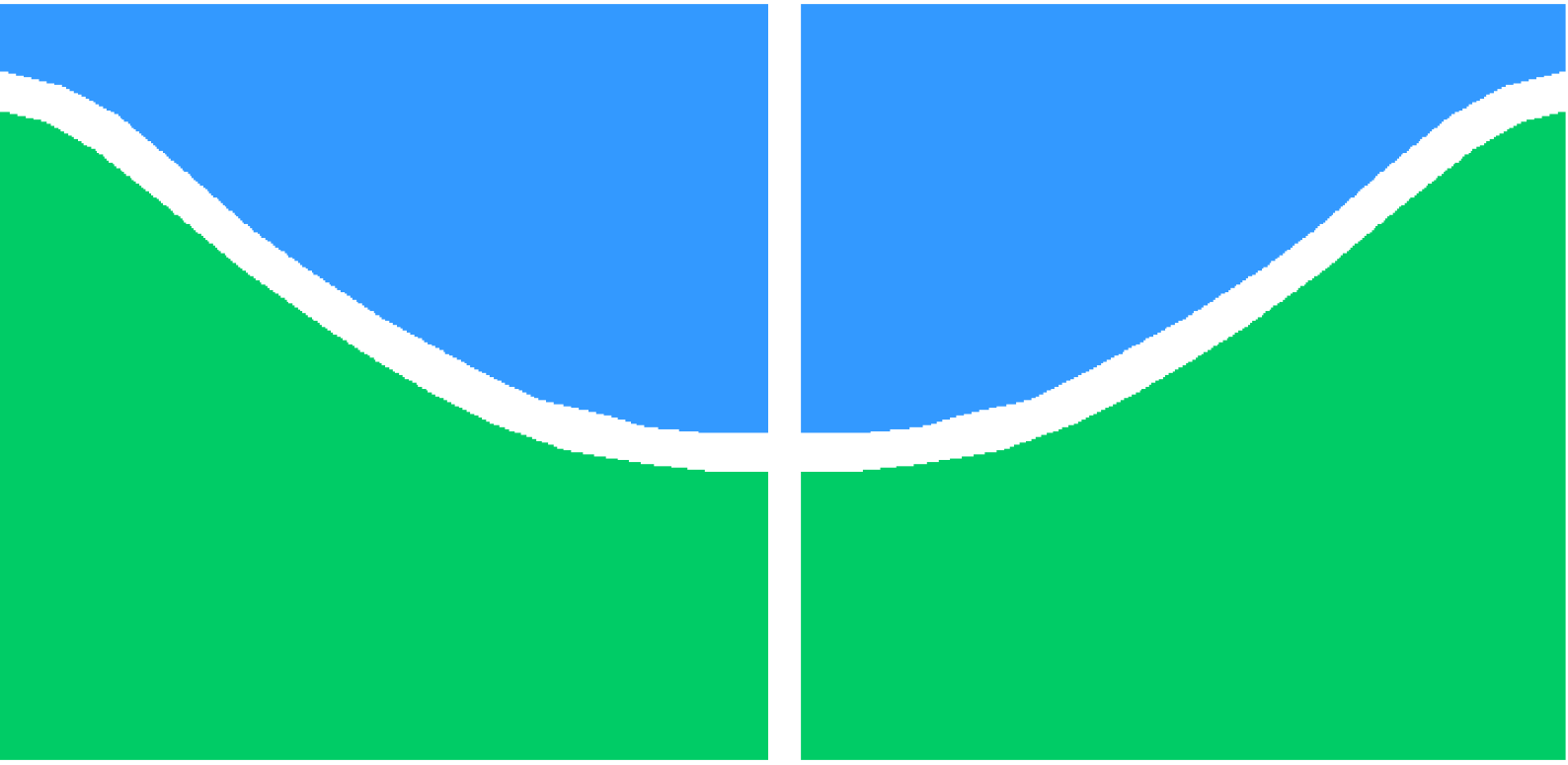


Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Gamma Online Judge: Desenvolvimento de uma plataforma para realização de contests baseados na Maratona UnB de Programação

Autor: João Pedro Silva de Carvalho

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2023

João Pedro Silva de Carvalho

Gamma Online Judge: Desenvolvimento de uma plataforma para realização de contests baseados na Maratona UnB de Programação

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

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2023

João Pedro Silva de Carvalho

Gamma Online Judge: Desenvolvimento de uma plataforma para realização de contests baseados na Maratona UnB de Programação

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

Brasília, DF
2023

Agradecimentos

Agradeço primeiramente a Deus por me dar força e motivação; agradeço aos meus pais, Regina e Manoel, por me apoiarem nesta caminhada e agradeço ao professor e treinador Edson pelo apoio neste trabalho e na Maratona de Programação.

"Deus dá as batalhas mais difíceis aos seus melhores soldados."
(Papa Francisco)

Resumo

A Maratona UnB de Programação é uma competição que ocorre anualmente há 10 anos no período da Semana Universitária UnB e por isso acumula mais de uma centena de problemas que podem ser utilizados pelos maratonistas e estudantes em sua preparação. Nesse contexto, inserido na Universidade de Brasília (UnB), o Gamma Online Judge é uma plataforma, que já possui uma parte desenvolvida, cujo objetivo é armazenar os problemas da Maratona UnB de Programação e permitir que os usuários estudem e pratiquem com base neles. O estágio de desenvolvimento da aplicação não está adequado para implantar em ambiente de produção. A realização de competições virtuais e autenticação dos usuários são funcionalidades vistas como necessária para a criação de uma primeira versão do projeto em produção. Com base em um método de ciclo de desenvolvimento baseado no Scrumban, este trabalho tem por objetivo desenvolver essas funcionalidades além de listar refinamentos para a evolução da aplicação.

Palavras-chave: Juiz Online, Maratona de Programação, Gamma Online Judge.

Abstract

The Maratona UnB de Programação is a competition that has been held annually for 10 years during the Semana Universitária UnB, and therefore accumulates more than a hundred problems that can be used by contestants and students in their preparation. In this context, inserted in the University of Brasília (UnB), Gamma Online Judge is a platform, which already has a developed part, whose objective is to store the problems of the Maratona UnB de Programação and allow users to study and practice based on them. The stage of development of the application, it is not suitable for deploying in a production environment. Conducting virtual competitions and user authentication are functionalities seen as necessary for creating a first version of the project in production. Based on a development cycle method based on Scrumban, the aim of this paper is to develop these functionalities in addition to listing refinements for the evolution of the application."

Key-words: Online Judges, Programming Contests, Gamma Online Judge

Lista de ilustrações

Figura 1 – Arquitetura Cliente Servidor 18
Figura 2 – Arquitetura geral Gamma Online Judge 19
Figura 3 – Arquitetura Gamma Judge API 21
Figura 4 – Arquitetura Gamma Judge Tools 22
Figura 5 – Exemplo de um quadro de tarefas do Kanban 24

Figura 6 – Quadro de tarefas 30
Figura 7 – Quadro de tarefas durante desenvolvimento 31
Figura 8 – Modelagem dos dados 32
Figura 9 – Notas ao final do texto de um problema 33
Figura 10 – Tela de Login 36
Figura 11 – Tela de Cadastro 37

Lista de tabelas

Tabela 1 – Lista de requisitos obtida 29
Tabela 2 – Lista de histórias de usuário 30

Lista de abreviaturas e siglas

API	*Application Programming Interface*
ABNT	Associação Brasileira de Normas e Técnicas
AWS	*Amazon Web Service*
GOJ	*Gamma Online Judge*
ICPC	*International College Programming Contest*
IOI	*International Olympiad of Informatics*
MVC	*Model View Controller*
OAS	*OpenAPI Specification*
OJ	*Online Judge*
PO	*Product Owner*
POO	Programação Orientada ao Objeto
S3	*Simple Storage Service*
SBC	Sociedade Brasileira de Programação
SQS	*Simple Queue Service*
SNS	*Simple Notification Service*
UI	*User Interface*
UnB	Universidade de Brasília

Sumário

1 INTRODUÇÃO 12
1.1 Contexto 12
1.2 Problema 14
1.3 Objetivos 14
1.4 Organização do Trabalho 15

2 REFERENCIAL TEÓRICO 16
2.1 Conceitos 16
2.1.1 *Frameworks* e Bibliotecas 16
2.1.2 Orientação ao componente 17
2.1.3 Arquitetura cliente-servidor e aplicações WEB 17
2.1.4 Swagger 17
2.2 Arquitetura legada 18
2.2.1 Arquitetura geral 18
2.2.2 Infraestrutura AWS 19
2.2.3 MongoDB 20
2.2.4 Gamma Judge API 20
2.2.5 Gamma Judge Tools 21
2.2.6 Gamma Judge UI e Gamma Judge Admin 22
2.3 Requisitos 22
2.4 Scrumban 23

3 MATERIAIS E MÉTODOS 25
3.1 Requisitos 25
3.2 Planejamento 25
3.3 Manutenção e evolução 26
3.3.1 Paradigmas de programação 27
3.3.2 Banco de dados 27
3.3.3 Ambiente de desenvolvimento 28

4 RESULTADOS 29
4.1 Requisitos 29
4.2 Aplicação 31
4.2.1 Modelagem 31
4.2.2 US01 - Campo de notas ao final do texto 32
4.2.3 US03 - Importação de problemas e de *constests* 32

4.2.4 US04 - Autenticação 35
4.2.5 Melhoria – Suporte a *tags* HTML 36
4.2.6 Melhoria - Armazenamento de parte das questões em Base64 37
4.2.7 Melhoria – Adição do suporte para múltiplas línguas na formatação dos problemas 38

5 CONSIDERAÇÕES FINAIS 39

REFERÊNCIAS 40

1 Introdução

1.1 Contexto

Online judges (OJ) são aplicações projetadas para coletar o código-fonte, compilar e executar o binário resultante a fim de avaliar se a solução gera os resultados corretos de acordo com um conjunto de casos de testes predefinido; por fim, um OJ computa o resultado da submissão (WASIK et al., 2016).

Diferentemente dos juízes eletrônicos, os *online judges* são aplicações que além de executar o processo de julgamento das soluções submetidas, disponibilizam os problemas publicamente na *internet* para que os usuários possam resolvê-los, sendo necessário apenas o registro no respectivo OJ (REVILLA; MANZOOR; LIU, 2008). Em síntese, o juiz eletrônico é uma aplicação focada apenas no julgamento dos problemas enquanto o *online judge* é uma aplicação, disponível na *internet*, mais robusta, possuindo funcionalidades como, por exemplo, um banco de problemas, registro de usuários e submissões e entre outras.

Juízes eletrônicos são amplamente utilizadas no ensino de programação. Segundo Galvão, Fernandes e Gadelha (2016) seu uso melhora consideravelmente o desempenho dos alunos no aprendizado de programação, seguido por Ribeiro et al. (2018), que acrescenta elementos de gamificação mais presentes nos *online judges*.

Tanto os *online judges* quanto os juízes eletrônicos permitiram a existência de competições em que o participante submete o código, recebendo a correção instantes depois junto a uma penalidade a depender do tempo e número de submissões necessárias para a solução correta. Os problemas utilizados são resolvidos por algoritmos previamente conhecidos ou por adaptação deles (HALIM; HALIM, 2013).

O International College Programming Contest (ICPC) é uma competição de programação de algoritmos que começou sua história em 1970 e é atualmente considerada a maior competição na área (ICPC , 2023). Já a International Olympiads of Informatics (IOI) com sua primeira edição ocorrendo em 1989 na Bulgária, é uma das 5 olimpíadas de ciências internacionais, com foco na ciência da computação (IOI, 2023). A principal diferença entre as duas principais é que o IOI foca em alunos nos anos finais do ensino médio e anos inicias do superior, enquanto o ICPC foca em alunos de nível superior.

No âmbito brasileiro, a Sociedade Brasileira de Computação (SBC) organiza a Maratona de Programação que existe desde 1996. O evento faz parte das competições classificatórias do ICPC e está incluída no regional sul-americano do evento (SBC, 2022).

Na Maratona de Programação da SBC, os competidores têm 5 horas para resolver os problemas recebendo para cada submissão incorreta, em qualquer um dos casos de testes secretos, uma penalidade de tempo quando enviar uma submissão correta. Vence quem resolver mais problemas, sendo o critério de desempate a soma acumulada do tempo e das penalidades de cada problema (SBC, 2022). Nessa competição é utilizado o sistema de entrega de exercícios BOCA responsável pela autenticação dos usuários, controle do tempo e processamento dos resultados (CAMPOS; FERREIRA, 2004).

Uma equipe de professores, estudantes, monitores e competidores do Distrito Federal promovem anualmente diversos eventos voltados à maratona de programação. Somente no ano de 2019, cerca de 700 participantes divididos em 315 equipes participaram dos 6 eventos que ocorreram naquele ano (SAAD, 2019). Estes dados ilustram o interesse da comunidade de estudantes de programação nessas competições.

Dentre os eventos promovidos no Distrito Federal cabe destacar a Maratona UnB de Programação que acontece anualmente desde a sua primeira edição em 2013 (RAMOS, 2021). Esse evento faz parte da Semana Universitária da UnB e conta com a participação de estudantes de outras instituições, majoritariamente do Distrito Federal, como o IFB, UniCEUB e o IESB.

A ideia de desenvolver o Gamma Online Judge nasce nesse ambiente de maratona de programação do Distrito Federal. As principais plataformas tanto para estudo quando armazenamento das questões não é adequado visto as necessidades, como o idioma, formatação e indexação de problemas por tema de estudo (LIMA; JUNIOR, 2021).

O Gamma Online Judge foi pensado como uma aplicação para ser uma ferramenta de suporte para o livro que, até a conclusão deste Trabalho, está sendo escrito pelo professor Dr. Edson Alves da Costa Júnior sobre a Maratona UnB de Programação.

Nesse contexto, Lima e Junior (2021) desenvolveram a aplicação visando armazenar e disponibilizar os problemas da Maratona UnB de programação para resolução por meio de submissão de código-fonte com o seguinte escopo.

1. armazenamento das questões passadas da Maratona UnB de Programação;
2. fornecer suporte para a resolução de questões por programadores iniciantes, como em tutoriais e dicas;
3. fazer a correção dos códigos submetidos;
4. agrupamento das questões por eventos.

1.2 Problema

O escopo inicialmente idealizado pelo professor Edson Alves da Costa Júnior envolvia uma série de funcionalidades e ferramentas que ofereciam suporte a quem deseja estudar os problemas da Maratona UnB de Programação. Em seu trabalho, Lima e Junior (2021) desenvolveram parte desse escopo.

Esta versão já desenvolvida do Gamma Online Judge possuía 4 serviços sendo eles: Gamma Judge UI, Gamma Judge Admin, Gamma Judge Tools e Gamma Judge API. Neste estado da aplicação, foram desenvolvidas funcionalidades que permitiam um usuário listar os *contests* e problemas e enviar a sua submissão, todavia não havia suporte para o *login* e registro dos usuários na plataforma.

Na parte de administração do sistema, encabeçado pelo Gamma Judge Admin, foi desenvolvido as funcionalidades de cadastro de *contests* e problemas por meio do preenchimento de formulários. Existia um suporte básico a instruções em Latex na formatação dos problemas, mas que possuíam algumas limitações no texto principalmente pela necessidade de adição de caracteres especiais, como barras e chaves, dificultando seu transporte e armazenamento em estruturas de chave-valor como o JSON e o Banco de dados MongoDB presentes no projeto.

Tendo isso em vista, o problema desse trabalho se baseia na manutenção e evolução do Gamma Online Judge visando desenvolver funcionalidades como a de autenticação e a realização de *contests* virtuais para poder ser gerada uma primeira versão apta para ser implantada em ambiente de produção.

1.3 Objetivos

Este trabalho tem por objetivo desenvolver as funcionalidades que solucionem os problemas abordados na Seção 1.2 e que são necessárias para implantar o Gamma Online Judge em produção.

Os objetivos específicos deste trabalho são:

- desenvolver a autenticação e autorização da aplicação.
- desenvolver a funcionalidade de *contest* virtuais.
- listar e desenvolver refinamentos na aplicação.

1.4 Organização do Trabalho

Este trabalho está organizado em 5 capítulos: na Introdução, foi apresentado brevemente todo o contexto que envolve este trabalho abordando também o problema e os objetivos; Referencial Teórico, onde são apresentados os principais fundamentos teóricos para o desenvolvimento da aplicação; Materiais e Métodos, onde são apresentados os métodos de desenvolvimento, as técnicas utilizadas, no planejamento e na organização; já o capítulo de Resultados apresenta os resultados obtidos no trabalho; por fim o capítulo de Considerações Finais conclui este trabalho abordando as últimas considerações acerca do que foi desenvolvido e possibilidades de trabalhos futuros.

2 Referencial Teórico

Neste capítulo são abordadas as principais bases teóricas usadas para o desenvolvimento do Gamma Online Judge e está organizado nas seções: Conceitos onde alguns conceitos utilizados neste trabalho são abordados; Arquitetura legada que detalha a arquitetura legada do Gamma Online Judge; já a Seção Requisitos fundamenta alguns termos sobre a gerência de requisitos; por fim a Seção Scrumban explica acerca do ciclo de desenvolvimento Scrumban.

2.1 Conceitos

Esta Seção irá abordar os principais termos e conceitos utilizados no desenvolvimento do Gamma Online Judge.

2.1.1 *Frameworks* e Bibliotecas

Os dois conceitos, *Frameworks* e bibliotecas, são semelhantes entre si, pois ambos oferecem um conjunto de funcionalidades que auxiliam o desenvolvimento. Em ambos os casos o programador os utiliza visando a aumentar a reutilização de código (WOZNIEWICZ, 2019).

O uso de ambas não são excludentes e em certos momentos até se completam. As bibliotecas, próprias ou de terceiros, oferecem a vantagem de serem mais flexíveis dando maior liberdade ao programador, mas demandam mais cuidado para manter o código em uma estrutura definida. Por outro lado, os *frameworks* não possuem esse problema, pois obriga ao programador a seguir a estrutura própria dele, o que pode ser um fator limitante para desenvolvimento de códigos mais complexos (WOZNIEWICZ, 2019).

Usando o próprio Gamma Online Judge como exemplo, explicado mais a fundo na Seção 2.2, o Gamma Judge API utiliza o *framework* .NET que possui uma estrutura defina de arquivos com um diretório para cada elemento da arquitetura *Model View Controller* (MVC) além de alguns arquivos padrões de configuração como o `Program.cs` e o `Startup.cs`.

As aplicações *frontend*, o Gamma Judge UI e o Gamma Judge Admin, utilizam a biblioteca do React que oferece uma série de funcionalidades, dentre elas, o uso de componentes, mas que não restringe a estrutura do código.

2.1.2 Orientação ao componente

O paradigma de orientação ao componente é voltado para a construção de componentes, que junto a outros se integram e formam a interface. Cada componente é um pedaço de código que representa um elemento visual e sua construção permite que seja renderizado como sua instância, eximindo a necessidade de codificar o mesmo elemento, favorecendo a reutilização de código (PRITCHARD, 2019).

A orientação ao componente possui várias semelhanças com a orientação ao objeto, principalmente ao facilitar a reutilização de código, entretanto a orientação a componente é focada em elementos visuais de aplicações *frontend* enquanto a orientação ao objeto busca acoplar dados e procedimentos em elementos básicos chamados de objetos (VINCENZI, 2004; PRITCHARD, 2019).

2.1.3 Arquitetura cliente-servidor e aplicações WEB

Segundo Volle (2022), podemos definir qualquer serviço oferecido na internet como uma forma de aplicação WEB. Sua principal vantagem é a possibilidade do usuário executar a aplicação em diversos ambientes computacionais, visto que o funcionamento depende do navegador e não do seu sistema operacional (VOLLE, 2022).

A principal arquitetura utilizada no desenvolvimento de aplicações WEB é a de cliente-servidor. Essa arquitetura é dividida em duas partes, o servidor que implementa um serviço específico podendo ser, por exemplo, um banco de dados, e o cliente que requisita um serviço ao servidor por meio de uma requisição e este retorna uma resposta de acordo com o serviço solicitado. Essa arquitetura necessita de uma rede que conecta as partes, podendo estar ou não conectadas à internet (TANENBAUM; STEEN, 2008).

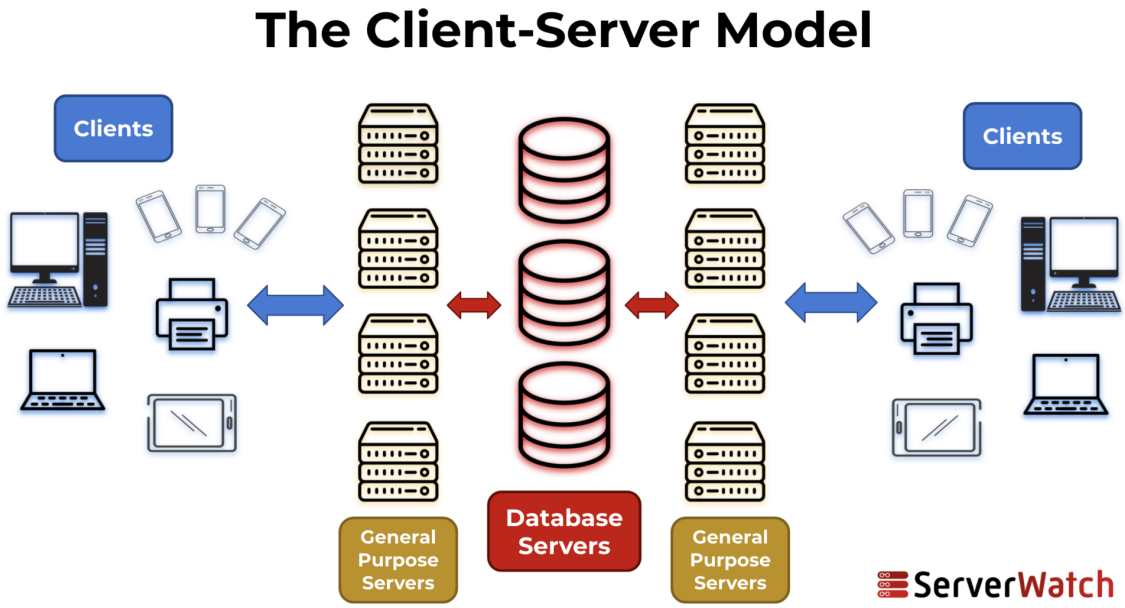
A Figura 1 exemplifica a arquitetura cliente-servidor de uma maneira geral, onde é possível ver os clientes (*clients*) se comunicando com os servidores de proposito geral (*general surpose servers*) processando as informações requisitadas dos servidores de banco de dados (*database servers*).

2.1.4 Swagger

O Swagger é um conjunto de programas que são ferramentas que auxiliam no desenvolvimento de APIs em todo o seu ciclo de vida. Desenvolvido pela SmartBear Software, o Swagger possui ferramentas de código aberto, grátis e pagas (Swagger, 2023).

O OpenAPI Specification (OAS), é um formato de descrição de APIs para o padrão. Essa especificação descrevem características como as rotas, operações, parâmetros de entrada e saída e os métodos de autenticação. Pode ser escrito no formato YAML ou JSON (Swagger, 2023).

Figura 1 – Arquitetura Cliente Servidor



Fonte: Ingalls (2021)

O conjunto de ferramentas do Swagger utiliza do OpenAPI Specification para projetar, desenvolver, documentar e consumir APIs. Uma das ferramentas mais usadas é o SwaggerUI que, a partir do OAS gerado, de forma manual ou automática, gera a documentação das rotas criadas e permite que haja uma interação com elas (Swagger, 2023).

2.2 Arquitetura legada

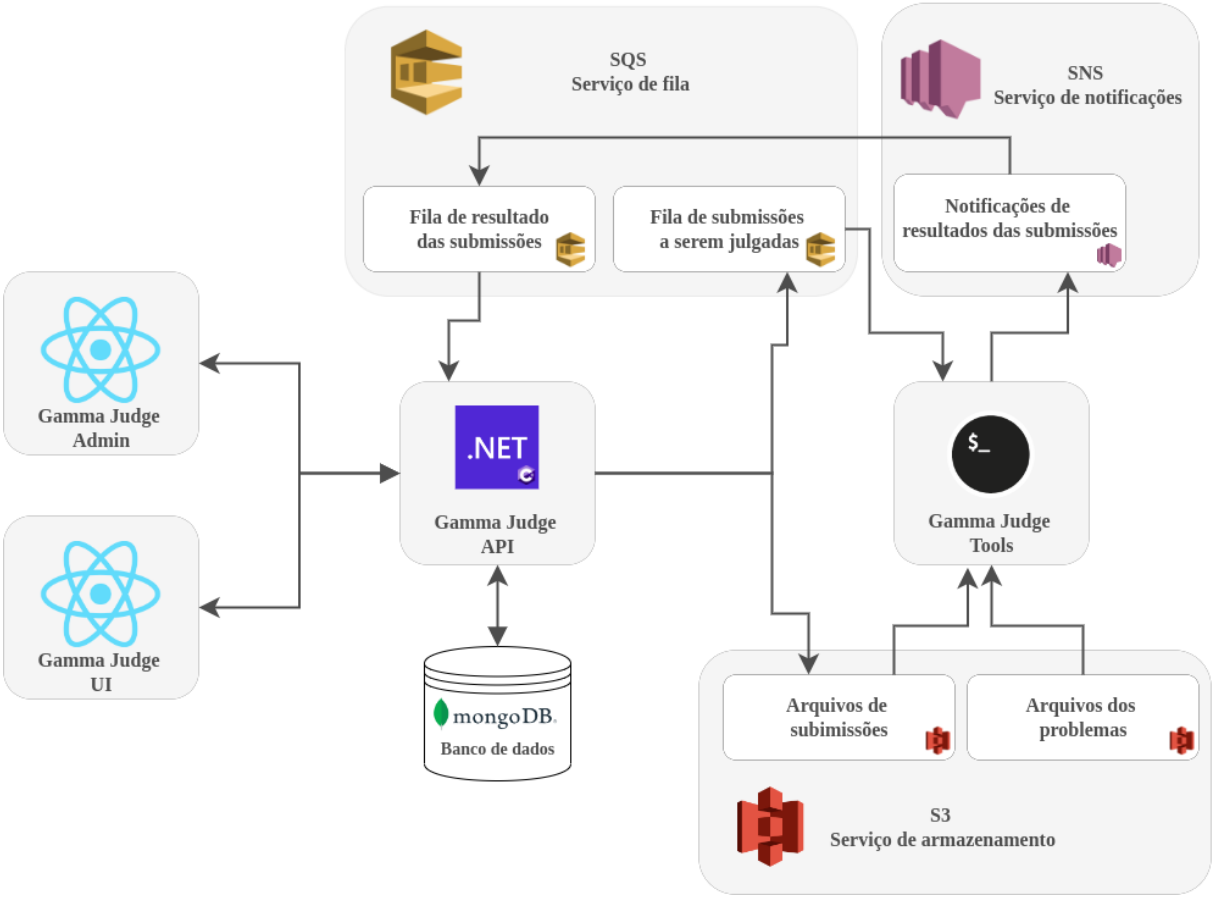
Esta Seção irá explicar conceitos necessários para o entendimento da aplicação além de mostrar como a arquitetura do projeto foi montada. A Seção está dividida nas seguintes Subseções: a Arquitetura geral introduz uma visão geral da arquitetura geral do projeto; MongoDB explica detalhes sobre a tecnologia utilizada para o principal componente de armazenamento de dados; Gamma Judge API, Gamma Judge Tools e Gamma Judge UI e Gamma Judge Admin aborda com mais detalhes cada aplicação desenvolvida e por fim a Infraestrutura AWS lista os serviços de nuvem da Amazon e detalha como estão inseridos na arquitetura.

2.2.1 Arquitetura geral

A Figura 2 sintetiza a arquitetura geral do Gamma Online Judge e como cada componente se relaciona entre si. O Gamma Judge UI e Gamma Judge Admin correspondem a interface do usuário com o sistema que utiliza o Gamma Judge API como servidor

dos dados e serviços. A Infraestrutura AWS é responsável por armazenar os arquivos de submissões e arquivos dos casos de teste e integrar junto ao Gamma Judge Tools que atua julgando as submissões e retornando o resultado.

Figura 2 – Arquitetura geral Gamma Online Judge



Fonte: Lima e Junior (2021)

2.2.2 Infraestrutura AWS

A infraestrutura do Amazon Web Services (AWS) presente na aplicação do GOJ é composta de 3 serviços: o Simple Notification Service (SNS), o Simple Queue Service (SQS) e o Simple Storage Service (S3).

O SNS é um sistema de notificação que envia mensagens para as pessoas e serviços cadastrados em seus tópicos (AMAZON, 2023b). Dentro da arquitetura do GOJ, quando acionado pelo Gamma Judge Tools, ele atua enviando a mensagem quando há alguma mudança no estado corrente do julgamento do da submissão.

O SQS é um serviço que simula o funcionamento de uma fila, porém em alguns casos o primeiro elemento a entrar não é o primeiro a sair (AMAZON, 2023c). Para que o julgamento possa ser uma atividade assíncrona com o resto da aplicação, o SQS

armazena uma fila de submissões guardando em cada elemento, dados de cada submissão como o identificador do problema, a referência do código-fonte no S3 e a linguagem de programação utilizada.

O S3 é um serviço de armazenamento voltado para objetos (AMAZON, 2023a). Nesse serviço são armazenados os códigos-fonte das submissões e os casos de teste secretos utilizado no julgamento.

2.2.3 MongoDB

No MongoDB, diferentemente dos bancos de dados relacionais, os dados são organizados por meio de documentos que os armazenam por meio de estruturas de chave-valor e, dentre as principais vantagens, está a representação de complexas hierarquias por meio de um único registro (BRADSHAW; BRAZIL; CHODOROW, 2020).

Uma das maiores diferenças entre os bancos de dados relacionais e o MongoDB é que este não possui tipagem de dado, dando mais liberdade para o desenvolvedor inclusive para adicionar e remover campos (BRADSHAW; BRAZIL; CHODOROW, 2020).

Segundo Bradshaw, Brazil e Chodorow (2020), os principais conceitos relativos ao MongoDB são:

- **Documento:** é representação básica de dado, semelhante à linha em bancos relacionais.
- **Coleção:** é o equivalente das tabelas dos brancos relacionais.
- **Base de dados:** uma instância de MongoDB consegue hospedar uma ou mais bases de dados, cada uma com as suas próprias coleções.

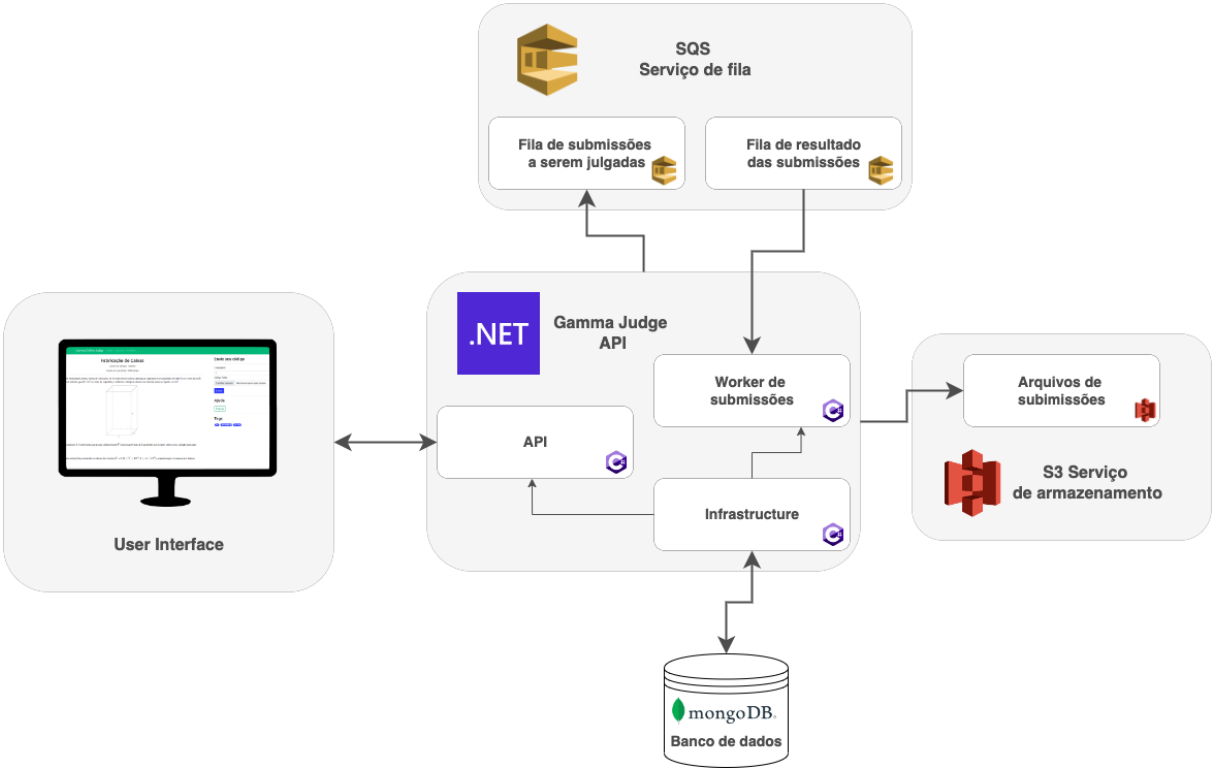
2.2.4 Gamma Judge API

O Gamma Judge API tem a função de ser o servidor para as aplicações *frontend* do GOJ. Sua principal função é fazer a disponibilização e manipulação das informações presentes no banco de dados (LIMA; JUNIOR, 2021). Utiliza a linguagem de programação C# na versão 10 junto ao *framework* .NET 6.0.

A Figura 3 demonstra graficamente como a arquitetura está estabelecida. O Gamma Judge API é dividido em 3 partes principais: a API, onde estão disponíveis as rotas usadas pelas aplicações *frontend* para a comunicação; o *Worker* de submissão, o qual consiste em um serviço que processa os resultados das submissões recebidas; por fim, a *Infrastructure*, que faz um mapeamento dos dados utilizados para a sua estrutura de armazenamento no banco de dados (veja Seção 2.2.3).

A Figura 3 mostra também os serviços externos ao Gamma Judge API, usados no seu funcionamento, dentre os quais estão: a interface de usuário (*user interface*), os serviços em nuvem da AWS SQS e S3 (veja Subseção 2.2.2) e o banco de dados *MongoDB* (veja Subseção 2.2.3).

Figura 3 – Arquitetura Gamma Judge API



Fonte: Lima e Junior (2021)

2.2.5 Gamma Judge Tools

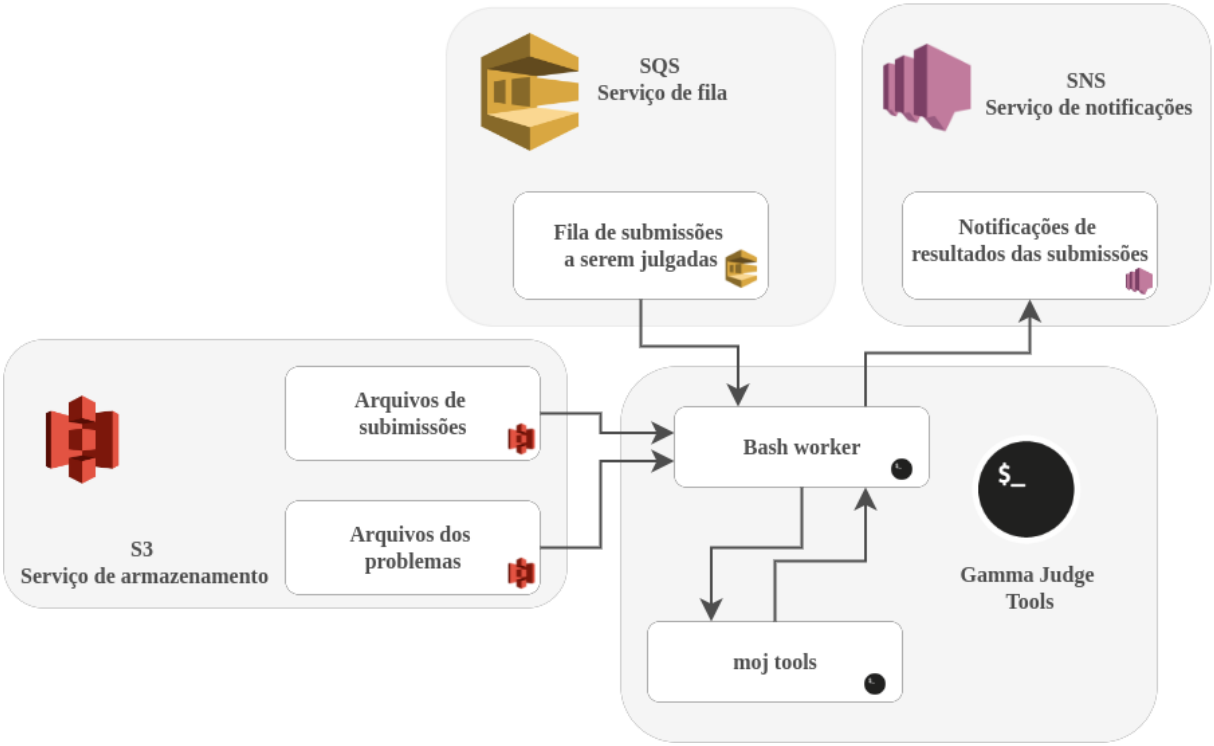
O Gamma Judge Tools, interagindo com os serviços da AWS, é responsável por processar as submissões, jugar e retornar o resultado. O serviço é escrito na linguagem de programação Bash e possui duas partes: o Bash Worker é responsável pelo monitoramento das filas SQS (veja Subseção 2.2.2) de submissões para o recebimento e envio das submissões; já o MOJ Tools é o serviço responsável pelo julgamento das submissões enviadas (LIMA; JUNIOR, 2021).

O MOJ Tools[1], de autoria do professor Dr. Bruno César Ribas, é um conjunto de ferramentas que calibram o tempo limite das questões além de executar o julgamento das questões. Dentro da arquitetura do Gamma Judge Tools, o MOJ Tools recebe o arquivo de submissão, compila se necessário, executa com os arquivo de entrada e compara a saída do programa com o resultado esperado, armazenado nos arquivos de saída. Os arquivos de entrada e saída são definidos por cada caso de teste.

[1] Código disponível em: <https://github.com/cd-moj/mojtools>

A Figura 4 mostra como se comporta a arquitetura do Gamma Judge Tools. O serviço observa a fila de submissões recebendo as informações do problema e um link para o arquivo de código-fonte presente no S3. As submissões são julgados pelo Moj Tools e os resultados enviados para o Bash Worker, que o retorna para o Gamma Judge API utilizando o SNS (veja Subseção 2.2.2).

Figura 4 – Arquitetura Gamma Judge Tools



Fonte: Lima e Junior (2021)

2.2.6 Gamma Judge UI e Gamma Judge Admin

O Gamma Judge UI e o Gamma Judge Admin são interfaces de usuário e utilizam o Gamma Judge API para o envio e recebimento de informações, porém, possuem objetivos distintos dentro da aplicação. O primeiro é a interface para o usuário final da aplicação, possibilitando a busca e solução de *contests* e problemas; já o segundo é onde o cadastro dos problemas e *contests* são feitos (LIMA; JUNIOR, 2021). Ambas interfaces utilizam como linguagem de programação principal o *TypeScript* versão 4.1.2 com o *NodeJS* na versão 16, além de algumas bibliotecas externas como o React na versão 17.

2.3 Requisitos

A Engenharia de Software engloba várias disciplinas que são necessárias para o desenvolvimento de um software. Segundo Vazquez e Simões (2016), a Engenharia de

Requisitos é uma dessas disciplinas que abordam como realizar a elicitação, documentação e gerenciamento dos requisitos do projeto além de garantir a sua qualidade.

A importância dos requisitos de software é endossado pela ABNT (2015) quando afirma que qualidade pode ser entendida como a capacidade do produto, nesse caso um software, de atender os requisitos. Sendo assim, a ABNT define requisito como: "Necessidade ou expectativa que é declarada, geralmente implícita ou obrigatória"(ABNT, 2015).

A atividade de elicitação dentro da Engenharia de Requisitos tem papel importante visto que é a etapa onde os requisitos são levantados. Nesse contexto, Mendonça (2014) listou técnicas para a elicitação, sendo elas: entrevistas, onde o entrevistador busca os requisitos por meio da exposição de ideias no entrevistado, podendo ter perguntas predefinidas; questionário, que se assemelha às entrevistas por se basear em perguntas, mas difere no ponto de abrangência, já que permite que seja feita com uma quantidade maior de pessoas; *joint application design* (JAD) é uma metodologia desenvolvida pela IBM e está embasa em dinâmicas em grupo junto a reuniões com estruturação e organização bem definidas; em prototipagem, é gerada uma versão inicial do produto utilizada para validação das ideias; por fim a técnica de *brainstorm* pode ser dividida em duas partes, na primeira, os envolvidos listam todas as ideias, e na segunda as ideias são refinadas e transformadas em requisitos.

2.4 Scrumban

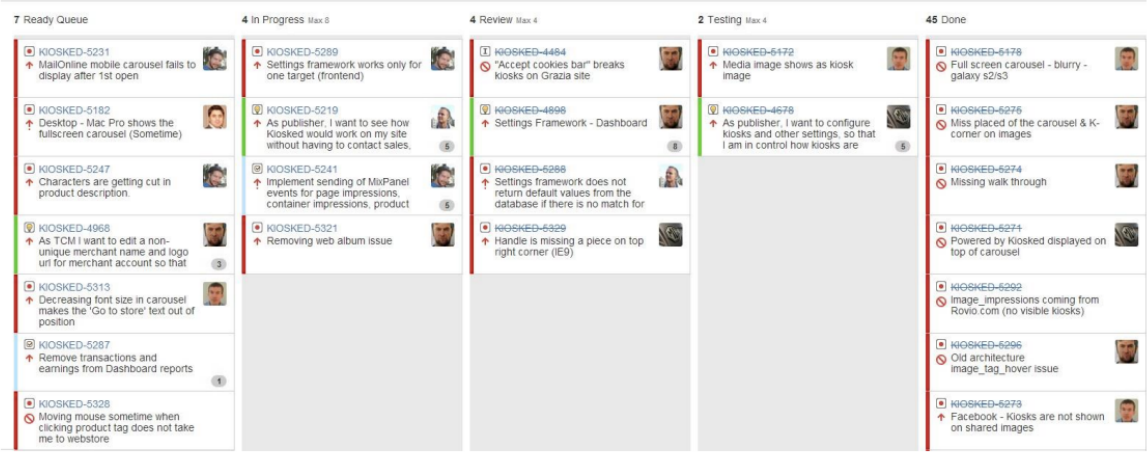
O Scrum é uma metodologia que preza pela agilidade, enquanto o Kanban preza pela melhoria contínua dos seus processos. O termo Scrumban nasce da junção dos dois termos, visando juntar as características de ambos em um único processo (KHAN, 2014).

A visualização do fluxo de trabalho, comumente visto na forma de quadro de tarefas, é uma importante ferramenta importada do Kanban. Essa ferramenta visa a visualização de todo o fluxo de trabalho desde o *backlog* do produto até a fase de conclusão. Por meio de quadros, digitais ou físicos, as tarefas são organizadas em estágios representados pelas colunas (LADAS, 2008). A Figura 5 mostra um exemplo de quadro de tarefas do Kanban separando-as em colunas.

Nas reuniões de planejamento, diferentemente do Scrum, o Scrumban define que elas devem ser os mais breves possíveis visando atualizar as tarefas a serem desenvolvidas. Já as reuniões de revisão é um momento em que o time tem a opinião das pessoas envolvidas no projeto, revisando o que foi desenvolvido no ciclo de desenvolvimento que está se encerrando (KHAN, 2014).

Esses ciclos são chamados de *sprints*, normalmente tendo duração de 4 semanas, podendo ser menos a depender da equipe. As *sprints* começam com a reunião de plane-

Figura 5 – Exemplo de um quadro de tarefas do Kanban



Fonte: Khan (2014)

jamento onde realiza a definição do escopo de tarefas e finaliza com a reunião de revisão (KHAN, 2014; LADAS, 2008).

Os papeis no Scrumban são importados do Scrum sendo eles: *scrum master* (SM), com a função de assegurar os valores e práticas do Scrum; o *product owner* (PO) que é, na tradução direta do termo, o dono do produto e fonte dos requisitos do *backlog*; e por fim o time de desenvolvimento é responsável pelo desenvolvimento do software (KHAN, 2014).

Originalmente derivada do Xtreming Programming (XP) (BECK; ANDRES, 2004), as histórias de usuário foram adotadas pelo Scrum, e consequentemente pelo Scrumban, sendo definidas como uma descrição breve do que o sistema deve fazer para o usuário. Segundo Vazquez e Simões (2016), a especificação da solução por parte do PO e comportamento do sistema não entram na escrita das histórias, usando os critérios de aceitação para realizar esse detalhamento.

3 Materiais e métodos

Com base no problema e dos objetivos apresentado no Capítulo 1, este capítulo irá discutir os materiais e métodos que foram utilizados no desenvolvimento do Gamma Online Judge.

O início do processo de desenvolvimento deste trabalho foi realizado sem uma metodologia claramente definida, o que ocasionou problemas no entendimento e na validação da funcionalidade de autenticação de usuários, a primeira a ser desenvolvida. O escopo da funcionalidade mudou algumas vezes, demonstrando a necessidade da aplicação de uma metodologia, não somente para o gerenciamento dos requisitos, mas de todo o ciclo de desenvolvimento.

3.1 Requisitos

Para um melhor mapeamento das funcionalidades e melhorias a serem implementadas, foi definida uma metodologia para o gerenciamento dos requisitos. Conforme as técnicas de elicitação de requisitos abordadas na Seção 2.3, este trabalho utilizou a técnica de *brainstorm*. Por meio dela foram listadas todas as ideias de funcionalidades e melhorias que seriam necessárias para cumprir o objetivo deste trabalho.

Após o processo elicitação, os requisitos foram organizados conforme a granularidade das tarefas definidas no Scrumban (veja Seção 2.4), definindo assim a hierarquia das tarefas em histórias de usuário e critérios de aceitação.

A organização dos requisitos e sua documentação foi hospedada na ferramenta Github Projects, o qual possui um quadro de tarefas em formato de colunas e cartões, cuja estruturação e as movimentações, baseadas no Scrumban, será abordada na Seção 3.2.

3.2 Planejamento

O desenvolvimento do GOJ utilizou uma variação do Scrumban, utilizando dos seus ritos, como o *planning* e o *review*; sua metodologia de organização de tarefas; a estruturação do ciclo de desenvolvimento em *sprints* e, por fim, a estratégia de quebrar as tarefas em pedaços menores para permitir uma entrega continua.

O *planning* foi um momento de reunião com o *product owner* do projeto a fim de listar as quais serão as atividades a serem executadas na *sprint*. Já a *review* foi um momento de validação das tarefas realizadas na *sprint* que se encerrou.

O quadro de tarefas, baseado no Scrumban, apresentou as seguintes colunas: Novas *Issues*, onde foram colocadas as novas tarefas a serem feitas; o *Backlog* do produto armazenou as tarefas já planejadas para entrar em desenvolvimento; *Sprint Backlog* onde foram definidas as tarefas prioritárias para a *sprint*; Em Andamento, que continha as tarefas que estavam sendo trabalhadas; Revisão, coluna que listava as tarefas finalizadas e que necessitavam de aprovação para serem consideradas concluídas; por fim, a coluna Feito, que registrava as tarefas concluídas.

O movimento entre as colunas do quadro de tarefas se deu da seguinte forma: uma tarefa era adicionada ao quadro na coluna de Novas *Issues*; após a aprovação do PO, o cartão passava para a coluna do *Backlog* do Produto; após o rito da *planning* e de ser definido que seria desenvolvido na *sprint* corrente, o cartão era movido para a coluna de *sprint backlog*; ao iniciar o trabalho em uma tarefa, o seu cartão passava para a coluna Em Andamento; ao ser finalizada, a tarefa era movida para a coluna Revisão, indicando que estava disponível para ser revisada e, por fim, chegava a coluna de Feito, onde a tarefa estaria caso tivesse passado com sucesso pela revisão e fosse considerada pronta.

A definição de uma tarefa pronta, neste trabalho, é a seguinte: todos os critérios de aceitação estabelecidos forem cumpridos, funcionando de forma integrada com todo o sistema.

A ferramenta utilizada para a organização das tarefas foi o Github Projects que permite a criação de quadros em repositórios e organizações do Github. Este quadro, criado dentro da organização que contém os repositórios de código, tinha acesso privado. Após a finalização deste Trabalho de Conclusão de Curso, será disponibilizado acesso público ao quadro e aos repositórios onde o código está hospedado.

As *sprints* do projeto foram definidas com a duração de uma semana, começando com o *planning* e finalizando com o *review*. Esse tempo para a *sprint* foi definido para ser possível conseguir acoplar melhor o escopo do que havia para ser desenvolvido no tempo disponível, considerando que um período mais curto implicaria em tarefas ainda mais divididas.

A prioridade das tarefas de cada *sprint* foi definida no *planning*. Sendo assim, melhorias e problemas encontrados durante a fase de desenvolvimento podem ganhar prioridade frente as histórias de usuário levantadas inicialmente neste trabalho.

3.3 Manutenção e evolução

Nesta seção será discutido o que foi usado e como se deu o processo de desenvolvimento da aplicação. A arquitetura, as linguagens de programação e seus respectivos *frameworks* e bibliotecas foram os mesmos em ambos os semestres de desenvolvimento

deste trabalho e no trabalho anterior (veja Subseções 2.2.4 e 2.2.6) (LIMA; JUNIOR, 2021).

3.3.1 Paradigmas de programação

O Gamma Judge API, como dito na Subseção 2.2.4, é uma aplicação escrita em C# na versão 10 com o *framework* .NET 6.0, linguagem essa orientada objeto, que permite que aplicações desenvolvidas nessa linguagem utilizem a programação orientada ao objeto. Com isso o desenvolvimento se deu neste paradigma, observando a reutilização de código e a modularização.

Já o Gamma Judge UI e o Gamma Judge Admin, como citado na Subseção 2.2.6 utilizaram o *TypeScript*, versão 4.1.2 como linguagem de programação e o *React*, versão 17, como biblioteca principal. Apesar dessa linguagem ser orientada ao objeto, nas aplicações *frontend* a estrutura de objeto foi utilizada apenas para representação dos dados.

As aplicações *frontend* utilizaram a orientação a componente presente no React. Quando um elemento gráfico precisou ser representado em várias partes do código, para esse elemento foi criado um componente, prezando assim pelo reaproveitamento de código.

3.3.2 Banco de dados

O Gamma Judge Online utiliza o banco de dados no-SQL *MongoDB*, versão 6, com arquitetura baseado em documentos. Essa arquitetura de banco não permite uma modelagem conforme feito em bancos de dados relacionais.

Observando os requisitos levantados, viu-se a necessidade de gerar uma modelagem do banco para validar seu entendimento. Todavia a arquitetura do banco atual não é a mesma dos bancos relacionais, e por isso foram usados os tipos de dados nativos da ferramenta na modelagem. Já para representação dos relacionamentos foi feita uma abstração do que ocorre em bancos de dados relacionais. Quando o relacionamento ocorre por meio de referência ao identificador do outro integrante da relação, foi adicionado uma legenda `FK` simbolizando esse relacionamento. Já quando o relacionamento ocorre por meio de listas, a estrutura de cada item é mostrada em um cartão separado e na coleção que possui essa relação, é simbolizado que o campo é uma lista com o sua respectiva estrutura dos seus itens. Os campos que possuem a legenda `PK` são os identificadores únicos no banco de dados.

Na arquitetura do MongoDB não existe a necessidade de definir uma estrutura para as *collections*, como seria nas tabelas dos bancos de dados relacionais. O MongoDB tem poucas restrições, sendo necessário apenas a criação da *collection* para a inserção dos dados. Nesse contexto, o Gamma Judge API tem a responsabilidade definir a estrutura dos dados e fazer a validação, como tipos dos dados e demais restrições.

3.3.3 Ambiente de desenvolvimento

O ambiente de desenvolvimento utilizado foi divido em duas partes: a parte local, composta pelas aplicações Gamma Judge API, Gamma Judge UI, Gamma Judge Admin, Gamma Judge Tools e o MongoDB que são executados localmente na máquina *host*, com o auxílio do *Docker* na versão 23, além da parte em nuvem, composta pelas ferramentas do AWS: o SQS, o SNS e o S3. O sistema operacional usado foi o PopOS 22.04, com o *kernel* do Linux na versão 6.0.12.Para a documentação automática das rotas do Gamma Judge API, foi utilizado o Swagger gerado automaticamente pelo *framework* .NET

As versões das imagens Docker utilizadas foram as seguintes: na API, para o ambiente de desenvolvimento, foi utilizada a imagem `mcr.microsoft.com/dotnet/sdk:6.0`; já as aplicações *frontend* usaram a imagem `node:16`.

4 Resultados

Observando a metodologia abordada no Capítulo 3, este capítulo irá abordar os resultados deste Trabalho de Conclusão de Curso. Os resultados obtidos estão divididos em duas partes: a primeira se refere aos Requisitos, onde foram elicitados e convertidos em histórias de usuário enquanto a segunda, aos resultados de codificação e modelagem na Aplicação.

4.1 Requisitos

Utilizando a técnica de *brainstorm* o processo de elicitação de requisitos foi dividido em duas fases. Na primeira fase foram listadas ideias de funcionalidades e melhorias na aplicação, gerando os seguintes requisitos presentes na Tabela 1. Nesta etapa, o objetivo era listar as ideias sem se preocupar com sua escrita.

Já na segunda fase, de refinamento, o objetivo foi elaborar uma redação mais definida para melhor entendimento. A partir desse ponto, foram criadas e listadas as seguintes histórias de usuário, mostradas na Tabela 2 baseada nos requisitos gerados na fase anterior.

Utilizando a ferramenta do Github Project, foi criado um quadro de tarefas baseado nas práticas do Scrumban e, seguindo sua organização das tarefas, foi criado um cartão para cada história de usuário, com o prefixo [USxx] sendo xx o identificador da história. Os cartões foram organizados e colocados na coluna de *backlog*, deixando apenas a US04 na coluna Em Andamento. A Figura 6 mostra as tarefas organizadas na ferramenta do Github.

Durante a etapa de desenvolvimento foi necessário algumas alterações no quadro de tarefas para melhor organização do trabalho. A primeira parte foi a tradução das colunas

Tabela 1 – Lista de requisitos obtida

Requisito	**Descrição**
1	Notas ao final do texto do problema
2	Carregar os testes secretos a partir do Gamma Judge Admin
3	Importar problemas do CP Tools
4	Sistema de autenticação
5	Estatísticas do usuário
6	Sistema para realizar *contests*

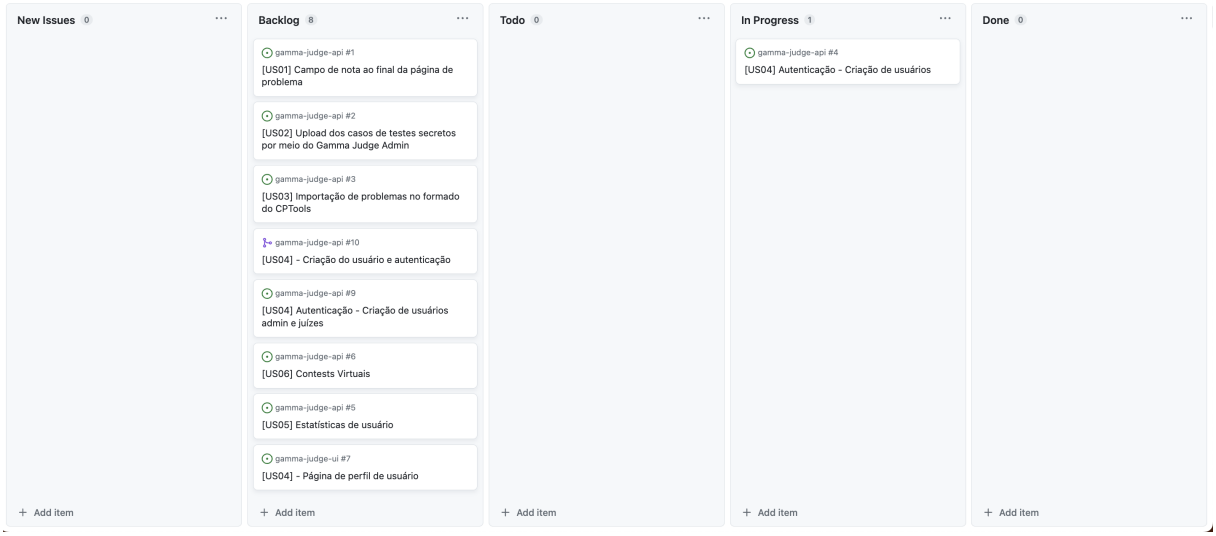
Fonte: o Autor

Tabela 2 – Lista de histórias de usuário

Identificador	**História de usuário**
US01	Eu, como juiz/administrador, desejo adicionar notas ao final do problema para explicar os casos de teste de exemplo
US02	Eu, como juiz/administrador, desejo poder carregar os testes secretos a partir da interface do Gamma Judge Admin
US03	Eu, como juiz/administrador, desejo importar um problema utilizando o formato do CPTools
US04	Eu, como usuário, desejo poder me cadastrar e realizar *login* na aplicação
US05	Eu, como usuário, desejo ter estatísticas além de histórico de submissões para acompanhar meu desenvolvimento
US06	Eu, como usuário, quero poder praticar por meio de *contests* virtuais

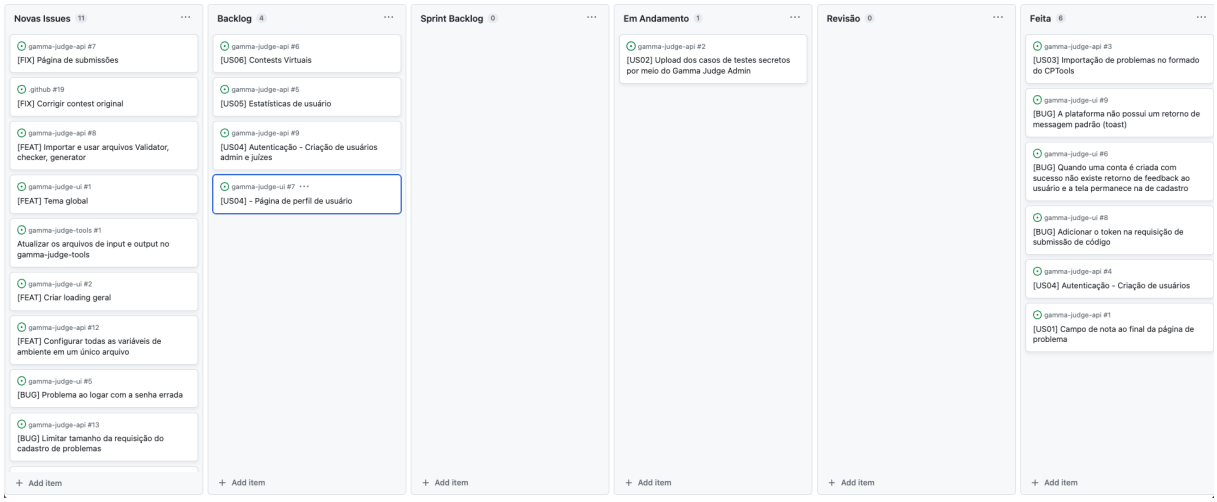
Fonte: o Autor

Figura 6 – Quadro de tarefas



Fonte: o Autor

Figura 7 – Quadro de tarefas durante desenvolvimento



Fonte: o Autor

na própria ferramenta para melhor paridade com as colunas definidas neste trabalho. Já a segunda foi o mapeamento das colunas, adicionando neste trabalho a coluna de *Sprint Backlog* e no quadro a coluna de Revisão. A Figura 7 mostra o estado do quadro de tarefas durante o desenvolvimento

Cabe destacar que o quadro comporta, em forma de cartões, novas melhorias e problemas encontrados durante a fase de desenvolvimento. Esses cartões têm o objetivo de manter o rastreamento das necessidades de melhoria e correção do Gamma Judge Online, durante e após a conclusão deste trabalho.

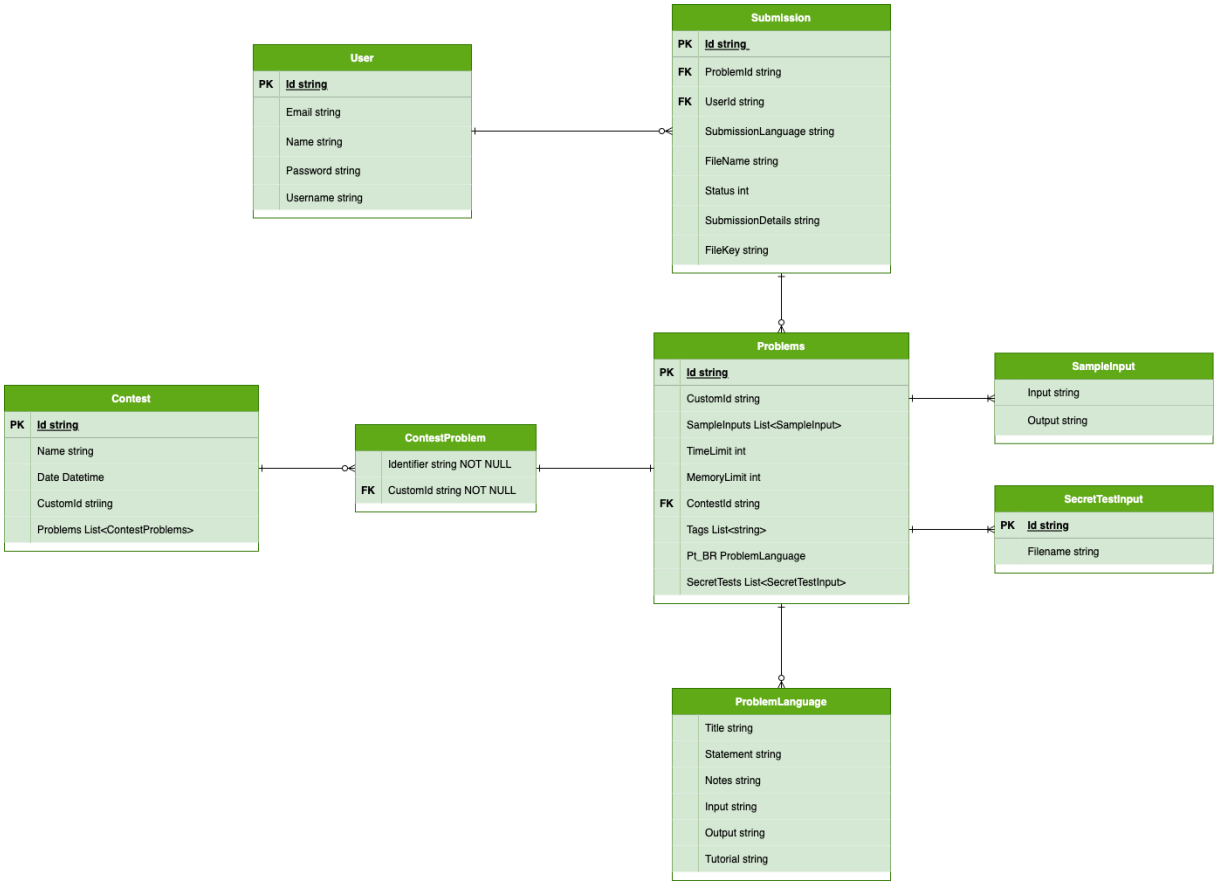
4.2 Aplicação

Esta seção aborda os resultados do desenvolvimento dividindo-os em duas partes: as histórias de usuário já previstas no *brainstorm* inicial e melhorias que foram entendidas como prioridade, em detrimento de outras histórias de usuário.

4.2.1 Modelagem

Com base na metodologia presente na Seção 3.3.2, os dados foram modelados considerando os requisitos elicitados na Seção 4.1, visando um melhorar o entendimento dos requisitos e da proposta de solução. A Figura 8 mostra a modelagem do banco de dados após os resultados obtidos no desenvolvimento.

Figura 8 – Modelagem dos dados



Fonte: o Autor

4.2.2 US01 - Campo de notas ao final do texto

Essa história de usuário originou da necessidade de colocar notas que explicam como entradas dos casos de teste de exemplo geram suas respectivas saídas. Esse campo é opcional, visto que a necessidade de explicar os casos de exemplo é definida pelo responsável pela elaboração do problema.

O Gamma Judge API já tratava o campo de notas nos problemas, mas esse campo não era mostrado corretamente no Gamma Judge UI. Corrigido o problema de integração entre ambas aplicações, o campo de notas foi mostrado corretamente conforme o problema cadastrado. A Figura 9 mostra parte de uma página de um problema exemplificando essa funcionalidade.

4.2.3 US03 - Importação de problemas e de *constests*

Essa história de usuário se refere a uma forma de importação que facilite a inclusão de problemas e de *contests* de forma automática. A ideia é que ambos possam ser codificados em um formato padrão e importados, eliminando a necessidade de preencher os campos no Gamma judge Admin.

Figura 9 – Notas ao final do texto de um problema

Fonte: o Autor

A solução desenvolvida foi definir um padrão de formatação tanto do problema quanto dos *contests* em JSON, o mesmo formato usado pelo Gamma Judge Admin para enviar ambos ao Gamma Judge API. Foi adicionado um botão para que o usuário possa carregar o JSON do problema ou do *contest* no Gamma Judge Admin. O Código 1 mostra o formato JSON definido para o problema, o qual possui os seguintes campos:

- `customId`: identificado único para o problema definido pelo usuário que está cadastrando o problema;
- `timeLimit`: tempo limite para execução das soluções em segundos;
- `memoryLimit`: limite de memória que a solução pode usar;
- `tags`: lista de temas que a questão aborda e é definida manualmente pelo usuário. Pode ser usada para listar os conteúdos que envolvem a solução do problema;
- `sampleInputs`: lista dos casos de testes de exemplos. No JSON é definida como uma lista objetos compostos pelos campos `input`, as entradas, e `output`, as saídas;
- `pt_BR`: campo destinado aos dados textuais do problema em português do Brasil (veja Subseção 4.2.7);
- `title`: título do problema;
- `statement`: descrição do problema, codificado em Base64;
- `input`: descrição da entrada do problema, codificado em Base64;

- `output`: descrição da saída esperada pelo problema, codificado em Base64;
- `tutorial`: tutorial de resolução do problema, codificado em Base64;
- `notes`: notas após os casos de testes de exemplo, codificado em Base64.

Código 1 – Formato do problema para importação

```
{
  "customId": "prob1",
  "timeLimit": 1,
  "memoryLimit": 256,
  "tags": [
      "Teste"
  ],
  "sampleInputs": [
      {
          "input": "1 2\n",
          "output": "3\n"
      }
  ],
  "pt_BR": {
      "title": "Divisores de Fatoriais",
      "statement": "<base64>",
      "input": "<base64>",
      "output": "<base64>",
      "tutorial": "<base64>",
      "notes": "<base64>"
    }
}
```

Fonte: o Autor

O Código 2 mostra o formato esperado para cadastro de um novo *contest*. O JSON possui os seguintes campos:

- `name`: nome do *contest*;
- `date`: data e hora no padrão ISO 8601;
- `customId`: identificador único definido pelo usuário;
- `problems`: lista dos problemas presentes no *contest*. Cada problema possui dois campos: `identifier`, que é um identificador para o problema dentro do contest, e `customId`, que referencia o problema a ser adicionado no *contest*.

Código 2 – Formato do *contest* para importação

```
{
  "name": "",
  "date": "2023-05-23T21:51:59.391Z",
  "customId": "1",
  "problems": [
    {
      "identifier": "A",
      "customId": "1A"
    },
    {
      "identifier": "B",
      "customId": "1B"
    }
  ]
}
```

Fonte: o Autor

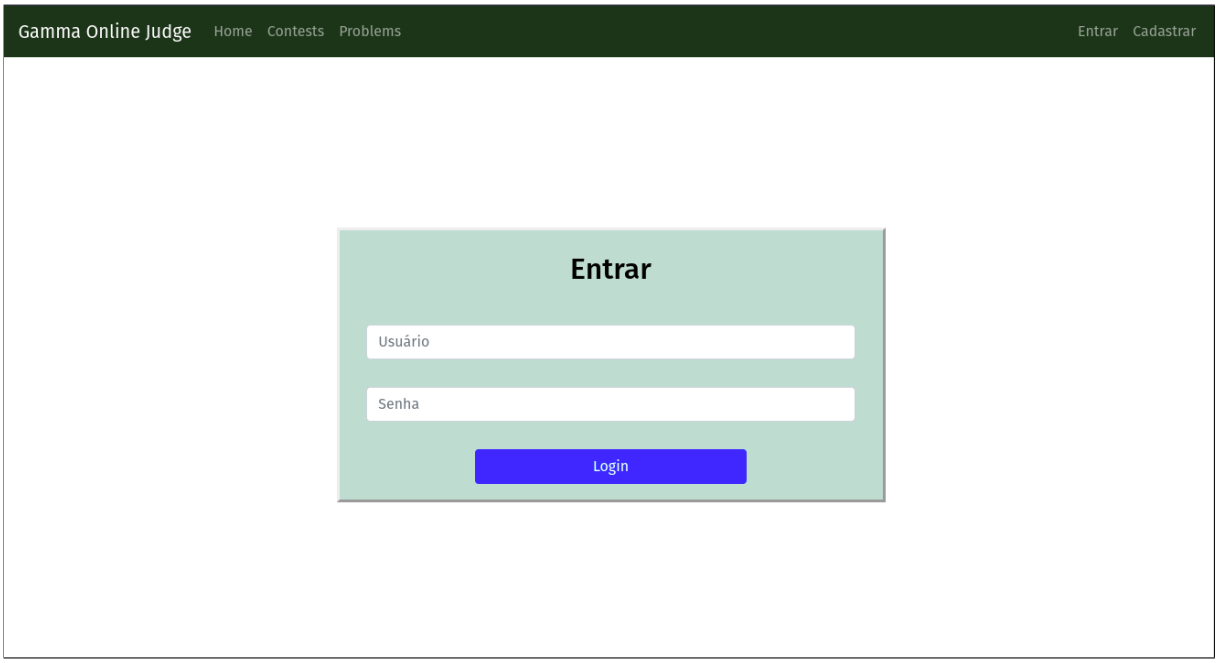
4.2.4 US04 - Autenticação

Essa história de usuário se originou da necessidade de identificar o usuário na plataforma, tanto para manter um histórico de uso, quanto para diferenciar os usuários nas submissões e nos *contests*. Para essa funcionalidade foram definidos 4 tipos de usuários: visitante, um usuário não autenticado que pode navegar entre os *contests* e problemas, mas sem poder submeter uma solução; usuário autenticado, com as mesmas permissões do visitante, mas podendo submeter sua solução aos problemas; o juiz, que além das permissões já citas, pode acessar o Gamma Judge Admin para cadastro e gerenciamento de problemas; por fim, o administrador, que possui todas as permissões possíveis.

Visto que essa história de usuário tinha um escopo muito aberto, foi necessário dividi-la em três partes: autenticação no Gamma Judge UI, com os usuários autenticados; no Gamma Judge Admin com os juízes e administradores; e a criação da página de perfil do usuário. Neste trabalho foi desenvolvido a primeira parte, com a autenticação Gamma Judge UI.

No Gamma Judge API foram criados as rotas para criação e leitura de usuários, além da rota que permite a autenticação recebendo o usuário e a senha e retornando um *token* JWT usado para autenticar nas demais rotas. A escolha do uso de *tokens* JWT para a autenticação evita o envio de usuário e senha toda vez que for necessário autenticar em uma rota. Ademais, para o armazenamento das senhas, foi utilizado a funcionalidade fornecida pelo .NET, no pacote `Microsoft.AspNetCore.Identity`, na versão 2.2.0, que armazena as senhas em *hash* e faz a sua validação.

Figura 10 – Tela de Login



Fonte: o Autor

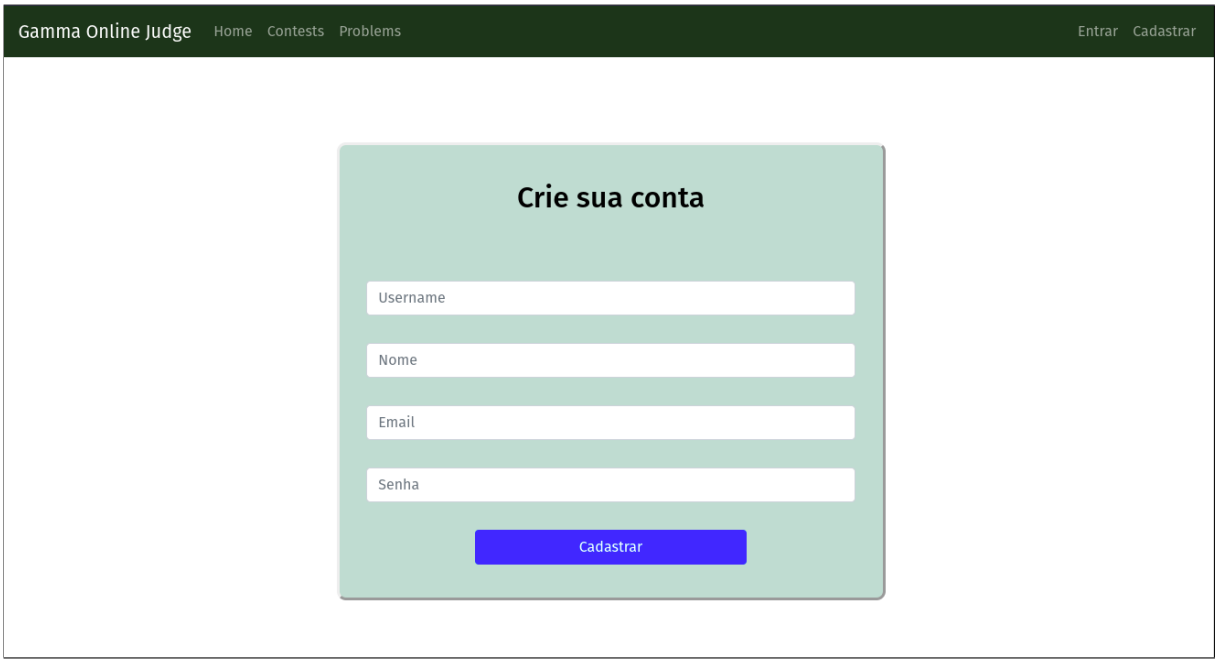
No Gamma Judge UI foram criadas as telas para *login* e cadastro mostradas nas Figuras 10 e 11, respectivamente. Além das telas criadas, foi criado um Context API, funcionalidade presente no React, que permite que o *token* obtido a partir do *login* no Gamma Judge API seja armazenado, permitindo ao usuário mudar de página ou de aba no navegador e continuar autenticado.

4.2.5 Melhoria – Suporte a *tags* HTML

A adição de imagens na formatação de um problema é mais um elemento que pode facilitar o entendimento do enunciado do problema. Sendo assim foi visto como necessário que o Gamma Online Judge tivesse suporte a funcionalidade de exibição de imagens nos problemas, tanto na parte dos enunciados, quanto na parte de notas e tutoriais. Além da inserção das imagens, a imagem deveria ter um posicionamento determinado pelo usuário dentro do corpo do texto. O LaTeX, que já é usado na formatação, oferece suporte para inserir as imagens, porém em formato de caminho do arquivo, de modo que é necessário que a imagem seja salva e disponibilizada online.

O LaTeX utilizado para formatação dos problemas possui suporte limitado à sua sintaxe. Algumas funcionalidades como o `enumerate` não foram renderizadas corretamente, além de não ser possível adicionar pacotes. Esses problemas levaram a necessidade de outra forma de formatar o texto. Utilizando o exemplo do `enumerate` que não funcionou corretamente, foi utilizado em seu lugar as *tags* de listas do HTML.

Figura 11 – Tela de Cadastro



Fonte: o Autor

Essa solução foi adotada, pois é bastante flexível, permitindo que uma imagem seja adicionada em qualquer lugar do texto e também que o usuário decida o formato mais adequado da imagem a ser inserida. Realizar o *upload* das imagens no S3 e referenciá-las no texto foi uma solução cogitada, mas apresentava três problemas: o primeiro é que permitia ao usuário salvar inúmeras imagens, possuindo, na prática, permissão de escrita no serviço da Amazon; já o segundo dizia respeito a referência da imagem do texto, que precisava ser corretamente posicionada; por fim, essa solução não contemplava os outros problemas relacionados ao suporte limitado do LaTeX no navegador.

4.2.6 Melhoria - Armazenamento de parte das questões em Base64

Tanto o LaTeX quanto o HTML possuem uma série de caracteres especiais que podem ocasionar erros no armazenamento e transporte do problema em formato JSON. Durante o cadastro de alguns problemas, foram encontrados erros na formatação, principalmente quando adicionava caracteres especiais como aspas duplas e barra invertida que levava a comportamentos inesperados na renderização do problema. Portanto, foi necessário definir uma forma que evitasse que esses caracteres especiais quebrassem a formatação do problema.

A primeira solução encontrada foi armazenar o problema inteiro em um único arquivo, porém isso dificultava a alocação os campos, como as notas e o tutorial, de forma separada. O uso de um arquivo para cada campo foi cogitada, porém aumentaria consideravelmente o uso do S3 ou do banco de dados para armazenamento dos arquivos.

Nesse contexto a solução encontrada foi continuar com o mesmo processo, entretanto codificando os campos em Base64, que possui apenas caracteres alfanuméricos, `+` e `=` e, portanto, sem caracteres especiais que possam impactar negativamente na formatação do problema.

Quando o usuário insere o texto de um problema no Gamma Judge Admin, esse texto é codificado em Base64 e enviado ao Gamma Judge API, que salva a informação no banco de dados. Quando o usuário acessa um problema, via Gamma Judge UI, a API retorna o problema com os devidos campos codificados e, ao exibir, os dados são decodificados. Apenas os campos `statement`, `input`, `output`, `tutorial` e `notes` que utilizam esse recurso, visto que são os que oferecem suporte tanto ao LaTeX quanto ao HTML.

4.2.7 Melhoria – Adição do suporte para múltiplas línguas na formatação dos problemas

Dado o estado de desenvolvimento do Gamma Judge Online, a funcionalidade de múltiplos idiomas não é vista como prioritária no contexto que envolve este trabalho. Entretanto, a separação dos campos traduzíveis em cada língua não foi uma tarefa custosa em relação ao tempo e evitou que o crescimento do Gamma Judge Online visse a dificultar o desenvolvimento dessa funcionalidade.

A solução desenvolvida foi separar, para cada língua, os campos `title`, `statement`, `input`, `output`, `tutorial` e `notes`. No atual ponto de desenvolvimento estes campos foram acrescidos apenas para o português do Brasil, visto que o Gamma Judge Admin e o Gamma Judge Admin não oferecem suporte a outra língua. Como exemplo do formato adotado, podemos ver no Código 1 o campo `pt_BR`, que representa o português do Brasil.

5 Considerações Finais

Ao longo deste trabalho foram implementadas algumas das histórias de usuários previstas no levantamento inicial dos requisitos além das melhorias que foram adicionadas no decorrer do processo de desenvolvimento.

O principal desafio foi evoluir uma aplicação já existente que envolve, além de entender a ideia do projeto, compreender a parte técnica do desenvolvimento, desde a instalação e configuração até os padrões de codificação aplicados no código. A evolução de um projeto limita as decisões a serem tomadas acerca da escolha da arquitetura e das tecnologias. Todavia este trabalho obteve vários progressos na manutenção e evolução do Gamma Online Judge.

Apesar do foco inicial deste trabalho ser os *contests* virtuais, as funcionalidades desenvolvidas reduzem as dependências necessárias para seu desenvolvimento. A autenticação e criação de usuários, permite que a aplicação diferencie as submissões, enquanto a importação de problemas e a melhorias na formatação melhoram sua exibição, facilitando o entendimento dos problemas.

O desenvolvimento das interfaces, o Gamma Judge UI e o Gamma Judge Admin, se baseou apenas na descrição textual das funcionalidades, sem um protótipo ou demonstração gráfica. Isso implicou em uma interface que não ficou consistente, pois, além de ser um código legado, é um projeto ainda no início e que não tem muitas telas para se basear. É interessante a construção de um protótipo de alta fidelidade que poderia ser usado para guiar a continuação do desenvolvimento das interfaces do Gamma Online Judge.

O Gamma Judge Online é um projeto mantido pelo professor Dr Edson Alves da Costa Júnior e tem por objetivo tornar o projeto *open source*, entretanto, o processo para abertura pública do código não é algo claro quanto a melhor metodologia que deve ser empregada, quais documentações são imprescindíveis e, principalmente, como atrair contribuintes. Nesse contexto uma sugestão de trabalho futuro é em relação a como realizar esse processo.

Referências

ABNT. *Sistemas de gestão da qualidade - Fundamentos e vocabulário*. 3. ed. Rio de Janeiro, Brasil, 2015. Citado na página 23.

AMAZON. *Armazenamento S3 - Simple Storage Service - Amazon Web Services*. 2023. Acesso em 5 de fevereiro de 2023. Disponível em: <https://aws.amazon.com/pt/s3/>. Citado na página 20.

AMAZON. *AWS SNS - Amazon Simple Notification Service - Amazon Web Services*. 2023. Acesso em 5 de fevereiro de 2023. Disponível em: <https://aws.amazon.com/pt/s3/>. Citado na página 19.

AMAZON. *Enfileiramento de mensagens totalmente gerenciado – Amazon Simple Queue Service – Amazon Web Services*. 2023. Acesso em 5 de fevereiro de 2023. Disponível em: <https://aws.amazon.com/pt/sqs/>. Citado na página 19.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley Professional, 2004. Citado na página 24.

BRADSHAW, S.; BRAZIL, E.; CHODOROW, K. *MongoDB The Definitive Guide: Powerful and Scalable Data Storage*. 3. ed. [S.l.]: O'Reilly, 2020. Citado na página 20.

CAMPOS, C. P. de; FERREIRA, C. E. Boca: um sistema de apoio a competições de programação. In: SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. *Workshop de Educação em Computação*. [S.l.], 2004. Citado na página 13.

GALVãO, L.; FERNANDES, D.; GADELHA, B. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: . [S.l.]: Sociedade Brasileira de Computação - SBC, 2016. v. 1, p. 140. Citado na página 12.

HALIM, S.; HALIM, F. *Competitive Programming 3: The New Lower Bound of Programming Contests*. Lulu.com, 2013. ISBN 9788392212355. Disponível em: <https://books.google.com.br/books?id=vUc-nwEACAAJ>. Citado na página 12.

ICPC . *ICPC - International Collegiate Programming Contest*. 2023. <https://icpc.global/>. Acesso em 30 de maio de 2023. Citado na página 12.

INGALLS, S. *What Is a Client-Server Model? A Guide to Client-Server Architecture*. 2021. Acesso em 2 de fevereiro de 2023. Disponível em: <https://www.serverwatch.com/guides/client-server-model/>. Citado na página 18.

IOI. *IOI - International Olympiad in Informatics*. 2023. <https://ioinformatics.org/>. Acesso em 30 de maio de 2023. Citado na página 12.

KHAN, Z. A. Scrumban - adaptive agile development process : Using scrumban to improve software development process. In: . [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 23 e 24.

LADAS, C. *Scrumban: Essays on Kanban Systems for Lean Software Development*. [S.l.]: Modus Cooperandi Press, 2008. Citado 2 vezes nas páginas 23 e 24.

LIMA, G. M.; JUNIOR, E. A. D. C. Gamma online judge: projeto de criação de uma plataforma para o armazenamento das questões das maratonas unb de programação. 2021. Citado 7 vezes nas páginas 13, 14, 19, 20, 21, 22 e 27.

MENDONçA, R. A. R. de. Levantamento de requisitos no desenvolvimento ágil de software. *Semana da Ciência e Tecnologia da PUC Goiás*, p. 12, 2014. Citado na página 23.

PRITCHARD, J. *Component Driven Development (CDD) In React*. 2019. Acesso em 31 de janeiro de 2023. Disponível em: <https://whatjackhasmade.co.uk/component-driven-development/>. Citado na página 17.

RAMOS, G. N. *IX Maratona UNB*. 2021. Disponível em: <http://maratona.unb.br/noticias/74-ix-maratona-unb-2021>. Citado na página 13.

REVILLA, M. A.; MANZOOR, S.; LIU, R. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, Institute of Mathematics and Informatics, v. 2, n. 10, p. 131–148, 2008. Citado na página 12.

RIBEIRO, R. B. et al. Gamificação de um sistema de juiz online para motivar alunos em disciplina de programação introdutória. In: . [S.l.]: Brazilian Computer Society (Sociedade Brasileira de Computação - SBC), 2018. v. 1, p. 805. Citado na página 12.

SAAD, D. *Maratonas DF: Restrospectiva 2019*. 2019. Acesso em 18 de janeiro de 2023. Disponível em: <https://danielsaad.com/maratona/retrospectivas/retrospectiva-2019/index.html>. Citado na página 13.

SBC. *O que é*. 2022. Acesso em 17 de janeiro de 2023. Disponível em: <https://maratona.sbc.org.br/sobre22.html>. Citado 2 vezes nas páginas 12 e 13.

Swagger. *Swagger: API Documentation*. 2023. <https://swagger.io/solutions/api-documentation/>. Acesso em 29 de junho de 2023. Citado 2 vezes nas páginas 17 e 18.

TANENBAUM, A. S.; STEEN, M. V. *Sistemas Distribuídos: princípios e paradigmas*. 2ª edição. ed. São Paulo: Pearson Education, 2008. Citado na página 17.

VAZQUEZ, C. E.; SIMõES, G. S. *Engenharia de requisitos: software orientado ao negócio*. [S.l.]: Brasport, 2016. Citado 2 vezes nas páginas 22 e 24.

VINCENZI, A. M. R. *Orientação a objeto: definição, implementação e análise de recursos de teste e validação*. Tese (Doutorado) — Universidade de São Paulo, 2004. Citado na página 17.

VOLLE, A. *Web application*. 2022. Acesso em 1 de fevereiro de 2023. Disponível em: <https://www.britannica.com/topic/Web-application>. Citado na página 17.

WASIK, S. et al. Survey on online judge systems and their applications. In: . [S.l.]: ACM Computing Surveys, 2016. v. 1. ISBN 9781450392983. Citado na página 12.

WOZNIEWICZ, B. *The Difference Between a Framework and a Library*. 2019. Acesso em 30 de janeiro de 2023. Disponível em: <https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/>. Citado na página 16.