INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS DE CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PEDRO PEREIRA DE MORAIS JÚNIOR

GitGame: GAME PARA FACILITAR APRENDIZAGEM DE GIT

CAJAZEIRAS
2023

Pedro Pereira de Morais Júnior

GitGame: GAME PARA FACILITAR APRENDIZAGEM DE GIT

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campus Cajazeiras, como requisito parcial para a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Fabio Gomes de Andrade

Cajazeiras
2023

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

M827g Morais Júnior, Pedro Pereira de.
GitGame : game para facilitar aprendizagem de Git / Pedro Pereira de Morais Júnior. – 2024.

43f. : il.

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2024.

Orientador(a): Prof. Dr. Fabio Gomes de Andrade.

1. Desenvolvimento de sistemas. 2. Gamificação. 3. Jogo educativo. 4. Ensino de Git. I. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba. II. Título.

IFPB/CZ CDU: 004.4(043.2)

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

PEDRO PEREIRA DE MORAIS JÚNIOR

GITGAME: GAME PARA FACILITAR APRENDIZAGEM DE GIT

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Dr. Fabio Gomes de Andrade

Aprovada em: **16 de Outubro de 2024.**

Prof. Dr. Fabio Gomes de Andrade - Orientador

Prof. Me. Michel da Silva - Avaliador

IFPB - Campus Cajazeiras

Prof. Dr. Hudson Geovane de Medeiros

IFPB - Campus Cajazeiras

Documento assinado eletronicamente por:

- **Fabio Gomes de Andrade**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 17/10/2024 19:58:10.
- **Hudson Geovane de Medeiros**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 17/10/2024 22:36:56.
- **Michel da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/10/2024 18:44:37.

Este documento foi emitido pelo SUAP em 16/10/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 620974
Código de Autenticação: aa1277eedf

AGRADECIMENTOS ESPECIAIS

Agradeço primeiramente a Deus, por me dar força e perseverança ao longo de todo o curso.

Agradeço também a minha família, que sempre me apoiaram e acreditaram em mim em todos os momentos.

Meus sinceros agradecimentos ao meu orientador, por sua paciência, orientação e valiosas contribuições ao desenvolvimento deste trabalho.

Também sou grato aos meus colegas de curso, pelas discussões, apoio e companheirismo ao longo dessa jornada acadêmica.

Agradeço também ao meu amigo Lyzzandro Dualano, por sua inestimável colaboração no desenvolvimento das páginas do frontend, proporcionando um design visualmente atraente e funcional através do uso da plataforma Figma. Sua expertise e criatividade foram essenciais para a qualidade final deste trabalho.

Por fim, agradeço a todos os professores, amigos e colaboradores que, de alguma forma, contribuíram para a realização deste projeto.

RESUMO

No dia a dia do trabalho de um desenvolvedor de software, o Git é uma ferramenta muito popular para o controle de versões do código. Por esse motivo, existem vários conteúdos para a aprendizagem deste gerenciador de versões, como cursos e tutoriais online. No entanto, muitos usuários podem ter dificuldade em aprender essa tecnologia usando as soluções existentes. Para resolver essa limitação, este trabalho propõe um jogo focado na aprendizagem do *Git*, que representa uma nova abordagem para o estudo e treino desse software. As metodologias aplicadas no desenvolvimento deste trabalho são baseadas em uma na abordagem teórica, prática e interativa.

Palavras-chave: git, jogo, software, tecnologias, aprendizagem, gamificação.

ABSTRACT

In the daily work of a software developer, Git is a very popular tool for version control. For this reason, there are many resources available for learning this version control system, such as online courses and tutorials. However, many users may struggle to learn this technology using existing solutions. To address this limitation, this paper proposes a game focused on learning Git, offering a new approach to studying and practicing this software. The methodologies applied in the development of this work are based on theoretical, practical, and interactive approaches.

Keywords: git, game, software, technologies, learning, gamification.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CLI	Command-Line Interface
CSS	Cascading Style Sheet
GIT	Global Information Tracker
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MIT	Massachusetts Institute of Technology
ORM	Object Relational Mapper
REST	Representational State Transfer
SGBD	Sistema de Gerenciamento de Banco de Dados
TCC	Trabalho de Conclusão de Curso
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier

SUMÁRIO

1. INTRODUÇÃO

Com o avanço do desenvolvimento de sistemas, os softwares foram se tornando cada vez mais complexos, fazendo assim que, com o passar dos anos, fosse necessário a criação de um sistema para facilitar o trabalho com versões de projetos. Sendo assim, em 2005, o famoso criador do sistema operacional *Linux*[1], Linus Torvalds, criou o *Git*, um projeto de código aberto que foi criado para suprir essa demanda, se tornando um dos gerenciadores de versão mais utilizados por desenvolvedores.

O *Git* é uma ferramenta poderosa e essencial no desenvolvimento de software, que permite aos desenvolvedores gerenciarem facilmente diferentes versões de seus projetos. Com o surgimento dessa ferramenta, o processo de controle de versões se tornou muito mais ágil, permitindo a colaboração de diversas pessoas em um mesmo projeto sem gerar problemas na integração das mudanças.

Atualmente, há diversas opções de aulas, artigos, tutoriais interativos e *websites* dedicados ao ensino do *Git*. Essas alternativas tornam possível que qualquer pessoa aprenda e se aprofunde nessa tecnologia. Entretanto, o esforço necessário para o aprendizado por meio dessas ferramentas pode tornar o processo de aprendizagem mais cansativo e desinteressante para o usuário. Para superar essa limitação, este Trabalho de Conclusão de Curso (TCC) tem como objetivo desenvolver uma ferramenta para permitir que as pessoas aprendam a usar o *Git* a partir de um jogo interativo, no qual o jogador aprende mais sobre a ferramenta e coloca em prática o aprendizado adquirido, podendo ter a sensação de evolução ao mesmo tempo em que aprende.

A proposta do jogo é proporcionar uma experiência de aprendizado divertida e envolvente, na qual os jogadores aprendam mais sobre as ferramentas e tenham a oportunidade de alinhar a teoria e a prática. O objetivo do trabalho proposto neste TCC é focar no ensino e aprimoramento das habilidades do *Git*. Assim, espera-se que este novo método de aprendizagem seja mais acessível e eficaz para alguns indivíduos que preferem aprender de forma lúdica e interativa.

---

[1] Sistema Operacional de código aberto

1.1 Motivação

Atualmente, existem diferentes formas de se aprender de maneira interativa uma nova tecnologia. Dentre elas, podemos citar a instalação da própria ferramenta em uma máquina local, esse método cria a oportunidade de realizar todo o treinamento no próprio computador, tendo total acesso e configurando como achar melhor. Entretanto, essa abordagem pode ser complicada para iniciantes. Assim, outros métodos podem ser usados, como jogos e tutoriais interativos.

Durante a concepção deste projeto, foi observada uma certa escassez de jogos para o ensino do Git, principalmente em nosso idioma nativo. Em meio às pesquisas realizadas, foi encontrado apenas um jogo. Diante dessa carência, cresceu a motivação de se criar mais uma aplicação usando conceitos de gamificação para facilitar a aprendizagem desta tecnologia, para que assim possa se ter mais uma opção de jogo.

O jogo desenvolvido neste projeto pode ser adotado por professores em sala de aula para aumentar a eficácia do ensino, tornando o aprendizado mais dinâmico, interativo e envolvente. Outro tipo de usuário que pode se beneficiar desse jogo são os estudantes que desejam complementar seus estudos fora da sala de aula, aproveitando para ver e rever conceitos e práticas da utilização da ferramenta.

Diante da ideia apresentada, a inovação que o *GitGame* traz com relação às ferramentas existentes com o mesmo propósito é a possibilidade da criação de uma conta por parte do usuário, o que permite que o jogador acesse seu progresso a partir de outro dispositivo a qualquer momento.

1.2 Objetivos

Nessa seção são apresentados os objetivos gerais e específicos definidos para este Trabalho de Conclusão de Curso (TCC). Os objetivos aqui descritos foram pensados para o ambiente de desenvolvimento de jogos, visando uma maior relação entre a aplicação e o propósito de ensinar.

1.2.1 **Objetivo Geral**

O objetivo geral deste TCC é desenvolver uma ferramenta que ofereça aos usuários uma maneira interativa e agradável de aprender e praticar comandos do *Git*.

1.2.2 **Objetivos Específicos**

O trabalho também possui os seguintes objetivos específicos:

- fornecer ao usuário a possibilidade de aprender e praticar o manuseio da ferramenta, acompanhado de uma sensação de evolução;
- fornecer ao usuário um ambiente que permita o treinamento de comandos com o acompanhamento visual da evolução do projeto;
- melhorar o processo de aprendizagem do *Git*, facilitando o entendimento e manuseio da ferramenta.

1.3 Trabalhos Relacionados

Alguns trabalhos já foram desenvolvidos para facilitar o ensino do *Git*. Dentre as soluções existentes, a ferramenta de maior destaque é o único jogo de *Git* encontrado, que é o *Oh My Git*[2]. A jogabilidade desse jogo consiste em utilizar *cards* dados pelo programa para realizar comandos ou a escrita direta de comandos no terminal. Assim, das duas formas é possível interagir com o projeto, e qualquer modificação realizada com a utilização do *Git* pode ser visualizada pela árvore do *Git*, que fica na parte esquerda da janela e possibilita a visualização de como está atualmente o Git no projeto.

O jogo está dividido em diferentes níveis, e cada nível possui uma história pequena e uma missão a ser realizada, com objetivos específicos para serem resolvidos. Dessa forma, deve-se realizar os comandos necessários para cumprir a missão atual. O jogo também tem o modo *SandBox* (um ambiente seguro e isolado usado para testar um programa ou sistema) que possibilita simular um projeto como se estivesse utilizando o Git apenas na máquina ou de forma local e com um repositório remoto.

O *Oh My Git* é sem dúvidas um ótimo jogo para se treinar e fixar mais o aprendizado sobre *Git.* Entretanto, ele não possui uma versão em português, o que pode vir a ser um problema para os usuários que venham a se sentir intimidados com o fato de que o jogo é todo em inglês.

Além do *Oh My Git,* existem outras aplicações a serem citadas. A primeira delas se trata do *Learn Git Branch*[3], que é um tutorial interativo, dividido em fases, no qual em cada fase se aprende uma funcionalidade de forma teórica e logo depois

[2] https://ohmygit.org/
[3] https://learngitbranching.js.org/

se aplica de forma prática o que foi aprendido, a partir de uma pequena lição, que pode ser resolvida a partir da utilização do comando aprendido. Além do modo de fases focado em passar de níveis, também existe o modo *SandBox*, que permite a simulação de comandos como se estivesse em um projeto real com ou sem a utilização de repositório remoto. O *Learn Git Branch* é uma boa forma de se aprender comandos básicos, mas possui algumas limitações a serem abordadas. Uma delas é o fato de não ser possível passar métodos para alguns comandos, como na utilização do Git normalmente acontece. Além disso, ele possui uma quantidade limitada de comandos e no terminal são demonstrados comandos e informações do terminal do próprio site, o que pode levar usuários iniciantes a acreditarem que se trata de comandos usados no *Git*.

Atualmente, percebe-se que existem poucas ferramentas que possibilitam o treinamento prático de *Git* diretamente na plataforma, mas existem *websites* e repositórios destinados a ensinar git de forma teórica. Alguns exemplos são: *Git4Noobs*[4], repositório mantido pelo *4noobs* para ensinar *Git*, *Think like a Git*[5], *website* focado em ensinar conceitos intermediários da ferramenta, *git How To*[6], um *website* guia para iniciar em *Git*, além de vários outros que fazem um trabalho semelhante. Esses *websites* são locais muito bons para se aprender sobre o *Git*, mas ainda assim não possuem uma forma interativa dentro da plataforma de treinar o *Git.*

1.4 Metodologia

Para o desenvolvimento bem sucedido de um projeto, é preciso esclarecer os processos e modos de execução planejados para se conseguir alcançar o objetivo final. Para isso, é importante ter uma visão clara não só do propósito do projeto, mas também das técnicas e abordagens que serão necessárias para atingi-lo. Assim, para a implementação deste TCC, foi estabelecido o desenvolvimento das seguintes atividades:

- **Análise do estado da arte (A_1)**: durante o decorrer da atividade, foi conduzida uma revisão minuciosa de estudos, materiais e projetos anteriores que abordam temas semelhantes aos tratados neste

[4] https://github.com/DanielHe4rt/git4noobs
[5] https://think-like-a-git.net/
[6] https://githowto.com/pt-BR

trabalho, isto é, aplicativos dedicados ao ensino de *Git*. O uso deste referencial ao longo do processo de desenvolvimento auxiliou na manutenção do foco na ideia principal e em garantir a originalidade do projeto;

- **Elaboração do documento (A_2)**: esta atividade está relacionada à redação do documento de conclusão de curso;
- **Determinar requisitos (A_3):** durante essa atividade, foram identificados e avaliados os requisitos fundamentais para alcançar o objetivo geral e os objetivos específicos, bem como determinar as funcionalidades e ferramentas que o sistema deveria oferecer aos seus usuários;
- **Definir a arquitetura do projeto (A_4):** nesta atividade foi estabelecida a arquitetura do projeto, os seus módulos e a lógica de comunicação entre os mesmos;
- **Projetar a base de dados (A_5)**: durante esta atividade, a estrutura e o esquema dos dados que seriam utilizados para a implementação do banco de dados da ferramenta foram definidos;
- **Design de interface do usuário (A_6):** esta atividade foi centrada em criar o projeto de interfaces com o usuário para o jogo educativo. A interface do usuário deve ser atraente, intuitiva e fácil de usar, levando em consideração as características do usuário-alvo. Oferecer uma experiência de usuário confortável e envolvente requer um planejamento cuidadoso do design visual, posicionamento de elementos e usabilidade;
- **Desenvolvimento do jogo educacional (A_7):** durante essa atividade o foco foi o desenvolvimento da aplicação abordada neste TCC, com base nas decisões tomadas nas atividades anteriores. Isso teve que ser feito utilizando-se as práticas modernas de desenvolvimento de jogos e tecnologias apropriadas, como frameworks de desenvolvimento e linguagens de programação;
- **Ajustes Finais (A_8):** com base nos resultados dos testes e da avaliação, esta etapa teve como foco a adaptação e o aperfeiçoamento do jogo educativo;

1.6 Organização do Documento

O restante deste documento segue uma estrutura composta por três capítulos. O Capítulo 2 tem como objetivo fornecer uma fundamentação teórica, onde são apresentados conceitos e tecnologias essenciais para a solução do problema abordado neste trabalho. Já o Capítulo 3 descreve a solução proposta, abrangendo os principais elementos produzidos durante o seu desenvolvimento, além de detalhar o processo de implementação. Finalmente, o Capítulo 4 traz as conclusões e considerações finais acerca do trabalho apresentado.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são discutidos os conceitos, definições e tecnologias utilizadas no desenvolvimento deste trabalho. Primeiro, este capítulo fala sobre o conceito de gamificação e suas utilizações. Depois, fala sobre a ferramenta *Git* e sua terminologia. A seguir, são apresentadas as tecnologias que foram utilizadas no desenvolvimento.

2.1 Gamificação

A gamificação se trata do uso de elementos e técnicas de jogos em contextos não relacionados a jogos, com o objetivo de engajar pessoas, motivar comportamentos, e facilitar o aprendizado, aplicando mecânicas como recompensas, níveis, desafios e competição (DETERDING et al., 2011). Esse é um conceito que pode ser usado para diversas áreas, como, por exemplo, para ambientes corporativos, onde geralmente é utilizada para treinar colaboradores, aumentar a produtividade, e melhorar o moral da equipe. Também pode ser aplicado na área da saúde, na qual pode ajudar no acompanhamento de tratamentos, promovendo hábitos saudáveis por meio de metas e feedbacks em tempo real.

O potencial da gamificação é um evento que já vem sendo observado há décadas, e teve a sua origem em programas de marketing e aplicações para web, com a finalidade de motivar, engajar e fidelizar clientes e usuários (ZICHERMANN e CUNNINGHAM, 2012). Entretanto, a evolução dos jogos e o aumento da sua popularidade trouxeram mais olhares para esse conceito, o que por consequência vem trazendo uma maior utilização prática dessas técnicas.

Outro ponto a ser destacado é a importância de uma boa implementação. Ao se gamificar algum processo antigo é necessário pensar bem como isso deve ser feito, pois uma má implementação pode causar uma sensação negativa no jogador, como, por exemplo, desmotivação, desinteresse ou até frustração. Então, para que a gamificação seja implementada de forma eficiente, é fundamental analisar o contexto e os desafios sob a perspectiva de um game designer (profissional focado no desenvolvimento de jogos eletrônicos), sempre pensando em realizar uma integração mais natural e equilibrando os desafios e recompensas (MCGONIGAL,2011). Essa visão é essencial para garantir que os elementos de jogo sejam adaptados corretamente à realidade do projeto e às necessidades dos usuários, evitando, assim, uma aplicação superficial ou desmotivadora.

2.1.1 **Gamificação na Educação**

A ideia principal da gamificação é simples, consistindo em trazer o prazer de se jogar para diferentes domínios de aplicação. Dentro do contexto educacional, é possível focar na sua utilização para aumentar a motivação dos estudantes, ao transformar o aprendizado em algo mais divertido e dinâmico, melhorando a retenção de informações e o desenvolvimento de habilidades.

Existem diversos projetos para trabalhar a gamificação para essa área, em diferentes vertentes, desde o ensino de matérias da educação básica, como português e matemática. Também há soluções para áreas mais específicas, como a área de desenvolvimento de software, conforme discutido na Seção 1.3.

2.1.2 **Bases teóricas na psicologia**

Um ponto forte a ser estudado na gamificação são os seus efeitos na psique humana. Os jogos tendem a prender e divertir os seus jogadores. Com isso, eles trabalham na motivação do jogador, para que ele se mantenha motivado a continuar jogando. Na psicologia isso é chamado de motivação intrínseca, a qual vem de dentro do indivíduo.

Além de atuarem na motivação o jogador, os jogos também trazem outro conceito, o estado de flow, que se caracteriza como o estado de total concentração na atividade que está sendo realizada, aumentando o foco e desligando o jogador do mundo exterior (CZIKSZENTMIHALYI, 1990). Ambos os efeitos combinados tornam a gamificação uma ferramenta forte, em especial para o contexto que esse trabalho visa acertar, que é um contexto mais educacional.

2.1.3 **Gamificação além das telas**

Geralmente, quando se fala de gamificação, automaticamente se pensa em plataformas e aplicações de software que implementam esse conceito. Mas, além dessa forma, também é comum adicionar essas técnicas para diferentes contextos da vida real (BURKE, 2015). Por exemplo, existem empresas que recompensam bem os seus funcionários que batem alguma meta ou até mesmo de clubes que realizam eventos com algum prêmio no final para quem tiver o melhor desempenho. Essas são maneiras possíveis de se aplicar conceitos de gamificação em problemas do mundo real.

2.2 Git

O *Git* é um sistema de controle de versão amplamente utilizado no desenvolvimento de software (CHACON e STRAUB, 2010). Ele foi criado por Linus Torvalds em 2005, com o objetivo de oferecer uma solução eficiente e confiável para gerenciar o histórico de alterações em projetos de código aberto do kernel do *Linux*. Desde então, o *Git* se tornou uma ferramenta fundamental para equipes de desenvolvimento em todo o mundo.

A finalidade do *Git* é permitir que desenvolvedores trabalhem em colaboração, acompanhem as alterações realizadas em arquivos e projetos, e revertam ou mesclem versões conforme necessário. Ao usarem o *Git*, os desenvolvedores podem controlar de forma eficiente as diferentes versões do código-fonte, gerenciando *branches* (ramificações) e fazendo *commits* (registros) de alterações.

A importância do *Git* reside em sua capacidade de facilitar a colaboração e o gerenciamento de projetos de software. Ele permite que várias pessoas trabalhem simultaneamente em um mesmo projeto, enquanto registra todas as alterações feitas em cada arquivo. Além disso, oferece recursos poderosos, como a mesclagem de branches e a resolução de divergências entre alterações, auxiliando no trabalho em equipe e na manutenção da integridade do código.

Ao adotarem o Git, as equipes de desenvolvimento podem se beneficiar de uma série de vantagens, como a capacidade de rastrear e entender facilmente as alterações feitas por cada desenvolvedor, reverter o código a versões anteriores, colaborar de forma eficiente com outros membros da equipe e gerenciar fluxos de trabalho complexos.

No contexto acadêmico e profissional, o *Git* é frequentemente utilizado em projetos de software, desde pequenos projetos individuais até grandes projetos colaborativos. Sua popularidade é atribuída à sua flexibilidade, eficiência e suporte à colaboração, tornando-o uma ferramenta essencial para o desenvolvimento de software moderno. As próximas subseções mostram alguns dos principais conceitos do Git.

2.2.1 ***Commit***

O *Commit* se trata de um registro de alterações realizadas em um projeto, realizando uma cópia exata da situação atual do projeto (CHACON e STRAUB, 2010). É por meio dele que é possível se fazer o versionamento do código, de forma a permitir um resgate mais fácil de informações e estado do projeto.

A principal vantagem do *commit* é a sua rastreabilidade. Ao registrar todas as alterações realizadas no código, o *Git* torna mais fácil para os desenvolvedores identificarem alterações específicas feitas a qualquer momento, rastrear quem fez essas alterações e entender o propósito por trás delas. Além disso, ele possibilita a recuperação de qualquer arquivo ou trecho de código que atualmente está inexistente, mas que já foi registrado antes.

2.2.2 **Branch**

As *branches* são literais ramificações independentes do código-fonte principal, permitindo que se tenha um desenvolvimento paralelo e uma organização eficiente de projetos de software (CHACON e STRAUB, 2010). Por padrão, o *Git* cria uma *branch* principal, geralmente chamada de *master* ou *main*, que contém a versão estável do código. No entanto, *branches* adicionais podem ser criadas a partir dessa branch principal para desenvolver recursos, corrigir erros ou executar testes sem afetar a estabilidade do projeto.

Graças a essa maneira de organização, o Git possibilita o desenvolvimento em paralelo, a colaboração em equipe e uma organização eficiente de projetos de software. Ao usarem as ramificações corretamente, os desenvolvedores podem otimizar o seu fluxo de trabalho e manter um histórico de desenvolvimento de projeto claro e organizado.

2.2.3 **Merge e Rebase**

Para atualizar as *branches* do projeto, combinando as alterações realizadas nas ramificações, o Git disponibiliza duas operações:, o *merge* e o *rebase*. Ambas operações resultam na atualização da *branch* escolhida, podendo ser inclusive a principal.

A operação de *merge* combina as alterações antes de atualizar a branch, criando um novo commit que une todas as alterações realizadas (CHACON e STRAUB, 2010). Ao visualizar os commits, é possível ver a conexão entre as

branches. Entretanto, se houver muitas *branches*, a visualização pode ser prejudicada mas, em contraponto, a ordem cronológica dos commits se manterá.

A operação *rebase*, por outro lado, move todos os commits para o final da linha de commits, resultando em uma linha de commits única (CHACON e STRAUB, 2010). Ela melhora a visualização do histórico de *commits*, porém não preserva a cronologia dos mesmos.

Depois de realizar as operações é possível haver conflito de código. Assim, cabe ao programador resolver manualmente tais conflitos antes de atualizar a *branch* desejada.

Após realizar operações de integração, o merge ou o rebase, pode ocorrer um conflito de código, que acontece quando duas ou mais alterações são feitas nas mesmas linhas ou regiões de um arquivo de forma incompatível. Nesses casos, o Git não consegue decidir automaticamente qual versão deve ser mantida, e cabe ao programador revisar e corrigir manualmente essas discrepâncias antes de atualizar a *branch* desejada.

2.2.4 **Repositórios**

O Git trabalha com uma arquitetura descentralizada, não possuindo um repositório central onde todas as modificações devem ser enviadas. Assim, o repositório pode ser local, ou seja, a máquina de quem estiver trabalhando no projeto, ou remoto, a partir de um servidor escolhido para manter o código. Alguns exemplos de repositórios são os projetos *GitHub*[7] e *GitLab*[8].

Ao se utilizar um repositório remoto, é necessário realizar um clone do projeto na máquina local. Depois de realizar alguma modificação nessa cópia local, os *commits* podem ser enviados para o repositório remoto, além de receber as modificações realizadas por outras pessoas

2.3 React

A *React.js* (FEDOSEJEV, 2015) é uma biblioteca JavaScript (FLANAGAN, 2012) de código aberto amplamente utilizada para criar interfaces de usuário em páginas web. Ela permite integrar a linguagem JavaScript no conteúdo e nos estilos

[7] https://github.com/
[8] https://gitlab.com/

de uma página, facilitando, assim, o desenvolvimento de aplicativos da Web responsivos e interativos (React, 2024).

Uma das principais características da *React* é o conceito de componentes. Os componentes são partes de código independentes e reutilizáveis que fazem parte da interface exibida na tela. Esses componentes podem ser agregados e combinados para criar interfaces complexas e ricas em recursos.

Uma das vantagens da *React* é a sua eficiência na atualização da interface. Ela usa um mecanismo chamado *Virtual DOM* para rastrear e gerenciar mudanças na interface. Isso permite que ela atualize apenas os componentes e elementos necessários, minimizando a quantidade de manipulação real do DOM. Isso resulta em atualizações de interface do usuário mais rápidas e eficientes (React, 2024), resultando em uma experiência mais fluida para os usuários.

Resumindo, a *React* é uma poderosa biblioteca JavaScript que facilita a criação de interfaces de usuário interativas e responsivas. Com o uso de componentes reutilizáveis e um mecanismo eficiente para atualizar a interface, ela fornece uma abordagem eficiente para o desenvolvimento de aplicativos da Web modernos e escaláveis.

2.4 Express

O *Express.js* (MARDAN, 2014) é um *framework* para desenvolvimento de aplicações *Node.js* (CANTELON, 2014) com javaScript. Ele foi criado em 2010 pelo desenvolvedor TJ Holowaychuk e é um projeto de código aberto licenciado sob a MIT (Massachusetts Institute of Technology), e se tornou em um dos *frameworks* mais famosos para a construção de aplicações e *back-end*, principalmente no uso de desenvolvimento de APIs (Application Programming Interface).

Como todo *framework,* o *Express* também possui características distintas. Por exemplo, ele fornece um sistema de roteamento completo que permite definir o comportamento do aplicativo para diferentes URLs, oferece suporte ao tratamento de exceções dentro do próprio aplicativo e habilita o tratamento de solicitações HTTP, entre outros recursos.

O fluxo de solicitação e resposta no *Express* é baseado em *middleware*. *Middlewares* são funções que controlam o processamento de informações ao longo do processo. É possível encadear vários *middlewares,* especificando-se a ordem em que eles serão executados.

2.5 Banco de Dados

O banco de dados é um sistema computadorizado de armazenamento de dados, que permite o registro de novos dados dentro de uma base de dados, nas quais são armazenados em arquivos digitais de dados (DATE, 2004). Em outras palavras, o banco de dados serve como um repositório ou recipiente para uma coleção de arquivos de dados computadorizados.

Outro conceito relevante a ser definido é o de Sistema de Gerenciamento de Banco de Dados (SGBD), que pode ser conceituado como um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações (ELMASRI, 2005). Em resumo, o SGBD permite ao usuário gerenciar de forma efetiva o banco de dados e definir o compartilhamento do mesmo com outros usuários.

Dentro de cada SGBD deve existir um suporte padrão a operações de CRUD (*Create, Read, Update, Delete*), que são as quatro principais operações para inserir, consultar, atualizar e excluir dados em um banco de dados. Essas operações são comumente realizadas a partir do uso de linguagens que são interpretadas e permitem ao usuário especificar a realização de operações desejadas para o SGBD. A linguagem mais predominante nesse cenário é o SQL (Structured Query Language) (DATE, 1989), linguagem que permite a especificação de operações a serem realizadas, escritas de forma estruturada e de fácil entendimento , sendo a linguagem padrão para banco de dados relacionais.

Atualmente, existem dois modelos de SGBDs: os relacionais e os noSQL. Os bancos de dados relacionais organizam os dados em tabelas com relações entre elas, enquanto que os bancos de dados noSQL armazenam os dados em formatos mais flexíveis, como documentos, chave e valor, família e colunas ou grafos, o que permite uma estrutura mais dinâmica para certos tipos de aplicações.

2.5.1 **PostgreSQL**

O PostgreSQL é um dos SGBDs objeto-relacionais mais usados dentro do mercado de programação. Possui código aberto e usa a linguagem SQL por padrão para realizar operações no banco de dados, mas aceita outras linguagens como PL/pgSQL (DOUGLAS, 2003).

O PostgreSQL dá suporte a diversas funcionalidades e permite o uso de extensões, que podem ser usadas para o desenvolvimento de aplicações específicas e bancos de dados. Por exemplo, a extensão PostGIS ao qual é utilizada para adicionar suporte a dados geoespaciais, permitindo realizar consultas espaciais complexas e manipular dados geográficos, ou a PgAudit oferece recursos de auditoria, essenciais para rastrear alterações e garantir a conformidade em ambientes sensíveis, como em sistemas financeiros, também temos o Hstore que oferece um tipo de dado chave-valor, ideal para armazenar informações semi-estruturadas em uma única coluna ou até mesmo o PLV8 que permite a escrita de funções em JavaScript dentro do PostgreSQL.

2.5.2 **MongoDB**

O MongoDB (BANKER, 2016) é um banco de dados noSQL que possui um modelo de base de dados orientado a documentos. Ele utiliza o formato Binary JSON (BSON), uma extensão do JSON, para armazenar os dados como arquivos binários (MongoDB, 2024). Por utilizar o JSON como um modelo para o formato dos dados, o MongoDB possibilita uma maior flexibilidade para lidar com dados dinâmicos e heterogêneos.

A sua flexibilidade traz grandes vantagens como, por exemplo, uma maior velocidade no desenvolvimento e uma maior escalabilidade em armazenar grandes volumes de dados não estruturados ou semiestruturados (BANKER, 2016).

2.5.3 **Redis**

O Redis (CARLSON, 2013) é um banco de dados noSQL no qual os dados são representados como pares de chave e valor e são armazenados em memória, ou seja, ele não necessita acessar o disco para acessar os dados. Isso possibilita uma alta performance para seus comandos de leitura e escrita, o tornando ideal para aplicações que necessitem de um menor tempo de resposta.

O Redis (CARLSON, 2013) pode ser usado de variadas formas, mas ele geralmente é utilizado como um complemento a um banco de dados principal, mas arcando com diferentes papeis importantes na aplicação. Por exemplo, ele pode servir como uma memória cache para os dados de retorno do banco de dados principal, que podem ser armazenados na memória durante um período de tempo

que pode ser pré-configurado, diminuindo, assim, a carga de trabalho do banco de dados principal.

2.5.4 **Prisma**

O *Prisma* (Prisma, 2024) é uma ORM (Object Relational Mapper) de código aberto, projetada para a utilização em aplicações *server-side* junto com o Node.js. oferecendo recursos avançados de mapeamento objeto-relacional. Ele permite que os desenvolvedores realizem operações de CRUD e outras tarefas relacionadas ao banco de dados de forma eficiente e simplificada.

O *Prisma* utiliza o conceito do *Prisma Client*, que é uma biblioteca gerada automaticamente a partir das especificações dadas através de um esquema de dados escrito pelo programador. Ao utilizar esse arquivo contendo o esquema de dados, o Prisma gera toda a estrutura do banco de dados e um conjunto de classes e métodos que permitem uma fácil interação com o banco de dados.

Outra característica importante é que o Prisma também oferece uma linguagem de consulta própria, chamada *Prisma Query Language* (PQL). Ela é uma linguagem tipada que permite que o desenvolvedor escreva as consultas ao banco de dados de forma declarativa. As especificações dessas consultas podem ser passadas dentro dos métodos gerados pelo Prisma.

Também vale a pena comentar que o Prisma oferece flexibilidade e suporte ao uso de vários banco de dados, em especial os mais populares. Ele também possui o suporte a migrations, que possibilitam um maior controle de alterações nos esquemas dos dados sem a necessidade de se escrever scripts, e o prisma CL, uma ferramenta de linha de comando que possibilita a interação com o Prisma.

3. A SOLUÇÃO PROPOSTA

Este capítulo aborda o desenvolvimento da ferramenta proposta neste Trabalho de Conclusão de Curso. Inicialmente, serão discutidos os tipos de usuários da aplicação e logo após serão apresentados os requisitos funcionais estabelecidos. Em seguida, será detalhado o processo de seleção do modelo de banco de dados e a criação da estrutura de dados. Em seguida, será apresentada a arquitetura do sistema, juntamente com uma descrição dos seus componentes. Por fim será descrito a implementação de cada um dos módulos apresentados na arquitetura do sistema.

3.1 Atores de Sistema

Esta seção tem como objetivo descrever os dois tipos de usuários que utilizarão a ferramenta proposta neste trabalho. São eles:

- **jogador:** esse é o usuário comum da aplicação e representa qualquer pessoa que tenha o interesse de jogar, podendo ser alunos ou programadores que tenham o interesse em aprender sobre o Git. Esse tipo de usuário apenas interage com o jogo e tem a possibilidade de enviar relatos de erros sobre os níveis do jogo;
- **administrador:** esse é o usuário que gerencia a aplicação, podendo gerenciar jogadores, modificar níveis, atividades e outros conteúdos da ferramenta. Ele pode ler os relatos de erro enviados por jogadores. Por ter esse nível de poder sobre a aplicação, esse tipo de usuário tem diferentes níveis de permissões.

3.2 Requisitos Funcionais

Em colaboração com o orientador deste trabalho, foram definidos os requisitos funcionais, com o objetivo de atingir os objetivos gerais e específicos definidos para este trabalho. Esses requisitos foram formulados para garantir que o jogo fosse uma nova maneira de se aprender a tecnologia, integrando teoria e prática no ensino do *Git*. O público-alvo é qualquer pessoa que queira aprender e aprimorar suas habilidades nessa ferramenta de gerenciamento de versão. Os requisitos propostos são os seguintes:

- **Sistema de Capítulos (RF1):** o jogo deveria conter um sistema de capítulos, em que cada capítulo teria como foco ensinar um conjunto

de termos e comandos para o usuário. A ideia central do capítulo era separar o nível de dificuldade do conteúdo e atividades, além de dar uma sensação de avanço para o jogador. Cada capítulo seria composto por níveis e por uma prova no final;

- **Sistema de Níveis (RF2):** o jogo deveria conter um sistema de níveis que seria focado em ensinar algum termo, conceito ou comando do Git. Cada nível deveria conter uma explicação teórica do conteúdo, para logo após ter duas ou três atividades práticas, cada uma contendo algum objetivo no qual o usuário teria que colocar o que acabou de aprender em prática;
- **Sistema de Provas (RF3):** o desempenho do usuário nas provas seria considerado para decidir se o usuário estaria apto a passar para o próximo capítulo. Para isso, cada prova deveria englobar todos os conteúdos ensinados durante o capítulo, sendo composta de um ou vários objetivos a serem realizados. Quando o usuário concluísse a prova, o capítulo atual seria marcado como concluído e o acesso ao próximo capítulo seria liberado;
- **Conteúdo Flexível (RF4):** cada nível do jogo seria composto de conteúdos, que seriam geralmente assuntos e atividades. Esses conteúdos deveriam ser flexíveis, permitindo que o administrador possa decidir a posição dos itens na tela e quais itens o usuário pode usar, como por exemplo, onde ficaria o texto.
- **Salvamento de progresso (RF5):** quando o usuário finalizasse uma atividade, nível, prova ou capítulo, o item finalizado deveria ser registrado na base de dados como concluído, permitindo que, na próxima vez que o usuário entrasse, não precisasse repetir todo o processo novamente;
- **Visualização do progresso para conclusão (RF6):** o jogo deveria retornar ao usuário o seu progresso atual para a conclusão do capítulo atual e para a conclusão do jogo como o todo, com o intuito de motivar o jogador a concluir 100% do jogo;
- **Jogar sem conta registrada (RF7):** o usuário poderia jogar sem estar autenticado na aplicação, mas não poderia ter informações sobre o seu progresso em outro computador;

- **Dashboard de informações do site (RF8):** o jogo deveria conter uma tela de *dashboard* acessível apenas aos seus administradores. A finalidade dessa tela seria trazer diversos dados sobre o site e qualquer relato informado pelos usuários;
- **Renderização de *Markdown* (RF9):** o jogo deveria conter um renderizador de texto *markdown*[9], que poderia ser usado para explicações de conteúdos;
- **Relatar problemas (RF10):** o jogo deveria possibilitar ao usuário relatar problemas encontrados durante a execução do jogo, por meio de um texto explicando o problema encontrado. Depois de o usuário ter relatado algum problema, os administradores poderiam resolver o problema e marcá-lo como concluído.

3.3 O armazenamento de dados

Para a implementação do jogo proposto neste TCC foram utilizados três bancos de dados: um banco de dados relacional, um banco de dados orientado a documentos e um banco de dados baseado em chave e valor. As próximas seções descrevem em mais detalhes como funcionam esses bancos de dados.

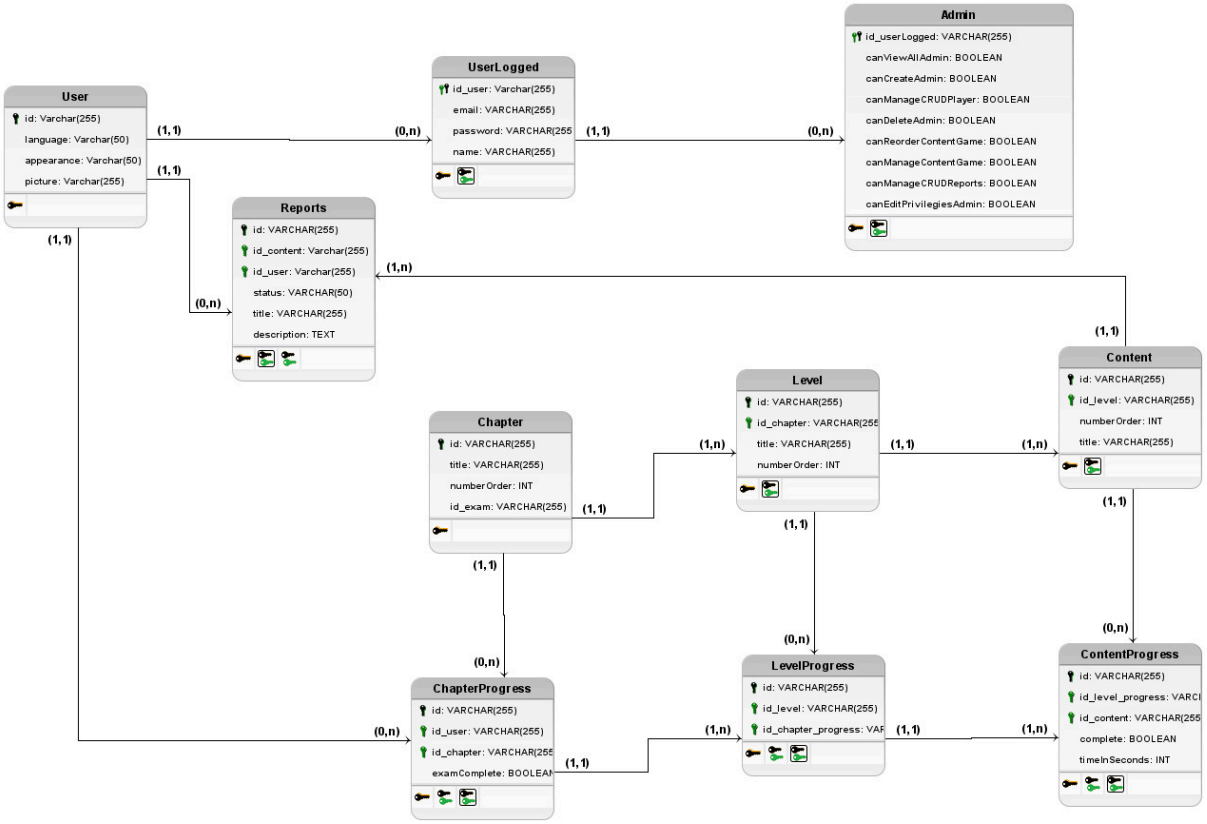
3.3.1 **O banco de dados relacional**

Para o desenvolvimento deste trabalho, optou-se pela utilização de um banco de dados relacional para o gerenciamento dos dados gerais do jogo. Esse banco de dados é responsável por armazenar os dados sobre o conteúdo do jogo e também as informações acerca dos usuários, além de guardar dados conteúdos do jogo, dados estes mais simples e que era desejável ter uma maior consistência nos seus valores, como por exemplo o título de cada um dos níveis e capítulos, garantindo assim que todos os registros das suas tabelas tenham os valores que foram especificados na sua construção.

Utilizando essa base de dados foi aplicado o método de divisão dos dados em várias tabelas como consequência, abaixo é possível visualizar como ficou a modelagem de dados para o banco postgreSQL:

[9] Linguagem de marcação simples

Figura 1 - Modelagem de dados



Fonte: Autor

De acordo com a Figura 1, observa-se que o banco de dados é composto por dez tabelas. Essas tabelas armazenam dados sobre três elementos: o jogo, os usuários e os progressos dos usuários no jogo.

Os dados dos usuários são armazenados na tabela *User*. Para cada usuário são armazenadas as seguintes informações: linguagem, aparência (Modo Dark ou Light do site) e imagem. A tabela *ChapterProgress* é usada para possibilitar que todos os usuários tenham o seu progresso armazenado na base de dados de forma transparente, inclusive para os usuários anônimos, que têm a impressão de que os dados estão sendo armazenados localmente. Ainda com relação à parte de usuários, a tabela *UserLogged* guarda dados acerca dos usuários cadastrados no jogo, mantendo informações como email e senha de forma criptografada. Por fim, a tabela de *Admin* serve para separar os usuários administradores, e guarda os privilégios de cada administrador cadastrado.

Saindo do contexto de usuário, o conteúdo do jogo é armazenado em três tabelas: *Chapter*, *Level* e *Content*. A tabela *Chapter* armazena todos os dados

referentes aos capítulos do jogo, como o título e a ordem em que ele deve ser apresentado. Esses dois dados também são guardados nas tabelas *Level* e *Content*, como visto na Figura 1. A tabela *Level* registra os dados de cada nível, contendo uma relação com a tabela *Chapter*. Por fim, a tabela *Content* armazena os mesmos dados genéricos da anterior, mais o identificador dos dados de conteúdo dentro da database do MongoDB.

Outro conteúdo a ser falado se trata da tabela *Report*, que guarda todos os relatos de problemas enviados pelos próprios jogadores. Para isso ela armazena o identificador do relato e o texto informando o problema, para que possa ser consertado.

Por fim, o banco de dados também tem a tabela *LevelProgress,* que registra o progresso que os usuários estão tendo com relação a um nível e a tabela *ContentProgress,* que armazena o progresso dos usuários com relação a um conteúdo, guardando se o mesmo foi concluído ou não, como pode ser observado na Figura 1.

3.3.2 **MongoDB**

O objetivo principal de se adicionar o MongoDB como um banco de dados é secundário permitir a utilização de formatos de dados mais dinâmicos. Com isso em mente, ele foi utilizado para armazenar dados de conteúdos do jogo, como itens que podem existir em atividades ou em exames. Um exemplo desse tipo de item são as questões de múltipla escolha que podem ser usadas nas atividades ou a ordem em que se deve apresentar os itens na tela e até mesmo resolução da atividade.

Outra vantagem do uso do MongoDB para conteúdos mais dinâmicos está na fácil implementação de novos itens para esses conteúdos, como, por exemplo, adicionar uma nova forma de explicar algum conteúdo ou um novo modo de resolver ou apresentar alguma atividade, nas quais é necessário apenas implementar essa feature no frontend e enviar o formato de dados que desejar para o banco de dados.

Além disso, a escalabilidade oferecida pelo MongoDB facilita futuras implementações como a adição de novos modos de jogo ou uma maior facilidade na inclusão de diferentes idiomas de forma mais simples e eficiente.

3.3.3 **Redis**

O terceiro e último banco de dados do jogo foi implementado usando o Redis. A sua função nesta aplicação é mais pontual, mas vital para algumas funcionalidades desenvolvidas, como, por exemplo, o armazenamento de *tokens* e sessões temporárias. Por exemplo, em um caso de um processo de recuperação de senha, no qual e existe um tempo limite para se usar o método de recuperar, ou para o caso dos dados de estatísticas e progresso do usuário, no qual são realizados cálculos a partir dos dados trazidos da base de dados principal e depois guardado com um espaço de tempo curto no Redis.

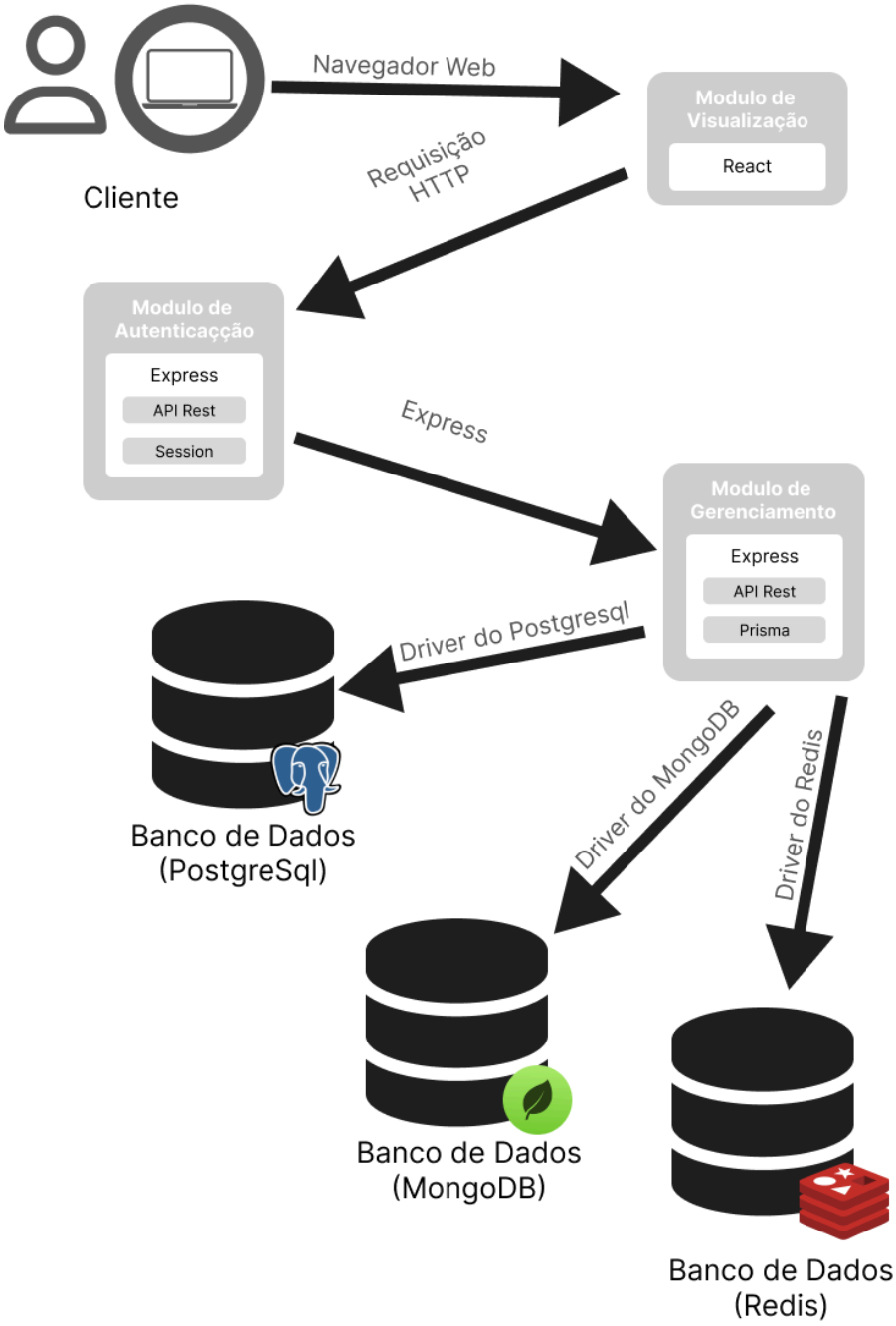
Assim, este banco de dados atua como uma memória cache, armazenando temporariamente a resposta de requisições anteriores realizadas nos endpoints e entregando como resposta para as novas requisições. Esses dados são mantidos geralmente por cinco minutos, para a existência dessa informação no Redis e sempre tendo o cuidado de manter atualizado a versão da resposta guardada no Redis.

A utilização do Redis se estende para todas as rotas da aplicação, cobrindo desde as rotas de usuários até as rotas direcionadas ao conteúdo do jogo. Com isso, há uma redução no tempo da resposta e um menor número de processamentos necessários para os bancos de dados principais, já que isso elimina a necessidade de sempre se realizar consultas para esses dois bancos de dados.

3.4 Arquitetura da Ferramenta

A arquitetura escolhida para a construção do jogo é composta de dois módulos principais, que separam as funções de *back-end* e *front-end* da aplicação. O primeiro módulo é o de visualização, que é a partir de onde o usuário interage com a aplicação (o *front-end)*. Essa interação ocorre por meio de páginas web. Já o segundo módulo é o de autenticação e é responsável pelo controle de usuários logados no sistema. Por fim, o módulo de gerenciamento é responsável por controlar a comunicação com o banco de dados. Os módulos de autenticação e de comunicação compõem o *back-end* do jogo. A relação entre os módulos da aplicação pode ser visualizada na Figura 2, que descreve de forma visual a arquitetura que foi utilizada.

Figura 2 - Arquitetura do projeto



Fonte: Autor

A arquitetura escolhida para a implementação do jogo traz uma série de vantagens, entre elas a independência e uma clara divisão de responsabilidades entre os diferentes módulos. Essa abordagem visa facilitar a manutenção do jogo, tornando cada módulo menos complexo. Além disso, a arquitetura prioriza a segurança, impedindo o acesso direto do módulo de visualização ao banco de

dados. Para que isso não ocorra, é usado o acesso por meio da API do módulo de gerenciamento, que controla e limita as consultas e o acesso ao banco de dados.

A função do módulo de visualização é interagir diretamente com o usuário final. Ele recebe requisições e apresenta os resultados obtidos a partir do *back-end* por meio de páginas geradas usando HTML, folhas de estilo CSS e linguagem de programação JavaScript. Todas essas tecnologias são amplamente utilizadas para criar e exibir páginas da web. Para proporcionar uma experiência mais agradável ao usuário, este módulo utiliza a biblioteca React.js, que é baseada na linguagem JavaScript. Essa biblioteca permite a criação de páginas dinâmicas, proporcionando uma interação mais responsiva e flexível.

O módulo de autenticação é centrado em verificar e gerenciar a autenticação de usuário na aplicação, além de verificar que tipo de usuário está autenticado durante a execução do jogo. Para todos os usuários é possível guardar o progresso realizado até o momento do jogo na base de dados, tendo acesso livre às rotas comuns dos jogadores. Já os usuários com uma conta do tipo administrador podem gerenciar qualquer conteúdo do jogo, seja prova, nível, atividades, entre outros, sendo limitado pelas suas permissões.

Por fim, o módulo de gerenciamento é responsável por estabelecer a comunicação com os três bancos de dados utilizados pelo jogo e fornecer os resultados das consultas de qualquer conteúdo relacionado ao jogo. Isso é feito por meio dos *endpoints* fornecidos pelo serviço REST.

Para a implementação dos módulos que compõem o *back-end* da aplicação foi utilizado o framework *Express*, que fornece um poderoso conjunto de recursos para o desenvolvimento de aplicações web. O *Express* facilita a criação de APIs de forma segura e confiável. Além disso, o acesso ao banco de dados relacional é feito por meio do *Prisma ORM*, que utiliza o driver do *PostgreSQL* para mapear as tabelas e executar as consultas necessárias, garantindo um acesso eficiente aos dados. Para os demais bancos de dados (MongoDB e Redis) foram usados apenas os seus respectivos drivers.

3.5 Implementação

Esta seção descreve a implementação dos módulos que compõem a arquitetura da ferramenta proposta por este TCC.

3.5.1 **Módulo de gerenciamento**

O módulo de gerenciamento é o módulo mais importante do backend da aplicação. Ele realiza todas as consultas aos bancos de dados e manipula corretamente os dados de resposta, retornando com o formato correto. Nesse módulo acontecem operações importantes para o administrador do sistema, como o gerenciamento de usuários, o gerenciamento do jogo, o gerenciamento de relatórios, a gravação do texto markdown e quaisquer outras utilidades vindas a partir do uso de uma base de dados.

Esse módulo foi construído usando as tecnologias TypeScript e Node.js. O *framework* de aplicações web *Express*, criando endpoints que retornam conteúdos no formato JSON. Para a comunicação com o banco de dados foi utilizada a ORM Prisma para realizar operações no banco de dados postgreSQL, e os drivers do MongoDB e Redis para realizar quaisquer operações para os seus respectivos bancos.

3.5.2 **Módulo de Autenticação**

O módulo de autenticação é o responsável pelo gerenciamento de usuários. O jogo aceita três tipos de usuários: não autenticados (anônimos), autenticados e administradores. A lógica por trás desse módulo é simples. Ele detecta o tipo de usuário que está acessando o jogo e controla o acesso ao módulo de gerenciamento. As rotas comuns de jogadores podem ser acessadas por qualquer usuário. Entretanto, as rotas com conteúdos mais delicados ou para a atualização do conteúdo do jogo só podem ser acessadas por usuários com permissão de administrador. O administrador apenas poderá ter contato com qualquer rota mais protegida se o mesmo contiver os privilégios necessários.

3.5.3 **Módulo de Visualização**

O módulo de visualização representa o frontend da aplicação. A partir desse módulo o usuário pode interagir com o conteúdo ao qual ele pode ter acesso. Por esse motivo tentou-se criar uma interface gráfica limpa e intuitiva, focando em aspectos como a beleza, acessibilidade, facilidade de uso e que fosse o mais completa possível. Para a construção deste módulo, foi utilizada a biblioteca React em conjunto com a linguagem TypeScript, para a construção das páginas webs. O acesso a funcionalidades do sistema está dividido em várias páginas. Entretanto,

pode-se dizer que o jogo tem três páginas principais: a lista de capítulos, a página responsável pela renderização do conteúdo do jogo e a página de Dashboard.

3.6 Telas do jogo

Esta seção apresenta as principais telas oferecidas pelo jogo proposto neste TCC.

3.6.1 **Listagem de Capítulos:**

Para o acesso a todas as funcionalidades oferecidas pelo jogo é necessário iniciar de uma tela de navegação. A tela de listagem e capítulos realiza esse trabalho, permitindo ao jogador escolher algum conteúdo que queira estudar e partir diretamente para ele. Isso proporciona uma flexibilização para os casos nos quais o jogador tem interesse em estudar algum tópico em específico.

A página de listagem de capítulos é descrita na Figura 3. Nela, é possível observar todos os capítulos do jogo e, abaixo de cada capítulo, a sua lista de níveis, que contém uma lista de botões para navegar nos tópicos desejados. Outro ponto importante de se observar é a visualização de uma barra de progresso na parte superior da página, que mostra o progresso do usuário no jogo, e os percentuais na parte da direita da página, que mostram o progresso do usuário em cada nível.

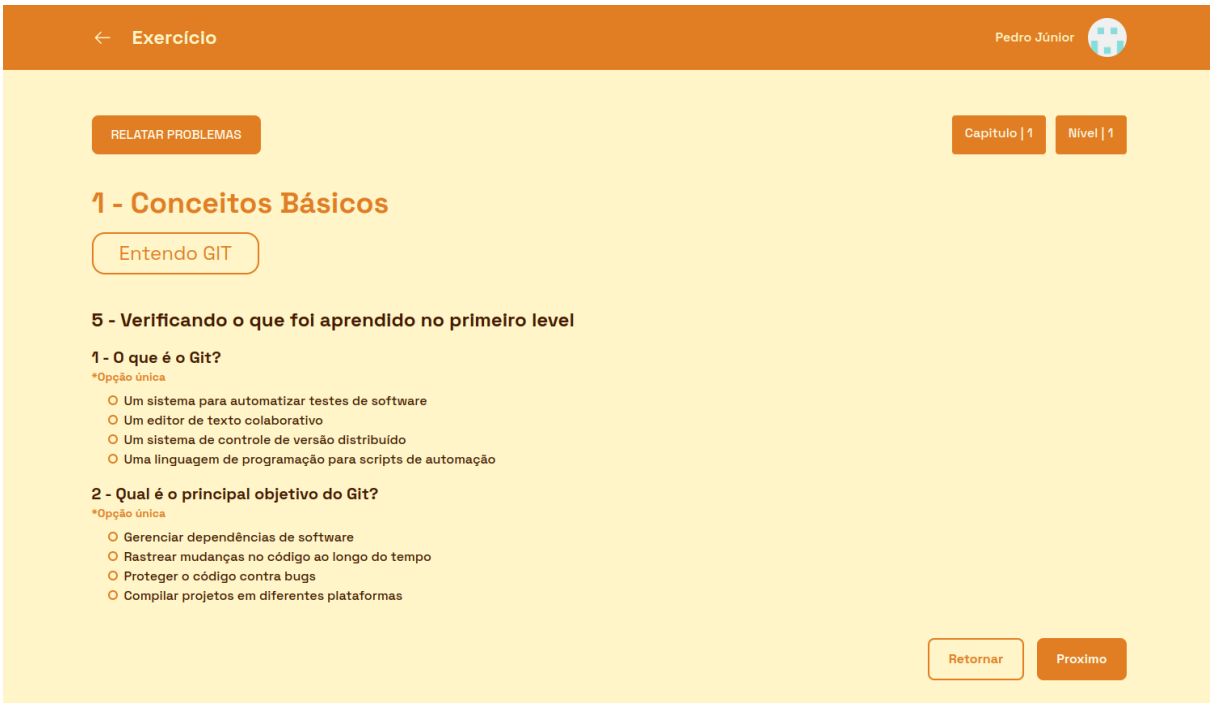
Figura 3 - Página de Listagem de Capítulos

Fonte: Autor

3.6.2 **Página de conteúdo**

A página e conteúdo é a página mais mutável do jogo, podendo disponibilizar qualquer conteúdo usado para o aprendizado do usuário. Cada conteúdo é dividido em duas partes: assunto e atividade. A Figura 4 mostra o exemplo uma atividade básica sobre o conteúdo do primeiro capítulo.

Figura 4 - Página de Atividade



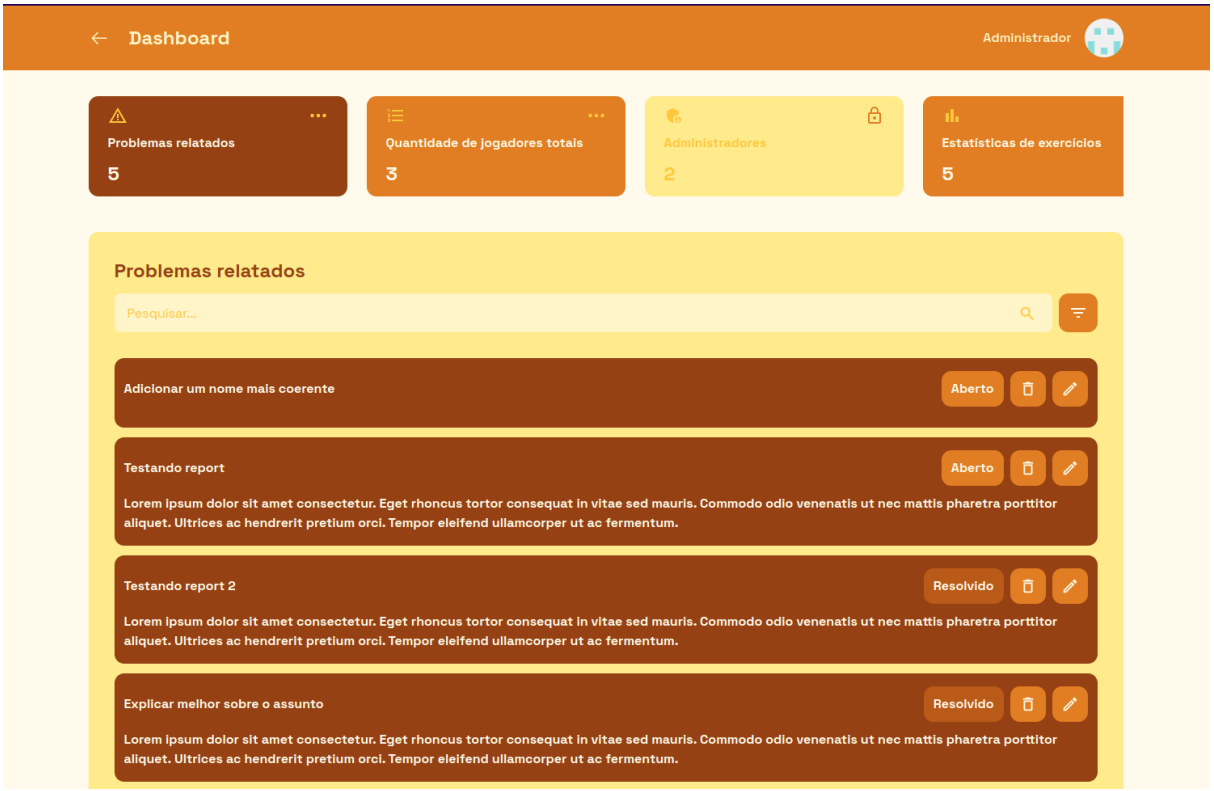
Fonte - Autor

Um ponto importante a se observar nessa tela é que as informações de capítulo e nível referentes à atividade são exibidas na parte superior da página. Também é importante destacar a existência do botão de relatar problemas, que serve para os jogadores avisarem qualquer problema que venha a aparecer no conteúdo que eles estão acessando, como, por exemplo, algum caso em que uma resposta errada esteja sendo aceita.

3.6.3 **Tela de Dashboard:**

A tela de dashboard é a tela na qual os administradores podem ter contato com as informações do jogo, que vão desde os problemas relatados até informações de usuários. As informações podem ser acessadas a partir de quatro cartões, que podem ou não estar habilitados de acordo com o nível de privilégio do administrador. A Figura 5 mostra a tela que lista todos os problemas relatados por outros usuários.

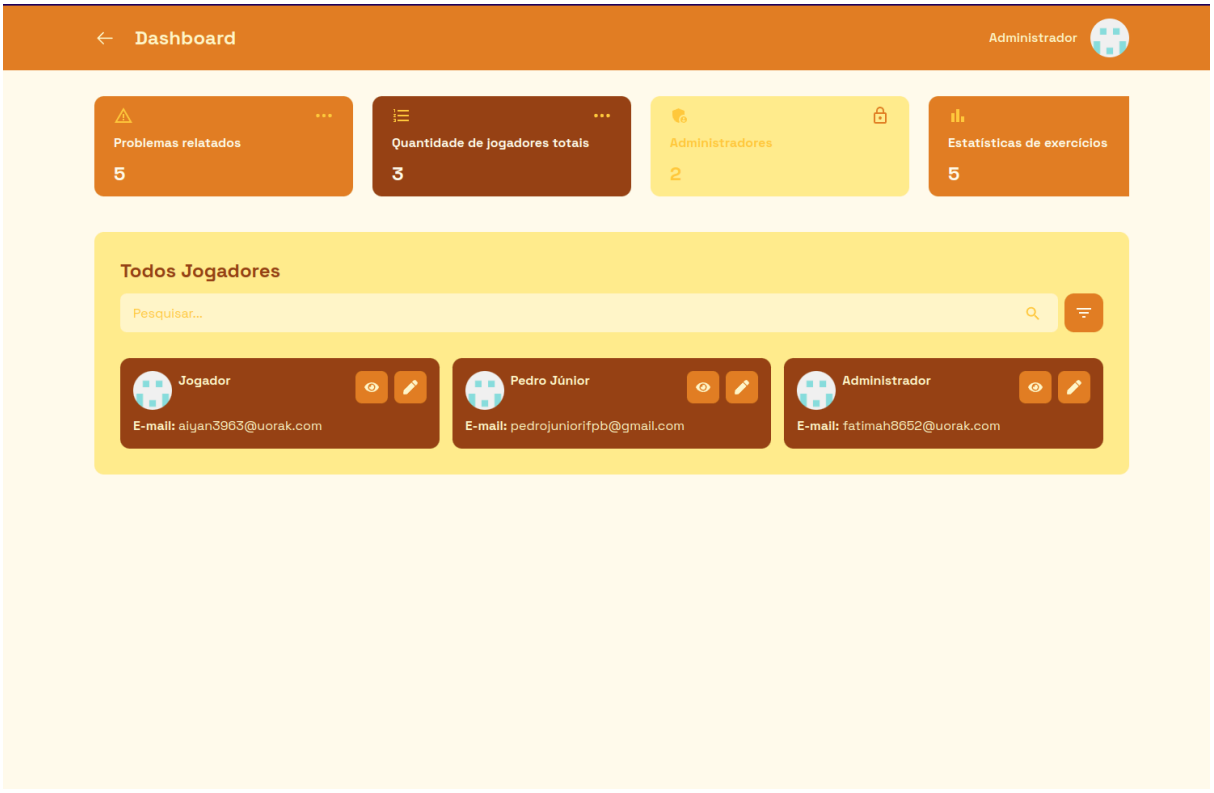
Figura 5 - Página de Dashboard - Reports



Fonte: Autor

É possível verificar que nessa página existe a opção de se utilizar um filtro e uma barra de pesquisa para facilitar a busca por um registro específico. Esses dois componentes também estão em outras páginas de dashboard, como pode ser visualizado na Figura 6, que mostra outra página de dashboard sobre algo.

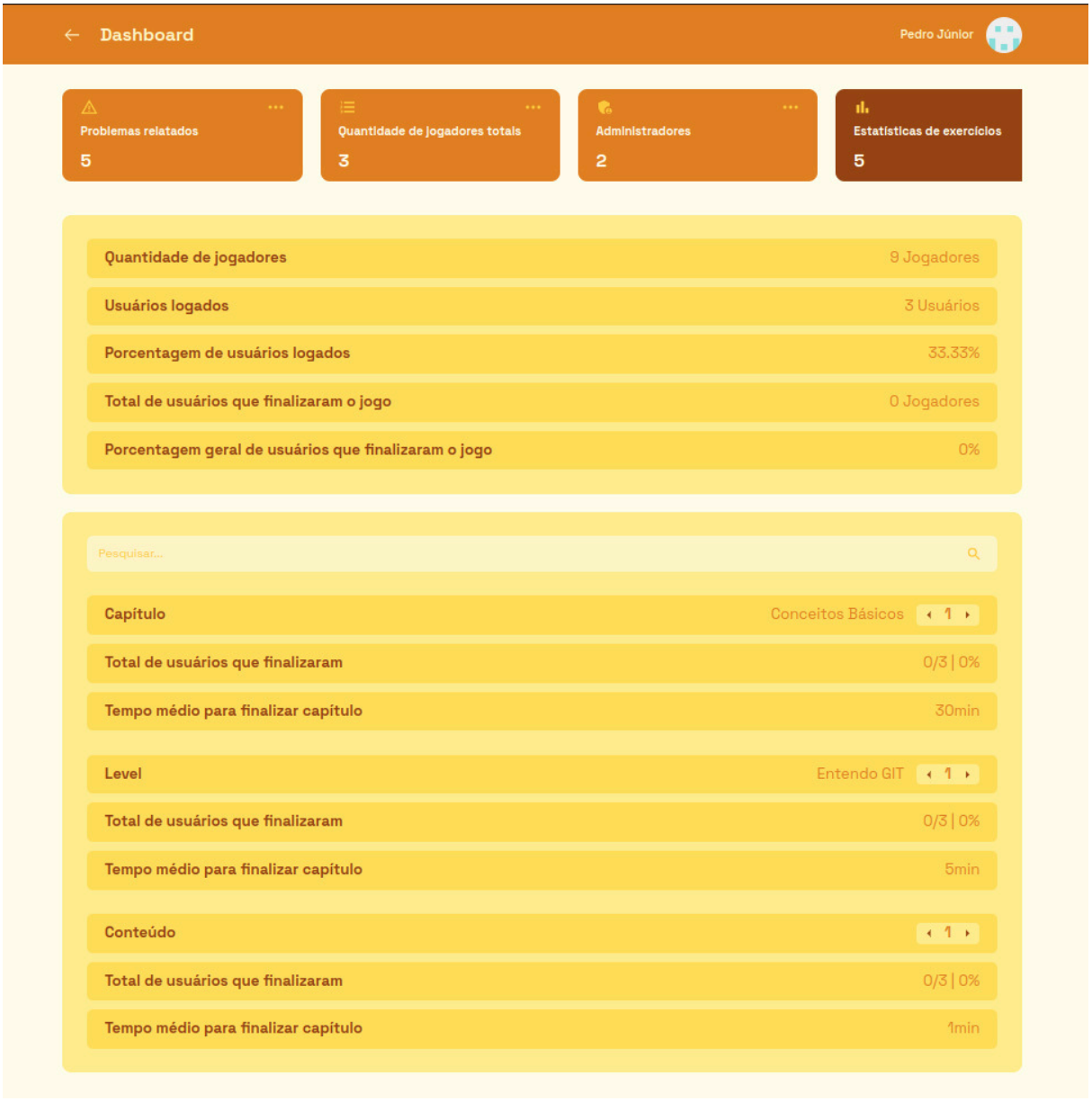
Figura 6 - Página de Dashboard - Jogadores



Fonte: Autor

Finalmente, é válido trazer a última página de dashboard, exibida na Figura 7, que se trata da página que oferece informações estatísticas do jogo, como, por exemplo, a quantidade de jogadores que finalizaram o jogo ou quantidade de jogadores anônimos. Além disso, ela mostra informações relacionadas a conteúdos do jogo, como, por exemplo, o tempo médio que os usuários estão levando para finalizar um determinado capítulo, nível ou atividade, ou a quantidade de pessoas que finalizaram, ajudando o administrador a ter uma melhor ideia de quais conteúdos estão difíceis ou fáceis demais e que devem passar por alguma modificação.

Figura 7 - Página de Dashboard - Estatísticas



Fonte: Autor

4. CONCLUSÃO

O Git é hoje uma das ferramentas mais populares para se realizar o controle de versões de um código. Entretanto, para muitos usuários, aprender essa tecnologia ainda é um desafio. Um problema recorrente no aprendizado de Git é a dificuldade em se manter o engajamento durante o processo de aprendizado, que muitas vezes se torna cansativo e desinteressante devido à variedade de aulas, artigos e tutoriais disponíveis. Em resposta a essa limitação, este TCC propôs um jogo interativo como alternativa, oferecendo uma maneira mais prática e envolvente de aprender Git. Essa abordagem proporciona uma experiência de aprendizado mais imersiva, permitindo que o usuário aprenda de maneira contínua e prática, sem os obstáculos típicos dos métodos tradicionais.

Assim, a criação de um jogo para o ensino do *Git* apresenta potenciais benefícios no ensino e aprendizado dessa ferramenta de controle de versão. Através de uma abordagem interativa e divertida, os jogadores podem ter a oportunidade de se familiarizar com os comandos e conceitos do *Git*. Essa experiência prática pode ajudar os usuários a entender melhor os recursos dessa ferramenta, como clonagem, confirmação e resolução de conflitos, preparando-os para trabalhar de forma colaborativa e gerenciar o código-fonte com mais eficiência.

Ao fornecer um ambiente virtual seguro para explorar e testar diferentes cenários, o jogo desenvolvido neste TCC permite que os usuários desenvolvam confiança em suas habilidades antes de aplicá-las em projetos reais. Além disso, o aspecto interativo do jogo pode aumentar o envolvimento e a motivação do jogador, tornando o processo de aprendizagem mais agradável. Com ele, espera-se que estudantes, desenvolvedores e profissionais de tecnologia se beneficiem dessa abordagem de aprendizado alternativa. A utilização desse jogo como ferramenta educacional pode contribuir para um melhor entendimento e uso do *Git*, aprimorando as práticas de desenvolvimento de software e facilitando a colaboração em equipe.

Por fim ainda existem pontos a serem melhorados e expandidos para essa aplicação, com a finalidade de a tornar cada vez mais superior às suas versões anteriores. Exemplos de possíveis trabalhos futuros são a internacionalização do jogo, trazendo para a aplicação diferentes idiomas que o usuário possa escolher, a implementação de mais e novos testes, guiando a aplicação para um caminho com uma maior confiabilidade, e até mesmo novos modos de jogo, como uma emulação de projetos, adicionando um terminal com visualização de arquivos, a visualização

da árvore do git em tempo real, o que facilitaria o aprendizado, e um modo sandbox para a aplicação, usando as outras três features anteriores.

REFERÊNCIAS

BANKER, Kyle et al. **MongoDB in action: covers MongoDB version 3.0**. Simon and Schuster, 2016.

BOTIAN, Igor. **Think Like a Git**. Disponível em: https://think-like-a-git.net/. Acesso em: 7 de julho de 2023.

BURKE, Brian. **Gamificar: como a gamificação motiva as pessoas a fazerem coisas extraordinárias**. DVS editora, 2015.

CANTELON, Mike et al. **Node. js in Action**. Greenwich: Manning, 2014.

CARLSON, J. **Redis in Action**. Manning, 2013.

CHACON, Scott; STRAUB, Ben. Pro Git. **Recuperado de: http://labs. kernelconcepts. de/downloads/books/Pro% 20Git**, 2010.

COTTLE, Peter. **learngitbranch**. Disponível em: https://learngitbranching.js.org/. Acesso em: 7 de julho de 2023.

CZIKSZENTMIHALYI, Mihaly. **Flow: The psychology of optimal experience**. New York: Harper & Row, 1990.

DA SILVA, Andreza Regina Lopes et al. **Gamificação na educação**. Pimenta Cultural, 2014.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. Elsevier Brasil, 2004.

DATE, Christopher J. **A Guide to the SQL Standard**. Addison-Wesley Longman Publishing Co., Inc., 1989.

DETERDING, Sebastian et al. From game design elements to gamefulness: defining" gamification". In: **Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments**. 2011.

DOUGLAS, Korry; DOUGLAS, Susan. **PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases**. SAMS publishing, 2003.

ELMASRI, Ramez et al. **Sistemas de banco de dados**. 2005.

FARDO, Marcelo Luis. A gamificação aplicada em ambientes de aprendizagem. **Revista Novas Tecnologias na Educação**, v. 11, n. 1, 2013.

FEDOSEJEV, Artemij. **React. js essentials**. Packt Publishing Ltd, 2015.

FLANAGAN, David. **JavaScript: o guia definitivo**. Bookman Editora, 2012.

Git. Software de controle de versão distribuído. Disponível em: https://git-scm.com/. Acesso em: 7 de julho de 2023.

GitHowTo. Disponível em: https://githowto.com/pt-BR. Acesso em: 7 de julho de 2023.

MARDAN, Azat. **Express.js Guide: The Comprehensive Book on Express.js**. Azat Mardan, 2014.

MCGONIGAL,Jane. **Reality Is Broken: Why Games Make Us Better and How They Can Change The World**. Nova Iorque: The Penguin Press, 2011.

Meta. Documentação do **React.js**. Disponível em: https://pt-br.react.dev/. Acesso em: 11 de outubro de 2024.

MongoDB. Documentação do **MongoDB**. Disponível em: https://www.mongodb.com/pt-br/docs. Acesso em: 11 de outubro de 2024.

OhMyGit. Oh My Git. In: GIT-LEARNING-GAME. Disponível em: https://ohmygit.org/. Acesso em: 11 de outubro de 2024.

OpenJs. Documentação do **Express.js**. Disponível em: https://expressjs.com/pt-br/. Acesso em: 11 de outubro de 2024.

Prisma. Documentação do Prisma. Prisma. Disponível em: https://www.prisma.io/docs/. Acesso em: 11 de outubro de 2024.

Redis. Documentação do **Redis**. Disponível em: https://redis.io/docs/latest. Acesso em: 11 de outubro de 2024.

REIS, Daniel. **4noobs**. Disponível em: https://github.com/DanielHe4rt/git4noobs. Acesso em: 11 de outubro de 2024.

SAPPENS. Gamificação e Psicologia: Como a psicologia se alia à gamificação para engajar pessoas?. Sappens, 2023. Disponível em: https://www.sappens.com.br/gamificacao-psicologia. Acesso em: 11 de outubro de 2024.

ZICHERMANN, Gabe; CUNNINGHAM, Christopher. **Gamification by Design. Implementing Game Mechanics in Web and Mobile Apps**. Canada: O'Reilly Media, 2011.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Cajazeiras - Código INEP: 25008978
Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

TCC

Assunto:	TCC
Assinado por:	Pedro Junior
Tipo do Documento:	Projeto
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Pedro Pereira de Morais Junior**, **DISCENTE (202112010020) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 23/10/2024 21:59:08.

Este documento foi armazenado no SUAP em 23/10/2024. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1289510
Código de Autenticação: 88bad9f0a2