Universidade Federal da Bahia
Instituto de Computação
Bacharelado em Sistemas de Informação

Trabalho de Conclusão de Curso

Jude 2.x.x: JULGADOR ONLINE DE SOLUÇÕES PARA PROBLEMAS DE PROGRAMAÇÃO

Lucas Borges Pereira

Salvador - Bahia

26 de julho de 2025

Jude 2.x.x: JULGADOR ONLINE DE SOLUÇÕES PARA PROBLEMAS DE PROGRAMAÇÃO

Lucas Borges Pereira

Trabalho de Conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas da informação.

Orientador: Prof. Dr. Rubisley de Paula Lemes.

Salvador - Bahia
26 de julho de 2025

Jude 2.x.x: JULGADOR ONLINE DE SOLUÇÕES PARA PROBLEMAS DE PROGRAMAÇÃO

Lucas Borges Pereira

Trabalho de Conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Banca Examinadora:

Prof. Dr. Rubisley de Paula Lemes (Orientador)
UFBA

Prof. Dr. Danilo Barbosa Coimbra
UFBA

Prof. Dr. Rodrigo Rocha Gomes e Souza
UFBA

Resumo

O ensino de programação enfrenta desafios significativos na avaliação prática de códigos, especialmente em turmas grandes, onde o feedback manual limita o progresso dos estudantes. Neste contexto, sistemas julgadores automáticos de código ganham relevância ao proporcionar correção automatizada, feedback instantâneo e incentivo à aprendizagem iterativa. Este trabalho apresenta as melhorias, correções e novas funcionalidades implementadas no Jude, sistema desenvolvido pela Universidade Federal da Bahia, utilizado no apoio ao ensino de programação. Foram realizadas melhorias técnicas, funcionais e estruturais, incluindo a substituição do Celery por Webhooks, reformulação da lógica de turmas e competições, implementação de sistema de chamadas em tempo real, novo modelo de versionamento semântico e melhorias no ambiente de desenvolvimento. A abordagem adotada baseou-se em princípios da engenharia de software e em ciclos iterativos de entrega contínua. Os resultados demonstram maior estabilidade, usabilidade e escalabilidade do sistema, além de facilitarem sua manutenção e evolução futura. Este trabalho contribui para o fortalecimento de ferramentas educacionais no ensino de programação e consolida boas práticas de desenvolvimento aplicadas ao contexto acadêmico.

Abstract

Teaching programming presents major challenges in the practical evaluation of code, especially in large classes, where manual feedback limits student progress. In this context, automatic code judge systems become increasingly relevant by offering automated correction, instant feedback, and encouragement for iterative learning. This study presents the improvements, fixes, and new features implemented in Jude, a system developed by the Federal University of Bahia to support programming education. Technical, functional, and structural enhancements were carried out, including the replacement of Celery with Webhooks, redesign of the logic for classes and competitions, implementation of a real-time attendance system, adoption of semantic versioning, and improvements to the development environment. The approach was based on software engineering principles and iterative delivery cycles. The results show improved system stability, usability, and scalability, while facilitating future maintenance and evolution. This work contributes to strengthening educational tools in programming education and consolidates good software development practices in the academic context.

Capítulo 1

Introdução

A evolução acelerada da tecnologia da informação e a crescente demanda por soluções computacionais robustas têm transformado significativamente o mercado de trabalho na área de software. Nesse contexto, formar profissionais capazes de projetar, desenvolver e validar sistemas eficientes tornou-se um dos principais desafios das instituições de ensino superior. Entre os componentes curriculares fundamentais para essa formação está o ensino da programação de computadores, que fornece a base lógica e técnica para o desenvolvimento de soluções computacionais em diferentes níveis de complexidade.

Entretanto, apesar da centralidade da programação nos currículos, sua avaliação prática ainda representa um gargalo. O processo manual de correção de códigos exige esforço e tempo significativos dos professores, além de limitar o volume e a diversidade de exercícios que podem ser propostos. Em turmas grandes, essa limitação é ainda mais evidente, comprometendo a frequência do feedback e, por consequência, o ritmo de aprendizagem dos alunos [9].

Como resposta a esse cenário, surgiram os **sistemas julgadores automáticos de códigos**, também conhecidos como **online judges**. Esses sistemas permitem que alunos submetam suas soluções para problemas de programação estruturados previamente, recebendo correções automáticas com base em critérios como tempo de execução, uso de memória e aderência ao resultado esperado [22, 21]. Além de aliviarem a carga de trabalho docente, essas ferramentas promovem autonomia, feedback instantâneo e uma cultura de aprendizado iterativo entre os estudantes [4].

No contexto internacional, plataformas como UVA Online Judge, HackerRank, LeetCode, Beecrowd e Sphere Online Judge (SPOJ) têm sido amplamente utilizadas tanto para ensino quanto para competições de programação [19, 11, 15, 20, 3]. No cenário brasileiro, destaca-se também o BOCA (sistema de apoio a competições de programação), utilizado em eventos como a Maratona de Programação da SBC [5].

Na Universidade Federal da Bahia (UFBA), destaca-se o uso do sistema **Jude On-**

line Judge [13], uma ferramenta desenvolvida internamente para aplicação em disciplinas de programação. O Jude possui uma arquitetura baseada em microserviços e contêineres Docker, utilizando tecnologias modernas como React no frontend, Django no backend, e Judge0 como motor de execução de código remoto [14]. A infraestrutura também conta com Redis para gerenciamento de cache e filas, WebSockets para notificações em tempo real, PostgreSQL como sistema gerenciador de banco de dados, além de serviços auxiliares como JPlag e Sherlock para análise de plágio.

Apesar de funcional e amplamente adotado no ambiente acadêmico da UFBA, o sistema ainda apresentava uma série de limitações, tanto do ponto de vista do usuário final (professores e alunos) quanto da equipe de desenvolvimento. Entre os principais problemas identificados estavam a dificuldade de configurar o ambiente local de desenvolvimento, a ausência de documentação clara, inconsistências no gerenciamento de turmas e competições, limitações na interface de submissões, e a falta de ferramentas práticas para controle de presença e manutenção das bases de dados de alunos.

Diante desse contexto, este trabalho propõe a realização de um conjunto de **melhorias estruturais, técnicas e funcionais** no sistema Jude. O objetivo é não apenas ampliar suas capacidades e facilitar a vida dos usuários, mas também promover um ambiente de desenvolvimento mais sustentável e acessível. As intervenções realizadas incluem a criação de documentação para o ambiente de desenvolvimento, correção de dependências quebradas, implementação de suporte a debugging com `debugpy`, reformulação do fluxo de submissões com remoção do uso do Celery, reestruturação da lógica de turmas e competições, criação de um sistema de chamadas online com WebSockets, e diversas melhorias na usabilidade da interface.

Além disso, para fundamentar a avaliação dos resultados, este trabalho propõe a aplicação de um instrumento de coleta de dados voltado para professores. O objetivo é medir, de forma empírica, o impacto das mudanças implementadas a partir de percepções práticas dos professores.

Assim, este trabalho contribui tanto para o aprimoramento de um sistema já em uso no contexto educacional quanto para a consolidação de boas práticas de engenharia de software aplicadas a sistemas acadêmicos.

Este documento está estruturado da seguinte maneira: o Capítulo 2 apresenta a fundamentação teórica relacionada aos sistemas julgadores automáticos de código, abordando tanto aspectos tecnológicos quanto pedagógicos; o Capítulo 3 descreve a metodologia adotada para conduzir o desenvolvimento e a avaliação das melhorias propostas; o Capítulo 4 detalha, em profundidade, as alterações técnicas e funcionais realizadas; o Capítulo 5 discute os resultados obtidos a partir da análise dos dados coletados junto aos usuários; e, por fim, o Capítulo 6 apresenta as conclusões do trabalho, suas limitações e

sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Sistemas Julgadores de Códigos

Sistemas julgadores de códigos são plataformas computacionais capazes de avaliar automaticamente a correção, eficiência e robustez de algoritmos submetidos por usuários. Essas ferramentas têm papel fundamental tanto no ensino de programação quanto em competições, hackathons e ambientes de avaliação técnica. Elas permitem que múltiplos códigos sejam avaliados simultaneamente, de forma padronizada e com critérios objetivos, como tempo de execução, uso de memória e correção funcional [22].

No contexto educacional, os online judges contribuem para a automação da avaliação, redução da carga docente e promoção do aprendizado baseado em tentativa e erro, onde os alunos recebem feedback quase instantâneo [4]. Essa estratégia favorece o engajamento dos discentes e o desenvolvimento de habilidades de resolução de problemas de forma incremental.

Esses sistemas geralmente operam por meio da execução dos códigos enviados em ambientes isolados (sandbox), comparando a saída produzida com a saída esperada definida em um conjunto de testes. Adicionalmente, podem empregar avaliações parciais (score fracionado), identificação de erros semânticos, e validação de casos extremos.

Entre os exemplos mais conhecidos no cenário internacional estão o **HackerRank** [11], **LeetCode** [15], **Beecrowd** (antigo URI Online Judge) [3] e o **SPOJ** [20]. Essas plataformas oferecem ambientes interativos, rankings, problemas categorizados por nível e suporte a múltiplas linguagens. Elas servem tanto como apoio educacional quanto como ferramentas de seleção profissional.

Sistemas como o **BOCA** [5], utilizado em competições como a Maratona de Programação da SBC, e o **UVa Online Judge** [19], voltado à prática sistemática de algoritmos, também se destacam como referências históricas e educacionais.

2.2 O Sistema Jude Online Judge

O **Jude Online Judge** é um sistema desenvolvido e mantido pela Universidade Federal da Bahia (UFBA), utilizado principalmente como ferramenta de apoio às disciplinas de programação. Seu objetivo é prover uma plataforma integrada e moderna para submissão, execução e avaliação automatizada de códigos, além de permitir a gestão eficiente de turmas, competições e históricos de desempenho [13].

O sistema é baseado em tecnologias contemporâneas como **React**, **Django**, **PostgreSQL**, **Docker** e **Judge0**, adotando uma arquitetura modular e escalável. Essa arquitetura foi desenhada para atender às necessidades de avaliação automática em grande escala, respeitando critérios de segurança, isolamento e flexibilidade na configuração de problemas e linguagens suportadas.

Entre as principais funcionalidades do sistema, destacam-se:

- **Criação e gerenciamento de competições:** Utilizadas em provas, listas de exercícios ou eventos de programação competitiva, com opções de tempo, participantes e visibilidade. (ver Figura 2.1)
- **Cadastro de problemas:** Professores podem registrar nome, restrições e casos de teste. (ver Figura 2.2)
- **Gerenciamento acadêmico:** Inclui cadastro de disciplinas, semestres e turmas. (ver Figura 2.3)
- **Relatório de notas:** Geração de relatório de notas com base nas submissões, organizadas por turma e competição. (ver Figura 2.4)
- **Dashboard:** Tela com visão geral das atividades recentes e dados sobre problemas. (ver Figura 2.5)

Além dessas, o sistema também conta com funcionalidades complementares como controle de presença, bloqueio de participantes, verificação de plágio e visualização do histórico de submissões, compondo uma ferramenta completa para o apoio ao ensino de programação.

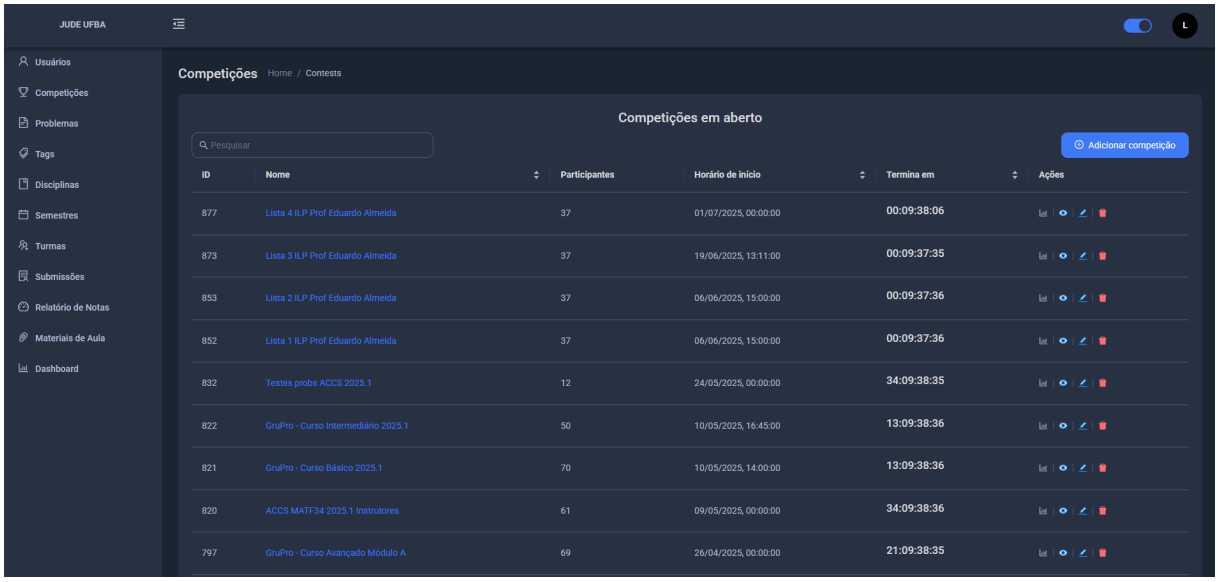


Figura 2.1: Interface de listagem de competições.

Figura 2.2: Tela de cadastro de problema.

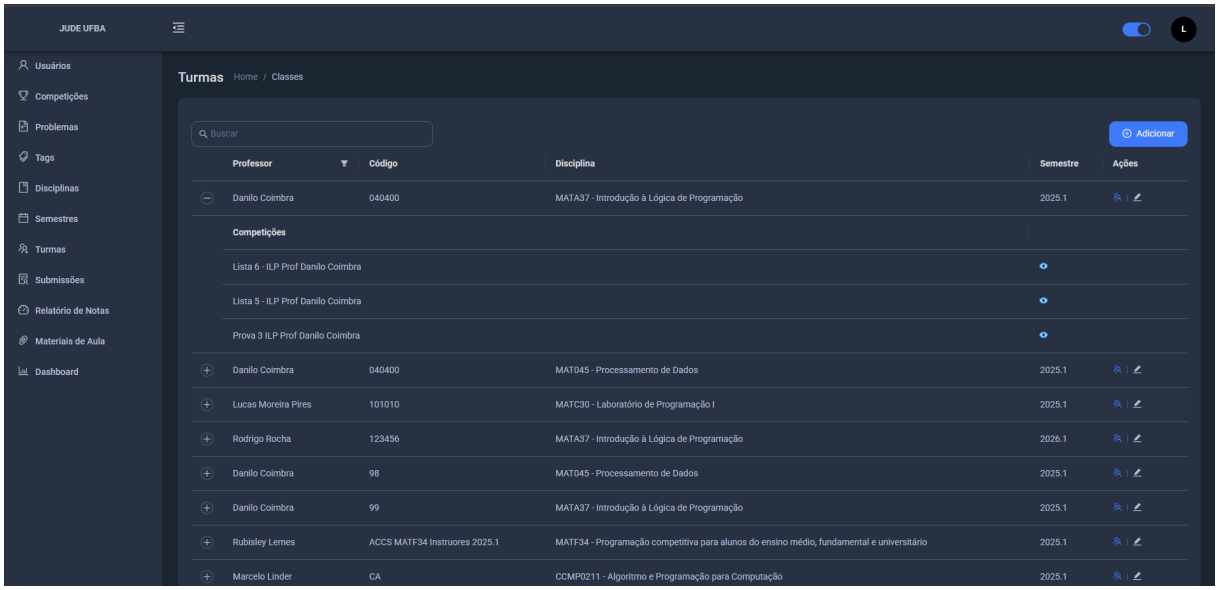


Figura 2.3: Gerenciamento de turmas.

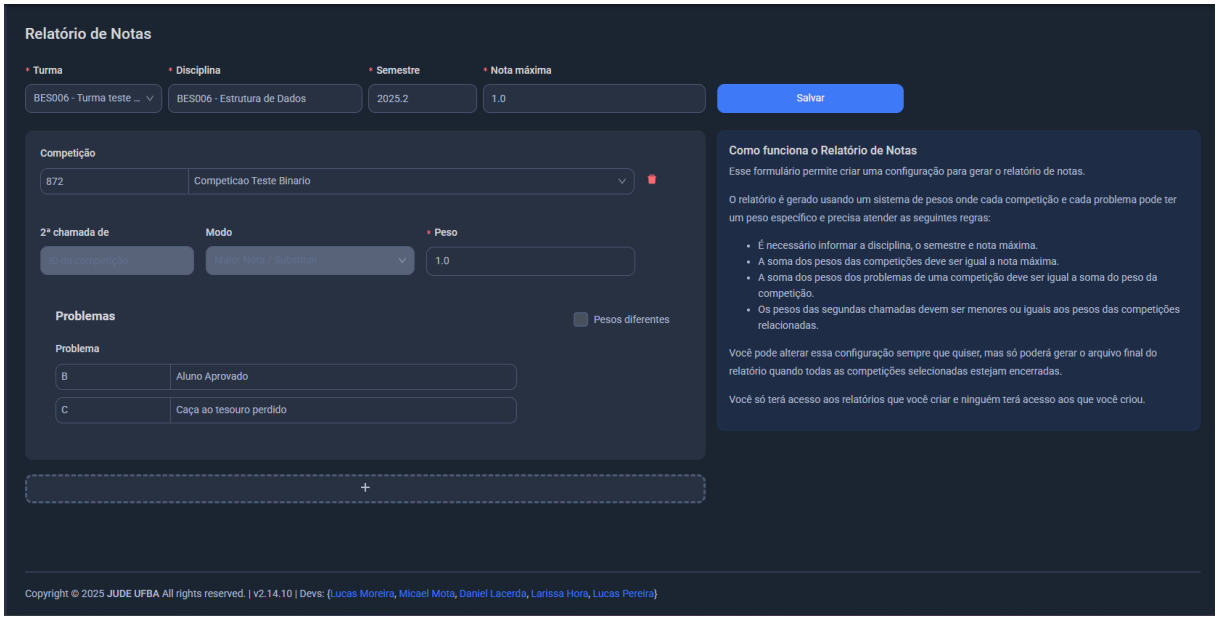


Figura 2.4: Cadastro de relatório de notas.

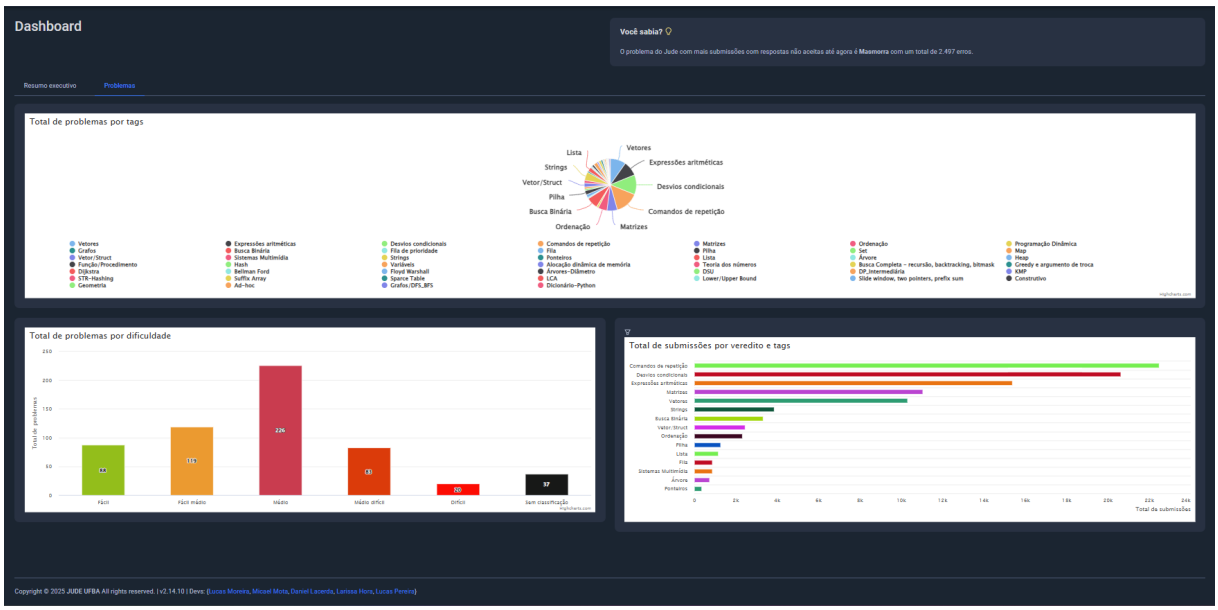


Figura 2.5: Dashboard do Jude.

2.2.1 Arquitetura do Sistema

O Jude é estruturado como um sistema distribuído em microserviços, no qual cada componente é encapsulado em um container Docker específico [7]. Essa separação de responsabilidades facilita o desenvolvimento, manutenção e escalabilidade do sistema como um todo (Figura 2.6).

Os principais componentes são:

- **Frontend (Interface Web)**: Desenvolvido em **React** [16], é responsável por fornecer a interface gráfica para alunos e professores. Permite a submissão de códigos, acesso ao histórico, visualização de resultados e gerenciamento de turmas. Os arquivos estáticos são servidos via **NGINX** [12].

- **Backend**: Construído com **Django** [6], expõe uma API RESTful para comunicação com o frontend e com os serviços auxiliares. Gerencia autenticação, submissões, competições, turmas e integração com o motor de execução de código e ferramentas de plágio.

- **Judge0**: O sistema de execução de código é baseado no **Judge0** [14], uma engine open source capaz de compilar e executar código em dezenas de linguagens diferentes, com isolamento via sandbox. São utilizados dois conjuntos distintos:

  - `server-ce` e `worker-ce`: para linguagens suportadas pela versão principal do Judge0.
  - `server-extra-ce` e `worker-extra-ce`: para linguagens adicionais suportadas pela versão estendida.

  Essa separação permite balancear a carga e ampliar a cobertura de linguagens conforme a necessidade.

- **Banco de Dados**: A base de dados é gerenciada pelo **PostgreSQL** [17], onde são armazenadas informações persistentes como usuários, registros de submissões, problemas, turmas e competições.

- **Cache e Fila de Tarefas**: O sistema utiliza o **Redis** [18] como mecanismo de cache e para controle assíncrono de tarefas. No modelo antigo, utilizava-se o **Celery** [1] para gerenciar a fila de submissões. No modelo atual, as requisições são feitas diretamente aos containers do Judge0 e o resultado é retornado via **webhook**, eliminando a necessidade de workers externos.

- **Plágio e Comparação de Códigos**: Para garantir a integridade acadêmica, o sistema inclui mecanismos de verificação de similaridade:

  - **JPlag**: realiza análise estrutural dos códigos.
  - **Sherlock**: aplica análise textual com foco em variações simples entre envios.

- **Servidor Web (NGINX)**: Atua como **proxy reverso**, roteando requisições entre o frontend e o backend, além de servir os arquivos estáticos da interface.

- **Docker e Orquestração**: Toda a infraestrutura é definida por meio de arquivos `docker-compose`, permitindo a rápida replicação do ambiente para novos desenvolvedores ou servidores.

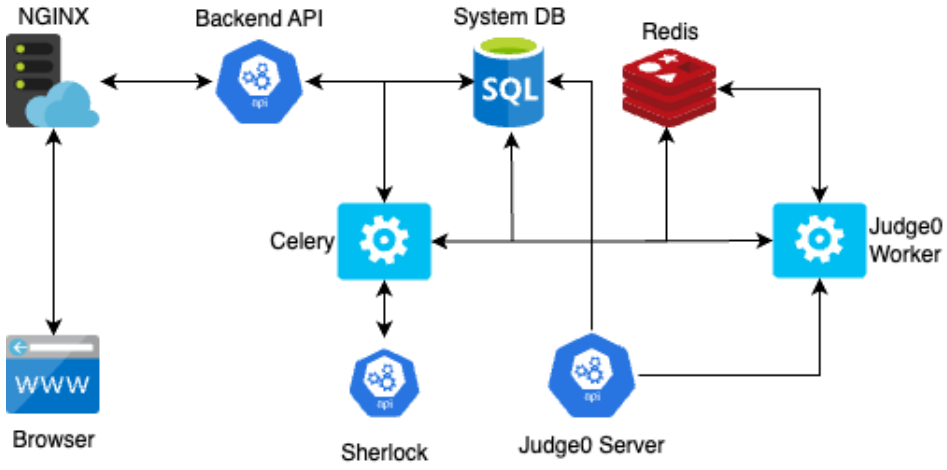


Figura 2.6: Fluxograma de funcionamento da infraestrutura do sistema Jude.

2.2.2 Benefícios da Arquitetura

A arquitetura baseada em microsserviços e containers traz diversos benefícios, especialmente no contexto educacional, onde a estabilidade, segurança e escalabilidade são cruciais:

- **Isolamento e segurança na execução dos códigos submetidos:** Cada submissão de código é executada dentro de um container Docker, garantindo que o código do aluno rode em um ambiente isolado, sem acesso ao sistema operacional do host. Isso protege o servidor contra comandos maliciosos, uso excessivo de recursos ou acesso indevido a arquivos do sistema. A separação por container também facilita a aplicação de limites de tempo, memória e CPU por execução.

- **Facilidade de manutenção e atualização individual dos serviços:** Com a adoção de microsserviços, cada componente do sistema (backend, frontend, executor, verificador de plágio, etc.) funciona de forma independente. Isso permite que atualizações, correções e melhorias sejam feitas em um serviço específico sem impactar os demais. Essa separação reduz riscos durante o desenvolvimento e simplifica a detecção de falhas.

- **Escalabilidade horizontal dos workers do Judge0:** O Judge0, responsável por executar os códigos submetidos, pode ser replicado facilmente. Em momentos de alta

demanda, novos containers “workers” podem ser instanciados para processar submissões simultaneamente, aumentando a capacidade do sistema sem reconfigurações complexas. Quando a carga diminui, os recursos podem ser liberados, otimizando o uso da infraestrutura.

- **Análise de plágio integrada ao fluxo de submissões, sem dependência de ferramentas externas:** Os verificadores de similaridade (como JPlag e Sherlock) foram integrados ao sistema como serviços internos, podendo ser ativados automaticamente após cada submissão. Isso evita que os professores precisem exportar códigos ou utilizar ferramentas externas, além de garantir que a análise seja feita de forma padronizada e segura, dentro do mesmo ambiente controlado.

Essa estrutura moderna garante que o sistema possa evoluir continuamente com base nas necessidades pedagógicas e técnicas da universidade, mantendo flexibilidade, desempenho e confiabilidade.

Capítulo 3

Metodologia

A condução deste trabalho seguiu uma abordagem prática, iterativa e incremental, fundamentada em princípios essenciais da engenharia de software e adaptada à realidade de projetos acadêmicos. O foco esteve na aplicação de boas práticas para análise de requisitos, organização de tarefas, desenvolvimento incremental e validação contínua das entregas [22, 4].

Nesta seção, são descritas as etapas percorridas durante o processo de desenvolvimento, os métodos utilizados para gerenciamento das tarefas, as ferramentas adotadas e os critérios que orientaram a implementação e validação das melhorias propostas para o sistema Jude.

Definição e Organização do Backlog

O ponto de partida do projeto consistiu na realização de uma reunião com o orientador responsável, com o objetivo de identificar as principais demandas do sistema, mapear funcionalidades desejadas, reconhecer falhas existentes e estabelecer prioridades iniciais.

No primeiro momento, o controle das tarefas era feito por meio de uma planilha no Google Sheets, gerenciada exclusivamente pelo orientador e com acesso eventual concedido aos desenvolvedores. No entanto, essa abordagem se mostrou limitada em termos de visibilidade, versionamento, rastreabilidade e colaboração.

Como alternativa, foi proposta e implementada a adoção do **Jira** [2], ferramenta amplamente utilizada na indústria de software para planejamento, acompanhamento de sprints e gerenciamento de backlog (Figura 3.1). Todas as tarefas identificadas na planilha foram migradas manualmente para o novo ambiente, que passou a centralizar a comunicação e o acompanhamento do progresso do projeto.

A estrutura de organização das tarefas no Jira seguiu a metodologia **Kanban**,

que consiste na visualização do fluxo de trabalho por meio de colunas como *To Do*, *In Progress* e *Done*. Esse modelo permite uma gestão mais visual e dinâmica das tarefas, facilitando o acompanhamento do status de cada item e a identificação de gargalos no desenvolvimento. A utilização do quadro Kanban também promoveu maior autonomia no gerenciamento do próprio fluxo de trabalho, mantendo alinhamento constante com o orientador.

A priorização das tarefas foi definida pelo orientador com base em critérios técnicos e pedagógicos, priorizando as entregas com maior impacto no uso do sistema por professores e alunos.

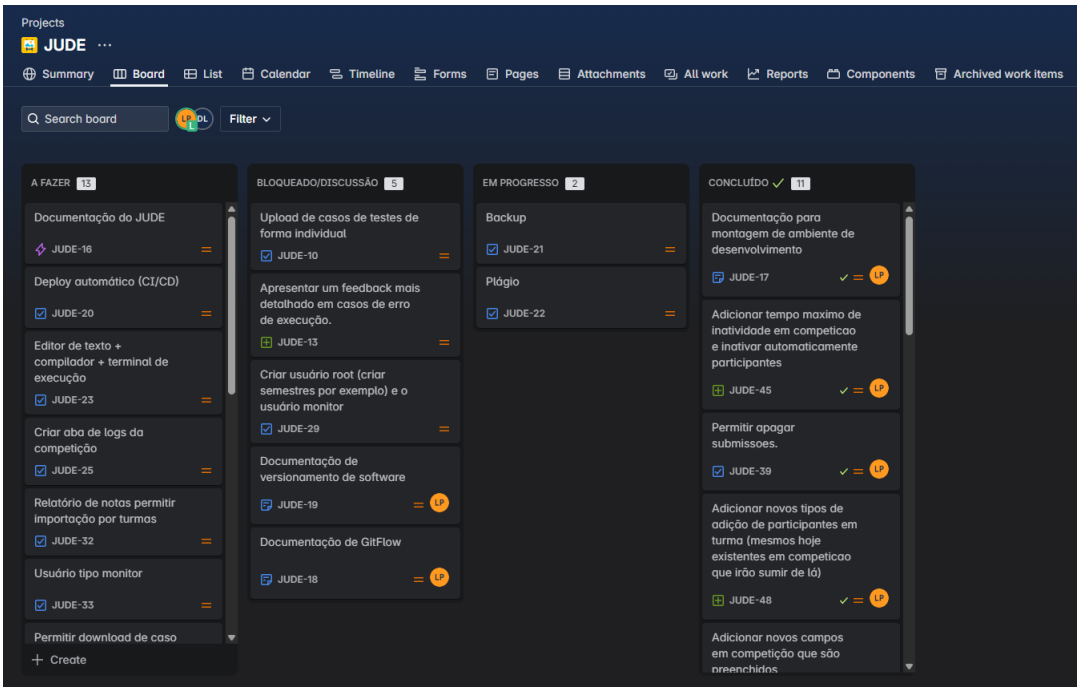


Figura 3.1: Quadro do projeto Jude no Jira.

Metodologia de Desenvolvimento

O processo de desenvolvimento adotado não seguiu um modelo formal como Scrum ou Kanban, mas manteve uma lógica sequencial, baseada em entrega contínua e ciclos de iteração curtos — abordagem comum em projetos de pequeno porte e escopo definido [22].

A dinâmica do trabalho foi centrada na execução de uma tarefa por vez, conforme a ordem de prioridade estabelecida no Jira. Após a conclusão de cada item, o desenvolvedor seguia para a próxima tarefa disponível, permitindo foco e controle sobre as entregas.

A comunicação com o orientador ocorreu em três momentos principais:

- Reunião inicial: definição do escopo, backlog e prioridades;

- Reuniões intermediárias: esclarecimento de dúvidas, validação técnica e realinhamento de escopo;
- Reunião final: entrega consolidada e orientações para validação em ambiente de homologação.

As etapas de trabalho podem ser resumidas da seguinte forma:

1. Levantamento e priorização do backlog;
2. Planejamento técnico individual por tarefa;
3. Desenvolvimento e testes locais;
4. Entrega no ambiente de homologação;
5. Teste funcional e validação com o orientador;
6. Atualização da tarefa no Jira e início da próxima.

Ferramentas e Ambiente de Desenvolvimento

Para garantir um ambiente produtivo, replicável e flexível, foi adotado um conjunto de ferramentas modernas compatíveis com o stack tecnológico do sistema Jude:

- **Visual Studio Code** – Editor de código com suporte a extensões, integração com Git e debugging Python;
- **WSL (Windows Subsystem for Linux)** – Utilizado como ambiente de desenvolvimento padrão, pela compatibilidade com comandos e ferramentas do Linux;
- **GitLab** – Repositório remoto e ferramenta de versionamento colaborativo [10];
- **Oracle VirtualBox** – Empregado nas fases finais do projeto para contornar incompatibilidades com containers do Judge0 que afetavam a execução no WSL;
- **Jira** – Plataforma de gerenciamento de tarefas, permitindo controle visual do fluxo, documentação de progresso e comunicação estruturada entre desenvolvedor e orientador.

Antes de iniciar a implementação das melhorias, optou-se por realizar uma tentativa de **atualização completa das dependências do projeto**, com o objetivo de modernizar bibliotecas e ferramentas utilizadas. Essa decisão foi motivada pela busca por

segurança, compatibilidade futura e alinhamento com boas práticas de desenvolvimento sustentável.

Entretanto, essa abordagem revelou-se extremamente complexa. Durante o processo, foram enfrentadas diversas dificuldades relacionadas à quebra de compatibilidade entre versões de bibliotecas, descontinuação de pacotes e mudanças no funcionamento interno de dependências críticas. Em especial, observou-se que bibliotecas centrais haviam sofrido alterações estruturais significativas, tornando a integração entre módulos altamente instável e gerando falhas em diferentes pontos da aplicação.

Embora tenha sido possível, após esforço significativo, construir uma versão funcional e moderna do projeto, concluiu-se que **o custo de manutenção e o risco associado à instabilidade do sistema não justificavam a mudança**, considerando o escopo do trabalho. Esse tipo de decisão reflete um princípio importante da engenharia de software: em alguns casos, preservar versões estáveis é preferível a buscar atualizações que alterem profundamente o ecossistema do sistema, especialmente quando não há infraestrutura contínua de suporte [22, 8].

Dessa forma, a estratégia foi reavaliada e optou-se por atualizar apenas as dependências necessárias para evitar incompatibilidades entre os componentes do projeto, em vez de realizar uma atualização completa para as versões mais recentes. Esse redirecionamento permitiu preservar a estabilidade do sistema, ao mesmo tempo em que viabilizou a implementação de melhorias estruturais e funcionais com base na versão vigente. Assim, os objetivos definidos puderam ser plenamente alcançados dentro do prazo disponível.

Critérios de Melhoria e Validação

Embora o backlog inicial tenha sido definido de forma colaborativa, durante o desenvolvimento foram identificadas novas oportunidades de aprimoramento que não estavam inicialmente planejadas. Essas melhorias surgiram da análise crítica do sistema em funcionamento, das dificuldades enfrentadas durante o desenvolvimento e da experiência prática do autor em engenharia de software [22, 8].

Entre os critérios utilizados para propor e implementar melhorias destacam-se:

- **Usabilidade da interface**: alterações que tornassem os fluxos mais intuitivos para professores e alunos;
- **Performance e limpeza do código**: remoção de dependências quebradas, padronização de lógicas duplicadas e melhoria da legibilidade;
- **Modularidade e manutenção**: criação de documentação de ambiente, suporte a debugging, simplificação do fluxo de submissões.

A validação das melhorias foi feita por meio de testes manuais conduzidos no ambiente local e em homologação. Esses testes consistiram em:

- Submissões simuladas por diferentes usuários;
- Validação da comunicação entre frontend, backend e containers do Judge0;
- Testes nos fluxos de chamada, cadastro de turmas, importação de alunos e análise de plágio;
- Verificação de retorno assíncrono via webhooks e atualização das submissões.

Não foram utilizados testes automatizados nem análise objetiva de desempenho (tempo de execução ou uso de recursos), uma vez que o foco do trabalho estava centrado na experiência do usuário final e na reestruturação da arquitetura do sistema.

Assim, a metodologia adotada garantiu não apenas o cumprimento das metas iniciais, mas também permitiu a identificação e resolução de pontos críticos da infraestrutura, ampliando o valor agregado do sistema de forma concreta e sustentável.

Capítulo 4

Desenvolvimento

O processo de desenvolvimento deste trabalho contemplou melhorias significativas tanto na estrutura técnica do sistema quanto na sua usabilidade para usuários finais e desenvolvedores. As intervenções realizadas foram organizadas em três frentes principais: melhorias no processo de desenvolvimento, desenvolvimento de novas funcionalidades e melhorias no sistema, e correções aplicadas a módulos existentes. Essa organização permite evidenciar o impacto técnico e funcional de cada grupo de alterações, bem como facilitar a compreensão da evolução do sistema Jude ao longo do projeto.

4.1 Melhorias voltadas ao processo de desenvolvimento

As melhorias descritas nesta subseção não impactam diretamente o produto final entregue ao usuário, mas foram fundamentais para tornar o sistema mais estável, facilitar a colaboração entre desenvolvedores e promover um ambiente mais ágil e sustentável para futuras evoluções.

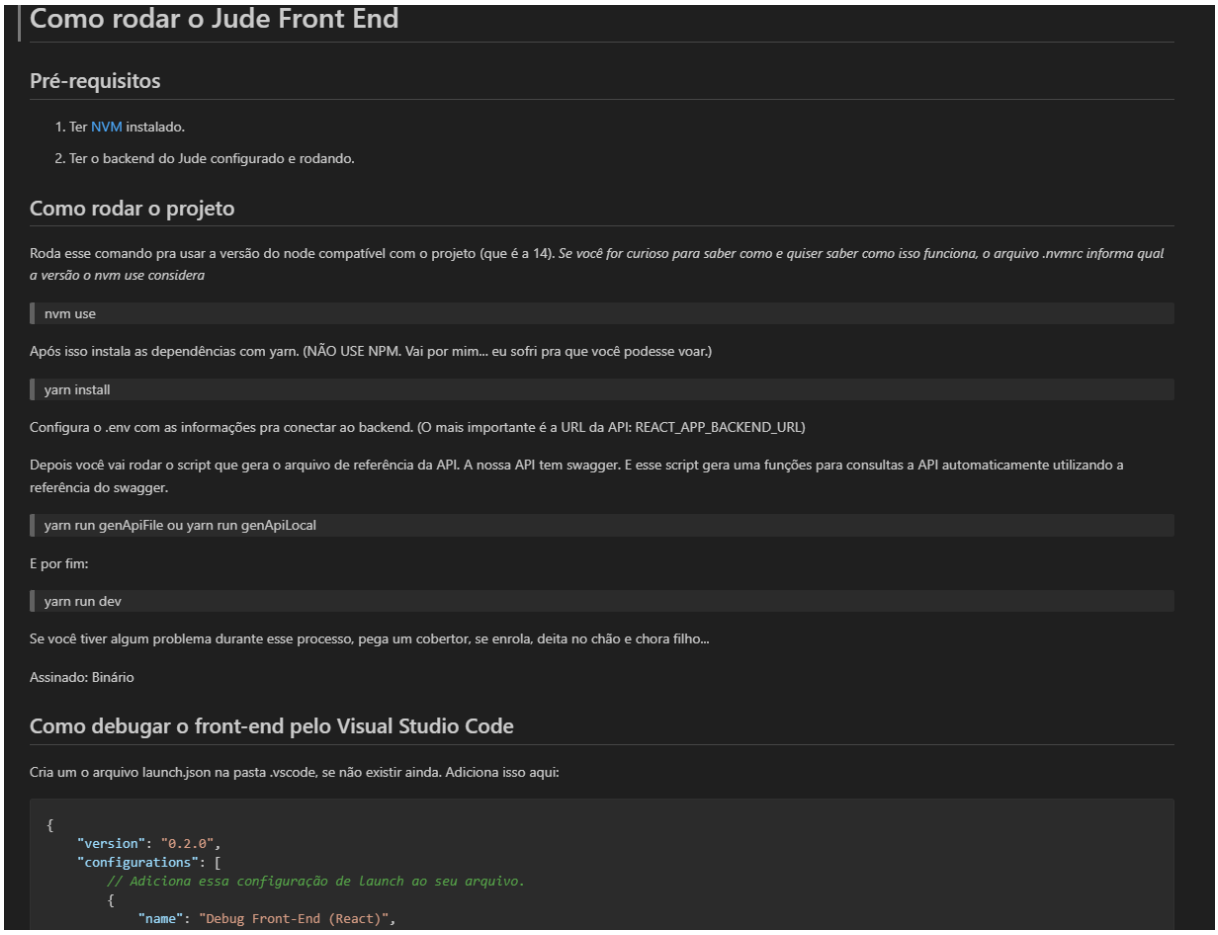
4.1.1 Documentação e configuração do ambiente

Logo no início do projeto, foi identificada a ausência de uma documentação clara sobre o processo de instalação e execução do sistema, o que dificultava o onboarding de novos desenvolvedores. Diversas dependências estavam quebradas ou em versões incompatíveis.

Para solucionar esses problemas, foram tomadas as seguintes medidas:

- Criação de documentação no arquivo README dos projetos frontend e backend, com instruções detalhadas para montagem do ambiente, execução e debugging (Figura 4.1);
- Correção de dependências quebradas e conflitos de versão;

- Uso da ferramenta *depcheck* para remoção de dependências obsoletas.



Como rodar o Jude Front End

Pré-requisitos

1. Ter NVM instalado.
2. Ter o backend do Jude configurado e rodando.

Como rodar o projeto

Roda esse comando pra usar a versão do node compatível com o projeto (que é a 14). *Se você for curioso para saber como e quiser saber como isso funciona, o arquivo .nvmrc informa qual a versão o nvm use considera*

```
nvm use
```

Após isso instala as dependências com yarn. (NÃO USE NPM. Vai por mim... eu sofri pra que você podesse voar.)

```
yarn install
```

Configura o .env com as informações pra conectar ao backend. (O mais importante é a URL da API: REACT_APP_BACKEND_URL)

Depois você vai rodar o script que gera o arquivo de referência da API. A nossa API tem swagger. E esse script gera uma funções para consultas a API automaticamente utilizando a referência do swagger.

```
yarn run genApiFile ou yarn run genApiLocal
```

E por fim:

```
yarn run dev
```

Se você tiver algum problema durante esse processo, pega um cobertor, se enrola, deita no chão e chora filho...

Assinado: Binário

Como debugar o front-end pelo Visual Studio Code

Cria um o arquivo launch.json na pasta .vscode, se não existir ainda. Adiciona isso aqui:

```
{
    "version": "0.2.0",
    "configurations": [
        // Adiciona essa configuração de launch ao seu arquivo.
        {
            "name": "Debug Front-End (React)",
```

Figura 4.1: README Frontend.

4.1.2 Implementação de debugging no backend

Para viabilizar o uso de *breakpoints* e facilitar o rastreamento do fluxo de execução, foi implementado o suporte ao módulo `debugpy`, integrando o backend com o Visual Studio Code para depuração em tempo real.

4.1.3 Correção da documentação Swagger

O projeto contava com integração Swagger para visualização e teste da API REST, mas a funcionalidade estava inoperante devido à configuração inadequada da autenticação nas rotas protegidas. A correção permitiu o uso pleno da ferramenta durante o desenvolvimento (Figura 4.2).

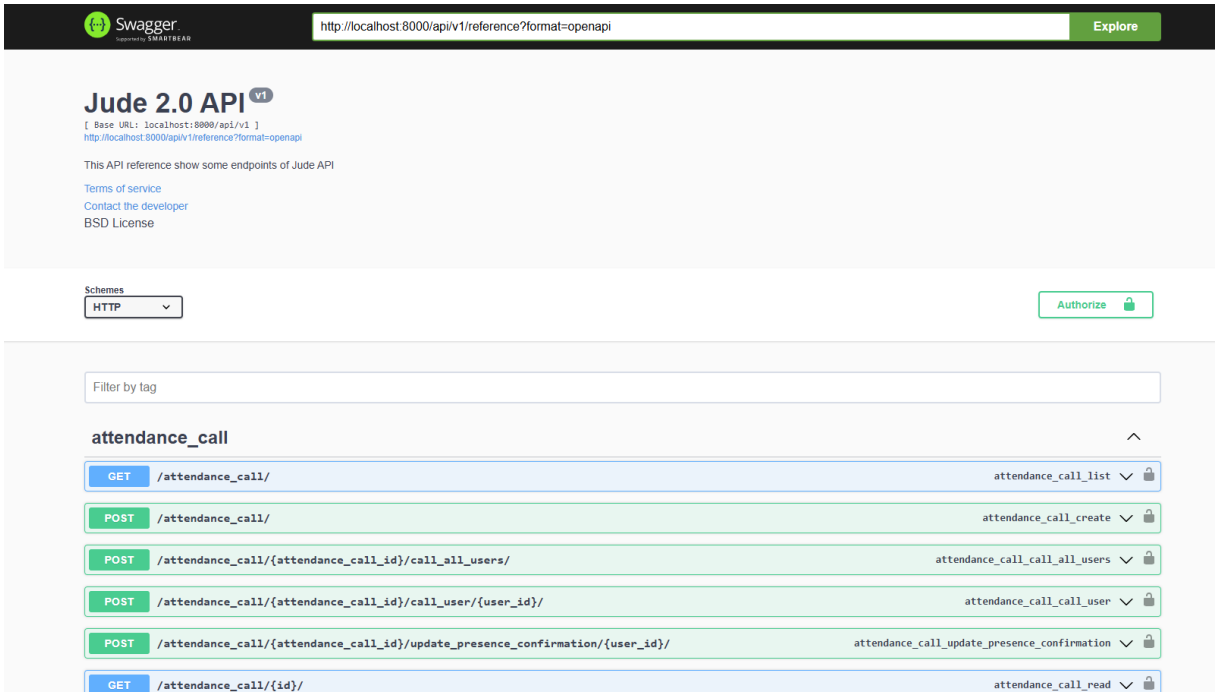


Figura 4.2: Documentação Swagger do Jude.

4.2 Desenvolvimento de novas funcionalidades e melhorias no sistema

Esta seção apresenta as funcionalidades e melhorias adicionadas ao sistema durante o desenvolvimento. Elas ampliam o conjunto de capacidades da plataforma, melhoram a experiência de uso e fortalecem a arquitetura técnica do Jude.

4.2.1 Integração entre turmas e competições

Foi implementada uma integração direta entre turmas e competições (Figura 4.3). A partir dessa modificação:

- Toda competição deve estar vinculada a uma turma;
- Os participantes da competição são definidos automaticamente com base nos integrantes da turma;
- Alunos adicionados à turma passam a integrar todas as competições vinculadas;
- Alunos removidos da turma:
  - são excluídos da competição, se não tiverem submissões;
  - são marcados como inativos, se já tiverem participado.

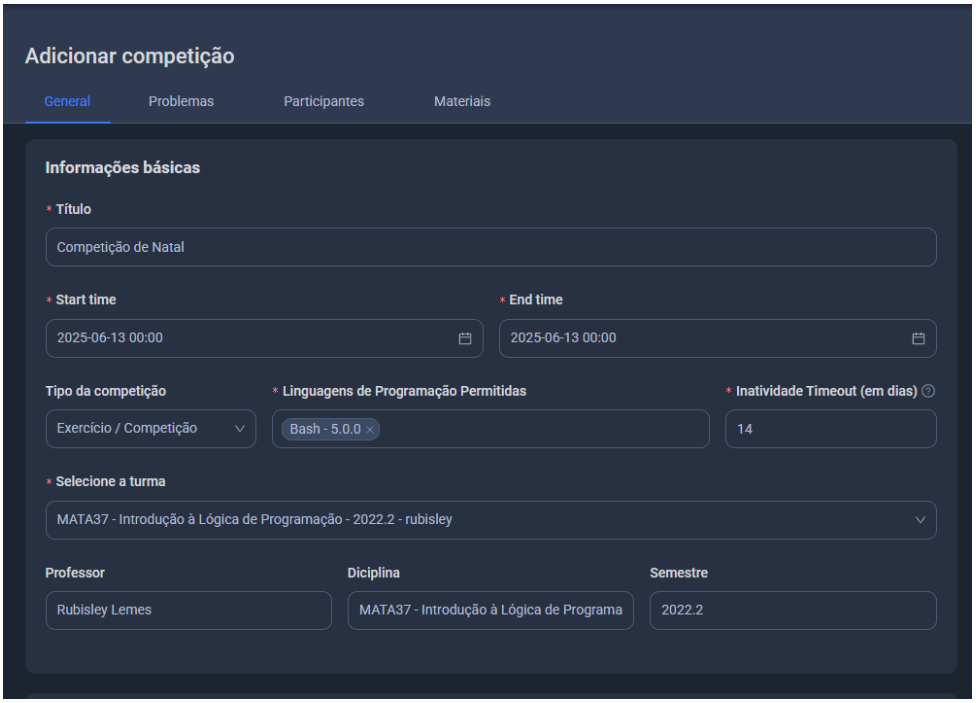


Figura 4.3: Criação de competição com integração entre turma.

4.2.2 Ampliação dos métodos de adição de alunos

Anteriormente, alunos só podiam ser adicionados a uma turma por meio da importação de arquivos do SIAC e apenas na edição da turma. Com a melhoria, foram adicionados os seguintes métodos (Figura 4.4):

- Inclusão manual por *username*;
- Inclusão em lote (lista de *usernames* separados por ponto e vírgula);
- Importação de alunos a partir de uma competição ou de outra turma;
- Inclusão de alunos já na criação da turma.

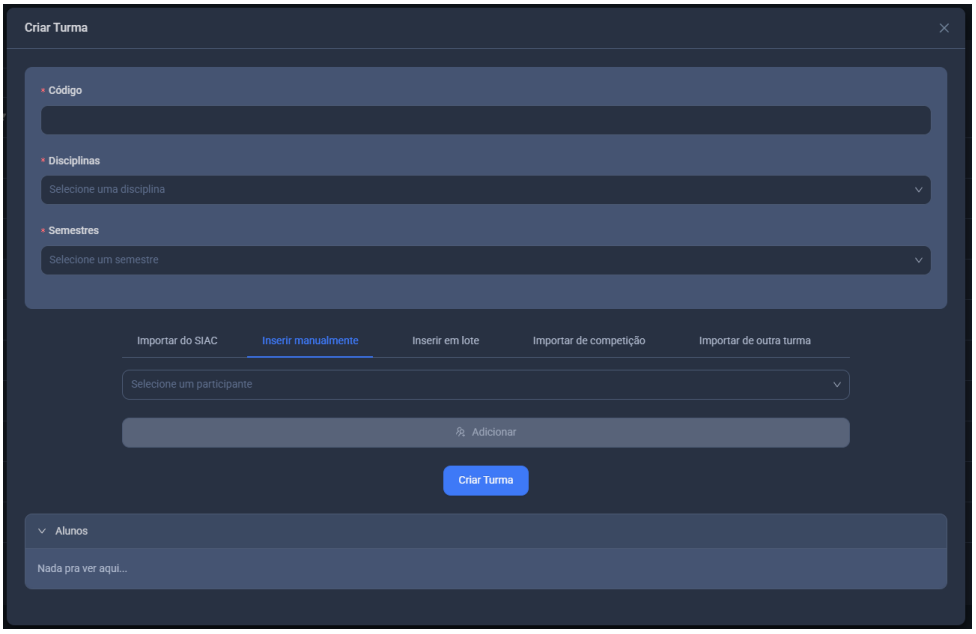


Figura 4.4: Criação de turma com múltiplas possibilidades de adição de alunos e layout reformulado.

4.2.3 Sistema de chamadas com WebSockets

Foi implementado um sistema de chamadas em tempo real, permitindo que professores iniciem chamadas para uma turma e que os alunos recebam notificações instantâneas para confirmar presença (Figuras 4.6, 4.7 e 4.8). A comunicação foi construída com uso de WebSockets, onde cada aluno se conecta aos canais das turmas que frequenta. A Figura 4.5 apresenta um fluxograma que demonstra o funcionamento do processo.

- Notificação automática ao aluno ao iniciar a chamada;
- Confirmação de presença via clique;
- Reenvio de chamada em lote ou individual;
- Possibilidade de o professor forçar presença ou ausência.

O professor pode ainda definir um tempo limite de resposta, comportamento visível na Figura 4.7. Após esse tempo, os alunos que não confirmarem presença são automaticamente marcados como ausentes, sendo possível apenas ao professor alterar seus status posteriormente.

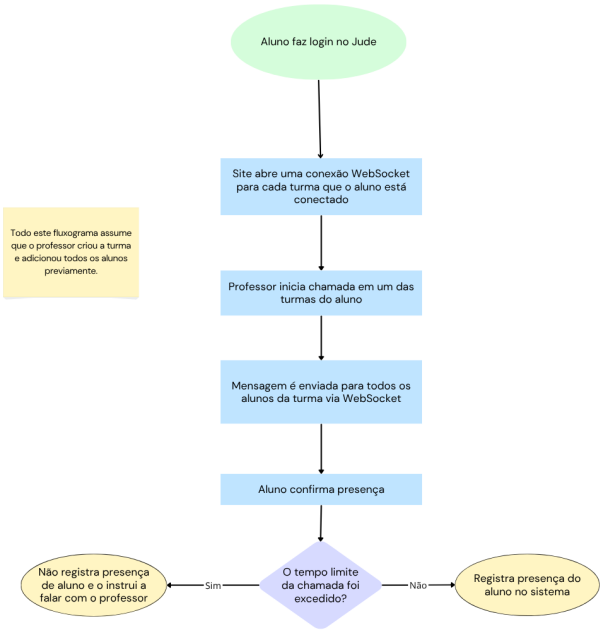


Figura 4.5: Fluxograma de funcionamento do sistema de chamadas.

Figura 4.6: Listagem de turmas com botão de ação para chamadas.

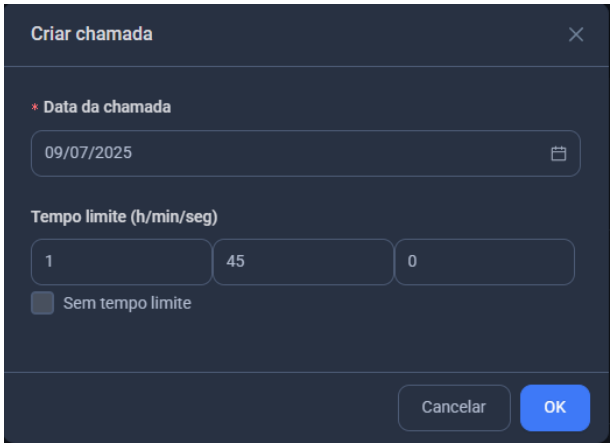


Figura 4.7: Criação de chamada com tempo limite.

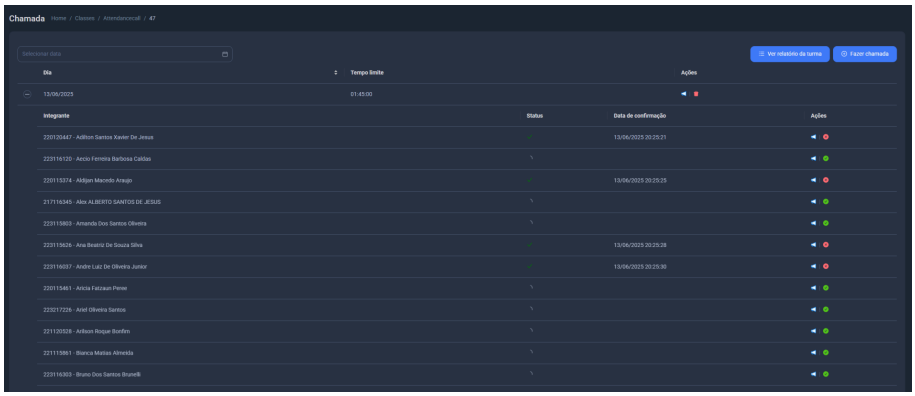


Figura 4.8: Visualização de uma chamada em andamento.

Relatório de presença e exportação

Complementando a funcionalidade, foi desenvolvido um relatório de presença consolidado por turma (Figura 4.9). O relatório apresenta:

- Nome do aluno;
- Número de presenças;
- Número de faltas;
- Histórico individual de chamadas.

Além da visualização, é possível exportar o relatório em formato CSV.

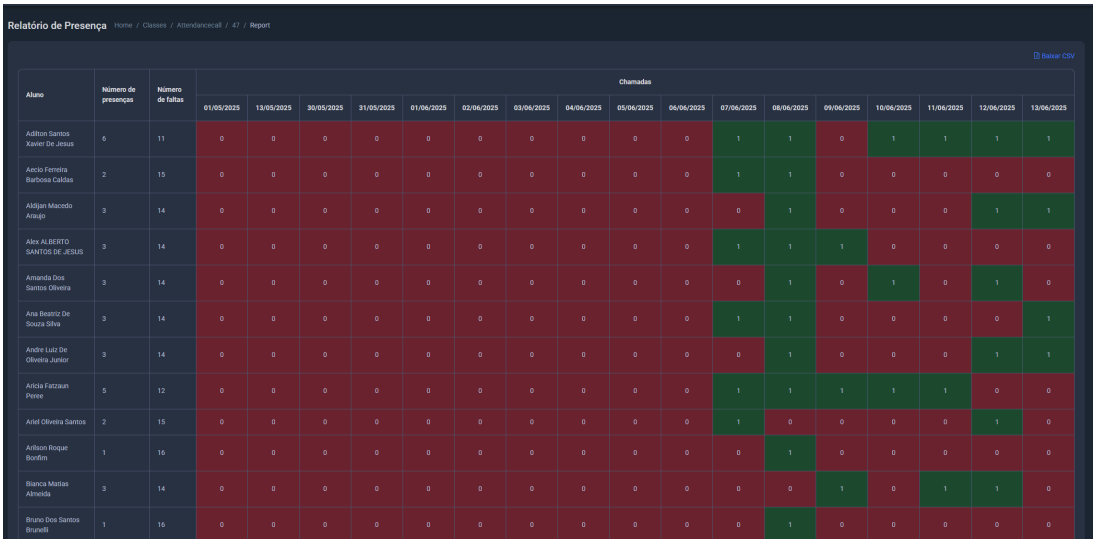


Figura 4.9: Relatório de chamadas.

4.2.4 Substituição do Celery por Webhooks

O sistema utilizava Celery para controle assíncrono de submissões. Contudo, como o Judge0 já trabalha com Webhooks, o Celery tornou-se redundante. Sua remoção trouxe:

- Redução de complexidade;
- Diminuição de dependências;
- Fluxo mais direto e moderno baseado em *callback*.

Essas mudanças resultaram em um novo fluxo de funcionamento, mais simples e eficiente, conforme ilustrado na Figura 4.10.

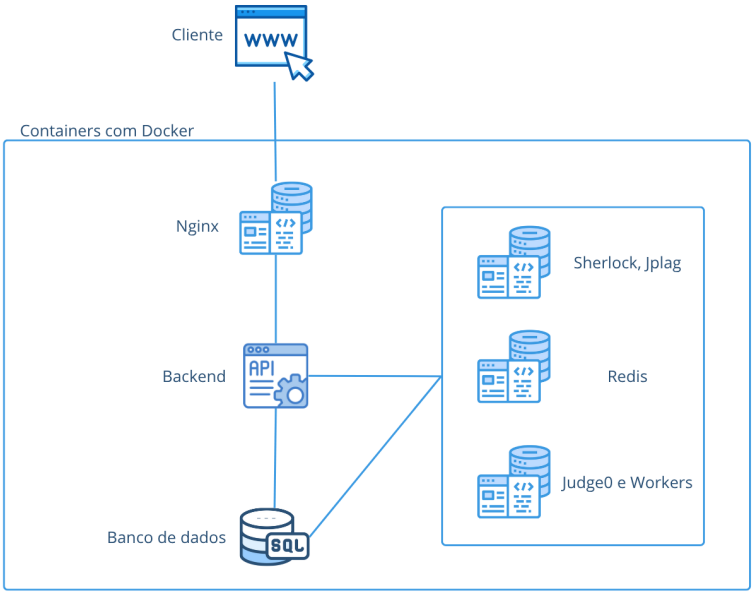


Figura 4.10: Novo fluxograma do Jude após remoção do Celery.

4.2.5 Relatório de notas

O módulo de relatório de notas foi aprimorado com a adição do campo de seleção da turma durante sua criação (Figura 4.11). A partir dessa mudança:

- Os campos **semestre** e **disciplina** passaram a ser preenchidos automaticamente com base na turma selecionada;
- O campo de **competições** agora é populado apenas com as competições vinculadas à turma, ao invés de listar todas as existentes no sistema.

Essa alteração tornou o preenchimento do formulário mais rápido e preciso, além de reduzir significativamente o tempo de carregamento das informações.

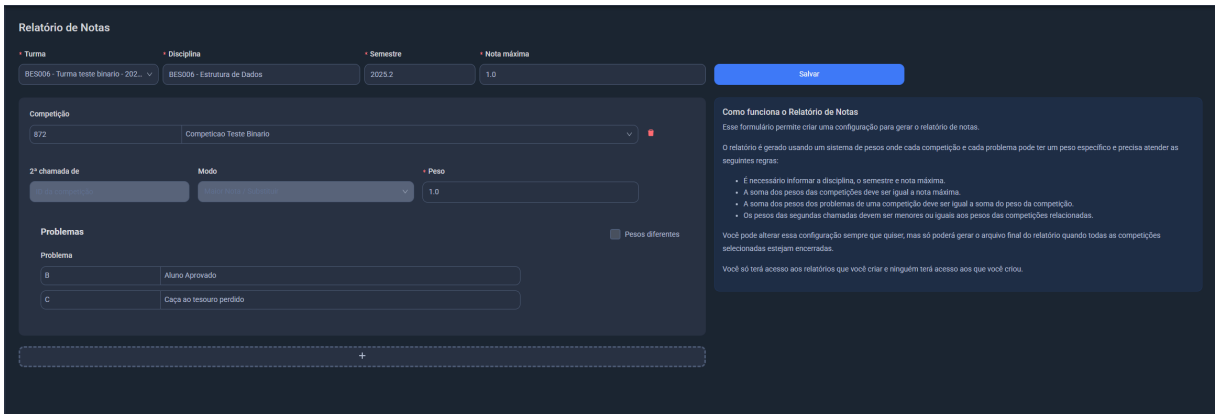


Figura 4.11: Novo preenchimento em relatório de notas.

4.2.6 Implementação de bloqueio de participantes

Dada a forte vinculação entre turmas e competições no sistema, foi implementada uma funcionalidade que permite ao professor **bloquear participantes** de uma competição de forma manual (Figuras 4.12 e 4.13).

Essa ação é útil em situações específicas em que o professor deseja restringir a participação de determinados alunos, evitando inconsistências ou usos indevidos da plataforma, como, por exemplo, em uma segunda chamada de prova, na qual o professor bloqueia todos os alunos que não fazem parte do grupo autorizado a realizar a avaliação.

Uma vez bloqueado, o aluno permanece vinculado à competição, mas é impedido de realizar novas submissões, até que seu status seja alterada novamente pelo professor.

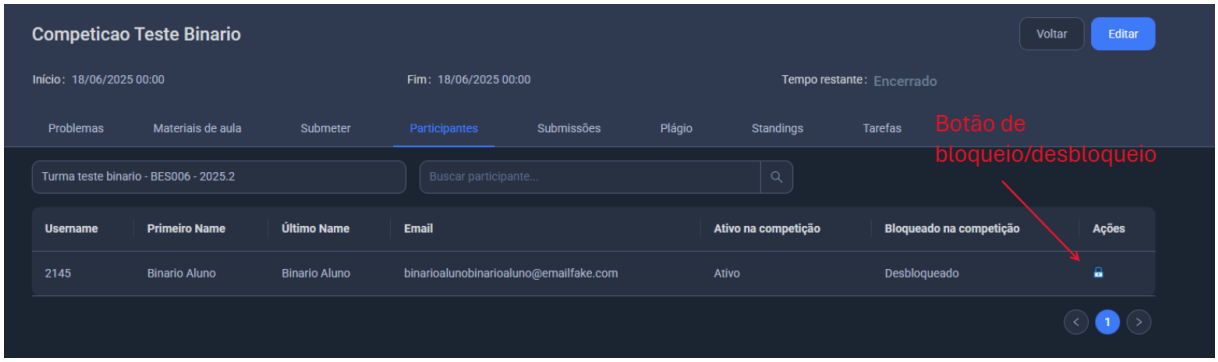


Figura 4.12: Usuário não bloqueado e botão de ação para bloquear/desbloquear usuário da competição.

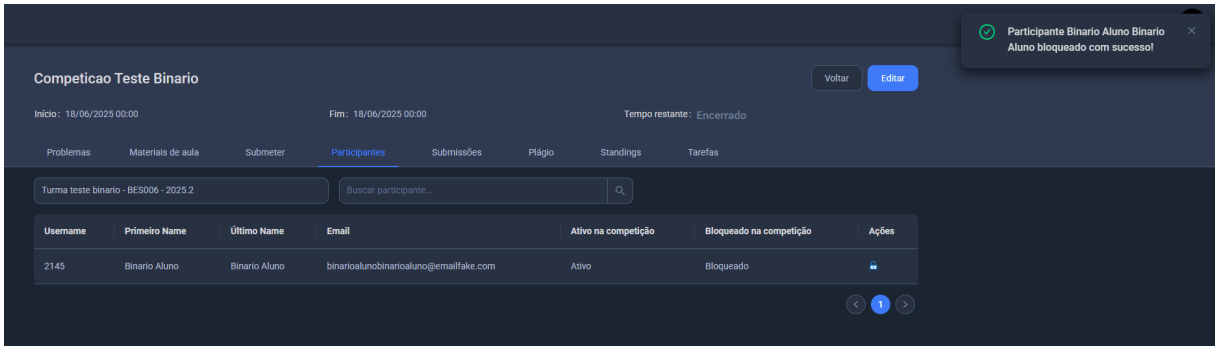


Figura 4.13: Usuário bloqueado.

4.2.7 Outras funcionalidades e melhorias

- Controle de inatividade automática em competições, com inativação de alunos sem submissões após período estipulado;
- Exibição de dados da turma diretamente na tela de competições;

- Adição do campo “ano” à entidade Semestre (Figura 4.14);
- Inclusão da funcionalidade de exclusão de submissões por professores (Figuras 4.15 e 4.16);
- Ordenação dos relatórios de notas por data de criação;

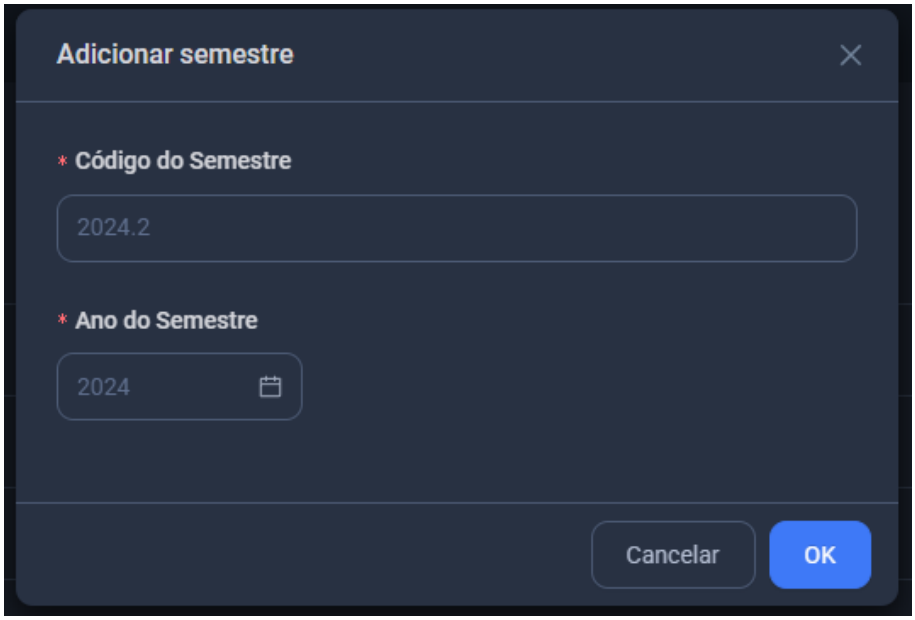


Figura 4.14: Criação de novo semestre.

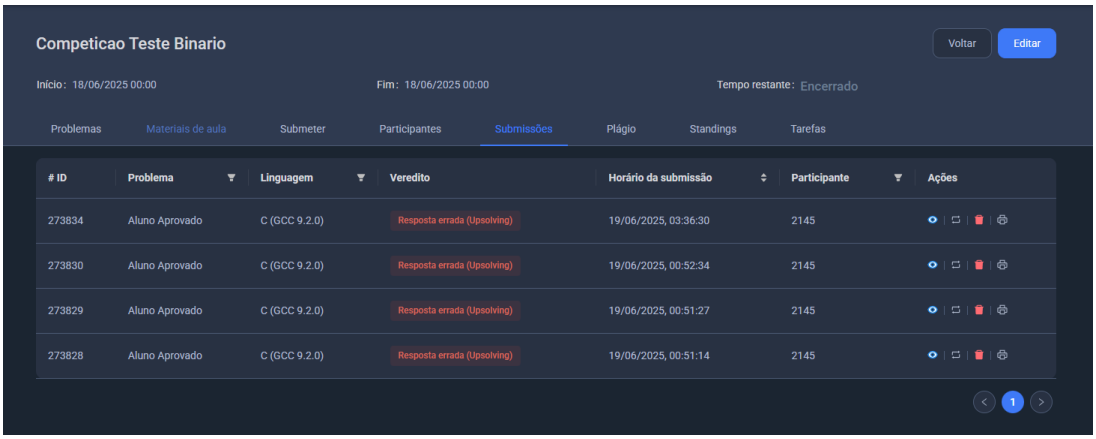


Figura 4.15: Funcionalidade para apagar submissões.

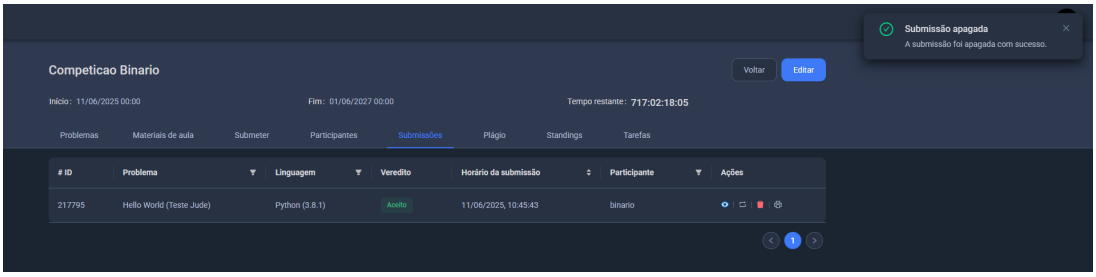


Figura 4.16: Mensagem de sucesso após apagar submissão.

4.3 Correções aplicadas ao sistema

Diversos comportamentos inadequados foram identificados e corrigidos, melhorando a robustez e a experiência do usuário:

- Corrigido erro ao editar usuários com ponto no *username*;
- Melhoria na tela de cadastro de usuários para tratamento de erros;
- Correção na listagem de participantes com e-mails repetidos;
- Problema de visualização de código em tarefas de competição resolvido;
- Otimização do *queryset* de competições, reduzindo tempo de carregamento pela metade;
- Correção em filtros de listas que travavam ao acessar páginas superiores à primeira;
- Correção na listagem de turmas, que não exibia todas corretamente;
- Professores agora podem visualizar todas as turmas, mas apenas o criador pode editá-las ou deletá-las.

4.4 Proposta de versionamento do sistema

Embora o sistema já estivesse na versão **4.0**, considerando as melhorias e correções realizadas até a data de elaboração deste documento, optou-se por adotar a versão **2.14.9** como referência. Essa escolha está alinhada ao modelo de *versionamento semântico*, que oferece uma forma clara e padronizada de indicar a natureza das alterações em cada nova versão.

O versionamento semântico segue o formato **MAJOR.MINOR.PATCH**, no qual:

- **MAJOR** é incrementado em mudanças incompatíveis com versões anteriores (quebra de retrocompatibilidade);
- **MINOR** em adições de funcionalidades retrocompatíveis;
- **PATCH** em correções de erros que não afetam a compatibilidade.

A escolha do número **2.14.9** foi feita com base nos seguintes critérios: o valor **2** representa a segunda grande versão do sistema, já que, até o momento, apenas uma mudança significativa comprometeu a retrocompatibilidade — a substituição da tecnologia do frontend para *React*. Considerando a adoção do modelo semântico neste ponto, foi realizado um *fresh start*, tratando a transição como o início da versão 2.0. A partir disso, foram contabilizadas **14 funcionalidades e melhorias** (incremento no campo **MINOR**) e **9 correções** (incremento no campo **PATCH**) implementadas até a data deste documento.

4.4.1 Atualização Automática de Versão

Com o avanço do projeto, foi implementado um mecanismo de atualização automática de versão com base no padrão de versionamento semântico (`MAJOR.MINOR.PATCH`). Esse recurso garante que cada nova versão do sistema represente, de forma clara e rastreável, as modificações aplicadas ao código, evitando inconsistências entre o estado real do sistema e sua numeração.

Para isso, foi criado um repositório central exclusivo para controle de versão, contendo um *script* responsável por gerar a nova versão com base nos commits mais recentes. Esse script interpreta as anotações de cada *commit* — classificando-os como mudanças de funcionalidade, correções ou quebras de compatibilidade — e, a partir disso, calcula e registra o novo número de versão.

A integração entre esse repositório de versionamento e os repositórios do *frontend* e *backend* foi realizada por meio de *webhooks*. Sempre que ocorre um *merge* na `branch` principal de qualquer um dos repositórios, o *webhook* é acionado automaticamente, executando o script de versionamento. Isso garante que a versão seja atualizada de forma contínua e sem intervenção manual, refletindo o estado mais recente do sistema.

A Figura 4.17 ilustra a estrutura do repositório de versionamento, onde estão definidos os scripts e arquivos que controlam o histórico de versões. Já as Figuras Figura 4.18 e Figura 4.19 mostram a configuração dos pipelines nos repositórios do *frontend* e *backend*, evidenciando os gatilhos que disparam a atualização de versão após os merges nas branches principais.

Para garantir consistência e automatizar a classificação semântica dos commits, foram adicionadas ferramentas de *pre-commit* nos dois repositórios. Essas ferramentas

validam se as mensagens de *commit* seguem o padrão **Conventional Commits**, evitando que alterações com mensagens fora do padrão sejam registradas.

O *Conventional Commits* é uma convenção que define um formato padronizado para as mensagens de commit, permitindo que ferramentas automatizadas interpretem corretamente o tipo da alteração. A estrutura básica de um commit segue o modelo:

```
tipo: descrição.
```

Exemplos válidos:

- `feat: adicionar sistema de chamadas em tempo real.`
- `fix: corrigir erro na listagem de participantes.`
- `chore: atualizar dependências do projeto.`

Essa padronização torna o histórico de mudanças mais legível e é fundamental para que o script de versionamento consiga identificar corretamente o tipo de mudança (MAJOR, MINOR ou PATCH). Com isso, o ciclo de versionamento automático se mantém estável, previsível e alinhado com boas práticas modernas de desenvolvimento de software.

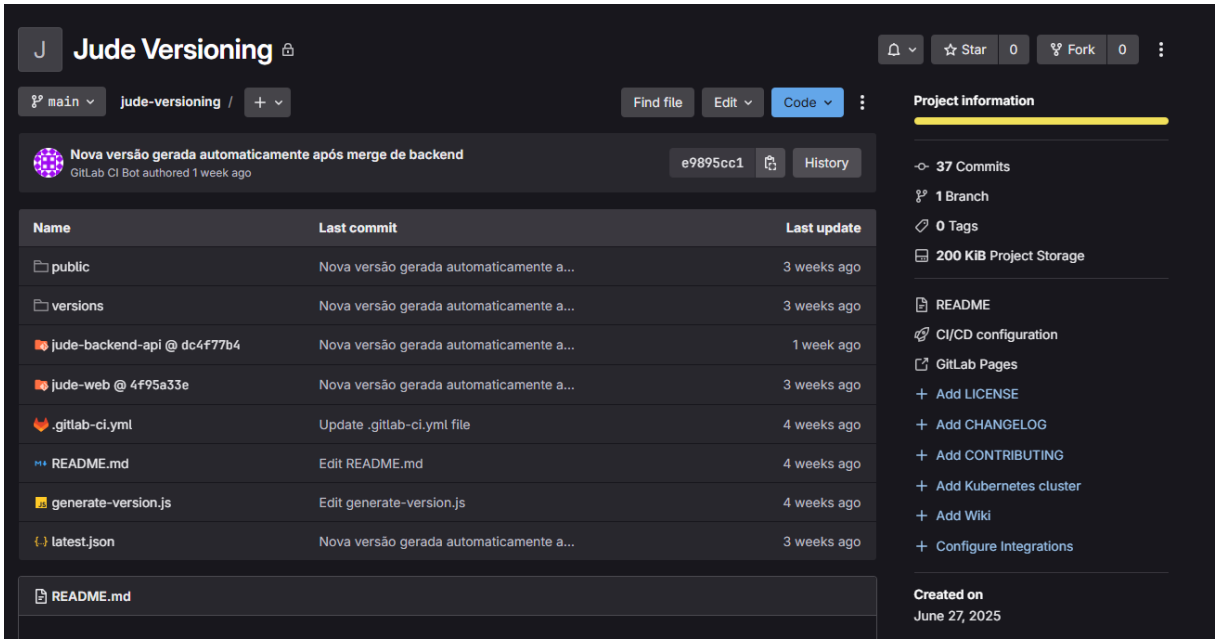


Figura 4.17: Repositório de versionamento com script automatizado.

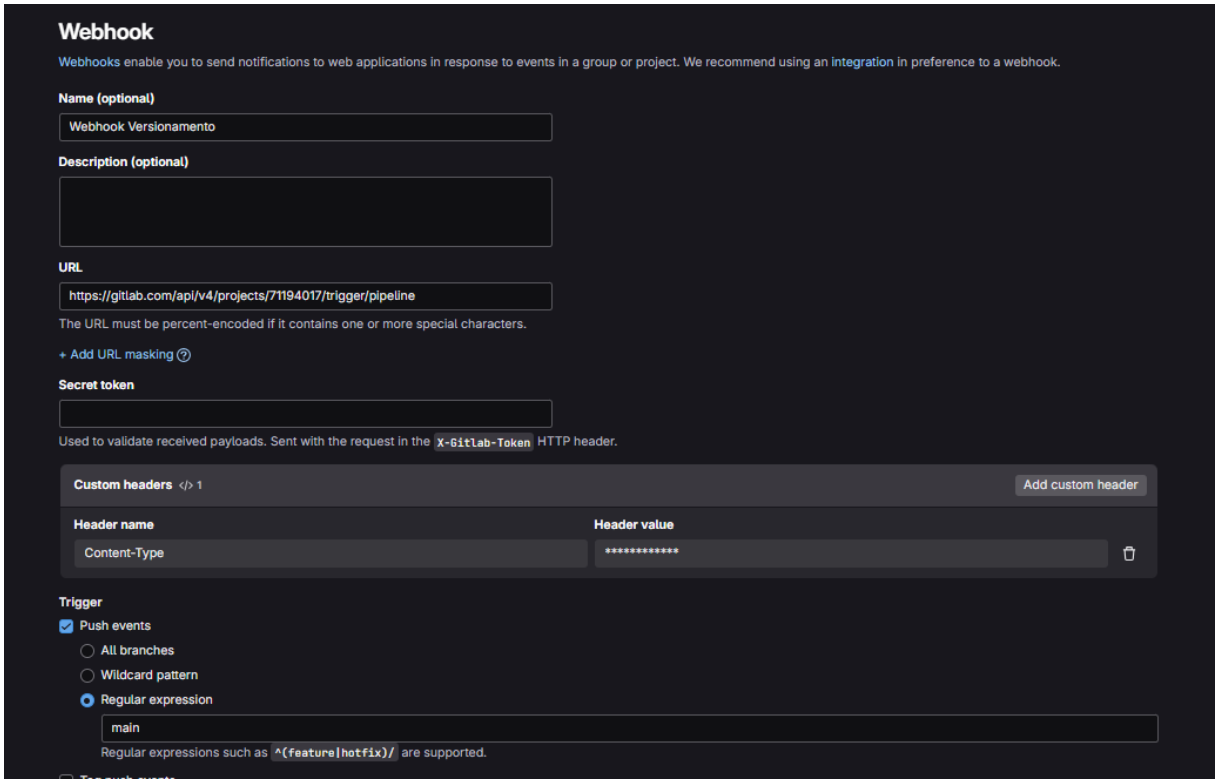


Figura 4.18: Configuração do pipeline de versionamento no repositório.

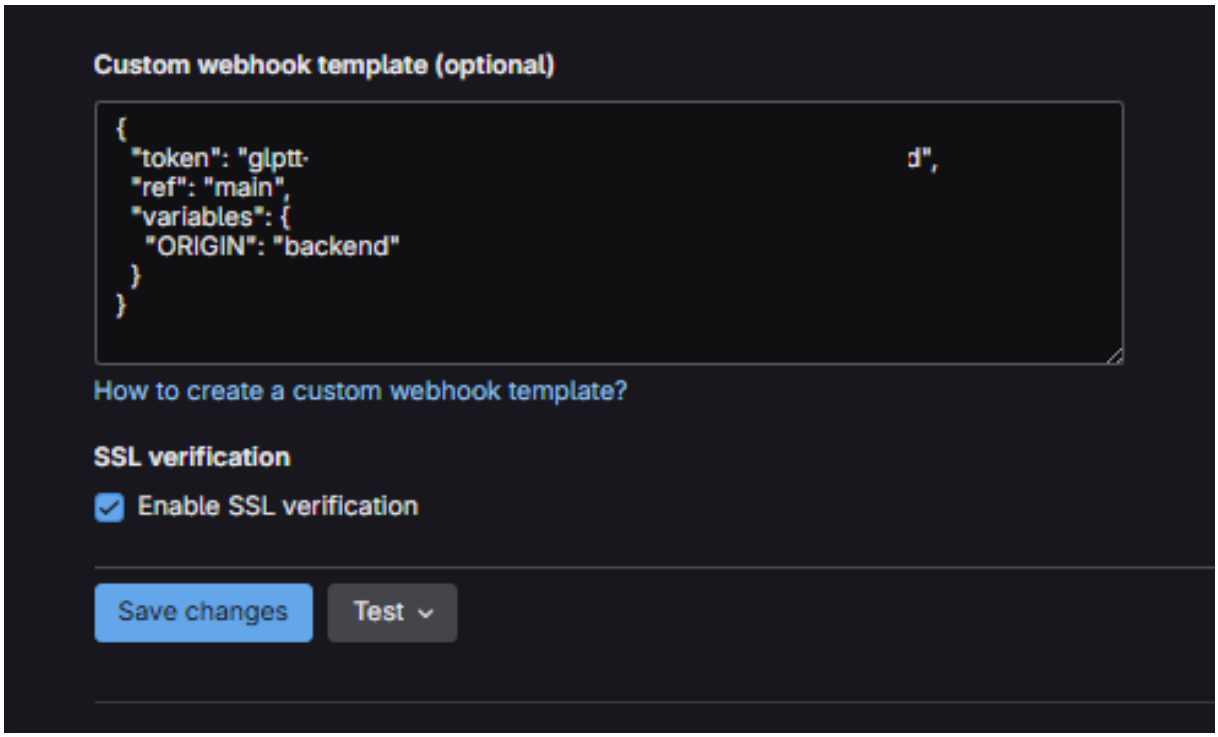


Figura 4.19: Configuração do pipeline de versionamento no repositório.

Capítulo 5

Resultados e discussões

Para a avaliação das melhorias implementadas na nova versão do sistema Jude, foi elaborado um formulário online destinado aos professores que utilizam o sistema. O objetivo desse formulário foi coletar a percepção dos docentes sobre as novas funcionalidades e correções introduzidas.

O formulário abordou questões relacionadas às funcionalidades implementadas, como o sistema de chamadas, cadastro de turmas, controle de presença e a remoção do Celery, entre outras. Além disso, foram avaliadas melhorias na interface, desempenho e usabilidade do sistema.

A seguir, apresento uma tabela com as perguntas realizadas no formulário, conforme enviadas aos docentes:

Tabela 5.1: Questionário de avaliação da nova versão do JUDE pelos docentes.

Variável	**Questão**
Q1	No sistema de chamadas o aluno só recebe a notificação se ele estiver logado no momento em que a chamada for disparada. Tendo em mente que é possível reenviar a requisição manualmente para toda a turma e para algum aluno indivualmente, este comportamento atende a sua expectativa?
Q2	No sistema de chamadas é possível atribuir presença ou ausência manualmente para os alunos. Esta parte da funcionalidade atendeu às suas expectativas? A funcionalidade de forçar presença, reenvio e tempo limite atendeu às suas expectativas?
Q3	No sistema de chamadas é possível reenviar a notificação de chamada para o aluno. Esta parte da funcionalidade atendeu às suas expectativas?

Continua na próxima página

Tabela 5.1 – Continuação da Tabela anterior

Variável	**Questão**
Q4	No sistema de chamadas, é possível definir um tempo limite para que os alunos respondam à notificação. Após o término desse tempo, não é mais possível enviar uma resposta. Essa funcionalidade atendeu às suas expectativas?
Q5	Como você avalia o relatório de chamadas?
Q6	No relatório de chamadas é possível exportar para CSV. Essa funcionalidade atendeu às suas expectativas?
Q7	Nesta atualização, a tela de cadastro de turmas foi completamente reformulada, com foco em usabilidade e design. O novo visual atendeu às suas expectativas?
Q8	Agora, além da importação de integrantes via arquivo, o cadastro de turmas permite adicionar usuários manualmente, um por um. Essa nova funcionalidade atendeu à sua expectativa?
Q9	Agora, além da importação de integrantes via arquivo, o cadastro de turmas permite adicionar usuários em lote utilizando os usernames separado por ponto e vírgula. Essa nova funcionalidade atendeu à sua expectativa?
Q10	Agora, além da importação de integrantes via arquivo, o cadastro de turmas permite importar usuários de uma competição. Essa nova funcionalidade atendeu à sua expectativa?
Q11	Agora, além da importação de integrantes via arquivo, o cadastro de turmas permite importar usuários de outra turma. Essa nova funcionalidade atendeu à sua expectativa?
Q12	A partir desta atualização, agora é possível adicionar participantes a uma turma já no momento da sua criação — antes, isso só era possível na etapa de edição. Essa melhoria atendeu à sua expectativa?
Q13	A partir desta atualização, passou a ser obrigatório vincular uma turma ao cadastrar uma competição. Os participantes são definidos automaticamente com base nos alunos da turma selecionada. Essa mudança foi positiva para você?
Q14	Ao atualizar os participantes de uma turma, os participantes de todas as competições são atualizados automaticamente. Essa funcionalidade atendeu às suas expectativas?
Q15	O tempo de inatividade atendeu às suas expectativas?

Continua na próxima página

Tabela 5.1 – Continuação da Tabela anterior

Variável	**Questão**
Q16	Um usuário inativo é reativado ao fazer uma submissão. Esse comportamento atende às suas expectativas?
Q17	O bloqueio de usuários atendeu às suas expectativas?
Q18	Um usuário bloqueado não pode visualizar a competição. Este comportamento atende às suas expectativas?
Q19	A remoção do Celery diminui a complexidade e tamanho do projeto e possívelmente diminui o tempo de julgamento das requisições. O quão importante você considera essa alteração?
Q20	A partir desta atualização, tornou-se obrigatório vincular uma turma ao gerar um relatório de notas. Essa mudança foi implementada com o objetivo de melhorar a performance do sistema. Essa mudança foi positiva para você?
Q21	Agora as informações de disciplina e semestre são preenchidas automaticamente de acordo com a turma. Essa funcionalidade atende às suas expectativas?
Q22	Nesta atualização, foi corrigido um erro que exibia indevidamente uma notificação de horário na tela inicial do Jude. Essa correção teve um impacto positivo para você?
Q23	Foi corrigido um erro que impedia a edição de usuários com ponto (.) no nome de usuário (username). Essa correção atendeu às suas expectativas?
Q24	Foi corrigido um erro que não permitia que o código da tarefa de impressão fosse visualizado. Essa correção atendeu às suas expectativas?
Q25	O queryset de competições foi alterado para diminuir tempo de obtenção de competições em qualquer tela do Jude. Em análises feitas utilizando a base de dados de produção percebeu-se que esse tempo diminuiu pela metade. Essa mudança foi positiva pra você?
Q26	Foi corrigido um erro que afetava todas as listas com filtro: ao realizar uma busca estando em uma página superior à primeira, o sistema apresentava falha e não exibia os resultados corretamente. Com essa atualização, o funcionamento das listas tornou-se mais estável e confiável. Essa correção atendeu às suas expectativas?

Continua na próxima página

Tabela 5.1 – Continuação da Tabela anterior

Variável	**Questão**
Q27	Foi corrigido um erro que impedia a visualização completa das turmas cadastradas no sistema — apenas uma parte delas era exibida anteriormente. Com essa atualização, todas as turmas cadastradas agora são corretamente listadas. Essa correção atendeu às suas expectativas?
Q28	A partir desta atualização, os professores só podem atualizar e deletar suas próprias turmas. Anteriormente qualquer professor poderia alterar qualquer turma. Essa mudança foi positiva pra você?
Q29	A partir desta atualização, na tela de detalhes da competição é possível deletar alguma submissão clicando no botão com ícone de lixeira. Essa mudança foi positiva pra você?
Q30	No cadastro de semestre, agora é possível adicionar a informação "Ano". Essa mudança foi positiva para você?
Q31	A partir desta atualização, a lista de relatório de notas é ordenado por data de criação. Do mais recente para o menos recente. Essa mudança foi positiva pra você?
Q32	Você usaria o Jude em substituição a algum outro sistema de submissão que você conheça?
Q33	Se sua resposta anterior foi não, o que você acha que falta no Jude para fazer com que você mude de ideia?
Q34	Você tem alguma sugestão de melhoria para o sistema Jude?

5.1 Avaliação de resultados

Após os professores responderem ao formulário, foram realizados os cálculos de média geral de acordo com as perguntas feitas e os fatores avaliados para cada funcionalidade. O formulário ficou disponível entre os dias **09/07 e 16/07**, totalizando **8 respostas** ao longo de uma semana.

Para aprimorar a experiência de leitura deste trabalho, foi adicionado um mapa de calor (5.2), que marca com diferentes intensidades de amarelo as notas mais frequentes em cada pergunta. Isso facilita a visualização da percepção dos professores quanto às melhorias e implementações da nova versão do sistema Jude.

5.1.1 Análise das respostas dos professores

Variáveis	Fator Avaliado	1	2	3	4	5	Média	Std. Desvio	Variância
Q1	Expectativa	0	0	2	1	5	4.38	0.857	0.7344
Q2	Expectativa	0	0	0	2	6	4.75	0.433	0.1875
Q3	Expectativa	0	0	0	2	6	4.75	0.433	0.1875
Q4	Expectativa	0	0	1	1	6	4.63	0.696	0.4844
Q5	Avaliação	0	0	0	1	7	4.88	0.331	0.1094
Q6	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q7	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q8	Expectativa	0	0	0	2	6	4.75	0.433	0.1875
Q9	Expectativa	0	0	0	2	6	4.75	0.433	0.1875
Q10	Expectativa	0	0	1	0	7	4.75	0.661	0.4375
Q11	Expectativa	0	0	0	1	7	4.88	0.331	0.1094
Q12	Expectativa	0	0	0	1	7	4.88	0.331	0.1094
Q13	Expectativa	1	0	0	0	7	4.50	1.323	1.7500
Q14	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q15	Expectativa	0	0	0	2	6	4.75	0.433	0.1875
Q16	Expectativa	0	0	1	1	6	4.63	0.696	0.4844
Q17	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q18	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q19	Importância	0	0	1	2	5	4.50	0.707	0.5000
Q20	Expectativa	0	0	0	1	7	4.88	0.331	0.1094
Q21	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q22	Impacto	0	0	0	0	8	5.00	0.000	0.0000
Q23	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q24	Expectativa	0	0	0	1	7	4.88	0.331	0.1094
Q25	Impacto	0	0	0	0	8	5.00	0.000	0.0000
Q26	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q27	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q28	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q29	Expectativa	0	0	0	1	7	4.88	0.331	0.1094
Q30	Expectativa	0	0	0	0	8	5.00	0.000	0.0000
Q31	Expectativa	0	0	1	0	7	4.75	0.661	0.4375
Média Geral	Geral	1	0	7	21	219	**4.84**	**0.487**	**0.2374**

Tabela 5.2: Jude - Mapa de calor das respostas dos professores (escala em amarelo).

Variáveis	Funcionalidade	Fator Avaliado
Q1	Sistema de chamadas (notificação de chamada e reenvio manual)	Expectativa
Q2	Sistema de chamadas (atribuição manual de presença, reenvio e tempo limite)	Expectativa
Q3	Sistema de chamadas (reenviar notificação para aluno)	Expectativa
Q4	Sistema de chamadas (tempo limite de resposta)	Expectativa
Q5	Relatório de chamadas	Avaliação
Q6	Relatório de chamadas (exportação para CSV)	Expectativa
Q7	Tela de cadastro de turmas (usabilidade e design)	Expectativa
Q8	Cadastro de turmas (adicionar usuários manualmente, um por um)	Expectativa
Q9	Cadastro de turmas (adicionar usuários em lote)	Expectativa
Q10	Cadastro de turmas (importar usuários de uma competição)	Expectativa
Q11	Cadastro de turmas (importar usuários de outra turma)	Expectativa
Q12	Cadastro de turmas (adicionar participantes na criação da turma)	Expectativa
Q13	Vinculação obrigatória de turma ao cadastrar competição	Expectativa
Q14	Atualização automática dos participantes das competições ao atualizar a turma	Expectativa
Q15	Tempo de inatividade do aluno	Expectativa
Q16	Reativação automática de usuário inativo ao fazer submissão	Expectativa
Q17	Bloqueio de usuários	Expectativa
Q18	Comportamento de usuário bloqueado (não pode visualizar competição)	Expectativa
Q19	Remoção do Celery (redução de complexidade e tempo de julgamento)	Importância
Q20	Vinculação obrigatória de turma ao gerar relatório de notas	Expectativa
Q21	Preenchimento automático de disciplina e semestre com base na turma	Expectativa
Q22	Correção de erro na notificação de horário na tela inicial do Jude	Impacto
Q23	Correção de erro na edição de usuários com ponto no username	Expectativa
Q24	Correção de erro na visualização do código da tarefa de impressão	Expectativa
Q25	Otimização do queryset de competições (redução do tempo de carregamento)	Impacto
Q26	Correção de erro nas listas com filtro em páginas superiores à primeira	Expectativa
Q27	Correção de erro na exibição de todas as turmas cadastradas	Expectativa
Q28	Restrição para que professores atualizem apenas suas próprias turmas	Expectativa
Q29	Exclusão de submissões na tela de detalhes da competição (ícone de lixeira)	Expectativa
Q30	Campo "Ano"no cadastro de semestre	Expectativa
Q31	Ordenação do relatório de notas por data de criação	Expectativa
Média Geral	Sistema Jude (visão geral)	Geral

Tabela 5.3: Tabela de perguntas, funcionalidades e fatores avaliados pelos professores.

A análise dos dados obtidos a partir da avaliação dos professores indica uma percepção amplamente positiva em relação às melhorias e novas funcionalidades da nova versão do sistema Jude.

A média geral das avaliações foi de 4,84, com um desvio padrão de 0,487, o que representa um nível de satisfação elevado e consistente entre os respondentes. Mais especificamente, a maioria das perguntas apresentou notas 5 (máxima), como evidenciado no mapa de calor da Tabela 5.2, no qual as células referentes à nota 5 estão majoritariamente preenchidas com tons escuros de amarelo, representando alta concentração de respostas nessa faixa.

Funcionalidades como a exportação de relatórios em CSV (Q6), a usabilidade da tela de cadastro de turmas (Q7), o bloqueio de usuários (Q17) e diversas correções de bugs (Q22 a Q30) receberam unanimidade com nota 5, o que indica que essas melhorias atenderam ou superaram as expectativas dos usuários.

Apenas duas perguntas receberam notas diferentes de 4 ou 5 com alguma frequência. A Q1 (notificação e reenvio manual de chamadas) obteve uma média de 4,38 e o maior desvio padrão entre as funcionalidades (0,857), sugerindo uma percepção menos uniforme por parte dos professores e possível margem para aprimoramentos. Já a Q13 (vinculação obrigatória de turma ao cadastrar competição) obteve nota 1 de um respondente, o que contribuiu para seu desvio padrão elevado (1,323), apesar de sua média ainda ser satisfatória (4,50). Esse dado pode indicar resistência pontual à funcionalidade ou falhas de comunicação quanto à sua utilidade.

Do ponto de vista da distribuição dos fatores avaliados, observa-se que a maioria das perguntas se refere ao fator "Expectativa", refletindo o quanto as funcionalidades implementadas ou aprimoradas se alinham com o que os usuários esperavam. Os poucos itens classificados como "Impacto" (Q22 e Q25) e "Importância" (Q19) também tiveram avaliações altas, indicando que os ajustes técnicos e estruturais do sistema foram reconhecidos como relevantes.

No conjunto, os resultados reforçam que a nova versão do Jude apresenta melhorias significativas em relação à anterior, sendo bem recebida pelos professores que atuam como usuários-chave do sistema. As pontuações elevadas, associadas à baixa variabilidade na maioria das respostas, indicam tanto o sucesso das implementações quanto a consistência na experiência dos usuários. Além disso, o uso do mapa de calor foi eficaz em destacar visualmente as percepções predominantes e contribui para uma interpretação mais intuitiva dos dados.

Como recomendação, sugere-se monitorar ao longo do tempo as funcionalidades que apresentaram maior dispersão nas avaliações (especialmente Q1 e Q13), buscando identificar causas específicas de insatisfação e promover ciclos de feedback mais contínuos.

Aceitação do JUDE como sistema principal de submissão

A pergunta "Você usaria o Jude em substituição a algum outro sistema de submissão que você conheça?" buscou avaliar o nível de aceitação do sistema JUDE entre os professores que já utilizam outras plataformas de submissão de códigos.

O resultado foi unanimemente positivo: 100% dos respondentes indicaram que utilizariam o JUDE como substituto de outros sistemas. Esse dado reforça a percepção de que o JUDE atende satisfatoriamente às necessidades dos docentes, sendo visto não apenas como uma alternativa viável, mas como uma solução preferencial para avaliação

de atividades práticas de programação.

A resposta unânime, ilustrada na Figura 5.1, sugere que, apesar de ainda haver pontos de melhoria mencionados em outras partes do questionário, o sistema já oferece um conjunto de funcionalidades e uma experiência suficientemente robusta para conquistar a confiança dos usuários.

Esse nível de aceitação também pode ser interpretado como um indicativo da carência de soluções completas ou acessíveis no contexto educacional brasileiro, o que posiciona o JUDE como uma ferramenta de destaque em seu nicho.

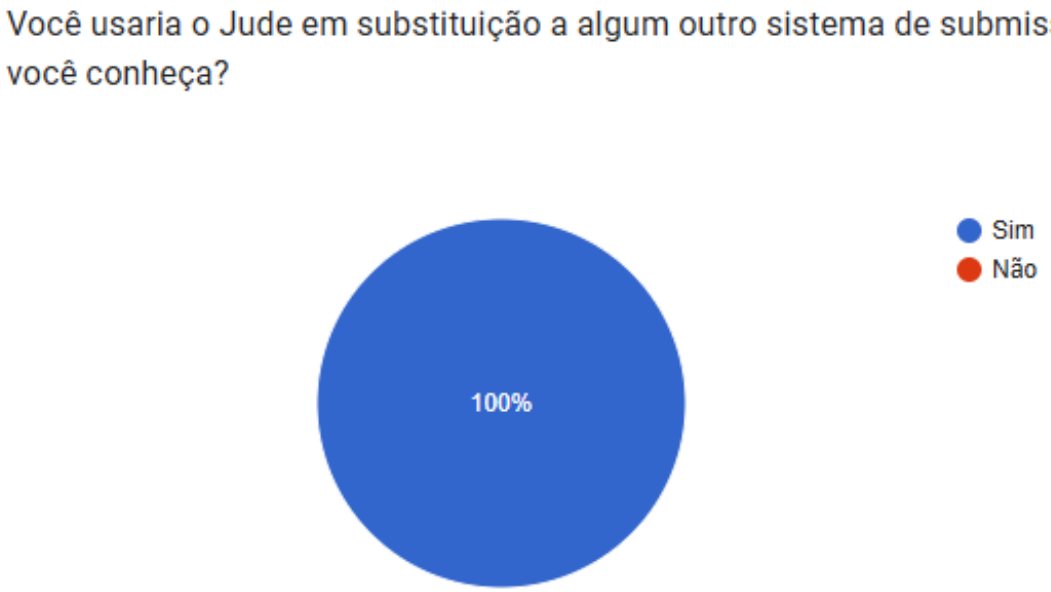


Figura 5.1: Distribuição das respostas à pergunta sobre substituição de outros sistemas pelo JUDE.

Análise das sugestões dos professores

As sugestões dos professores revelam pontos específicos em que o sistema Jude pode ser aprimorado, tanto em termos de usabilidade quanto em funcionalidades mais avançadas. A seguir, sintetizam-se as principais contribuições:

- **Clareza nas interfaces de entrada**: Um dos professores destacou que a funcionalidade de adicionar alunos manualmente a uma turma não deixa claro que os usernames devem ser separados por ponto-e-vírgula. O campo de entrada apresenta apenas um username como exemplo, o que pode induzir o usuário ao erro. Sugere-se melhorar o texto do placeholder ou incluir uma instrução explícita sobre o formato exigido.

- **Importação de usuários de competições**: Há dificuldade na importação de

usuários a partir do ID de competições, uma vez que o usuário precisa sair da tela atual para localizar esse ID. Além disso, foi apontada uma inconsistência na terminologia: o campo utiliza o termo "contest" em vez de "competição", o que pode gerar confusão. Recomenda-se alinhar a terminologia ao padrão adotado no restante do sistema e considerar alternativas de busca automática ou seleção via interface.

- **Importação de usuários do SIAC**: Os procedimentos para importar arquivos do SIAC (.xls e .txt) não estão suficientemente claros, especialmente em relação à origem e função do arquivo .txt (lista de e-mails gerada pelo SIAC). Uma documentação mais clara ou instruções na interface podem reduzir erros e aumentar a autonomia do usuário.

- **Competições de segunda chamada**: Foi sugerido que o sistema permita habilitar apenas os alunos que efetivamente realizarão a prova, evitando a necessidade de bloquear manualmente os demais. Essa melhoria seria especialmente útil em turmas com grande número de estudantes e poucos participantes na segunda chamada.

- **Segurança e controle de acesso**: Alguns professores sugeriram a implementação de bloqueio por faixa de IP, restringindo o acesso ao sistema durante provas apenas a dispositivos conectados à rede da UFBA. Além disso, foi recomendada a criação de um log de acessos com registros de horário e IP, o que aumentaria a transparência e a segurança em avaliações online.

- **Criação de competições para testes**: A obrigatoriedade atual de vincular uma competição a uma turma foi apontada como um obstáculo para professores que desejam criar competições de teste. Sugeriu-se a possibilidade de criar competições desvinculadas, o que facilitaria o uso do sistema em contextos experimentais ou de simulação.

- **Ambiente de desenvolvimento integrado**: Por fim, foi sugerida a criação de um ambiente completo que incluísse editor de texto, compilador e terminal, oferecendo maior independência e flexibilidade ao aluno dentro da própria plataforma.

Essas sugestões oferecem um direcionamento valioso para as próximas versões do sistema. A maioria dos pontos envolve ajustes de usabilidade e comunicação, mas também há propostas que envolvem maior complexidade técnica, como controle de acesso por IP e integração de ferramentas de desenvolvimento. Tais contribuições demonstram o engajamento dos professores e reforçam a importância de manter canais abertos de escuta ativa para evolução contínua do Jude.

Capítulo 6

Conclusão

O presente trabalho teve como objetivo reestruturar e aprimorar o sistema Jude Online Judge, ferramenta desenvolvida e utilizada na Universidade Federal da Bahia como suporte ao ensino de programação. A iniciativa surgiu a partir da identificação de limitações técnicas e funcionais da versão anterior do sistema, que comprometiam tanto a experiência dos usuários finais quanto a manutenção por parte dos desenvolvedores.

Ao longo do projeto, foram implementadas melhorias significativas na arquitetura, usabilidade e desempenho do sistema. Destacam-se a substituição do Celery por Webhooks, a implementação de um sistema de chamadas com uso de WebSockets, a integração automática entre turmas e competições, e a ampliação das formas de adicionar usuários às turmas. Além disso, foi estruturado um novo modelo de versionamento semântico, com geração automatizada baseada em webhooks, promovendo maior rastreabilidade e organização das entregas.

Do ponto de vista metodológico, adotou-se uma abordagem prática, incremental e centrada no uso de ferramentas modernas como o Jira e o GitLab, bem como no uso de ambientes replicáveis com Docker e WSL. A estratégia de desenvolvimento priorizou entregas de alto impacto para professores, além da manutenção da estabilidade e coerência do sistema.

Para validar as melhorias, foi conduzida uma avaliação com os professores usuários do sistema, utilizando um formulário estruturado. Os resultados demonstraram um alto grau de satisfação: a média geral das avaliações foi de 4,84 (em uma escala de 1 a 5), com baixíssima variabilidade nas respostas. Funcionalidades como exportação de relatórios, bloqueio de usuários, cadastro de turmas e correções de bugs obtiveram unanimidade em avaliações máximas. Adicionalmente, todos os professores afirmaram que utilizariam o Jude em substituição a outras plataformas de submissão de código, o que reforça sua aceitação e relevância prática no ambiente educacional.

As sugestões dos professores também trouxeram insights valiosos para futuras

versões do sistema, incluindo a necessidade de melhorar mensagens de erro, implementar restrições por IP em provas, e adicionar um ambiente completo de codificação diretamente na plataforma. Essas contribuições demonstram o alto nível de engajamento da comunidade usuária e apontam caminhos promissores para evolução contínua.

Conclui-se, portanto, que os objetivos deste trabalho foram plenamente alcançados. As melhorias implementadas tornaram o Jude um sistema mais robusto, intuitivo e alinhado com as demandas contemporâneas do ensino de programação. Além disso, o processo seguido neste projeto consolidou boas práticas de engenharia de software no contexto acadêmico, servindo como modelo para iniciativas semelhantes. Espera-se que os resultados aqui obtidos contribuam para a continuidade do desenvolvimento da ferramenta, beneficiando diretamente estudantes, professores e demais colaboradores da área de Computação na UFBA e, potencialmente, em outras instituições.

6.1 Trabalhos Futuros

Com base nas limitações e possibilidades observadas durante o desenvolvimento, algumas melhorias e extensões podem ser consideradas para versões futuras do sistema:

- **Substituição do uso de *polling* por *WebSockets*:** Atualmente, o site utiliza intensivamente o *polling*, com diversas telas realizando requisições ao servidor a cada 5 segundos. Isso gera sobrecarga no backend e resulta em uma experiência menos eficiente para o usuário. A adoção de *WebSockets* permitiria atualizações em tempo real apenas quando necessário, otimizando recursos e melhorando a performance.

- **Revisão nas mensagens de erro do sistema:** Muitas mensagens de erro apresentadas ao usuário não são claras ou informativas. Um exemplo recorrente é na criação de usuários, onde erros específicos não são descritos, sendo exibida apenas uma mensagem genérica. A melhoria na clareza das mensagens facilitaria a identificação e correção de problemas por parte dos usuários.

- **Redirecionamento automático ao expirar o token de autenticação:** Atualmente, quando o token expira, o usuário não é redirecionado automaticamente para a tela de login, o que pode causar confusão. Uma verificação constante e o redirecionamento automático ao fim da sessão melhorariam a usabilidade e a segurança.

- **Tela de notícias e novidades do sistema:** A criação de uma tela dedicada à exibição das atualizações e novidades da última versão publicada permitiria aos usuários compreender melhor o propósito do versionamento e explorar de forma mais eficaz os novos recursos implementados.

- **Validação visual no upload de casos de teste:** No cadastro de problemas, ao fazer o upload do arquivo compactado (`.zip`) contendo os casos de teste, não há atualmente uma confirmação visual de que o processo foi bem-sucedido ou se o conteúdo está em conformidade com o formato esperado. A implementação de uma visualização prévia ou mensagens de validação reduziria erros e melhoraria a experiência do usuário.

- **Qualidade de software:** Atualmente, o sistema verifica apenas se o código enviado produz a saída esperada. No entanto, é desejável implementar uma funcionalidade que vá além da correção funcional, analisando também a estrutura e as boas práticas de programação utilizadas na solução. Essa análise pode considerar aspectos como legibilidade, organização, nomenclatura de variáveis e uso adequado de estruturas de controle, contribuindo para o incentivo à escrita de códigos mais limpos e sustentáveis.

Referências Bibliográficas

[1] Ask Solem Hoel. Celery. Disponível em: `https://docs.celeryq.dev/en/stable/`. Acesso em: 07 de julho de 2025.

[2] Atlassian. Jira. Disponível em: `https://www.atlassian.com/br/software/jira`. Acesso em: 07 de julho de 2025.

[3] beecrowd. Beecrowd. Disponível em: `https://www.beecrowd.com.br/`. Acesso em: 04 de novembro de 2022.

[4] Daramsenge Bilegjargal and Nien-Lin Hsueh. Understanding students' acceptance of online judge system in programming courses: A structural equation modeling approach. *IEEE Access*, PP:1–1, 11 2021.

[5] C. P. Dd Campos and C. E. Ferreira. Boca: um sistema de apoio a competições de programação. *Anais do Congresso da SBC*, 2004.

[6] Django Software Foundation. Django. Disponível em: `https://www.djangoproject.com/`. Acesso em: 26 de novembro de 2022.

[7] Docker Inc. Docker. Disponível em: `https://www.docker.com/`. Acesso em: 26 de novembro de 2022.

[8] Herman Zvonimir Došilović and Igor Mekterović. Robust and scalable online code execution system. pages 1627–1632, 2020.

[9] George E. Forsythe and Niklaus Wirth. Automatic grading programs. *Commun. ACM*, 8(5):275–278, 1965.

[10] Git SCM. Git. Disponível em: `https://git-scm.com/`. Acesso em: 07 de novembro de 2022.

[11] HackerRank. Hackerrank. Disponível em: `https://www.hackerrank.com/`. Acesso em: 12 de junho de 2023.

[12] Igor Sysoev. Nginx. Disponível em: `https://nginx.org/`. Acesso em: 07 de julho de 2025.

[13] Jude. A modern judge designed for on-site programming contests. Disponível em: `http://200.128.51.30/`. Acesso em: 13 de novembro de 2022.

[14] Judge0. Judge0 - the most advanced open-source online code execution system in the world. Disponível em: `https://judge0.com/`. Acesso em: 07 de novembro de 2022.

[15] LeetCode. Leetcode. Disponível em: `https://leetcode.com/`. Acesso em: 12 de junho de 2023.

[16] Meta (Facebook). React - uma biblioteca javascript para criar interfaces de usuário. Disponível em: `https://pt-br.reactjs.org/`. Acesso em: 13 de novembro de 2022.

[17] PostgreSQL Global Development Group. Postgresql: The world's most advanced open source database. Disponível em: `https://www.postgresql.org/`. Acesso em: 13 de novembro de 2022.

[18] Redis Ltd. Redis. Disponível em: `https://redis.io/`. Acesso em: 19 de novembro de 2022.

[19] Miguel A. Revilla, Shahriar Manzoor, and Rujia Liu. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, 2(10):131–148, 2008.

[20] Spoj. Spoj - sphere online judge. Disponível em: `https://www.spoj.com/info/`. Acesso em: 04 de novembro de 2022.

[21] M. Wang, W. Han, and W. Chen. Metaoj: A massive distributed online judge system. *Tsinghua Science and Technology*, 26:548–557, 2021.

[22] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *ACM Comput. Surv.*, 51(1), jan 2018.