration into an existing system without the need to create or adapt a database to store them. This component allows a range of operations:

- Creating a new exercise with unique identifiers, details, and the code itself.
- Saving the necessary details of an exercise into respective files within the Webpal local storage.
- Reading all previously stored exercises from the local storage and loading them back into the system.

- Deleting specific exercises based on unique identifiers. This process removes data from both the internal system and the local storage.
- Fetching all details of a specific exercise based on its unique identifier.
- Retrieving specific types of data, such as tests or assignments, of a particular exercise, using its unique identifier.
- Get all stored exercises in the system.

When creating exercises, it's necessary to provide the reference code and functional tests. The reference files are stored in a JSON file consisting of an array of objects, with each object containing two attributes - filename and code. An example of the structure is available at [4.1](#).

```
1  [
2    {
3      "filename": "index.html",
4      "code": "<!DOCTYPE html>
5      <html lang='en'>
6      <head>
7        <link rel='stylesheet' href='style.css'>
8        <title>Assignment</title>
9      </head>
10     <body>
11       <div>Hello World</div>
12     </body>
13   </html>"
14   },
15   {
16     "filename": "style.css",
17     "code": "div {color: red}"
18   }
19 ]
```

Listing 4.1: JSON File Example

The functional tests must be a Javascript file that imports the Chai/Mocha libraries. A simple functional test file is presented in [4.2](#).

## 4.2 Request Validation

Each time an exercise is created, or a student submits an attempt, Webpal must verify whether the input data aligns with a predefined structure. To simplify the integration with code playgrounds, all files that are processed through Webpal can be condensed into a single JSON file. This approach eliminates the need to create a separate file for each component of an exercise and compress them for Webpal to decompress upon receiving the data. To ensure compliance with the requisite file format, a JSON schema was established. Both exercise and attempt data must be presented as



```
1  const assert = chai.assert;
2
3  describe('Test app', function() {
4    const teste = document.getElementById('testDiv');
5
6
7    it('test starts with 0', function(done) {
8      assert.equal(0, teste.innerHTML);
9      done();
10   });
11 }
```

Listing 4.2: Functional Tests Example

an array of objects. Each object should contain two properties: **filename** and **code**. **Filename**, a string, denotes the designated name for a specific file and must include the extension for subsequent identification. **Code**, also a string, corresponds to the code itself that matches the specified filename within the same object.

In order to validate the schema, we employed an established *npm* package: **ajv**<sup>1</sup>. **ajv** holds a renowned reputation as a prominent JSON schema validator. Its creators promote it as the fastest JSON validator for Node.js and browsers.

### 4.3 Syntax Validation

Syntax validation is an essential preliminary step in identifying potential code anomalies. By analyzing the syntax of the code, we can detect errors in the initial stages, eliminating the need to execute the entire code and functional tests. This also facilitates the provision of informative feedback about issues within the code, such as missing variable declarations and unclosed tags. Given that Webpal handles three different file types (HTML, CSS, and JavaScript), each requires a distinct approach for syntax validation. For HTML files, we utilized the *npm* package **html-validator**<sup>2</sup>. This package validates HTML files using the Nu HTML Checker by W3 or html-validate for offline validation. An example of an HTML error identified by this package is displayed in listing 4.3.

For syntax validation of CSS files, we used the *npm* package **w3c-css-validator**<sup>3</sup>. This package utilizes W3C's public CSS validator service<sup>4</sup> for its operations. An example of an error is available at 4.4.

---

<sup>1</sup><https://www.npmjs.com/package/ajv>

<sup>2</sup><https://www.npmjs.com/package/html-validator>

<sup>3</sup><https://www.npmjs.com/package/w3c-css-validator>

<sup>4</sup><https://jigsaw.w3.org/css-validator/>

```

1   <footer>
2     <fieldset>
3       <p>Lorem ipsum dolor sit amet</p>
4       <legend>Consectetur adipiscing elit</legend>
5     </fieldset>
6
7     <main>
8       <blink>(c) 2018 Initech</blink>
9     </main>
10
11  </footer>
12  -----
13  error: Element <legend> must be used before <p>
14  in this context (element-permitted-order) at inline:4:6:
15    2 | <fieldset>
16    3 |   <p>Lorem ipsum dolor sit amet</p>
17  > 4 |   <legend>Consectetur adipiscing elit</legend>
18    |     ^^^^^^
19    5 | </fieldset>
20    6 |
21    7 | <main>

```

Listing 4.3: HTML Syntax Validation Example

```

1   .foo {
2     text-align:
3   }
4   -----
5   {
6     "valid": false,
7     "errors": [
8       {
9         "line": 2,
10        "message": "Parse Error"
11      }
12    ]
13  }

```

Listing 4.4: CSS Syntax Validation Example

Lastly, for JavaScript file syntax validation, we utilized the *npm* package **JSHint**<sup>5</sup>. **JSHint** is a community-driven tool designed with the intent to identify errors and potential issues within JavaScript code. Listing 4.5 shows an example of an error that JSHint is capable of pinpointing.

```
1   window.onload = function(){
2       let a;
3   }
4   -----
5   [
6       {
7           id: '(error)',
8           raw: "'{a}' is defined but never used.",
9           code: 'W098',
10          evidence: 'window.onload = function(){let a}',
11        }
12   ]
```

Listing 4.5: JS Syntax Validation Example

Webpal performs a static evaluation on all submitted files, generating an array of objects in the process. Each object contains a property named 'type', which can be either 'attempt' or 'solution', a 'file' property that holds the file name, and a 'messages' property, represented by an array of objects. Each object in this array represents an error found in the corresponding file. This array of objects, generated through syntax validation, is then passed on to the Feedback Manager. Then, the Feedback Manager selects the most relevant feedback to send to the student.

## 4.4 HTML and CSS Processing

The processes of extraction and analysis of HTML and CSS elements are executed independently. Initially, Webpal extracts elements from the exercise and the student's attempt, generating a separate list of elements for each. This list of elements can be perceived as a flattened tree, representing the elements in their order of appearance, while also storing their children within an element-specific property. This phase necessitates accessing the DOM to verify element properties, facilitating their comparison later on. However, since Node.js lacks a DOM, it was necessary to identify an alternative approach that would simulate a browser environment.

### 4.4.1 Browser Emulation

The evaluation of interfaces requires the emulation of web pages to discern the relationships and positions of elements. Two options that offer a simulation of the DOM in Node.js, JSDOM<sup>6</sup> and Cheerio<sup>7</sup>, were initially considered. However, as per their documentation, neither of these

<sup>5</sup><https://jigsaw.w3.org/css-validator/>

<sup>6</sup><https://www.npmjs.com/package/jsdom>

<sup>7</sup><https://www.npmjs.com/package/cheerio>

tools supports the simulation of element positions. A more effective approach turned out to be headless browsers, which operate a browser instance without a graphical interface and facilitate the extraction and manipulation of the DOM. Playwright<sup>8</sup>, a framework allowing web testing and automation, was selected to take this role. An alternative, Puppeteer<sup>9</sup>, was also evaluated but proved to be less suitable due to being older and lacking cross-browser support. To analyze the interfaces, Playwright requires a URL of the web page. A function to create routes dynamically was developed, which inspects the JSON file containing the web page files and serves them to the local host using the Express<sup>10</sup> *npm* package. Express is a widely-used package for Node.js that facilitates the creation and serving of routes. The implementation of this package allowed for effective communication between the Playwright framework and the locally served exercise and attempt files.

#### 4.4.2 Element Extraction

In order to retrieve the necessary information from the HTML files of the exercise and the student's attempt, a simple process is employed. This process ensures the extraction of the elements and their properties, which are stored in a format that allows for an effective comparison between the two sets of elements. The following is a step-by-step explanation of this process, which provides a full overview of how the HTML files are analyzed:

1. The process starts by creating two separate arrays to store the elements extracted from the exercise and the student's attempt. Each HTML file in the data set is processed one by one.
2. For every HTML file, a new browser instance is launched with Playwright. This instance opens a web page pointing to the HTML file served on the local server.
3. Identification of all the direct child elements of the body on the web page. Each of these elements is examined in detail to gather specific information.
4. For each element, various properties are extracted, including the tag name, the element's position, its inner text, and its unique identifier. Additionally, the process retrieves the computed styles for each element to analyze its appearance.
5. Explore child elements recursively, extracting similar data. This approach ensures that the element hierarchy is fully captured, effectively creating a tree-like structure that represents the structure of the HTML document.
6. The information for each element, along with data about its child elements, is added to its respective array (either for the exercise or for the student's attempt) as an object.

---

<sup>8</sup><https://www.npmjs.com/package/playwright>

<sup>9</sup><https://www.npmjs.com/package/puppeteer>

<sup>10</sup><https://www.npmjs.com/package/express>

7. At this stage, the arrays only contain the top-level elements from the HTML file. Therefore, a function is used to fetch all nested child elements and add them to their respective arrays. This action completes the representation of the HTML document.

#### 4.4.3 Element Relationships

The procedure for comparing elements between the attempt and reference interfaces is based on the web interface matching algorithm developed by Leal and Primo [20]. This algorithm conducts an examination of spatial relationships, styling, and text content present in the interfaces, generating a match score for each pair of elements. Subsequently, the pairs that generated higher scores are selected for further steps in the process. This ensures an effective matching system that takes multiple aspects of each element into consideration. Here is an outline of the process:

1. Each element from both interfaces is represented as a node in a structured tree, where each node has four sectors. These sectors are determined based on the relative positions of other elements to the node: upper left, upper right, lower left, and lower right.
2. Within each sector, elements are organized and sorted based on their relative position to provide an orderly structure for comparison.
3. A tree-like structure is then established for each node. This structure uses an element as a root and the remaining elements in the same sector as the descendants. This hierarchical organization is recursively done for each descendant.
4. The next step involves the comparison of the styling properties of two elements. This is achieved by checking the common properties between the two elements and determining how many of these common properties have identical values.
5. A match score is then calculated for each pair of elements. This score is dependent on three main factors: the similarity of styles, the similarity of the distribution of surrounding elements, and the presence of text content in both elements.
6. Every element from the solution interface is compared with each element from the attempt interface. For each pair, the match score is calculated, and all these pairs with their corresponding scores are added to a list.
7. After all possible pairs have been evaluated, the list is sorted in descending order based on the match scores. This way, the pairs with the highest match scores come first, implying they are the best matches.
8. The best pairs are selected such that no element from the solution or the attempt is repeated in more than one pair. This ensures that each selected pair is unique, representing the best possible match between elements of the solution and the attempt.

Upon completion of the element comparison process, the selected pairs, representing the optimal matches between elements in the student's attempt and the reference solution, are prepared for the next stage of evaluation. This array of the best pairs is forwarded to the functional assessment component. In this subsequent phase, a series of functional tests are performed directly on the student's code. These tests aim to assess the functionality of the code, examining how effectively it carries out its intended tasks and responds to various inputs and conditions.

## 4.5 Functional Testing

The process of functional testing is carried out in the next segment of the evaluation pipeline, and its steps are outlined as follows:

1. The pairs identified as the best matches in the previous phase are utilized as input to this process.
2. A mapping is created that relates each element's ID from the reference solution to the corresponding element's ID in the student's attempt. This mapping is vital as it links the elements that have been identified as corresponding matches between the two interfaces.
3. The test data are adjusted to match the IDs in the student's attempt, replacing the original IDs of the solution interface elements. This update ensures that the functional tests are appropriately applied to the elements in the student's attempt.
4. A new browser instance is launched for the student's attempt interface, and it is prepared for the execution of the tests. This preparation includes loading the necessary testing libraries and setting up the testing environment within the page.
5. The updated test data are then injected into the student's attempt page.
6. The functional tests are executed within the browser environment, and the results are collected. Each test's status (whether it passed or failed) is tracked, and if a test fails, the associated error message is also recorded.
7. After all the tests have been run, the browser instance is closed, and the results of the functional tests are returned.

The mapping process is an essential step in functional testing by transforming the IDs from the reference interface into the corresponding student attempt IDs. This ensures that students are not restricted to specific IDs in their solutions, allowing for more flexibility in their submissions.

Following this step, we have an array of results from the functional tests, each indicating whether the corresponding functionality in the student's attempt was successful or not.

## 4.6 Feedback Manager

The Feedback Manager analyzes the outputs of previous steps and generates helpful feedback, which helps students improve their code. It does so by employing a series of methods, each designed to handle the results from different parts of the evaluation process, such as schema validation, syntax validation, best pairs mapping, and functionality evaluation.

For instance, in the case of schema validation, should the result indicate a non-conformance with the required file format in the student's submission, the Feedback Manager promptly creates feedback, advising the student to ensure their file format aligns with the expected format.

Similarly, when handling syntax validation results, the Feedback Manager handles its outputs. Each error is acknowledged with a message that pinpoints the problematic file and the error message, thereby enabling students to address these issues.

During the best pairs mapping stage, the Feedback Manager examines the correspondence between the student's attempt and the reference solution. If a lack of matches is observed, it suggests a review of their HTML structure and element relationships through a feedback message.

The functionality test results are also very relevant for feedback generation. In scenarios where a test fails due to mapping errors between the student's HTML elements and those in the solution, the Feedback Manager provides feedback, suggesting to the student to reassess their interface or the JavaScript code that manipulates the Document Object Model (DOM).

Moreover, the Feedback Manager creates explicit feedback messages for any other test failures, incorporating the title of the test and the specific error message. This detailed feedback guides students toward understanding precise functional issues within their attempts.

Following the generation of all feedback messages, the Feedback Manager employs a strategy to select a final feedback message for the student.

Each feedback message carries a weight indicative of its importance. Using these weights, the Feedback Manager randomly picks a message, with a higher chance of picking a feedback message with a bigger weight.

On occasions when no feedback messages need to be generated, implying that all tests and validations were successful, the Feedback Manager returns a message congratulating the student. Conversely, in cases where persistent errors surface in consecutive evaluations, the Feedback Manager motivates the student to keep striving and tackle the previously noted errors.

## 4.7 Webpal API

The Webpal API is the interface that facilitates communication between the VLE and Webpal. This API incorporates several functionalities, each corresponding to a different aspect for managing and evaluating exercises. The capabilities of this API include:

- `createExercise(exerciseData, testData, assignment)`: Generates an unique identifier automatically for each new exercise. Input data is validated for syntax and against

the JSON schema. If validations pass, a new exercise instance is created with the provided data and stored. If there are any validation errors, an error is thrown.

- `deleteExercise(id)`: Removes an exercise from the system using the unique identifier. The corresponding data and files in the local storage are deleted. If the exercise isn't found, an error is thrown.
- `getFullExercise(id)`: Retrieves a specific exercise's data, test data, and assignment using its unique identifier. If the exercise is not found, an error is thrown.
- `getExerciseData(id)`: Fetches the exercise data for a specific exercise identified by the unique identifier. If the exercise is not found, an error is thrown.
- `getExerciseTestData(id)`: Retrieves the test data for a specific exercise identified by the unique identifier. If the exercise is not found, an error is thrown.
- `getExerciseAssignment(id)`: Retrieves the assignment instructions for a specific exercise identified by the unique identifier. If the exercise is not found, an error is thrown.
- `getAllExercises()`: Returns a list of all the exercises that are currently stored in the system.
- `evaluateAttempt(exerciseID, attemptData, previousFeedback)`: This function is responsible for constructing a new evaluation instance. It accepts three parameters: the unique identifier for the exercise, the student's attempt data, and an array of strings consisting of previously given feedback. After these parameters are processed, the function executes an internal routine that encompasses all the mechanisms outlined in this chapter. The outcome of this operation is an insightful feedback report on the student's attempt.
- `evaluateAttemptWithoutStatic(exerciseID, attemptData, previousFeedback)`: Similar to the `evaluateAttempt` operation, this function omits the static/syntax evaluation phase. Its existence serves the purpose of the validation demo, as detailed in Chapter 5, enabling a focus on other aspects of the tool while bypassing syntax validation.



## Chapter 5

# Proof of Concept

To evaluate the capabilities of Webpal and test its integration within code playgrounds, a proof of concept was created to provide an interactive experience with the tool. The backend server was developed using Node.js, while the frontend interface was designed using Vue.js.

This proof of concept was structured around four key components:

- **Main Page (5.1)** — This was the interactive platform where students engaged with and solved the exercises while receiving feedback.
- **Updated Page (5.3)** — A parallel version of the Main Page. Its main difference lay in the feedback given: it only communicates if a student's attempt is "Correct" or "Incorrect". To facilitate immediate visual recognition during the experiment, this page was designed with a distinct background color.
- **Sidebar (5.5)** — Acting as a navigation and information panel, the Sidebar allows students to select exercises and check the objective of each exercise.
- **Backoffice (5.6)** — Simplified interface, designed for teachers to create new exercises.

### 5.1 Main Page

The Main Page simulates the intuitive interface of *Codepen*<sup>12</sup>, containing three code editor input boxes where students can directly write their code. These boxes are supported by the *npm* package *Codemirror*, a tool that brings features like syntax highlighting, auto-completion, and line numbering to the interface. Refer to Figure 5.1 for a graphical depiction of the Main Page interface.

Beneath these *Codemirror* components exist three additional elements:

- An 'Attempt' IFrame allows students to view the current state of their code's execution.
- A 'Reference' IFrame offers a reference point by presenting the interface and functionality of the selected exercise, designed by the educator.

---

<sup>12</sup><https://codepen.io/>



Figure 5.1: Webpal Main Page

- A 'Feedback Log' box is present to provide a log of feedback, aiding students in understanding and rectifying the flaws in their code.

This interface also includes three buttons:

- **"Hamburger" Button** — This triggers the sidebar, enabling students to select an exercise.
- **"Execute" Button** — This initiates the execution of the student's code, which is then evaluated against the reference code.
- **"Clear Log" Button** — This offers the student the ability to clear the feedback log, especially helpful when it's filled with numerous previously addressed feedback.

### 5.1.1 Feedback Log

The Feedback Log is where students can analyze their feedback, each accompanied by a timestamp for easy reference. Refer to Figure 5.2 for an illustrative representation of a populated Feedback Log.

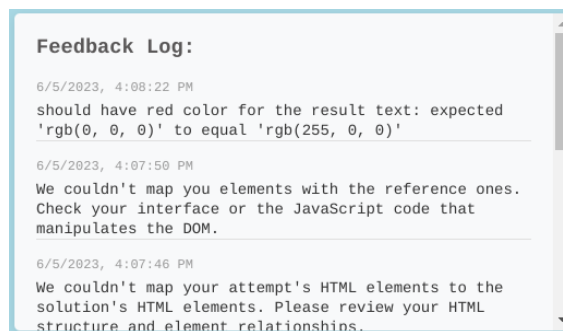


Figure 5.2: Webpal Feedback Log

## 5.2 Updated Page

The Updated Page replicates the Main Page but introduces a difference in the feedback it provides. It employs a different validation function, specifically `evaluateAttemptWithoutStatic`, as detailed in Chapter 4. Contrasting with the Main Page that utilizes the `evaluateAttempt` function, the Updated Page omits the syntax validation process. The exclusion of syntax validation is intentional as it often generates feedback on coding best practices that, while not strictly necessary for functionality, contribute to maintaining clean and better code, such as the inclusion of `<!DOCTYPE html>` in an HTML file, for example. This modification will simplify the work in chapter 6.

Figure 5.3 depicts the graphical interface of the updated page.

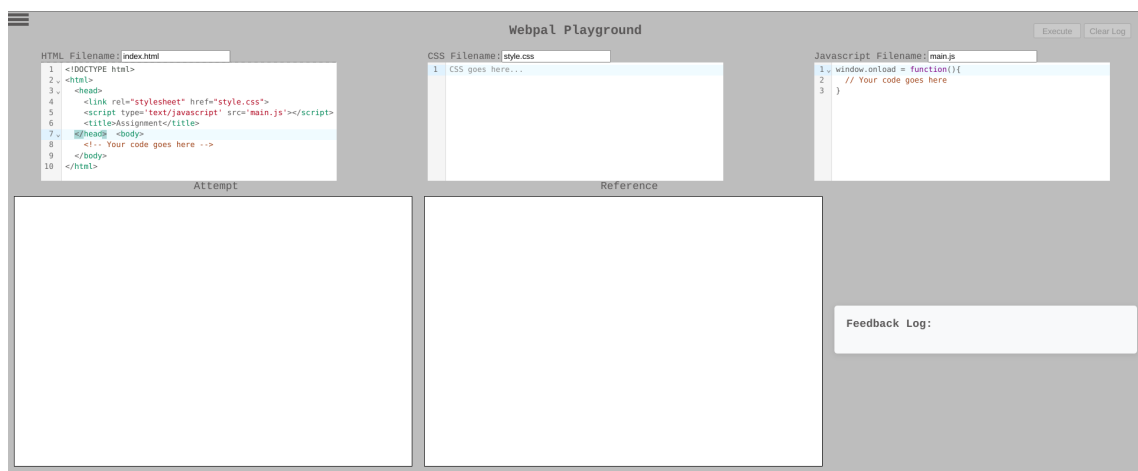


Figure 5.3: Webpal Updated Page

### 5.2.1 Feedback Log

The feedback on this interface diverges from that of the Main Page by solely providing an indication of the correctness of an attempt. This simplified feedback system is illustrated in Figure 5.4.

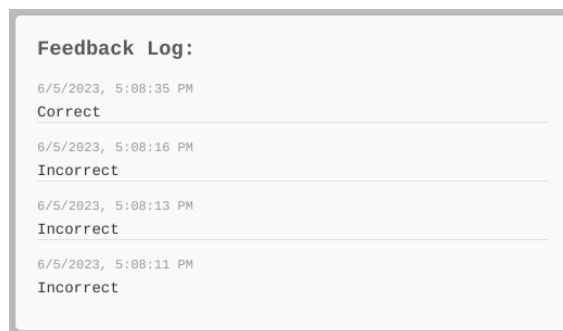


Figure 5.4: Webpal Feedback Log Updated

## 5.3 Sidebar

The Sidebar serves an important role in the student's navigation, enabling the selection of an exercise from all the available options. Upon a student's selection, the exercise interface is displayed within the reference iframe. Each exercise is further associated with an 'Info' button which, when clicked, opens a popup that provides a detailed description of the exercise's objective. Figure 5.5 provides a visual depiction of the Sidebar, showing the set of exercises incorporated in the experiment.

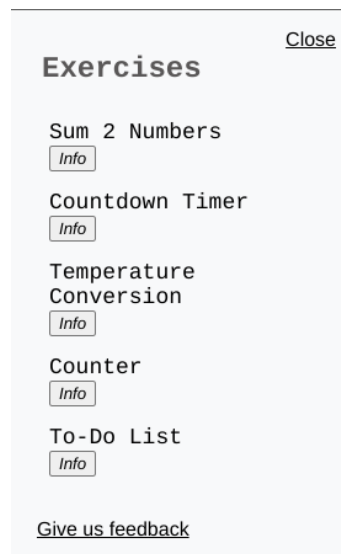


Figure 5.5: Webpal Sidebar

### 5.3.1 Exercise List

A set of five simple exercises were created. These exercises were designed to evaluate a broad range of web programming skills, including creating relationships between elements, applying style to elements, and implementing element functionality. The following provides an overview of the exercises incorporated with the associated description:

1. **Sum Two Numbers** — Create a webpage that accepts two numerical inputs from the user. There is a 'Sum' button that, when clicked, displays the sum of these two numbers. Additionally, there's a 'Reset' button that clears the input fields and resets the sum to zero upon clicking.
2. **Counter** — Create a webpage that displays a counter to the user. The counter starts at zero and is incremented or decremented by one when the '+' or '-' button is clicked, respectively. Additionally, there's a 'Reset' button that resets the counter back to zero upon clicking.

3. **Temperature Conversion** — Create a webpage that allows users to convert temperatures between Celsius and Fahrenheit. Users should be able to enter a temperature in either Celsius or Fahrenheit, and the application should automatically display the equivalent temperature in the other unit. When a user enters a non-numeric input, the corresponding field in the other unit should be cleared. The conversions should correctly apply the formulas for transforming Celsius to Fahrenheit and vice versa.
4. **Countdown Timer** — Create a webpage that allows users to set a countdown timer. Users should be able to enter a duration in seconds and start the countdown by clicking the 'Start' button. The countdown should decrement every second and display the remaining time. Implement a 'Pause' button that pauses the countdown and a 'Reset' button that stops the countdown and resets it to zero. The countdown timer should stop automatically when it reaches zero.
5. **To-Do List** — Develop a webpage that enables users to manage a simple to-do list. Users should be able to input tasks in a text field, and clicking the 'Add' button adds these tasks to a list. Each item in the list should include a 'Delete' button that removes the item from the list when clicked. There should also be a 'Reset' button that clears all tasks from the list.

Each exercise is structured to evaluate a particular aspect of web programming, with an emphasis on testing the creation of relationships between different elements in the web page. This way encourages students to think about how different parts of a webpage interact and work together, thereby improving their ability to design and develop efficient and interactive web applications.

In the forthcoming subsections, it's provided a description of the test executions associated with each exercise. This will inform about the specific objectives each exercise is expected to fulfill, the steps involved in their execution, and how they help in evaluating the student's understanding and application of web programming concepts.

#### 5.3.1.1 Sum two Numbers

The "Sum Two Numbers" exercise is structured to evaluate a student's ability to interact with the DOM and manipulate its elements. Specifically, this exercise tests the student's ability to capture input values, perform a calculation, and display the result, as well as handle events and change element styles.

The tests executed for this exercise are:

- **Sum Calculation** — This test checks whether the sum of two numbers, entered in the respective input fields ('number1' and 'number2'), is correctly calculated and displayed when the 'sum' button is clicked. For instance, if '5' and '3' are entered in the 'number1' and 'number2' fields respectively, clicking the 'sum' button should result in '8' being displayed in the 'resultText' field.

- **Reset Functionality** — This test evaluates whether the application resets correctly, clearing input fields and resetting the result to '0' when the 'reset' button is clicked. If '5' and '3' are in the input fields and the result displays '8', clicking the 'reset' button should empty the input fields and change the 'resultText' to '0'.
- **Style Application** — Lastly, this test is designed to check whether the text displaying the result ('resultText') is styled correctly with a red color. The `getComputedStyle` function verifies that the color of the 'resultText' field is `'rgb(255, 0, 0)'`, the RGB equivalent of red.

### 5.3.1.2 Counter

The tasks for the "Counter" exercise involve manipulating a counter using various controls - increment, decrement, and reset buttons.

The tests executed for this exercise are:

- **Increment Operation** — This test checks whether the counter increments correctly when the increment button is clicked. Clicking the increment button should increase the count displayed in the 'count' field by 1.
- **Reset Operation** — This test validates the reset functionality of the application. When the 'reset' button is clicked, the 'count' field should reset to '0', regardless of its previous value.
- **Sequence of Operations** — Finally, this test is designed to verify the correct functionality of the counter through a series of operations. Here, a sequence of increment and decrement operations are performed, and the final count is validated. For example, if the sequence of operations is increment, decrement, increment, increment, decrement, increment, the 'count' field should display '2'.

### 5.3.1.3 Temperature Conversion

The "Temperature Conversion" exercise is designed to evaluate the student's ability to implement conversion logic. The goal of this exercise is to create an application that correctly converts temperatures between Celsius and Fahrenheit scales.

The tests executed for this exercise are:

- **Celsius to Fahrenheit Conversion** — This test ensures the application accurately converts temperatures from Celsius to Fahrenheit. For instance, if a value of '25' is entered in the Celsius field, the Fahrenheit field should display '77.00' after the conversion.
- **Fahrenheit to Celsius Conversion** — This test checks the application's ability to correctly convert temperatures from Fahrenheit to Celsius. For example, an input of '98.6' in the Fahrenheit field should yield '37.00' in the Celsius field after the conversion.

- **Non-numeric Input (Celsius to Fahrenheit)** — This test evaluates how the application responds to non-numeric input in the Celsius field. Upon entering a non-numeric value, the Fahrenheit field should be cleared.
- **Non-numeric Input (Fahrenheit to Celsius):** — Similarly, this test checks the application's behavior in response to non-numeric input in the Fahrenheit field. If a non-numeric value is given, the Celsius field should be cleared.

#### 5.3.1.4 Countdown Timer

The "Countdown Timer" exercise focuses on the student's ability to work with timers. This involves creating a countdown timer that starts, pauses, resets, and counts down correctly, emphasizing the need for precise control of the application state.

The tests executed for this exercise are:

- **Initial Values** — This test checks if the initial values of the countdown timer and seconds input are correct. Both the seconds input field and the countdown timer should start as empty and '0', respectively.
- **Start Countdown** — In this case, the test verifies if the countdown starts correctly when the start button is clicked. When a numeric value is entered in the seconds input field, and the start button is clicked, the countdown timer should start from the same value.
- **Pause Countdown** — Check whether the countdown timer pauses correctly when the pause button is clicked. For example, if a countdown from '5' is in progress and the pause button is clicked immediately, the countdown timer should remain at '5'.
- **Reset Countdown** — Verify if the countdown timer resets to '0' and clears the seconds input field when the reset button is clicked.
- **Countdown to Zero** — Validate if the countdown timer correctly counts down to '0'. For instance, when a countdown from '1' is started, the countdown timer should reach '0' after one second.

#### 5.3.1.5 To-Do List

Lastly, the "To-Do List" exercise helps evaluate a student's ability to create and manipulate dynamic elements in the DOM, handle user events, and implement basic styling through CSS.

Here is the description of each test executed for this exercise:

- **Initial Values** — This test checks whether the initial values of the task input field and the task list are correct. Initially, the task input should be empty, and there should be no tasks in the list.

- **Add Task** — Validate the function of the 'add' button. When a task is written, and the add button is clicked, the task should be added to the list, increasing the number of children in the task list by one.
- **Delete Task** — Check the functionality of the 'delete' button. After adding a task to the list, clicking the delete button should remove the task, returning the task list's children count to zero.
- **Reset List** — Confirm that clicking the reset button clears all tasks from the list, making the task list empty.
- **Cursor Style** — Check if the cursor changes to a pointer when hovering over the add and reset buttons.

## 5.4 Backoffice

The Backoffice of the proof of concept was designed to provide educators with a user-friendly interface for creating new exercises. There are five input fields for teachers to fill out:

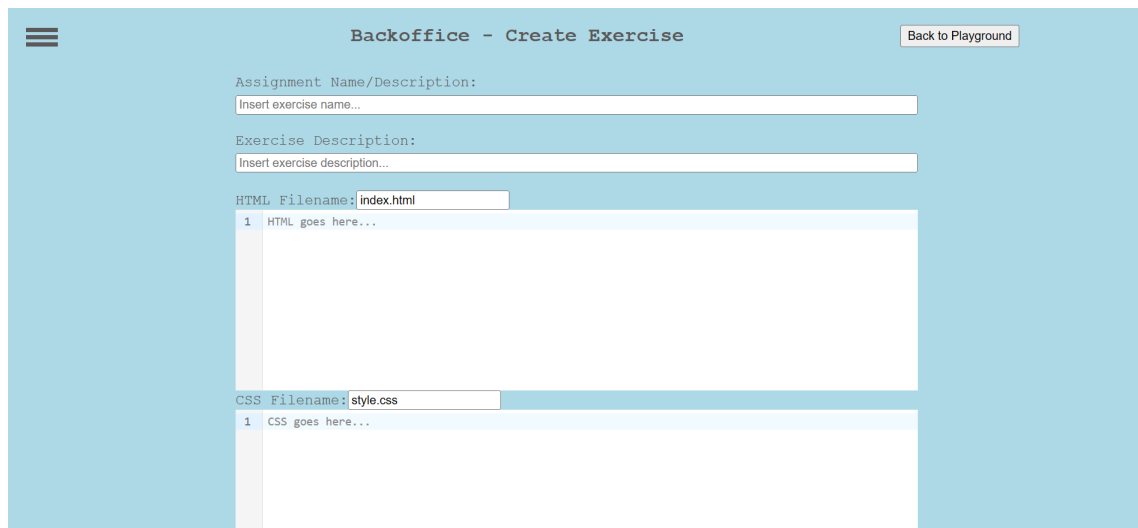
- **Assignment Name:** This is the name or brief description of the exercise. It helps users identify exercises quickly.
- **Assignment Description:** Created specifically for the proof of concept. This field allows for a more detailed description of the exercise, helping students understand the objective of the task.
- **HTML/CSS/JS Code Editors:** Where teachers enter the code for the exercise. These inputs, like the ones in the main and updated pages, are supported by the *Codemirror* package to provide code highlighting and autocomplete. The filename for each of these files can also be specified in the respective input fields above each editor.
- **JS Mocha Tests:** Teachers can write the Javascript tests for the exercise. These tests can be written using the *Mocha* or *Chai* testing frameworks.

Upon form submission, all the input data is merged into a JSON object, adhering to the pre-defined schema required for the Request Validation process. This JSON object, along with the JavaScript tests, assignment name, and description, is subsequently sent to Webpal. As a result, a new exercise is created. Figure 5.6 shows the visual interface of the Backoffice page.

## 5.5 Log Files

Another important functionality of the backend server, implemented for this proof of concept, includes two important endpoints: one for storing logs and another for downloading them.





The screenshot shows a web interface titled "Backoffice - Create Exercise" with a "Back to Playground" button in the top right. The form contains several input fields and text areas:

- Assignment Name/Description:** A text input field with the placeholder "Insert exercise name..."
- Exercise Description:** A text input field with the placeholder "Insert exercise description..."
- HTML Filename:** A text input field containing "index.html". Below it is a text area with a line number "1" and the text "HTML goes here..."
- CSS Filename:** A text input field containing "style.css". Below it is a text area with a line number "1" and the text "CSS goes here..."

Figure 5.6: Webpal Backoffice

The first endpoint, for storing logs, is labeled `/log`. Its primary purpose is to receive and store the log data sent via POST requests. Each incoming log request is packed with the user's unique ID and the associated log content. Using the user ID, a unique filename is established for each user, thereby providing individual activity tracking for each student. The log content is converted into a tab-separated string, which is appropriate for a `.tsv` (Tab Separated Values) file. Depending on the existence of a log file for the specific user, the system will append the log entry to the current file or create a new one.

The second endpoint for downloading logs, `/downloadLogs`, takes responsibility for compiling and retrieving all user log files. The server initiates a writable stream associated with a `.zip` file and creates a pipeline for the `'zip'` archiver. The archiver is directed to include all files from the log directory. Once the archive is ready, the server commences a download response with the `.zip` file.

Log data is recorded for both versions: the one that provides feedback, and the other that does not. Once the data collection phase is complete, all the logs from the feedback version are merged into a single file, and similarly, the logs from the version without feedback are combined into another separate file. This process simplifies the subsequent analysis of logs, making it easier to compare and evaluate the two versions.

### 5.5.1 Log File Parameters

The logs generated by Webpal capture specific parameters of student interactions. The parameters stored in a log file include the `'studentID'`, `'exerciseID'`, `'exerciseName'`, `'timestamp'`, `'withFeedback'`, `'type'`, and `'feedback'`. These are described below:

- **studentID:** Anonymous unique identifier representing each student, allowing individual tracking.

- **exerciseID**: This is the unique identifier for each exercise. It provides a reference for connecting student interactions to specific exercises.
- **exerciseName**: This is the name of the exercise that the student attempted. It offers a more human-readable reference for identifying exercises.
- **timestamp**: This is the exact time when a student's action took place. The timestamp allows chronological mapping of student interactions.
- **withFeedback**: This binary parameter represents whether the student interacted with the version of the proof of concept that provides feedback. If 'true', the student used the feedback-enabled version, otherwise, they used the version without feedback.
- **type**: This parameter identifies the nature of the student's interaction, specified as either 'action' or 'feedback'. An 'action' signifies the selection of an exercise, whereas 'feedback' denotes an instance where feedback was generated in response to the student's interaction.
- **feedback**: This is the feedback provided by Webpal in response to a student's action. It records the Webpal response to the student's approach.

An Installation Guide is available in Appendix A and a User Guide in Appendix B to ease the installation and usage process for future integrations.

# Chapter 6

## Validation

To validate Webpal, an experiment (6.1) was conducted by replicating its usage in a code playground. The results (6.2) obtained from this experiment are presented, showing the findings and initiating a discussion (6.3) of their implications. The results are analyzed, taking into account any limitations or challenges encountered during the validation. To aid in the analysis of the results, a hypothesis test and a correlation analysis were conducted. These statistical techniques allowed for a more rigorous examination of the data, enabling insights into the relationships and potential patterns present in the validation results. Potential areas for improvement and future work are also highlighted based on the insights gained from the validation that will inform about future steps to further enhance Webpal capabilities.

### 6.1 Experiment

An experiment was conducted at ESMAD - a Polytechnic Institute of Porto school with a class of the Undergraduate Degree in Web Information Systems and Technologies course. The class was separated into two groups, one serving as an experimental group and the other as the control group, and had to solve 5 web programming exercises. The experimental group had access to a version of the tool that would allow them to execute and submit their attempts and receive custom feedback about their attempt code. The other group didn't receive detailed feedback and was only capable of checking if their attempt was correct or not. The experiment occurred in two different classes of 40 minutes. There was a total of 9 students in the control group with an average age of 20.89 years and with a gender distribution (6.1) of 33.30% female and 66.70% male. Conversely, the experimental group was composed of 7 students, with an average age of 19 years, and displayed a slightly more balanced gender distribution (6.2): 42.90% females and 57.10% males.

While the students solved the exercises, logs were stored for posterior analysis.

After analyzing the logs, the objective was to get answers to the following key questions:

- What was the total count of exercises initiated by each student on average?
- How many exercises were successfully completed by students on average?

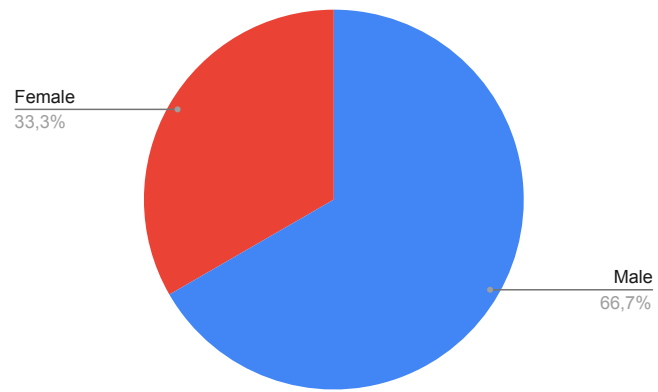


Figure 6.1: Gender Distribution - Control Group

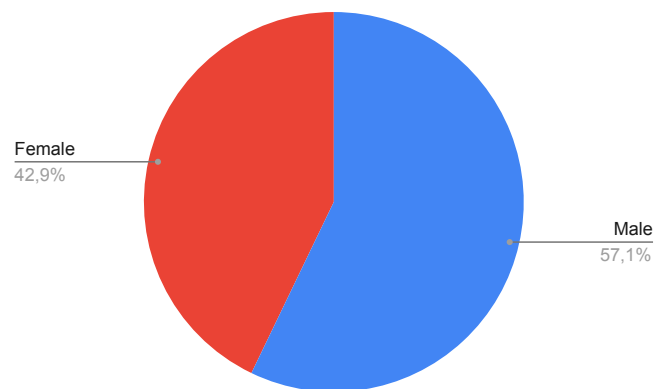


Figure 6.2: Gender Distribution - Experimental Group

- What was the average number of attempts made by the students?
- What was the average time spent on the first exercise "Sum Two Numbers"?

### 6.1.1 Definition of Instruments

To realize this experiment it was necessary that the platform used by the students were similar to code playgrounds. For that, it was necessary to build some instruments to aid in the validation process.

The proof of concept 5 created to test the integration and functionalities of Webpal in a code playground was ideal for this validation. It contains 5 simple exercises capable of assessing various web programming aspects and provides a simple and interactive platform for students to use.

The experimental group utilized the 5.1 version, which offered specific feedback for their solution attempts, whereas the control group used the 5.2 version, which provided information solely on the correctness of their attempts. The latter lacked the syntax validation functionality, as it could potentially trigger an "Incorrect" feedback message due to bad programming practices - attempts can be correct even with bad programming practices - potentially causing students to become stuck in an exercise.

In the sidebar of the replicated code playground was added a button called "Give Us Feedback", which opens a questionnaire designed to gather students' experiences and impressions of using Webpal. The questionnaire is structured around the following questions:

- Were the instructions for using the Webpal demo clear and easy to understand??
- How would you rate the difficulty of the exercises?
- Was the feedback provided by Webpal helpful for improving your code?
- Did you enjoy using Webpal for the coding exercises?

Additionally, some students contributed their ideas on potential enhancements or modifications that could be incorporated into Webpal for an improved user experience.

The questionnaire and individual results tables can be found in Appendix E, providing comprehensive data for reference. Additionally, visual representations in the form of graphs have been generated to facilitate the visualization of the response patterns, and these graphs are also included.

The proof of concept server and interface was initially hosted on Amazon Web Services (AWS), leveraging the free tier provided for student projects. However, when a stress test was conducted — simulating the simultaneous interaction of twenty browser instances with Webpal — it became clear that the servers under the AWS free tier were incapable of effectively handling this level of concurrent requests. This stress test was made possible using the Selenium WebDriver<sup>11</sup>, a tool designed to facilitate web testing. More details of the stress test implementation are available in Appendix C. Therefore, as an alternative, the entire setup was hosted on a Debian machine provided by the Faculty of Sciences of the University of Porto.

<sup>11</sup><https://www.selenium.dev/documentation/webdriver/>

## 6.2 Results

This section provides an evaluation of the outcomes derived from the experiment. The results presented here were obtained through the analysis and interpretation of data gathered during the experiment. The primary objective of this section is to look into the effectiveness, efficiency, and usability of Webpal, while concurrently identifying potential areas of enhancement and refinement.

Throughout the course of the experiment, it was gathered over 400 distinct log lines. Subsequently, all the logs from the experimental group were consolidated into one file, and similarly, the logs from the control group were merged into another separate file. With this data, it was possible to get answers to the four key questions that were initially questioned at the beginning of this chapter:

1. What was the total count of exercises initiated by each student on average?
2. How many exercises were successfully completed by students on average?
3. What was the average number of attempts made by the students?
4. What was the average time spent on the first exercise "Sum Two Numbers"?

In order to facilitate this analysis, some Python scripts were created leveraging the capabilities of the *pandas* library, which is well-suited for handling tab-separated values (TSV) files. The average results are presented in the following table:

Table 6.1: Response to the key questions - Average

Question	Control Group	Experimental Group
What was the total count of exercises initiated by each student on average?	2.22	3.71
How many exercises were successfully completed by students on average?	1.33	1.5
What was the average number of attempts made by the students?	21.33	30.43
What was the average time spent on the first exercise "Sum Two Numbers"?	00:20:26	00:24:53

To gain better insights and assess whether the differences observed between the experimental and control groups are statistically significant, a hypothesis test was conducted on the data. This required additional data preparation to generate comparable values.

In response to the first question, the data was narrowed down to just the number of exercises started by each student. The second question was simplified to the number of exercises each student completed. For the third question, the data was adjusted to reflect the number of attempts made by each student. For the fourth and final question, the data was simplified to show the time, in seconds, each student spent on the "Sum Two Numbers" exercise (the first exercise of the experiment).

A more detailed approach is available in Appendix D, containing tables for each question and group with the respective values.

### 6.2.1 Hypothesis Test

Hypothesis testing is a statistical method used to draw conclusions about a population based on sample data. In this study, this method was used to determine if there were significant differences between the experimental and control groups. The four questions from the previous section were used: the number of exercises initiated, the number of exercises successfully completed, the number of attempts made per exercise, and the time spent solving an exercise. The process of hypothesis testing involves defining a null hypothesis, which indicates no significant difference between the groups, and an alternative hypothesis that suggests a difference [9].

Before testing these hypotheses, the data was checked for normality and homogeneity of variances using the Shapiro-Wilk and Levene's tests, respectively. These checks are important because they help to decide which type of statistical test should be used.

The Shapiro-Wilk test checks the null hypothesis that the data is drawn from a normal distribution. If the p-value is larger than the significance level (0.05 in this case), the test fails to reject this hypothesis, indicating that the data do not significantly deviate from normality. Levene's test checks the null hypothesis that the variances in the two groups are equal. Again, if the p-value is greater than 0.05, the test fails to reject the hypothesis, suggesting that the variances of the groups do not significantly differ.

Following these checks, a Mann-Whitney U Test was performed. This test was chosen because it is a non-parametric statistical test that does not require the assumptions of normal data distribution and equal variances across compared groups. This makes it a suitable choice for this analysis, given that not all data followed a normal distribution and didn't have equal variances [31]. The null hypothesis for this test is that the distributions of both groups are equal. If the resulting p-value is larger than the significance level, it fails to reject the null hypothesis, suggesting that the distributions of the two groups are not significantly different.

The next descriptive list shows the results for each test:

#### Exercises Started

Both the feedback and non-feedback groups displayed normal distribution, while the variances were also found to be equal. The Mann-Whitney U Test produced a p-value of 0.052, failing to reject the null hypothesis. This suggests that there is not a statistically significant difference between the two groups in terms of the number of exercises started.

#### Exercises Completed

Neither the feedback nor the non-feedback groups displayed normal distribution, but the variances were found to be equal. The p-value from the Mann-Whitney U Test was 0.406, failing to reject the null hypothesis. Hence, there is no statistically significant difference between the groups in terms of the number of exercises completed.

### Attempts Made

The number of attempts made in both groups followed a normal distribution, and their variances were equal. The p-value of 0.079 from the Mann-Whitney U Test means it fail to reject the null hypothesis, indicating no statistically significant difference between the groups in the number of attempts made.

### Time Spent on "Sum Two Numbers"

Both groups showed normal distribution and equal variances for the time spent on the first exercise. The p-value of 0.210 from the Mann-Whitney U Test indicates no significant difference between the two groups. ...

The results of these tests indicate that there are no significant differences between the feedback and non-feedback groups for any of the four questions studied. One possible explanation for these findings could be the relatively small sample size used in this study. Additionally, the limited duration of the experiment, which was only 40 minutes, might also have influenced the outcomes. It's worth noting that the validation was conducted during the end of a semester — a time when students are typically occupied with exams and final projects — constraining their available time. Future research could benefit from a larger sample size, which would increase the power of the statistical tests and potentially enable the use of parametric testing methods. However, it's also possible that there truly are no differences between the groups for these parameters.

After performing the first part of the experiment, the participants were asked to fill out a questionnaire about their experience using Webpal. In order to identify the students, the anonymous student ID passed automatically from the Webpal playground to a dedicated text field in the form.

Some participants also answered the optional open-ended question "If there is one thing you could improve or change in Webpal, what would it be?". The answers to this question were mainly about the design of the playground rather than the generated feedback. The most common and relevant answers were about the input fields to write code being too small and the necessity for better auto-completion like Visual Studio Code for the JavaScript code.

For each question, an average of the responses were captured for each group and are specified in table 6.2.

Table 6.2: Response to the questions of the experiment

Question	Control	Experimental
Were the instructions for using the Webpal demo clear and easy to understand?	3.56	4.29
How would you rate the difficulty of the exercises?	2.22	2.86
Was the feedback provided by Webpal helpful for improving your code?	2.56	2.71
Did you enjoy using Webpal for the coding exercises?	3.11	4



At initial observation, the results appear to favor the experimental group. However, to determine any correlation between the form data and log data, a more comprehensive analysis is necessary. The subsequent section presents a study on the data correlation.

## 6.2.2 Correlation of Data

In order to conduct the correlation analysis, it was necessary to prepare both the form and log data. Initially, relevant information was extracted from the logs within the TSV file, which was then processed and stored in a new CSV file. Subsequently, the form data was also extracted and, to facilitate the process, column names were adjusted. Once prepared, the data sets were merged using the student ID as the common key, resulting in the generation of a correlation matrix. The script employed to create this matrix can be found in Appendix F.

The resulting correlation matrix for the control group are presented in the tables 6.3 and 6.4.

Table 6.3: Correlation Matrix (Part 1) - Control Group

	instructions	difficulty	feedback	enjoyment
instructions	1.000	0.297	0.547	0.725
difficulty	0.297	1.000	0.777	0.627
feedback	0.547	0.777	1.000	0.637
enjoyment	0.725	0.627	0.637	1.000

Table 6.4: Correlation Matrix (Part 2) - Control Group

	started_exercises	completed_exercises	attempts	time_spent
instructions	0.452	0.421	-0.283	-0.656
difficulty	-0.112	0.647	-0.395	-0.086
feedback	-0.092	0.641	-0.202	-0.238
enjoyment	0.302	0.580	-0.174	-0.180

Legend:

- **"instruction"**: Were the instructions for using the Webpal demo clear and easy to understand?
- **"difficulty"**: How would you rate the difficulty of the exercises?
- **"feedback"**: Was the feedback provided by Webpal helpful for improving your code?
- **"enjoyment"**: Did you enjoy using Webpal for the coding exercises?
- **"started\_exercise"**: What was the total count of exercises initiated by each student on average?
- **"completed\_exercises"**: How many exercises were successfully completed by students on average?

- **"attempts"**: What was the average number of attempts made by the students?
- **"time\_spent"**: What was the average time spent on the first exercise "Sum Two Numbers"?

After analyzing the Correlation Matrix for the control group, there are some points worth discussing:

1. **"instructions" and "enjoyment"** — Strong positive correlation (0.725). This could mean that students who found the instructions clear and easy to understand also enjoyed better using Webpal for the coding exercises.
2. **"difficulty" and "feedback"** — Also a strong positive correlation (0.777). This could indicate that students who rated the exercises as more difficult also found the feedback provided by Webpal to be more helpful for improving their code. This suggests that the feedback mechanism may be valuable when students are dealing with difficult exercises.
3. **"difficulty" and "completed\_exercises"** — Positive correlation (0.647). This suggests that students who found the exercises more difficult actually completed more of them.
4. **"started\_exercise" and "time\_spent"** — There is a strong negative correlation (-0.665). This indicates that students who started more exercises spent less time on the first exercise.
5. **"instructions" and "time\_spent"** — There is a strong negative correlation (-0.656). This means students who found the instructions clear spent less time on the first exercise.

For the experimental group the correlation matrix is shown in the tables 6.5 and 6.6.

Table 6.5: Correlation Matrix (Part 1) - Experimental Group

	instructions	difficulty	feedback	enjoyment
instructions	1.000	0.091	0.636	0.296
difficulty	0.091	1.000	0.194	-0.810
feedback	0.636	0.194	1.000	0.105
enjoyment	0.296	-0.810	0.105	1.000

Table 6.6: Correlation Matrix (Part 2) - Experimental Group

	started_exercises	completed_exercises	attempts	time_spent
instructions	0.789	0.411	-0.568	0.407
difficulty	-0.320	0.417	0.410	0.538
feedback	0.589	0.081	-0.628	0.224
enjoyment	0.519	-0.135	-0.532	-0.598

The Correlation Matrix (6.5 and 6.6) for the experimental group also got some points worth analyzing:

1. **"instructions" and "started\_exercises"** — There's a strong positive correlation (0.789). This could mean students who found the instructions easy to understand started more exercises.
2. **"difficulty" and "enjoyment"** — Huge negative correlation (-0.810). This can indicate that as the difficulty of the exercises increased, students enjoyed using Webpal less.
3. **"instructions" and "feedback"** — A significant positive correlation (0.636). This can indicate that students who found the instructions easy to understand also found the feedback helpful.
4. **"attempts" and "feedback"** — Negative correlation (-0.628). Suggests that the more attempts students made, the less helpful they found the feedback. This could be because more attempts might indicate that students are struggling, so they may not find the feedback as useful.
5. **"enjoyment" and "attempts"** — Moderate negative correlation (-0.532). Implies that students who had to make more attempts enjoyed less using Webpal.
6. **"difficulty" and "time\_spent"** — Positive correlation (0.538), which could mean that students that think that the exercises were more difficult also spent more time trying to solve the first exercise.

### 6.3 Discussion of Results

Observing the data in Table 6.2, it's noteworthy that, on average, the experimental group initiated and completed more exercises, made more attempts, and spent more time on the first exercise compared to the control group. However, further statistical tests revealed that these differences were not statistically significant. This could imply that the feedback doesn't drastically influence the number of exercises started or completed, the number of attempts made, or the time spent on the exercises in the scope of this study.

When looking at the p-values derived from the Mann-Whitney U Test, they suggest that the feedback manager doesn't significantly affect the students' interaction with Webpal. This could mean that students' performance is determined by factors beyond the provision of feedback such as their motivation, prior knowledge of the topic, or time constraints.

The time of conducting the experiment (end of the semester) might have also impacted the results. Students are generally more focused on exams and final projects during this period, limiting the time and attention they can devote to these types of experiments. Future studies could consider conducting similar experiments at different times of the academic year.

On the other hand, the correlation analysis revealed some interesting relationships. For instance, students who found the instructions clear also reported enjoying using Webpal more and

started more exercises. Also, the strong positive correlation between difficulty and feedback implies that feedback becomes more valuable as the exercises get harder, reinforcing the role of feedback in supporting learning during challenging assignments.

However, it is worth noting some counter-intuitive results as well. For example, the correlation between attempts and feedback is negative, suggesting that students who made more attempts found the feedback less helpful. This could be due to the possibility that struggling students may not fully understand or effectively use the feedback provided.

In conclusion, while the feedback manager did not show a significant impact on student performance in this study, its role in student experience and in difficult tasks indicates that it is a feature worth retaining and optimizing in future versions of Webpal.

## Chapter 7

# Conclusion and Future Work

This dissertation aimed to dive into the automation of web application assessments and the development of a tool capable of evaluating simple web applications automatically. The tool was designed to offer incremental, non-repetitive feedback and is intended to ease the workload of teachers on the evaluation of student submissions for web programming assignments. This tool was built as an npm package to be easily integrated into VLEs. To achieve these goals, the following steps were followed:

- Analysis of existent tools, frameworks, and platforms capable of assessing interfaces and functionality and generating feedback.
- Design, specification, and architecture of Webpal to facilitate the development process.
- Exploration and integration of existing technologies, such as packages for static assessment and browser emulation.
- Development of a proof of concept interface for the validation experiment.
- Execution of a validation experiment to verify the effectiveness of the tool.

The initial design and objectives of Webpal were first introduced in a short paper published at the International Computer Programming Education Conference (ICPEC) in 2023. This paper provided a preliminary overview of the Webpal system, outlining its objectives and functionality. However, one of the limitations acknowledged in the paper was the lack of validation and results to support Webpal effectiveness.

This dissertation expands upon the work initially presented in the ICPEC 2023 paper. It dives into a comprehensive examination of all aspects of Webpal and enhances the prior work through the inclusion of a validation process. As a result of this validation, this dissertation is also able to present results.

In summary, this work has accomplished the following objectives:

- Development of Webpal<sup>13</sup>, an npm package capable of automatically assessing web applications built with HTML, CSS, and Javascript.
- Creation of an online code playground, available online<sup>14</sup>.
- Obtaining insights from the validation experiment, which involved the use of the code playground.
- Presentation of a paper at the international conference ICPEC 2023, showcasing the developed tool [5].

These achievements demonstrate the successful realization of the goals set forth in this proposed research, highlighting the contributions made to the field of automated assessment for web applications.

Although the developed tool is fully functional for assessing basic web applications, several aspects need further consideration for future refinements:

- A new, more complete validation should be considered. In the validation described in this work, it was verified that some results were inconclusive due to the conditions of the validation, such as the number of participants, motivation due to the end of the semester, and time constraints. Because of that, a new validation should be held at the beginning of a new academic semester since usually there are bigger presence and motivation rates at that time.
- Enhancing Webpal to effectively assess more complex web programming assignments, expanding its capabilities to evaluate advanced concepts and functionalities.
- Enabling the assessment of multi-page web applications submitted by students, allowing for a comprehensive evaluation of their ability to design and implement interconnected web pages.
- Usability tests of the validation interface can also be improved. Following the suggestions of the previous validation, a more organized layout with more space to write code and an auto-completion for the JavaScript code should be implemented.
- To use Webpal for grading students in the future, the inspect elements code should be obfuscated since participants can view the exercise code when inspecting the elements with the browser.

---

<sup>13</sup><https://www.npmjs.com/package/webpal>

<sup>14</sup><https://www.dcc.fc.up.pt/webpal/#/main>

# References

- [1] Kirsti Ala-Mutka, Toni Uimonen, and Hannu-Matti Järvinen. Supporting students in c++ programming courses with automatic program style assessment. *JITE*, 3:245–262, 01 2004.
- [2] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.
- [3] Valentina Arkorful, Nelly Abaidoo, et al. The role of e-learning, advantages and disadvantages of its adoption in higher education. *International journal of instructional technology and distance learning*, 12(1):29–42, 2015.
- [4] Tapio Auvinen. Harmful study habits in online learning environments with automatic assessment. *Proceedings - 2015 International Conference on Learning and Teaching in Computing and Engineering, LaTiCE 2015*, pages 50–57, 2015.
- [5] Luís Maia Costa, José Paulo Leal, and Ricardo Queirós. Automated assessment of simple web applications. In *Fourth International Computer Programming Education Conference (ICPEC 2023)*, volume 112. Schloss Dagstuhl- Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, 2023.
- [6] Pierre Dillenbourg, Daniel Schneider, and Paraskevi Synteta. Information and communication technologies in education, 2002.
- [7] Rhino Documentation. Rhino.
- [8] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *J Educ Resour Comput*, 5, 09 2005.
- [9] Frank Emmert-Streib and Matthias Dehmer. Understanding Statistical Hypothesis Testing: The Logic of Statistical Inference. *Machine Learning and Knowledge Extraction*, 1(3):945–962, 2019.
- [10] John English. Automated assessment of gui programs using jewl. *SIGCSE Bull.*, 36(3):137–141, jun 2004.
- [11] Man Feng and Andrew McAllister. A tool for automated gui program grading. In *Proceedings. Frontiers in Education. 36th Annual Conference*, pages 7 – 12. IEEE, 12 2006.
- [12] Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu. Apogee-automated project grading and instant feedback system for web based computing. *SIGCSE’08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, pages 77–81, 2008.
- [13] Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and L. Thomas van Binsbergen. Ask-elle: an adaptable programming tutor for haskell giving automated feedback. *International Journal of Artificial Intelligence in Education*, 27:65–100, 3 2017.

- [14] Geoffrey R. Gray and Colin A. Higgins. An introspective approach to marking graphical user interfaces. *ITiCSE06 - Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2006:43–47, 2006.
- [15] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, page 86–93, New York, NY, USA, 2010. Association for Computing Machinery.
- [16] David Jackson and Michelle Usher. Grading student programs using assyst. *SIGCSE Bull.*, 29(1):335–339, mar 1997.
- [17] Ville Karavirta and Petri Ihantola. Serverless automatic assessment of javascript exercises. *ITiCSE'10 - Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education*, page 303, 2010.
- [18] Ayaan M. Kazerouni, James C. Davis, Arinjoy Basak, Clifford A. Shaffer, Francisco Servant, and Stephen H. Edwards. Fast and accurate incremental feedback for students' software tests using selective mutation analysis. *Journal of Systems and Software*, 175:110905, 2021.
- [19] Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37:83–137, 2005.
- [20] José Paulo Leal and Marco Primo. A matching algorithm to assess web interfaces. In Teresa Guarda, Filipe Portela, and Maria Fernanda Augusto, editors, *Advanced Research in Technologies, Information, Innovation and Sustainability*, pages 115–126, Cham, 2022. Springer Nature Switzerland.
- [21] Guodong Li, Esben Andreasen, and Indradeep Ghosh. Symjs: Automatic symbolic testing of javascript web applications. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 16-21-Nov:449–459, 2014.
- [22] Ali Mesbah. Chapter five - advances in testing javascript-based web applications. volume 97 of *Advances in Computers*, pages 201–235. Elsevier, 2015.
- [23] Shabnam Mirshokraie. Effective test generation and adequacy assessment for javascript-based web applications. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 453–456, New York, NY, USA, 2014. Association for Computing Machinery.
- [24] Mark Noone and Aidan Mooney. Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education*, 5:149–174, 2018.
- [25] RL O'Leary and AJ Ramsden. *Virtual Learning Environments*, pages 1 – 30. Economics Learning and Teaching Support Network, 2002.
- [26] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. Automated assessment in computer science education: A state-of-the-art review. *ACM Transactions on Computing Education*, 22:1–40, 9 2022.
- [27] Matthew Peveler, Evan Maicus, and Barbara Cutler. Automated and manual grading of web-based assignments. *SIGCSE 2020 - Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, page 1373, 2020.



- [28] Ricardo Queirós. Webpuppet - a tiny automated web ui testing tool. In *Third International Computer Programming Education Conference (ICPEC 2022)*, volume 102. Schloss Dagstuhl- Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, 7 2022.
- [29] Khirulnizam Rahman and Md Jan Nordin. A review on the static analysis approach in the automated programming assessment systems. 07 2007.
- [30] Kenneth A. Reek. The try system -or- how to avoid testing student programs. *SIGCSE Bull.*, 21(1):112–116, feb 1989.
- [31] David J. Sheskin. The Mann–Whitney U Test. *Handbook of Parametric and Nonparametric Statistical Procedures*, pages 531–594, 2020.
- [32] Antonio C. Siochi and William R. Hardy. Webwolf: Towards a simple framework for automated assessment of webpage assignments in an introductory web programming class. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 84–89, 2015.
- [33] G. Suresh Babu and K. Sridevi. Importance of e-learning in higher education: a study. *International Journal of Research Culture Society*, 2(5):84–88, 2018. KerkoCite.ItemAlsoKnownAs: 2339240:W5NFPK3J 2405685:ZMXT6JIP.
- [34] Mate’ Sztipanovits, Kai Qian, and Xiang Fu. The automated web application testing (awat) system. *Proceedings of the 46th Annual Southeast Regional Conference on XX, ACM-SE 46*, pages 88–93, 2008.
- [35] Russell Thackston. Exploring the use of xpath queries for automated assessment of student web development projects. *SIGITE 2020 - Proceedings of the 21st Annual Conference on Information Technology Education*, pages 255–259, 2020.
- [36] Nghi Truong, Paul Roe, and Peter Bancroft. Static analysis of students’ java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, ACE ’04*, page 317–325, AUS, 2004. Australian Computer Society, Inc.
- [37] Jiří Štěpánek and Monika Šimková. Comparing web pages in terms of inner structure. *Procedia - Social and Behavioral Sciences*, 83:458–462, 2013.

# Appendix A

## Installation Guide

This appendix serves as a guide for installing and setting up the Webpal package<sup>12</sup> on your machine. As Webpal is an *npm* package, its installation is very straightforward.

### A.1 Requirements

Webpal runs on a Node.js server and hence, cannot be installed directly on frontend frameworks as some dependencies are incompatible with the client side. Consequently, it is necessary to have both Node.js and Node Package Manager (npm) installed on your machine.

### A.2 Installation

To install Webpal, you need first to create a Node.js project. This can be achieved by executing the following command in your terminal:

```
1 npm init
```

Listing A.1: Start Project

Upon execution, you will be prompted to select certain options such as project name, version, and so forth.

Next, you need to install the Webpal package. This can be done by executing the following command in the terminal, within the directory where your project is located:

```
1 npm install --save webpal
```

Listing A.2: Webpal Installation

---

<sup>12</sup><https://www.npmjs.com/package/webpal>

This command will install Webpal into your newly created project.

### A.3 Importing

To utilize Webpal in your project, it needs to be imported. This can be done as shown below:

```
1 const webpal = require("webpal");
```

Listing A.3: Importing Webpal

For users operating on ECMAScript 6, the import can be executed in the following manner:

```
1 import webpal from 'webpal';
```

Listing A.4: Importing Webpal - ES 6

### A.4 Troubleshoot

In some instances, you may encounter issues when running the Node server containing Webpal, particularly on a Linux machine without a graphical interface. This problem arises due to the *Playwright* requirement to launch headless browsers, and the Linux machine lacking the necessary dependencies for that.

To resolve this issue, navigate to your Linux terminal and execute the following command:

```
1 sudo npx playwright install-deps
```

Listing A.5: Installing Dependencies in Linux Server

# Appendix B

## User Guide

This appendix aims to guide you on how to import files into Webpal and how to appropriately call functions from the Webpal API.

### B.1 Importing Exercise Related Files

Files can be either directly dispatched from a directory to Webpal or their contents can be transmitted through an API.

For importing from a file directory, two dependencies are required: file-system and path. They can be included in your project as follows:

```
1  const fs = require("fs").promises;  
2  const path = require("path");
```

Listing B.1: Importing Webpal

Once the dependencies are included, you can proceed to read the files within a try-catch block. The example below demonstrates how solution data and test data are read for the creation of an exercise:

```
1  const solutionData = await fs.readFile(path.join  
2      (__dirname, "exercises", "solution.json"), "utf8");  
3  const testData = await fs.readFile(path.join  
4      (__dirname, "exercises", "test.js"), "utf8");
```

Listing B.2: Read Files Content

## B.2 Executing Webpal API Functions

An exercise can be created by invoking the `createExercise` function from the Webpal API. The contents of the files read in the previous step are used as arguments for this function. Here's how:

```
1  const id = await webpal.createExercise
2      (solutionData, testData, "Assignment Description");
```

Listing B.3: Create Exercise Function

For details on other functions of the Webpal API, refer to Chapter 4 of this document or consult the README file of Webpal, available at <https://www.npmjs.com/package/webpal>.

# Appendix C

## Stress Test

This appendix contains the script, written in Python, used for stress testing the servers hosting Webpal.

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3 from selenium.webdriver.support.ui import WebDriverWait
4 from selenium.webdriver.support import expected_conditions as EC
5 from selenium.webdriver.chrome.service import Service
6
7 def click_button(driver):
8     button = WebDriverWait(driver, 10)
9         .until(EC.element_to_be_clickable
10             ((By.ID, "executeButton")))
11         )
12     button.click()
13
14
15 def main():
16     num_instances = 20
17
18     drivers = []
19     for _ in range(num_instances):
20         driver = webdriver
21             .Chrome(service=Service("/home/bin/chromedriver"))
22         drivers.append(driver)
23
24     for driver in drivers:
25         driver.get("'aws-server-url'#/main")
26
27     for driver in drivers:
28         click_button(driver)
29
30     for driver in drivers:
31         driver.quit()
```

```
32  
33 if __name__ == "__main__":  
34     main()
```

---

Listing C.1: Stress Test Source Code

# Appendix D

## Extracted Data

This appendix contains the raw data extracted from the log files and the hypothesis testing source code.

### D.1 Auxiliar Tables for Hypothesis Tests

Table D.1: Number of exercises started by each student - Experimental Group

Student ID	Number of Exercises Started
8cc29071-b42c-4acf-b451-fbc2ce97b599	5
ae62e551-fa72-4432-a6aa-94351829acdc	5
d59b615b-31b0-43dc-a101-048efe51661c	4
e5c3374a-d517-4e73-a29f-8920b5510f1d	4
a0df9180-d6e8-405b-a42b-4b4ac3d5a4ff	3
ea0613e3-82b8-4ab2-b22b-e40ae301a77c	3
5d9b1552-4384-4841-9655-29fb4779dff4	2

Table D.2: Number of exercises started by each student - Control Group

Student ID	Number of Exercises Started
25cea295-612d-47f4-a0c5-33b4459f16f3	5
d3efa0b2-289f-45c0-902c-c92f8f56e657	4
fac05b3b-9f08-441a-9c35-59e354bc097e	3
64135f45-96ac-4762-b36a-509ade2f77e3	2
fd9e3074-7bf6-452e-afa3-a325b65eebea	2
358ead22-c542-4d03-9a1e-fc0b26a8bf5	1
73b8e912-fea7-4478-9da3-c168c5d47a3c	1
cb7f8e2f-ca38-48e7-9197-dcd6ae0128d1	1
d2a32b51-43e7-45d5-a54e-b6a8eb5b4315	1

Table D.3: Number of exercises completed by each student - Experimental Group



Student ID	Number of Exercises Completed
5d9b1552-4384-4841-9655-29fb4779dff4	0
8cc29071-b42c-4acf-b451-fbc2ce97b599	0
a0df9180-d6e8-405b-a42b-4b4ac3d5a4ff	1
ae62e551-fa72-4432-a6aa-94351829acdc	3
d59b615b-31b0-43dc-a101-048efe51661c	1
e5c3374a-d517-4e73-a29f-8920b5510f1d	0
ea0613e3-82b8-4ab2-b22b-e40ae301a77c	1

Table D.4: Number of exercises completed by each student - Control Group

Student ID	Number of Exercises Completed
25cea295-612d-47f4-a0c5-33b4459f16f3	1
358ead22-c542-4d03-9a1e-fc0b26a8bfb5	0
64135f45-96ac-4762-b36a-509ade2f77e3	0
73b8e912-fea7-4478-9da3-c168c5d47a3c	0
cb7f8e2f-ca38-48e7-9197-dcd6ae0128d1	0
d2a32b51-43e7-45d5-a54e-b6a8eb5b4315	0
d3efa0b2-289f-45c0-902c-c92f8f56e657	0
fac05b3b-9f08-441a-9c35-59e354bc097e	1
fd9e3074-7bf6-452e-afa3-a325b65eebea	2

Table D.5: Number of attempts by each student - Experimental Group

Student ID	Number of Attempts
5d9b1552-4384-4841-9655-29fb4779dff4	33
8cc29071-b42c-4acf-b451-fbc2ce97b599	9
a0df9180-d6e8-405b-a42b-4b4ac3d5a4ff	43
ae62e551-fa72-4432-a6aa-94351829acdc	32
d59b615b-31b0-43dc-a101-048efe51661c	33
e5c3374a-d517-4e73-a29f-8920b5510f1d	21
ea0613e3-82b8-4ab2-b22b-e40ae301a77c	42

Table D.6: Number of attempts by each student - Control Group

Student ID	Number of Attempts
25cea295-612d-47f4-a0c5-33b4459f16f3	28
358ead22-c542-4d03-9a1e-fc0b26a8bfb5	9
64135f45-96ac-4762-b36a-509ade2f77e3	18
73b8e912-fea7-4478-9da3-c168c5d47a3c	33
cb7f8e2f-ca38-48e7-9197-dcd6ae0128d1	25
d2a32b51-43e7-45d5-a54e-b6a8eb5b4315	14
d3efa0b2-289f-45c0-902c-c92f8f56e657	17
fac05b3b-9f08-441a-9c35-59e354bc097e	28
fd9e3074-7bf6-452e-afa3-a325b65eebea	20

Table D.7: Time spent by each student in exercise "Sum Two Numbers" - Experimental Group

Student ID	Number of Attempts
5d9b1552-4384-4841-9655-29fb4779dff4	1504.12 sec
8cc29071-b42c-4acf-b451-fbc2ce97b599	1995.94 sec
a0df9180-d6e8-405b-a42b-4b4ac3d5a4ff	1101.39 sec
ae62e551-fa72-4432-a6aa-94351829acdc	1599.83 sec
d59b615b-31b0-43dc-a101-048efe51661c	1197.53 sec
e5c3374a-d517-4e73-a29f-8920b5510f1d	609.11 sec
ea0613e3-82b8-4ab2-b22b-e40ae301a77c	2254.29 sec

Table D.8: Time spent by each student in exercise "Sum Two Numbers" - Control Group

Student ID	Number of Attempts
25cea295-612d-47f4-a0c5-33b4459f16f3	482.17 sec
358ead22-c542-4d03-9a1e-fc0b26a8bfb5	791.24 sec
64135f45-96ac-4762-b36a-509ade2f77e3	1409.96 sec
73b8e912-fea7-4478-9da3-c168c5d47a3c	1459.83 sec
cb7f8e2f-ca38-48e7-9197-dcd6ae0128d1	1265.28 sec
d2a32b51-43e7-45d5-a54e-b6a8eb5b4315	1492.14 sec
d3efa0b2-289f-45c0-902c-c92f8f56e657	989.72 sec
fac05b3b-9f08-441a-9c35-59e354bc097e	1214.23 sec
fd9e3074-7bf6-452e-afa3-a325b65eebea	1118.78 sec

## D.2 Hypothesis Testing Code

```

1  import numpy as np
2  from scipy import stats
3
4  # Data

```

```
5     exercises_started_feedback =
6         np.array([5, 5, 4, 4, 3, 3, 2])
7     exercises_started_non_feedback =
8         np.array([5, 4, 3, 2, 2, 1, 1, 1, 1])
9
10    exercises_completed_feedback =
11        np.array([0, 0, 1, 3, 1, 0, 1])
12    exercises_completed_non_feedback =
13        np.array([1, 0, 0, 0, 0, 0, 0, 1, 2])
14
15    attempts_feedback =
16        np.array([33, 9, 43, 32, 33, 21, 42])
17    attempts_non_feedback =
18        np.array([28, 9, 18, 33, 25, 14, 17, 28, 20])
19
20    time_spent_feedback = np.array([1504.123, 1995.942, 1101.388,
21        1599.834, 1197.525, 609.107, 2254.294])
22    time_spent_non_feedback = np.array([482.171, 791.239, 1409.955,
23        1459.83, 1265.281, 1492.139, 989.718, 1214.229, 1118.78])
24
25    data = {
26        'Exercises Started':
27            (exercises_started_feedback,
28             exercises_started_non_feedback),
29        'Exercises Completed':
30            (exercises_completed_feedback,
31             exercises_completed_non_feedback),
32        'Attempts':
33            (attempts_feedback, attempts_non_feedback),
34        'Time Spent on First Exercise':
35            (time_spent_feedback, time_spent_non_feedback),
36    }
37
38    # Significance level
39    alpha = 0.05
40
41    for measure, (feedback, non_feedback) in data.items():
42        print(f"\n=== {measure} ===")
43
44        # Normality check
45        _, p_normality_feedback = stats.shapiro(feedback)
46        _, p_normality_non_feedback = stats.shapiro(non_feedback)
47        print(f'Normality check for feedback group:
48            {p_normality_feedback > alpha}')
49        print(f'Normality check for non-feedback group:
50            {p_normality_non_feedback > alpha}')
51
52        # Homogeneity of variance check
53        _, p_variance_equality = stats.levene(feedback, non_feedback)
```

```
54     print(f'Variance equality check:
55           {p_variance_equality > alpha}')
56
57     # Mann-Whitney U test
58     _, p = stats.mannwhitneyu(feedback, non_feedback)
59     print(f'Mann-Whitney U Test: p={p:.3f}')
60     if p > alpha:
61         print('Same distribution (fail to reject H0)')
62         print(measure)
63     else:
64         print('Different distribution (reject H0)')
65         print(measure)
```

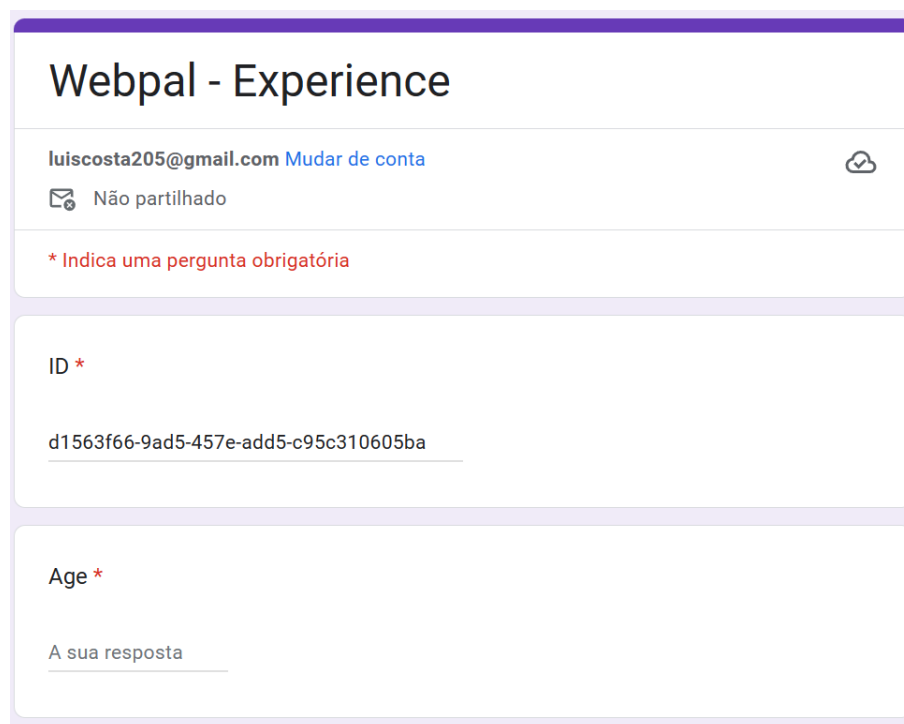
Listing D.1: Hypothesis Test Source Code

## Appendix E

# Questionnaire Structure and Data

This appendix contains all the additional information related to the experience form provided to the students at the end of the validation experiment.

### E.1 Questionnaire Structure



The image shows a screenshot of a Google Form titled "Webpal - Experience". The form is displayed in a mobile or tablet view. At the top, the title "Webpal - Experience" is centered. Below the title, the user's email address "luiscosta205@gmail.com" is shown with a link to "Mudar de conta" (Change account) and a share icon. Below that, it says "Não partilhado" (Not shared). A red asterisk indicates a mandatory question: "\* Indica uma pergunta obrigatória". The first question is "ID \*", with a text input field containing the value "d1563f66-9ad5-457e-add5-c95c310605ba". The second question is "Age \*", with a text input field containing the value "A sua resposta".

Figure E.1: Form - 1st Part

Gender \*

Male

Female

Were the instructions for using the Webpal demo clear and easy to understand? \*

1 2 3 4 5

Strongly Disagree      Strongly Agree

How would you rate the difficulty of the exercises? \*

1 2 3 4 5

Very Easy      Very Hard

Figure E.2: Form - 2nd Part

Was the feedback provided by Webpal helpful for improving your code? \*

1 2 3 4 5

Strongly Disagree      Strongly Agree

Did you enjoy using Webpal for the coding exercises? \*

1 2 3 4 5

Not at all      Very much

If there is one thing you could improve or change in Webpal, what would it be?

A sua resposta

Enviar [Limpar formulário](#)

Figure E.3: Form - 3rd Part

## E.2 Individual Responses - Tables

Table E.1: Responses from the control group

Student ID	Age	Gender	(1)	(2)	(3)	(4)
358ead22-c542-4d03-9a1e-fc0b26a8bfb5	21	Male	4	3	3	3
25cea295-612d-47f4-a0c5-33b4459f16f3	22	Female	4	2	3	3
d3efa0b2-289f-45c0-902c-c92f8f56e657	18	Female	4	1	1	3
64135f45-96ac-4762-b36a-509ade2f77e3	26	Male	3	2	2	3
d2a32b51-43e7-45d5-a54e-b6a8eb5b4315	17	Female	3	3	3	3
cb7f8e2f-ca38-48e7-9197-dcd6ae0128d1	22	Male	4	1	3	3
73b8e912-fea7-4478-9da3-c168c5d47a3c	19	Male	2	1	1	1
fd9e3074-7bf6-452e-afa3-a325b65eebea	20	Male	4	4	4	4
fac05b3b-9f08-441a-9c35-59e354bc097e	23	Male	4	3	3	5

Table E.2: Responses from the experimental group

Student ID	Age	Gender	(1)	(2)	(3)	(4)
8cc29071-b42c-4acf-b451-fbc2ce97b599	19	Male	5	2	4	5
e5c3374a-d517-4e73-a29f-8920b5510f1d	19	Male	4	2	4	5
ae62e551-fa72-4432-a6aa-94351829acdc	20	Female	5	4	4	4
a0df9180-d6e8-405b-a42b-4b4ac3d5a4ff	20	Female	4	2	1	5
d59b615b-31b0-43dc-a101-048efe51661c	18	Female	4	2	1	4
5d9b1552-4384-4841-9655-29fb4779dff4	19	Male	4	4	2	3
ea0613e3-82b8-4ab2-b22b-e40ae301a77c	18	Male	4	4	3	2

### Legend:

- (1) Were the instructions for using the Webpal demo clear and easy to understand?
  - 1 – Strongly Disagree
  - 5 – Strongly Agree
- (2) How would you rate the difficulty of the exercises?
  - 1 – Very Easy
  - 5 – Very Hard
- (3) Was the feedback provided by Webpal helpful for improving your code?
  - 1 – Strongly Disagree
  - 5 – Strongly Agree
- (4) Did you enjoy using Webpal for the coding exercises?
  - 1 – Not at all
  - 5 – Very much

### E.3 Graphs

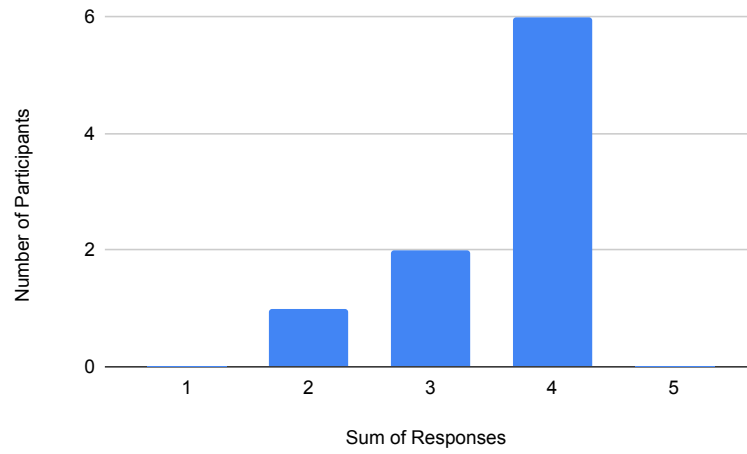


Figure E.4: Were the instructions for using the Webpal demo clear and easy to understand? - Control Group

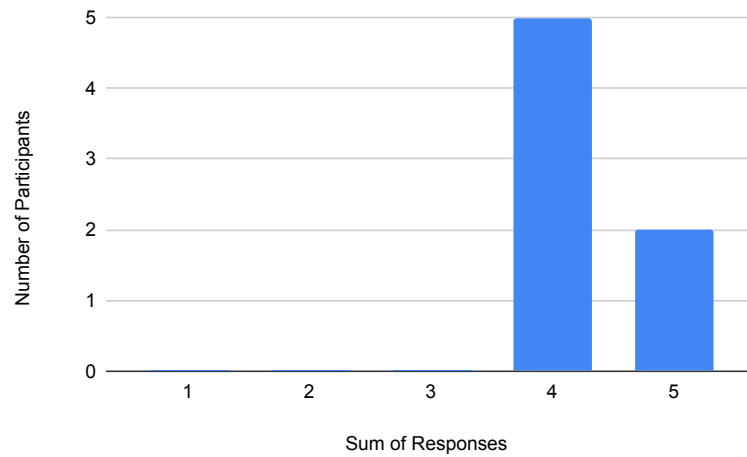


Figure E.5: Were the instructions for using the Webpal demo clear and easy to understand? - Experimental Group



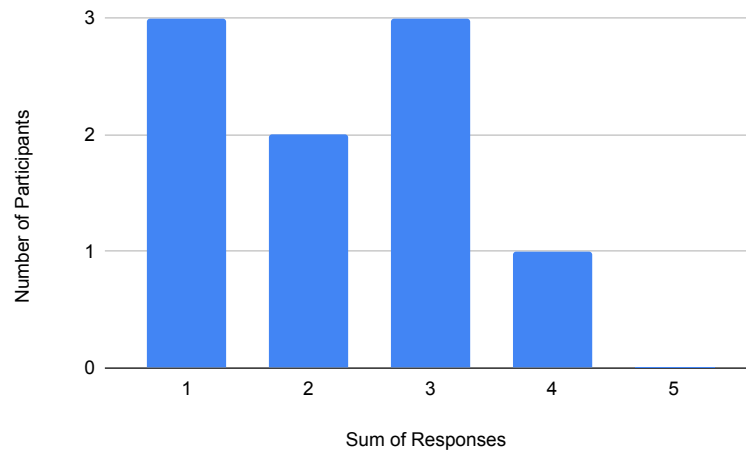


Figure E.6: How would you rate the difficulty of the exercises? - Control Group

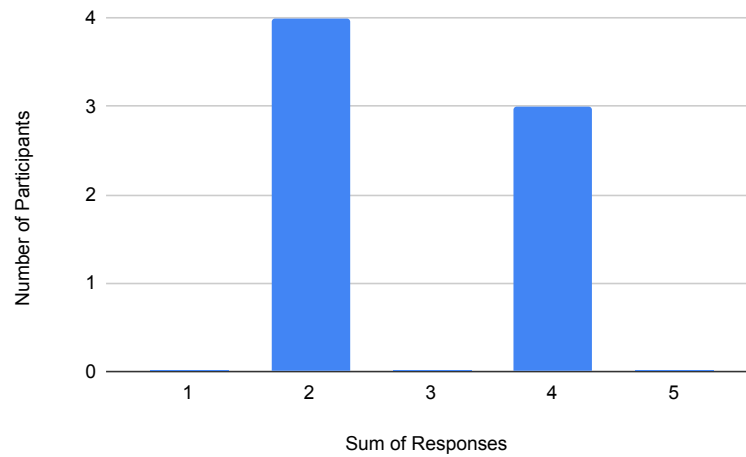


Figure E.7: How would you rate the difficulty of the exercises? - Experimental Group

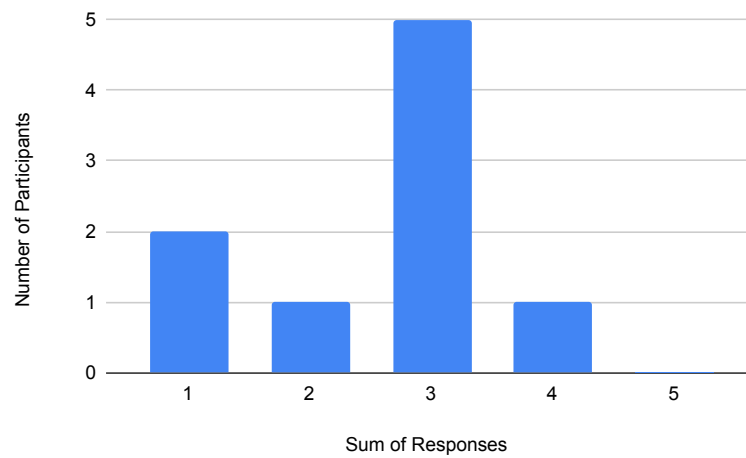


Figure E.8: Was the feedback provided by Webpal helpful for improving your code? - Control Group

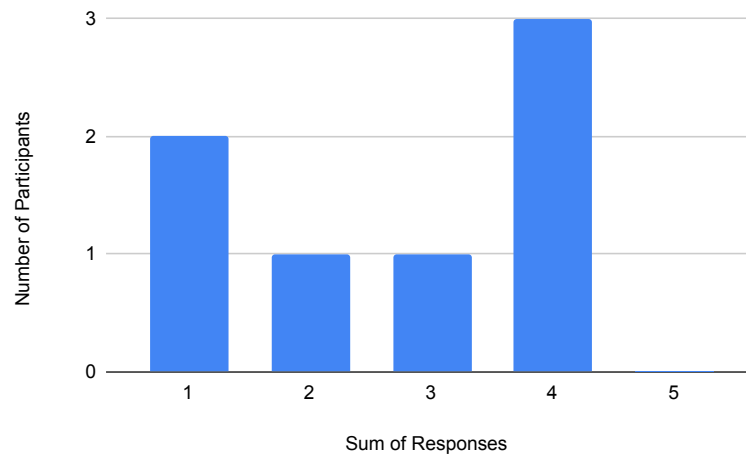


Figure E.9: Was the feedback provided by Webpal helpful for improving your code? - Experimental Group

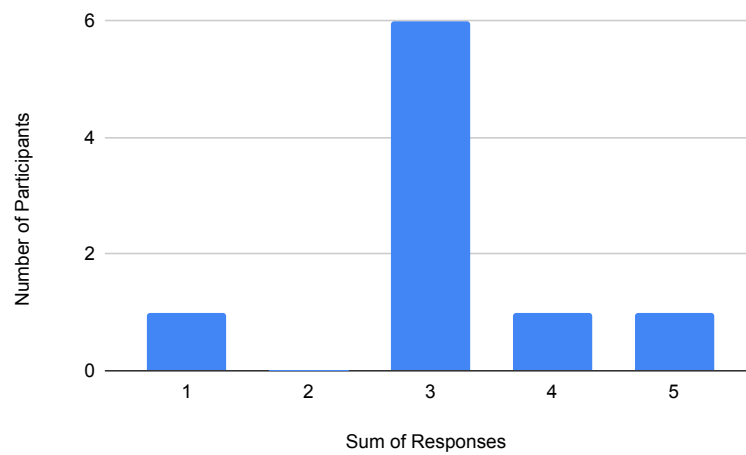


Figure E.10: Did you enjoy using Webpal for the coding exercises? - Control Group

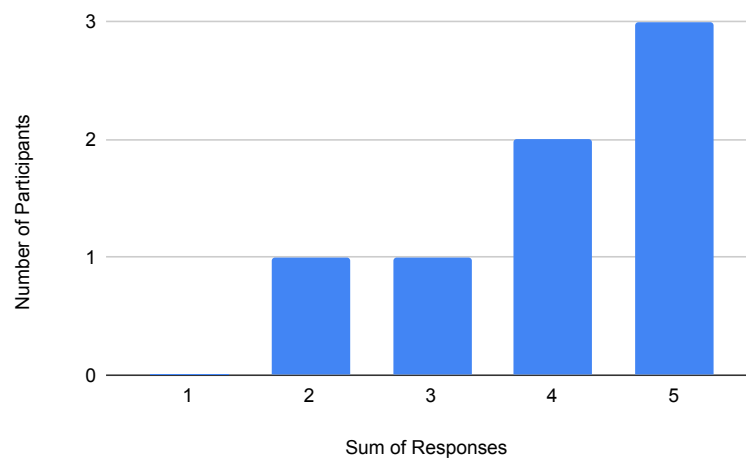


Figure E.11: Did you enjoy using Webpal for the coding exercises? - Experimental Group

## Appendix F

# Correlation Matrix Generation

This appendix contains the script, written in Python, used to perform a correlation analysis on the data.

```
1  import pandas as pd
2  import numpy as np
3
4  # Data
5  form_data = pd.read_csv('form-data.csv')
6  log_data = pd.read_csv('log-data.csv')
7  merged_data = pd.merge(form_data, log_data, on='studentID')
8
9  # Remove studentID from data
10 numeric_data = merged_data.drop(columns=['studentID'])
11
12 # Generate Correlation Matrix
13 correlation_matrix = numeric_data.corr()
14 pd.set_option('display.precision', 3)
15
16 print(correlation_matrix)
```

Listing F.1: Correlation Matrix Source Code