



INSTITUTO
UNIVERSITÁRIO
DE LISBOA

A Hint Generation System for Introductory Programming Exercises in Java

Jorge Alexandre da Silva Gonçalves

Master of Computer Engineering

Supervisor:

PhD, André Leal Santos, Assistant Professor

Iscte - Instituto Universitário de Lisboa

Co-Supervisor:

PhD, Joana Martinho de Almeida Costa Pardal, Assistant Professor

Iscte - Instituto Universitário de Lisboa

November, 2022



TECHNOLOGY
AND ARCHITECTURE

Department of Information Sciences and Technologies

A Hint Generation System for Introductory Programming Exercises in Java

Jorge Alexandre da Silva Gonçalves

Master of Computer Engineering

Supervisor:

PhD, André Leal Santos, Assistant Professor

Iscte - Instituto Universitário de Lisboa

Co-Supervisor:

PhD, Joana Martinho de Almeida Costa Pardal, Assistant Professor

Iscte - Instituto Universitário de Lisboa

November, 2022

*For the women who follow me on this journey,
my two daughters Carmo and Maria
and my wife Soraia.*

Acknowledgements

I would like to thank Prof. André Santos Santos and Prof. Joana Pardal, for all the support provided in this project. A special thanks to Prof. André Santos, for your mentoring, and your guidance, and for every minute spent during the last year helping me with this challenge.

Special thanks to my parents, their support, for raising me and giving me the best guidance to achieve my goals and be the person I am today. To my sisters, thank you, even far away, for all your support and for always being so happy with my achievement.

Finally, a very special thanks to my daughters Maria and Carmo, and my wife Soraia, who were with me from the first minute, giving support, courage and unconditional love. They are only people who really know how difficult this long journey has been, and without them, this objective could not be achieved.

Resumo

Ensinar e aprender programação é um verdadeiro desafio para professores e alunos. É normal os alunos que começam a codificar pela primeira vez sentirem-se bloqueados ao tentar resolver um exercício de programação, ficando dependentes de ajuda de professores ou alguém mais experiente para progredir. Porém, o aluno que trabalha de forma autónoma pode desenvolver uma atitude proativa em relação à sua educação. Com a atual evolução da tecnologia e os acontecimentos mundiais, o ambiente remoto para educação é uma realidade, e logo, ferramentas que possam auxiliar os alunos, orientando-os quando precisam, podem ser importantes para dar-lhes a autonomia necessária. Esta dissertação propõe um sistema de recomendação que pode ser integrado em qualquer plataforma de exercícios para reconhecer um código parcialmente escrito e gerar dicas confiáveis e úteis (combinadas com feedback positivo) para ajudar no progresso sempre que o aluno precisar de apoio. As dicas são geradas com base numa solução previamente definida pelo professor. Esta dissertação descreve a implementação de tal sistema de recomendação, baseado em árvores de sintaxe abstrata e distância de edição das mesmas. Foi desenvolvido um protótipo de sistema composto por uma API REST e um frontend Web como materialização da abordagem proposta.

Palavras-chave: Introdução à programação, geração de dicas, sistemas de recomendação, trabalho autónomo.

Abstract

Teaching and learning programming is a real challenge for teachers and students. Students starting to code for the first time, feeling stuck when trying to solve a programming exercise is normal, and teachers should support them in this case. The student who works autonomously can create a proactive attitude towards his education. With the current evolution of technology and world events, the remote environment for education is a reality, so tools that can help students, guiding them when they need it, can be important to give them the necessary autonomy. This dissertation proposes a recommendation system that can be integrated into any exercise platform to recognize a partially written code and generate reliable and useful tips (combined with positive feedback) whenever the student needs support, based on a solution previously set by the teacher. This dissertation describes the implementation of such a recommendation system, based on abstract syntax trees and tree editing distance. A system prototype composed of a REST API and a Web frontend was developed as a materialization of the proposed approach.

Keywords: Introductory programming, hint generation, recommender systems, autonomous work.

Contents

Acknowledgements	iii
Resumo	v
Abstract	vii
List of Figures	xi
List of Tables	xv
Code Listings	xvi
Abbreviations	xvii
Chapter 1. Introduction	1
1.1. Motivation	1
1.2. Context	2
1.3. Research Questions	3
1.4. Objectives	3
1.5. Research Methodology	3
Chapter 2. Literature Review	5
2.1. Four-Component Instructional Design (4C/ID) Model	5
2.2. Existing Solutions For Learning Tools	8
2.3. A Tool For Any Programming Exercise Platform	12
Chapter 3. Automated Hint Generation	13
3.1. Overview of the Approach	13
3.2. Solution Strategy	15
3.3. Abstract Syntax Tree	16

3.4. Tree Edit Distance	18
3.5. Nodes Matching	20
3.6. Nodes Pairs Analysis And Hints Generation	22
3.7. Exercise Scenario Walkthrough	32
3.8. Limitations	37
Chapter 4. Prototpe Design and Implementation	39
4.1. High Level System Architecture	39
4.2. User Interface	41
4.3. Exercise Hint API	42
Chapter 5. Conclusions and Future Work	53
Bibliography	55

List of Figures

1	Design Science Research (DSR) methodology process model [11]	4
2	Overview of 4C/ID model	6
3	Parsons Problem - A 2D Parsons problem with paired distractors [19]	8
4	Generated questions after a code submission. [21]	9
5	The framework offering recommendations using markers based on the best matching target implementation. [22]	10
6	Javardise editor. Two placeholders (gray) yet to be filled with expressions. [23]	11
7	Trace Generator - Questions displayed tracing the code $b = a * 2$ [24]	12
8	Example of wrong solution to the maximum problem	14
9	Example of incomplete solution	14
10	Example of positive feedback provided for a solution	15
11	High Level of Hint Generation Process	15
12	Abstract Syntax Tree (AST) representation of Is Even Exercise (Listing 1). The number on each element represents the root's key.	16
13	AST representation of Is Even Wrong Solution (Listing 2). The number on each element represents the root's key.	17
14	Tree Edit Distance (TED) result after comparison the AST from code submitted (Listing 2) versus solution (Listing 1).	19
15	Gale-Shapley algorithm applied to TED matrix	21
16	Hint message for a wrong operator	22
17	Types Currently Handled	23

18 Combining Analysis	31
19 Exercise Setup Example	33
20 Sequence Diagram of Exercise Hint Platform	33
21 Code Correction Hint	34
22 Next-Step Hint	34
23 Hint Based in Variable Roles	35
24 Different semantics to correctly solve the same solution.	36
25 Test Data Results	36
26 Semantic equivalent of the solution considered wrong.	37
27 High Level System Architecture	40
28 Main Page	41
29 Exercise Code Editor	42
30 Exercise Hint API Architecture	43
31 Available Endpoints	44
32 Example of structure generated by Strudel for Is Even exercise (Listing 1)	51

Code Listings

1	Is Even Exercise Example	16
2	Is Even Exercise Wrong Solution Example	17
3	Get All Exercises Request and Response	45
4	Get Exercise by identifier Request and Response	46
5	Exercise compilation	46
6	Test solution with an unsuccessful execution	47
7	Solution compilation	48
8	Hints Request	48
9	Hints Response	49
10	HintGeneratorService method to generate hints (Kotlin)	50

List of Tables

1	Value type hints	23
2	Reference type hints	25
3	Return type hints	25
4	Hints based in the number of parameters/variables	26
5	Hints based in variable roles	27
6	Expressions Examples	28
7	Hints based in expressions	30
8	Examples of semantic equivalences	30
9	Some Example of Positive Feedback Provided	32

Abbreviations

API: Application Programming Interface. 39, 41, 44, 54

AST: Abstract Syntax Tree. xi, 15-17, 19

CD: Continuous Delivery. 39, 40

CI: Continuous Integration. 39, 40

DDD: Domain Driven Design. 43

DSR: Design Science Research. xi, 3, 4

HTTPS: Hypertext Transfer Protocol Secure. 40

IDE: Integrated Development Environment. 2, 13

JSON: JavaScript Object Notation. 52

PaaS: Platform as a Service. 39

QLC: Questions about Learns' Code. 9, 12

REST: Representational State Transfer. 41

TED: Tree Edit Distance. xi, 15, 18, 19, 21

UI: User Interface. 39, 41, 53

URL: Uniform Resource Locator. 45

CHAPTER 1

Introduction

1.1. Motivation

Programming is an essential skill that all computer science students must master. Teaching and learning programming is a real challenge for teachers and students, respectively. In introductory programming courses, students exhibit various difficulties in syntactic knowledge, conceptual knowledge, and strategic knowledge [1].

Syntactic knowledge error is common in introductory programming courses. Some studies based on Java programming have identified that the most frequent error is mismatched parentheses, brackets, or quotation marks [2], unsolvable symbols (e.g. failing to declare variable before using it), missing semicolons and using illegal start of expressions [3].

The conceptual knowledge is the misunderstanding of programming constructs or machine operation. Studies reveal that students may fail to understand that variables can only hold one value at a time or that the order of statements assigning values to variable is important [1], the students don't understand where the data come from and how it is stored in memory [4]. Conditionals and looping construct are another difficult concept. Some students even mistakenly think that if the condition of an *if-statement* is *false*, the execution of the whole program stops or what the scope of loop is, which the lines will be repeated, how many times the code inside the loop will be executed, etc [5].

Strategic knowledge of programming, which is also labeled as conditional knowledge in cognitive psychology, refers to expert-level knowledge about planning, writing, and debugging programs for solving novel problems using syntactic and conceptual knowledge [1]. The term strategy is a generic term exemplified by problem solving ideas such as *plans*, *patterns*, *algorithms* and other methodologies, together with means of integrating these ideas to form a single solution [6]. Raadt concludes in his study that

“programming knowledge is a prerequisite for programming strategies” [6], therefore, without adequate syntactical and conceptual knowledge, students will have difficulties, for example to choose an appropriate loop construct in a specific context or to test and debug programs.

Learning programming is a time-consuming and difficult process as it requires students to master different cognitive skills, requires effort and dedication from beginners, and can therefore make students feel frustrated and abandon their Computer Science programs. Teachers of introductory programming courses try to discover strategies that facilitate student learning, studies have been carried out on the most suitable language for teaching computer programming, as well as the most suitable integrated development environments (Integrated Development Environment (IDE)) [7], or solutions using microworlds (e.g. *LOGO*, *Alice Project* or *Scratch*), created with the purpose of helping and motivating students to learn programming [8].

1.2. Context

The first steps in learning to program can be difficult, in the courses for introduction to programming taught by universities, teaching assistants are required to manually go through each student submission and provide qualitative feedback outlining exactly what is wrong and how to fix it. Novice students feeling stuck when trying to implement a particular programming task is normal, and teachers should be supportive in this case, but facing a classroom with students all asking different questions at the same time can be overwhelming. Consequently, due to lack of time, teachers may fail to provide the best guidance to new students.

There are a growing number of tools to generate data-driven hints, next-step hints that suggest how students should edit their code to resolve errors and make progress. The focus of these tools is to be a system that can reliably provide hints to students and how much data is necessary to do so [9]. The pioneering Hint Factory [10], for example, relies on previously collected data to fill in possible pathways, giving help to students based on what other students have done before them, this approach may be limited as it cannot provide help for students in states that have not been seen before, and it

does not ensure that all students will always be able to get a hint. This dissertation is an attempt to provide a prototype that can provide reliable hints to students with minimal data, in this case the solution to exercises provided by the teacher.

1.3. Research Questions

Taking into consideration many misconceptions and other difficulties that students have when learning programming, and the purpose of providing teachers with a tool that facilitates learning programming, the research questions that motivate an in-depth analysis are as follows:

- What kind of hints can be used to provide useful student guidance in introductory programming exercises?
- How to compute reliable hints for assisting the process of solving programming exercises based in a single template solution?

1.4. Objectives

The main purpose of this research is the design and implementation of a platform that allows teachers/instructors to configure sequences of programming exercises that provide an automated technique to provide hints to students, not only to help correct almost finished programs, but also to help them when they have difficulty finding a good starting point and provide feedback when a step in the exercise is correct, so that they achieve the objectives proposed by the exercise.

1.5. Research Methodology

The research methodology used for reaching the objectives proposed for this project is based on the Design Science Research (DSR) methodology process model that incorporates six activities in a nominal sequence (Figure 1) [11]. This methodology follows a problem-centered approach, and is oriented to the creation of artifacts [12].

Adapting this model to the context of this dissertation, starting first with “Problem Identification & Motivation”, activity described in the motivation section of this document, where the existing problem that gave rise to the idea of this research is described.

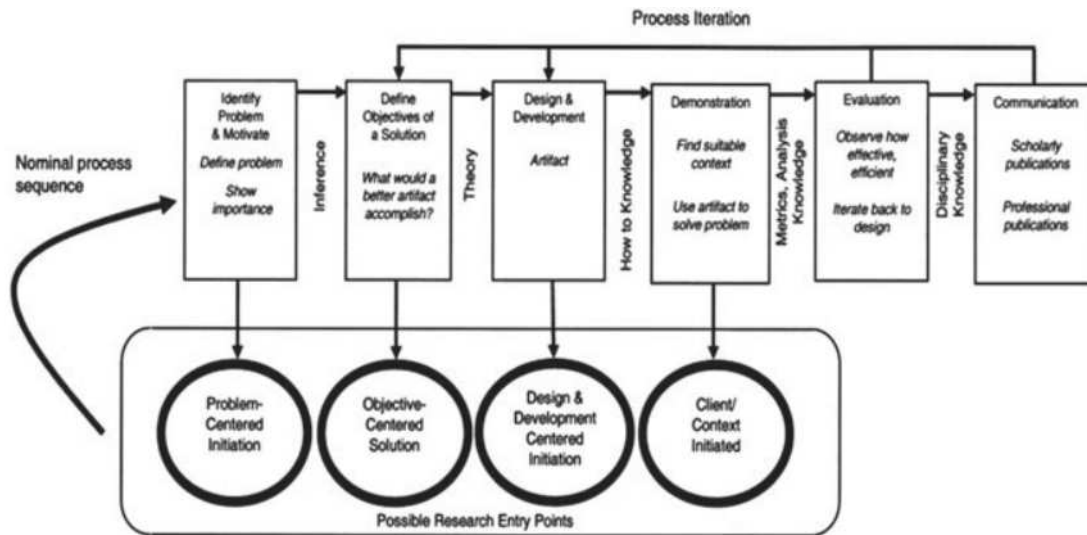


Figure 1. DSR methodology process model [11]

Second, “Objectives of a Solution”, the research objectives were defined, and the proposal of a solution is a configurable exercise tool that, following the 4C/ID principles, will provide an efficient and effective programming teaching process in the acquisition of different technical skills. Then we have the “Design & Development” activity, where the functionality and architecture of the artifact are defined, as well as its implementation. In the, “Demonstration” activity, the effectiveness of the generated artifact is demonstrated to solve the problem, which consists of experimentation in a given context [11][12]. The “Evaluation” is the activity that involves comparing the “Objectives of a Solution” to actual observed results from use of the artifact in the “Demonstration”, at the end of this activity based on results we can decide to iterate back to “Design & Development” to try improve the effectiveness or to continue to the next activity [11][12]. Finally, “Communication”, in this activity, the problem and its importance, and the usefulness of the artifact are presented through the writing of the dissertation, that is, when the work is published and/or presented [11][12].

CHAPTER 2

Literature Review

The strategy used to find relevant literature, as an alternative for searching databases, was the snowball approach. Snowballing consists of defining an start set of relevant articles and then identifying additional articles using the list of references and citations for each article. It is a reliable and efficient way to carry out systematic literature studies [13]. The first step is to identify an initial set of articles on a topic [13], to build it, literature was searched on the b-on platform ¹ and Google Scholar ² using the keywords including “4C ID model” or “four components instructional design model”, “recommendation framework”, “hints”, “programming learning”, “learning evaluation”, “education”, and the most relevant were selected. From the starter set, was used the reference list to locate additional relevant articles.

2.1. Four-Component Instructional Design (4C/ID) Model

The 4C/ID is a instructional model suitable for teaching complex skills or professional competencies [1]. It is composed of four components (Figure 2):

- a) *learning tasks*, which are the core of instructional process, they are whole tasks of increasing levels of difficulty, preferably based on real-world tasks/experiences;
- b) *supportive tasks*, consist of the knowledge prerequisite for performing tasks, they help students by providing the link between what they already know and what they need to know to perform the non-routine aspects of learning tasks;
- c) *procedural information*, also called just-in-time information, provides to the students ‘how-to’ instructions to perform the routine aspects of learning tasks;

¹<https://www.b-on.pt/>

²<https://scholar.google.com/>

d) *part-task practice*, is required when the learning tasks do not provide the required amount of tasks, it allows students to practice skills that require a high level of automaticity;

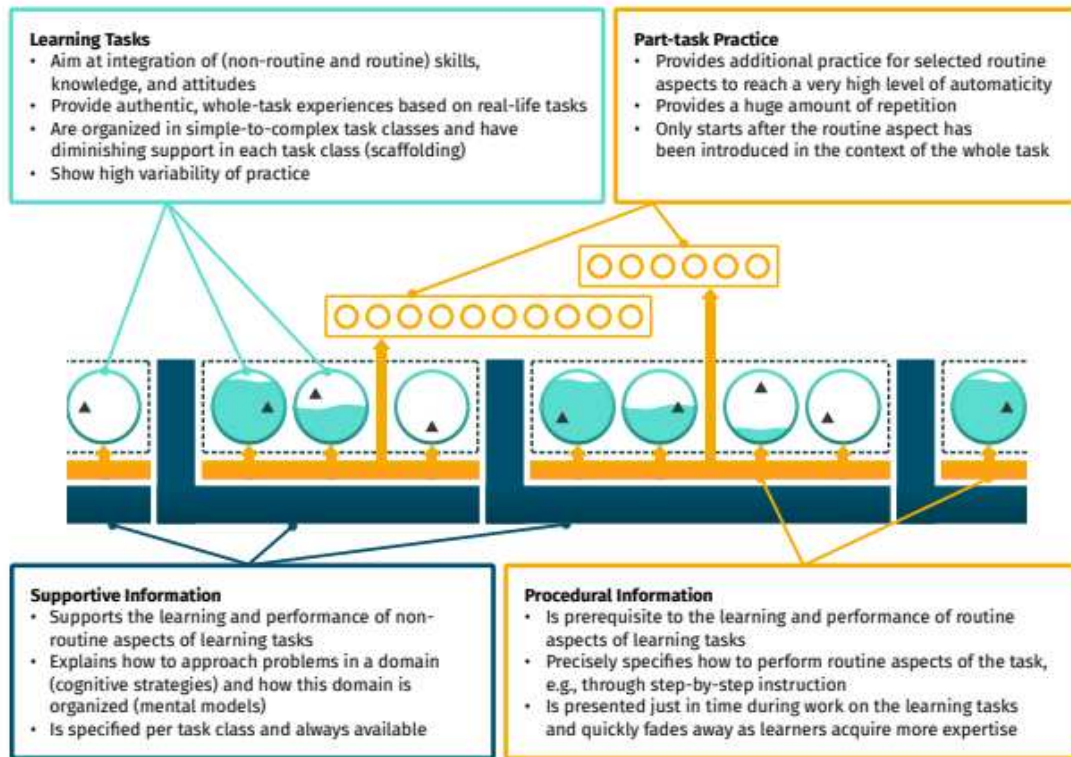


Figure 2. Overview of 4C/ID model ^a

^a<https://www.4cid.org/>

The learning task consists of completing, preferentially, real-life based tasks that focus on authentic, hands-on tasks such as project-based training, event methodology, problem-based learning, and competency-based learning. It is what distinct this model of others models, that in most cases, divide the overall learning tasks into sub-tasks of easier achievement (e.g. Programmed Learning) [14]. Software developed following these principles can serve to develop software skills such as learning, organization and analytical thinking [15].

The use of the 4C/ID model has increased in several areas and contributed to the improvement of online and face-to-face learning environments, there has been high quality research on this model in several domains such as health and medical education,

problem-solving and higher education [16]. Task-centric ID models such as 4C/ID create learners who are able to transfer their knowledge from learning to the professional environment, provide educational programs that correspond to market demands, producing students endowed with knowledge, skills (e.g. logical reasoning, problem solving, analytical thinking, etc.) and attitudes that perform high-performing tasks [17]. Technology-based instruction used without reference to instructional design principles that flow from human cognition is likely to be haphazard in its effectiveness [18]. Due to the complexity of this type of learning tasks, it is crucial to accurately manage the cognitive load imposed on students. Cognitive load theory provides instructional recommendations based on our knowledge of human cognition. To manage these types of cognitive load imposed on students, generated by the complexity of learning tasks, the 4C/ID model suggests the following specific strategies in terms of task sequencing and information presentation, summarized in a research that reviewed the use and effect on performance of educational programs developed with the 4C/ID model [16]:

- a) *sequence the learning tasks from simple to complex*, to avoid cognitive overload for students, the first task should be the simplest and the complexity should increase for each task. The last task should be the most complex, including real-life tasks;
- b) *sequencing learning tasks with decreasing student support*, another way to prevent student cognitive overload is to decrease support for each task, from high built-in support to conventional unsupported tasks;
- c) *sequence learning tasks in a variable order*, research indicates that high variability of practice affects the development of schemata and promotes subsequent learning transfer;
- d) *present supporting information before students start working on learning tasks and make it accessible to students during practice*;
- e) *present procedural information when students need it*, to reduce ineffective cognitive load;

2.2. Existing Solutions For Learning Tools

During the process of learning programming, many skills can be acquired such as logical thinking, algorithm training, problem solving skills and analytical thinking skills. To avoid cognitive overload, in the 4C/ID model, students start from simple learning tasks and, as their knowledge increases, they work on more complex tasks, support and guidance will gradually decrease in a process known as “scaffolding”³. We have researched possible solutions that can be used in the implementation of exercises applying the concept of “scaffolding”.

2.2.1. Parsons Problem

Parsons problems are a type of code completion practice problem in which the learner must place blocks of mixed up program code in the correct order 3), that should have a lower cognitive load than a problem that requires the student to write a code from scratch [19].

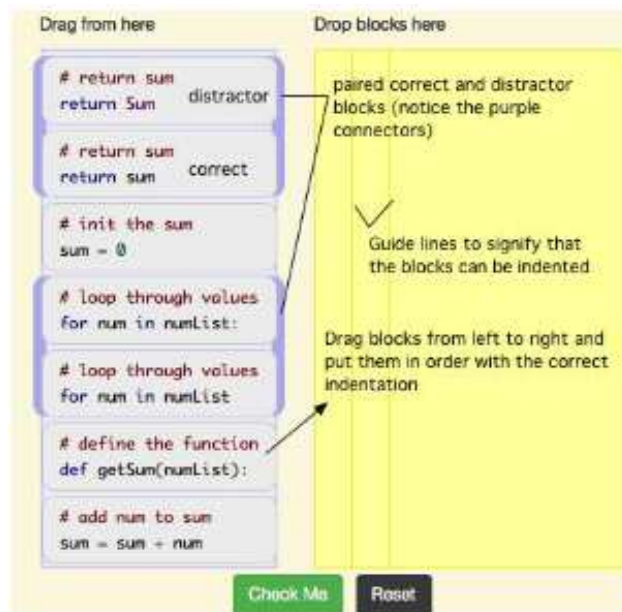


Figure 3. Parsons Problem - A 2D Parsons problem with paired distractors [19]

³<https://www.4cid.org/>

Parsons problems was design to following principles: maximising student engagement, separating logic from syntax, providing immediate feedback, and modelling good design [20].

2.2.2. Automatic Questions About Learners' Code (Questions about Learns' Code (QLC)s) Generation

Automated QLCs are questions about program code that a student has written, generated automatically from an analysis of the code produced. QLCs can encourage students to reflect on their code and their programming knowledge, and can also be valuable for teachers to verify students' programming knowledge [21].

Answer to the following questions about your code:

1. How many parameters does function `fact` have?

Confidence level for the given answer:

2. Is function `fact` recursive?

True

False

Confidence level for the given answer:

3. What are the parameters of function `fact`?

Confidence level for the given answer:

Figure 4. Generated questions after a code submission. [21]

2.2.3. Automated Framework for Recommending Program Elements to Novices

Zimmerman and Rupakheti [22] created a Java framework to integrate with the Eclipse IDE to recommend specific code edits relevant to students' problems when they

are trying to solve a specific programming exercise. The framework offers recommendations using markers based on the best match between the student's solution and the teacher's suggested solution (Figure 5).

```
40 public int source(int[] a, int key) {  
5     int lo = 0;  
6     int hi = a.length - 1;  
7     while (lo <= hi) {  
8         int mid = lo + (hi - lo) / 2;  
9         BrChange '+' to '-' (key < a[mid]) hi = mid + 1;  
10        else if (key > a[mid]) lo = mid + 1;  
11    }  
12    return -1;  
13 }
```

(a) Source code written by a novice

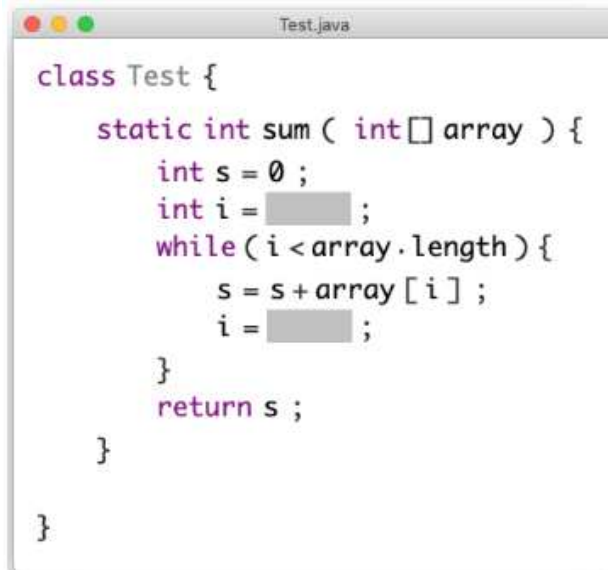
```
39 public int target(int array[], int value) {  
4     int left = 0;  
5     int right = array.length - 1;  
6     while(left <= right) {  
7         int middle = left + (right - left) / 2;  
8         if(value < array[middle])  
9             right = middle - 1;  
10        else if(value > array[middle])  
11            left = middle + 1;  
12        else  
13            return middle;  
14    }  
15    return -1;  
16 }
```

(b) Target code provided by an instructor

Figure 5. The framework offering recommendations using markers based on the best matching target implementation. [22]

2.2.4. Structured Code Editor

The purpose of a structured code editor “is to constrain editing to syntactically valid program code, that is, the modifications ensure that the source code always conforms to grammar” [23]. For example, Javardise consists of a Java structured code editor whose purpose is to aid programming pedagogy, the main purpose is to avoid the syntax barrier in the early stages of introductory programming, students will focus on semantics and not waste time with the syntax obstacles [23].

The image shows a window titled "Test.java" containing the following Java code:

```
class Test {  
    static int sum ( int[] array ) {  
        int s = 0 ;  
        int i =  ;  
        while ( i < array . length ) {  
            s = s + array [ i ] ;  
            i =  ;  
        }  
        return s ;  
    }  
}
```

Two gray rectangular placeholders are visible in the code, one after the assignment of `i` and one after the increment of `i` in the `while` loop.

Figure 6. Javardise editor. Two placeholders (gray) yet to be filled with expressions. [23]

2.2.5. Automated Code Tracing

The automated code tracing generates questions about the code executed by the student. The questions are generated for the student to explain the code, data type, execution result value or variable value (Figure 7). The main objective is to provide students with the ability to read and understand code, which is a fundamental skill in programming [24].

Address	Name	Type	Value
0	a	int	12

Order	Explanation	Code Executed	Result	Type of Result
1	Value loaded from variable	a	12	int
2	literal constant in code	2	2	int
3	Multiplication	12 * 2	24	int
4	Assignment	b = 24	24	int

Address	Name	Type	Value
0	a	int	12
1	b	int	24

Figure 7. Trace Generator - Questions displayed tracing the code $b = a * 2$ [24]

2.3. A Tool For Any Programming Exercise Platform

Based on the research carried out, this dissertation focuses on the development of a tool for platforms for introductory programming exercises, some of the tools and/or ideas above can be used to apply the concept of “scaffolding”. parsons or a structured code editor would be interesting solutions to implement exercises where the task complexity is low, the automatic generation of QLCs or automated code tracing exercises would be the most suitable solution for intermediate levels of complexity. An *environment with built-in hints* that suggests what is needed to solve based on a basic solution, combined with *intermediate positive feedback* when an exercise step is correct. It could be an important tool to be part of a “scaffolding” process, so this dissertation will be based on these two concepts to provide a prototype to be integrated into any platforms for introductory programming exercises, regardless of the programming language, providing guidance for students during their tasks to solve programming problems and also providing an computer-assisted feedback which is one of most effective forms of feedback provide cues or reinforcement to learners [25].

CHAPTER 3

Automated Hint Generation

This chapter will present the basis for an automated hint generation system that implements an approach that provides hints by structurally comparing the student's solutions with the solution model created/provided by the teacher. This comparison is made for each element that can compose a program, for example, *return type*, *parameters*, *variables*, statements like *for-statement*, *if..else-statement*, *return-statement*, etc. This approach aims to guide students and help them when they feel stuck, providing hints that can help them achieve their goal.

3.1. Overview of the Approach

To demonstrate what this approach consists of, let's consider a simple problem to find the maximum integer value contained in an array of integers. This problem teaches concepts of conditionals and iteration over arrays. For this problem, of course, some students struggled with many low-level Java semantic issues such as array indexing and iteration limits. The IDEs have many time-saving features for developers, such as intelligent code completion and automated code generation, which eliminates the need to type complete strings. IDEs analyze the code as it is written, so errors caused by human error are identified in real time, but all these features will only help the student to overcome some syntax difficulties, and thus, perhaps, focus on the problem.

In the example (Figure 8) the solution may be obvious, but it may also be a problem that some students can only overcome with the help of the teacher. New programming students can face many challenges when learning to program, in addition to syntax issues, so the goal is to help newbies in their learning process by recommending specific code edits relevant to their problems. In the example bellow (Figure 8) we exemplify a solution that is almost correct, what if the student feels stuck at the beginning?

```
1 v class Solution {
2 v static int max(int[] array) {
3 v int m = array[0];
4 v for (int i = 0; i < array.length; i++){
5 v
6 v
7 v
8 v }
9 v return m;
10 v }
11 v }
```

💡 You should try to use a different literal value, it can affect the result of your solution

Figure 8. Example of wrong solution to the maximum problem

Among other relevant help, the application will provide students with some guidance in designing the solutions (Figure 9). A student writes some code for a proposed problem, when he feels stuck he will press help button in the application. The application will analyze his code, compare it with the professor's proposed solution and recommend the next step using alert markers. The reduced number of visible hints per request (one) is important to reduce the cognitive load. That way, novices will work independently while getting help overcoming their learning barriers, giving teachers more time that can be spent providing better guidance on more complex tasks.

```
1 v class Solution {
2 v static int max(int[] array) {
3 v
4 v
5 v }
```

💡 You will need to declare some variables in your solution.

📖 Variables are containers for storing data values. To declare (create) a variable, you must specify the type and assign it a value:

E.g.
int variableName = 1;

Figure 9. Example of incomplete solution

The tool also provides positive feedback for all lines that have been well formed (Figure 10), the combination of hints and feedback can improve students' learning. Hattie and Timperley [25] noted that targeted feedback at the right level can help students

```

1  class Solution {
2  static int max(int[] array) {
3      ✓ Good, this parameter is correct.
4      ✓ Great, return type is correct.
5      ✓ The number of parameters is ok, well done.
6
7      }
8  }
9  return m;
10 }
11 }

```

Figure 10. Example of positive feedback provided for a solution

understand, engage, or develop effective strategies for processing the information they are trying to learn.

3.2. Solution Strategy

Figure 11 gives us an overview of the approach to generating hints. The process begins by parsing both solution versions to produce an AST (Section 3.3) of each code solution. Each node in the generated AST represents one of twelve nodes type that we have predefined to be parsed differently and to provide better accuracy when we try to combine each node. To find a set of differences, we compare the two AST, obtaining their TED matrix (Section 3.4), from the produced matrix a new matrix is produced which contains the "stable match" of each node and finally each pair will be compared based on its own node type (there are no pairs with two different node types) to give hints based on their differences.

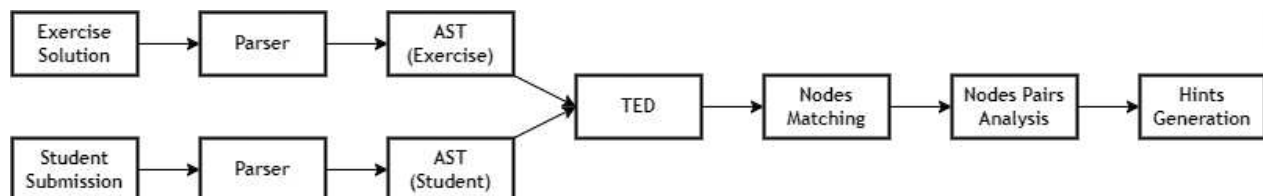


Figure 11. High Level of Hint Generation Process

3.3. Abstract Syntax Tree

An Abstract Syntax Tree (AST) is a tree representations of code created by the compiler during the process of converting the program from text to binary code, they are a fundamental part of the way a compiler works.

```
1 class Exercise {  
2     static boolean isEven(int number)  
3     {  
4         boolean isEven = false;  
5         if (number % 2 == 0)  
6             isEven = true;  
7         return isEven;  
8     }  
9 }
```

Listing 1. Is Even Exercise Example

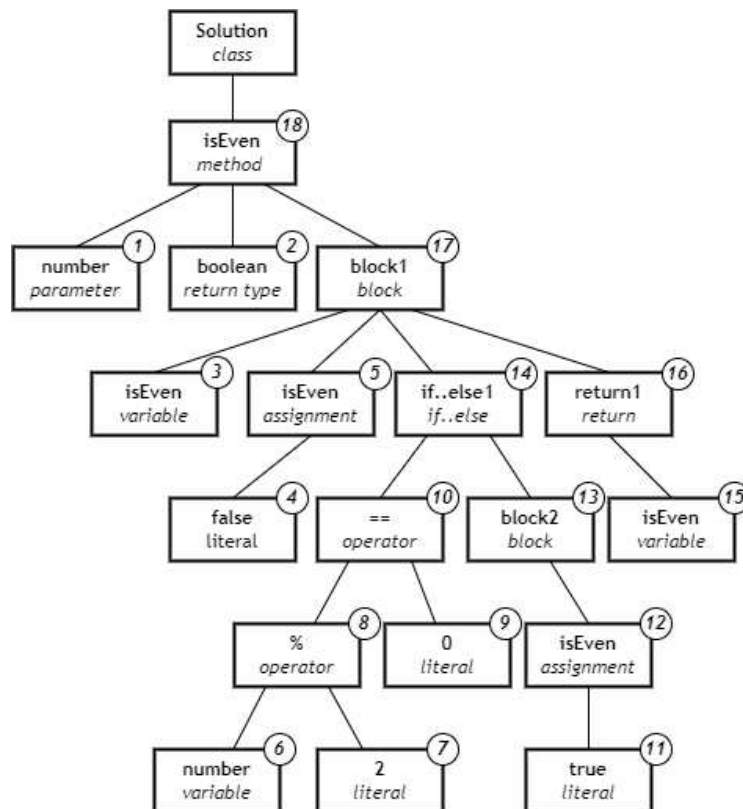


Figure 12. AST representation of Is Even Exercise (Listing 1). The number on each element represents the root's key.

There are numerous uses of AST with application in compilers, as abstract syntax trees are data structures widely used in compilers to represent the structure of program code. Once the ASTs were stored as data structures, the tree's edit distance can be used to compare them directly. (Figure 12)

```

1 class Submission {
2     static boolean even(int number)
3     {
4         boolean e = false;
5         if (number / 2 == 0)
6             return true;
7     }
8 }

```

Listing 2. Is Even Exercise Wrong Solution Example

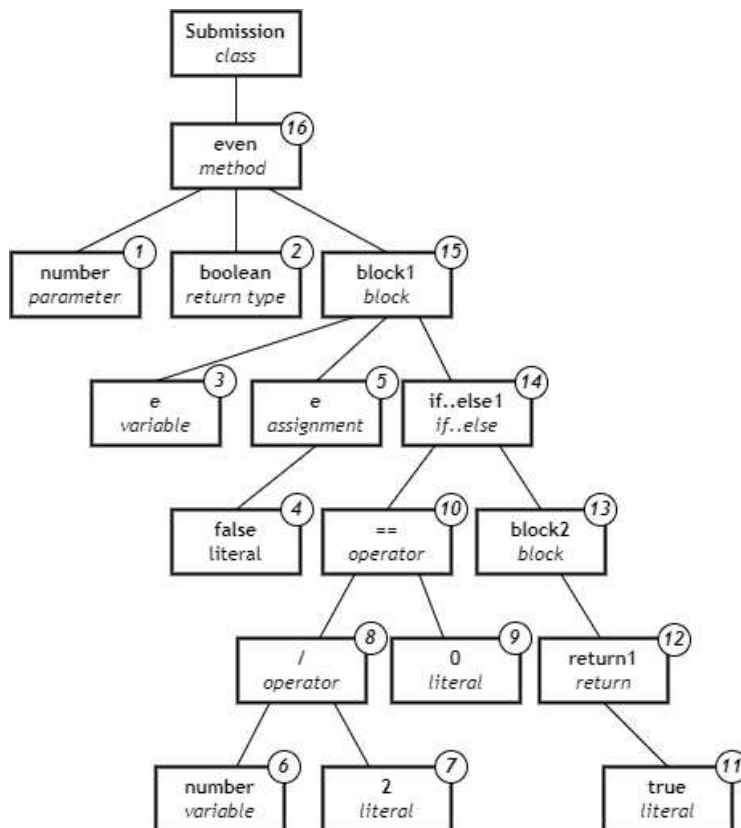


Figure 13. AST representation of Is Even Wrong Solution (Listing 2). The number on each element represents the root's key.

3.4. Tree Edit Distance

Tree Edit Distance (TED) refers to the minimum number of node insertions, node deletions, and node reclassifications required to transform a given tree T into a desired target T' . The tree edit distance problem can be comparable to the string edit distance. In the case of the string, if $S_1[i] S_2[j]$, then the distance between $S_1[1..i-1]$ and $S_2[1..j-1]$ is the same as between $S_1[1..i]$ and $S_2[1..j]$ [26].

Algorithm 1 Tree Edit Distance - Zhang-Shasha Algorithm [26]

Require: Tree T_1 and T_2

Ensure: Preprocessing (to compute $l()$, $LRkeyroots1$ and $LRkeyroots2$)

```

for  $i \leftarrow 1$  to  $LRkeyroots(T_1)$  do
  for  $j \leftarrow 1$  to  $LRkeyroots(T_2)$  do
     $i' \leftarrow LRkeyroots1[i']$ 
     $j' \leftarrow LRkeyroots2[j']$ 
     $forestdist(\emptyset, \emptyset) \leftarrow 0$ 
    for  $i_1 \leftarrow l(i)$  to  $i$  do
       $forestdist(T_1[l(i)..i_1], \emptyset) \leftarrow forestdist(T_1[l(i)..i_1 - 1], \emptyset) + \gamma(T_1[i_1] \rightarrow \Lambda)$ 
    end for
    for  $j_1 \leftarrow l(j)$  to  $j$  do
       $forestdist(\emptyset, T_2[l(j)..j_1]) \leftarrow forestdist(\emptyset, T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \rightarrow T_2[j_1])$ 
    end for
    for  $i_1 \leftarrow l(i)$  to  $i$  do
      for  $j_1 \leftarrow l(j)$  to  $j$  do
        if  $l(i_1) = l(i)$  and  $l(j_1) = l(j)$  then
           $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1]) = \min\{$ 
             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1]) + \gamma(T_1[i_1] \rightarrow \Lambda)$ 
             $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \rightarrow T_2[j_1])$ 
             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1 - 1]) + \gamma(T_1[i_1] \rightarrow T_2[j_1])\}$ 
           $treedist(i, j) \leftarrow forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1])$ 
        else
           $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1]) = \min\{$ 
             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1]) + \gamma(T_1[i_1] \rightarrow \Lambda)$ 
             $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \rightarrow T_2[j_1])$ 
             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1 - 1]) + treedist(i, j)\}$ 
        end if
      end for
    end for
  end for
end for

```

In the case of the tree, the main difficulty is that preserving the ancestral relationships in the mapping between the trees prevents the analogous implication from being valid. Zhang and Shasha [26] presented a simple dynamic programming algorithm (Algorithm 1) to find the edit distance between ordered labeled trees, with time complexity $O(n^2 \cdot \log^2(n))$ where n is the number of nodes in the larger of the two trees, has better time and space complexity than others algorithms to solve the tree edit distance problem and is generalizable with the same time complexity for approximating tree matching problems. To demonstrate the TED algorithm, the code illustrated in Listing 1 and in Listing 2 are transformed into two different ASTs (T - Figure 12) and T' - Figure 13 respectively), after execute the algorithm to compute the tree edit distance between the two ASTs produced, the final result is the matrix represented in Figure 14. The value in the lower right corner of the matrix (Figure 14) is the distance between the T and the T' , it means that we need to do n operations (insertion or deletion) in the tree T to get to the same tree represented by the T' tree.

		AST Exercise - T																			
		keyroot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
AST Solution - T'	1	0	1	1	1	2	1	1	3	1	5	1	2	3	9	1	2	15	17		
	2	1	0	1	1	2	1	1	3	1	5	1	2	3	9	1	2	15	17		
	3	1	1	0	1	2	1	1	3	1	5	1	2	3	9	1	2	14	17		
	4	1	1	1	0	1	1	1	3	1	5	1	2	3	9	1	2	14	17		
	5	2	2	2	1	0	2	2	3	2	5	2	2	3	9	2	2	13	16		
	6	1	1	1	1	2	0	1	2	1	4	1	2	3	8	1	2	14	17		
	7	1	1	1	1	2	1	0	2	1	4	1	2	3	8	1	2	14	17		
	8	3	3	3	3	3	2	2	1	3	3	3	3	4	7	3	3	13	16		
	9	1	1	1	1	2	1	1	3	0	3	1	2	3	8	1	2	14	17		
	10	5	5	5	5	5	4	4	3	4	1	5	5	5	5	5	5	11	14		
	11	1	1	1	1	2	1	1	3	1	5	0	1	2	8	1	2	14	17		
	12	2	2	2	2	2	2	2	3	2	5	1	1	2	8	2	2	14	17		
	13	3	3	3	3	3	3	3	4	3	5	2	2	1	7	3	3	13	16		
	14	9	9	9	9	9	8	8	7	8	5	8	8	7	3	9	9	9	12		
	15	13	13	12	12	11	12	12	11	12	9	12	12	11	7	13	13	5	8		
	16	15	15	15	15	14	15	15	14	15	12	15	15	14	10	16	16	8	6		

Figure 14. TED result after comparison the AST from code submitted (Listing 2) versus solution (Listing 1).

For the implementation of the algorithm, it was defined for each node its own cost function, during the calculation the node types are checked, when the type is the same the cost function is executed to verify more details of the nodes (for example, type of data, roles of the variables, if it is inside or outside the loop, etc.) if all the characteristics are equal, the cost is considered zero. The characteristics to check depend on the node type, and what was defined in its cost function.

3.5. Nodes Matching

The matrix produced by the Zhang-Shasha algorithm (Figure 14) gives us similarities between each $node_x$ and $node_y$, now we need to prepare a list of the nodes to be analyzed, the defined rule is that a $node_x$ can only be paired with a single $node_y$ and vice versa. The pair created must be the best match between the nodes.

The Gale-Shapley Algorithm is an efficient algorithm that is used to solve the Stable Matching problem, whose time complexity is $O(n^2)$ where n is the number of elements involved. The stable correspondence problem, simplifying, of two different sets of equal size (e.g., n men and n women, or n students and n universities), and the order of preference of each element of the first set by for each element of the second set, can find the stable match for all elements of both sets. The Gale-Shapley algorithm guarantees to produce a stable match for all elements[27].

Algorithm 2 Stable Matching - Gale-Shapley Algorithm

Require: Initialize $m \in M$ and $w \in W$ to *free*
while \exists *free* man m who has a woman w to propose to **do**
 $w \leftarrow$ first woman on m 's list to whom m has not yet proposed
 if \exists some pair (m', w) **then**
 if w prefers m to m' **then**
 m' becomes *free*
 (m, w) become *engaged*
 end if
 else
 (m, w) become *engaged*
 end if
end while

Applying this algorithm to the matrix obtained in the TED calculation, where each column will be an element of a set A and each row will be an element of a set B , and each value that composes each row or column corresponds to the preference value in the element of the opposite set (where the lowest value corresponds to the most preferred value), the best match for each node will be obtained.

		AST Exercise - T																		
		keyroot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
AST Solution - T	1	0	1	1	1	2	1	1	3	1	5	1	2	3	9	1	2	15	17	
	2	1	0	1	1	2	1	1	3	1	5	1	2	3	9	1	2	15	17	
	3	1	1	0	1	2	1	1	3	1	5	1	2	3	9	1	2	14	17	
	4	1	1	1	0	1	1	1	3	1	5	1	2	3	9	1	2	14	17	
	5	2	2	2	1	0	2	2	3	2	5	2	2	3	9	2	2	13	16	
	6	1	1	1	1	2	0	1	2	1	4	1	2	3	8	1	2	14	17	
	7	1	1	1	1	2	1	0	2	1	4	1	2	3	8	1	2	14	17	
	8	3	3	3	3	3	2	2	1	3	3	3	3	4	7	3	3	13	16	
	9	1	1	1	1	2	1	1	3	0	3	1	2	3	8	1	2	14	17	
	10	5	5	5	5	5	4	4	3	4	1	5	5	5	5	5	5	11	14	
	11	1	1	1	1	2	1	1	3	1	5	0	1	2	8	1	2	14	17	
	12	2	2	2	2	2	2	2	3	2	5	1	1	2	8	2	2	14	17	
	13	3	3	3	3	3	3	3	4	3	5	2	2	1	7	3	3	13	16	
	14	9	9	9	9	9	8	8	7	8	5	8	8	7	3	9	9	9	12	
	15	13	13	12	12	11	12	12	11	12	9	12	12	11	7	13	13	5	8	
	16	15	15	15	15	14	15	15	14	15	12	15	15	14	10	16	16	8	6	
	17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
	18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	

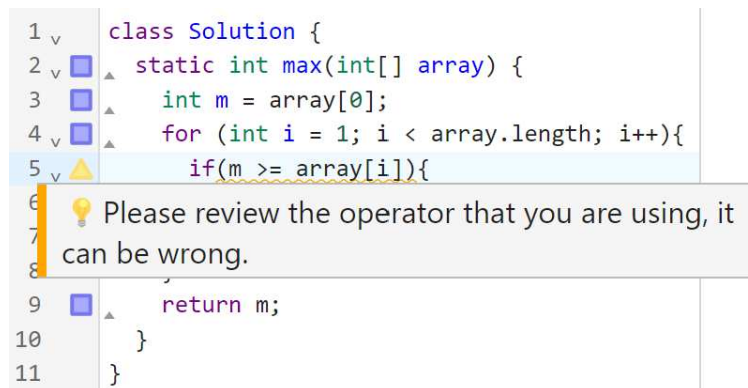
Figure 15. Gale-Shapley algorithm applied to TED matrix 14

This algorithm also needs to be adapted to our context, as the algorithm assumes that we are going to deal with two sets of the same size, which is not always true for our process, as normally the trees will have different sizes. In the first example, the resulting matrix is square, therefore ideal for the application of the algorithm. So, consider that we need to produce a tip, when the student is writing his first lines and he feels stuck, the resulting matrix with the TED calculation will not be square, so there are no conditions to produce two sets with equal sizes using each row for produce one set and each column to produce another set. The solution is to add the missing columns or rows where all elements are represented by -1, and during the calculation assume

that the new row/column is not ready to “engage” as well as all remaining elements of the opposite set when there are no free elements to match. The purpose of using this algorithm is an efficient way to create the pairs to be compared and then produce hints as the results of your comparison, the nodes that did not find a pair can be used to generate next-step hints (see columns 12 and 15 in Figure 15).

3.6. Nodes Pairs Analysis And Hints Generation

The analysis of each node is different, the analysis of ReturnTypeNameNode consists only in checking the data type being used, while LoopNode consists in checking several characteristics, for example, the variable to initialize, the expression of the condition, if there is or not an alternative statement, etc...). For each verified characteristic, a specific hint is provided, to try to help the student during his attempt to produce a solution for the proposed exercise. Hints messages are suggestions of what the student can do, not exactly what he should do, for example a wrong operator, the message informs that the operator is wrong but does not say which is the right operator (Figure 16).



```
1 class Solution {
2     static int max(int[] array) {
3         int m = array[0];
4         for (int i = 1; i < array.length; i++){
5             if(m >= array[i]){
6                 // Hint message: Please review the operator that you are using, it
7                 // can be wrong.
8             }
9         }
10        return m;
11    }
```

Figure 16. Hint message for a wrong operator

3.6.1. Type Analysis

Type analysis is the analysis done to the data type of a variable, parameter, or return type. The data type specifies the different sizes and values that can be stored in variables (Figure 17). During node pairs analysis, some nodes need to compare their data types (e.g. return type, variable, parameter), which can give rise to some of the hints described below.

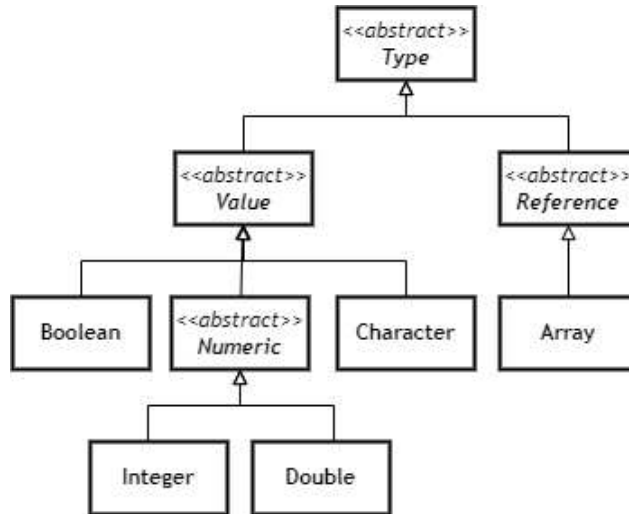


Figure 17. Types Currently Handled

Hint Type	When	Hint Message
Value Type	A reference type was used, but the solution is expecting a value type.	<i>The solution is not expecting a reference type as (return type, or parameter, or variable).</i>
Different Type	The types (Value or Reference type) are the same, but the data type is not.	<i>You must use a different type than {submission.type}.</i>
Number Type	A different number type was used.	<i>Very good, you are using a value type as (return type, or parameter, or variable), but you can use a different numeric data type than {submission.type}.</i>

Table 1. Value type hints

Hint Type	When	Hint Message
Reference Type	A value type was used, but the solution is expecting a reference type.	<i>The solution is not expecting a value type as (return type, or parameter, or variable).</i>
Array Type	An array is expected, but another reference type data type was used.	<i>You must use an array as (return type, or parameter, or variable).</i>
Array Elements Value Type	The solution is expecting an array and the type of its elements is of value type, but a reference type has been used for its elements.	<i>To reach the objective, the elements of array should be specified as value type.</i>
Array Elements Reference Type	The solution is expecting an array and the type of its elements is of reference type, but a value type has been used for its elements.	<i>To reach the objective, the elements of array should be specified as reference type.</i>
Array Different Type	The solution is expecting an array whose elements are of a different data type than the given data type.	<i>You must use a data type other than {submission.type}, for array elements.</i>

Array Number Type	The solution expects an array whose elements are of a different number type than the given data type.	<i>Great job, you are using a value type as data type for array elements, but you can use a different number type than {submission.type}.</i>
Array Dimensions	The solution expects an array with different dimension than the dimension used in submission.	<i>Data type is OK, but now you need to check the array dimensions.</i>

Table 2. Reference type hints

3.6.2. Return Type Analysis

The return type analysis basically consists of checking the defined return type (Table 1 and Table 2), but return type has a specific keyword (*void*) that is handled as a type, and will provide specific hints when it is used in the student's solution or in the teacher's solution (Table 3).

Hint Type	When	Hint Message
Void	A type was used, but the solution is expecting the void keyword.	<i>Your solution have been specified to return a data type, but is not expected to return a value.</i>
Type	The void keyword was used, but the solution is expecting a type.	<i>Your solution was specified not to return a value, but you must define a data type for the return value.</i>

Table 3. Return type hints

3.6.3. Parameter and Variable Analysis

Parameter node and Variable node analysis is a bit more complex than return node type analysis, as in return type node analysis, the types are checked (Table 1 and Table 2) for both, but some hints are generated based on the number of expected parameters and/or variables. Here starts the first next-step hints when the parameters or variables are missing in the student solution (Table 4).

Hint Type	When	Hint Message
Unnecessary	The number of (parameters, or variables) sent is greater than the number of (parameters, or variables) in the solution.	<i>You can consider remove this (parameter, or variable).</i>
Needs Parameters	The submitted solution does not contain any parameters, but the exercise expects at least one parameter	<i>You need to set some parameters in your function.</i>
Add More Parameters	The submitted solution does not contain the number of parameters expected.	<i>The number of parameters is not correct, try adding one or more parameters to your function.</i>
Needs Variable	The submitted solution does not contain any variable, but the exercise expects at least one variable	<i>You will need to declare some variables in your solution.</i>

Table 4. Hints based in the number of parameters/variables

Hint Type	When	Hint Message
Fixed Value	The implemented variable should receive a value that then does not change for a duration of a loop.	<i>You should consider making this variable's role as a fixed value.</i>
Stepper	The variable is expected to move through an array or other data structure, typically going towards a fixed value and looping through the elements in an array.	<i>You should consider making this variable's role as a stepper.</i>
Gathered	The student's solution must implement the variable in such a way that it accumulates or records a set of data and inputs.	<i>You should consider making this variable's role as a gathered.</i>
Most Wanted Holder	The solution must have a variable that tracks the lowest or highest value in a set of inputs.	<i>You should consider making this variable's role as a most wanted holder.</i>
One-Way Flag	The variable (<i>boolean</i>) is a one-way flag when can effectively only be changed once, although the new value can be reassigned multiple times.	<i>You should consider making this variable's role as a one-way flag.</i>

Table 5. Hints based in variable roles

Variables also produce hints based on their roles, results from one study suggested that knowledge about variable roles allows students to process information similarly to good coders [28], so this type of hint can be very useful to help students (Table 5).

3.6.4. Expression Analysis

Expression Analysis consists of analyzing the expressions defined for an assignment, loop statement, if-else statement, or return statement. The treated expressions are described in Table 6. As with type analysis, during the pairs of nodes analysis, some nodes need to compare their expressions, which can give rise to some of the tips described below (Table 9).

Expression	Example
Literal	<i>1, 'a', true</i>
Variable Element	<i>x</i>
Array Element	<i>array</i>
Array Allocation	<i>array[expression]</i>
Array Length	<i>array.length</i>
Unary Expression	<i>NOT(a == b), -(1 + 2)</i>
Binary Expression	<i>a + 2, a + b - c, a AND c</i>

Table 6. Expressions Examples

Hint Type	When	Hint Message
Literal	The exercise solution is using a literal expression but the student solution is using a different expression.	<i>For this (assignment, or condition, or return statement) is expected to use a literal value instead of a variable or an expression.</i>

Literal Value	The exercise is expecting a literal value different from the one defined by the student.	<i>You should try to use a different literal value, it can affect the result of your solution.</i>
Variable	The exercise solution is using a variable expression but the student solution is using a different expression.	<i>For this (assignment, or condition, or return statement) is expected to use a variable value instead of a literal value or expression.</i>
Array Element	The exercise solution is using an value from an specific array element as expression but the student solution not.	<i>It is expected to (assign, or use) a value of an array element to this assignment.</i>
Array Allocation	The exercise solution is allocating an array but the student solution is using a different expression.	<i>It is expected to allocate an array on this assignment.</i>
Array Length	The exercise solution is using the attribute length from an array in its expression but the student solution not.	<i>You can use the the attribute length to determines the length of an array.</i>
Binary Expression	The exercise solution is using a binary expression but the student's solution not.	<i>You can use a expression instead of a single variable or literal value.</i>

Operator	The operator used by student is not equal or equivalent to the solution's operator.	<i>Please review the operator that you are using, it can be wrong.</i>
----------	---	--

Table 7. Hints based in expressions

As syntactically different programs can behave in an equivalent way, the nodes that can contain expressions are able to treat equivalent expressions as an equal expression, avoiding incorrect classifications and producing hints that can lead the student to error. Below are some examples of expressions that can be handled (Table 8).

Semantic Equivalences			
$a \geq b$	$b \leq a$	$\neg(a \leq b)$	$\neg(b \geq a)$
$a \leq b$	$b \geq a$	$\neg(a \geq b)$	$\neg(b \leq b)$
$a > b$	$b < a$	$\neg(a < b)$	$\neg(b > a)$
$a < b$	$b > a$	$\neg(a > b)$	$\neg(b < b)$
$a = b$	$b = a$	$\neg(a \neq b)$	$\neg(b \neq a)$
$a \neq b$	$b \neq a$	$\neg(a = b)$	$\neg(b = a)$
$a + b$	$b + a$		
$a * b$	$b * a$		
$a ++$	$a = a + 1$	$a = 1 + a$	
$a --$	$a = a - 1$		

Table 8. Examples of semantic equivalences

3.6.5. Combining Analysis

For Assignment (Figure 18a), Loop (Figure 18b), Selection (Figure 18c) and Return (Figure 18d) nodes, the application will combine the previous analyses. For example, for Loop Analysis, the three parts that compose the loop-statement (*initializer*, *condition* and *increment*), will be analyzed separately. That is, for *initializer* the Variable Analysis (Section 3.6.3) and Assignment Analysis (Figure 18a, which performs the Type Analysis and Expression Analysis) will be performed, for *condition* the Expression Analysis (Section 3.6.4) will be executed, and finally, for *increment* will be the Assignment Analysis.

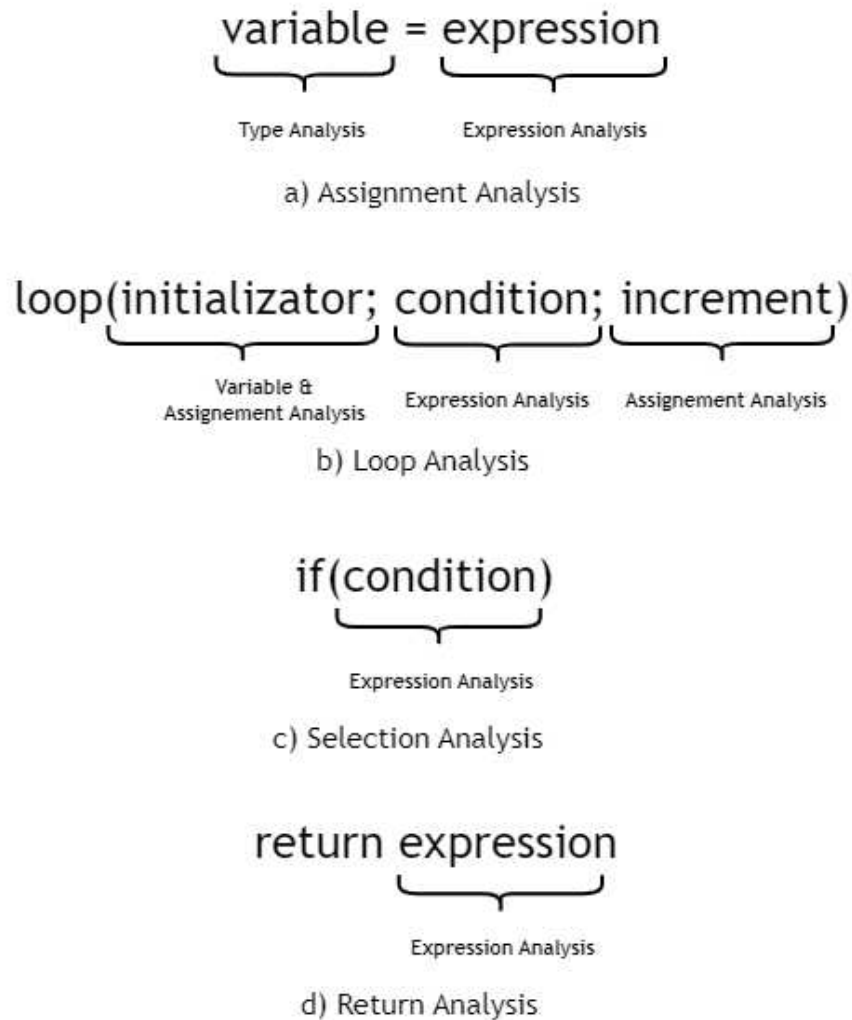


Figure 18. Combining Analysis

3.6.6. Positive Feedback

In addition to generating hints, feedback is provided for each well-implemented element (node). Positive feedback can help students stay focused on their goal [25], i.e. solving the exercise. The messages provided are specific about what the student did well so the student can understand them and what they should continue to do.

Type	When	Message
Parameter	A parameter node was well formed	<i>Good, this parameter is correct.</i>

Return Type	Return type node was well formed	<i>Great, return type is correct.</i>
Variable	Variable node was well formed	<i>Good, this variable is correct.</i>
Assignment	Assignment node was well formed	<i>Good, this variable assignment is correct.</i>
If..Else Statement	Selection node condition was well defined	<i>Well done, the condition is correct.</i>
Loop Statement	Loop node condition was well defined	<i>Good job, the condition is well formed.</i>
Return Statement	Expression of return is right	<i>Well done, the return statement is right.</i>

Table 9. Some Example of Positive Feedback Provided

3.7. Exercise Scenario Walkthrough

This section describes a scenario where a teacher creates exercises, in order that students will try to solve them with the support and guidance of our tool.

3.7.1. Teacher - Exercise Setup

The teacher needs to create one or more exercises in the platform. Each exercise should contain a title, a description of what the purpose is (which may provide some guidance to the student, e.g. expected data type, expected input parameter, etc.), some input parameters and expected output for the exercise (which can be used by the prototype to test the student’s solution), and a proposed solution (a template). (Figure 19).

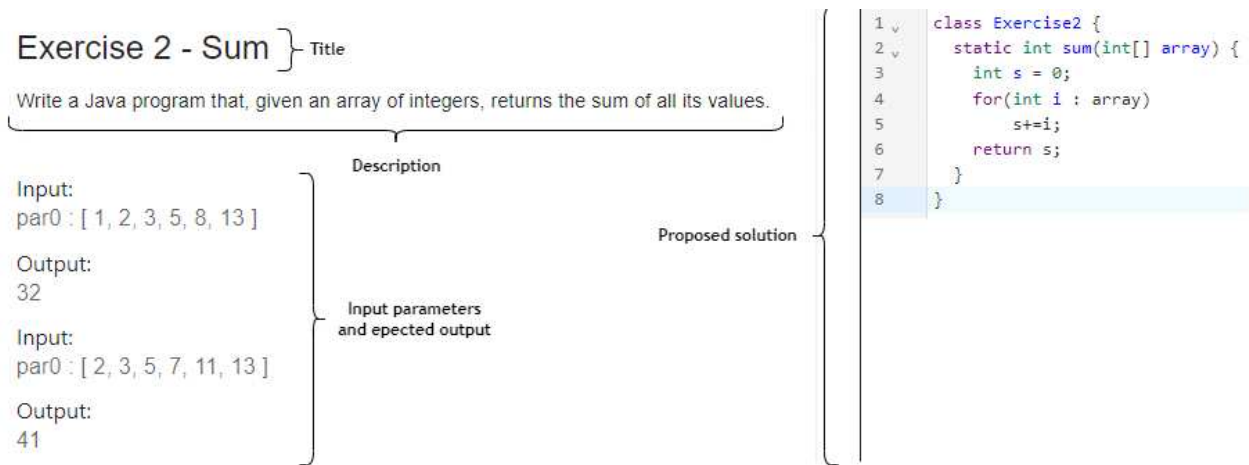


Figure 19. Exercise Setup Example

3.7.2. Student - Solving Exercise

a) A student starts writing a few lines of code for a proposed problem posted on the platform. When the student feels stuck, he will press the “Get Hint?” button available on the platform. This action will send a request (which includes the exercise id and the student’s code solution) to the API that will analyze the code against the code related to the provided exercise ID, and then retrieves a response to the platform (Figure 20).

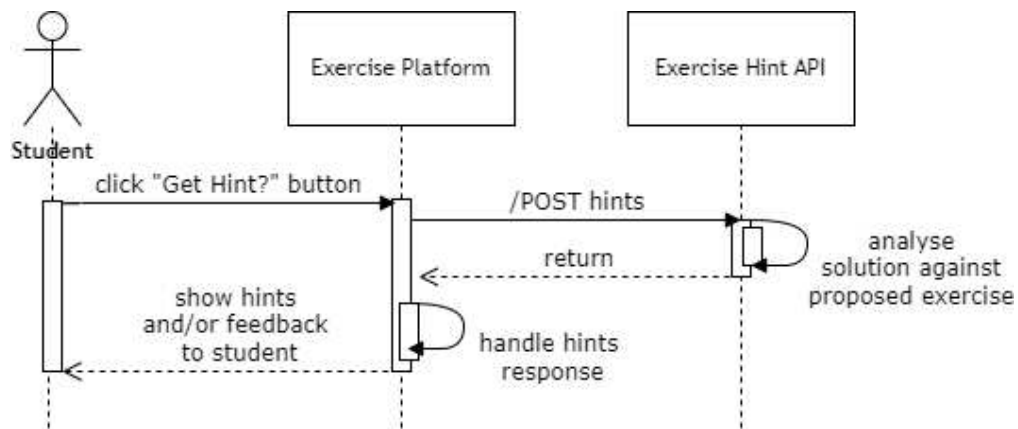


Figure 20. Sequence Diagram of Exercise Hint Platform

b) The API response is handled by the platform and rendered, showing hints and/or feedback for the student by adding markers to code. Figure 21 presents a hint, suggesting the student to change the return type of his solution.

```
1 v class Solution {
2 v static double sum(int[] array){
3
4
5
```

💡 Very good, you are using a value type as return type, but you can use a different numeric data type than double.

📖 Remember, double is a floating point values that provide a fractional part, which means they have decimal components in them. int (integer) refer to whole numbers, they do not have a fractional part. In programming due to the advantages like less memory occupancy, faster processing and rounding, integers are preferred.

Figure 21. Code Correction Hint

c) Step b can be repeated until the student gets the solution right. For example, the student follows the hint (or ignores it) and clicks the button to get a new hint. Figure 22 shows a new hint that can be provided to the student, this time it is not a correction hint, but rather a hint for the next step that suggests the student to add a variable to his solution.

```
1 v class Solution {
2 v static int sum(int[] array){
3
4
5
```

💡 You will need to declare some variables in your solution.

📖 Variables are containers for storing data values. To declare (create) a variable, you must specify the type and assign it a value:

E.g.
int variableName = 1;

Figure 22. Next-Step Hint

- d) The prototype also provides hints based on the roles of variables in the proposed solution (Figure 23), the results of one study suggested that knowledge about the roles of variables allows students to process information similarly to good code connoisseurs [28].

```
1 class Solution {
2     static int sum(int[] array){
3         int x = 0;
4
5
```

💡 You should consider making this variable's role as a most wanted holder.

📖 Remember, the role of the variable is a most wanted holder when a variable that keeps track of the lowest or highest value in a set of inputs.

E.g.

```
int mostWantedHolder = set[0];
for (int stepper = 1; i < fixedValue;
stepper++) {
    if (mostWantedHolder < set[i]) {
        mostWantedHolder = set[i];
    }
}
```

Figure 23. Hint Based in Variable Roles

- e) The prototype reduces to a canonical form some semantic equivalences, such as handling all loops (simple *for-loop* Figure 24c, *while-loop* Figure 24b, enhanced *for-loop* Figure 24a) in a normalized way and also normalizing the relational and arithmetic operators (see some examples in Table 8).

```

1 class Solution {
2     static int sum(int[] array) {
3         int s = 0;
4         for(int i : array)
5             s+=i;
6         return s;
7     }
8 }

```

Well done, the return statement is right.

a) Code Written Using Enhanced For-Loop

```

1 class Solution {
2     static int sumArray(int[] integers) {
3         int sum = 0;
4         int j = 0;
5         while(j < integers.length){
6             sum+=integers[j];
7             j = j + 1;
8         }
9         return sum;
10    }
11 }

```

Well done, the return statement is right.

b) Code Written Using While-Loop

```

1 class Solution {
2     static int sumArray(int[] integers) {
3         int sum = 0;
4         for(int i = 0; integers.length > i; i++)
5             sum = sum + integers[i];
6         return sum;
7     }
8 }

```

Well done, the return statement is right.

c) Code Written Using Simple For-Loop

Figure 24. Different semantics to correctly solve the same solution.

f) Finally, the prototype provides a button to test the student's solution, using the parameters previously configured by the teacher, in the configuration of the exercise (Figure 25).

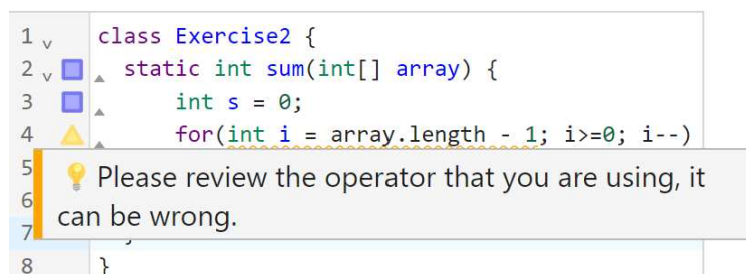
✓ Test1 -> Input: par0 : [1, 2, 3, 5, 8, 13] -> Output: 32
 ✓ Test2 -> Input: par0 : [2, 3, 5, 7, 11, 13] -> Output: 41

Figure 25. Test Data Results

3.8. Limitations

Although the prototype largely fulfills the proposed objectives, it has some identified characteristics that can be explored (and/or added) in future works.

- The current prototype is performing an intraprocedural analysis, that is, it is analyzing only one function (or method), implementing an interprocedural analysis (it covers several functions) is possible and can be an improvement. One approach that can be followed is to convert the main block of each nested function as the main function block when the AST is built. This allows us to compare several semantically equivalent solutions, even if they don't contain nested functions to be executed.
- Creating a set of solutions for the same exercise could improve the quality of the hints provided, the TED could be calculated by comparing the solution provided by the student with each solution in the set and using the solution that has the smallest TED to provide hints.
- Strudel provides many features, an interesting feature is to run the solution after deserialization, this allows us to provide hints for more complex exercises. For example, an exercise requires a minimum of iterations within a loop. Strudel can be also used for the following example:



```
1 class Exercise2 {
2     static int sum(int[] array) {
3         int s = 0;
4         for(int i = array.length - 1; i >= 0; i--)
5             s += array[i];
6     }
7 }
8 }
```

Please review the operator that you are using, it can be wrong.

Figure 26. Semantic equivalent of the solution considered wrong.

Figure 26 represents a different way of solving the *sum* exercise, but instead of starting at the beginning of the array it starts at the end, but the solution is

semantically equivalent to the teacher's solution. In this case, Strudel can be used to check if the solution provides the expected result and considers it as a valid solution and inhibits hints for code correction or next step.

CHAPTER 4

Prototype Design and Implementation

The main purpose of this application is to provide hints throughout the student problem solving process, this means that students should not only be given hints on how to correct near-completed programs, but also students who struggle to find a good starting point. However, from a more technical point of view, there was a concern to build an application with an architecture that would allow its reuse and easy integration with other systems. In addition, a development environment was also created with the Continuous Integration (CI)/Continuous Delivery (CD) method, which provides an automatic way for deployments and ensures application quality by running existing unit tests before deployment. In this chapter, the architecture of the system and the technologies used for its implementation will be described, followed by how the core component of the application (main focus of the dissertation), called Exercise Hint API, was implemented. It then describes how the Exercise Hint API can be integrated into an User Interface (UI) and how the user can interact with it. Finally, it will describe some challenges that arose during the development of the prototype.

4.1. High Level System Architecture

The entire system (Figure 27) was built around the Exercise Hint API as it is the core of the prototype. But to demonstrate how the API can be used, a UI was implemented in a web environment, so that users have an interactive environment to try and test the Application Programming Interface (API). The system was built following a Platform as a Service (PaaS) (Platform as a Service) cloud model architecture, that is, it offers a flexible and scalable cloud platform to develop, deploy, run and manage applications. The cloud service provider (Google Cloud⁴) provides the necessary networks, servers, and storage to host an application while the end user oversees software deployment and

⁴<https://cloud.google.com/>

configuration settings. A CI/CD pipeline was also set up which made it easier to maintain the integrity of the deployment.

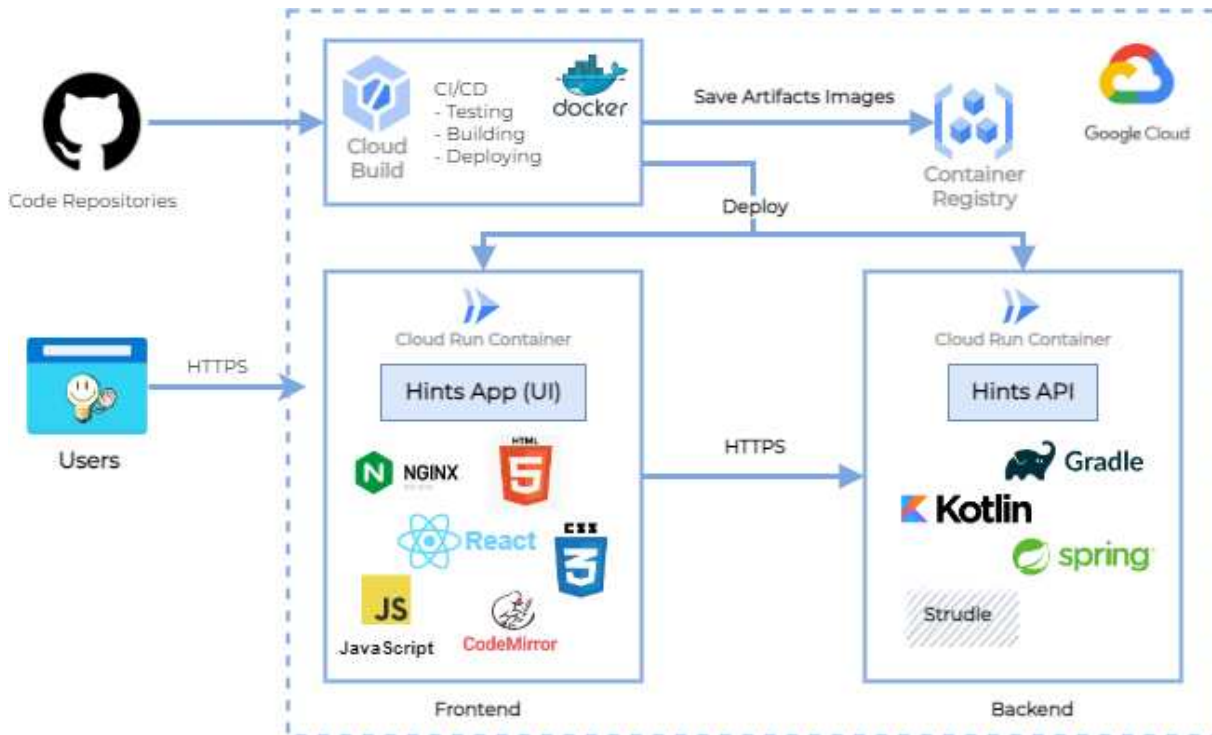


Figure 27. High Level System Architecture

In this environment, the code is written and uploaded to the source repository (GitHub⁵), as soon as the code arrives in the repository, Cloud Build⁶ is triggered and runs tests and security checks, builds a docker image and pushes it to the Container Registry⁷. Then the produced containers are deployed on the production clusters (Cloud Run⁸), finally available to be accessed anywhere. Following a client-server architectural pattern, applying the separation of concerns between the presentation layer (frontend) and the business logic and/or physical logic (backend), the layers communicate with each other through the Hypertext Transfer Protocol Secure (HTTPS) protocol.

⁵<https://github.com/>

⁶<https://cloud.google.com/build>

⁷<https://cloud.google.com/container-registry/>

⁸<https://cloud.google.com/run/>

The frontend was developed with JavaScript⁹ language using mainly the libraries, React¹⁰ and CodeMirror¹¹. The backend is a Representational State Transfer (REST) API [29], developed using Kotlin¹² language, and Spring¹³ framework and Strudel library (which provides a powerful tool to deserialize code in text format to an object). The API includes all the business logic, that is, all the core functionalities related to the suggested approach for this dissertation.

4.2. User Interface

To allow interaction with the API and its implemented logic, a simple user interface (UI) was developed. The main page contains some exercises that can be selected (Figure 28).

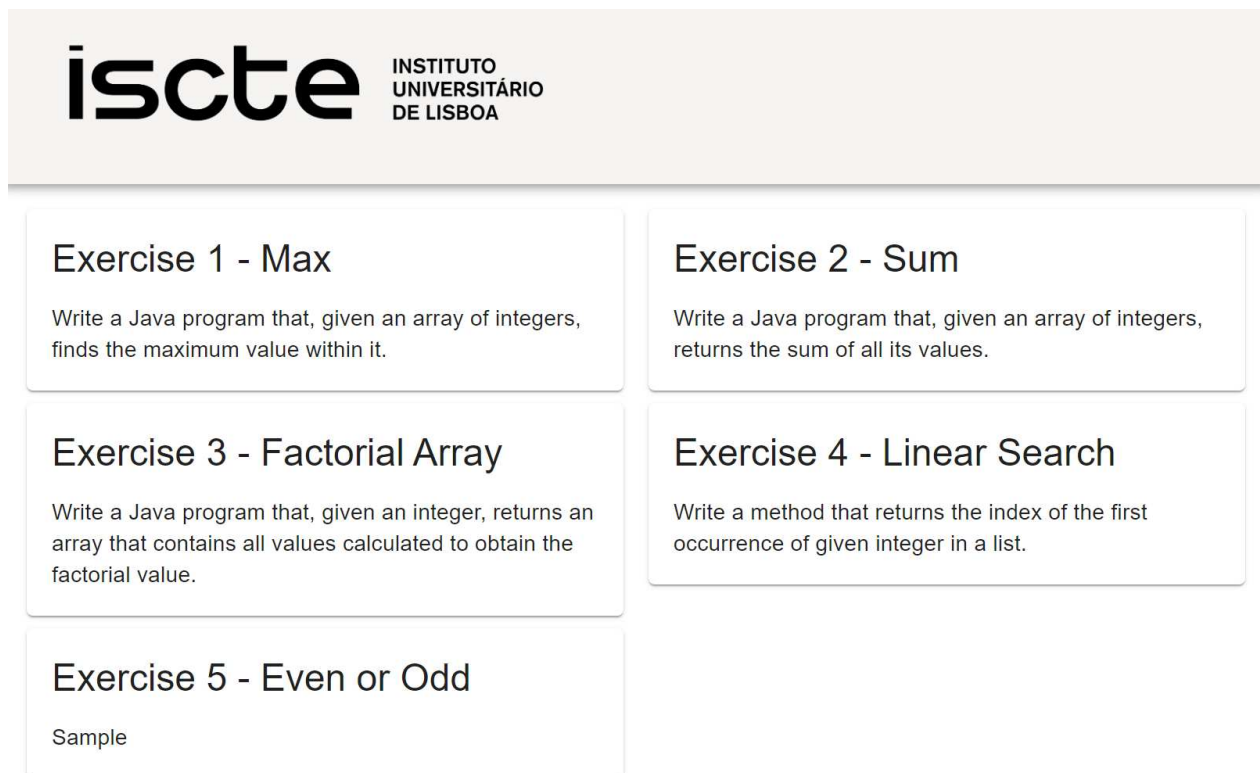


Figure 28. Main Page

⁹<https://www.javascript.com/>

¹⁰<https://reactjs.org/>

¹¹<https://codemirror.net/>

¹²<https://kotlinlang.org/>

¹³<https://spring.io/>

After selecting the exercise, a new page will be displayed where the student can start solving the exercise (Figure 29). The page is divided into three sections, on the right side the exercise content (Figure 29a), which contains the description of the exercise and the expected results for specific entries. The left side is divided into two parts, in the upper part the code editor, where the student can start writing his solution (Figure 29b) and where two buttons are also available (Figure 29c), one to get tips and the other to submit and test his solution and, finally, at the bottom, a third section where the results of submitted solution and some error messages when something goes wrong with the communication between the layers will be displayed (Figure 29d).

The screenshot shows the ISCTE (Instituto Universitário de Lisboa) Exercise Code Editor interface. The page is divided into four sections:

- a) Exercise Content:** Titled "Exercise 1 - Max", it asks the student to write a Java program that finds the maximum value in an array of integers. It provides two test cases:
 - Input: par0 : [0, 1, 2, 3, 5]
Output: 5
 - Input: par0 : [9, 1, 81, 3, 27]
Output: 81
- b) Code Editor:** Contains a Java code snippet for a class `Exercise1` with a `static int max(int[] array)` method. The code is:


```

class Exercise1 {
    static int max(int[] array) {
        int m = array[0];
        for (int i = 1; i < array.length; i++) {
            if (m < array[i]) {
                m = array[i];
            }
        }
        return 5;
    }
}
      
```

 A hint message is displayed: "You should try replacing the return value with a single variable."
- c) Action Buttons:** Two buttons are visible: "GET HINT?" (with a lightbulb icon) and "SUBMIT" (with a play icon).
- d) Test Results:** A table showing the results of two test cases:
 - Test1 -> Input: par0 : [0, 1, 2, 3, 5] -> Output: 5 (Passed, green checkmark)
 - Test2 -> Input: par0 : [9, 1, 81, 3, 27] -> Output: 5 (Failed, red circle)

Figure 29. Exercise Code Editor

4.3. Exercise Hint API

As described earlier, the Exercise Hint API is a REST API that provides an endpoint, which, upon receiving a request containing a code snippet and the exercise ID, will return

a list of tips and positive feedback based on the solution related to the given exercise ID. In addition to the main endpoint, there is two more endpoint to support the frontend with exercise data. There is an endpoint that can be accessed to get the API documentation (Document OpenApi¹⁴) that describes its elements.

Designed using a *clean architecture* [30], putting the business logic and application model at the center of the application, rather than having the business logic dependent on data access or other infrastructure issues, this dependence is reversed: the infrastructure and implementation details depend on the application core. Following the Dependency Inversion Principle [31], as well as Domain-Driven Design (Domain Driven Design (DDD))[32] principles, this functionality is achieved by defining abstractions, or interfaces, at the domain layer, which are then implemented by types defined at the infrastructure layer (Figure 30).

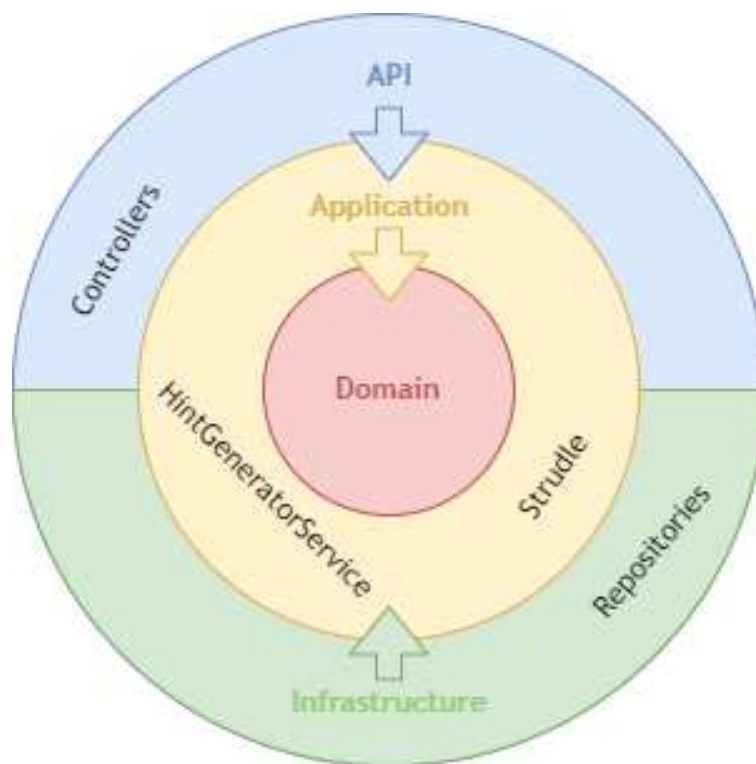


Figure 30. Exercise Hint API Architecture

¹⁴<https://spec.openapis.org/oas/latest.html>

4.3.1. API Endpoints

The API exposes six endpoints that will support the entire flow defined for the prototype to present the idealized approach to provide tips and feedback to students. Controllers are responsible for controlling the way a user interacts with an application. A controller contains the flow control logic and determines what response to send back to a user when they make a request. The controllers are located in the API layer of the application (Figure 30), and each controller is represented by one of the three available resources (Exercises, Compiler and Hints) that will be described below.

Exercise Hint API ^{1.0}

[Base URL: localhost:8080]

Exercise Hint API

compiler-controller Compiler Controller

POST /api/v1/compiler post

GET /api/v1/compiler/{id} getById

POST /api/v1/compiler/{id}/test postById

exercise-controller Exercise Controller

GET /api/v1/exercises get

GET /api/v1/exercises/{id} getById

hint-controller Hint Controller

POST /api/v1/hints post

Figure 31. Available Endpoints

4.3.1.1. *Exercises* This resource is responsible for providing, as the name implies, the required exercise data to be displayed in the user interface for the student. There are two endpoints available, one to get all available exercises (Listing 3) and another to get a specific exercise by identifier, providing the identifier in the Uniform Resource Locator (URL) (Listing 4).

```
1 Request:
2   GET /api/v1/exercises HTTP/1.1
3 Response:
4 [
5   {
6     "id": "f04e081b-c0e7-4f04-a517-89ccc322b054",
7     "name": "Exercise 4 - Linear Search",
8     "description": "Write a method that returns the index of ...",
9     "additionalDescription": "Assume that the index of the first ...",
10    "testData": [
11      {
12        "input": [
13          { "value": "5" },
14          { "value": "[ 1, 2, 3, 5, 8, 13 ]" }
15        ],
16        "output": { "value": "3" }
17      }
18    ]
19  }
20 ]
```

Listing 3. Get All Exercises Request and Response

The exercise data model contains *id* which is the exercise identifier, *description* that is a short description of the exercise, *additionalDescription* is a more detailed description of the exercise and *testData* which represents the expected result for the specific input parameters.

```

1 Request:
2   GET /api/v1/exercises/6d959c20-a370-4e97-b448-a37a39a35328 HTTP/1.1
3 Response:
4 {
5   "id": "6d959c20-a370-4e97-b448-a37a39a35328",
6   "name": "Exercise 1 - Max",
7   "description": "Write a Java program that, given an array of integers...",
8   "additionalDescription": null,
9   "testData": [
10    {
11      "input": [ {"value": "[ 0, 1, 2, 3, 5 ]" } ],
12      "output": {"value": "5"}
13    }
14  ]
15 }

```

Listing 4. Get Exercise by identifier Request and Response

4.3.1.2. *Compiler* Resource responsible for compiling a previously created exercise or executing a code snippet provided upon request. Three endpoints are available, one to run a specific exercise by identifier using random parameters (Listing 5), a second to run the provided solution also using random parameters (Listing 7).

```

1 Request:
2   GET /api/v1/compiler/6d959c20-a370-4e97-b448-a37a39a35328 HTTP/1.1
3 Response:
4 {
5   "id": "6d959c20-a370-4e97-b448-a37a39a35328",
6   "result": {
7     "input": [ "[0, 6, 4, 2, 8, 5, 8, 2]" ],
8     "output": "8",
9     "wasSuccessful": true,
10    "errors": [],
11    "hasPassed": true
12  }
13 }

```

Listing 5. Exercise compilation

The latter is used to test the solution based on the parameters provided in the request (Listing 6). The request's compiler data model contains an *id* for the solution

and the *code* that contains the code snippet to execute, for testing it is also necessary the *testData*, which contains a list of *input* parameters and expected *output*.

```
1 Request:
2   POST /api/v1/compiler/6d959c20-a370-4e97-b448-a37a39a35328/test HTTP/1.1
3 {
4   "id": "6d959c20-a370-4e97-b448-a37a39a35328",
5   "code": "class Exercise1 { static int max(int[] array) { return 0; }}",
6   "testData": [
7     {
8       "input": [ { "value": "[ 0, 1, 2, 3, 5 ]" } ],
9       "output": { "value": "5" }
10    }
11  ]
12 }
13 Response:
14 [
15   {
16     "id": "6d959c20-a370-4e97-b448-a37a39a35328",
17     "result": {
18       "input": [ "[0, 1, 2, 3, 5]" ],
19       "output": "null",
20       "wasSuccessful": false,
21       "errors": [ "variable not initialized" ],
22       "hasPassed": false
23     }
24   }
25 ]
```

Listing 6. Test solution with an unsuccessful execution

The response data model is composed of the *id* of the solution, and the *result*, whose data model contains the *input* parameters used to run the solution, the *output* after o execution , an attribute that indicates whether the execution was successful (*wasSuccessful*), a list of *errors* encountered during execution and whether the solution passed (*hasPassed*) the tests, that is, is returning the expected result.

```

1 Request:
2   POST /api/v1/compiler
3 {
4   "id": "6d959c20-a370-4e97-b448-a37a39a35328",
5   "code": "class Exercise1 { static int max(int[] array) { return 0; } }"
6 }
7 Response:
8 {
9   "id": "6d959c20-a370-4e97-b448-a37a39a35328",
10  "result": {
11    "input": [ "[0, 3, 4, 3, 1]" ],
12    "output": "0",
13    "wasSuccessful": true,
14    "errors": [],
15    "hasPassed": false
16  }
17 }

```

Listing 7. Solution compilation

4.3.1.3. *Hints* It is the main resource, responsible for providing hints and positive feedback on the code snippet (*code*) based on the exercise id (*solutionId*), both sent on request body (Listing 8) .

```

1 Request:
2   POST /api/v1/hints HTTP/1.1
3 {
4   "solutionId": "6d959c20-a370-4e97-b448-a37a39a35328",
5   "submissionId": "6d959c20-a370-4e97-b448-a37a39a35328",
6   "code": "class Exercise1 { static int max(int[] array) { return m; } }"
7 }
8

```

Listing 8. Hints Request

```

1 Response:
2 {
3   "solutionId": "6d959c20-a370-4e97-b448-a37a39a35328",
4   "submissionId": "6d959c20-a370-4e97-b448-a37a39a35328",
5   "hints": [
6     {
7       "id": "m",
8       "location": {
9         "line": 3,
10        "start": 12,
11        "end": 27
12      },
13      "hint": {
14        "id": "HNT003001014",
15        "type": "VARIABLE",
16        "messageType": "EXPECTING_MOST_WANTED HOLDER",
17        "message": "You should consider making this variable's ...",
18        "complementaryMessage": "Remember, the role of ...",
19        "whenHint": "When the role of solution is most wanted ...",
20        "isPositiveFeedback": false
21      }
22    }
23  ]
24 }

```

Listing 9. Hints Response

The Hints response data model (Listing 9) contains the exercise id (*solutionId*), the id that the service consumer gave to its request (*submissionId*), and a list of (*hints*). The data model defined for each hint is composed of an *id*, which represents the parsed element (for example, the name of the variable, or a loop ("Loop(depth)"), or a declaration of return("Return(depth)"), etc.), the *location* in the code snippet sent on request, which contains the *line* number and where the expression starts (*start*) and where it ends (*end*). There is one more attribute (*hint*) which contains the hint object, which contains the *message* and a complementary *complementaryMessage* message to be displayed, and some more details about the hint, like the internal *textid*, and the hint *type*, type of message (*messageType*), when this message occurs (*whenHint*), and whether it is a hint or positive feedback (*isPositiveFeedback*).

4.3.2. Hint Generator Service

The Hint Generator Service, located in application layer, is the component that contains the flow described in our approach (Section 3.2). This component will execute sequentially each task described in the flow.

```
1 fun generateHints(solutionsPair: Pair<Code,Code>):List<AnalysisResult> =
2     try{
3         solutionsPair
4             .parse()
5             .toAbstractSyntaxTree()
6             .getTreeEditDistance()
7             .getNodesMatching()
8             .generateHints()
9     } catch (e : SubmissionNotWellFormedException){
10         getHint(e.id, e.location, HintCode.SUBMISSION_NOT_WELL_FORMED)
11     } catch (e : SolutionNotWellFormedException){
12         getHint(e.id, e.location, HintCode.SOLUTION_NOT_WELL_FORMED)
13     }
```

Listing 10. HintGeneratorService method to generate hints (Kotlin)

4.3.2.1. *parse* It is the method responsible for deserializing the text code (Figure 32). It's a simple method that uses the implementation of the Strudel library to perform deserialization.

4.3.2.2. *toAbstractSyntaxTree* This method convert the object (IProcedure) created by the *parse* method to a new object (ITree) that represents an abstract syntax tree (Section 3.3).

4.3.2.3. *getTreeEditDistance* This method performs the implementation of the Zhang-Shasha algorithm, to provide a matrix representing the tree-editing distance between previously generated trees (Section 3.4).

4.3.2.4. *getNodeMatching* Based on Gale-Shapley algorithm (Section 3.5), this method will provide a list of matched nodes based in the matrix generated by the method *getTreeEditDistance*.

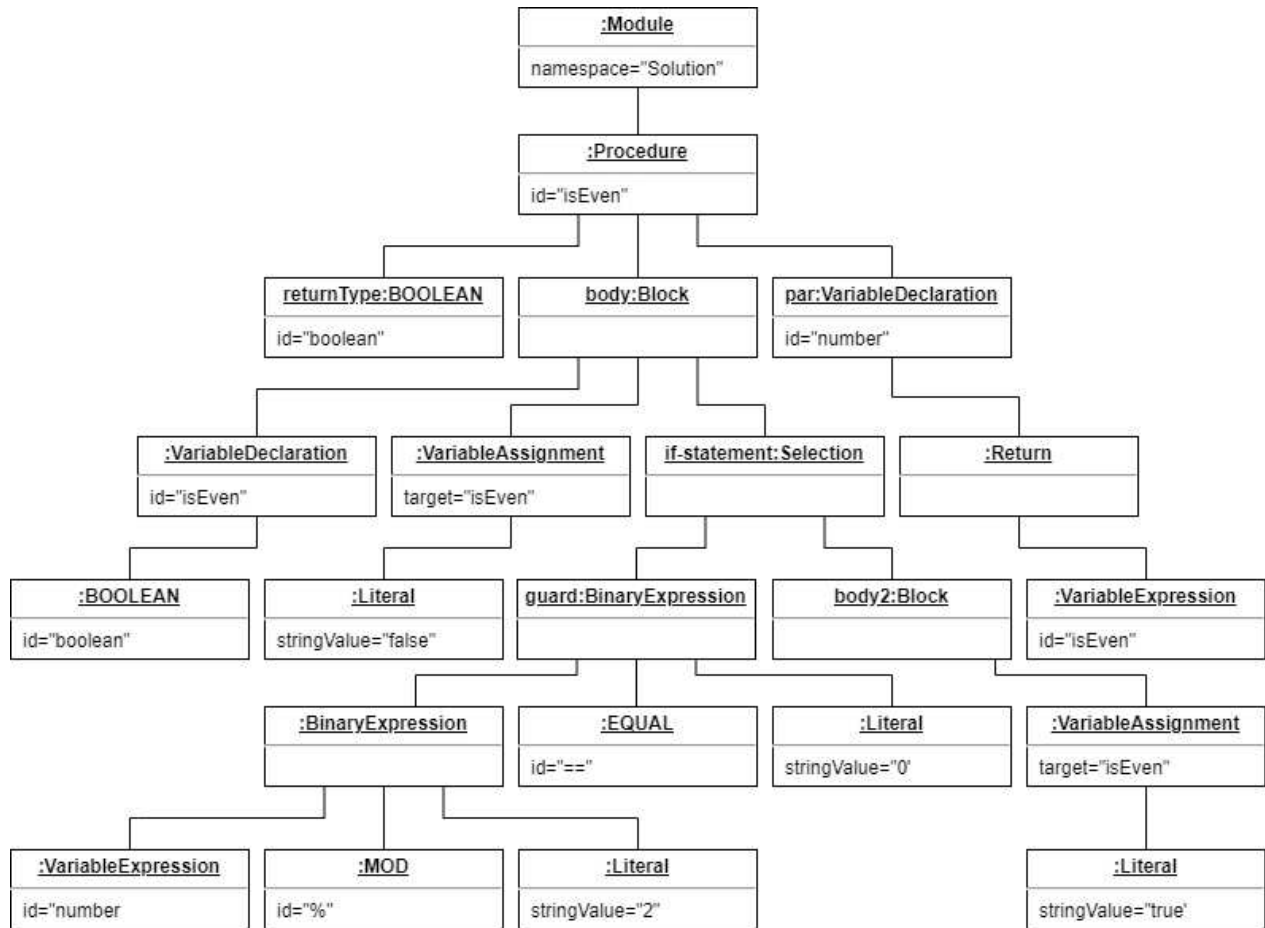


Figure 32. Example of structure generated by Strudel for Is Even exercise (Listing 1)

4.3.2.5. *generateHints* Finally, this method will compare each pair from the previously created list, and based on the results of that comparison, it will give you tips or positive feedback (if there are no hints) based on each generated pair.

4.3.3. Domain and Infrastructure Layers

The domain layer (Figure 30) contains all the entities, enums, exceptions, interfaces, types and logic specific to this layer, it is the layer that should contain the business logic of the application, and has no dependencies on anything external. The domain layer is the layer where the data model used by the application was defined.

The infrastructure layer (Figure 30) should contain classes for accessing external resources such as file systems, web services, and so on. These classes are based on

interfaces defined in the application layer. In this layer, data access was implemented, implementing the repository pattern. For the prototype was decided to store the data in two JavaScript Object Notation (JSON) files (*exercise.json* and *hints.json*), but with the architecture defined, it can be easily replaced by technologies that allow persisting new data for the available resources.

CHAPTER 5

Conclusions and Future Work

In this dissertation, the objective was to demonstrate the idea of a tool based on a single model/exercise solution that can compare with a student's code snippet and then provide useful and reliable programming tips and positive feedback to the student, aiming at providing some guidance and keep them motivated while solving the exercise. One of the research questions was to find out what kind of hints can be useful for the student's guidance, it was concluded that hints including textual explanation can improve the student's immediate programming performance [33], and students who received this type of hint perceived that the provided support is significantly more useful, relevant, and interpretable, and had a better understanding of the provided hint than students who only received code hints [34]. Another question to be answered, and important for the construction of a possible solution, how to compute reliable hints for help the process of solving programming exercises based on a single solution model, we conclude that using abstract syntactic trees combined with an efficient algorithm to compare them is a good start [22][35][36]. Combining this approach with a library (Strudel) that provides important details about each code element, it is possible to overcome some of the limitations found in previous works.

To demonstrate the idea, an API prototype was built that can be integrated into any platform for introductory programming exercises, an UI was also built to visually demonstrate the results achieved. The Strudel library is an important piece of this prototype, allowing deserialization of code text and providing an easy-to-handle object structure. It also provides functionalities that allow, for example, to check the role of a variable, to execute the provided code after deserialization (independently of the programming language), etc. It also means that if Strudel is improved, it could be reflected in an

improvement in this prototype, for example, deserialization of different programming languages.

Our preliminary results establish the technical viability of the approach proposed in this dissertation. In the future, conducting student studies to test the usefulness of the prototype is part of the plan. It is also part of the plan to be able to integrate this API into an introductory programming platform that can provide the innovative feature provided by this prototype API.

Bibliography

- [1] Y. Qian and J. Lehman, 'Students' misconceptions and other difficulties in introductory programming: A literature review', *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1-24, 2017.
- [2] N. C. Brown and A. Altadmri, 'Investigating novice programming mistakes: Educator beliefs vs. student data', in *Proceedings of the tenth annual conference on International computing education research*, 2014, pp. 43-50.
- [3] J. Jackson, M. Cobb and C. Carver, 'Identifying top java errors for novice programmers', in *Proceedings frontiers in education 35th annual conference*, IEEE, 2005, T4C-T4C.
- [4] P. Bayman and R. E. Mayer, 'A diagnosis of beginning programmers' misconceptions of basic programming statements', *Communications of the ACM*, vol. 26, no. 9, pp. 677-679, 1983.
- [5] D. Sleeman *et al.*, 'Pascal and high-school students: A study of misconceptions. technology panel study of stanford and the schools. occasional report# 009.', 1984.
- [6] M. De Raadt, 'Teaching programming strategies explicitly to novice programmers', Ph.D. dissertation, University of Southern Queensland, 2008.
- [7] S. M. M. Rubiano, O. López-Cruz and E. G. Soto, 'Teaching computer programming: Practices, difficulties and opportunities', in *2015 IEEE Frontiers in Education Conference (FIE)*, IEEE, 2015, pp. 1-9.
- [8] J. M. Costa, 'Microworlds with different pedagogical approaches in introductory programming learning: Effects in programming knowledge and logical reasoning', *Microworlds with different pedagogical approaches in introductory programming learning: effects in programming knowledge and logical reasoning*, no. 1, pp. 145-174, 2019.

- [9] T. W. Price, Y. Dong, R. Zhi *et al.*, ‘A comparison of the quality of data-driven programming hint generation algorithms’, *International Journal of Artificial Intelligence in Education*, vol. 29, no. 3, pp. 368-395, 2019.
- [10] T. Barnes and J. Stamper, ‘Toward automatic hint generation for logic proof tutoring using historical student data’, in *International conference on intelligent tutoring systems*, Springer, 2008, pp. 373-382.
- [11] J. Vom Brocke, A. Hevner and A. Maedche, *Design Science Research. Cases*. Springer, 2020.
- [12] K. Pfeffers, T. Tuunanen, C. E. Gengler *et al.*, ‘The design science research process: A model for producing and presenting information systems research’, in *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006)*, Claremont, CA, USA, 2006, pp. 83-106.
- [13] C. Wohlin, ‘Guidelines for snowballing in systematic literature studies and a replication in software engineering’, in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1-10.
- [14] M. Melo and G. L. Miranda, ‘Applying the 4c-id model to the design of a digital educational resource for teaching electric circuits: Effects on student achievement’, in *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*, 2014, pp. 8-14.
- [15] Z. Güney, ‘Four-component instructional design (4c/id) model approach for teaching programming skills.’, *International Journal of Progressive Education*, vol. 15, no. 4, pp. 142-156, 2019.
- [16] J. M. Costa, G. L. Miranda and M. Melo, ‘Four-component instructional design (4c/id) model: A meta-analysis on use and effect’, *Learning Environments Research*, pp. 1-19, 2021.
- [17] J. Frerejean, J. J. van Merriënboer, P. A. Kirschner, A. Roex, B. Aertgeerts and M. Marcellis, ‘Designing instruction for complex learning: 4c/id in higher education’, *European Journal of Education*, vol. 54, no. 4, pp. 513-524, 2019.

- [18] J. Sweller, *Cognitive load theory and educational technology. education tech research dev* 68, 1-16, 2020.
- [19] B. J. Ericson, L. E. Margulieux and J. Rick, 'Solving parsons problems versus fixing and writing code', in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, 2017, pp. 20-29.
- [20] Y. Du, A. Luxton-Reilly and P. Denny, 'A review of research on parsons problems', in *Proceedings of the Twenty-Second Australasian Computing Education Conference*, 2020, pp. 195-202.
- [21] T. Lehtinen, A. L. Santos and J. Sorva, 'Let's ask students about their programs, automatically', in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, IEEE, 2021, pp. 467-475.
- [22] K. Zimmerman and C. R. Rupakheti, 'An automated framework for recommending program elements to novices (n)', in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2015, pp. 283-288.
- [23] A. L. Santos, 'Javardise: A structured code editor for programming pedagogy in java', in *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, 2020, pp. 120-125.
- [24] S. Russell, 'Automated code tracing exercises for cs1', in *Computing Education Practice 2022*, 2022, pp. 13-16.
- [25] J. Hattie and H. Timperley, 'The power of feedback', *Review of educational research*, vol. 77, no. 1, pp. 81-112, 2007.
- [26] K. Zhang and D. Shasha, 'Simple fast algorithms for the editing distance between trees and related problems', *SIAM journal on computing*, vol. 18, no. 6, pp. 1245-1262, 1989.
- [27] D. Gale and L. S. Shapley, 'College admissions and the stability of marriage', *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9-15, 1962.
- [28] M. Kuittinen and J. Sajaniemi, 'Teaching roles of variables in elementary programming courses', in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, 2004, pp. 57-61.

- [29] R. Richards, ‘Representational state transfer (rest)’, in *Pro PHP XML and web services*, Springer, 2006, pp. 633-672.
- [30] R. C. Martin, ‘Clean architecture: A craftsman’s guide to’,
- [31] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.
- [32] E. Evans and E. J. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
- [33] S. Marwan, J. Jay Williams and T. Price, ‘An evaluation of the impact of automated programming hints on performance and learning’, in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 2019, pp. 61-70.
- [34] S. Marwan, N. Lytle, J. J. Williams and T. Price, ‘The impact of adding textual explanations to next-step hints in a novice programming environment’, in *Proceedings of the 2019 ACM conference on innovation and technology in computer science education*, 2019, pp. 520-526.
- [35] B. Fein, F. Obermüller and G. Fraser, ‘Catnip: An automated hint generation tool for scratch’, in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, 2022, pp. 124-130.
- [36] R. Singh, S. Gulwani and A. Solar-Lezama, ‘Automated feedback generation for introductory programming assignments’, in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013, pp. 15-26.