Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

OJTracker: Uma aplicação para acompanhamento pessoal em juízes online

Autor: Thiago Guilherme Muniz Ferreira
Orientador: Dr. Edson Alves da Costa Júnior

Brasília, DF
2023

Thiago Guilherme Muniz Ferreira

OJTracker: Uma aplicação para acompanhamento pessoal em juízes online

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Dr. Edson Alves da Costa Júnior

Brasília, DF

2023

Thiago Guilherme Muniz Ferreira
OJTracker: Uma aplicação para acompanhamento pessoal em juízes online/ Thiago Guilherme Muniz Ferreira. – Brasília, DF, 2023-
70 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Acompanhamento pessoal em Juiz Online. 2. Recomendação de problemas. I. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. OJTracker: Uma aplicação para acompanhamento pessoal em juízes online

CDU 02:141:005.6

Thiago Guilherme Muniz Ferreira

OJTracker: Uma aplicação para acompanhamento pessoal em juízes online

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, :

Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Vinícius Ruela Pereira Borges
Convidado 1

Prof. Dr. Bruno Cesar Ribas
Convidado 2

Brasília, DF
2023

"A verdadeira sabedoria é reconhecer o valor da própria ignorância.
(Lee Sin, League Of Legends)

Resumo

As competições de programação cresceram bastante de popularidade nos últimos anos. A principal forma dos competidores treinarem para esses eventos é através de juízes *online*. A vasta quantidade de questões disponíveis dificulta a decisão do estudante de escolher o próximo problema a ser resolvido. Ter uma ferramenta de apoio, capaz de permitir acompanhar a evolução do estudante nos principais juízes disponíveis e indicar problemas, permitem uma melhor percepção dos pontos fortes e fracos do indivíduo e direcionam o estudante ao próximo passo. Atualmente existem algumas soluções que podem ser utilizadas para essas finalidades. Entretanto cada uma delas é focada em apenas um único juiz, sendo necessário utilizar uma plataforma diferente para cada *site* de treinamento. Este trabalho tem como objetivo criar uma solução capaz de centralizar as informações obtidas entre os diversos juízes além de recomendar questões a serem resolvidas nessas plataformas. A extração dos dados de um estudante nas plataformas será realizada por meio das *APIs* de cada *website* ou através de *web scraping*. A recomendação de problemas será feita de forma simples, extraindo da lista de problemas disponíveis as cinco primeiras que não foram resolvidas pelo usuário.

Palavras-chave: juiz *online*. acompanhamento em juízes *online*. indicação de problemas. programação competitiva. competição de programação. *web scraping*.

Abstract

Programming contests have grown in popularity in the last years. The main way for competitors to prepare themselves for these events is through online judges. The vast amount of available questions makes it difficult for the student to choose the next problem to be solved. Having a support tool, capable of allow follow-up track the student's evolution in the main online judges available and indicate problems, allow a better perception of strengths and remain of the individual and direct the student to the next step. Currently there are some solutions that can be used for these purposes. However, each of them is focused on just a single online judge, being necessary to use a different platform for each site. This work aims to create a solution capable of centralizing information transiting between the various online judges in addition to recommending problems to be resolved in these platforms. The student's data on the platforms will be obtained by through the APIs of each site or through web scraping. The problem recommendation will be done in a simple way, extracting from the list of available problems the first five that have not been resolved by the user.

Key-words: online judges. tracking in online judges. problems recommendation. competitive programming. programming contests. web scraping.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACM	*Association for Computing Machinery*
API	*Application Programming Interface*
CORS	*Cross-origin Resource Sharing*
CSS	*Cascading Style Sheets*
DACU	*Distinct Accepted Users*
HTTP	*Hypertext Transfer Protocol*
ICPC	*International Collegiate Programming Contest*
IOI	*International Olympiad in Informatics*
JSON	*Java Script Object Notation*
LTS	*Long-Term Support*
OJ	*Online Judge*
SBC	Sociedade Brasileira de Computação
SPA	*Single-Page Application*
SPOJ	*Sphere Online Judge*

Sumário

1 Introdução

Programação competitiva é um esporte mental no qual os competidores devem resolver uma lista de questões dentro de um intervalo de tempo limitado. A participação pode ser, a depender das normas da competição, individual ou em equipe. Os problemas são resolvidos por meio da programação de computadores. Os competidores implementam uma solução que acreditam funcionar e enviam o código-fonte para o sistema.

Segundo RIBEIRO e GUERREIRO (2008), é um clichê dizer que computadores são excelentes em realizar tarefas repetitivas. Por isso, a correção das soluções enviadas ao sistema durante uma competição é feita através de uma ferramenta chamada juiz eletrônico. Para proporcionar um *feedback* acerca da corretude da solução proposta, o juiz eletrônico executa diversos testes com o código recebido e verifica se a saída gerada consiste na solução do problema.

Para toda competição é necessária uma preparação, logo ter algo ou alguém que auxilie nesta etapa pode encurtar o tempo necessário para alcançar o objetivo final. Em programação isso não é diferente, sendo necessário praticar cada tópico de forma a entender as mais diversas nuances para melhor aplicá-los na construção de uma solução de um problema durante a competição.

Tendo isto em vista, os juízes *online* foram criados. Tratam-se de ferramentas *online* que possuem uma lista de problemas, que podem ou não pertencer a alguma competição realizada, e um programa para correção automatizada das submissões. Deste modo, é o ambiente ideal para os estudantes praticarem o conhecimento teórico adquirido previamente e se acostumarem com o formato de uma competição de programação.

Muitos dos juízes *online* possuem um sistema de ranqueamento, o qual pode fornecer uma ou mais métricas, as quais permitem aos usuários visualizarem sua evolução ao longo da caminhada como programador competitivo. Portanto, ter uma ferramenta analítica de apoio pode ser útil para identificar quais tópicos necessitam de uma maior atenção no momento.

Escolher um dentre tantos problemas e tópicos disponíveis não é uma tarefa fácil, e tentar resolver um problema ou aprender um conteúdo sem ter o conhecimento prévio necessário pode ser mais frustrante do que motivador. Sendo assim, receber uma recomendação do próximo passo é fundamental para manter a motivação durante o processo de preparação.

1.1 Motivação

Um juiz *online* é uma ferramenta utilizada pelos estudantes para desenvolverem suas habilidades em programação competitiva. As competições hospedadas nestas plataformas ajudam o competidor a se acostumar com o formato e o ambiente de uma competição real. Além disso, a vasta quantidade de problemas oferece ao aluno a opção de praticar algum tópico específico de computação até se sentirem satisfeitos com o nível obtido.

Atualmente existem algumas ferramentas que permitem ao competidor o acompanhamento de seu progresso em alguns dos principais juízes *onlines* e recebimento de recomendação de problemas a serem resolvidos nessas plataformas. O uHunt[1] oferece esses recursos para o *site* Online Judge[2], o A2OJ Ladders[3] para o Codeforces[4] e o AtCoder Problems[5] para o AtCoder[6].

Entretanto, cada uma das soluções citadas acima é especializada em uma única plataforma, fazendo com que o estudante tenha que utilizar cada ferramenta separadamente, a depender do *site* escolhido. Com este trabalho, será possível ter acesso a alguns dos recursos das soluções anteriormente citadas através de uma única interface.

1.2 Objetivos

Esse trabalho tem como objetivo geral criar uma aplicação capaz de centralizar as informações de um usuário nas plataformas Codeforces, AtCoder, Online Judge, SPOJ e Codechef, além de recomendar problemas a serem resolvidos dentro destas plataformas.

Para alcançar o objetivo geral, os seguintes objetivos específicos precisam ser alcançados:

- extrair informações dos problemas e usuários das plataformas Codeforces, AtCoder, Online Judge, SPOJ e Codechef;
- implementar um algoritmo de recomendação de problemas com as questões disponíveis nas plataformas acima;
- criar uma interface para que o usuário possa utilizar os recursos definidos; e
- realizar *deploy* dos serviços de forma a disponibilizá-los mais facilmente aos usuários.

[1] <https://uhunt.onlinejudge.org>
[2] <https://onlinejudge.org>
[3] <https://earthshakira.github.io/a2oj-clientside/server/Ladders.html>
[4] <https://codeforces.com>
[5] <https://kenkoooo.com/atcoder/>
[6] <https://atcoder.jp>

1.3 Estrutura do trabalho

Este trabalho está dividido em cinco capítulos, Introdução, Fundamentação Teórica, Metodologia, Resultados e Considerações finais. A introdução apresenta o contexto e a motivação para a criação do OJTracker, bem como os objetivos a serem cumpridos para a realização deste trabalho. A Fundamentação teórica analisa trabalhos semelhantes a este e explica alguns conceitos referentes ao domínio do projeto. A Metodologia expõe como se deu o desenvolvimento do trabalho, detalhando a arquitetura do *software* e as ferramentas utilizadas. Resultados apresenta o que foi concluído. Por fim, o capítulo de considerações finais discorre sobre o que deu certo e o que poderia ter sido feito melhor durante o desenvolvimento do trabalho, além de propor futuras melhorias.

2 Fundamentação Teórica

Este capítulo tem como objetivo especificar os principais conceitos que moldam o presente trabalho. A Seção 2.1 descreve trabalhos que possuem semelhanças com o que será desenvolvido. A Seção 2.2 abre a descrição dos termos referentes ao projeto, explicando o que é um juiz eletrônico. A Seção 2.3 define o que é programação competitiva e como ela se difere do desenvolvimento tradicional de *software*. A Seção 2.4 descreve juízes *online* e cita alguns dentre os principais disponíveis. A Seção 2.5 discorre sobre as regras das competições de programação e cita algumas dentre as principais competições. Por fim, a Seção 2.6 comenta sobre recomendações de problemas em juízes *online*.

2.1 Trabalhos Correlatos

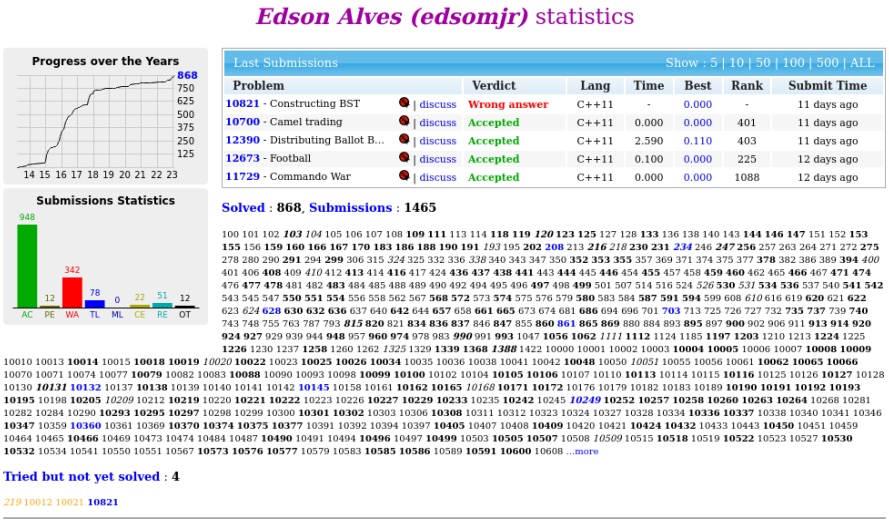
Esta seção tem como objetivo expor as ferramentas uHunt, A2OJ Ladders, AtCoder Problems e alguns projetos *open source* hospedados no Github. Estes trabalhos possuem funcionalidades que se assemelham ao que foi proposto nesse trabalho de conclusão de curso. A análise dos trabalhos existentes é fundamental para identificar lacunas de pesquisa, abordagens utilizadas e contribuições já realizadas.

2.1.1 uHunt

O uHunt[1] é um projeto criado por Felix Halim, um dos autores da série de livros *Competitive Programming*. O propósito da plataforma é manter as estatísticas dos problemas que os usuários do Online Judge (OJ)[2] já solucionaram, ou tentaram, como mostra a Figura 1. Os problemas listados podem ser filtrados com base na edição do livro em que aparecem, veja a Figura 2.

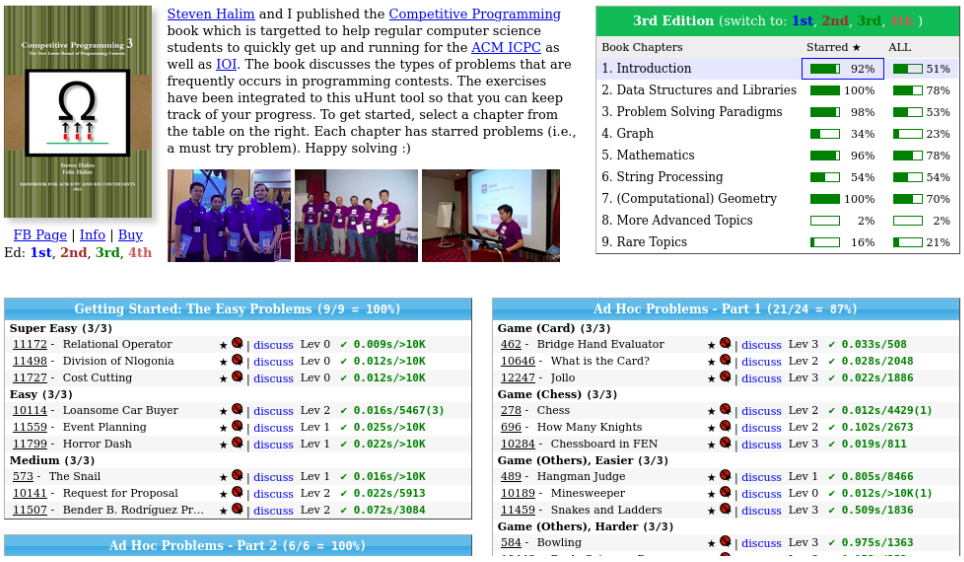
[1] <https://uhunt.onlinejudge.org>
[2] <https://onlinejudge.org>

Figura 1 – Seção de estatísticas do uHunt



Fonte: uHunt (2023)

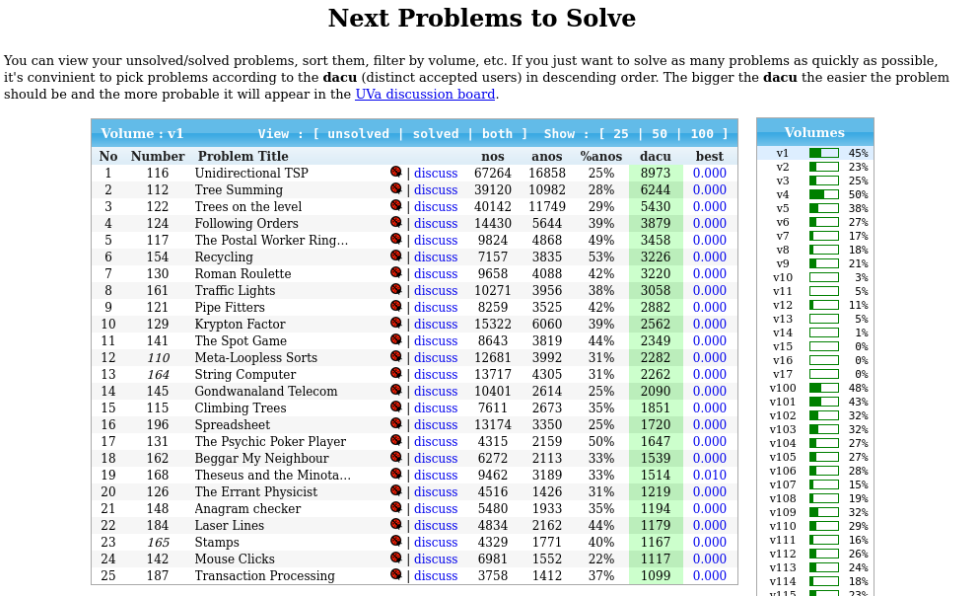
Figura 2 – Seção de problemas dos livros *Competitive Programming* na plataforma uHunt



Fonte: uHunt (2023)

Outra funcionalidade desse projeto é a seção de recomendação de problemas, como pode ser visto na Figura 3. O sistema indica aos usuários uma lista de questões com base no *rank* interno da plataforma, o *dacu (distinct accepted users)*, no qual, quanto mais alto o valor, mais fácil será a questão em comparação com questões com menor *dacu*.

Figura 3 – Seção de recomendação de problemas do uHunt



Next Problems to Solve

You can view your unsolved/solved problems, sort them, filter by volume, etc. If you just want to solve as many problems as quickly as possible, it's convinient to pick problems according to the **dacu** (distinct accepted users) in descending order. The bigger the **dacu** the easier the problem should be and the more probable it will appear in the UVa discussion board.

Volume : v1 View : [unsolved | solved | both] Show : [25 | 50 | 100]

No	Number	Problem Title		nos	anos	%anos	dacu	best
1	116	Unidirectional TSP	| discuss	67264	16858	25%	8973	0.000
2	112	Tree Summing	| discuss	39120	10982	28%	6244	0.000
3	122	Trees on the level	| discuss	40142	11749	29%	5430	0.000
4	124	Following Orders	| discuss	14430	5644	39%	3879	0.000
5	117	The Postal Worker Ring...	| discuss	9824	4868	49%	3458	0.000
6	154	Recycling	| discuss	7157	3835	53%	3226	0.000
7	130	Roman Roulette	| discuss	9658	4088	42%	3220	0.000
8	161	Traffic Lights	| discuss	10271	3956	38%	3058	0.000
9	121	Pipe Fitters	| discuss	8259	3525	42%	2882	0.000
10	129	Krypton Factor	| discuss	15322	6060	39%	2562	0.000
11	141	The Spot Game	| discuss	8643	3819	44%	2349	0.000
12	*110*	Meta-Loopless Sorts	| discuss	12681	3992	31%	2282	0.000
13	*164*	String Computer	| discuss	13717	4305	31%	2262	0.000
14	145	Gondwanaland Telecom	| discuss	10401	2614	25%	2090	0.000
15	115	Climbing Trees	| discuss	7611	2673	35%	1851	0.000
16	196	Spreadsheet	| discuss	13174	3350	25%	1720	0.000
17	131	The Psychic Poker Player	| discuss	4315	2159	50%	1647	0.000
18	162	Beggar My Neighbour	| discuss	6272	2113	33%	1539	0.000
19	168	Theseus and the Minota...	| discuss	9462	3189	33%	1514	0.010
20	126	The Errant Physicist	| discuss	4516	1426	31%	1219	0.000
21	148	Anagram checker	| discuss	5480	1933	35%	1194	0.000
22	184	Laser Lines	| discuss	4834	2162	44%	1179	0.000
23	*165*	Stamps	| discuss	4329	1771	40%	1167	0.000
24	142	Mouse Clicks	| discuss	6981	1552	22%	1117	0.000
25	187	Transaction Processing	| discuss	3758	1412	37%	1099	0.000

Volumes

v1	45%
v2	23%
v3	25%
v4	50%
v5	38%
v6	27%
v7	17%
v8	18%
v9	21%
v10	3%
v11	5%
v12	11%
v13	5%
v14	1%
v15	0%
v16	0%
v17	0%
v100	48%
v101	43%
v102	32%
v103	32%
v104	27%
v105	27%
v106	28%
v107	15%
v108	19%
v109	32%
v110	29%
v111	16%
v112	26%
v113	24%
v114	18%

Fonte: uHunt (2023)

2.1.2 A2OJ Ladders

O A2OJ Ladders[3] é uma ferramenta criada dentro do antigo A2 judge, no qual é sugerida aos usuários uma lista com as questões hospedadas no site Codeforces[4]. Segundo uma publicação[5] de Ahmed Ahly, o autor da ferramenta, no blog do Codeforces, inicialmente foram criadas 7 ladders com 100 problemas cada, como ilustrado na Figura 4. Cada problema foi extraído dos *rounds*[6] dos Codeforces e as dificuldades eram baseadas no índice da questão dentro da competição.

[3] <https://earthshakira.github.io/a2oj-clientside/server/Ladders.html>
[4] <https://codeforces.com>
[5] <https://codeforces.com/blog/entry/16443>
[6] Competições de programação hospedadas no Codeforces

Figura 4 – Ladders baseado nas divisões do Codeforces

ID	Name	Problems Count
4	Codeforces Div. 2, A	100
5	Codeforces Div. 2, B	100
6	Codeforces Div. 2, C	100
7	Codeforces Div. 2, D	100
8	Codeforces Div. 2, E	100
9	Codeforces Div. 1, D	100
10	Codeforces Div. 1, E	100

Fonte: Ladders (2023)

Posteriormente mais ladders foram adicionadas ao juiz, assim como mostra a Figura 5. Essa lista de questões, conforme Ahly descreve em outra publicação[7], foi gerada baseada em 700 usuários do Codeforces que aumentaram seu *rating* em mais de 600 pontos em um período de até 2 anos, participaram de, pelo menos, 20 competições oficias da plataforma e resolveram ao menos 150 questões no mesmo período. Dentro de um intervalo de valores de *rating* desses usuários, foram encontrados até 100 problemas em comum, sendo criada uma *ladder* para cada um dos intervalos.

[7] <https://codeforces.com/blog/entry/47442>

Figura 5 – Ladders gerados a partir de problemas em comum entre usuários com uma evolução no *rating* de até 600 pontos

ID	Name	Problems Count
11	Codeforces Rating < 1300	100
12	1300 <= Codeforces Rating <= 1399	100
13	1400 <= Codeforces Rating <= 1499	100
14	1500 <= Codeforces Rating <= 1599	100
15	1600 <= Codeforces Rating <= 1699	100
16	1700 <= Codeforces Rating <= 1799	100
17	1800 <= Codeforces Rating <= 1899	100
18	1900 <= Codeforces Rating <= 1999	100
19	2000 <= Codeforces Rating <= 2099	100
20	2100 <= Codeforces Rating <= 2199	100
21	Codeforces Rating >= 2200	100
22	Codeforces Rating < 1300 (Extra)	200
23	1300 <= Codeforces Rating <= 1399 (Extra)	200
24	1400 <= Codeforces Rating <= 1499 (Extra)	200
25	1500 <= Codeforces Rating <= 1599 (Extra)	200
26	1600 <= Codeforces Rating <= 1699 (Extra)	200
27	1700 <= Codeforces Rating <= 1799 (Extra)	200
28	1800 <= Codeforces Rating <= 1899 (Extra)	200
29	1900 <= Codeforces Rating <= 1999 (Extra)	200
30	2000 <= Codeforces Rating <= 2099 (Extra)	200
31	2100 <= Codeforces Rating <= 2199 (Extra)	200
32	Codeforces Rating >= 2200 (Extra)	200

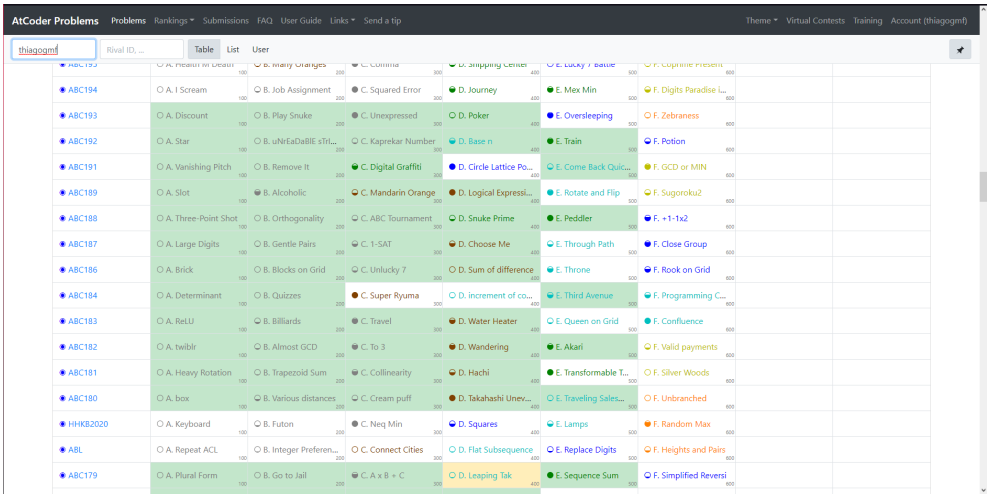
Fonte: Ladders (2023)

O A2OJ foi descontinuado e não é mais possível acessá-lo. Entretanto, as ladders citadas ainda estão disponíveis em uma página estática.

2.1.3 AtCoder Problems

O AtCoder Problems[8] é um website que permite ao usuário do AtCoder[9] visualizar seu progesso pessoal de forma eficiente e centralizada. O usuário tem acesso à todas as competições hospedadas no juiz *online* japonês, além de especificar quais questões já foram resolvidas, como mostra a Figura 6.

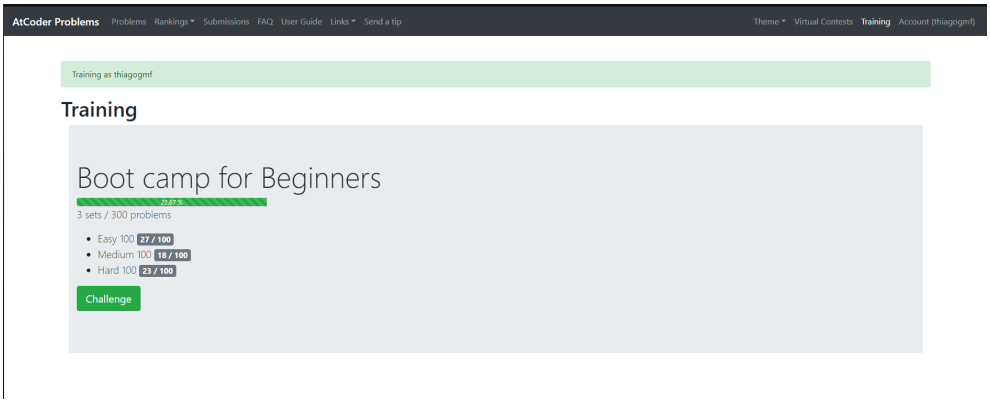
Figura 6 – Seção de problemas do AtCoder Problems



Fonte: Problems (2023)

Essa ferramenta também possui duas seções de recomendação de problemas. A primeira, chamada de *Training* (vide Figura 7), conta com uma lista fixa de 300 questões distribuídas uniformemente em 3 diferentes níveis: fácil, médio e difícil. A segunda seção, chamada de *Recommendation* (vide Figura 8), conta com uma lista de questões também subdividida nos níveis fácil, médio e difícil, porém aqui é dada ao usuário a liberdade de escolher quantos itens serão exibidos.

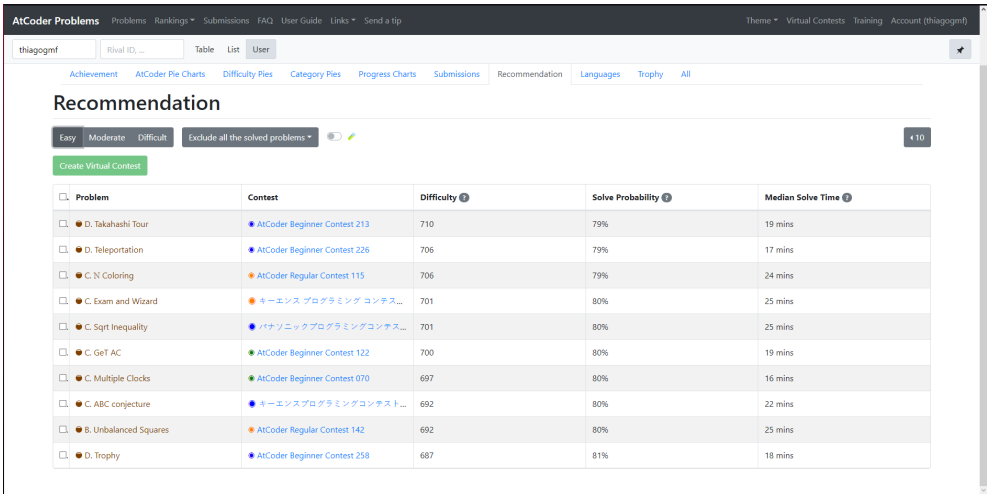
Figura 7 – Seção de treinamento do AtCoder Problems



Fonte: Problems (2023)

8 <https://kenkoooo.com/atcoder/>
9 <https://atcoder.jp>

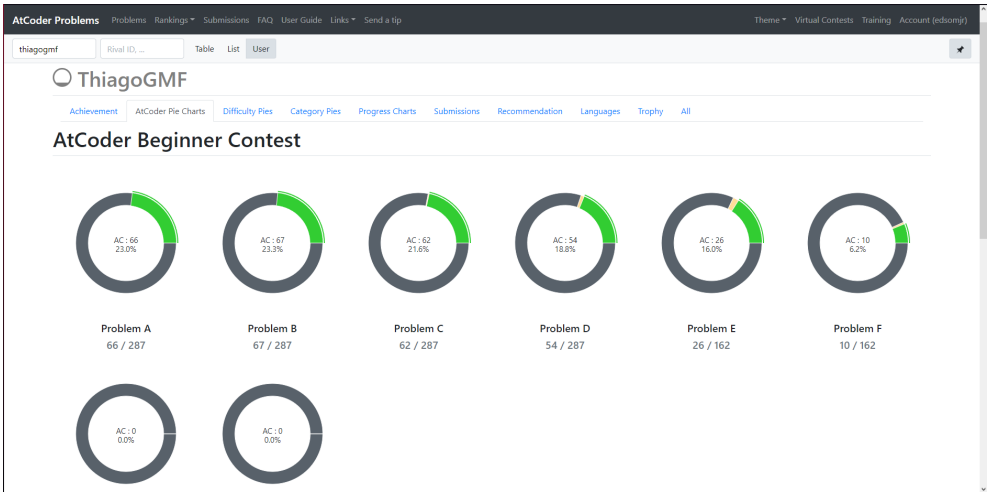
Figura 8 – Seção de recomendação de problemas do AtCoder Problems



Fonte: Problems (2023)

Além disso, outra seção interessante da ferramenta é a de estatísticas do usuário. A Figura 9 exibe graficamente algumas informações sobre o desempenho do estudante, como, por exemplo, a distribuição de questões resolvidas baseada no índice da mesma dentro de uma competição oficial do AtCoder.

Figura 9 – Seção estatísticas do usuário na plataforma AtCoder Problems



Fonte: Problems (2023)

2.1.4 Projetos *Open Source*

As soluções acima são as mais conhecidas desse segmento, porém também existem alguns projetos disponíveis no Github[10] que podem ser utilizados pela comunidade. Os mais simples entre eles são apenas *API*s que retornam um *JSON (Javascript Object Notation)* com algumas informações disponíveis por um juiz *online* com base no

[10] <https://github.com>

nome do usuário, como por exemplo os projetos *all-cp-platforms-API*[11] e o *Competitive-Programming-Score-API*[12].

2.1.5 Artigos sobre recomendações de problemas

Como um dos objetivos específicos deste trabalho é a recomendação de problemas a serem resolvidos pelos estudantes nas plataformas suportadas, faz-se necessário um algoritmo de recomendação de problemas, por mais simples que seja. Esse tema foi alvo de diversos artigos que utilizam de abordagens mais complexas do que as serão utilizadas neste trabalho, como por exemplo através de inteligência artificial, em FANTOZZI e LAURA (2020), que utiliza de redes neurais, ou até mesmo em TOLEDO, MOTA e MARTÍNEZ (2018), que utiliza de modelagem de informações *fuzzy*.

2.2 Juízes eletrônicos

Conforme descrito em RIBEIRO e GUERREIRO (2008), as principais competições de programação são baseadas em problemas de *input-output*, ou seja, dada uma entrada específica existe uma, ou várias, saídas que correspondem à solução de um problema. Um juiz eletrônico é um programa que fornece correções automatizadas das submissões recebidas pelos competidores.

O juiz eletrônico compila o código-fonte recebido e então, caso não haja erros, executa o programa verificando se a saída gerada corresponde a uma solução do problema. Uma submissão só é considerada correta caso gere soluções corretas para todos os casos de testes. Como relatado em BECARES (2017), um juiz eletrônico, normalmente, notifica apenas o sucesso ou insucesso da resolução, sem detalhar o que está errado. Alguns dos possíveis vereditos podem ser observados no Quadro 1.

[11] <https://github.com/SysSn13/all-cp-platforms-API>
[12] <https://github.com/Abhijeet-AR/Competitive_Programming_Score_API>

Tabela 1 – Vereditos dos juízes eletrônicos

Sigla	**Veredito**	**Descrição**
WA	*Wrong Answer*	A solução proposta gerou uma saída inválida para algum dos casos de testes.
CE	*Compilation Error*	O código-fonte submetido não pôde ser compilado.
TLE	*Time Limit Exceeded*	A solução proposta não encontrou uma solução dentro do limite de tempo definido.
MLE	*Memory Limit Exceeded*	A solução proposta utilizou mais memória do que o limite máximo permitido para o problema.
RE	*Run Time Error*	O programa não terminou de ser executado.
AC	*Accepted*	A submissão proposta gerou saídas corretas para todos os casos de testes.

Fonte: Autor

> Tradicionalmente, o processo de fornecimento de informações sobre erros cometidos em problemas de programação fazem parte da convicção de que existe uma solução canônica que o aluno deve alcançar. No entanto, esses programas são baseados na realização de análises estática do código, já que o objetivo final é que o envio do aluno assemelha-se à saída canônica que foi predefinido (BECARES, 2017).

O Quadro 1 exibe alguns dos vereditos retornados por um juiz eletrônico após o código-fonte submetido pelo competidor ser avaliado. Os vereditos permitem que os participantes identifiquem erros e aprimorem suas soluções.

2.3 Programação competitiva

Programação competitiva é a combinação entre *design* de algoritmo e a implementação do algoritmo. O *design* do algoritmo é a idealização de um software capaz de resolver um problema, através de métodos e estruturas de dados já bem definidos, juntamente com novos *insights*[13]. A implementação do algoritmo é a escrita do código-fonte da solução para um problema, o qual será testado através de um juiz eletrônico (LAAKSONEN, 2018).

A programação competitiva é um esporte mental onde os participantes podem competir, a depender das regras da competição, de forma individual ou em equipe. Os competidores recebem um *feedback* imediato tanto em uma página individual como no placar geral. No campo educacional, como citado por RIBEIRO e GUERREIRO (2008), ter um placar gera uma competição saudável entre os estudantes e isso aumenta a produtividade dos alunos, medida pela quantidade de código escrito durante o evento.

É necessário conhecer e entender diversos algoritmos e estruturas de dados para ter um bom desempenho em programação competitiva. Algumas empresas utilizam problemas

[13] percepções sobre um determinado assunto ou situação.

desta natureza como forma de selecionar um candidato durante o processo seletivo. O que difere a programação competitiva do desenvolvimento de software tradicional é o tamanho do código-fonte, que normalmente em competições não ultrapassa 100 linhas, e não há necessidade de mantê-lo após a entrega (LAAKSONEN, 2017).

2.4 Juízes *Online*

BECARES (2017) descreve juízes *online* como sistemas que, inicialmente, foram criados para hospedar questões que apareceram anteriormente em alguma competição de programação mas que com o tempo ganharam tanta popularidade que começaram a serem utilizados também como ferramenta educativa.

Em REVILLA, MANZOOR e LIU (2008) é dito que um usuário de qualquer lugar do mundo pode se registrar em um desses sistemas e resolver quantos problemas quiser, enviando quantas submissões forem necessárias até receber o *feedback* desejado. As respostas retornadas pelo juiz eletrônico incluem o veredito, o tempo de execução do programa ou até mesmo a quantidade de memória utilizada.

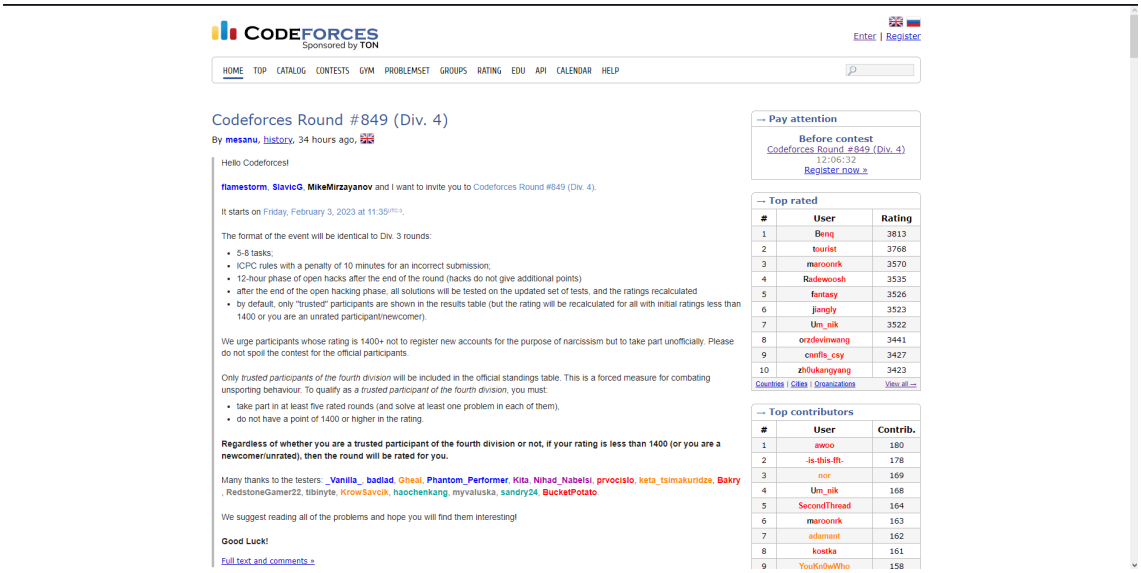
Com base nisso, um juiz *online* pode ser definido como um *software* que possui um sistema de usuários, uma lista de problemas e um juiz eletrônico para efetuar as correções das submissões recebidas e enviar o *feedback* apropriado. Alguns dos juízes *online* hospedam suas próprias competições, distribuindo pontos entre os participantes para gerar um *ranking*.

Diversos estudantes utilizam juízes *online* com o intuito de se preparem para as competições de programação. Uma vez logados no juiz *online*, os estudantes escolhem uma dentre as várias questões disponíveis, leem o enunciado do problema, escrevem o código-fonte que resolve a tarefa e os enviam para a plataforma. A submissão é verificada, testando a correção e a performance do software (FANTOZZI; LAURA, 2020).

2.4.1 Codeforces

O Codeforces é um juiz *online* russo que hospeda competições de programação, organiza treinamentos de programação e dispõe de um *blog* em que se discute as competições e questões. O site possui material nos idiomas inglês e russo. A Figura 10 ilustra a página inicial da plataforma Codeforces.

Figura 10 – Página inicial do Codeforces.

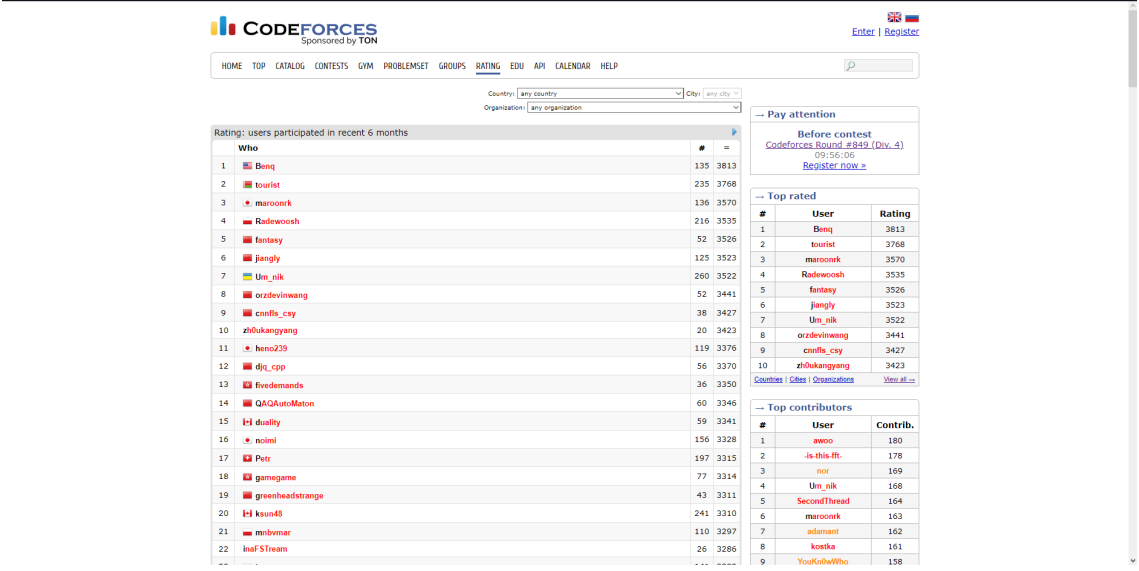


Fonte: Codeforces (2023)

Esta plataforma conta com um sistema de *rating*, que, conforme descrito em uma publicação[14] oficial, é aumentado ou diminuído com base em uma fórmula que leva em consideração a posição esperada de um participante em uma competição e a posição real.

Para evitar a situação de todos os participantes terminarem na posição esperada, fazendo com que a mudança de *rating* seja nula para todos, uma determinada quantidade de competidores são previamente escolhidos para fazerem parte de um grupo em que não receberão ou perderão pontos. Os competidores são classificados com base nessa pontuação, como pode ser observado a partir da Figura 11.

Figura 11 – *Ranking* do Codeforces



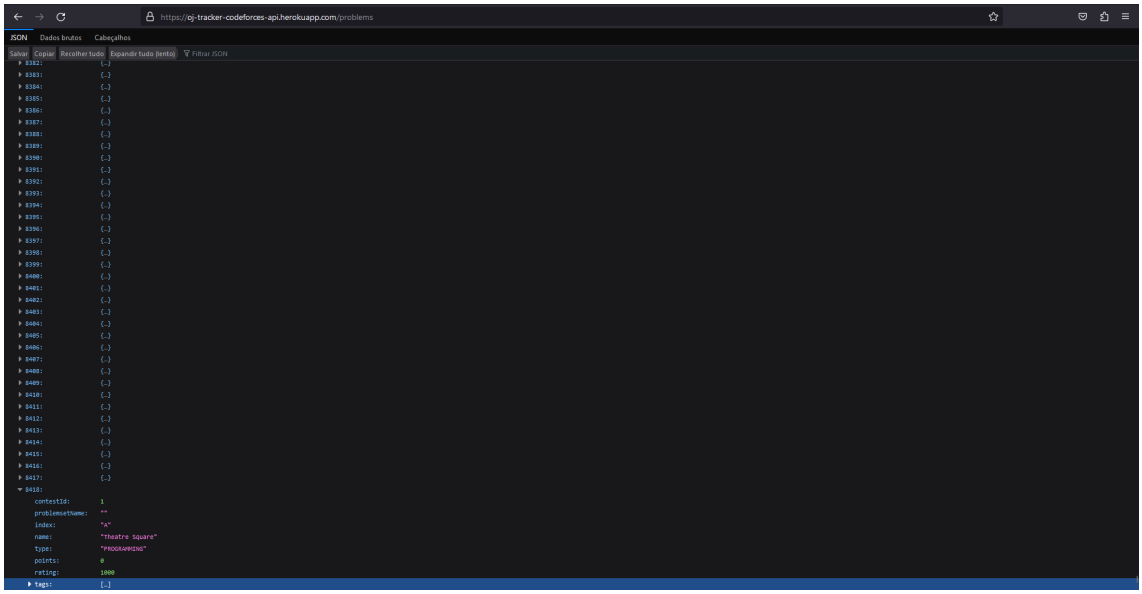
Fonte: Codeforces (2023)

14 <https://codeforces.com/blog/entry/20762>

Até o ano de 2023 as competições oficiais da plataforma são divididas em 4 divisões. Os competidores precisam ter um *rating* abaixo do limite definido para serem hábeis a pontuar nas divisões de 2 a 4, onde para a divisão 2 o limite é de 2100, na 3 de 1600 e na 4 de 1400.

Além disso, o Codeforces conta com uma *API*[15] pública, a qual foi utilizada para extrair as informações sobre os *contests*, lista de problemas (Figura 12), lista de submissões e informações dos usuários.

Figura 12 – Lista de problemas do Codeforces extraída via API.



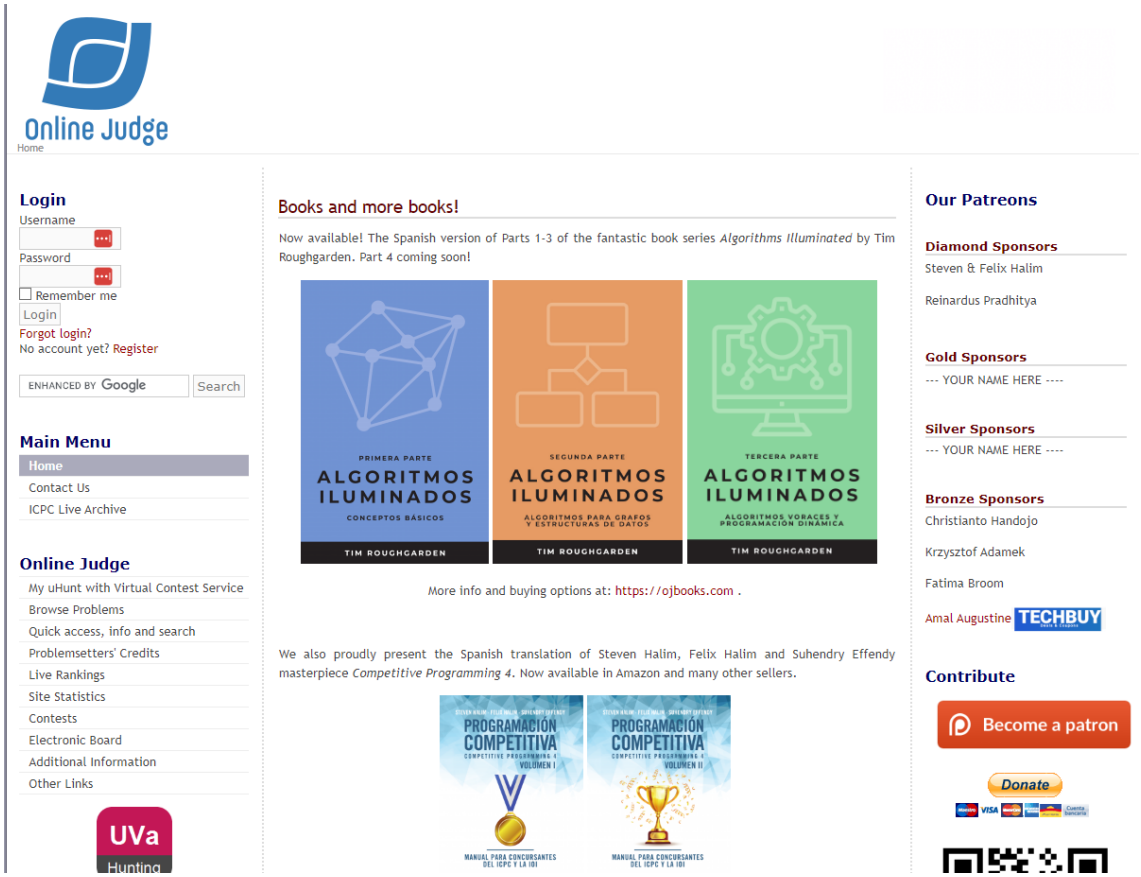
Fonte: Autor

2.4.2 Online Judge

O Online Judge, exibido na Figura 13, segundo REVILLA, MANZOOR e LIU (2008), foi criado em 1995 com o objetivo de selecionar um time da *UVa (University of Valladolid)* para participar do ACM-ICPC daquele ano. A plataforma era fechada aos estudantes da universidade, funcionava apenas algumas horas por dia e era mantida por apenas uma pessoa.

[15] <https://codeforces.com/apiHelp>

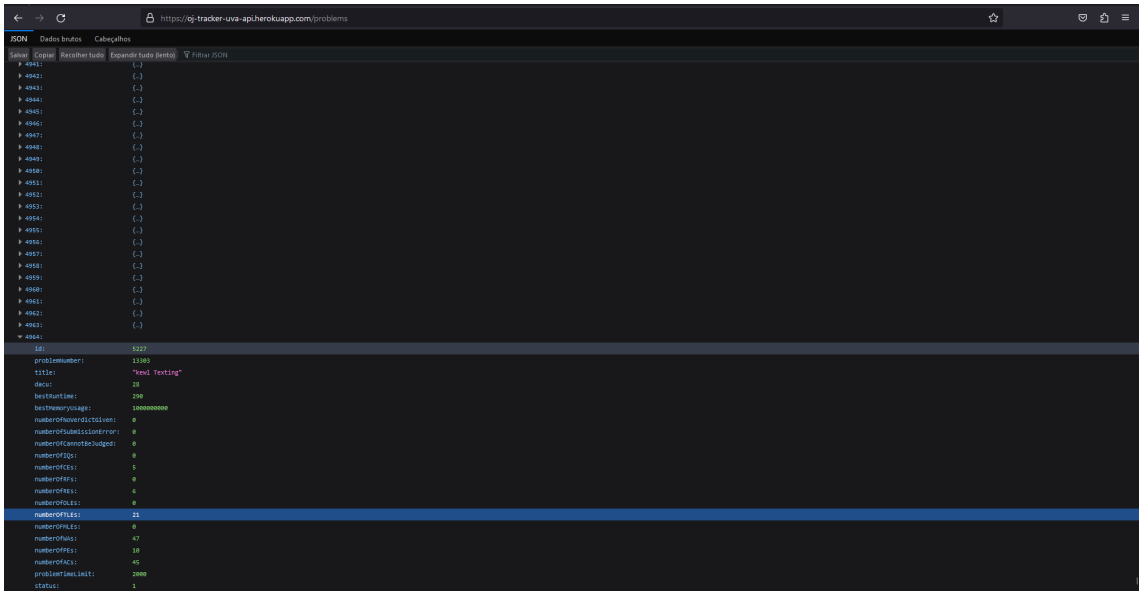
Figura 13 – Página inicial do *Online Judge*



Fonte: OnlineJudge (2023)

O site conta, em 2023, com uma lista de mais de 4900 problemas, como pode ser observado na Figura 14. Também é possível acessar uma série de competições pelo Online Judge, tendo como verificar a lista de submissões e o placar final de cada uma.

Figura 14 – Quantidade de problemas no *Online Judge*



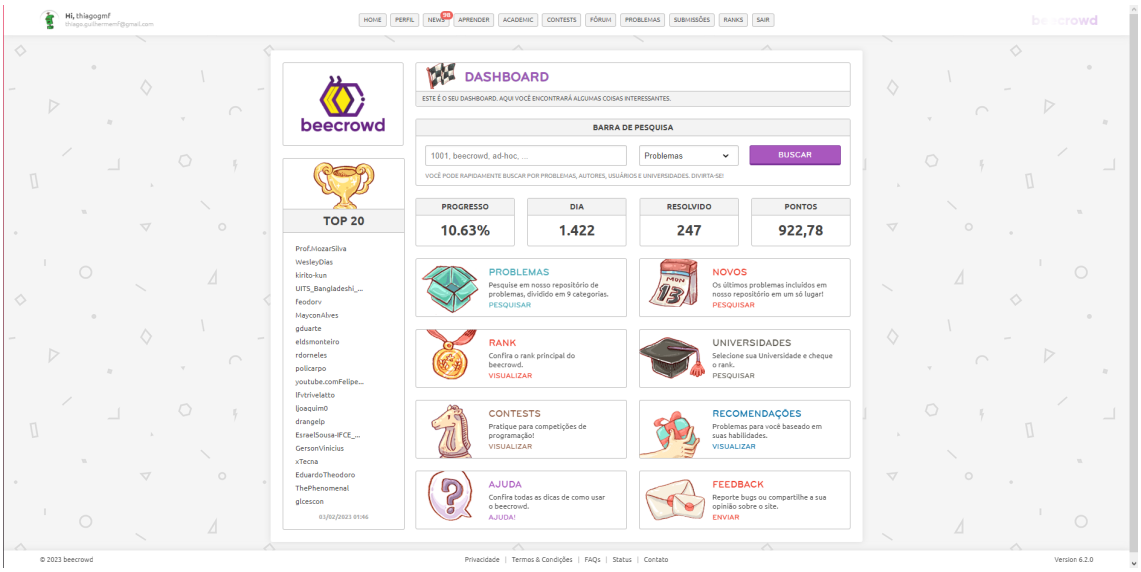
Fonte: Autor

Assim como no Codeforces, é possível extrair as informações disponíveis no Online Judge através de uma *API*[16]. Entretanto, a mesma não é disponibilizada pela própria plataforma, e sim através do site uHunt.

2.4.3 Beecrowd

O Beecrowd, cuja página inicial é ilustrada na Figura 15, é uma plataforma brasileira que foi criada, conforme citado em BEZ, TONIN e RODEGHERI (2014), no departamento de ciência da computação da Universidade Regional Integrada (URI) e teve como objetivo criar um ambiente *online* capaz de ajudar tanto alunos como professores de algoritmos e linguagens de programação.

Figura 15 – Página inicial do Beecrowd.

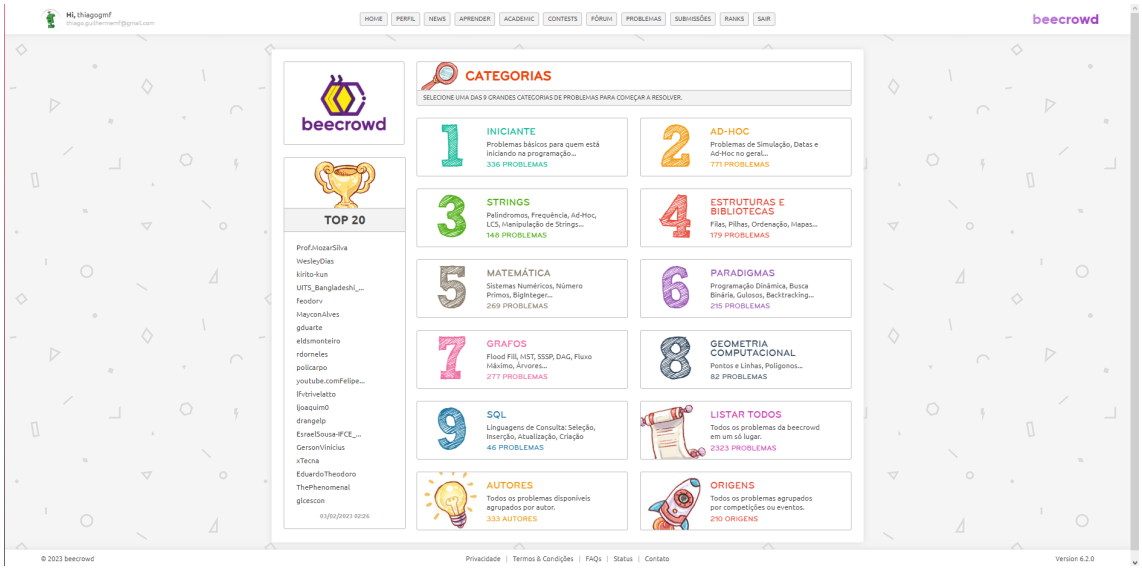


Fonte: Beecrowd (2023)

A plataforma possui uma lista de problemas de programação organizada por tópico, como pode ser observado na Figura 16. Pode-se ver um fórum onde é possível discutir com os outros usuários as questões disponíveis no juiz, um blog para conversar sobre temas diversos e uma sessão de treinamentos em outras áreas de desenvolvimento de *software*, conforme mostra a Figura 17. Além disso, o Beecrowd hospedou uma série de competições, inclusive o espelho da Maratona SBC de Programação.

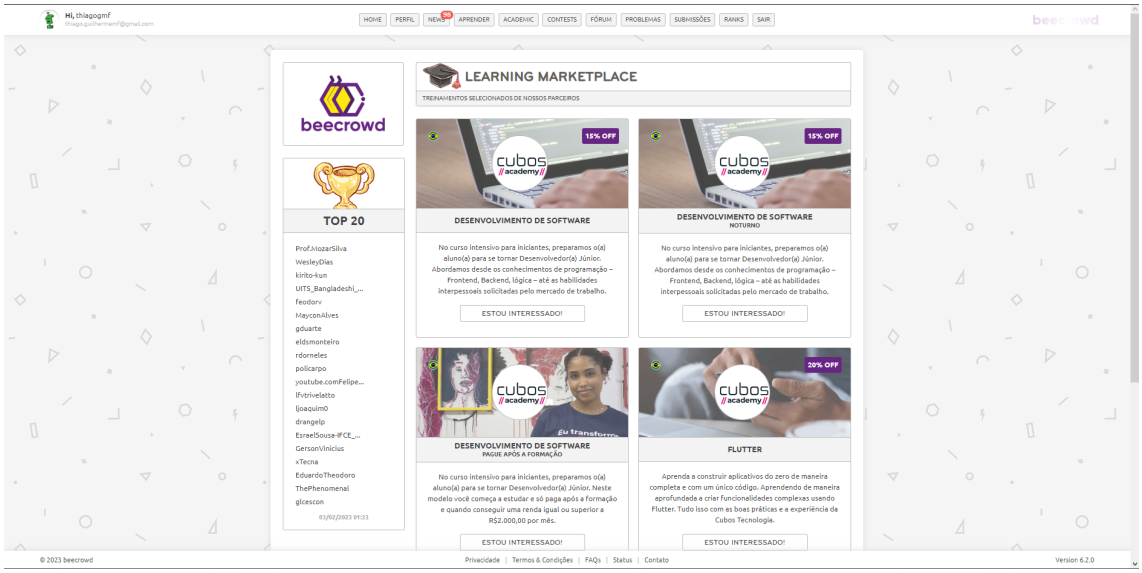
[16] <https://uhunt.onlinejudge.org/api>

Figura 16 – Lista de problemas do Beecrowd



Fonte: Beecrowd (2023)

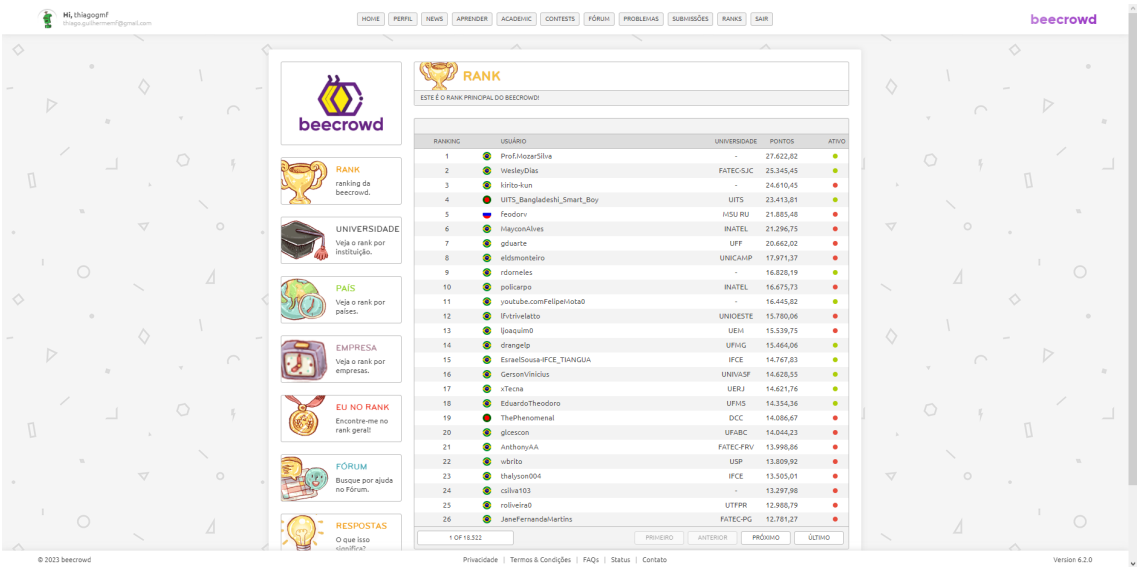
Figura 17 – Seção *Academic* do Beecrowd



Fonte: Beecrowd (2023)

Cada questão disponível na plataforma conta com um nível de dificuldade, variando de 1 a 10, o que está proporcionalmente ligado à pontuação entregue por cada uma. A soma das pontuações obtidas pelos usuários ao resolverem um problema é utilizada para gerar o *ranking*, onde, quanto maior a pontuação, melhor a posição no ranking, que pode ser visto na Figura 18.

Figura 18 – *Ranking* do Beecrowd.



Fonte: Beecrowd (2023)

2.5 Competições de programação

As competições de programação são eventos em que os competidores participantes devem responder uma série de problemas utilizando programação competitiva. As soluções propostas são testadas por meio de um juiz eletrônico: caso esteja correta o participante obtém a pontuação atribuída ao problema.

Tendo início em 1970, a principal competição de programação, o ACM-ICPC (Association for Computing Machinery International Collegiate Programming Contest)[17] é uma competição anual envolvendo equipes universitárias. Inicialmente as equipes disputam fases regionais, onde os vencedores disputam vaga para a final mundial. Cada prova tem duração de 5 horas, normalmente.

Nas regras do *ICPC* ao acertar uma questão a equipe recebe como penalidade o tempo, em minutos, requerido para finaliza-la juntamente com um acréscimo de 20 minutos por cada tentativa falha. Ao fim da competição o vencedor é aquele que conseguiu resolver a maior quantidade de questões, e em caso de empate, aquele com menor tempo acumulado.

No Brasil, os estudantes do ensino fundamental e médio podem competir na Olímpiada Brasileira de Informática (OBI)[18], um evento promovido pela Sociedade Brasileira de Computação (SBC) em conjunto com a Unicamp, cujo principal objetivo é aumentar o interesse dos alunos pela ciência da computação por meio de atividades que envolvem desafio, criatividade e uma dose saudável de competição.

[17] <https://icpc.global>
[18] <https://olimpiada.ic.unicamp.br>

A OBI é composta por duas diferentes categorias, a de programação, para alunos do ensino médio, e a de lógica, para alunos do fundamental. Ambas são subdivididas em outros níveis, a depender da idade do competidor. Os vencedores da categoria programação disputam vagas para a IOI (*International Olympiad in Informatics*)[19] (OLIVEIRA; MENDERICO, 2007).

Conforme descrito no regulamento[20] oficial da IOI, o evento se trata de uma competição internacional, anual e individual de programação. Ademais, é acompanhada de eventos sociais e programas culturais. Um dos principais objetivos desta competição é encontrar e encorajar jovens talentos da computação.

Para os programadores que não se encaixam nos níveis dos eventos acima, o Meta Hacker Cup[21] é uma competição internacional, anual, aberta a todos, hospedada e administrada pela Meta. Foi criada em 2021 com o propósito de identificar talentos de engenharia a serem recrutados para a Meta.

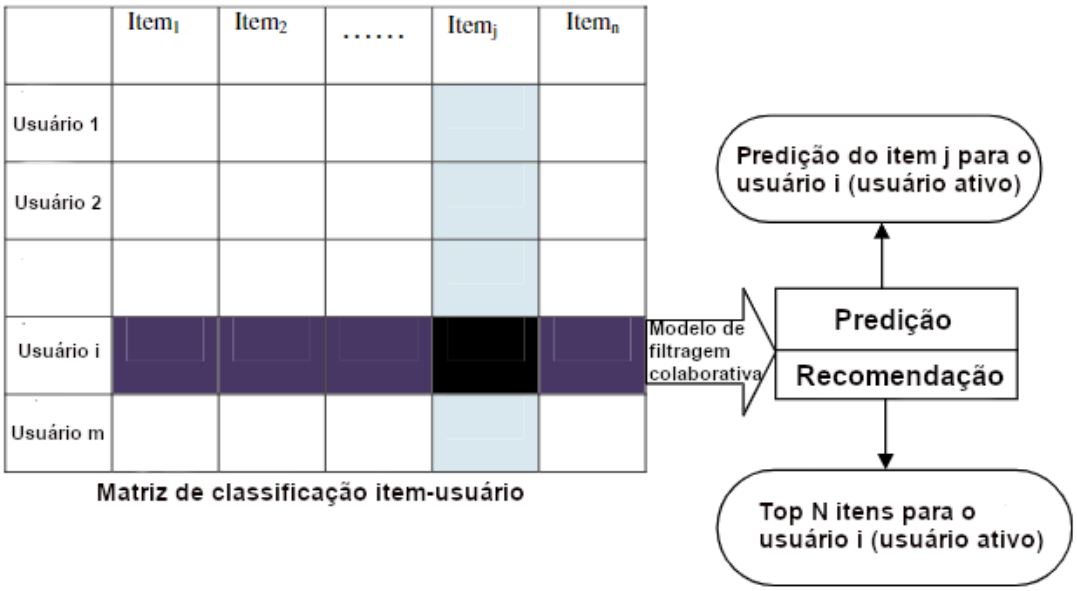
2.6 Sistemas de recomendações

Na Internet, onde a quantidade de opções e dados disponíveis aumentam cada vez mais, é essencial filtrar, priorizar e entregar informações relevantes de forma eficiente, a fim de lidar com o problema de sobrecarga de informações. Os sistemas de recomendação são desenvolvidos para enfrentar esse desafio, realizando pesquisas em grandes volumes de informações em constante atualização, a fim de fornecer aos usuários conteúdos e serviços personalizados (ISINKAYE; FOLAJIMI; OJOKOH, 2015).

Tradicionalmente, os sistemas de recomendações são divididos em duas categorias. Aqueles baseados em conteúdo, nas quais as recomendações derivam de características dos itens a serem sugeridos, como também as abordagens de filtragens colaborativas. A Figura 19 ilustra uma sugestão baseada nos itens escolhidos pelos usuários semelhantes ao atual (FANTOZZI; LAURA, 2020).

[19] <https://ioinformatics.org>
[20] <https://ioinformatics.org/files/regulations21.pdf>
[21] <https://www.facebook.com/codingcompetitions/hacker-cup>

Figura 19 – Processos da filtragem colaborativa.



Fonte: Isinkaye, Folajimi e Ojokoh (2015)

Como dito por TOLEDO, MOTA e MARTÍNEZ (2018), no contexto de juízes online, em que existem várias questões de programação disponíveis, encontrar um problema que seja compatível com as habilidades atuais do estudante é uma tarefa árdua. Tal fato pode gerar um desânimo e frustração quando o nível de dificuldade do problema escolhido não combina com o nível do usuário.

Nos juízes *online*, a dificuldade de uma questão é definida por meio de heurísticas definidas por cada plataforma, assim como o tópico tal qual a questão é categorizada. Em algumas delas, o nível do usuário é medido de acordo com um *rating* interno, o que interfere no *ranking* do usuário.

Audrito. et al. (2019) cita algumas características em problemas de programação competitiva que impedem a utilização de uma recomendação tradicional. Uma é que o conceito de preferência não é aplicável, uma vez que ao resolver uma questão pela primeira vez a torna mais fácil nas vezes subsequentes. A outra é que um grupo de usuário ao qual se pretende recomendar um determinado conjunto de tarefas podem ter comportamentos diferentes, como por exemplo, alguns podem resolver todas as questões de um determinado tópico enquanto outros apenas algumas tarefas e então partir para o próximo conteúdo.

Apesar de existirem sistemas de recomendações em plataformas *online* de aprendizagem, o mesmo esforço não tem sido aplicado em juízes *online*. Uma boa recomendação para problemas de programação é aquela onde o usuário consegue resolver a questão sugerida durante a fase de obtenção do conhecimento, evitando a frustração (TOLEDO; MOTA; MARTÍNEZ, 2018).

3 Metodologia

Este Capítulo tem o objetivo de explicitar como se deu o desenvolvimento deste trabalho. A Seção 3.1 descreve o estudo dos juízes *online*. A Seção 3.2 apresenta como foi o processo de desenvolvimento do trabalho, tanto em software quanto em texto. Por fim, a Seção 3.3 expõe quais serviços foram utilizados para realizar o *deploy* da aplicação.

3.1 Estudo dos juízes *online*

Antes de iniciar o desenvolvimento do OJTracker foi necessário avaliar os dados em comum que podem ser extraídos de diferentes juízes *online*, o que pode ser visto no Quadro 2. Esse estudo foi necessário para identificar quais informações podiam ser extraídas de cada juiz sem a necessidade de realizar um processo de autenticação nas plataformas. Com essas informações foi possível definir com maior clareza quais seriam as *features* do OJTracker.

Tabela 2 – Recursos em comum entre as *APIs*

	OJ	**Codeforces**	**AtCoder**	***SPOJ***	**Codechef**
Informações do usuário	x	x	x	x	x
Base de problemas	x	x	x	x	x
Submissões de um usuário	x	x	x	x	x
Acesso ao código fonte		x	x		x
Tag da questão	x	x		x	x
Dificuldade questão	x	x	x	x	x

Fonte: Autor

No Quadro 2, informações do usuário são dados referentes à evolução do usuário dentro do juiz *online* e informações pessoais; a base de problemas é uma lista com todas as questões disponíveis na plataforma; as submissões de um usuário indica todas as tentativas do estudante de resolver os problemas do OJ; o acesso ao código-fonte é a possibilidade de visualizar o algoritmo que o usuário submeteu; a *tag* se refere a quais abordagens podem ser utilizadas para tentar solucionar uma questão; e, por fim, a dificuldade é um valor estimado do quão fácil, ou difícil, é solucionar um problema.

Referente ao acesso do código-fonte, a plataforma SPOJ só permite a visualização por parte do dono da submissão, enquanto os outros juízes *online* com essa funcionalidade o acesso é livre. Em relação a *tag*, o Online Judge classifica somente aquelas que aparecem na série de livros *Competitive Programming*, escrita pelos irmãos Halim[1].

[1] <https://cpbook.net>

Tratando-se da dificuldade da questão, cada plataforma lida com esta informação de forma diferente. O Online Judge utiliza o *dacu* (distinct accepted users), uma contagem de quantos usuários distintos resolveram um determinado problema. O Codeforces utiliza o *rating* (uma pontuação interna obtida pelos competidores durante as competições oficiais do Codeforces) dos usuários que resolveram a questão durante o *round*. Para o SPOJ, o Codechef e para o AtCoder não foram identificados os métodos utilizados.

Para este trabalho as características utilizadas serão a base de problemas, as informações dos usuários e as submissões. A base de problemas é fundamental para realizar a recomendação. As informações do usuário servem para validar se o identificador do usuário (*handle*) inserido é válido para a plataforma em questão. As submissões permitem o acompanhamento da evolução do usuário dentro de cada juiz *online*, e quando possível uma submissão poderá ser acessada para revisão do usuário.

3.2 Desenvolvimento do trabalho

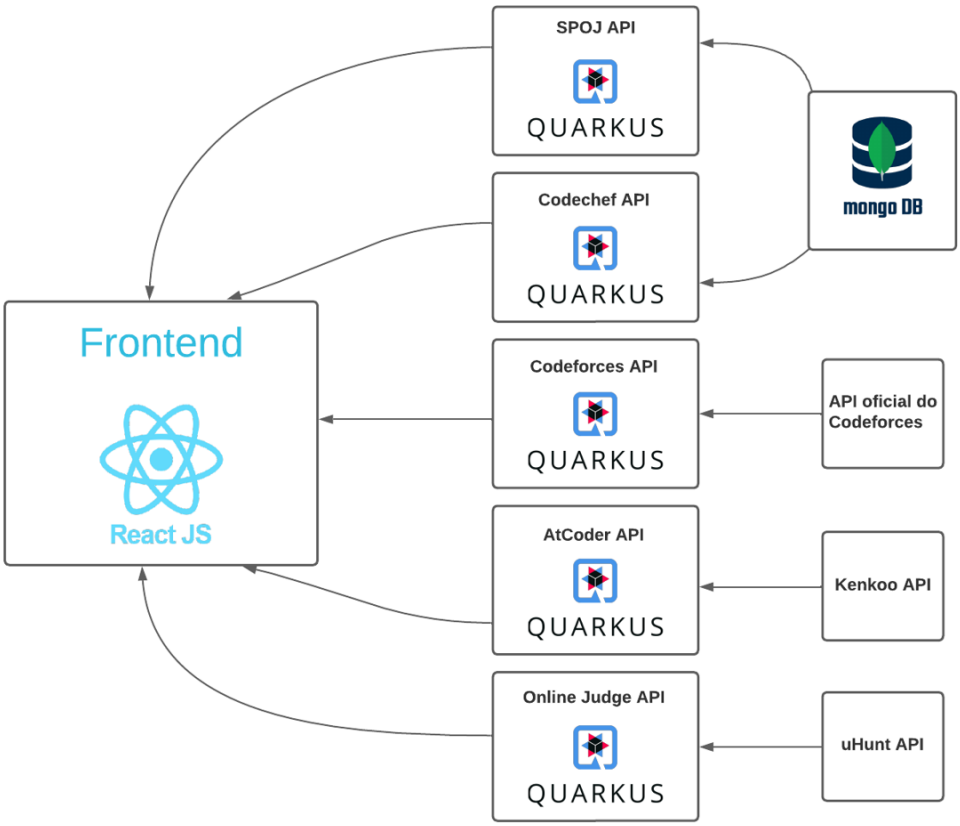
Esta seção busca explicar os processos do desenvolvimento do OJTracker, a arquitetura do *software* e qual a responsabilidade de cada componente que o compõe. Além de explicitar os processos do desenvolvimento do texto e quais ferramentas foram utilizadas durante a construção do projeto.

3.2.1 Desenvolvimento do *software*

O OJTRacker foi dividido em *backend* e *frontend*. O *backend* trata a extração dos dados dos usuários nas plataformas suportadas, persiste a lista de questões em um banco de dados mongoDB[2] e expõe uma API para que seja possível acessar as informações via requisições HTTP. Já o *frontend* é a parte responsável por exibir os dados coletados e tratados no *backend* de uma forma organizada, facilitando a interação do usuário com o sistema. A arquitetura do OJTracker pode ser visualizada na Figura 20.

[2] <https://www.mongodb.com>

Figura 20 – Arquitetura do software



Fonte: Autor

Os juízes *online* Atcoder, Online Judge e Codeforces possuem APIs que podem ser consultadas para realizar a coleta das informações dos usuários diretamente. Já no caso do SPOJ e Codechef é necessário utilizar uma abordagem que permita buscar os dados diretamente do HTML da página em que as informações são exibidas nesses juízes. Essa abordagem é conhecida como *web scraping*.

O editor de texto utilizado para o desenvolvimento do código foi o Visual Studio Code[3], o qual possui integração com diversos *plugins* que auxiliaram no desenvolvimento do projeto. A ferramenta Git[4] foi utilizada para realizar o versionamento dos códigos, os quais foram hospedados em repositórios no Github[5]. Tanto o *backend* quanto o *frontend* foram desenvolvidos utilizando o sistema operacional Ubuntu 22.04 LTS (*Long-Term Support*).

O *backend* do projeto foi desenvolvido utilizando o *framework* Quarkus[6], na versão 2.15.1.Final. Se trata de uma ferramenta Java *open source* que têm como principal característica permitir que os desenvolvedores Java sejam capazes de criar aplicações *cloud*

[3] <https://code.visualstudio.com>
[4] <https://git-scm.com>
[5] <https://github.com>
[6] <https://quarkus.io>

native. O paradigma utilizado majoritariamente no desenvolvimento do *backend* foi a orientação a objetos.

Para a recomendação de problemas entre as plataformas suportadas foram implementados algoritmos que se baseiam tanto no nível do usuário dentro de cada juiz online como também na dificuldade de cada questão. Essa metodologia de recomendação foi escolhida pois dessa forma espera-se que o nível de habilidade do usuário seja o suficiente para resolver o problema recomendado.

Além disso, foi desenvolvido uma recomendação baseada em um intervalo (*range*) de dificuldade, onde o algoritmo de recomendação irá escolher aleatoriamente 5 questões dentro do intervalo de dificuldade indicado. Essa abordagem permite que o usuário ajuste manualmente o nível de dificuldade das questões.

O *frontend* foi construído utilizando a biblioteca ReactJS[7], na versão 17.0.1. O React é uma ferramenta que permite o desenvolvedor criar *SPAs (Single-Page Application)*, através da união de componentes simples e independentes que podem ser reutilizados para a criação de páginas complexas. Um dos principais problemas da utilização desta biblioteca é a possibilidade de vazamento de *CSS (Cascading Style Sheets)* entre os componentes, e, dentre as diversas soluções existentes, foi utilizada a de `css-modules`[8]. O paradigma utilizado majoritariamente no desenvolvimento do *frontend* foi o funcional.

Apesar de não seguir estritamente uma metodologia de desenvolvimento de software, o trabalho foi evoluindo semanalmente durante os semestres letivos, a partir do dia 21 de novembro de 2022 até o dia 06 de julho de 2023. O que foi realizado em cada semana pode ser observado nas Tabelas 3 e 4.

3.2.2 Escrita do texto

Para o desenvolvimento do texto foi utilizado o Overleaf, um editor LaTeX baseado em nuvem. Assim como o código-fonte, o documento de texto do trabalho também foi evoluindo semanalmente. O que foi realizado em cada semana pode ser observado no Quadro 3 e 4.

3.3 Ambiente de produção

Para realizar o *deploy* do *backend* foi utilizado o serviço do Heroku[9], já no *frontend* a ferramenta utilizada foi o Netlify[10]. O custo de utilização desses serviços podem ser observados na Tabela 5.

[7] <https://reactjs.org>
[8] <https://github.com/css-modules/css-modules>
[9] <https://heroku.com>
[10] <https://www.netlify.com>

Tabela 3 – Cronograma TCC1

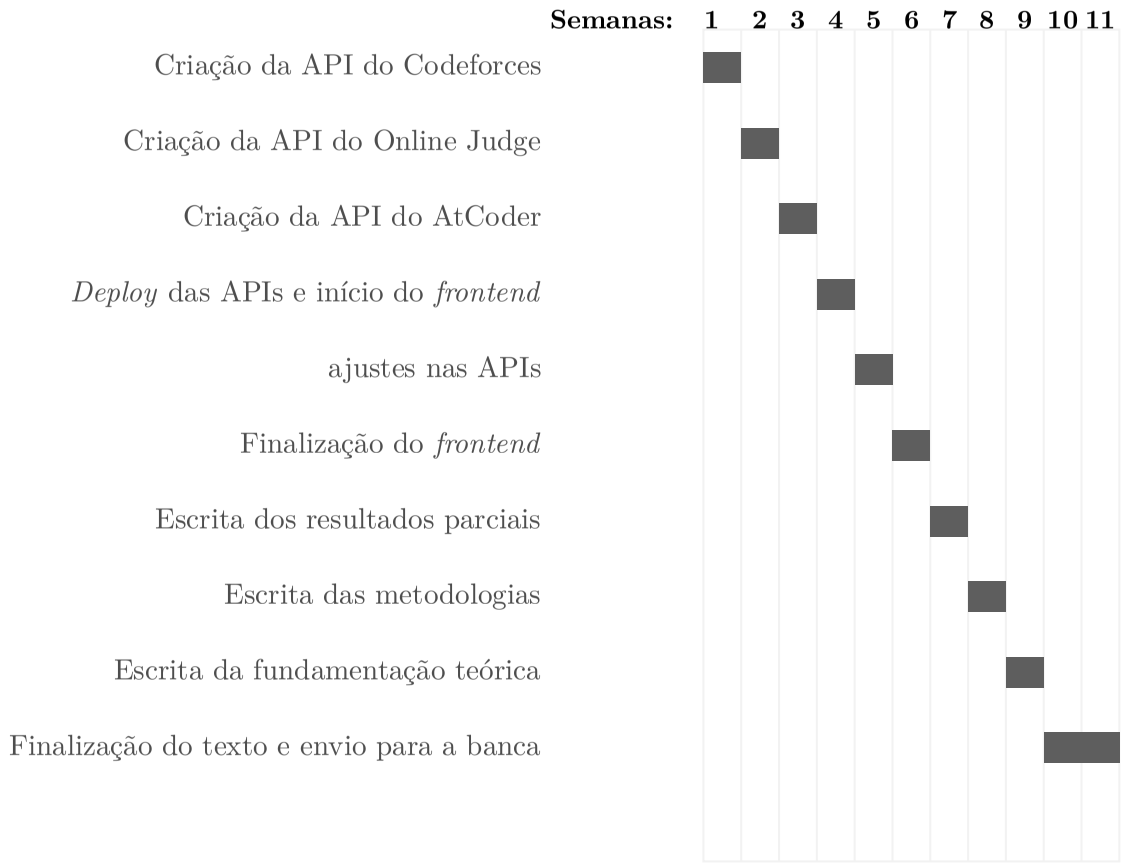


Tabela 4 – Cronograma TCC2

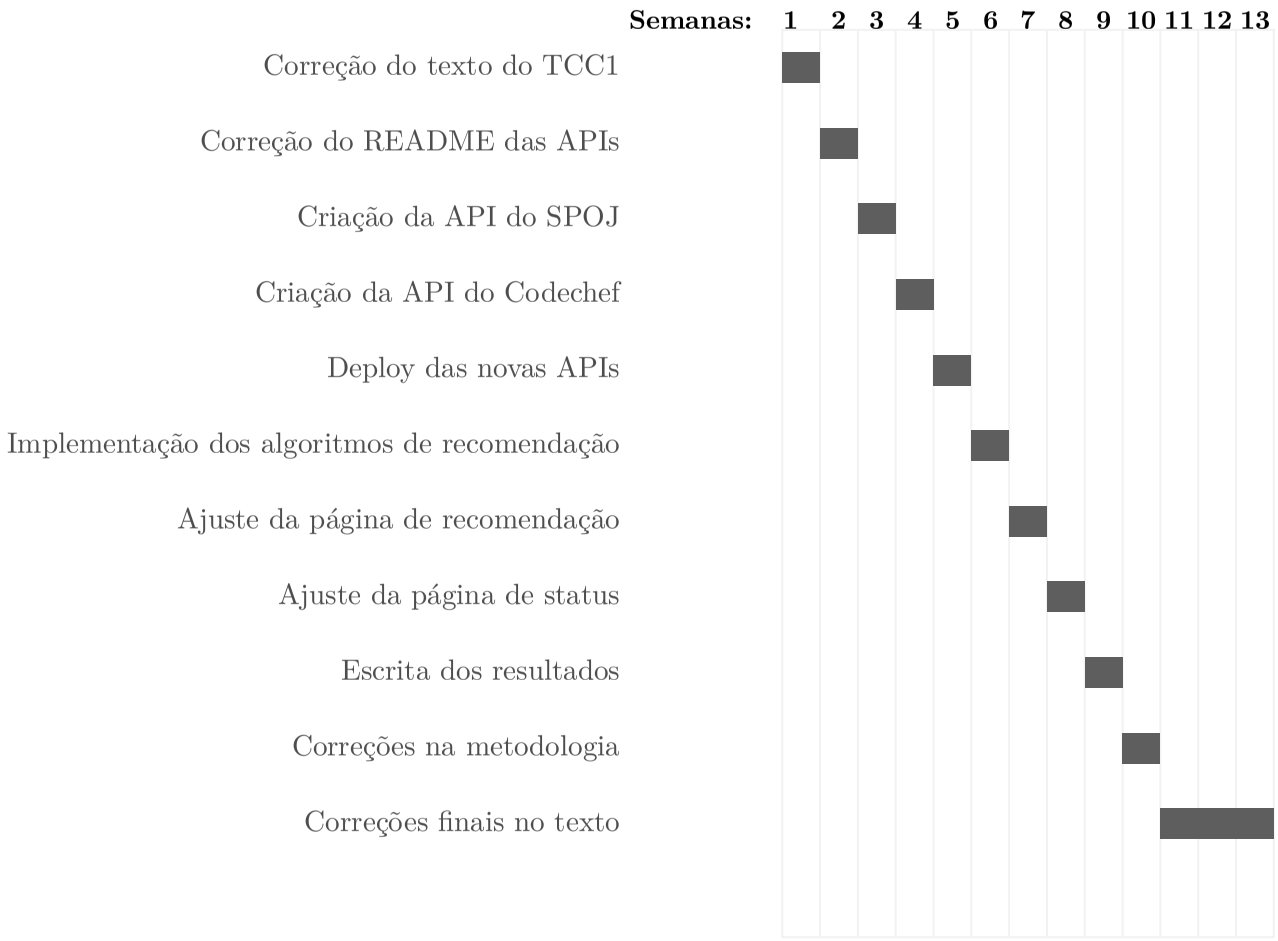


Tabela 5 – Preços dos serviços de *deploy*

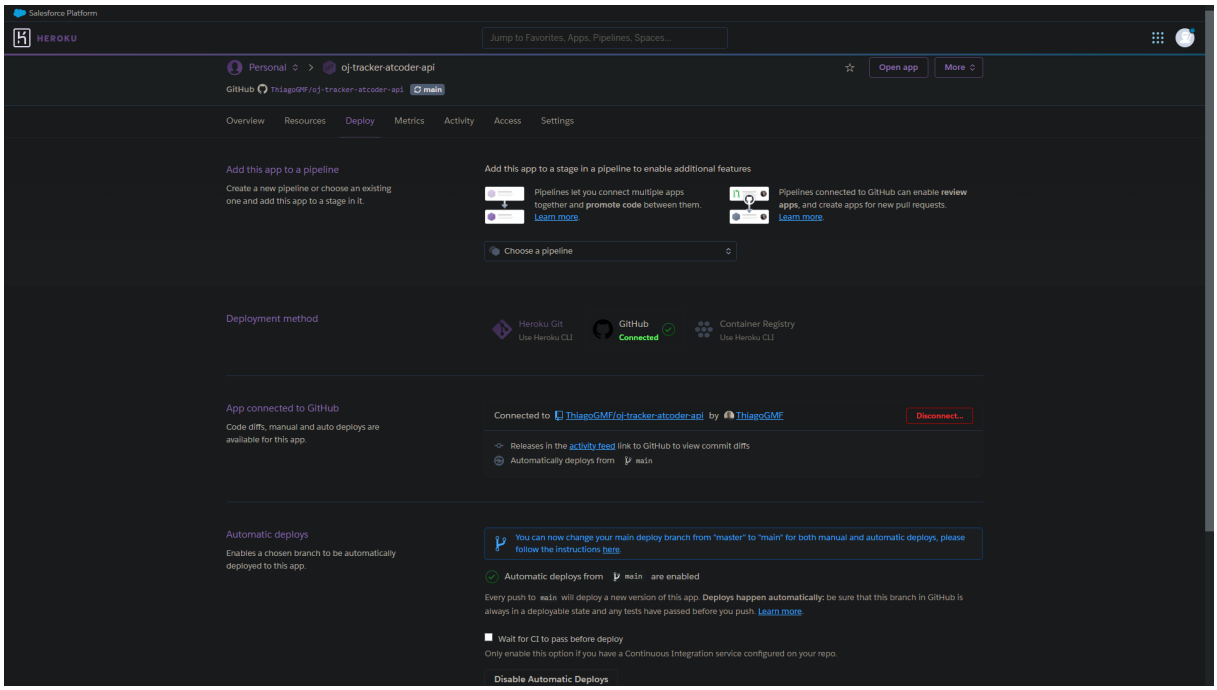
Nome	Preço (dólar)	Plataforma	Recursos (mensal)
Dyno Economic	5$	Heroku	1000 *dyno* horas
Starter	0$	Netlify	100GB de largura de banda
Starter	0$	Netlify	300 minutos de *build*

Fonte: Autor

O *deploy* no Heroku e no Netlify foram feitos através da integração com o Github, onde o as plataformas solicitam uma autenticação para ter acesso aos repositórios do usuário. É possível realizar o *deploy* tanto de forma manual quanto automática. No caso da automática, como está sendo utilizada neste trabalho, basta selecionar qual *branch* a plataforma deve observar. Assim, a cada mudança realizada uma nova versão da aplicação é gerada. As configurações de *deploy* para as APIs podem ser observadas na Figura 21, e as configurações do *frontend* na Figura 22.

Segundo a documentação[11] do Heroku, as aplicações implantadas na plataforma ficam em *dynos*, um contêiner Unix leve, seguro e virtualizado, que contêm os sistemas de arquivos necessários para executar o projeto. Quando uma nova versão da aplicação é gerada, os *dynos* antigos são apagados e novos são gerados com a versão atualizada.

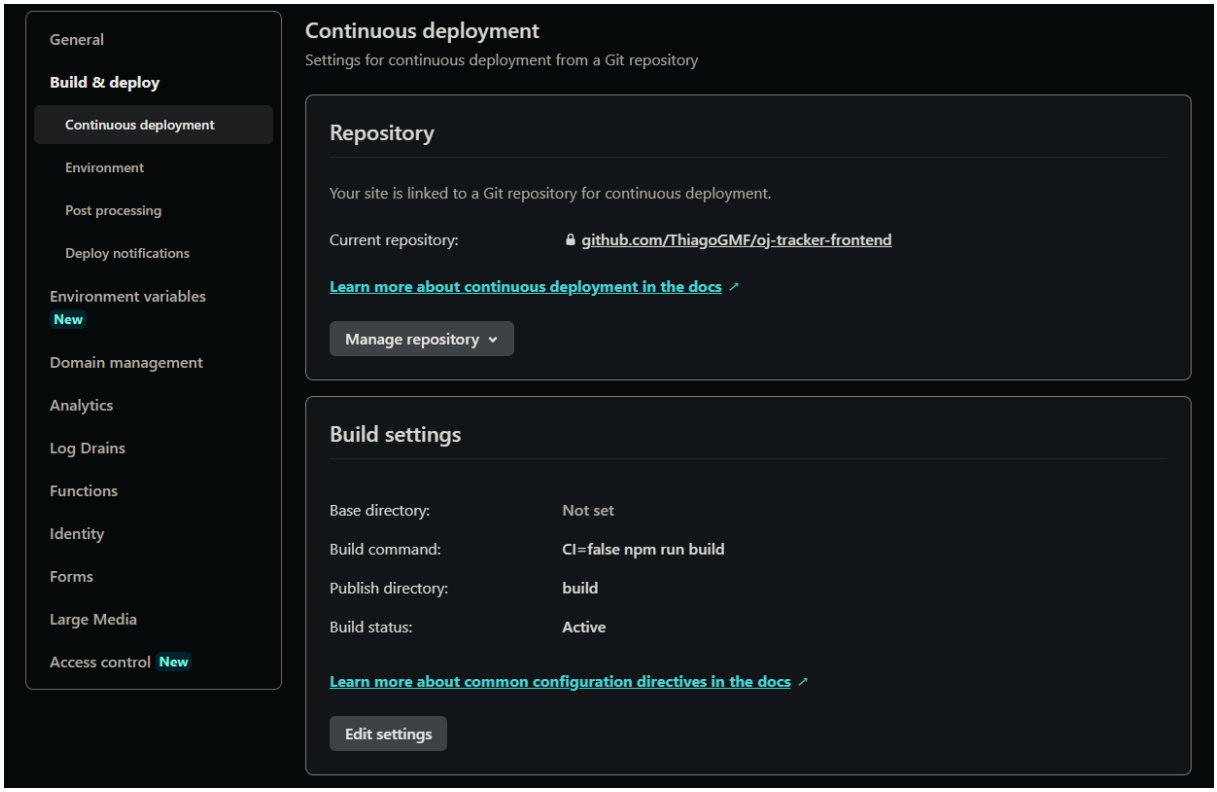
Figura 21 – Configurações de *deploy* do Heroku



Fonte: Autor

[11] <https://devcenter.heroku.com/articles>

Figura 22 – Configurações de *deploy* do Netlify



Fonte: Autor

A comunicação entre o *frontend* e o *backend* é feito através do protocolo HTTP *(Hypertext Transfer Protocol)*, onde são enviadas requisições, por meio do método *GET*, nos *endpoints* específicos. Cada requisição retorna um arquivo no formato JSON *(Javascript Object Notation)*. Para que a comunicação entre essas duas camadas aconteça, é necessário habilitar o CORS *(Cross-origin resource sharing)* no *backend*. No Quarkus isso é feito conforme mostra o trecho de Código 1. O CORS é um mecanismo que permite que aplicações de fora do domínio ao qual o conteúdo pertence possam recuperar o conteúdo solicitado.

Código 1 – Habilitando CORS no Quarkus

```
1 quarkus.http.cors=true
2 quarkus.http.cors.methods=GET
```

4 Resultados

Este capítulo tem como objetivo principal descrever os resultados obtidos e quais foram as principais dificuldades encontradas durante o desenvolvimento do trabalho, para isto, o capítulo encontra-se dividido em quatro seções.

A Seção 4.1 explica como foi realizada a coleta de dados dos usuários nos juízes, como são as respostas das APIs e como é feita a exibição no *frontend*. A Seção 4.2 demonstra como funciona o algoritmo de recomendação de problemas. A Seção 4.3 expõe os *feedbacks* coletados pelos usuários e a Seção 4.4 descreve algumas das dificuldades encontradas durante o trabalho.

4.1 Coleta de dados

Para realizar a extração dos dados de um usuário nas plataformas Codeforces, Online Judge e Atcoder foram criados as aplicacões oj-tracker-codeforces-api[1], oj-tracker-atcoder-api[2] e oj-tracker-uva-api[3], que realizam chamadas às APIs desses juízes. Estas chamadas permitem, por exemplo, extrair a lista de problemas, lista de submissões de um usuário e informações dos usuários, e tratar as informações recebidas para serem exibidas posteriormente no *frontend*.

Já para a coleta de dados nas plataformas Codechef e SPOJ foi necessário extrair as informações dos usuários diretamente do *HTML* das páginas. Assim como para as plataformas anteriores, foram criadas as aplicações oj-tracker-spoj-api[4] e oj-tracker-codechef-api[5]. Em algumas páginas o conteúdo exibido é carregado dinamicamente, dessa forma é preciso de um navegador para renderizar o conteúdo carregado pelo *script*.

Em ambos os casos, a busca pela base de problemas aumenta o tempo de resposta quando comparado com as plataformas que possuem API. Para reduzir esse tempo foi, criado um banco de dados para armazenar essas informações, de modo que, após a primeira requisição, a base é populada e então passa a ser utilizada como fonte para a busca.

A diferença entre os tempos de resposta para buscar os 3961 problemas hospedados no SPOJ e os 4255 problemas do Codechef, no dia 03 de julho de 2023, podem ser observados na Tabela 6.

[1] <https://oj-tracker-codeforces-api.herokuapp.com/docs/>
[2] <https://oj-tracker-atcoder-api.herokuapp.com/docs/>
[3] <https://oj-tracker-uva-api.herokuapp.com/docs/>
[4] <https://oj-tracker-spoj-api.herokuapp.com/docs/>
[5] <https://oj-tracker-codechef-api.herokuapp.com/docs/>

Tabela 6 – Tempo de requisição para o *endpoint* de proplemas no SPOJ e Codechef

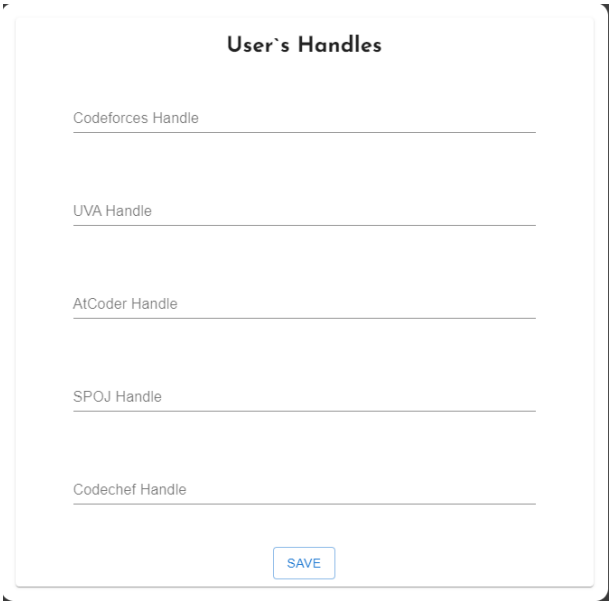
Juiz *Online*	**Método**	**Tempo (ms)**
SPOJ	Sem banco de dados e sem *tags*	61000
SPOJ	Sem banco de dados e com *tags*	7058129
SPOJ	Com banco de dados	49
Codechef	Sem banco de dados e sem *tags*	309600
Codechef	Sem banco de dados e com *tags*	22376840
Codechef	Com banco de dados	50

Fonte: Autor

Além disso, como a proposta do trabalho é produzir uma plataforma que unifica as informações de vários juízes, foi criado uma interface[6], onde o usuário pode visualizar os dados obtidos através de um navegador *web*.

Inicialmente, o usuário precisa inserir o seu *handle* no campo correspondente a cada uma das plataformas suportadas, para que, após validação, a aplicação possa buscar as informações por meio de requisições às APIs. A Figura 23 ilustra a tela inicial. A validação dos *handles* inseridos são realizados através do *endpoint* de informações do usuário: caso o usuário exista algo semelhante ao exibido no JSON 2 será retornado, caso contrário, a resposta é similar ao que é apresentado no trecho 3.

Figura 23 – Modal onde os *handles* são inseridos



Fonte: Autor

Código 2 – Informações de um usuário do Codeforces - API

[6] <https://oj-tracker.netlify.app>

```
{
  "status": "OK",
  "result": [
    {
      "handle": "edsomjr",
      "country": "Brazil",
      "city": "Brasília",
      "organization": "University of Brasilia",
      "contribution": 0,
      "rank": "candidate master",
      "rating": 1907,
      "maxRank": "candidate master",
      "maxRating": 1907,
      "lastOnlineTimeSeconds": 1687630014,
      "registrationTimeSeconds": 1443286475,
      "friendOfCount": 124,
      "avatar": "https://userpic.codeforces.org/no-avatar.jpg",
      "titlePhoto": "https://userpic.codeforces.org/no-title.jpg"
    }
  ]
}
```


Código 3 – Usuário não encontrado no Codeforces - API


```
{
  "status": "FAILED",
  "message": "handles: User with handle retainer2 not found"
}
```


A lista de submissões (veja o JSON 4) é a principal fonte de informações e é utilizada para gerar as estatísticas de um usuário, conforme mostrado no JSON 5. Através dela é possível saber, por exemplo, a quantidade de questões resolvidas e quais vereditos foram obtidos.

Essas informações são exibidas para o usuário conforme a Figura 24. Como a lista de submissões pode conter um grande volume de dados, foi implementado um sistema de paginação para que seja possível exibir as informações de forma parcial, de forma que não sobrecarregue o navegador durante a exibição da tabela (veja a Figura 25).

Código 4 – Lista de submissões de um usuário na plataforma Atcoder - API


```
{
  "status": "OK",
  "result": [
    {
      "submissionId": 2323455,
```

```
      "epochSecond": 1523125144,
      "problemId": "arc094_a",
      "contestId": "arc094",
      "userId": "edsomjr",
      "language": "C++14 (GCC 5.4.1)",
      "point": 300.0,
      "length": 1046,
      "verdict": "Accepted",
      "executionTime": 168
    },
    {
      "submissionId": 2359510,
      "epochSecond": 1523727296,
      "problemId": "arc095_a",
      "contestId": "arc095",
      "userId": "edsomjr",
      "language": "C++14 (GCC 5.4.1)",
      "point": 300.0,
      "length": 702,
      "verdict": "Accepted",
      "executionTime": 57
    },
    {
      "submissionId": 2359563,
      "epochSecond": 1523728560,
      "problemId": "arc095_b",
      "contestId": "arc095",
      "userId": "edsomjr",
      "language": "C++14 (GCC 5.4.1)",
      "point": 0.0,
      "length": 630,
      "verdict": "Wrong answer",
      "executionTime": 19
    },
    {
      "submissionId": 2359646,
      "epochSecond": 1523731123,
      "problemId": "arc095_b",
      "contestId": "arc095",
      "userId": "edsomjr",
      "language": "C++14 (GCC 5.4.1)",
      "point": 400.0,
      "length": 727,
      "verdict": "Accepted",
      "executionTime": 18
    }
  ],
```

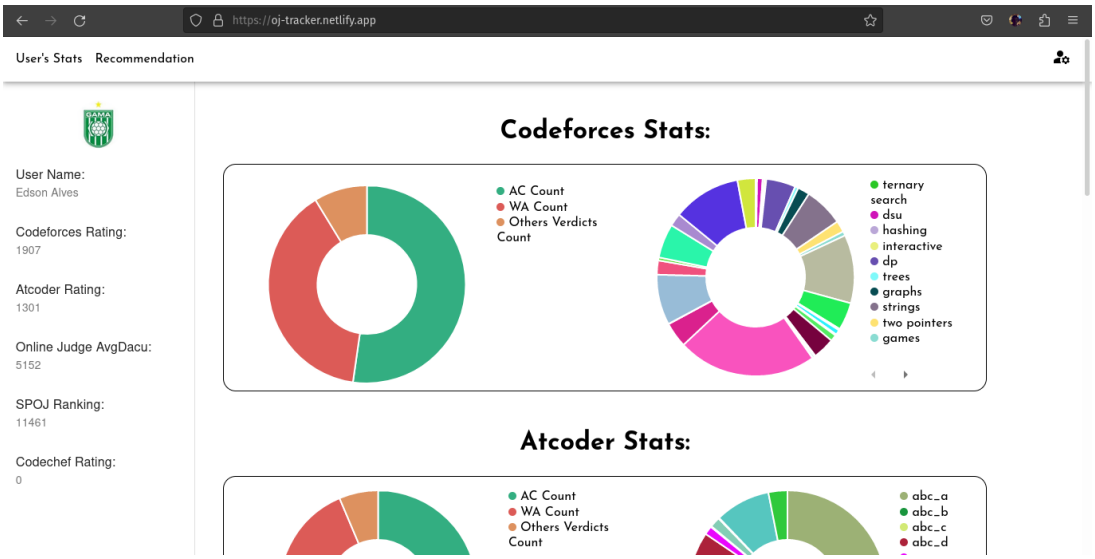
```
  "page": 1,
  "count": 4,
  "totalPages": 251
}
```

Código 5 – Estatísticas de um usuário - API


```
{
  "status": "OK",
  "result": [
    {
      "acCount": 875,
      "waCount": 653,
      "othersVerdictCount": 146,
      "tagsCount": {
        "ternary search": 5,
        "dsu": 18,
        "hashing": 6,
        "interactive": 6,
        "dp": 96,
        "trees": 11,
        "graphs": 39,
        "strings": 128,
        "two pointers": 37,
        "games": 14,
        "greedy": 223,
        "data structures": 90,
        "divide and conquer": 5,
        "bitmasks": 16,
        "matrices": 1,
        "probabilities": 3,
        "combinatorics": 21,
        "binary search": 71,
        "meet-in-the-middle": 4,
        "expression parsing": 6,
        "graph matchings": 2,
        "implementation": 454,
        "constructive algorithms": 82,
        "string suffix structures": 1,
        "brute force": 164,
        "dfs and similar": 46,
        "shortest paths": 10,
        "sortings": 111,
        "geometry": 42,
        "math": 222,
        "number theory": 59
      }
```

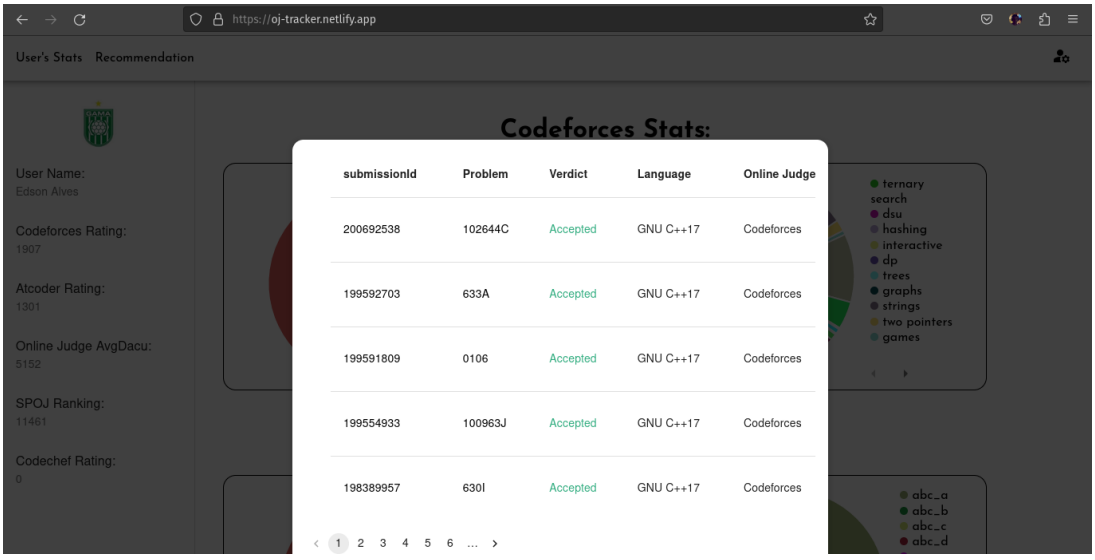
```
41       }
42    ]
43 }
```

Figura 24 – Página com as estatísticas do usuário



Fonte: Autor

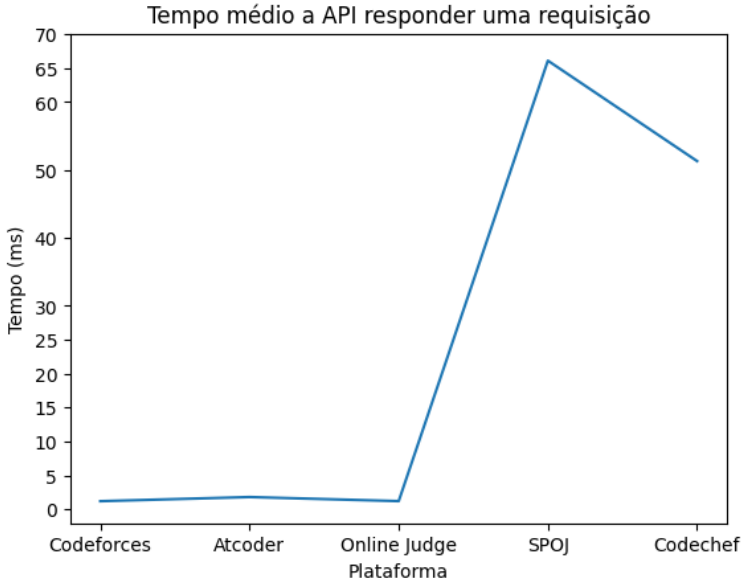
Figura 25 – Página com a lista de submissões de um usuário



Fonte: Autor

Para ter uma noção de quanto tempo, em média, as APIs demoram para responder uma requisição é necessário levar em consideração a quantidade de dados a ser retornada. Para as submissões, por exemplo, o tempo médio para coletar uma única tentativa pode ser observado na Figura 26. Os tempos exibidos foram obtidos realizando chamadas a *API* de cada plataforma e dividindo o tempo total pela quantidade de submissões de cada usuário.

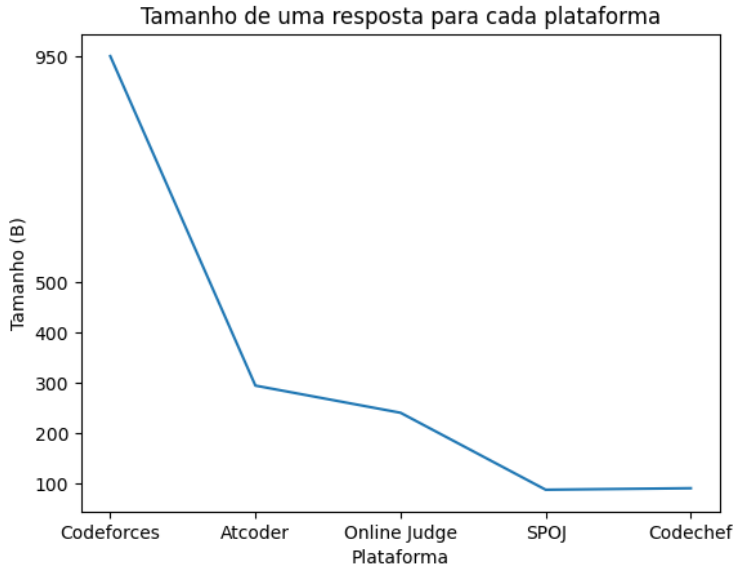
Figura 26 – Tempo médio para a API responder uma submissão ao endpoint de submissões



Fonte: Autor

A mesma ideia foi aplicada para se ter conhecimento do tamanho, em KB, de cada requisição com base na quantidade de submissões de um usuário. A Figura 27 demonstra o quanto cada tentativa aumenta a memória requerida.

Figura 27 – Tamanho médio, em KB, de cada submissão retornada pela API

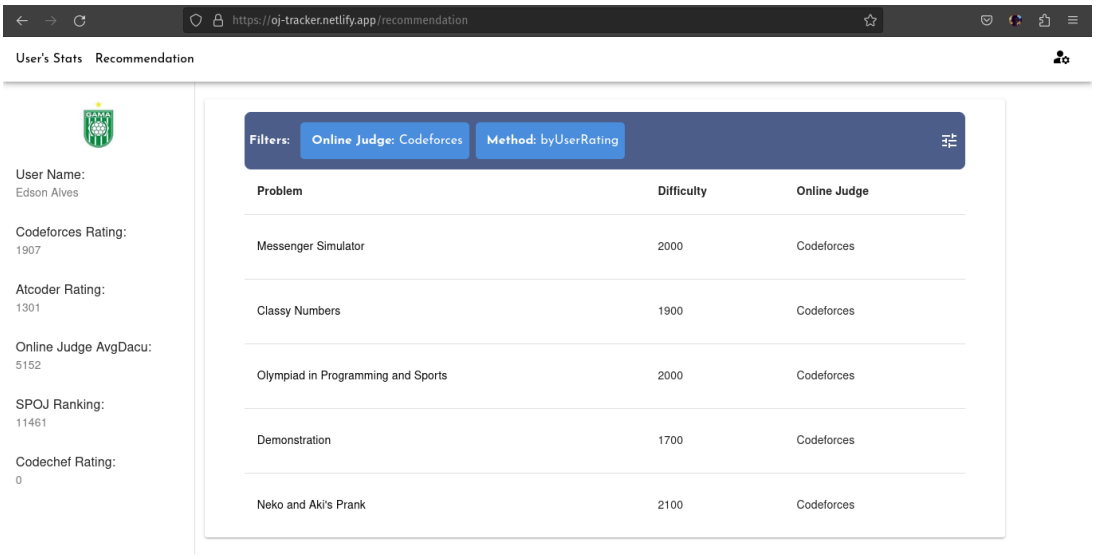


Fonte: Autor

4.2 Recomendação de problemas

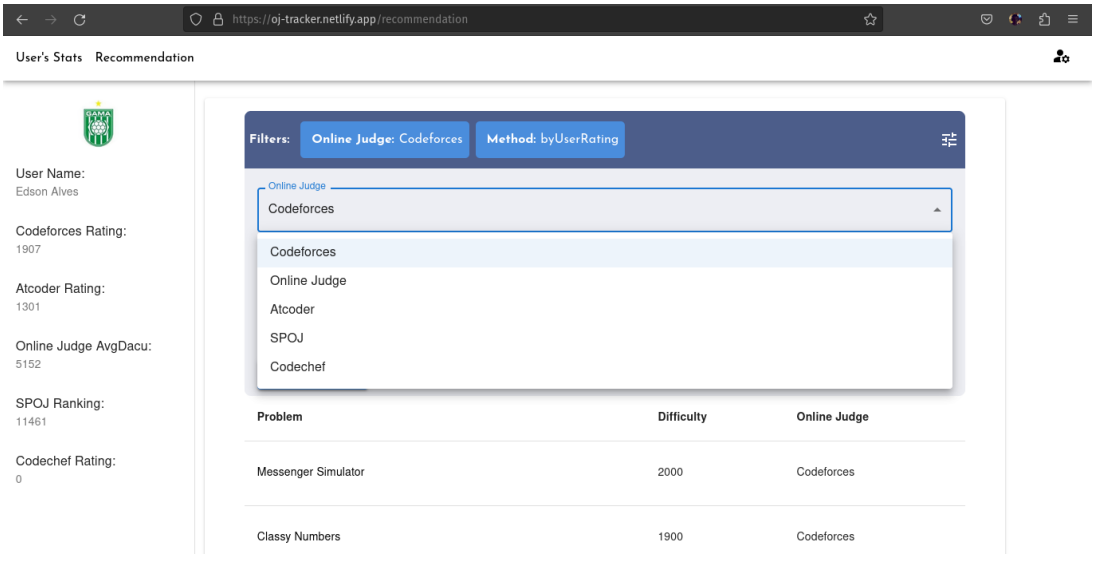
A parte de recomendação de problemas foi, inicialmente, feita de uma forma simples, havendo uma única abordagem, onde era extraída da lista completa de problemas de cada juiz as 5 primeiras que o usuário não resolveu por completo. Posteriormente, foram adicionadas novas abordagens para o sistema de recomendação. Essa sugestão é exibida ao usuário conforme mostra a Figura 28. Além disso, os filtros referentes ao juiz *online* e qual método de recomendação será utilizado pode ser observado nas Figuras 29 e 30.

Figura 28 – Página de recomendação de problemas.



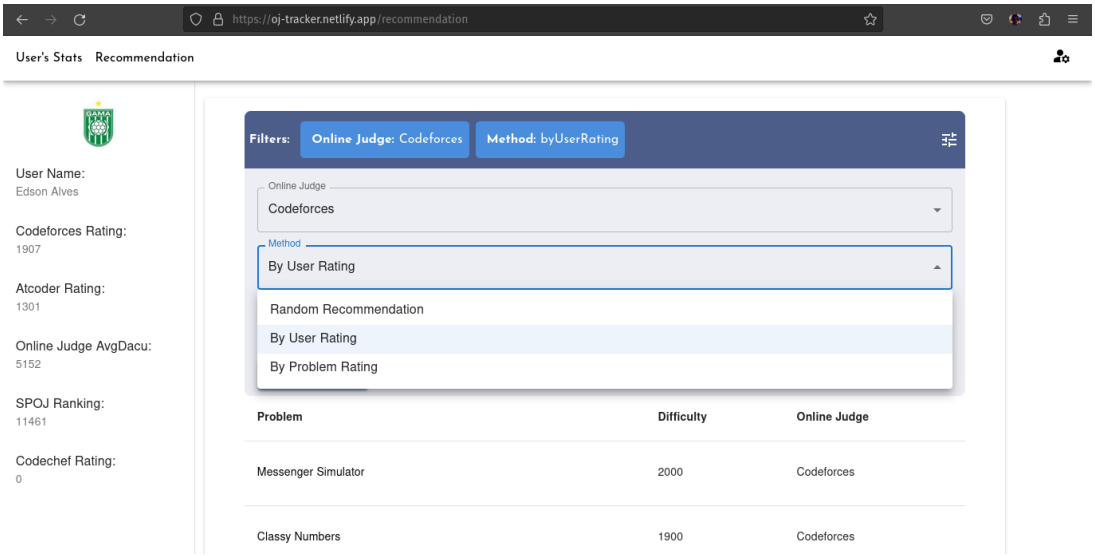
Fonte: Autor

Figura 29 – Página de recomendação de problemas - Filtro de juiz *online*.



Fonte: Autor

Figura 30 – Página de recomendação de problemas - Filtro de métodos.



Fonte: Autor

As sugestões de problemas foram criadas a partir da lista de questões (ver JSON 6) disponíveis em cada plataforma e a lista de submissões (JSON 4). A primeira abordagem, e também a mais simples, extrai 5 questões, que usuário não obteve o veredito *Accepted*, aleatórias da lista de problemas. Um exemplo da resposta para este *endpoint* pode ser visualizado no JSON 7.

Código 6 – Lista de problemas da plataforma *Online Judge* - API


```
{
  "status": "OK",
  "result": [
    {
      "id": 36,
      "problemNumber": 100,
      "title": "The 3n + 1 problem",
      "dacu": 103633,
      "bestRuntime": 0,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 6949,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 133649,
      "numberOfRFs": 0,
      "numberOfREs": 94727,
      "numberOfOLEs": 369,
      "numberOfTLEs": 74602,
      "numberOfMLEs": 5209,
      "numberOfWAs": 346552,
```

```
      "numberOfPEs": 6347,
      "numberOfACs": 242207,
      "problemTimeLimit": 3000,
      "status": 1
    },
    {
      "id": 37,
      "problemNumber": 101,
      "title": "The Blocks Problem",
      "dacu": 17269,
      "bestRuntime": 0,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 933,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 15745,
      "numberOfRFs": 0,
      "numberOfREs": 24961,
      "numberOfOLEs": 21,
      "numberOfTLEs": 12521,
      "numberOFMLEs": 200,
      "numberOfWAs": 28421,
      "numberOfPEs": 8072,
      "numberOfACs": 31170,
      "problemTimeLimit": 3000,
      "status": 1
    }
  ],
  "page": 1,
  "count": 2,
  "totalPages": 2483
}
```


Código 7 – Exemplo de resposta da API de recomendação de problemas no juiz AtCoder


```
{
  "status": "OK",
  "result": [
    {
      "problemId": "ttpc2019_f",
      "contestId": "ttpc2019",
      "problemIndex": "F",
      "name": "Road Construction",
      "title": "F. Road Construction",
      "point": 0.0,
```

```
      "solverCount": 185,
      "difficulty": 0
    },
    {
      "problemId": "kupc2021_d",
      "contestId": "kupc2021",
      "problemIndex": "D",
      "name": "Stones",
      "title": "D. Stones",
      "point": 0.0,
      "solverCount": 145,
      "difficulty": 0
    },
    {
      "problemId": "arc137_d",
      "contestId": "arc137",
      "problemIndex": "D",
      "name": "Prefix XORs",
      "title": "D. Prefix XORs",
      "point": 700.0,
      "solverCount": 843,
      "difficulty": 2191
    },
    {
      "problemId": "dwango2016qual_b",
      "contestId": "dwango2016-prelims",
      "problemIndex": "B",
      "name": "",
      "title": "B. ",
      "point": 0.0,
      "solverCount": 1901,
      "difficulty": 750
    },
    {
      "problemId": "arc121_c",
      "contestId": "arc121",
      "problemIndex": "C",
      "name": "Odd Even Sort",
      "title": "C. Odd Even Sort",
      "point": 500.0,
      "solverCount": 1258,
      "difficulty": 1856
    }
  ]
}
```

Outra metodologia para o algoritmo de recomendação de problemas leva em consideração a dificuldade das questões, independente de como a plataforma calcula essa informação, onde o usuário indica a menor e maior dificuldade desejadas e então o software busca questões dentro desse alcance que ainda não foram resolvidas. Um exemplo de resposta para essa recomendação pode ser observada no *JSON* 8.

Código 8 – Exemplo de resposta da API de recomendação de problemas por *range* de dificuldade


```
{
  "status": "OK",
  "result": [
    {
      "problemId": "abc203_e",
      "contestId": "abc203",
      "problemIndex": "E",
      "name": "White Pawn",
      "title": "E. White Pawn",
      "point": 500.0,
      "solverCount": 2006,
      "difficulty": 1750
    },
    {
      "problemId": "abc218_f",
      "contestId": "abc218",
      "problemIndex": "F",
      "name": "Blocked Roads",
      "title": "F. Blocked Roads",
      "point": 500.0,
      "solverCount": 2558,
      "difficulty": 1753
    },
    {
      "problemId": "abc017_3",
      "contestId": "abc017",
      "problemIndex": "C",
      "name": "",
      "title": "C. ",
      "point": 0.0,
      "solverCount": 2590,
      "difficulty": 1607
    },
    {
      "problemId": "abc150_d",
      "contestId": "abc150",
      "problemIndex": "D",
```

```
      "name": "Semi Common Multiple",
      "title": "D. Semi Common Multiple",
      "point": 400.0,
      "solverCount": 4883,
      "difficulty": 1534
    },
    {
      "problemId": "abc006_4",
      "contestId": "abc006",
      "problemIndex": "D",
      "name": "",
      "title": "D. ",
      "point": 0.0,
      "solverCount": 3476,
      "difficulty": 1696
    }
  ]
}
```


Além das recomendações anteriormente citadas também foi criada uma abordagem que leva em consideração o nível do usuário na plataforma buscada, pesquisando por questões que estão no nível do *rating* do usuário e questões que possuem um nível de dificuldade 5% e 10% acima e abaixo do *rating* do usuário.

Para as plataformas SPOJ e Online Judge, que não possuem ranqueamento, foi calculado o *dacu* (distinct accepted users) médio para realizar a recomendação. Nessa abordagem também é possível adicionar um filtro por *tags* nas plataformas que possuem essa funcionalidade (Codeforces, SPOJ e Codechef). Um exemplo de resposta para a chamada desse *endpoint* pode ser visualizado no JSON 9.

Código 9 – Exemplo de resposta da API de recomendação de problemas baseado no *rating* do usuário


```
{
  "status": "OK",
  "result": [
    {
      "problemId": "abc268_d",
      "contestId": "abc268",
      "problemIndex": "D",
      "name": "Unique Username",
      "title": "D. Unique Username",
      "point": 400.0,
      "solverCount": 3338,
      "difficulty": 1309
    },
```

```
    {
      "problemId": "arc030_2",
      "contestId": "arc030",
      "problemIndex": "B",
      "name": "",
      "title": "B. ",
      "point": 0.0,
      "solverCount": 1703,
      "difficulty": 1298
    },
    {
      "problemId": "abc212_e",
      "contestId": "abc212",
      "problemIndex": "E",
      "name": "Safety Journey",
      "title": "E. Safety Journey",
      "point": 500.0,
      "solverCount": 3790,
      "difficulty": 1410
    },
    {
      "problemId": "indeednow_2015_quala_3",
      "contestId": "indeednow-quala",
      "problemIndex": "C",
      "name": "",
      "title": "C. ",
      "point": 0.0,
      "solverCount": 1591,
      "difficulty": 1200
    },
    {
      "problemId": "agc007_b",
      "contestId": "agc007",
      "problemIndex": "B",
      "name": "Construct Sequences",
      "title": "B. Construct Sequences",
      "point": 400.0,
      "solverCount": 2455,
      "difficulty": 1611
    }
  ]
}
```


Por fim, na plataforma Online Judge, também foi criada uma recomendação baseada nas listas de recomendações disponíveis nos livros *Competitive Programming* nas versões de 1 a 3, onde também é possível filtrar por *tag*, que nesse caso são os capítulos dos livros. Um exemplo de resposta para a chamada desse *endpoint* pode ser visualizado

no JSON 10.

Código 10 – Exemplo de resposta da API de recomendação de problemas baseado nas questões recomendadas nos livros *Competitive Programming*


```
{
  "status": "OK",
  "result": [
    {
      "id": 863,
      "problemNumber": 922,
      "title": "Rectangle by the Ocean",
      "dacu": 98,
      "bestRuntime": 0,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 0,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 11,
      "numberOfRFs": 0,
      "numberOfREs": 14,
      "numberOfOLEs": 0,
      "numberOfTLEs": 15,
      "numberOFMLEs": 0,
      "numberOfWAs": 119,
      "numberOfPEs": 3,
      "numberOfACs": 142,
      "problemTimeLimit": 3000,
      "status": 1
    },
    {
      "id": 2916,
      "problemNumber": 11816,
      "title": "HST",
      "dacu": 149,
      "bestRuntime": 10,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 2,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 44,
      "numberOfRFs": 0,
      "numberOfREs": 54,
      "numberOfOLEs": 0,
      "numberOfTLEs": 58,
```

```
      "numberOFMLEs": 0,
      "numberOfWAs": 773,
      "numberOfPEs": 6,
      "numberOfACs": 264,
      "problemTimeLimit": 3000,
      "status": 1
    },
    {
      "id": 2613,
      "problemNumber": 11566,
      "title": "Let\u0027s Yum Cha!",
      "dacu": 879,
      "bestRuntime": 0,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 5,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 193,
      "numberOfRFs": 0,
      "numberOfREs": 370,
      "numberOfOLEs": 0,
      "numberOfTLEs": 319,
      "numberOFMLEs": 0,
      "numberOfWAs": 3083,
      "numberOfPEs": 9,
      "numberOfACs": 1707,
      "problemTimeLimit": 1000,
      "status": 1
    },
    {
      "id": 2338,
      "problemNumber": 11353,
      "title": "A Different Kind of Sorting",
      "dacu": 811,
      "bestRuntime": 10,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 3,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 82,
      "numberOfRFs": 0,
      "numberOfREs": 149,
      "numberOfOLEs": 0,
      "numberOfTLEs": 126,
      "numberOFMLEs": 0,
```

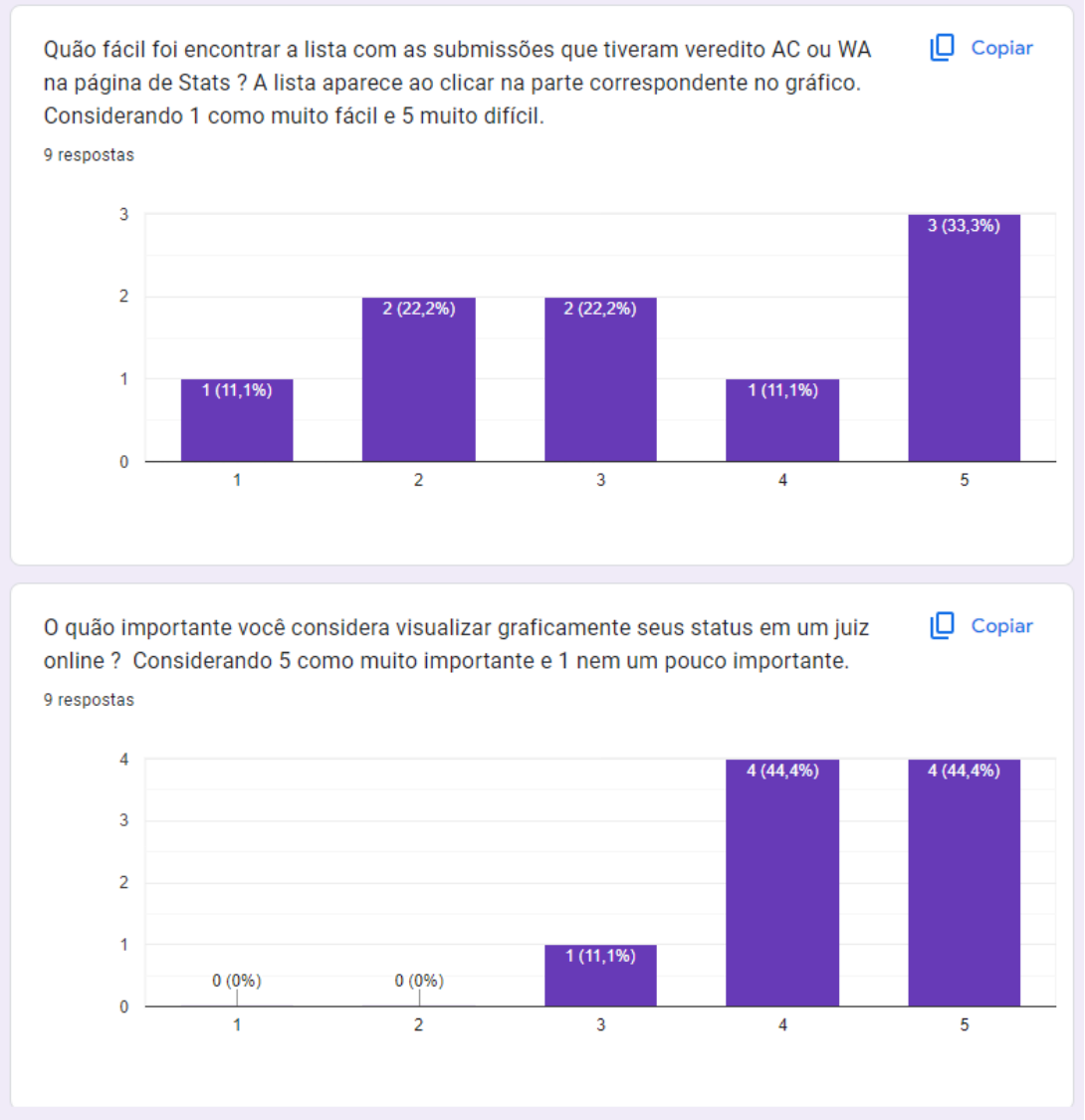
```
      "numberOfWAs": 735,
      "numberOfPEs": 15,
      "numberOfACs": 1362,
      "problemTimeLimit": 5000,
      "status": 1
    },
    {
      "id": 3714,
      "problemNumber": 12293,
      "title": "Box Game",
      "dacu": 1336,
      "bestRuntime": 0,
      "bestMemoryUsage": 1000000000,
      "numberOfNoVerdictGiven": 0,
      "numberOfSubmissionError": 15,
      "numberOfCannotBeJudged": 0,
      "numberOfIQs": 0,
      "numberOfCEs": 224,
      "numberOfRFs": 0,
      "numberOfREs": 104,
      "numberOfOLEs": 0,
      "numberOfTLEs": 153,
      "numberOFMLEs": 0,
      "numberOfWAs": 1698,
      "numberOfPEs": 6,
      "numberOfACs": 2748,
      "problemTimeLimit": 1000,
      "status": 1
    }
  ]
}
```


4.3 Validação

Para realizar a validação do OJTracker em relação à usabilidade da interface e as funcionalidades propostas pelo projeto, foi criado um formulário para que os usuários pudessem fornecer sua opinião após usarem o *software*.

Esse formulário foi compartilhado, juntamente com o link para acessar a aplicação, em grupos de estudos para maratonas de competições da Universidade de Brasília - Gama. Espera-se que o nível dos usuários seja tanto de iniciantes em maratonas como também competidores experientes. Ao todo o formulário obteve apenas 9 respostas dos usuários. As respostas coletadas podem ser observadas nas Figuras: 31, 32, 33, 34, 35, 36, 37 e 38.

Figura 31 – Respostas coletadas durante validação - Parte 1



Fonte: Autor

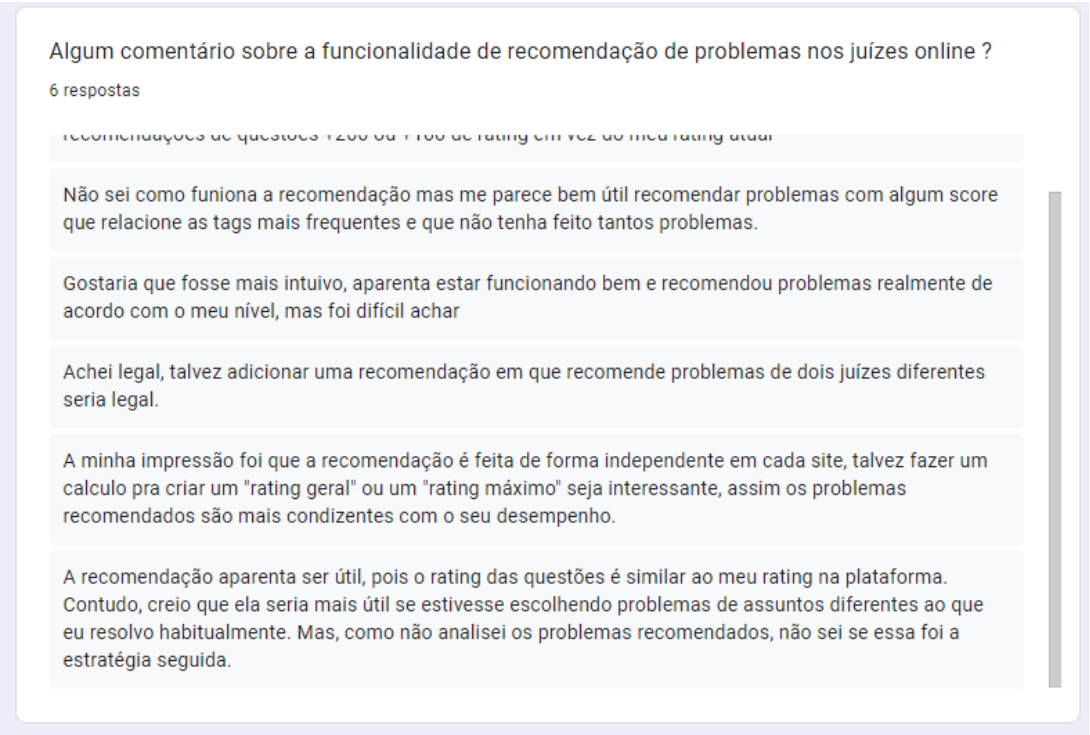
Figura 32 – Respostas coletadas durante validação - Parte 2

Fonte: Autor

Figura 33 – Respostas coletadas durante validação - Parte 3

Fonte: Autor

Figura 34 – Respostas coletadas durante validação - Parte 4

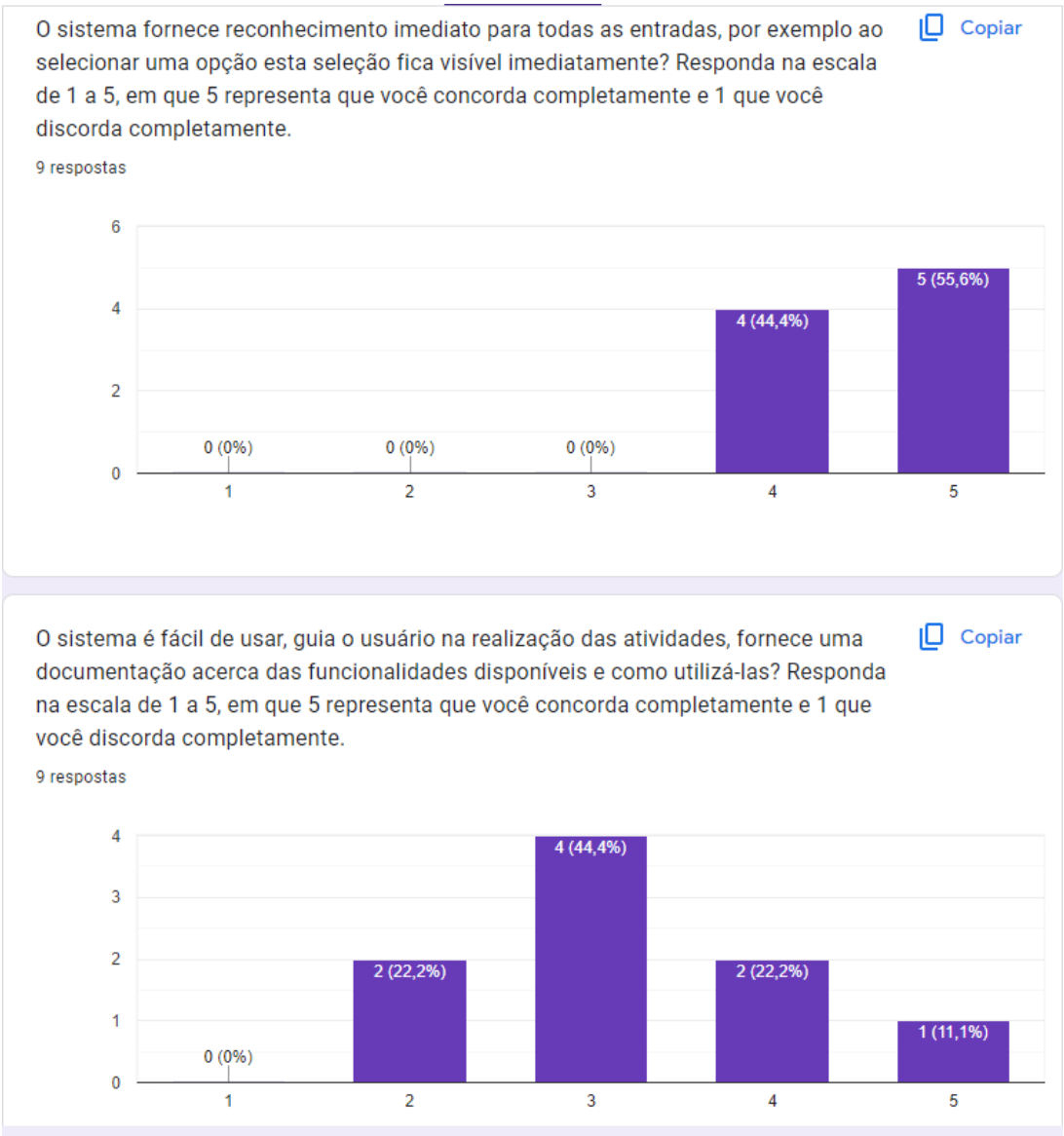


Fonte: Autor

Figura 35 – Respostas coletadas durante validação - Parte 5

Fonte: Autor

Figura 36 – Respostas coletadas durante validação - Parte 6

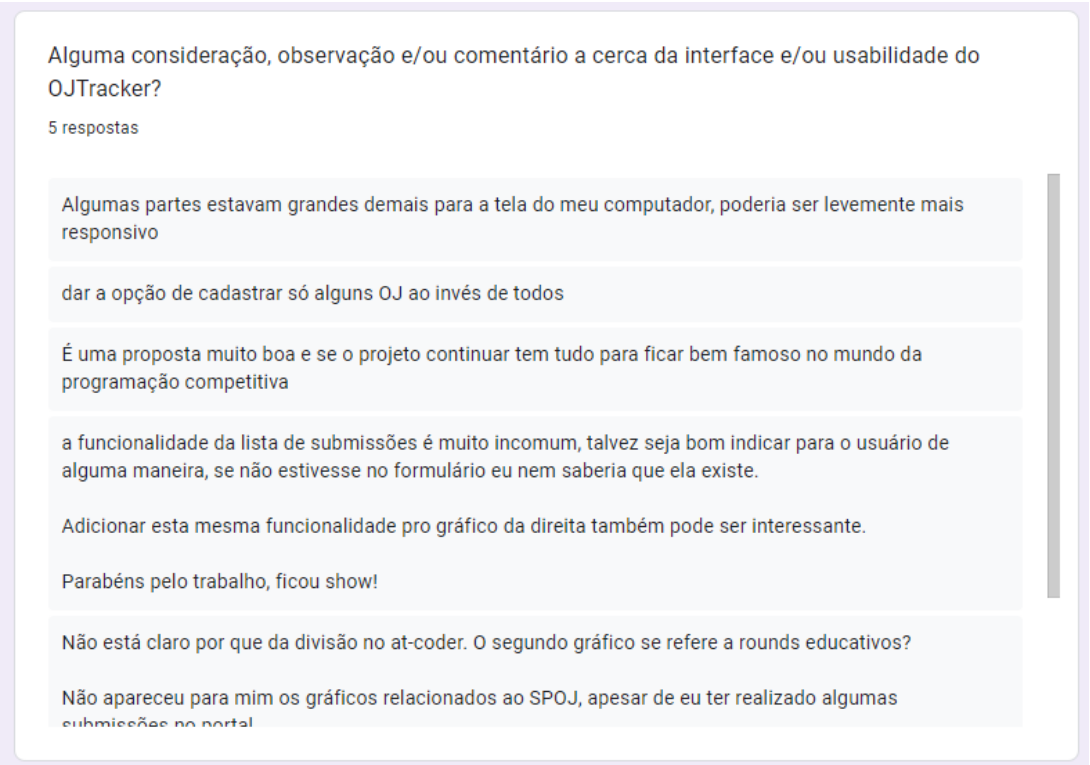


Fonte: Autor

Figura 37 – Respostas coletadas durante validação - Parte 7

Fonte: Autor

Figura 38 – Respostas coletadas durante validação - Parte 8



Fonte: Autor

Essas respostas indicam que o *frontend* do OJTracker ainda conta com alguns problemas tanto em responsividade, quando acessado em diferentes tamanhos de telas, como também na experiência do usuário, uma vez que não fica totalmente claro para o usuário onde algumas das funcionalidades se encontram.

Em relação à *feature* de recomendação de problemas, apesar de a maioria dos usuários que testaram a aplicação terem achado útil ter algum tipo de recomendação, foram apontados alguns problemas. Por exemplo, indicar questões com base no *ranking* do usuário em cada plataforma separadamente pode gerar uma sugestão com problemas fora do nível real do competidor. Tratando-se das estatísticas do usuário, a funcionalidade também foi bem aceita pelos usuários apesar das sugestões de melhorias propostas para a interface no geral.

4.4 Dificuldades encontradas

A primeira dificuldade encontrada foi o retorno das APIs, que não contém exatamente o que seria utilizado nesse trabalho, o que motivou a criação de APIs, que fazem o tratamento dos dados obtidos no lado do servidor. Além disso, nas plataformas AtCoder e Online Judge, em algumas ocasiões, foi necessário consultar mais de um *endpoint* para se obter as informações que são obtidas utilizando-se apenas um no Codeforces, o que aumenta o tempo de resposta da consulta.

Outro problema encontrado foi a mudança de preços nos serviços do Heroku, que deixaram de serem gratuitos. Entretanto, a plataforma disponibilizou créditos entre os usuários que tinham vínculo comprovado com alguma universidade, o que permitiu continuar com a solução inicial para o *deploy* do *backend*. Além disso, outra dificuldade encontrada foi realizar a configuração da aplicação do Codechef, no qual é necessário um navegador para renderizar a página. Para contornar esse problema, a solução encontrada foi popular o banco de dados a partir da aplicação local.

5 Considerações Finais

Este trabalho teve como objetivo a criação de um *software* que extrai dados em 5 juízes *online*: Codeforces, AtCoder, Online Judge, SPOJ e Codechef, e exibe essas informações a partir de uma interface centralizada. Este software, chamado OJTracker, também recomenda problemas a serem resolvidos nessas plataformas, utilizando diferentes métodos.

Ambas as funcionalidades foram implementadas, apesar da necessidade de melhoria na usabilidade da interface. Além disso, a realização de melhorias no *backend* agregariam mais valor ao projeto, como por exemplo, um conjunto de testes automatizados. Esse incremento aumentaria a confiabilidade do código-fonte além de facilitar a manutenção e futuras contribuições ao *software*.

O OJTracker possui um *deploy* automático tanto no *backend* como no *frontend*, o que permitiu coletar *feedbacks* por parte dos usuários após o uso. Com esse retorno dos maratonistas, foi possível identificar alguns dos problemas na usabilidade e também algumas sugestões de melhorias nas funcionalidades entregues, assim como validar a importância e necessidade da proposta deste trabalho.

Tratando-se da execução local do projeto, também se faz necessário alguns aprimoramentos, principalmente tratando-se das aplicações referentes aos juízes SPOJ e Codechef, uma vez que popular a base de dados com as *tags* das questões disponíveis nessas plataformas pode levar horas, já que é preciso realizar a leitura do HTML da página de cada questão, individualmente.

A divisão do projeto utilizando APIs permite que futuras melhorias no projeto sejam feitas de forma separada. Caso haja uma ampliação da cobertura de juízes *online* não há obrigatoriedade em utilizar os mesmos *frameworks* que foram utilizado. Já para o *frontend* as melhorias futuras exigirão conhecimentos das ferramentas utilizadas.

5.1 Trabalhos Futuros

Futuros trabalhos podem envolver tanto melhorias na interface, criação de testes automatizados no *backend*, no *frontend* ou *end-to-end*, ampliação dos juízes *online* mapeados, melhorias e/ou refatoração do sistema de recomendação de problemas, assim como acréscimos de novas funcionalidades. Todos estes pontos podem servir de guia para novos trabalhos de conclusão de curso voltados para a evolução do OJTracker.

Referências

AUDRITO., G. et al. Recommending tasks in online judges. *International Conference in Methodologies and intelligent Systems for Techhnology Enhanced Learning*, 2019. Citado na página 33.

BECARES, J. Feedback en jueces online. *Universidad Complutense de Madrid*, 2017. Citado 3 vezes nas páginas 23, 24 e 25.

BEECROWD. *Beecrowd*. 2023. Disponível em: <https://www.beecrowd.com.br>. Citado 3 vezes nas páginas 29, 30 e 31.

BEZ, J.; TONIN, N.; RODEGHERI, P. Uri online judge academic: A tool for algorithms and programming classes. *ICCSE*, 2014. Citado na página 29.

CODEFORCES. *Codeforces*. 2023. Disponível em: <https://codeforces.com>. Citado na página 26.

FANTOZZI, P.; LAURA, L. Recommending tasks in online judges using autoencoder neural networks. *Olympiads in Informatics*, 2020. Disponível em: <https://doi.org/10.15388/ioi.2020.05>. Citado 3 vezes nas páginas 23, 25 e 32.

ISINKAYE, F.; FOLAJIMI, Y.; OJOKOH, B. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 2015. Citado 2 vezes nas páginas 32 e 33.

LAAKSONEN, A. *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*. [S.l.]: Springer, 2017. 283 p. Citado na página 25.

LAAKSONEN, A. *Competitive Programmer's Handbook*. [S.l.: s.n.], 2018. 296 p. Citado na página 24.

LADDERS, A. *A2OJ Ladders*. 2023. Disponível em: <https://earthshakira.github.io/a2oj-clientside/server/Ladders.html>. Citado 2 vezes nas páginas 19 e 20.

OLIVEIRA, R. ANIDO de; MENDERICO, R. Brazilian olympiad in informatics. *Olympiads in Informatics*, 2007. Citado na página 32.

ONLINEJUDGE. *OnlineJudge*. 2023. Disponível em: <https://onlinejudge.org>. Citado na página 28.

PROBLEMS, A. *AtCoder Problems*. 2023. Disponível em: <https://kenkoooo.com/atcoder/>. Citado 2 vezes nas páginas 21 e 22.

REVILLA, M.; MANZOOR, S.; LIU, R. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, 2008. Citado 2 vezes nas páginas 25 e 27.

RIBEIRO, P.; GUERREIRO, P. Early introduction of competitive programming. *Olympiads in Informatics*, 2008. Citado 3 vezes nas páginas 13, 23 e 24.

TOLEDO, R.; MOTA, Y.; MARTÍNEZ, L. A recommender system for programming online judges using fuzzy information modeling. *Informatics*, 2018. Disponível em: <https://doi.org/10.3390/informatics5020017>. Citado 2 vezes nas páginas 23 e 33.

UHUNT. *uHunt*. 2023. Disponível em: <https://uhunt.onlinejudge.org>. Citado 2 vezes nas páginas 17 e 18.