UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ
CENTRO DE CIÊNCIAS, TECNOLOGIA E SAÚDE
CURSO TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÃO

Fabio Junior Barros Barbosa

AMPLIAÇÃO E MODERNIZAÇÃO DO PROJETO MEU-TCC:
UMA ABORDAGEM PARA APRIMORAR A EFICIÊNCIA E A ESCALABILIDADE DO BACK-END

Araranguá
2023

Fabio Junior Barros Barbosa

AMPLIAÇÃO E MODERNIZAÇÃO DO PROJETO MEU-TCC:

UMA ABORDAGEM PARA MELHORAR A EFICIÊNCIA E A USABILIDADE DO BACK-END

Trabalho de Conclusão do Curso de Graduação em Tecnologias da informação e comunicação do Centro de Araranguá, da Universidade Federal de Santa Catarina como requisito para a obtenção do Título de Bacharel/Licenciado em Tecnologias da informação e comunicação.
Orientador: Profª Christian Cechinel.

Araranguá

2023

Barbosa, Fabio Junior Barros
AMPLIAÇÃO E MODERNIZAÇÃO DO PROJETO MEU-TCC : UMA ABORDAGEM PARA MELHORAR A EFICIÊNCIA E A USABILIDADE DO BACK-END / Fabio Junior Barros Barbosa ; orientador, Cristian Cechinel, 2023.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Tecnologias da Informação e Comunicação, Araranguá, 2023.

Inclui referências.

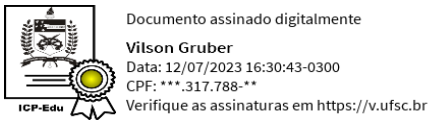
1. Tecnologias da Informação e Comunicação. 2. Gestão de projetos. 3. TCC. 4. Ampliação de back-end. 5. Trabalhos acadêmicos. I. Cechinel, Cristian. II. Universidade Federal de Santa Catarina. Graduação em Tecnologias da Informação e Comunicação. III. Título.

Fabio Junior Barros Barbosa

AMPLIAÇÃO E MODERNIZAÇÃO DO PROJETO MEU-TCC:
UMA ABORDAGEM PARA MELHORAR A EFICIÊNCIA E A USABILIDADE DO BACK-END

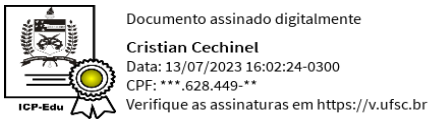
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel e aprovado em sua forma final pelo Curso Tecnologias da Informação e Comunicação.

Araranguá, 05 de Julho de 2023.

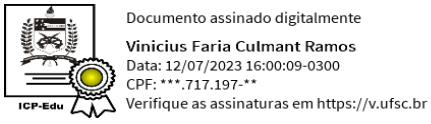


Prof. Vilson Gruber, Dr.
Coordenador do Curso

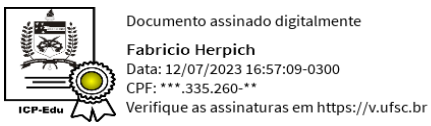
Banca Examinadora



Prof.ª Cristian Cechinel, Dr.
Orientador
Universidade Federal de Santa Catarina



Prof.ª Vinicius Faria Culmant Ramos, Dr.
Universidade Federal de Santa Catarina



Prof. Fabricio Herpich, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado a minha querida família e amigos
que me apoiaram desde o início da jornada acadêmica.

AGRADECIMENTOS

Gostaria de aproveitar este momento para expressar minha gratidão a todas as pessoas que contribuíram para o sucesso da minha jornada na elaboração deste trabalho. Sem o apoio e incentivo delas, esse projeto não seria possível.

Primeiramente, gostaria de agradecer à minha família pelo constante apoio e compreensão ao longo de toda a minha trajetória acadêmica. Agradeço por estarem sempre ao meu lado, me encorajando e me motivando a buscar sempre o melhor em todas as etapas desta caminhada. Vocês são minha base e minha fonte de inspiração.

À minha rede de amigos, meu mais sincero agradecimento. A cada um de vocês que esteve presente ao longo desses anos, compartilhando risadas, desafios e conquistas, minha gratidão é imensa. Seus estímulos e encorajamentos foram fundamentais para me manter motivado durante as dificuldades e para celebrar as vitórias alcançadas.

Gostaria de expressar minha profunda gratidão ao meu orientador, Cristian Cechinel, pela orientação e apoio ao longo deste projeto. Sua dedicação, paciência e comprometimento foram essenciais para o desenvolvimento desta pesquisa. Sou imensamente grato pela sua orientação experiente, pelas discussões enriquecedoras e pelos insights valiosos que contribuíram para a evolução deste trabalho.

Não posso deixar de mencionar também o Ezequiel Matos, que esteve presente durante todo o projeto, oferecendo sua expertise e auxiliando nas etapas de desenvolvimento. Sua contribuição foi fundamental para o sucesso desta pesquisa, e sou grato pela sua disponibilidade em compartilhar conhecimentos e ajudar a superar os desafios encontrados.

Também gostaria de agradecer ao professor Alexandre Leopoldo Gonçalves, que me auxiliou de forma significativa na estruturação do banco de dados. Sua orientação técnica e conhecimento profundo na área foram fundamentais para a construção de uma base sólida e eficiente para este projeto.

A todos vocês, meu sincero agradecimento. Seu apoio foi fundamental e fez toda a diferença. Esse trabalho não seria possível sem vocês.

Muito obrigado!

O modo como você reúne, administra e usa a informação determina se vencerá ou perderá. (Gates, 1999)

RESUMO

O projeto de ampliação do back-end do sistema MeuTCC visa aprimorar a eficiência e a usabilidade dessa plataforma, desenvolvida para facilitar a criação e gerenciamento de trabalhos acadêmicos. O objetivo principal é expandir as capacidades do software por meio da implementação de novas tecnologias, como Docker e Swagger. O sistema MeuTCC permite o cadastro de usuários, possibilitando que eles tenham um ou mais perfis. Os usuários com perfil de aluno têm a capacidade de criar projetos e envolver outros usuários, como orientadores, coorientadores e membros da banca, em suas atividades acadêmicas. O projeto visa simplificar o processo de criação de trabalhos acadêmicos, promovendo a colaboração e facilitando a gestão dos envolvidos. A ampliação do back-end do MeuTCC envolve a implementação de tecnologias como Docker, que oferece a possibilidade de empacotar o sistema em contêineres independentes, facilitando sua implantação e escalabilidade. Além disso, a integração do Swagger permite uma documentação completa e interativa da API, facilitando o desenvolvimento de novos recursos e a interação com outras aplicações.

Palavras-chave: ampliação do back-end; trabalhos acadêmicos; colaboração; Docker; Swagger.

ABSTRACT

The project to expand the MeuTCC system's back-end aims to improve the efficiency and usability of this platform, developed to facilitate the creation and management of academic papers. The main objective is to expand the capabilities of the software through the implementation of new technologies such as Docker and Swagger. The MeuTCC system allows the registration of users, allowing them to have one or more profiles. Users with a student profile have the ability to create projects and involve other users, such as advisors, co-advisors and committee members, in their academic activities. The project aims to simplify the process of creating academic papers, promoting collaboration and facilitating the management of those involved. The expansion of MeuTCC's back-end involves the implementation of technologies such as Docker, which offers the possibility of packaging the system in independent containers, facilitating its deployment and scalability. In addition, the Swagger integration allows a complete and interactive documentation of the API, facilitating the development of new features and the interaction with other applications.

Keywords: back-end augmentation; academic work; collaboration; Docker; Swagger.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID - Atomicidade, Consistência, Isolamento e Durabilidade

API - Application Programming Interface

CRUD - Create, Read, Update e Delete

HTML - HyperText Markup Language

ID - Identity

JSON - JavaScript Object Notation

JWT - JSON Web Token

ORM - Object-Relational Mapping

PHP - Hypertext Preprocessor

SBC - Sociedade Brasileira de Computação

SMTP - Simple Mail Transfer Protocol

SQL - Structured Query Language

TCC - Trabalho de conclusão de curso

URL - Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

No ambiente acadêmico, a elaboração de trabalhos científicos é uma atividade essencial para o desenvolvimento de pesquisas e a disseminação do conhecimento. Inicialmente, um grupo de alunos empreendeu esforços para criar um software de gerenciamento de projetos acadêmicos, reconhecendo a complexidade e a natureza colaborativa dessas iniciativas. Contudo, ao se depararem com desafios e a necessidade de ampliar a funcionalidade e a escalabilidade do sistema, foi tomada a decisão de reformular o projeto original.

Assim, em conjunto com o orientador e os alunos que iniciaram o trabalho, nos reunimos para repensar a arquitetura, ferramentas e tecnologias utilizadas. Esta pesquisa apresenta o desenvolvimento subsequente do software MeuTCC, resultado dessa iniciativa de aprimoramento e expansão. A partir dessa reestruturação, o projeto ganhou um novo direcionamento, visando facilitar ainda mais o gerenciamento de projetos acadêmicos, promover a colaboração entre os envolvidos e melhorar a produção de trabalhos científicos.

Este projeto tem como objetivo apresentar o desenvolvimento dessa nova versão do software, que permite o cadastro de usuários com diferentes perfis e desempenha um papel central no processo de criação dos projetos acadêmicos. Serão abordados os principais requisitos do sistema, as tecnologias adotadas e os desafios encontrados durante a implementação. Além disso, serão discutidos os resultados alcançados até o momento e as perspectivas para futuras melhorias e expansões.

Em suma, este projeto é uma continuação do trabalho inicial desenvolvido por um grupo de alunos, que identificaram a necessidade de aprimorar a arquitetura e as funcionalidades do software MeuTCC. Através da reestruturação e da adoção de novas tecnologias, busca-se proporcionar uma solução ainda mais eficiente e colaborativa para o gerenciamento de projetos acadêmicos, contribuindo para a produção de trabalhos científicos de qualidade.

1.1 OBJETIVOS

1.2 OBJETIVO GERAL

O objetivo geral deste projeto é realizar a ampliação do back-end do sistema MeuTCC, visando aprimorar a eficiência e a usabilidade da plataforma para facilitar a criação e gestão de trabalhos acadêmicos colaborativos.

1.2.1 Objetivos Específicos

a) Analisar o funcionamento atual do back-end do sistema MeuTCC, identificando os pontos de melhoria e as funcionalidades que podem ser adicionadas para aprimorar a experiência do usuário.
b) Analisar as tecnologias e ferramentas existentes no contexto de desenvolvimento de aplicações web para identificar as melhores práticas e soluções adotadas no mercado.
c) Pesquisar e avaliar a viabilidade técnica e os benefícios da integração das tecnologias Docker e Swagger no back-end do MeuTCC, visando melhorar a escalabilidade, a implantação simplificada e a documentação da API.
d) Elaborar uma proposta detalhada de implementação para a ampliação do back-end do MeuTCC, incluindo a definição das novas funcionalidades, arquitetura de software e métodos de integração das tecnologias selecionadas.

1.3 PROBLEMÁTICA/JUSTIFICATIVA

A problemática identificada no projeto original reside na complexidade e limitações das ferramentas e tecnologias utilizadas, como PHP, Vue.js e banco de dados SQLite. Embora essas tecnologias tenham permitido o desenvolvimento de um sistema funcional, a escalabilidade, a flexibilidade e a eficiência foram comprometidas, dificultando a manutenção e o crescimento do projeto.

Diante disso, a justificativa para a ampliação do back-end do sistema MeuTCC com o uso de tecnologias como Node.js, Express, TypeORM, Docker, Docker Compose e PostgreSQL é promover uma significativa melhoria na arquitetura e na infraestrutura da aplicação.

A escolha de Node.js como linguagem de programação para o back-end traz inúmeras vantagens, como a possibilidade de utilizar JavaScript tanto no front-end quanto no back-end, facilitando a colaboração entre desenvolvedores e proporcionando uma curva de aprendizado suave para a equipe. Além disso, a eficiência e o desempenho do Node.js permitem uma resposta mais rápida aos clientes, melhorando a experiência do usuário.

A adoção do framework Express proporciona uma estrutura robusta e flexível para o desenvolvimento de APIs, permitindo a construção de rotas e a implementação de middlewares de forma simplificada e eficiente. Combinado com o TypeORM, um ORM (Object-Relational

Mapping) baseado em TypeScript, torna-se mais fácil mapear objetos para tabelas no banco de dados PostgreSQL, simplificando as operações de persistência e manipulação dos dados.

A utilização de Docker e Docker Compose oferece a possibilidade de empacotar e implantar o sistema em contêineres isolados, garantindo a portabilidade, a consistência e a escalabilidade do ambiente de desenvolvimento. Essas tecnologias também facilitam a configuração e o gerenciamento de dependências, tornando a implantação e a execução do sistema mais ágeis e confiáveis.

A escolha do banco de dados PostgreSQL traz benefícios como maior desempenho, escalabilidade e suporte a recursos avançados, como transações de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) e integridade referencial.

Assim, a ampliação do back-end do sistema MeuTCC com essas tecnologias modernas e poderosas visa solucionar as limitações identificadas no projeto original, proporcionando um sistema mais robusto, escalável, flexível e eficiente. Essas melhorias trarão benefícios significativos tanto para os usuários finais, que terão uma experiência de uso aprimorada, quanto para a equipe de desenvolvimento, que poderá trabalhar de forma mais produtiva e eficiente.

1.4 ORGANIZAÇÃO DO DOCUMENTO

A organização do documento é fundamental para guiar o leitor ao longo da sua jornada no trabalho. Nesse contexto, o presente projeto foi estruturado de forma a apresentar uma sequência lógica de conteúdo.

O documento inicia com a seção de Introdução, onde são expostos os objetivos gerais e específicos do trabalho, bem como a problemática e justificativa que motivaram a sua realização.

Em seguida, a seção de Fundamentação Teórica aborda o uso de software para gestão de projetos e o papel transformador dessas ferramentas no contexto acadêmico.

O Capítulo 3 apresenta a proposta do software para gestão de TCCs, incluindo a metodologia adotada, a descrição do projeto, o público-alvo e o planejamento.

No Capítulo 4, são abordadas as ferramentas e conceitos teóricos relevantes, como banco de dados, linguagem de programação e gerenciamento de ambientes de desenvolvimento.

O Capítulo 5, aborda o desenvolvimento do back-end com detalhes seguido da documentação da API.

O Capítulo 6, apresenta os resultados obtidos, onde são destacados aspectos como a implementação de tecnologias avançadas, ferramentas de padronização de código, histórico de

banco de dados, segurança das rotas e acesso controlado, bem como a implementação do CRUD e operações no banco de dados.

O Capítulo 7, apresenta as considerações com ênfase na experiência e aprendizados adquiridos, avaliação do resultado final e recomendações para melhorias.

O Capítulo 8 apresenta a conclusão do trabalho, consolidando os principais resultados alcançados e reforçando a importância do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será apresentada uma fundamentação teórica abrangente que embasará o desenvolvimento do projeto de ampliação do back-end do sistema MeuTCC. Serão discutidos conceitos, teorias e práticas relevantes relacionadas ao uso de ferramentas de gestão no contexto acadêmico, além de tecnologias e softwares utilizados para aprimorar a gestão de projetos.

2.1 USO DE SOFTWARE PARA GESTÃO DE PROJETOS

O gerenciamento de projetos é orientado por pessoas e precisa ser feito de forma técnica. Envolve a compreensão das causas e efeitos das relações e interações entre as dimensões sociotécnicas dos projetos. Aprimorar competências nessas dimensões melhorará enormemente a sua vantagem competitiva como gerente de projetos. A área de gerenciamento de projetos tem crescido rapidamente em importância. É quase impossível imaginar qualquer carreira de gerente no futuro que não inclua o gerenciamento de projetos. (GRAY; LARSON, 2009).

É indiscutível a importância da informação no processo de uma gestão ágil, eficaz e cristalina, portanto, o sistema de gestão acadêmica exerce um papel importante no processo de gestão acadêmica. A transparência das informações proporcionadas por estes sistemas viabiliza o controle e acompanhamento mais eficiente do processo administrativo, e, por conseguinte, possibilitam planejar as ações de forma simples, e competente, otimizando o uso dos recursos e diminuindo os custos (SARAIVA, 2010).

Em suma, o uso de software para gestão de projetos se mostra indispensável no ambiente atual, onde a complexidade e a dinâmica das atividades projetuais exigem uma abordagem estruturada e eficiente. Através dessas ferramentas, os gerentes de projetos são capazes de planejar, controlar e acompanhar cada etapa do ciclo de vida do projeto, garantindo o cumprimento de prazos, alocação adequada de recursos e a obtenção dos resultados desejados. Além disso, o emprego de sistemas de gestão acadêmica possibilita a transparência das informações, facilitando a tomada de decisões, a otimização de processos e o aprimoramento da gestão acadêmica como um todo. Com a utilização dessas soluções tecnológicas, é possível aumentar a eficiência, a produtividade e a qualidade dos projetos desenvolvidos, contribuindo para o sucesso das organizações e instituições de ensino.

2.2 O PAPEL TRANSFORMADOR DO SOFTWARE PARA GESTÃO DE PROJETOS ACADÊMICOS

A gestão de projetos acadêmicos demanda uma abordagem eficiente e estruturada, especialmente diante da complexidade e diversidade das atividades envolvidas. Nesse contexto, o uso de software para gestão de projetos se revela como uma ferramenta transformadora. Essas soluções tecnológicas oferecem recursos e funcionalidades que vão além da organização e planejamento básicos, permitindo um gerenciamento aprimorado das atividades acadêmicas.

Esses softwares proporcionam uma visão abrangente do projeto, possibilitando a definição de metas claras, a alocação eficiente de recursos, o monitoramento do progresso em tempo real e a identificação de eventuais desvios ou problemas. Com recursos como gráficos de Gantt, calendários compartilhados, controle de tarefas e colaboração online, os softwares de gestão de projetos acadêmicos viabilizam uma comunicação efetiva e uma colaboração mais fluida entre os membros da equipe acadêmica.

Além disso, essas ferramentas oferecem uma série de benefícios no contexto acadêmico, como o aumento da produtividade, a otimização do tempo, a padronização de processos e a melhoria da qualidade dos resultados obtidos. Ao centralizar informações, simplificar fluxos de trabalho e oferecer relatórios e análises detalhadas, os softwares para gestão de projetos acadêmicos contribuem para uma tomada de decisão embasada em dados e facilitam a identificação de oportunidades de melhoria.

De acordo com Johnson (2001) a área de software e tecnologia da informação (TI) este assunto assume a cada dia uma importância maior. Isto se deve, em parte, pelo entendimento de que parte significativa do insucesso em projetos de software está relacionada com uma má gerência de projetos ou, algumas vezes, por uma ausência completa de gerenciamento (apud Torreão, 2010).

Dessa forma, fica evidente que o uso de software para gestão de projetos acadêmicos é essencial para aprimorar a eficiência e a eficácia das atividades acadêmicas, potencializando o sucesso dos projetos desenvolvidos no meio acadêmico e promovendo um ambiente propício à inovação, colaboração e excelência acadêmica.

3 PROPOSTA DE UM SOFTWARE PARA GESTÃO DE TCCS

A proposta deste projeto é desenvolver um software inovador e eficiente para auxiliar os estudantes na gestão de seus Trabalhos de Conclusão de Curso (TCC). O objetivo principal é fornecer uma solução tecnológica que simplifique e otimize todo o processo envolvido na elaboração, acompanhamento e apresentação dos TCCs, desde a seleção do tema até a entrega final.

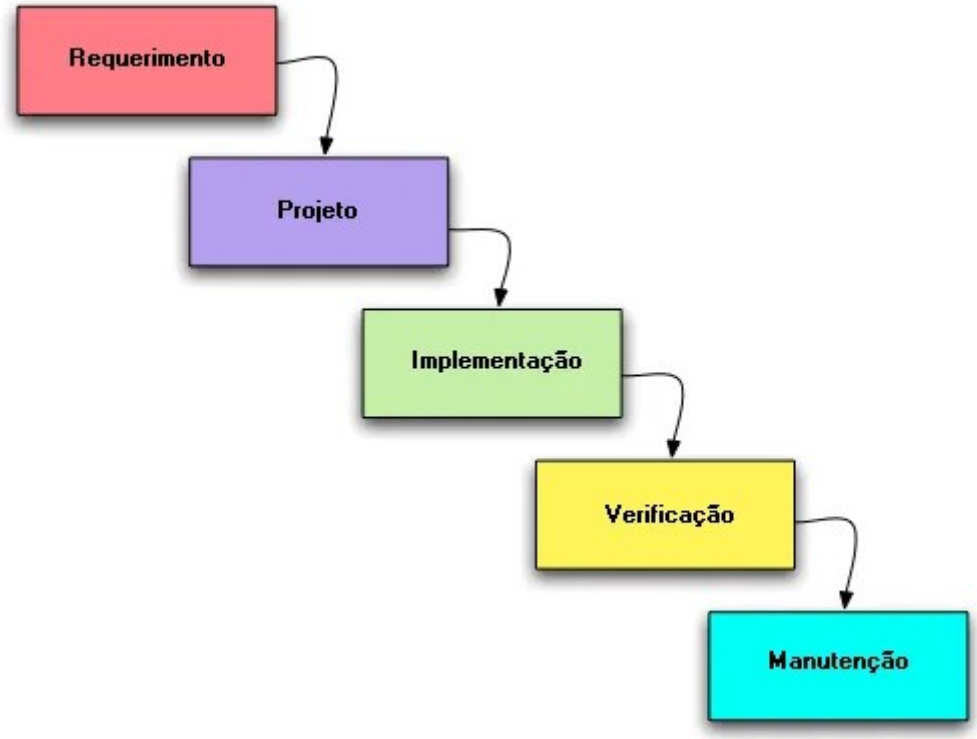
O software será desenvolvido com base nas necessidades e desafios enfrentados pelos estudantes durante essa etapa crucial de sua formação acadêmica. Além disso, o software oferecerá recursos avançados de organização, colaboração, controle de prazos, gerenciamento de referências bibliográficas e geração de relatórios, visando facilitar o trabalho dos estudantes, aumentar sua produtividade e garantir a qualidade e o cumprimento dos requisitos acadêmicos estabelecidos pela instituição de ensino.

3.1 METODOLOGIA

O projeto de metodologia de desenvolvimento de um sistema é visto geralmente como uma atividade intelectual complexa. Ele exige habilidades em, pelo menos, duas disciplinas: o domínio do problema ou da aplicação, que é a área do problema a ser resolvido, e o domínio da solução, que é a área de sistemas de informação e de software. É importante “compreender profundamente o universo da aplicação e do público-alvo para quem será desenvolvido o sistema: seus valores, sua cultura, suas necessidades, entre outros” (BISSARRO, 2002, apud Freitas et al. 2006).

Para o desenvolvimento do software MeuTCC, está sendo adotada uma abordagem metodológica que visa garantir a eficiência e a qualidade do processo de pesquisa e desenvolvimento. A metodologia compreende a análise e avaliação dos métodos disponíveis para a realização de um projeto acadêmico como mostra a imagem 1, buscando alcançar os objetivos propostos. Nesse sentido, o projeto está seguindo o método científico, que consiste em um conjunto de procedimentos adotados com o propósito de buscar conhecimento e soluções para os problemas apresentados.

Figura 1 - Etapas do desenvolvimento do projeto

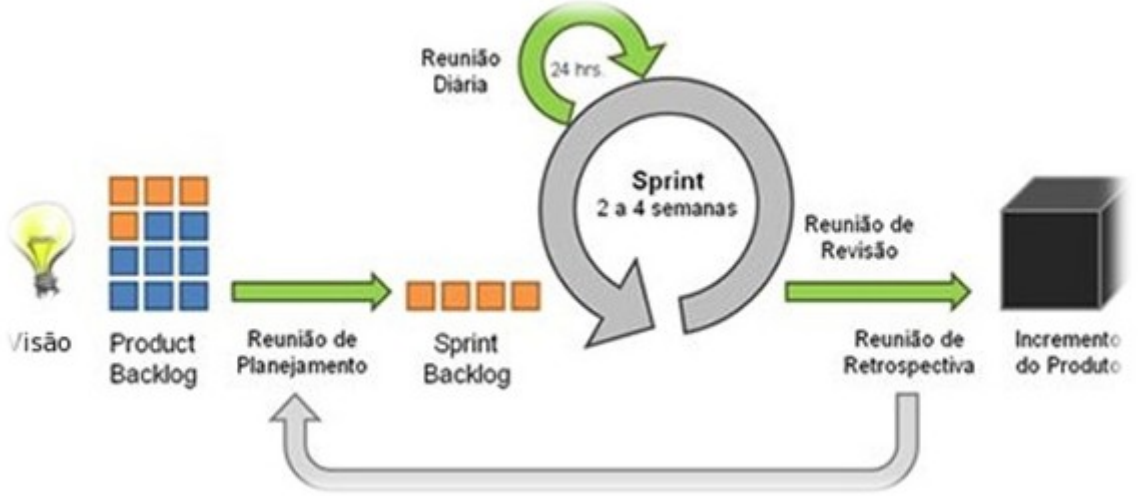


Fonte: https://www.diegomacedo.com.br/modelos-de-ciclo-de-vida/

A pesquisa científica, realizada com uma abordagem hipotético-dedutiva, teve início com a formulação clara e precisa do problema a ser solucionado pelo MeuTCC. Por meio de reuniões e análises, foram identificadas as prioridades e necessidades da ferramenta, bem como as etapas e recursos a serem desenvolvidos no back-end. Durante o processo, a equipe se dedicará à criação das funcionalidades essenciais do software, seguindo os princípios do método científico.

Para garantir um controle efetivo do tempo de criação dos recursos, adotamos métodos ágeis que proporcionam uma abordagem flexível e adaptável. Utilizamos práticas como o Scrum, que possibilita a visualização e o gerenciamento das tarefas em um fluxo contínuo. Por meio dessa metodologia, conseguimos acompanhar o progresso das atividades, identificar possíveis gargalos e ajustar o planejamento conforme necessário como demonstra a imagem 2.

Figura 2 - Metodologias ágeis utilizada no projeto



Fonte: https://www.devmedia.com.br/modelos-de-desenvolvimento-agil/27660

Embora a implementação da interface, os testes unitários e o deploy em nuvem não tenham sido abordados neste momento, serão considerados em sugestões futuras para aprimorar a solução desenvolvida. Os envolvidos ao projeto MeuTCC está consciente da importância desses aspectos e planeja incluí-los em etapas subsequentes, buscando aperfeiçoar a experiência do usuário, garantir a qualidade do software e otimizar sua disponibilidade.

Portanto, a metodologia adotada no desenvolvimento do MeuTCC visa assegurar a eficácia e a eficiência na gestão de TCC, concentrando-se, neste momento, no desenvolvimento do back-end para atender às necessidades dos estudantes e contribuir para aprimorar a qualidade e o cumprimento dos requisitos acadêmicos.

3.2 DESCRIÇÃO DO PROJETO

O projeto MeuTCC consiste em um software desenvolvido para auxiliar na gestão de projetos acadêmicos, com foco especial em TCC. O objetivo principal é fornecer uma plataforma intuitiva e eficiente que permita aos estudantes e pesquisadores organizar, planejar e acompanhar todas as etapas do processo de elaboração do trabalho. O software oferece recursos como o registro de atividades, definição de prazos, atribuição de tarefas, controle de versões, compartilhamento de arquivos e comunicação entre os membros da equipe. Além disso, o MeuTCC visa facilitar a colaboração e a troca de conhecimentos entre os envolvidos no projeto, proporcionando uma experiência mais eficaz e produtiva. Com essa ferramenta, busca-se otimizar o tempo e recursos investidos no desenvolvimento do TCC, contribuindo para a qualidade e o sucesso dos trabalhos acadêmicos.

3.3 PUBLICO/PERFIL

O software MeuTCC é uma solução desenvolvida para atender às necessidades de gestão de projetos acadêmicos, com ênfase em TCC. Destina-se a estudantes, pesquisadores e profissionais envolvidos na elaboração de trabalhos acadêmicos, que buscam uma ferramenta eficiente para auxiliar na organização, rastreamento e gerenciamento de suas atividades e tarefas relacionadas aos projetos acadêmicos. O MeuTCC possui uma arquitetura sólida e escalável, com foco no back-end, utilizando tecnologias modernas e boas práticas de desenvolvimento para garantir a segurança, confiabilidade e desempenho das operações do sistema. Sua interface de programação (API) permite a integração com outras ferramentas e sistemas utilizados no contexto acadêmico, possibilitando uma maior automatização e otimização dos fluxos de trabalho. O público-alvo do MeuTCC compreende desde estudantes individuais até instituições de ensino que desejam adotar uma solução personalizada para a gestão de projetos acadêmicos em seus ambientes.

3.4 PLANEJAMENTO

Planejamento estratégico é o processo de avaliar “o que somos” e decidir e implementar “o que pretendemos ser e o que faremos para consegui-lo” (Gray, 2009). No tópico de planejamento do trabalho, foi realizada uma análise cuidadosa das etapas e atividades necessárias para o desenvolvimento do projeto, também incluiu a identificação de possíveis riscos e a definição de estratégias para mitigá-los. Dessa forma, o planejamento detalhado permitiu uma gestão adequada do projeto.

3.5 REQUISITOS

A definição dos requisitos do projeto MeuTCC foi realizada por meio de um processo de pesquisa e diálogo entre a equipe de desenvolvimento e o orientador do projeto. Utilizando uma abordagem de “perguntas e respostas”, foram levantadas informações detalhadas sobre as necessidades dos usuários e as funcionalidades essenciais do sistema. O orientador desempenhou um papel fundamental na orientação da equipe, fornecendo insights valiosos e direcionamento para garantir que os requisitos atendessem às expectativas e demandas do público-alvo. Essa abordagem colaborativa permitiu uma compreensão aprofundada das necessidades dos usuários e contribuiu para a definição precisa dos requisitos funcionais e não funcionais do projeto.

3.5.1 Requisitos funcionais

Os requisitos funcionais estão intimamente ligados às funcionalidades propostas pelo sistema, e que serão usadas na resolução do problema do contratante, e atenderá todas as suas necessidades funcionais. Resumidamente, são os requisitos que objetivamente cumprem as reais necessidades do usuário do sistema. Exemplo: Controle financeiro, controle de tráfego aéreo, controle de produção (ROCHA; MAGALHÃES, 2019). Com base nesse conceito, o planejamento do projeto incluiu a definição de alguns requisitos não funcionais, os quais desempenham um papel crucial na performance e usabilidade da ferramenta conforme mostra a tabela 1.

Tabela 1 - Requisitos Funcionais

Código	Descrição
RF001	Deve permitir que o usuário possa se cadastrar no sistema.
RF002	Deve permitir o login do usuário.
RF003	Deve permitir ao usuário realizar a solicitação de recuperação de senha.
RF004	Deve permitir ao usuário a criação de uma nova senha.
RF005	Deve permitir que o usuário altere seus dados.
RF006	Deve permitir que o usuário cadastre o seu projeto – TCC.
RF007	Deve permitir que o usuário cadastre a instituição.
RF008	Deve permitir que o usuário cadastre os cursos.
RF009	Deve permitir que o usuário cadastre as linhas de pesquisas.
RF010	Deve permitir que o usuário cadastre o calendário do semestre.
RF011	Deve permitir que o usuário cadastre as tarefas do projeto.

Fonte: Autor do trabalho (2023)

3.5.2 Requisitos não funcionais

Os requisitos não-funcionais geralmente estão ligados à qualidade do produto como, por exemplo, robustez, segurança ou integridade. Dividem-se, assim como a qualidade de produtos, em

dois sub-ramos, dependência positiva e negativa (ROCHA; MAGALHÃES, 2019). Com base nesse conceito, o planejamento do projeto incluiu a definição de alguns requisitos não funcionais, os quais desempenham um papel crucial na performance e usabilidade da ferramenta conforme mostra a tabela 2.

Tabela 2 - Requisitos não Funcionais

Código	Descrição
RNF001	O sistema deve ser capaz de lidar com um grande volume de dados e processar solicitações de forma eficiente, garantindo tempos de resposta rápidos e minimizando a latência.
RNF002	O sistema deve ser capaz de lidar com o aumento da carga de trabalho e do número de usuários sem comprometer sua performance, permitindo a escalabilidade horizontal ou vertical conforme necessário.
RNF003	O sistema deve implementar medidas de segurança robustas para proteger os dados.
RNF004	O sistema deve ser altamente confiável, garantindo que esteja sempre disponível e que as transações sejam processadas corretamente, mesmo em situações de falhas ou interrupções temporárias.
RNF005	O código deve ser modular, bem estruturado e de fácil manutenção, permitindo a realização de atualizações, correções e melhorias de forma eficiente e sem impactos indesejados em outras partes do sistema.
RNF006	O back-end deve ser capaz de se integrar com outros sistemas e APIs, permitindo a troca de dados e a comunicação eficiente com diferentes componentes do ecossistema tecnológico.

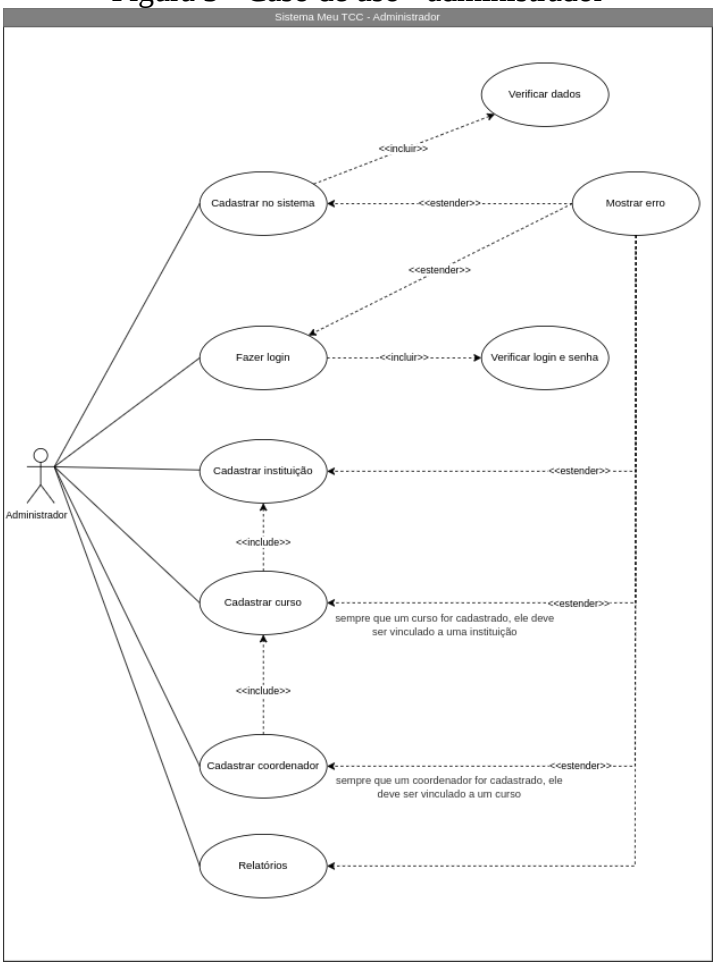
Fonte: Autor do trabalho (2023)

3.5.3 Caso de uso do administrador

O caso de uso do administrador do sistema envolve uma série de funcionalidades essenciais para a gestão do sistema (como demonstra a figura 3). O administrador tem a capacidade de cadastrar instituições, fornecendo informações relevantes sobre cada uma delas, tais como nome, endereço e contato. Além disso, ele também pode cadastrar cursos e vinculá-los às instituições correspondentes, garantindo a organização e a estruturação adequada dos dados. Para facilitar a

administração dos cursos, o administrador pode cadastrar coordenadores de curso e associá-los aos cursos já cadastrados. Essa funcionalidade permite uma gestão eficiente das responsabilidades e atribuições dentro do sistema. O administrador também tem a capacidade de gerar relatórios, fornecendo informações consolidadas e relevantes sobre as instituições, cursos, projetos e usuários

Figura 3 - Caso de uso - administrador



Fonte: Autor do trabalho (2023)

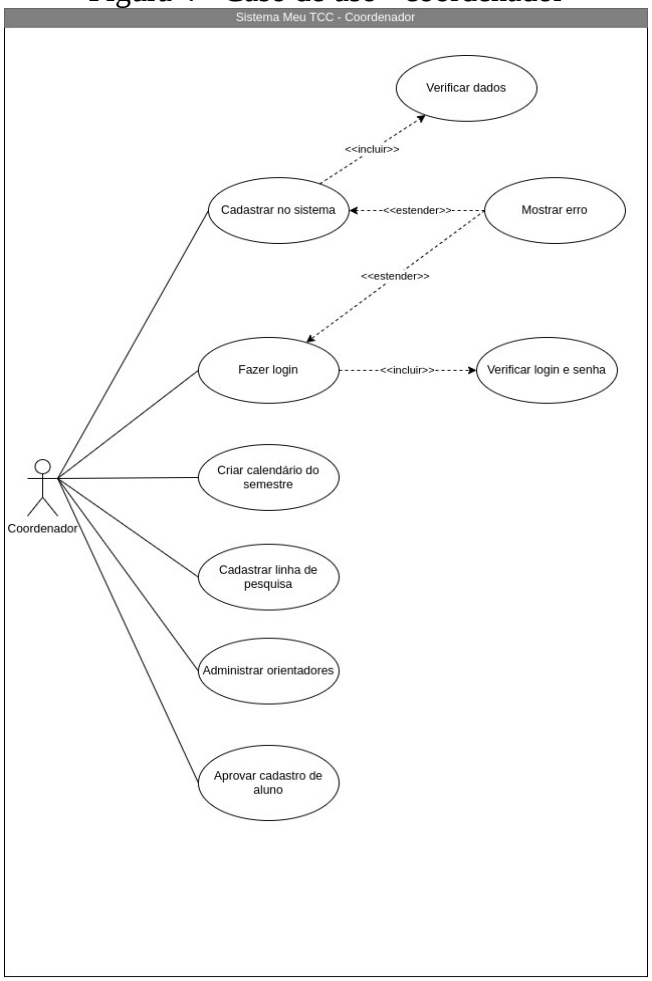
3.5.4 Caso de uso coordenador

O caso de uso do coordenador do curso abrange diversas responsabilidades fundamentais para o gerenciamento eficiente do processo de TCC como mostra a imagem 4. O coordenador desempenha um papel crucial na gestão dos orientadores/coorientadores e aprovação de cadastro dos novos alunos no sistema. Após o aluno realizar o cadastro, é enviado um e-mail de notificação ao coordenador, que deve revisar e aprovar o cadastro.

Outra função importante do coordenador é a criação do calendário do semestre, estabelecendo as etapas do TCC. Essas etapas incluem a definição do tema, da linha de pesquisa e

do orientador, a entrega da proposta de projeto, a definição da banca, o agendamento da defesa do TCC, o envio do TCC para a banca e a entrega da versão final do TCC.

Figura 4 - Caso de uso - coordenador



Fonte: Autor do trabalho (2023)

3.5.5 Caso de uso orientador/orientando

O caso de uso do orientador/orientando é fundamental para a interação e colaboração entre o aluno e seu orientador ao longo do projeto de TCC (conforme a figura 5). Nesse caso de uso, o aluno desempenha várias ações para gerenciar o projeto, enquanto o orientador desempenha um papel de aprovação e orientação.

Inicialmente, o aluno cadastra o seu projeto de TCC no sistema, incluindo informações como título, objetivo, metodologia, entre outros detalhes relevantes. Em seguida, o aluno vincula o orientador ao projeto, indicando o professor responsável por orientá-lo.

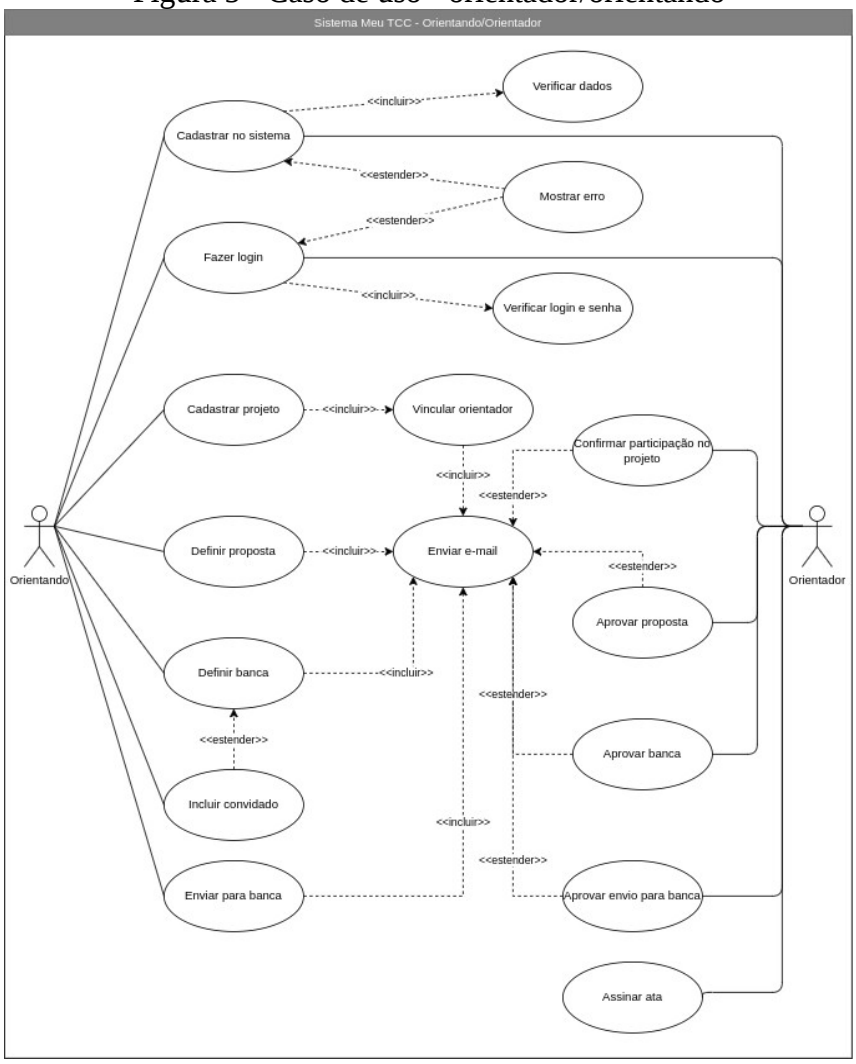
Após cadastrar o projeto, o aluno define uma proposta, que é enviada automaticamente por e-mail ao orientador para aprovação. O orientador tem a responsabilidade de revisar e aprovar a proposta, fornecendo feedback e orientações, se necessário.

Em seguida, o aluno avança para a etapa de definição da banca, selecionando os membros que irão compor o comitê avaliador do TCC. Novamente, um e-mail é enviado ao orientador para que ele aprove a composição da banca. Além disso, se o aluno desejar incluir um convidado especial na banca, é enviado um e-mail ao orientador solicitando sua aprovação.

Por fim, quando o aluno conclui o seu trabalho e está pronto para enviá-lo à banca, o projeto final é submetido pelo aluno através do sistema. Nesse momento, um e-mail é encaminhado ao orientador para que ele aprove o envio do projeto final.

Essas interações entre aluno e orientador garantem um acompanhamento adequado do projeto de TCC, permitindo que o orientador forneça orientações, sugestões e aprovações necessárias ao longo de todas as etapas do trabalho.

Figura 5 - Caso de uso - orientador/orientando



Fonte: Autor do trabalho (2023)

3.6 METODOLOGIA ÁGIL

O manifesto ágil ressalta o que mais tem valor para as metodologias ágeis, a importância de como saber lidar com pessoas, assim como ter o cliente colaborando para encontrar a melhor solução, entregar o software com qualidade e do que se adaptar às mudanças. Isto oculta parte das dificuldades dos processos, contratos, documentação e planejamento para o desenvolvimento de um software (FADEL; SILVEIRA, 2010).

Existem 12 princípios fundamentais e determinantes para obtenção de bons resultados quando se trata de desenvolvimento de software (FADEL; SILVEIRA, 2010, apud Filho, 2008).

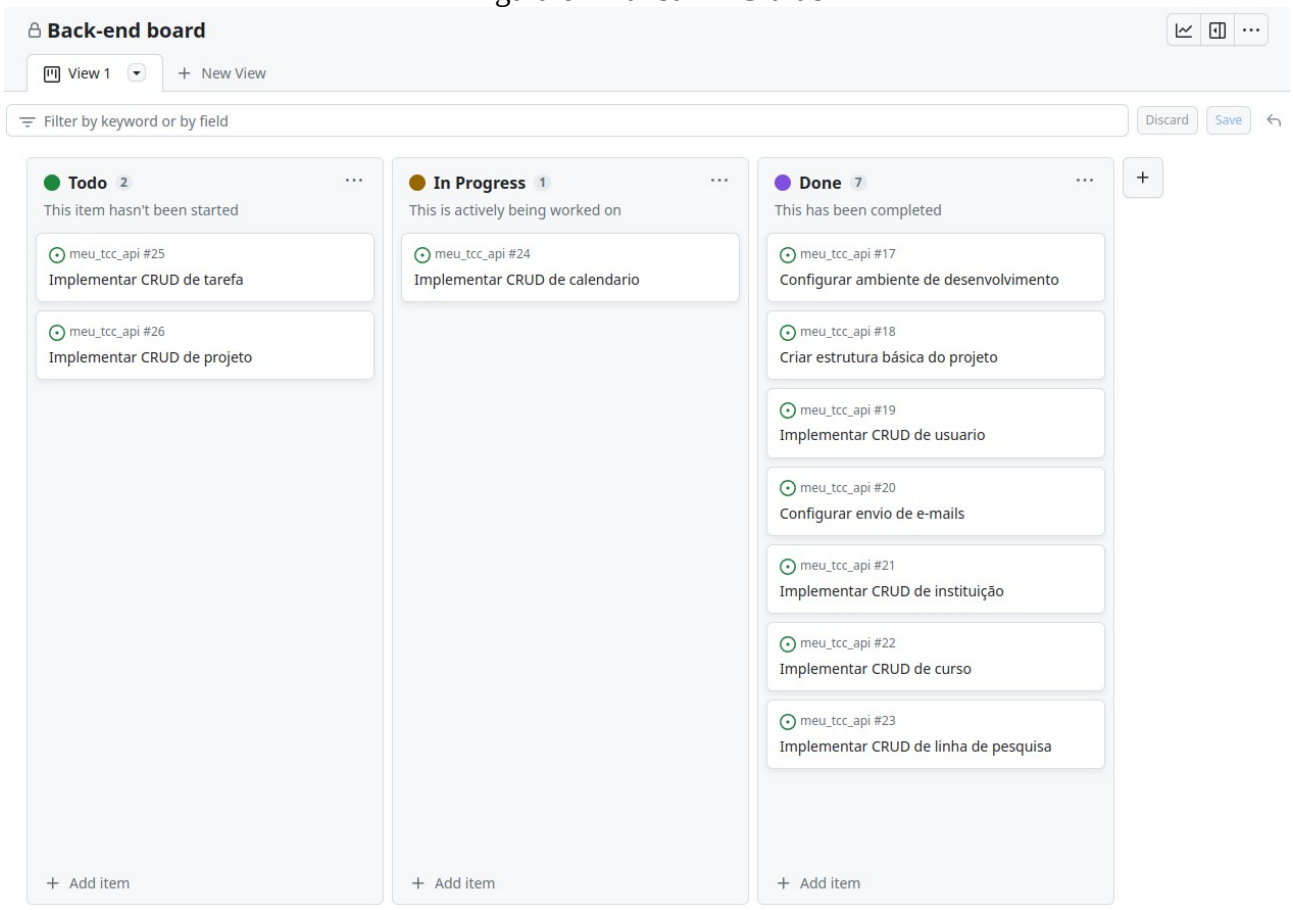
a) Prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
b) As mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
c) Frequentes entregas do software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
d) As pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
e) Os projetos devem ser construídos em torno de indivíduos motivados. Dando o ambiente e o suporte necessário e confiança para fazer o trabalho;
f) O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
g) Software funcionando é a medida primária de progresso;
h) Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
i) Contínua atenção a excelência técnica e bom design aumentam a agilidade;
j) Simplicidade para maximizar, a quantidade de trabalho não realizado é essencial;
k) As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis;
l) Em intervalos regulares, a equipe deve refletir sobre como se tornar mais efetiva, e então, ajustar-se de acordo com seu comportamento.

Com base nisso, Para o desenvolvimento do projeto MeuTCC, foi feito uso da metodologia do Scrum, acompanhado do quadro Kanban do GitHub para monitoração do mesmo. A escolha do Scrum como metodologia se baseou em sua eficácia comprovada em diversos contextos, oferecendo uma abordagem ágil e adaptativa.

O Scrum é uma metodologia de gerenciamento de projetos que prioriza a entrega incremental e iterativa do produto. Ele enfatiza a colaboração, a comunicação constante e a flexibilidade para lidar com mudanças ao longo do processo. Ao adotar o Scrum, a equipe pode se organizar em Sprint definidas, que são períodos de tempo fixos para o desenvolvimento e entrega de incrementos do projeto.

Conforme mostra a figura 6, para acompanhar e visualizar a Sprint do projeto, foi utilizado o quadro Kanban do GitHub. O Kanban é uma ferramenta visual que permite o acompanhamento das tarefas em um fluxo contínuo. Com o quadro Kanban do GitHub, é possível criar colunas representando as etapas do processo, como "A fazer", "Em andamento" e "Concluído" As tarefas são representadas por cartões que podem ser movidos entre as colunas à medida que avançam no fluxo de trabalho.

Figura 6 - Kanbam - Github



Fonte: Autor do trabalho (2023)

A combinação do Scrum com o quadro Kanban do GitHub proporcionou uma abordagem ágil e transparente para o desenvolvimento do projeto. Com o Scrum, pude dividir o trabalho em sprints gerenciáveis, enquanto o quadro Kanban do GitHub facilitou o monitoramento visual das tarefas em tempo real, promovendo a colaboração e o acompanhamento efetivo do progresso do projeto.

4 FERRAMENTAS E CONCEITOS TEÓRICOS

Neste capítulo, descrevemos as ferramentas adotadas para o desenvolvimento do software. O projeto do MeuTCC foi conduzido seguindo boas práticas de desenvolvimento de software, com o objetivo de garantir escalabilidade e eficiência. Para Marques (2022) boas práticas são entendidas como “método ou técnica que tem consistentemente mostrado resultados superiores aos alcançados com outros meios e que é usado como referência”.

Para a criação do projeto, foram selecionadas ferramentas e linguagens de código aberto e licença livre, que proporcionam flexibilidade e ampla comunidade de suporte. Essa abordagem permite uma maior personalização do software e facilita a colaboração entre os desenvolvedores. Ao empregar essas tecnologias, buscamos assegurar a qualidade, desempenho e robustez do software.

Para o desenvolvimento do back-end do projeto, foram utilizadas diversas ferramentas que desempenham funções fundamentais no processo. O brModelo foi adotado para mapear e visualizar o relacionamento das tabelas do banco de dados, facilitando a compreensão da estrutura e organização dos dados. O PostgreSQL foi escolhido como o banco de dados principal devido à sua robustez e flexibilidade, proporcionando um ambiente confiável para armazenamento e manipulação dos dados. O pgAdmin foi utilizado como a interface de administração do banco de dados, permitindo gerenciar de forma eficiente os esquemas, tabelas e consultas.

Além disso, o Swagger foi empregado para documentar o código do back-end, fornecendo uma documentação clara e precisa das APIs desenvolvidas. As tecnologias Node.js e Express foram escolhidas como base para a construção do servidor back-end, fornecendo uma plataforma ágil e eficiente para o desenvolvimento de aplicações web. O TypeORM, acompanhado da utilização de migrations, foi adotado para facilitar a manipulação e organização das entidades e operações do banco de dados.

Por fim, o Docker e o Docker Compose foram utilizados para criar e gerenciar os containers que encapsulam o ambiente de desenvolvimento do projeto. Essas ferramentas garantem a consistência e a portabilidade do ambiente de execução, simplificando o processo de implantação e facilitando a reprodução do ambiente em diferentes máquinas.

Cada uma dessas ferramentas desempenha um papel fundamental no desenvolvimento do back-end, oferecendo recursos e funcionalidades que contribuem para a construção de um sistema eficiente, escalável e de fácil manutenção. Nos próximos sub-tópicos, detalharemos cada uma dessas ferramentas e sua importância no contexto do projeto.

4.1 BANCO DE DADOS

O banco de dados desempenha um papel fundamental em uma aplicação, sendo um dos componentes mais importantes para o armazenamento, gerenciamento e recuperação de dados. Sua importância está diretamente relacionada à necessidade de persistência de informações em uma aplicação, permitindo que os dados sejam armazenados de forma estruturada e organizada.

Um banco de dados eficiente e bem projetado oferece várias vantagens para uma aplicação. Primeiramente, ele permite que os dados sejam armazenados de maneira confiável e segura, evitando perdas ou corrupção das informações. Além disso, o banco de dados possibilita a realização de consultas e operações complexas sobre os dados, permitindo a obtenção de informações relevantes de forma rápida e eficiente.

Outro aspecto importante é a escalabilidade. Com o crescimento da aplicação e o aumento do volume de dados, um banco de dados adequado permite lidar com o aumento da demanda sem comprometer o desempenho ou a integridade dos dados. A capacidade de expansão e adaptação do banco de dados é crucial para acompanhar o crescimento e as necessidades da aplicação.

Além disso, um banco de dados bem projetado e otimizado pode contribuir para o desempenho da aplicação como um todo, garantindo respostas rápidas às consultas e minimizando o tempo de acesso aos dados. Isso resulta em uma melhor experiência para o usuário final, com tempos de resposta mais curtos e maior eficiência operacional.

Em resumo, o banco de dados desempenha um papel central no desenvolvimento de uma aplicação, sendo responsável pelo armazenamento seguro e eficiente dos dados. Sua importância está diretamente relacionada à integridade dos dados, escalabilidade, desempenho e capacidade de adaptação da aplicação às necessidades dos usuários.

4.1.1 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto amplamente utilizado em aplicações modernas. Ele é conhecido por sua confiabilidade, flexibilidade e recursos avançados, tornando-o uma escolha popular para muitos projetos. As origens do PostgreSQL remontam a 1986 como parte do projeto POSTGRES da Universidade da Califórnia em Berkeley e tem mais de 35 anos de desenvolvimento ativo na plataforma principal (POSTGRESQL..., [s.d.]).

Uma das principais vantagens do PostgreSQL é sua capacidade de lidar com grandes volumes de dados e cargas de trabalho intensas. Ele oferece suporte a transações ACID, o que

garante a integridade dos dados mesmo em cenários complexos. Além disso, o PostgreSQL possui um otimizador de consultas sofisticado que melhora o desempenho das consultas e permite a execução eficiente de operações complexas.

Outro aspecto importante do PostgreSQL é sua extensibilidade. Ele oferece suporte a uma ampla gama de tipos de dados, incluindo tipos personalizados, e permite a criação de funções e procedimentos armazenados personalizados. Isso permite que os desenvolvedores estendam as capacidades do banco de dados de acordo com as necessidades específicas de sua aplicação.

O PostgreSQL também é conhecido por sua segurança robusta. Ele oferece recursos avançados de autenticação e controle de acesso, permitindo que os administradores definam permissões granulares para usuários e grupos. Além disso, o PostgreSQL suporta criptografia de dados em trânsito e em repouso, garantindo a confidencialidade das informações armazenadas.

Outra vantagem do PostgreSQL é sua ampla compatibilidade com padrões SQL. Ele suporta a maioria dos recursos do SQL padrão, tornando-o interoperável com outras ferramentas e sistemas que também seguem os padrões SQL. Além disso, o PostgreSQL oferece suporte a recursos avançados, como consultas complexas, junções, subconsultas e operações espaciais.

Em resumo, o PostgreSQL é um sistema de gerenciamento de banco de dados poderoso e confiável, adequado para uma ampla gama de aplicações. Sua capacidade de lidar com grandes volumes de dados, sua extensibilidade, segurança e compatibilidade com padrões SQL o tornam uma escolha sólida para projetos que exigem um banco de dados robusto e flexível.

4.2 LINGUAGEM DE PROGRAMAÇÃO

Uma linguagem de programação é um conjunto de regras e instruções que permitem que os programadores escrevam códigos para criar programas de computador. Elas fornecem uma maneira estruturada e formal de se comunicar com o computador, permitindo especificar ações a serem executadas.

As linguagens de programação possuem uma sintaxe específica, que define a forma correta de escrever o código, incluindo palavras-chave, símbolos, estruturas de controle e convenções de formatação. Cada linguagem tem sua própria sintaxe e conjunto de recursos, que determinam a maneira como os programas são escritos e interpretados pelo computador.

Existem diferentes categorias de linguagens de programação, como linguagens de alto nível, baixo nível e de script, cada uma com suas características e finalidades específicas.

Essas linguagens são usadas para desenvolver uma ampla variedade de aplicativos e sistemas, desde aplicativos de desktop, móveis e web até sistemas operacionais e software de

controle de hardware. A escolha da linguagem de programação depende dos requisitos do projeto, das preferências do programador e do ambiente de desenvolvimento.

Em resumo, as linguagens de programação são ferramentas que permitem aos programadores escreverem instruções para o computador, de forma estruturada e precisa. Elas possibilitam a criação de programas e aplicativos para atender a diversas necessidades e finalidades.

4.2.1 JavaScript/NodeJs

O JavaScript figura hoje como uma das linguagens mais utilizadas, e em grande parte isso se deve ao fato de ser uma linguagem base para dezenas de frameworks com alta popularidade e adesão na comunidade de desenvolvimento (BESSA, 2023). Originalmente concebida para ser executada nos navegadores, o JavaScript permite a criação de interações dinâmicas em páginas HTML. No entanto, com o advento do Node.js, o JavaScript passou a ser utilizado também no lado do servidor.

“Node.js é um ambiente de execução do código JavaScript do lado servidor (server side), que na prática se reflete na possibilidade de criar aplicações standalone (autossuficientes) em uma máquina servidora, sem a necessidade do navegador” (BESSA, 2023). Ele permite que o JavaScript seja executado fora do navegador, possibilitando a construção de aplicativos do lado do servidor, ferramentas de linha de comando e outros tipos de aplicações que não estão diretamente relacionadas à web.

Uma das principais vantagens do Node.js é sua natureza assíncrona e orientada a eventos. Isso significa que o Node.js é altamente eficiente para lidar com um grande número de conexões simultâneas, o que o torna ideal para o desenvolvimento de aplicações em tempo real, como bate-papo em tempo real, streaming de dados e aplicações colaborativas.

No contexto do desenvolvimento web, o JavaScript e o Node.js trabalham em conjunto para oferecer uma solução abrangente. Enquanto o JavaScript é utilizado no lado do cliente para manipular o DOM, lidar com eventos e fornecer uma experiência interativa para o usuário, o Node.js é usado no lado do servidor para processar solicitações, gerenciar banco de dados, realizar lógica de negócio e fornecer dados para o cliente.

Além disso, o ecossistema do Node.js é rico em bibliotecas e frameworks, como o Express.js, que facilitam o desenvolvimento de aplicativos web robustos e escaláveis. O Node.js também possui recursos poderosos, como o gerenciamento de pacotes NPM (Node Package Manager), que oferece acesso a uma ampla gama de módulos e bibliotecas reutilizáveis.

Em resumo, o JavaScript e o Node.js trabalham juntos para fornecer uma solução completa de desenvolvimento web. Enquanto o JavaScript atua no lado do cliente, oferecendo interatividade e comportamentos dinâmicos às páginas web, o Node.js possibilita a construção de aplicações do lado do servidor, eficientes, escaláveis e em tempo real. Essa combinação poderosa torna o JavaScript e o Node.js uma escolha popular para o desenvolvimento de aplicações web modernas.

4.3 GERENCIAMENTO DE AMBIENTES DE DESENVOLVIMENTO

No desenvolvimento de software, o gerenciamento de ambientes de desenvolvimento desempenha um papel essencial para garantir a consistência e a eficiência no processo de criação de aplicações. Um ambiente de desenvolvimento consiste em todos os recursos necessários, como sistemas operacionais, bibliotecas, frameworks e ferramentas, que permitem aos desenvolvedores escrever, testar e depurar o código de forma eficiente.

O gerenciamento de ambientes de desenvolvimento envolve a criação e manutenção desses ambientes, garantindo que todos os membros da equipe tenham acesso às mesmas configurações e dependências. Isso é fundamental para garantir que a aplicação funcione corretamente em diferentes estágios do desenvolvimento, desde o ambiente de desenvolvimento local até a implantação em produção.

4.3.1 Docker e Docker Compose

Docker e Docker Compose são ferramentas amplamente utilizadas para simplificar o gerenciamento de ambientes de desenvolvimento. Para Willians (2022) o Docker é uma plataforma de virtualização de nível de sistema operacional que permite empacotar e executar aplicativos em contêineres isolados. Esses contêineres encapsulam todas as dependências necessárias para a execução da aplicação, como bibliotecas, frameworks e até mesmo o sistema operacional.

Com o Docker, os desenvolvedores podem criar contêineres independentes, garantindo que a aplicação seja executada de maneira consistente, independentemente do ambiente de desenvolvimento em que estejam trabalhando. Isso elimina problemas comuns relacionados a diferenças de configuração e dependências entre os desenvolvedores, facilitando a colaboração e reduzindo os conflitos.

Docker Compose é uma solução multi-container que utiliza um arquivo declarativo para definir os serviços que você quer rodar, sendo que cada serviço é basicamente um container que você pode executar (WILLIANS, 2022). Com o Docker Compose, é possível especificar as

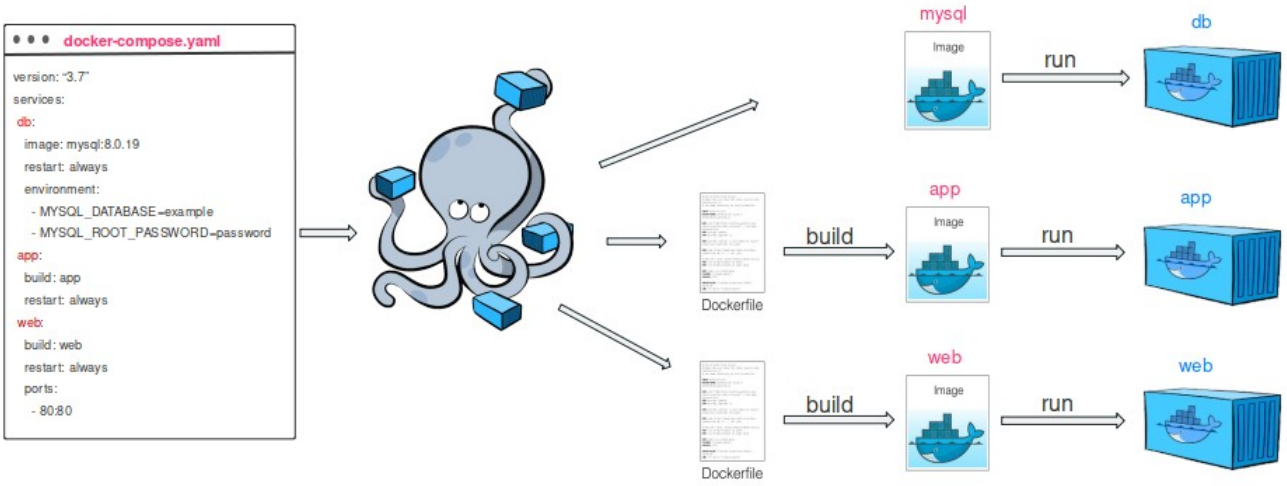
dependências entre os contêineres, as configurações de rede e outras configurações relevantes. Isso simplifica o processo de configuração e execução de aplicativos complexos, garantindo a consistência e a integração entre os diferentes componentes da aplicação.

A utilização do Docker e Docker Compose traz diversos benefícios para o gerenciamento de ambientes de desenvolvimento (como exemplifica a imagem 7). Eles oferecem portabilidade, permitindo que a aplicação seja executada em diferentes ambientes sem alterações significativas. Além disso, facilitam a reprodução do ambiente em diferentes máquinas e etapas do ciclo de vida do software.

O uso do Docker e Docker Compose também promove a escalabilidade e a flexibilidade, permitindo aumentar ou diminuir o número de contêineres de acordo com a demanda. Isso é especialmente importante quando se lida com picos de tráfego ou quando há a necessidade de implantar a aplicação em diferentes ambientes, como desenvolvimento, teste e produção.

Em resumo, o uso do Docker e Docker Compose no gerenciamento de ambientes de desenvolvimento traz eficiência, consistência e escalabilidade para o processo de desenvolvimento de software. Essas ferramentas simplificam a criação e o compartilhamento de ambientes de desenvolvimento, proporcionando um fluxo de trabalho mais eficiente e colaborativo para a equipe de desenvolvimento.

Figura 7 - Docker e docker compose



Fonte: https://pavankumar07.hashnode.dev/docker-compose

5 DESENVOLVIMENTO DO BACK-END

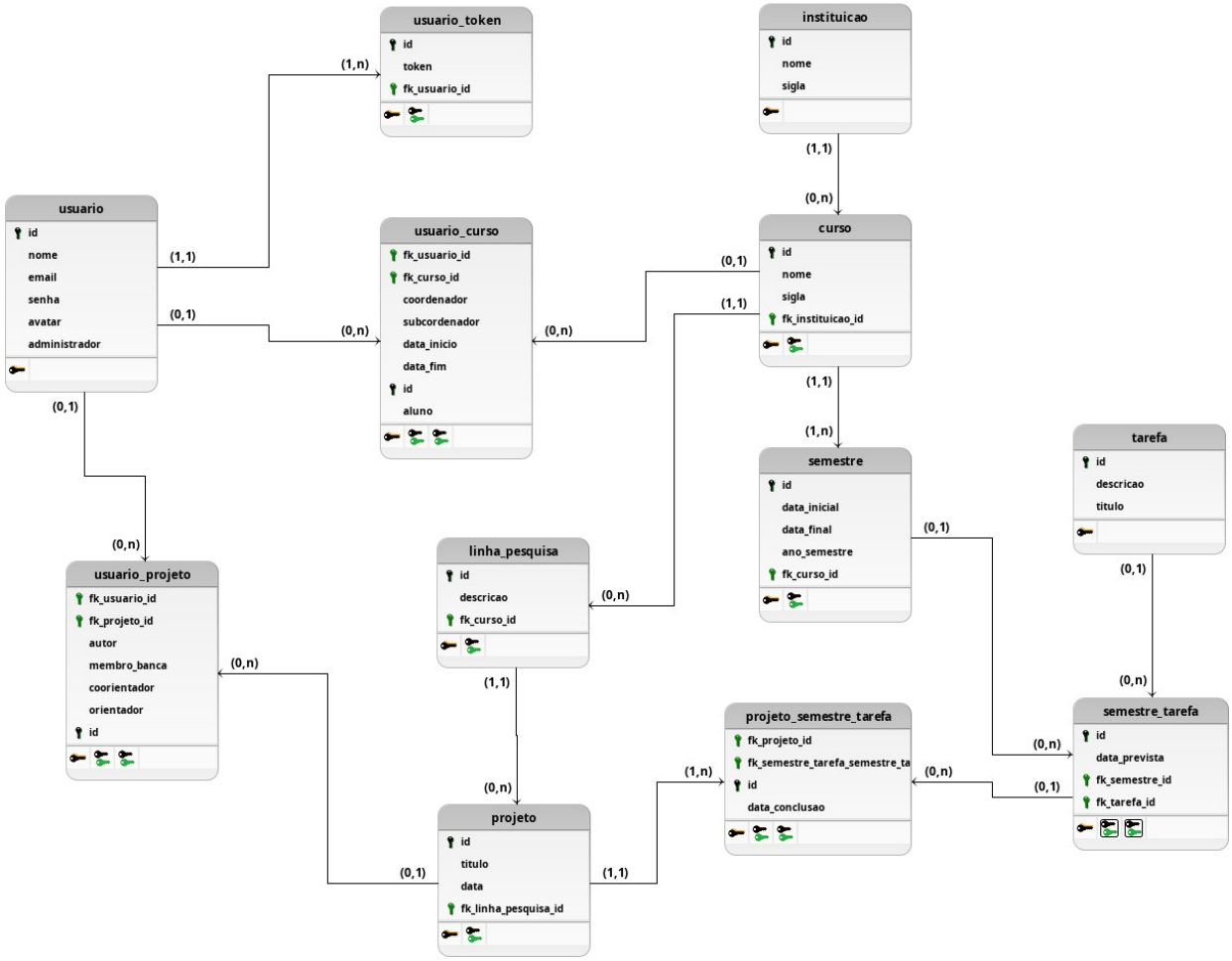
O desenvolvimento do back-end é uma etapa essencial no processo de criação de aplicações web. O back-end é responsável por processar as solicitações dos usuários, gerenciar os dados e fornecer as respostas adequadas. Nessa fase, são utilizadas linguagens de programação, como JavaScript, e frameworks, como Node.js, para construir a lógica de negócios, integrar com bancos de dados e implementar as funcionalidades da aplicação. O desenvolvimento do back-end requer o conhecimento de conceitos como rotas, controladores, modelos e serviços, que permitem criar uma arquitetura robusta e escalável. Além disso, é comum utilizar ferramentas como o Express.js e o TypeORM para facilitar o desenvolvimento, fornecendo recursos e abstrações que agilizam o processo de criação de APIs e a interação com o banco de dados.

5.1 MAPEAMENTO DO BANCO DE DADOS

No processo de desenvolvimento do back-end, uma etapa fundamental é o mapeamento do banco de dados e a definição das tabelas. Para realizar essa tarefa, foi utilizado o brModelo na sua versão desktop, uma ferramenta desenvolvida na linguagem java que auxilia o usuário durante a elaboração do mapeamento do banco de dados. O brModelo permitiu visualizar e planejar a estrutura do banco de dados de forma intuitiva e organizada (conforme mostra a figura 8).

Por meio dessa ferramenta, foi possível criar entidades, atributos e relacionamentos, além de estabelecer as chaves primárias e estrangeiras. O brModelo facilitou o processo de definição das tabelas e seus atributos, fornecendo uma representação gráfica clara e compreensível do esquema do banco de dados. Com esse mapeamento bem estruturado, foi possível garantir a integridade dos dados e a eficiência nas operações realizadas pelo sistema, contribuindo para um desenvolvimento mais fluido e consistente.

Figura 8 - Modelagem do banco de dados



Fonte: Autor do trabalho (2023)

5.2 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

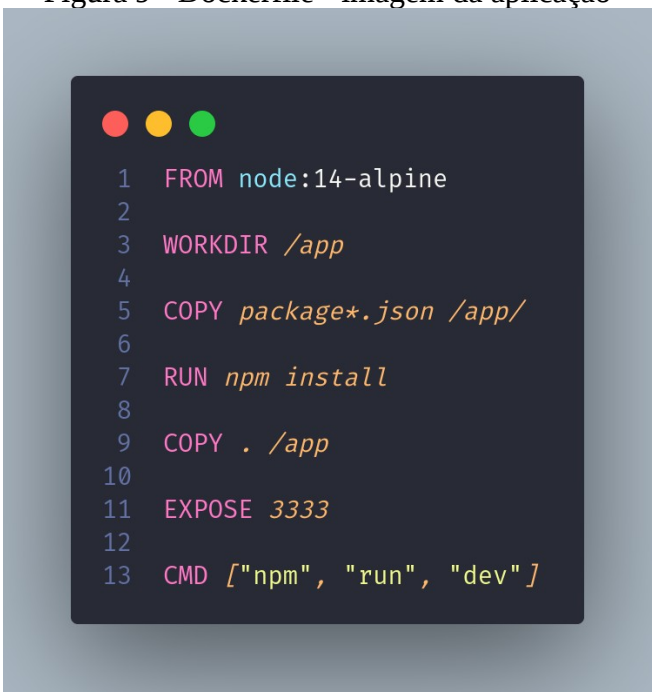
Nesta etapa, é realizada a configuração do ambiente de desenvolvimento para a aplicação. Uma abordagem utilizada é o uso do Docker e do Docker Compose, que facilitam a criação e o gerenciamento de containers para os componentes necessários.

5.2.1 Containers

O Docker permite empacotar a aplicação, suas dependências e configurações em um container isolado, garantindo a portabilidade e a consistência do ambiente de desenvolvimento. Isso significa que os desenvolvedores podem trabalhar em um ambiente controlado e reproduzível, independentemente do sistema operacional e das configurações locais.

Uma das etapas iniciais é a criação do Dockerfile, que define as instruções para construir a imagem do container da aplicação. Neste arquivo, é possível especificar a versão do Node.js a ser utilizada, copiar o código fonte da aplicação para dentro do container e configurar as dependências necessárias (como demonstra a imagem 9).

Figura 9 - Dockerfile - imagem da aplicação



Fonte: Autor do trabalho

Além disso, o Docker Compose simplifica o gerenciamento de múltiplos containers, permitindo definir e configurar todos os serviços necessários em um arquivo YAML. Isso facilita a configuração do banco de dados PostgreSQL e do pgAdmin, que são componentes essenciais para o desenvolvimento do back-end.

No contexto do projeto, são criados três containers principais:

a) Container da aplicação: contém o código fonte e todas as dependências necessárias para a execução da aplicação. Esse container pode ser construído a partir de uma imagem base, como uma imagem Node.js, e configurado com as bibliotecas e frameworks específicos do projeto.
b) Container do PostgreSQL: responsável por hospedar o banco de dados utilizado pela aplicação. É configurado com as credenciais de acesso, as configurações de segurança e as definições de volume para persistência dos dados.

c) Container do pgAdmin: fornece uma interface gráfica para gerenciamento e visualização do banco de dados PostgreSQL. É configurado com as credenciais de acesso e as configurações de conexão com o banco de dados.

A figura 10 ilustra a estrutura básica do ambiente de desenvolvimento com Docker e Docker Compose:

Figura 10 - Docker compose

```yaml
version: '3.4'

services:
  app:
    container_name: container-meutcc-api
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - '3333:3333'
    networks:
      - meu-tcc-network
    stdin_open: true
    tty: true
    volumes:
      - ".:/app"
      - "/app/node_modules"
    environment:
      TZ: America/Sao_Paulo

  database_postgres:
    container_name: container-meutcc-postgres
    image: postgres:14.2-alpine
    environment:
      - POSTGRES_USER=${DB_USERNAME}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
      - POSTGRES_DB=${DB_NAME}
    ports:
      - '${DB_PORT}:5432'
    networks:
      - meu-tcc-network
    restart: always

  pgadmin:
    container_name: container-meutcc-pgadmin
    image: dpage/pgadmin4
    environment:
      - PGADMIN_DEFAULT_EMAIL=meutcc@meutcc.com
      - PGADMIN_DEFAULT_PASSWORD=meutcc123
    ports:
      - '5050:80'
    networks:
      - meu-tcc-network
    restart: always
    volumes:
      - pgadmin:/root/.pgadmin
    logging:
      driver: none
volumes:
  app:
  database_postgres:
  pgadmin:

networks:
  meu-tcc-network:
```

Fonte: Autor do trabalho (2023)

Ao utilizar o Docker e o Docker Compose para configurar o ambiente de desenvolvimento, é possível ter um ambiente consistente e replicável em diferentes máquinas, simplificando a configuração inicial e evitando possíveis conflitos entre dependências e configurações.

5.2.2 Conexão com o banco de dados

No processo de conexão com o banco de dados, é utilizado o arquivo *ormconfig.ts* para configurar as opções de conexão do TypeORM. Esse arquivo é responsável por definir os parâmetros necessários para estabelecer a comunicação entre a aplicação e o banco de dados PostgreSQL (conforme mostra a imagem 11).

Figura 11 - Conexão com banco de dados

```
import { ConnectionOptions } from 'typeorm';

const config: ConnectionOptions = {
  type: 'postgres',
  host: process.env.DB_HOST,
  port: Number(process.env.DB_PORT),
  username: process.env.DB_USERNAME,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  synchronize: false,
  logging: false,
  entities: ['src/app/entity/**/*.{ts,js}'],
  migrations: ['src/database/migrations/**/*.{ts,js}'],
  subscribers: ['src/subscriber/**/*.{ts,js}'],
  cli: {
    entitiesDir: 'src/app/entity',
    migrationsDir: 'src/database/migrations',
    subscribersDir: 'src/subscriber',
  },
};

export default config;
```

Fonte: autor do trabalho (2023)

5.2.3 Variáveis de ambiente

Variáveis de ambiente são aquelas que definimos fora de um programa, geralmente por um provedor da nuvem ou um sistema operacional. No Node, as variáveis de ambiente são uma ótima maneira de configurar com segurança e conveniência questões que não se alteram com frequência, como URLs, chaves de autenticação e senhas (STORK, 2022).

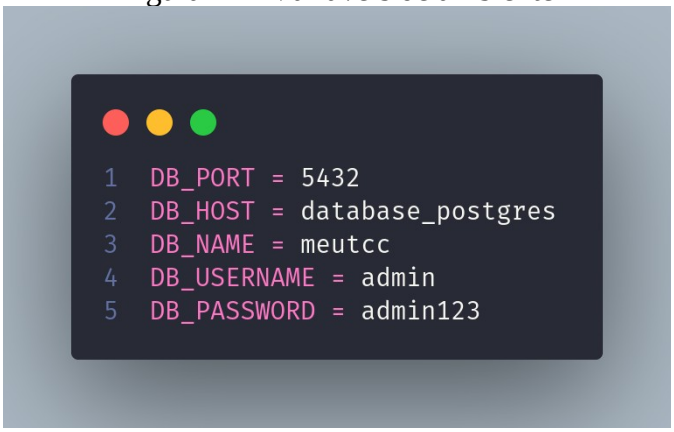
A configuração das variáveis de ambiente é essencial para a correta conexão e utilização do banco de dados. No arquivo *.env*, são definidas as variáveis que contêm informações como a porta, o host, o nome do banco de dados, o nome de usuário e a senha. Essas informações são importantes para estabelecer a comunicação entre a aplicação e o banco de dados PostgreSQL.

No contexto do projeto, as variáveis de ambiente são utilizadas tanto no arquivo *docker-compose.yml*, para configurar o container do banco de dados, quanto no arquivo *ormconfig.ts*, utilizado pelo TypeORM para estabelecer a conexão com o banco de dados durante a execução da aplicação.

Através dessas configurações, é possível garantir que a aplicação se conecte ao banco de dados corretamente, utilizando as credenciais e as configurações específicas definidas nas variáveis de ambiente.

Como ilustra a imagem a utilização das variáveis de ambiente no arquivo .env, no docker-compose.yml e no ormconfig.ts:

Figura 12 - Variáveis de ambiente



Fonte: Autor do trabalho (2023)

Ao utilizar variáveis de ambiente, é possível manter as informações sensíveis, como senhas e chaves de acesso, fora do código fonte e protegê-las adequadamente. Isso aumenta a segurança da aplicação e facilita a configuração e o gerenciamento das diferentes configurações de ambiente, tornando o desenvolvimento mais flexível e adaptável às necessidades do projeto.

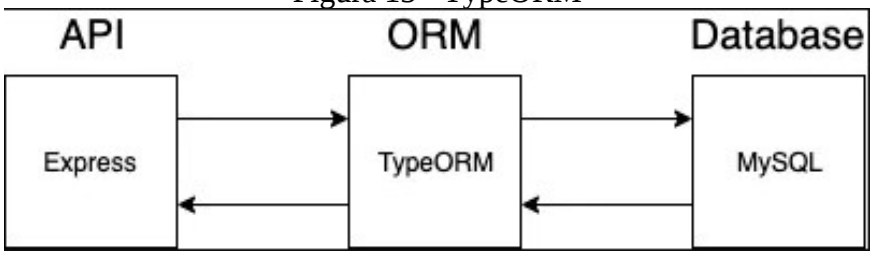
5.3 DESENVOLVIMENTO DA LÓGICA DE NEGÓCIO

No estágio de desenvolvimento da lógica de negócio, ocorre a implementação das funcionalidades centrais da aplicação. Nessa etapa, são criados os modelos e entidades do banco de dados usando o TypeORM, que permite definir a estrutura dos dados de forma intuitiva e interagir com o banco de dados de maneira simplificada. Além disso, é realizada a implementação dos controladores e rotas utilizando o framework Express.js, que facilita o gerenciamento das requisições e respostas HTTP. Também são definidas as regras de negócio e implementados os serviços necessários para o funcionamento adequado do sistema. O desenvolvimento da lógica de negócio é crucial para construir uma aplicação completa e funcional, que atenda às necessidades do usuário e alcance os objetivos do negócio.

5.3.1 Criação dos modelos e entidades do banco de dados usando TypeORM

O TypeORM é um framework que pode ser utilizado com Node.js, utilizando Javascript ou Typescript. Essa ferramenta possui um recurso muito útil chamado *migration*, que pode ser descrito como uma espécie de controle de versão do banco de dados (MENDES, 2022).

Figura 13 - TypeORM



Fonte: https://betterprogramming.pub/typeorm-migrations-explained-fdb4f27cb1b3

O TypeORM é uma ferramenta muito útil para trabalhar com bancos de dados relacionais. Ele facilita o acesso aos dados e torna o processo de criação de consultas e relacionamentos mais simples e intuitivo, oferecendo muitas funcionalidades para o nosso projeto (KRIGER, 2023).

Além de definir as entidades, o TypeORM também fornece recursos para criar e gerenciar as migrações do banco de dados. As migrações são scripts que representam alterações na estrutura do banco de dados, como a criação de tabelas, adição de colunas, alteração de relacionamentos e muito mais. Com as migrações, é possível manter um histórico das alterações realizadas no banco de dados ao longo do tempo e garantir que todas as instâncias do sistema estejam sincronizadas. Essas migrações são executadas de forma sequencial e controlada, permitindo que o banco de dados

evolua de maneira consistente, acompanhando as mudanças no modelo de negócio e as necessidades da aplicação. Como exemplo, na imagem a seguir, podemos observar uma migração do projeto que adiciona uma nova tabela "Usuario" ao banco de dados.

Figura 14 - Migration - usuário

```
export class CriaUsuario1647559969442 implements MigrationInterface {
  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query('CREATE EXTENSION IF NOT EXISTS "uuid-ossp"');
    await queryRunner.createTable(
      new Table({
        name: 'usuario',
        columns: [
          {
            name: 'id',
            type: 'uuid',
            isPrimary: true,
            generationStrategy: 'uuid',
            default: 'uuid_generate_v4()',
          },
          {
            name: 'nome',
            type: 'varchar',
            length: '100',
          },
          {
            name: 'email',
            type: 'varchar',
            isUnique: true,
          },
          {
            name: 'senha',
            type: 'varchar',
          },
          {
            name: 'created_at',
            type: 'timestamp with time zone',
            default: 'now()',
          },
          {
            name: 'updated_at',
            type: 'timestamp with time zone',
            default: 'now()',
          },
        ],
      }),
    );
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropTable('usuario');
    await queryRunner.query('DROP EXTENSION "uuid-ossp"');
  }
}
```

Fonte: Autor do trabalho (2023)

O uso das migrações com o TypeORM facilita a evolução do banco de dados de forma controlada e consistente, permitindo que a estrutura do banco acompanhe as mudanças no modelo de negócio e as necessidades da aplicação. Isso proporciona uma maior flexibilidade e segurança no desenvolvimento, garantindo a integridade dos dados e simplificando a implementação de novas funcionalidades.

5.3.2 Implementação dos controladores e rotas usando o framework Express.js

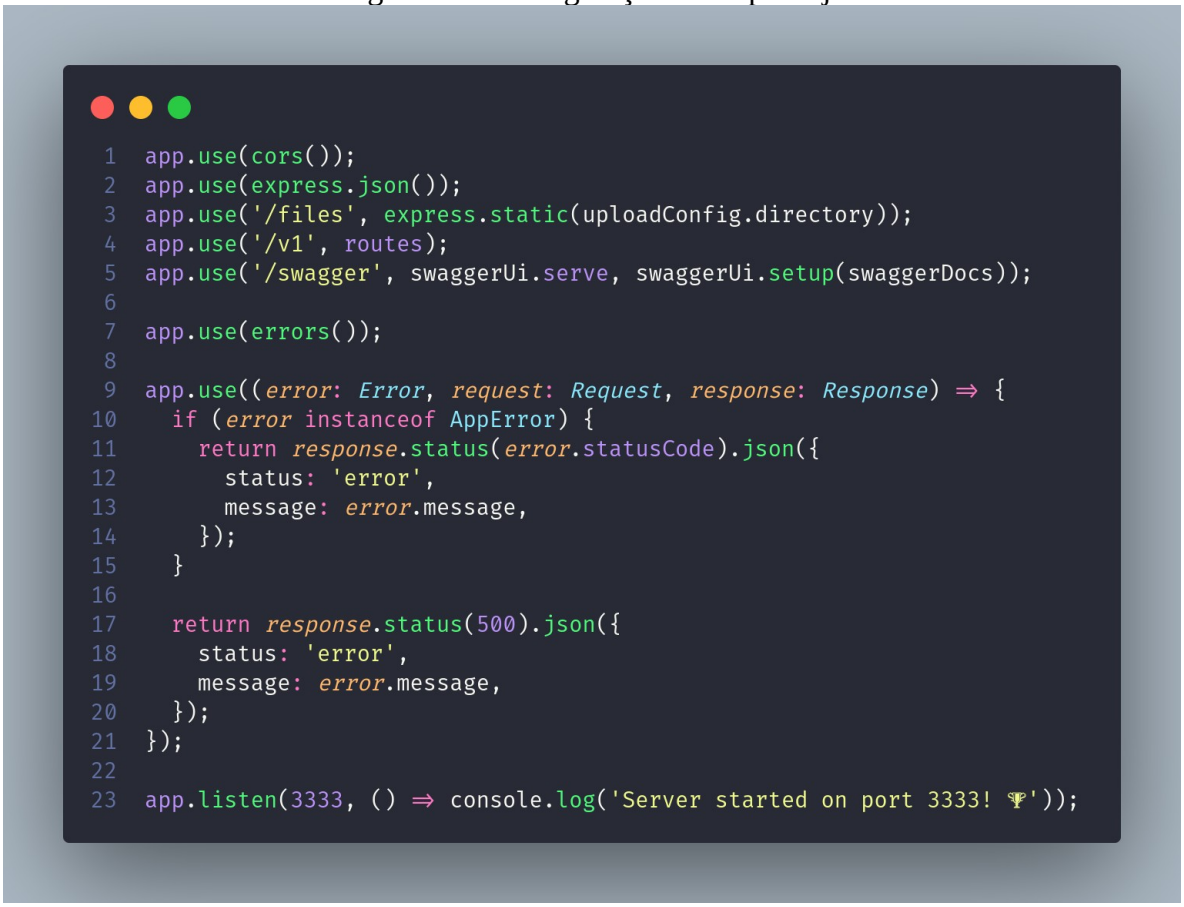
O Express.js é um Framework rápido e um dos mais utilizados em conjunto com o Node.js, facilitando no desenvolvimento de aplicações back-end e até, em conjunto com sistemas de templates, aplicações full-stack. Escrito em JavaScript, o Express.js é utilizado por diversas empresas ao redor do mundo, dentre elas a Fox Sports, PayPal, IBM, Uber, entre outras. Muito popular tanto em grandes empresas quanto na comunidade, o Express facilita a criação de aplicações utilizando o Node em conjunto com o JavaScript, tornando este ecossistema ainda mais poderoso (ANDRADE, 2021).

A implementação dos controladores e rotas usando o framework Express.js desempenha um papel essencial no desenvolvimento do back-end do projeto. Os controladores atuam como intermediários entre as requisições recebidas dos clientes e as ações a serem executadas. Eles lidam com a lógica de negócio da aplicação, processando os dados, realizando operações no banco de dados e retornando as respostas apropriadas. Através dos controladores, é possível definir as diferentes funcionalidades e endpoints da API.

As rotas, por sua vez, são responsáveis por mapear as URLs e direcionar as requisições para os controladores correspondentes. Elas estabelecem as diferentes vias de acesso à API, permitindo que os clientes interajam com o sistema por meio de solicitações HTTP. Cada rota define um endpoint específico e define o método HTTP associado (como GET, POST, PUT, DELETE, entre outros) que será utilizado para acessá-la.

No contexto do projeto, a implementação dos controladores e rotas usando o Express.js pode ser visualizada nas imagens 15 e 16.

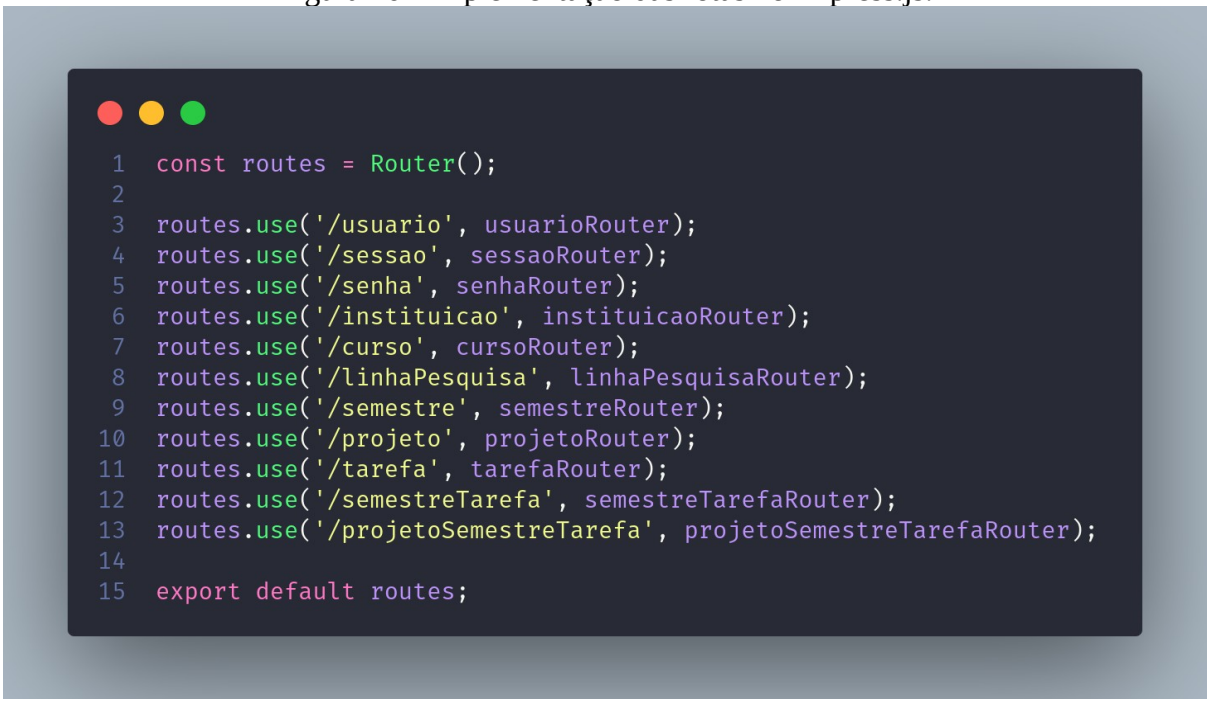
Figura 15 - Configuração do Express.js



```
app.use(cors());
app.use(express.json());
app.use('/files', express.static(uploadConfig.directory));
app.use('/v1', routes);
app.use('/swagger', swaggerUi.serve, swaggerUi.setup(swaggerDocs));

app.use(errors());

app.use((error: Error, request: Request, response: Response) => {
  if (error instanceof AppError) {
    return response.status(error.statusCode).json({
      status: 'error',
      message: error.message,
    });
  }

  return response.status(500).json({
    status: 'error',
    message: error.message,
  });
});

app.listen(3333, () => console.log('Server started on port 3333! 🏆'));
```

Fonte: Autor do trabalho (2023)

Figura 16 - Implementação das rotas no Express.js.



```
const routes = Router();

routes.use('/usuario', usuarioRouter);
routes.use('/sessao', sessaoRouter);
routes.use('/senha', senhaRouter);
routes.use('/instituicao', instituicaoRouter);
routes.use('/curso', cursoRouter);
routes.use('/linhaPesquisa', linhaPesquisaRouter);
routes.use('/semestre', semestreRouter);
routes.use('/projeto', projetoRouter);
routes.use('/tarefa', tarefaRouter);
routes.use('/semestreTarefa', semestreTarefaRouter);
routes.use('/projetoSemestreTarefa', projetoSemestreTarefaRouter);

export default routes;
```

Fonte: Autor do trabalho (2023)

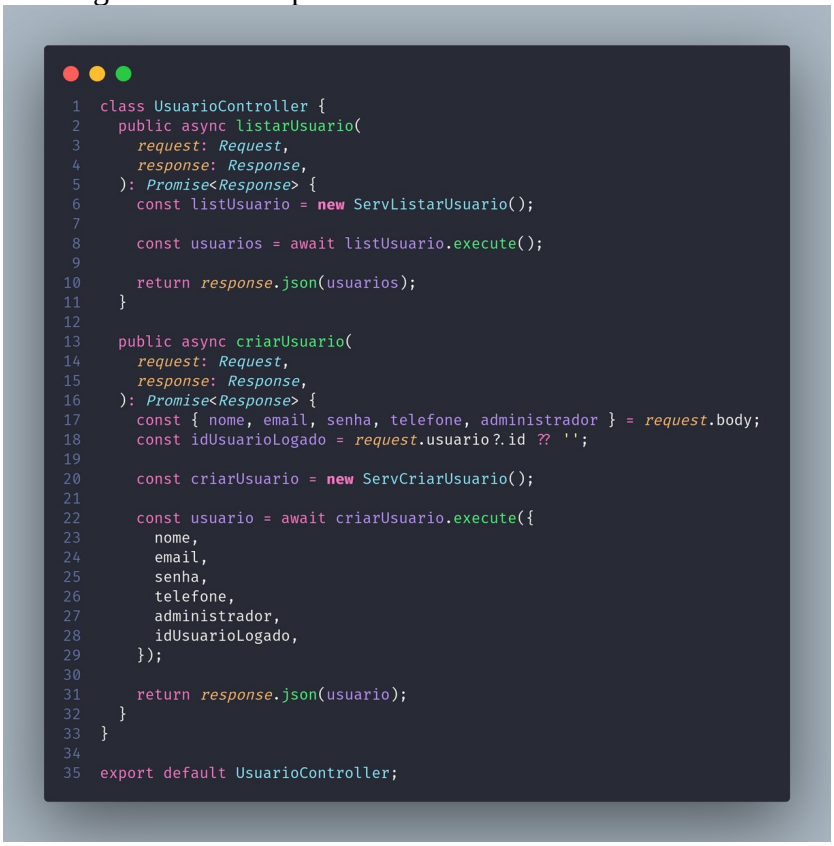
Os controladores são componentes cruciais na implementação da lógica de negócio da aplicação. Eles atuam como intermediários entre as requisições recebidas dos clientes e as ações a serem executadas. Os *controllers* atuam como intermediários entre a camada de transporte (HTTP) e a lógica de negócio da aplicação são responsáveis por processar os dados e fazem chamadas aos serviços (conforme ilustra as imagens 17 e 18), que contêm a lógica do negócio. Com a ajuda dos *controllers*, é possível definir as diferentes funcionalidades e endpoints da API, organizando e estruturando a lógica de negócio de forma modular.

Figura 17 - Rota - usuário

```
const usuarioRouter = Router();
const usuarioController = new UsuarioController();
const usuarioAvatarController = new UsuarioAvatarController();
const upload = multer(uploadConfig);

const defaultData = {
  nome: Joi.string().required(),
  email: Joi.string().email().required(),
  senha: Joi.string().required().min(8),
  telefone: Joi.string().required(),
};

usuarioRouter.get('/', isAuthenticated, usuarioController.listarUsuario);

usuarioRouter.post(
  '/',
  isAuthenticated,
  celebrate({
    [Segments.BODY]: {
      ...defaultData,
      administrador: Joi.boolean(),
    },
  }),
  usuarioController.criarUsuario,
);

usuarioRouter.post(
  '/aluno',
  celebrate({
    [Segments.BODY]: {
      ...defaultData,
      matricula: Joi.string().required(),
      idCurso: Joi.string().uuid().required(),
    },
  }),
  usuarioController.criarAluno,
);

usuarioRouter.put(
  '/',
  celebrate({
    [Segments.BODY]: {
      ...defaultData,
      id: Joi.string().uuid().required(),
      confirmacaoSenha: Joi.string().required().valid(Joi.ref('senha')),
    },
  }),
  usuarioController.atualizarUsuario,
);

usuarioRouter.patch(
  '/avatar',
  isAuthenticated,
  upload.single('avatar'),
  usuarioAvatarController.update,
);

usuarioRouter.patch(
  '/aprovar_cadastro/:id',
  celebrate({
    [Segments.PARAMS]: { id: Joi.string().uuid().required() },
    [Segments.BODY]: { cadastroAprovado: Joi.boolean().required() },
  }),
  usuarioController.aprovarCadastro,
);

export default usuarioRouter;
```

Fonte: Autor do trabalho (2023)

Figura 18 - Exemplo da classe de usuário no controller



```
class UsuarioController {
  public async listarUsuario(
    request: Request,
    response: Response,
  ): Promise<Response> {
    const listUsuario = new ServListarUsuario();

    const usuarios = await listUsuario.execute();

    return response.json(usuarios);
  }

  public async criarUsuario(
    request: Request,
    response: Response,
  ): Promise<Response> {
    const { nome, email, senha, telefone, administrador } = request.body;
    const idUsuarioLogado = request.usuario?.id ?? '';

    const criarUsuario = new ServCriarUsuario();

    const usuario = await criarUsuario.execute({
      nome,
      email,
      senha,
      telefone,
      administrador,
      idUsuarioLogado,
    });

    return response.json(usuario);
  }
}

export default UsuarioController;
```

Fonte: Autor do trabalho (2023)

Dessa forma, a combinação do Express.js, das rotas e dos controladores proporciona uma arquitetura sólida e escalável para o back-end do projeto, permitindo o processamento eficiente das requisições e a implementação das regras de negócio de forma organizada e modularizada.

5.3.3 Implementação dos serviços

A implementação dos serviços desempenha um papel fundamental no desenvolvimento do back-end. Os serviços são responsáveis por encapsular a lógica de negócio da aplicação e fornecer funcionalidades específicas. Eles atuam como intermediários entre os controladores e o acesso aos dados no banco de dados, permitindo a execução de operações como cadastrar, atualizar, remover e listar informações.

Os serviços são criados com base nas regras de negócio do projeto, definindo as operações e validações necessárias para garantir a integridade e consistência dos dados. Essa abordagem modularizada facilita a manutenção do código e promove a reutilização de lógica em diferentes partes do sistema (como exemplifica a imagem 19).

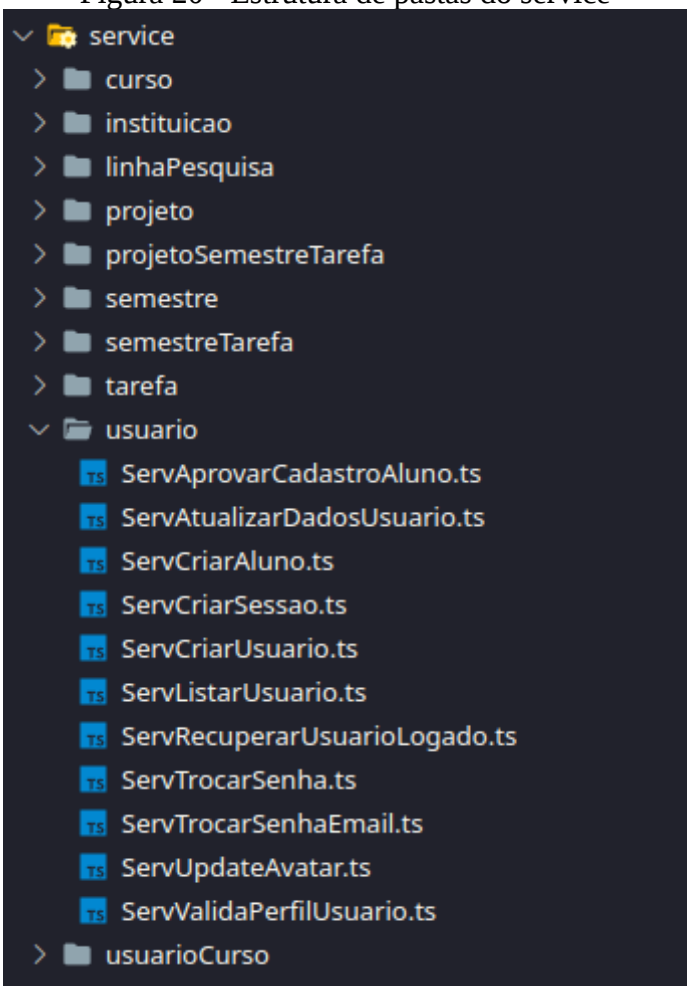
Figura 19 - Serviço para criar sessão

```
class ServCriarSessao {
  public async execute({ email, senha }: IRequest): Promise<IResponse> {
    const repUsuario = getCustomRepository(RepUsuario);
    const usuario = await repUsuario.findByEmail(email);

    if (!usuario) {
      throw new AppError('E-mail não cadastrado.', 401);
    }

    const servRecuperarAluno = new ServRecuperarAluno();
    const aluno = await servRecuperarAluno.execute(usuario.id);

    if (aluno && !usuario.cadastroAprovado) {
      throw new AppError(
        'Caro aluno, seu cadastro ainda não foi aprovado, por favor contate o coordenador do seu curso!',
      );
    }

    const senhaVerificada = compare(senha, usuario.senha ?? '');

    if (!senhaVerificada) {
      throw new AppError('Senha incorreta.', 401);
    }

    const token = sign(
      {
        idUsuario: usuario.id,
        nome: usuario.nome,
      },
      authConfig.jwt.secret,
      {
        subject: usuario.id,
        expiresIn: authConfig.jwt.expiresIn,
      },
    );

    delete usuario.senha;

    return { usuario, token };
  }
}

export default ServCriarSessao;
```

Fonte: Autor do trabalho (2023)

Conforme demonstra a imagem 20, podemos ver uma estrutura de pastas representando os serviços implementados. Cada serviço é responsável por um conjunto específico de funcionalidades e possui seus próprios métodos e lógica de negócio. Essa organização facilita a compreensão e manutenção do código, além de permitir a adição de novos serviços de forma modular.

Figura 20 - Estrutura de pastas do service



Fonte: Autor do trabalho (2023)

Ao implementar os serviços, é importante considerar boas práticas de programação, como a separação de responsabilidades, o tratamento adequado de erros e exceções, e a adoção de padrões de projeto que tornem o código mais legível e escalável.

Dessa forma, a implementação dos serviços é uma etapa crucial no desenvolvimento do back-end, permitindo a execução das regras de negócio e fornecendo funcionalidades específicas para atender às necessidades da aplicação.

5.3.3.1 Regras de negócio relacionadas ao serviço

A implementação dos serviços desempenha um papel fundamental no desenvolvimento do back-end, permitindo a execução de funcionalidades específicas da aplicação. Cada serviço é responsável por lidar com um conjunto de operações relacionadas a uma determinada entidade ou funcionalidade. Ao implementar os serviços, é necessário definir as regras de negócio que serão aplicadas em cada contexto. Essas regras estabelecem as restrições, validações e lógicas específicas

que garantem a integridade dos dados e a correta execução das funcionalidades. Neste contexto, exploraremos alguns serviços importantes do projeto e discutir as regras de negócio associadas a cada um deles, destacando como eles contribuem para o funcionamento adequado e eficaz da aplicação.

5.3.3.1.1 Serviço de cadastro de usuário

O serviço de cadastro de usuário desempenha um papel fundamental na aplicação, permitindo a criação de novos usuários no sistema. Ao utilizar esse serviço, é possível fornecer informações como nome, e-mail, senha e telefone para realizar o cadastro. Antes de efetuar o registro, o serviço verifica se o e-mail já está cadastrado, evitando duplicações. Caso o e-mail já exista, uma exceção é lançada indicando a situação. Além disso, o serviço permite definir se o usuário cadastrado terá perfil de administrador. Nesse caso, é realizada uma verificação para garantir que apenas usuários com perfil administrador possam cadastrar outros administradores. Após a criação do usuário, a senha é criptografada e o cadastro é salvo no banco de dados. O serviço retorna os dados do novo usuário, excluindo a informação sensível da senha. Uma visão geral do serviço de cadastro de usuário pode ser visualizada na imagem 21.

Figura 21 - Serviço para cadastro de usuário

```
class ServCriarUsuario {
  public async execute({
    nome,
    email,
    senha,
    telefone,
    administrador = false,
    idUsuarioLogado,
  }: ICadastrarUsuario): Promise<Usuario | void> {
    const repUsuario = getCustomRepository(RepUsuario);
    const emailExists = await repUsuario.findByEmail(email);

    if (emailExists) {
      throw new AppError('Este e-mail já esta cadastrado.');
    }

    let usuarioLogado;

    if (idUsuarioLogado) {
      usuarioLogado = await repUsuario.findById(idUsuarioLogado);
    }

    if (administrador && !usuarioLogado?.administrador) {
      throw new AppError(
        'Apenas usuario com perfil administrador pode cadastrar outro administrador',
      );
    }

    let novoCadastro = repUsuario.create({
      nome,
      email,
      senha: await hash(senha, 8),
      telefone,
      administrador,
    });

    novoCadastro = await repUsuario.save(novoCadastro);

    delete novoCadastro.senha;

    return novoCadastro;
  }
}

export default ServCriarUsuario;
```

Fonte: Autor do trabalho (2023)

5.3.3.1.2 Serviço para aprovar cadastro de aluno

O serviço de aprovação de cadastro de aluno é responsável por validar e atualizar o status de aprovação do cadastro de um aluno no sistema. Ao utilizar esse serviço, é fornecido o ID do aluno e um parâmetro indicando se o cadastro foi aprovado ou não. O serviço busca o aluno no banco de dados com base no ID fornecido e verifica se o aluno existe. Caso o aluno não seja encontrado, uma exceção é lançada indicando a impossibilidade de encontrar o cadastro do aluno. Em seguida, o status de aprovação do cadastro é atualizado de acordo com o valor fornecido. Após a atualização, o serviço realiza o envio de um e-mail para o aluno, informando-o sobre a aprovação ou não do seu cadastro. O conteúdo do e-mail é definido com base no status de aprovação, e um template correspondente é selecionado. O serviço utiliza a biblioteca *EtherealMail* para enviar o e-mail. Uma visão geral do serviço de aprovação de cadastro de aluno pode ser visualizada na imagem 22.

Figura 22 - Serviço para aprovar o cadastro de aluno

```
class ServAprovarCadastroAluno {
  public async execute({
    id,
    cadastroAprovado = false,
  }: IAprovarCadastroAluno): Promise<string | void> {
    const repUsuario = getCustomRepository(RepUsuario);
    const aluno = await repUsuario.findById(id);

    if (!aluno) {
      throw new AppError(
        `Não possivel encontrar o cadastro do aluno, id: ${id}`,
      );
    }

    aluno.cadastroAprovado = cadastroAprovado;

    await repUsuario.save(aluno);
    await this.enviaEmailParaAluno(aluno);

    return 'Dados salvos com sucesso!';
  }

  private async enviaEmailParaAluno(aluno: Usuario) {
    const { cadastroAprovado, nome, email } = aluno;
    const template = cadastroAprovado
      ? 'cadastro_aluno_aprovado.hbs'
      : 'cadastro_aluno_nao_aprovado.hbs';

    const file = path.resolve(__dirname, '..', '..', 'views', template);

    await EtherealMail.sendMail({
      to: {
        nome,
        email,
      },
      subject: 'Meu TCC - Aprovação de cadastro!',
      template: {
        file,
        variaveis: {
          link: 'http://localhost:3000',
        },
      },
    });
  }
}

export default ServAprovarCadastroAluno;
```

Fonte: Autor do trabalho (2023)

5.3.3.1.3 Serviço para criar o curso

O serviço de criação de curso permite a criação de um novo curso no sistema. Ao utilizar esse serviço, são fornecidos os dados necessários para criar um curso, como o nome, sigla, status de ativação, ID da instituição, ID do coordenador e ID do subcoordenador (opcional). Primeiro, o serviço verifica se já existe um curso cadastrado com o mesmo nome, evitando duplicações. Caso o nome já esteja em uso, uma exceção é lançada indicando que já existe um curso com esse nome. Em seguida, o serviço verifica se o coordenador e, se fornecido, o subcoordenador existem no sistema com base nos IDs fornecidos. Se algum deles não for encontrado, uma exceção é lançada indicando que não foi possível encontrar o coordenador ou subcoordenador.

Após as verificações, o serviço cria o novo curso com os dados fornecidos e o salva no banco de dados. Além disso, o serviço utiliza outro serviço chamado "ServCriarUsuarioCurso" para associar automaticamente o coordenador e, se fornecido, o subcoordenador ao curso. O serviço de criação de curso retorna o curso recém-criado como demonstra a imagem 23.

Figura 23 - Serviço para criar curso

```
class ServCriarCurso {
  public async execute({
    nome,
    sigla,
    ativo,
    idInstituicao,
    idCoordenador,
    idSubcoordenador,
  }: ICriarCurso): Promise<Curso> {
    const repCurso = getCustomRepository(RepCurso);
    const repUsuario = getCustomRepository(RepUsuario);
    const servCriarUsuarioCurso = new ServCriarUsuarioCurso();

    const nomeExists = await repCurso.findByName(nome.trim());

    if (nomeExists) {
      throw new AppError('Já existe um curso cadastrado com este nome!');
    }

    const coordenador = await repUsuario.findById(idCoordenador);

    if (!coordenador) {
      throw new AppError('Não foi possível encontrar o Coordenador!');
    }

    let curso = repCurso.create({
      nome,
      sigla,
      ativo,
      instituicao: { id: idInstituicao },
    });

    curso = await repCurso.save(curso);

    await servCriarUsuarioCurso.execute({
      perfilUsuario: 'coordenador',
      idUsuario: idCoordenador,
      idCurso: curso.id,
    });

    if (idSubcoordenador) {
      const subcoordenador = await repUsuario.findById(idSubcoordenador);

      if (!subcoordenador) {
        throw new AppError('Não foi possível encontrar o Subcoordenador!');
      }

      await servCriarUsuarioCurso.execute({
        perfilUsuario: 'subcoordenador',
        idUsuario: idSubcoordenador,
        idCurso: curso.id,
      });
    }

    return curso;
  }
}

export default ServCriarCurso;
```

Fonte: Autor do trabalho (2023)

5.3.3.1.4 Serviço para criar semestre

O serviço de criação de semestre permite a criação de um novo semestre para um determinado curso. Ao utilizar esse serviço, são fornecidos os dados necessários para criar o semestre, como a data inicial, a data final e o ID do curso. O serviço realiza algumas validações para garantir a consistência dos dados. Primeiro, verifica se a data inicial e a data final são obrigatórias. Caso alguma delas não seja fornecida, uma exceção é lançada indicando que ambas são necessárias.

Em seguida, o serviço verifica se a data final é posterior à data inicial. Caso contrário, uma exceção é lançada indicando que a data final deve ser posterior à data inicial. Além disso, o serviço verifica se a data inicial é anterior ao dia atual. Caso seja, uma exceção é lançada indicando que não é permitido cadastrar um semestre com data inicial anterior ao dia atual.

Após as validações, o serviço consulta o banco de dados para verificar se já existe um semestre vigente para o curso fornecido. Se não houver nenhum semestre vigente, o serviço cria o novo semestre com os dados fornecidos e o salva no banco de dados.

Caso já exista um semestre vigente, o serviço verifica se a data inicial do novo semestre é anterior ou igual à data final do semestre vigente. Se for, uma exceção é lançada indicando que já existe um semestre vigente até a data final do semestre existente conforme demonstra a imagem 24.

Figura 24 - Serviço para criar semestre

```
class ServCriarSemestre {
  public async execute({
    dataInicial,
    dataFinal,
    idCurso,
  }: IRequest): Promise<Semestre | undefined> {
    const repSemestre = getCustomRepository(RepSemestre);
    const _dataInicial = moment(dataInicial, 'DD/MM/YYYY');
    const _dataFinal = moment(dataFinal, 'DD/MM/YYYY');
    const dataAtual = moment(moment(), 'DD/MM/YYYY');
    let novoSemestre;

    if (!_dataInicial || !_dataFinal) {
      throw new AppError('Data inicial e data final são obrigatórias!');
    }

    if (_dataInicial ⩾ _dataFinal) {
      throw new AppError('A data final deve ser posterior a data inicial!');
    }

    if (moment(_dataInicial).isBefore(dataAtual, 'day')) {
      throw new AppError(
        'Não é permitido cadastrar semestre com data inicial anterior ao dia atual',
      );
    }

    const semestreCurso = await repSemestre.findUltimoSemestreByCurso(idCurso);

    if (!semestreCurso) {
      return await cadastraSemestre();
    }

    if (_dataInicial ⩽ moment(semestreCurso.dataFinal, 'DD/MM/YYYY')) {
      throw new AppError(
        `Já existe um semestre vigente até ${moment(
          semestreCurso.dataFinal,
        ).format('DD/MM/YYYY')}`,
      );
    }

    return await cadastraSemestre();

    async function cadastraSemestre() {
      novoSemestre = repSemestre.create({
        dataInicial: _dataInicial,
        dataFinal: _dataFinal,
        curso: { id: idCurso },
      });

      return await repSemestre.save(novoSemestre);
    }
  }
}

export default ServCriarSemestre;
```

Fonte: Autor do trabalho (2023)

5.3.3.1.5 Serviço que cria o relacionamento entre Semestre e Tarefa

O serviço de criação de “SemestreTarefa” permite vincular uma tarefa a um semestre específico. Ele recebe como entrada a data prevista para a conclusão da tarefa, o ID do semestre e o ID da tarefa. O serviço realiza algumas validações para garantir a integridade dos dados.

Primeiro, ele obtém os repositórios necessários para acessar as tabelas envolvidas: “RepSemestreTarefa”, “RepSemestre” e “RepTarefa”. Em seguida, ele verifica se o semestre e a tarefa correspondentes aos IDs fornecidos existem no banco de dados. Caso contrário, uma exceção é lançada indicando que o semestre ou a tarefa não foram encontrados.

Após a verificação, o serviço extrai as datas de início e fim do semestre encontrado e formata a data prevista para a conclusão da tarefa. Em seguida, ele verifica se a data prevista está dentro do período do semestre. Caso a data prevista seja anterior à data de início do semestre ou posterior à data de término do semestre, uma exceção é lançada informando que a data prevista deve estar dentro do período do semestre.

Se todas as validações forem bem-sucedidas, o serviço cria uma nova instância de “SemestreTarefa” e a salva no banco de dados, vinculando o semestre e a tarefa correspondentes aos IDs fornecidos conforme demonstra a imagem 25.

Figura 25 - Serviço que cria vinculo entre Semestre e Tarefa

```
class ServCriarSemestreTarefa {
  public async execute({
    dataPrevista,
    idSemestre,
    idTarefa,
  }: IRequest): Promise<SemestreTarefa | undefined> {
    const repSemestreTarefa = getCustomRepository(RepSemestreTarefa);
    const repSemestre = getCustomRepository(RepSemestre);
    const repTarefa = getCustomRepository(RepTarefa);
    const dataPrevistaFormatada = moment(dataPrevista, 'DD/MM/YYYY');
    const semestre = await repSemestre.findById(idSemestre);
    const tarefa = await repTarefa.findById(idTarefa);

    if (!semestre) {
      throw new AppError('Semestre não encontrado');
    }

    if (!tarefa) {
      throw new AppError('Tarefa não encontrada');
    }

    const dataInicialSemestre = moment(semestre.dataInicial, 'DD/MM/YYYY');
    const dataFinalSemestre = moment(semestre.dataFinal, 'DD/MM/YYYY');

    if (
      !dataPrevistaFormatada.isSameOrAfter(dataInicialSemestre, 'day') ||
      !dataPrevistaFormatada.isSameOrBefore(dataFinalSemestre, 'day')
    ) {
      const _dataInicialSemestre = dataInicialSemestre.format('DD/MM/YYYY');
      const _dataFinalSemestre = dataFinalSemestre.format('DD/MM/YYYY');

      throw new AppError(
        `A data prevista da tarefa deve estar dentro do periodo do semestre
        que inicia em ${_dataInicialSemestre} e finaliza em ${_dataFinalSemestre}`,
      );
    }

    const semestreTarefa = repSemestreTarefa.create({
      dataPrevista,
      semestre: { id: idSemestre },
      tarefa: { id: idTarefa },
    });

    await repSemestreTarefa.save(semestreTarefa);

    return semestreTarefa;
  }
}

export default ServCriarSemestreTarefa;
```

Fonte: Autor do trabalho (2023)

5.3.3.2 Autenticação com JWT

A autenticação é um aspecto fundamental na segurança de uma aplicação. Ela garante que apenas usuários autorizados tenham acesso aos recursos e funcionalidades do sistema. A autenticação com JWT (JSON Web Token) é uma abordagem amplamente utilizada para fornecer autenticação em aplicações web.

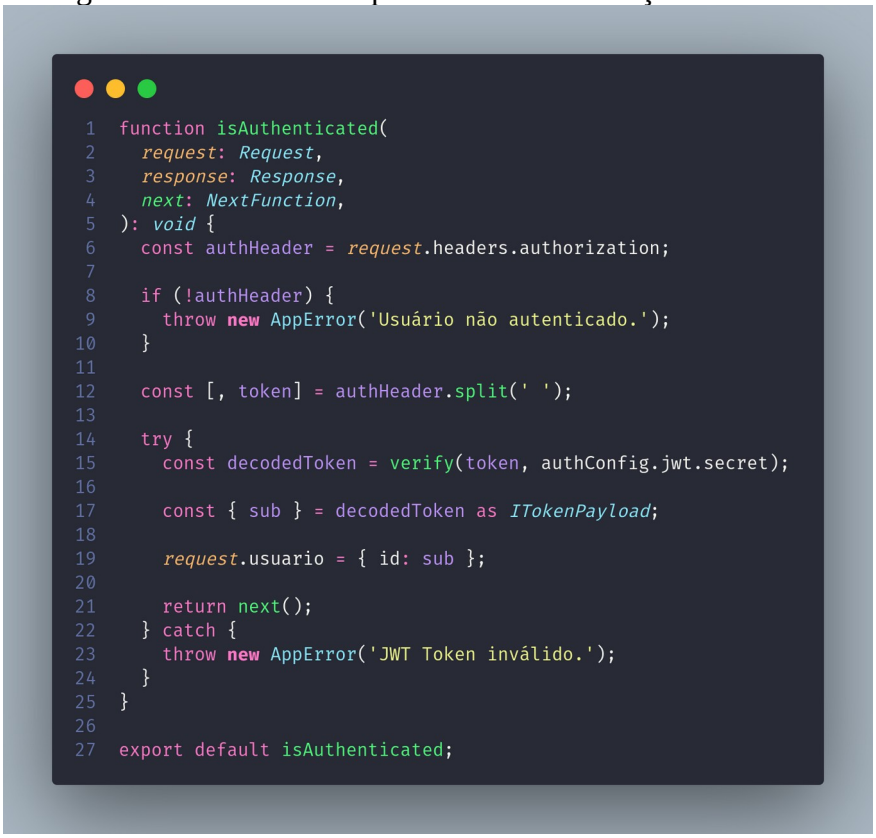
JWT ou JSON Web Token é um padrão da indústria definido pela *RFC7519* que tem como objetivo transmitir ou armazenar de forma compacta e segura objetos JSON entre diferentes aplicações. O JWT é digitalmente assinado usando uma chave secreta com o algoritmo HMAC ou um par de chaves pública e privada RSA ou ECDSA (LIMA, 2021). Ele é composto por três partes: o cabeçalho, o payload e a assinatura. O cabeçalho contém informações sobre o algoritmo de criptografia usado para assinar o token. O payload contém os dados do usuário, como o ID ou o nome de usuário. E a assinatura é uma sequência de caracteres que é usada para verificar a integridade do token.

Ao utilizar a autenticação com JWT, quando um usuário realiza o login com sucesso, o servidor gera um token JWT e o envia de volta para o cliente. Esse token é então armazenado pelo cliente (geralmente em cookies ou no armazenamento local) e enviado em cada requisição subsequente como parte do cabeçalho de autorização.

No servidor, o middleware de autenticação é responsável por verificar e decodificar o token recebido do cliente. Ele verifica a assinatura do token para garantir que ele não tenha sido alterado e extrai as informações do usuário contidas no payload. Essas informações são então disponibilizadas para as rotas subsequentes, permitindo que o servidor verifique se o usuário tem permissão para acessar determinados recursos.

Como demonstra a imagem 26, a função isAuthenticated é um middleware de autenticação que verifica a presença e a validade do token JWT no cabeçalho de autorização da requisição. Se o token estiver ausente ou inválido, um erro é lançado, impedindo o acesso à rota protegida. Caso o token seja válido, as informações do usuário são adicionadas ao objeto request para uso posterior pelas rotas.

Figura 26 - Middleware que valida a autenticação do usuário



```
function isAuthenticated(
  request: Request,
  response: Response,
  next: NextFunction,
): void {
  const authHeader = request.headers.authorization;

  if (!authHeader) {
    throw new AppError('Usuário não autenticado.');
  }

  const [, token] = authHeader.split(' ');

  try {
    const decodedToken = verify(token, authConfig.jwt.secret);

    const { sub } = decodedToken as ITokenPayload;

    request.usuario = { id: sub };

    return next();
  } catch {
    throw new AppError('JWT Token inválido.');
  }
}

export default isAuthenticated;
```

Fonte: Autor do trabalho (2023)

Ao utilizar o middleware isAuthenticated nas rotas (conforme exemplifica a imagem), é garantindo que apenas usuários autenticados e com um token válido possam acessar essas rotas específicas. Isso ajuda a proteger recursos sensíveis da aplicação e controlar o acesso aos mesmos.

Figura 27 - Exemplo de uso do middleware - isAuthenticated

```
const defaultData = {
  nome: Joi.string().required(),
  sigla: Joi.string(),
  ativo: Joi.boolean(),
};

instituicaoRouter.post(
  '/',
  isAuthenticated,
  celebrate({ [Segments.BODY]: defaultData }),
  instituicaoController.criarInstituicao,
);
```

Fonte: Autor do trabalho (2023)

A utilização do JWT na autenticação traz diversas vantagens, como a escalabilidade da aplicação, já que o servidor não precisa armazenar informações sobre os usuários autenticados, e a possibilidade de trocar informações seguras entre diferentes serviços. No entanto, é importante garantir a correta implementação e configuração do JWT, incluindo o uso de chaves secretas fortes e a definição adequada do tempo de expiração dos tokens, a fim de manter a segurança da aplicação.

5.3.3.3 Serviço de envio de e-mail

O serviço de envio de e-mail é uma parte essencial de muitas aplicações, pois permite a comunicação com os usuários através de mensagens eletrônicas. É comum utilizar serviços de e-mail externos, como o Gmail, para enviar e-mails a partir de uma aplicação.

Conforme demonstra a imagem 28, há uma classe *EtherealMail* que contém um método estático *sendMail* responsável por enviar o e-mail. O método recebe os parâmetros necessários para o envio, como o destinatário, remetente, assunto e o template do e-mail.

Figura 28 - Serviço que realiza o envio de e-mail

```
export default class EtherealMail {
  static async sendMail({
    to,
    from,
    subject,
    template,
  }: ISendMail): Promise<void> {
    const mailTemplate = new HandlebarsMailTemplate();
    const transporter = nodemailer.createTransport({
      host: 'smtp.gmail.com',
      port: 587,
      auth: {
        user: process.env.USER_EMAIL,
        pass: process.env.USER_EMAIL_PASSWORD,
      },
    });

    await transporter.sendMail({
      from: {
        name: from?.nome || 'Meu TCC',
        address: from?.email || 'meutcc@dev.com.br',
      },
      to: {
        name: to?.nome,
        address: to?.email,
      },
      subject,
      html: await mailTemplate.parser(template),
    });
  }
}
```

Fonte: Autor do trabalho (2023)

Para o envio do e-mail, é utilizado o pacote *nodemailer*, que fornece uma API para envio de e-mails no Node.js. O código configura um transportador (transporter) que define as informações do servidor SMTP a ser utilizado. No exemplo, o servidor SMTP do Gmail é utilizado, com as credenciais de e-mail e senha definidas nas variáveis de ambiente USER_EMAIL e USER_EMAIL_PASSWORD, respectivamente.

O método sendMail utiliza o transportador configurado para enviar o e-mail. Ele define o remetente e destinatário do e-mail, o assunto e o conteúdo HTML do e-mail. O conteúdo HTML é gerado a partir de um template utilizando a classe *HandlebarsMailTemplate*. O template é lido de um arquivo e, em seguida, é compilado e renderizado utilizando a biblioteca handlebars. As variáveis do template são passadas como parâmetro para a função *parseTemplate*, que retorna o conteúdo HTML final do e-mail.

A utilização desse serviço de envio de e-mail permite que a aplicação envie notificações, confirmações, lembretes ou qualquer outro tipo de comunicação por e-mail aos usuários. Essa comunicação é fundamental para manter os usuários informados sobre atividades, atualizações ou eventos relacionados à aplicação.

5.4 DOCUMENTAÇÃO DA API

A documentação da API desempenha um papel crucial no desenvolvimento, manutenção e uso de uma aplicação. Podem ser listados alguns pontos importantes sobre a importância da documentação e o uso do Swagger:

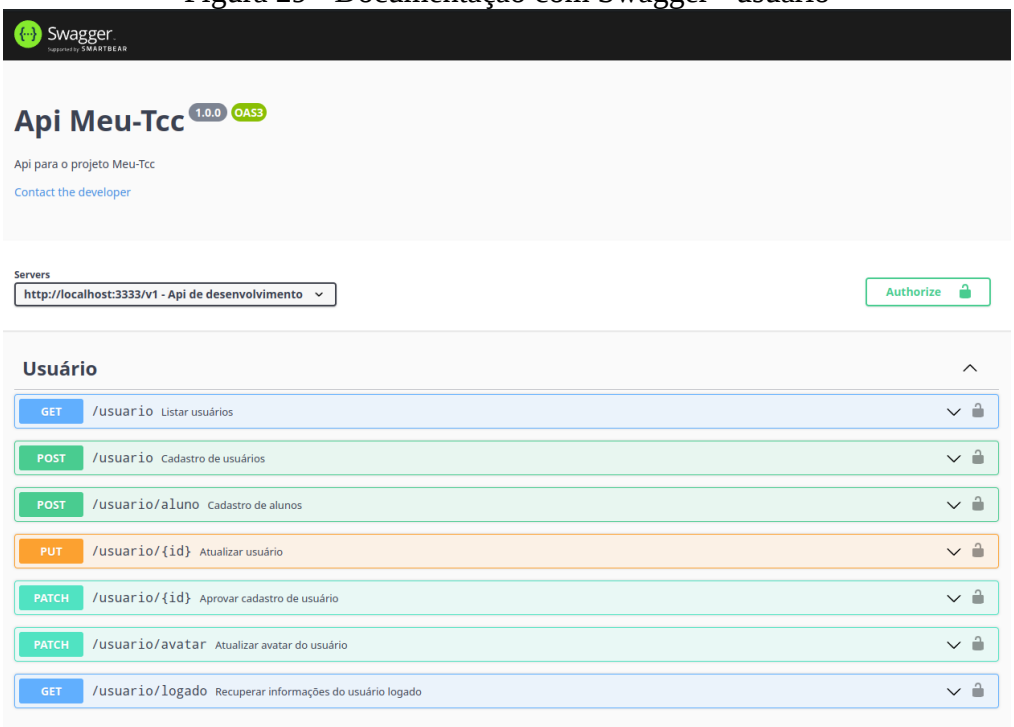
a) Comunicação eficaz: A documentação fornece uma forma clara e concisa de comunicar as funcionalidades e recursos da API para a equipe de desenvolvimento, clientes e usuários. Isso ajuda a evitar mal-entendidos e garante que todos estejam na mesma página em relação ao uso correto da API.
b) Facilita o desenvolvimento: Uma documentação bem estruturada permite que os desenvolvedores entendam rapidamente como usar os endpoints, quais parâmetros devem ser enviados, quais respostas esperar e como lidar com possíveis erros. Isso acelera o processo de desenvolvimento, economizando tempo e esforço.
c) Onboarding de desenvolvedores: Quando novos desenvolvedores ingressam em um projeto, a documentação da API os ajuda a se familiarizar com a estrutura e os recursos disponíveis. Eles podem entender facilmente como interagir com a API e começar a contribuir rapidamente.

d) Melhora a experiência do cliente: Para clientes e usuários que desejam integrar suas aplicações com a sua API, uma documentação completa e precisa é fundamental. Ela fornece todas as informações necessárias para que eles possam aproveitar ao máximo os recursos oferecidos pela sua aplicação.

e) Testes e depuração: A documentação também pode incluir exemplos de solicitações e respostas, permitindo que os desenvolvedores testem e depurem suas implementações antes de usar a API em produção. Isso ajuda a identificar e corrigir problemas com antecedência.

Swagger é um kit de ferramentas de código aberto construído em torno da especificação OpenAPI que nos ajuda a projetar, construir, documentar e consumir APIs REST (BATISTA, 2022). O Swagger permite criar uma documentação interativa, fácil de navegar e explorar. Ele gera automaticamente uma interface de usuário onde os desenvolvedores podem testar os endpoints, visualizar os modelos de dados, verificar os parâmetros e até mesmo realizar autenticação, tudo dentro do próprio ambiente de documentação. O Swagger também oferece recursos para exportar a documentação em diferentes formatos, como HTML, PDF ou Markdown.

Figura 29 - Documentação com Swagger - usuário



Fonte: Autor do trabalho (2023)

O uso do Swagger simplifica a criação e a manutenção da documentação da API (conforme demonstra a imagem), tornando-a atualizada e precisa. Além disso, fornece uma experiência interativa para os desenvolvedores, melhorando a compreensão e o uso correto dos recursos disponíveis.

6 RESULTADOS

Esta seção apresenta os resultados obtidos no desenvolvimento do projeto MeuTCC, destacando as conquistas e avanços alcançados ao longo do processo. Por meio da aplicação de tecnologias avançadas, adoção de práticas de qualidade de código e implementação de recursos de segurança, o projeto MeuTCC obteve resultados significativos que contribuem tanto para o presente trabalho quanto para o futuro do projeto. Serão abordados tópicos como a padronização do código, o histórico do banco de dados, a implementação de rotas seguras e o desenvolvimento do CRUD das tabelas. Além disso, serão apresentadas sugestões de melhorias e possíveis direcionamentos futuros para a evolução contínua do projeto MeuTCC.

6.1 IMPLEMENTAÇÃO DE TECNOLOGIAS AVANÇADAS

Durante o desenvolvimento do projeto MeuTCC, foram aplicadas tecnologias avançadas que proporcionaram benefícios significativos em termos de qualidade de código, escalabilidade e segurança.

6.2 FERRAMENTAS DE PADRONIZAÇÃO DE CÓDIGO

Por meio da integração de ferramentas como ESLint e Prettier, foi estabelecido um padrão de código consistente e aderente às melhores práticas de mercado. Isso facilitou o entendimento e a manutenção do código-fonte, permitindo que desenvolvedores futuros possam trabalhar de forma mais eficiente e produtiva.

6.3 HISTÓRICO DE BANCO DE DADOS

Através da configuração do TypeORM, foi possível manter um histórico detalhado da estrutura das tabelas do banco de dados. Esse recurso possibilitou o acesso ao modelo das tabelas, bem como rastrear alterações realizadas, garantindo maior transparência e facilidade na evolução do banco de dados.

6.4 SEGURANÇA DAS ROTAS E ACESSO CONTROLADO

Com a implementação de recursos avançados de segurança, apenas usuários autorizados podem interagir com o sistema MeuTCC. Isso garante a proteção dos dados e funcionalidades, permitindo que apenas usuários autenticados possam criar, editar ou visualizar informações sensíveis.

6.5 IMPLEMENTAÇÃO DO CRUD E OPERAÇÕES NO BANCO DE DADOS

Foi desenvolvida a estrutura completa para realizar operações de criação, leitura, atualização e exclusão (CRUD) nas tabelas do banco de dados. As rotas foram devidamente configuradas, permitindo a manipulação eficiente dos registros e fornecendo uma base sólida para requisitos futuros mais complexos.

7 CONSIDERAÇÕES

Durante o desenvolvimento do projeto MeuTCC, foram vivenciadas diversas experiências e ocorreram aprendizados significativos que contribuíram para o crescimento e aprimoramento da equipe envolvida. Abordaremos alguns dos principais pontos observados ao longo do processo.

7.1 EXPERIÊNCIA E APRENDIZADOS

Uma das principais experiências foi a imersão nas tecnologias e ferramentas utilizadas no projeto. Onde surgiu a oportunidade de explorar e trabalhar com frameworks, bibliotecas e padrões de desenvolvimento que eram novos para alguns membros. Essa vivência proporcionou um ambiente de aprendizado constante, no qual foi possível adquirir conhecimentos práticos e aprofundar-se em conceitos como arquitetura de software, boas práticas de codificação e integração de diferentes componentes do sistema.

Durante o desenvolvimento do projeto MeuTCC, adquiri uma variedade de aprendizados significativos. A exploração de novas tecnologias e ferramentas me permitiu expandir meus conhecimentos em arquitetura de software, boas práticas de codificação e integração de componentes. Lidar com a complexidade do projeto exigiu habilidades analíticas e de resolução de problemas, além de aprimorar minhas capacidades de documentação. A valorização da escalabilidade e manutenibilidade do sistema também foi um aprendizado importante. Essa experiência me preparou para enfrentar futuros desafios com mais confiança e conhecimento.

7.2 AVALIAÇÃO DO RESULTADO FINAL

A avaliação do resultado final do projeto foi realizada com base em critérios objetivos e testes realizados internamente. Devido ao estágio atual do projeto, ainda não foi possível obter feedbacks diretos dos usuários, uma vez que a API desenvolvida ainda não foi implantada em um ambiente de produção e disponibilizada para uso público.

No entanto, foram realizados testes extensivos para verificar a funcionalidade, desempenho e usabilidade da API. Foram criados casos de teste abrangentes que abordaram diferentes cenários de uso, garantindo a correta execução das operações e a consistência dos dados.

Além disso, foram aplicadas boas práticas de desenvolvimento, como o uso de padrões de projeto, a adoção de ferramentas de análise de código para garantir a qualidade do código

produzido. Essas práticas contribuíram para um código mais legível, modular e de fácil manutenção.

Embora a ausência de feedbacks diretos dos usuários seja uma limitação deste estágio do projeto, acredita-se que a API desenvolvida atende aos requisitos funcionais e possui uma base sólida para futuras implementações e iterações. Sugere-se, no entanto, a realização de testes com usuários reais assim que a API for implantada em um ambiente de produção, a fim de obter feedbacks mais abrangentes e validar sua eficácia na prática.

Em resumo, a avaliação do resultado final do projeto foi baseada em testes internos abrangentes, que evidenciaram a funcionalidade e o desempenho satisfatório da API. Apesar da ausência de feedbacks diretos dos usuários, acredita-se que a solução desenvolvida está preparada para futuras melhorias e validações por meio de interações com usuários reais.

7.3 RECOMENDAÇÕES E SUGESTÕES DE MELHORIAS

Com base na experiência adquirida durante o desenvolvimento do projeto, bem como considerando a integração com o ambiente acadêmico da faculdade, surgem duas recomendações principais para futuras melhorias.

A primeira recomendação é realizar o deploy da API em um ambiente de produção, tornando-a acessível para uso real pelos usuários. A implantação da API em um servidor público permitirá que alunos, professores e demais membros da comunidade acadêmica tenham acesso às funcionalidades disponibilizadas pelo projeto. Isso proporcionará a oportunidade de coletar feedbacks e avaliar o desempenho da API em um contexto real, possibilitando ajustes e aprimoramentos adicionais conforme necessário.

A segunda sugestão de melhoria é a integração da autenticação com o idUFSC que por sua vez necessita de autorização da Superintendência de Governança Eletrônica e Tecnologia da Informação e Comunicação (SETIC). Ao estabelecer uma integração direta entre a API e o idUFSC, os usuários poderão utilizar suas credenciais para acessar o software MeuTCC e desfrutar das funcionalidades oferecidas pelo projeto. Isso simplificará o processo de autenticação e permitirá uma experiência mais fluida e familiar para os usuários, evitando a necessidade de criar contas ou fazer login separadamente na API.

Além dessas duas recomendações específicas, é importante destacar a importância contínua da coleta de feedbacks dos usuários reais e da realização de testes de usabilidade. Essas atividades podem fornecer insights valiosos sobre possíveis áreas de melhoria, identificar novas necessidades dos usuários e direcionar o desenvolvimento futuro do projeto.

Em suma, as recomendações para melhorias futuras incluem o deploy da API em um ambiente de produção e a integração com o Moodle para autenticação dos usuários. Essas melhorias proporcionarão uma experiência mais completa e integrada, atendendo às necessidades da comunidade acadêmica e aprimorando a utilidade e a usabilidade do projeto.

8 CONCLUSÃO

O projeto MeuTCC foi uma jornada significativa que envolveu a criação de uma API para gerenciamento de projetos de TCC, voltada para a comunidade acadêmica. Ao longo desse trabalho, foram exploradas tecnologias modernas e metodologias de desenvolvimento para alcançar os objetivos propostos. Neste contexto, este projeto proporcionou a oportunidade de aplicar os conhecimentos adquiridos ao longo da formação acadêmica, bem como de adquirir novas habilidades técnicas e desenvolver competências profissionais.

Durante o desenvolvimento, foram implementadas funcionalidades essenciais para o gerenciamento de projetos de TCC, como o cadastro de usuários, registro de projetos, atribuição de orientadores, acompanhamento do progresso e avaliação. A utilização de tecnologias como Node.js, Express e typeORM permitiu a criação de uma API robusta e flexível, capaz de atender às necessidades dos usuários e proporcionar uma experiência intuitiva.

Ao longo do processo, foram enfrentados desafios e obstáculos que exigiram pesquisa, análise e tomada de decisões. A abordagem adotada para garantir a qualidade do código, a padronização, bem como a preocupação com a segurança e a privacidade dos dados, foram aspectos essenciais considerados no desenvolvimento da aplicação.

Apesar de ser um projeto desenvolvido individualmente, foi possível adquirir conhecimentos sobre o trabalho em equipe, uma vez que foram realizadas pesquisas, consultas e interações com professores e outros colegas para obter feedbacks valiosos e orientações durante o processo de desenvolvimento.

No entanto, é importante ressaltar que o trabalho apresentado aqui é uma versão inicial e que ainda existem oportunidades para melhorias e expansões futuras. Recomenda-se a implementação de recursos como o deploy da API em um ambiente de produção, a integração com o Moodle para autenticação dos usuários e a coleta de feedbacks reais para aprimorar ainda mais a aplicação.

Em suma, o projeto MeuTCC representa a aplicação prática dos conhecimentos adquiridos ao longo da graduação e proporcionou uma experiência enriquecedora de desenvolvimento. Através da implementação de funcionalidades relevantes, a adoção de boas práticas de programação e a consideração dos feedbacks recebidos, espera-se que este projeto contribua para a melhoria do gerenciamento de projetos de TCC na comunidade acadêmica, facilitando o processo e proporcionando uma experiência mais eficiente e eficaz para todos os envolvidos.

REFERÊNCIAS

GRAY, Clifford F.; LARSON, Erik W.. **Gerenciamento de projetos: o processo gerencial.** 4. ed. São Paulo: Mc Graw Hill, 2010. 589 p. Disponível em: https://edisciplinas.usp.br/pluginfile.php/5687071/mod_resource/content/1/Gerenciamento%20de%20Projetos%20(Gray%20-%20Larson)%20-%20McGrawHill.pdf. Acesso em: 05 maio 2023.

CARVALHO, Rosângela Saraiva. **Sistemas de gestão da aprendizagem e sistemas de gestão acadêmica: avaliados pela ótica do docente.** 2010. 175 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2010. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/2450/1/arquivo3458_1.pdf. Acesso em: 08 maio 2023.

TORREÃO, Paula Geralda Barbosa Coelho. **Gerenciamento de projetos.** [entre 2005 e 2010]. 23 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Pernambuco, [entre 2005 e 2010]. Cap. 3. Disponível em: https://www.cin.ufpe.br/~if717/leituras/artigo-gerenciamento-de-projetos-paula-coelho.pdf. Acesso em: 08 maio 2023.

ANDRADE, A. **O que é o Express.js?** Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-express-js>. Acesso em: 6 jun. 2023.

BATISTA, S. S. Desarrollo de un sistema de gestión de API´s en la plataforma Moonshot. p. 67, 2022.

BESSA, A. **O que é Node.JS? Como funciona e um Guia para iniciar**. Disponível em: <https://www.alura.com.br/artigos/node-js>.

FADEL, A. C.; SILVEIRA, H. DA M. Metodologias ágeis no contexto de desenvolvimento de software: XP, Scrum e Lean. 2010.

FREITAS, A. et al. **Metodologia de Desenvolvimento do Software Sispaf**. , 2006. Disponível em: <https://www.infoteca.cnptia.embrapa.br/bitstream/doc/50402/1/Doc147.pdf>. Acesso em: 5 maio. 2023

KRIGER, B. **Typeorm o que é: conceito, para que serve e como instalar?** Disponível em: <https://kenzie.com.br/blog/typeorm/>. Acesso em: 5 jun. 2023.

LIMA, C. **O que é JWT?** Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-jwt>.

MARQUES, S. **As 5 boas práticas de desenvolvimento de software**. **Blog UDS**, 9 nov. 2022. Disponível em: <https://uds.com.br/blog/boas-praticas-de-desenvolvimento-de-software/>

MENDES, M. **Proposta de Nova Implementação do Sistema Online de Distribuição de Disciplinas**. , 2022. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/34324/1/PropostaNovaImplementa%c3%a7%c3%a3o.pdf>. Acesso em: 5 jun. 2023

POSTGRESQL. **PostgreSQL: Sobre**. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 29 jun. 2023.

ROCHA, R.; MAGALHÃES, T. **ENGENHARIA DE REQUISITOS**. , 2019.

STORK, V. **Como usar variáveis de ambiente do Node com um arquivo DotEnv para Node.js e npm**. Disponível em: <https://www.freecodecamp.org/portuguese/news/como-usar-variaveis-de-ambiente-do-node-com-um-arquivo-dotenv-para-node-js-e-npm/>. Acesso em: 30 maio. 2023.

WILLIANS, W. **Docker vs Docker Compose - Full Cycle**. Disponível em: <https://fullcycle.-com.br/docker-vs-docker-compose/>.