



Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

CODEFLEX – APLICAÇÃO WEB DE PROGRAMAÇÃO COMPETITIVA

MIGUEL ANTÓNIO FERRÃO BRITO

RELATÓRIO PARA A OBTENÇÃO DO GRAU DE LICENCIADO
EM ENGENHARIA INFORMÁTICA

Julho 2018

Ficha de Identificação

Aluno: Miguel António Ferrão Brito

Nº 1011695

Licenciatura: Engenharia Informática

Estabelecimento de Ensino:

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

Orientador de Estágio:

Nome: Professor Celestino Gonçalves

Grau académico: Mestre

Agradecimentos

Queria começar por agradecer ao meu orientador, Professor Celestino Gonçalves, por toda a disponibilidade mostrada ao longo do desenvolvimento em especial pelas reuniões realizadas praticamente todas as semanas. Agradeço também por todas as críticas e sugestões que tiveram sem dúvida um papel importante no desenvolvimento do projeto. O meu sincero obrigado.

Agradeço também ao Professor José Fonseca por todo o conhecimento transmitido durante as aulas de projeto, e sobretudo pelas críticas que ajudaram na identificação de algumas lacunas.

Por fim, agradeço à minha família por todo o apoio e ânimo prestado nesta fase mais difícil e stressante.

Resumo

O presente documento descreve o processo de planeamento e implementação de uma aplicação *web* no âmbito da unidade curricular de Projeto de Informática, integrada na Licenciatura de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O meio académico é sem dúvida uma das principais vias de introdução na área da programação, no entanto, neste meio onde a prática é crucial para o desenvolvimento de bases sólidas, a quantidade de exercícios realizados fica bastante aquém da pretendida, em especial pelo processo trabalhoso que é a avaliação de código.

Numa tentativa de colmatar esta falha, foi implementada uma aplicação *web* que servirá como um repositório de problemas (típicos da área da programação) para prática e aprendizagem, mas que também contará com uma parte competitiva, e essencialmente, terá de ser capaz de avaliar as submissões dos utilizadores no momento. A solução para a avaliação das submissões passa pela utilização de compiladores associados à respetiva linguagem de programação e uma execução paralela para melhorias na eficiência com que os recursos do módulo de avaliação são utilizados.

Palavras-Chave: Aplicação *Web*, *web services*, Java EE, *Spring*, Programação Competitiva, *React*

Abstract

This document describes the process of planning and implementation of a web applicaton within the curricular unit of Project at the Polytechnic Institute of Guarda.

The academic environment is undoubtedly one of the main means of introduction to the field of programming, however, in this environment where practice is crucial for the development of solid foundations, the number of given assignments falls short from ideal, especially because of the time it takes to evaluate.

To fill this gap, a web application has been implemented to serve as a repository of problems(typical to programming) for practice and learning, but will also contain a competitive section. Above all things, the app should be capable of evaluating user's submissions at the time. The solution for evaluating submissions involves the use of compilers associated with their programming language and a parallel execution to bring improvements in resource usage efficiency.

Keywords: Web Application, web services, Java EE, Spring, Competitive Programming, React

Índice

Agradecimentos	III
Resumo	IV
Abstract.....	V
Índice	VI
Índice de Figuras	IX
Índice de Tabelas	XI
Siglário	XII
1 Introdução	1
1.1 Motivação.....	1
1.2 Objetivos	1
1.3 Estrutura do Documento.....	2
2 Estado da Arte.....	3
2.1 Aplicações existentes	4
2.1.1 LeetCode.....	5
2.1.2 HackerRank	6
2.2 Análise crítica das soluções existentes	8
3 Metodologia	11
3.1 Metodologia de desenvolvimento: Scrum.....	12
4 Análise de Requisitos.....	15
4.1 Diagrama de Contexto.....	17
4.2 Atores e respetivos casos de uso	18
4.3 Diagrama de casos de uso	19

4.4	Descrição dos casos de uso e Diagramas de Sequência	20
4.4.1	Submeter solução para um problema.....	20
4.4.2	Criação de problema público	23
4.5	Diagrama de Classes	25
5	Implementação da Solução	29
5.1	Tecnologias Utilizadas	29
5.1.1	Java EE	29
5.1.2	JavaScript.....	30
5.1.3	ReactJS.....	31
5.1.4	MySQL	32
5.1.5	HTML	32
5.1.6	CSS	32
5.1.7	Git	33
5.1.8	Compiladores	33
5.2	Arquitetura do Sistema.....	35
5.3	Compilação, Execução e Avaliação de Submissões	37
5.3.1	Segurança.....	39
5.3.2	Performance	40
5.4	Classificação Elo	43
5.5	Interfaces da aplicação <i>web</i>	47
5.5.1	Interface inicial (<i>Default</i>).....	47
5.5.2	Interface <i>Login/Registo</i>	48
5.5.3	Interface de Prática	49
5.5.4	Interface de Listagem de Problemas	50

5.5.5 Interface do Problema.....	51
5.5.6 Interface da Vista de Resultados.....	54
5.5.7 Interface do Perfil	55
5.5.8 Interface de Gestão de Conteúdos	56
6 Testes	57
7 Processo de deploy da aplicação <i>web</i>	61
8 Conclusão.....	63
Referências Bibliográficas.....	65
9 Anexos	67
9.1 Código	67
9.1.1 Cálculo de <i>rating</i> para participantes num torneio.....	67
9.1.2 Compilação, Execução e Avaliação.....	68
9.2 Interfaces	77

Índice de Figuras

Figura 1 - Categorias da plataforma LeetCode.....	5
Figura 2 - Landing page da plataforma HackerRank	7
Figura 3 - Diagrama de funcionamento do Scrum	13
Figura 4 - Diagrama de Contexto	17
Figura 5- Diagrama de Casos de Uso	19
Figura 6 - Diagrama de Sequência "Submeter solução para um problema"	22
Figura 7 - Diagrama de Sequência "Criação de Problema Público"	24
Figura 8- Diagrama de Classes	26
Figura 9 - Modelo ER.....	27
Figura 10 - React Virtual DOM vs Typical DOM.....	31
Figura 11- Arquitetura do sistema	36
Figura 12 - Comparação dos resultados dos tempos de execução.....	42
Figura 13 - Classe de Cálculo do Rating	46
Figura 14- Interface inicial	48
Figura 15- Interface de Login/Registo.....	48
Figura 16- Página de Prática.....	49
Figura 17 - Listagem de problemas	50

Figura 18 - Interface do Problema - parte1.....	51
Figura 19 - Interface do Problema - parte2.....	52
Figura 20- Editor ACE	53
Figura 21 - Sumário de submissões.....	53
Figura 22 - Interface de vista de resultados.....	54
Figura 23 - Código submetido.....	55
Figura 24 - Página de perfil	55
Figura 25 - Interface Manage Content.....	56
Figura 26- Coleções no Postman.....	58
Figura 27 - Snippet de testes	59

Índice de Tabelas

Tabela 1- Comparação de funcionalidades por plataformas	4
Tabela 2 - Atores e respectivos casos de uso	18
Tabela 3 - Descrição de Caso de Uso “Submeter solução para um problema”	21
Tabela 4 - Descrição de Caso de Uso "Criação de Problema Público"	23
Tabela 5 - Linguagem e seu compilador	33
Tabela 6 - Testes de performance - Concorrente.....	41
Tabela 7 - Testes de Performance - Paralela	41
Tabela 8- Confronto com adversários	43

Siglário

API	Application Programming Interface
AWS	Amazon Web Services
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DOM	Document Object Model
EE	Enterprise Edition
ER	Entity Relationship
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWT	JSON Web Token
SQL	Structured Query Language
SSH	Secure Shell
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1 Introdução

O presente relatório descreve o projeto desenvolvido pelo aluno Miguel António Ferrão Brito, no âmbito da unidade curricular de Projeto de Informática, unidade curricular do terceiro ano do curso de Engenharia Informática.

1.1 Motivação

Nos dias que correm existe uma grande procura no que diz respeito à área das Tecnologias da Informação, sendo a programação um dos focos do mercado. Nesta área, a prática e o estudo de conceitos como algoritmos e estruturas de dados é um dos fatores nucleares para o crescimento enquanto programador.

Sendo o meio académico uma das principais vias onde é feita a introdução à programação e onde são desenvolvidas bases fundamentais, é essencial que haja um aumento na quantidade de exercícios resolvidos por parte dos alunos, que se encontra, no entanto, longe do ideal devido ao trabalho e complexidade envolvido na avaliação dos mesmos. [1]

Neste contexto, as aplicações web de programação competitiva têm uma grande importância como um método de preparação, estudo e avaliação, e como tal, existe a necessidade de uma aplicação que possa reunir problemas tipo para facilitar uma maior imersão dos utilizadores com a resolução de problemas recorrendo à programação.

A nível pessoal, o interesse surgiu pela participação em alguns torneios no passado e pela consideração deste desafio como interessante para aplicar e solidificar os conhecimentos adquiridos ao longo da licenciatura, além de permitir explorar novas tecnologias.

1.2 Objetivos

Este projeto tem como objetivo a implementação de uma aplicação *web* que permitirá a prática de programação competitiva. Esta prática consiste na criação de soluções para um

determinado problema, proposto por outros utilizadores, recorrendo a código que é posteriormente compilado e executado relativamente a *inputs* predefinidos. Caso o código submetido pelo utilizador gere os *outputs* pretendidos considera-se a resposta como válida. Tendo em vista o proposto, a aplicação *web* deverá cumprir os seguintes objetivos:

1. Aplicação *web*.
2. Sistema de autenticação de utilizadores.
3. Desafios para prática sem limitações de tempo, organizados por categorias.
4. Torneios limitados por tempo.
5. Classificação de utilizadores utilizando o *rating* Elo¹.
6. Editor de texto embutido.
7. Compilação e avaliação das soluções submetidas.
8. Criação de torneios privados por parte do utilizador final.

1.3 Estrutura do Documento

O relatório encontra-se dividido em nove capítulos. Neste primeiro é feita uma introdução e descrita a motivação e os objetivos pretendidos para o projeto. No segundo capítulo realiza-se um estudo e análise de outras plataformas com o objetivo de identificar lacunas e oportunidades de melhoria. No terceiro capítulo é descrita a metodologia de trabalho aplicada a este projeto. De seguida, no quarto capítulo, são apresentados alguns elementos do planeamento e *design* da aplicação. No quinto capítulo é descrito o processo de implementação: desde as tecnologias usadas à arquitetura e algumas interfaces. No sexto capítulo é feita uma breve descrição do processo de testes, seguido dum capítulo de exposição do processo de disponibilização da aplicação (*deploy*). Por último, são apresentadas as conclusões retiradas e sugestões de trabalho futuro, seguidas de um capítulo com as referências bibliográficas.

¹ Método estatístico de cálculo da habilidade de um jogador relativamente a outro.

2 Estado da Arte

A realização do estudo e de uma análise crítica às plataformas existentes na área é um passo essencial para identificar e procurar colmatar as falhas existentes no modelo de aplicações atual.

Inicialmente foram realizadas pesquisas acerca das principais aplicações atuais e das suas principais características, das quais resultou a informação reunida na Tabela 1, relativa às aplicações LeetCode², HackerRank³, CodeChef⁴, HackerEarth⁵, Codeforces⁶, TopCoder⁷.

Esta pesquisa foi baseada nos seguintes parâmetros:

1. Características inovadoras
2. Semelhança com os objetivos apresentados
3. Conhecimento e experiência anterior de algumas das plataformas

² Aplicação de programação competitiva focada em preparação para entrevistas de emprego, <https://leetcode.com/>

³ Aplicação de programação competitiva focada em aprendizagem, <https://www.hackerrank.com/>

⁴ Aplicação de programação competitiva focada em aprendizagem, <https://www.codechef.com/>

⁵ Aplicação de programação competitiva focada em aprendizagem, <https://www.hackerearth.com/>

⁶ Aplicação de programação competitiva focada em competição, <https://codeforces.com/>

⁷ Aplicação de programação competitiva focada em competição, <https://www.topcoder.com/>

Plataforma Características	Codeflex ⁸	LeetCode ²	HackerRank ³	CodeChef ⁴	HackerEarth ⁵	Codeforces ⁶	TopCoder ⁷
Torneios	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Prática de desafios	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Simulação de Torneios	Não	Sim	Não	Não	Não	Não	Não
Conquistas	Não	Não	Sim	Sim	Não	Não	Sim
Classificação	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Procura de Emprego	Não	Não	Sim	Não	Sim	Não	Sim
Networking/Discussão	Não	Sim	Sim	Sim	Sim	Sim	Sim
Mock interviews	Não	Sim	Não	Não	Não	Não	Não
Torneios Privados	Sim	Não	Não	Não	Não	Não	Não
Classificação global Alexa ⁹	--	2878	3844	7145	7381	7524	27773

Tabela 1- Comparação de funcionalidades por plataformas

2.1 Aplicações existentes

Das aplicações apresentadas anteriormente foram escolhidas a *LeetCode* e *HackerRank* para uma análise mais aprofundada e crítica comparativamente à solução que se pretende desenvolver, tendo em vista as suas semelhanças e completude.

⁸ Solução que se pretende desenvolver

⁹ Classificação atribuída pela empresa Alexa, subsidiária da Amazon, a um site de acordo com o seu tráfego. Quanto menor o valor, melhor a classificação do site.

2.1.1 LeetCode

A plataforma *LeetCode* [2] é uma das plataformas mais populares da atualidade e conta em especial com uma enorme variedade de problemas de preparação para entrevistas de emprego na área da programação. Existem categorias de problemas redirecionados para entrevistas a grandes empresas da área, como por exemplo Google, Facebook, Amazon, Microsoft e Apple entre outras. Oferece também sessões de *mock interviews*¹⁰ para utilizadores com subscrição *premium*, na tentativa de simular o ambiente de pressão de uma entrevista.

Para além das categorias de preparação focadas em empresas, possui também uma variedade de tópicos para aprendizagem genérica de Algoritmos e Estruturas de Dados. Até à atualidade a plataforma disponibiliza um total de 816 problemas para resolução. [3]

Na Figura 1 podem ser observadas algumas das categorias anteriormente referidas.

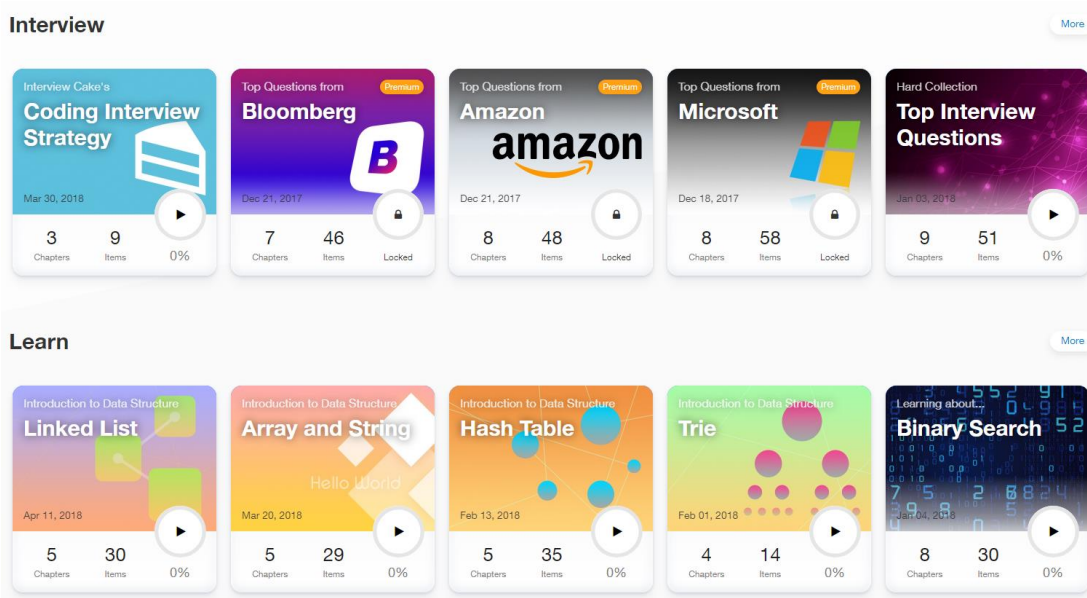


Figura 1 - Categorias da plataforma LeetCode
Fonte: <https://leetcode.com/>

¹⁰ Simulação de entrevistas realizadas com foco em certa categoria ou empresa.

Apesar da plataforma conter todo um foco para preparação e aprendizagem não deixa aquém na parte competitiva. São realizados torneios semanais regularmente e cada um destes torneios conta com um conjunto de 4 questões de dificuldade progressiva, com uma duração total de 90 minutos.

Outro dos pontos de inovação desta plataforma é a possibilidade de um utilizador poder simular a realização de um torneio que já decorreu.

A plataforma contém um editor de texto *online*, proporcionado pela *framework Codemirror* [4] que pretende simular a existência de um *IDE*. Esta *framework* processa também *syntax* de acordo com a linguagem escolhida, tornando o editor mais apelativo e o código mais legível. Possui ainda algumas características básicas de *code completion*. Existe também a possibilidade de escolher o modo do editor de texto, *Vim*, *Emacs* ou Normal, e de escolher entre uma variedade de temas disponibilizados.

A plataforma disponibiliza compiladores para as seguintes linguagens: C++, Java, Python, Python 3, C, C#, Javascript, Ruby, Swift, Go, Scala e Kotlin. Não disponibiliza, no entanto, qualquer informação sobre se utiliza um serviço de APIs para a compilação e avaliação dos problemas, como por exemplo o *Sphere Engine*, ou se contém uma API proprietária.

No geral, esta aplicação oferece uma interface intuitiva e fácil de utilizar com uma enorme variedade e qualidade de problemas para qualquer utilizador que tenha a curiosidade e/ou necessidade de estudar as áreas propostas.

2.1.2 **HackerRank**

A plataforma *HackerRank* [5] é uma das plataformas mais completas da atualidade. Comparativamente com o *LeetCode*, esta plataforma contém algum foco na preparação para entrevistas e na procura de trabalho, sendo usada a nível empresarial para realização de entrevistas técnicas.

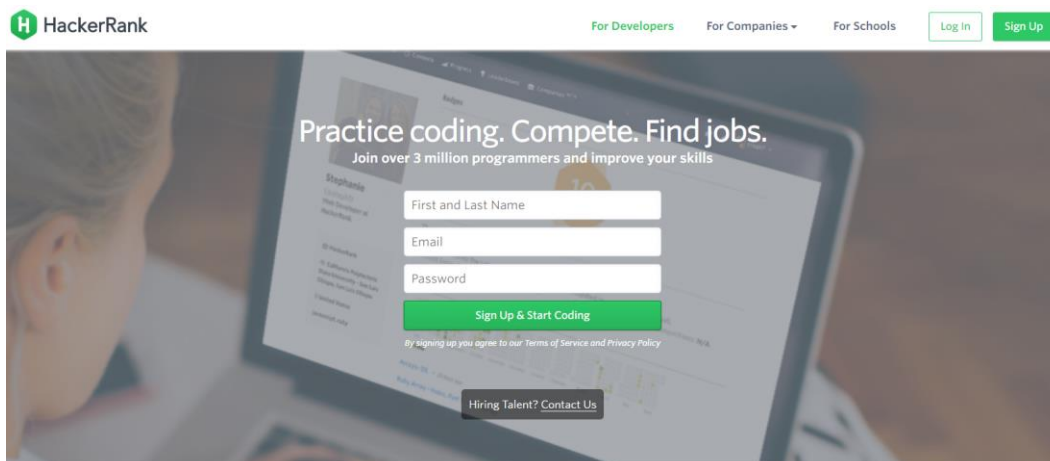


Figura 2 - Landing page da plataforma HackerRank
Fonte: <https://www.hackerrank.com/>

A oferta de problemas tipo estende-se a várias categorias, como por exemplo Algoritmos, Estruturas de Dados, Inteligência Artificial, Bases de Dados entre outras.

No que diz respeito ao nível competitivo, a oferta é bem mais variada pela oferta de problemas de duração semanal, *Week of Code*, que se realizam uma vez por mês, pelas competições de duração de 60 minutos, *Hour of Code*, também realizadas uma vez por mês, para além de todas as outras competições e *Hackathons* promovidos por empresas.

A plataforma contém um editor de texto *online*, proporcionado também pela *framework Codemirror* [4], não variando muito a sua configuração relativamente à plataforma anterior.

A lista de compiladores disponíveis por esta plataforma é bastante extensa, oferecendo BASH, C, Clojure, C++, C++14, C#, D, Erlang, F#, Go, Groovy, Haskell, Java 7, Java 8, Javascript, LOLCode, Lua, Objective-C, OCaml, Pascal, Perl, PHP, Python 2, Python 3, R, Racket, Ruby, Rust, Scala, Swift e VB.Net. Estima-se que a plataforma recorra a um serviço de compilação e avaliação externo e não a software proprietário, dada a variedade de compiladores disponibilizados.

Esta plataforma possibilita a criação e gestão de torneios por parte dos utilizadores algo que não é possível na maioria das plataformas estudadas, no entanto estes torneios são sempre públicos e estão disponíveis para todos os utilizadores.

No geral esta plataforma obteve a sua popularidade pela facilidade com que um novo utilizador se pode iniciar no mundo da programação competitiva, quer pelo seguimento de problemas aconselhados, quer pela regularidade de torneios mais ‘amigáveis’.

2.2 Análise crítica das soluções existentes

Ambas as soluções apresentadas são excelentes soluções que abrangem na sua maioria os objetivos propostos para a criação da nova solução.

A plataforma *LeetCode* tem como principais pontos de destaque o seu foco para preparação de entrevistas e aprendizagem, no entanto não possibilita ao utilizador a criação de torneios e a diversidade de torneios é inferior ao *HackerRank*.

O *HackerRank* é uma solução bastante sólida e é a plataforma que mais se assemelha aos objetivos pretendidos, no entanto a funcionalidade de criação de torneios não permite que sejam classificados como privados.

A criação de torneios privados é uma das principais lacunas que a nova solução pretende colmatar. Fornecerá uma maior facilidade ao utilizador por exemplo em situações de realização de *Hackathons*/competições *onsite* e também poderá ser utilizado como um método de avaliação e/ou preparação de alunos para disciplinas envolvendo programação.

Tendo em conta que um dos desafios destas plataformas passa pela compilação e avaliação do código submetido pelos utilizadores, será feita uma breve análise do principal

serviço da área. O serviço *Sphere Engine* [6] é utilizado por várias aplicações (ex.: *ideone*¹¹, *codechef*), para realizar a compilação e execução das submissões dos utilizadores. É um serviço pago com foco em prestação de serviços a nível empresarial. A oferta de compiladores é extremamente extensiva abrangendo um total de 66 linguagens de programação [7].

As principais características que o serviço fornece são as seguintes:

1. Compilação de submissões
2. Execução de casos de teste relativos a uma submissão
3. Avaliação parcial das submissões
4. Limitações de execução para tempo de CPU e memória
5. Análise da complexidade dos algoritmos

Espera-se que a solução apresentada seja capaz de realizar os 4 primeiros pontos listados anteriormente.

¹¹ Serviço que permite a compilação de código no *browser*. Link: <https://ideone.com/>

3 Metodologia

Uma metodologia de *software* é um procedimento que tem um papel fundamental na estruturação, planeamento e controlo do processo de desenvolvimento de *software*. É o seguimento de uma metodologia que permite que o desenvolvimento seja realizado de forma focada para com as exigências do cliente e com um maior grau de organização e qualidade.

A metodologia Ágil surgiu com o objetivo de trazer mais flexibilidade para o processo de planeamento e desenvolvimento de *software* relativamente às metodologias clássicas (ex.: *Waterfall*), nas quais os seus processos rígidos e pouco iterativos dificultam os ambientes que necessitam de atualização constante, quer por parte da introdução de novas tecnologias, ou por exigências do cliente.

Sendo assim, a metodologia Ágil caracteriza-se, segundo o seu manifesto [8], por priorizar:

- Indivíduos e interações sobre processos e ferramentas – Existe a participação de todos os intervenientes do processo (designers, gestores de projeto e produto, programadores, utilizadores, etc.) a cada iteração.
- *Software* funcional sobre documentação – A implementação dos requisitos e a sua validação por testes são preferíveis à extensiva realização de documentação
- Dar resposta e adaptar às mudanças sobre seguir a rigidez do plano – As mudanças que se consideram como importantes para a evolução do projeto devem ser aplicadas mesmo que não façam parte do plano inicial.

Dentro do espectro da metodologia Ágil optou-se pela utilização do modelo Scrum.

3.1 Metodologia de desenvolvimento: Scrum

O Scrum [9] é uma *framework* baseada em metodologia Ágil que se destaca pela grande iteratividade e possibilidade da realização de um processo incremental. É um modelo eficaz quando ainda não está claro o que se pretende desenvolver na totalidade do projeto. O funcionamento deste modelo pode ser observado na Figura 3 e descreve-se segundo os seguintes pontos:

1. Os requisitos obtidos pelo estudo do mercado e a interação com o cliente são armazenados (*Product Backlog*) de forma ordenada de acordo com a sua prioridade.
2. São selecionados n requisitos do *Product Backlog* para o planeamento de um *sprint*¹² de acordo com a sua prioridade e métricas¹³. Os *sprints* duram tipicamente entre 1 a 4 semanas. Nesta fase determina-se essencialmente quem faz o quê.
3. Ao longo do *sprint* são realizadas reuniões diárias de curta duração e informais, envolvendo toda a equipa onde cada interveniente deve procurar responder às seguintes questões:
 - a. “O que fiz desde a última reunião?”
 - b. “O que pretendo ter feito até à próxima reunião?”
 - c. “Que dificuldades estou a ter?”

Estas questões pretendem transmitir à equipa o estado do trabalho atual, planear trabalho futuro e criar um espírito de entreajuda para situações de dificuldade.

4. No fim de cada *sprint* é feita uma revisão para avaliar quais os objetivos cumpridos. Por norma, a equipa demonstra as funcionalidades implementadas. É também

¹² Período de tempo no qual a equipa se propõe a realizar o que é definido.

¹³ Obtidas essencialmente por experiência passada de desenvolvimento. Uma das principais será o tempo de implementação.

importante nesta fase realizar uma retrospectiva tendo em vista analisar o que pode ser melhorado para *sprints* futuros.

5. Cada *sprint* contará como um incremento no avanço do projeto. O processo descrito realiza-se várias vezes até que o cliente se encontre satisfeito com o resultado.

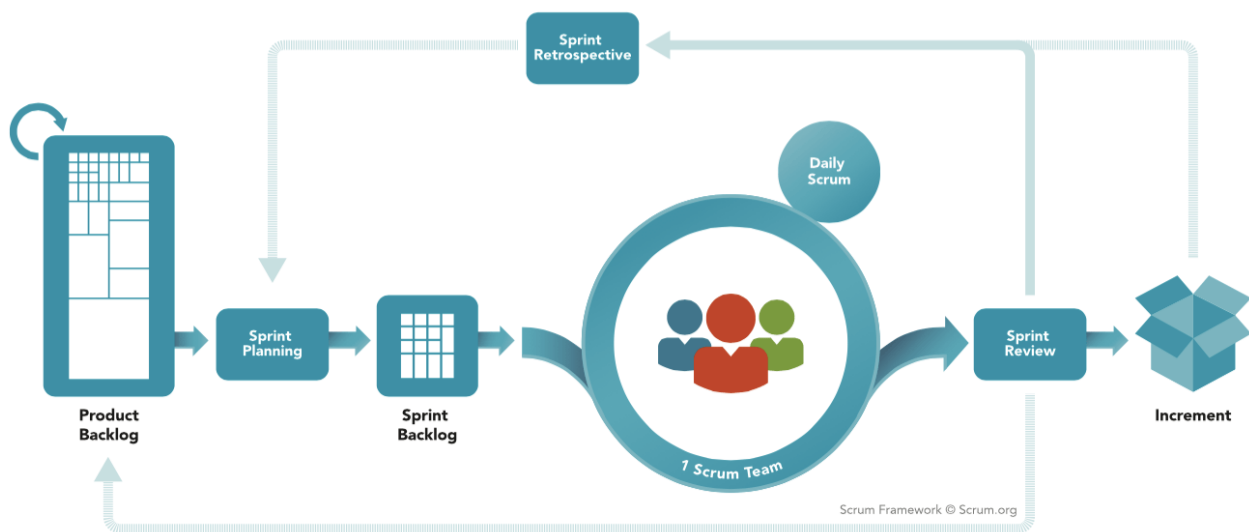


Figura 3 - Diagrama de funcionamento do Scrum

Fonte [9]

A escolha do Scrum para a utilização como metodologia de planeamento e desenvolvimento resultou das incógnitas iniciais associadas ao rumo de alguns módulos da aplicação e da pouca experiência inicial com algumas das tecnologias utilizadas. A realização de reuniões semanais com o orientador é também um facto que suporta o uso deste modelo, visto que simula as revisões e o curto período de tempo de um Sprint. Nestas reuniões pretende-se apresentar e avaliar o que foi feito e planear o que será feito na próxima semana.

4 Análise de Requisitos

Os requisitos de um sistema são um conjunto de informação sobre o que se espera que a aplicação faça e permita fazer. Foi feita uma análise tendo em conta os objetivos inicialmente traçados com suporte num conjunto de diagramas recorrendo à linguagem UML, com a qual será planeada a implementação de casos de uso.

Considerando a primeira interação do utilizador com a aplicação após realizar o *login*, o caminho seguinte será deslocar-se até à página da parte prática onde é feita uma listagem das categorias de problemas disponíveis. As categorias são um meio para se melhor organizar a listagem dos problemas e no fundo identificar sobre de que se tratam os problemas de uma forma genérica. Cada problema é um desafio algorítmico e lógico para o utilizador resolver. Estes problemas estão por norma associados à área da programação em especial algoritmos e estruturas de dados, mas qualquer problema que possa ser testado recorrendo a *inputs* e *outputs* de texto poderá ser disponibilizado. Este conjunto de *inputs* e *outputs* são os casos de teste de cada problema, ou seja, cada par *input-output* terá uma pontuação associada, e é a partir dum conjunto de vários pares *input-output* que a solução (código) submetida pelo utilizador é avaliada. Os utilizadores deverão ter a possibilidade de participar em torneios, sejam eles públicos, torneios que se encontram disponibilizados para todos, ou privados, torneios aos quais só é possível fazer registo por um código privado.

De acordo com a performance do utilizador nas suas submissões, é inserido numa série de listas classificatórias: classificação global, diz respeito à classificação de todos os utilizadores na aplicação, classificação por torneio, diz respeito à classificação dos utilizadores participantes do torneio e classificação de problema, que diz respeito à classificação para cada problema individual.

Tendo em conta as considerações anteriores, o sistema deverá possibilitar:

- Consultar categorias de problemas práticos
- Consultar lista de problemas de uma categoria

- Submeter solução(código) para o problema
- Consultar classificação global
- Consultar classificação de torneio
- Consultar submissões para determinado problema
- Consultar torneios
- Participar em torneios
- Adicionar, editar, eliminar torneio privado
- Adicionar, editar, eliminar problemas de um torneio
- Adicionar, editar, eliminar casos de teste de um problema
- Gerir categorias
- Gerir problemas públicos
- Gerir casos de teste associados a problemas públicos

4.1 Diagrama de Contexto

O diagrama de contexto é um diagrama de fluxo de dados que representa todo o sistema como um único processo. Este diagrama permite representar o objeto de estudo, o projeto e a sua relação com o ambiente. É apresentado de seguida, Figura 4, o diagrama de contexto referente à aplicação.



Figura 4 - Diagrama de Contexto

4.2 Atores e respetivos casos de uso

A Tabela 2 define os atores intervenientes na plataforma e os respetivos casos de uso.

Atores	Casos de Uso
Programador	Consultar categorias de problemas práticos Consultar lista de problemas de uma categoria Submeter solução(código) para o problema Consultar classificação global Consultar classificação de torneio Consultar submissões para determinado problema Consultar torneios Participar em torneios Adicionar, editar, eliminar torneio privado Adicionar, editar, eliminar problemas de um torneio Adicionar, editar, eliminar casos de teste de um problema
Gestor de Conteúdos	Todas as funcionalidades do utilizador “Programador” Gerir categorias Gerir problemas públicos Gerir casos de teste associados a problemas públicos

Tabela 2 - Atores e respetivos casos de uso

4.3 Diagrama de casos de uso

Encontrando-se já identificados os casos de uso dos respectivos atores, segue-se agora a representação dessa informação num diagrama, ilustrado na Figura 5.

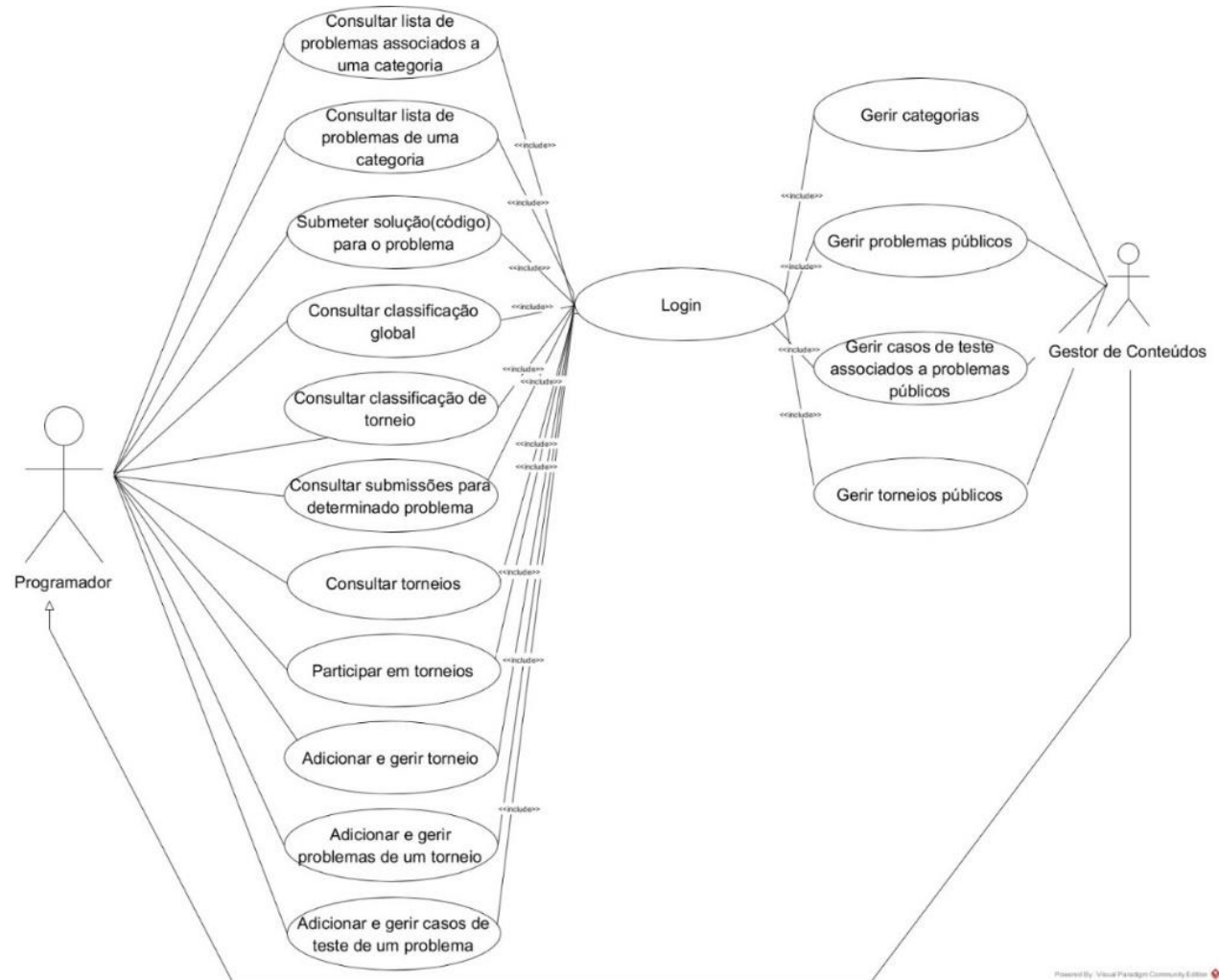


Figura 5- Diagrama de Casos de Uso

4.4 Descrição dos casos de uso e Diagramas de Sequência

A descrição de um caso de uso é essencial para se obter uma visão mais aprofundada de como o utilizador fará a interação com a aplicação e assim planejar a interface e a lógica de negócio, mas também verificar onde poderão estar os pontos de falha associados a uso indevido [10].

Esta descrição será feita segundo uma estrutura predefinida na qual se explora os seguintes tópicos:

- **Descrição:** Descrição curta e sucinta do caso de uso em questão. Deverá ser perceptível o que se pretende numa curta frase;
- **Pré-Condição:** Condição inicial necessária para que o caso de uso decorra com sucesso;
- **Caminho Principal:** Descrição de como o utilizador deve proceder para que tudo decorra com sucesso
- **Caminhos Alternativos:** Descrição do que poderá correr mal em determinado passo do caminho principal. Esta é uma parte essencial, uma vez que os desenvolvedores devem estar conscientes do que poderá levar a falhas no sistema.

4.4.1 Submeter solução para um problema

Este caso de uso pode ser despertado por duas situações distintas: a resolução de problemas na parte prática ou a resolução de problemas em contexto de competição. A Tabela 3 permite ver em maior detalhe o desenvolvimento do caso de uso.

Nome	Submeter solução para um problema
Descrição	Este caso de uso tem como objetivo descrever o processo de submissão de solução para um problema.
Pré-Condição	<i>Login</i> válido
Caminho Principal	<ol style="list-style-type: none"> 1. O ator clica no problema que pretende resolver 2. O sistema mostra uma página com o problema e as suas características (dificuldade, score máximo, criador do problema, data de criação) 3. O ator introduz o seu código e submete-o. 4. O sistema compila, executa e avalia o código e mostra ao utilizador uma página com os seus resultados.
Caminhos Alternativo	<p>2.a) Se o problema já não existir, o ator é redirecionado para uma página genérica de página não encontrada.</p> <p>4.a) Caso haja uma falha na conexão, tenta-se a reconexão periodicamente.</p> <p>4.b) Qualquer outra falha neste processo é informada ao ator com uma mensagem genérica e de que este deverá tentar novamente.</p>

Tabela 3 - Descrição de Caso de Uso “Submeter solução para um problema”

Na Figura 6 apresenta-se o diagrama de sequência relativo ao caminho principal do caso de uso “Submeter solução para um problema”

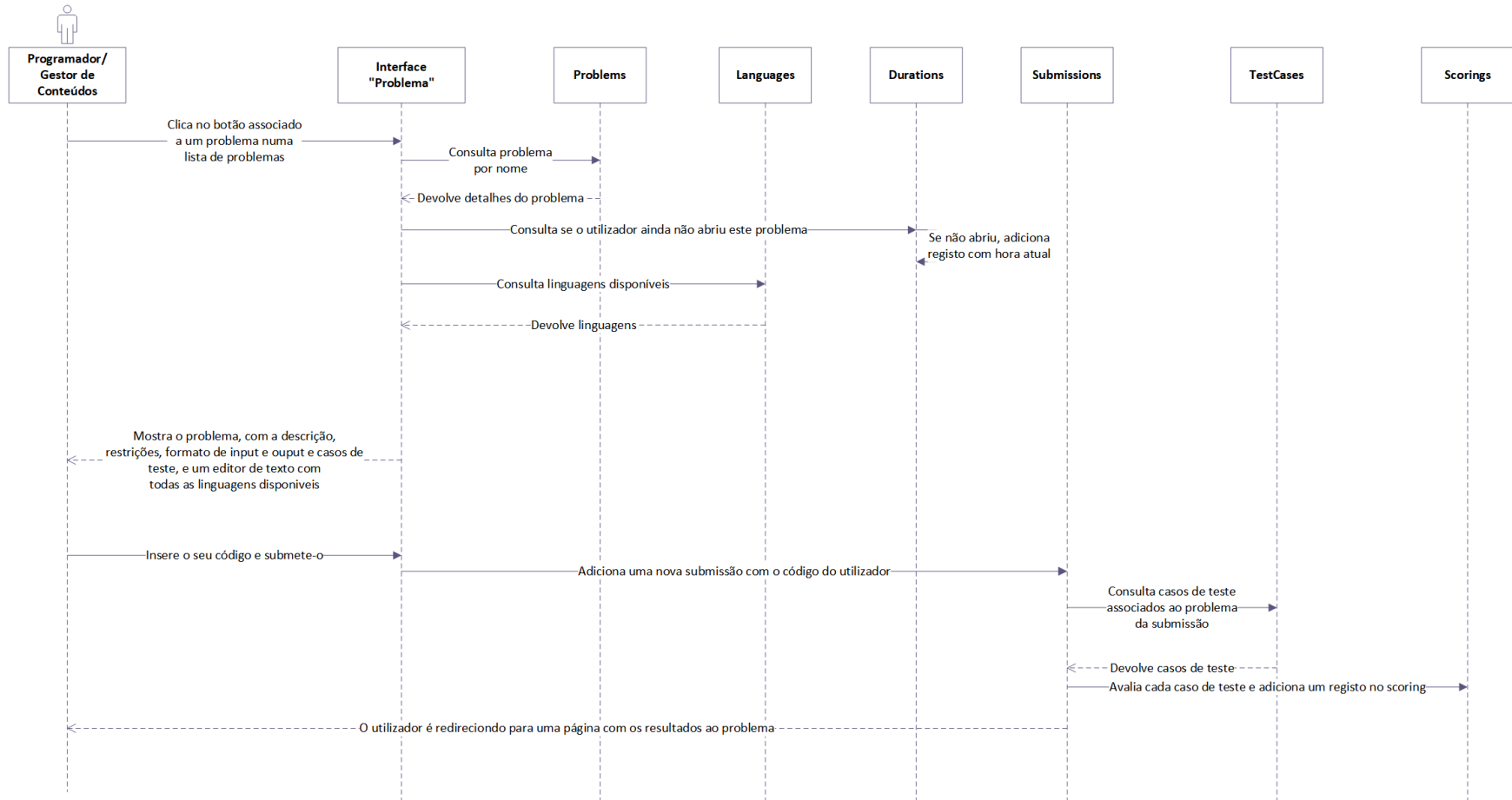


Figura 6 - Diagrama de Sequência "Submeter solução para um problema"

4.4.2 Criação de problema público

A descrição do caso de uso que se segue, Tabela 4, representa a criação de um problema público por parte do Gestor de Conteúdos. Os problemas públicos são os que se encontram inseridos numa categoria e como tal estão visíveis para qualquer utilizador que consulte a área de Prática do *site*.

Nome	Criação de problema público
Descrição	Este caso de uso tem como objetivo descrever o processo de criação de um problema público
Pré-Condição	<i>Login</i> válido
Caminho Principal	<ol style="list-style-type: none">1. O ator clica em “Adicionar novo problema”2. O sistema mostra uma página para preenchimento (Nome, Categoria, Dificuldade, Descrição, Restrições, Formato de Input e Output e Score máximo).3. O ator preenche os diversos campos e faz a submissão4. O sistema guarda o problema e redireciona o ator para a zona de gestão de problemas
Caminhos Alternativo	<ol style="list-style-type: none">3.a) O sistema alerta o ator que não preencheu todos os campos3.b) O sistema alerta o ator que o nome de problema já existe

Tabela 4 - Descrição de Caso de Uso "Criação de Problema Público"

Na Figura 7 encontra-se ilustrado o diagrama de sequência relativo à “Criação de Problema”

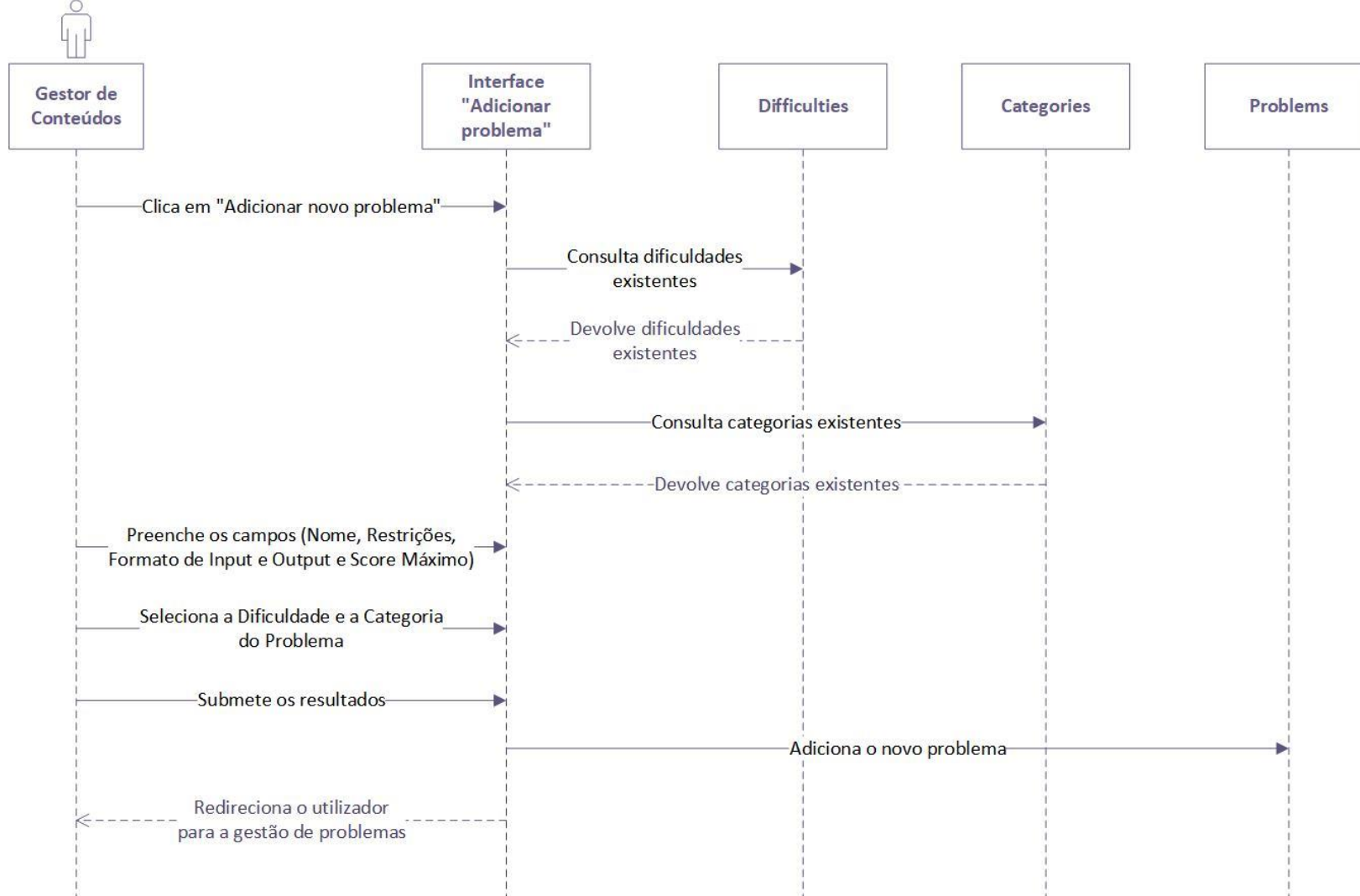


Figura 7 - Diagrama de Sequência "Criação de Problema Público"

4.5 Diagrama de Classes

Neste subcapítulo é apresentado o diagrama de classes relativo à aplicação *web* desenvolvida. O principal objetivo deste diagrama é demonstrar a relação entre as diversas classes que compõem o sistema. Para cada classe é apresentado o seu nome, atributos (dados que se pretende armazenar) e os principais métodos lógicos que o compõem.

A classe *Problems* é uma das principais classes uma vez que interage com uma grande quantidade de outras classes, como por exemplo a classe *Users*, para fazer o armazenamento do criador do problema, a classe *Submissions*, que permite indicar a que problema diz respeito determinada submissão, a classe *TestCases*, que permite associar os casos de teste a um determinado problema, a classe *Tournaments*, que permite registar os problemas de cada torneio, entre outras. Além disso, os atributos desta classe são os que terão mais consultas visto que uma parte considerável da aplicação é passada na resolução de problemas, tanto em parte prática como em parte competitiva.

A classe *Submissions* é também crucial na aplicação uma vez que é aqui que está associada a maior parte da lógica de compilação, execução e avaliação das submissões realizadas pelos utilizadores. É também utilizada para revelar ao utilizador o seu histórico de participações na plataforma, quer pela página de perfil, quer pelo resumo de submissões associados a cada problema.

Sendo a parte competitiva um dos pontos essenciais da aplicação, a classe *Tournaments* permite a gestão de dados associados aos torneios. Permite, além disso, gerir a diferença entre torneios públicos e privados. A sua ligação com a classe *Ratings*, e posteriormente a classe *Users*, permite manter um registo dos utilizadores registados no torneio e fazer o cálculo do seu desempenho.

É possível observar o diagrama de classes na página que se segue na Figura 8.

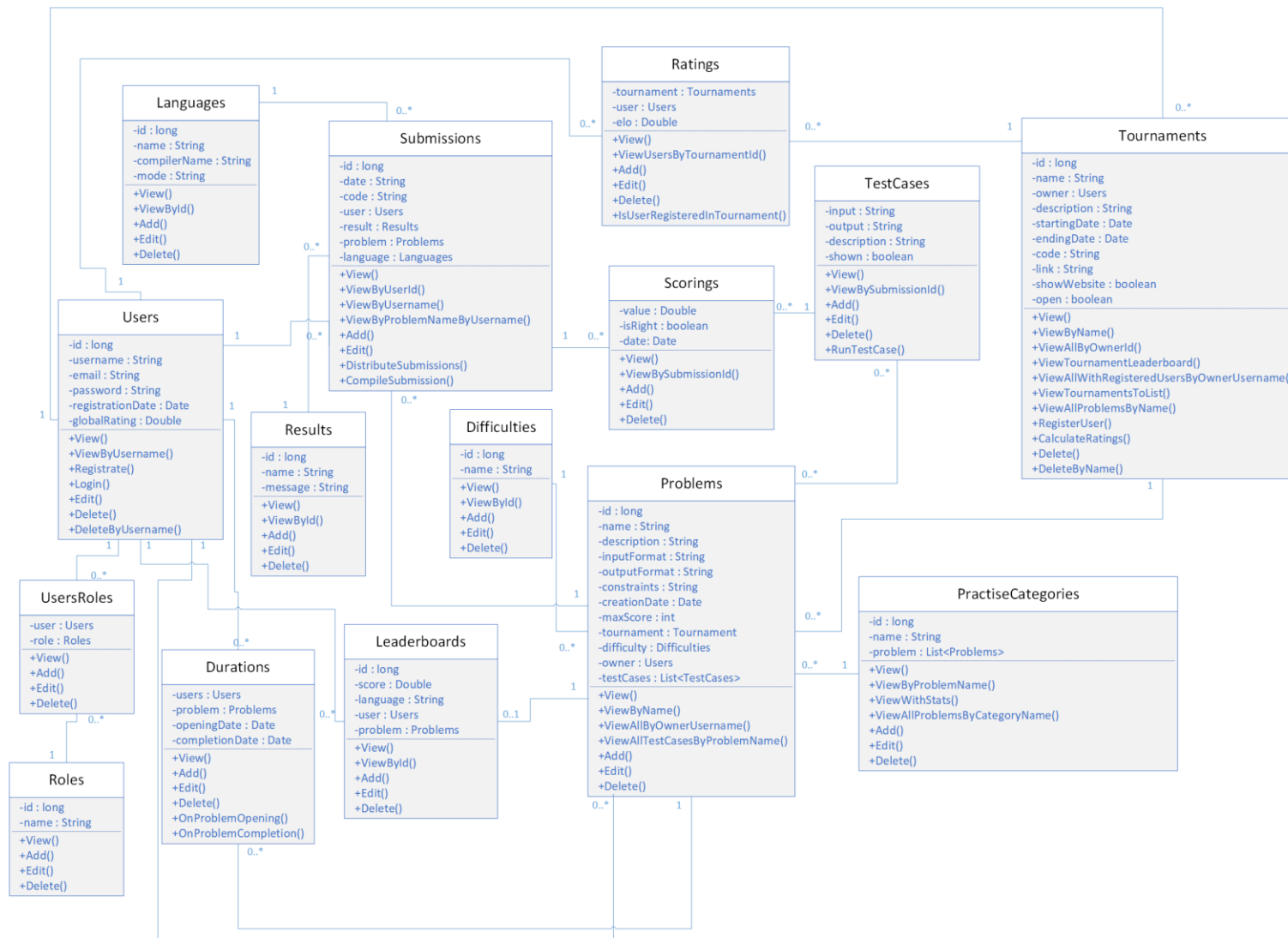


Figura 8- Diagrama de Classes

Segue-se o modelo ER obtido através do mapeamento de objetos realizado pelo *Hibernate* (*Framework* que realiza o mapeamento de objetos para uma estrutura relacional).

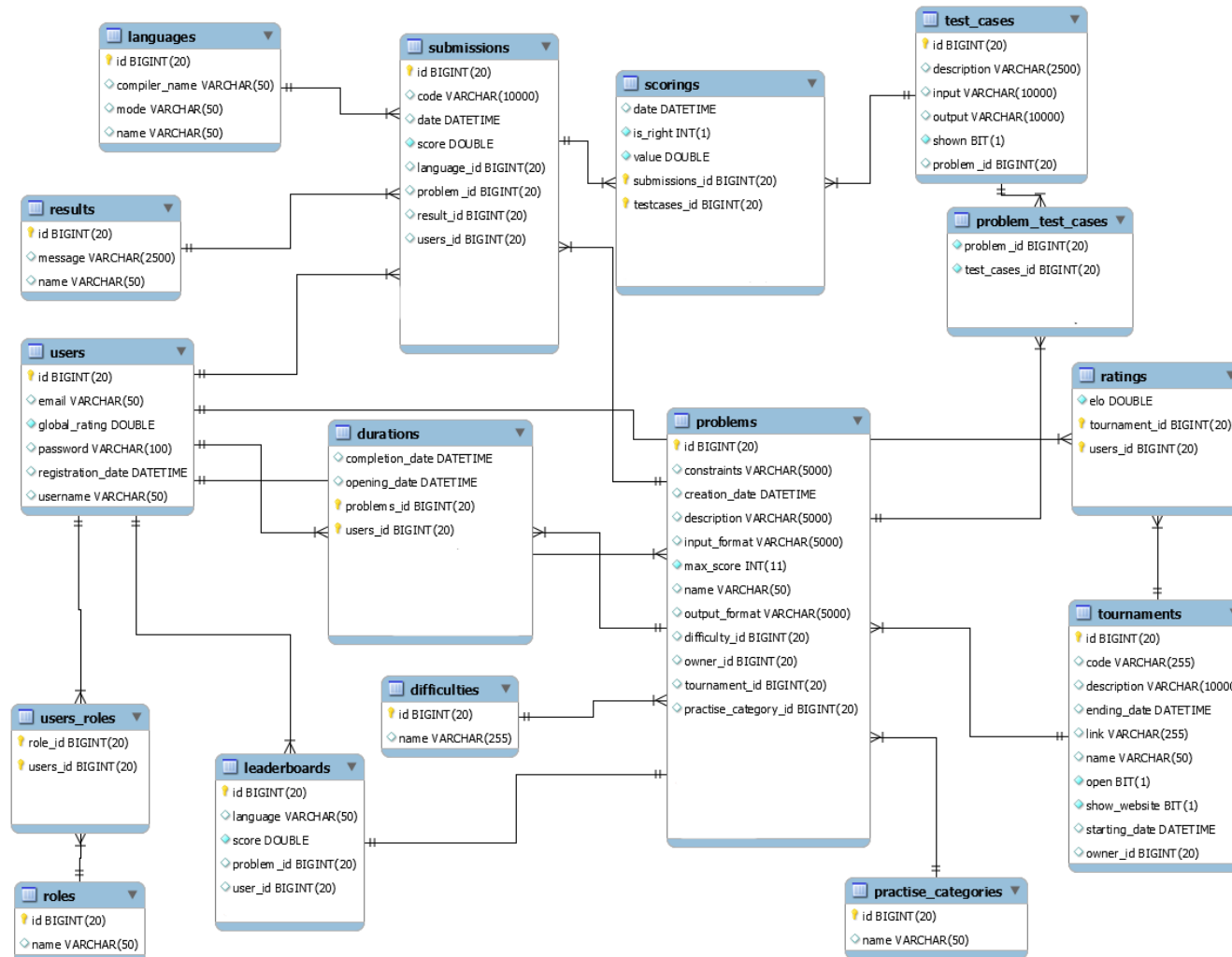


Figura 9 - Modelo ER

5 Implementação da Solução

Feito o estudo de algumas das soluções existentes no mercado e a análise do rumo que se pretende dar à aplicação, através da recolha e planeamento de requisitos, segue-se o desenvolvimento da aplicação *web*.

Como tal, foi escolhido um grupo de tecnologias que permita o desenvolvimento da forma mais adequada e eficiente possível.

5.1 Tecnologias Utilizadas

5.1.1 Java EE

A linguagem *Java* encontra-se no topo como uma das linguagens mais utilizadas das últimas décadas [11]. O facto da *Java Virtual Machine* ser multiplataforma concede uma enorme versatilidade a esta linguagem sendo usada para desenvolvimento em dispositivos móveis, aplicações *desktop*, *backend* de aplicações *web*, robótica, entre outros.

Para o desenvolvimento de *backend* de aplicações *web* recorre-se à versão *Enterprise Edition* desta linguagem, que fornece um conjunto de APIs para gestão de pedidos *HTTP*, *WebSockets*, *web services RESTful*, processamento de *JSON*, persistência de dados, etc.

A curiosidade de explorar a parte *enterprise* desta linguagem teve um grande peso na escolha relativamente ao *nodejs*¹⁴, em especial por ser uma solução bastante sólida e por se encontrar em utilização na indústria desde o final dos anos 90 [12].

Na tentativa de tornar o processo de desenvolvimento mais eficiente, optou-se pelo uso de várias *frameworks*.

¹⁴ Interpretador de código *JavaScript* que permite a sua execução fora do *browser*

5.1.1.1 Spring

O uso desta *framework* tem vindo a crescer nos últimos tempos de forma exponencial, encontrando-se já no topo de popularidade [13] de *frameworks Java* mais utilizadas. É constituída por uma grande variedade de módulos que variam desde *cloud* a *mobile*. No desenvolvimento do projeto foram utilizados os seguintes:

- Spring Boot: Módulo que facilita a criação e toda a configuração inicial e que como tal permite ter um projeto inicial pronto em poucos minutos.
- Spring Data JPA: Módulo que recorre à *Java Persistence API* para facilitar a criação de modelos de dados.
- Spring Security: Módulo que facilita a implementação de autenticação na aplicação.

5.1.1.2 Hibernate

Esta *framework* tem como principal função mapear um conjunto de objetos em lógica relacional de base de dados, removendo assim a necessidade de haver a criação manual das tabelas e a lógica que ligará essas tabelas aos objetos. Conta também com uma linguagem própria para fazer *queries* orientadas a objetos (*Hibernate Query Language*), no entanto o suporte a *SQL* também está presente.

5.1.2 JavaScript

JavaScript é uma linguagem de programação interpretada (não necessita de ser compilada) que conta com características do paradigma orientado a objetos [14]. É conhecida sobretudo por ser utilizada num ambiente de páginas *web* na qual permite uma interação com

o *DOM* e assim uma maior interatividade entre o utilizador e a página, no entanto esta não se restringe ao ambiente do *browser* e é usada em especial pelo *nodejs* e *electron* [15].

As principais bibliotecas *JavaScript* utilizadas nesta aplicação são: jQuery, ReactJS e MathJax.

5.1.3 ReactJS

React é uma framework baseada em *JavaScript* criada pelo *Facebook*. Esta *framework* possui uma versão para desenvolvimento *mobile*, denominada de *React Native* e também a sua versão *frontend* para criação de páginas *web* interativas, denominada de ReactJS.

Um dos principais pontos fortes desta *framework* passa pela modularidade com que as interfaces são desenvolvidas recorrendo ao uso de componentes. Esta modularidade permite que recorrendo a uma *Virtual DOM* apenas os componentes que são alterados são renderizados de novo, algo que não acontece com o típico *DOM*, onde cada mudança exige uma renderização completa da página o que implica um maior uso de recursos e mais tempo de processamento. Esta diferença pode ser observada na Figura 10.

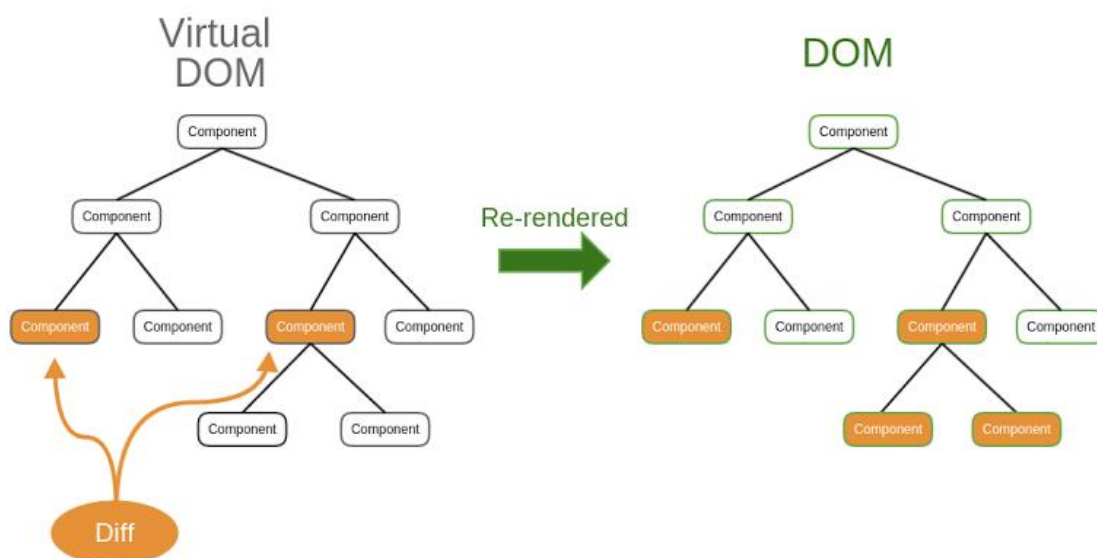


Figura 10 - React Virtual DOM vs Typical DOM

5.1.4 MySQL

O MySQL é um sistema de gestão de bases de dados relacionais *open-source* baseado em SQL. A sua popularidade resulta sobretudo de estar presente nas famosas *stacks WAMP/LAMP*. Este sistema de gestão de BD foi escolhido essencialmente por ser *open-source*, fácil de instalar e possuí todas as características necessárias.

5.1.5 HTML

Acrónimo para *Hypertext Markup Language*, é a linguagem de marcação padrão para páginas *web*. É com esta linguagem que se criam os elementos fundamentais de uma página *web* recorrendo a *tags*, que são depois interpretadas por um *web browser* e exibidos ao utilizador.

5.1.6 CSS

CSS é o mecanismo que permite ao programador adicionar estilos e animações aos elementos criados em linguagens de marcação, na maioria dos casos *HTML*. Por norma esta aplicação de estilos e animações é feita num ficheiro à parte, o que permite uma melhor organização, sendo, no entanto, possível colocar estilos juntamente com o *HTML*. O seu nome surge da prioridade de como são aplicadas as formatações.

Para que a aplicação da formatação seja possível é necessário saber em primeiro lugar qual/quais os elementos a que se pretende aplicar a formatação. De forma a resolver este problema são usados seletores que fazem a seleção dos elementos e depois aplicam a formatação indicada aos mesmos.

Com base em alguma experiência passada, optou-se pela utilização da *framework bootstrap* [16] visto que esta é composta por uma enorme variedade de classes predefinidas que foram desenhadas e otimizadas para uma experiência responsiva e amigável para um grande leque de resoluções e *browsers*.

5.1.7 Git

Git é um sistema de gestão de versões grátis e *open-source*. Um sistema de gestão de versões permite a criação de diferentes versões ao longo do desenvolvimento do projeto através de *commits*¹⁵, versões essas que podem ser consultadas a qualquer altura. Este tipo de *software* também fornece funcionalidades que facilitam o desenvolvimento em equipas de trabalho pela gestão de conflitos e a possibilidade de criação de *branches*¹⁶.

Para este projeto recorreu-se à plataforma GitHub onde foi armazenado todo o código e registada a evolução do projeto. Consultável em <http://github.com/miguelfbrito/codeflex>

5.1.8 Compiladores

Tendo em vista a avaliação das soluções submetidas pelos utilizadores, estas têm de ser compiladas e executadas de acordo com um compilador específico à linguagem. A lista de compiladores disponíveis pode ser observada pela Tabela 5.

LINGUAGEM	COMPILADOR
JAVA	Java 8
C#	Mono 4.2.1
C++	GCC 5.4.0
PYTHON	Python 2.7.12

Tabela 5 - Linguagem e seu compilador

¹⁵ Ato de adicionar alterações ao repositório do projeto

¹⁶ Ramos de desenvolvimento em paralelo

5.2 Arquitetura do Sistema

A arquitetura de um sistema é o modelo conceptual como a estrutura, o comportamento e todas as interações entre os diversos componentes ocorrem [17]. A definição da arquitetura de um sistema deve ser um passo tomado com consciência de todos os elementos envolventes no projeto e em possíveis elementos a serem adicionados futuramente.

No presente projeto optou-se pelo uso de uma arquitetura de *web services*, que é uma arquitetura bastante flexível e modular visto que cria um nível de abstração entre o que se pretende expor (dados, lógica de negócio) e o que se pretende consumir. Esta exposição-consumo de dados é feita através de uma linguagem universal, sendo as mais usadas *JSON* e *XML*. Como tal podem existir servidores a correrem aplicações em *Java* a comunicar com um servidor que corre em *JavaScript*. Outro dos principais pontos que levou à escolha desta arquitetura é a possibilidade de fazer uma separação *frontend-backend* que permite uma melhor organização do projeto, melhor controlo tanto da parte de negócio como da parte da interface do utilizador e utilizar a mesma lógica de negócio para consumir dados para diferentes aplicações, quer sejam *web*, *desktop*, *mobile*, *Smart Tv*, etc.

Utilizando uma arquitetura *web services* é necessário ter em consideração a segurança do mesmo, caso contrário qualquer utilizador pode fazer pedidos e obter informação à qual não deveria ter acesso. Para tal, adicionou-se um sistema de *tokens*¹⁷, recorrendo ao JWT (JSON Web Tokens) [18] para gerir quem tem acesso ao quê. Este sistema, permite transmitir informação num *token*, acompanhada de uma chave de verificação. Sempre que um pedido é feito ao sistema e o *token* não condiz com os *tokens* gerados, o acesso é negado.

¹⁷ Conjunto de caracteres com um significado coletivo

A arquitetura do sistema implementado é ilustrada na Figura 11.

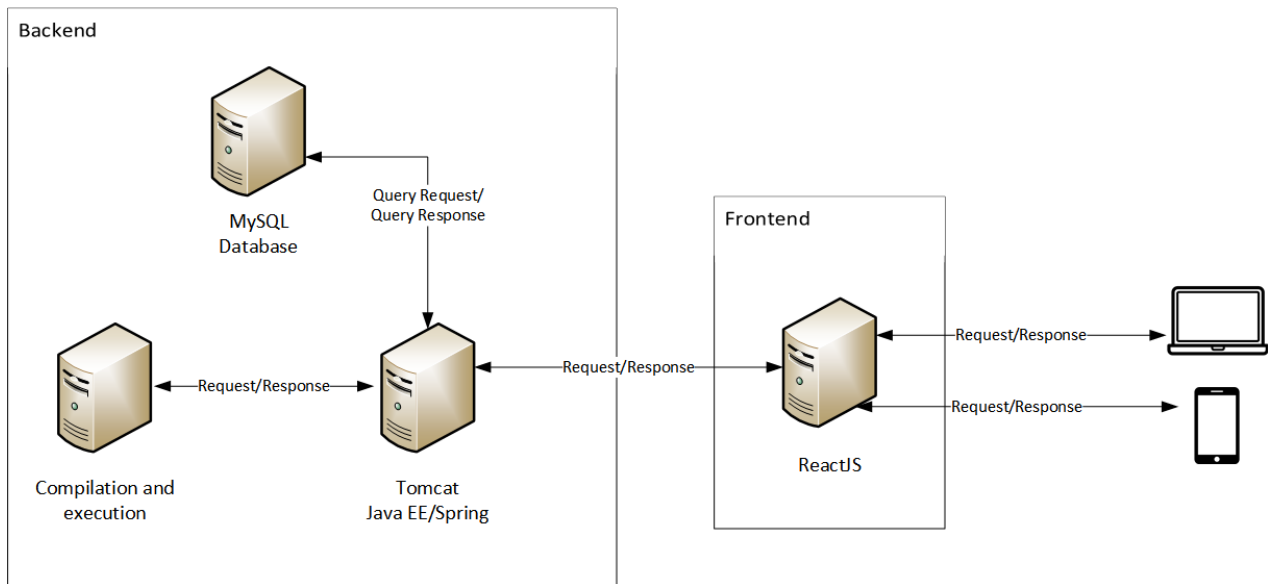


Figura 11- Arquitetura do sistema

5.3 Compilação, Execução e Avaliação de Submissões

Um dos objetivos do presente projeto passa por avaliar o código submetido pelos utilizadores para determinado problema. Numa fase inicial optou-se por fazer uma pesquisa dos serviços disponíveis numa tentativa de incorporar uma API já capaz de realizar todo o processo de compilação, execução e avaliação de forma eficiente e consistente.

Realizaram-se abordagens a dois serviços distintos com o objetivo de obter informação sobre os custos do serviço e se haveria algum tipo de programa académico disponível.

O primeiro serviço, *SphereEngine*, um serviço já analisado anteriormente, informou que só fornecem “pacotes” de serviço a partir das 10000 submissões e visto que se trata de um projeto académico estariam dispostos a proporcionar submissões grátis, desde que fosse usado um *widget* seu. Apesar desta proposta parecer positiva, o uso do *widget* deste serviço retira por completo a possibilidade de haver um processo de avaliação uma vez que só funcionaria como um avaliador de código para *inputs* e *outputs* inseridos pelo próprio utilizador.

O segundo serviço, *HackerEarth*, já referido no capítulo 2, contém várias indicações para documentação pública de uma API de avaliação, no entanto, indicou que o serviço se encontra descontinuado e toda a API se encontra como sendo privada e apenas é usada na sua aplicação.

Assim, optou-se pela criação de um sistema capaz de avaliar as submissões e como tal possuir as seguintes capacidades:

1. Compilação de submissões

Qualquer linguagem não interpretada terá de passar por um processo de compilação antes de poder ser executada. Os erros obtidos no processo de compilação devem ser transmitidos ao utilizador.

2. Execução de casos de teste relativos a uma submissão

Cada submissão estará associada a um problema, e esse mesmo problema associada a n casos de teste.

3. Avaliação parcial das submissões

O utilizador deverá receber uma pontuação cumulativa baseada nos casos de teste válidos e não apenas uma resposta correta ou incorreta.

4. Limitações para cada execução

Cada execução será limitada para evitar que perturbe o normal funcionamento do sistema. Por exemplo, uma submissão que tenha uma complexidade algorítmica muito superior à pretendida e como tal a sua execução será muito mais prolongada deve ser terminada após um tempo limite.

Havendo outros objetivos para concretizar e uma janela de tempo reduzida, optou-se pela integração da lógica deste sistema diretamente com a lógica principal da aplicação já existente, ao invés da criação de uma API separada com o servidor onde se encontrariam todos os compiladores, o que seria a solução ideal para este problema.

De uma forma muito simples, o sistema criado, passa pela construção de vários comandos que são executados paralelamente numa máquina Linux relativamente a um *input*, através de SSH(*Secure Shell*). O *output* obtido é depois validado de acordo com o *output* definido. Este processo pode ser descrito da seguinte maneira:

1. A submissão do utilizador é adicionada a uma fila de submissões.
2. Enquanto a fila não estiver vazia
 - a. Retira-se o primeiro elemento da fila e este é compilado
 - b. Após a compilação a submissão é executada relativamente a todos os casos de teste associados ao problema
 - i. É guardado um resultado para cada execução de acordo com o seu *output* relativamente ao predefinido.

O código relativo a este processo encontra-se no anexo 9.1.2.

Exemplo de estrutura de comando de compilação de uma submissão em Java:

```
javac Solution.java > [ficheiro_erro_compilação]
```

Exemplo de estrutura de comando de execução de uma submissão em Java:

```
parallel -j `nproc` < jobs.txt
```

Sendo o ficheiro jobs.txt constituído por um conjunto de comandos relativos aos casos de teste da submissão com a seguinte estrutura:

```
firejail --private=[path] --quiet --net=none cat [testcase_output_file] | timeout  
3s java Solution 2> [error_file] > [output_file]
```

5.3.1 Segurança

Possibilitar ao utilizador que execute o código que pretender nas máquinas que correm a aplicação é de facto o sonho de qualquer atacante informático. Como tal, foram tomadas um conjunto de medidas com o objetivo de lidar com esta situação.

A mais significativa passou pelo uso de um ambiente *sandbox*¹⁸ para execução das submissões do utilizador. Este tipo de ambientes evita que código malicioso se propague para a restante máquina ficando isolado no seu ambiente. O ambiente *sandbox* foi aplicado recorrendo ao *Firejail*, um programa focado essencialmente em reduzir o risco de quebras de segurança [18].

¹⁸ Mecanismo de segurança que permite isolar o que se pretende executar do restante sistema operativo

5.3.2 Performance

Tendo em conta que a compilação e execução de programas é um processo que exigirá dos recursos de processamento da máquina, em especial para casos de teste com vários milhares de *inputs*, é importante ter em consideração o tempo de processamento das submissões uma vez que se pretende que haja um *feedback* em tempo real.

A capacidade de processamento do *hardware* em utilização será sempre um grande fator na velocidade com que o *feedback* é transmitido ao utilizador, no entanto, podem ser aplicadas algumas medidas para o uso mais eficiente de recursos, sendo a principal a execução em paralelo dos casos de teste relativos às submissões.

Para tal, foi feita uma implementação recorrendo a programação multitarefa que permitirá que o servidor continue a expor todos os métodos para o *frontend* enquanto realiza a avaliação das submissões uma vez que os métodos não são assíncronos.

Feita a implementação, foram testados os tempos de processamento para vários conjuntos de submissões tanto para a execução de forma concorrente simulando o ambiente normal de execução, como para execução de forma paralela, a estratégia que pretende melhorar a velocidade de execução. A máquina onde serão executados estes testes possui um CPU i5-8250u de 4 núcleos de processamento e 3.4GHz de frequência máxima, e um total de 3GB de memória RAM (*Random Access Memory*).

No seguinte conjunto de testes, Tabela 6, a execução no servidor de compilação e execução é realizada de forma concorrente.

<i>Nº Submissões</i>	<i>Nº de casos de teste totais</i>	<i>Duração (segundos)</i>
5	50	26
10	100	53

20	200	68
45	450	135

Tabela 6 - Testes de performance - Concorrente

No conjunto de testes seguinte, Tabela 7, a execução no servidor de compilação e execução é realizada de forma paralela.

<i>Nº Submissões</i>	<i>Nº de casos de teste totais</i>	<i>Duração (segundos)</i>
5	50	15
10	100	32
20	200	45
45	450	82

Tabela 7 - Testes de Performance - Paralela

Comparando agora os testes pelo gráfico da Figura 12, é possível observar que de facto existem vantagens a nível de velocidade numa solução relativamente à outra. Existe sempre um tempo de “arranque” para a adição das submissões que poderá afetar ligeiramente os resultados, em especial para testes de curta duração, no entanto para testes de maior quantidade de submissões a diferença torna-se mais acentuada.

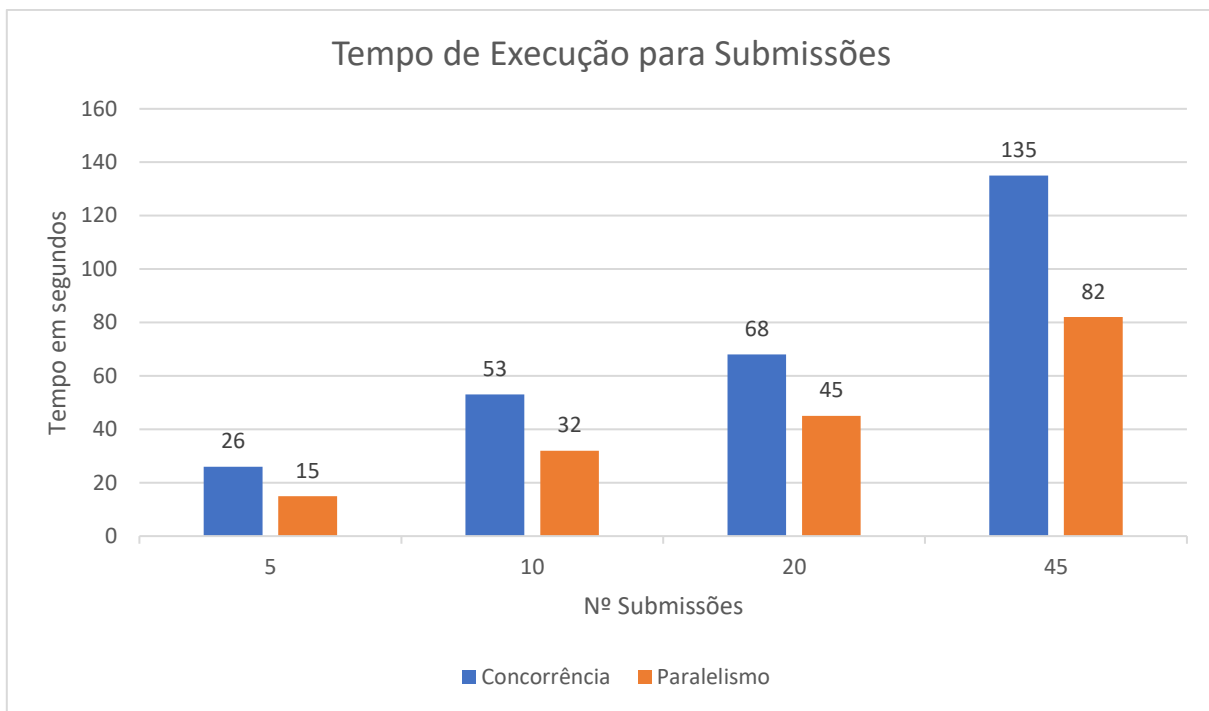


Figura 12 - Comparação dos resultados dos tempos de execução

O método de execução em paralelo apresenta melhorias de tempo de execução na ordem dos 50% a 70% (58%, 60%, 67%, e 60%) relativamente ao método de execução concorrente para os testes realizados. Sendo esta execução feita num processador com 4 núcleos, esperava-se que o método em paralelo obtivesse tempos de execução perto de 4 vezes inferiores, no entanto, o tempo de envio do código e dos casos de teste para o servidor de compilação é significativo visto que todas as execuções se realizam de forma relativamente rápida. Por outro lado, para execuções em que ocorre o pior caso possível¹⁹ o tempo utilizado a transferir o código e os casos de teste para o servidor será menos significativo relativamente ao tempo de execução. Nesse caso, as melhorias no tempo de execução serão mais visíveis e aproximar-se-ão do esperado.

¹⁹ A execução prolonga-se pelo tempo máximo atribuído, sendo este um valor arbitrário para cada uma das linguagens.

5.4 Classificação Elo

O sistema de classificação Elo [19], assim apelidado pelo seu autor Arpad Elo, é um método estatístico de cálculo da habilidade de um jogador relativamente a outro. A cada jogador é associado um valor positivo, chamado de *Elo*. No final de cada partida o vencedor retira pontos ao vencido. A quantidade de pontos perdidos ou ganhos dependerá da diferença entre os *elos* dos dois jogadores. A popularidade deste sistema surgiu principalmente pelo seu uso como sistema de classificação em campeonatos de xadrez.

Segue-se um exemplo retirado de [19], que pretende demonstrar o funcionamento deste sistema.

Considerando o jogador A de *rating* 1613 que participa num torneio e defronta 5 jogadores com os resultados da Tabela 8. O *rating* apresentado na tabela é o *rating* inicial de cada jogador.

<i>Adversário</i>	<i>Rating</i>	<i>Resultado</i>
1	1720	Derrota
2	1388	Vitória
3	1586	Vitória
4	1477	Empate
5	1609	Derrota

Tabela 8- Confronto com adversários

O rating atualizado do jogador A será calculado pela sua prestação relativamente a cada jogador, se ganhou, perdeu ou empatou e pela diferença entre os *elos* de cada jogador. Inicialmente calcula-se a pontuação esperada do jogador A relativamente ao adversário 1, que será identificado como B1, usando a seguinte fórmula:

$$E_A = \frac{1}{1 + 10^{(R_{B1} - R_A)/400}} = \frac{1}{1 + 10^{(1720 - 1613)/400}} = 0,354$$

É realizado o mesmo processo para os restantes 4 adversários:

Adversário 2, de *rating* 1388.

$$E_A = \frac{1}{1 + 10^{(R_{B2} - R_A)/400}} = \frac{1}{1 + 10^{(1388 - 1613)/400}} = 0,785$$

Adversário 3, de *rating* 1586

$$E_A = \frac{1}{1 + 10^{(R_{B3} - R_A)/400}} = \frac{1}{1 + 10^{(1586 - 1613)/400}} = 0,539$$

Adversário 4, de *rating* 1477

$$E_A = \frac{1}{1 + 10^{(R_{B4} - R_A)/400}} = \frac{1}{1 + 10^{(1477 - 1613)/400}} = 0,686$$

Adversário 5, de *rating* 1609

$$E_A = \frac{1}{1 + 10^{(R_{B5} - R_A)/400}} = \frac{1}{1 + 10^{(1609 - 1613)/400}} = 0,506$$

Após calculado os valores esperados relativamente a este jogador pode ser feita a atualização do seu *rating* recorrendo à seguinte equação:

$$R'_A = R_A + K(S_A - E_A)$$

Sendo K uma constante de valor 32, e S_A a soma dos pontos de acordo com o resultado contra cada adversário. A vitória conta como 1 ponto, o empate 0.5 e a derrota nenhum.

$$S_A = 0 + 1 + 1 + 0.5 + 0 = 2.5$$

O E_A representa o somatório de todos os resultados esperados (E_A) contra os 5 adversários.

Aplicando a fórmula obtêm-se o novo *rating* do utilizador após a participação no torneio.

$$R'_A = 1613 + 32(2.5 - 2.87) = 1601$$

Para atualizar o *rating* dos restantes adversários o mesmo conjunto de cálculos terá de ser realizado.

A implementação feita para o cálculo do *rating* na aplicação é muito semelhante a este exemplo. Existe uma tarefa no *backend* que verifica a cada segundo o estado dos torneios, se existirem torneios para os quais a data final seja superior a data atual este fecha o torneio a nível de lógica de *backend* e realiza o cálculo recorrendo ao código presente no Anexo 9.1.1.

A classe responsável por fazer a aplicação das fórmulas associados ao *rating* é representada pela Figura 13.

```

public class RatingCalculator {

    // set constant used for calculation. Impacts the rating variability
    public static final int K = 32;
    // Equation to calculate expected value
    public static double expectedRating(double ratingA, double ratingB) {
        return 1 / (1 + Math.pow(10, (ratingB - ratingA) / 400));
    }

    // Return points based on which player has the highest score
    public static double pointsComparasion(double ratingA, double ratingB) {
        if (ratingA > ratingB) {
            return 1;
        } else if (ratingA < ratingB) {
            return 0;
        }
        return 0.5;
    }
}

```

Figura 13 - Classe de Cálculo do Rating

Este sistema de classificação apresenta a sua fraqueza quando a quantidade de utilizadores é muito pequena ou os próprios jogadores apresentam performances muito inconsistentes, no entanto assim que exista uma base sólida de utilizadores é um sistema de classificação que permite obter mais informação relativamente a um sistema que apenas realize a média de pontuação dos participantes.

5.5 Interfaces da aplicação *web*

Foi logo à partida tomada a decisão de separar a parte prática e competitiva da aplicação para se tornar um ambiente mais organizado e mais simpático para novos utilizadores. É de esperar que um utilizador completamente inexperiente em programação competitiva não queira embarcar num torneio sem qualquer conhecimento, e por isso esta divisão vai-lhe permitir explorar e aprender, sem a ansiedade criada pela limitação de tempo.

A nível de *design* optou-se por um aspeto simples, mas apelativo, de combinação de cores, que envolva o mínimo de imagens possível visto que por norma as imagens são um dos principais fatores de atraso no carregamento das páginas, mas também porque haverá várias *frameworks* em uso que são exigentes a nível de recursos. Como tal é importante ter em consideração todos os pontos que afetem o atraso no carregamento da página.

Segue-se uma apresentação de algumas das interfaces implementadas.

5.5.1 Interface inicial (*Default*)

Esta é a interface com que o utilizador se depara quando acede à aplicação, ainda sem ter realizado qualquer tipo de *login* ou registo e que se ilustra na Figura 1 .

O único objetivo desta interface é captar o interesse do utilizador para o uso da aplicação e para a programação competitiva. Referem-se de seguida os 3 pontos de destaque desta plataforma:

- **Aprendizagem:** O utilizador pode explorar um repositório de problemas que se encontra organizado por categorias e dificuldade e encontrar um desafio à sua medida.
- **Competição:** Existe uma série de torneios onde o utilizador se pode inscrever para competir com os seus amigos, colegas, etc.
- **Organizar:** Ao contrário de outras plataformas, é aqui possível organizar torneios privados que podem ser partilhados através de um código.

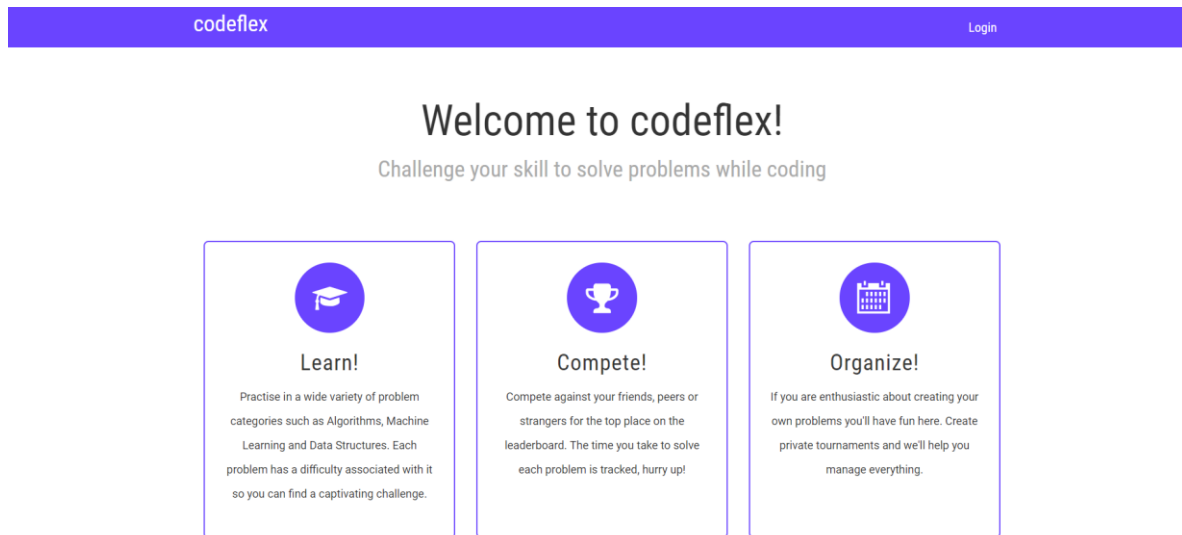


Figura 14- Interface inicial

5.5.2 Interface *Login/Registo*

Caso o utilizador se tenha sentido cativado a testar a plataforma deve-se agora dirigir à página de registo, ilustrada na Figura 15. O registo e *login* são realizados na mesma interface e para ambos é feita validação de dados.

The image shows the Codeflex login/register interface. At the top is a blue header with 'codeflex' on the left and 'Login' on the right. Below the header, the text 'Login or create your Codeflex account today!' is centered. The main content area features a white box with a blue border. Inside the box, there is a circular profile picture of a man with a beard. Below the profile picture, the text 'Account Details' is centered. There are four input fields: 'Username', 'Email', 'Password', and 'Confirm password'. At the bottom of the box, there are two green buttons: 'Login' and 'Sign Up'.

Figura 15- Interface de Login/Registo

5.5.3 Interface de Prática

As categorias e os problemas publicados nesta secção (ver Figura 16) são da inteira responsabilidade do Gestor de Conteúdos e, portanto, cabe a este tipo de utilizador gerir o que acha de bem apresentar.

O utilizador pode consultar um grupo de categorias de problemas a resolver e para cada uma delas o utilizador recebe *feedback* sobre a quantidade de problemas já resolvidos através de uma barra de progresso.

Esta secção é importante no sentido de permitir que os utilizadores se foquem na aprendizagem de determinados tópicos para desenvolver um conhecimento mais sólido acerca do mesmo.

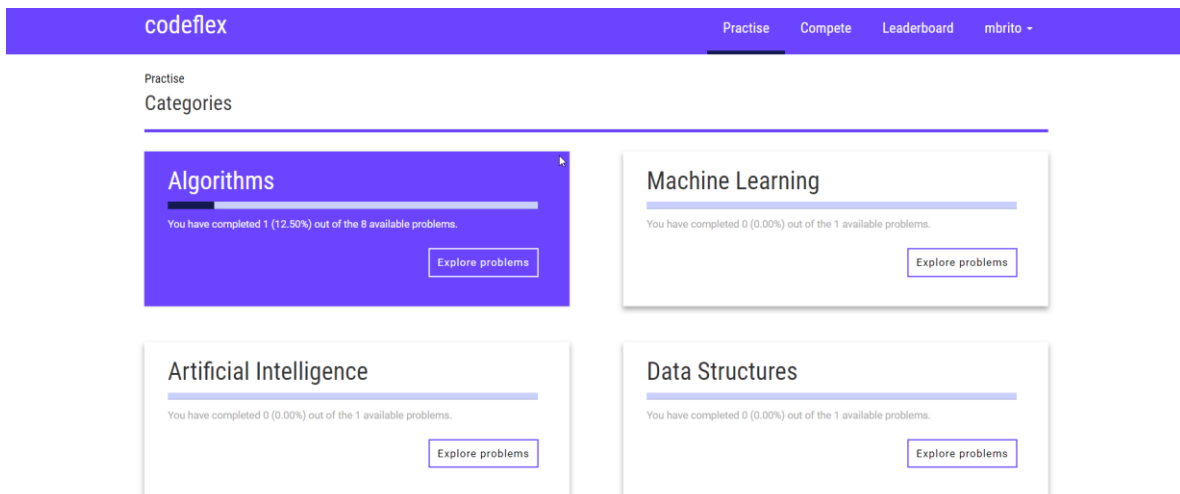


Figura 16- Página de Prática

Clicando em “*Explore Problems*”, em qualquer uma das categorias, o utilizador é redirecionado para uma secção onde são listados todos os problemas associados a essa categoria.

5.5.4 Interface de Listagem de Problemas

Nesta interface (Figura 17) são listados todos os problemas associados à categoria em questão. Os problemas são apresentados ordenados por dificuldade, mas existe um conjunto de filtros na parte lateral direita dos problemas que permitem ao utilizador filtrar a seu gosto, quer por estado de resolução ou por dificuldade.

O botão de cada problema será exposto como “*Solve again*”, caso o utilizador já tenha resolvido o problema com uma pontuação de 100%. Caso contrário mantém-se o botão “*Solve problem*” padrão.

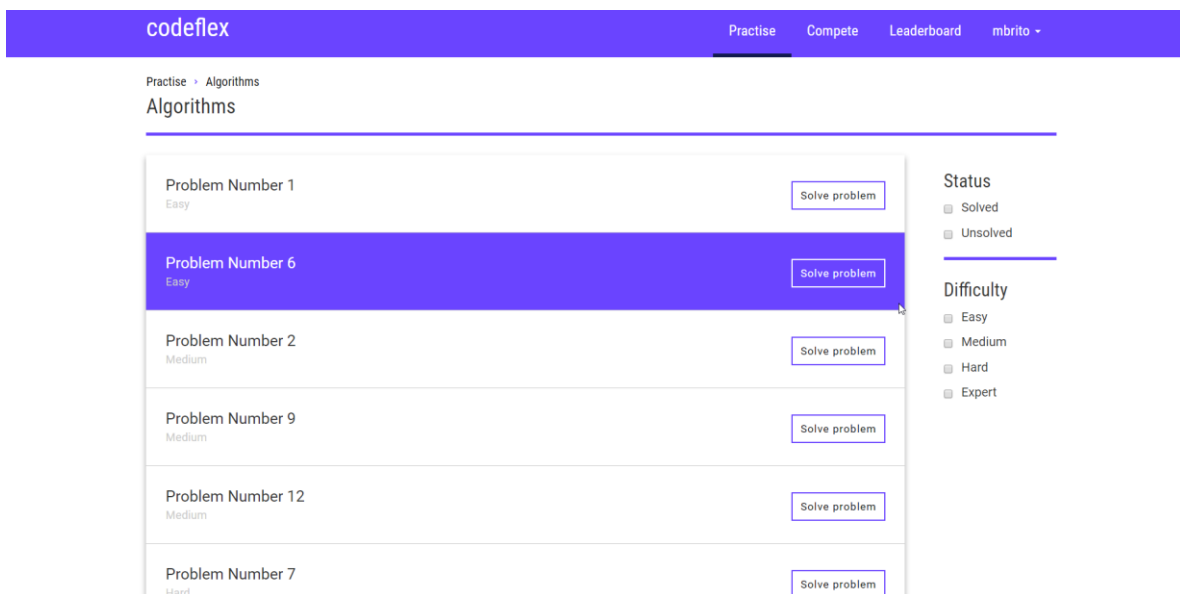


Figura 17 - Listagem de problemas

5.5.5 Interface do Problema

Esta é uma das principais interfaces da aplicação *web*. É usada tanto pela parte prática como pela parte competitiva e é aqui que é descrita a lógica do problema e todos os seus detalhes. Divide-se em três secções: a secção de detalhes, as submissões e a *leaderboard*. Na secção do problema é apresentada toda a informação inserida pelo criador do torneio (ilustrada na Figura 18 e Figura 19), nomeadamente:

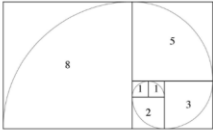
- Descrição do problema: Parte introdutória onde se pretende fazer passar de que se trata o problema e o que se pretende dos utilizadores.
- Restrições: Informação sobre as restrições do problema, normalmente tratam-se das limitações do tamanho dos *inputs* dos casos de teste.
- Formato de Input: Descrição do formato de *input* que será fornecido ao programa na avaliação.
- Formato de *Output*: Descrição do formato esperado do *output*.
- Exemplos de casos de teste: O criador do torneio pode escolher mostrar alguns dos casos de teste para auxiliar na compreensão do problema

Practise > Algorithms > Fibonacci-Sequence
Fibonacci Sequence

Problem Submissions Leaderboard

Problem Statement

Each term in the Fibonacci sequence is generated by adding the previous two terms
$$fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)$$



Create a solution that takes n numbers as an input and outputs the first n numbers of the Fibonacci sequence.

Constraints

$1 < n < 80$

Difficulty	Medium
Creator	mbrito
Date	7/6/2018
Max Score	125

Figura 18 - Interface do Problema - parte I

The screenshot shows a web interface for a programming problem. It is divided into three main sections: 'Input Format', 'Output Format', and 'Test Case 3'. The 'Input Format' section states that 'n' represents the quantity of numbers to be printed. The 'Output Format' section states that numbers should be printed separated by a space, with an example '0 1 1 2 3'. The 'Test Case 3' section contains two input fields: 'Input' with the value '5' and 'Output' with the value '0 1 1 2 3'.

Section	Text
Input Format	n representing the quantity of numbers to be printed
Output Format	The numbers should be printed separated by a space 0 1 1 2 3
Test Case 3	Input: 5 Output: 0 1 1 2 3

Figura 19 - Interface do Problema - parte2

É também apresentada uma tabela lateral onde se indica a dificuldade, o criador, com um link para a página de perfil do mesmo, a data de criação e a pontuação máxima que se pode obter.

Ainda na mesma secção, após a parte introdutória da descrição encontra-se um editor de texto embutido (ver Figura 20), disponibilizado pela *framework* ACE-Editor. Este editor de texto permitirá a qualquer utilizador que não tenha acesso a um editor de texto ou *IDE*, ou simplesmente prefira a portabilidade de usar um editor *online* escrever o seu código e submetê-lo para avaliação.

A esta *framework* foram adicionadas coleções de *highlight de syntax* de acordo com as linguagens disponíveis pela aplicação para tornar o código mais legível e também esquemas de cores para deixar o editor de texto mais apelativo.


```

1 import java.util.Scanner;
2
3 public class Solution {
4
5     long[] dictionary;
6
7     public static void main(String[] args) {
8         Scanner in = new Scanner(System.in);
9         int n = in.nextInt();
10
11         Solution f = new Solution();
12         f.getFibWithMem(n);
13         f.printFibonacci();
14
15         // for(int i = 0 ; i<n; i++) {
16         //     System.out.print(f.fibonacciRecursive(i) + " ");
17         // }
18     }
19
20     // Get Fibonacci with Memoization
21     public long getFibWithMem(int n) {
22         if (dictionary == null) {
23             dictionary = new long[n];
24         }
25
26         if (dictionary[n - 1] == 0) {
27             if (n <= 2) {
28                 dictionary[n - 1] = n - 1;
29             } else {
30                 dictionary[n - 1] = getFibWithMem(n - 1) + getFibWithMem(n - 2);
31             }
32         }
33     }
34
35     void printFibonacci() {
36         // ...
37     }
38 }

```

Submit your code!

Figura 20- Editor ACE

Clicando em “*Submit your code!*” a submissão do utilizador é adicionada a uma fila de submissões. Após a sua submissão ser avaliada o utilizador é redirecionado para a interface que se segue (Interface da Vista de Resultados).

Na secção das submissões (ver Figura 21) é possível ver um resumo de todas as submissões realizadas para o problema em questão, e consultar de forma mais pormenorizada cada uma delas clicando em “*View Results*”.

Problem Submissions Leaderboard				
Result	Score	Language	Date	
Correct	125.00	Java	15:55 9/7/2018	View Results
Incorrect	0.00	Java	15:55 9/7/2018	View Results
Incorrect	89.29	Java	15:55 9/7/2018	View Results
Correct	125.00	Java	15:54 9/7/2018	View Results
Compiler Error	0.00	Java	15:54 9/7/2018	View Results
Correct	125.00	Java	15:39 9/7/2018	View Results

Figura 21 - Sumário de submissões

5.5.6 Interface da Vista de Resultados

Nesta interface é apresentado aos utilizadores a pontuação da sua solução de acordo com os casos de teste que acertaram.

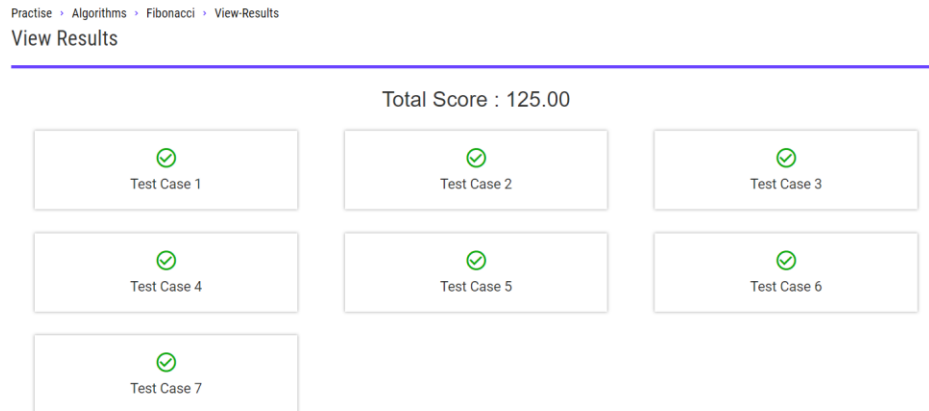


Figura 22 - Interface de vista de resultados

Cada caso de teste conta com três resultados possíveis:

1. Correto: O código submetido produz o *output* esperado
2. Incorreto: O código submetido produz um *output* incorreto
3. *Runtime Exception*: O código compilou, mas falhou na execução ou excedeu o tempo limite de execução.

É também apresentado o código submetido pelo utilizador para que o mesmo possa consultar as suas submissões passadas e rever o seu código (Figura 23).

Code submitted

```
6 // ...
7
8 public static void main(String[] args) {
9     Scanner in = new Scanner(System.in);
10    int n = in.nextInt();
11
12    Solution f = new Solution();
13    f.getFibWithMem(n);
14    f.printFibonacci();
15
16    // for(int i = 0; i < n; i++) {
17    //     System.out.print(f.fibonacciRecursive(i) + " ");
18    // }
19
20    // Get Fibonacci with Memoization
21    public long getFibWithMem(int n) {
22        if (dictionary == null) {
23            dictionary = new long[n];
24        }
25
26        if (dictionary[n - 1] == 0) {
27            if (n <= 2) {
28                dictionary[n - 1] = n - 1;
29            } else {
30                dictionary[n - 1] = getFibWithMem(n - 1) + getFibWithMem(n - 2);
31            }
32        }
33
34        return dictionary[n - 1];
35    }
36
37    public void printFibonacci() {
38        for (int i = 0; i < dictionary.length; i++) {
39            System.out.print(dictionary[i] + " ");
40        }
41    }
42 }
```

Figura 23 - Código submetido

5.5.7 Interface do Perfil

O principal objetivo da página de perfil é transmitir ao utilizador um histórico da sua interação na plataforma. Para este fim, foi adicionado um calendário do estilo *github* que apresenta o número de submissões diárias e um resumo das submissões mais recentes (Figura 24)

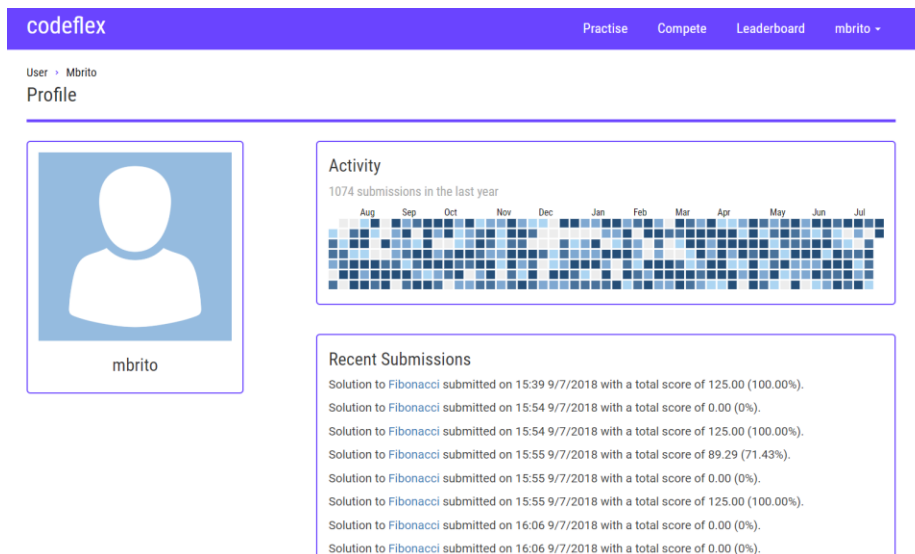


Figura 24 - Página de perfil

5.5.8 Interface de Gestão de Conteúdos

Todos os conteúdos apresentados publicamente aos utilizadores são geridos por um Gestor de Conteúdos, e dada a importância dessa tarefa optou-se por criar uma secção onde se encontram todos os elementos que podem ser geridos. Esta secção pode ser acedida clicando no nome de utilizador na *navbar* e de seguida em “*Manage Content*”.

É apresentada uma interface (ver Figura 25) na qual o gestor pode escolher o que pretende adicionar, editar, eliminar quer seja relativo a Torneios, Categorias ou Problemas. Cada uma destas interfaces pode ser encontra em anexo.

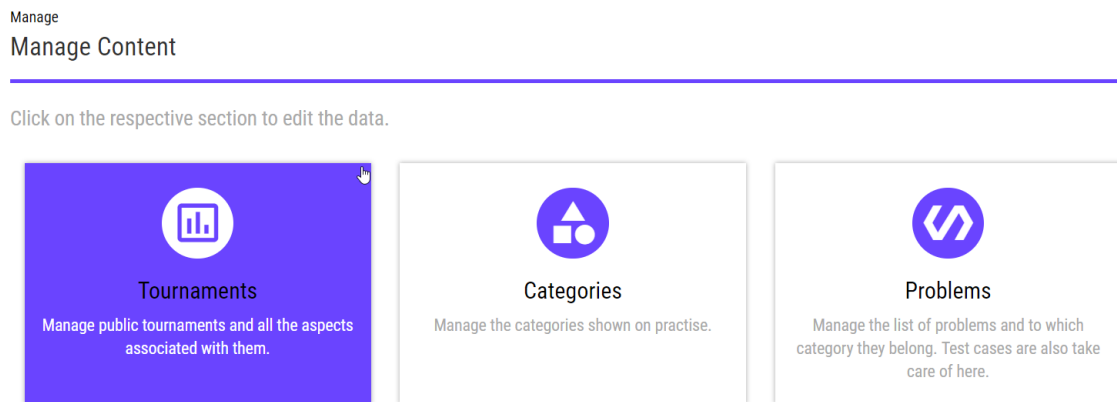


Figura 25 - Interface Manage Content

6 Testes

O desenvolvimento de *software* é um processo propenso a erros, e como tal a implementação de uma metodologia para verificação e validação das funcionalidades de acordo com o esperado é essencial. É igualmente importante ter em conta que um módulo ou funcionalidade que se encontra a operar de acordo com os parâmetros na atualidade, pode assim não se encontrar no futuro.

Para a realização dos testes optou-se pela utilização do *Postman* [21], um *software* muito popular para interação com APIs e execução de pedidos HTTP, que conta também com uma área para realização de testes. A realização de testes por este *software* contrariamente a uma *framework* como o JUnit²⁰ deveu-se essencialmente ao elevado uso já em curso do *Postman* para interação com a API do *backend*.

A fácil organização pelo uso de coleções²¹ (Figura 26) facilita o processo de gestão de pedidos²², que permite depois uma aplicação de testes a grupos de coleções.

²⁰ *Framework* para criação e execução de testes automatizados em Java

²¹ Agrupamento de pedidos por pastas

²² Mensagem HTTP que é enviada para o servidor, para com ele interagir

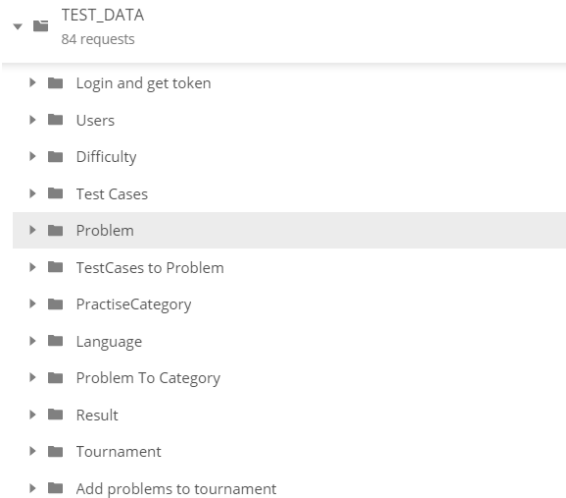


Figura 26- Coleções no Postman

Os testes em si são aplicados por *test scripts*. Estes *scripts* podem ser aplicados como pré-pedido (executados antes do pedido ser enviado) ou executados após a resposta ao pedido ser obtida. A sua execução é feita num ambiente *sandbox* executado sobre Nodejs, sendo assim possível recorrer a *javascript* para a criação do teste. Este ambiente permite verificar qualquer tipo de dados relativos aos pedidos e respostas, desde *cookies*²³, *headers*²⁴, *request body*²⁵, *response body*²⁶.

Segue-se um exemplo de um teste aplicado a uma coleção de inserção de dados relativos aos problemas.

²³ Pacote de dados onde são armazenadas informações

²⁴ Campos pertencentes ao cabeçalho de um pedido HTTP

²⁵ Corpo do pedido a ser efetuado, constituído pela informação que se pretende enviar para o servidor

²⁶ Semelhante ao *request body*, mas estes tratam-se dos dados que são enviados pelo servidor como resposta a um pedido

```
1 pm.test("status code 200", function () {  
2   pm.response.to.have.status(200);  
3 });  
4  
5 pm.test("correct data inserted", () => {  
6   pm.response.to.have.jsonBody(JSON.parse(request.data));  
7 });  
8
```

Figura 27 - Snippet de testes

No *snippet*²⁷ apresentado na Figura 27 são aplicados dois testes à coleção. O primeiro verifica se o pedido foi efetuado com sucesso e devolve uma resposta de código 200. Qualquer erro na adição dos dados a nível de *backend* resultaria num *internal error* de código 500, ou caso o URI não seja encontrado, um erro 404. O segundo teste verifica se os dados enviados são os dados que realmente foram inseridos na base de dados.

Foram aplicados testes como o anterior aos diversos métodos usados para interação com o *backend*, em especial os que dizem respeito à interação com a base de dados.

²⁷ Bloco de código

7 Processo de deploy da aplicação *web*

Deploy é o processo que torna a aplicação disponível para uso. Para tal, será necessário em primeiro lugar facultar a infraestrutura para a aplicação, que necessitará de dois servidores. No primeiro, correrão todas as aplicações de *backend* e *frontend* (Tomcat (Spring), MySQL, ReactJS) e no segundo serão disponibilizados os compiladores e é onde toda a compilação e execução será realizada.

O caminho mais óbvio, prático e barato será certamente recorrer a um serviço *cloud* para a disponibilização da aplicação. Existe uma grande oferta destes serviços, que variam desde os serviços das gigantes Google, Amazon, Microsoft até serviços de *cloud* nacionais como o caso da Claranet. Tanto a Google como a Microsoft e a Amazon oferecem horas ou créditos para novos utilizadores e/ou estudantes, no entanto no processo de registo e tentativa de obtenção deste serviço gratuito, tanto a Google como a Microsoft revelaram uma certa amargura quanto ao uso de um cartão *MB.Net* e apenas por esse motivo se optou pelo uso dos serviços *cloud* da Amazon.

Dentro da enorme variedade de serviços disponibilizados pela AWS, destaca-se o EC2 (*Elastic Computing*) que na prática se trata da disponibilização de uma máquina a qual o cliente tem acesso por SSH. Nesta máquina o cliente tem controlo total e assim sendo, torna-se intuitivo fazer a instalação e execução de todos os serviços necessários para o funcionamento da aplicação *web*. Este serviço permite também o rápido *upgrade* e *downgrade* dos recursos da máquina.

Após a obtenção do serviço e realizado o acesso às máquinas o processo foi o seguinte:

1. Fazer a clonagem do repositório do *Github*.
2. Instalar todos os serviços e dependências necessários para a execução dos diversos serviços que compõem a aplicação *web*.
3. Iniciar os serviços e servidores (Spring, MySQL, React, etc.)
4. Alterar o *security group* para disponibilizar o acesso pelo porto 80.
5. A aplicação *web* encontra-se disponível pelo endereço público associado à máquina em questão.

8 Conclusão

No decorrer deste projeto foi desenvolvida uma aplicação *web* de programação competitiva com o objetivo de oferecer um meio onde os utilizadores, em especial os estudantes da área da programação, possam consultar um conjunto de repositórios de problemas tipo para desenvolver as suas capacidades de programação e resolução de problemas. Além da parte prática, os utilizadores são desafiados a participar em torneios onde a sua capacidade é testada num ambiente cronometrado.

A aplicação *web* desenvolvida cumpre com todos os objetivos iniciais estabelecidos, e como tal encontra-se preparada para acolher utilizadores dispostos a aprender ou participar e realizar os seus torneios. Tanto no desenvolvimento do *backend* como do *frontend* foi fulcral recorrer a *frameworks* para que o processo se tornasse mais eficiente.

O desenvolvimento do módulo de avaliação foi sem dúvida das partes mais trabalhosas do projeto uma vez que não existem muitas soluções do género, e como tal, pouca informação sobre o tópico. Optou-se por ir mais além no desenvolvimento deste módulo e procurar melhorar o seu desempenho e torná-lo mais seguro, algo que foi atingido, mas fica uma grande margem para melhoria e aperfeiçoamento. O desenvolvimento deste módulo permitiu obter novos conhecimentos sobre tecnologias nunca antes utilizadas (ex: *sandboxes*, execução de processos em paralelo) e adquirir uma melhor perceção sobre o sistema operativo *Linux*.

O uso de um serviço *cloud* como o da *Amazon* foi também algo de novo para com as tecnologias conhecidas. Fica para recordar que o *deploy* é bastante mais trabalhoso do que aparente, e nesse processo, surgem um conjunto de outros problemas não existentes até à data.

Como trabalho futuro, e de forma a tornar a aplicação mais completa existem um conjunto de funcionalidades que poderão ser consideradas. Começando pelo módulo de avaliação das submissões, implementar o uso de *control groups* nas *sandboxes* utilizadas para garantir que o processamento é consistente para todas as submissões. Para tornar a aplicação mais dinâmica em contexto académico seria interessante introduzir um sistema de deteção

de plágio e um maior controlo e visão por parte do organizador dos torneios, quer pela consulta individual das submissões de cada utilizador, quer pela gestão dos participantes no torneio.

Referências Bibliográficas

- [1] Oonc, B. Cheanga, A. Kurniaa, A. Limb e Wee-Chong, “Avaliação automatizada de exercícios de programação numa instituição académica,” *On automated grading of programming assignments in an academic institution*, Setembro 2003.
- [2] “LeetCode Landing Page,” [Online]. Available: <https://leetcode.com/>. [Acedido em 12 Junho 2018].
- [3] “Leet Code Problem Set,” [Online]. Available: <https://leetcode.com/problemset/all/>. [Acedido em 13 Junho 2018].
- [4] “Codemirror Landing Page,” [Online]. Available: <https://codemirror.net/>. [Acedido em 12 Junho 2018].
- [5] “HackerRank Landing Page,” [Online]. Available: <https://www.hackerrank.com/>. [Acedido em 12 Junho 2018].
- [6] “Sphere Engine Landing Page,” [Online]. Available: <https://sphere-engine.com/>. [Acedido em 13 Junho 2018].
- [7] “Sphere Engine compiler support,” [Online]. Available: <https://sphere-engine.com/demo/1-online-compiler>. [Acedido em 13 Junho 2018].
- [8] “Agile Manifesto,” [Online]. Available: <http://agilemanifesto.org/>. [Acedido em 18 Junho 2018].
- [9] “Scrum Portugal,” [Online]. Available: <http://www.scrumportugal.pt/scrum/>. [Acedido em 14 Junho 2018].
- [10] “What is a use case,” [Online]. Available: <http://www.bridging-the-gap.com/what-is-a-use-case/>. [Acedido em 13 7 2018].
- [11] “Ranking linguagens,” [Online]. Available: <https://www.tiobe.com/tiobe-index/>. [Acedido em 2 7 2018].
- [12] “Timeline da evolução do Java,” [Online]. Available: <http://oracle.com.edgesuite.net/timeline/java/>. [Acedido em 2 7 2018].
- [13] “Gráficos de comparação de uso de frameworks Java,” [Online]. Available: <https://redmonk.com/fryan/2017/06/22/language-framework-popularity-a-look-at-java-june-2017/>. [Acedido em 2 7 2018].

- [14] D. Flanagan, “JavaScript is an interpreted programming language with object-oriented(OO) capabilities,” *JavaScript: The Definitive Guide*, p. 1, Agosto 1996.
- [15] “Javascript | MDN,” [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. [Acedido em 2 Agosto 2018].
- [16] “Bootstrap,” [Online]. Available: <https://getbootstrap.com/>. [Acedido em 2 Julho 2018].
- [17] Thalheim, H. Jaakkola e Bernhard, “Architecture-driven modelling methodologies,” em *Proceedings of the 2011 conference on Information Modelling and Knowledge Bases XXII.*, 2011.
- [18] “JSON Web Tokens,” [Online]. Available: <https://jwt.io/>. [Acedido em 15 7 2018].
- [19] “Firejail,” [Online]. Available: <https://firejail.wordpress.com/>. [Acedido em 8 7 2018].
- [20] “Rating ELO,” [Online]. Available: https://pt.wikipedia.org/wiki/Rating_ELO. [Acedido em 7 7 2018].
- [21] “Postman,” [Online]. Available: <https://www.getpostman.com/>. [Acedido em 16 7 2018].

9 Anexos

9.1 Código

9.1.1 Cálculo de *rating* para participantes num torneio

```
1. @PostMapping(path = "/Tournament/calculateRatings/{tournamentId}")
2. public void calculateTournamentRatings(@PathVariable long tournamentId) {
3.
4.     List<Users> usersUpdated = new ArrayList<>();
5.     Tournament tournament = viewTournamentById(tournamentId);
6.
7.     if (tournament == null) {
8.         return;
9.     }
10.
11.     List<Users> usersPerTournament = viewUsersByTournamentId(tournamentId);
12.
13.     for (int i = 0; i < usersPerTournament.size(); i++) {
14.
15.         Users currentUser = usersPerTournament.get(i);
16.
17.         double sumExpectedRating = 0;
18.         double sumPoints = 0;
19.
20.         for (int j = 0; j < usersPerTournament.size(); j++) {
21.
22.             Users opponent = usersPerTournament.get(j);
23.             if (currentUser == opponent)
24.                 continue;
25.
26.             sumExpectedRating += RatingCalculator.expectedRating(currentUser.getGlobalRating(),
27.                 opponent.getGlobalRating());
28.
29.             double scoreA = tournamentRepository.findScoreOfUserInTournament(currentUser.getId(),
30.                 tournament.getId());
31.             double scoreB = tournamentRepository.findScoreOfUserInTournament(opponent.getId(), tournament.getId());
32.
33.             sumPoints += RatingCalculator.pointsComparasion(scoreA, scoreB);
34.
35.         }
36.
37.         double calculatedRating = (currentUser.getGlobalRating()
38.             + RatingCalculator.K * (sumPoints - sumExpectedRating));
39.
40.         // Updates rating for current tournament
41.         Optional<Rating> currentRating = ratingRepository
42.             .findById(new RatingID(tournament.getId(), currentUser.getId()));
43.
44.         if (currentRating.isPresent()) {
45.             currentRating.get().setElo(calculatedRating);
```

```

45.     }
46.
47.     // Updates rating on overall leaderboard
48.     Users user = currentUser;
49.     user.setGlobalRating(calculatedRating);
50.     usersUpdated.add(user);
51. }
52.
53. // saves all ratings after calculating.
54. // Can't be done on the for loop because updating a rating before calculating
55. // them all would result in incorrect data
56. usersRepository.saveAll(usersUpdated);
57.
58. }

```

9.1.2 Compilação, Execução e Avaliação

```

1. package pt.codeflex.evaluatesubmissions;
2.
3. import java.io.File;
4. import java.io.FileNotFoundException;
5. import java.io.IOException;
6. import java.io.PrintWriter;
7. import java.io.UnsupportedEncodingException;
8. import java.util.ArrayDeque;
9. import java.util.ArrayList;
10. import java.util.Base64;
11. import java.util.Calendar;
12. import java.util.List;
13. import java.util.Optional;
14. import java.util.Queue;
15. import java.util.concurrent.SynchronousQueue;
16. import java.util.logging.Level;
17. import java.util.logging.Logger;
18.
19. import javax.transaction.Transactional;
20.
21. import org.springframework.beans.factory.annotation.Autowired;
22. import org.springframework.context.annotation.Scope;
23. import org.springframework.stereotype.Component;
24.
25. import net.schmizz.sshj.common.IOUtils;
26. import net.schmizz.sshj.connection.ConnectionException;
27. import net.schmizz.sshj.connection.channel.direct.Session;
28. import net.schmizz.sshj.connection.channel.direct.Session.Command;
29. import net.schmizz.sshj.transport.TransportException;
30. import net.schmizz.sshj.xfer.FileSystemFile;
31. import pt.codeflex.controllers.DatabaseController;
32. import pt.codeflex.databasesmodels.Durations;
33. import pt.codeflex.databasesmodels.Leaderboard;
34. import pt.codeflex.databasesmodels.Problem;
35. import pt.codeflex.databasesmodels.Result;
36. import pt.codeflex.databasesmodels.Scoring;

```



```

37. import pt.codeflex.databasesmodels.Submissions;
38. import pt.codeflex.databasesmodels.TestCases;
39. import pt.codeflex.databasesmodels.Tournament;
40. import pt.codeflex.models.Host;
41. import pt.codeflex.models.SubmissionWithTestCase;
42. import pt.codeflex.models.TasksBeingEvaluated;
43. import pt.codeflex.models.TestCaseForExecution;
44. import pt.codeflex.repositories.LeadersboardRepository;
45. import pt.codeflex.repositories.ResultRepository;
46. import pt.codeflex.repositories.ScoringRepository;
47. import pt.codeflex.repositories.SubmissionsRepository;
48. import pt.codeflex.repositories.UsersRepository;
49.
50. @Component
51. @Transactional
52. @Scope("prototype")
53. public class EvaluateSubmissions implements Runnable {
54.
55.     @Autowired
56.     private DatabaseController db;
57.
58.     @Autowired
59.     private UsersRepository usersRepository;
60.
61.     @Autowired
62.     private ScoringRepository scoringRepository;
63.
64.     @Autowired
65.     private SubmissionsRepository submissionsRepository;
66.
67.     @Autowired
68.     private ResultRepository resultRepository;
69.
70.     @Autowired
71.     private LeadersboardRepository leadersboardRepository;
72.
73.     private Host host;
74.
75.     private static Queue<Submissions> submissionsQueue = new ArrayDeque();
76.
77.     private static final String SERVER_USER = "ubuntu";
78.     private static final String PATH_SPRING = System.getProperty("user.home") + File
79.     .separator + "Submissions";
80.     private static final String PATH_SERVER = "Submissions";// "/home/mbrito/Desktop
81.     /Submissions"
82.     private static final String PATH_FIREJAIL = "/home/" + SERVER_USER + "/" + PATH_
83.     SERVER;// "/home/mbrito/Desktop/Submissions"
84.
85.     private volatile static long count = 0;
86.     private long uniqueId;
87.
88.     @Override
89.     public void run() {
90.         System.out.println("Thread starting!");
91.         System.out.println("Connection established!");
92.         // getSubmissions();
93.         distributeSubmissions();
94.     }
95.
96.     public List<Submissions> getSubmissions() {

```

```

94.
95.     List<Submissions> submissions = submissionsRepository.findSubmissionsToAvali
ate();
96.     List<Submissions> finalSubmissions = new ArrayList<>();
97.
98.     for (Submissions s : submissions) {
99.         Optional<Submissions> submission = submissionsRepository.findById(s.getI
d());
100.        if (submission.isPresent() && !submissionsQueue.contains(submission.get(
))) {
101.            // finalSubmissions.add(submission.get());
102.            submissionsQueue.add(submission.get());
103.        }
104.    }
105.
106.    return finalSubmissions;
107.
108. }
109.
110. public synchronized void distributeSubmissions() {
111.     while (!submissionsQueue.isEmpty()) {
112.         Submissions submission = submissionsQueue.poll();
113.         compileSubmission(submission);
114.     }
115. }
116.
117. public void compileSubmission(Submissions submission) {
118.     Problem problem = submission.getProblem();
119.     System.out.println(submission.toString() + "\n\n\n\n");
120.     uniqueId = submission.getId();
121.     Session session = null;
122.     try {
123.         session = host.getSsh().startSession();
124.     } catch (ConnectionException | TransportException e2) {
125.         e2.printStackTrace();
126.     }
127.
128.     String fileName = "Solution";
129.     String suffix = "";
130.     String compilerError = "compiler_error_" + uniqueId + ".txt";
131.     String command = "cd " + PATH_SERVER + "/" + uniqueId + "_" + submission.get
Language().getName() + " && ";
132.     // TODO : add memory limit
133.
134.     switch (submission.getLanguage().getCompilerName()) {
135.     case "Java 8":
136.         command += "javac " + fileName + ".java 2> " + compilerError + " "; // &&
disown
137.         suffix = ".java";
138.         break;
139.     case "C++11 (gcc 5.4.0)":
140.         command += "g++ -std=c++11 -
o " + fileName + "_exec_" + uniqueId + " " + fileName + ".cpp 2> "
+ compilerError + " ";
141.         suffix = ".cpp";
142.         break;
143.     case "Python 2.7":
144.         break;
145.     case "C# (mono 4.2.1)":

```

```

147.         command += "mcs -
out:" + fileName + "_exec_" + uniqueId + " " + fileName + ".cs 2> " + compilerError + "";
148.         suffix = ".cs";
149.         break;
150.     default:
151.         break;
152.     }
153.
154.     Command cmd;
155.     try {
156.         cmd = session.exec("mkdir " + PATH_SERVER + "/" + uniqueId + "_" + submission.getLanguage().getName());
157.         cmd.close();
158.     } catch (ConnectionException | TransportException e1) {
159.         e1.printStackTrace();
160.     }
161.
162.
163.     // Create and send the code to the server
164.     createFile(new String(Base64.getDecoder().decode(submission.getCode())), "Solution");
165.     scp(PATH_SPRING + "/" + fileName,
166.         PATH_SERVER + "/" + uniqueId + "_" + submission.getLanguage().getName() + "/Solution" + suffix);
167.
168.     try {
169.         session = host.getSsh().startSession();
170.         cmd = session.exec(command);
171.         cmd.close();
172.
173.         // Verifica se houve erro
174.         try {
175.             session = host.getSsh().startSession();
176.             command = "cat " + PATH_SERVER + "/" + uniqueId + "_" + submission.getLanguage().getName() + "/"
+ compilerError;
177.             cmd = session.exec(command);
178.             String output = IOUtils.readFully(cmd.getInputStream()).toString();
179.
180.
181.             if (!output.equals("")) {
182.                 List<Result> current = resultRepository.findAllByName("Compiler
Error");
183.
184.                 boolean exists = false;
185.                 for (Result r : current) {
186.                     if (r.getMessage() != null && r.getMessage().equals(output))
{
187.                         exists = true;
188.                         submission.setResult(r);
189.                     }
190.                 }
191.
192.                 if (!exists) {
193.                     Result r = new Result("Compiler Error", output);
194.                     resultRepository.save(r);
195.                     submission.setResult(r);
196.                 }
197.

```

```

198.         Scoring sc = new Scoring(submission, new TestCases(), 0, 0);
199.         scoringRepository.save(sc);
200.
201.         submissionsRepository.save(submission);
202.         System.out.println("Compiler error!");
203.         return;
204.     }
205.
206.     } catch (IOException e) {
207.
208.         e.printStackTrace();
209.     }
210.
211.     List<TestCases> testCases = submission.getProblem().getTestCases();
212.     String commandsToExecute = "";
213.     String dirName = submission.getId() + "_" + submission.getLanguage().get
Name() + "/";
214.     String obtainOutput = "(";
215.
216.     for (TestCases tc : testCases) {
217.
218.         createFileInFolder(tc.getInput(), dirName, String.valueOf(tc.getId()
));
219.
220.         commandsToExecute += getRunCommand(submission, tc) + "\n";
221.         obtainOutput += " echo '%%output_' + tc.getId() + ' ' && cat output_
" + submission.getId() + "_"
222.             + tc.getId() + ".txt && echo 'end%%' &&";
223.     }
224.     scp(PATH_SPRING + "/" + dirName,
225.         PATH_SERVER + "/" + submission.getId() + "_" + submission.getLan
guage().getName() + "/");
226.
227.     // creates jobs to be executed by parallel
228.     createFile(commandsToExecute, "jobs.txt");
229.     scp(PATH_SPRING + "/" + "jobs.txt",
230.         PATH_SERVER + "/" + submission.getId() + "_" + submission.getLan
guage().getName() + "/");
231.
232.     // removes the last unnecessary &&
233.     if (obtainOutput.length() > 2) {
234.         obtainOutput = obtainOutput.substring(0, obtainOutput.length() - 2);
235.     }
236.     obtainOutput += " ) 2> err";
237.
238.     // executes submissions in parallelism
239.
240.     String output = runTestCasesForSubmission(submission, obtainOutput);
241.     System.out.println("\n\n" + output + "\n\n");
242.
243.     //
244.     // Evaluation
245.     //
246.
247.     int totalTestCasesForProblem = problem.getTestCases().size();
248.     int givenTestCases = 0;
249.
250.     // checks all testcases
251.     for (TestCases tc : testCases) {

```

```

252.         String tcName = "%%output_" + tc.getId();
253.         String s = output;
254.         if (!s.equals("")) {
255.             s = s.substring(s.indexOf(tcName));
256.             s = s.substring(0, s.indexOf("end%%"));
257.             s = s.replace(tcName, "").trim();
258.         }
259.
260.         System.out.println("TESTCASE " + tc.getId());
261.         System.out.println(s.trim());
262.
263.         // Evaluates test case
264.         int isRight = validateResult(tc.getOutput(), s);
265.
266.         if (tc.isShown()) {
267.             givenTestCases++;
268.         }
269.
270.         double score = isRight == 1
271.             ? ((double) submission.getProblem().getMaxScore()
272.               / ((double) totalTestCasesForProblem - (double) give
nTestCases))
273.             : 0;
274.         System.out.println("Score " + score);
275.         Scoring sc = new Scoring(submission, tc, score, isRight);
276.         scoringRepository.save(sc);
277.     }
278.
279.
280.     List<Scoring> scoringBySubmission = scoringRepository.findAllBySubmissio
ns(submission);
281.     int totalScoring = scoringBySubmission.size();
282.
283.     int countCorrectScoring = 0;
284.     if (totalScoring == totalTestCasesForProblem) {
285.         double totalScore = 0;
286.         for (Scoring s : scoringBySubmission) {
287.             if (s.getIsRight() == 1) {
288.                 countCorrectScoring++;
289.             }
290.             totalScore += s.getValue();
291.         }
292.
293.         if (countCorrectScoring == totalTestCasesForProblem) {
294.             System.out.println("Correct problem!");
295.             submission.setResult(resultRepository.findByName("Correct"));
296.
297.             // Updates the completion date in order to calculate how much ti
me a user
298.             // took
299.             // // to solve the problem
300.             Durations currentDuration = db.viewDurationsById(submission.getU
sers().getId(),
301.                 submission.getProblem().getId());
302.             db.updateDurationsOnProblemCompletion(currentDuration);
303.         } else {
304.             submission.setResult(resultRepository.findByName("Incorrect"));
305.         }
306.         submission.setScore(totalScore);

```

```

307.         submissionsRepository.save(submission);
308.
309.         List<Leaderboard> highestSubmissionOnLeaderboard = leaderboardRepository
310.             .findHighestScoreByUserByProblem(submission.getUsers(), submission.getProblem());
311.
312.         Leaderboard newLeaderboard = new Leaderboard(totalScore, submission.getUsers(), submission.getProblem(),
313.             submission.getLanguage().getName());
314.
315.         Tournament currentTournament = submission.getProblem().getTournament();
316.         // if the tournament has closed already, won't change the leaderboard.
317.         if (currentTournament != null && !currentTournament.getOpen()) {
318.             cmd.close();
319.             return;
320.         }
321.
322.         if (!highestSubmissionOnLeaderboard.isEmpty()) {
323.
324.             Leaderboard maxScoreSubmission = highestSubmissionOnLeaderboard.get(0);
325.
326.             // ugly, can't figure out how to load an object using only the id from the
327.             // previous query
328.             Optional<Leaderboard> currentLeaderboard = leaderboardRepository
329.                 .findById(maxScoreSubmission.getId());
330.
331.             if (totalScore > maxScoreSubmission.getScore()) {
332.                 if (maxScoreSubmission.getScore() == -1) {
333.                     maxScoreSubmission = newLeaderboard;
334.                 } else {
335.                     if (currentLeaderboard.isPresent()) {
336.                         maxScoreSubmission = currentLeaderboard.get();
337.                         maxScoreSubmission.setScore(totalScore);
338.                     }
339.                 }
340.                 leaderboardRepository.save(maxScoreSubmission);
341.             }
342.             } else {
343.                 leaderboardRepository.save(newLeaderboard);
344.             }
345.         }
346.
347.     } catch (ConnectionException | TransportException e) {
348.         e.printStackTrace();
349.     }
350.
351. }
352.
353. public String runTestCasesForSubmission(Submissions submission, String obtainOutput) {
354.
355.     count++;
356.     System.out.println("\nCOUNT " + count + "\n");
357.

```

```

358.     Session session = null;
359.     try {
360.         session = host.getSsh().startSession();
361.     } catch (ConnectionException | TransportException e) {
362.
363.     }
364.
365.     String dirName = submission.getId() + "_" + submission.getLanguage().getName
    ();
366.     String command = " cd " + PATH_SERVER + "/" + dirName + " && parallel -
    j `nproc` < jobs.txt ; ";
367.     command = command.concat(obtainOutput);
368.
369.     try {
370.
371.         Command cmd = session.exec(command);
372.         String output = IOUtils.readFully(cmd.getInputStream()).toString();
373.
374.         System.out.println("OUTPUT " + output);
375.
376.         cmd.close();
377.         return output;
378.     } catch (IOException e) {
379.         e.printStackTrace();
380.     }
381.
382.     return null;
383.
384. }
385.
386. public String getRunCommand(Submissions submission, TestCases testCase) {
387.
388.     String dirName = submission.getId() + "_" + submission.getLanguage().getName
    ();
389.     String command = "firejail --private=" + PATH_FIREJAIL + "/" + dirName + " -
    -quiet --net=none cat " + dirName
390.         + "/" + testCase.getId() + " | ";
391.
392.     String fileName = "Solution";
393.     String runError = "runtime_error_" + submission.getId() + ".txt";
394.     String runOutput = "output_" + submission.getId() + "_" + testCase.getId() +
    ".txt";
395.     String outputPath = PATH_FIREJAIL + "/" + dirName + "/" + runOutput;
396.
397.     switch (submission.getLanguage().getCompilerName()) {
398.     case "Java 8":
399.         command += "timeout 3s java " + fileName + " 2> " + runError + " > " + r
    unOutput + "";
400.         break;
401.     case "C++11 (gcc 5.4.0)":
402.         command += " timeout 2 ./" + fileName + "_exec_" + uniqueId + " 2> " + r
    unError + " > " + runOutput + "";
403.         break;
404.     case "Python 2.7":
405.         command += "timeout 10 python " + fileName + ".py 2> " + runError + " >
    " + runOutput + "";
406.         break;
407.     case "C# (mono 4.2.1)":
408.         command += "timeout 3 ./" + fileName + "_exec_" + uniqueId + " 2> " + r
    unError + " > " + runOutput + "";

```

```

409.         break;
410.     default:
411.         break;
412.     }
413.
414.     return command;
415.
416. }
417.
418. private int validateResult(String tcOutput, String output) {
419.     System.out.println("\n\nComparing results");
420.     System.out.println(tcOutput + " - " + output + "\n\n");
421.
422.     if (tcOutput.trim().equals(output.trim())) {
423.         return 1;
424.     } else if (output.trim().equals("")) {
425.         return -1;
426.     }
427.     return 0;
428. }
429.
430. public void createFile(String text, String fileName) {
431.     PrintWriter writer = null;
432.     try {
433.         writer = new PrintWriter(PATH_SPRING + "/" + fileName, "UTF-8");
434.         writer.print(text);
435.         writer.close();
436.     } catch (FileNotFoundException e) {
437.         e.printStackTrace();
438.     } catch (UnsupportedEncodingException e) {
439.         Logger.getLogger("Create File").log(Level.SEVERE, "Error creating file",
440. e);
441.         e.printStackTrace();
442.     }
443.
444. public void createFileInFolder(String text, String folder, String fileName) {
445.     PrintWriter writer = null;
446.     try {
447.         new File(PATH_SPRING + "/" + folder).mkdirs();
448.         writer = new PrintWriter(PATH_SPRING + "/" + folder + "/" + fileName, "U
449. TF-8");
450.         writer.print(text);
451.         writer.close();
452.     } catch (FileNotFoundException e) {
453.         e.printStackTrace();
454.     } catch (UnsupportedEncodingException e) {
455.         Logger.getLogger("Create File").log(Level.SEVERE, "Error creating file",
456. e);
457.         e.printStackTrace();
458.     }
459.
460. public void scp(String src, String dest) {
461.     int count = 0;
462.     int maxCount = 5;
463.     while (count < maxCount) {
464.         try {

```



```

465.         System.out.println("Sending file from : " + src + " to " + dest + "\n\n");
466.         host.getSsh().newSCPFileTransfer().upload(new FileSystemFile(src), dest);
467.         break;
468.     } catch (IOException e) {
469.         e.printStackTrace();
470.         count++;
471.         System.out.println("Trying SCP for the " + count + " time. ");
472.     }
473. }
474. }
475.
476. public Host getHost() {
477.     return host;
478. }
479.
480. public void setHost(Host host) {
481.     this.host = host;
482. }
483.
484. }
















```

9.2 Interfaces

“Gerir Conteúdos - Torneios”

Manage > Tournaments


Manage Tournaments


Status	Name	Starting On	Ending On	Code	Registered Users	
Ongoing	Tournament Number 1	1:00 1/6/2018	1:00 31/8/2018		4	  
Finished	Tournament Number 2	1:00 6/6/2018	1:00 1/7/2018	CODEFLEX	0	  
Finished	Tournament Number 3	23:30 1/6/2018	13:45 5/6/2018		0	  
Finished	Tournament Number 4	14:07 9/6/2018	14:09 9/6/2018		0	  
Finished	Tournament Number 5	14:09 9/6/2018	14:11 9/6/2018		0	  

“Gerir Conteúdos – Categorias”

Manage Categories

Categories with at least one problem will be shown on 'Practise' section.



Algorithms 

Problem Number 1

Problem Number 2

Problem Number 6

Problem Number 7


Problem Number 9

Problem Number 10


Problem Number 12

Problem Number 13


Fibonacci

Machine Learning 

Problem Number 8

Artificial Intelligence 





Problem Number 3

Data Structures 

“Gerir Conteúdos – Problemas”

Manage > Problems

Manage Problems

Name	Difficulty	Max Score	#TestCases	
Problem Number 7	Hard	0	0 	 
Problem Number 8	Easy	0	0 	 
Problem Number 9	Medium	0	0 	 
Problem Number 10	Expert	0	0 	 
Problem Number 12	Medium	0	0 	 
Problem Number 13	Expert	0	0 	 
Problem Number 14	Medium	0	0 	 
Problem Number 15	Expert	0	0 	 
Fibonacci	Medium	125	7 	 