Codeflex2.0

Competitive Programming web application

Miguel M. Soares
Polytechnic Institute of Guarda

Celestino Gonçalves
Polytechnic Institute of Guarda

Abstract **—**Codeflex is a competitive programming platform that enables the creation and participation of users in competitive programming tournaments through the automatic compilation and evaluation of code submissions. Developed by a former student of the Polytechnic Institute of Guarda's Computer Engineering program in 2018 **[1] [2]**, the platform is implemented over a web services architecture and aims to provide students with a platform for practicing programming problems.

In 2022, Codeflex was extended with new functionalities and updates to better support the evaluation of students in competitive programming tournaments. This includes the expansion of the compilation capability to include programming languages currently taught in the course, as well as the implementation of tools for improved performance visualization and plagiarism detection. These updates have further enhanced the platform's value as a reliable and powerful resource for students and instructors at the Institute.

Keywords - Web application; Web services; Automatic evaluation of code submissions; Competitive Programming, Plagiarism detection.

I. MOTIVATION

The motivation for this project was twofold: to acquire knowledge in the latest technologies such as React, Java, MySQL, HTML, and CSS, and to tackle the challenge of developing functionalities in a competitive programming application. Working on a project that builds upon an existing architecture presented the added complexity of understanding its operation and the technologies and techniques used. However, this also provided an opportunity to apply and further develop technical skills.

II. OBJECTIVES

The main objective of this project was to include the ability to compile and evaluate submissions in Haskell and Prolog, improve the visualization of user submissions, and implement plagiarism detection tools. The following steps were taken to achieve this goal:

- Analysis of existing technologies and architecture.
- Deployment and testing of the original platform.
- Adding predefined functionalities to support a competitive environment.
- Adaptation for compiling and evaluating Haskell and Prolog submissions.
- Redesign of the user interface, including the change from class to functional components.
- Implementation of tournament submission viewing for the manager.
- Addition of plagiarism detection functionality for tournament submissions.
- Testing of the platform in a real-world context.
- Documentation to support future platform expansion.

III. STATE OF THE ART

This study is an essential step to identify and try to close the existing gaps in the current application model and to support decision making about which tools and technologies are suitable to be used. Initially, research was conducted about the current competitive programming applications, Table 1

Table 1- Comparison of Functionalities by Platform

Platform / Features	Codeflex [2]	LeetCode [3]	HackerRank [4]	Codeforces [5]	CodeChef [6]	HackerEarth [7]	TopCoder [8]	Jutge [9]
Tournaments	✓	✓	✓	✓	✓	✓	✓	✓
Challenge Practice	✓	✓	✓	✓	✓	✓	✓	✓
Tournament simulation		✓						
Achievements			✓		✓		✓	✓
Ranking	✓	✓	✓	✓	✓	✓	✓	✓
Job search			✓			✓	✓	
Networking		✓	✓	✓	✓	✓	✓	✓
Mock interviews		✓						
Private Tournaments	✓							
Plagiarism detection	✓		✓		✓	✓		
Ranking similarweb	--	2134	6,730	6,845	25,510	31,282	103,488	883,75

A. Existing applications competitive programming

From the applications presented above, the LeetCode, HackerRank and Jutge platforms were chosen for a deeper and more critical analysis compared to the solution to be developed.

HackerRank and LeetCode were selected because they are the most complete platforms and have the best ranking according to similarweb, Jutge is analyzed because it has a lower ranking.

B. *Critical Analysis*

The main highlights of the LeetCode platform are its focus on interview preparation and learning, however it does not allow the user to create tournaments, and the diversity of tournaments is inferior to HackerRank.

HackerRank proves to be a very solid solution and is the platform that most closely resembles the intended goals, however the functionality for creating tournaments does not allow them to be classified as private. Jutge, on the other hand, is a platform with a great diversity of exercises **[10]**, games, reward systems, but it is more focused on learning than on competitiveness.

One of the main gaps that the new solution intends to fill is the creation of private tournaments, which will provide a greater facility to the user in situations such as Hackathons/onsite competitions and can also be used as a method of evaluation and/or preparation of students for disciplines involving programming.

Next, a second survey was performed about the available plagiarism detection tools and their characteristics, Table 2.

Table 2- Feature comparison per plagiarism detection tool

Tools / Features	SIM [11]	AC2 [12]	MOSS [13]	Jplag [14]	Plaggie [15]	Sherlock [16]	Holmes [17]
Open source	✓	✓			✓		✓
Supported programming languages	8	9	40	12	1	3	1
Extensible	✓	✓		✓			
Local / Web	Loc.	Loc.	Web	Web	Loc.	Loc.	Loc.
Haskell support			✓				✓
Prolog support							

C. *Existing tools for Plagiarism analysis*

MOSS (Measure of software similarity) **[13]** and SIM (Software Similarity Tester) **[18]** were selected because they have opposite characteristics.

D. *Critical analysis*

As shown in Table 2, MOSS and SIM offer distinct advantages and disadvantages when considering their use for plagiarism analysis. It is important to consider both tools based on the application's needs, considering several key criteria. Online/local availability, extensibility, and performance.

- Online / Local – In one of the tests performed we verified instability with the MOSS service **[19] [20]**, the integration of a tool provided as an online service would create a dependency on third parties, which could affect the performance of the platform if implemented.
- Extensible – Although many of the platforms offer the ability to analyze several programming languages, none of the tools analyzed allow for the analysis of Prolog code. However, a tool that allows extensibility can be adapted as needed.

IV. Methodology

The process of updating a platform for real use can be complex, especially when working on a large project. My solution was to use Scrum, a type of agile project management. However, since I was the only one working on the project, I had to modify the process from the typical Scrum setup. In my internship, I met regularly with my internship supervisor, who was also my project manager, to discuss what should be implemented in the platform. This approach allowed me to have a clear and comprehensive approach to developing the platform and ensured that the project was fine-tuned and ready for use in the real context.

V. Requirements Analysis

To meet the requirements of the system, an analysis was conducted using the Unified Modeling Language (UML). This visual modeling language was used to plan some of the application's use cases and establish the initial objectives:

- Adding predefined features and rules to tournaments to allow the platform to be used in a competitive environment.
- Haskell and Prolog compilation and evaluation.
- Greater control and vision by the manager.
- Plagiarism detection in submissions.

VI. System Architecture

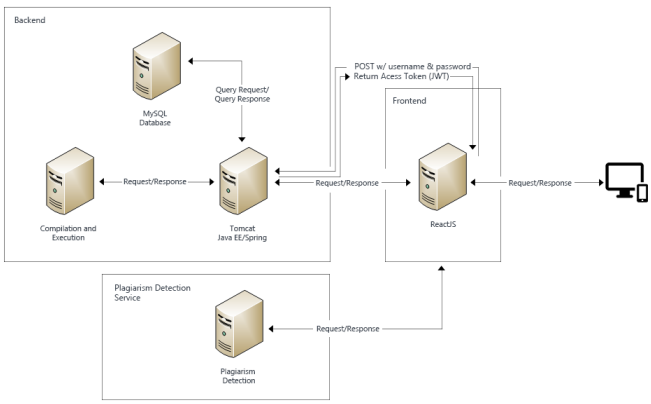

Figure 1- System architecture

The platform is supported by a web services architecture, Figure 1 that promotes modularity and flexibility. All services were then converted to Docker containers, allowing for better organization, and simplifying the application deployment process. Additionally, the plagiarism system was added in a modular form, allowing it to work independently and without compromising the current performance of the platform.

The platform security is done through a token authentication system using JWT (JSON Web Tokens) **[21]** that guarantees which accesses and information are sent to the users.

VII. Compilation Process

The compilation process, as illustrated in Figure 2, begins when the user submits a solution for a given exercise. This submission is sent to the backend through a secure SSH (Secure Shell) connection and compiled to produce a result. This result is then compared with the solution to the problem in order to evaluate the submission.

Figure 2- Submission Compilation Process

On the compilation server the file, depending on the language, needs instructions to be compiled. The following command illustrates the instruction for compiling Java via the javac compiler, assuming the file is in the location where the command is executed.

```
javac submissao.java 2> compilerError;
```

This will compile the file submission and any errors will be saved in the compilerError file. Once the file is compiled, it can be executed with the following command:

```
java submission 2> run_error.txt > run_output.txt;
```

In the case of an error, the output will be placed in runerror.txt, while successful execution will be placed in runoutput.txt. If there are no errors detected in the compilation, the result will be placed in the last file, which will then be compared with the test cases to come up with the evaluation.

VIII. Alernative Compilation Process – Drivers

Not all compilers use the same parameters and not all compilers work in the same way. In order to be able to compile the new languages it was necessary to create drivers that allowed this compilation without compromising the functionality of the existing compilation processes.

The simplest example of necessary changes is the import of libraries, assuming that it is necessary to use a certain library in the solution to be submitted by the student, and that this library as it is present in most of the IDE´s, is considered default, and it does not make much sense for the student to include the library along with the code.

The drivers come to solve this problem, through code injection, transparently where needed.

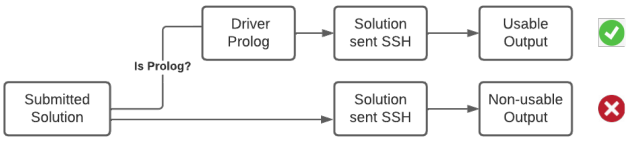

Figure 3- Driver and driverless submission process

When the submission is sent, depending on the language selected, the corresponding driver is used, Figure 3

IX. Plagiarism Analysis

The evaluation of the available tools had a great influence on the decision about the possible solution. It was decided that the tool should work locally, thus eliminating the dependency on an external service, which could fail and compromise the platform's operation at any time. Additionally, the extensibility of the tool was considered. For the development, it was always kept in mind that the platform should be updated, and that new programming languages should be supported. Therefore, it was necessary for the tool to be expandable, in order to keep up with the evolving needs of the platform.

A. Extension of SIM

The decision fell on the SIM Tool, Figure 4, because except for the support of out-of-the-box languages all other characteristics were favorable for integration with the application.

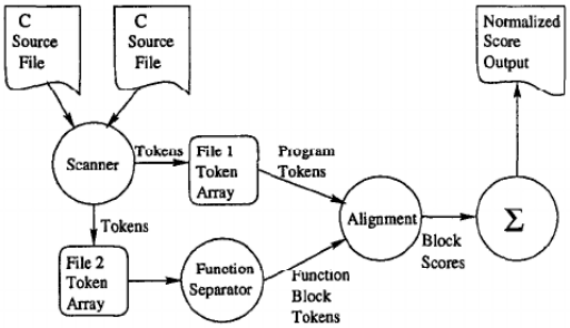

Figure 4- SIM Tool Operation process [18]

Plagiarism detection methods fall into three main categories in the way they evaluate code, through structure, semantics and behavior **[22] [23]**. SIM uses a structural analysis approach, where similarity is calculated by matching the code structure between two files. It works by converting input files to tokens through lexical analysis, where each token can represent an arithmetic function, a logical operation, a string, a comment, etc. In the case of the following example.

```
for (i = 0; i < max; i++)
```

It is converted to:

```
TKN_FOR TKN_LPAREN TKN_ID_I TKN_EQUALS TKN_ZERO TKN_SEMICLN
```

After both files are converted, two sets of tokens result, one is split into sections that represent a module of the other set. Each of these modules is then aligned separately. This technique allows similarity detection even when the positions of the modules are swapped.

The form of extension is through the addition of dictionaries, for each language, which are used when converting the files to tokens. So a dictionary was created for the Prolog language.

As a result we have a bash script capable of evaluating Prolog files by inputting two files.

```
./script_name file1.pl file2.pl
```

B. *Parsing Service*

Considering the initial objectives, it was necessary to add the ability to detect similar submissions. This automated analysis simplifies and streamlines the manual analysis process, which is quite laborious.

The Analysis Service provides a solution to interconnect the previously developed solution with the Codeflex platform, enabling analysis of user submissions in a tournament. The service was designed to be used within the platform, but its modular design as a script with no dependencies makes it usable as an independent tool.

A NodeJS server was created to receive submissions through an API, recreate them on the server, and perform analysis to produce a report for the manager. The server uses the following process to parse submissions:

- Receive JSON submissions.
- Remove any folder from previous analysis.
- Create a /tmp folder.
- Create folders for each user and subfolders for each problem.
- Decode the submissions, storing each in a Prolog file within the corresponding folders.
- Generate commands for all submissions for a problem.
- Evaluate all submissions using the previously developed script.
- Return the analysis report.

The service organizes submissions by problem by creating subfolders, storing the path of each in an object for easy execution:

```
 }
  problem: '5',
   exec_path: ' ./tmp/RicardoAndrade/5/24.pl ./tmp/RicardoAndrade
./tmp/PauloTomasLda/5/29.pl ./tmp/PauloTomasLda/5/32.pl ./tmp/Paul
./tmp/Guilherme Alves/5/31.pl ./tmp/NobleLip/5/26.pl ./tmp/NobleLi
./tmp/RodrigoMartins/5/39.pl'
 },
 {
   problem: '6',
   exec_path: ' ./tmp/RicardoAndrade/6/30.pl ./tmp/NobleLip/6/37.pl'
 }
```

Then the script is run for each problem, the Result has the following structure:

```
File ./tmp/mustafa.bukh/4/7.pl: 25 tokens, 3 lines
File ./tmp/mustafa.bukh/4/14.pl: 25 tokens, 3 lines
File ./tmp/NobleLip/4/12.pl: 29 tokens, 6 lines
File ./tmp/NobleLip/4/13.pl: 29 tokens, 6 lines
File ./tmp/NobleLip/4/15.pl: 33 tokens, 7 lines
File ./tmp/Rui Condesso/4/16.pl: 27 tokens, 2 lines
File ./tmp/RodrigoMartins/4/18.pl: 27 tokens, 2 lines
File ./tmp/afonso/4/20.pl: 20 tokens, 2 lines
File ./tmp/DiogoNeto/4/33.pl: 33 tokens, 9 lines

./tmp/PauloTomasLda/4/4.pl consists for 100 % of ./tmp/DiogoNeto/4/33
./tmp/Rui Condesso/4/16.pl consists for 100 % of ./tmp/RodrigoMartins
./tmp/1705091/4/41.pl consists for 86 % of./tmp/raquelvidal99@hotmail
./tmp/NobleLip/4/15.pl consists for 76 % of ./tmp/afonso/4/38.pl mate
```

This result is sent back to the platform via the API, thus making it an independent plagiarism detection service.

X. Validation in Real Context

The use of Codeflex in real context was tested through programming tournaments in the Haskell and Prolog languages at the Polytechnic Institute of Guarda.

A. *Haskell Tournament*

The first tournament, for the Haskell language, took place on May 06, 2022, and involved 5 students competing to solve 5 problems. During the tournament, various issues were identified, including a lack of visualization of student submissions, no plagiarism detection, and a crash due to an invalid character in a student submission. These issues highlighted the need for further development and improvements to the platform before the Prolog Tournament.

B. *Prolog Tournament*

The Prolog tournament took place on December 14, 2022, as part of the Artificial Intelligence unit. The tournament had 23 students, who competed in solving 4 problems., which was in the final stages of development, showed that the platform performed as expected and provided an effective means for evaluating student performance and detect plagiarism on the submissions.

I. Conclusion

In conclusion, the development of the extended version of Codeflex was a challenging project that required a thorough understanding of the existing architecture and technologies. By implementing new functionalities such as the ability to compile and evaluate submissions in Haskell and Prolog, improved visualization of user submissions, and plagiarism detection tools, the platform has become a valuable resource for students and instructors. These changes bring significant benefits to the institution, instructors, and students by streamlining the code evaluation process and ensuring that plagiarism is effectively detected. The newly developed functionalities are a significant improvement over traditional methods and are poised to have a positive impact on the field of competitive programming and student evaluation.

BIBLIOGRAPHIC REFERENCES

[1] M. Brito e C. Goncalves, "Codeflex: a web-based platform for competitive programming.," In 2019 14th Iberian Conference on Information Systems and Technologies (CISTI) (pp. 1-6). IEEE., 2019, June.

[2] "Repositório Github Codeflex - Miguel Brito," [Online]. Available: https://github.com/miguelfbrito/Codeflex. [Acedido em 10 12 2022].

[3] "Leet Code Landing page," [Online]. Available: https://leetcode.com/. [Acedido em 15 12 2022].

[4] "HackerRank Landing page," [Online]. Available: https://www.hackerrank.com/. [Acedido em 15 12 2022].

[5] "Codeforces Landing page," [Online]. Available: https://codeforces.com/. [Acedido em 15 12 2022].

[6] "Codechef Landing page," [Online]. Available: https://www.codechef.com/. [Acedido em 15 12 2022].

[7] "HackerEarth Landing page," [Online]. Available: https://www.hackerearth.com/. [Acedido em 15 12 2022].

[8] "TopCoder Landing page," [Online]. Available: https://www.topcoder.com/. [Acedido em 15 12 2022].

[9] "Jutge Landing page," [Online]. Available: https://jutge.org/. [Acedido em 15 12 2022].

[10] J. Petit, . S. Roura, J. Carmona, J. Cortadella, F. I. J. Duch, O. Gimenez, A. Mani, J. Mas, E. Rodriguez-Carbonell, E. Rubio, E. de San Pedro e D. Venkataramani, "Jutge.org: Characteristics and Experiences," IEEE Transactions on Learning Technologies (Volume: 11, Issue: 3), 01 July-Sept. 2018.

[11] "Repositório Github Software similarity tester SIM - mpanczyk," [Online]. Available: https://github.com/mpanczyk/sim. [Acedido em 15 12 2022].

[12] "Repositório Github AC2 Source code plagiarism detection tool - manuel-freire," [Online]. Available: https://github.com/manuel-freire/ac2. [Acedido em 15 12 2022].

[13] "Moss - A System for Detecting Software Similarity," [Online]. Available: https://theory.stanford.edu/~aiken/moss/. [Acedido em 23 11 2022].

[14] "Repositório Github JPlag Token-Based Software Plagiarism Detection - jplag," [Online]. Available: JPLAG. [Acedido em 18 12 2022].

[15] "Plaggie Landing page," [Online]. Available: https://www.cs.hut.fi/Software/Plaggie/. [Acedido em 15 12 2022].

[16] "Repositório Github The Sherlock Plagiarism Detector - diogocabral," [Online]. Available: https://github.com/diogocabral/sherlock. [Acedido em 18 12 2022].

[17] J. Hage, B. Vermeer e G. Verburg, "Plagiarism Detection for Haskell with Holmes," 2013.

[18] D. Grune, "The software and text similarity tester SIM," [Online]. Available: https://dickgrune.com/Programs/similarity_tester/. [Acedido em 23 11 2022].

[19] "Moss is not working #56," [Online]. Available: https://github.com/soachishti/moss.py/issues/56. [Acedido em 15 12 2022].

[20] "Moss not working? #55," [Online]. Available: https://github.com/soachishti/moss.py/issues/55. [Acedido em 15 12 2022].

[21] "JSON Web Tokens," [Online]. Available: https://jwt.io/. [Acedido em 15 9 2022].

[22] A. S. M. BEJARANO, L. . E. GARCI´A e E. E. ZUREK, "Detection of Source Code Similitude in Academic Environments," Barranquilla, Colombia,, 2012.

[23] J. C. Paiva, J. P. Leal e Á. Figueira, "Automated assessment in computer science education: A state-of-the-art review.," ACM Transactions on Computing Education (TOCE), 22(3), 1-40., 2022.