UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOÃO AUGUSTO GROBE CASTILHO

DESENVOLVIMENTO DE UMA PLATAFORMA ON-LINE DE SUBMISSÃO E EXECUÇÃO EM LOTES DE ROTINAS COMPUTACIONAIS DESENVOLVIDAS EM OCTAVE

PONTA GROSSA

2022

JOÃO AUGUSTO GROBE CASTILHO

DESENVOLVIMENTO DE UMA PLATAFORMA ON-LINE DE SUBMISSÃO E EXECUÇÃO EM LOTES DE ROTINAS COMPUTACIONAIS DESENVOLVIDAS EM OCTAVE

Development of an Online Platform for Submission and Execution in Batch of Computational Routines Developed in Octave

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Diego Roberto Antunes

PONTA GROSSA

2022

JOÃO AUGUSTO GROBE CASTILHO

DESENVOLVIMENTO DE UMA PLATAFORMA ON-LINE DE SUBMISSÃO E EXECUÇÃO EM LOTES DE ROTINAS COMPUTACIONAIS DESENVOLVIDAS EM OCTAVE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 25/maio/2022

Diego Roberto Antunes
Doutor
Universidade Tecnológica Federal do Paraná

Richard Duarte Ribeiro
Doutor
Universidade Tecnológica Federal do Paraná

Luiz Rafael Schmitke
Mestre
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2022

AGRADECIMENTOS

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas, as quais sou imensamente grato.

A minha família e amigos, por toda a colaboração, suporte e motivação.

As pessoas com quem eu trabalho, pela compreensão aos momentos de ausência e atrasos.

Ao meu orientador e professores, os quais me forneceram a base para desenvolver este trabalho.

RESUMO

O ensino de Cálculo Numérico está inserido na grade de diversos cursos de graduação, tais como Licenciatura e Bacharelado em Matemática e Física, em cursos de engenharias e áreas afins. Esta disciplina conta com o estudo de métodos numéricos para resolução de problemas em gerais. Alguns autores tratam o ensino da disciplina de forma tradicional, abordando puramente o lado teórico matemático da disciplina; outros compreendem que tais métodos numéricos tem aplicabilidade limitada se não forem implementados como programas de computador. Dessa forma, defendem o ensino do cálculo numérico utilizando *softwares*, entretanto, esta prática pode gerar uma carga de trabalho adicional para o professor. Dentre os desafios, está a correção de um algoritmo submetido pelo aluno, a qual destaca-se neste trabalho a execução do algoritmo do aluno por meio de forma automatizada, que tradicionalmente é feita de forma manual, e no qual o professor precisa realizar a execução de cada uma das submissões em diversos casos distintos, a fim de testar o comportamento do código.
Este trabalho visa apresentar o processo de desenvolvimento de uma plataforma que permite a execução em lotes de algoritmos desenvolvidos na linguagem Octave, agilizando a correção da atividade pelo professor e, consequentemente, gerando o *feedback* mais rápido para o aluno. A ferramenta desenvolvida permite o gerenciamento de turmas, exercícios e testes, os quais são executados pela plataforma e os dados são apresentados para o professor realizar a correção sem a necessidade da execução manual dos algoritmos submetidos. A plataforma ainda permite escalabilidade, pois exercícios podem ser executados em paralelo, melhorando o tempo de resposta para o professor e para o aluno. Foram simulados testes em turmas fictícias em três cenários distintos, a fim de testar a estabilidade e coletar *feedbacks* iniciais do sistema, o qual apresentou-se capaz de realizar os exercícios submetidos para todos os testes cadastrados, cumprindo o objetivo proposto pelo trabalho.

Palavras-chave: tecnologia educacional; inovações educacionais; ensino - meios auxiliares; gnu octave (programa de computador).

ABSTRACT

Numerical Calculus is present in the grade of several undergraduate courses, such as bachelor's degree in mathematics and physics, in engineering courses and related areas. The course studies numerical methods for solving general problems. Some authors approach the teaching of the subject in the traditional way, approaching purely the mathematical side of the subject; others understand that such numerical methods have limited applicability if not implemented as computer programs. Thus, defend the teaching of numerical calculation using software, however, this practice can generate an additional workload for the professor. Among the challenges of performing the correction of an algorithm submitted by the student, the execution of the code stands out in this paper, which is traditionally done manually, where the professor needs to run each of the submissions in several different cases, in order to test the behavior of the algorithm. This work aims to present the process of developing a platform that allows the execution in batches of algorithms developed in the Octave language, speeding up the correction of the activity by the professor, and consequently generating faster feedback for the student. The developed tool allows the management of classes, exercises and tests, which are executed by the platform, and the data is presented for the professor to perform the correction without the need to manually run the submitted algorithms. The platform also allows scale, where exercises can run in parallel, improving the response time for the professor and the student. Tests were simulated in fictitious classes in three different scenarios, in order to test the stability and collect initial feedback of the system, which was able to perform the exercises for all registered tests, fulfilling the objective proposed by the work.

Keywords: technology, educational; education - innovations; teaching materials; gnu octave (software) .

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMQP	*Advance Message Queuing Protocol*
API	*Application Programming Interface*
BSON	*Binary Javascript Object Notation*
CLI	*Command Line Interface*
CSR	*Client Side Rendering*
CSV	*Comma Separated Values*
DNS	*Domain Name System*
FIFO	*First In First Out*
GUI	*Graphical User Interface*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
RA	Registro Acadêmico
RAM	*Random Access Memory*
SPA	*Single Page Application*
SSR	*Server Side Rendering*
TCP	*Transmission Control Protocol*
vCPU	*Virtual Central Processing Unit*

SUMÁRIO

1 INTRODUÇÃO

O Cálculo Numérico pode ser considerado como a sinergia entre computadores programáveis, análise matemática e a oportunidade de resolver problemas grandes e complexos (SIAM, 2006). Em seu conteúdo programático, a disciplina de Cálculo Numérico estuda métodos para encontrar zeros de funções, interpolar e aproximar funções, aproximar integrais e resolver sistemas lineares (PIRES, 2015). Áreas como a previsão balística e dinâmica de fluidos multidimensional não-estacionária, entre muitas outras, devido às suas necessidades por avanço, impulsionaram o desenvolvimento do computador e dependeram fortemente dos avanços de métodos numéricos e modelagem matemática (SIAM, 2006).

Em nosso país, tradicionalmente, a disciplina de Cálculo Numérico é lecionada de forma totalmente teórica, isto é, sem o uso de ferramentas como linguagens de programação e *softwares*. Entretanto, como já observou Freire (2016), “ensinar não é transferir conhecimento, mas sim criar as possibilidades para a sua própria produção ou a sua construção". Desta forma, restringir o ensino do Cálculo Numérico apenas à reprodução de seus conteúdos, é deixar de aproveitar a enorme gama de aplicabilidade real que esta disciplina tem e, com isso, desperdiçar a possibilidade de estimular no aluno a criatividade e o desenvolvimento de atitudes científicas. Neste sentido, alguns autores, como Arenales (2016) e Pires (2015) optam por utilizar abordagens com o auxílio de *softwares*. Entre eles, se destaca o *MATLAB* (MATHWORKS, 1984), o qual é o *software* mais utilizado nesse contexto. O *Octave* (EATON, 1988) é uma alternativa para a resolução de problemas numéricos, oferecendo grande compatibilidade com o primeiro (SHARMA; GOBBERT, 2010) e é gratuito.

Um dos empecilhos para a adoção de *softwares* no ensino de Cálculo Numérico, é que a correção destes algoritmos, por meio da execução manual de cada algoritmo, acrescido da análise do professor, consome muito tempo. A correção de um código requer diversas execuções com parâmetros diferentes para cobrir vários casos. Além disso, a execução e gerenciamento de tantas execuções não é trivial. O professor normalmente precisa realizar esses testes de forma manual, um a um, e gerenciar os parâmetros de entrada e de saída do código de cada aluno. Este processo, além de consumir tempo do professor, está sujeito a erros humanos. Adicionalmente a isso, é comum um professor de Cálculo Numérico lecionar para diversas turmas, o que torna quase inviável realizar atividades práticas, uma vez que cada atividade pode resultar em centenas, ou até milhares de execuções dos algoritmos. Por exemplo em uma turma de 40 alunos, um exercício contendo 5 testes necessitam 200 execuções para a devida correção.

Além do enorme consumo de tempo de correção que a execução manual de algoritmos acarretaria para um professor, no lado do aluno, as coisas também seriam dificultadas, pois, uma atividade em que o aluno demore para receber um *feedback* pode prejudicar seu desempenho na disciplina. Visto que alguns métodos são dependentes de conteúdos estudados anteriormente, um *feedback* atrasado pode gerar um acúmulo de dúvidas no aluno, impedindo-o de fixar o primeiro conteúdo antes de avançar para temas mais complexos.

As dificuldades acima mencionadas, levam ao seguinte questionamento: seria possível obter uma forma de otimizar os tempos de correção do professor? A resposta para este problema, juntamente com o inegável ganho acadêmico que o ensino de Cálculo Numérico tem com a utilização de *softwares*, justificam a necessidade de se desenvolver uma ferramenta para auxiliar a execução de códigos de forma programática e em lote, uma vez que tal ferramenta pode ser adaptada para a correção de atividades voltadas para o ensino prático e contextualizado do Cálculo Numérico.

1.1 Objetivo geral

Desenvolver uma ferramenta para a execução de códigos desenvolvidos na linguagem *Octave* de forma programática e em lote para o auxiliar a correção de exercícios de programação na disciplina de Calculo Numérico.

1.2 Objetivos específicos

Para atingir o objetivo geral proposto, os seguintes objetivos foram necessários:

- Modelar a arquitetura do sistema;
- Modelar e desenvolver as telas e as interfaces;
- Desenvolver uma API para comunicação da interface com banco de dados e coordenar a execução dos algoritmos;
- Desenvolver um ambiente para execução isolada do código do aluno e realizar a execução de forma programática.
- Avaliar a ferramenta proposta.

1.3 Organização do trabalho

O presente texto está dividido em cinco capítulos. Neste primeiro, é apresentada uma breve contextualização do tema, bem como o problema a ser resolvido e uma justificativa para este trabalho. Além disso, os objetivos gerais e específicos são também apresentados.

O segundo capítulo consta de uma breve revisão sobre a disciplina de Cálculo Numérico, como sua origem histórica e seu estado atual, desde seu conteúdo programático até a metodologia através da qual este conteúdo é ministrado. Posteriormente, são apresentados dois trabalhos similares a este. Por fim, neste capítulo, são apresentadas, brevemente, as principais etapas para a arquitetura de um *software* e definidos alguns conceitos importantes utilizados no trabalho.

Durante o terceiro capítulo, a arquitetura da ferramenta aqui desenvolvida é apresentada de forma detalhada, descrevendo cada um de seus componentes, juntamente com a comunicação entre os serviços e forma em que ocorre o processo de execução utilizando a plataforma desenvolvida.

O quarto capítulo apresenta os resultados obtidos através do funcionamento da plataforma desenvolvida, bem como uma discussão sobre eles.

Por último, as conclusões deste trabalho e seus impactos, bem como uma visão geral do que foi feito e sugestões para trabalhos futuros, serão apresentadas no capítulo 5.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo faz uma breve introdução sobre a disciplina de Cálculo Numérico e sua forma de ensino, ferramentas similares e ainda a introdução de alguns conceitos de Engenharia de *software* utilizados no trabalho.

2.1 Disciplina de Cálculo Numérico

O Cálculo Numérico, de uma maneira geral, pode ser visto como o estudo dos métodos numéricos para a resolução de problemas em geral, isto é, os métodos por meio dos quais se encontram soluções aproximadas de problemas mais complexos (AMARAL; LEITE; SILVA, 2013). Como disciplina, o Cálculo Numérico pode estar inserido em vários cursos de graduação, tais como Licenciatura e Bacharelado em Matemática e em Física, nas diversas engenharias e nos cursos de Economia e áreas afins, visto que suas ferramentas buscam a solução de problemas que, muitas vezes, seriam de difícil resolução através do Cálculo Analítico. Enquanto ciência, o Cálculo Numérico surge a partir do século XVIII, não desvinculado do Cálculo Analítico, desta forma, não existe uma precisão de datas separando os trabalhos dessas duas áreas, já que, nessa época, os problemas eram resolvidos sem que se pensasse na possibilidade de soluções numéricas de forma isolada. Foi com Newton (1642-1727), Lagrange (1736-1813), Gauss (1777-1855), Raphson (1648-1715) entre outros, que o Cálculo Numérico ganhou muito destaque (SPERANDIO; MENDES; SILVA, 2003).

No Brasil, o ensino de Cálculo Numérico e da Análise Numérica começou em 1953, na recém-criada Escola de Engenharia de São Carlos-USP (EESC-USP), que incluiu pela primeira vez Cálculo Numérico como disciplina obrigatória. Anos depois, pela popularização dos computadores, esta disciplina esta presente em praticamente todas as outras instituições de ensino superior nas áreas de Ciências Exatas e Tecnologia (FILHO, 2018). Atualmente, em nosso país, o ensino do Cálculo Numérico segue algumas tendências. No que se refere a conteúdo programático, a grande maioria dos cursos, tem como parte comum de suas ementas: interpolação, ajuste de curvas, sistemas lineares, integração, equações (diferenciais, algébricas e transcendentes). Já no que tange a abordagem deste conteúdo, verifica-se a existência de uma ênfase diferenciada que depende do curso de graduação no qual a disciplina está sendo lecionada: métodos numéricos acompanhados de aplicações tecnológicas relacionadas a cada área do curso nas diferentes engenharias, o ensino mais detalhado de conceitos e teoremas no curso de Matemática, uma abordagem mais teórica voltada para a produção de softwares numéricos que envolvem os métodos apresentados nos cursos de Computação (VASCONCELLOS; BARROSO, 1994).

Além das tendências verificadas no conteúdo programático e na abordagem deste, uma outra tendência notória que surge no ensino do Cálculo Numérico no Brasil se refere à metologia usada que, muitas vezes, caracteriza-se apenas como tradicional, isto é, aquela em que o pro-

fessor apresenta a teoria, exemplos e os alunos resolvem exercícios e realizam avaliações como forma de verificar a aprendizagem do conteúdo passado. Alguns autores abordam essa disciplina em sua forma tradicional, isto é, sem o uso de linguagens de programação e softwares, abordando puramente o lado matemático da disciplina, veja por exemplo Filho (2018). Outros como Rajan (2016) compreendem que os métodos numéricos têm aplicabilidade limitada se não forem implementados como programas de computador.

O Desenvolvimento tecnológico naturalmente acarreta muitas mudanças no processo educacional e, tais mudanças, impulsionam a criação de novos métodos de ensino e processos de aprendizagem baseados nas tecnologias que são frutos desse desenvolvimento. O rápido avanço das tecnologias computacionais é algo inegável, hoje, os computadores são exímias ferramentas de cálculo e, como tais, não podem ser desprezadas, podendo ter todo esse potencial usado a favor do ensino. Em outras palavras, os aspectos computacionais podem ser usados para enriquecer e estimular o interesse pelos métodos numéricos (GAMA; GOMES; PIRES, 2018).

O ensino do Cálculo Numérico deve estimular a criatividade, de forma que o conteúdo a ser aprendido ou ministrado deve desenvolver atitudes científicas nos alunos. Assim, a metodologia utilizada para ministrar esses conteúdos deve promover um acesso ao conhecimento, de modo rápido e, ao mesmo tempo, permitir ser constantemente atualizado, de forma a acompanhar o rápido desenvolvimento tecnológico e as mudanças que este acarreta (VASCONCELLOS; BARROSO, 1994). Neste sentido, Bhuyan e Khan (2016) afirmam que a disciplina de Cálculo Numérico tem aplicações reais nas áreas da engenharia, e que a disciplina tem que ser ensinada de forma eficaz para que os alunos possam aplicar os conhecimentos aprendidos neste curso na resolução de seus problemas práticos, onde, para cumprir tais propósitos, os autores propõem o ensino da disciplina utilizando o MATLAB, que é atualmente a linguagem mais utilizada na computação numérica (SHARMA; GOBBERT, 2010).

2.2 Ferramentas de submissão e correção de exercícios

Segundo Ihantola *et al.* (2010), a avaliação contínua com *feedback* durante um curso de programação garantem que os alunos tenham prática o suficiente e soluções de qualidade. Os autores ainda afirmam que a avaliação manual, mesmo em turmas pequenas, faz com que o *feedback* não seja instantâneo como em uma aula particular e que, conforme o tamanho da turma cresce, a quantidade de trabalho avaliado tem que ser cortada ou racionalizada de alguma forma.

Sistemas de múltiplas escolhas com avaliação automática são de grande interesse para cursos universitários com grande quantidade de alunos, e são usados em diversas instituições (INSA; SILVA, 2015). Gibbs (1999) afirma que estes tipos de testes desencorajam o aluno a ter uma compreensão mais profunda do assunto, e usar testes de múltiplas escolhas para exercí-

cios de programação não é recomendado por Ahoniemi e Reinikainen (2006), que dizem que a maior parte do aprendizado é baseado no estudante programando por si mesmo.

Entretando, nas palavras de Gupta, Dubey *et al.* (2012, p. 107), "a avaliação automática de atribuições de programação é uma das principais tarefas nas aulas de programação para avaliar e marcar os exercícios de programação dos alunos com o auxílio do computador".

Romli, Sulaiman e Zamli (2010) dizem que a avaliação do programa requer a avaliação da qualidade do programa, e para isso é essencial que o programa seja testado. Sommerville (2010) propõe dois tipos de testes para analisar a qualidade de *softwares*: caixa branca, onde as informações internas do software são conhecidas e utilizadas para desenvolver os casos de testes e medir a qualidade do código; e o teste caixa preta, onde o software é testado apenas analisando as entradas e saídas, sem o conhecimento das instruções internas do *software*.

> O teste de caixa preta é a técnica mais amplamente usada para teste dinâmico em vez de teste de caixa branca. Isso devido que a corretude do programa é o fator de qualidade mais importante deve ser incluído na avaliação dos programas dos alunos (ROMLI; SULAIMAN; ZAMLI, 2010, p. 1187).

A análise de caixa preta é baseada em testes comparando a saída do código do aluno com o resultado esperado informado pelo professor. Este tipo de avaliação apresenta diversos problemas, por exemplo, é difícil saber qual a cobertura dos testes, não é possível saber se o programa do aluno está realmente certo e nem saber se ele esta usando boas técnicas de programação (GERDES; JEURING; HEEREN, 2010). Dessa forma, checar apenas a saída do programa do aluno normalmente não é o suficiente para garantir a qualidade no código (INSA; SILVA, 2015).

Este trabalho propõe a utilização de testes de caixa preta para auxiliar o professor fazendo uma triagem de erros comuns através dos testes, deixando para o professor realizar uma análise complementar na submissão para avaliar a lógica e a solução do algoritmo.

2.2.1 Trabalhos similares

Os corretores automáticos de código possuem como objetivo validar a consistência de um algoritmo através de vários casos teste. Este processo é semelhante ao utilizado pela engenharia de *software* no teste da caixa preta ou funcional, acima mencionado. Pois, no teste da caixa preta não se tem conhecimento interno do algoritmo, os dados de entrada são fornecidos, a partir deles o algoritmo interpreta os dados recebidos e o resultado de saída gerado pelo algoritmo é comparado com o conhecido (PRESSMAN; MAXIM, 2021).

As atividades computacionais costumam ser individuais e a correção delas demandam muito tempo, pois o professor precisa executar as rotinas dos alunos uma por uma. Neste sentido, a contribuição dos corretores está ligada à comodidade, facilidade e agilidade que pode proporcionar ao professor e aos estudantes, pois elas permitem a redução do tempo de res-

posta, evita a quebra de raciocínio lógico do estudante, diminui a carga de correção sobre o professor, além de procurar motivar o aluno para um melhor aproveitamento na disciplina (NAZÁRIO; SOUZA, 2010).

De forma a otimizar o tempo de correção e para que os alunos tenham um *feedback* mais rápido e preciso em relação ao seu desempenho, já existem algumas ferramentas de correções de rotinas computacionais. Como um primeiro exemplo, podemos mencionar aqui o BOCA-Lab, utilizado no ensino da disciplina Linguagem de Programação I, no curso de Bacharelado em Ciência da Computação, da Universidade Estadual de Santa Catarina (UDESC). Esta ferramenta, além da correção automática, realiza a orientação por conteúdo a cada problema submetido pelo estudante, através de dicas, e trabalha com rotinas nas linguagens C, C++ e *Java* (NAZÁRIO; SOUZA, 2010).

Como segundo exemplo, podemos mencionar a *Octave Correction Tool*, ferramenta de correção de rotinas implementadas em GNU Octave/MATLAB, destinada, *a priori*, para a disciplina de Cálculo Numérico, que permite aos professores fazerem uma correção em segundos e terem acesso a planilhas de notas e relatórios de desempenho dos alunos de forma automática e precisa. Já os alunos, por sua vez, têm um *feedback* mais rápido, permitindo a eles saber se tiveram ou não êxito na tarefa de forma também rápida e precisa. Esta ferramenta tem como vantagem, em relação ao BOCA-Lab, que ela pode ser usada abertamente, ao contrário da primeira, que não é disponibilizada para as outras instituições. Contudo, vale ressaltar que a *Octave Correction Tool* também apresenta suas próprias desvantagens como, por exemplo, está disponível apenas para o sistema operacional *Linux* (SILVA *et al.*, 2020) e tem uma interface difícil para manter turmas e exercícios, através de uma *Command Line Interface* (CLI) e da manipulação de arquivos.

2.3 Arquitetura de software

Alguns termos já citados ou que serão apresentados no decorrer deste trabalho necessitam de uma breve contextualização para melhor compreensão, os seguintes a seguir: *Application Programming Interface* (API), *Front End*, *Conteinerização*, *Sistema de mensageria*, sanam essa necessidade.

2.3.1 API

API é um conjunto de funções que uma aplicação fornece para outras aplicações (OREIZY; TAYLOR, 2003), permitindo que dois softwares se comuniquem através de um protocolo estabelecido. Dessa forma, não é necessário que a aplicação conheça os detalhes de implementação do software, apenas ter conhecimento da interface oferecida pela API.

As funções internas do sistema, disponibilizadas através da API permitem o usuário interagir indiretamente com software, por exemplo permitindo a comunicação com o banco de dados, inserindo, removendo, atualizando e recebendo os dados do sistema de acordo com as regras de negócio estabelecidas pela API. Dessa forma, é possível, por exemplo, impedir que um professor realize a submissão de uma atividade, e impedir que um aluno realize o cadastro de novos testes, ou ainda impedir que um teste seja cadastrado com dados inválidos.

2.3.2 Front End

Aplicações *Web* raramente trabalham sozinhas. O que as fazem interessantes é o fato que elas nos possibilitam comunicar com a Internet e inovar de formas que não eram possíveis anos atrás (PASSAGLIA, 2017). O desenvolvimento *Web* moderno normalmente utiliza duas estratégias: *Server Side Rendering (SSR)*, na qual o servidor tem a responsabilidade de entregar a página com os dados dinâmicos e estáticos, ou *Client Side Rendering* (CSR), na qual o servidor envia a página contendo apenas os dados estáticos, e a página solicita os dados dinamicamente de acordo com a necessidade (MORINIGO, 2020).

O *Front End* é a camada executada no lado do cliente, ou seja, pelo navegador do usuário. Utilizando a estratégia CSR, a página que o usuário recebe contem apenas a estrutura geral, os dados serão gerados e requisitados separadamente através da *API* por meio de requisições *HyperText Transfer Protocol Secure* (HTTPS), e o *Front End* realiza a renderização destes dados de acordo com a estrutura definida na página.

2.3.3 Conteinerização

Para Oliveira (2020), a conteinerização é um método usado no processo de implantação de aplicações altamente distribuídas. Nesse modelo, várias soluções isoladas são executadas em um host, compartilhando o mesmo kernel do sistema operacional.

Ao executar softwares em um contêiner, os processos têm isolamento em nível de rede, memória e disco, através do uso de *namespaces* e *cgroups* do *kernel* (GRUNERT, 2019). Ferramentas como o *Docker* fazem a abstração desses recursos do *kernel* e oferecem uma interface simples para o desenvolvimento de *softwares* utilizando os conceitos de contêineres, entretanto é possível usar nativamente esses recursos e rodar processos isolados no sistema operacional, sem o uso de ferramentas adicionais como o *Docker*.

Os contêineres *Docker* são uma ferramenta poderosa para construir e implantar serviços de forma consistente e confiável (SMITH, 2017). Um contêiner é um conjunto de um ou mais processos organizados isoladamente do sistema. Todos os arquivos necessários para executá-los são fornecidos por uma imagem de sistema (REDHAT, 2018).

> Ao executar um contêiner, ele usa um sistema de arquivos isolado. Esse sistema de arquivos personalizado é fornecido por uma imagem de contêiner. Como a imagem contém o sistema de arquivos do contêiner, ela deve conter tudo o que é necessário para executar um aplicativo - todas as dependências, configurações, *scripts*, binários, etc. A imagem também contém outras configurações para o contêiner, como variáveis de ambiente, um comando padrão a ser executado, e outros *metadados* (DOCKER, 2013).

Existem situações onde a troca de arquivos do sistema hospedeiro com o contêiner, ou ainda, entre contêineres são necessárias. Além disso, cada contêiner conta com seu próprio sistema de arquivos e não persiste seus dados após a finalização do contêiner. O *Docker* tem duas opções para armazenar arquivos no sistema hospedeiro, para que os arquivos persistam mesmo após a interrupção do contêiner: volumes e *bind mounts*, onde o primeiro é o mecanismo preferencial de persistência de dados, pois é totalmente gerenciado pelo *Docker*, enquanto a outra estrutura depende do sistema de arquivos montado no sistema hospedeiro (DOCKER, 2013).

Em diversos cenários também é necessária a comunicação através da rede entre contêineres. O *Docker* oferece *drivers* para a comunicação, permitindo, por exemplo, remover o isolamento de rede, desativar toda a rede do contêiner ou, por padrão, uma comunicação em *bridge*, a qual permite redirecionar tráfego entre segmentos da rede, permitindo definir grupos de contêineres capazes de se comunicar entre si. Por padrão, todos os contêineres se encontram no mesmo grupo, e os contêineres podem se comunicar utilizando o endereço *Internet Protocol* (IP) uns dos outros, ou utilizando o nome do contêiner, que é resolvido por um serviço de *Domain Name System* (DNS) interno.

O desenvolvimento da aplicação contou com um ambiente baseado em contêineres por meio do *Docker*. Com essa divisão em contêineres são evitados conflitos entre versões de dependências dos aplicativos necessários para a aplicação, pois as dependências são empacotadas juntamente com a aplicação na imagem do contêiner (ANIL, 2022). Um ambiente isolado e configurável facilita a entrada de novos contribuidores do projeto e melhora a experiência de desenvolvimento RedHat (2018). O processo de implantação utilizou a mesma estratégia, facilitando colocar a plataforma em ambiente de produção facilmente, onde o sistema hospedeiro tem como dependência apenas o *Docker* e o *Docker Composer*.

Além da experiência de desenvolvimento, o teste dos códigos dos alunos necessita que a execução ocorra de forma consistente, isso é, sem interferência externa que poderia acontecer por conta de diferentes versões do Octave, programas com dependências conflitantes, atualizações indesejadas; de forma isolada de outras execuções, impossibilitando o código do aluno ter qualquer acesso ao servidor ou a códigos de outros alunos; e de forma segura, pois um código desconhecido pode conter códigos maliciosos, podendo causar danos a máquina do professor ou ao ambiente de execução. Dessa forma, o *Docker* também é usado para executar o código do aluno de forma isolada e consistente.

2.3.4 Sistema de mensageria

Segundo Brito (2019), mensageria é um conceito que define que sistemas distribuídos, possam se comunicar por meio de troca de mensagens ou eventos, sendo estas mensagens “gerenciadas” por um *Message Broker*.

De forma geral, um *message broker* recebe e compartilha mensagens, permitindo que aplicações troquem mensagens e se comuniquem entre si, compartilhando dados ou até mesmo solicitando serviços.

O *RabbitMQ* (VMWARE, 2007) é um dos *message brokers* de código aberto mais populares, contendo dezenas de milhares de usuários, entre eles grandes corporações (RABBITMQ, 2007), e será utilizada no desenvolvimento deste trabalho.Entretanto existem diversas soluções de *message brokers* como o Apache Kafka (APACHE, 2011) e o *Amazon MQ* (AMAZON, 2017) que poderiam ser utilizadas com o mesmo propósito.

Por padrão, o *RabbitMQ* utiliza o protocolo *Advance Message Queuing Protocol* (AMQP), um protocolo aberto leve criado pelo setor corporativo e projetado para ter confiabilidade, segurança e interoperabilidade (NAIK, 2017), e a garantia de entrega das mensagens, o que a torna ideal para comunicação entre sistemas.

O *RabbitMQ* e outros sistemas mensageiros em geral usam os mesmos jargões (RABBITMQ, 2007)

- *Producer*:
  O papel do *producer* é enviar um evento. Um programa que envia um evento para a fila é denominado *producer*, ou produtor. Um exemplo de evento pode ser um *e-mail* a ser enviado, tal evento é gerado pela API e consumido por um serviço especializado em envio de *e-mails*

- *Queue*:
  Uma *Queue*, ou fila, é o canal de comunicação entre o *consumer* e o *producer*. Quando um evento é enviado, ele é armazenado em uma fila. Esta fila contém um identificador, e é normal ter uma fila para cada tipo de evento. Uma fila pode ter diversos produtores e consumidores.

- *Consumer*
  O papel do *consumer* é receber, ou consumir os eventos de uma fila. Um programa que boa parte do tempo espera por eventos é chamado *consumer*.

> A principal ideia por trás de Filas de Trabalho (também conhecido como Filas de Tarefas) é evitar fazer uma tarefa que requer muitos recursos imediatamente e ter que esperar que ela seja concluída. Em vez disso, agendamos a tarefa para ser feita mais tarde. Encapsulamos uma tarefa como uma mensagem e a enviamos para a fila. Um processo de trabalho executado em segundo plano exibirá as tarefas e, eventualmente, executará o

> trabalho. Quando você executa muitos *workers*[1], as tarefas serão compartilhadas entre eles.
>
> Esse conceito é especialmente útil em aplicativos da Web em que é impossível lidar com uma tarefa complexa durante uma pequena janela de solicitação HTTP (RABBITMQ, 2007).

Dessa forma, a execução de um algoritmo pode ser encapsulada como tarefa, enviadas para uma fila e consumidas posteriormente por *worker*.

Uma tarefa pode levar certo tempo pare executar e, nesse intervalo, o *worker* pode ser finalizado ou ter algum problema antes que a tarefa seja concluída. Para evitar que a tarefa seja perdida, o *RabbitMQ* tem o suporte a *acknowledgment*, onde o *consumer* envia uma mensagem para a fila, sinalizando que o evento foi recebido e processado com sucesso. Por padrão, se em 30 minutos o evento não for marcado como realizado, o *RabbitMQ* irá entender que a tarefa falhou, e enviá-la novamente para um *worker* disponível (RABBITMQ, 2007).

O *RabbitMQ* também consegue gerenciar o número de eventos que cada *consumer* recebe por vez. Dessa forma, é possível ter um *worker* realizando mais de uma tarefa ao mesmo tempo, como por exemplo, enquanto o *worker* está aguardando a resposta de uma requisição *web*, ele pode realizar outra requisição. Esta configuração é definida pelo lado do *consumer*, dessa forma, é possível permitir que em uma máquina com mais recursos o *worker* processe diversas tarefas simultaneamente, enquanto em uma máquina com menos poder de processamento o trabalhador execute uma tarefa por vez.

[1] Um *worker* é algo a que se dá a uma tarefa e continua em seu processo, enquanto o trabalhador (ou vários trabalhadores) processa a tarefa em uma *thread* diferente. Quando eles terminam, informam novamente o processo que iniciou a tarefa (SCHAUDER, 2007)

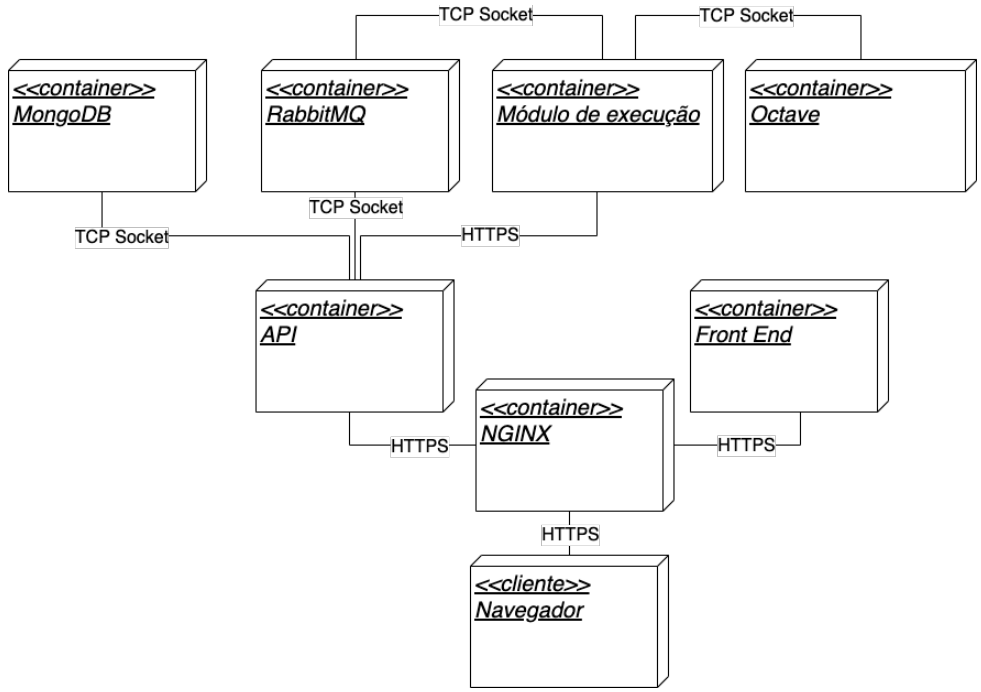
3 DESENVOLVIMENTO

Nessa seção são apresentados os componentes desenvolvidos para a construção do sistema de submissão *online* das atividades computacionais.

3.1 Arquitetura do software

O desenvolvimento da plataforma contou com o uso do *Docker* e do *Docker Composer*, onde cada serviço é executado e exposto através de um contêiner. A arquitetura do sistema representada pela Figura 1.

Figura 1 – Diagrama de implantação



Fonte: Autoria própria (2022).

Com o *Docker Composer* é possível definir ambientes com múltiplos contêineres, e definir a dependência e relação entre eles. Por exemplo, é possível definir que o contêiner da API depende do *MongoDB*, dessa forma, API só será executado quando todas suas dependências estiverem prontas. Estas definições ocorrem em um arquivo denominado `docker-compose.yml`, seguindo o exemplo da Figura 2.

Por padrão, a execução de um contêiner não tem seus arquivos persistidos. Para salvar estes dados, ou permitir que parte do sistema de arquivos do sistema hospedeiro sejam compartilhados com o contêiner, é necessário definir volumes.

No exemplo da Figura 2, o contêiner mongo apresenta dois volumes, o primeiro compartilha o arquivo *env/prod/mongo/mongo-init.js* do sistema hospedeiro para o contêiner do *MongoDB*, com destino em */docker-entrypoint-initdb.d/mongo-init.js*, diretório definido em MongoDB (2016), onde tal arquivo será executado na inicialização do contêiner. Já o segundo volume é

Figura 2 – Exemplo de arquivo *docker-compose.yml*

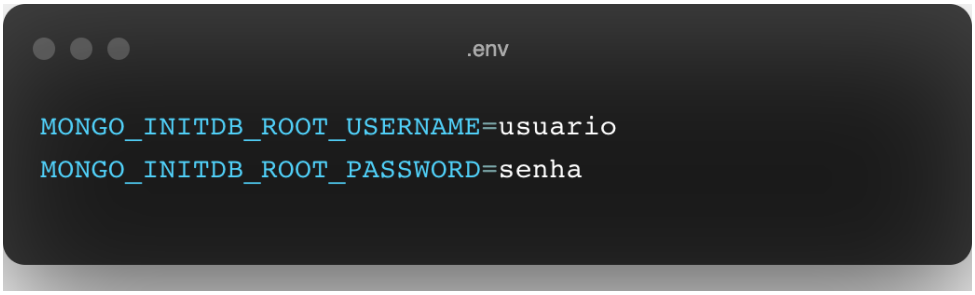
Fonte: Autoria própria (2022).

gerenciado pelo *Docker* no sistema hospedeiro e permite a persistência de dados do contêiner. Este diretório também é definido por MongoDB (2016) e se trata do diretório destinado a persistência de dados do banco de dados.

Também é possível injetar variáveis de ambiente no contêiner. Segundo Fayock (2020), as variáveis de ambiente são valores predeterminados que normalmente são usados para fornecer a capacidade de configurar um valor em seu código de fora de seu aplicativo. O *Docker Composer* realiza a leitura de um arquivo `.env` e essas variáveis ficam disponíveis para uso no arquivo de definição dos ambientes. Dessa forma, informações como credenciais de acesso ao banco de dados, porta, endereços de IP, *tokens* para comunicação entre serviços são definidas como variáveis de ambiente. A Figura 3 apresenta um exemplo de arquivo `.env`, com variáveis referentes as credenciais de acesso ao MongoDB.

Através das variáveis definidas no arquivo `.env` é possível acessá-las no arquivo `docker-compose.yml`, e também definir um valor padrão, caso a variável não seja encontrada.

Figura 3 – Exemplo de arquivo *.env*



Fonte: Autoria própria (2022).

3.2 NGINX

NGINX é um software *open source* para servir aplicações *WEB*, balanceamento de carga, cacheamento, entre outros.

Este serviço é utilizado como porta de entrada do servidor, expondo a porta *80* publicamente e redirecionando o tráfego de acordo com a rota. Para rotas com o prefixo */api* é feito um *proxy* reverso [1], redirecionando o tráfego para a API. As demais rotas são redirecionadas para o módulo *Front End*. Além disso, o serviço força o uso do protocolo *HTTPS*, garantindo uma comunicação segura entre o cliente e o servidor.

3.3 API

Um sistema rapidamente fica complexo, precisando atender várias regras de negócios, validações de entradas, controle de acesso, etc. A API é a porta de entrada para o sistema e, por meio dela, o usuário consegue através do *front end* interagir com o sistema. A API encapsula funções complexas, como a execução de uma resolução, em uma interface simples, e a expõe através de uma requisição *HyperText Transfer Protocol* (HTTP) que são acessadas pela interface do usuário.

A maioria das rotas definidas na API são protegidas, isso é, elas não podem ser acessadas publicamente, e necessitam de um *token* de acesso. Para obter o *token*, é necessário realizar a autenticação. A API expõe uma rota para autenticação, a qual requer o e-mail do usuário e a senha. A partir do e-mail, é recuperado o usuário do banco de dados e, para finalizar a autenticação, a senha informada deve ser comparada com a senha do usuário.

A arquitetura mencionada considera que a senha do usuário esta salva em modo textual no banco de dados, prática que é insegura. Afim de melhorar a segurança do sistema, a senha deve ser salva após ser criptografada. Alguns algoritmos de *hash* populares são *MD5* e *SHA1*. Para validar a identidade do usuário, basta aplicar a mesma função para criptografar a senha

[1] O proxy reverso Nginx atua como um servidor intermediário que intercepta as solicitações do cliente e as encaminha para o servidor apropriado e, posteriormente, encaminha uma resposta do servidor de volta ao cliente (SHINDE, 2021)

informada na requisição e comparar se as duas senhas criptografadas são iguais (KUMAR, 2018).

Caso o usuário entre com a senha correta é gerado um *JSON Web Token* (JWT), o qual contém informações de usuários, tais como o *id*, nome, e se o usuário é um aluno ou professor. Este *token* é enviado na resposta da requisição de autenticação, e é utilizado para validar as futuras autenticações, assim como para definir quais paginas serão renderizadas para o usuário pelo *front end*.

> *JSON Web Token* (JWT) é um padrão aberto (RFC 7519) que define uma maneira compacta e independente de transmitir informações com segurança entre as partes como um objeto *JavaScript Object Notation* (JSON). Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente. Os JWTs podem ser assinados usando um segredo (com o algoritmo HMAC) ou um par de chaves pública/privada usando RSA ou ECDSA. (JWT, 2010)

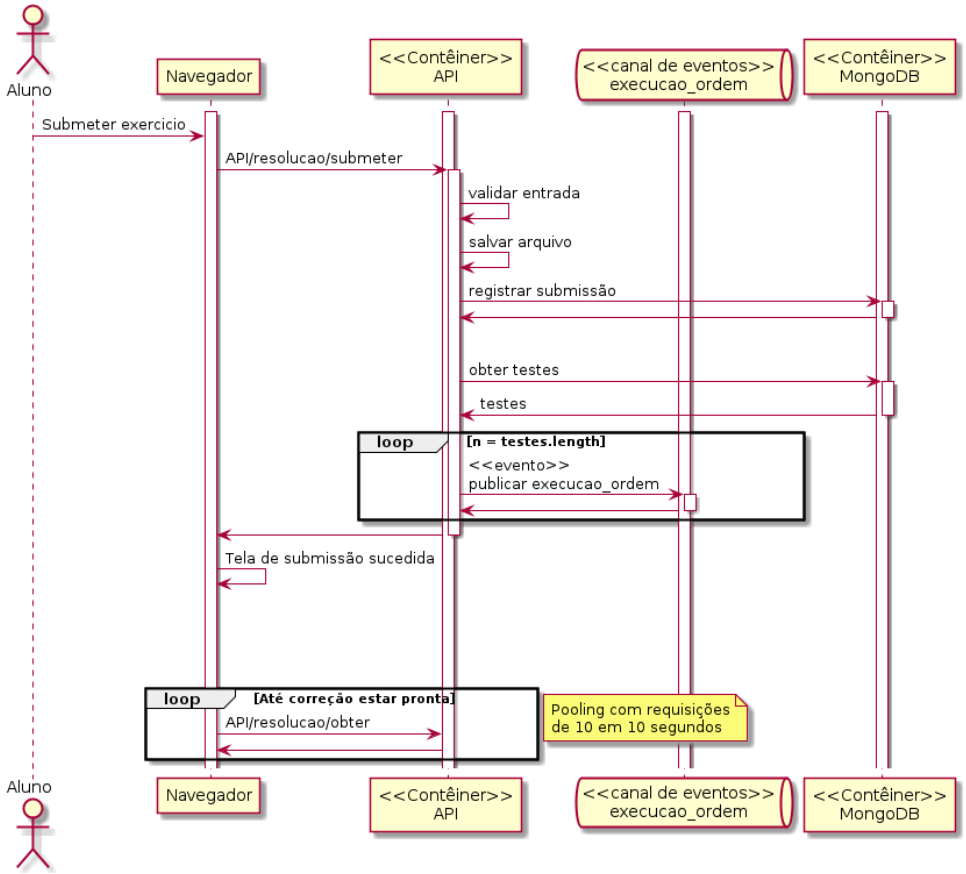
Através de um *middleware* a requisição é interceptada e é realizada a verificação do JWT, verificando se o *token* foi alterado e se o usuário tem acesso determinada rota.

A API expõem rotas para o professor gerenciar exercícios, matérias e testes; assim como rotas para o aluno se matricular e cancelar a matricula em matérias, visualizar exercícios e matérias, e realizar submissões de um exercício. Cada rota tem seus dados validados, para impedir por exemplo a criação de um exercício contendo o nome de função inválido, que não será aceito para execução no Octave; Ou para impedir o aluno a se matricular em uma turma que já está cheia.

Este módulo também possui um *producer* para a fila `execucao-ordem`, e insere um evento quando uma submissão é realizada. O fluxo da requisição de uma submissão é descrito no diagrama de sequência, presente na Figura 4. Dessa forma, a execução do algoritmo é executada pelo *worker* em segundo plano, e a resposta da requisição é devolvida para o navegador.

Além disso, a API também possui um *consumer*, atendendo os eventos do canal `exercicio_resposta`. Quando um novo evento é recebido, a resposta do exercício é comparada com a resposta esperada, cadastrada no teste, e o teste do aluno é marcado como correto ou incorreto. O professor pode alterar esta informação, caso tenha sido identificado de forma errada, mas este processo faz uma triagem inicial de erros comuns cometidos pelos alunos, e o mesmo tem a chance de submeter novamente afim de receber um acerto no teste. Além disso, o professor tem a chance de sobrescrever a nota automática gerada com base nos acertos do teste. O fluxo deste evento é demonstrado no diagrama de sequência, apresentado na Figura 5.

Figura 4 – Diagrama de sequência - Submissão

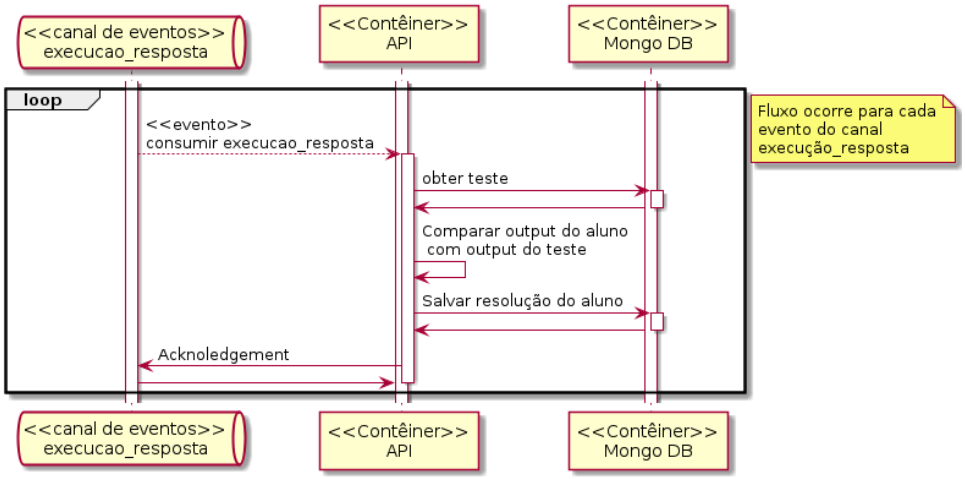


Fonte: Autoria própria (2022).

3.4 MongoDB

Afim de persistir os dados no sistema é necessária a utilização de um banco de dados. O banco de dados escolhido foi o *MongoDB* (MERRIMAN, 2007). O principal motivo da sua

Figura 5 – Diagrama de sequência do evento "execucao-resposta".



Fonte: Autoria própria (2022).

escolha foi o uso de Documentos para armazenamento de dados. Documentos *MongoDB* são compostos por pares de chave e valor (MONGODB, 2007), seguindo a estrutura apresentada na Figura 6.

Figura 6 – Exemplo de documento MongoDB

Fonte: Autoria própria, adaptado de MongoDB (2007).

O *MongoDB* armazena registros em forma de documentos *Binary Javascript Object Notation* (BSON), que é uma representação binaria de documentos JSON, porém com mais tipos de dados, além de permitir mais de uma chave com o mesmo nome MongoDB (2007).

Devido a essa semelhança com o JSON, a integração entre o *JavaScript* e o *MongoDB* ocorre de maneira facilitada, permitindo utilizar as mesmas abstrações da linguagem de programação no banco de dados.

O *MongoDB* oferece um contêiner oficial para o *Docker*, o *mongo* (MONGODB, 2016). Devido a execução de contêineres ocorrerem de maneira isolada, os dados gerados pelo contêiner não serão mantidos quando ele for finalizado, o que se torna um problema para a execução de um banco de dados. Embora existam outras formas de persistir os dados, o *MongoDB* recomenda a utilização de volumes, compartilhando o diretório de dados do sistema *host* com o contêiner.

Ao iniciar um contêiner sem dados, um *script* inicial é inicializado, onde é realizada criação de usuários para acesso no banco de dados, junto com a criação de coleções, que são similares a uma tabela em um banco de dados relacional. O nome e senha de usuário, assim como o nome do banco de dados inicial e porta são conhecidos pelo contêiner do *MongoDB* e da API, desta forma, o outro contêiner consegue realizar a conexão. Todas estas informações são definidas a nível de ambiente, através do *Docker Composer*, como demonstrado na sessão 3.1

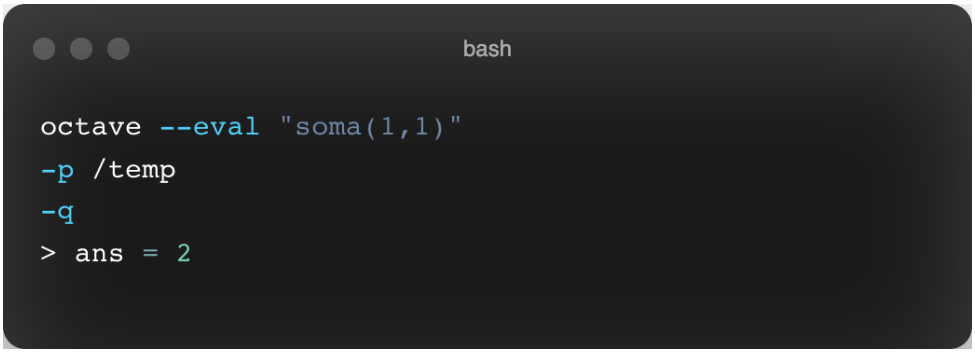
3.5 Octave

O Octave pode ser acessado na forma de *Graphical User Interface* (GUI) ou através da CLI. A primeira normalmente é acessada por usuários finais, devido a facilidade de uso através da interface gráfica. Entretanto, a segunda opção se torna o ideal para integrações com outros

softwares. CLI é uma interface de usuário baseada em texto, usada para executar programas e interagir com o computador. Dessa forma, a CLI permite a criação de *scripts* (LOSHIN, 2021)

Através da CLI do Octave, este contêiner consegue executar o algoritmo com os parâmetros definidos. O comando ilustrado na Figura 7 mostra como poderia ser feita a execução do algoritmo `soma`, encontrado na pasta `/temp`. Adicionalmente, é passada a *flag* `-q` para executar o Octave em modo *quiet*, removendo a mensagem de saudação e a versão ao iniciar o Octave (OCTAVE, 2022).

Figura 7 – Execução de código por meio do Octave CLI.



Fonte: Autoria própria (2022).

O contêiner utilizado foi o *mtmiller/octave*, desenvolvido por Miller (2020), e se trata de um ambiente baseado em *Linux* com a instalação do Octave. Utilizando volumes do *Docker* é possível criar um diretório com o arquivos necessários para a execução dentro deste contêiner. Através do *Docker*, também é possível executar o comando de acordo com a necessidade, alterando, por exemplo, a função a ser executada, os parâmetros de entrada e o diretório do arquivo. Os detalhes deste processo são descritos na sessão 3.6.

3.6 Módulo de execução

Executar um código desconhecido no servidor pode ser extremamente perigoso. Diversas linguagens de programação incluem uma API para acessar recursos do sistema operacional, dando acesso desde ao dispositivo de saída padrão, utilizado para realizar *print*, acesso ao sistema de arquivos, até acesso a rede. Em um cenário normal, é esperado que a linguagem disponibilize esse tipo de interface para comunicação com o sistema operacional, mas isso se torna um grande problema quando estamos executando um código desconhecido, pois a máquina fica exposta a códigos maliciosos presentes no algoritmo do usuário. Por conta disso, o algoritmo deve ser executado isoladamente.

Além disso, os algoritmos devem ser executados de forma consistente, para isso, o mesmo ambiente deve ser utilizado para realizar a execução dos códigos, sem terem variações de softwares instalados, sistema operacional, versões de *softwares* instalados, ou resquícios deixados pela execução de outros algoritmos.

Para atingir esses objetivos, o *Docker* foi utilizado. Com ele, é possível criar um contêiner com o ambiente contendo apenas os softwares necessários para a execução do algoritmo. Para isso, encontra-se alguns desafios:

1. Realizar a execução do contêiner programaticamente;
2. Disponibilizar os arquivos necessários no contêiner do Octave;
3. Permitir que a API comunique com o modulo de execução para executar uma nova execução;
4. Obter a saída do algoritmo executado e devolver para a API;
5. Realizar o gerenciamento do número de instâncias executando em paralelo;

A execução do contêiner pode ser feita através de linha de comando no terminal, porém o *Docker* oferece uma API que permite interagir com o *Docker daemon* [2] (DOCKER, 2013), através de requisições *HTTP* ou um *socket Transmission Control Protocol* (TCP), sendo possível executar programaticamente novos contêineres e receber eventos, como *outputs* de um contêiner ou quando o mesmo for encerrado.

Devido ao módulo de execução e a API estarem em ambientes isolados, eles não possuem acesso ao sistema de arquivos um do outro, impedindo que o módulo de execução consiga acessar diretamente o arquivo de correção do aluno. Uma solução seria a utilização de volumes, recurso do *Docker* que permite o compartilhamento de arquivos entre o sistema operacional e o contêiner. Dessa forma, poderia ser criado um volume compartilhado entre a API e o módulo de execução para armazenar as submissões do aluno.

Outra solução seria o módulo de execução realizar o *download* da submissão utilizando a API. Para isso, é necessária a utilização de um meio de autenticação, para impedir que as submissões fiquem expostas publicamente, como um *token*, similar ao acesso dos usuários. Esse *token* de acesso deve ser conhecido pelos dois contêineres, e pode ser definido como variável de ambiente no momento de inicialização do sistema.

A primeira solução torna o sistema mais acoplado, na qual o módulo de execução necessita ter acesso ao sistema de arquivos da API; já a segunda solução não possui essa dependência, e ainda permite a API fazer um gerenciamento de permissão para permitir que apenas submissões em estado de execução sejam acessíveis pelo módulo de execução. Além disso permite uma auditoria maior, sendo possível visualizar quais arquivos estão sendo acessados pelo *token*. Por conta disso, a segunda solução foi adotada.

Com isso, o módulo de execução tem acesso ao arquivo submetido, o qual deve ser passado para o contêiner do Octave. Este processo é feito através de um volume compartilhado

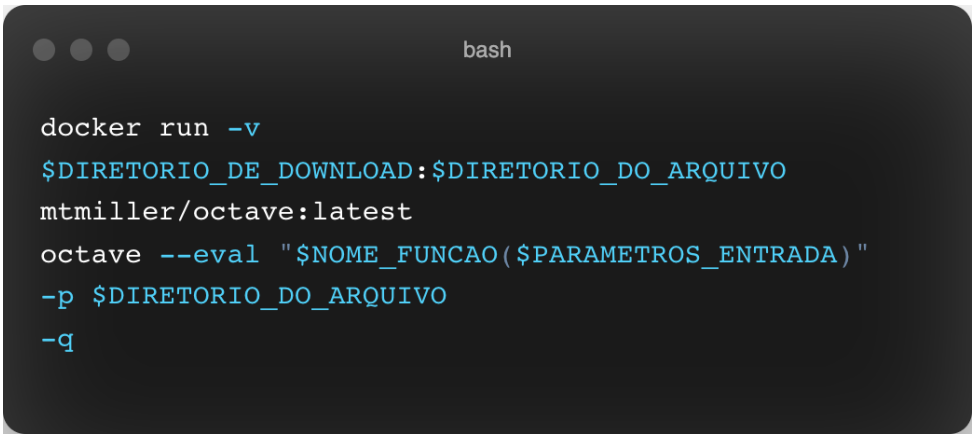
[2] O Docker *daemon* é um processo persistente executado em segundo plano que gerencia os contêineres em um único *host*. É uma execução autossuficiente que gerencia objetos do Docker, como imagens, contêineres, rede e armazenamento. O *daemon* do Docker ouve as solicitações da API REST e executa uma série de operações de contêiner de acordo (CHEN, 2020)

entre ambos os contêineres, montando um diretório dentro do segundo contêiner contendo a submissão do aluno. Ao realizar a execução deste contêiner, a única tarefa do mesmo é executar o Octave através da CLI.

Este serviço foi projetado como um *worker*, desenvolvido em *Node.js*. Este recebe ordens de execução através de eventos do *RabbitMQ*. Os detalhes deste evento e o diagrama de sequência, assim como a solução do problema 5 estão disponíveis na sessão 3.7.

Assim que um evento é recebido pelo canal *execucao-ordem*, é realizado *download* da submissão utilizando a API, utilizando um *token* definido nas variáveis de ambiente do sistema, e executa o contêiner do Octave. Para a geração do contêiner, o comando é gerado utilizando o código apresentado na Figura 8.

Figura 8 – *Template* do comando para execução do Octave.



Fonte: Autoria própria (2022).

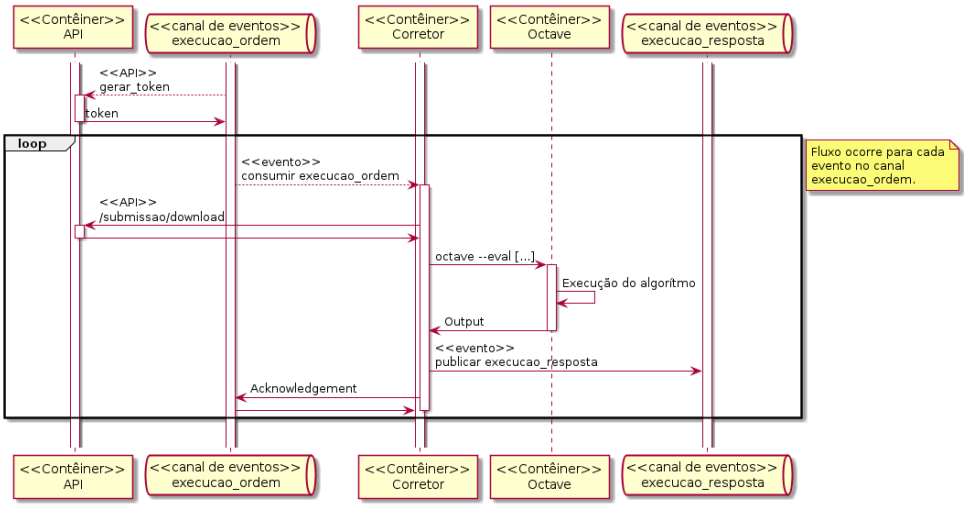
O diretório de *download* e diretório do arquivo são compostos por um diretório base, definido nas variáveis de ambiente, e composto pelo *id* da matéria, do exercício e do aluno, respectivamente. O nome da função e parâmetros de entrada do teste definidos pelo professor na criação de exercícios e dos testes, e esta informação esta presente no evento. O comando `-v`, presente na linha 1 da Figura 8 cria o volume compartilhando parte do sistema de arquivos do contêiner do módulo de execução com o contêiner do Octave. Toda a estrutura de arquivos presente no diretório `$DIRETORIO_DE_DOWNLOAD` será mantida no contêiner do módulo de execução, no caminho `$DIRETORIO_DO_ARQUIVO`.

Como é demonstrado na Figura 7, a saída do algoritmo ocorre na saída padrão do sistema. É possível escrever a saída em um arquivo utilizando os operadores *pipe* do *Linux* (MALIK, 2021), e utilizando os volumes do *Docker* obter esta informação, mas ainda seria necessário identificar quando a execução terminou e o arquivo foi escrito. Outra solução mais viável para isso é a utilização os eventos do *Docker API*, executando funções no módulo de execução toda vez que novos dados chegam do contêiner, e informando quando a execução acabou. Entretanto, ambas as soluções podem gerar carácteres inválidos no meio da solução, o que pode influenciar na comparação do *output* do teste em relação ao código executado. Para isso, foi criada uma lista de carácteres inválidos encontrados em alguns testes, e estes são re-

movidos do resultado. Além disso, é possível que o professor altere o resultado do teste, caso ele apresente alguma anomalia.

Assim que a execução é finalizada, um evento será publicado no canal *execucao-resposta* e o evento *execucao-ordem* é marcado como consumido, enviando um sinal de *acknowledgement* para o *RabbitMQ*, processo descrito em Figura 9.

Figura 9 – Diagrama de sequência do evento "execucao-ordem".



Fonte: Autoria própria (2022).

3.7 *RabbitMQ*

Com a execução em massa de exercícios, podem-se deparar com sobrecargas e ou problemas de desempenho. Para isso se torna necessário gerenciar a quantidade de execuções simultâneas, utilizando por exemplo uma estrutura *First In First Out* (FIFO), onde o primeiro exercício a ser solicitado para ser corrigido é o primeiro a ser corrigido. Utilizando o *RabbitMQ* é possível criar essa estrutura, para a correção. Utilizando dois canais de eventos é possível realizado a orquestração da correção.

3.7.1 Canal *execucao-ordem*

Este canal recebe eventos para realizar a execução de um algoritmo. O conteúdo desse evento é um JSON contendo informações do aluno, do exercício e do teste. Cada evento resulta em uma execução.

Este evento é publicado pela API, sempre que um aluno enviar uma submissão e é consumido pelo módulo de execução. A quantidade de eventos gerados é igual a quantidade de testes cadastrados para o exercício.

3.7.2 Canal *execucao-resposta*

A função desse canal é salvar os dados de execução. Para isso, um evento é gerado pelo módulo de execução, contendo o *output* gerado pelo algoritmo do aluno, informações do teste e do aluno. O *consumer* da API recebe esse evento e realiza o processamento do mesmo.

Vale ressaltar que esta arquitetura permite múltiplos *producers* e *consumers*, dessa forma, seria possível ter diversas instâncias do módulo de execução, presentes em um único servidor ou espalhados por diferentes máquinas. Além disso, o *RabbitMQ* permite definir o número de eventos a serem consumidos simultaneamente por cada *consumer*, permitindo a execução simultânea de mais de uma execução. A nível de testes, foi definido uma única instância do módulo de execução.

3.8 Front End

O *Front End* é executado do lado do cliente, ou seja, pelo navegador do usuário. Este serviço expõe páginas *web* com o *template* da aplicação, e utiliza a *API* para popular os dados necessários em cada página, e para controlar o acesso de cada usuário.

O *Front End* foi desenvolvido utilizando o *framework VueJs*, uma ferramenta popular para o desenvolvimento de interfaces (KROTOFF, 2020), componentes e páginas web responsivas, ou seja, que se adaptam em qualquer tipo de dispositivo, e com uso da biblioteca Vuetify (VUETIFY, 2016) é possível utilizar os componentes, botões e ícones do Material Design, padrão adotado por grandes empresas e desenvolvido pelo Google (2014). Esse contêiner expõe uma *Single Page Application* (SPA) para o *NGINX*. Essa página possui *scripts* para carregar o conteúdo dinamicamente através das requisições da API, preenchendo a página com informações do aluno/professor, exercícios e entre outros dados.

Através do *token* de autorização, comentado na seção 3.3, o *Front End* tem como renderizar conteúdo específico para o cargo do usuário. O conteúdo gerado para o professor tem sessões para a administração de matérias e exercícios, como por exemplo alterar matérias e criar testes para determinado exercício.

Ao acessar o site, o usuário precisa realizar a autenticação utilizando o formulário de login, o qual redireciona o usuário para o portal do aluno ou do professor.

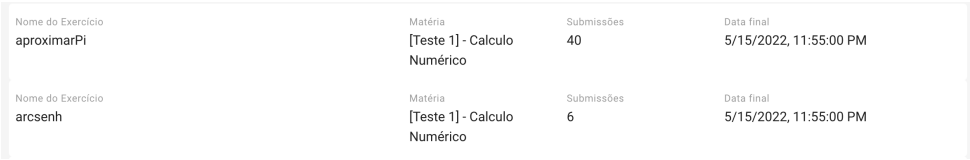
3.8.1 Professor

No portal do professor, o usuário tem acesso as disciplinas e aos exercícios cadastrados pelo usuário, ilustrados pelas Figuras 10 e 11, respectivamente.

Figura 10 – Fragmento da tela de matérias disponibilizada para professores

Fonte: Autoria própria (2022).

Figura 11 – Fragmento da tela de exercícios disponibilizada para professores

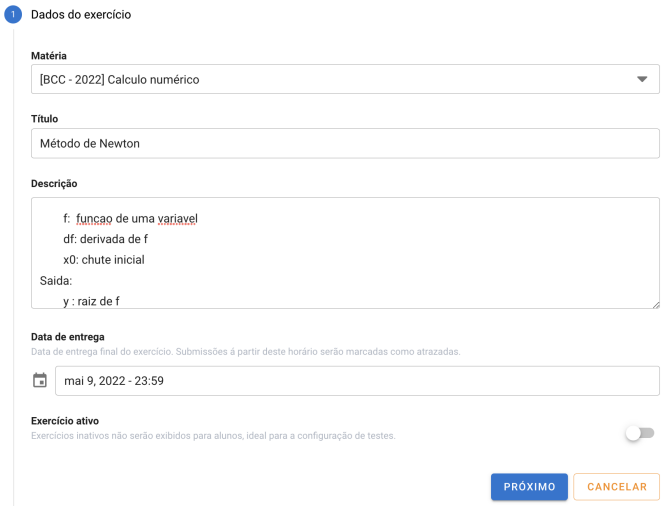


Fonte: Autoria própria (2022).

A partir da sessão de Matérias é possível criar ou editar uma matéria, clicando na mesma, e visualizar as informações da matéria, como capacidade, vagas disponíveis, e a listagem de alunos matriculados, e clicando no botão de criação de matérias, um formulário solicita o registro da nova matéria.

Já na página com a listagem de exercícios o professor pode criar um novo exercício, onde são solicitados os dados do exercício e dados da função, como mostra as Figuras 12 e 13, ou ainda navegar para um exercício, onde são exibidas as informações do mesmo, junto com a listagem de submissões, apresentada em Figura 14.

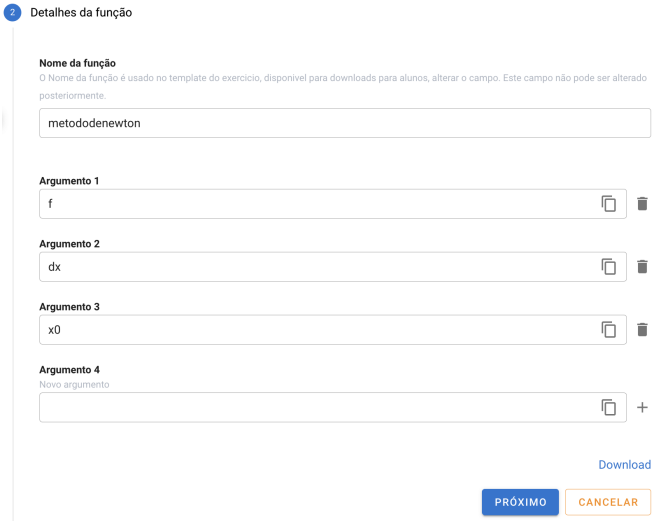
Figura 12 – Cadastro de exercício - Dados do exercício



Fonte: Autoria própria (2022).

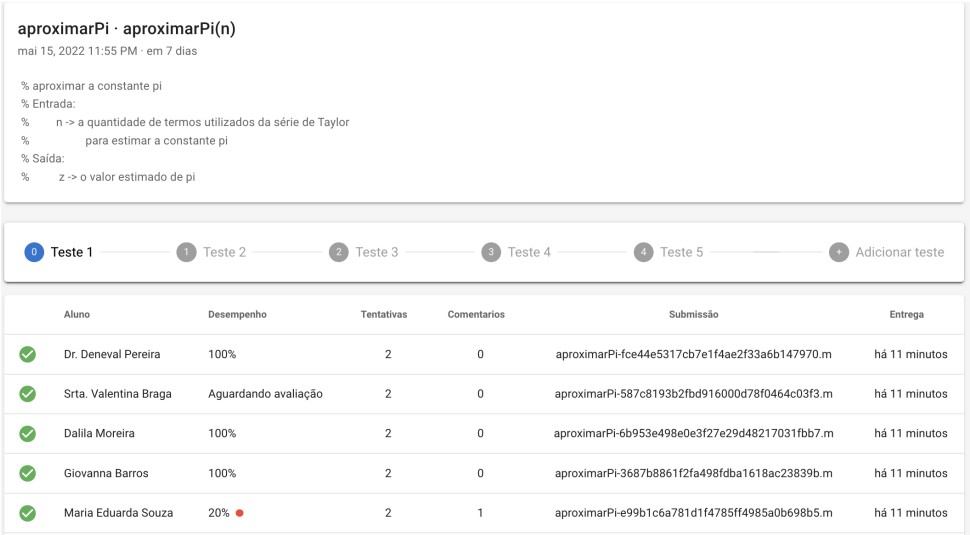
Vale notar que o cadastro do exercício é feito em três etapas. Na primeira são solicitados dados do exercício, como título, descrição, data de entrega, e se o exercício é visível para o aluno ou não. Já na segunda etapa os dados referentes a função submetida são solicitados, definindo a assinatura da função através do nome da função e dos parâmetros de entrada. Na terceira etapa, o professor é apresentado a tela para navegar para o exercício, onde pode criar os testes. Com esses dados das duas primeiras etapas, é possível realizar a criação de

Figura 13 – Cadastro de exercício - Dados da função



Fonte: Autoria própria (2022).

Figura 14 – Detalhes de exercício



Fonte: Autoria própria (2022).

um *template*, disponível para o aluno baixar e realizar a sua submissão usando esta estrutura, que se trata de um arquivo `.m` contendo a descrição do exercício e a assinatura da função, como mostra a Figura 15. A partir do *template*, o aluno consegue desenvolver o exercício dentro dos padrões estabelecidos pelo professor, e ainda o aluno tem a descrição do exercício como comentário no algoritmo, facilitando a consulta de informações definidas pelo professor para a realização do exercício.

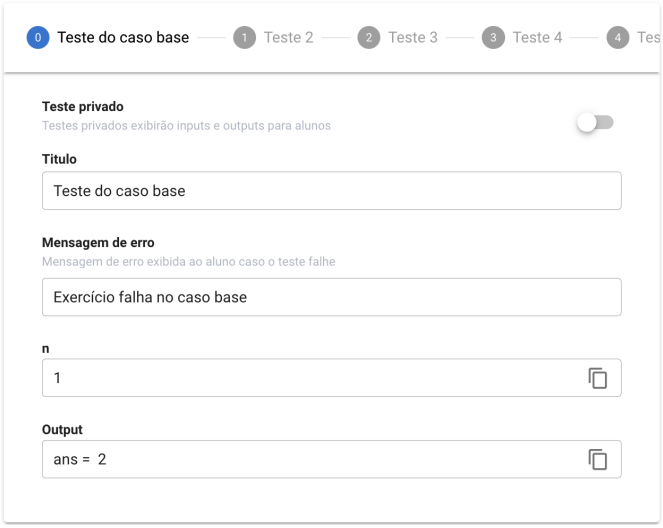
Ao abrir as configurações do exercício, é apresentado para o professor mais detalhes do exercício, junto com a possibilidade de edição dos atributos, e são exibidos os testes do exercício, como é mostrado na figura 16. Um teste é sempre associado a um exercício, e define os parâmetros para a execução do algoritmo do aluno, assim que o mesmo for submetido. Um teste pode ser publico ou privados, o primeiro apresenta os parâmetros de entrada e saida para o aluno, e podem auxiliar o aluno a desenvolver o seu algoritmo; já testes privados tem essas

Figura 15 – *Template* do exercício

Fonte: Autoria própria (2022).

informações ocultas, e ao submeter o exercício o aluno tem um feedback apenas se o teste passou ou não. Adicionalmente, um teste contem uma mensagem de erro, que é apresentada para o aluno sempre que o teste falha, podendo dar sugestões de como corrigir a implementação, ou apontar erros que causam determinado erro. Por fim, testes também possuem propriedades como titulo, para ajudar na identificação do teste; e parâmetros de entrada e saida, que são necessários para a execução do algoritmo do aluno.

Figura 16 – Exibição de testes de exercício



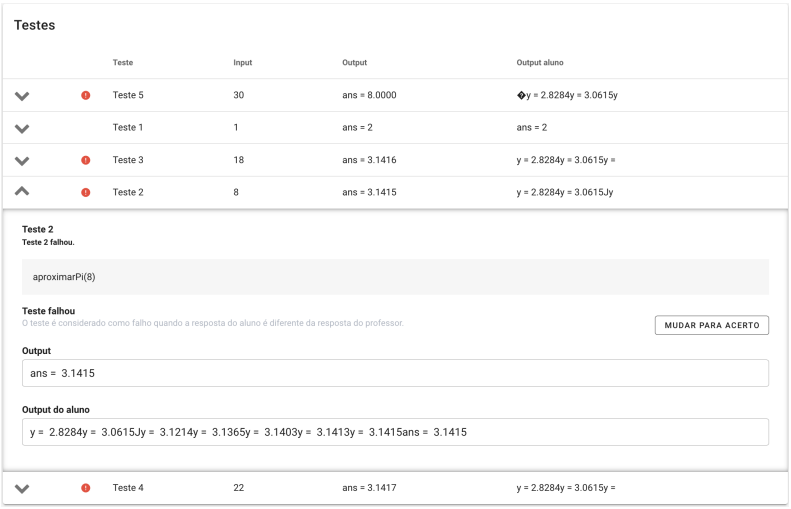
Fonte: Autoria própria (2022).

O professor ainda conta com a tela de correção do exercício, onde é possível visualizar a submissão do aluno, os dados de execução em cada teste, o *output* do aluno e se o *output* é o esperado no teste, e também é possível sobrescrever a nota para a resolução, a qual é gerada automaticamente de acordo com o número de erros e acertos nos testes. Também é possível adicionar comentários na resolução. Esta tela ainda conta com botões para navegar rapidamente entre submissões de um mesmo exercício, facilitando o processo geral de correção. Tal processo demonstrado nas Figuras 17, 18 e 19.

Figura 17 – Correção da submissão - Submissão do aluno

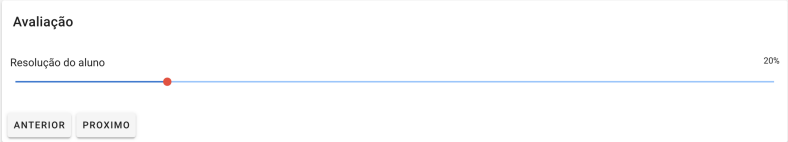
Fonte: Autoria própria (2022).

Figura 18 – Correção da submissão - Dados de execução



Fonte: Autoria própria (2022).

Figura 19 – Correção da submissão - Avaliação



Fonte: Autoria própria (2022).

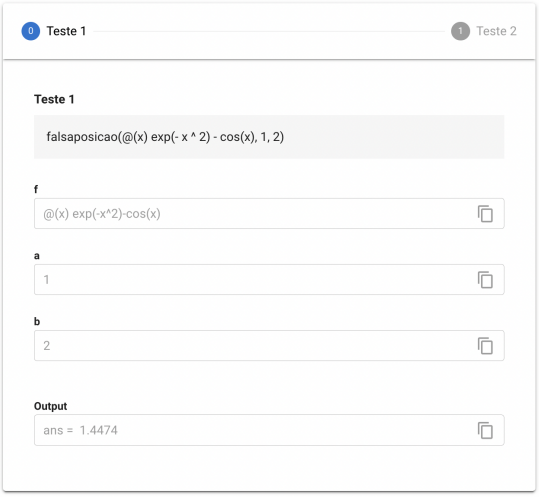
3.8.2 Aluno

A partir da página inicial, o aluno recebe a listagem de disciplinas no qual o mesmo está matriculado, assim como a listagem de disciplinas disponíveis para matricula. Para se matricular em uma disciplina é necessário entrar com a senha definida pelo professor no momento do cadastro da disciplina.

Dentre as opções disponíveis, como a matrícula em disciplinas, visualização de detalhes de matérias, destaca-se a visualização de exercícios e testes e a submissão de um exercício.

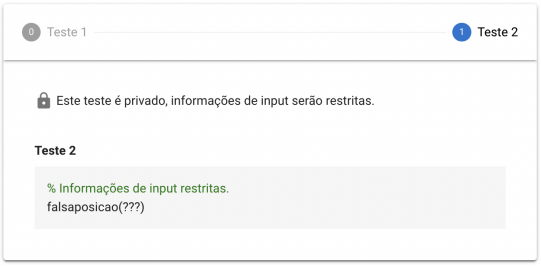
A visualização de exercício exibe informações como o título e descrição do exercício, prazo de entrega, e principalmente os testes. O aluno pode visualizar os dados de entrada e saída de testes públicos, como é mostrado em 20 e 21.

Figura 20 – Visualização de teste público



Fonte: Autoria própria (2022).

Figura 21 – Visualização de teste privado



Fonte: Autoria própria (2022).

A submissão de uma resolução requer o envio de um arquivo, e após o envio, o aluno recebe uma nova tela com a informação dos testes, processo ilustrado nas Figuras 22, 23 e 24. Assim que a submissão for enviada, serão exibidas as telas com a submissão do aluno e as informações de execução, semelhante a visão do professor, demonstradas nas Figuras 17 e 18.

Para notificar o aluno sobre o estado da execução, o cliente precisa solicitar as informações para o servidor, pois na arquitetura desenvolvida, a comunicação feita para a API utiliza o método HTTP, onde a requisição é sempre iniciada pelo cliente. Logo que o aluno submete o código, o *Front End* começa a realizar requisições para obter as informações da submissão, verificando se a mesma foi corrigida. Através da técnica de *pooling* é possível realizar requisições em um intervalo de tempo para solicitar essas informações, até que o exercício tenha sido executado.

Outras técnicas poderiam ser utilizadas para notificar a interface sobre o estado de execução, como por exemplo o uso de *websockets*, onde é criado um canal de conexão entre o cliente e o servidor, onde ambas as partes podem enviar mensagens. Dessa forma, a interface não precisaria ficar solicitando dados em um intervalo de tempos, pois o servidor notificaria a mesma com os dados atualizados.

Logo que o exercício é executado, o aluno tem acesso aos dados de execução, assim como o professor. Os testes públicos têm os parâmetros de entrada e saída visíveis para o

Figura 22 – Submissão de exercício - Envio de arquivo

Fonte: Autoria própria (2022).

Figura 23 – Submissão de exercício - Exercício submetido

Fonte: Autoria própria (2022).

Figura 24 – Submissão de exercício - Exercício executado

Fonte: Autoria própria (2022).

aluno, permitindo que ele faça testes locais com os mesmos valores. Já os testes privados são fechados para o aluno, pois ele não tem acesso aos parâmetros de entrada e de saída e, dessa forma, apenas a mensagem de erro é mostrada para o aluno. Após a submissão do exercício, execução e avaliação pelo servidor, o aluno tem ao feedback do professor, como nota final e comentários do exercício.

4 TESTES E RESULTADOS

Para validar a plataforma, foram realizados testes simulando uma turma de cálculo numérico com 40 alunos, e utilizando um *script* em *Node.js* foi possível automatizar a criação da turma, exercícios e testes pelo professor, e também automatizar a submissão dos exercícios pelos alunos. Neste capítulo é analisada a funcionalidade de plataforma desenvolvida, utilizando diferentes cenários para tentar analisar como a plataforma se comportaria dentro de uma sala de aula, com informações reais. É importante estes testes serem realizados em um ambiente controlado, para observar a estabilidade da plataforma em um cenário que se aproxima de um cenário real, de forma que, em caso de falhas, o sistema possa ser corrigido sem afetar o desempenho de uma turma real e, consequentemente, possa vir a atrapalhar o desenvolvimento da matéria.

A plataforma desenvolvida foi implantada servidores da Vultr (2014), em uma máquina com 4 *gigabytes* de memória *Random Access Memory* (RAM), e dois *Virtual Central Processing Unit* (vCPU), e os testes foram feitos utilizando a plataforma em um ambiente de produção.

Os testes contaram com diversos dados obtidos pela plataforma *Octave correction tool* Silva *et al.* (2020). O repositório disponibilizado pelo projeto conta com diversos exercícios, contendo informações de entrada de testes, disponíveis em um arquivo *Comma Separated Values* (CSV), submissões de alunos e do professor ou monitor. As submissões contém em sua nomenclatura o nome do exercício, seguido pelo Registro Acadêmico (RA) do aluno, a palavra chave "*default*"para submissões do professor, ou "MONITOR"para submissões enviadas pelo monitor.

A plataforma *Octave correction tool* não guarda o *output* dos testes, invés disso, ele executa o arquivo do professor a fim de obter a saída do mesmo. Por conta disso, foi necessária a execução dos testes antes de realizar o cadastro na plataforma. Para a execução dos testes, o módulo de execução foi adaptado provisoriamente, de forma que o *script* de testes e o módulo se comuniquem, utilizando a mesma lógica de filas apresentada no capítulo 2.3.4. Dessa forma, o *script* de testes recebe os dados de saída do algoritmo do professor e armazena os resultados completos de testes, contendo as entradas disponíveis originalmente no repositório, e os dados de saída recém obtidos.

Utilizando expressões regulares, foi possível extrair informações dos exercícios, como por exemplo, a assinatura da função, RA, descrição do exercício. Todas as submissões tiveram que passar por um processo de padronização, onde as submissões tiveram suas assinaturas adaptadas, removendo o RA do aluno. Além disso, todas as ocorrências de RA do aluno do arquivo, mesmo que presentes no código em forma de comentário, foram removidas, não sendo possível vincular um aluno com uma submissão. Através disso, foi gerado um novo repositório de submissões, com os dados padronizados e anônimos, isto é, sem dados que possam identificar os alunos.

O módulo de testes utilizou a biblioteca *faker-js* (FAKERJS, 2022) para a criação de alunos com nomes e endereços de e-mail falsos. Como as submissões não possuem identificador

do aluno, quando um dos alunos gerados irá submeter uma resolução, uma das resoluções disponíveis é atribuída aleatoriamente para ser submetida. Uma submissão pode ser enviada mais de uma vez por alunos diferentes, e em diferentes submissões um aluno pode submeter diferentes arquivos.

Com os dados dos exercícios, testes, submissões e alunos foi realizada a simulação de uma turma de Cálculo Numérico. A Tabela 1 sumariza os exercícios disponíveis utilizados para testes. Utilizando a API, o *script* se autentica como professor, verifica a existência de matéria especificada, se a mesma não existir ela é criada. O mesmo processo ocorre para cada exercício e teste. Para cada aluno gerado é realizado a autenticação, assim como a matrícula na disciplina.

Tabela 1 – Sumário do repositório de dados gerado.

Exercício	Número de testes	Arquivos de submissão distintos	Execuções necessárias
aproximarpi	5	23	200
arcsenh	5	28	200
falsaposicao	4	22	160
GaussSeidel	4	18	160
InterBilinear	4	19	160
newton	5	21	200
pontofixo	6	16	240
arcsenh	5	28	200
Total	38	175	1520

Fonte: Autoria própria (2022).

A fim de coletar dados para a análise, foram realizados ajustes ao módulo de execução e a API, registrando o momento em que o teste é inserido na fila, o momento que a execução é inicializada e o momento em que a execução termina. Com esses dados são geradas informações como o tempo que um teste levou para ser executado, o tempo de espera na fila, e também é possível calcular o tempo que uma resolução demorou para ter todos os seus testes executados.

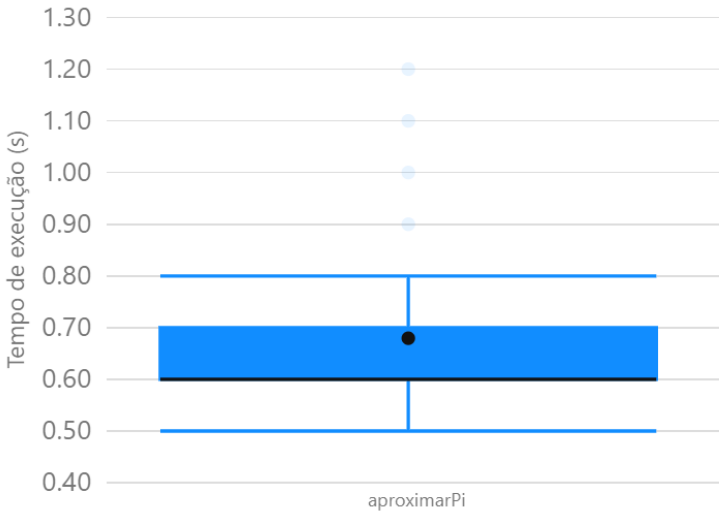
Foram realizadas simulações de submissão com todos os exercícios apresentados anteriormente na Tabela 1. Por questões de apresentações, as seções a seguir apresentam apenas os gráficos e detalhes de um exercício, porém todos os exercícios testados apresentarem os mesmos padrões dos apresentados a seguir.

4.1 Teste 1

A submissão de exercícios em uma turma real teoricamente esta espalhada em um intervalo de diversas horas, se não dias. Isto é, os alunos não enviam todos os exercícios de uma única vez. A fim de testas os limites da plataforma, foi simulado o pior caso, que seria onde todos os alunos submetem suas resoluções para um exercício simultaneamente, ou em um intervalo curto de tempo.

O primeiro exercício executado foi para a aproximação de π através da série de Taylor. Esta função tem como entrada um valor n determina o número de termos da série para a aproximação. O Gráfico 1 mostra um gráfico do tipo *boxplot* onde é apresentado o tempo de duração das execuções junto com a distribuição dos seus valores. É possível identificar que o tempo de execução deste algoritmo está concentrado no intervalo entre 0.6 segundos e 0.7 segundos. Pode-se observar casos que se afastam do intervalo, tendo instâncias de execuções que levaram desde 0.5 segundos e outras que chegaram a levar 1.2 segundos para executar.

Gráfico 1 – Tempo de execução do exercício Aproximar PI.



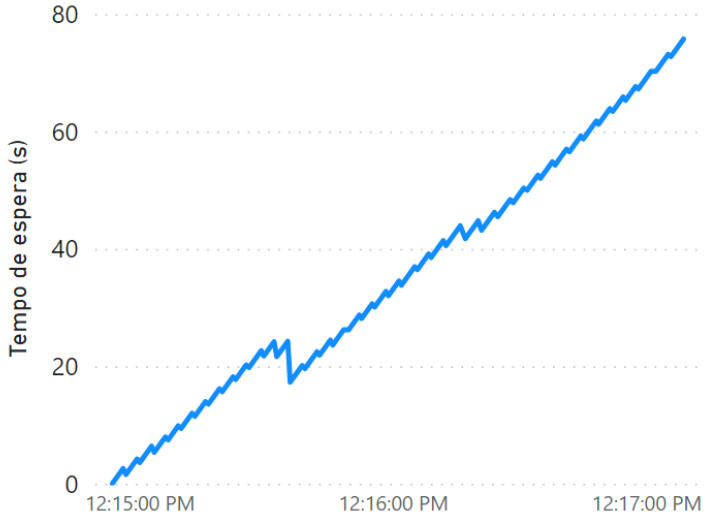
Fonte: Autoria própria (2022).

Além do tempo em que o código levou para ser executado, outra métrica importante é o tempo em que um exercício passou na fila, esperando ser atendido por um *worker*. Para isso, foi calculada a diferença entre o momento que a submissão foi submetida e o momento que ele foi recebido pelo *worker* para iniciar a execução. O Gráfico 2 mostra que os primeiros exercícios submetidos foram rapidamente atendidos, porém com o passar do tempo o *worker* levou mais tempo para atendê-los, chegando a tempos de espera superiores a um minuto.

Com isso, pode-se assumir que a frequência de entrada de dados é maior do que o *worker* consegue atender, isto é, tarefas são adicionadas a fila mais rapidamente do que elas são atendidas, causando um aumento no tamanho da fila.

Uma fila grande tem total impacto no tempo total que uma execução leva para ser executada e, consequentemente, aumenta o tempo de espera do aluno e do professor. O Gráfico 3 mostra uma *timeline* mostrando o tempo total de execução de uma resolução, que é dado pela data de agendamento do primeiro teste de uma submissão até a finalização da execução do último teste da resolução, no caso deste exercício, a execução completa de uma resolução se caracteriza pela execução de cada um dos cinco testes cadastrados no exercício. É notado o aumento da duração do tempo total de uma resolução de acordo com o tamanho da fila, que acompanha o mesmo padrão do Gráfico 1, e pode-se notar que a queda no tamanho da fila

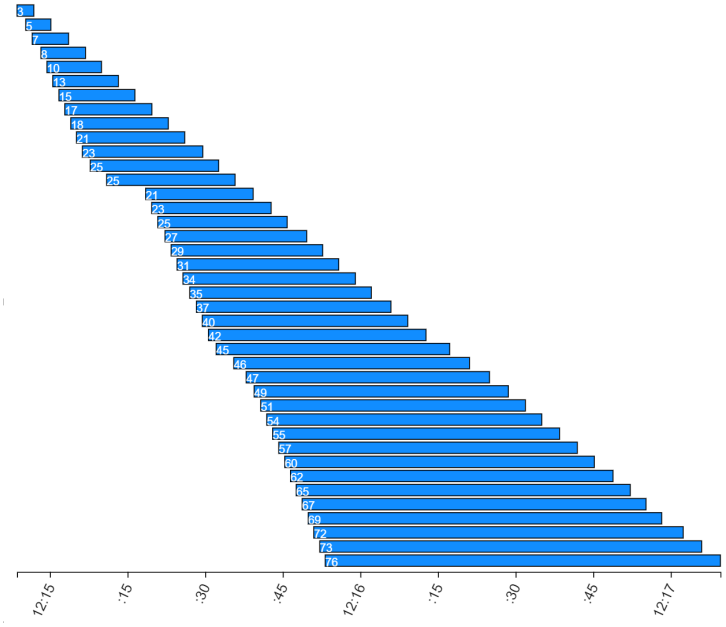
Gráfico 2 – Tempo de espera da execução ao decorrer do tempo.



Fonte: Autoria própria (2022).

apresentada no primeiro gráfico tem impacto direto no segundo, diminuindo o tempo total de execução da resolução.

Gráfico 3 – Tempo total de execução de resolução.



Fonte: Autoria própria (2022).

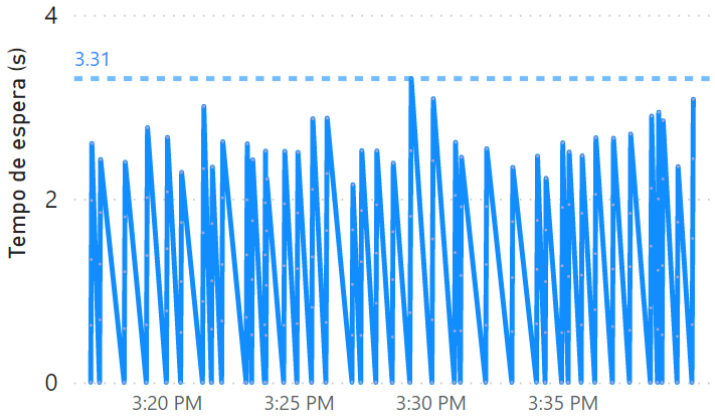
A fim de tentar reduzir o tamanho fila e, consequentemente, melhorar o tempo de execução, são apresentadas duas alternativas: diminuir a frequência de entrada de dados na fila; e aumentar a frequência em que as tarefas são processadas. As seções a seguir apresentam os resultados para ambas as alternativas.

4.2 Teste 2

Além do pior caso, apresentado anteriormente, também foi simulado um caso que se assemelha com um comportamento em uma turma real, onde os exercícios são submetidos no decorrer de um período e, consequentemente, diminuindo a frequência de novas tarefas adicionadas a fila. Para realizar esta simulação, foi adicionado um *delay* entre a submissão de cada aluno. O tempo de espera entre as submissões é escolhido aleatoriamente, onde alunos podem enviar suas submissões em sequência, ou ter um intervalo de até um minuto. Vale ressaltar que, em uma turma de aula real, o intervalo entre submissões dos alunos depende de inúmeros fatores, e não é trivial simular esse fator com precisão. Além disso, em uma turma com atividades semanais, como é proposto neste trabalho, o aluno tem 7 dias para realizar a sua submissão, um intervalo muito maior do que o proposto por este teste e, consequentemente o cenário real geraria uma carga ainda menor para a fila e para o módulo de execução. Desta forma, o teste proposto apresenta um caso mais realista que o pior caso, mas ainda teoricamente pior do que o observado em uma turma real.

Através do segundo teste é observado que tempo de cada execução tem o mesmo comportamento apresentado no Gráfico 1 do teste anterior, com o tempo de execução médio de 0.65 segundos. Entretanto, pode-se observar através do Gráfico 4 que o tempo de espera não teve grandes picos, e o maior tempo de espera registrado durante a execução do exercício foi de 3.31 segundos, e que o módulo de execução, na maioria dos casos, conseguiu finalizar a execução de todos os exercícios da fila antes de receber mais carga, diferentemente do teste anterior. Este comportamento é esperado neste teste, dado que o *worker* tem um tempo maior para processar as tarefas.

Gráfico 4 – Tempo de espera da execução ao decorrer do tempo.

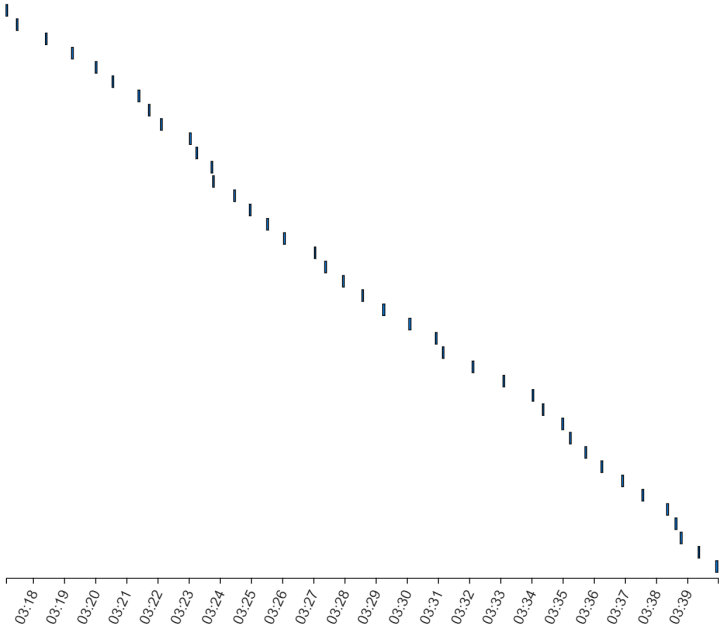


Fonte: Autoria própria (2022).

Em adição a isso, através da *timeline* presente no Gráfico 5, é possível observar como o intervalo entre as submissões afeta o tempo de espera, pois com as submissões dispersas em um intervalo de tempo maior, o *worker* teve tempo de processar as atividades pendentes e ficar

sem carga, e assim que uma nova submissão é enviada, o mesmo esta pronto para executar o algoritmo.

Gráfico 5 – Tempo total de execução de resolução.



Fonte: Autoria própria (2022).

4.3 Teste 3

Não é possível garantir a frequência de entrada de submissões se comporte como o segundo teste. Em diversas situações, a plataforma pode encontrar uma carga de trabalho maior, devido a diversas submissões submetidas em um curto intervalo de tempo, dessa forma, as tarefas da fila devem ser processadas mais rapidamente, e para isso é possível aumentar o número de instâncias do módulo de execução. Para realizar esta simulação, este teste apresenta o teste número 1, apresentado anteriormente, com dois módulos de execução trabalhando em paralelo, em busca de aumentar a frequência com que as tarefas da fila são processados e, consequentemente, ter uma fila menor.

O módulo de execução tem dependência do *RabbitMQ* e da API, desde que o *worker* consiga se comunicar com ambos os serviços, é possível que a execução do mesmo ocorra no mesmo servidor, ou em um servidor distinto.

Devido a utilização de contêineres a plataforma pode ser facilmente escalável através do *Docker Composer*, é possível replicar um contêiner e ter diversas instâncias. Através do comando apresentado em Figura 25 a plataforma é iniciada com duas instâncias do módulo de execuções. Cada uma dessas instâncias recebe um identificador único nas variáveis de ambiente, e esse identificador é registrado junto com os dados de cada execução.

Submetendo os testes com a plataforma inicializada com dois módulos de execução pode-se observar a partir do Gráfico 6 que a carga de trabalho foi distribuída entre os dois

Figura 25 – Comando de inicialização da plataforma com escala do módulo de execução.

Fonte: Autoria própria (2022).

workers, e, consequentemente, o tamanho da fila teve maior resistência para crescer, em comparação com o primeiro teste.

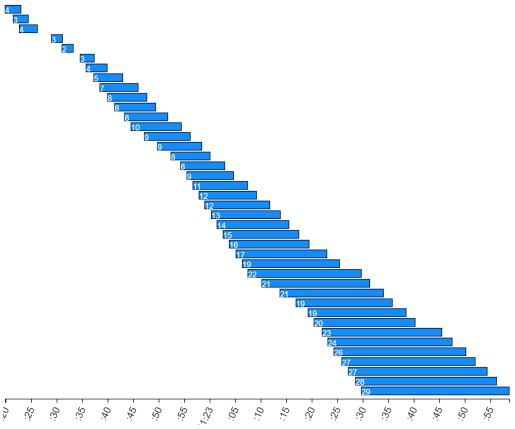
Gráfico 6 – Tempo de espera da execução ao decorrer do tempo.

Fonte: Autoria própria (2022).

Também é notado que o tempo total de execução da resolução é inferior em relação ao primeiro teste. A presença de dois ou mais módulos de execução permitem que execuções sejam realizadas em paralelo, isso é, duas ou mais execuções ocorrem ao mesmo tempo, desta forma, aumentando a frequência que tarefas são executadas e consequentemente saem da fila. O Gráfico 7 apresenta o tempo de execução total de uma resolução, enquanto a Gráfico 8 permite visualizar a execução em paralelo de tarefas da primeira resolução, do início ao final da execução, ainda é possível ver que, assim que um módulo de execução finaliza uma tarefa, outra começa ser executada.

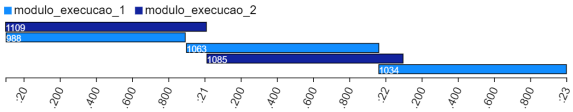
O aumento do número de módulo de execuções resultou em um tempo de execução superior para cada execução em relação aos dois testes anteriores. O comportamento pode ser observado no Gráfico 9, e pode ser associado a um maior uso dos recursos da máquina, dado que ambos os módulos estão executando no mesmo ambiente. Ainda que o tempo médio de execução tenha aumentado, o tempo total levado para executar uma resolução é inferior ao primeiro teste, e estes valores poderiam ser otimizados com os módulos de execução hospedados em uma máquina dedicada, sem disputarem recursos entre si.

Gráfico 7 – Tempo total de execução de resolução.



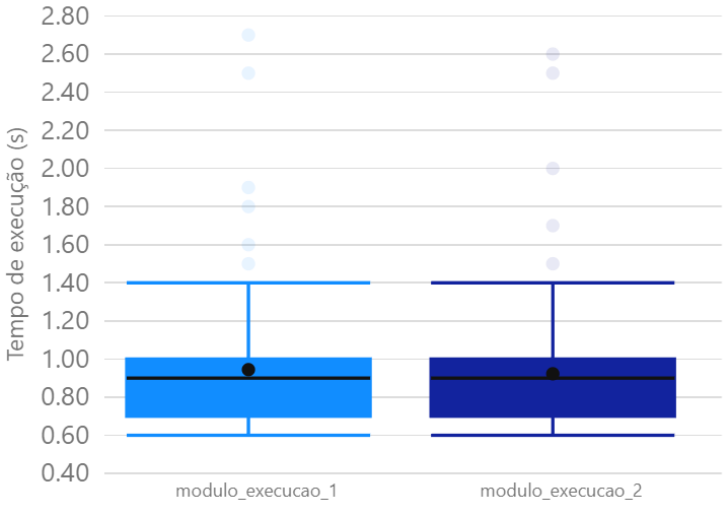
Fonte: Autoria própria (2022).

Gráfico 8 – *Timeline* de execuções da primeira resolução



Fonte: Autoria própria (2022).

Gráfico 9 – Tempo de execução para cada módulo de execução.



Fonte: Autoria própria (2022).

5 CONCLUSÕES

O desenvolvimento prático de atividades de Cálculo Numérico é essencial para garantir a fixação do conteúdo pelo aluno, entretanto, a correção de atividades gera carga extra para o professor devido a diversos fatores, entre eles, a necessidade de executar cada algoritmo diversas vezes para cobrir casos de testes distintos a fim de garantir que o algoritmo do aluno se comporta corretamente. Por conta disso, muitas vezes o professor demora para realizar a correção das atividades dos alunos, ou até mesmo é diminuída a frequência de atividades para que a correção seja feita em tempo hábil.

A plataforma desenvolvida nesse trabalho facilita parte desse processo através da execução em lote das atividades, e na qual o algoritmo submetido pelo aluno é automaticamente executado em todos os casos de testes definidos pelo professor, sem que haja a necessidade de o professor realizar manualmente a execução do código e testar diferentes casos.

Utilizando a plataforma desenvolvida, o professor consegue navegar entre as submissões e realizar a correção dos exercícios, que neste ponto já foram executados e fornecem diversos dados para auxiliar o professor no processo de correção. Para realizar a correção do exercício, o professor tem acesso a submissão do código do aluno, junto com os dados de execuções, como entrada de cada um dos testes, e o *output* gerado pelo Octave. Dessa forma, o professor pode adicionar comentários diretamente na submissão do aluno, e sobrescrever a nota do aluno, gerada automaticamente de acordo com os acertos e erros nos testes.

Ainda que os testes apresentados ocorram em um ambiente controlado, pode-se observar que em todos os casos, a plataforma conseguiu realizar a execução dos testes submetidos, desta forma, cumprindo o objetivo proposto de implementar a plataforma para a execução em lotes de algoritmos desenvolvidos na linguagem Octave. Em adição a plataforma gera um feedback instantâneo para o aluno com base nos testes informados pelo professor, e ainda recusa a submissão de exercícios fora do padrão especificado, por exemplo, a assinatura da função não esteja de acordo com a assinatura especificada pelo professor. Dessa forma, o aluno tem a oportunidade alterar sua submissão e corrigir o código conforme o necessário.

A plataforma mostrou-se efetiva em termos de execução de exercícios e a apresentar um rápido feedback para o aluno nos testes apresentados, entretanto alguns pontos podem ser trabalhados para melhorar a experiência do usuário na utilização da plataforma, aumentar o desempenho da mesma, entre outros.

5.1 Trabalhos futuros

5.1.1 Visualização de matrizes

Na disciplina de Cálculo Numérico, problemas envolvendo vetores e matrizes são recorrentes. A apresentação desses dados não é muito amigável pela plataforma. Estes dados são apresentados como uma única linha de texto, e dificulta o entendimento do dado, por exemplo, uma matriz em Octave pode ser definida na forma $x = [1, 2; 3, 4]$. Poderia ser desenvolvido uma melhor representação para estes tipos de dados através do *Front End*, onde os dados em forma de texto seriam convertidos para a forma vetorial ou matricial.

5.1.2 Suporte a linguagens

A problemática central enfrentada pode ser encontrada em outras disciplinas que envolvem o desenvolvimento de atividades de programação pelo aluno. A plataforma poderia permitir a execução de códigos desenvolvidos em outras linguagens. A plataforma poderia ter uma lista de linguagens, e através do módulo de execução, seria decidido qual contêiner seria executado para a execução do código. Linguagens como C necessitam um contêiner que seja capaz de realizar a compilação do código e então a execução.

5.1.3 Eliminação de palavras chaves

O Octave permite uma série de funções predefinidas. Em algumas situações, o professor pode querer passar um exercício que envolve a implementação pelo aluno de um método que já existe na linguagem, por exemplo, em Cálculo Numérico o Octave tem funções prontas para realizar a multiplicação de matrizes, ou encontrar o zero da função. Estas funções poderiam ser bloqueadas pelo professor para determinado exercício, para impedir que o aluno faça uso e não precise realizar a implementação do método.

O bloqueio também poderia ser realizado em APIs para acesso a recursos do sistema, por exemplo, bloquear a função `system()`, que permite a execução de códigos no terminal do sistema em que o Octave esta executando.

5.1.4 Escalonamento automático do módulo corretor

Como foi apresentado nos testes, o tamanho da fila de execução pode crescer de acordo com a frequência em que os alunos submetem seus exercícios. Executar diversos módulos de execução simultaneamente pode consumir muitos recursos da máquina, e em diversos casos,

a maioria dos módulos ficaram inativos, aguardando tarefas. Uma boa ideia seria criar e excluir instâncias do módulo de execução sob demanda, utilizando por exemplo um serviço em nuvem.

Existem diversas estratégias para escalar um serviço, uma forma simples de fazer isso seria determinado um valor x de tempo de espera máximo, e ao ultrapassar esse valor, uma nova instância do módulo de execução seria alocado. Por exemplo, se o tempo de espera na fila for superior a 20 segundos, uma nova instância deve ser alocada. Esta instância poderia ser removida após um limite inferior, por exemplo, de 10 segundos.

5.1.5 Exportação de dados

A plataforma consegue gerenciar as notas dos alunos para atividades individuais, entretanto, não é possível visualizar todas as notas de um aluno em uma matéria, nem visualizar todas as notas dos alunos, ou ainda exportar estes dados para fora da plataforma. Seria útil visualizar todas as notas dos alunos de uma matéria, e ainda exportar todas as notas de uma turma para uma planilha.

Vale notar que embora os testes apresentados se tratarem de simulações em turmas, eles serviram para testar a estabilidade da plataforma, coletar sugestões de trabalhos futuros e testar o fluxo completo de uma submissão, e a plataforma se mostrou capaz de lidar com a turma. Entretanto, a plataforma deve ser testada em uma turma real para obter mais *feedbacks* e obter dados em cenários reais.

REFERÊNCIAS

AHONIEMI, T.; REINIKAINEN, T. Aloha - a grading tool for semi-automatic assessment of mass programming courses. In: **Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006**. New York, NY, USA: Association for Computing Machinery, 2006. (Baltic Sea '06), p. 139–140. ISBN 9781450378383. Disponível em: https://doi.org/10.1145/1315803.1315830.

AMARAL, T. R. do; LEITE, N. G.; SILVA, A. O. da. O ensino de cálculo numérico utilizando o scilab. In: **VI Congresso Internacional de Ensino de Matemática-2013**. [*S.l.*: *s.n.*], 2013.

AMAZON. **Amazon MQ**. 2017.

ANIL, T. J. N. **Introdução aos contêineres e ao Docker**. 2022. Microsoft. Disponível em: https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/container-docker-introduction/.

APACHE. **Apache Kafka**. 2011.

ARENALES, S. **Cálculo numérico : aprendizagem com apoio de software.** Cengage Learning, 2016. ISBN 9788522112876. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000010630&lang=pt-br&site=eds-live&scope=site.

BHUYAN, M. H.; KHAN, S. S. A. Teaching numerical analysis course for electrical engineering students using matlab. **Southeast University Journal of Science and Engineering**, v. 10, n. 2, p. 38–46, 2016.

BRITO, T. **01 — Mensageria**. 2019. Disponível em: https://medium.com/@devbrito91/mensageria-1330c6032049.

CHEN, J. **Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed**. 2020. Disponível em: https://unit42.paloaltonetworks.com/attackers-tactics-and-techniques-in-unsecured-docker-daemons-revealed.

DOCKER. **Docker Engine API**. 2013. Docker Docs. Disponível em: https://docs.docker.com/engine/api/.

EATON, J. W. **GNU Octave**. 1988.

FAKERJS. **Generate massive amounts of fake (but realistic) data for testing and development.** 2022. Disponível em: https://github.com/faker-js/faker.

FAYOCK, C. **What Are Environment Variables and How Can I Use Them with Gatsby and Netlify?** 2020. Freecodecamp. Disponível em: https://www.freecodecamp.org/news/what-are-environment-variables-and-how-can-i-use-them-with-gatsby-and-netlify/#what-are-environment-variables.

FILHO, F. F. C. **Algoritmos numéricos : uma abordagem moderna de cálculo numérico.** LTC, 2018. 7 p. ISBN 9788521635550. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000012409&lang=pt-br&site=eds-live&scope=site.

FREIRE, P. **Pedagogia da autonomia : saberes necessários à prática educativa.** [*S.l.*]: Paz e Terra, 2016. ISBN 9788577531639.

GAMA, C. L. G.; GOMES, M. das N.; PIRES, L. A. Da teoria à prática: problematização e metodologias diferenciadas no cálculo numérico. **Ensino em Re-Vista**, v. 25, n. 1, p. 234–255, 2018.

GERDES, A.; JEURING, J. T.; HEEREN, B. J. Using strategies for assessment of programming exercises. In: **Proceedings of the 41st ACM Technical Symposium on Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2010. (SIGCSE '10), p. 441–445. ISBN 9781450300063. Disponível em: https://doi.org/10.1145/1734263.1734412.

GIBBS, G. **Assessment Matters in Higher Education**: Choosing and using diverse approaches. Philadelphia, USA: SRHE and Open University Press, 1999. 465 p.

GOOGLE. **Material Design**. 2014. Material Design. Disponível em: https://material.io/.

GRUNERT, S. **Demystifying Containers - Part I: Kernel Space**. 2019. Medium. Disponível em: https://medium.com/@saschagrunert/demystifying-containers-part-i-kernel-space-2c53d6979504.

GUPTA, S.; DUBEY, S. K. *et al.* Automatic assessment of programming assignment. **Computer Science & Engineering**, Academy & Industry Research Collaboration Center (AIRCC), v. 2, n. 1, p. 67, 2012.

IHANTOLA, P. *et al.* Review of recent systems for automatic assessment of programming assignments. In: **Proceedings of the 10th Koli Calling International Conference on Computing Education Research**. New York, NY, USA: Association for Computing Machinery, 2010. (Koli Calling '10), p. 86–93. ISBN 9781450305204. Disponível em: https://doi.org/10.1145/1930464.1930480.

INSA, D.; SILVA, J. Semi-automatic assessment of unrestrained java code: A library, a dsl, and a workbench to assess exams and exercises. In: **Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2015. (ITiCSE '15), p. 39–44. ISBN 9781450334402. Disponível em: https://doi.org/10.1145/2729094.2742615.

JWT. **Introduction to JSON Web Tokens**. 2010. Disponível em: https://jwt.io/introduction.

KROTOFF, T. **Front-end frameworks popularity**. 2020. GitHub. Disponível em: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190.

KUMAR, S. **How to store a password in database?** 2018. Disponível em: https://www.geeksforgeeks.org/store-password-database/.

LOSHIN, A. S. G. P. **command-line interface (CLI)**. 2021. Techtarget. Disponível em: https://www.techtarget.com/searchwindowsserver/definition/command-line-interface-CLI.

MALIK, T. S. **How Do You Pipe the Output of a Command to a File in Linux**. 2021. Linux Hint. Disponível em: https://linuxhint.com/how-do-you-pipe-the-output-of-a-command-to-a-file-in-linux/.

MATHWORKS. **MATLAB**. 1984.

MERRIMAN, E. H. e. K. R. D. **Mongo DB**. 2007. Disponível em: https://www.mongodb.com/.

MILLER, M. **GNU Octave Docker images**. 2020. Docker Hub. Disponível em: https://vuex.vuejs.org/.

MONGODB. **Documents - MongoDB Manual**. 2007. Disponível em: https://www.mongodb.com/.

MONGODB. **mongo**. 2016. Docker Hub. Disponível em: https://hub.docker.com/_/mongo.

MORINIGO, L. **Web Architecture: Server-Side Render or Client Side Render?** 2020. Medium. Disponível em: https://medium.com/samsung-internet-dev/web-architecture-server-side-render-or-client-side-render-f16b61cd6299.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: **2017 IEEE International Systems Engineering Symposium (ISSE)**. [*S.l.*: *s.n.*], 2017. p. 1–7.

NAZÁRIO, D. C.; SOUZA, A. de. Boca-lab: Corretor automático de código adaptado ao ensino de linguagem de programação. **Anais do Computer on the Beach**, p. 42–46, 2010.

OCTAVE. **Command Line Options**. 2022. Disponível em: https://octave.org/doc/v7.1.0/Command-Line-Options.html.

OLIVEIRA, B. **Containers**. 2020. Disponível em: https://medium.com/sysadminas/containers-e4adf391de87.

OREIZY, P.; TAYLOR, R. N. **Decentralized software evolution**. [*S.l.*], 2003.

PASSAGLIA, A. **Vue.js 2 Cookbook.** Packt Publishing, 2017. ISBN 9781786468093. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1513366&lang=pt-br&site=eds-live&scope=site.

PIRES, A. d. A. **Cálculo numérico : prática com algoritmos e planilhas.** Atlas, 2015. ISBN 9788522498819. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000006759&lang=pt-br&site=eds-live&scope=site.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. [*S.l.*]: McGraw Hill Brasil, 2021.

RABBITMQ. **RabbitMQ Introduction**. 2007. Disponível em: https://www.rabbitmq.com/tutorials/tutorial-one-python.html.

RAJAN, S. Teaching object-oriented numerical analysis. **Journal of Engineering Education Transformations**, 2016.

REDHAT. **O que é um container Linux?** 2018. RedHat. Disponível em: https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container.

ROMLI, R.; SULAIMAN, S.; ZAMLI, K. Z. Automatic programming assessment and test data generation a review on its approaches. In: **2010 International Symposium on Information Technology**. [*S.l.*: *s.n.*], 2010. v. 3, p. 1186–1192.

SCHAUDER, J. **What does the concept of "worker"mean in programming?** 2007. Disponível em: https://stackoverflow.com/questions/10300251/what-does-the-concept-of-worker-mean-in-programming.

SHARMA, N.; GOBBERT, M. K. A comparative evaluation of matlab, octave, freemat, and scilab for research and teaching. **UMBC Faculty Collection**, 2010.

SHINDE, A. K. **Understanding Nginx As A Reverse Proxy**. 2021. Disponível em: https://medium.com/globant/understanding-nginx-as-a-reverse-proxy-564f76e856b2.

SIAM. **The history of numerical analysis and scientific computing**. 2006. Disponível em: http://history.siam.org/.

SILVA, W. *et al.* Octave correction tool: uma ferramenta de correção de rotinas computacionais para a disciplina de cálculo numérico. **Revista Eletrônica de Iniciação Científica em Computação**, v. 18, n. 2, 2020.

SMITH, R. **Docker Orchestration.** Packt Publishing, 2017. ISBN 9781787122123. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1457766&lang=pt-br&site=eds-live&scope=site.

SOMMERVILLE, I. **Software Engineering**. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1.

SPERANDIO, D.; MENDES, J. T.; SILVA, L. H. M. e. **Cálculo numérico: características matemáticas e computacionais dos métodos numéricos**. [*S.l.*]: Prentice Hall, 2003.

VASCONCELLOS, C. H.; BARROSO, L. C. Cálculo numérico e desenvolvimento tecnológico. **Educação & Tecnologia**, v. 1, n. 1, 1994.

VMWARE. **Rabbit MQ**. 2007.

VUETIFY. **Why you should be using Vuetify**. 2016. Vuetify. Disponível em: https://vuetifyjs.com/.

VULTR. **High Performance Cloud Servers.** 2014. Disponível em: https://www.vultr.com/.