

UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Aplicação *web* para organizar o estudo  
para programação competitiva**

Gustavo de Medeiros Carlos

MONOGRAFIA FINAL

MAC 499 — TRABALHO DE  
FORMATURA SUPERVISIONADO

Supervisor: Carlos Eduardo Ferreira

São Paulo  
2022

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0  
(Creative Commons Attribution 4.0 International License)*

*Loucura tem limite e a gente não chegou nem perto.*

*— Kobus, Giovanna*



# Agradecimentos

Ao MaratonUSP, o grupo de extensão para estudos em programação competitiva da USP, por me introduzir a essa modalidade, que pratico desde o meu ano de ingresso.

Ao Enrique e ao Nathan por fazerem parte do meu time de 2021. Foi com eles o meu treinamento mais intenso, que gerou várias das ideias para este trabalho.

Ao Carlinhos pela orientação durante o ano todo.

Ao João Francisco Daniel pelas dicas de ferramentas para o desenvolvimento da aplicação.

E principalmente, à minha mãe por investir com tanto esforço em meus estudos, o que permitiu a minha mudança de estado para cursar a graduação.



# Resumo

Gustavo de Medeiros Carlos. **Aplicação web para organizar o estudo para programação competitiva**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

A programação competitiva reúne milhares de participantes do mundo inteiro em múltiplas competições. São várias as plataformas na *internet* que disponibilizam problemas e recursos para a prática para essas competições. O *Codeforces* é uma das mais utilizadas e possui uma *API* para o acesso aos seus dados. O treino para programação competitiva exige centenas de horas de dedicação ao estudo de algoritmos e estruturas de dados e à resolução de problemas. Existem algumas estratégias de estudo recomendadas para torná-lo mais eficiente. Para auxiliar os competidores nesse processo, foi desenvolvida uma aplicação *web* que se propõe a incentivar o emprego dessas estratégias em conjunto com uma integração com o *Codeforces* pelo uso de sua *API*. A aplicação pode ser utilizada a partir do navegador e conta com variados componentes que ajudam o treino em diferentes aspectos. Durante seu desenvolvimento, foi considerada uma infraestrutura que facilita a adição futura de outras funcionalidades, além das já implementadas.

**Palavras-chave:** Programação competitiva. *Software* para estudo. Aplicação *web*.





# Abstract

Gustavo de Medeiros Carlos. **Web application to organize the study for programming competitions**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Competitive programming gathers thousands of students around the world in a varied set of competitions. Multiple online tools have their own collection of problems and contests that allow practicing for them. Codeforces is one of the most used websites for this purpose, and it provides an API that makes its data easily accessible. Training for those competitions requires hundreds of hours dedicated to learning algorithms and data structures and solving problems. There are recommended study strategies to ensure this is done efficiently. A web application was developed to help competitors in this matter. It encourages the use of some of those strategies with an integration with Codeforces using its API. The website can be used in the browser, and it has components to help practice in different aspects. Its development considered a range of possible features, so it allows adding new ones in addition to the ones already implemented.

**Keywords:** Competitive programming. Study software. Web application.



# Lista de figuras

3.1	Exemplo de problema do <i>Codeforces</i> . . . . .	7
3.2	Vereditos do juiz virtual do <i>Codeforces</i> . . . . .	8
3.3	Representação dos recursos da <i>API</i> em um diagrama <i>UML</i> . . . . .	11
5.1	Modelo conceitual inicial da aplicação. . . . .	19
5.2	Modelo conceitual para os dados do usuário. . . . .	20
5.3	Modelo genérico de informações derivadas dos recursos da <i>API</i> . . . . .	22
5.4	Informações derivadas da <i>API</i> do <i>Codeforces</i> . . . . .	22
5.5	Relação dos dados do servidor com as páginas do cliente. . . . .	26
5.6	Arquitetura do servidor. . . . .	30
5.7	Sequência usual de carregamento de dados no cliente. . . . .	31
5.8	Arquitetura da aplicação. . . . .	32
6.1	Barra lateral de navegação expandida. . . . .	33
6.2	Página inicial ( <i>Home</i> ). . . . .	34
6.3	Página de problemas do <i>Codeforces</i> ( <i>CF Problems</i> ). . . . .	35
6.4	Página de competições internas do <i>Codeforces</i> ( <i>CF Contests</i> ). . . . .	36
6.5	Página de competições da <i>Gym</i> do <i>Codeforces</i> ( <i>CF Gym</i> ). . . . .	37
6.6	Problemas do usuário ( <i>My problems</i> ). . . . .	38
6.7	Formulário de edição de problema. . . . .	39
6.8	Lista de problemas do usuário ( <i>My problem lists</i> ). . . . .	40
6.9	Página para uma lista de problemas ( <i>Problem list</i> ). . . . .	41
6.10	Diálogo com a lista de problemas não contidos na lista em questão. . . . .	41
6.11	Diálogo para edição de informações de uma lista de problemas. . . . .	42

## Lista de tabelas

5.1	Relação de operações <i>CRUD</i> com as requisições <i>HTTP</i> . . . . .	21
5.2	Relação de operações para a associação entre problemas e listas de problemas.	21

## Lista de programas

5.1	Classe <code>UrlCache</code> . . . . .	28
5.2	Exemplo de métodos da classe <code>CfApiResponseFetcher</code> . . . . .	28
5.3	Exemplo de processador de informação. . . . .	29

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Programação competitiva</b>	<b>3</b>
2.1	Definição . . . . .	3
2.2	Principais competições . . . . .	4
2.3	Sites para estudo . . . . .	4
<b>3</b>	<b>Codeforces</b>	<b>7</b>
3.1	Principais funcionalidades . . . . .	8
3.1.1	Juiz virtual . . . . .	8
3.1.2	Sistema de pontuação de desempenho ( <i>rating</i> ) . . . . .	8
3.1.3	Competições internas . . . . .	9
3.1.4	<i>Problemset</i> . . . . .	9
3.1.5	Competições da <i>Gym</i> . . . . .	10
3.2	<i>API</i> . . . . .	10
3.2.1	Recursos disponíveis . . . . .	10
3.2.2	Limitações . . . . .	11
<b>4</b>	<b>Estratégias de estudo</b>	<b>13</b>
4.1	Visões de competidores . . . . .	13
4.2	Principais estratégias de estudo . . . . .	15
4.2.1	Objetivo da aplicação . . . . .	16
<b>5</b>	<b>Desenvolvimento da aplicação</b>	<b>17</b>
5.1	Ideias iniciais . . . . .	17
5.1.1	Funcionalidades . . . . .	17
5.1.2	Modelo conceitual . . . . .	18
5.2	Análise de requisitos . . . . .	19
5.3	Modelo de dados final . . . . .	20

5.3.1	Dados do usuário . . . . .	20
5.3.2	Informações da <i>API</i> . . . . .	21
5.3.3	Uso dos dados do servidor pelas páginas da aplicação . . . . .	25
5.4	Detalhes de implementação . . . . .	26
5.4.1	Servidor ( <i>back end</i> ) . . . . .	27
5.4.2	Cliente ( <i>front end</i> ) . . . . .	30
5.4.3	Interação entre os componentes . . . . .	32
<b>6</b>	<b>Aplicação <i>web</i> desenvolvida</b>	<b>33</b>
6.1	Páginas da aplicação . . . . .	33
6.1.1	Página inicial ( <i>Home</i> ) . . . . .	34
6.1.2	Páginas relacionadas ao <i>Codeforces</i> . . . . .	34
6.1.3	Páginas do usuário . . . . .	37
6.2	Fluxos de uso . . . . .	42
<b>7</b>	<b>Trabalhos futuros</b>	<b>45</b>
7.1	Auto-avaliação em tópicos . . . . .	45
7.2	Recomendação de problemas . . . . .	45
7.3	Funcionalidades de <i>time</i> . . . . .	46
<b>8</b>	<b>Conclusão</b>	<b>47</b>
	<b>Referências</b>	<b>49</b>

# Capítulo 1

## Introdução

A programação competitiva é uma das modalidades de competição relacionada à computação que conta com milhares de participantes no mundo todo. A maior disputa entre estudantes universitários é a *International Collegiate Programming Contest (ICPC)*. Seu formato consiste em uma prova de cinco horas que requer a resolução do maior número de problemas de programação no menor tempo possível.

O estudo de algoritmos e estruturas de dados e a prática de resolução de vários problemas é essencial para quem quer ter um bom desempenho nesse estilo de competição. Na *internet*, existem inúmeras ferramentas para treino e uma das principais delas é o *Codeforces*.

Para otimizar o uso do tempo necessário para o estudo para programação competitiva, é importante selecionar os problemas adequados para serem resolvidos, além de provas para serem simuladas com o intuito de replicar as oficiais. Adicionalmente, é essencial garantir que o estudo está realmente trazendo os resultados esperados.

O objetivo deste trabalho é proporcionar uma ferramenta para auxiliar os competidores nesse processo. Para isso, foi desenvolvida uma aplicação *web* com funcionalidades que buscam auxiliá-los na realização de um treino eficiente e incentivar o uso de estratégias recomendadas de estudo.

A aplicação é integrada com o *Codeforces* com o uso de sua *API (Application Programming Interface)*. Isso permite utilizar informações públicas do *site* para complementar as suas funcionalidades. Nela, o usuário consegue, por exemplo, selecionar problemas para resolver e competições para simular.

Os primeiros capítulos têm o objetivo de oferecer contexto para o que motivou o desenvolvimento da aplicação. O Capítulo 2 explica sobre programação competitiva, com a citação de algumas competições e plataformas *online* para o treinamento. Já o Capítulo 3 enfatiza o *Codeforces*, com detalhes sobre suas funcionalidades, que inclui a *API* concedida pela plataforma para o acesso aos seus dados. Como forma de orientar a escolha das funcionalidades para a aplicação, o Capítulo 4 traz algumas perspectivas sobre estratégias para um treino proveitoso para as competições.

Os próximos capítulos tratam da aplicação desenvolvida. Primeiro, o Capítulo 5 detalha

o processo de desenvolvimento. Ele começa com as primeiras ideias de funcionalidades para a aplicação, depois descreve o modelo para os dados dos usuários e para as informações provindas da *API* do *Codeforces* e termina com alguns detalhes sobre a implementação. Em seguida, o Capítulo 6 apresenta a aplicação desenvolvida, com as especificidades das páginas que ela disponibiliza e termina com alguns exemplos para o seu uso durante os treinamentos. Já o Capítulo 7 lista algumas funcionalidades que podem ser futuramente implementadas ao considerar a infraestrutura já desenvolvida.

Para concluir, o Capítulo 8 traz os principais resultados da aplicação desenvolvida.



# Capítulo 2

## Programação competitiva

Existem múltiplas categorias de competições relacionadas à computação. Em *hackatons*, o objetivo é desenvolver um protótipo de *software* em um período limitado de tempo, que pode ser de algumas horas ou de vários dias seguidos. Há também competições voltadas à programação paralela e ao desenvolvimento de uma inteligência artificial. A modalidade central no desenvolvimento deste trabalho é denominada por programação competitiva.

### 2.1 Definição

Programação competitiva é um esporte mental que envolve a resolução de um conjunto de problemas em um tempo limitado com o uso de algoritmos e estruturas de dados conhecidos da computação.

As questões dessa categoria de competição possuem um enunciado que descreve uma situação-problema que deve ser resolvida com a construção de um algoritmo. No enunciado, o formato dos dados de entrada para o algoritmo é detalhado e ele deve produzir os dados de saída correspondentes, conforme o especificado. Em geral, o enunciado estabelece limites para os parâmetros de entrada e a solução deve ser eficiente o suficiente para ser executada com tais parâmetros sem ultrapassar os limites de tempo e memória predefinidos.

As soluções são enviadas a um juiz virtual que executa o programa com vários casos de testes não visíveis aos competidores. Esse juiz irá verificar se a resposta está correta e se é gerada sem ultrapassar os limites determinados para o problema. Se isso acontecer, o código é considerado correto e o problema resolvido. Caso contrário, o competidor será informado e poderá realizar outras tentativas de resolução.

Em uma competição, várias pessoas tentam solucionar esses problemas simultaneamente e vence quem tiver resolvido o maior número de problemas no final. Em caso de empate, outros critérios são utilizados para definir os premiados. Habitualmente, são critérios que consideram a quantidade de envios ao juiz e o tempo para a resolução de cada problema.

## 2.2 Principais competições

A *IEEEExtreme* ([ieeextreme.org](http://ieeextreme.org)) é uma competição organizada pelo *Institute of Electrical and Electronics Engineers (IEEE)*. Ela tem um formato de 24 horas e admite a participação de estudantes membros do instituto.

Outra competição é a *International Olympiad in Informatics (IOI)* ([ioinformatics.org](http://ioinformatics.org)), dedicada aos alunos do mundo inteiro que estão no ensino secundário. No Brasil, a Olimpíada Brasileira de Informática (OBI) ([olimpiada.ic.unicamp.br](http://olimpiada.ic.unicamp.br)), organizada pela Sociedade Brasileira de Computação (SBC), tem o mesmo público-alvo. Os mais bem classificados nela podem participar de uma prova seletiva para a *IOI*, conforme o regulamento (OBI, 2022).

A *Google Code Jam* ([codingcompetitions.withgoogle.com/codejam](http://codingcompetitions.withgoogle.com/codejam)) e a *Facebook Hacker Cup* ([facebook.com/codingcompetitions/hacker-cup](http://facebook.com/codingcompetitions/hacker-cup)) são exemplos de competições *online* organizadas por empresas multinacionais de tecnologia, com somente as últimas fases realizadas presencialmente.

A principal competição de programação do mundo é a *International Collegiate Programming Contest (ICPC)* ([icpc.global](http://icpc.global)). Ela reuniu 59 mil estudantes em mais de 3.400 universidades de 104 países em competições regionais na edição de 2019 (DR. MICHAEL JEFFRY DONAHOO, 2020). Os estudantes universitários participam em equipes de três pessoas e devem resolver o máximo de problemas em um período de cinco horas com somente um computador à disposição para a escrita dos códigos das soluções.

Para *ICPC*, o mundo é dividido em múltiplas regiões, com o Brasil sozinho definindo uma delas. Cada região possui uma determinada quantidade de vagas dedicadas à final mundial da disputa. Essas vagas são preenchidas por competições regionais organizadas por inúmeras outras instituições em parceria com a *ICPC*.

A Maratona SBC de Programação ([maratona.sbc.org.br](http://maratona.sbc.org.br)), também organizada pela SBC, é responsável pela seleção de vagas brasileiras para a final mundial da *ICPC* e faz isso em duas fases. Na primeira, denominada sub-regional, as universidades do país são agrupadas em regiões e a competição ocorre em diversos polos regionais concorrentemente. Alguns critérios são considerados para a classificação dos times para a segunda fase, a final brasileira. A prova final é realizada em um único local com 60 equipes que totalizam 180 estudantes e define os classificados para o mundial.

## 2.3 Sites para estudo

São várias ferramentas *online* dedicadas ao estudo para programação competitiva. Entre elas, estão *Codechef* ([codechef.com](http://codechef.com)), *Sphere Online Judge (SPOJ)* ([spoj.com](http://spoj.com)) e o *TopCoder* ([topcoder.com](http://topcoder.com)). Essas plataformas possibilitam a resolução de problemas e oferecem seus próprios juízes virtuais para o julgamento das soluções. Elas também organizam competições internas nas quais qualquer pessoa pode participar.

Outras plataformas, como *LeetCode* ([leetcode.com](http://leetcode.com)) e *HackerRank* ([hackerrank.com](http://hackerrank.com)), ajudam no estudo para entrevistas técnicas de emprego. Muitas empresas utilizam proble-

mas semelhantes aos de competições de programação e solicitam para os candidatos os resolverem detalhadamente durante as entrevistas.

No Brasil, uma ferramenta de estudos muito utilizada é o *Neps Academy* ([neps.academy](#)). Ela contém tanto materiais para estudo quanto problemas que podem ser resolvidos pela própria plataforma, além de também organizar competições internas.

Entre as plataformas mais conhecidas para o estudo de programação competitiva e, em especial, para a *ICPC*, está o *Codeforces* ([codeforces.com](#)). O *site* é utilizado por pessoas do mundo inteiro e conta com várias das características presentes nas outras plataformas mencionadas nesta seção. Como essa plataforma foi utilizada substancialmente neste trabalho, ela será posta em evidência no próximo capítulo.



# Capítulo 3

## Codeforces

O *Codeforces* é uma plataforma utilizada por centenas de milhares de pessoas para o estudo relacionado a programação competitiva. Criado por Mike Mirzayanov e lançado em janeiro de 2010, o *site* tem o objetivo de oferecer competições e outros recursos para treino, o que inclui um *blog* interno para interações entre seus usuários (MIKE MIRZAYANOV, 2015a).

O *site* organiza suas próprias competições e disponibiliza as realizadas em outros lugares para a simulação na plataforma. Ela permite a resolução de problemas com envios de códigos ao seu juiz virtual e conta com um sistema de pontuação de desempenho (*rating*). Além disso, conta com uma Interface de Programação de Aplicações (*API*, do inglês *Application Programming Interface*) para o acesso facilitado aos seus dados.

**D. Multiplication Table**

time limit per test: 1 second  
memory limit per test: 256 megabytes  
input: standard input  
output: standard output

Bizon the Champion isn't just charming, he also is very smart.

While some of us were learning the multiplication table, Bizon the Champion had fun in his own manner. Bizon the Champion painted an  $n \times m$  multiplication table, where the element on the intersection of the  $i$ -th row and  $j$ -th column equals  $i \cdot j$  (the rows and columns of the table are numbered starting from 1). Then he was asked: what number in the table is the  $k$ -th largest number? Bizon the Champion always answered correctly and immediately. Can you repeat his success?

Consider the given multiplication table. If you write out all  $n \cdot m$  numbers from the table in the non-decreasing order, then the  $k$ -th number you write out is called the  $k$ -th largest number.

**Input**  
The single line contains integers  $n$ ,  $m$  and  $k$  ( $1 \leq n, m \leq 5 \cdot 10^5$ ;  $1 \leq k \leq n \cdot m$ ).

**Output**  
Print the  $k$ -th largest number in a  $n \times m$  multiplication table.

**Examples**

<b>input</b>	Copy
2 2 2	
<b>output</b>	Copy
2	

**Figura 3.1:** Exemplo de problema do Codeforces.

A Figura 3.1 mostra um problema típico do *Codeforces*, com o título, os limites de tempo e de memória, o enunciado que descreve a situação a ser resolvida, as especificações

de entrada, o que inclui restrições para ela, as de saída e também exemplos de casos de teste.

## 3.1 Principais funcionalidades

### 3.1.1 Juiz virtual

O juiz virtual é responsável por avaliar os códigos enviados pelos usuários. Ele compila o código utilizando o compilador especificado no momento do envio e então o executa para vários casos de teste do problema correspondente. Esses casos procuram testar a correção do algoritmo por executá-lo com variadas combinações dos parâmetros de entrada. Além disso, ele mede a quantidade de tempo e de memória utilizados. Se o algoritmo gerar as saídas corretas nos limites especificados pelo problema, a solução é aceita. Caso contrário, informa, através de um veredito, o motivo de não a aceitar.

Os vereditos mais comuns na plataforma são: tempo limite ultrapassado, solução incorreta, solução aceita e memória limite estourada, como é observável na Figura 3.2. O número do caso de teste que gerou a falha é evidenciado no *Codeforces*, o que não está disponível em muitas outras plataformas.

Problem	Lang	Verdict	Time	Memory
<a href="#">K - Knowledge Testing Problem</a>	GNU C++20 (64)	Time limit exceeded on test 10	3000 ms	3600 KB
<a href="#">K - Knowledge Testing Problem</a>	GNU C++20 (64)	Wrong answer on test 2	78 ms	2400 KB
<a href="#">A - AppendAppendAppend</a>	GNU C++17	Accepted	61 ms	165800 KB
<a href="#">H - Hanoi</a>	GNU C++17	Memory limit exceeded on test 1	764 ms	262100 KB
<a href="#">H - Hanoi</a>	GNU C++17	Wrong answer on test 7	15 ms	300 KB
<a href="#">H - Hanoi</a>	GNU C++17	Wrong answer on test 1	0 ms	0 KB

Figura 3.2: Vereditos do juiz virtual do Codeforces.

### 3.1.2 Sistema de pontuação de desempenho (*rating*)

O *Codeforces* atribui uma pontuação de desempenho (ou *rating*, como é denominado na plataforma) aos usuários que participaram de pelo menos uma competição pontuada (*rated*). Essa pontuação segue um sistema semelhante ao da *Rating Elo*, utilizado no xadrez. A ideia desse sistema é comparar o desempenho dos competidores. É esperado que um competidor com uma pontuação maior tenha mais chances de terminar uma competição com uma classificação melhor que o outro que possui uma pontuação menor (MIKE MIRZAYANOV, 2010).

Em cada competição pontuada do *Codeforces*, é estimada a classificação esperada de cada competidor em relação a todos os outros e, no fim dela, é comparada a classificação esperada com a concretamente atingida. A partir disso, a pontuação de cada competidor é alterada de modo a refletir seu desempenho real.

Essa pontuação atribuída aos usuários é utilizada para agrupá-los no que é denominado por **divisão** pela plataforma. A divisão 1 é composta por aqueles com as maiores pontuações. Atualmente, existem quatro divisões.

Adicionalmente, a relação das pontuações dos usuários com os problemas que eles resolveram em uma competição é usada para atribuir uma pontuação de dificuldade (também denominada por *rating* na plataforma) para cada problema.

### 3.1.3 Competições internas

O *Codeforces* realiza competições internas ([codeforces.com/contests](https://codeforces.com/contests)) cuja duração é aproximadamente duas horas e são organizadas pelos próprios usuários do *site*. A maior de 2022 até o momento teve mais de 36 mil competidores registrados<sup>1</sup>.

Qualquer usuário da plataforma pode participar das competições, mas, em geral, cada competição tem uma ou mais divisões-alvo para a participação, o que indica o nível de dificuldade de suas questões. Somente as pessoas das divisões-alvo terão sua pontuação de desempenho alterada após a participação síncrona da competição.

Em 2021, foram realizadas 141 competições no total, o que implica uma média entre duas e três por semana. Dentre elas, competições da categoria *Codeforces Round* totalizaram 89. Essa categoria é a padrão da plataforma e vem sendo realizada desde 2010. Outras 19 foram da categoria *Educational Codeforces Round*, cujos problemas envolvem conceitos mais simples com o intuito de ensiná-los aos competidores (MIKE MIRZAYANOV, 2015b).

Os usuários podem participar das competições mesmo após elas terem terminado, em uma modalidade denominada participação virtual. A competição original é efetivamente simulada, já que é possível ver quais problemas estão sendo resolvidos por outros usuários como se estivessem os fazendo no momento da simulação. Isso significa que a classificação parcial é visível para quem está em uma participação virtual. A maior diferença da participação síncrona é o fato de a participação virtual não influenciar a pontuação de desempenho do usuário.

### 3.1.4 *Problemset*

Os problemas das competições internas são agregados em um conjunto enorme de problemas denominado *Problemset* ([codeforces.com/problemset](https://codeforces.com/problemset)). Alguns deles podem estar em mais de uma competição, como é o caso de quando há duas competições para divisões diferentes acontecendo simultaneamente. O *Problemset* contém somente uma dessas instâncias, o que garante a unicidade dos problemas.

---

<sup>1</sup> *Codeforces Round #817 (Div. 4)* (<https://codeforces.com/contest/1722>)

### 3.1.5 Competições da Gym

A área *Gym* do *Codeforces* ([codeforces.com/gyms](https://codeforces.com/gyms)) permite a adição de competições realizadas externamente à plataforma. Dessa forma, seus usuários podem simular essas competições dentro dela, o que inclui o julgamento de suas soluções pelo juiz virtual e a classificação parcial dos participantes durante a simulação da prova.

Várias das provas oficiais da *ICPC* estão incluídas na *Gym*, o que torna o *Codeforces* um excelente meio para praticar para essa competição. Em muitos casos, os participantes oficiais dessas competições também são adicionados à simulação. Assim, é possível comparar o desempenho próprio com os daqueles que participaram da competição real.

## 3.2 API

O *Codeforces* é uma das únicas plataformas de treino para programação competitiva que disponibiliza uma *API* de acesso público. Disponibilizada desde 2014 (IVAN FEFER, 2014), ela permite requisitar diversas das informações contidas no *site* e as devolve no formato *JSON* (*JavaScript Object Notation*).

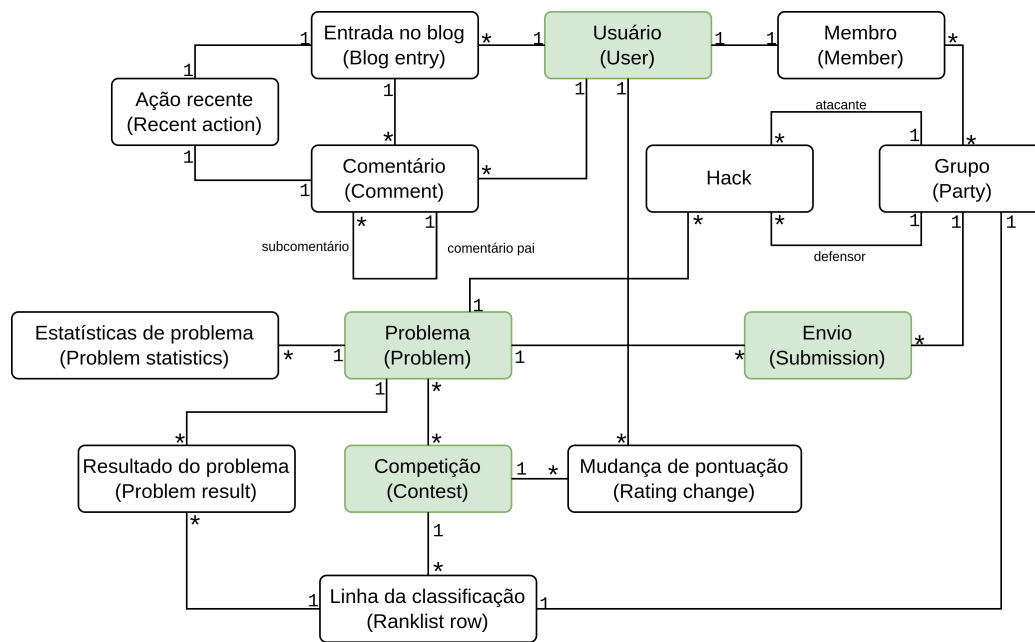
### 3.2.1 Recursos disponíveis

Os recursos disponíveis pela *API* estão listados e descritos em [codeforces.com/apiHelp/objects](https://codeforces.com/apiHelp/objects). Para facilitar a análise de quais deles utilizar para este trabalho, foi criado um diagrama na Linguagem de Modelagem Unificada (*UML*, do inglês *Unified Modeling Language*) com as entidades disponibilizadas por ela (Figura 3.3). É possível observar que, entre os recursos disponíveis pela *API*, estão as postagens no *blog* interno (*Blog entry*), os comentários delas (*Comment*), os times formados internamente pelos usuários (*Party*) e as classificações finais das competições (*Ranklist row*). Para este trabalho, foram utilizadas as informações sobre os usuários (*User*), os problemas (*Problem*), as competições (*Contest*) e os envios de código (*Submission*).

Esses recursos são obtíveis por métodos disponibilizados pela *API*, listados em [codeforces.com/apiHelp/methods](https://codeforces.com/apiHelp/methods). Os seguintes métodos foram utilizados neste trabalho:

- **contest.list?gym={}** - retorna uma lista de competições; se o parâmetro `gym` for `false`, retorna a lista de competições internas, se for `true`, retorna as competições da *Gym*;
- **contest standings?contestId={}** - entrega informações da competição cujo número identificador é o valor do parâmetro `contestId`; também entrega a lista de problemas dela e classificação dos usuários participantes;
- **problemset.problems** - fornece todos os problemas do *Problemset*;
- **user.info?handles={}** - retorna informações dos usuários indicados no parâmetro `handles`;
- **user.status?handle={}** - oferece a lista de envios do usuário cujo nome é o valor do parâmetro `handle`.





**Figura 3.3:** Representação dos recursos da API em um diagrama UML, com destaque para os que foram efetivamente utilizados neste trabalho.

Com a utilização da própria API, é possível verificar que, em 19/12/2022, o Codeforces possuía um total de 1.684 competições internas<sup>2</sup>, que contribuem para a quantidade de 8.313 problemas no *Problemset*<sup>3</sup>. Já a *Gym* totaliza 1.490 competições<sup>4</sup>.

Em relação aos usuários da plataforma, são 501.757 que possuíam a pontuação de desempenho definida, ou seja, que participaram em pelo menos uma competição pontuada<sup>5</sup>. Desses, 114.797 estiveram *online* na plataforma no último mês, com 49.593 participando de pelo menos uma competição pontuada no último mês.

### 3.2.2 Limitações

A API estabelece um limite de uma requisição a cada dois segundos. Isso faz com que seja necessário a usar com moderação. Para a obtenção de informações úteis ao treino, foi considerado que algumas informações precisam ser atualizadas com maior frequência, enquanto outras podem ser requisitadas somente uma vez e armazenadas por um longo período. Isso será detalhado no próximo capítulo.

Além desse limite, existem algumas informações que não são diretamente recuperadas com o uso dela. Um exemplo é a lista de problemas das competições. Para obtê-la, deve-se fazer uma requisição para cada uma das milhares de competições disponibilizadas pela plataforma. Com somente uma requisição ao método `problemset.problems`, é possível recuperar todos os problemas de competições internas da plataforma. Porém, como

<sup>2</sup> `api/contest.list`

<sup>3</sup> `api/problemset.list`

<sup>4</sup> `api/contest.list?gym=true`

<sup>5</sup> `api/user.ratedList?activeOnly=false&includeRetired=true`

mencionado anteriormente, essa lista não inclui os problemas duplicados, o que significa que com ela não é possível definir as competições que possuem as duplicações de um problema.

# Capítulo 4

## Estratégias de estudo

A programação competitiva agrega comunidades de competidores em grupos de universidades e em plataformas *online* para treino. Muitos deles oferecem orientações sobre as melhores técnicas para o treinamento para competições oficiais, algumas delas em forma de artigos na *internet* ou em postagens no *blog* do *Codeforces*.

### 4.1 Visões de competidores

Um artigo da *GeeksForGeeks*, *site* com variados conteúdos para o aprendizado de assuntos relacionados à computação, oferece algumas sugestões para quem está iniciando o treino em programação competitiva ([GEEKSFORGEEKS, 2022](#)). Primeiramente, ele salienta a importância de se aprender uma linguagem a ser utilizada na construção dos códigos de solução para os problemas. Em seguida, sugere o aprendizado dos principais algoritmos e estruturas de dados que costumam aparecer nas questões de programação.

Para a parte prática, o texto propõe a dedicação na resolução de problemas, a começar pelos mais simples. No caso do gasto de muito esforço em um problema sem se chegar na solução, recomenda a leitura de editoriais e de soluções de outras pessoas. Adicionalmente, instrui a participação em competições *online*, o que inclui algumas das ferramentas mencionadas na Seção 2.3. O artigo é finalizado com destaque para a importância da consistência no treino e de não se desmotivar após erros serem cometidos.

Outro artigo no *website Medium*, de Andrei Margeloiu, corrobora com alguns desses pontos ([ANDREI MARGELOIU, 2016](#)). Escrita por um medalhista de ouro na Olimpíada de Informática da Romênia, a publicação aconselha o uso das linguagens *C++* e *Java* para competições de programação. A primeira pela rapidez de execução do seu código e a segunda pela presença da classe `BigInteger`, que permite operações com inteiros muito grandes.

Similarmente ao dito no *GeeksForGeeks*, o autor indica a prática de resolver problemas para o aperfeiçoamento das habilidades de implementação e o estudo de algoritmos e estruturas de dados para o aprendizado de novos conceitos. Em relação ao tempo dedicado a cada problema, ele sugere a insistência na resolução por pelo menos uma hora e, somente após esse tempo, considerar a leitura de soluções oficiais ou de outros competidores.

Ademais, aponta que um bom sinal para a escolha de exercícios é o aprendizado de algo novo a cada três resolvidos.

Alex Danilyuk, conhecido por *um\_nik*, está entre os 10 usuários com maiores pontuações de desempenho do *Codeforces*. Ele criou uma postagem no *blog* interno do *site* com algumas sugestões próprias para a prática em programação competitiva (ALEX DANILYUK, 2022). O competidor resume o estudo em duas componentes que afirma serem quase que independentes: a capacidade de resolver problemas e a velocidade de resolução.

Para o treino da capacidade de resolução de problemas, recomenda a seleção de questões com um nível de dificuldade próximo da habilidade do estudante e a dedicação de muito tempo nas tentativas de resolução. Em contraponto aos artigos citados anteriormente, ele discorda da leitura do editorial como uma boa prática de estudo. Ao invés dela, sugere pausar a tentativa e retornar ao problema em torno de um mês depois.

Em relação à velocidade, prescreve a participação em competições. Pelo fato de uma competição possuir tanto problemas muito fáceis e muito difíceis para quem participa, sobra pouco espaço para problemas do nível adequado ao competidor, o que as tornam desfavoráveis para o aprendizado da capacidade de resolução. No entanto, elas são importantes para praticar a rapidez de solução para problemas que já se deve saber resolver.

Ainda no *blog* do *Codeforces*, outro usuário, o Masataka Yoneda (nome de usuário E869120), publicou um documento com sugestões para o treino que considera técnicas diferentes para níveis de habilidades variados (MASATAKA YONEDA, 2019).

Em seu texto, ele cita o sistema de pontuação de dificuldade de problemas do *Codeforces* como uma das melhores formas de selecionar problemas para estudo. Complementarmente, também recomenda o uso de outras plataformas, como o *AtCoder* e *TopCoder*.

Ele compartilha a opinião dos artigos citados no início desta seção de realizar a leitura dos editoriais após um certo tempo sem conseguir decifrar a resposta para um exercício. Adicionalmente, indica que o tempo que se deve tentar um problema antes disso é relativo à sua dificuldade. Mesmo em questões resolvidas sem ajuda externa, ele aconselha a leitura de soluções de outros competidores mais experientes, para o potencial aprendizado de técnicas novas.

Além disso, o autor recomenda a participação em competições virtuais para o aprimoramento do desempenho nas oficiais. Ele introduz uma ideia nova de simulação de competição: selecionar de quatro a cinco problemas com um nível de dificuldade compatível às habilidades do competidor e resolvê-los em um período de duas a três horas. Essa proposta parece combinar as ideias de treinar a capacidade de resolução e de rapidez, mencionadas por Danilyuk.

No documento, Yoneda detalha que existem alguns *sites* úteis para estudo que não providenciam uma maneira de marcar quais problemas já foram resolvidos. Para resolver isso, ele orienta a criação de planilhas com anotações sobre os problemas, com reflexões sobre estratégias de estudos e aprendizados após a leitura do editorial.

O competidor Colin Galen possui um canal no *Youtube*<sup>1</sup> em que faz vídeos com orien-

---

<sup>1</sup> <https://www.youtube.com/c/ColinGalen>

tações para o treino e ensinamentos sobre algoritmos e estruturas de dados. Seu nome de usuário no *Codeforces* é `galen_colin`.

Em um desses vídeos, ele recomenda uma leitura gradual do editorial ([COLIN GALEN, 2021](#)). Após um determinado tempo na tentativa de resolução do problema, deve-se iniciar a leitura do editorial até o competidor conseguir continuar sozinho. Ademais, também propõe a criação de um histórico de problemas, com anotações sobre a dificuldade, o estado de resolução, os tópicos envolvidos, o que aconteceu durante a tentativa de resolução e o que foi aprendido.

## 4.2 Principais estratégias de estudo

Ao ponderar sobre as visões descritas na seção anterior, consegue-se agrupar as principais técnicas de estudo recomendadas.

### Estudo de algoritmos e estruturas de dados

Muitos problemas de programação utilizam algoritmos e estruturas de dados específicos. Quanto mais desses o competidor tiver conhecimento, mais ferramentas terá para a resolução de exercícios.

### Seleção de problemas adequados

Os autores recomendam a seleção de problemas com dificuldade próximo do nível de habilidade do competidor. Isso evita o gasto de tempo em problemas muito fáceis, que introduziriam poucos conceitos novos, e em problemas muito difíceis, que, apesar de envolverem técnicas desconhecidas pelo competidor, ainda são muito avançadas para ele aprender naquele momento.

O sistema de pontuação de desempenho de usuários e de dificuldade de problemas do *Codeforces* é um ótimo jeito de fazer essa seleção de problemas.

### Análise das dificuldades enfrentadas

Autores como Yoneda e Galen recomendam o registro das dificuldades enfrentadas em cada problema, como maneira de monitorar o que deve ser focado para o aperfeiçoamento das habilidades, de modo a amenizá-las para problemas futuros.

### Participação em competições

O treino em velocidade é muito importante, como defendido por Danilyuk. A participação tanto em competições oficiais quanto em competições simuladas pode ser uma forma de se praticar esse aspecto.

### Consulta a editoriais e outras soluções

Uma sugestão muito frequente é a leitura de soluções oficiais, feitas pelos criadores das questões, ou mesmo de soluções explicadas por outros competidores. Algumas plataformas, como o *Codeforces*, permitem a visualização do código de quem resolveu o problema.

Essa prática é benéfica para que não se gaste muito tempo sem estar aprendendo algo na realização do treino, que acontece quando se fica preso em um problema. No entanto, é importante que o usuário seja sincero em sua tentativa de resolução e evite consultar outras fontes antes que tenha se esforçado o suficiente nela.

### **4.2.1 Objetivo da aplicação**

O desenvolvimento da aplicação é guiado pelo objetivo de facilitar o uso dessas estratégias de estudo para quem está treinando para competições de programação. O próximo capítulo trata de descrever as funcionalidades implementadas.

Ao considerar as estratégias descritas nesta seção, a aplicação deve contribuir no processo de seleção de problemas e competições, prover o registro dos problemas feitos e das dificuldades enfrentadas, como recomendado por Yoneda e Galen, e incentivar a leitura do editorial da forma que o usuário achar mais apropriado em seu estudo.

# Capítulo 5

## Desenvolvimento da aplicação

A aplicação tem o objetivo de oferecer diferentes ferramentas para auxiliar o treino para programação competitiva em um mesmo lugar. A pesquisa por estratégias descritas no capítulo anterior foi realizada de modo a orientar o desenvolvimento de algumas funcionalidades que visam permitir o uso de algumas delas.

Ela também se propõe a utilizar os recursos da *API* do *Codeforces* de forma eficiente, de modo a enfrentar a limitação de requisições e permitir um acesso rápido e contínuo às informações importantes para o treino.

### 5.1 Ideias iniciais

Antes do início efetivo do desenvolvimento, foi necessário listar as potenciais funcionalidades que poderiam ser implementadas. A partir disso, foi criado um modelo conceitual das informações que seriam necessárias para a construção de uma aplicação que contivesse todas essas funcionalidades. Isso foi importante para a tomada de decisões relacionadas à arquitetura.

#### 5.1.1 Funcionalidades

As principais funcionalidades presentes na proposta inicial do projeto são listadas nessa seção.

##### **Histórico de problemas**

O histórico de problemas consiste no registro de exercícios de programação competitiva feitos pelo usuário. Essa funcionalidade tem a finalidade de agregar os problemas de diversas fontes em um mesmo lugar para o monitoramento do progresso de resolução. Nela, os problemas não resolvidos podem ser guardados para futura resolução e as dificuldades enfrentadas podem ser registradas.

##### **Listas de problemas**

A criação de listas de problemas permite agrupar os problemas adicionados pelo usuário para que o progresso de resolução deles seja monitorado em conjunto.

### **Problemas do *Codeforces***

Os problemas do *Codeforces*, que podem ser obtidos por meio da *API*, são listados para que o usuário possa aplicar filtros para escolher aqueles que deseja resolver. Adicionalmente, o estado do usuário nos problemas também é exibido.

### **Competições do *Codeforces***

Além dos problemas, as competições do *Codeforces* são listadas para que o usuário selecione qual delas deve ser a próxima a ser simulada.

### **Tópicos**

Essa funcionalidade permite a auto-avaliação do usuário nos diferentes tópicos de programação competitiva, com o objetivo final de focar naqueles que se possui menor conhecimento.

### **Time**

As funcionalidades de time englobam as versões correspondentes das criadas para o treino individual ao treino em equipe. Elas incluem a listagem de problemas de competições nas quais o time participou e não foram resolvidos durante a sua execução. Também incluem a listagem de competições da *Gym* do *Codeforces* para a seleção de simulações a serem feitas.

### **Estatísticas**

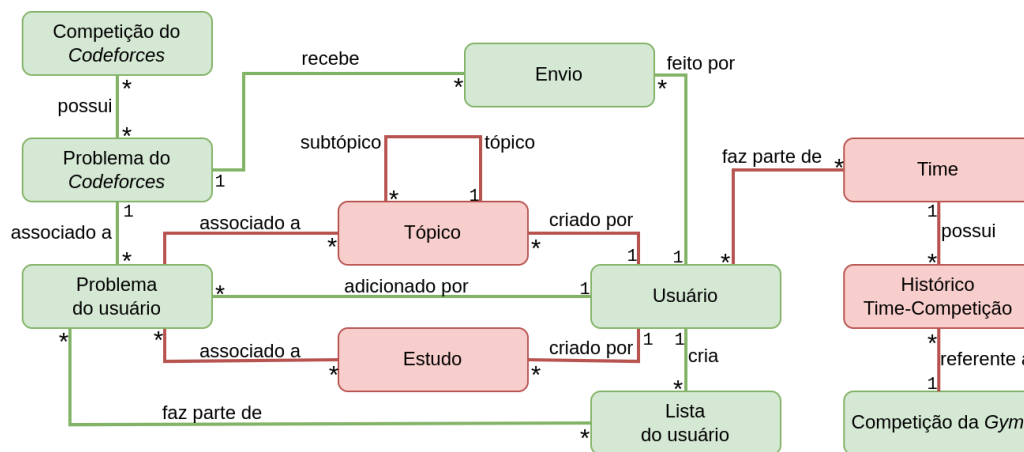
As estatísticas do treino do usuário são métricas para a auto-avaliação do treino com o intuito de ajudar nas decisões sobre seus métodos. Além disso, também servem de incentivo para a consistência do treino.

## **5.1.2 Modelo conceitual**

Além do modelo da *API* do *Codeforces* apresentado na Figura 3.3 (página 11), foi criado um modelo conceitual inicial que considerou as ideias de funcionalidades descritas anteriormente. Esse modelo está exposto na Figura 5.1.

A aplicação desenvolvida possui um subconjunto dessas funcionalidades, visto que, individualmente, requerem uma quantidade significativa de tempo para a implementação. No entanto, todas foram consideradas durante o desenvolvimento para que a aplicação tenha uma arquitetura que permita eventualmente a implementação de cada uma delas.





**Figura 5.1:** Modelo conceitual inicial da aplicação. Em verde estão as entidades relacionadas às funcionalidades implementadas na versão atual.

## 5.2 Análise de requisitos

Algumas das funcionalidades mencionadas exigem a requisição de recursos da *API* do *Codeforces*. No entanto, como mencionado em 3.2.2, a *API* limita a taxa de requisições para duas por segundo. Para amenizar esse problema, um sistema de *cache* pode ser implementado. Outra complicação que precisa ser resolvida é o processamento necessário para gerar novas informações a partir daquelas disponibilizadas pela *API*.

Essas soluções exigem uma capacidade de armazenamento e de processamento que pode ser excessiva para o computador do usuário. Assim, é benéfico que sejam realizadas em um servidor central que, então, envia os dados já processados para ele. Com esse servidor, é possível aproveitar uma mesma requisição à *API* para vários usuários, além de viabilizar funcionalidades que exigem o compartilhamento de dados, como as relacionadas ao treino em equipe.

Um servidor central requer que o usuário esteja conectado à *internet* para o uso da aplicação. No entanto, isso não é grande problema para o contexto da aplicação, pois a conexão já é necessária para acessar as plataformas com problemas e realizar envios de códigos aos respectivos juízes virtuais. Como vantagem, a existência de um servidor central proporciona a utilização da plataforma de qualquer dispositivo com um navegador.

Também é importante para o usuário dispor de um acesso rápido às informações que contribuirão para o seu treino. Elas devem ser constantemente atualizadas e exibidas em seu navegador. Além disso, ele deve poder alternar frequentemente entre as diferentes funcionalidades da aplicação sem ser interrompido por recarregamentos de página.

Isso motivou o uso do padrão Aplicação de Página Única (*SPA*, do inglês *Single-Page Application*). Ele permite o uso contínuo da aplicação sem a atualização da página do navegador. As trocas de páginas aparentes são realizadas, na verdade, pela execução de código *JavaScript*.

## 5.3 Modelo de dados final

Na modelagem final da aplicação, os dados foram divididos em duas categorias. A primeira é a dos dados criados, modificados e removidos pelo usuário. A segunda, consiste nos dados derivados dos recursos da *API* do *Codeforces*.

### 5.3.1 Dados do usuário

#### Modelo conceitual

O modelo conceitual para as informações criadas pelo usuário pode ser visualizado na Figura 5.2. Ele pode adicionar vários problemas e criar várias listas de problemas, enquanto cada problema e cada lista pertence somente a um usuário. Cada problema pode estar associado a várias listas e cada lista pode conter vários problemas. Existe também uma restrição implícita de que os problemas de uma lista devem também pertencer ao criador dela.

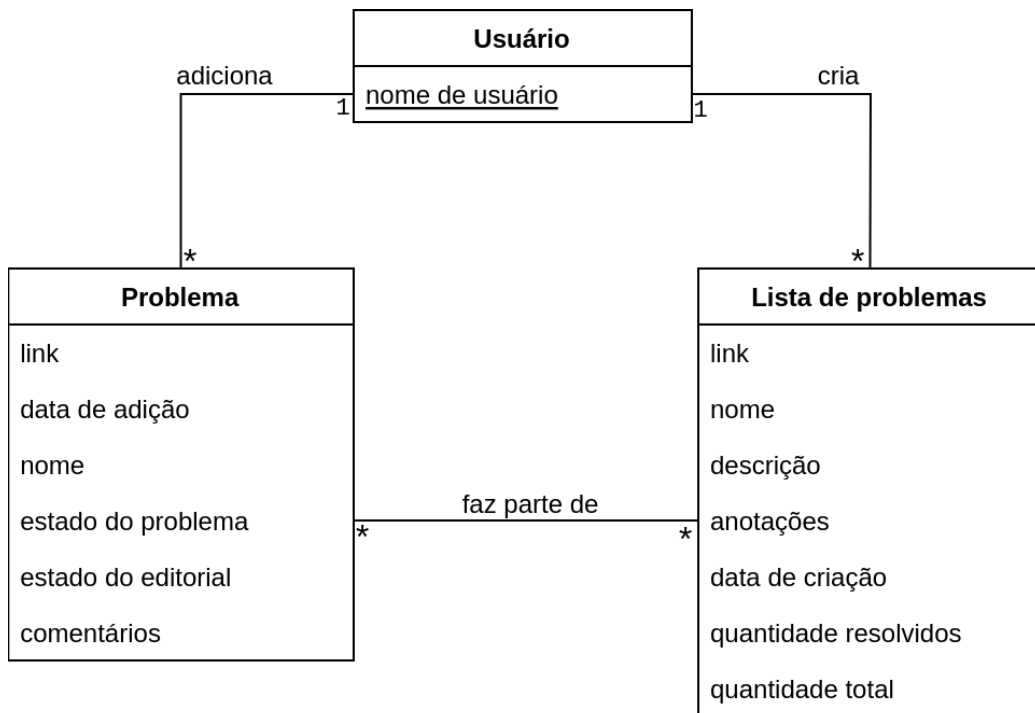


Figura 5.2: Modelo conceitual para os dados do usuário.

#### Implementação

As informações do usuário ficam guardadas no servidor principal — o *back end* da aplicação. Para interagir com elas, é disponibilizada uma *API REST* (do inglês, *Representational state transfer*). Com isso, o usuário consegue realizar as operações de criação, de leitura, de atualização e de deleção em suas informações, conhecidas como operações *CRUD* (*create, read, update, delete*).

Essa *API* do servidor pode ser utilizada por requisições feitas ao caminho {endereço-do-servidor}/api. A relação delas com as operações *CRUD* aplicadas aos problemas do usuário está presente na tabela 5.1.

Operação CRUD	Método HTTP	Caminho (base: /api)	Corpo da requisição
Criação de um novo problema ( <i>Create</i> )	POST	/problems	Campos do novo problema
Leitura de todos os problemas ( <i>Read</i> )	GET	/problems	Vazio
Leitura de um problema ( <i>Read</i> )	GET	/problems/{id}	Vazio
Atualização de um problema ( <i>Update</i> )	PATCH	/problems/{id}	Campos modificados
Deleção de um problema ( <i>Delete</i> )	DELETE	/problems/{id}	Vazio

**Tabela 5.1:** Relação de operações *CRUD* com as requisições *HTTP* que causam a sua execução no servidor principal da aplicação.

A relação para as operações que afetam as listas de problemas é similar, com a diferença de que são feitas no caminho /api/problemLists. Já as operações referentes a associação entre problemas e listas estão descritas na tabela 5.2.

Operação CRUD	Método HTTP	Caminho (base: /api/problemLists/{lid})	Corpo da requisição
Associação de um problema à lista	POST	/problems	Caminho para o problema
Leitura dos problemas associados à lista	GET	/problems	Vazio
Leitura de um problema associado à lista	GET	/problems/{pid}	Vazio
Remoção de associação	DELETE	/problems/{pid}	Vazio

**Tabela 5.2:** Relação de operações para a associação entre problemas e listas de problemas. *lid* e *pid* são os códigos identificadores da lista e do problema, respectivamente.

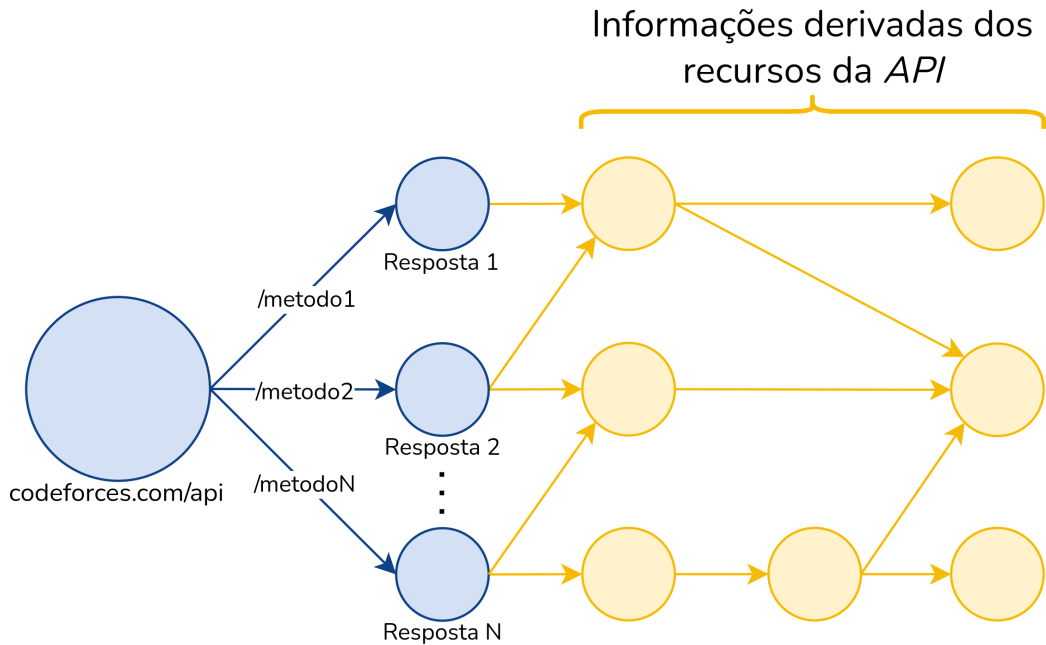
### 5.3.2 Informações da API

#### Modelo de informações

Para as informações derivadas dos recursos da *API* do *Codeforces*, foi construído um modelo recursivo em que as informações são geradas a partir do processamento de outras. O modelo se aproxima de um grafo acíclico direcionado. Cada vértice representa uma informação e os arcos, as dependências entre elas. Um arco do vértice que representa a informação *A* incidente sobre o da *B*, significa que a informação *B* depende da *A*.

A API do *Codeforces* também é representada por um vértice e é o único que não possui dependências. Os arcos que partem dele incidem sobre vértices que representam os objetos devolvidos pelas chamadas aos métodos da API.

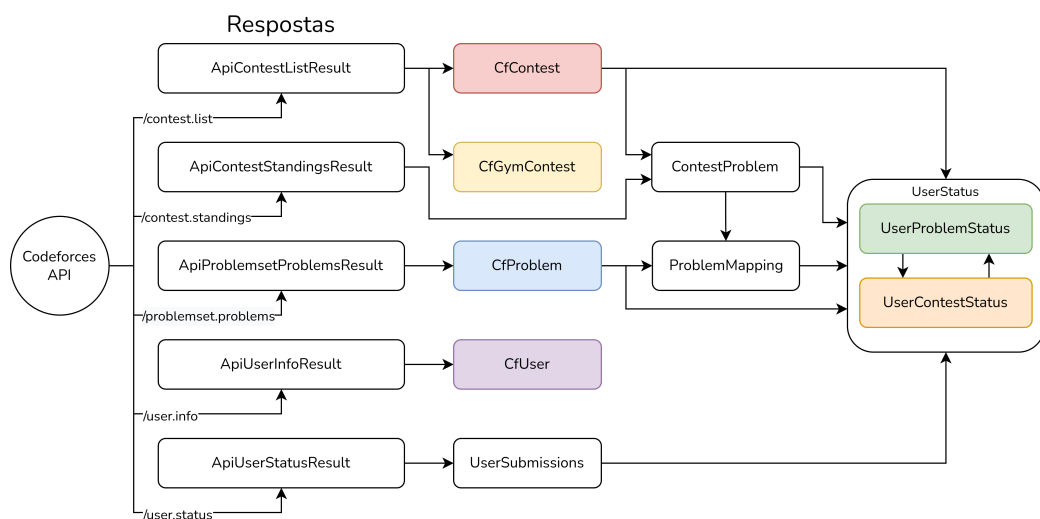
O modelo baseado nesse grafo está representado na Figura 5.3.



**Figura 5.3:** Modelo genérico de informações derivadas dos recursos da API do Codeforces.

### Informações implementadas

A Figura 5.4 exibe as informações utilizadas pela aplicação desenvolvida.



**Figura 5.4:** Informações derivadas da API do Codeforces, com destaque para as que são requisitadas diretamente pelo cliente (front end).

As respostas da *API* são transformadas em objetos de tipo `Result` (Resultado). Por exemplo, o retorno do método `contest.list` é colocado em um objeto `ApiContestListResult`. A partir desses objetos, as informações derivadas são criadas, que são:

- **CfContest** - competições internas;
- **CfGymContest** - competições da *Gym*;
- **CfProblem** - problemas do *Problemset*;
- **CfUser** - dados dos usuários;
- **UserSubmissions** - envios dos usuários;
- **ContestProblem** - problemas das competições internas;
- **ProblemMapping** - mapeamento entre problemas de competições internas e os problemas do *Problemset*;
- **UserStatus** - informações de estado do usuário:
  - **UserProblemStatus** - estado do usuário para um problema;
  - **UserContestStatus** - estado do usuário para uma competição interna.

As informações **CfContest** e **CfGymContest** são obtidas por chamadas ao método `contest.list`. A primeira determina o valor `false` para o parâmetro `gym` enquanto a segunda utiliza o valor `true`. Já a informação **CfProblem** se refere à lista de problemas do *Problemset* obtida pelo método `problemset.problems`.

A informação **CfUser** é recuperada com chamadas ao método `user.info`, desde que especificado o nome de usuário do *Codeforces* que se deseja. Já a informação **UserSubmissions** se preocupa em armazenar a lista de envios de um usuário, recebida como resposta do método `user.status`, que também recebe um nome de usuário como parâmetro.

Essas informações citadas são as mais básicas da aplicação, elas procuram servir como uma representação interna dos recursos do *Codeforces*. Para isso, são guardadas em um formato mais compacto, com somente a inclusão dos campos úteis às funcionalidades implementadas.

Outras informações mais complexas foram criadas para a atribuição correta de estado de usuário aos problemas e às competições. Como a *API* não disponibiliza uma maneira de identificar os problemas duplicados, foi preciso fazer isso na aplicação. A **ContestProblem** armazena os problemas de competições internas *com a inclusão dos problemas duplicados*. Para isso, ela recolhe as informações sobre competições internas em **CfContest** e faz requisições ao método `contest standings` com o objetivo de obter a lista de problemas para cada uma delas.

Já a informação **ProblemMapping** realiza um mapeamento dos problemas das competições internas aos problemas do *Problemset*. Problemas duplicados têm sempre um representante nesse conjunto, o mapeamento tem a função de apontar os problemas duplicados a ele. Para a detecção de duplicação, são verificados o momento de início da competição que contém o problema e o nome dele, ambos obtidos da informação **ContestProblem**. Se dois problemas combinarem para esses dois campos, eles são considerados

iguais. Nesse contexto, a informação **CfProblem** é utilizada para obter a lista de problemas do *Problemset* para a definição dos problemas representantes.

Por fim, a informação **UserStatus** é a que mais interessa e tornou necessária a criação das outras duas descritas nos parágrafos anteriores. Ela agrupa duas informações menores: o **UserProblemStatus** e **UserContestStatus**. Esse agrupamento é feito de modo a não se quebrar a propriedade acíclica do grafo de informações.

Primeiramente, é calculado o estado dos problemas do *Problemset* para um usuário. A lista desses problemas é obtida com a informação **CfProblem**. Depois, recolhem-se os envios do usuário com o uso da informação **UserSubmissions**. Eles trazem o veredito do envio e o problema da competição a que se refere. O problema é mapeado ao seu representante no *Problemset* com a **ProblemMapping** e atribui-se o estado de **Solução aceita** se pelo menos um dos envios possui o veredito desse tipo. Se houver pelo menos um envio e nenhum deles corresponder à resolução do problema, então ele recebe o estado de **Solução incorreta**. Caso não haja nenhum envio, o problema fica com o estado **Não tentado**.

Após isso, é calculado o estado das competições internas, obtidas da informação **CfContest**. O estado da competição é definido conforme o estado de seus problemas, recolhidos da informação **ContestProblem**. Caso todos os problemas dela possuam estado **Não tentado**, a competição terá o estado **Limpa**. Caso todos estejam com **Solução aceita**, a competição recebe o estado **Completa**. Se nenhuma dessas situações ocorrer, a competição recebe o estado **Tentada**. Em uma última etapa que gera a ciclicidade dessas duas informações, o estado da competição de cada problema do *Problemset* é adicionado à informação **UserProblemStatus**, para uso posterior no cliente (*front end*).

As informações relacionadas à *API* do *Codeforces* estão disponíveis no caminho `{endereco-do-servidor}/info`.

### Sistema de *cache*

Para lidar com o limite imposto pela *API* do *Codeforces* e para permitir um acesso contínuo às informações processadas, foi incluído um sistema de *cache* das informações.

A primeira forma de *cache* foi adicionada diretamente nas requisições à *API*. Qualquer que for realizada pela aplicação terá o seu retorno armazenado em um repositório específico que guarda a sua resposta e o momento em que ela foi recebida. Isso possibilita a recuperação de dados que já estão no *cache* ao invés de se executar uma nova requisição.

Para isso, foi definida uma tolerância de quão recentes os recursos da *API* precisam estar. Se o recurso no *cache* estiver dentro dessa tolerância, ele é recuperado dele, caso contrário, a requisição é feita. Essa tolerância varia de recurso a recurso e, inclusive, depende da informação que o solicita. Por exemplo, a lista de competições do *Codeforces* atualiza somente quando há novas competições, o que acontece aproximadamente três vezes por semana. Já os envios de usuário são feitos muito mais frequentemente, a depender de quantos problemas ele resolve por dia. Um usuário ativo pode realizar dezenas de envios diariamente, muitos deles em um curto tempo, como durante a participação em uma competição.

Para a lista de competições, uma nova requisição a cada hora é suficiente para a adição de novas competições pouco tempo depois que elas se tornarem disponíveis na plataforma. Já para envios do usuário, é necessário manter uma frequência de alguns segundos, já que após um envio é essencial que o usuário tenha a informação atualizada sobre os problemas resolvidos na aplicação, de modo que possa já escolher o próximo sem complicações.

Uma categoria de informação que será requisitada poucas vezes são aquelas relacionadas a competições finalizadas. A lista de problemas da competição, por exemplo, dificilmente irá mudar, então é possível colocar um intervalo de tolerância muito grande. Isso é bem importante para informações que dependem de milhares de requisições à *API*, como a **ContestProblem**. Elas podem ser feitas uma única vez para adicionar as informações à aplicação e depois disso só será necessário fazê-las para novas competições.

Um exemplo de variação de tolerância para o mesmo recurso é para uma informação hipotética para a recomendação de problemas: pode ser necessário pegar as listas de envios de milhares de usuários para criar a recomendação, mas não seria necessário ter essas listas frequentemente atualizadas como é o caso para o processamento do estado de problemas para os usuários que estão utilizando a aplicação no momento.

Também é realizado o *cache* das informações após elas serem processadas, de modo a evitar a repetição de processamento com os mesmos dados. Isso é necessário porque o usuário vai continuamente requisitá-las ao servidor e seria dispendioso recalculá-las toda vez. No que foi implementado, uma informação somente é reprocessada caso alguma de suas dependências seja mais recente que o seu último processamento ou se algum recurso da *API* tiver fora da validade especificada pela tolerância.

É importante notar que as informações são recursivas e a verificação da necessidade de um reprocessamento também é feita dessa forma. Por exemplo, ao verificar se **UserStatus** deve ser atualizada, será verificado se suas dependências devem ser ou já foram atualizadas. Em específico, a verificação de atualização da **UserSubmissions** pode detectar que a lista de envios do usuário precisa ser requisitada novamente. Após a requisição, será atualizada a **UserSubmissions** e, por fim, a **UserStatus**.

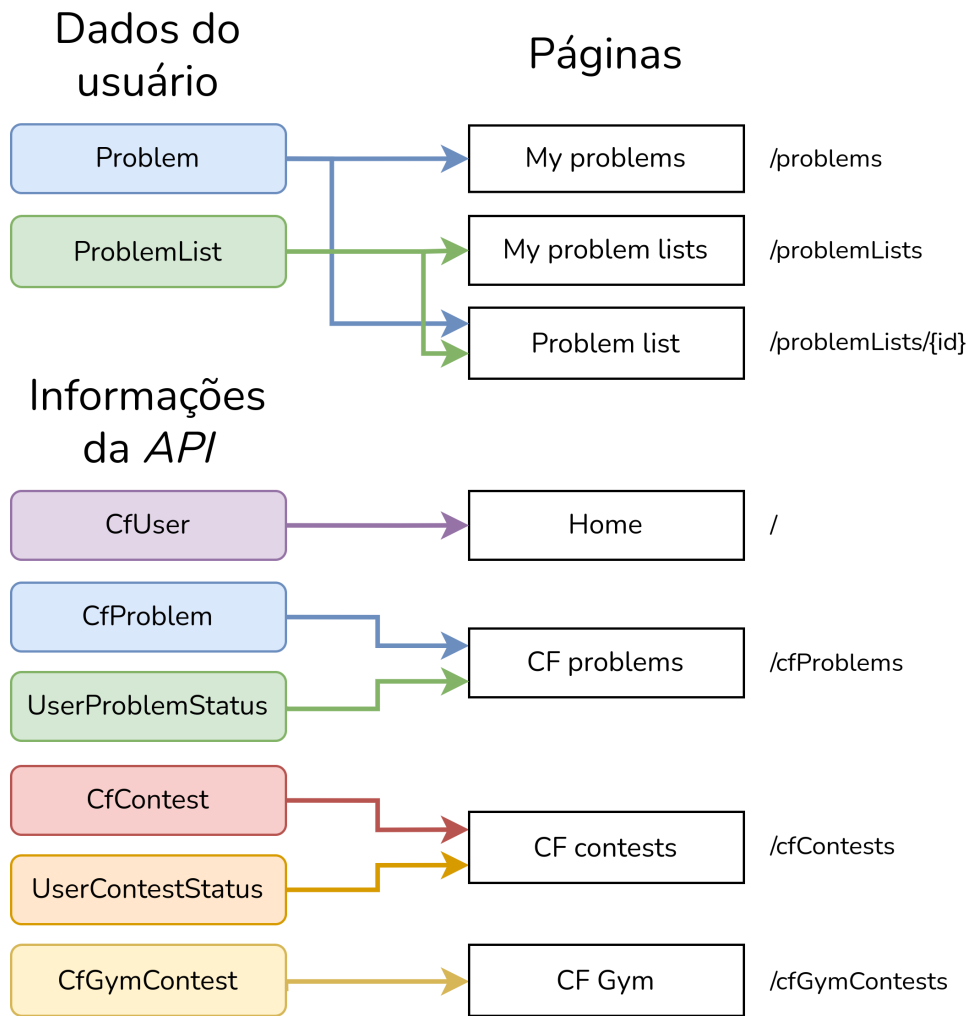
### 5.3.3 Uso dos dados do servidor pelas páginas da aplicação

Foram desenvolvidas páginas da aplicação para a criação e modificação de dados do usuário e para a consulta das informações da *API* do *Codeforces*. A relação dos dados com as páginas está representada na Figura 5.5. As páginas são descritas com detalhes na Seção 6.1 do próximo capítulo.

Os dados sobre **problemas** (*Problem*) e **listas de problemas** (*ProblemList*) são utilizados nas páginas **Problemas do usuário** (*My problems*) e **Lista de problemas do usuário** (*My problem lists*), respectivamente. Ambos são também utilizados na página **Lista de problema** (*Problem list*).

A informação *CfUser* é usada na **Página inicial** (*Home*) para a exibição das informações do usuário selecionado.

As informações *CfProblem* e *UserProblemStatus* são usadas na página **Problemas do Codeforces** (*CF problems*) para mostrar a lista de problemas do *Problemset* e o estado



**Figura 5.5:** Relação dos dados do servidor com as páginas do cliente.

do usuário selecionado em relação a eles, respectivamente.

De forma semelhante, as informações *CfContest* e *UserContestStatus* são usadas na página **Competições do Codeforces (CF contests)** para mostrar as competições internas e o estado do usuário nelas, respectivamente.

Finalmente, a informação *CfGymContest* é utilizada para a listagem na página **Competições da Gym (CF Gym)**.

## 5.4 Detalhes de implementação

A aplicação é *full stack*, ou seja, consiste em um servidor (*back end*) e um cliente (*front end*).



### 5.4.1 Servidor (*back end*)

#### Tecnologias usadas

O servidor (*back end*) foi desenvolvido com o *framework Spring*. Para um início rápido com várias configurações comuns já definidas, utilizou-se o *Spring Boot*. A linguagem *Kotlin* foi escolhida para a escrita do código do servidor. A *API REST* para os dados do usuário foi implementada com *Spring Data REST*, que só exige a definição dos recursos em classes *Kotlin* e já disponibiliza a modificação deles pelas requisições descritas em 5.3.1. Em relação ao banco de dados, foi utilizado o *MongoDB*, baseado em documentos. Já a interface desse banco com o servidor foi construída com o uso da biblioteca *Spring Data MongoDB*.

O *Gradle* ([gradle.org](http://gradle.org)) foi usado como ferramenta de automação de compilação de código. Ele permite a fácil adição de dependências com o uso do arquivo de configuração `build.gradle.kts`.

Para a conversão dos dados no formato *json* obtidos da *API* do *Codeforces* para objetos em *Kotlin*, foi usada a biblioteca *Gson*.

#### Estrutura do código

O código relevante do servidor principal pode ser encontrado no caminho `/server/src/main/kotlin/com/gustavo/competitiveprogrammingapp` a partir do repositório da aplicação<sup>1</sup> e é organizado nos seguintes pacotes:

- **cfApi** - interface com a *API* do *Codeforces*
  - **resources** - classes em *Kotlin* que representam os recursos presentes nas respostas da *API*
- **config** - arquivos de configuração para o *Spring*
- **information** - informações derivadas dos recursos da *API*
  - **domain** - classes das unidades de informação
  - **processors** - classes que processam informações
  - **repositories** - repositórios das informações
- **rest** - definição das informações passíveis de operações *CRUD*
  - **problem** - classes relacionadas aos problemas de usuário
  - **problemList** - classes relacionadas às listas de problemas de usuário
- **util** - classes de utilidade geral

#### Interface com a *API* do *Codeforces*

O pacote **cfApi** contém as classes que fazem a interação com a *API* do *Codeforces*. O arquivo `responses.kt` contém as classes do tipo `Result` citadas em 5.3.2. A classe

<sup>1</sup> <https://github.com/gustavoM32/competitive-programming-app>

**Fetcher** é responsável por fazer as requisições ou recuperar o que já está guardado no *cache*, a depender da tolerância especificada.

O *cache* das requisições é definido por objetos do tipo **UrlCache**, que podem ser vistos no Programa 5.1. O campo `apiResource` é a chave identificadora de cada recurso e é definida pelo método da *API*, com a inclusão dos parâmetros usados. Por exemplo, para as competições da *Gym*, a chave seria `/contest.list?gym=true`. O campo `json` guarda a resposta no formato *JSON* dessa requisição e o campo `lastUpdate` guarda o momento em que ela foi obtida.

---

**Programa 5.1** Classe `UrlCache`, que armazena respostas da *API* do *Codeforces*.

---

```

1  @Document("urlCache")
2  data class UrlCache(
3      @Id
4      val apiResource: String,
5      val json: String,
6      val lastUpdate: LocalDateTime
7  )

```

---

A classe **CfApiResponseFetcher** replica os métodos da *API* em métodos de *Kotlin*. Ela procura seguir a mesma nomenclatura deles e de seus parâmetros. Os métodos relacionados à lista de competições são mostrados no Programa 5.2. O método `getContestListLastUpdate` devolve o último momento em que a lista de competições foi guardada no *cache*. Já o método `getContestList` recupera a lista de competições dados os parâmetros `gym` e `requiredRecency`. O primeiro é o parâmetro do método da *API* e o segundo é a tolerância para a recuperação do que está no *cache*.

---

**Programa 5.2** Exemplo de métodos da classe `CfApiResponseFetcher`.

---

```

1  @Component
2  class CfApiResponseFetcher(val fetcher: Fetcher) {
3      companion object {
4          val DEFAULT_RECENCY: Duration = Duration.ofSeconds(1000L)
5      }
6      ...
7      fun getContestListLastUpdate(gym: Boolean = false): LocalDateTime {
8          return fetcher.getLastUpdate("/contest.list?gym=$gym")
9      }
10
11     fun getContestList(gym: Boolean = false, requiredRecency: Duration =
12         DEFAULT_RECENCY): ApiResponseResult {
13         return fetcher.getResource("/contest.list?gym=$gym",
14             ApiResponseResult::class.java, requiredRecency)
15     }
16     ...
17 }

```

---

## Informações derivadas da API

No pacote **information** estão as classes relacionadas às informações. As classes que fazem o processamento das informações estão no pacote **information.processors**. Eles possuem um formato específico, como é possível ver no exemplo resumido para a informação **CfGymContest** no Programa 5.3.

---

### Programa 5.3 Exemplo de processador de informação.

---

```

1  @Component
2  class CfGymContestProcessor(
3      val repository: CfGymContestRepository,
4      val informationService: InformationService,
5      private val cfApiResponseFetcher: CfApiResponseFetcher
6  ) {
7      /* Campos estáticos da classe. */
8      companion object {
9          const val INFORMATION_ID = "CfGymContest"
10         var isUpdating = false
11         // Tolerância para requisições para a lista de competições
12         val CONTEST_LIST_CACHE_TOLERANCE: Duration = Duration.ofHours(1)
13     }
14
15     /* Verifica se uma atualização deve ocorrer.
16      * Se sim, reprocessa a informação. */
17     fun update(): UpdateResponse {...}
18
19     /* Realiza o processamento da informação. */
20     fun process() {
21         // Recolhe dependências
22         val contestList = cfApiResponseFetcher.getContestList(true,
23             CONTEST_LIST_CACHE_TOLERANCE)
24         // Processa a informação
25         ...
26         // Salva o que foi processado
27         repository.saveAll(cfGymContests)
28     }
29
30     /* Devolve a informação armazenada. */
31     fun get(): List<CfGymContest> { return repository.findAll() }
32
33     /* Remove a informação armazenada. */
34     fun reset() {
35         repository.deleteAll()
36         informationService.delete(INFORMATION_ID)
37     }
38 }

```

---

Os momentos de último processamento de cada informação são armazenados em objetos do tipo **Information**, de forma semelhante ao que é feito com **UrlCache**.

A classe **InformationController** expõe as informações no caminho `/info` para a interação com elas. Como exemplo, considerando a informação **CfGymContest**, as requisições **HTTP** com o método **GET** para o caminho `/info/cfGymContests` devolvem as

informações armazenadas e as para o caminho `/info/cfGymContests/update` executam a verificação de atualização da informação.

A arquitetura geral do servidor é exibida na Figura 5.6.

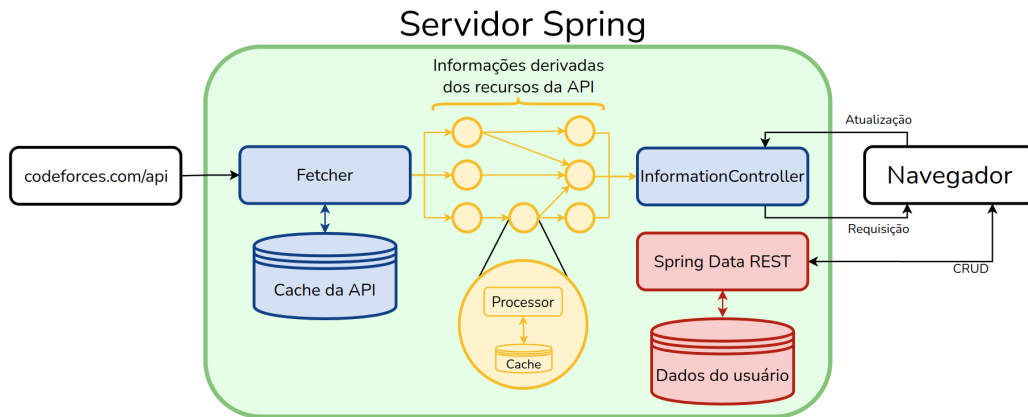


Figura 5.6: Arquitetura do servidor.

## 5.4.2 Cliente (*front end*)

### Tecnologias usadas

O cliente (*front end*) possui páginas para a modificação dos dados do usuário e para a visualização das informações do *Codeforces*. O seu desenvolvimento foi feito com *React* pelo uso do *Create React App*, que permite iniciar o trabalho com o *framework* mais rapidamente. A maior parte do código foi escrito com *TypeScript*, uma versão de *JavaScript* com especificação de tipos para variáveis. Para verificação de problemas de sintaxe no código foi utilizado a biblioteca *ESLint*.

Os dados do servidor são obtidos com a execução de requisições a ele. Essas requisições são feitas nos componentes do *React* com o uso das bibliotecas *Axios* e *React Query*. Esta última disponibiliza várias customizações sobre elas, como a configuração de requisições automáticas periódicas e também em mudança de foco do navegador. Isso foi utilizado para manter os dados constantemente atualizados enquanto o usuário está na aplicação e também para desencadear uma atualização quando troca as abas do navegador (ou as janelas de seu sistema) para visualizar a aplicação.

Para a interface do usuário (*UI*, do inglês *User Interface*), foram usados os componentes da *Material UI*, uma implementação da *Material Design* específica para o uso com *React*. Complementarmente, foi utilizado *Bootstrap* em algumas partes para pequenos ajustes. Por fim, a biblioteca *ag-grid-react* facilitou a criação das tabelas presentes na maioria das páginas da aplicação.

### Estrutura do código

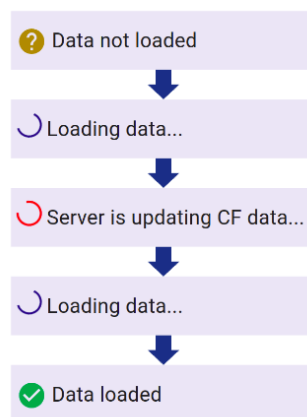
O código relevante do cliente pode ser encontrado no caminho `/client/src` do repositório da aplicação. Sua estrutura é definida a seguir.

- **api** - arquivos responsáveis pela interação do cliente com o servidor
- **components** - componentes<sup>2</sup> do *React* criados para a aplicação
  - **problem** - componentes relacionados aos problemas do usuário
  - **problemList** - componentes relacionados às listas de problemas do usuário
- **constants** - definição de constantes
- **hooks** - *hooks*<sup>3</sup> do *React* customizados
- **pages** - componentes das páginas da aplicação
- **styles** - arquivos de estilos *CSS* para a aplicação
- **utils** - arquivos de utilidade geral

Entre os principais componentes definidos na pasta **components**, estão: **DataGrid**, tabela utilizada na maioria das páginas para a exibição das informações; **DataLoadingInfo**, informação sobre o carregamento dos dados do servidor; **Sidebar**, que define a barra lateral de navegação entre as páginas; **ProblemDialog** e **ProblemListDialog**, diálogos de criação e modificação de problemas e listas de problemas, respectivamente.

### Interação com o servidor

As requisições para executar as operações *CRUD* estão definidas no arquivo `api/crud.ts`. Já os *fetchers*, métodos utilizados pela biblioteca *React Query* para requerer os dados para as páginas, estão definidos em `api/fetchers.ts`. Para a execução das operações *CRUD* em diferentes componentes, foram criados *hooks* para elas no arquivo `hooks/crudHooks.ts`.



**Figura 5.7:** Sequência usual de carregamento de dados no cliente.

O processo de carregamento e atualização de informações do servidor é mostrado para o usuário como na Figura 5.7. Primeiramente, não há dados no navegador (**Data not loaded**).

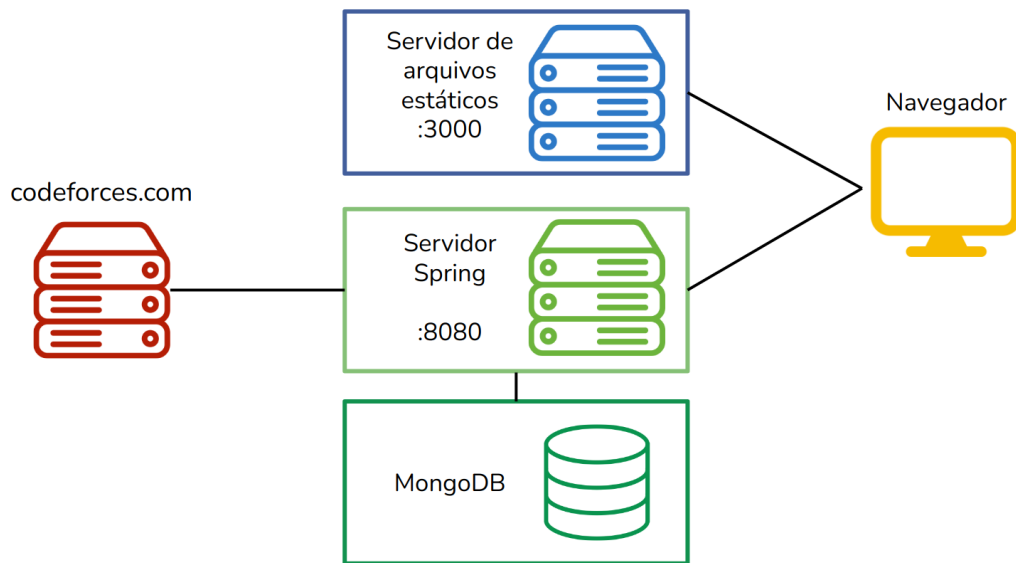
<sup>2</sup> <https://reactjs.org/docs/components-and-props.html>

<sup>3</sup> <https://reactjs.org/docs/hooks-intro.html>

Rapidamente, é iniciado o processo de requisição ao servidor (**Loading data...**). Para o caso de uma informação derivada da API do *Codeforces*, os dados armazenados no *cache* são devolvidos imediatamente e é feito um pedido de atualização. Se essa atualização for confirmada, o servidor informa o navegador (**Server is updating CF data...**), e a frequência de requisições aumenta para que os dados atualizados sejam refletidos no navegador em tempo real (**Loading data...**). Assim que a atualização terminar, mais uma requisição é feita para recuperar os dados atualizados e o processo termina (**Data loaded**).

### 5.4.3 Interação entre os componentes

Para a compilação dos arquivos tanto do servidor quanto do cliente ser facilmente executada em qualquer sistema, foi utilizado o *Docker*. Assim, os arquivos são compilados dentro de contêineres e a única dependência que precisa ser instalada diretamente no sistema é o próprio *Docker*. Os arquivos do servidor são compilados em um executável de *Java* que abre um servidor *Spring* na porta 8080 por padrão. Os arquivos do cliente são empacotados em arquivos estáticos em HTML, CSS e JavaScript. Para que tais arquivos possam ser requisitados pelos navegadores dos usuários, eles são servidos com o uso da biblioteca *serve* do *Node.js* pela porta 3000. Além disso, também é necessário ter o banco de dados em *MongoDB* para o servidor *Spring*.



**Figura 5.8:** Arquitetura da aplicação.

O *Docker Compose* foi utilizado para simplificar a criação de contêineres para a compilação dos códigos e para execução dos dois servidores e do banco de dados. Ele consegue fazer tudo isso com a execução de poucos comandos. A arquitetura geral da aplicação está exposta na Figura 5.8, com destaque para os três contêineres do *Docker* necessários para ela funcionar corretamente e a interação entre eles.

# Capítulo 6

## Aplicação *web* desenvolvida

### 6.1 Páginas da aplicação

Foram desenvolvidas sete páginas no total para a aplicação. Uma delas é a inicial, a primeira que o usuário visualiza ao acessá-la. Outras três são dedicadas a funcionalidades relacionadas ao *Codeforces* e as três remanescentes priorizam a administração de informações criadas pelo próprio usuário. Uma barra lateral expansível (Figura 6.1) foi adicionada para facilitar a navegação por essas páginas.

A maioria das páginas contém o botão **Atualizar informações** (*Update data*), que aciona o processo de requisição de dados ao servidor. Próximo a ele, está o estado dessa requisição. As requisições também são executadas de forma automática periodicamente e ao ocorrer a mudança de foco do navegador (quando o usuário troca de abas, por exemplo).

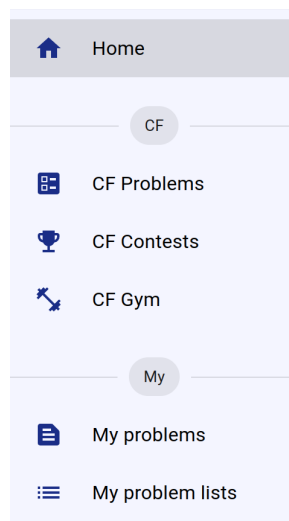


Figura 6.1: Barra lateral de navegação expandida.

### 6.1.1 Página inicial (*Home*)

Na **Página inicial (*Home*, Figura 6.2)**, o usuário pode definir o seu nome de usuário no *Codeforces*, chamado por *handle* na plataforma. Para isso, ele deve preencher o campo *CF user name* e pressionar o botão **Definir (*Set*)**.

A alteração só irá ser realizada se o campo não estiver vazio e se o usuário existir no *Codeforces*. Se for confirmada, esse nome fica guardado na memória do navegador e irá permanecer mesmo se o usuário sair da aplicação e voltar posteriormente.

Todas as outras páginas da aplicação passarão a se referir ao usuário selecionado. Adicionalmente, serão mostradas algumas informações sobre esse usuário nessa mesma página, como o seu nome completo e sua pontuação no *Codeforces*, caso estejam presentes na plataforma. O nome do usuário também possui um *link* que leva para o seu perfil nela.

Existe também o botão **Remover definição (*Unset*)** para remover a seleção de um usuário específico.



Figura 6.2: Página inicial (*Home*).

### 6.1.2 Páginas relacionadas ao *Codeforces*

Nas páginas relacionadas ao *Codeforces*, são exibidas informações provenientes da *API* disponibilizada pela plataforma. Algumas delas referentes ao usuário selecionado na página inicial.

#### Problemas do *Codeforces*

A página **Problemas do *Codeforces* (CF Problems, Figura 6.3)** apresenta os problemas das competições regulares da plataforma ou, mais precisamente, aqueles do *Problemset*.



O topo da página contém filtros para a tabela de problemas. O primeiro filtra os problemas pelo nome, sem a diferenciação maiúsculas de minúsculas. Também é possível filtrá-los pelo código. Já o segundo permite filtrá-los pelo estado de resolução do usuário, que pode ser:

- **Não tentado (*Not tried*)** — o usuário não enviou nenhuma tentativa de resolução para o problema;
- **Solução incorreta (*Wrong answer*)** — o usuário enviou pelo menos solução no *Codeforces*, mas nenhuma foi sucedida;
- **Solução aceita (*Accepted*)** — o usuário enviou uma solução aceita pelo juiz virtual do *Codeforces*.

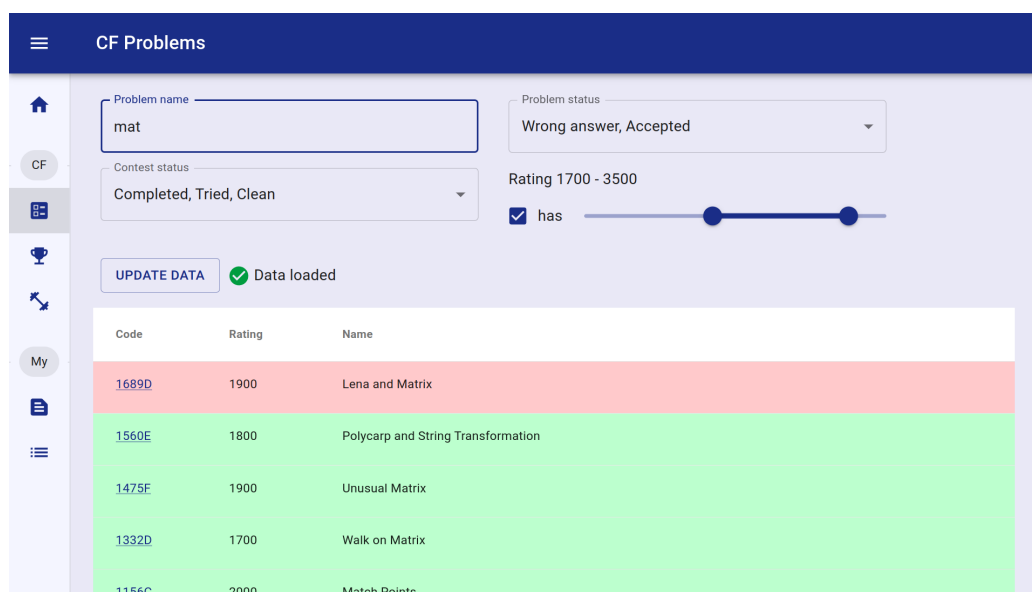


Figura 6.3: Página de problemas do Codeforces (*CF Problems*).

O terceiro filtro considera o estado da competição da qual o problema faz parte, que assume uma das seguintes opções:

- **Limpa (*Clean*)** — não foi feita nenhuma tentativa em nenhum dos problemas da competição;
- **Tentada (*Tried*)** — algum envio foi feito a pelo menos um dos problemas da competição;
- **Completa (*Completed*)** — todos os problemas da competição foram resolvidos.

Esses dois últimos filtros permitem a seleção de mais um estado em simultâneo.

O último filtro é aplicado sobre a pontuação de dificuldade do problema (*rating*). Um intervalo de pontuação pode ser selecionado. Também há uma caixa de seleção para filtrar somente problemas que possuem essa pontuação especificada.

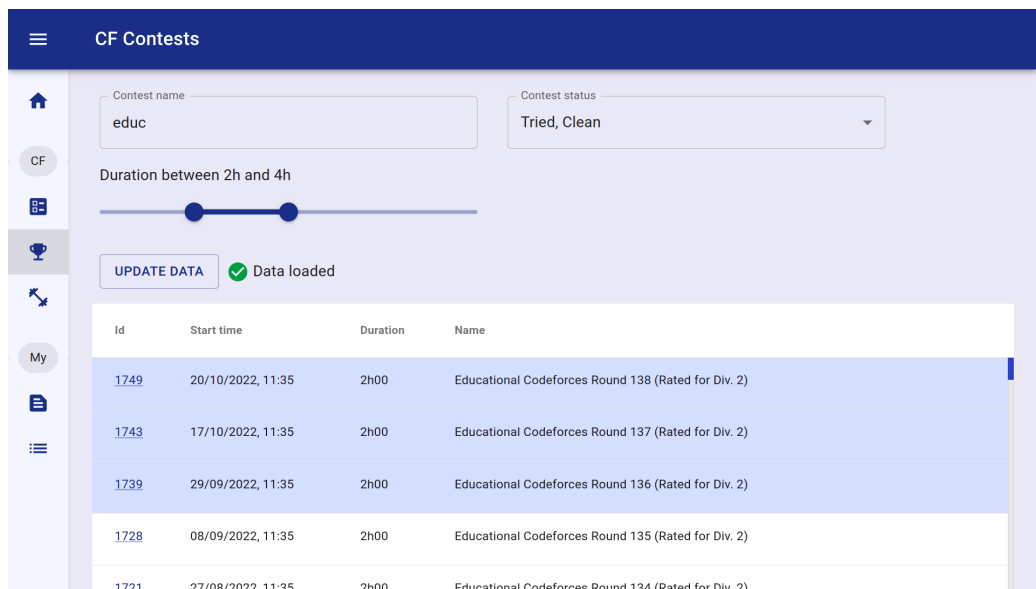
Abaixo dos filtros, está a tabela com os problemas. A primeira coluna possui o **código do problema (*Code*)**, composto pelo identificador de sua competição concatenado com a letra

dele. Essa coluna também possui um *link* que abre a página do problema no *Codeforces* em uma nova aba do navegador. A segunda, possui a **pontuação de dificuldade** do problema (**Rating**). Por último, está o **nome** do problema (**Name**).

Cada linha da tabela é colorida conforme o estado do problema: vermelho para os com somente tentativas falhas de resolução, e verde para os resolvidos.

### Competições internas do Codeforces

A página **Competições internas do Codeforces (CF Contests, Figura 6.4)** oferece as competições organizadas por usuários da plataforma e realizadas nela própria.



The screenshot shows the 'CF Contests' interface. At the top, there are filters for 'Contest name' (set to 'educ') and 'Contest status' (set to 'Tried, Clean'). Below these is a 'Duration between 2h and 4h' slider. An 'UPDATE DATA' button is visible with a 'Data loaded' confirmation. The main part of the page is a table of contests with the following data:

Id	Start time	Duration	Name
1749	20/10/2022, 11:35	2h00	Educational Codeforces Round 138 (Rated for Div. 2)
1743	17/10/2022, 11:35	2h00	Educational Codeforces Round 137 (Rated for Div. 2)
1739	29/09/2022, 11:35	2h00	Educational Codeforces Round 136 (Rated for Div. 2)
1728	08/09/2022, 11:35	2h00	Educational Codeforces Round 135 (Rated for Div. 2)
1721	27/08/2022, 11:35	2h00	Educational Codeforces Round 134 (Rated for Div. 2)

**Figura 6.4:** Página de competições internas do Codeforces (CF Contests).

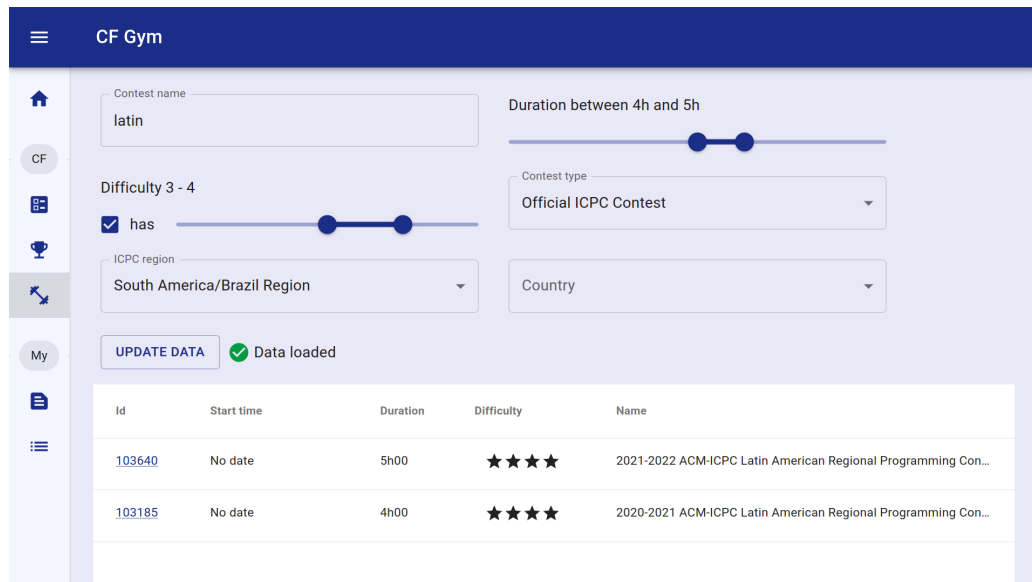
Na parte superior, estão três filtros para as competições. Os dois primeiros são semelhantes aos contidos na página de problemas. Eles filtram as competições por nome e por estado, respectivamente. O filtro por nome é útil para encontrar competições de categorias específicas como, por exemplo, *Educational Codeforces Round* ou *Codeforces Global Round*. Adicionalmente, esse filtro também considera o número identificador da competição. O terceiro filtro permite selecionar as competições pela duração. Nele, é possível definir o intervalo de durações que se deseja.

Adiante, está a tabela de competições. A primeira coluna contém o **identificador** da competição (**Id**), que também possui um *link* para ela no *Codeforces*. Em seguida, está a **data de início** da competição (**Start time**), acompanhada pela **duração** (**Duration**) e o seu **nome** (**Name**).

Cada linha é colorida segundo o estado da competição para o usuário do *Codeforces* selecionado: azul para aquelas que possuem algum problema com pelo menos uma tentativa, e verde para as que tiveram todos os seus problemas resolvidos.

## Competições da Gym do Codeforces

A última página relacionada ao *Codeforces* é a **Competições da Gym (CF Gym, Figura 6.5)**. Ela contém competições da seção *Gym* da plataforma.



**Figura 6.5:** Página de competições da Gym do Codeforces (CF Gym).

No alto da página, estão seis filtros para essas competições. Os dois primeiros são definidos de forma semelhante àqueles da página de competições internas. O terceiro filtro se refere à dificuldade da competição especificada por quem a adicionou ao *Codeforces*. É possível definir um intervalo para dificuldade e ativar a caixa de seleção para requerer somente competições que a possuem definida. O quarto é sobre a categoria da competição. Com ele, é possível filtrar, por exemplo, somente as oficiais da *ICPC*. O quinto concerne sobre a região oficial da *ICPC* em qual a competição foi realizada. Por último, o sexto permite filtrar pelo país de execução da competição.

A tabela de competições é análoga a da página descrita na seção anterior. Com a adição de uma coluna que define a **dificuldade** da competição (*Difficulty*).

### 6.1.3 Páginas do usuário

Nas páginas do usuário, o competidor pode adicionar seus próprios problemas, do *Codeforces* ou não, e fazer anotações sobre eles. Ademais, também é possível compor listas de problemas e monitorar o progresso da resolução deles.

#### Problemas

Na página **Problemas do usuário (My problems, Figura 6.6)** é possível registrar problemas de quaisquer fontes que se desejar, não limitados àqueles do *Codeforces*.

Ela contém uma tabela com todos os problemas adicionados pelo usuário. Acima dela, está o botão **Criar (Create)** que, ao ser pressionado, abre um diálogo com um formulário

para a adição de um novo problema. Esse formulário é semelhante ao de edição do problema, que será descrito posteriormente.

Addition date ↓	Link	Name	Status	Editorial	Action
06/12/2022, 10:32	<a href="#">Link</a>	Transform the Expression	Accepted	Not read	<a href="#">EDIT</a> <a href="#">DELETE</a>
05/12/2022, 13:13	<a href="#">Link</a>	Helga Hufflepuff's Cup	Read	Read before AC	<a href="#">EDIT</a> <a href="#">DELETE</a>
04/12/2022, 10:56	<a href="#">Link</a>	Remove Directed Edges	Wrong answer	Read before AC	<a href="#">EDIT</a> <a href="#">DELETE</a>
03/12/2022, 11:21	<a href="#">Link</a>	UN Finals	Accepted	Read after AC	<a href="#">EDIT</a> <a href="#">DELETE</a>
02/12/2022, 10:43	<a href="#">Link</a>	AquaMoon and Chess	Accepted	Read after AC	<a href="#">EDIT</a> <a href="#">DELETE</a>

Figura 6.6: Problemas do usuário (*My problems*).

Cada linha da tabela representa um problema. A **data de adição** do problema (*Addition date*) é armazenada de modo a manter uma ordem fixa e padrão, para que o usuário lembre facilmente do seu progresso e dos problemas que adicionou.

Também é permitido incluir um **link** para o enunciado do problema. Assim, ele pode ser lido facilmente e, a depender de qual *site* ele está localizado, pode ser resolvido nele também. Desse modo, o usuário consegue ter acesso a exercícios de fontes diversas em um único lugar.

Existe uma coluna para o **nome** do problema (*Name*), que pode ser preenchida da maneira que o usuário achar mais conveniente para o identificar. Em geral, as plataformas que disponibilizam questões de programação competitiva providenciam um nome próprio para eles ou, pelo menos, um código que os identificam.

Duas colunas são reservadas para tipos diferentes de estado do problema. A primeira é relacionada ao **estado de resolução** do problema (*Status*), que pode ser:

- **Não lido** (*Not read*) — o enunciado do problema não foi lido pelo usuário;
- **Lido** (*Read*) — o enunciado do problema foi lido, mas ele ainda não enviou uma solução;
- **Solução incorreta** (*Wrong answer*) — o usuário enviou pelo menos solução, mas nenhuma foi sucedida;
- **Solução aceita** (*Accepted*) — o usuário enviou uma solução aceita pelo juiz virtual.

Ela, em conjunto com as cores das linhas na tabela, ajuda o usuário a distinguir os problemas que já resolveu daqueles que ainda precisam ser resolvidos.

Já a segunda define o **estado de leitura do editorial** (*Editorial*), ou a solução do problema, que pode ser:

- **Não lido** (*Not read*) – o editorial não foi lido;
- **Lido antes de resolver** (*Read before AC*) – o editorial foi lido antes do problema ser resolvido;
- **Lido após resolver** (*Read after AC*) – o editorial foi lido após a resolução do problema.

Ela é importante para identificar o quanto de ajuda o usuário precisou para resolver o problema. Se ele resolveu o problema sem consultar o editorial, ele pode usar essa coluna para marcar que leu o editorial posteriormente. Dessa maneira, ele é incentivado a ler outras soluções para a questão, que podem diferir da que ele encontrou e podem trazer novas ideias para problemas futuros. Caso tenha resolvido após ler o editorial, ele pode registrar isso para indicar que precisou de ajuda para encontrar a solução.

A última coluna consiste em **ações** relacionadas ao problema da linha correspondente (*Action*). O botão **Editar** (*Edit*) abre um diálogo com um formulário para a edição das informações do problema e o botão **Deletar** (*Delete*) remove o problema completamente da aplicação.

**Edit problem**

Name  
Transform the Expression

Link  
https://www.spoj.com/problems/ONP/en/

Status  
 Not read  Read  WA  AC

Editorial  
 Not read  Read before  Read after

Comments  
Problema de implementação que envolve transformar uma expressão infixa para pós-fixa.

CANCEL SAVE

**Figura 6.7:** Formulário de edição de problema.

O diálogo de edição pode ser visto na Figura 6.7. Os dois primeiros campos definem o nome e o *link* do problema. Nos dois seguintes, é possível definir o estado de resolução de problema e o de leitura do editorial, respectivamente. O último campo de texto permite adicionar **comentários** (*Comments*) sobre o problema. Isso é útil para o registro de informações importantes como, por exemplo, o tópico utilizado para a sua resolução,

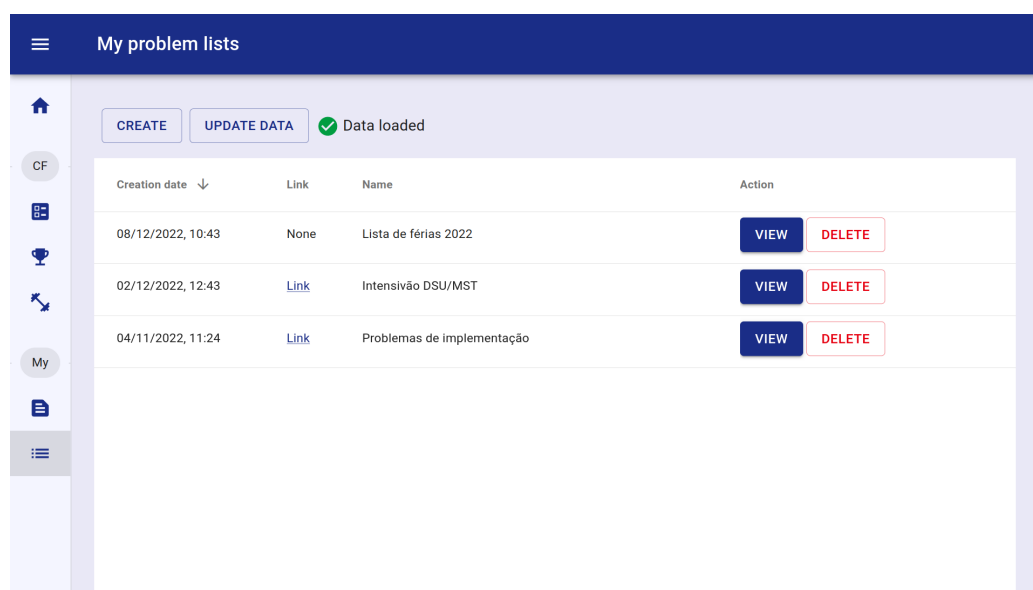
ou dificuldades enfrentadas que devem ser trabalhadas para um melhor desempenho futuro.

Os diálogos de adição e de edição de problemas são semelhantes. A diferença entre eles é que o de adição é inicialmente vazio, enquanto o de edição é preenchido com as informações do problema que se está editando.

## Listas de problemas

A página **Lista de problemas do usuário** (*My problem lists*, **Figura 6.8**) contém listas criadas pelo usuário. Os problemas contidos na página descrita na última seção podem ser adicionados a múltiplas listas.

No topo da página, encontra-se o botão **Criar** (*Create*) para a criação de uma nova lista. A seguir, está a tabela com as listas de problemas. Cada linha contém a **data de criação** da lista (*Creation date*), o **link**, o **nome** (*Name*), o botão **Visualizar** (*View*) para visualizar a página da lista correspondente, que será descrita adiante, e o botão **Deletar** (*Delete*) para remover a lista da aplicação. Esta última ação não remove da aplicação os problemas que estavam contidos na lista.



**Figura 6.8:** Lista de problemas do usuário (*My problem lists*).

A criação de uma lista traz um diálogo semelhante ao de edição dela, assim como acontece na página de problemas. Ele será descrito em seguida.

## Lista de problemas

Ao pressionar o botão de visualizar uma lista, o usuário é redirecionado para a página **Lista de problemas** (*Problem list*), com informações sobre ela. Caso tenha especificado um *link* para uma fonte externa que a representa, ele pode acessá-la ao clicar em seu nome. Abaixo, ele encontra a sua **descrição** (*Description*) e as **anotações** (*Notes*) que fez sobre o progresso da resolução dos problemas.

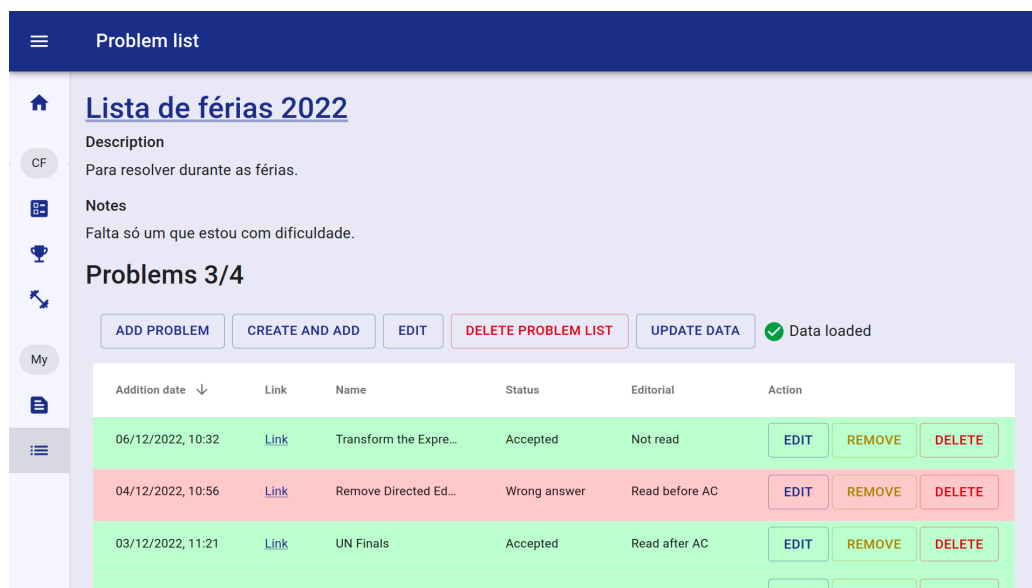


Figura 6.9: Página para uma lista de problemas (Problem list).

Logo abaixo, está a contagem dos problemas resolvidos e do total de problemas da lista. Ela inclui os problemas adicionados explicitamente na lista e os problemas que estão sendo contados implicitamente. A contagem implícita é alterada no formulário de edição da lista e é útil para quando o usuário não deseja adicionar os problemas na aplicação, pois consegue monitorar o progresso da resolução em outro lugar.

Em seguida, está localizada uma tabela com os problemas associados à lista, similar à encontrada na página de problemas. Existem quatro botões acima dela para a sua modificação.


O primeiro é o **Adicionar problema (Add problem)**, para adicionar um problema existente na aplicação à lista. Um diálogo é mostrado com uma lista de problemas que ainda não estão incluídos, como mostra a Figura 6.10.



Figura 6.10: Diálogo com a lista de problemas não contidos na lista em questão.

O segundo, **Criar e adicionar (Create and add)**, permite adicionar um novo problema à aplicação e, imediatamente, associá-lo à lista atual. Ele abre um diálogo idêntico ao da página de problemas.

O botão **Editar** (*Edit*) possibilita a edição das informações da lista por um diálogo que pode ser visto na Figura 6.11. Além de campos para a edição do nome, do *link*, da descrição e das anotações da lista, ele possui dois campos para a contagem dos problemas adicionados implicitamente a ela.



The image shows a dialog box titled "Edit problem list". It contains the following fields and values:

- Name: Lista de férias 2022
- Link: https://docs.google.com/spreadsheets/d/1UYEz\_PT-p6tsDUUWImWdUvj
- Description: Para resolver durante as férias.
- Notes: Falta só um que estou com dificuldade.
- Implicit solved: 0
- Implicit total: 0

At the bottom right, there are two buttons: "CANCEL" and "SAVE".

**Figura 6.11:** Diálogo para edição de informações de uma lista de problemas.

Por fim, o botão **Deletar lista de problemas** (*Delete problem list*) permite a remoção da lista. Os problemas associados persistem na plataforma.

A tabela com os problemas explicitamente associados à lista é semelhante àquela encontrada na página de problemas. Os problemas podem ser editados ou removidos da aplicação pelos botões **Editar** (*Edit*) e **Deletar** (*Delete*), respectivamente. Adicionalmente, o problema pode ser apenas desassociado da lista pelo botão **Remover** (*Remove*).

## 6.2 Fluxos de uso

Considerando as páginas implementadas, alguns fluxos de uso da aplicação podem ser executados pelo usuário para auxiliar o seu estudo em programação competitiva de diferentes maneiras.



### Resolução de problemas do *Problemset*

No caso do usuário desejar resolver problemas do *Codeforces*, independente da competição da qual fazem parte, ele pode ir para a página **Problemas do Codeforces**, selecionar questões pelos estados **Não tentado** e **Solução incorreta**, escolher um determinado intervalo de pontuação de dificuldade e resolver os problemas da lista resultante.

### *Upsolving* das competições internas

Se a prioridade for resolver problemas não resolvidos das competições internas (denominado por *upsolving*), o usuário pode ir para essa mesma página e selecionar os filtros mencionados acima, com a adição do filtro por estado da competição do problema. Ao selecionar somente o estado **Tentada**, ele verá apenas problemas de competições em que já tentou algum outro problema.

Essa estratégia de resolução também evita a leitura de problemas de competições cujo estado é **Limpa**, o que inviabilizaria a simulação virtual dela seguindo as regras da plataforma (MIKE MIRZAYANOV, 2011), que exige que o usuário não tenha visto nenhuma das questões antes de a simular.

### Encontrar competições internas para serem simuladas

A página de **Competições internas do Codeforces** facilita a busca de competições a serem simuladas. Com o filtro **Limpa** selecionado para o estado da competição, o usuário consegue ter a certeza que não enviou tentativas a nenhum dos problemas e então pode seguir para a plataforma para realizar a participação virtual.

### Encontrar competições da *Gym* para simular em equipe

Na página de **Competições da Gym**, podem ser selecionadas competições para o treino em equipe. Competições de regiões específicas da *ICPC* podem ser encontrados tanto pela busca pelo nome quanto pelo filtro de região. O filtro de dificuldade também pode ser bem importante para escolher a competição adequada às habilidades do time. Outro filtro útil é o de duração, pois o tempo dedicado ao treino em equipe pode ser limitado entre seus membros.

### Resolver problemas pendentes de qualquer lugar

Quando existem questões de outras fontes os quais o usuário gostaria de monitorar o progresso, ou até mesmo para as do *Codeforces*, ele pode registrá-las na página de **Problemas do usuário**.

Primeiramente, ele adiciona problemas na aplicação por meio do botão **Criar (Create)**. Após isso, ele pode registrar continuamente as mudanças de estado e fazer anotações nos exercícios em que trabalhar. Para os problemas que ele não conseguiu resolver na primeira tentativa, ele pode retornar depois de uns dias e tentar novamente. Tudo isso com os *links* para os enunciados agregados em um mesmo lugar.

### **Organizar listas de exercícios em um mesmo lugar**

Além da lista geral de problemas, a página **Lista de problemas do usuário**, permite agrupá-los em listas menores. Tais listas podem conter, inclusive, problemas definidos implicitamente. É o caso de uma lista já estar definida fora da plataforma e não for do interesse do usuário adicionar as informações dele manualmente.

Esse agrupamento também é um incentivo à resolução de problemas, visto que ele será instigado a completar as listas que adicionou.

# Capítulo 7

## Trabalhos futuros

Este trabalho teve como objetivo principal reunir múltiplas funcionalidades para auxiliar o treino em um mesmo lugar. Para isso criou-se uma arquitetura com os dados do usuário armazenados em um servidor central e uma interface para a visualização e modificação deles. O servidor também possui uma estrutura para a requisição e processamento de dados da *API* do *Codeforces* com um sistema de *cache* que faz com que eles estejam rapidamente disponíveis aos seus usuários. Essas características permitem que outras funcionalidades úteis ao treino possam ser implementadas no futuro com maior facilidade. Este capítulo descreve algumas delas.

### 7.1 Auto-avaliação em tópicos

O usuário poderia monitorar o seu desempenho em relação a determinados assuntos comumente encontrados em questões de programação competitiva. A aplicação poderia auxiliá-lo nesse processo ao disponibilizar uma seção para auto-avaliação e também ao permitir associar tópicos aos problemas.

No *Codeforces*, os problemas do *Problemset* já têm tópicos atribuídos. No entanto, para a *Gym*, a situação é diferente. Como os exercícios dessa parte da plataforma são mais próximos aos de provas oficiais da *ICPC* que aqueles das competições internas, seria importante ter a informação dos tópicos dos exercícios contidos nela para um estudo mais abrangente.

Dado que a aplicação desenvolvida já possui um banco de dados central, seria possível permitir que seus usuários atribuam tópicos aos problemas da *Gym* de forma coletiva. Seria como um sistema de *crowdsourcing* para que todos sejam beneficiados com a definição dos assuntos dos problemas, o que permitiria uma seleção mais específica deles.

### 7.2 Recomendação de problemas

Uma funcionalidade útil que visa incentivar a resolução eficiente de exercícios pelo usuário seria a recomendação de problemas. Com ela, o usuário poderia receber listas de problemas que devem contribuir com o seu aprendizado. Assim como as listas criadas por

ele, a recomendação em pacotes de questões incentivaria a sua conclusão, o que motiva a resolução dos exercícios contidos neles.

Para esse sistema de recomendação, poderiam ser usados múltiplas categorias de dados da *API* para garantir sua eficiência. É possível, por exemplo, acessar dados de milhares de usuários e analisar quais problemas eles estão fazendo. O desempenho desses usuários pode ser avaliado pela respectiva pontuação de desempenho e suas alterações nas últimas competições. Além disso, é possível comparar esses usuários para a definição de quais deles são mais próximos do usuário que está treinando na aplicação, para usá-los como referência para a recomendação.

Adicionalmente, se os problemas da *Gym* tiverem seus tópicos definidos, seria viável utilizá-los como critério para incrementar a sugestão de problemas. Desse modo, criaria-se uma priorização dos assuntos a receberem maior foco pelo usuário para garantir um repertório robusto e diverso, o que aumenta suas chances de resolver questões nas competições oficiais.

### 7.3 Funcionalidades de time

O banco de dados central permite a criação de seções da aplicação com dados compartilhados a usuários conectados de diferentes lugares simultaneamente, como é o caso para as informações do *Codeforces*, já implementadas, e para a atribuição de tópicos, a ideia mencionada na primeira seção deste capítulo. A sincronização constante dos dados do navegador com aqueles do servidor permite, além da leitura, a modificação por múltiplos usuários.

Essa característica possibilita a criação de seções da aplicação dedicadas ao treino em equipe. Algumas das funcionalidades para o estudo individual poderiam ser replicadas para essa outra modalidade, como é o caso do histórico de problemas. Poderia ser criado um histórico para as competições que um time participou em conjunto e, a partir dele, gerar uma lista de exercícios não resolvidos para a realização do *upsolving* deles coletivamente pelos membros da equipe.

Outra utilidade seria a seleção de competições da *Gym* que considera os problemas resolvidos por cada pessoa da equipe. Adicionalmente, poderia ter um histórico de competições nas quais outros times participaram, o que também pode ser usado como referência para essa seleção.

A auto-avaliação em tópicos de cada um dos integrantes do time poderia ser compartilhada para análise da coletânea de assuntos que a equipe domina e quais que devem ser mais estudados. Além disso, existe o potencial de desenvolvimento de um sistema de recomendação de problemas para os membros da equipe, que visaria preencher os buracos no conhecimento coletivo.

## Capítulo 8

### Conclusão

A aplicação desenvolvida cumpre com o seu objetivo inicial: providencia uma maneira para seus usuários organizarem o seu estudo para competições de programação. O seu uso permite o registro de problemas de qualquer fonte da *internet* para o monitoramento do progresso de resolução deles. Ela também permite registrar a leitura de editoriais ou de soluções de outros usuários, o que incentiva a realização dessa prática, como sugerido por variadas fontes descritas no Capítulo 4. Complementarmente, oferece uma maneira de manter problemas não resolvidos para resolução posterior, após o estudo de novos conteúdos e técnicas.

Além disso, a integração com as informações disponibilizadas pela *API* do *Codeforces* faz com que ela seja útil para a seleção de problemas e de competições da plataforma. A infraestrutura desenvolvida permite, inclusive, a adição futura de ainda mais funcionalidades que façam o uso dessas informações.

Com o sistema de *cache* implementado para as informações da *API*, o usuário pode ter acesso contínuo aos dados da plataforma com um impacto minimizado para o *Codeforces*. Assim, ele pode rapidamente tomar decisões sobre o seu treino e continuá-lo com agilidade.



# Referências

- [ALEX DANILYUK 2022] ALEX DANILYUK. *How to practice Competitive Programming [Um\_nik version]*. 2022. URL: <https://codeforces.com/blog/entry/98806> (acesso em 22/12/2022) (citado na pg. 14).
- [ANDREI MARGELOIU 2016] ANDREI MARGELOIU. *How to prepare for competitive programming?* 2016. URL: <https://medium.com/@andreimargeloiu/how-to-prepare-for-competitive-programming-396d557e0c12> (acesso em 20/12/2022) (citado na pg. 13).
- [COLIN GALEN 2021] COLIN GALEN. *Candidate Master in 1 Year - This Strategy Works Wonders*. 2021. URL: <https://www.youtube.com/watch?v=9M5voWYmie4> (acesso em 22/12/2022) (citado na pg. 15).
- [DR. MICHAEL JEFFRY DONAHOO 2020] DR. MICHAEL JEFFRY DONAHOO. “2020 brochure - icpc world finals moscow” (2020). URL: <https://icpc.global/community/history/2020Brochure.pdf> (acesso em 17/12/2022) (citado na pg. 4).
- [GEEKSFORGEEKS 2022] GEEKSFORGEEKS. *How to Get Started with Competitive Programming?* URL: <https://www.geeksforgeeks.org/how-to-get-started-with-competitive-programming> (acesso em 20/12/2022) (citado na pg. 13).
- [IVAN FEFER 2014] IVAN FEFER. *Codeforces API*. 2014. URL: <https://codeforces.com/blog/entry/12520> (acesso em 20/12/2022) (citado na pg. 10).
- [MASATAKA YONEDA 2019] MASATAKA YONEDA. *A Way to Practice Competitive Programming*. 2019. URL: <https://codeforces.com/blog/entry/66909> (acesso em 22/12/2022) (citado na pg. 14).
- [MIKE MIRZAYANOV 2010] MIKE MIRZAYANOV. *Codeforces Rating System*. 2010. URL: <https://codeforces.com/blog/entry/102> (acesso em 20/12/2022) (citado na pg. 8).
- [MIKE MIRZAYANOV 2015a] MIKE MIRZAYANOV. “Codeforces 5 years!” (2015). URL: <https://codeforces.com/5years> (acesso em 23/12/2022) (citado na pg. 7).
- [MIKE MIRZAYANOV 2015b] MIKE MIRZAYANOV. *New: Educational Rounds on Codeforces!* 2015. URL: <https://codeforces.com/blog/entry/21496> (acesso em 18/12/2022) (citado na pg. 9).

- [MIKE MIRZAYANOV 2011] MIKE MIRZAYANOV. *Codeforces: Virtual Contests are here!* 2011. URL: <https://codeforces.com/blog/entry/2607> (acesso em 17/12/2022) (citado na pg. 43).
- [OBI 2022] OBI. *Regulamento OBI*. URL: <https://olimpiada.ic.unicamp.br/info/regulamento> (acesso em 17/12/2022) (citado na pg. 4).