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

BROMS: *Brazilian Online Marathon Scoreboard*

Autor: Rafael Makaha Gomes Ferreira
Orientador: Professor Dr. Edson Alves da Costa Júnior

Brasília, DF
2021

Rafael Makaha Gomes Ferreira

BROMS: *Brazilian Online Marathon Scoreboard*

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: Professor Dr. Edson Alves da Costa Júnior

Brasília, DF

2021

Rafael Makaha Gomes Ferreira

BROMS: *Brazilian Online Marathon Scoreboard*/ Rafael Makaha Gomes Ferreira. – Brasília, DF, 2021-

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

Orientador: Professor Dr. Edson Alves da Costa Júnior

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

1. Scoreboard. 2. Competition. I. Professor Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. BROMS: *Brazilian Online Marathon Scoreboard*

CDU 02:141:005.6

Rafael Makaha Gomes Ferreira

BROMS: *Brazilian Online Marathon Scoreboard*

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, 28 de outubro de 2021 – Data da aprovação do trabalho:

Professor Dr. Edson Alves da Costa Júnior
Orientador

Professor Dr. Bruno Cesar Ribas
Convidado 1

Professor MsC. Daniel Saad Nogueira Nunes
Convidado 2

Brasília, DF
2021

Dedico este trabalho à minha mãe
que sempre acreditou em mim.

Agradecimentos

Agradeço primeiramente à minha mãe que sempre apoiou minhas decisões e cuidou de mim. Agradeço, também, aos meus queridos amigos Nicholas Muller e Beatriz Pontes, os quais carrego em meu peito com muito carinho e me ajudam diariamente a seguir com a mente sã. À minha parceira e companheira Cleia Oliveira pelo enorme carinho e apoio. Aos amigos Humberto Salviolo, Pedro Matos, Ian Goes e Wesley Adriann pela duradoura amizade ao longo de tantos anos. Aos meu companheiros de graduação João Vitor Morandi e Pedro Langendorf pela força e apoio. Agradeço, também, ao meu orientador Prof. Edson Alves, pelas inúmeras vezes que me motivou a continuar firme e forte na graduação. E à todos os que de alguma forma me ajudaram nesta jornada.

Resumo

Diante do crescimento da popularidade de eventos voltados à programação competitiva, criou-se a necessidade de se transmitir as informações destes eventos para o público. Em competições as informações são transmitidas por meio de placares. O presente trabalho trata-se do desenvolvimento de um placar dinâmico com minimização de dependências para simplificar suas etapas de execução. Com foco no funcionamento em navegadores web, o BROMS realiza a renderização das informações de *contests* diversos. O Objetivo principal seria auxiliar os organizadores de eventos de maratona de programação a realizar uma apresentação em tempo real do andamento do evento. Esta ferramenta se alimentará de informações provindas de requisições web de um servidor capaz de prover os dados necessários para a montagem e atualização do placar como informações das equipes participantes, suas pontuações e suas penalidades.

Palavras-chave: placar. juiz eletrônico. programação competitiva. desenvolvimento web.

Abstract

Facing the growing popularity of competitive programming events, the need to transmit information from these events to the public arose. During competitions the infomations are shared throw scoreboards. The following work is about developing a dinamic scoreboard with minimum dependencies to simplify its execution steps. Focused on running on web browsers, BROMS performs the information rendering of miscellaneous contests. The main goal is to assist organizers of programming marathon to perform a real-time presentation of the event's progress. This tool will feed information from web requests from a server capable of providing the necessary data needed to assemble and update the scoreboard such as information on teams members, their scores and penalties.

Key-words: scoreboard. eletronic judge. competitive programming. web development.

Lista de ilustrações

Lista de Quadros

Lista de códigos fonte

Lista de abreviaturas e siglas

BROMS	Brazilian Online Marathon Scoreboard
API	Application Programming Interface
ICPC	*International Collegiate Programming Contest*
OBI	Olimpíada Brasileira de Informática
SBC	Sociedade Brasileira de Computação
SVG	*Scalable Vector Graphics*
HTML	*Hyper Text Markup Language*

Sumário

Introdução

Competições comumente trazem placares para apresentar pontuações, posições e informações gerais de seus participantes. Em competições de programação não é diferente, os espectadores demandam informações sobre a competição que se ocorre. Isso faz com que as organizações destas competições necessitem de placares que apresentem os dados e informações dos competidores bem como prendam a atenção do público.

Em uma competição de programação os participantes são dispostos em equipes para responder questões de computação de diferentes níveis de dificuldade. Para verificação e validação das respostas são utilizados sistemas como juízes eletrônicos. Estes juízes promovem o controle dos acertos e erros das questões dos participantes. São estes sistemas que fornecem os dados necessários para se montar placares para apresentação do desempenho das equipes. Ferramentas como Maratona Live (SEGUNDO, 2021), o Maratona Animeitor (OSHIRO, 2021) e o Maratona Rustrimeitor (WUERGES, 2021) foram geradas para apresentação destas informações. Entretanto, há possibilidade de melhoras em questão de instalação e portabilidade de código nestas aplicações.

A necessidade de instalação de todos os recursos de uma linguagem de programação e pacotes adicionais desta linguagem para o funcionamento de um software são pontos que podem melhorar nestas ferramentas, pois dependendo da largura de banda do usuário, o tempo de instalação da ferramenta pode vir a consumir muito tempo. Utilizar uma linguagem de programação comumente presente nos computadores da maioria dos usuários facilita na portabilidade da ferramenta. Consoante a isso, remover as dependências não nativas à linguagem utilizada também ajuda no processo de instalação da aplicação bem como na confiabilidade de que algum pacote não manutenido impeça o seu funcionamento.

Este trabalho de conclusão de curso propõe o desenvolvimento de um placar virtual de simples instalação, boa portabilidade e de mínimas dependências, sendo capaz de funcionar na maioria dos computadores por meio de um simples navegador de internet capaz de realizar requisições web para se alimentar de informações provindas de um servidor que forneça informações sobre a competição em um formato específico. Este trabalho visa, também, facilitar a manutenibilidade e desenvolvimento do código-fonte do placar para que futuros desenvolvedores que venham a implementar novas funcionalidades com uma maior facilidade para isso.

Objetivos

Este trabalho de conclusão de curso propõe um software denominado *Brazilian Online Marathon Scoreboard* (BROMS) que apresente, em tempo real, os dados e informações de uma competição de programação, por meio de uma estrutura a nível de código que seja tanto escalável quanto manutenível para que novas funcionalidades possam ser implementadas sem dificuldades após o término do ciclo deste projeto.

Os objetivos específicos são:

- minimizar dependências: a aplicação deve ter o mínimo de dependências para funcionar;
- redimensionar automaticamente a aplicação para que a mesma seja responsiva para diferentes tamanhos de telas;
- apresentar listagem de times participantes e suas pontuações;
- atualizar pontuações e posições dos times participantes;
- indicar classificados ao final da competição;
- indicar medalhistas;
- ter suporte para grande número de times.

Estrutura do Trabalho

Este trabalho está organizado em quatro capítulos. No Capítulo 1, Fundamentação Teórica, são apresentados trabalhos relacionados às definições e conceitos presentes no projeto. No Capítulo 2, Metodologia, são apresentados os métodos e ferramentas adotados para a produção deste projeto. No Capítulo 3, Resultados, são apresentados os resultados obtidos na implementação do projeto. E, por fim, no Capítulo 4, Considerações Finais, são apresentadas uma análise do projeto, seus impactos e previsões para próximas evoluções do trabalho.

1 Fundamentação Teórica

Este capítulo tem por finalidade definir e explicar os principais termos necessários para o entendimento do projeto, no qual a Seção 1.1 define e apresenta o funcionamento de uma Maratona de Programação, a Seção 1.2 define e explica um juiz eletrônico, a Seção 1.3 define o BOCA, a Seção 1.4 define e explica um juiz online e a Seção 1.5 faz um levantamento geral sobre desenvolvimento de aplicações web.

1.1 Competição de Programação

As competições de programação são eventos voltados à solução de problemas relacionados à algoritmos. Os participantes se reúnem para resolver um caderno de questões o mais rápido possível. As equipes submetem suas respostas à correção automática podendo acertar ou errar a questão. Em caso de acerto, a equipe recebe um ponto. Já em caso de erro, a equipe acumula uma penalidade de tempo por questão. Ao acertar uma questão com penalidade acumulada, a equipe recebe seu ponto e o acúmulo de penalidade é somado à penalidade total da equipe. A equipe que resolver mais questões tendo menor penalidade acumulada ao fim do tempo determinado para a maratona se torna vitoriosa. Comumente, nestas maratonas os vitoriosos podem receber três tipos de medalhas – ouro, prata ou bronze – a depender de sua classificação ao final da maratona. “As competições de programação constituem uma metodologia já difundida e bastante utilizada em todo o mundo” (KIKUTI, 2015).

Estas competições surgem na década de 1970 por meio do *International Collegiate Programming Contest* (ICPC), o qual rapidamente cresceu em popularidade em meio às universidades ao redor do mundo. Sendo segmentada em fases eliminatórias até ocorrer a final mundial, a qual, em 2019, envolveu 58.963 participantes de 3.406 universidades de 104 países (ICPC, 2021).

Além do ICPC existem outras organizações que se utilizam do modelo de competições, ou *contests*. É o caso do Google Code Jam, organizado pela empresa Google. Neste evento existem três etapas classificatórias e uma etapa final. A primeira etapa é subdividida em três subetapas, as quais classificarão os primeiros 1.500 participantes de cada sub-etapa. Estes classificados competirão na segunda etapa pelas primeiras 1.000 posições. A competição na terceira etapa selecionará os primeiros 25 colocados para participar da *Virtual World Finals* pelo prêmio final (GOOGLE, 2021).

No Brasil, a Olimpíada Brasileira de Informática (OBI) realiza eventos de competição de programação voltados para alunos do ensino fundamental, médio e cursantes

do primeiro ano do ensino superior OBI (2021). Os participantes competem em três fases pelas medalhas de ouro, prata e bronze entregues aos vencedores.

Outro evento de programação competitiva bastante popular é a Maratona SBC de Programação, a qual existe desde o ano ano de 1996. Sendo parte da América Latina do ICPC, os alunos das universidades se reúnem em times para competir por vagas para as próximas fases da competição. Em 2019, a XXIV Maratona SBC de Programação envolveu 47 sedes com 726 times de 224 escolas (SBC, 2021).

1.2 Juízes Eletrônicos

Os juízes eletrônicos são sistemas desenhados para verificar e validar respostas de questões implementadas em códigos-fonte (ZHIGANG et al., 2001). Primeiramente, no caso de códigos em linguagens compiladas, o juiz eletrônico executa a compilação do código fonte e, caso exista algum problema, retorna erro de compilação. Em linguagens interpretadas, caso ocorra algum problema, o juiz eletrônico retorna erro em tempo de execução. Em seguida, o código-fonte recebe as entradas e retorna as suas saídas. Estes códigos devem responder as questões dentro de um tempo limite determinado por questão, caso a execução ultrapasse este tempo limite, o juiz retorna tempo limite excedido. O juiz eletrônico, então, realiza a comparação das saídas obtidas com as saídas esperadas. Caso haja alguma diferença entre as saídas comparadas, o juiz eletrônico indicará que o código submetido não resolve todos os casos de teste. O código submetido aos testes do sistema só será aceito caso passe em todos os casos.

Um exemplo de juiz eletrônico é o BOCA (*BOCA Online Contest Administrator*), o qual é um administrador de competições e possui um módulo responsável por desempenhar o papel de juiz eletrônico. Amplamente utilizado no Brasil por instituições de ensino e por organizações de eventos de programação competitiva como a Maratona SBC. Esta ferramenta é capaz de verificar se um código submetido retorna as saídas esperadas baseando-se nas entradas inseridas. Auxiliando na correção de avaliações de professores e respostas de questões em maratonas, o juiz eletrônico pode apresentar diferentes mensagens para o resultado da avaliação do código-fonte como é mostrado no Quadro 1.1.

Quadro 1.1: Possíveis retornos do juiz eletrônico para submissões

Abreviação	**Significado**	**Descrição**
AC	*Accept*	Os retornos obtidos condizem com os retornos esperados
WA	*Wrong Answer*	Ao menos um retorno não condiz com os retornos esperados
TLE	*Time Limit Exceeded*	Tempo limite para execução das respostas foi ultrapassado
RE	*Runtime Error*	Erro durante execução do código
PE	*Presentation Error*	Os retornos obtidos não se encontram no padrão de apresentação requerido pela questão
CE	*Compilation Error*	Erro durante compilação do código

1.3 BOCA

O BOCA é um administrador de competições desenvolvido para ser usado na Maratona SBC de Programação. Esta ferramente pode, também, ser utilizado didaticamente para auxiliar disciplinas que utilizem submissão e correção de trabalhos durante as aulas (CAMPOS; FERREIRA, 2004).

O BOCA funciona seguindo o mesmo fluxo de outros juízes. A Figura 1 mostra as etapas realizadas em cima de um código-fonte submetido aos testes do seu módulo de juiz eletrônico. Primeiramente, no caso de códigos compilados, o BOCA realiza a compilação do código-fonte submetido. Caso haja algum problema na compilação, retorna *compilation error*. Caso não haja nenhum erro, são executados testes em cima do programa gerado. No caso de códigos interpretados, o BOCA inicia diretamente a execução dos testes para se obter as saídas esperadas. Entretanto, caso haja algum problema com o código, retorna erro em tempo de execução. Em seguida, caso não haja nenhum erro anteriormente, comparam-se as saídas obtidas pelo programa com as saídas esperadas. Caso ocorra alguma divergência, o BOCA retornará uma mensagem de erro de acordo com o Quadro 1.1.

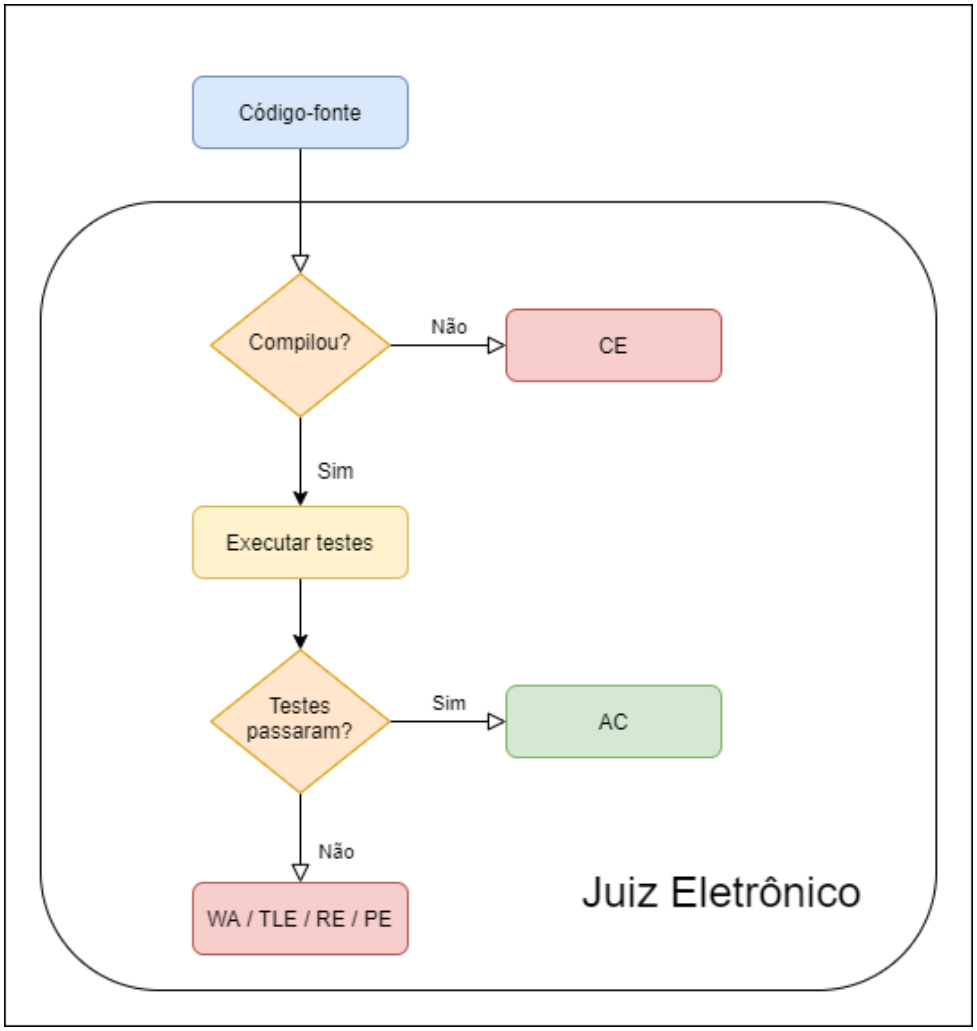


Figura 1 – Fluxo de validação de submissão

O BOCA é capaz de exportar informações do estado atual do *contest*. Esta exportação ocorre por meio de dois arquivos principais: *contest* e *runs*. O primeiro apresenta informações do *contest* – como indicado na Figura 2 – como tempo total, penalidade por erro, quantidade de questões; bem como informações e pontuações de cada time. O segundo arquivo apresenta um histórico de todas as submissões realizadas durante a competição, como apresentado na Figura 3. Estas informações possibilitam a montagem gráfica de placares em aplicações independentes do BOCA como, por exemplo, o BROMS – foco deste trabalho – e as aplicações Maratona Live (SEGUNDO, 2021), Maratona Animeitor (OSHIRO, 2021) e Maratona Rustrimeitor (WUERGES, 2021).

Figura 2 – Trecho do arquivo contest fornecido pelo BOCA

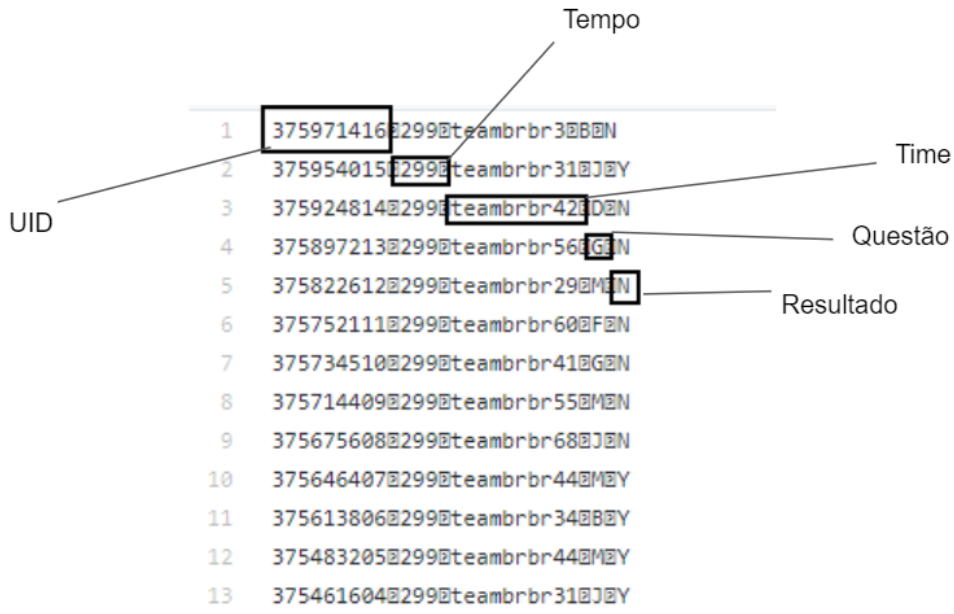


Figura 3 – Trecho do arquivo runs fornecido pelo BOCA

1.4 Juízes Online

Os juízes online são a plataforma para os usuários terem acesso aos problemas, provas e submissões (MARANHÃO; CARVALHO, 2017). Os juízes online são muito populares, pois alguns deles realizam competições periodicamente. Dentro destas plataformas, as competições geram pontuações internas para os usuários, o que aumenta ainda

mais a motivação dos mesmo em solucionar as questões apresentadas.

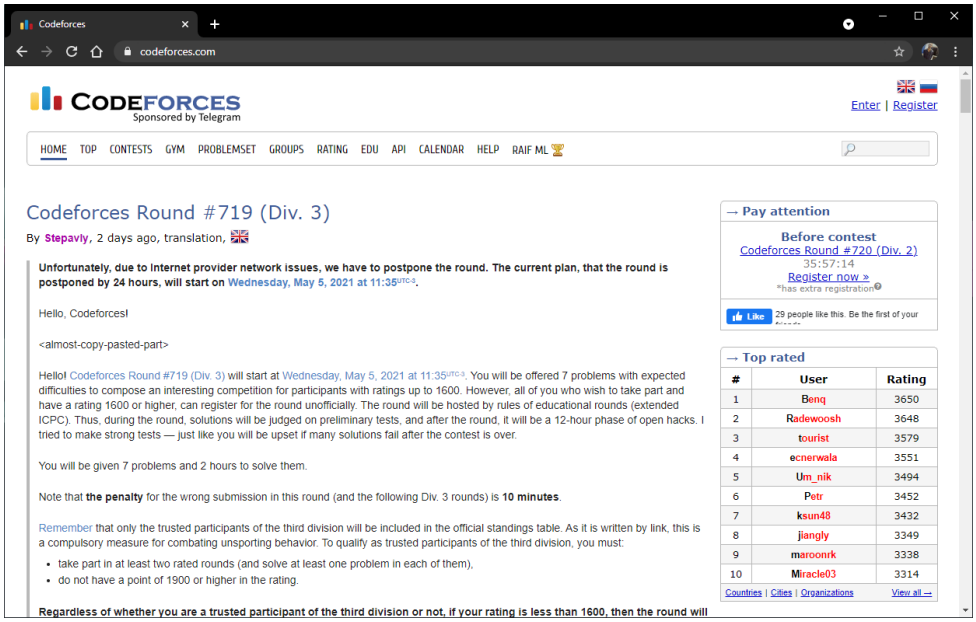


Figura 4 – Página principal do CodeForces

O CodeForces (2021) é um dos juízes online mais populares no mundo atualmente. Algumas das funcionalidades fornecidas pela plataforma se encontram: apresentação de ao menos dois *contests* por semana, código aberto de todas as submissões dos usuários, possibilidade de participação virtual de maratonas já encerradas e possibilidade de *contests* personalizados. A Figura 4 mostra a página principal do CodeForces.

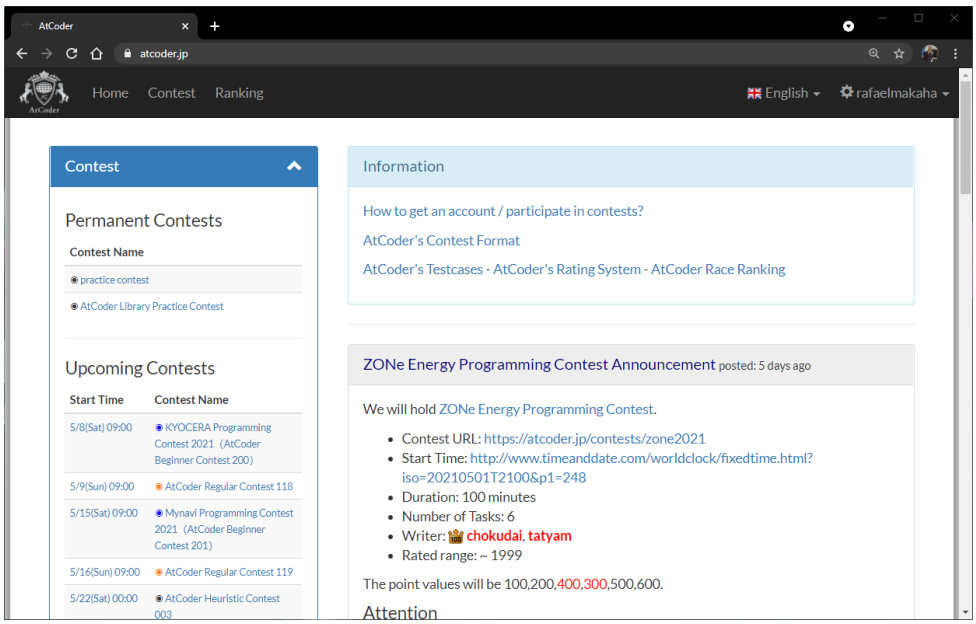


Figura 5 – Página principal do atCoder

Outra plataforma bastante popular mundialmente é a AtCoder (2021). Esta plataforma é bastante popular no Japão, onde empresas de tecnologia patrocinam *contests*

à procura de talentos na área de programação. A Figura 5 mostra a página principal do AtCoder.

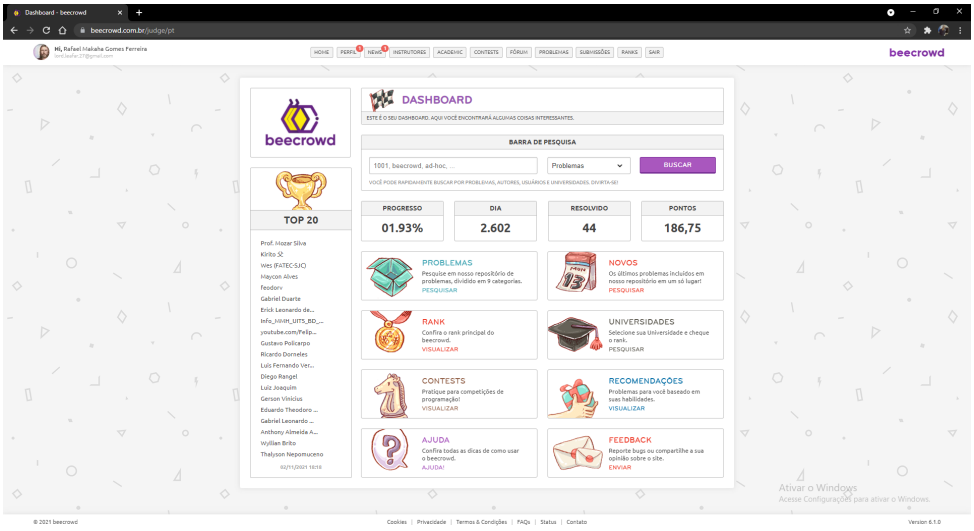


Figura 6 – Página principal do BeeCrowd

No Brasil, um dos juízes onlines mais utilizados é o BeeCrowd (2021). Esta plataforma apresenta um sistema único voltado para professores, onde os mesmos podem adicionar exercícios tanto novos quanto já existentes na plataforma em módulos e disciplinas. A Figura 6 mostra a página principal do *BeeCrowd*.

Existem, também, diversas outras plataformas com o mesmo foco em competições de programação como o CD-MOJ Ribas (2021), CodeChef (2021), UVA (2021), Spoj (2021) e CodeCup (2021). Todas com suas respectivas comunidades voltadas para a resolução de problemas de computação. Estas plataformas se utilizam de sistemas web para fornecerem seus serviços às organizações de eventos e seus participantes.

1.5 Desenvolvimento Web

O desenvolvimento de aplicações web, assim como diversos artefatos, visam facilitar e melhorar a vida das pessoas. Atualmente, existem milhares de serviços voltados desde entretenimento até como meio de ensino e fonte de informações. Sistemas de *streaming*, ligações e vídeo chamadas pela internet, bancos digitais, compras online e muitos outros serviços foram construídos, cada qual com suas metodologias e arquiteturas únicas, seguindo princípios de desenvolvimento de aplicações web. Estas aplicações web comumente seguem um padrão de desenvolvimento por meio da modularização de suas dependências, separando suas funcionalidades em servidores *back-end* e servidores *front-end*. A Figura 7 apresenta, de maneira simplificada, a comunicação entre os serviços *back-end*, *front-end* e usuário.

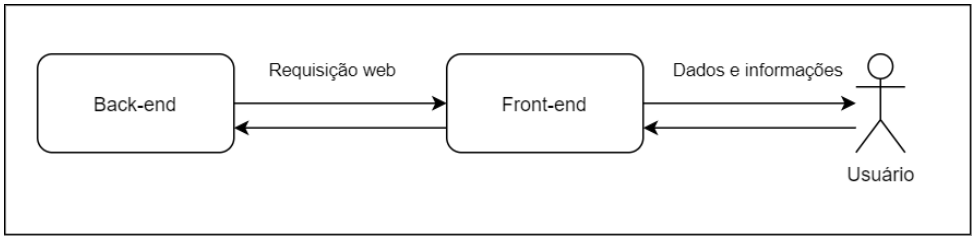


Figura 7 – Comunicação entre *back-end*, *front-end* e usuário

O *back-end* é responsável por toda a lógica de negócios, ou seja, realiza toda a manipulação de dados de acordo com o serviço que é fornecido sem interação direta com o usuário final. Disponibilizado em um servidor, este serviço se comunica com outras aplicações implementando uma interface de comunicação – *Application Programming Interface* (API) – que se utiliza de *endpoints*, os quais são portas especializadas e expostas no servidor capazes de realizar e esperar requisições web. Estas requisições web promovem a troca de mensagens entre aplicações via internet seguindo protocolos de comunicação HTTP. O Quadro 1.2 apresenta os métodos de requisição web mais comuns.

Quadro 1.2: Métodos de requisição HTTP

Método	**Descrição**
GET	Requisição de um recurso específico do servidor
POST	Envia um recurso para o servidor
PUT	Substitui um recurso no servidor
DELETE	Deleta um recurso no servidor

O serviço do *front-end* fica localizado em um servidor separado do *back-end*. Este serviço é responsável por enviar as interface gráfica para o navegador web do usuário para que este possa interagir com os fluxos de trabalho presentes no serviço do *back-end*. Esta interface gráfica fará as requisições web responsáveis tanto por pedir dados para o servidor *back-end* e apresentá-los ao usuário quanto por enviar dados e informações inseridas por este usuário e enviá-los para o *back-end*, o qual irá tratá-los em sua lógica de negócio.

Esta divisão de um serviço em *back-end* e *front-end* promove a redução da carga de processamento de um único servidor responsável tanto pela lógica de negócios quanto pela renderização de interfaces para comunicação com o usuário. Com esta especialização de responsabilidades, há uma maior liberdade de implementação de interfaces de interação com o usuário, onde um mesmo *back-end* pode fornecer os mesmo *endpoints* para comunicação com aplicações *front-end* distintos. Por exemplo, uma aplicação web que funcione em um navegador e um aplicativo de dispositivo móvel podem se alimentar de informações provindas dos mesmos *endpoints* servidos por um *back-end* único.

2 Metodologia

Este capítulo tem por finalidade apresentar os métodos adotados para a construção deste trabalho. A Seção 2.1 identifica as ferramentas utilizadas para o desenvolvimento do trabalho, a Seção 2.2 faz a apresentação da metodologia aplicada e a Seção 2.3 mostra as ferramentas utilizadas para o levantamento bibliográfico.

2.1 Ferramentas de Desenvolvimento e Ambiente

O código-fonte do projeto foi desenvolvido em Javascript e HTML5. As duas tecnologias foram selecionadas para que o BROMS seja executado utilizando-se apenas um navegador, ou seja, sem a necessidade de pacotes adicionais, atendendo um dos objetivos específicos apontados anteriormente neste trabalho. Para que os módulos internos ao trabalho pudessem ser importados e utilizados no navegador, é necessário um servidor mínimo que forneça o conteúdo do trabalho. Utilizou-se, então, o Python v3.8.5 – devido a sua ampla presença nos Sistemas Operacionais atuais – para se gerar este servidor local mínimo possibilitando que módulos Javascript implementados fossem capazes de interagir entre si.

Como ambiente de desenvolvimento foram utilizadas as ferramentas presentes no Quadro 2.1, o qual apresenta o editor de texto utilizado, o gerenciador de versionamento de códigos-fonte em repositórios, a plataforma utilizada para armazenar o repositório do trabalho e o navegador utilizado para se utilizar e testar os resultados das implementações.

Quadro 2.1: Listagem de ferramentas utilizadas no desenvolvimento

Ferramenta	**Versão**	**Descrição**
Visual Studio Code	v1.55.1	Editor de textos
Git	v2.31.0	Gerenciador de versionamento de repositórios
GitHub	—	Provedor de repositórios remotos
Python	v3.8.5	Interpretador Python
FastAPI	v0.65.2	FrameWork Python para construção de APIs
Uvicorn	v0.14.0	Framework de servidor HTTP com requisições assíncronas
HTML	v5.0	Linguagem de Marcação de Hipertexto
Javascript	ES6	Linguagem de programação
Google Chrome	v89.0	Navegador web com suporte a HTML5 e Javascript ES6

O navegador *Google Chrome* v89.0 foi escolhido pelo seu suporte aos recursos pre-

sentes no Javascript e HTML5 como, por exemplo, a *tag Canvas* do HTML5 – responsável por toda a renderização dos gráficos e tabelas do projeto – bem como suporte aos módulos Javascript citados anteriormente.

Pensando em uma evolução futura do BOCA servindo como um *back-end* fornecedor dos dados a respeito de um *contest* qualquer que esteja sendo executado, desenvolveu-se uma *API mock* capaz de simular a execução de uma competição baseada em arquivos estáticos de uma competição já ocorrida. Para o desenvolvimento desta *API* utilizou-se a linguagem Python3 com o *framework fastAPI* v0.65.2 devido ao seu conjunto de ferramentas para rápido desenvolvimento de *API's web*. Consoante a isto, para subir o servidor implementado no *fastAPI* utilizou-se a ferramenta *Uvicorn* v0.14.0, com o propósito de promover um servidor *web* capaz de lidar com requisições assíncronas. Esta *API*, como já dito, se utiliza de arquivos de uma competição já finalizada para simular a execução da mesma. Tendo a simulação ocorrendo, foram gerados *endpoints* responsáveis por fornecer os dados a respeito desta competição de acordo com as rotas indicadas no Quadro 2.2.

Quadro 2.2: Listagem de rotas

Rota	**Parâmetros**	**Descrição**
/contest	—	Retorna nome, duração, congelamento, *blind*, penalidade, quantidade de equipes, quantidade de questões e equipes participantes do *contest*
/contest/finish	—	Retorna verdadeiro ou falso para verificar se o *contest* já foi finalizado
/runs	—	Retorna uma lista com todas submissões de respostas das equipes
/runs/diff	runID	Recebe o ID de uma submissão de resposta e retorna as submissões mais recentes a partir deste ID

2.2 Metodologia de Desenvolvimento

O trabalho seguiu conforme a metodologia ágil com pequenos ciclos iterativos para construção de incremento tanto do presente documento quanto do software em desenvolvimento. Uma reunião inicial com o Professor Orientador Edson Alves e o Professor Bruno Ribas proporcionou a elicitação dos requisitos do software, apresentados no Quadro 2.3. Estes requisitos foram quebrados em partes menores e distribuídas em *sprints* cabíveis no tempo de realização deste trabalho. O foco em entregas rápidas do *Scrum* consoante ao curto tempo do semestre letivo tornou bem viável a escolha desta metodologia de desenvolvimento. Se fez uso de elementos base da metodologia para se ter um bom andamento nas iterações do projeto ao longo do semestre.

Quadro 2.3: Listagem de requisitos do BROMS

ID	Requisito	Descrição
R01	Visualizar aplicação em navegador	A aplicação deve funcionar em um navegador
R02	Rolar informações do placar	A aplicação deve ser capaz de realizar rolagem pelas linhas do placar
R03	*Deploy* simples	A aplicação deve possuir poucos passos para realização de *deploy*
R04	Dependências mínimas	A aplicação deve possuir quantidade baixa de dependências
R05	Indicar classificados	A aplicação deve destacar os classificados ao fim da competição
R06	Indicar medalhistas	A aplicação indicar os medalhistas ao final da competição
R07	Apresentar informações sobre times	A aplicação deve apresentar informações referentes aos times
R08	Tocar músicas dos times	A aplicação deve ser capaz de tocar a música registrada para cada time

Partindo da elicitação dos requisitos, seguiu-se para a determinação do tempo de cada *sprint*. Determinando o tempo médio de uma a duas semanas – a depender da semana – devido ao curto espaço de tempo do semestre letivo, reuniões foram agendadas para as terça-feiras. Estas reuniões foram responsáveis pela revisão da *sprint* passada, verificando as implementações ocorridas na mesma, e planejamento da *sprint* seguinte, levando em consideração possíveis débitos técnicos provindos da *sprint* anterior. O Quadro 2.4 apresenta o planejamento de cada *sprint* executada neste trabalho.

2.3 Levantamento Bibliográfico

Para o levantamento do referencial teórico, foi utilizado o motor de buscas de artigos acadêmicos da *Google*, o *Google Scholar*. A busca foi feita em cima de palavras chaves relacionadas ao assunto tratado em diversos pontos deste projeto.

A primeira busca no *Google Scholar* foi por artigos relacionados às maratonas de programação. Com 3760 resultados apresentados, o artigo do Kikuti (2015) foi o primeiro da listagem. Foi lido e verificado que possuía informações relevantes para a definição do termo de maratona de programação.

Foi utilizado, também, o buscador da Biblioteca Digital da Produção Intelectual Discente da Universidade de Brasília para se encontrar trabalhos de conclusão de curso relacionados à juízes onlines. Com 1745 resultados, verificou-se que existiam trabalhos orientados pelo orientador deste mesmo trabalho, o Professor Edson. Verificou-se que existia o trabalho de conclusão de curso de Maranhão e Carvalho (2017), o qual estava

Quadro 2.4: Listagem de *sprints*

Etapa	**Sprint**	**Tarefa**
TCC1	01	Familiarização Canvas
TCC1	02	Classe Scoreboard
TCC1	03	Classe Row
TCC1	04	Redimensionalisação
TCC1	05	Camera e Scroll
TCC1	06	Singleton
TCC1	07	Documentação TCC1
TCC1	09	Apresentação TCC1
TCC2	01	Simulação em API Mock
TCC2	02	Endpoints API Mock
TCC2	03	Novo layout: Paralelogramos
TCC2	04	Novo layout: Constantes
TCC2	05	Refatoração: Renderização de Paralelogramos
TCC2	06	Refatoração: Renderização de Textos
TCC2	07	Refatoração: Eventos
TCC2	08	Documentação TCC2
TCC2	09	Apresentação TCC2

relacionado ao termo buscado.

As definições dos juízes online indicados neste projeto foram baseadas nas informações contidas em suas respectivas páginas *web*.

3 Resultados

Este capítulo tem por finalidade apresentar os resultados obtidos com o desenvolvimento deste trabalho. A Seção 3.1 apresenta o BROMS e suas funcionalidade em sua primeira iteração, a Seção 3.2 apresenta a segunda iteração do BROMS, a Seção 3.3 mostra o BROMS comparado a outros placares e a Seção 3.4 expõe os problemas encontrados na evolução do trabalho.

3.1 O BROMS - Brazilian Online Marathon Scoreboard

O BROMS é um *scoreboard* dinâmico, capaz de desenhar a listagem dos times durante uma maratona de programação que se utilize do juiz eletrônico BOCA. Na primeira iteração do BROMS, a apresentação das informações seguiu um padrão geral de linhas e colunas convencionais em uma tabela, desenhando, por meio da biblioteca Canvas do Javascript, as informações comuns às maratonas de programação – posição, time, score, penalidade, questões realizadas. A Figura 8 mostra o BROMS em sua versão mais atual renderizando os elementos citados e o Código 1 apresenta a estrutura da classe `Singleton` do Canvas.

Como o BROMS necessita de um servidor local mínimo, devido a limitação dos navegadores de não permitirem acesso direto à arquivos locais do computador utilizado, utilizou-se a biblioteca `http.server` do Python para realizar esta tarefa. O Código 2 apresenta a linha de comando responsável por iniciar este servidor mínimo passando a *flag* `-m` para execução de módulos Python como *script*, o módulo `http.server` nativo ao Python responsável por executar o servidor e a porta `3000` para expor a comunicação com a aplicação. Este servidor Python fornece os arquivos necessários ao navegador para a aplicação funcionar corretamente.

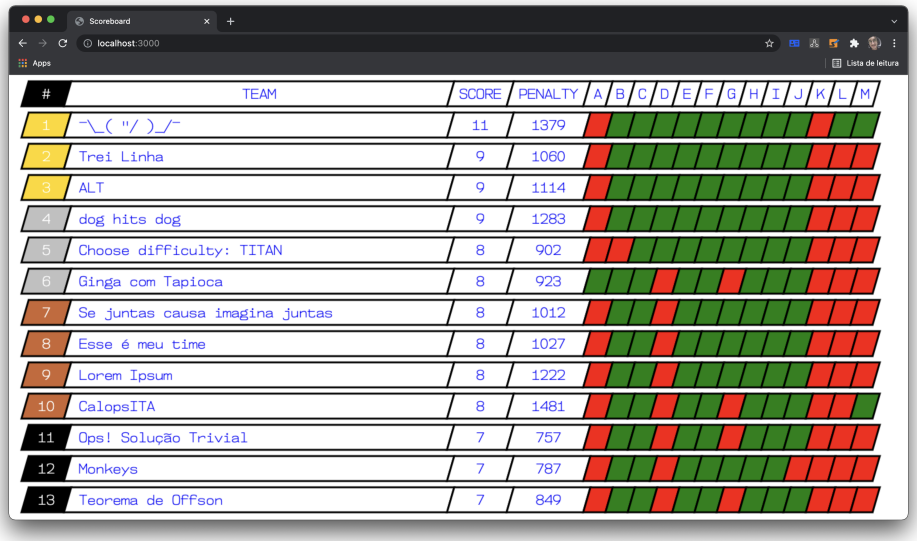


Figura 8 – BROMS renderizando posição, time, score, penalidade e questões realizadas.

```
let canvasSingleton = (function () {
    var instance;
    function createInstance() {
        var object = document.getElementById('canvas')
        return object;
    }
    return {
        getInstance: function () {
            if (!instance) {
                instance = createInstance();
            }
            return instance;
        }
    };
})();

export default canvasSingleton;
```

Código 1: Classe Singleton Canvas.

```
python3 -m http.server 3000
```

Código 2: Comando para executar o servidor Python.

Como o BROMS tem finalidade de ser utilizado em diversas competições e visa se alimentar de informações provindas do BOCA, a construção do placar está diretamente relacionada às informações que venham de requisições *web*. Informações importantes como os nomes dos times e quantidade de questões são fundamentais para a definição e dimensionamento da aplicação gráfica.

Na primeira iteração do BROMS estas requisições *web* provêm de um arquivo estático com informações de uma competição finalizada como mostra a Figura 2 apresentada previamente na Seção 1.3. Estas informações são requeridas pela aplicação por meio de leitura direta dos arquivos e fornecidas ao *Scoreboard* para serem processadas. A variável *url* contida no Código 3 espera o caminho para os arquivos estáticos da competição. O BROMS pode interagir com estes arquivos estáticos pois foram servidos ao navegador por meio do servidor Python já mencionado.

```
export const getContest = (url) => {
  return new Promise((resolve, reject) => {
    fetch(url)
    .then(response => response.text())
    .then(text => {
      text = text.split('\n')
      let [eventTitle, eventInfo, values] = text
      const [duration, frozen, blind, penality] = eventInfo.split(FILE_SEPARATOR)
      eventInfo = {duration, frozen, blind, penality}
      const [qtdTeams, qtdProblems] = values.split(FILE_SEPARATOR)
      text = text.slice(3)
      const teams = text.slice(0,qtdTeams)
      return {eventTitle, eventInfo, values, qtdTeams, qtdProblems, teams}
    })
    .then(resolve)
    .catch(reject)
  })
}

export const getRuns = (url) => {
  return new Promise((resolve, reject) => {
    fetch(url)
    .then(response => response.text())
    .then(text => {
      text = text.split('\n').slice(0,-1);
      const runs = text.map((run,i) => {
        const [runid, time, teamUid, problem, verdict] = run.split(FILE_SEPARATOR);
        return [parseInt(runid), parseInt(time), teamUid, problem, verdict]
      })
      return runs.reverse()
    })
    .then(resolve)
    .catch(reject)
  })
}
```

Código 3: Módulo de leitura de arquivos estáticos do Contest.

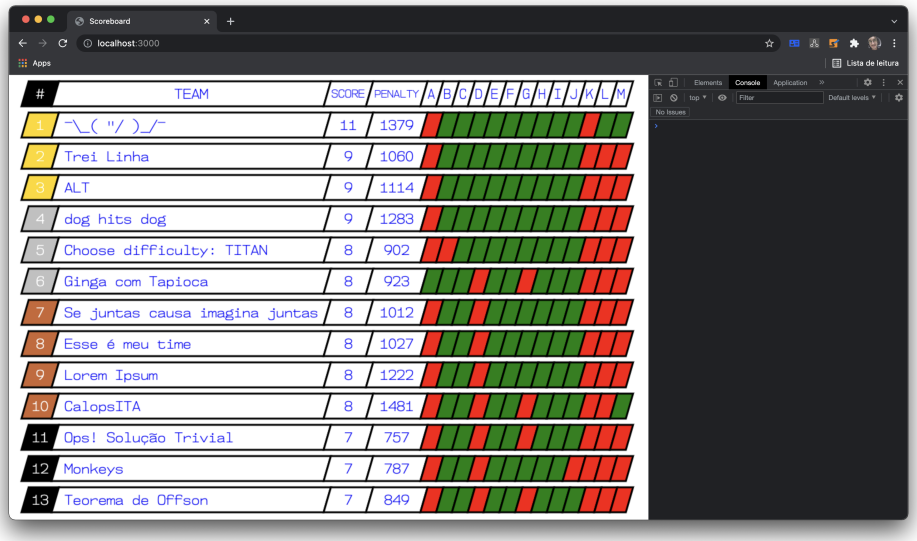


Figura 9 – Renderização do placar em 1314x616 pixeis.

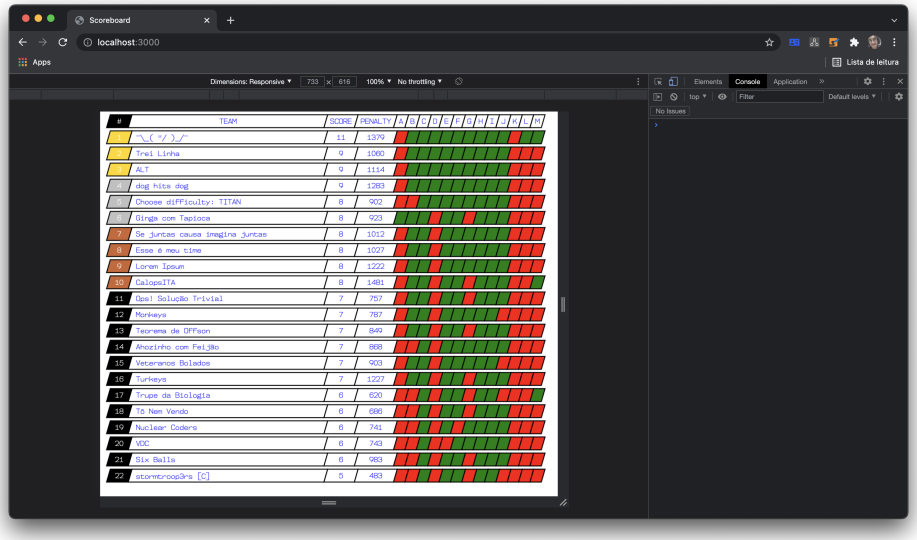


Figura 10 – Renderização do placar em 773x616 pixeis.

Além de renderizar a listagem dos participantes, a aplicação é capaz, também, de se redimensionar automaticamente. A Figura 9 e a Figura 10 mostram o BROMS sendo renderizado em diferentes resoluções sem ocorrer a atualização da página no navegador, ou seja, na mesma execução da aplicação pode-se alterar as dimensões do placar sem problemas. Esta atualização é feita por meio de um evento Javascript que observa a alteração da resolução da janela como mostrado no código 4.

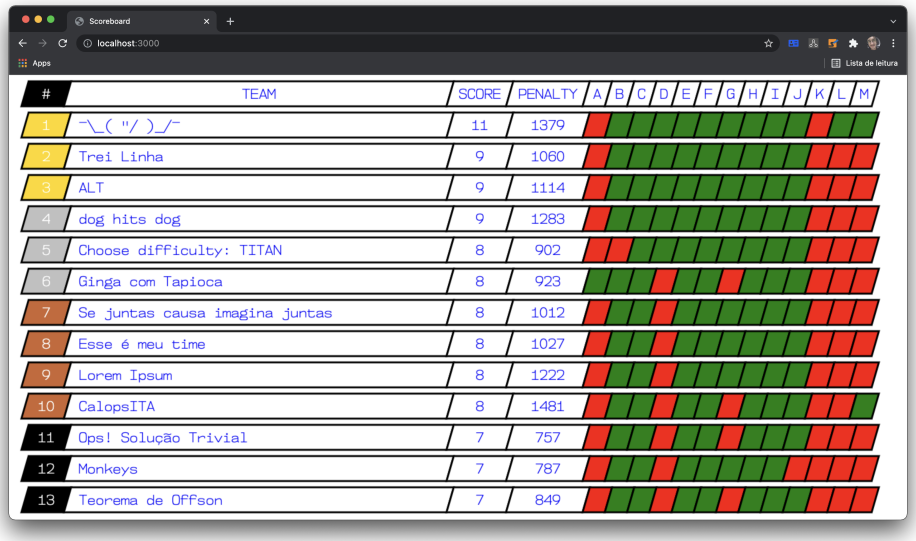


Figura 11 – Renderização dos times de 1 a 13.

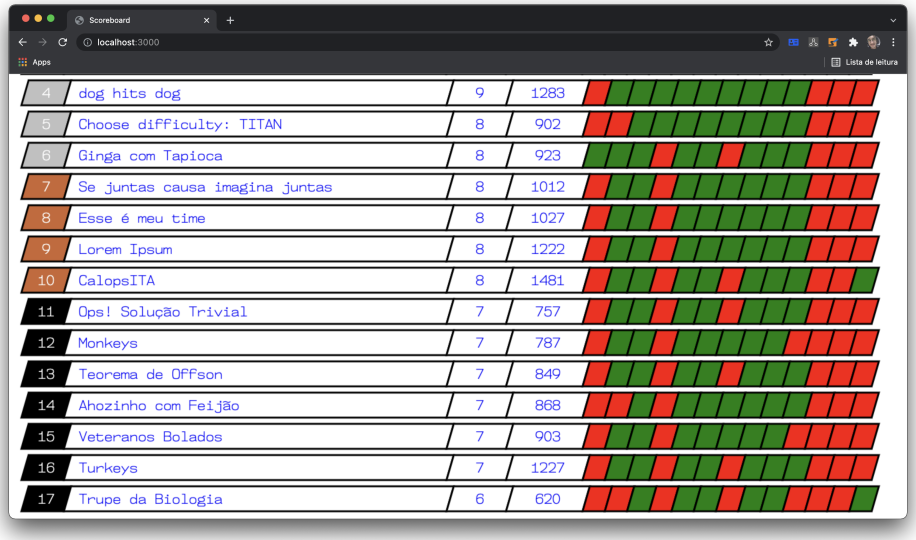


Figura 12 – Renderização dos times de 4 a 17 após rolagem para baixo.

```
window.addEventListener('resize', () => {
  canvas.width = window.innerWidth - 5;
  canvas.height = window.innerHeight - 5;
  const camera = cameraSingleton.getInstance()
    .updateSize(window.innerWidth, window.innerHeight)
  redrawAll()
}, {passive: true})
```

Código 4: Evento de redimensionamento da tela.

Como o placar realiza uma listagem dos times, nem todos os times aparecem de uma mesma vez. O BROMS é capaz de apresentar todos os times por meio de rolagem,

ou seja, a aplicação recebe uma entrada para rolar tanto para cima quanto para baixo no placar, havendo a possibilidade de se visualizar os times que estão ocultos devido ao tamanho da tela renderizada. A Figura 11 e Figura 12 mostram esta rolagem e apresentação de times que antes não eram visíveis no BROMS. O código 5 mostra a implementação do evento responsável pela rolagem, o qual atualiza os valores de posicionamento da classe Câmera, a qual é encarregada de definir o que será renderizado no Canvas.

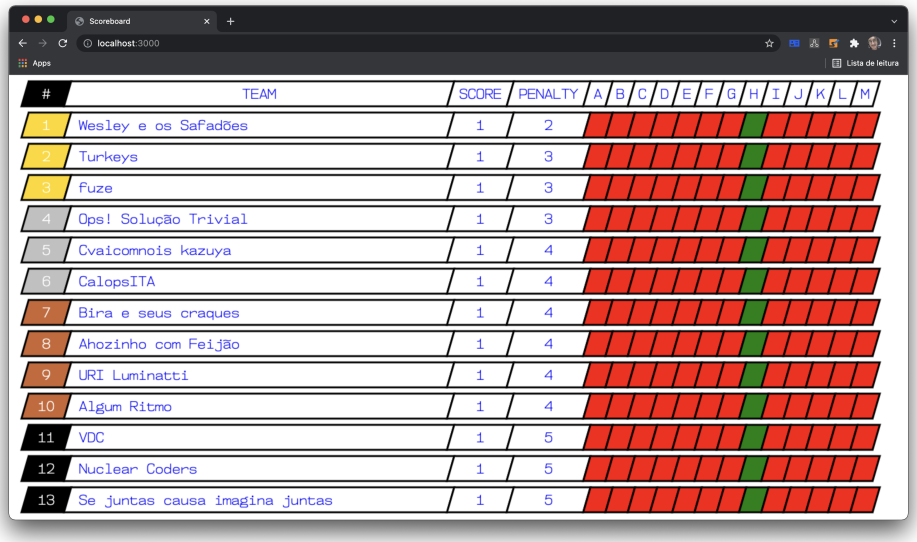


Figura 13 – Primeiro estado de submissões

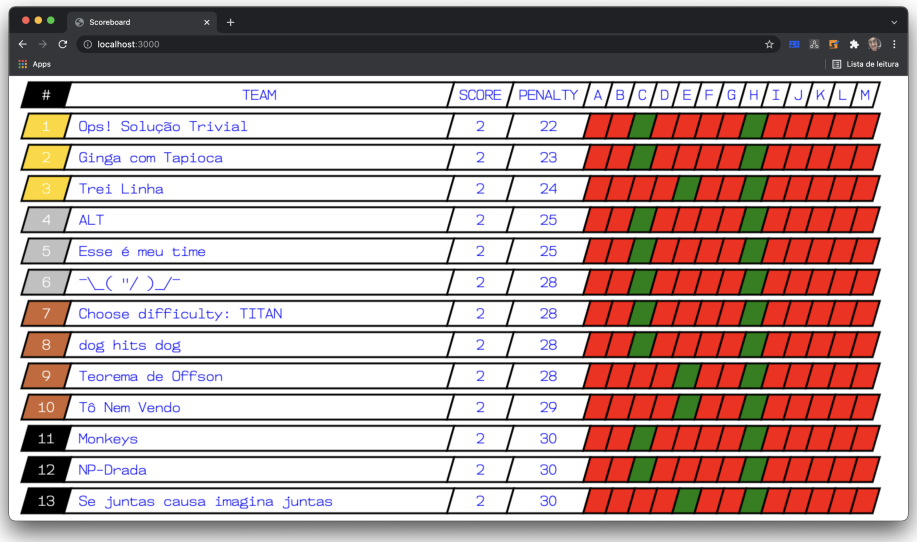


Figura 14 – Segundo estado de submissões

O BROMS é capaz tanto de computar quanto atualizar as informações dos times. Tendo uma submissão, sendo ela um acerto ou um erro, esta informação é repassada para a parte visual do placar, a qual é atualizada 24 vezes a cada segundo. A aplicação realiza

```
window.addEventListener('wheel', (event) => {
    if (event.deltaY < 0) {
        camera.move(0, camera.y - 20)
        redrawAll()
    }
    else if (event.deltaY > 0) {
        camera.move(0, camera.y + 20)
        redrawAll()
    }
    }, {passive: true})

    window.addEventListener('keydown', (event) => {
    if (event.code === "ArrowUp") {
        camera.move(0, camera.y - 40)
        redrawAll()
    }else if (event.code === "ArrowDown") {
        camera.move(0, camera.y + 40)
        redrawAll()
    }
}, {passive: true})
```

Código 5: Evento de rolagem da tela.

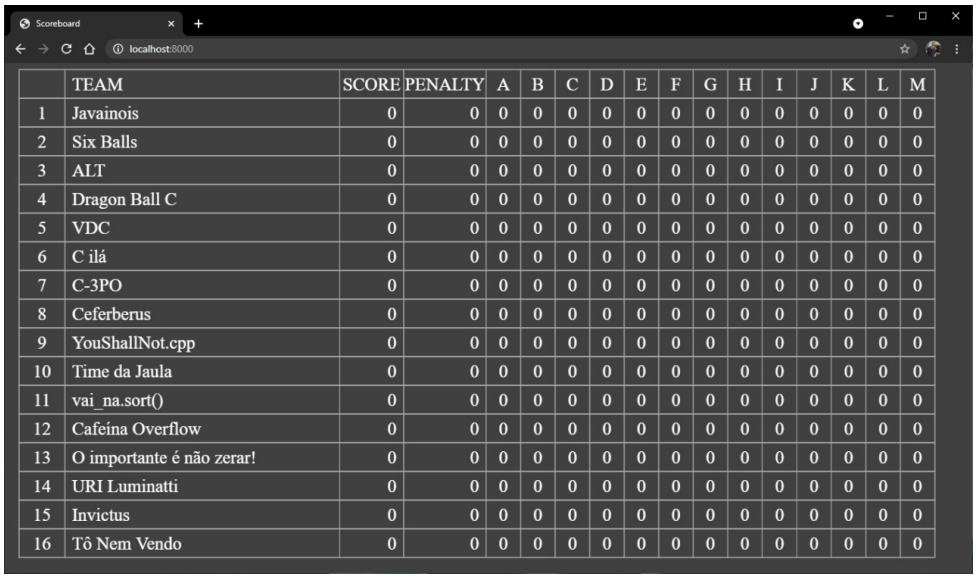
o processamento da submissão e faz a atualização necessária na tela. A Figura 13 e a Figura 14 mostram a atualização das informações das equipes após serem computadas algumas submissões pelo BROMS.

3.2 Evolução

O BROMS sofreu uma evolução em sua segunda fase de implementação promovendo melhorias na manutenibilidade do código-fonte do trabalho e um novo visual na apresentação das linhas contendo as informações dos participantes. Esta evolução gerou uma API *web* para simular a competição realizada pelo BOCA, permitindo que o BROMS realizasse requisições *web* reais para se alimentar das informações em vez de depender da leitura de arquivos estáticos locais. A evolução também focou na refatoração do código-fonte do próprio BROMS em especial na arquitetura de como eram chamados os métodos responsáveis pela renderização das imagens no *Canvas* desacoplando os métodos de desenho encontrados na classe *Row* e delegando para classes especializadas como a classe *Parallelog* e a classe *Text*, responsáveis por desenhar, respectivamente, os paralelogramos e as informações presentes no BROMS.

3.2.1 Scoreboard

Um novo layout foi implementado baseado no desenho de paralelogramos para substituir a apresentação de uma tabela comum – como era no layout anterior presente na Figura 15. A Figura 16 apresenta o novo desenho dos paralelogramos bem como o destaque dos possíveis medalhistas, ou seja, há um destaque dos primeiros colocados. Isso facilita na visualização de qual time receberá sua medalha de bronze, prata e ouro. Os demais participantes são identificados com sua posição em fundo preto.



	TEAM	SCORE	PENALTY	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Javainois	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Six Balls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	ALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Dragon Ball C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	VDC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	C ilá	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	C-3PO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Ceferberus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	YouShallNot.cpp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Time da Jaula	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	vai_na.sort()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Cafeína Overflow	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	O importante é não zerar!	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	URI Luminatti	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Invictus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	Tô Nem Vendo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 15 – Demonstração de layout anterior.

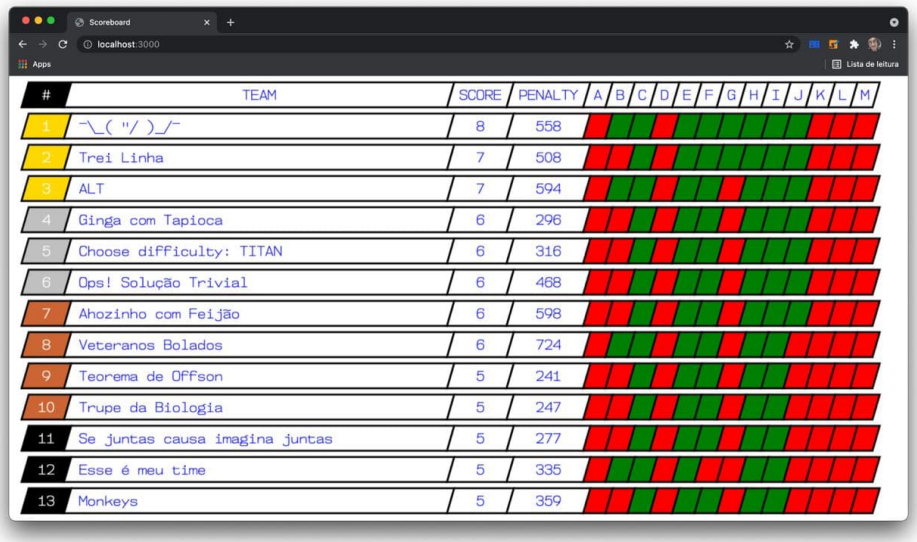


Figura 16 – Demonstração de novo layout.

O desenho das linhas em que os times se encontram está especializada na classe *Parallelog* como apresentado no código 6. Esta nova classe assume a responsabilidade de desenho dos elementos presentes nas linhas do BROMS, liberando esta para se responsabilizar apenas no controle de informações de um dos times participantes. A classe *Parallelog* recebe as coordenadas da classe que o instancia e conversa diretamente com a classe *Canvas* para realizar a renderização dos paralelogramos.

Assim como a classe *Parallelog*, a classe *Text* também conversa diretamente com o *Canvas* para realizar renderização de imagem. Entretanto, esta é encarregada dos textos recebidos pela classe que a instancia e, por meio de seu método *align* - apresentado pelo Código 7 - consegue definir o tamanho e posicionamento ideais para o texto dentro das coordenadas recebidas pela sua classe pai.

A refatoração também adicionou a classe *EventsManager*, a qual fica responsável por conter os eventos presentes no BROMS. O redimensionamento e rolagem da tela agora são definidos por esta classe para centralizar os eventos presentes na aplicação. O Código 8 apresenta a implementação desta classe.

Após o incremento do trabalho, o BROMS recebe as informações não mais por meio de leitura de arquivos presentes localmente, mas por requisições *web* que conversam com a API *mock* implementada para simular a competição ocorrida no BOCA. Estas requisições se utilizam da interface *fetch* presente no Javascript, a qual fornece os recursos necessários para interação do BROMS com a API sem a necessidade de instalação de pacotes não nativos à linguagem. Estas requisições são responsáveis por adquirir as informações necessárias para a montagem e atualização do placar. As requisições pedem para o simulador informações como a listagem dos times e suas submissões para que o BROMS funcione. Ao iniciar a aplicação no navegador do usuário, o BROMS realizará uma requisição web com todas as submissões já disponibilizadas pela API. O Código 9 apresenta a implementação da requisição que pede as novas submissões da competição.

```
class Parallelogram {
    constructor(Parent=undefined, x, y, wRel, h,
                text=undefined, fillColor='white', borderColor='black'){
        this.Parent = Parent;
        this.x = x;
        this.y = y;
        this.wRel = wRel;
        this.h = h;
        this.fillColor = fillColor;
        this.borderColor = borderColor;
        this.text = text;
        if(this.text) this.text.Parent = this;
        this.w = this.wRel * this.Parent.w;
    }
    setText(text, {selfAlign=this.text.selfAlign}={}){
        this.text.setText(text, selfAlign);
    }
    draw(){
        const ctx = canvasSingleton.getInstance().getContext();
        const ang = CONSTANTS.ang
        ctx.beginPath()
        const dx = -this.h * Math.tan(ang) / 2;
        ctx.lineWidth = 3
        ctx.strokeStyle = this.borderColor
        ctx.fillStyle = this.fillColor
        ctx.moveTo(this.x, this.y)
        ctx.lineTo(this.x + this.w, this.y)
        ctx.lineTo(this.x + this.w - dx, this.y - this.h)
        ctx.lineTo(this.x - dx, this.y - this.h)
        ctx.lineTo(this.x, this.y)
        ctx.fill()
        ctx.closePath()
        ctx.stroke()
        if(this.text){
            this.text.draw()
        }
    }
    update(x, y, {fillColor=this.fillColor}={}) {
        this.w = this.wRel * this.Parent.w;
        this.x = x;
        this.y = y;
        this.fillColor = fillColor;
    }
}
```

Código 6: Classe Paralelogramo.

Em suma, o BROMS foi implementado esperando retornos específicos em formato *JSON* provindos do simulador por meio de suas requisições web. Sendo o retorno da

```
align(value, type, width='', height='', font=FONTS.default) {
        const ctx = canvasSingleton.getInstance().getContext();
        ctx.font = `${font.size}px ${font.name}`;
        const textMetrics = ctx.measureText(value)
        const text_w = textMetrics.width
        const text_h = textMetrics.actualBoundingBoxAscent
        const offset = Math.max(width * 0.02, 1);
        const diff = height * Math.tan(CONSTANTS.ang) / 2 // é o cateto oposto
        const ans = {
          center: [diff + (width - diff - text_w)/2, (height - text_h)/2, offset],
          left: [offset + diff,(height - text_h)/2, offset],
          right: [
                width - text_w - offset < 0 ?
                        width - text_w :
                        width - text_w - offset, (height - text_h)/2, offset],
        }
        if (text_w > width - diff)
                return this.align(value, type,
                        width, height, {...font, size: font.size - 1})
        return ans[type]
}
```

Código 7: Implementação do método align contido na classe Text

```
class EventsManager {
    constructor(){
        this.events = {}
    }
    registerListener(eventType, listener){
        if(this.events[eventType])
            this.events[eventType] = [...this.events[eventType], listener];
        else
            this.events[eventType] = [listener];
    }
    unregisterListener(eventType, listener){
        if(this.events[eventType])
            this.events[eventType].pop(listener)
    }
    onEvent(eventType, event){
        this.events[eventType].map(listener => listener.onEvent(eventType, event))
    }
    notify(eventType, event) {
        this.onEvent(eventType, event)
    }
}
```

Código 8: Classe EventsManager.

```
export const getNewRuns = (runId) => {
  const url = `${baseUrl}/runs/diff`;
  const method = 'POST';
  return new Promise((resolve, reject) => {
        fetch(url, {
          method,
          headers: {
                'Access-Control-Allow-Origin':'*'
          },
          body: JSON.stringify(runId)
        })
        .then(response => response.json())
        .then(resolve)
        .catch(reject)
  })
}
```

Código 9: Implementação de requisição de novas submissões

requisição das informações da competição um objeto *JSON* composto por chaves com dados da competição e uma listagem com os objetos contendo informações das equipes – como apresentado no Código 10 –; e o retorno da requisição das submissões das equipes sendo uma lista de objetos com informações de cada submissão – como apresentado no Código 11.

```
{
  "name": "LATAM ACM ICPC",
  "duration": 300,
  "frozen": 285,
  "blind": 240,
  "penalty": 20,
  "n_teams": 72,
  "n_questions": 13,
  "teams": [
    {
      "teamId": "teambrbr7",
      "college": "CEULJI-ULBRA",
      "name": "Javainois"
    },
    {
      "teamId": "teambrbr11",
      "college": "UNESP-Bauru",
      "name": "UNESP Bauru"
    }
  ]
}
```

Código 10: Formato esperado pelo retorno da rota /contest

```
[
  {
    "runId": 197701401,
    "time": 2,
    "teamUid": "teambrbr4",
    "problem": "H",
    "verdict": "Y"
  },
  {
    "runId": 198956310,
    "time": 4,
    "teamUid": "teambrbr26",
    "problem": "H",
    "verdict": "Y"
  }
]
```

Código 11: Formato esperado pelo retorno da rota /runs e /runs/diff

3.2.2 API

A refatoração do BROMS voltada para a implementação de requisições *web* foi focada em gerar uma aplicação que simule a competição. Esta aplicação foi gerada em um repositório separado do placar. Esta API *mock* implementa os *endpoints* necessários para o placar ser montado e atualizado. A simulação da competição se baseia na leitura do arquivo estático de uma competição finalizada. Esta API foi implementada na linguagem Python utilizando o framework *fastAPI* pela sua facilidade e velocidade na construção de APIs *web*.

Como indicado no Capítulo 2, a API necessita de um servidor para executar a aplicação. O comando necessário para executar esta aplicação está indicado no Código 12.

```
uvicorn main:app
```

Código 12: Comando para executar a API.

As informações presentes neste arquivo estático são passadas à memória em variáveis de controle, como apresentado no Código 13, sendo um objeto que armazena os dados referentes aos times e à competição. As submissões são armazenadas em duas listas distintas: uma contendo todas as submissões e outra que contém apenas as submissões até o tempo atual da simulação da competição.

A simulação da competição ocorre por meio de uma variável de tempo. O Código 14 apresenta esta lógica incluindo o incremento da variável de tempo e a atualização da lista de submissões até o momento em que a competição se encontra.

```
contest = {}
total_runs = []
runs = []
t = 0
```

Código 13: Variáveis de controle da API

```
@app.on_event("startup")
@repeat_every(seconds=1, wait_first=True,
max_repetitions=global_vars.contest["duration"]/10)
def periodic():
    global_vars.t += 10
    print(global_vars.t, len(global_vars.runs))
    for index, run in enumerate(global_vars.total_runs):
        if(run['time'] > global_vars.t):
            global_vars.runs =  global_vars.total_runs[:index]
            return

    global_vars.runs = global_vars.total_runs
```

Código 14: Controle de tempo da simulação

A implementação das rotas está dividida em dois módulos: um para informações da competição e outro para as submissões. O Código 15 mostra a implementação das rotas de informações da competição e verificação de que a competição já foi encerrada. O Código 16 mostra às implementação das rotas referentes às submissões, onde a primeira rota retorna todas as submissões até o ponto atual da competição e a segunda rota retorna apenas as novas submissões a partir da última submissão presente no BROMS.

```
@router.get("/", tags=["contest"])
async def get_contest():
    return global_vars.contest

@router.get("/finish", tags=["contest"])
async def get_contest():
    if global_vars.t >= global_vars.contest["duration"]:
        return True
    return False
```

Código 15: Rotas de informações da competição.

```python
@router.get("/", tags=["runs"])
async def get_runs():
    return global_vars.runs

@router.post("/diff", tags=["runs"])
async def get_diff_runs(request: Request):
    uid = await request.json()
    i = 0
    for index,run in enumerate(global_vars.runs):
        if run["runId"] == uid:
            i = index
            break
    if i > 0:
        return global_vars.runs[i+1:]
    return []
```

Código 16: Rotas de submissões.

3.2.3 Progresso

Para este trabalho, o BROMS foi planejado para possuir os seguintes pontos presentes no Quadro 3.1. Alguns dos pontos inicialmente indicados não puderam ser implementados até o fim deste trabalho devido a necessidade de refatoração do código para definição de uma arquitetura que isolasse as responsabilidades de renderização de imagens.

Quadro 3.1: Progresso do BROMS

Feito	Planejado
✔	Visualizar aplicação em navegador
✔	Rolar informações do placar
✔	Deploy simples
✔	Dependências mínimas
✖	Indicar classificados
✔	Indicar medalhistas
✖	Apresentar informações sobre os time
✖	Tocar músicas dos times

3.3 Comparativo Scoreboards

Atualmente, o Maratona Live (SEGUNDO, 2021), o Maratona Animeitor (OSHIRO, 2021), e o Maratona Animatior Rust (WUERGES, 2021) conseguem suprir a necessidade de um placar para apresentação das informações das maratonas de programação, entretanto, possuem a necessidade de dependências de pacotes externos tanto aos sistemas operacionais quanto as linguagens de programação utilizadas em seus desenvolvimentos.

Desta forma, o BROMS surgiu para ser um placar com dependências mínimas, boa portabilidade e de fácil execução, removendo a etapa de instalação de pacotes, bem como a etapa de compilação.

Segundo as instruções presentes no repositório do Maratona Live, há a necessidade de instalação de bibliotecas no sistema operacional bem como a configuração de múltiplos arquivos para realizar a inicialização da aplicação. Além disso, a implementação da aplicação na linguagem C++ torna o processo de compilação não trivial em sistemas que não possuam esta linguagem instalada nativamente. A Figura 17 apresenta o Maratona Live em execução.

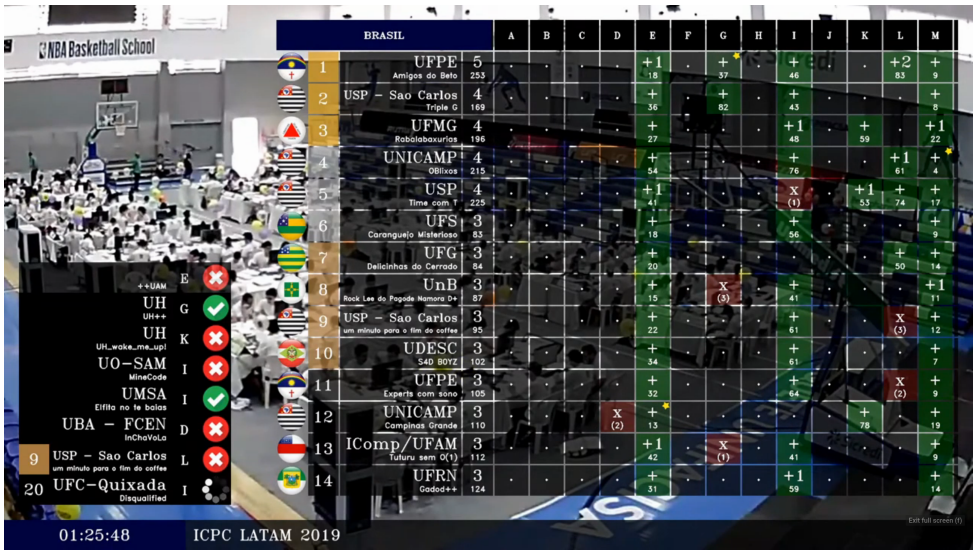


Figura 17 – Maratona Live em execução

O Maratona Animeitor, implementado em linguagem Python, não necessita de instalação de pacotes no sistema operacional, mas de pacotes Python por meio do gerenciador de pacotes Python, o PiP. A dependência de pacotes externos pode auxiliar na implementação da aplicação, mas, também, aumenta a quantidade de etapas para execução bem como aumenta o risco de ausência de compatibilidade caso os pacotes externos parem de ser manutenidos.

O Maratona Animeitor Rust necessita da instalação de pacotes adicionais no sistema operacional bem como a instalação da própria linguagem Rust, pois esta não é encontrada na maioria dos sistemas operacionais. Em seguida, há, ainda, a necessidade de compilação do código para que, por fim, a aplicação seja executada.

No caso do BROMS não há necessidade de instalação de pacotes no sistema operacional. Para sua execução, é necessário apenas um servidor mínimo Python, função presente nativamente na linguagem, com a finalidade de servir os arquivos para o navegador do usuário.

3.4 Problemas Encontrados

O primeiro problema encontrado foi a necessidade de um servidor local para que o código fonte funcionasse corretamente no navegador. Sem um servidor local, os módulos Javascript não poderiam ser importados para a aplicação em tempo de execução no navegador.

Um segundo problema fora o acesso de referência à instância global do módulo Canvas, o qual não se tinha certeza de que todos os módulos estavam realmente alterando a mesma instância ou se apenas uma cópia. Para isso, gerou-se uma classe Singleton, a qual garante a existência de uma instância única de uma classe na execução da aplicação.

O retorno das requisições de informações a respeito da competição provindas do BOCA se encontram em formato ZIP, o que exige manipulação de arquivos antes do acesso real às informações da maratona. Para contornar isto, foram utilizados arquivos de uma competição finalizada e gerada uma API web para realizar o fornecimento das informações por meio de requisições web.

Havia o planejamento de implementação de animações já neste trabalho. Entretanto, devido aos métodos de renderização de imagens estarem muito acoplados à classe Row, esta implementação não foi possível. Houve uma refatoração focada neste desacoplamento de responsabilidades para que, na próxima evolução do trabalho, haja a implementação das animações bem como um aumento na manutenibilidade do código-fonte.

4 Considerações Finais

O BROMS já tem uma versão preliminar funcional, com a listagem dos times e computação dos valores de pontuação e penalidades, bem como a posição em que o time se encontra no *contest*. A simulação da competição por meio da API também fornece uma boa base para a conversação *web* entre o BROMS e as futuras rotas do BOCA.

Até o fim desta evolução do projeto, não foi possível implementar todos os pontos e ideias definidas para o BROMS devido a necessidade de realizar refatoração. Esta refatoração foi necessária para garantir a manutenibilidade do código-fonte da aplicação, pois a partir do início da implementação das animações no placar, percebeu-se que os métodos de renderização de imagem no *Canvas* estavam muito acopladas à classe *Row*. O desacoplamento facilita na implementação tanto das animações de movimento das linhas quanto de futuras animações que possam vir a ser idealizadas na aplicação.

O BROMS foi pensado para ajudar na apresentação das competições de programação competitiva com uma aplicação que funcione em navegadores na maioria dos computadores atuais. Esta ferramenta auxiliará os apresentadores da maratona com uma maior qualidade de informações para os espectadores da competição.

A aplicação tem um tempo de instalação e preparo extremamente menor do que os outros placares atualmente utilizados para as transmissões ao vivo. Isso se deve ao foco da implementação em reduzir a quantidade de pacotes utilizados na produção do BROMS.

O grande foco na minimização de dependências trouxe alguns desafios para a implementação do BROMS. Um deles foi a familiarização com a API Canvas, nativa ao Javascript. Este pacote fornece boas ferramentas e recursos para realização de desenhos. Entretanto, sem uma experiência prévia com desenhos por meio de códigos, renderizar formas geométricas simples fora um desafio a ser superado. Outro ponto envolvendo o Canvas fora identificar o local correto para chamar a renderização dos desenhos, pois, a principio, implementar os desenhos na classe Row pareceu uma boa ideia, o que se mostrou inviável ao se tentar implementar as animações. Já a renderização dos textos fora um pouco mais simples no começo, porém realizar o reajuste de tamanho dos textos para que coubesse dinamicamente em seus espaços corretos demandou bastante tempo para ser realizado.

4.1 Trabalhos futuros

Para as próximas iterações de evolução do BROMS, a princípio, serão iniciadas pela implementação de animações das linhas e, em seguida, a apresentação de destaques

após o término da competição. Outro ponto de interesse será a interação dos usuários com o placar, possibilitando a visualização de informações extras de cada time como foto dos competidores, sua bandeira e dados sobre participações anteriores. Estes pontos servirão de guia para novos trabalhos de conclusão de curso voltados à evolução do BROMS.

Referências

ATCODER. *AtCoder*. 2021. <https://atcoder.jp/>. Acessado em: 2021-05-05. Citado na página 30.

BEECROWD. *BeeCrowd*. 2021. <https://beecrowd.com.br/>. Acessado em: 2021-05-01. Citado na página 31.

CAMPOS, C. P. de; FERREIRA, C. E. Boca: um sistema de apoio a competições de programação. 2004. Citado na página 27.

CODECHEF. *CodeChef*. 2021. <https://www.codechef.com/>. Acessado em: 2021-05-05. Citado na página 31.

CODECUP. *CodeCup*. 2021. <https://www.codecup.nl>. Acessado em: 2021-05-05. Citado na página 31.

CODEFORCES. *CodeForces*. 2021. <https://codeforces.com/>. Acessado em: 2021-05-01. Citado na página 30.

GOOGLE. *Google Code Jam*. 2021. <https://codingcompetitions.withgoogle.com/codejam>. Acessado em: 2021-05-05. Citado na página 25.

ICPC. *ICPC International Collegiate Programming Contest*. 2021. <https://icpc.global/>. Acessado em: 2021-05-01. Citado na página 25.

KIKUTI, A. P. e Mauro Miazaki e Tony Hild e Mauro Mulati e D. A metodologia das maratonas de programação em um projeto de extensão: um relato de experiência. *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, v. 4, n. 1, p. 1246, 2015. ISSN 2316-8889. Disponível em: <https://br-ie.org/pub/index.php/wcbie/article/view/6276>. Citado 2 vezes nas páginas 25 e 35.

MARANHÃO, C. N.; CARVALHO, S. C. L. d. Broj: juiz eletrônico e online para ensino e aprendizagem. 2017. Citado 2 vezes nas páginas 29 e 35.

OBI. *OBI*. 2021. <https://olimpiada.ic.unicamp.br/>. Acessado em: 2021-05-05. Citado na página 26.

OSHIRO, M. *Maratona Animeitor*. 2021. <https://github.com/maratona-linux/maratona-animeitor>. Acessado em: 2021-05-05. Citado 3 vezes nas páginas 23, 28 e 51.

RIBAS, B. *CD-MOJ*. 2021. <https://moj.naquadah.com.br/cgi-bin/index.sh>. Acessado em: 2021-11-02. Citado na página 31.

SBC. *SBC Maratona SBC de Programação*. 2021. <http://maratona.sbc.org.br/>. Acessado em: 2021-05-01. Citado na página 26.

SEGUNDO, M. P. *Maratona Live*. 2021. <https://github.com/maups/maratona-live>. Acessado em: 2021-05-05. Citado 3 vezes nas páginas 23, 28 e 51.

SPOJ. *Spoj*. 2021. <https://www.spoj.com/>. Acessado em: 2021-05-05. Citado na página 31.

UVA. *UVA*. 2021. <https://onlinejudge.org/>. Acessado em: 2021-05-05. Citado na página 31.

WUERGES, E. *Maratona Animeitor*. 2021. <https://github.com/wuerges/maratona-animeitor-rust>. Acessado em: 2021-05-05. Citado 3 vezes nas páginas 23, 28 e 51.

ZHIGANG, S. et al. Moodle plugins for highly efficient programming courses. 2001. Citado na página 26.