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

Gamma Online Judge: projeto de criação de uma plataforma para o armazenamento das questões das Maratonas UnB de programação

Autor: Gustavo Marques Lima
Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF
2021

Gustavo Marques Lima

Gamma Online Judge: projeto de criação de uma plataforma para o armazenamento das questões das Maratonas UnB de programação

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: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2021

Gustavo Marques Lima

Gamma Online Judge: projeto de criação de uma plataforma para o armazenamento das questões das Maratonas UnB de programação/ Gustavo Marques Lima. – Brasília, DF, 2021-

75 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2021.

1. projeto. 2. Online Judge. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Gamma Online Judge: projeto de criação de uma plataforma para o armazenamento das questões das Maratonas UnB de programação

CDU 02:141:005.6

Gustavo Marques Lima

Gamma Online Judge: projeto de criação de uma plataforma para o armazenamento das questões das Maratonas UnB de programação

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, 30 de setembro de 2022 – Data da apresentação do trabalho:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Bruno César Ribas
Convidado 1

Prof. Daniel Saad Nogueira Nunes
Convidado 2

Brasília, DF
2021

Agradecimentos

Agradeço aos professores Bruno Ribas e Daniel Saad por participarem da banca de avaliação; agradeço ao professor Edson pelas orientações durante o semestre e a minha esposa que me apoiou bastante durante a escrita do texto, e que me encorajou a continuar até o fim.

"Já fui jovem e agora sou velho,
mas nunca vi o justo desamparado
nem seus filhos mendigando o pão
(Bíblia Sagrada, Salmos 37, 25)

Resumo

O *Gamma Online Judge* é um juiz online desenvolvido visando reunir os problemas das maratonas UnB de programação. O projeto armazena os problemas e as separa por evento, possibilitando que usuários testem suas soluções para os problemas. Separados em micro-serviços, o sistema possui uma interface, uma API e um juiz eletrônico.

Palavras-chave: Online judge. Maratonas de programação. Gamma Online Judge.

Abstract

The Gamma Online Judge is an online judge designed to bring together the problems of the maratonas UnB de programação. The project stores the problems and separates them by event, allowing users to test their solutions to problems. Separated into microservices, the system has an interface, an API and an electronic judge.

Key-words: Online judge. Programming contests. Gamma Online Judge.

Lista de ilustrações

Lista de tabelas

Lista de Quadros

Lista de abreviaturas e siglas

HTML	*HyperText Markup Language*
TLE	*Time Limit Exceeded*
MLE	*Memory Limit Exceeded*
WA	*Wrong Answer*
UnB	Universidade de Brasília
OBI	Olimpíada brasileira de informática
GOJ	*Gamma Online Judge*
NPM	*Node Package Manager*
ICPC	*International Collegiate Programming Contest*
API	*Application Programing Interface*
LINF	Laboratório de Informática
UFU	Universidade Federal de Uberlândia
USP	Universidade de São Paulo
PUC-GO	Pontifícia Universidade Católica de Goiás

Sumário

1 Introdução

Este capítulo aborda as justificativas e motivações do trabalho. Sua estrutura é composta com pelas seções: Contextualização, que traz um contexto geral para o entendimento do trabalho; Justificativa/Motivação, que aborda a justificativa e motivação; Objetivos, que apresenta os objetivos gerais e específicos; e, por fim, Estrutura do Trabalho, que apresenta a estrutura do trabalho.

1.1 Contextualização

Maratonas de programação são competições onde os participantes utilizam conhecimentos de ciência da computação para resolução de problemas. São grupos de competidores, ou competidores individuais, disputando para uma melhor colocação na competição. As equipes recebem problemas e devem resolvê-los utilizando ciência da computação, os problemas são resolvidos utilizando alguma linguagem de programação e julgados por juízes eletrônicos. A colocação dos participantes depende do formato da maratona; um dos formatos de maratona é o International Collegiate Programming Contest (ICPC), onde os participantes competem em trios. Esse formato utiliza o número de problemas resolvidos e o tempo como critérios de colocação, assim, os competidores devem resolver o maior número de problemas, o mais rápido possível.

A Maratona UnB de Programação é uma maratona de programação que segue o formato de maratona ICPC, realizada desde 2013 e acumula até hoje 10 edições. Seu público varia, não se limitando apenas aos estudantes da UnB. Além disso, o nível dos competidores é abrangente, contendo estudantes iniciantes em programação e estudantes mais experientes.

Outro exemplo é a Olimpíada brasileira de Informática. Diferentemente da Maratona UnB de Programação, ela é voltada para o público iniciante em programação e a participação é individual. Os níveis dos competidores podem variar entre estudantes do ensino fundamental, ensino médio, técnico ou início de graduação (OBI2021, 2021).

As Maratonas UnB de Programação foram realizadas, na maioria das edições, sem auxílio de plataformas onlines, dificultando os competidores a encontrar problemas de edições passadas, buscar resolução e tutoriais para as questões e, o mais importante, uma plataforma para testar seus códigos – processo fundamental para aprendizagem e o desenvolvimento dos competidores.

Existem vários juízes onlines que fazem o trabalho de armazenar e julgar proble-

mas. No Brasil, existem juizes populares como: Beecrowd[1], Codeforces[2] e Online Judge[3]. Dentre os citados, é possível cadastrar novas questões utilizando o Codeforces ou o Beecrowd, sendo o Beecrowd o único em português. A indexação dos problemas para buscas, separação em eventos e escrita de tutoriais não é flexível e, em alguns casos, nem é possível. Além disso, alguns desses juízes possuem problemas de usabilidade e um *feedback* não assertivo acerca do resultado quando o público são programadores iniciantes, questão apontada também por Francisco, Júnior e Ambrósio (2016, p. 15):

> [...] Talvez devido à falta de maturidade e à origem baseada em competições, os sistemas apresentam problemas de usabilidade. O processo de submissão de programas ainda apresenta problemas, e o *feedback* não é suficiente para que alunos consigam corrigir muitos dos erros.

Conforme se tem observado, a utilização de ferramentas existentes atualmente para o cadastro de questões das edições das Maratonas de programação UnB dificulta a indexação dos problemas e limita a flexibilidade. Além disso, é interessante que as questões dos eventos fiquem organizadas em um único local. Essa ideia se assemelha ao site da Olimpíada brasileira de Informática[4], que concentra as questões e provas das edições anteriores do evento.

1.2 Justificativa/Motivação

Os principais juízes online não possuem um suporte flexível para a indexação de questões das Maratonas UnB de Programação. Além disso, seria interessante a concentração das questões desses eventos em um site próprio, facilitando a busca por problemas.

Atualmente os juízes online tem um formato voltado para competidores de programação. Devido a isso, o retorno das plataformas pode não ser suficiente para alunos iniciantes encontrarem erros presentes em seus códigos (FRANCISCO; JÚNIOR; AMBRÓSIO, 2016), um fator desmotivante para quem não é experiente e busca uma melhora em programação.

1.3 Objetivos

Diante do exposto, torna-se objetivo geral do trabalho criar o *Gamma Online Judge* (GOJ), uma plataforma para armazenar e julgar questões das edições anteriores das Maratonas UnB de programação.

[1] Disponível em: <https://www.beecrowd.com.br/>.
[2] Disponível em: <https://codeforces.com/>.
[3] Disponível em: <https://onlinejudge.org/>.
[4] Disponível em: <https://olimpiada.ic.unicamp.br/>.

Os objetivos específicos deste trabalho são:

1. armazenar questões anteriores das maratonas de programação UnB;
2. proporcionar ferramentas de auxílio para programadores iniciantes, como tutoriais e dicas para resolução dos problemas;
3. julgar as questões e entregar uma resposta aos usuários acerca dos códigos enviados à plataforma e;
4. organizar as questões por eventos possibilitando buscas por questões e eventos na plataforma.

1.4 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma: na Fundamentação Teórica são apresentados os conceitos fundamentais para o entendimento do desenvolvimento deste trabalho; na Metodologia são definidos os requisitos e a arquitetura do trabalho; os Resultados com resultados do desenvolvimento do GOJ; e por fim, a Conclusão, que apresenta as considerações finais do trabalho.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos fundamentais para o embasamento da reflexão proposta, serão abordados os conceitos a respeito de juízes eletrônicos e juízes online, maratonas de programação e tecnologias utilizadas na arquitetura do GOJ. O capítulo está dividido nas seguintes seções: Maratonas de programação, onde são abordados os conceitos de maratonas de programação e programação competitiva; Maratonas UnB de Programação, que contextualiza os eventos das Maratonas UnB de programação; Juízes Onlines, que trata da definição de um juiz online e o impacto do contato de alunos com essa tecnologia; Arquitetura em micro-serviços, que aborda a definição da arquitetura de micro-serviços. Web APIs, que discorre sobre a definição de Web APIs e Desenvolvimento WEB, que aborda sobre desenvolvimento web.

2.1 Maratonas de programação

Programação competitiva é um termo usado para competições, onde os participantes resolvem problemas conhecidos de Ciência da Computação o mais rápido possível. Os problemas resolvidos não são inéditos, são baseados em conhecimentos de Ciência da Computação e conhecidos por pelo menos o autor ou autores do problema, possuindo uma solução. Para los resolver, os participantes devem colocar em prática os conhecimentos de ciência da computação, elaborando uma solução que passará por testes ocultos; os testes realizados na solução do participante devem ter resultados iguais aos resultados da solução do problema. A competitividade se encontra com a resolução dos problemas o mais rápido possível.

> *The core directive in 'Competitive Programming' is this: "Given well-known Computer Science (CS) problems, solve them as quickly as possible!".*
>
> *Let's digest the terms one by one. The term 'well-known CS problems' implies that in competitive programming, we are dealing with solved CS problems and not research problems (where the solutions are still unknown). Some people (at least the problem author) have definitely solved these problems before. To 'solve them' implies that we must push our CS knowledge to a certain required level so that we can produce working code that can solve these problems too—at least in terms of getting the same output as the problem author using the problem author's secret test data within the stipulated time limit. The need to solve the problem 'as quickly as possible' is where the competitive element lies—speed is a very natural goal in human behavior.* (HALIM et al., 2013, p. 1)

Maratona de programação é um termo adaptado do inglês *Programing Contest*; são competições utilizando programação competitiva. As competições possuem uma lista de problemas de programação competitiva, onde os competidores devem tentar resolvê-los no menor tempo possível. Para cada tentativa de resolução do problema, o competidor consegue acompanhar se sua solução apresentou o comportamento esperado ou não[1]; soluções incorretas ou que não apresentaram o comportamento esperado podem penalizar o competidor, encorajando que os competidores sejam não somente rápidos, como também precisos. Muitas maratonas de programação são em equipe, o que faz com que os competidores coloquem em prática não só os conhecimentos de ciência da computação, como também o trabalho em equipe.

> *It's clear that a programming contest is, by its own definition, a competitive activity, where there are winners and others (not really losers, in general).[...]. Moreover, many of the programming contests are team competitions and they involve a lot of collaborative work to prepare them.* (REVILLA; MANZOOR; LIU, 2008, p. 132)

O formato mais comum das maratonas é o utilizado no ICPC. O ICPC teve início no ano de 1977 em Atlanta, e conta com o total de 44 edições até a edição de 2020. A competição é feita entre equipes, que recebem uma lista de problemas para resolver em um tempo pré-estabelecido de prova. Cada equipe é composta por três participantes[2], que trabalham em equipe para resolver os problemas. Os times são classificados pelo maior número de problemas resolvidos; caso mais de uma equipe tenha o mesmo número de problemas resolvidos, a classificação desses times é feita pelo menor tempo total gasto para resolver os problemas. O tempo gasto para resolver um problema se dá pelo intervalo de tempo do início da competição até a primeira submissão aceita pelo juiz, mais 20 minutos de penalidade por cada submissão anterior errada (ICPC, 2021).

2.2 Maratonas UnB de Programação

As Maratonas UnB de Programação são eventos baseados no ICPC e realizados pela UnB (Universidade de Brasília). Os eventos se iniciaram em 2013, e os idealizadores foram: o Prof. Dr. Edson Alves da Costa Junior, professor adjunto da UnB desde 2009, com graduação e mestrado em Matemática, e doutorado em Engenharia Elétrica pela UnB[3]; o

[1] Além de ter as respostas esperadas em cada um dos casos de testes ocultos, é importante que a solução utilize os recursos de tempo e memória estabelecidos no problema. Logo, uma solução que apresentou o comportamento esperado não se limita apenas às soluções com as saídas corretas para os casos de teste.

[2] Em alguns casos o equipe pode conter um participante reserva, formando uma equipe com 4 participantes, porém apenas 3 participantes podem participar da competição.

[3] <http://lattes.cnpq.br/2105379147123452>.

Figura 1 – I Maratona UnB de programação

Fonte: <http://maratona.unb.br/>.

Prof. Dr. Diego de Freitas Aranha, professor adjunto da UnB entre 2011 e 2014, formado em Ciência da Computação na UnB e com mestrado e doutorado também em Ciência da Computação pela Universidade Estadual de Campinas[4]; e o Prof. Dr. Guilherme Novaes Ramos, professor adjunto da UnB desde 2011, formado em Engenharia Mecatrônica na UnB e com mestrado e doutorado em Ciência da Computação pelo Instituto de Tecnologia de Tóquio[5].

A I Maratona UnB de programação foi realizada em 2013 no LINF (Laboratório de Informática) da UnB — campus Darcy Ribeiro. O evento teve a participação de 2 escolas, e contou com o total de 11 equipes e 33 pessoas participando. A competição teve como vencedora a equipe da Faculdade UnB Gama (FGA), "Lone Wolves". Em 2014 foi realizada a II Maratona UnB de programação, também no LINF; nessa edição do evento houveram 12 equipes participantes e 9 problemas no total. Como na última edição, a equipe vencedora também foi a equipe "Lone Wolves". Na III Maratona UnB de programação, em 2015, o evento teve 21 equipes participando, 4 escolas, e 3 *problem setters*. Nessa edição a equipe vencedora foi "Teorema de Offson", também da FGA.

[4] <http://lattes.cnpq.br/9788199690491510>.
[5] <http://lattes.cnpq.br/7879595143050087>.

Figura 2 – II Maratona UnB de programação

Fonte: <http://maratona.unb.br/>.

Figura 3 – III Maratona UnB de programação

Fonte: <http://maratona.unb.br/>.

Figura 4 – IV Maratona UnB de programação

Fonte: <http://maratona.unb.br/>.

A IV Maratona UnB de programação recebeu o total de 5 escolas, 26 participantes e teve como equipe vencedora "Turkeys", uma equipe de Ciência da Computação da UnB (CiC). A quinta edição do evento ocorreu presencialmente na Faculdade UnB Gama e teve a participação remota de algumas equipes. No total, foram 58 equipes participantes de 12 estados diferentes; nessa edição, a equipe vencedora foi "Ahozinho com Feijão", da Universidade Federal de Uberlândia.

O sistema utilizado até a V edição do evento foi o BOCA (CAMPOS; FERREIRA, 2004), mesmo sistema utilizado em competições ICPC regionais e nacionais. Assim como o sistema, a competição segue as regras de uma competição ICPC. O tempo de prova é de 5 horas e as equipes são formadas por 3 pessoas[6]. Na VI Maratona UnB de programação em 2018, o sistema utilizado diferiu, a maratona aconteceu no juiz online Codeforces e contou com 13 equipes presenciais no LINF e na FGA. Além disso, houveram 41 equipes que participaram remotamente; a competição teve 14 problemas e a equipe vencedora foi a equipe do CiC "100% é Pouco, Pagode Importa D+".

[6] Dada a informalidade do evento, é possível que uma equipe participe sem os 3 participantes presentes.

A sétima edição do evento em 2019 contou com 29 equipes presenciais e 20 equipes remotas, o evento teve o total de 12 problemas e 382 submissões, a equipe vencedora dessa edição foi "ICPC top 46 super team", equipe do CiC. Em 2020, por conta da pandemia de COVID-19, a VIII Maratona UnB de programação ocorreu remotamente pelo Codeforces, o evento teve o total de 16 problemas e 31 equipes participantes; nessa edição, a equipe vencedora foi ":ultra_fast_parrot:", equipe da Universidade Federal do Ceará — Campus Quixadá.

A IX Maratona UnB de programação em 2021 ocorreu remotamente, contou com o total de 43 equipes, 15 problemas e 9 *problem setters*; a equipe vencedora dessa edição foi a equipe da Universidade Federal de Goiás "Cadê as pizza?". Em 2022, a X Maratona UnB de programação teve o retorno presencial; como na primeira edição, a maratona aconteceu no LINF, e o sistema utilizado foi o BOCA; a maratona teve o total de 30 equipes e 15 problemas e a equipe vencedora foi a equipe do CiC "Meianoite eu te conto".[7]

O evento teve 10 edições até o momento, e como observado, a participação das equipes nas maratonas UnB de programação é variada. Não limitada apenas à UnB, também houve a participação de universidades de outros estados no evento. O evento também não se limita a universidades públicas, o Instituto Federal de Brasília (IFB) e a faculdade IESB, por exemplo, já estiveram presentes em edições do evento: o IFB com a primeira participação na terceira edição e o IESB na quarta edição (UNB, 2022). As Maratonas UnB de Programação tem motivado alunos iniciantes com um bom desempenho em programação a se desafiarem, competindo contra programadores mais experientes. Além disso, muitas equipes que competiram nessa maratona alcançaram bons resultados em edições do ICPC.

2.3 Juízes Onlines

Em maratonas de programação são utilizados juízes eletrônicos, corretores automáticos de problemas. No momento em que um competidor submete o código de um problema, o juiz eletrônico é responsável por compilar o código recebido, gerando um programa, executar o programa em uma lista de testes e determinar o veredito da solução. O veredito da solução se dá pelas saídas do problema nos testes e pelo comportamento do programa durante a execução, além de possuir as saídas esperadas para cada um dos testes executados, o programa deve utilizar os recursos de tempo e memória determinados no problema; além disso, o programa executado não pode apresentar um comportamento inesperado, como parar a execução no meio dos testes, ou utilizar recursos não permitidos

[7] Os números foram obtidos na apresentação dos resultados da X Maratona UnB, os quais não foram publicados ainda.

pelo problema, como, por exemplo, paralelismo[8].

Ao enviar um problema para um juiz eletrônico, sua correção pode variar, dependendo das saídas esperadas do programa e os recursos gastos para execução. Quando o programa tem saídas diferentes das esperadas, o veredito retornado é *Wrong Answer* (WA); caso o tempo de execução do programa seja maior que o esperado, o veredito é *Time Limit Exceeded* (TLE); se o software utilizar mais memória do que permitido, o juiz retorna *Memory Limit Exceeded* (MLE); e por fim, se as saídas do software forem iguais aos resultados esperadas e o software utilizar os recursos permitidos, o veredito é *Accepted* (AC). Existem outros vereditos como: *Compilation Error* (CE), retornado quando existe algum erro com a compilação do código; *Runtime Error* (RTE) retornado quando o programa executado apresenta um erro durante a execução; e outros vereditos que podem ser retornados dependendo do juiz eletrônico e o problema para o qual o código foi enviado.

Um juiz online é em uma plataforma online com problemas de programação competitiva, que permite que soluções sejam enviadas para seus problemas e que essas soluções sejam julgadas. São exemplos de juízes onlines plataformas como: Codeforces[9], Becrowd[10] e LeetCode[11]; cada uma dessas plataformas possui um juiz eletrônico próprio para julgar os problemas enviados. Além dos juízes eletrônicos próprios das plataformas, existem juízes eletrônicos de código aberto como: BOCA[12], MOJ TOOLS[13] e Ejudge[14].

Atualmente, além de maratonas de programação, juízes online são bastante utilizados para o ensino de programação básica. Francisco, Júnior e Ambrósio (2016) ressaltam os benefícios da utilização de um juiz online no ensino de matérias inicias de programação:

> "[...] Aprendizagem no ritmo do aluno, auto-aprendizagem e redução da carga de trabalho do professor, são alguns dos benefícios apontados que contribuem não só em ambientes tradicionais de ensino, mas em ambientes de Educação a Distancia (EAD) e em MOOC's. A liberdade de definir listas de exercício e a disponibilidade de instrumentos para acompanhar os alunos são questões importantes para o professor." (FRANCISCO; JÚNIOR; AMBRÓSIO, 2016, p. 18-19)

Com a introdução de alunos aos juízes online, o ambiente de maratonas de programação se torna mais familiar. Isso se dá ao fato das maratonas utilizarem um formato de questões parecido ou até mesmo igual ao formato de juízes online. O envolvimento de

[8] Os juízes podem ser configurados para não permitir que o programa execute tarefas em paralelo, porém essa não é uma regra, existem casos onde esse recurso pode ser permitido.
[9] Disponível em: <https://codeforces.com/>.
[10] Disponível em: <https://www.beecrowd.com.br/>.
[11] Disponível em: <https://leetcode.com/>.
[12] Disponível em: <https://www.ime.usp.br/~cassio/boca/>.
[13] Disponível em: <https://github.com/cd-moj/mojtools>.
[14] Disponível em: <https://ejudge.ru/>.

alunos com maratonas se mostra positivo em relação aos seus resultados com disciplinas gerais de programação. Em um estudo realizado no curso de Engenharia de Software na Universidade de Brasília, Pereira et al. (2016) ressaltam que:

> "Entre as influencias encontradas, quanto ao desempenho individual, com base nos resultados da analise de desempenho geral do aluno, observou que 69,9% dos alunos tiveram um desempenho maior do que o desempenho anterior a esse contato. Ainda sobre a analise desempenho individual destes alunos em disciplinas de programação antes e depois da utilização desta estratégia de ensino, observou-se que 50,3% dos alunos apresentaram um aumento no desempenho em disciplinas de programação" (PEREIRA et al., 2016, p. 218)

2.4 Arquitetura em micro-serviços

A arquitetura de micro-serviços consiste em variados *softwares* que trabalham independentemente. Em contraposição, existe a abordagem monolítica onde o serviço concentra todas as responsabilidades do software. De fato, a abordagem monolítica é bastante eficaz para um sistema pequeno, pois facilita o *deploy* e desenvolvimento da aplicação; no entanto, a medida que cresce a concentração de funcionalidades em um só sistema, a complexidade do código também aumenta, dificultando seu entendimento e manutenção, conforme afirmam Dmitry e Manfred (2014).

A separação de aplicações em módulos visa a flexibilidade de escalar os micros serviços independentemente: "[...] Com a arquitetura monolítica, não é possível escalar cada componente de maneira independente" (DMITRY; MANFRED, 2014, p. 24, tradução nossa)[15]. Outro benefício na separação das responsabilidades em dois sistemas independentes é o desenvolvimento de novas *features*, manutenção e possível troca de *framework* ou de tecnologia presente no software para algo mais adequado, ou algo mais novo.

Em uma arquitetura monolítica, mudar o código pode se tornar muito difícil dependendo do tamanho e complexidade da aplicação, de modo coaduno, Dmitry e Manfred (2014, p. 24, tradução nossa) afirmam que: "[...] Com a arquitetura monolítica, é muito difícil (se lê impossível) realizar mudanças."[16].

2.5 Web APIs

Application Programing Interface (API) é o termo usado para a representação da interface de uma aplicação. Essa interface dispõe as funções para interação com o

[15] *"[...] With a monolithic architecture, we can not scale each component independently".*
[16] *"[...] With the monolithic architecture, it is very difficult (read impossible) to change it.".*

sistema. Uma *Web* API é a interface de um serviço *web*, responsável por receber e atender as requisições enviadas ao sistema (MASSE, 2011; RICHARDSON et al., 2013).

A comunicação com uma Web API é feita por *requests* utilizando o protocolo HTTP de comunicação. Essa comunicação pode ou não ter envio de informações atreladas. Quando há o envio de informações, um formato utilizado para o envio de dados é o JSON, padrão utilizado para a representação de estruturas de dados (RICHARDSON et al., 2013).

Aplicações modernas e sistemas baseados em micro-serviços necessitam de comunicação com serviços externos. A utilização de Web APIs é uma alternativa para estabelecer uma ponte de comunicação entre serviços. A API oferece uma lista de funções, que funcionam como uma camada de abstração para o sistema interno. Com isso, os sistemas compartilham informações sem comprometer suas independências (GHEBREMICAEL, 2017).

2.6 Desenvolvimento WEB

A definição de Aplicação Web de Hadley é: "[...] um aplicativo Web é definido como um aplicativo dinâmico baseado em comunicação HTTP cujas interações são passíveis de processamento por máquina" [17] (HADLEY, 2006, p. 1, tradução nossa). Um exemplo de aplicação web é o site, conjunto de páginas web que compõem uma aplicação.

Para o desenvolvimento de sites existem várias abordagens. Dentre elas, uma abordagem popular é a utilização do *frameweork React* — uma biblioteca *javascript open-source* desenvolvida pelo *Facebook*. O conceito principal do *React* é a reutilização de componentes de interfaces para a aceleração do processo de desenvolvimento (RAWAT; MAHAJAN'S, 2020).

Para reutilização de componentes *React* são utilizados *Node Packages* — bibliotecas *javascritpt*. Para gerir essas bibliotecas é utilizado o NPM (*Node Package Manager*), um recurso utilizado pelo React para controle de pacotes. Esse recurso permite a utilização de bibliotecas já desenvolvidas na aplicação, visando a otimização do trabalho, de acordo com Rawat e Mahajan's (2020) "[...] A utilização de pacotes npm em seu empreendimento pode diminuir o tempo esperado para a realização de uma tarefa."[18](RAWAT; MAHAJAN'S, 2020, p. 699, tradução nossa).

[17] *[...] a Web application is defined as a dynamic HTTP-based application whose interactions are amenable to machine processing. [...].*

[18] *"[...] Utilizing npm packages in your venture can diminish the measure of time expected to accomplish the errand.".*

3 Metodologia

Nesse capítulo serão abordados os métodos utilizados para encontrar e selecionar juízes *online*, os métodos de desenvolvimento do GOJ, os critérios para seleção de juízes *online*, os métodos de contagem de seus problemas, a arquitetura geral do GOJ, a arquitetura e tecnologias utilizadas em cada um dos módulos do sistema.

3.1 Produtos relacionados

Para o desenvolvimento do GOJ, uma das etapas foi identificar os principais juízes online, para observação das funcionalidades e características de cada um deles. Para isso, foi realizada uma busca por juízes online e uma seleção de alguns desses juízes. Depois, foi feito um levantamento de funcionalidades e do número de problemas de cada um.

Nessa seção serão descritos os métodos para a identificação dos principais juízes onlines existentes, os métodos para levantamento do número de questões e das funcionalidades em cada um deles, além disso, serão apresentados os critérios de seleção dos juízes.

3.1.1 Identificação dos principais juízes online existentes

Conforme definido na Seção 2.3, um juiz *online* é um site ou página que armazena problemas e possibilita ao usuário enviar suas soluções para o problema, para que estas soluções sejam avaliadas e que o usuário receba um veredito. A identificação dos principais juízes *online* existentes foi feita com base em pesquisas de termos específicos no *Google*[1], mecanismo de pesquisas *online*. Para a seleção de resultados foram considerados apenas os sites que se encaixaram na definição de juiz *online* citada. Apenas as duas primeiras páginas retornadas na pesquisa foram consideradas para cada uma das buscas. Os resultados estão enumerados em ordem de relevância, determinada pelo mecanismo de pesquisa do Google, que lista os resultados mais relevantes primeiro, ou seja, em ordem de aparição. Vale ressaltar que os resultados das buscas listam páginas de anúncios, porém, estes anúncios foram desconsiderados.

A primeira busca realizada foi com o termo "*online judge*", e obteve os seguintes resultados: "*online judge*": o Online Judge[2], conhecido anteriormente como *UVa*; o Bee-

[1] Disponível em: <https://www.google.com/>.
[2] Disponível em: <https://www.onlinejudge.org/>.

crowd[3], antigo URI, o Sphere Online Judge (SPOJ)[4], e o PKU JudgeOnline[5], o juiz *online* da Universidade de Pequim.

A segunda busca foi realizada com o termo "*online programming contests*" e os resultados foram:

1. CodeChef
2. Google's Coding Competitions
3. Beecrowd
4. Sphere Online Judge (SPOJ)
5. Hackerearth

O novo termo foi utilizado com o propósito de identificar sites que, além de serem juízes *online*, possuem competições de programação. Como consequência, o resultado listou juízes diferentes daqueles relacionados na busca anterior. São eles: CodeChef[6], um juiz *online* que possui alguns diferenciais como competições e o armazenamento de questões das competições passadas para práticas posteriores; Goolge's Coding Competitions[7], site onde estão reunidos *links* para competições do Google como Kick Start[8], Hash Code[9] e CodeJam[10]; e por fim, o Hackerearth[11], site muito usado para prática de entrevistas técnicas de programação e que oferece um serviço de entrevista técnica para empresas, onde permite que os candidatos realizem entrevistas técnicas na plataforma, com o acompanhamento de um entrevistador.

O terceiro termo pesquisado foi "*programming competitions and contests*", e os resultados foram:

1. Google's Coding Competitions
2. CodeChef
3. Codeforces
4. Hackerearth

[3] Disponível em: <https://www.beecrowd.com.br>.
[4] Disponível em: <https://www.spoj.com/>.
[5] Disponível em: <http://poj.org/>.
[6] Disponível em: <https://www.codechef.com/>.
[7] Disponível em: <https://codingcompetitions.withgoogle.com/>.
[8] Disponível em: <https://codingcompetitions.withgoogle.com/kickstart/about/>.
[9] Disponível em: <https://codingcompetitions.withgoogle.com/hashcode/about/>.
[10] Disponível em: <https://codingcompetitions.withgoogle.com/codejam/about/>.
[11] Disponível em: <https://www.hackerearth.com/>.

Este terceiro termo tinha como objetivo ressaltar sites de competição e o único site novo que apareceu foi o Codeforces[12], um site voltado para competições de programação que possui um vasto número de problemas para práticas posteriores. Assim, se tornam diferenciais do site as competições e as questões de competições passadas para práticas posteriores.

A quarta busca foi utilizando o termo "juiz *online* programação", e os resultados foram:

1. Beecrowd
2. Neps Academy
3. CodeBench

O quarto termo foi construído propositalmente usando duas palavras da língua portuguesa, para deste modo tentar encontrar juízes *online* com questões em português, ou que possuem o português do Brasil como um dos idiomas dos textos. Esta quarta busca obteve dois novos resultados: Neps Academy[13], um site voltado para o ensino de programação, que possui como diferenciais aulas, competições e o armazenamento das questões de competições para práticas posteriores; e o CodeBench[14], site voltado para ensino, com criação de turma e inscrição de alunos.

Também foram pesquisados os termos "programação" e"*contests* de programação", porém não foram encontrados novos resultados.

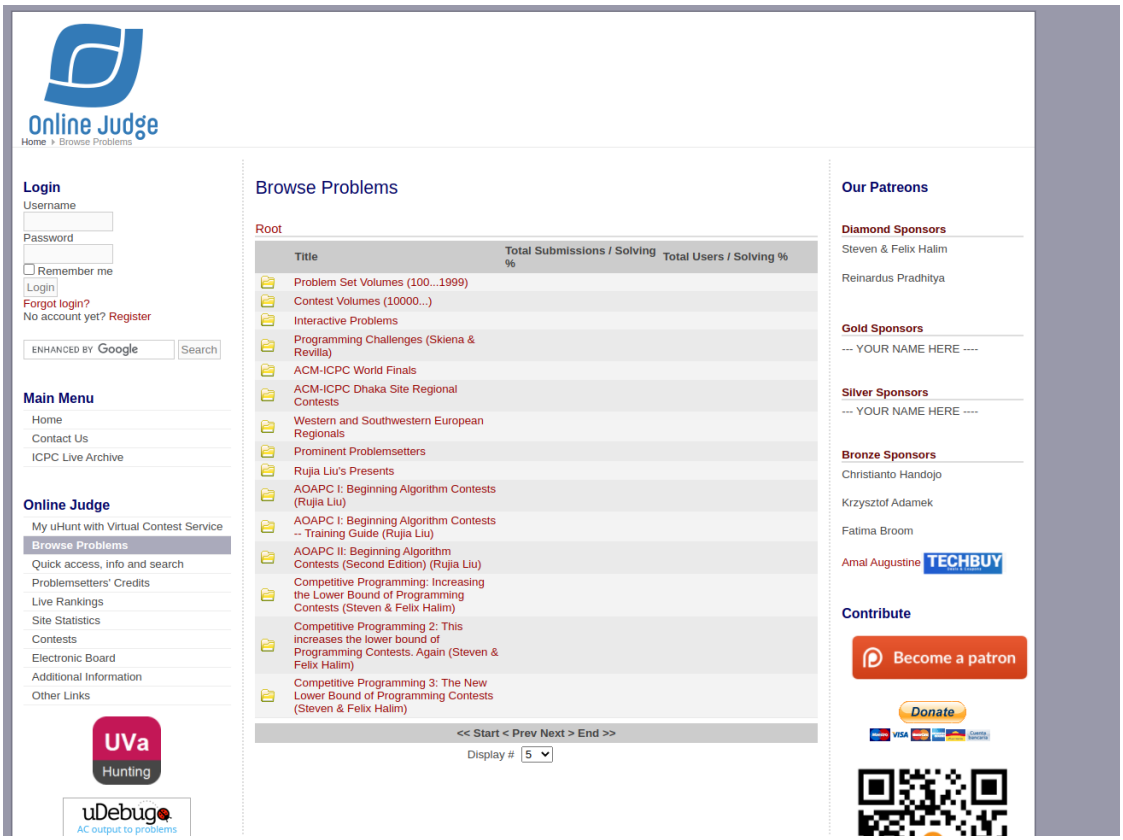
3.1.2 Levantamento do número de problemas

Em cada um dos juízes selecionados na Subseção 3.1.1 foi realizado um levantamento do número total de problemas disponíveis para seus usuários. Para isso, distintas abordagens foram escolhidas conforme as diferentes características de cada juiz *online*. Nessa subseção serão apresentadas as abordagens e estratégias utilizadas para esta contagem de problemas.

Para o levantamento dos problemas da plataforma Online Judge foi acessada a aba *Browse Problems*[15], conforme mostra a Figura 5. Em cada uma das categorias foi contado o número de problemas presentes (veja a Figura 6). Esta contagem foi realizada por um *script* em linguagem de programação Python, disponível no Apêndice A.1, com a estratégia de localizar o padrão em *links* de categorias e *links* de problemas. O primeiro padrão observado é que as pastas ficam em uma tabela; ao localizar a tabela pelo HTML

[12] Disponível em: <https://codeforces.com/>.
[13] Disponível em: <https://neps.academy/>.
[14] Disponível em: <https://codebench.icomp.ufam.edu.br/>.
[15] Disponível em: <https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8>.

Figura 5 – Online Judge — Browse Problems



Fonte: Online Judge <https://onlinejudge.org>.

da página é possível encurtar a busca pelos links. O segundo padrão que vale ressaltar é que os problemas possuem o trecho "`page=show_problem`" na (URL), e isso diferencia um *link* de outra pasta de um *link* de problema.

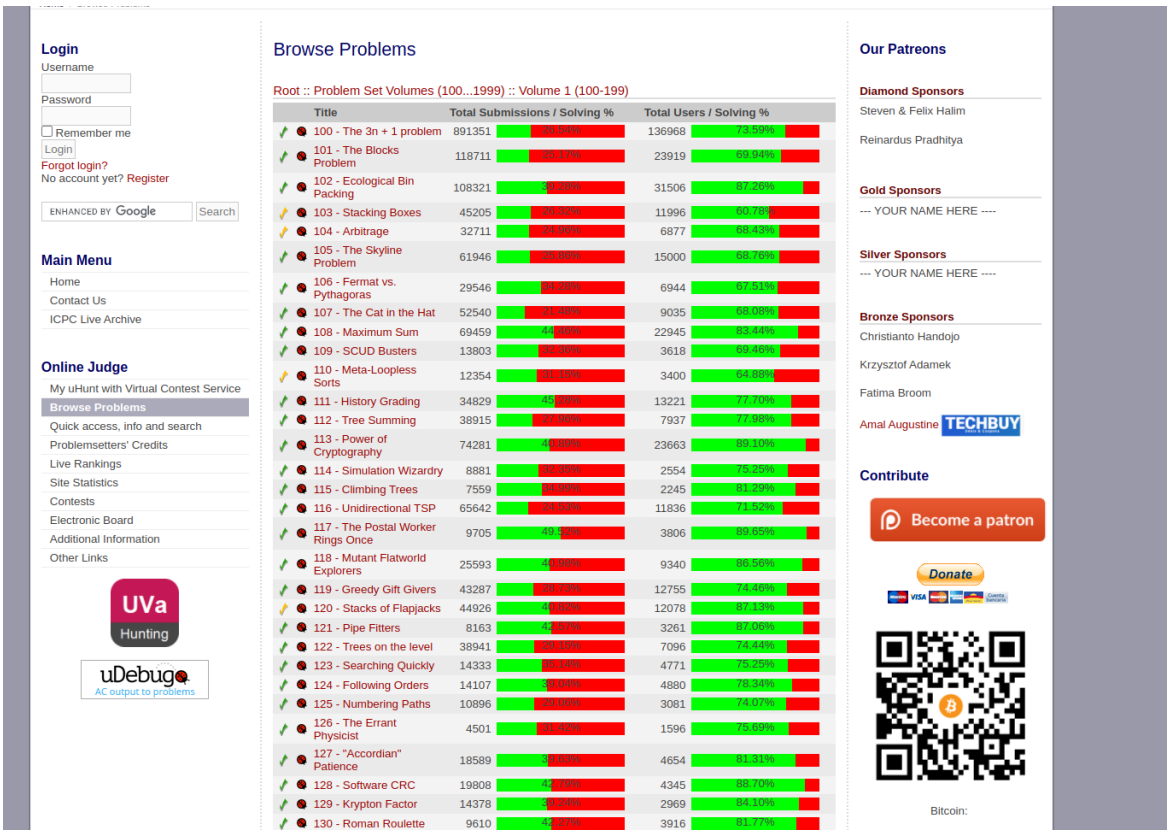
No site Beecrowd foi preciso acessar a *URL* de categorias[16], onde o número de problemas é exibido na categoria "Listar todos", conforme apresentado na Figura 7.

No Codeforces foi utilizada a *API* pública do site[17], que possui recursos para acessar as informações da plataforma; a *API* é acessada via protocolo *HTTP* e possui *endpoints* públicos e privados. Um dos *endpoints* públicos permite a visualização das informações dos problemas, retornando todos os problemas quando nenhum filtro é informado. Para recuperação do número total de problemas, foi utilizado um *script* em linguagem *bash*, disponível no Apêndice A.2.

Para o levantamento do total de problemas do site CodeChef foi acessada a página de problemas, e dentro dela, foi utilizado o filtro *"All Levels"*, que revelou os problemas de todos os níveis, como apresentado na Figura 8. Com isso, foi possível observar o número de problemas mais abaixo, conforme mostra a Figura 9.

[16] Disponível em: <https://www.beecrowd.com.br/judge/pt/categories>.
[17] Disponível em: <https://codeforces.com/apiHelp>.

Figura 6 – Online Judge — Browse Problems (Detalhes)



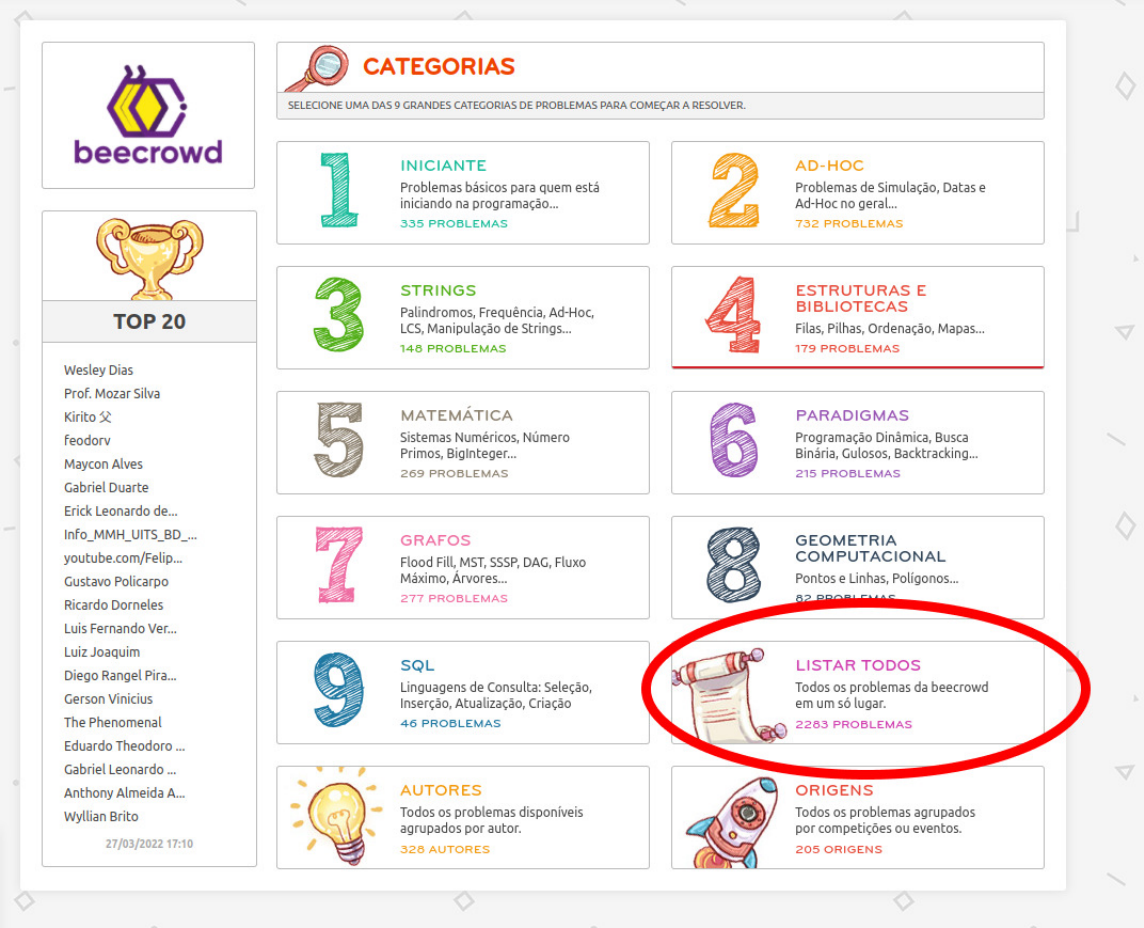
Fonte: Online Judge <https://onlinejudge.org>.

Para o levantamento do número de problemas no Google's Coding Competition foram separadas 3 categorias diferentes, correspondentes aos 3 eventos anuais do Google: Code Jam, KickStart e HashCode. Na categoria Code Jam existem 8 rounds diferentes, dentre eles *rounds* de prática, classificatórios e os oficiais. O número de problemas das finais variam de 5 a 6 problemas, enquanto nos demais *rounds* varia de 3 a 4 problemas; como até hoje foram realizadas 4 competições, começando em 2018, foi estimado um total de 120 problemas para essa categoria. Na categoria KickStart são 8 *rounds* em cada edição, sendo que cada *round* contém 3 problemas. A competição teve início em 2018, totalizando 4 edições, de modo que o número estimado de problemas é igual a 96. Por fim, na categoria HashCode há apenas 2 problemas por edição: 1 problema do *round* classificatório e 1 do *round* final, totalizando 8 problemas nas 4 edições que se iniciaram em 2018.

Não foi possível fazer o levantamento do número de problemas na plataforma CodeBench, pelo fato dos problemas serem privados. A plataforma disponibiliza para professores a criação de turma, onde o acesso é controlado, e por conta disso, não foi feita uma estimativa do número de problemas na plataforma.

Para o levantamento de problemas no Sphere Online Judge foi acessado o menu de problemas. Nele foram localizadas 6 categorias: "*Classical*", "*Chalenge*", "*Partial*", "*Tu-*

Figura 7 – Beecrowd — Total de problemas



Fonte: Beecrowd <https://www.beecrowd.com.br>.

Tabela 1 – Informações das categorias — Sphere Online Judge [18]

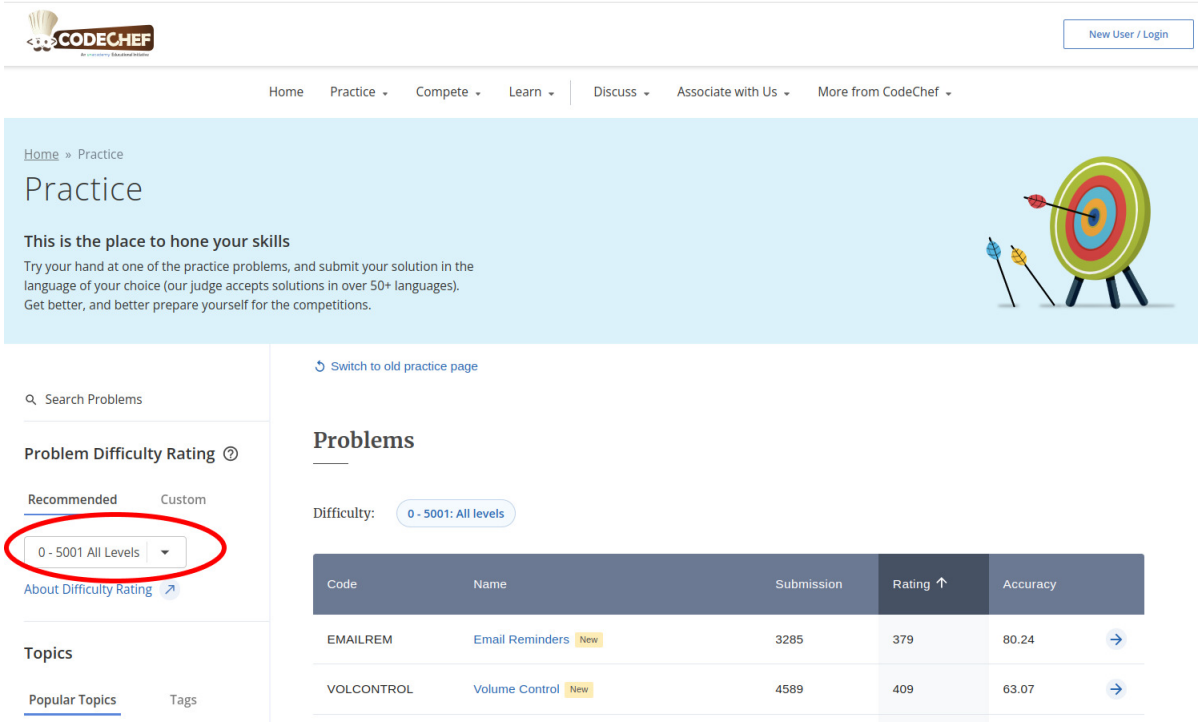
Categoria	**Número de páginas**	**Problemas na última página**
Classical	79	50
Chalenge	4	10
Tutorial	28	49
Partial	1	35
Basics	7	3

Fonte: Sphere Online Judge <https://www.spoj.com>.

torial", "*Riddle*" e "*Basics*", como mostra a Figura 10. Para o levantamento do número de problemas foram acessadas cada uma das categorias, e o número de páginas por categoria foi contado. Com exceção da última página, as demais apresentam 50 problemas; portanto, para ter o total de problemas, foram contadas as páginas — exceto a última — e o resultado foi multiplicado por 50. Em seguida, foi adicionado o total de problemas da última página ao valor; na Tabela 1 são apresentadas as informações citadas por categoria.

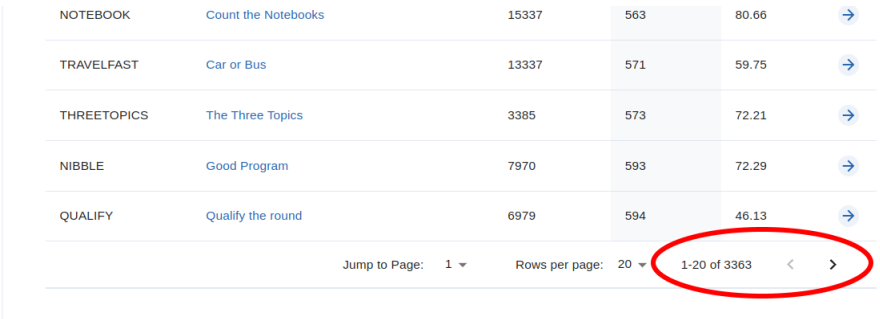
[18] Data da observação: 05/04/2022.

Figura 8 – Code Chef — Filtro de problemas



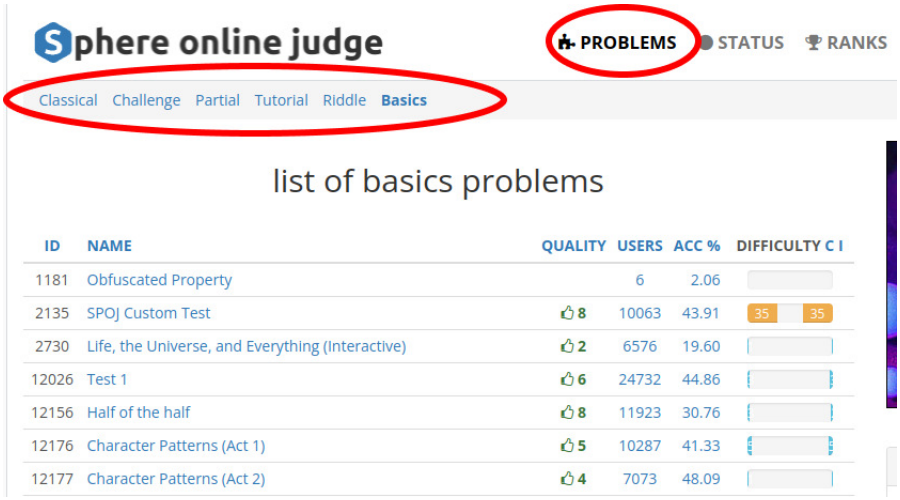
Fonte: Code Chef <https://www.codechef.com>.

Figura 9 – Code Chef — Número de problemas



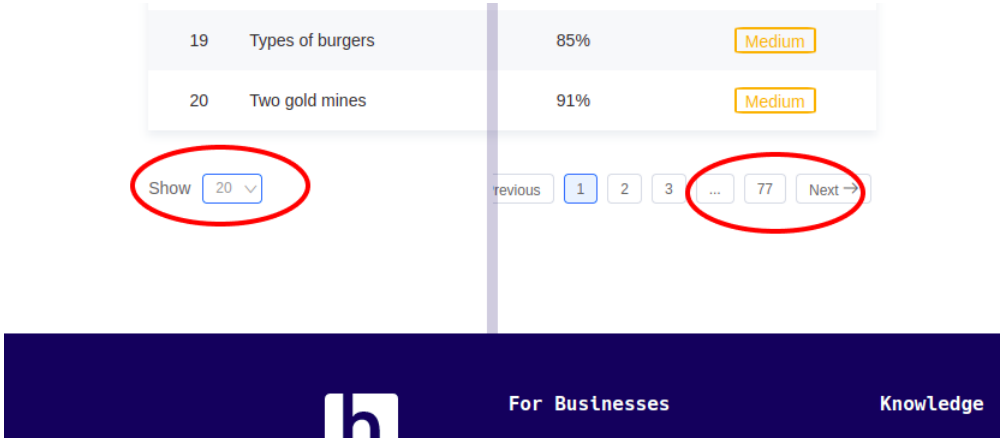
Fonte: Code Chef <https://www.codechef.com>.

Figura 10 – Sphere Online Judge — Categorias de problemas



Fonte: Sphere Online Judge <https://www.spoj.com>.

Figura 11 – Hackerearth — Número de problemas



Fonte: Hackerearth <https://www.hackerearth.com>.

Os demais juízes onlines possuem alguma forma de apresentar todos os problemas separados por páginas, com um valor fixo de problemas em cada uma delas. Com isso, o método para contagem nos demais sites foi: o valor do número de problemas por página é multiplicado pelo total de páginas — com exceção da página final —, e depois, o número de problemas na última página é adicionado ao valor total. Na Figura 11 é possível observar esse padrão na plataforma Hackerearth, que apresenta 77 páginas e um total de 20 problemas em cada; sem contar com a última página, é possível estimar o número de problemas em 1520.

Na Tabela 2 o levantamento dessas informações para os demais juízes é representado nas colunas: “Problemas por página”, “Total de páginas” e “Problemas na última página”; com essas informações é possível fazer uma estimativa do número total de problemas em cada uma das plataformas disponíveis na Tabela.

Tabela 2 – Coleta do número de problemas por página

Juíz online	Problemas por página	Total de páginas	Problemas na última página
Hackerearth	20	77	6
PKU Judge Online	100	31	54
Neps Academy	16	67	13

Fonte: Hackerearth <https://www.hackerearth.com>.

3.1.3 Levantamento das funcionalidades dos juízes

Nessa subseção será abordada a maneira onde o levantamento das funcionalidades existentes em cada um dos juízes online foi realizada. As observações de cada um dos juízes foi efetuada de maneira diferente, apesar da abordagem ter sido similar em alguns.

As funcionalidades do site *Online Judge* foram levantadas navegando pelo site, enviando problemas olhando a resposta e buscando por tutoriais e ferramentas adicionais no próprio site. O site possui links para ferramentas externas como *uDebug*[19], plataforma que auxilia a solução dos problemas disponibilizado um código correto para comparação de entradas e saídas, e o *uHunt*, uma plataforma direcionada para o site *Online Judge* que deixa a resolução de questões mais iterativa, mostrando estatísticas dos problemas resolvidos e separando as questões em categorias.

Para o levantamento das funcionalidades no site *PKU Online Judge*, foi feita uma navegação no site e acessando as funcionalidades foi possível coletar algumas informações. Ao enviar uma questão, e é possível perceber que o site aceita algumas linguagens e compiladores como mostrado na Figura **??**, porém não foi possível receber o veredicto de uma questão enviada, pois no dia 5 de abril às 11:43, horário em que o teste foi feito, o site retornava um erro ao enviar a questão;

Já no *Codeforces*, as funcionalidades foram observadas navegando e enviando questões. O foco do site está em competições, e ocorrem competições semanalmente. Além disso, a maioria dos problemas possui tutoriais e os códigos e soluções de outras pessoas são abertos.

3.2 Levantamento de requisitos

Antes e durante o desenvolvimento do GOJ, foi necessário realizar um levantamento de requisitos. Essa etapa é importante para obter detalhadamente os recursos que englobam a aplicação (YOUNG, 2002).

Semanalmente, foram feitas entrevistas para entendimento do GOJ, as responsa-

[19] Disponível em: <https://www.udebug.com/UVa/>.

bilidades do sistema e como ele deveria se comportar. Também foram feitas prototipagens para assegurar o correto andamento do desenvolvimento.

Os requisitos levantados da aplicação se dividem em três principais, descritos no Quadro 1.

Quadro 1: Requisitos gerais

Id	**Requisito**
1	É necessário armazenar questões e eventos das edições das maratonas UnB de programação, que ocorrem anualmente.
2	O usuário deve conseguir enviar sua solução em código para ela ser julgada, retornando um veredicto de sua submissão.
3	O usuário deve conseguir ter acesso aos problemas e eventos.

Fonte: o Autor.

Os requisitos citados no Quadro 1 são requisitos amplos e com pouca especificidade. Para não gerar ambiguidade ou incompletude para os mesmos, eles foram divididos em requisitos menores e mais específicos. Com isso, mais detalhes foram listados na busca de uma maior completude e clareza. Os requisitos menores serão descritos nas seguintes subseções: Armazenamento das questões e eventos, Juiz eletrônico e Interface de usuário.

3.2.1 Armazenamento das questões e eventos

As informações dos eventos e questões das maratonas UnB de programação precisam ser armazenadas e disponibilizadas para o funcionamento do GOJ; os requisitos dessa subseção dizem respeito ao armazenamento dessas informações. No Quadro 2 são mostrados os requisitos referentes ao armazenamento de questões e eventos do GOJ.

Os requisitos citados no Quadro 2 tem uma granularidade maior que os citados no Quadro 1. Com isso, o desenvolvimento das funcionalidades se torna mais preciso, evitando que expressões genéricas gerem ambiguidades com as necessidades da aplicação.

3.2.2 Juiz eletrônico

Quando o usuário escreve um código com sua solução para algum problema, o código deve ser julgado e o usuário deve conseguir ver o veredito de sua submissão. Para isso o GOJ precisa de um juiz eletrônico. No Quadro 3, estão listados requisitos específicos para o juiz eletrônico do GOJ.

[20] Os arquivos necessários de cada questão dependem da questão e do juiz eletrônico utilizado para julgar as submissões enviadas.

Quadro 2: Requisitos de armazenamento

Id	Requisito
1.1	As questões devem possuir título, enunciado, tempo limite de execução, limite de memória utilizado pelo programa, lista de rótulos que categorizam o problema, ID predefinido para localização e identificação das questões.
1.2	As questões possuem arquivos, utilizados para julgar as submissões enviadas.[20]
1.3	Os eventos devem possuir uma data, que identifica quando o evento aconteceu, um nome e uma lista de problemas, tal como o rótulo do problema naquele evento, ex: "Questão A".
1.4	Além das informações dos problemas, as soluções e lista de testes precisam ser armazenadas.

Fonte: o Autor.

Quadro 3: Requisitos do juiz

Id	Requisito
2.1	O sistema deve receber arquivos ou textos contendo códigos em linguagens de programação pré-definidas.
2.2	As linguagens suportadas pelo sistema devem ser as mesmas linguagens utilizadas no ICPC.
2.4	O sistema deve conseguir saber a memória e tempo utilizados para execução do programa.

Fonte: o Autor.

Os requisitos citados no Quadro 3 se relacionam com a parte do GOJ que julgará os códigos enviados pelos usuários. Os requisitos citados estão relacionados com os formatos das questões do GOJ. Por seguir o padrão de maratona ICPC, o juiz eletrônico do GOJ deve ter um suporte similar.

3.2.3 Interface de usuário

A interface de usuário é responsável por apresentar as informações do GOJ ao usuário e proporcionar mecanismos de iteração do usuário com a aplicação. Os requisitos listados no Quadro 4 são referentes à interface de usuário do GOJ e às informações das questões que serão disponibilizadas ao usuário.

Os requisitos no Quadro 4 descrevem como a interface deve ser. Ela se assemelha a juízes onlines presentes hoje, porém um foco direcionado para essa interface é ser amigável para estudantes iniciantes em programação e ser compatível com os elementos não textuais necessários para as questões das maratonas UnB de programação.

Quadro 4: Requisitos da interface

Id	Requisito
3.1	As informações das questões devem estar disponíveis ao usuário, podendo conter fórmulas matemáticas e imagens.
3.2	Os eventos devem ser apresentados ao usuário de maneira organizada, com a lista dos respectivos problemas dos eventos.
3.3	As informações de dicas e tutoriais das questões devem ser escondidas do usuário e disponíveis apenas se o usuário optar por elas.

Fonte: o Autor.

3.3 Arquitetura

A arquitetura escolhida para construção do GOJ foi a de micro-serviços, separando a aplicação em módulos com pequenos conjuntos de responsabilidades individuais. Além dos módulos desenvolvidos, foram utilizados os seguintes serviços externos: serviço de filas, de notificações e de armazenamento. A Figura 12 representa a arquitetura do Gamma Online Judge e os módulos desenvolvidos: "Gamma Judge Admin", "Gamma Judge UI", "Gamma Judge API" e "Gamma Judge Tools".

Nessa seção serão abordadas as arquiteturas, tecnologias e os métodos utilizados para o desenvolvimento dos módulos. Também serão abordadas as ferramentas e serviços externos utilizados no desenvolvimento do GOJ.

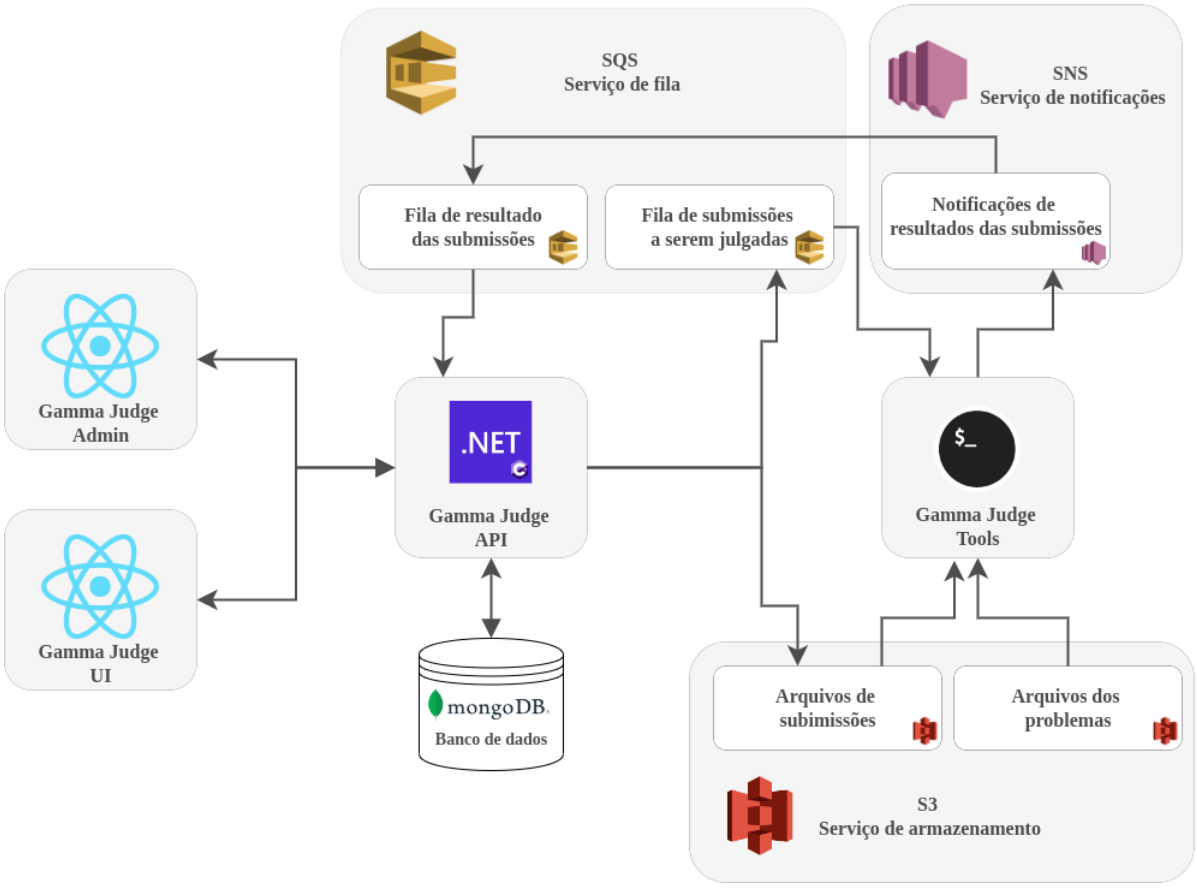
3.3.1 Serviços externos

Na arquitetura proposta, foram utilizados recursos do AWS Console[21], o qual disponibiliza ferramentas diversas para a manutenção, processamento e armazenamento de aplicações na nuvem; além disso, existem serviços prontos que auxiliam o desenvolvimento de aplicações. No GOJ, os serviços AWS que fazem parte da arquitetura são: Simple Queue Service (SQS), o Simple Notification Service (SNS) e o Simple Storage Service (S3).

O SQS consiste em um serviço online de filas, onde mensagens podem ser enviadas, lidas e consumidas. A ideia da utilização desse serviço é manter a consistência do funcionamento da aplicação em casos de intermitência ou erros do sistema. No GOJ, o serviço SQS é utilizado para que a comunicação entre o módulo Gamma Judge API — que recebe as submissões do usuário — e o módulo Gamma Judge Tools — que julga as submissões lidas da fila — seja consistente. No caso do Gamma Online Judge apresentar falhas pontuais para julgar as submissões — como falta de memória na máquina, inatividade do serviço, etc. — a mensagem não será consumida ou ficará no sistema de filas até

[21] Disponível em: <https://aws.amazon.com/pt/console/>.

Figura 12 – Arquitetura geral GOJ



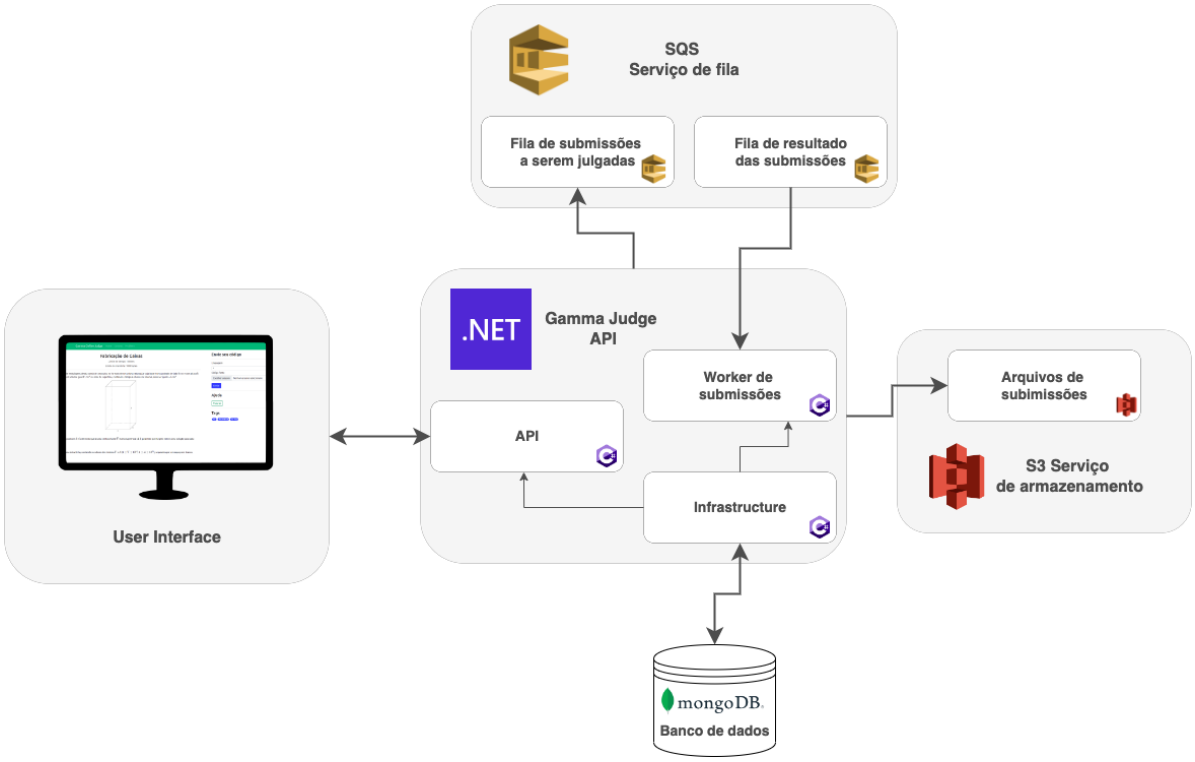
Fonte: o Autor.

isso acontecer; esta abordagem resulta em maior consistência, pois, em caso de inatividade ou intermitência do Gamma Judge Tools, o envio de submissões não será comprometido.

Já o SNS é um serviço de notificações, que permite que uma mensagem enviada possa ser distribuída para outros serviços. Sua utilização no GOJ consiste em notificar quando uma submissão for julgada, sendo esta notificação enviada através de um tópico SNS e os serviços inscritos nesse tópico recebem a mensagem. A "Fila de resultados das submissões" é inscrita no tópico SNS "Notificações de resultados das submissões", como apresentado na Figura 12, e todas as mensagens enviadas nesse tópico são enviadas para a fila, para que os resultados possam ser consumidos e apresentados ao usuário final.

Por fim, o S3 é um serviço de armazenamento de arquivos. Ele é utilizado para ler e armazenar os arquivos enviados nas submissões e os arquivos necessários para julgar os problemas. A utilização de um serviço de armazenamento permite que os arquivos sejam acessados por mais de um módulo da aplicação, como acontece na arquitetura do GOJ.

Figura 13 – Arquitetura — Gamma Judge API



Fonte: o Autor.

3.3.2 Gamma Judge API

O módulo Gamma Judge API é responsável por fazer uma ponte de comunicação com a interface, disponibilizando as informações armazenadas e recebendo arquivos para serem julgados pelo juiz eletrônico. Além disso, é responsabilidade desse módulo processar as informações dos resultados das submissões. Na Figura 13 também são apresentados os fluxos descritos e as interações com a interface e os serviços externos.

Conforme dito anteriormente, o sistema faz o uso de 2 serviços externos: SQS e S3. O SQS é utilizado para receber as informações dos resultados das submissões e para o envio de submissões a serem julgadas. Já o serviço S3 é utilizado para o armazenamento dos arquivos de submissões.

O Gamma Judge API se divide em 3 grandes partes: “API”, “Worker de submissões” e “Infrastructure”, como mostra a Figura 13. A “API” é responsável pela comunicação externa, recebendo e enviando informações para uma interface de usuário; o “Worker de submissões” é um *background service* que processa as submissões recebidas; e a “Infrastructure” é uma abstração dos serviços externos, que se comunica diretamente com o banco de dados e os serviços SQS e S3, disponibilizando funções para os demais módulos.

Para o envio de uma submissão, a API recebe um arquivo, que contém a solução proposta pelo usuário, as configurações (como linguagem e compilador que serão utiliza-

dos) e as informações do problema, as quais determinarão quais são as entradas e saídas esperadas. Após recebidas, as informações da submissão são armazenadas no banco de dados, o código-fonte é enviado para o S3 com um identificador único, que associa aquele arquivo à submissão, e as informações são enviadas para a "Fila de submissões a serem julgadas" no SQS; posteriormente um juiz eletrônico é responsável por reunir esses dados e julgar o problema. Após julgada, o resultado da submissão é enviado para a fila SQS "Fila de resultado das submissões", onde os resultados são processados e atualizados no banco de dados.

Esse módulo foi desenvolvido utilizando o .NET Frameweork com C# de linguagem de programação principal. O banco de dados é um MongoDB, para o armazenamento das informações de questões, eventos e submissões. A comunicação com a "User interface" é feita via protocolo HTTP, para o recebimento e disponibilização de informações (veja a Seção 2.5). Para melhora da compatibilidade e facilidade com a entrega desse sistema foi utilizada a ferramenta Docker, um sistema de virtualização que permite que o projeto seja executado em qualquer plataforma que suporte a ferramenta.

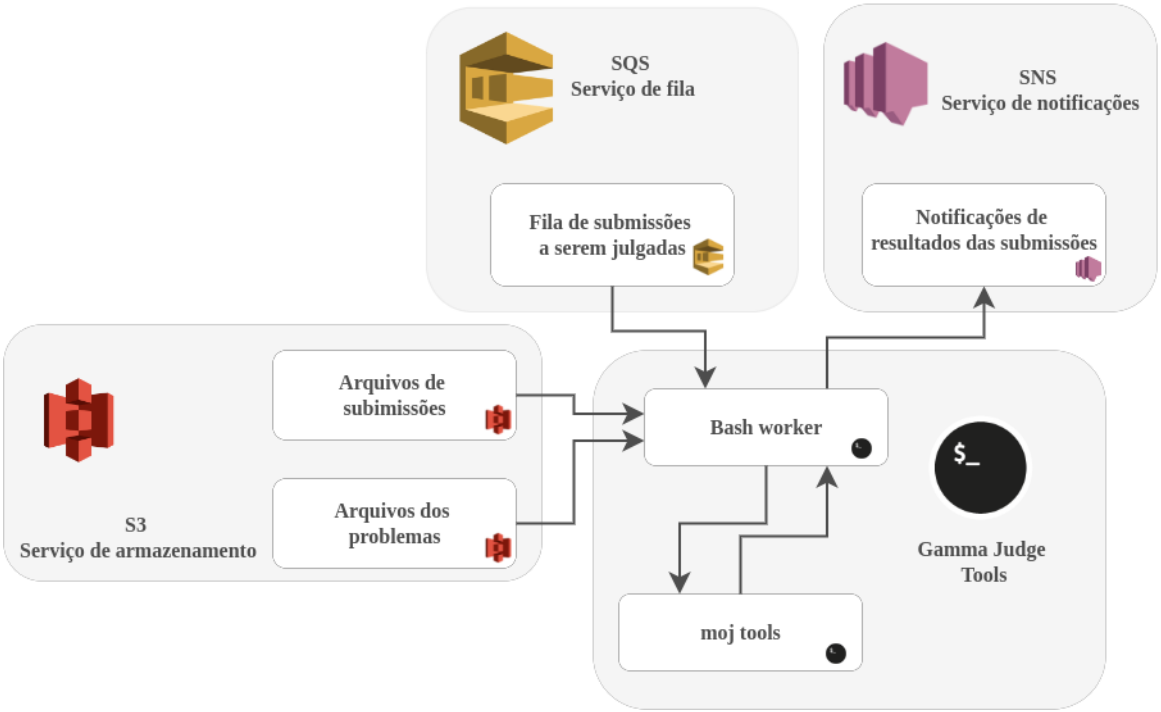
3.3.3 Gamma Judge Tools

O Gamma Judge Tools é o módulo responsável por julgar submissões e retornar o veredicto. Esse módulo realiza o papel de um juiz eletrônico e funciona com o auxílio dos serviços externos citados na Subseção 3.3.1. Os serviços externos utilizados nesse módulo são: SQS, SNS e S3, como mostra a Figura 14.

O módulo foi desenvolvido em linguagem de programação Bash e possui um *worker* para consumir as submissões. Cada submissão recebida possui as informações do problema correspondente: linguagem, identificador do problema e o identificador do arquivo armazenado no S3. Reunidas as informações, o módulo vai produzir, a partir do código-fonte, um código executável. Este executável será alimentado com várias entradas que correspondem aos testes do problema, e para cada entrada ele produzirá uma saída correspondente, denominada resposta. A resposta para cada entrada será confrontada com as respostas esperadas, armazenadas no S3. A comparação entre estas respostas e o comportamento do executável serão avaliados de modo a produzir um veredito para a submissão; o veredito da execução é enviado para uma notificação SNS, podendo ser: AC, WA, TLE, MLE ou RTE (veja a Seção 2.3). Em caso de falha do juiz na execução, a mensagem não é consumida e volta para a fila SQS para ser processada novamente.

Por esse módulo ser um serviço separado do Gamma Judge API, não é necessária sua exposição à *web*. Além disso, as filas SQS realizam o controle de concorrência caso mais de um consumidor esteja acessando esse recurso, possibilitando um escalonamento do sistema, ou seja, caso seja necessário mais processamento ou paralelismo para julgar as questões, basta rodar o Gamma Judge Tools em uma máquina adicional, sem se preocupar

Figura 14 – Arquitetura — Gamma Judge Tools



Fonte: o Autor.

com a concorrência entre os consumidores e a fila SQS.

3.3.4 Gamma Judge UI e Gamma Judge Admin

O Gamma Judge UI é a interface de usuário que se comunica com a API, recebendo e enviando informações. Ela recebe as informações dos problemas, eventos e submissões, e envia os códigos para serem processados. O Gamma Judge Admin tem um comportamento similar, porém as informações enviadas são para edição, criação e exclusão de problemas e eventos.

Ambos os módulos foram desenvolvidos em Node JS, com Typescript sendo a linguagem principal. Nos módulos foram utilizados *Node Packages*, bibliotecas externas que facilitam o desenvolvimento web (veja a Seção 2.6). Das bibliotecas utilizadas, as principais foram: *React Bootstrap*[22], que disponibiliza componentes, como botões, textos e tabelas; *React Router*[23], que disponibiliza recursos para navegação; e *React LaTeX*[24], que permite a renderização de elementos HTML e LaTeX.

O Gamma Judge UI é responsável por renderizar elementos não textuais, como fórmulas, imagens e elementos necessários para a visualização das informações de um

[22] Disponível em: <https://react-bootstrap.github.io/>.
[23] Disponível em: <https://v5.reactrouter.com/web/guides/quick-start>.
[24] Disponível em: <https://www.npmjs.com/package/react-latex>.

problema. As informações são recebidas em texto (texto bruto, HTML ou LaTeX) da API, e o sistema é responsável por transformar essas informações em elementos visuais como fórmulas, tabelas ou imagens. Para isso, o pacote *React LaTeX* é utilizado, pois, permite a inserção de trechos em LaTeXou HTML para a visualização na página WEB.

4 Resultados

Este capítulo apresenta os resultados de pesquisa e de desenvolvimento do GOJ. Os resultados das contagens de problemas dos juízes onlines selecionados e a seleção dos juízes estão na seção Identificação dos principais juízes online exisitentes; os resultados do desenvolvimento da aplicação e seus devidos módulos se encontram na seção Módulos do GOJ.

4.1 Identificação dos principais juízes online exisitentes

Nessa seção serão abordados resultados do levantamento dos juízes online existentes, o que embasou e motivou a criação de um novo juiz online. Esta seção está dividida em duas subseções: a subseção Identificação e seleção de juízes online, que mostra os juízes selecionados para as análises; a subseção Funcionalidades, que apresenta funcionalidades relevantes para o GOJ e a presença delas nos juízes selecionados.

4.1.1 Identificação e seleção de juízes online

Foram selecionados 10 juízes onlines com base nos critérios e buscas descritas na Subseção 3.1.1. Os juízes selecionados foram:

1. OnlineJudge
2. Beecrowd
3. Sphere Online Judge (SPOJ)
4. PKU JudgeOnline
5. CodeChef
6. Google's Coding Competitions
7. Hackerearth
8. Codeforces
9. Neps Academy
10. CodeBench

Para o melhor entendimento do impacto e relevância de cada um dos juízes, foi realizado um levantamento do número de problemas de cada um deles, o que pode indicar o engajamento da comunidade, o tempo da plataforma no ar ou a frequência de atualização dos mantenedores; esse levantamento pode ser observado na Tabela 3.

Tabela 3 – Detalhes dos juízes online selecionados

Juiz online	**Número de Problemas**	**Data da consulta**
Online Judge	12 483	27/03/2022
Beecrowd	2 283	27/03/2022
Sphere Online Judge (SPOJ)	6 027	05/04/2022
PKU Judge Online	3 054	05/04/2022
CodeChef	3 363	05/04/2022
Google's Coding Competitions	224	05/04/2022
Hackerearth	1 526	05/04/2022
Codeforces	7 642	27/03/2022
Neps Academy	1 069	05/04/2022
CodeBench	—	05/04/2022

Fonte: o Autor.

A Tabela 3 mostra uma coluna com o título "Problemas", a qual indica o número de problemas estimado na Subseção 3.1.1 disponíveis para treino; na coluna "Consulta" é registrada a data em que o levantamento foi feito. Na última linha da tabela, o juiz online *"Code Bench"* não possui registro de problemas, pois a plataforma funciona com um sistema de turmas fechadas. Deste modo, não foi possível estimar o número de problemas.

4.1.2 Funcionalidades

Para os juízes selecionados foi realizado um levantamento de funcionalidades consideradas relevantes para o projeto GOJ. O objetivo desse levantamento é apresentar a motivação da criação de um novo juiz online, diante dos vários disponíveis. Para se tornar eficaz, esse novo juiz deve ter como funcionalidades: plataforma em português, visto que o público alvo do sistema é um público falante da língua portuguesa; código aberto, para que possíveis melhorias no sistema sejam efetuadas posteriormente por contribuidores interessados; possibilidade de enviar problemas customizados para a plataforma e poder de inserir tutoriais para as questões, para auxiliar os competidores na resolução posterior.

O Quadro 5 apresenta as funcionalidades relevantes para o projeto GOJ na coluna "Funcionalidade". As demais colunas listam os juízes por sigla. O Quadro assinala com um "x" o juiz que possui a funcionalidade. O nome dos juízes foi abreviado, porém, eles seguem a ordem da Tabela 3 e estão apresentados em nota.

Quadro 5: Funcionalidades Dos Juízes

Funcionalidade	**OJ**[1]	**BC**[2]	**SPOJ**[3]	**PKU**[4]	**CC**[5]	**GCC**[6]	**HH**[7]	**CF**[8]	**NA**[9]	**CB**[10]
Plataforma em português		x							x	x
Código aberto										
Possível enviar problemas customizados		x						x		
Possui tutoriais dos problemas					x	x	x	x	x	

Fonte: o Autor.

Conforme dito, "Plataforma em português" representa os juízes que possuem o site em português ou suporte para a língua. Os únicos juízes que possuem tal funcionalidade são o *Beecrowd* e o *Neps Academy*. Apesar de outras plataformas possuírem problemas na língua portuguesa, como o *Codeforces* por exemplo, estes problemas não são oficiais, ou seja, foram publicados por usuários na plataforma.

Já "Código aberto" indica os juízes que possuem código aberto para contribuidores externos, como no caso do juiz eletrônico "BOCA", por exemplo, o que possibilita a contribuição ou cópia do mesmo; porém, dentre os juízes pesquisados, nenhum possui esta funcionalidade.

"Possível enviar problemas customizados" é aplicado para juízes que permitem o envio de questões customizadas. Apesar de alguns juízes estarem marcados com essa funcionalidade, não é trivial o envio: é preciso preencher alguns pré-requisitos para enviar problemas ou entrar em contato diretamente com a equipe do site para que as questões sejam aprovadas. Apesar disso, foram marcados as plataformas com essa possibilidade.

Por fim, "Possui tutoriais dos problemas" identifica os juízes com suporte para a adição de tutoriais. Algumas das plataformas não tem esse suporte ou não tem a intenção de ter tutoriais nos problemas, como é o exemplo dos juízes com propostas primariamente educativas, voltados para o uso acadêmico em exercícios e provas. Em geral, as plataformas voltadas para competições e treinos de programação competitiva possuem essa funcionalidade.

Conforme mostrado no Quadro 5, nenhum dos juízes selecionados abrange todas as funcionalidades relevantes, sendo que o máximo de abrangência foram 2 funcionalidades

[1] Online Judge.
[2] Beecrowd.
[3] Sphere Online Judge.
[4] PKU JudgeOnline.
[5] CodeChef.
[6] Google's Coding Competitions.
[7] Hackerearth.
[8] Codeforces.
[9] Neps Academy.
[10] CodeBench.

em um total de 4, motivando a criação desse novo juiz online, cujo objetivo é implementar todas estas quatro funcionalidades.

4.2 Módulos do GOJ

O GOJ foi dividido em 4 partes: o Gamma Judge UI, que representa a interface; Gamma Judge API, que além de ser a API ainda funciona com um *worker*, ou seja, trabalha recebendo e processando informações das submissões; Gamma Judge Tools, assim denominado por conta do MOJ Tools[11], módulo responsável por julgar as submissões; e por fim, o Gamma Judge Admin, que também é uma interface, voltada para a edição dos eventos e problemas.

4.2.1 Gamma Judge API

Esse módulo da aplicação foi dividido em 3 partes: *API*, *Infrastructure* e *Workers*. A *API* é responsável por efetuar a comunicação externa da aplicação via *HTTP*; a *Infrastructure* é responsável por concentrar a infraestrutura do sistema, como a comunicação com o banco de dados; e os *Workers* são funções executadas periodicamente — nesse caso, para fazer o acesso com o sistema de filas.

Na Figura 15 é apresentado o esquema de pastas, cada um dos blocos representa um projeto do *.NET* e a organização das principais pastas ou arquivos. Na Api temos 2 pastas principais: *"Controllers"*, que armazena os *endpoints* expostos, onde é feita a comunicação de outros serviços; e *"Models"*, onde estão as estruturas de algumas requisições. *"Infrastructure"* possui em sua maioria funções para comunicação com o banco de dados; já *"Workers"*, possui um importante arquivo que faz a comunicação com o sistema de filas e recebe o veredicto das questões julgadas.

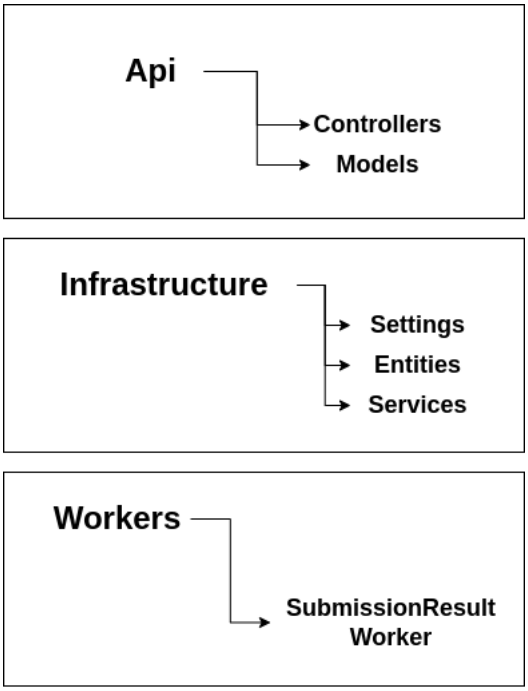
4.2.1.1 Módulo de problemas

O módulo de problemas é responsável pelos problemas e todas as funcionalidades relacionadas às informações expostas para a interface da aplicação. A Figura 16 representa os endereços expostos via API pelo módulo de problemas.

A Figura 16 apresenta em "Endereços expostos" os endereços que devem ser públicos, acessados pela interface via API; os "Endereços internos" listam funções da *API* realizados para uso interno, possibilitando o cadastro, edição e remoção de um problema sem a necessidade da comunicação direta com o banco de dados.

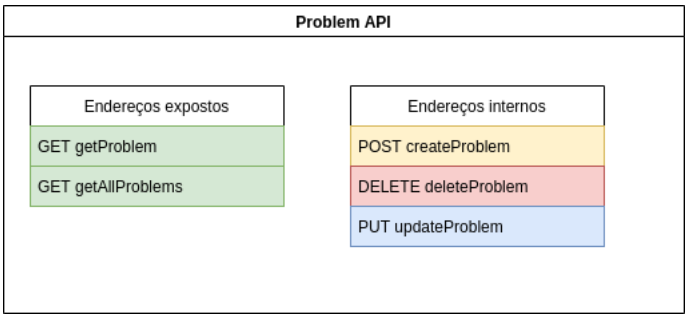
[11] Disponível em: <https://github.com/cd-moj/mojtools>.

Figura 15 – Esquema de pastas Gamma Judge API



Fonte: o Autor.

Figura 16 – Funções expostas do módulo de problemas



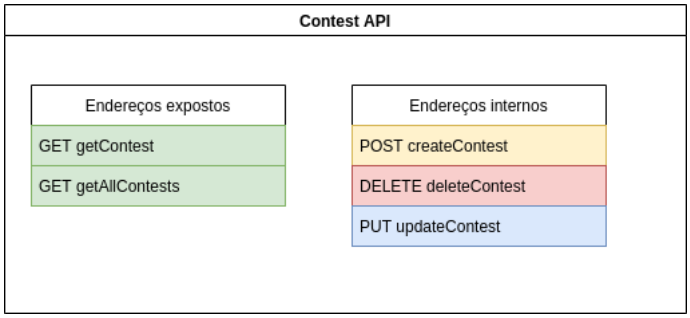
Fonte: o Autor.

4.2.1.2 Módulo de eventos

O módulo de eventos é responsável pelas informações expostas relacionadas aos eventos cadastrados no banco de dados. A Figura 17 exemplifica o funcionamento desse módulo da API.

O comportamento desse módulo é similar ao do módulo de problemas (Subseção 4.2.1.1). Ele expõe endereços internos, compostos por funções de criação, edição e remoção de eventos. Além disso, são disponibilizadas funções para recuperação de um único evento ou de vários eventos.

Figura 17 – Funções expostas do módulo de eventos



Fonte: o Autor.

4.2.2 Gamma Judge UI

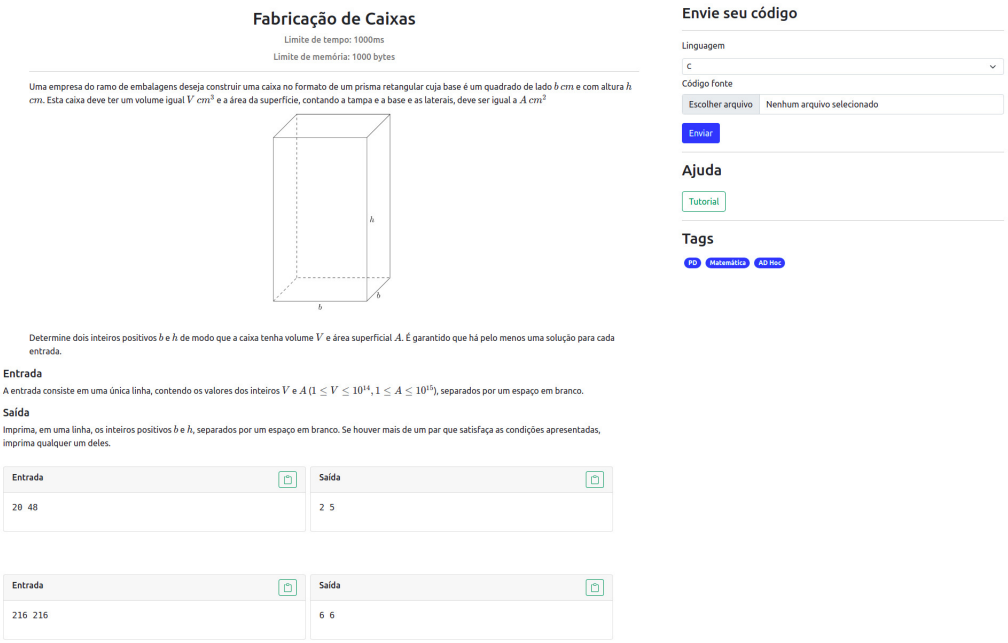
A interface do GOJ é um site, com páginas que apresentam as questões e os eventos. O objetivo da interface é organizar as informações da API para o usuário e permitir as submissões. O site pode ser dividido em 2 diferentes módulos: “Questões”, que representa as páginas que apresentam informações sobre os problemas cadastrados na plataforma; e “Eventos”, com as páginas responsáveis por exibir as informações sobre os eventos.

4.2.2.1 Questões

O módulo de questões da interface é responsável por disponibilizar ao usuário as informações a respeito das questões. Os dados disponíveis aqui são referentes aos dados da Subseção 4.2.1.1. Na Figura 18 é possível observar a organização e a maneira com que os dados são disponibilizados ao usuário.

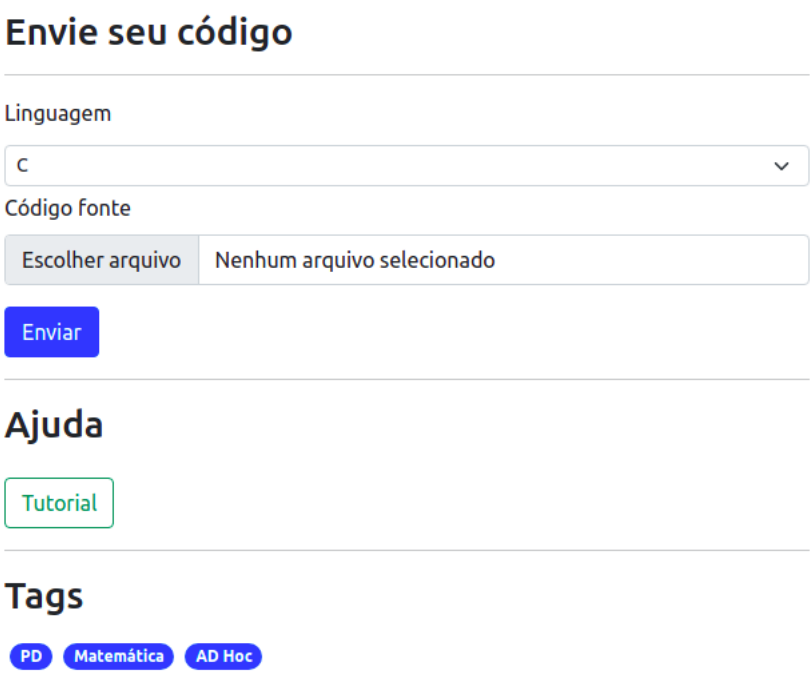
A interface das questões é dividida em algumas partes: o menu lateral, observável na Figura 19; as entradas e saídas, disponíveis na Figura 20; e o enunciado, disponível na Figura 21.

Figura 18 – Interface - informações da questão



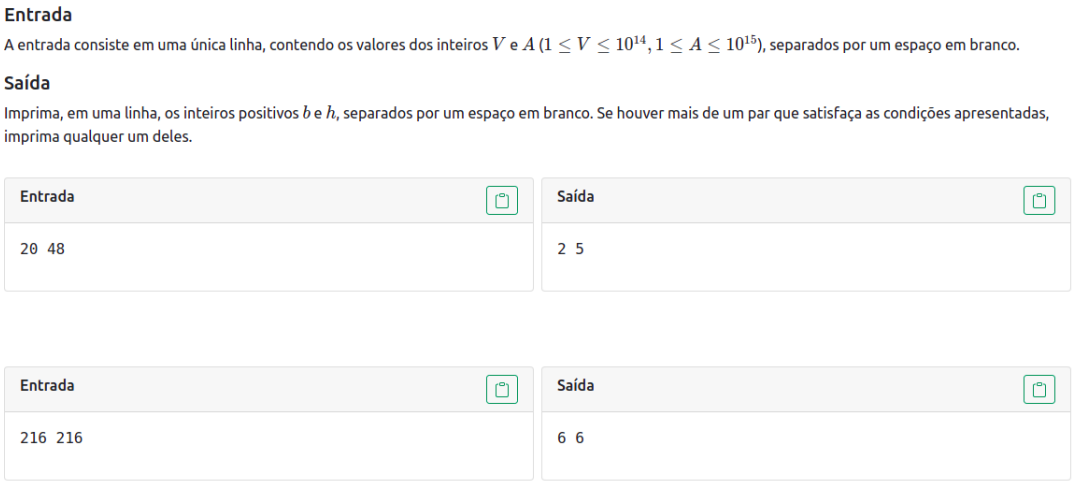
Fonte: o Autor.

Figura 19 – Interface — envio de questão



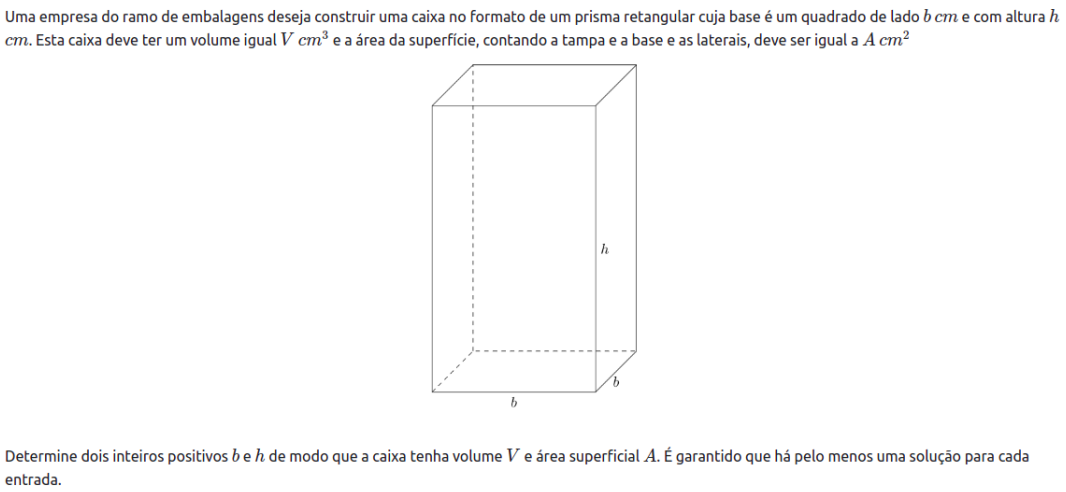
Fonte: o Autor.

Figura 20 – Interface — entradas e saídas da questão



Fonte: o Autor.

Figura 21 – Interface — Enunciado da questão

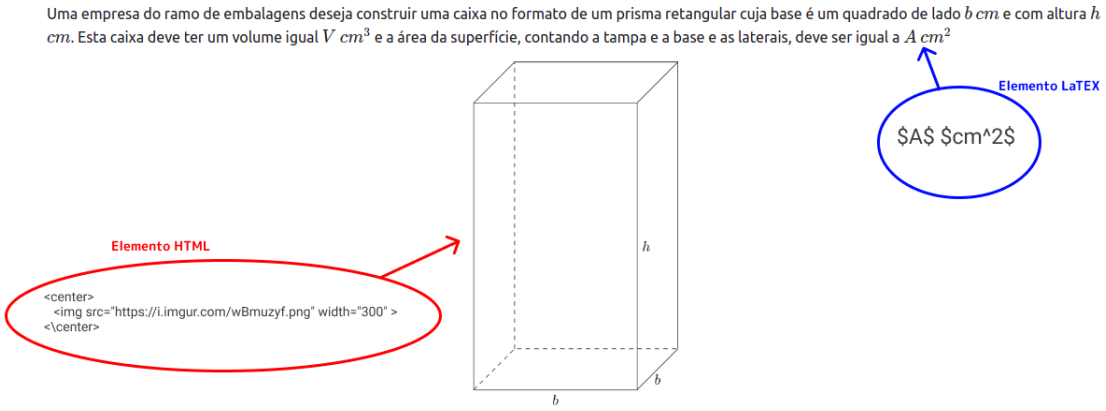


Fonte: o Autor.

Em juízes online, elementos não textuais são usados para dar mais riquezas e detalhes aos problemas. Assim sendo, é comum a utilização de imagens, fórmulas matemáticas e elementos para enfatizar trechos dos problemas, como textos em negrito ou itálico. Além de apresentar as informações do problema, a interface também é responsável por representar elementos da API[12], em HTML[13] ou LaTeX[14], em elementos não textuais; a Figura 22 exemplifica a utilização desses recursos.

[12] Ver em 3.3.4.
[13] RAGGETT, Dave et al. HTML 4.01 Specification. W3C recommendation, v. 24, 1999.
[14] LAMPORT, Leslie. LaTeX. Company Cyfronet, 1991.

Figura 22 – Interface – demonstração de elementos não textuais



Fonte: o Autor.

A Figura 22 foi renderizada automaticamente a partir de um texto plano vindo da API. O texto utilizado está disponível na Figura 23, tal como o resultado da sua renderização realizada pela aplicação, disponível na Figura 21. Como apresentado na Figura 22, neste enunciado foram utilizados elementos da linguagem LaTeXem azul e elementos HTML em vermelho.

Figura 23 – Texto bruto enunciado

```
Uma empresa do ramo de embalagens deseja construir uma caixa no formato de um prisma retangular
cuja base é um quadrado de lado $b$ $cm$ e com altura $h$ $cm$. Esta caixa deve ter um volume igual
$V$ $cm^3$ e a área da superfície, contando a tampa e a base e as laterais, deve ser igual a
$A$ $cm^2$

<br/>
<center>
  <img src="https://i.imgur.com/wBmuzyf.png" width="300" >
</center>
<br/>

Determine dois inteiros positivos $b$ e $h$ de modo que a caixa tenha volume $V$ e área
superficial $A$. É garantido que há pelo menos uma solução para cada entrada.
```

Fonte: o Autor.

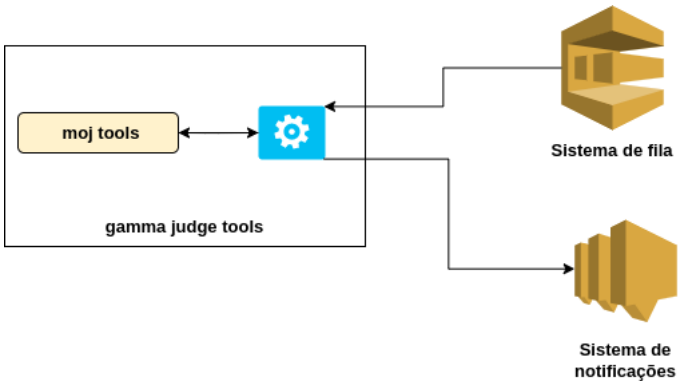
4.2.2.2 Eventos

O módulo de eventos concentra as informações referentes a um determinado evento da Maratona UnB de Programação. Nesse módulo está concentrada a lista de questões do evento e a data em que o evento ocorreu. Os problemas de um evento podem ser reaproveitados em outros, portanto, também é apresentada a origem do problema, que pode ser original do evento ou apresentar onde ele teve origem. A Figura 24 mostra a maneira em que as informações são disponibilizadas ao usuário na tela.

Figura 24 – Interface – Detalhes de um evento

Fonte: o Autor.

Figura 25 – Gamma judge tools – Diagrama



Fonte: o Autor.

Como apresentado na Figura 24, são disponibilizadas informações do rótulo do problema[15], título, marcações[16] e o evento original da questão. Diferentemente do que mostra a Figura 24, é possível que nos eventos os problemas sejam marcados com rótulos diferentes das letras do alfabeto. Além disso, o campo "Evento original" pode mostrar um botão para o evento em que aquele problema foi criado, caso ele tenha sido reaproveitado nessa maratona.

4.2.3 Gamma Judge Tools

Essa parte do sistema é responsável por julgar as submissões enviadas e retornar o veredito; para isso foi utilizada a ferramenta de código aberto MOJ Tools[17], responsável por compilar o código recebido, gerar um programa executável, e executá-lo em uma lista de testes (Seção 2.3).

[15] O rótulo é representado na Figura 24 com um ícone de balão.
[16] As marcações são representadas na Figura 24 como *tags*.
[17] Disponível em: <https://github.com/cd-moj/mojtools>.

Na Figura 25, o GOJ lê do sistema de filas uma mensagem contendo as informações da submissão. Após o uso do MOJ Tools para julgar, o veredito é retornado para um sistema de notificações; com isso, as outras aplicações são responsáveis por lidar com a notificação enviada e processar o veredito da submissão.

4.2.4 Gamma Judge Admin

O módulo *Gamma Judge Admin* permite o uso de funções de leitura e escrita da API, com mecanismos para ler, criar, editar e deletar um evento ou um problema. Esse módulo permite a manutenção e edição do conteúdo dos problemas armazenados. Sua estrutura se divide em 3 páginas: *"HomePage"*, que possui apenas botões para as outras páginas; *"ProblemPage"*, que possui os mecanismos descritos, direcionados para os problemas; e *"ContestPage"*, que se assemelha à *"ProblemPage"*, porém direcionada para eventos.

4.2.4.1 Problem Page

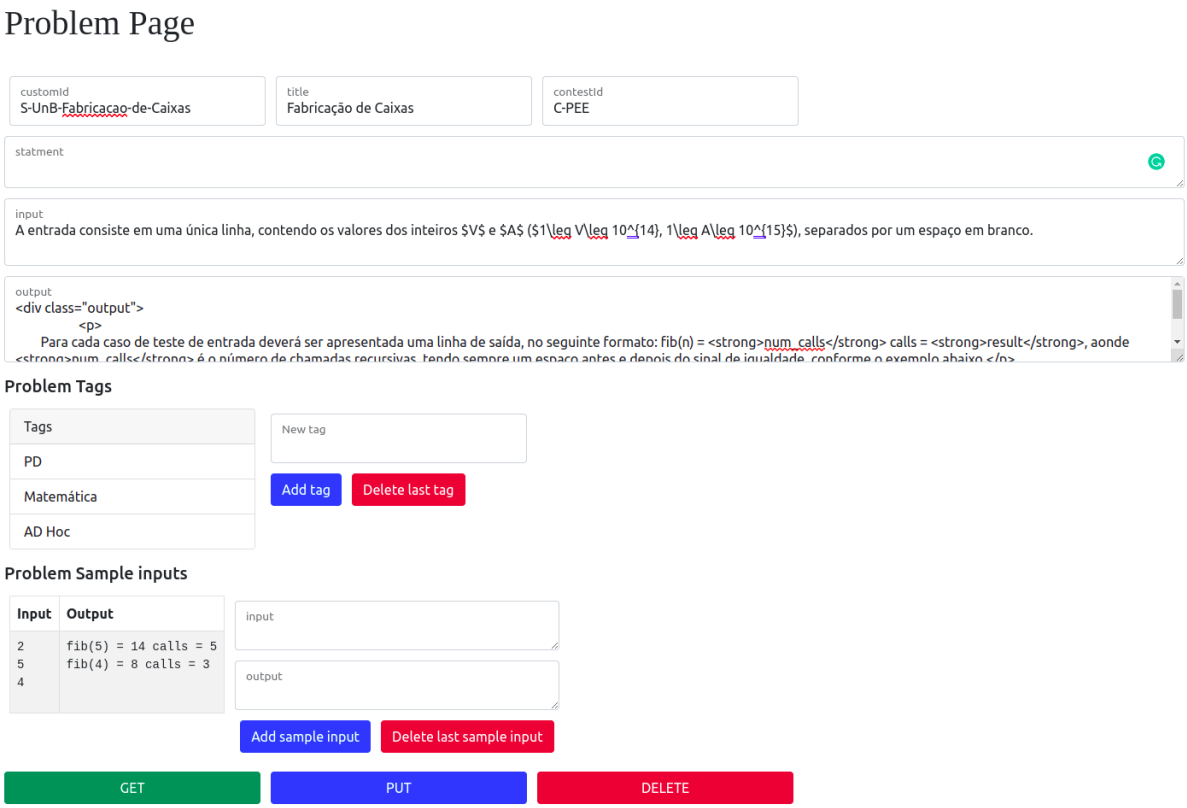
A página de problemas acessa a API apresentada na Subseção 4.2.1, que tem como base o *customId*, uma sequência de caracteres que identifica cada um dos problemas unicamente. A Figura 26 ilustra a página descrita: com o botão *"GET"*, em verde, e o *customId* preenchido, é possível recuperar as informações de um problema; *"PUT"*, em azul, cria, caso não exista, ou sobrescreve as informações do problema; *"DELETE"*, em vermelho, apaga o problema.

4.2.4.2 Contest Page

A página de eventos tem o comportamento similar à página de problemas, com os mesmos botões e mesmo funcionamento, porém com menos informações. Na página de eventos é possível adicionar ou remover problemas ao evento utilizando o *customId* do problema e o *identifier* será o identificador do problema naquele evento, como mostra a Figura 27.

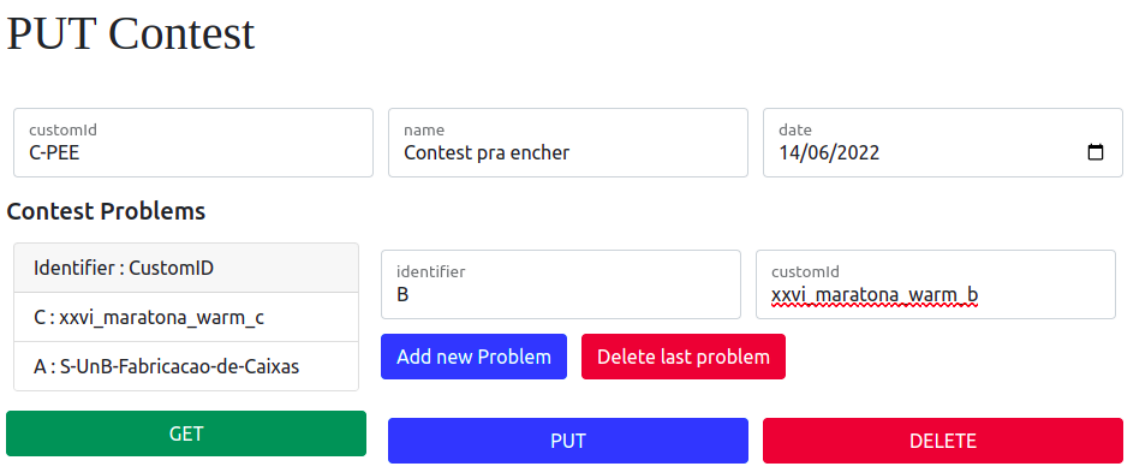
Conforme descrito, a ferramenta possui as funcionalidades almejadas, apresentadas na subseção 4.1.2, possibilitando a sua utilização para armazenamento das questões das Maratonas UnB de Programação. Com isso, o GOJ provê as funcionalidades não observadas nos juízes online pesquisados, fato que justifica seu desenvolvimento e adoção.

Figura 26 – Gamma judge Admin – ProblemPage



Fonte: o Autor.

Figura 27 – Gamma judge Admin – ContestPage



Fonte: o Autor.

5 Conclusão

Conforme visto ao longo do trabalho, juízes onlines são ferramentas importantes para treinos de programação. Eles possuem problemas que podem ser oriundos de maratonas de programação, e assim, permitem que os usuários testem suas soluções.

Existem diversos juízes disponíveis para treinos de programação. Alguns deles foram analisados no levantamento realizado, o que permitiu identificar a viabilidade das plataformas e as funcionalidades disponíveis. Porém, como observado na subseção 4.1.2, nenhum dos juízes da amostra possuía todas as funcionalidades consideradas relevantes para um projeto como o Gamma Online Jugde.

Então, objetivando reunir as questões das maratonas UnB de programação, o GOJ permite que os usuários acessem os problemas de eventos passados, e testem suas soluções para tais problemas. Suas funcionalidades incluem: a separação das questões por eventos, uma plataforma em português, o envio dos problemas das maratonas, a adição de tutorias nos problemas - funcionalidades estas não reunidas nos juízes levantados.

Desse modo, o novo juiz online criado atende todos os requisitos do projeto, e preenche, de certo modo, uma lacuna existente na seara dos juízes online.

Ademais, como mostra a seção 4.2, a aplicação foi dividida em módulos, contendo um juiz eletrônico, uma API, uma interface de usuário e uma interface para os administradores, corroborando o atendimento dos requisitos e possibilitando o uso eficiente da plataforma por usuários e administradores.

Com isso, conclui-se que o juiz criado atende os objetivos e consegue armazenar as questões passadas das maratonas UnB de programação. As questões são divididas por eventos, possuem tutoriais e todo o projeto é open source, melhorando a possibilidade de suporte de terceiros.

Referências

CAMPOS, C. P. de; FERREIRA, C. E. Boca: um sistema de apoio a competiçoes de programaçao. 2004. Disponível em: <http://www.eeecs.qub.ac.uk/~c.decampos/publist/papers/decampos2004d.pdf>. Citado na página 31.

DMITRY, N.; MANFRED, S.-S. On micro-services architecture. *International Journal of Open Information Technologies*, v. 2, n. 9, 2014. Disponível em: <https://cyberleninka.ru/article/n/on-micro-services-architecture>. Citado na página 34.

FRANCISCO, R.; JÚNIOR, C. P.; AMBRÓSIO, A. P. Juiz online no ensino de programação introdutória-uma revisao sistemática da literatura. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [s.n.], 2016. v. 27, n. 1, p. 11. Disponível em: <http://www.br-ie.org/pub/index.php/sbie/article/view/6676>. Citado 2 vezes nas páginas 24 e 33.

GHEBREMICAEL, E. S. *Transformation of REST API to GraphQL for OpenTOSCA*. Dissertação (Mestrado), 2017. Disponível em: <https://elib.uni-stuttgart.de/handle/11682/9369>. Citado na página 35.

HADLEY, M. J. *Web application description language (WADL)*. Sun Microsystems, Inc., 2006. Disponível em: <https://dl.acm.org/doi/pdf/10.5555/1698142>. Citado na página 35.

HALIM, S. et al. *Competitive programming 3*. [S.l.]: Citeseer, 2013. Citado na página 27.

ICPC. *About ICPC*. 2021. Disponível em: <https://icpc.global/regionals/abouticpc>. Citado na página 28.

MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 35.

OBI2021. *Sobre a OBI*. 2021. Disponível em: <https://olimpiada.ic.unicamp.br/info/>. Citado na página 23.

PEREIRA, T. G. et al. Utilização de problemas da maratona de competição de programação e juízes eletrônicos como estratégia de ensino em um curso de graduação em engenharia de software. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [s.n.], 2016. v. 27, n. 1, p. 210. Disponível em: <http://br-ie.org/pub/index.php/sbie/article/view/6701>. Citado na página 34.

RAWAT, P.; MAHAJAN'S, A. N. React js: A modern web development framework. *International Journal of Innovative Science and Research Technology*, v. 5, n. 11, 2020. Disponível em: <https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf>. Citado na página 35.

REVILLA, M. A.; MANZOOR, S.; LIU, R. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, Institute of Mathematics and

Informatics, v. 2, n. 10, p. 131–148, 2008. Disponível em: <https://ioinformatics.org/journal/INFOL035.pdf>. Citado na página 28.

RICHARDSON, L. et al. *RESTful Web APIs: Services for a Changing World*. [S.l.]: "O'Reilly Media, Inc.", 2013. Citado na página 35.

UNB. *Maratona de Programação*. 2022. Disponível em: <http://maratona.unb.br/>. Citado na página 32.

YOUNG, R. R. Recommended requirements gathering practices. *CrossTalk*, Citeseer, v. 15, n. 4, p. 9–12, 2002. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.643.8197&rep=rep1&type=pdf>. Citado na página 45.

Apêndices

APÊNDICE A – Scripts

Listing A.1 – Script para coletar número de problemas — Online Judge

```
import urllib.request
import urllib.error
import urllib.parse
from bs4 import BeautifulSoup
import threading

sem = threading.Semaphore()

BASE_URL = 'https://onlinejudge.org/'
MAX_THREADS = 10


def get_soup(url):
    html = urllib.request.urlopen(url).read()
    return BeautifulSoup(html, 'html.parser')


def isProblemCategory(tag):
    try:
        return 'sectiontableentry' in tag['class'][0]
    except:
        return False


def getProblemsCategory(soup):
    return list(filter(lambda tag: isProblemCategory(tag),
                       soup.find_all('tr')))


links_set = set()


def getCategoryLinks(soup):
```

```
    link_list = []
    categories = getProblemsCategory(soup)
    for category in categories:
        for tag in category.find_all('a'):
            try:
                link = tag['href']
                if link not in links_set:
                    sem.acquire()
                    link_list.append(link)
                    links_set.add(link)
                    sem.release()
            except:
                pass
    return link_list


links = ['index.php?option=com_onlinejudge&Itemid=8']
threads = []


def processLink(url, links):
    soup = get_soup(url)
    otherLinks = getCategoryLinks(soup)
    sem.acquire()
    links += otherLinks
    sem.release()


while len(links) > 0:
    selected_link = links.pop(0)
    if 'https://www.udebug.com/UVa' in selected_link:
        continue
    if 'page=show_problem' in selected_link:
        print(selected_link)
        continue
    url = BASE_URL + selected_link

    t = threading.Thread(target=processLink, args=(url, links))
    t.start()
```

```
        threads.append(t)
        if len(links) == 0 or len(threads) > MAX_THREADS:
            threads.pop(0).join()

for t in threads:
    t.join()
```

Listing A.2 – Script para coletar número de problemas — Codeforces

```
problems_url="https://codeforces.com/api/problemset.problems"
response=$(curl --location --request GET $problems_url)
echo $response | jq -r ".result.problems | length"
```