UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM ENGENHARIA AEROESPACIAL

Erick Rogério da Silva

GREENLEARN - APLICAÇÃO WEB PARA EDUCAÇÃO EM SUSTENTABILIDADE E ENERGIAS

Santa Maria, RS
2023

Erick Rogério da Silva

GREENLEARN - APLICAÇÃO WEB PARA EDUCAÇÃO EM SUSTENTABILIDADE E ENERGIAS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia Aeroespacial da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia Aeroespacial**.

ORIENTADORA: Prof.ª Natália de Freitas Daudt

Santa Maria, RS
2023

Erick Rogério da Silva

GREENLEARN - APLICAÇÃO WEB PARA EDUCAÇÃO EM SUSTENTABILIDADE E ENERGIAS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia Aeroespacial da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia Aeroespacial**.

Aprovado em 20 de julho de 2023:

Natália de Freitas Daudt, Prof. Dra. (UFSM)
(Presidenta/Orientadora)

Eduardo Escobar Bürger, Prof. Dr. (UFSM)

Lucas Vizzotto Bellinaso, Prof. Dr. (UFSM)

Santa Maria, RS
2023

RESUMO

GREENLEARN - APLICAÇÃO WEB PARA EDUCAÇÃO EM SUSTENTABILIDADE E ENERGIAS

AUTOR: Erick Rogério da Silva
ORIENTADORA: Natália de Freitas Daudt

Apoiando-se na ascensão das ferramentas tecnológicas no âmbito da educação, o presente trabalho apresenta o desenvolvimento da aplicação *web* de cunho educacional *greenlearn*, que tem como objetivo complementar o ensino em sustentabilidade e energias renováveis no âmbito do projeto de extensão "Energizando o Futuro". A aplicação foi desenvolvida utilizando as tecnologias *React.js* e *Node.js*, e oferece uma abordagem interativa por meio de perguntas, distribuídas em categorias de diferentes níveis de dificuldade. O *greenlern* busca proporcionar aos usuários uma experiência de aprendizado dinâmica e progressiva, incentivando o engajamento e a aquisição de novos conhecimentos sobre os temas abordados.

Palavras-chave: Aplicação web. Desenvolvimento web. Educação. Energias renováveis. Ensino virtual. Recursos didáticos digitais. Sustentabilidade. Tecnologia.

ABSTRACT

GREENLEARN - WEB APPLICATION FOR SUSTAINABILITY AND ENERGY EDUCATION

AUTHOR: Erick Rogério da Silva
ADVISOR: Natália de Freitas Daudt

Leveraging the rise of technological tools in education, this paper presents the development of the educational web application greenlearn, aiming to complement the teaching of sustainability and renewable energies within the scope of the "Energizando o Futuro" extension project, affiliated with the Universidade Federal de Santa Maria. Greenlearn was built using React.js and Node.js technologies. It offers an interactive approach through a series of questions organized into categories of varying difficulty levels. The greenlearn application seeks to provide users with a dynamic and progressive learning experience, fostering engagement and the acquisition of new knowledge in the covered subjects.

Keywords: Digital educational resources. Education. Renewable energies. Sustainability. Technology. Virtual learning. Web application. Web development.

LISTA DE FIGURAS

Figura 3.1 – Usuários ativos de aplicativos de educação por ano. Fonte: Apps (2022) 12
Figura 3.2 – Mapa de jornada de usuário. Fonte: Acervo do autor. 17
Figura 3.3 – Fluxograma de navegação de usuário comum. Fonte: Acervo do autor. . 18
Figura 3.4 – Fluxograma de navegação de usuário administrador. Fonte: Acervo do autor. 18
Figura 3.5 – Logotipo *greenlearn*. Fonte: Acervo do autor. 20
Figura 3.6 – Paleta de cores do greenlearn. Fonte: Acervo do autor. 20
Figura 5.1 – Modelo do banco de dados da aplicação *greenlearn* desenvolvido no *DB-Designer*. Fonte: Acervo do autor. 24
Figura 5.2 – Hierarquia de diretórios e arquivos do *backend*. Fonte: Acervo do autor. 26
Figura 5.3 – Hierarquia de diretórios e arquivos do frontend do projeto *greenlearn*. Fonte: Acervo do autor. 56
Figura 5.4 – Captura de tela das páginas de cadastro e *login* do projeto *greenlearn*. Fonte: Acervo do autor. 58
Figura 5.5 – Captura de tela das páginas de categorias (*home*) e níveis do projeto *greenlearn*. Fonte: Acervo do autor. 59
Figura 5.6 – Captura de tela das páginas de perguntas e de nível em construção do projeto *greenlearn*. Fonte: Acervo do autor. 60
Figura 5.7 – Captura de tela da página de perguntas e suas variações de acordo com o envio de respostas. Fonte: Aplicação desenvolvida pelo autor. 61
Figura 5.8 – Captura de tela do ranking do projeto *greenlearn*. Fonte: Acervo do autor. 62
Figura 5.9 – Captura de tela do menu de administrador e da página de usuários do *greenlearn*. Fonte: Acervo do autor. 63
Figura 5.10 – Captura de tela das opções de adição de conteúdo do *greenlearn*. Fonte: Acervo do autor. 64
Figura 6.1 – Capturas de tela da aplicação *spacestudy*. Fonte: Acervo do autor. 67
Figura 6.2 – Resultado da pesquisa de satisfação do teste de aceitação. Fonte: Acervo do autor. 69
Figura 6.3 – Sugestões de disciplinas por parte dos participantes do teste de aceitação. Fonte: Acervo do autor. 70

LISTA DE CÓDIGOS

5.1 Configuração do servidor 27
5.2 Roteamento da aplicação 28
5.3 Rotas de autenticação - authRouter.js 29
5.4 *Schema* de cadastro - *signUpSchema.js* 30
5.5 *Middleware* de validação de *schemas* - *schemaValidator.js* 30
5.6 Função *signUp* - *authController.js* 31
5.7 Função *getUserByEmail* - *userRepository.js* 32
5.8 Função *createUser* - *userRepository.js* 32
5.9 *Schema* de *login* - *signInSchema.js* 33
5.10 Função *signIn* - *authController.js* 34
5.11 Função *createSession* - *sessionRepository.js* 34
5.12 Instância de roteamento *questionRouter* - *questionsRouter.js* 35
5.13 *Middleware* de autenticação - *validateAuth.js* 37
5.14 Função *getSessionByToken* - *sessionRepository.js* 38
5.15 Função *getCategories* - *questionsController.js* 38
5.16 Função *getCategories* - *questionsRepository.js* 38
5.17 Função *getLevels* - *questionsController.js* 39
5.18 Função *getCategoryById* - *questionsRepository.js* 39
5.19 Função *getLevels* - *questionsRepository.js* 39
5.20 Função *getQuestionsByCategoryAndLevel* - *questionsController.js* 40
5.21 Função *getQuestionsByCategoryAndLevel* - *questionsRepository.js* 41
5.22 *getQuestionsDoneByCategoryLevelAndUser* - *questionsRepository.js* 42
5.23 Função *getQuestionById* - *questionsController.js* 43
5.24 Função *getQuestionById* - *questionsRepository.js* 43
5.25 Função *getAnswersByQuestionId* - *questionsRepository.js* 44
5.26 Função *getProgressByUserAndQuestion* - *questionsRepository.js* 44
5.27 Função *postProgress* - *questionsController.js* 45
5.28 Função *postProgress* - *questionsRepository.js* 45
5.29 Instância de rotas de usuário - *userRouter.js* 46
5.30 Função controladora *getUsersProgress* - *usersController.js* 47
5.31 Função getUsersProgress - *userRepository.js* 47
5.32 *Middleware* de validação de permisões *validateAdm* - *admValidator.js* 48
5.33 Função controladora *postCategory* - *questionsController.js* 49
5.34 Função *getCategoryByName* - *questionsRepository.js* 49
5.35 Função *postCategory* - *questionsRepository.js* 50
5.36 *Schema* de perguntas - *questionSchema.js* 50
5.37 Função controladora *postQuestion* - *questionsController.js* 51
5.38 Função *postQuestion* - *questionsRepository.js* 52
5.39 Função *postAnswers* - *questionsRepository.js* 52
5.40 Função controladora *getUsers* - *usersController.js* 53
5.41 Função *getUsers* - *userRepository.js* 53
5.42 Função controladora *upgradePermition* - *usersController.js* 54
5.43 Função *upgradePermition* - *userRepository.js* 54

SUMÁRIO

1 INTRODUÇÃO 8
2 OBJETIVOS E JUSTIFICATIVAS 9
2.1 OBJETIVO GERAL 9
2.2 OBJETIVOS ESPECÍFICOS 9
2.3 JUSTIFICATIVA 10
3 REVISÃO DE LITERATURA 11
3.1 TECNOLOGIA COMO FERRAMENTA EDUCACIONAL 11
3.2 MODELAGEM DE PRODUTO 13
3.2.1 Interface e experiência de usuário 13
3.2.2 Análise de mercado 13
3.2.2.1 Duolingo 14
3.2.2.2 Perguntados 15
3.2.3 Decisões de projeto 15
3.2.3.1 Jornada do usuário e fluxo de navegação 16
3.2.3.2 Identidade visual 19
4 METODOLOGIA 22
5 DESENVOLVIMENTO DA APLICAÇÃO 23
5.1 BANCO DE DADOS 23
5.2 APLICAÇÃO BACKEND 25
5.2.1 Arquitetura de backend 25
5.2.2 Configuração e conexão 27
5.2.2.1 Configuração do servidor 27
5.2.2.2 Conexão com o banco de dados 28
5.2.3 Rotas de autenticação 29
5.2.3.1 Fluxo de cadastro 29
5.2.3.2 Fluxo de login 32
5.2.4 Fluxos de navegação 35
5.2.4.1 Listagem de categorias, níveis e perguntas 36
5.2.4.2 Busca de perguntas específicas 42
5.2.4.3 Registro de progresso 44
5.2.4.4 Ranking 46
5.2.5 Ferramentas administrativas 48
5.2.5.1 Gerenciamento de conteúdo 49
5.2.5.2 Gerenciamento de permissões de usuário 52
5.3 APLICAÇÃO FRONTEND 55
5.3.1 Arquitetura de frontend 55
5.3.2 Interface e navegação 58
6 RESULTADOS 65
6.1 TESTES EM AMBIENTE DE DESENVOLVIMENTO 65
6.2 TESTES EM AMBIENTE DE PRODUÇÃO 65
6.2.1 Teste de aceitação 65
6.2.2 Teste de *rollout* gradual 70
7 CONCLUSÃO 72
REFERÊNCIAS BIBLIOGRÁFICAS 73

1 INTRODUÇÃO

Com o avanço da tecnologia e a crescente demanda por soluções inovadoras na área da educação, surgem novas oportunidades para o desenvolvimento de aplicativos educacionais. Neste trabalho, é apresentado o processo de criação da aplicação *greenlearn*, um aplicativo educacional concebido com o objetivo de auxiliar o projeto de extensão "Energizando o Futuro", da Universidade Federal de Santa Maria (UFSM).

A ideia de criar o *greenlearn* surgiu a partir da necessidade de ampliar o alcance e o impacto do Energizando o Futuro, que busca disseminar conhecimentos sobre sustentabilidade e eficiência energética para a jovens estudantes de ensino fundamental e médio. Reconhecendo o potencial dos dispositivos móveis como ferramentas de aprendizagem acessíveis e interativas, decidiu-se desenvolver um aplicativo que pudesse complementar as atividades presenciais do projeto.

O processo de criação do *greenlearn* envolveu diversas etapas, desde a pesquisa de mercado até a implementação do aplicativo em si. Inicialmente, realizou-se uma pesquisa detalhada sobre o mercado de aplicativos educacionais, analisando tendências, necessidades dos usuários e concorrência existente. Essa pesquisa embasou as decisões tomadas no projeto, permitindo a identificação de oportunidades e o desenvolvimento de um produto capaz de atender o público alvo.

Além disso, foram considerados aspectos de usabilidade e experiência do usuário (UI/UX) durante todo o processo de desenvolvimento do *greenlearn*. A interface do aplicativo foi projetada de forma intuitiva e atrativa, visando proporcionar uma experiência agradável aos usuários e facilitar a navegação e interação com os conteúdos educacionais oferecidos.

O desenvolvimento do aplicativo foi realizado utilizando boas práticas de programação e tecnologias adequadas ao contexto, visando garantir estabilidade, segurança e desempenho. Foram empregados métodos ágeis de desenvolvimento, permitindo uma maior flexibilidade e adaptabilidade às demandas do projeto.

Para avaliar a eficácia e a aceitação do *greenlearn*, foi realizado um teste de aceitação envolvendo um grupo controlado de usuários. Essa etapa foi fundamental para identificar possíveis melhorias, coletar avaliações e validar a proposta do aplicativo.

Ao longo deste trabalho, serão abordadas de forma detalhada cada uma dessas etapas, destacando os desafios enfrentados, as soluções adotadas e os resultados obtidos. O objetivo é apresentar um panorama completo do processo de criação do aplicativo *greenlearn*, evidenciando seu potencial como ferramenta educacional e seu papel no projeto "Energizando o Futuro"da UFSM.

2 OBJETIVOS E JUSTIFICATIVAS

Nesta fase, decisões e abordagens são definidas para o desenvolvimento do projeto, tais como objetivos geral e específicos, assim como a justificativa por trás de sua realização.

2.1 OBJETIVO GERAL

O trabalho tem por objetivo geral o projeto de um aplicativo que atue como ferramenta educacional auxiliar para o projeto de extensão Energizando o Futuro, da Universidade Federal de Santa Maria. A aplicação tem como finalidade fortalecer a compreensão e fixação do conteúdo abordado nos cursos e aulas ministrados em escolas de ensino fundamental e médio.

2.2 OBJETIVOS ESPECÍFICOS

Os seguintes objetivos específicos foram definidos para o projeto:

- Definição de funcionalidades da aplicação;
- Modelagem e população do banco de dados;
- Definição e construção das rotas do *backend*;
- Definição e construção das páginas de *frontend*;
- Testes em ambiente de desenvolvimento;
- Testes em ambiente de produção com coleta de avaliações de usuários através de uma versão paralela simplificada, voltada para a disciplina de Projeto Conceitual de Sistemas Espaciais do curso de Engenharia Aeroespacial da Universidade Federal de Santa Maria.

2.3 JUSTIFICATIVA

A presente pesquisa propõe o desenvolvimento da aplicação *greenlearn* como tema de Trabalho de Conclusão de Curso com base na necessidade de promover a educação em sustentabilidade e energias renováveis. A escolha desse tema se deve à importância crescente de conscientização e adoção de práticas sustentáveis no contexto atual. Além disso, a aplicação visa preencher uma lacuna existente no campo da educação, proporcionando uma abordagem inovadora e interativa para o ensino desses temas, por meio de recursos digitais e tecnológicos, atuando como uma valiosa adição à didática do projeto de extensão Energizando o Futuro. A relevância da pesquisa está na possibilidade de oferecer uma ferramenta educacional eficaz e acessível, que contribua para a formação de indivíduos conscientes e engajados com tecnologia e questões ambientais.

3 REVISÃO DE LITERATURA

Nos últimos anos, tem-se testemunhado uma rápida evolução tecnológica, que tem afetado positivamente diversos setores da sociedade, incluindo a educação. A psicologia da educação, um campo de estudo que busca compreender os processos mentais envolvidos na aprendizagem e no ensino, tem sido uma importante área de pesquisa na exploração do potencial da tecnologia na educação. Neste contexto, os aplicativos educacionais têm surgido como ferramentas inovadoras que podem revolucionar a forma como aprendemos e ensinamos.

3.1 TECNOLOGIA COMO FERRAMENTA EDUCACIONAL

A psicologia da educação destaca a importância da individualização do ensino, reconhecendo que cada aluno possui características, ritmos e estilos de aprendizagem diferentes. Nesse sentido, os aplicativos educacionais oferecem a possibilidade de personalizar o processo de aprendizagem, adaptando-se às necessidades de cada estudante. Estudos, como o proposto por Hattie (2009), enfatizam que a personalização aumenta a eficácia do ensino, melhorando os resultados acadêmicos e a motivação dos alunos. Nessa obra, Hattie apresenta uma lista de influências educacionais ordenadas por sua eficácia relativa, destacando fatores que têm um impacto significativo no processo de ensino-aprendizagem.

Outros aspectos cruciais no processo de aprendizagem são o engajamento e a motivação dos estudantes. De acordo com Reeve (2009), é fundamental que os professores adotem um estilo motivacional que promova a autonomia dos alunos ao invés a um estilo controlador. A pesquisa destaca a importância de os educadores se tornarem mais favoráveis à autonomia dos estudantes, fornecendo apoio e encorajamento, evitando a imposição de pressão ou controle excessivo. Essa abordagem mais autônoma de motivação por parte dos professores tem sido associada a um maior engajamento e motivação intrínseca por parte dos alunos, produzindo melhores resultados acadêmicos, maior satisfação com o processo de aprendizagem, além de aumento na propensão a aprender e a reter o conhecimento. Os aplicativos educacionais oferecem recursos interativos, jogos educativos, recompensas e desafios que despertam o interesse e incentivam a participação ativa dos alunos.

Além disso, torna-se relevante citar a teoria da aprendizagem significativa (AUSUBEL, 1963), que enfatiza a importância de relacionar novos conhecimentos com conceitos já existentes na mente do aluno. Os aplicativos educacionais têm o potencial de proporcionar experiências de aprendizagem significativas, por meio de recursos visuais, interativos e multimídia, que facilitam a compreensão e a assimilação do conteúdo. De acordo com

Novak e Gowin (1984), a aprendizagem significativa está diretamente ligada à retenção de conhecimento e à transferência de habilidades para situações reais.

Por fim, é válido ressaltar que a tecnologia, aliada aos aplicativos educacionais, permite que os estudantes acessem uma quantidade vasta de informações e recursos educacionais, proporcionando um aprendizado contínuo fora da sala de aula. Estudos, como o de Dweck (2006), enfatizam a importância do aprendizado ao longo da vida e destacam que a mentalidade de crescimento e o hábito de aprender constantemente são fundamentais para o sucesso acadêmico e profissional; e os aplicativos educacionais facilitam esse processo, fornecendo acesso a materiais atualizados, cursos online, tutoriais e comunidades de aprendizagem.

Um levantamento feito pela *Business of Apps* - uma plataforma online que fornece informações e percepções de análises de mercado, estatísticas, tendências, estudos de caso e notícias relevantes sobre mercado dos aplicativos - indica uma evolução na quantidade de usuários de aplicativos de educação ao longo dos anos, como mostra o gráfico na Figura 3.1.

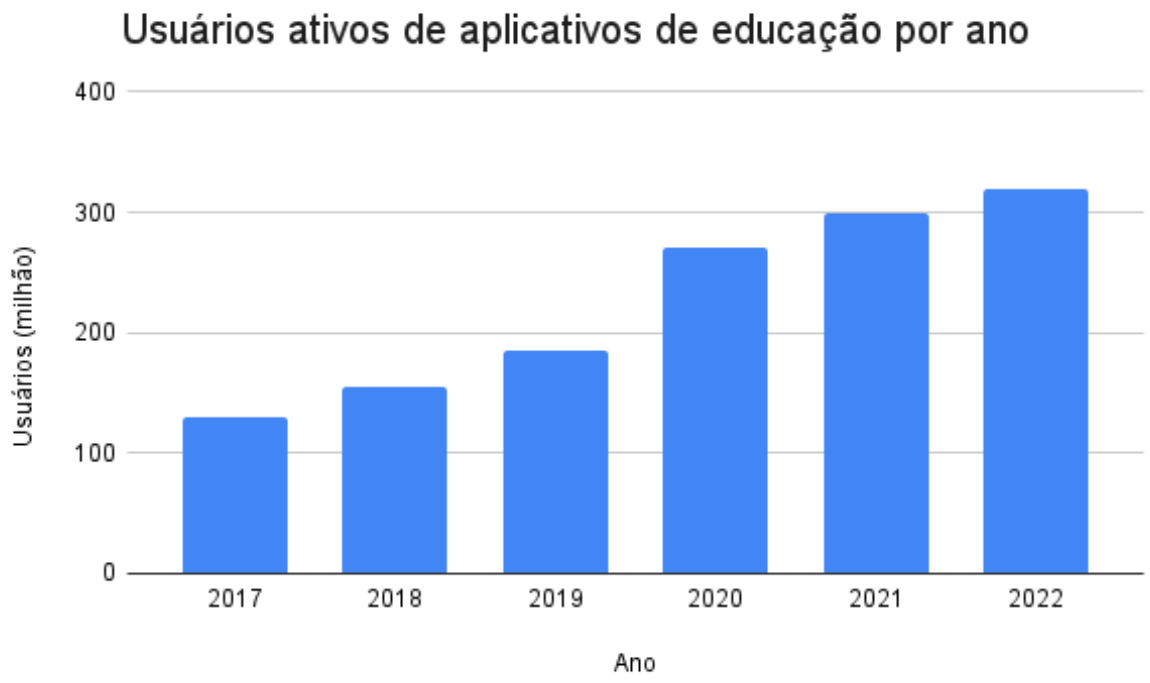


Figura 3.1 – Usuários ativos de aplicativos de educação por ano. Fonte: Apps (2022)

É possível notar um salto significativo na quantidade de usuários no ano de 2020, em função da grande digitalização de processos causada pela pandemia de COVID-19. Porém, mesmo após o fim das políticas de confinamento, não houve regressão no número de usuários ativos. Pelo contrário, a quantidade de usuários continuou a crescer, indicando que, embora essa parcela dos usuários tenha começado a utilizar as aplicações pela necessidade imposta pela mudança de rotina, as vantagens trazidas pelas ferramentas se

mostraram úteis o suficiente para provocar sua retenção.

3.2 MODELAGEM DE PRODUTO

Previamente ao início do processo de desenvolvimento da aplicação, fez-se necessário a definição do formato da mesma, dando-se início a um estudo sobre interface e experiência de usuário (UI/UX).

3.2.1 Interface e experiência de usuário

O *design* de interface e a experiência do usuário têm um impacto significativo na eficácia e no sucesso das aplicações *web* e devem ser intuitivos, agradáveis, eficientes e acessíveis.

Estudos, como os de Nielsen e Norman (1993) e de Schneiderman (1998) destacam a importância da usabilidade na interface de um sistema. A usabilidade refere-se à facilidade com que os usuários podem interagir com a aplicação, realizar tarefas e atingir seus objetivos de forma eficiente. Um design de interface bem estruturado e uma UX (experiência do usuário) cuidadosamente projetada contribuem para aprimorar a usabilidade, garantindo que os usuários encontrem facilmente o que precisam e executem suas ações de forma intuitiva.

Ademais, segundo Norman (2013), é importante existir um *design* centrado no usuário, que leve em consideração suas necessidades, expectativas e características durante todo o processo. Ao compreender o público-alvo da aplicação *web*, é possível criar uma interface e projetar uma experiência de usuário que atendam às suas preferências, tornando a experiência de uso mais personalizada e relevante.

Por fim, a existência de *feedback* contínuo, por meio de elementos visuais, ajuda os usuários a entenderem o estado da aplicação e a obterem confirmação de suas ações, aumentando os níveis de satisfação que, de acordo com Hassenzahl e Tractinsky (2006), desempenham um papel fundamental na retenção e fidelização dos usuários.

3.2.2 Análise de mercado

Durante o processo criativo do desenvolvimento de um novo produto, faz-se necessário compreender a situação atual do mercado no segmento específico em que se pretende entrar. Existem diversas ferramentas que podem auxiliar nesse proceso, dentre as quais, destaca-se o *benchmarking*.

Segundo McNair e Scriven (2002), o *benchmarking* é uma técnica que consiste em analisar e comparar o desempenho, os processos, as práticas e os resultados de uma organização com outras organizações líderes no mesmo setor. No contexto do desenvolvimento de uma nova aplicação *web*, o *benchmarking* é uma ferramenta valiosa para avaliar o desempenho de outras aplicações semelhantes, identificar melhores práticas e buscar inspiração para aprimorar a própria aplicação. Através da análise de outras aplicações líderes no mercado no nicho desejado, é possível identificar referências e tendências, melhorar a usabilidade e experiência do usuário, avaliar e comparar performance e eficiência e compreender as preferências, expectativas e necessidades do público alvo.

Segue um estudo das principais referências de marcado para aplicações *web* educacionais utilizadas.

3.2.2.1 Duolingo

O *Duolingo* é uma aplicação *web* e móvel voltada para o aprendizado de idiomas. Ela oferece uma abordagem "gamificada" e interativa, com exercícios de vocabulário, gramática, compreensão auditiva e leitura. Embora possua uma gama ampla e diversificada de perfis de usuário, o *Duolingo* foi escolhido como uma das principais referências devido à sua abordagem baseada em elementos de jogos, capaz de atrair e conversar muito bem com um público mais jovem, como adolescentes e jovens adultos.

Alguns aspectos do *Duolingo* podem ser destacados como pontos de referência para o desenvolvimento de uma nova aplicação web educacional:

1. Design intuitivo e envolvente: O *Duolingo* possui uma interface intuitiva e amigável, com elementos visuais atraentes e cores vibrantes. O design cuidadoso contribui para uma experiência de usuário envolvente, tornando o aprendizado mais agradável e motivador;

2. Personalização e progresso: A aplicação oferece um sistema de acompanhamento do progresso do usuário, permitindo que ele defina metas e visualize seu avanço. Também oferece sugestões personalizadas de atividades com base no nível de proficiência do usuário, adaptando-se ao seu ritmo de aprendizado;

3. "Gamificação" e recompensas: O *Duolingo* utiliza elementos de jogos para motivar os usuários, oferecendo recompensas virtuais, como pontos, níveis e conquistas desbloqueáveis. Essa abordagem lúdica cria um senso de conquista e incentiva a prática constante;

4. Exercícios interativos: A aplicação apresenta exercícios como preencher lacunas, escolher a palavra correta, pronunciar frases e traduzir textos. Essa interatividade

mantém o aprendizado interessante e desafiador.

3.2.2.2 Perguntados

O Perguntados é um aplicativo de perguntas e respostas que oferece uma experiência de aprendizado em jogo interativa e divertida. O aplicativo atrai principalmente usuários jovens e adultos que estão em busca de uma experiência divertida e educacional. No entanto, devido à sua natureza de jogo interativo, o Perguntados também pode atrair usuários mais jovens, desde que sejam capazes de compreender as perguntas e participar ativamente do jogo.

Assim como o *Duolingo*, o Perguntados apresenta pontos fortes que podem servir como referência para a construção de uma nova aplicação web, são eles:

1. Engajamento e diversão: O Perguntados é conhecido por oferecer uma experiência de jogo envolvente e divertida, o que contribui para o engajamento dos usuários. Ao transformar o aprendizado em uma atividade lúdica, o aplicativo torna o processo educacional mais agradável e motivador;

2. Aprendizado baseado em perguntas: O Perguntados utiliza um formato de pergunta e resposta para fornecer informações e conhecimentos aos usuários. Esse formato permite que os usuários testem seus conhecimentos, identifiquem lacunas em sua compreensão e aprendam com os *feedbacks* fornecidos nas respostas corretas e incorretas;

3. Variedade de categorias temáticas: O aplicativo oferece uma ampla gama de categorias temáticas, abrangendo áreas como cultura geral, ciências, história, esportes, entretenimento e muito mais. Essa variedade permite que os usuários explorem diferentes tópicos e expandam seus conhecimentos em diversas áreas;

4. Competição amigável: O Perguntados incorpora um elemento competitivo por meio de desafios entre amigos ou jogadores aleatórios. Essa competição amigável estimula os usuários a se envolverem mais com o aplicativo, compararem seu desempenho e buscarem aprimorar seus conhecimentos.

3.2.3 Decisões de projeto

Considerando o objetivo de desenvolver uma aplicação web para o projeto de extensão "Energizando o Futuro", com foco na sustentabilidade, foi realizada uma análise

das características mais relevantes encontradas nos aplicativos Duolingo e Perguntados por meio do processo de *benchmarking* anteriormente apresentado.

Durante essa análise, buscou-se identificar os pontos fortes e as melhores práticas presentes nos aplicativos mencionados. Foram considerados aspectos como integração de elementos de jogos, aprendizado baseado em perguntas, diversidade temática, interação social e *design* intuitivo direcionado ao tema proposto e ao perfil dos usuários.

Ao incorporar essas características no planejamento e desenvolvimento do *greenlern*, pretende-se proporcionar uma experiência de aprendizagem envolvente e condizente com o público alvo do projeto de extensão, constituído por adolescentes e pré-adolescentes dos ensinos fundamental II e médio.

Acredita-se que ao utilizar elementos como a "gamificação", que transforma o aprendizado em uma atividade lúdica, e a ideia do aprendizado baseado em perguntas, apresentada na forma de preenchimento de lacunas, que permite aos usuários testar seus conhecimentos e receber *feedback* imediato, será possível gerar engajamento e incentivar o aprofundamento nas temáticas de sustentabilidade e energias alternativas.

Além disso, a interação social e competição amigável serão exploradas através de um ranking de pontuações, a fim de criar um ambiente interessante e com elementos capazes de gerar motivação e retenção de usuários.

Portanto, busca-se embasar o desenvolvimento do *greenlearn* de forma a cumprir as decisões acima definidas, visando a construção de uma jornada de usuário capaz de oferecer uma experiência educacional significativa e alinhada aos propósitos do projeto "Energizando o Futuro".

3.2.3.1 Jornada do usuário e fluxo de navegação

A jornada do usuário é o conjunto de etapas ou interações que um usuário realiza ao interagir com um produto, serviço ou sistema. Ao mapear a jornada do usuário, é possível identificar pontos de fricção, oportunidades de melhoria e obter percepções valiosas para o *design* e aprimoramento do produto.

De acordo com Garrett (2002), a elaboração de um mapa de jornada de usuário ajuda a compreender melhor suas necessidades, expectativas, motivações e emoções em cada etapa, possibilitando o projeto de uma jornada mais envolvente, personalizada e significativa, visando proporcionar uma experiência memorável e satisfatória aos usuários. A Figura 3.2 apresenta o mapa de jornada de usuário desenvolvido para o *greenlearn*.

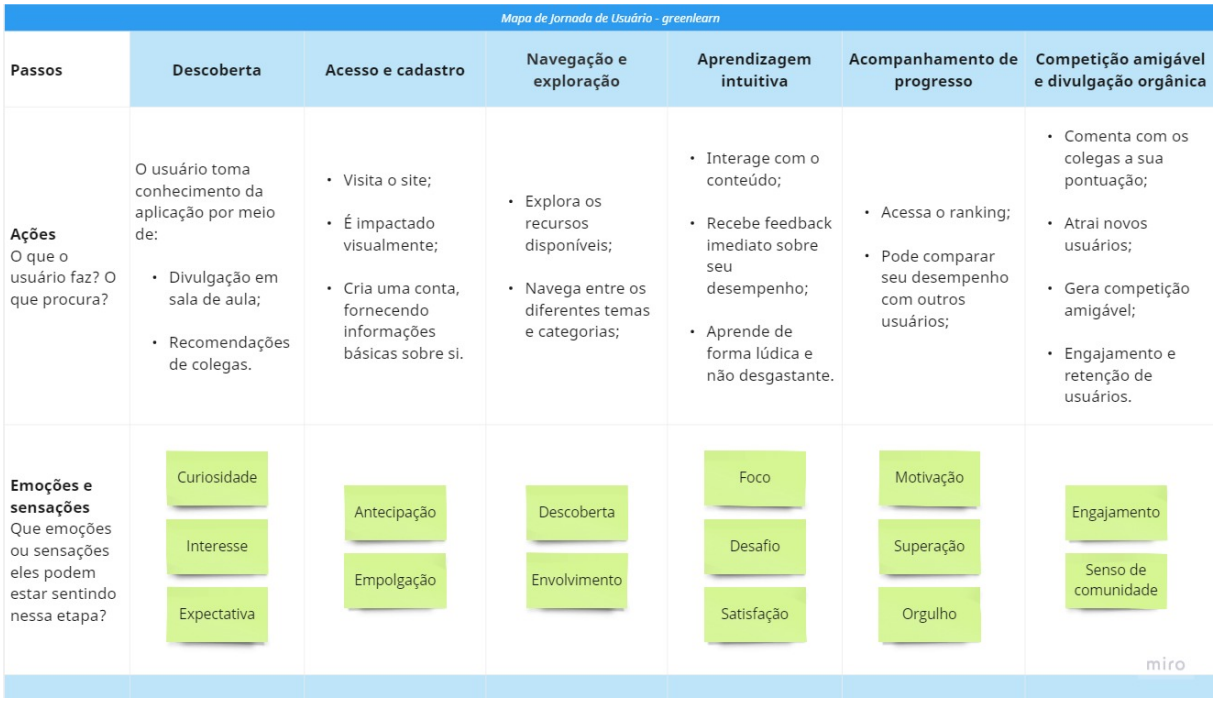


Figura 3.2 – Mapa de jornada de usuário. Fonte: Acervo do autor.

Com base nos passos e emoções levantados pelo mapa de jornada, é possível construir um fluxo de navegação que supra as necessidades do usuário. O fluxo de navegação determina como o usuário se desloca pela aplicação, quais caminhos são possíveis de seguir e como as informações estão organizadas, analisando a estrutura lógica de interação da aplicação. A Figura 3.3 mostra o fluxograma de navegação do usuário padrão ao utilizar o *greenlearn*.

Com o objetivo de dar mais independência para a aplicação, para que os membros do projeto Energizando o Futuro consigam realizar alterações sem a necessidade dos serviços de um desenvolvedor, fez-se necessário a construção de ferramentas específicas para usuários administradores e, portanto, a construção de um fluxo de navegação próprio. O fluxograma de navegação do usuário administrador é apresentado na Figura 3.4.

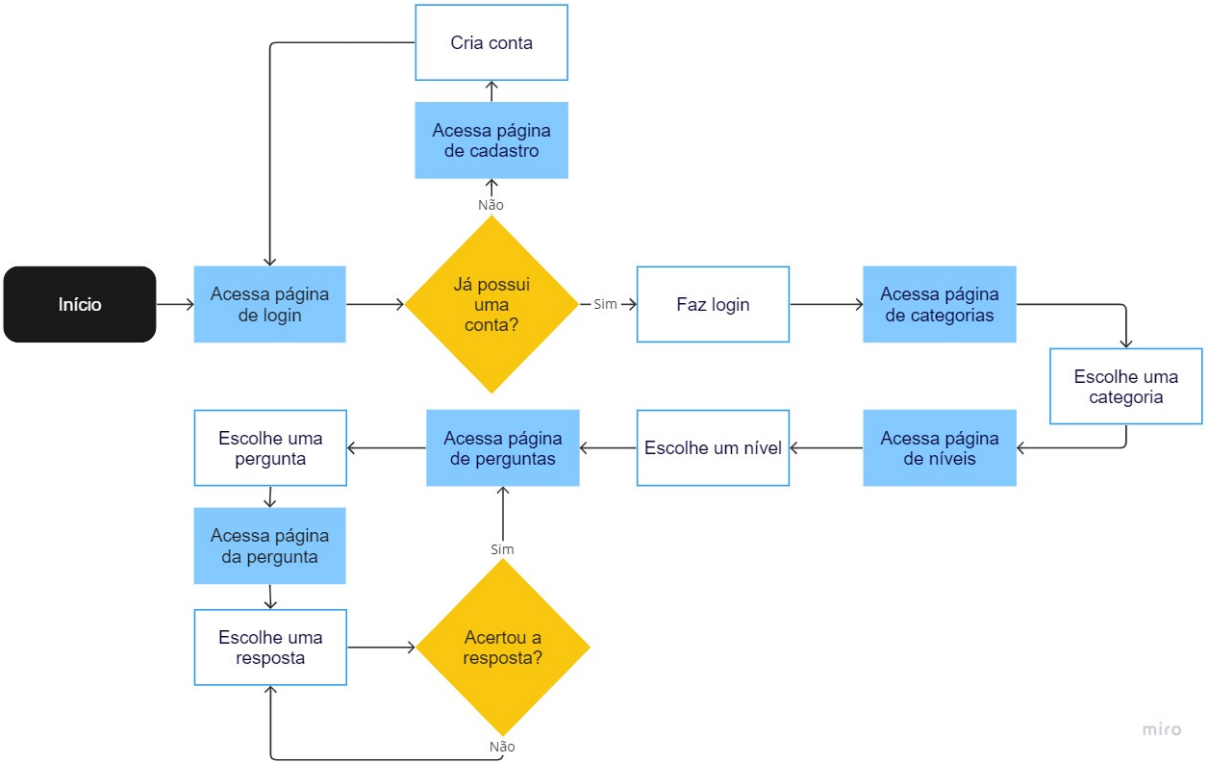


Figura 3.3 – Fluxograma de navegação de usuário comum. Fonte: Acervo do autor.

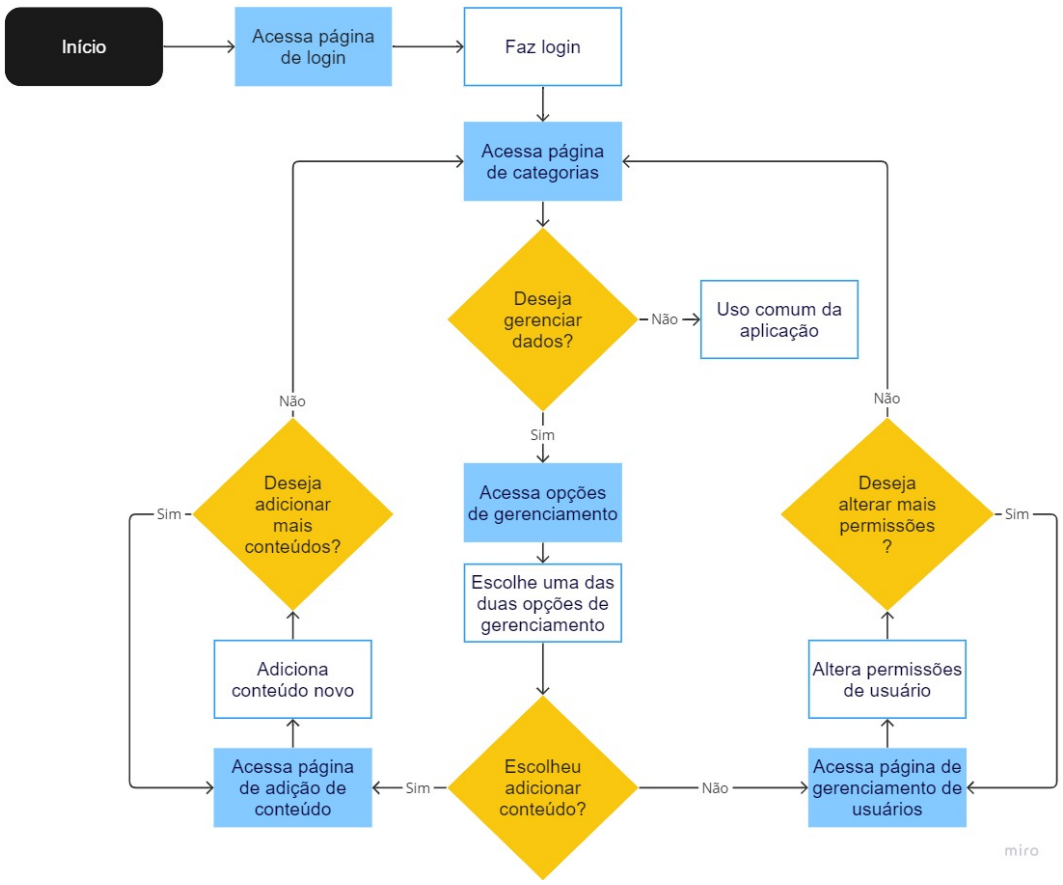


Figura 3.4 – Fluxograma de navegação de usuário administrador. Fonte: Acervo do autor.

Uma vez que as funcionalidades foram definidas em um fluxo claro, outro aspecto fundamental para experiência do usuário e de suma importância para a criação de uma interface gráfica pode ser definido: a identidade visual.

3.2.3.2 Identidade visual

A identidade visual de um produto é o conjunto de elementos visuais que representam e comunicam sua essência e personalidade. Ela é responsável por criar uma imagem consistente e reconhecível, transmitindo a mensagem desejada por meio de elementos visuais como logotipos, cores, tipografia, ícones e elementos gráficos.

O *branding* e posicionamento desempenham um papel fundamental na identidade visual, sendo essenciais para estabelecer a identidade da marca, diferenciá-la da concorrência e criar conexões significativas com o público-alvo. Segundo Kotler (2000), *branding* é um conceito amplo e multidimensional que abrange a gestão estratégica da marca, visando posicioná-la de forma distintiva no mercado, influenciando a percepção dos consumidores e gerando fidelidade ao longo do tempo.

Um dos principais elementos englobados pelos conceitos de *branding* e posicionamento é o nome da marca, que deve estar alinhado com a identidade da empresa e/ou produto e seu posicionamento estratégico. Portanto, deve comunicar os valores, personalidade e propósito da marca de forma consistente e coerente com a mensagem que se deseja transmitir.

O nome definido para a aplicação foi *"greenlearn"*. A palavra "*green*", em inglês, significa "verde", sendo frequentemente associado à natureza, ecologia e sustentabilidade. Essa cor é simbólica e representa a harmonia com o meio ambiente, a preservação dos recursos naturais e a conscientização sobre questões ambientais. Ao utilizar "*green*" no nome da aplicação, é possível evocar visualmente a ideia de um ambiente voltado para a sustentabilidade. Já o termo "*learn*", também em inglês, significa "aprender" e é uma palavra simples e direta que reforça o principal objetivo da aplicação: proporcionar uma experiência de aprendizado significativa sobre sustentabilidade e energias renováveis. Portanto, a combinação dos termos "green" e "learn" no nome cria uma sinergia entre os aspectos ambientais e educacionais.

Em relação a tipografia, foi escolhida a fonte *Roboto*, por apresentar um *design* limpo e moderno, com linhas retas e proporções equilibradas, transmitindo uma sensação de atualidade, modernidade, clareza e legibilidade. O logotipo, que carrega o próprio nome da aplicação por motivos de fixação por repetição visual, pode ser conferido na Figura 3.5.

greenlearn

Figura 3.5 – Logotipo *greenlearn*. Fonte: Acervo do autor.

Outra etapa essencial do processo de construção da identidade visual é o estudo das cores e sua influência psicológica. Estudos importantes na área de psicologia das cores, como os propostos por Itten (1970) e por Birren (1978), enfatizam a conexão entre as cores e as emoções humanas. Ao selecionar a paleta de cores para a aplicação, é fundamental entender seu impacto no engajamento dos estudantes e garantir a transmissão da mensagem desejada.

A paleta de cores escolhida é apresentada na Figura 3.6, sendo cuidadosamente selecionada pensando na conexão a ser estabelecida com o usuário, de forma passar mensagens relacionadas ao tema ou gerar emoções e sensações que tragam efeitos positivos para o processo de aprendizagem.

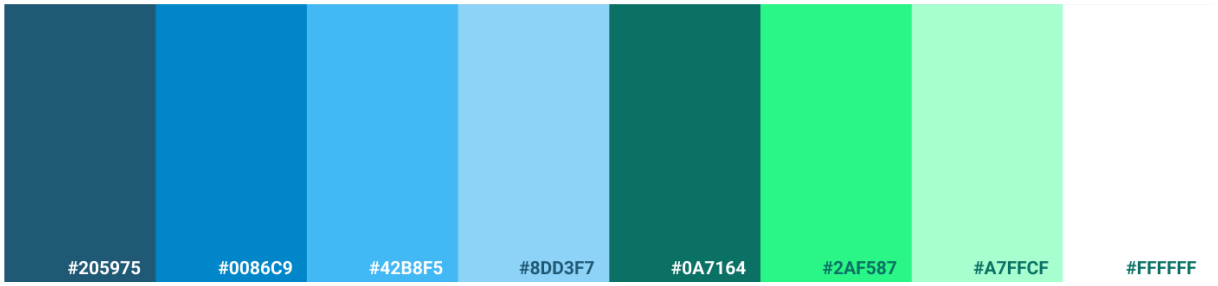


Figura 3.6 – Paleta de cores do greenlearn. Fonte: Acervo do autor.

As cores estão individualmente descritas na lista abaixo, assim como seus respectivos motivos de escolha. As cores referente aos códigos hexadecimais podem ser visualmente verificadas na Figura 3.6.

- #205975: Este tom de azul escuro passa a ideia de estabilidade e seriedade, sendo ideal para transmitir confiança e segurança aos estudantes;
- #0086C9: O azul intenso e vibrante é associado à comunicação eficiente e transmite uma sensação de autoridade, podendo ser utilizado para capturar a atenção dos estudantes e incentivar a exploração de conteúdos relevantes;
- #42B8F5: Um azul fresco e energético, associado à criatividade. Pode ser utilizado para criar um ambiente agradável e amigável, propenso ao surgimento de novas ideias;
- #8DD3F7: Essa tonalidade de azul, mais clara e suave, atua criando uma atmosfera calma e acolhedora, estimulando a concentração e o aprendizado;

- #0A7164: Esse tom de verde profundo está relacionado a equilíbrio, crescimento e sustentabilidade, destacando tópicos relacionados ao meio ambiente e transmitindo uma sensação de responsabilidade e consciência ambiental;
- #2AF587: Um verde mais claro e vibrante, que transmite otimismo e positividade. Pode ser usado para representar a inovação nas energias renováveis e estimular a curiosidade dos estudantes, incentivando a exploração de soluções sustentáveis;
- #A7FFCF: Esse tom de verde claro e suave transmite tranquilidade e serenidade. Pode ser utilizado para criar uma atmosfera relaxante, tornando o ambiente virtual mais agradável e convidativo para os estudantes;
- #FFFFFF: O branco é uma cor neutra que representa simplicidade e clareza, trazendo uma sensação de organização e limpeza que conversa perfeitamente com um ambiente de aprendizado.

Em posse de um fluxograma de navegação bem definido e dos principais elementos da identidade visual do produto, tem-se uma base sólida para a criação de uma experiência agradável, visualmente atraente e coesa, dando conclusão as etapas que, no presente trabalho, precedem o desenvolvimento propriamente dito da aplicação.

4 METODOLOGIA

O presente trabalho relata o processo de desenvolvimento de um aplicação *web* para fins educacionais e, portanto, engloba de forma descritiva as etapas que o envolvem.

Inicialmente, foi realizado o estudo da situação atual mercado para tecnologias que atuam como parte de processos educativos, em especial, aplicações *web*. De forma complementar, realizou-se uma análise dos impactos de tais ferramentas no processo de aprendizagem, baseando-se em estudos de psicologia da educação, que se mostraram de grande pertinência.

Subsequentemente, efetuou-se uma busca por formatos de maior relevância para a estrutura da aplicação, conhecendo práticas bem sucedidas empregadas por entidades que já possuem conhecimento de mercado estabelecido. Para esse propósito, o autor faz uso do *benchmarking*, que - segundo Cotter (1997) - é um conceito retirado do mundo dos negócios que tem ganhado importância positiva crítica em ambientes educacionais.

Uma vez que os dados coletados foram analisados, definiu-se o formato da aplicação através das etapas de estudo e elaboração de jornada do usuário e identidade visual, a fim de realizar um projeto de interface e experiência do usuário (UI/UX) em conformidade com os conceitos antepostos para a aplicação, que se encaixe nos tópicos educação e sustentabilidade anteriormente propostos e que disponha de formas acessíveis de implementação e coleta de resultados.

Em seguida, deu-se início ao desenvolvimento propriamente dito da aplicação, onde ocorreu a esquematização e construção do banco de dados, e um levantamento das informações necessárias para que a estrutura anteriormente definida possa ser executada, dispondo de todas as suas funcionalidades. Por conseguinte, iniciou-se a construção do servidor, responsável por fazer a comunicação entre a interface e o banco de dados, e da interface em si, linha de frente da aplicação e ponto de contato direto com o usuário. O fluxo de trabalho utilizado é conhecido no meio da engenharia de *software* como desenvolvimento em fatias, e consiste no avanço intercalado do *backend* e do *frontend* da aplicação, tendo por objetivo produzir funcionalidades completas ao longo do processo de desenvolvimento, entregando valor de forma contínua.

Ao finalizar o desenvolvimento da versão inicial da aplicação, iniciou-se a fase de testes e coleta de dados. A etapa englobou a disponibilização da aplicação para um público limitado, responsável por trazer opiniões e criticas que atuam como um retorno de sucesso do aplicativo. Por fim, ocorreu o período de análise de resultados, onde o retorno obtido na etapa anterior foi estudado e interpretado, revelando a viabilidade do projeto.

5 DESENVOLVIMENTO DA APLICAÇÃO

A seguir, será discutido o processo de desenvolvimento da aplicação, abordando todos os elementos que o compõem, assim como ferramentas utilizadas e seu funcionamento.

5.1 BANCO DE DADOS

Para a criação do banco de dados consumido no presente projeto, foi utilizado o *PostgreSQL*. Segundo The PostgreSQL Global Development Group (2021), o *PostgreSQL* é um sistema de gerenciamento de banco de dados objeto-relacional (SGBDR) capaz de oferecer muitas funcionalidades modernas, tais como, consultas complexas, chaves estrangeiras, gatilhos, visões atualizáveis, integridade transacional e controle de concorrência multiversão. Além disso, suporta uma ampla gama de tipos de dados, incluindo tipos personalizados definidos pelo usuário, o que o torna altamente flexível e adaptável às necessidades específicas de diferentes aplicações. Como a ferramenta possui licença liberal, pode ser utilizada livre de encargos por qualquer pessoa e para qualquer finalidade, seja particular, comercial ou acadêmica.

Por se tratar de um sistema de gerenciamento de bancos de dados relacionais, tem como função gerenciar dados armazenados em relações, isto é, tabelas. Cada tabela é uma coleção nomeada de linhas e cada linha da tabela possui o mesmo conjunto de colunas nomeadas, sendo cada coluna um tipo de dado específico.

As tabelas são agrupadas em bancos de dados, e uma coleção de bancos de dados gerenciados por uma única instância do servidor *PostgreSQL* forma um *cluster*.

O formato das tabelas, bem como os tipos de dados que nelas são armazenados e suas relações, variam de acordo com o objetivo para o qual o banco é desenvolvido. O processo de definição dos formatos das tabelas é chamado de modelagem do banco de dados, e interações com bancos relacionais criados com o *PostgreSQL* utilizam *Structured Query Language (SQL)*.

Existem ferramentas gráficas que auxiliam de forma visual na modelagem dos bancos de dados, entre elas, pode-se citar o *DBDesigner*, que permite aos usuários criar, editar e gerenciar esquemas de bancos de dados de forma intuitiva, oferecendo recursos poderosos para a criação de diagramas ER (Entidade-Relacionamento). A Figura 5.1 mostra o esquema do banco de dados desenvolvido no *DBDesigner* para a presente aplicação.

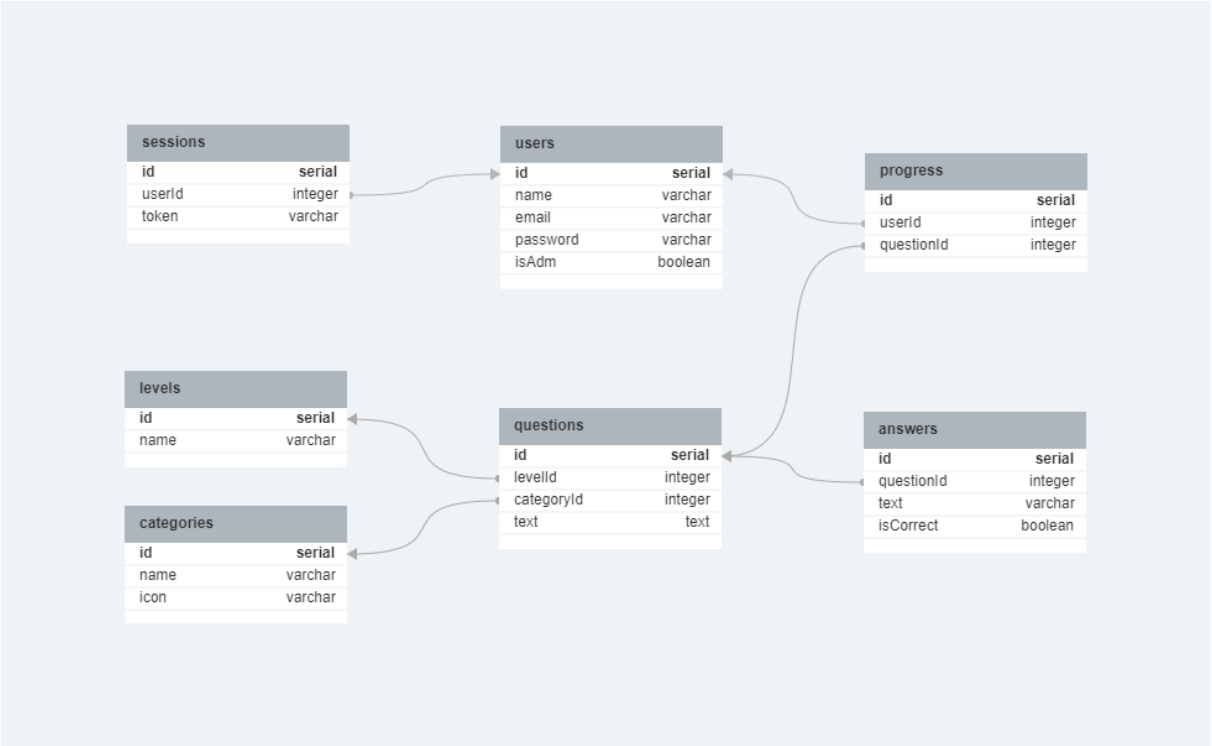


Figura 5.1 – Modelo do banco de dados da aplicação *greenlearn* desenvolvido no *DBDesigner*. Fonte: Acervo do autor.

Como mostra a Figura 5.1, foram criadas sete tabelas que, juntas, gerenciam todos os dados necessários para o funcionamento da aplicação. Todas as tabelas contam com uma coluna *id* do tipo *serial* responsável por identificar de forma única cada conjunto de dados inserido em uma linha.

A tabela *users* é responsável por armazenar os dados dos usuários cadastrados na aplicação. Para isso, conta com as colunas *name*, armazenando o nome do usuário; *email*, que armazena seu e-mail e possui a propriedade *unique*, o que impede a inserção de dados em duplicidade; *password*, que, por motivos de segurança, armazena uma versão criptografada da senha do usuário; e *isAdm*, que armazena um valor booleano, encarregado de identificar se é um usuário comum ou um administrador.

O banco conta com a tabela *sessions*, que se responsabiliza por gerenciar as sessões ativas dos usuários. Para o desempenho dessa função, possui a coluna *userId*, que é uma chave estrangeira vinculada a culuna *id* da tabela *users*, tendo por função identificar a qual usuário aquela sessão pertence; e contém a coluna *token*, que armazena uma string aleatória gerada no momento de início da sessão, responsável por conceder permissões de autenticação ao usuário.

As tabelas *levels* e *categories* armazenam, respectivamente, os níveis de dificuldade e as categorias que podem existir na aplicação. Cada uma conta com a coluna *name*, responsável por armazenar o nome do nível ou da categoria. Além disso, a tabela *categories* possui a coluna *icon* que armazena uma *string* de identificação de um ícone da

biblioteca *ion-icons* a ser usado para representar a categoria.

A tabela *questions* armazena todas as perguntas presentes na aplicação e possui as colunas *categoryId* e *levelId*, que são chaves estrangeiras que apontam para as colunas *id* das tabelas *categories* e *levels*, respectivamente, a fim de identificar a categoria e o nível aos quais a pergunta pertence. Conta, ainda, com a coluna *text*, que armazena o texto da pergunta.

Para armazenar as respostas de cada pergunta, tem-se a tabela *answers*. Esta conta com a coluna *questionId*, uma chave estrangeira que aponta para o *id* da tabela *questions*, com o objetivo de identificar a qual pergunta aquela resposta pertence. Ainda contém as colunas *text*, que guarda o texto da resposta, e *isCorrect*, que armazena um booleano incubido de identificar se é a resposta correta.

Por fim, para gerenciar o progresso do usuário na aplicação, existe a tabela *progress*. Esta conta com duas chaves extrangeiras, sendo a primeira a coluna *userId*, que aponta para o *id* do usuário na tabela *users*, e a segunda a coluna *questionId*, que está vinculada ao *id* da pergunta na tabela *questions*. Dessa forma, é possível identificar quais questões cada usuário já respondeu.

O banco de testes foi criado na máquina do autor e o banco de produção foi criado utilizando o plano estudantil da extensão *Heroku Postgres*, da plataforma *Heroku*.

5.2 APLICAÇÃO BACKEND

A aplicação *backend* é responsável por processar e gerenciar as funcionalidades lógicas e de dados de uma aplicação, sendo a camada do sistema que lida com o processamento das requisições dos usuários, a manipulação e organização dos dados, a comunicação com bancos de dados e a implementação das regras de negócio. Para o presente projeto, trata-se de uma API (*Application Programming Interface*) construída utilizando *Node.js*, um ambiente de execução de *JavaScript* assíncrono projetado para o desenvolvimento de aplicações escaláveis de rede. (Open JS Foundation, 2021)

Com o intuito de otimizar a construção da API, foi utilizado o *Express.js*, um *framework* de desenvolvimento *web* minimalista e flexível, baseado em *Node.js*, que facilita a criação de aplicações robustas e escaláveis, fornecendo um conjunto de recursos e funcionalidades para lidar com rotas, *middlewares* e requisições HTTP.

5.2.1 Arquitetura de backend

A arquitetura de backend empregada na aplicação utiliza *Repository Pattern*, um dos padrões de projeto mais utilizados e conhecidos no desenvolvimento de sistemas. O

método consiste em isolar toda a lógica responsável por fazer alterações no banco de dados, de forma que o restante da aplicação apenas consuma as funções responsáveis por tais ações. Entre suas maiores vantagens, é possível citar a facilitação da manutenção, testabilidade e reutilização do código, além da promoção da separação de responsabilidades e da modularidade do sistema.

Com a lógica de comunicação com o banco concentrada nos *repositories*, cabe aos *controllers* encapsular a lógica responsável por ler o corpo das requisições, identificar a viabilidade da operação solicitada através das funções declaradas nos *repositories*, e enviar uma resposta.

Os *routers* definem as rotas existentes e passíveis de receber requisições, fazendo contato tanto com os *middlewares*, quanto com os *controllers*. Os *middlewares* são responsáveis por fazer verificações de formatos de dados, através dos *schemas*, e identificar as credenciais de acesso do usuário, em caso de rotas autenticadas. Ao identificar problemas em alguma dessas verificações, os *middlewares* interrompem o fluxo antes que os dados cheguem aos controllers, retornando o erro cabível a situação como resposta da requisição. Caso as verificações sejam bem sucedidas, a requisição é passada aos *controllers* para que seu trabalho, acima descrito, seja executado.

A Figura 5.2 mostra a hierarquia de diretórios e arquivos da aplicação, seguindo o padrão de projeto acima retratado. As camadas da aplicação serão abordadas através o fluxo de dados de cada funcionalidade, individualmente detalhadas nos tópicos seguintes.

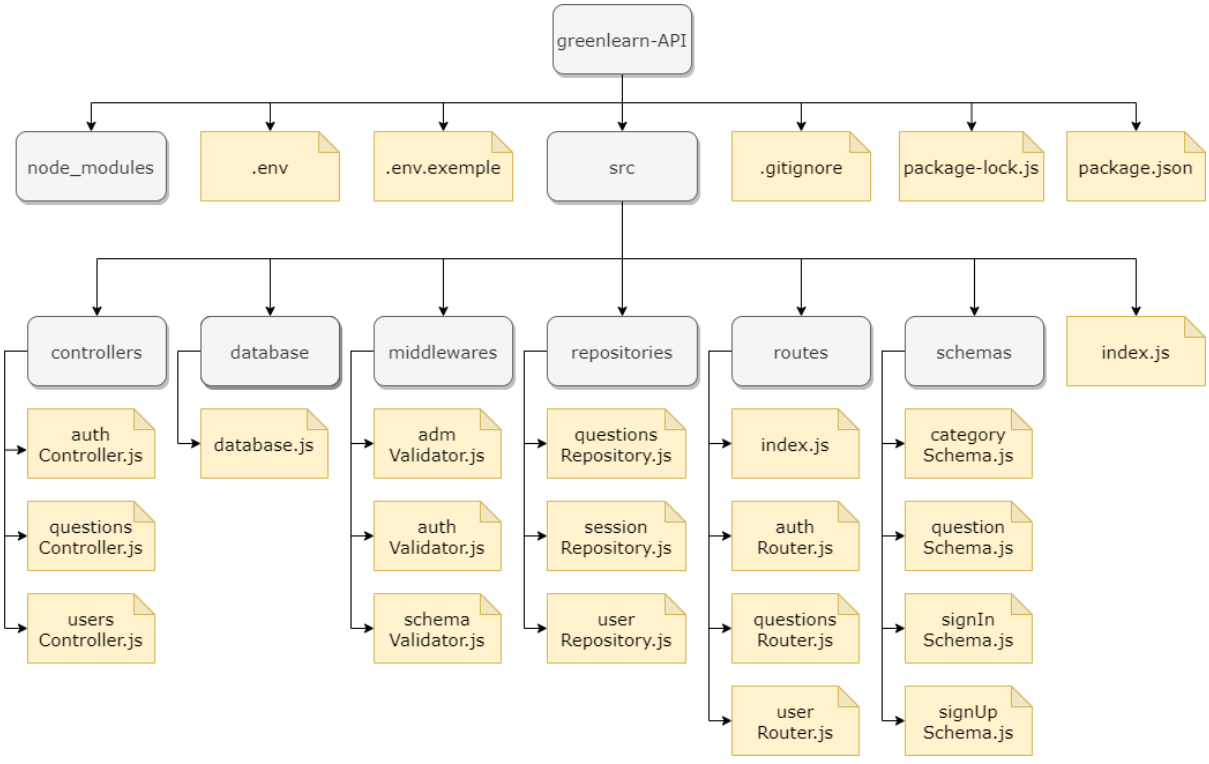


Figura 5.2 – Hierarquia de diretórios e arquivos do *backend*. Fonte: Acervo do autor.

5.2.2 Configuração e conexão

Para viabilizar o funcionamento correto e eficiente do sistema, é necessário realizar a configuração do servidor e a conexão com o banco de dados.

5.2.2.1 Configuração do servidor

A configuração do servidor envolve a definição das rotas, *middlewares* integrados e a porta em que o servidor irá escutar as requisições. É através da configuração que o servidor é preparado para receber e processar as solicitações dos clientes.

A configuração do servidor ocorre no arquivo *index.js* da pasta *src*, estando descrito no Bloco de Código 5.1.

```
1  import express, { json } from "express";
2  import dotenv from "dotenv";
3  import cors from "cors";
4  import router from "./routes/index.js";
5
6  dotenv.config();
7
8  const app = express();
9  app.use(json());
10 app.use(cors());
11 app.use(router);
12
13 const port = process.env.PORT || 4000;
14
15 app.listen(port, () => {
16   console.log(`Server running on port ${port}`);
17 });
```

Código 5.1: Configuração do servidor

No código acima, primeiramente, são importados os pacotes necessários, como o próprio *Express.js*, o módulo json para processamento de dados em formato JSON, o dotenv para carregar variáveis de ambiente a partir de um arquivo .env, e o pacote cors para lidar com a política de mesma origem (*Cross-Origin Resource Sharing*).

Em seguida, é criada uma instância do *Express*, que será responsável por lidar com as requisições e respostas do servidor. Os *middlewares* são definidos utilizando o método `use()` do aplicativo *Express*. O *middleware* `json()` é utilizado para processar os dados recebidos em formato JSON nas requisições. Já o *middleware* `cors()` é responsável por permitir que o servidor aceite requisições de diferentes origens, o que é especialmente útil quando se trabalha com APIs.

O próximo passo é a configuração das rotas da aplicação, que estão definidas no arquivo *index.js* localizado na pasta *routes*, e se responsabiliza por criar um objeto *router* de roteamento para cada arquivo de rota e, em seguida, exportá-los (Bloco de Código 5.2). O módulo *router* é importado utilizado como um *middleware* através do método `use()`, para que as rotas sejam corretamente tratadas.

Por fim, é definida a porta em que o servidor irá escutar as requisições, através da variável de ambiente `PORT`, caso ela esteja configurada. Caso contrário, é utilizada a porta 4000 como padrão. Após a configuração da porta, o servidor é iniciado através do método `listen()`, exibindo uma mensagem no console indicando que o servidor está em execução na porta especificada.

```
1  import { Router } from "express";
2  import authRouter from "./authRouter.js";
3  import questionRouter from "./questionsRouter.js";
4  import userRouter from "./userRouter.js";
5
6  const router = Router();
7
8  router.use(authRouter);
9  router.use(questionRouter);
10 router.use(userRouter);
11
12 export default router;
```

Código 5.2: Roteamento da aplicação

5.2.2.2 Conexão com o banco de dados

O diretório *database* é utilizado para armazenar arquivos relacionados à configuração de conexão e interação com o banco de dados. O arquivo *database.js* dentro desse diretório tem a função de estabelecer a conexão com o banco de dados, configurar as opções de acesso e prover métodos para realizar operações de leitura, escrita e manipulação de dados. Além disso, desempenha um papel crucial na camada de persistência da aplicação ao permitir que as informações sejam armazenadas, recuperadas e atualizadas de forma eficiente e segura. Nesse arquivo, o *Pool* de conexão é criado através da *connection string* passada por meio de uma variável de ambiente, e exportado a fim de ser utilizado nos arquivos onde a comunicação com o banco é realizada.

5.2.3 Rotas de autenticação

As rotas de autenticação são um conjunto de *endpoints* que lidam com o processo de autenticação e autorização de usuários, sendo responsáveis por receber as solicitações relacionadas ao *login*, cadastro e outras funcionalidades que dizem respeito à identificação do usuário.

Para este trabalho, as rotas de autenticação estão contidas no diretório *routes*, no arquivo `authRouter.js`, representado no Bloco de Código 5.3. Trata-se das rotas responsáveis pelo cadastro e *login*, acessadas através das rotas `'/sign-up'` e `'/sign-in'`, respectivamente, que serão discutidas nos tópicos seguintes.

```
1  import { Router } from "express";
2  import signUpSchema from "../schemas/signUpSchema.js";
3  import signInSchema from "../schemas/signInSchema.js";
4  import { validateSchema } from "../middlewares/schemaValidator.js";
5  import { signIn, signUp } from "../controllers/authControler.js";
6
7
8  const authRouter = Router();
9
10 authRouter.post('/sign-up', validateSchema(signUpSchema), signUp);
11 authRouter.post('/sign-in', validateSchema(signInSchema), signIn);
12
13 export default authRouter;
```

Código 5.3: Rotas de autenticação - authRouter.js

5.2.3.1 Fluxo de cadastro

O fluxo de dados para a funcionalidade de cadastro da aplicação se inicia ao realizar uma requisição do tipo *post* para a rota `'/sign-up'`, descrita na linha 10 do Bloco de Código 5.3.

Ao receber a chamada, ocorre a validação dos dados recebidos por meio do corpo da requisição através da chamada do *validateSchema*. Trata-se de um *middleware* reutilizável para validação de diferentes padrões de dados (*schemas*), que recebe o *schema* por parâmetro no momento de sua chamada.

Um *schema* é uma representação estruturada e definida de como os dados devem ser organizados e quais são as regras que eles devem seguir em relação a formatos e tipos. Para o caso da funcionalidade de cadastro, o *schema* passado é o *signUpSchema*, que determina se os dados recebidos para cadastro têm as propriedades: *name*, que deve ser uma *string*; *email*, que também deve ser uma string e apresentar a formatação de um

e-mail; e *password* e *confirmPassword*, ambas do tipo *string* e de valores idênticos, como se pode verificar no Bloco de Código 5.4.

Ao receber o *schema*, o *middleware* se responsabiliza por validar se o corpo da requisição apresenta o padrão esperado, usando o método `validate()` para encontrar diferenças e apontá-las como erros. Uma vez que um erro é identificado, o fluxo da aplicação é interrompido e o erro *422 - Unprocessable Entity* é retornado, indicando que o servidor não é capaz de processar os dados no formato recebido, como é possível observar no Bloco de Código 5.5.

```
1  import joi from 'joi';
2
3  const signUpSchema = joi.object({
4    name: joi.string().required(),
5    email: joi.string().email().required(),
6    password: joi.string().required(),
7    confirmPassword: joi.ref("password")
8  });
9
10 export default signUpSchema;
```

Código 5.4: *Schema* de cadastro - *signUpSchema.js*

```
1  export function validateSchema(schema){
2      return (req, res, next) => {
3          const error = schema.validate(
4              (req.body),
5              {abortEarly: false}
6          ).error;
7
8          if(error){
9              return res.status(422).send(
10                 error.details.map((detail) => {
11                     detail.message
12                 })
13             );
14         }
15
16         next();
17     }
18 }
```

Código 5.5: *Middleware* de validação de *schemas* - *schemaValidator.js*

Em seguida, se nenhum erro for capturado durante a validação de padrão dos dados, a informação é passada para a função *signUp*, importada dos *controllers*, responsável pela lógica envolvida na tentativa de registro das informações no banco de dados e na resposta da requisição de acordo com o sucesso do processo. Trata-se de uma função assíncrona que utiliza os métodos *try/catch* para chamar as *queries* de comunicação direta com o banco, contidas na camada *repositories*, como mostra o Bloco de Código 5.6.

```
1  export async function signUp(req, res){
2
3      const { name, email, password } = req.body;
4
5      try{
6          const userAlreadyExists = await userRepository.getUserByEmail(
               email);
7
8          if(userAlreadyExists){
9              return res.status(409).send("Usuario ja cadastrado");
10         }
11
12         await userRepository.createUser(name, email, password);
13
14         res.status(201).send('Usuario registrado com sucesso');
15     }
16     catch (error){
17         return res.status(500).send(error);
18     }
19 }
```

Código 5.6: Função *signUp* - *authController.js*

Na função *signUp*, após a desestruturação dos dados oriundos do corpo da requisição em variáveis homônimas, ocorre a utilização do *try/catch*, afim de tratar casos de sucesso e erro de execução das funções assíncronas. Assim, quaisquer adversidades ao executar os comandos dentro do bloco *try* resultam em um desvio para o bloco *catch*, interrompendo o fluxo da aplicação e retornando um erro *500 - Internal Server Error*, indicando problemas com o servidor.

Iniciando o bloco *try*, a função *getUserByEmail*, importada dos *repositories*, é chamada, recebendo como parâmetro o e-mail do usuário. A função se responsabiliza por realizar uma busca pelo e-mail do usuário no banco de dados e pode ser conferida no Bloco de Código 5.7. Em seguida, ocorre a validação do resultado da busca. Se o usuário foi encontrado, o erro *409 - Conflict* é retornado, indicando que já existe um usuário cadastrado com esses dados e interrompendo o fluxo. Caso a busca não traga resultados, o processo continua. Por fim, ocorre a chamada da função *createUser* que recebe como parâmetro o nome, o e-mail e a senha do usuário a ser cadastrado (Bloco de Código 5.8).

A função é importada dos *repositories* e é responsável por fazer o registro dos dados do usuário no banco. Antes do cadastro no banco, por motivos de segurança, a senha do usuário passa por um processo de criptografia utilizando a biblioteca *bcrypt* e, portanto, o banco de dados jamais chega a armazenar a senha em seu formato original. Caso nenhum problema ocasione o redirecionamento para o bloco *catch* e o registro seja bem sucedido, a requisição é respondida com o *status code 201 - Created*, informando que o usuário foi cadastrado com sucesso.

```
1 export async function getUserByEmail(email){
2     const user = await connection.query(`
3         SELECT * FROM users WHERE email = $1;
4     `, [email]);
5 
6     return user.rows[0];
7 }
```

Código 5.7: Função *getUserByEmail - userRepository.js*

```
1  export async function createUser(name, email, password){
2 
3      const passwordHash = bcrypt.hashSync(password, 12);
4 
5      const createdUser = await connection.query(`
6          INSERT INTO users (name, email, password)
7          VALUES ($1, $2, $3);
8      `, [name, email, passwordHash]);
9 
10     return createdUser.rows[0];
11 }
```

Código 5.8: Função *createUser - userRepository.js*

5.2.3.2 Fluxo de login

O fluxo de dados da funcionalidade de *login*, tal qual o de cadastro, é iniciado com uma requisição do tipo *post* para a rota `'/sign-in'`. Muitos dos processos ocorrem de forma análoga ao descrito anteriormente para a rota de cadastro, entre eles, a validação de padrão de dados recebidos realizada pelo *middleware*, cuja única diferença está no *schema* a ser considerado. O *schema* passado como parâmetro é o *signInSchema*, cuja declaração pode ser encontrada no Bloco de Código 5.9, e se incube de definir o padrão esperado para o *login*, no caso, um objeto contendo as propriedades *email*, que deve ser uma *string* na formatação de um e-mail, e *password*, também do tipo *string*.

Em seguida, caso as validações do *middleware* sejam bem sucedidas, a função *signIn*, importada dos *controllers* e descrita no Bloco de Código 5.10, é executada. A função realiza, dentro do bloco *try*, a busca por um usuário cadastrado com o e-mail recebido no corpo da requisição, através da execução da função *getUserByEmail* detalhada no tópico anterior e disponível para consulta no Bloco de Código 5.7.

Uma vez em posse do resultado da busca do e-mail recebido nos os dados do banco, ocorre a validação de existência de cadastro. Se a busca não retornou resultados, o fluxo é interrompido com o retorno do erro *401 - Unauthorized*, acompanhado da mensagem *"E-mail e/ou senha incorretos"*. Caso a existência de um cadastro com o e-mail recebido seja validada, ocorre a análise de compatibilidade entre a senha recebida e a versão criptografada da senha do usuário encontrada no banco; processo realizado através do método `compareSync()` da biblioteca *bcrypt*. Caso a análise apresente resultado negativo, a requisição é respondida com o erro *401 - Unauthorized*. Caso contrário, o processo continua e ocorre a criação de um *token* de acesso para o usuário.

Em seguida, a função *createSession* é chamada, com o objetivo de criar uma sessão vinculada ao usuário, como mostra o Bloco de Código 5.11. Em caso de sucesso na criação da sessão, a requisição é respondida com o *status code 201 - Created* e um objeto contendo o *token* e o nome do usuário. Em caso de falha em qualquer um dos processos descritos dentro do bloco *try*, o fluxo é desviado para o bloco *catch* e o erro *500 - Internal Server Error* é retornado.

```
1 import joi from 'joi';
2
3 const signInSchema = joi.object({
4   email: joi.string().email().required(),
5   password: joi.string().required()
6 });
7
8 export default signInSchema;
```

Código 5.9: *Schema* de *login* - *signInSchema.js*

```
export async function signIn(req, res){
    const { email, password } = req.body;

    try{
        const user = await userRepository.getUserByEmail(email);

        if(!user){
            return res.status(401).send("Email e/ou senha incorretos");
        }

        const isUserAuthenticated = bcrypt.compareSync(password, user.
           password);

        if(!isUserAuthenticated){
            return res.status(401).send("Email e/ou senha incorretos");
        }

        const token = uuid();

        await sessionRepository.createSession(user.id, token);

        res.status(201).send({token: token, userName: user.name});
    }
    catch(error){
        res.status(500).send(error);
    }
}
```

Código 5.10: Função *signIn* - *authController.js*

```
export async function signIn(req, res){
export async function createSession(userId, token){
    const session = await connection.query(`
        INSERT INTO
            sessions ("userId", token)
        VALUES
            ($1, $2);
    `, [userId, token]);

    return session;
}
```

Código 5.11: Função *createSession* - *sessionRepository.js*

5.2.4 Fluxos de navegação

Na construção do roteamento da aplicação, como foi apresentado no Bloco de Código 5.2, ocorre a importação e utilização do *questionRouter* - instância de roteamento que se encarrega da definição das rotas que fazem referência as perguntas da aplicação e suas características, tais como categoria e nível.

A instância possui sete rotas, que serão discutidas nas seções a seguir, e estão explicitadas no Bloco de Código 5.12.

```
1  import { Router } from "express";
2  import { validateAuth } from "../middlewares/authValidator.js";
3  import { validateSchema } from "../middlewares/schemaValidator.js";
4  import questionSchema from "../schemas/questionSchema.js";
5  import { validateAdm } from "../middlewares/admValidator.js";
6  import categorySchema from "../schemas/categorySchema.js";
7  import {
8      getCategories,
9      getLevels,
10     getQuestionById,
11     getQuestionsByCategoryAndLevel,
12     postProgress,
13     postQuestion,
14     postCategory
15 } from "../controllers/questionsController.js";
16
17
18 const questionRouter = Router();
19
20 questionRouter.get('/categories', validateAuth, getCategories);
21 questionRouter.get('/categories/:categoryId/levels', validateAuth,
       getLevels);
22 questionRouter.get('/categories/:categoryId/levels/:levelId/questions',
       validateAuth, getQuestionsByCategoryAndLevel);
23 questionRouter.get('/question/:questionId', validateAuth,
       getQuestionById);
24 questionRouter.post('/question/progress', validateAuth, postProgress);
25 questionRouter.post('/categories', validateAdm, validateSchema(
       categorySchema), postCategory)
26 questionRouter.post('/questions', validateAdm, validateSchema(
       questionSchema), postQuestion);
27
28 export default questionRouter;
```

Código 5.12: Instância de roteamento *questionRouter* - *questionsRouter.js*

5.2.4.1 Listagem de categorias, níveis e perguntas

As rotas dessa seção são autenticadas, isso é, apenas usuários logados possuem permissões de acesso. Tais permissões são validadas e verificadas através do *middleware validateAuth*, que pode ser observado no Bloco de Código 5.13. A função se encarrega de acessar o cabeçalho (*headers*) da requisição e resgatar o parâmetro *authorization*, que contém o *token* de acesso do usuário, e realizar a retirada da formatação padrão *Bearer*, deixando apenas a *string* original do *token*.

Por conseguinte, uma verificação de existência do *token* é realizada e, em casos nos quais o dado não existe, a requisição é respondida com o erro *401 - Unauthorized*, acompanhado da mensagem *"Não há um token de acesso"*. Se o *token* existir, o fluxo continua e o bloco *try* se inicia, chamando a função *getSessionByToken* e passando o *token* como parâmetro. Tal função se encarrega de realizar uma busca por sessões que possuem o token informado no banco de dados, retornando as informações da sessão caso a encontre, como mostra o Bloco de Código 5.14.

Uma vez que a busca é retornada, o *middleware* realiza uma validação do resultado e, se uma sessão não for encontrada para o *token* em questão, a função retorna com o erro *401 - Unauthorized*, seguido da mensagem *"Sessão não encontrada"*.

Com os dados da sessão em mãos, é possível encontrar o usuário a quem ela pertence, através da função *getUserById*. Tal função se encarrega de buscar um usuário no banco pelo seu *id* e, caso não encontre, o *middleware* encerra o fluxo, devolvendo o erro *401 - Unauthorized* como resposta. Caso contrário, os dados do usuário são armazenados no objeto de variáveis locais (`res.locals`) e ficam disponível até que a requisição seja encerrada. Caso algum erro de comunicação com o banco ocorra ao longo do bloco *try*, acontece o desvio para o bloco *catch*, onde a requisição é encerrada com o erro *500 - Internal Server Error*. É importante ressaltar que esse processo de validação de *token* para autenticação de usuário é perfeitamente reutilizável e se repete para todas as rotas autenticadas.

```
1  import * as sessionRepository from "../repositories/sessionRepository.js
       ";
2  import * as userRepository from "../repositories/userRepository.js";
3
4  export async function validateAuth(req, res, next){
5      const { authorization } = req.headers;
6      const token = authorization?.replace("Bearer ", "");
7
8      if(!token){
9          return res.status(401).send("Nao ha token de acesso");
10     }
11
12     try{
13         const session = await sessionRepository.getSessionByToken(token)
               ;
14
15         if(!session){
16             return res.status(401).send("Sessao nao encontrada");
17         }
18
19         const user = await userRepository.getUserById(session.userId);
20
21         if(!user){
22             return res.status(401).send("Usuario nao encontrado");
23         }
24
25         res.locals.user = user;
26         next();
27     }
28     catch(error){
29         return res.sendStatus(500);
30     }
31 }
```

Código 5.13: *Middleware* de autenticação - *validateAuth.js*

A rota `'/categories'` possui o objetivo de listar todas as categorias existentes na aplicação e, após passar pelo *middleware* de autenticação, executa a função *getCategories*, importada dos *controllers* (Bloco de Código 5.15). Seu papel como controladora é a chamada a função homônima *getCategories*, importada dos *repositories*, que faz a busca por todas categorias no banco (Bloco de Código 5.16).

Caso não haja problemas de comunicação com o banco que redirecionem para o bloco *catch*, a função controladora retorna o *status code 200 - OK*, em conjunto com a lista de categorias encontradas.

```
export async function getSessionByToken(token){
    const session = await connection.query(`
        SELECT * FROM sessions WHERE token = $1;
    `, [token]);

    return session.rows[0];
}
```

Código 5.14: Função *getSessionByToken* - *sessionRepository.js*

```
export async function getCategories(req, res){
    try{
        const categories = await questionRepository.getCategories();

        return res.status(200).send(categories);
    }
    catch(error){
        return res.status(500).send(error);
    }
}
```

Código 5.15: Função *getCategories* - *questionsController.js*

```
export async function getCategories(){
    const categories = await connection.query(`
        SELECT * FROM categories;
    `);

    return categories.rows;
}
```

Código 5.16: Função *getCategories* - *questionsRepository.js*

A rota `'/categories/:categoryId/levels'` é responsável pela busca de todos os níveis de uma determinada categoria. Seu funcionamento ocorre de forma bem parecida com o da rota `'/categories'`, diferindo-se na existência do parâmetro de rota `:categoryId`, que indica a categoria para a qual a busca de níveis será feita.

Como é uma rota privada, passa pelo *middleware* de autenticação e, uma vez que o *token* de acesso é validado, a função controladora *getLevels* é executada. Tal função se responsabiliza pela validação da existência da categoria e busca a categoria pelo seu *id* no banco, através da função de *repository getCategoryById*, retornando o erro *404 - Not Found* caso a busca não encontre resultados. Em seguida, chama a função de *repository getLevels*, que busca todos os níveis existentes no banco.

Por fim, a função controladora retorna o resultado da busca, com o *status code 200*. A função controladora *getLevels* e as funções de *repository getCategoryById* e *getLevels* podem ser verificadas, respectivamente, nos Blocos de Código 5.17, 5.18 e 5.19.

```
export async function getLevels(req, res) {
    const { categoryId } = req.params;

    try {
        const category = await questionRepository.getCategoryById(
            categoryId);

        if (!category) {
            return res.status(404).send("Categoria não encontrada");
        }

        const levels = await questionRepository.getLevels();

        const result = {
            id: category.id,
            categoryName: category.name,
            levels: levels
        }

        return res.status(200).send(result);
    }
    catch (error) {
        return res.status(500).send(error);
    }
}
```

Código 5.17: Função *getLevels* - *questionsController.js*

```
export async function getCategoryById(categoryId) {
    const category = await connection.query(`
        SELECT * FROM categories WHERE id = $1;
    `, [categoryId]);

    return category.rows[0];
}
```

Código 5.18: Função *getCategoryById* - *questionsRepository.js*

```
export async function getLevels() {
    const levels = await connection.query(`
        SELECT * FROM levels;
    `);

    return levels.rows;
}
```

Código 5.19: Função *getLevels* - *questionsRepository.js*

A rota `'/categories/:categoryId/levels/:levelId/questions'`, tem por objetivo listar todas as perguntas pertencentes a uma categoria e a um nível específico. As primeiras etapas, que envolvem a autenticação do usuário e a validação da existência da categoria e do nível ocorrem de forma análoga a rota anterior e, para evitar redundância, não serão explicadas aqui.

Analisando então a função controladora *getQuestionsByCategoryAndLevel*, descrita no Bloco de Código 5.20, a partir da linha 16, nota-se a chamada da funções de *repository getQuestionsByCategoryAndLevel* e *getQuestionsDoneByCategoryLevelAndUser*, que tem o objetivo de buscar no banco, respectivamente, as questões existentes para dada categoria e nível e as questões da categoria e nível em questão que já foram respondidas pelo usuário (Blocos de Código 5.21 e 5.22).

Em seguida, o *array* contendo todas as questões é percorrido e, para cada questão, sua existência é verificada no *array* de questões respondidas. Caso exista, a variável *isDone* é preenchida com o valor booleano *true* e caso contrário, com o valor *false*. Tal formatação serve para produzir uma lista de perguntas personalizada, capaz de informar quais perguntas o usuário em questão respondeu ou não. O acompanhamento das respostas do usuário é feito através da tabela *progress*, acessada pela função *getQuestionsDoneByCategoryLevelAndUser*.

Por fim, se todas as buscas correram bem no banco, o bloco *try* é finalizado com o envio do *array* de questões formatado, acompanhado do *status code 200 - OK*. Caso haja algum problema na comunicação com o banco, o desvio para o bloco *catch* ocorre e o retorno é o *status code 500 - Internal Server Error*.

```
1  export async function getQuestionsByCategoryAndLevel(req, res) {
2      const { categoryId, levelId } = req.params;
3      const { authorization } = req.headers;
4      const token = authorization?.replace("Bearer ", "");
5
6      try {
7          const category = await questionRepository.getCategoryById(
               categoryId);
8          const level = await questionRepository.getLevelById(levelId);
9          const session = await sessionRepository.getSessionByToken(token)
               ;
10         const userId = session.userId;
11
12         if (!category || !level) {
13             return res.status(404).send("Categoria e/ou nível não
                   encontrados");
14         }
15
16         const questions = await questionRepository.
```

```
            getQuestionsByCategoryAndLevel(categoryId, levelId);
        const questionsDone = await questionRepository.
            getQuestionsDoneByCategoryLevelAndUser(categoryId, levelId,
            userId);

        const formatedQuestions = questions.map((question) => {
            let isDone = false;

            questionsDone.forEach((questionDone) => {
                if (question.id === questionDone.id) {
                    isDone = true
                }
            });

            return (
                {
                    id: question.id,
                    levelId: question.levelId,
                    categoryId: question.categoryId,
                    text: question.text,
                    isDone: isDone
                }
            )
        });

        return res.status(200).send(formatedQuestions);
    }
    catch (error) {
        return res.status(500).send(error);
    }
}
```

Código 5.20: Função *getQuestionsByCategoryAndLevel - questionsController.js*

```
export async function getQuestionsByCategoryAndLevel(categoryId, levelId
    ) {
    const questions = await connection.query(`
        SELECT * FROM
            questions
        WHERE
            questions."categoryId" = $1
        AND
            questions."levelId" = $2;
    `, [categoryId, levelId]);

    return questions.rows;
}
```

Código 5.21: Função *getQuestionsByCategoryAndLevel - questionsRepository.js*

```
export async function getQuestionsDoneByCategoryLevelAndUser(categoryId,
    levelId, userId) {
    const questions = await connection.query(`
        SELECT
            questions.*
        FROM
            questions
        JOIN
            progress ON questions.id = progress."questionId"
        WHERE
            questions."categoryId"=$1 AND questions."levelId"=$2 AND
               progress."userId" = $3
        ;
    `, [categoryId, levelId, userId]);

    return questions.rows;
}
```

Código 5.22: *getQuestionsDoneByCategoryLevelAndUser - questionsRepository.js*

5.2.4.2 Busca de perguntas específicas

A busca de perguntas específicas ocorre através de uma requisição do tipo *get* para a rota `'question/:questionId'`, Como se trata de uma rota autenticada, o *middleware* de autenticação é chamado e, após suas validações serem realizadas, ocorre a execução da função controladora *getQuestionById*, descrita no Bloco de Código 5.23.

A função se recebe o *id* da pergunta por parâmetros de rota e dá inicio ao bloco *try*, onde as funções *getQuestionById* e *getAnswersByQuestionId* são chamadas dos *repositories*, com o objetivo de obter, respectivamente, o enunciado da pergunta desejada e suas quatro opções de resposta. As funções estão descritas, de maneira respectiva, nos Blocos de Código 5.24 e 5.25

Em seguida, ocorre uma validação da busca realizada com o intuito de verificar a existência de uma questão ou de respostas referentes ao *id* recebido. Caso uma das condições não se verifique, é retornado o *status code 404 - Not Found*, acompanhado da mensagem "Pergunta não encontrada". Caso a busca encontre respostas, os dados são formatados em um objeto, que é enviado como resposta, acompanhado do *status code 200 - OK*.

Se algum problema de conexão surgir durante as buscas, ocorre o desvio para o bloco *catch* e o retorno do *status code 500 - Internal Server Error*

```
export async function getQuestionById(req, res) {
    const { questionId } = req.params;

    try {
        const question = await questionRepository.getQuestionById(
            questionId);
        const answers = await questionRepository.getAnswersByQuestionId(
            questionId);

        if (!question || !answers) {
            return res.status(404).send("Pergunta não encontrada");
        }
        const formatedQuestion = {
            id: question.id,
            categoryId: question.categoryId,
            categoryName: question.categoryName,
            levelId: question.levelId,
            levelName: question.levelName,
            text: question.text,
            answers: answers
        };
        return res.status(200).send(formatedQuestion);
    }
    catch (error) {
        return res.status(500).send(error);
    }
}
```

Código 5.23: Função *getQuestionById - questionsController.js*

```
export async function getQuestionById(questionId) {
    const question = await connection.query(`
        SELECT
            questions.*, categories.name as "categoryName", levels.name
                as "levelName"
        FROM
            questions
        JOIN
            categories ON categories.id = questions."categoryId"
        JOIN
            levels ON levels.id = questions."levelId"
        WHERE
            questions.id=$1;
    `, [questionId]);

    return question.rows[0];
}
```

Código 5.24: Função *getQuestionById - questionsRepository.js*

```
export async function getAnswersByQuestionId(questionId) {
    const answers = await connection.query(`
        SELECT * FROM answers WHERE answers."questionId" = $1;
    `, [questionId]);

    return answers.rows;
}
```

Código 5.25: Função *getAnswersByQuestionId - questionsRepository.js*

5.2.4.3 Registro de progresso

O registro de progresso é realizado através de requisições do tipo *post* enviadas para a rota `'/question/progress'` que, após as validações do *middleware* de autenticação, chama a função controladora *postProgress* (Bloco de Código 5.27).

A função controladora identifica o usuário através de uma busca por sua sessão, utilizando o *token* de acesso na função *getSessionByToken*, e faz uma busca da questão pelo *id* recebido via corpo de requisição.

Em seguida, acontece uma validação para ambas as buscas. Se a sessão ou a questão não forem encontradas, o fluxo é interrompido com o retorno do *status code 404*, acompanhado da mensagem "sessão ou questão não encontradas". Caso esse cenário não ocorra, o processo de registro de progresso é continuado.

Com a chamada da função de *repository getProgressByUserAndQuestion* (Bloco de Código 5.26), é verificado se o usuário já respondeu aquela pergunta. Em casos onde a pergunta já foi respondida, não ocorre alteração no banco e o *status code 200* é enviado junto à mensagem "questão já concluída". Se o usuário não tiver respondido a questão, o progresso é registrado no banco por meio da função de *repository postProgress* (Bloco de Código 5.28) e a requisição é respondida com o *status code 201 - created*, com a mensagem "progresso registrado".

```
export async function getProgressByUserAndQuestion(userId, questionId){
    const progress = await connection.query(`
        SELECT * FROM progress WHERE "userId" = $1 AND "questionId" = $2
            ;
    `, [userId, questionId]);

    return progress;
}
```

Código 5.26: Função *getProgressByUserAndQuestion - questionsRepository.js*

```
export async function postProgress(req, res) {
    const { questionId } = req.body;
    const { authorization } = req.headers;
    const token = authorization?.replace("Bearer ", "");

    try {
        const session = await sessionRepository.getSessionByToken(token)
            ;
        const question = await questionRepository.getQuestionById(
            questionId);

        if (!session || !question) {
            return res.status(404).send('Sessão ou questão não
                encontradas')
        };

        const userId = session.userId;

        const questionDone = await questionRepository.
            getProgressByUserAndQuestion(userId, questionId);

        if (questionDone.rowCount !== 0) {
            return res.status(200).send('Questão já concluída');
        }

        const progress = await questionRepository.postProgress(userId,
            questionId);

        return res.status(201).send('Progresso registrado');
    }
    catch (error) {
        return res.status(500).send(error);
    }
}
```

Código 5.27: Função *postProgress* - *questionsController.js*

```
export async function postProgress(userId, questionId){
    const progress = await connection.query(`
        INSERT INTO
            progress("userId", "questionId")
        VALUES
            ($1, $2);
    `, [userId, questionId]);

    return progress;
}
```

Código 5.28: Função *postProgress* - *questionsRepository.js*

5.2.4.4 Ranking

O *ranking* de usuários pode ser obtido através de uma requisição do tipo *get* para a rota `'/users/progress'`, que se encontra no arquivo *userRouter* (Bloco de Código 5.29), no diretório de *routes*. Trata-se de uma rota privada e, portanto, a validação de autenticação é feita pelo *middleware validateAuth*.

Em seguida, ocorre a execução da função controladora *getUsersProgress*, demonstrada no Bloco de Código 5.30. Nela, é chamada a função de *repository* homônima *getUsersProgress*, com o objetivo de buscar no banco o progresso dos usuários. Tal função está descrita no Bloco de Código 5.31, e contem uma *query* que faz uma junção das tabelas *progress* e *users*, agrupando os resultados por usuário e ordenando de forma decrescente pelo número de questões respondidas. Dessa forma, seu retorno é uma lista de usuários por ordem de progresso, do maior para o menor, ou seja, um *ranking*.

Uma vez em posse do *ranking*, a função controladora formata a lista, transformando o número de perguntas respondidas em pontuação ao multiplicá-lo por 100. Por fim, a lista formatada é enviada como resposta da requisição, acompanhada do *status code 200*.

```
1  import { Router } from "express";
2  import { validateAdm } from "../middlewares/admValidator.js";
3  import { downgradePermition, getUserProgress, getUsers, getUsersProgress
       , upgradePermition } from "../controllers/usersController.js";
4  import { validateAuth } from "../middlewares/authValidator.js";
5
6  const userRouter = Router();
7
8  userRouter.get('/users', validateAdm, getUsers)
9  userRouter.put('/users/upgrade', validateAdm, upgradePermition);
10 userRouter.put('/users/downgrade', validateAdm, downgradePermition);
11 userRouter.get('/users/progress', validateAuth, getUsersProgress);
12
13 export default userRouter;
```

Código 5.29: Instância de rotas de usuário - *userRouter.js*

```
export async function getUsersProgress(req, res) {
    try {
        const usersProgress = await userRepository.getUsersProgress();

        const usersScore = usersProgress.map((user) => {
            return {
                name: user.name,
                email: user.email,
                score: user.questionsDone * 100
            }
        })

        return res.status(200).send(usersScore);
    }
    catch (err) {
        return res.status(500).send(err);
    }
}
```

Código 5.30: Função controladora *getUsersProgress* - *usersController.js*

```
export async function getUsersProgress() {
    const progress = await connection.query(`
        SELECT
            users.name, users.email, COUNT(progress."questionId") as "
               questionsDone"
        FROM
            progress
        JOIN
            users
        ON
            progress."userId" = users.id
        GROUP BY
            users.email, users.name
        ORDER BY
            "questionsDone"
        DESC;
    `);

    return progress.rows;
}
```

Código 5.31: Função getUsersProgress - *userRepository.js*

5.2.5 Ferramentas administrativas

As rotas administrativas possuem algumas particularidades, pois necessitam de permissões de acesso especiais. Como são rotas de gerenciamento, só podem ser acessadas por usuários administradores e, por isso, um *middleware* de verificação de nível de permissão entra em ação: o *validateAdm* (Bloco de Código 5.32).

Seu início ocorre de forma similar ao middleware de autenticação, encontrando a sessão pelo *token* e o usuário pela sessão, mas, ao encontrar o usuário, verifica se a propriedade *isAdm* é verdadeira. Se a propriedade for validada, o fluxo é continuado. Caso contrário, o acesso é negado e o *status code 401 - unauthorized* é enviado como resposta.

```
1  export async function validateAdm(req, res, next) {
2      const { authorization } = req.headers;
3      const token = authorization?.replace("Bearer ", "");
4
5      if (!token) {
6          return res.status(401).send("Não há token de acesso");
7      }
8
9      try {
10         const session = await sessionRepository.getSessionByToken(token)
               ;
11
12         if (!session) {
13             return res.status(401).send("Sessão não encontrada");
14         }
15
16         const user = await userRepository.getUserById(session.userId);
17
18         if (!user) {
19             return res.status(401).send("Usuário não encontrado");
20         }
21
22         if (!user.isAdm) {
23             return res.status(401).send("Usuário sem permissões de
                   acesso");
24         }
25
26         res.locals.user = user;
27         next();
28     }
29     catch (error) {
30         return res.sendStatus(500);
31     }
```

```
32 }
```

Código 5.32: *Middleware* de validação de permisões *validateAdm* - *admValidator.js*

5.2.5.1 Gerenciamento de conteúdo

A rota *post* `'/categories'` tem como objetivo a adição de novas categorias e conta com dois *middlewares*. O primeiro deles é o *validateAdm*, responsável por veficar se o usuário tem permissões de administrador para acessar a rota. O segundo, é o *validateSchema*, que é chamado passando o *categorySchema* como argumento.

Caso nenhum *middleware* barre a requisição, ocorre a chamada da função controladora *postCategory*. A função valida se a categoria já existe através da chamada da função de *repository getCategoryByName* e, caso a resposta seja positiva, retorna o *status code 409 - Conflict*, juntamente com a mensagem "a categoria já existe". Caso contrário, insere a categoria nova por meio da função de *repository postCategory*. A função controladora *postCategory* e as funções de *repository getCategoryByName* e *postCategory* podem ser conferidas, respectivamente, nos Blocos de Código 5.33, 5.34 e 5.35.

```
1  export async function postCategory(req, res) {
2      const { categoryName } = req.body;
3
4      try {
5          const categoryAlreadyExists = await questionRepository.
               getCategoryByName(categoryName);
6
7          if (categoryAlreadyExists) {
8              return res.status(409).send("A categoria já existe");
9          }
10
11         await questionRepository.postCategory(categoryName);
12
13         return res.status(201).send("Categoria criada com sucesso!")
14     }
15     catch {
16         return res.sendStatus(500);
17     }
18 }
```

Código 5.33: Função controladora *postCategory* - *questionsController.js*

```
1  export async function getCategoryByName(categoryName) {
2      const category = await connection.query(`
3          SELECT * FROM categories WHERE name = $1;
```

```
    `, [categoryName]);

    return category.rows[0]
}
```

Código 5.34: Função *getCategoryByName* - *questionsRepository.js*

```
export async function postCategory(categoryName) {
    const category = await connection.query(`
        INSERT INTO
            categories(name, icon)
        VALUES
            ($1, 'book-outline');
    `, [categoryName]);

    return category;
}
```

Código 5.35: Função *postCategory* - *questionsRepository.js*

A rota *post* `'/questions'` segue o mesmo padrão, passando pelos mesmos dois *middlewares*, porém o *validateSchema* recebe o padrão *questionSchema* (Bloco de Código 5.36) via parâmetro.

Em seguida, caso a passagem pelos *middlewares* seja bem sucedida, ocorre a execução da função controladora *postQuestion*, disponível no Bloco de Código 5.37. A função faz uma busca pela categoria e pelo nível, passados pelo corpo da requisição, para verificar se eles existem no banco. Caso a busca não encontre resultado, retorna o *status code 404*, acompanhado da mensagem "categoria e/ou nível inválidos". Caso contrário, dá prosseguimento ao processo, chamando a função de *repository postQuestion* (Bloco de Código 5.38) responsável pela adição da pergunta na tabela *questions*.

Após a adição da pergunta, faz-se necessário inserir as respostas, o que ocorre através da chamada da função de *repository postAnswers* (Bloco de código 5.39) dentro de um laço de repetição *forEach*, que percorre o *array* de respostas recebido no corpo da requisição, realizando, uma a uma, a inserção das respostas na tabela *answers*. Por fim, ocorre o envio do *status code 201*, com a mensagem "pergunta criada com seucesso".

```
import joi from 'joi';

const questionSchema = joi.object({
  categoryName: joi.string().required(),
  levelName: joi.string().required(),
  questionText: joi.string().required(),
  answers: joi.array().required()
});
export default questionSchema;
```

Código 5.36: *Schema* de perguntas - *questionSchema.js*

```
export async function postQuestion(req, res) {
    const { categoryName, levelName, questionText, answers } = req.body;

    try {
        const category = await questionRepository.getIdByCategoryName(
            categoryName);
        const level = await questionRepository.getIdByLevelName(
            levelName);

        if (!category || !level) {
            return res.status(404).send("Categoria e/ou nível inválidos"
                );
        }

        const categoryId = category.id;
        const levelId = level.id;

        const question = await questionRepository.postQuestion(
            categoryId, levelId, questionText);

        if (!question) {
            return res.status(400).send("Erro ao inserir pergunta");
        }

        const questionId = question.id;

        answers.forEach(async (answer) => {
            const text = answer.text;
            const isCorrect = answer.isCorrect;
            const answerResponse = await questionRepository.postAnswers(
                questionId, text, isCorrect);
        });

        return res.status(201).send('Pergunta criada com sucesso!');
    }
    catch {
        return res.sendStatus(500);
    }
}
```

Código 5.37: Função controladora *postQuestion* - *questionsController.js*

```
export async function postQuestion(categoryId, levelId, text) {
    const question = await connection.query(`
        INSERT INTO
            questions("categoryId", "levelId", text)
        VALUES
            ($1, $2, $3)
        RETURNING
            id;
    `, [categoryId, levelId, text]);

    return question.rows[0];
}
```

Código 5.38: Função *postQuestion* - *questionsRepository.js*

```
export async function postAnswers(questionId, text, isCorrect) {
    const answer = await connection.query(`
        INSERT INTO
            answers("questionId", text, "isCorrect")
        VALUES
            ($1, $2, $3);
    `, [questionId, text, isCorrect]);

    return answer;
}
```

Código 5.39: Função *postAnswers* - *questionsRepository.js*

5.2.5.2 Gerenciamento de permissões de usuário

A rota *get* `'/users'` se responsabiliza por trazer uma listagem de todos os usuários existentes na aplicação, trazendo informações armazenadas na tabela *users*. Por se tratar de uma rota de gerenciamento, o *middleware validateAdm* é chamado com o intuito de garantir as permissões corretas de acesso.

Uma vez que o *middleware* desempenhou o seu papel, ocorre a chamada da função controladora *getUsers*, descrita no Bloco de Código 5.40. A função chama a função de *repository* de mesmo nome, *getUsers* (Bloco de Código 5.41), com o intuito de fazer uma busca no banco por todos os usuários e retorná-los ordenados alfabeticamente por nome. Se nenhum usuário é encontrado, a função controladora envia uma resposta com o *status code 404 - Not Found* e a mensagem "não há usuários cadastrados".

Caso não ocorra, acontece uma reordenação para que os usuários administradores fiquem no topo da lista, seguidos por usuários comuns, ambos os grupos ordenados alfabeticamente por nome. Por fim, a resposta ordenada é enviada com o *status code 200*.

```
export async function getUsers(req, res) {
    try {
        const users = await userRepository.getUsers();

        if (users.length === 0) {
            return res.status(404).send("Não há usuários cadastrados");
        }

        const admUsers = [];
        const regularUsers = [];

        users.forEach((user) => {
            if (user.isAdm) {
                admUsers.push(user);
            }
            else {
                regularUsers.push(user);
            }
        })

        const sortedUsers = [...admUsers, ...regularUsers]

        const formatedUser = sortedUsers.map((user) => {
            return ({
                name: user.name,
                email: user.email,
                isAdm: user.isAdm
            })
        });

        return res.status(200).send(formatedUser);
    }
    catch (err) {
        return res.status(500).send(err);
    }
}
```

Código 5.40: Função controladora *getUsers* - *usersController.js*

```
export async function getUsers() {
    const users = await connection.query(`
        SELECT * FROM users ORDER BY name;
    `);
    return users.rows;
}
```

Código 5.41: Função *getUsers* - *userRepository.js*

Finalmente, restam as rotas de *put* `'/users/upgrade'` e `'/users/downgrade'`, responsáveis por fazer, respectivamente, a promoção de usuários comuns a administradores e o rebaixamento de usuários administradores a usuários comuns.

Após a execução do *middleware validateAdm*, a rota de *upgrade* chama a função controladora *upgradePermition*, descrita no Bloco de Código 5.42. A função se encarrega de encontrar o usuário através da função de *repository getUserByEmail* e, se não obtiver sucesso, retorna o *status code 404* com a mensagem "usuário não encontrado".

Caso a busca pelo usuário seja bem sucedida, executa-se a função de *repository upgradePermition* (Bloco de Código 5.43), responsável por alterar o valor da coluna *isAdm* de falso para verdadeiro para o usuário em questão. Então, a função controladora retorna a mensagem "usuário promovido com sucesso", com o *status code 201*.

Os passos para a rota de *downgrade* são idênticos, e a únicas diferença estão na alteração do valor booleano de *isAdm* de verdadeiro para falso e na mudança nos nomes das funções de *upgrade* para *downgrade*.

```
export async function upgradePermition(req, res) {
    const { email } = req.body;
    try {
        const user = userRepository.getUserByEmail(email);

        if (!user) {
            return res.status(404).send("Usuário não encontrado")
        }
        userRepository.upgradePermition(email);

        return res.status(201).send("Usuário promovido com sucesso");
    }
    catch (err) {
        return res.status(500).send(err);
    }
}
```

Código 5.42: Função controladora *upgradePermition - usersController.js*

```
export async function upgradePermition(email) {
    const user = await connection.query(`
        UPDATE
            users
        SET
            "isAdm" = true
        WHERE
            email = $1;
    `, [email]);
    return user.rows[0];
}
```

Código 5.43: Função *upgradePermition - userRepository.js*

5.3 APLICAÇÃO FRONTEND

O *frontend* da aplicação é a interface de contato com o usuário, responsável por interpretar os comandos recebidos e reagir aos mesmos, de acordo com as funcionalidades desejadas para a aplicação.

As aplicações podem possuir interfaces estáticas ou dinâmicas; sendo chamadas de estáticas aquelas que apresentam dados fixos exibidos igualmente para todos os usuários, normalmente escritas inteiramente em *HTML* e *CSS*; já as interfaces dinâmicas podem exibir diferentes conteúdos e proporcionar uma gama incrivelmente maior de interações com o usuário, utilizando tópicos mais avançados de programação, outras linguagens em paralelo, como *JavaScript*, e consumo de APIs para comunicação com bancos de dados.

O *frontend* do presente trabalho é uma interface dinâmica escrita em *JavaScript*, utilizando a biblioteca *React* e seus recursos para a construção das funcionalidades necessárias para que a aplicação cumpra seu propósito, fornecendo uma boa experiência ao usuário.

O *React* possibilita a criação de UIs (*User Interfaces* ou Interfaces do Usuário) interativas com ferramentas declarativas, gerando um código limpo e fácil de depurar. A biblioteca permite a criação de componentes encapsulados que gerenciam o próprio estado e que podem ser reutilizados ao longo do desenvolvimento, evitando repetição e gerando um código mais limpo, além de fornecer diversas ferramentas que permitem atualizar renderizações de tela de forma veloz e eficiente. (Facebook Open Source, 2022)

5.3.1 Arquitetura de frontend

A arquitetura da aplicação foi desenvolvida de forma que a organização pudesse ser mantida ao localizar os arquivos, categorizando-os de acordo com o contexto lógico e com a camada da aplicação em que estão inseridos. O padrão implementado está descrito na Figura 5.3.

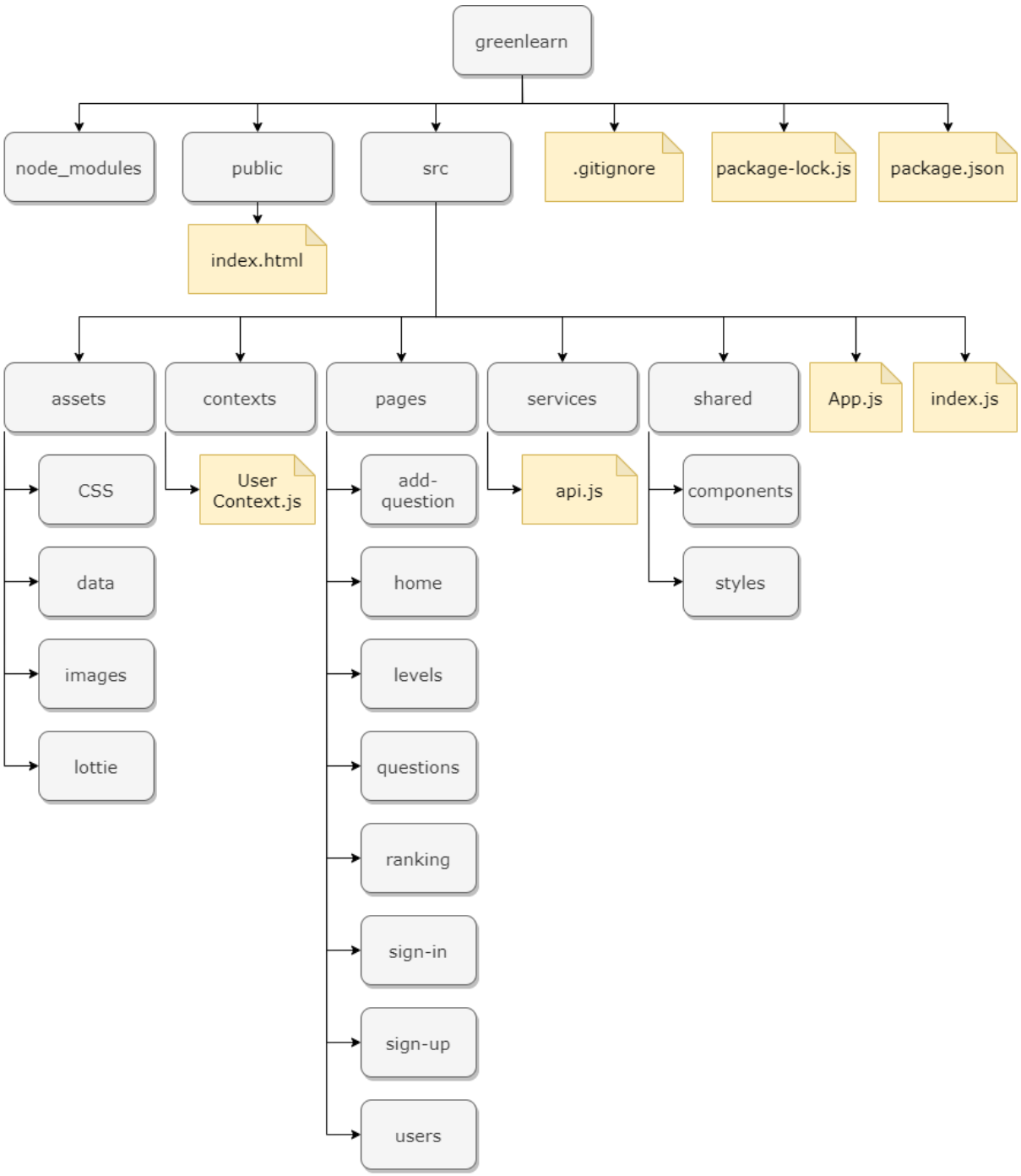


Figura 5.3 – Hierarquia de diretórios e arquivos do frontend do projeto *greenlearn*. Fonte: Acervo do autor.

Todas as pastas e arquivos relacionados a construção da lógica da aplicação estão contidos na pasta *src*, que está subdividida de acordo com o papel desempenhado pelos arquivos em questão na aplicação.

A pasta *assets* é onde estão armazenados os recursos locais da aplicação, na qual estão contidos os diretórios *CSS*, que armazena o arquivo *reset.css* utilizado para reconfiguração das estilizações padrão dos navegadores antes da aplicação das estilizações da aplicação; a pasta *data*, que contém algumas listas de objetos utilizados para configuração

de cores; a pasta *images*, contendo as imagens utilizadas no *layout* da aplicação; e a pasta *lottie*, que contem arquivos *json* referentes a animações implementadas na interface.

O diretório *contexts* contém o arquivo *UserContext.js*, que realiza a configuração de um contexto para o usuário. Os contextos são utilizados como uma forma de armazenar e transferir informações para componentes da aplicação que não possuem uma ligação direta entre si. Assim sendo, foi criado um contexto de usuário para que os dados ali armazenados, tais como nome e *token* de acesso, possam ser utilizados por qualquer parte da aplicação.

A pasta *pages* se responsabiliza por armazenar os diretórios referentes a cada página da aplicação. Cada uma dessas subpastas contém um arquivo principal responsável pela renderização e todos os componentes necessários para tal.

A pasta *service* contém a lógica de comunicação com a API. Nela, está contido o arquivo *api.js*, que contém a URL do *deploy* do *backend* realizado no *heroku* e todas as funções responsávies por disparar as requisições. O processo de disparo das requisições é feito através dos métodos da biblioteca *axios*, um cliente HTTP baseado em promessas e isomórfico, isto é, pode ser executado no navegador e no ambiente *node.js* com a mesma base de código (Axios, 2022). Na aplicação, é utilizado em funções assíncronas responsáveis por disparar requisições para rotas específicas da API anteriormente descrita.

O diretório *shared* tem por objetivo armazenar componentes compartilhados por diversas páginas. Possui dois subdiretórios: o *components*, que armazena os componentes *react* que aparecem em mais de uma página, como o *header*, que representa o menu topo; e o diretório *styles* que armazena os componentes de estilo compartilhados, construídos através da biblioteca *styled components*. Trata-se de uma biblioteca para *React* que permite a criação de estilos a nível de componente, utilizando CSS, além de remover o mapeamento generalizado entre componentes e estilos, permitindo flexibilidade de estilizações. (STYLED COMPONENTS, 2022)

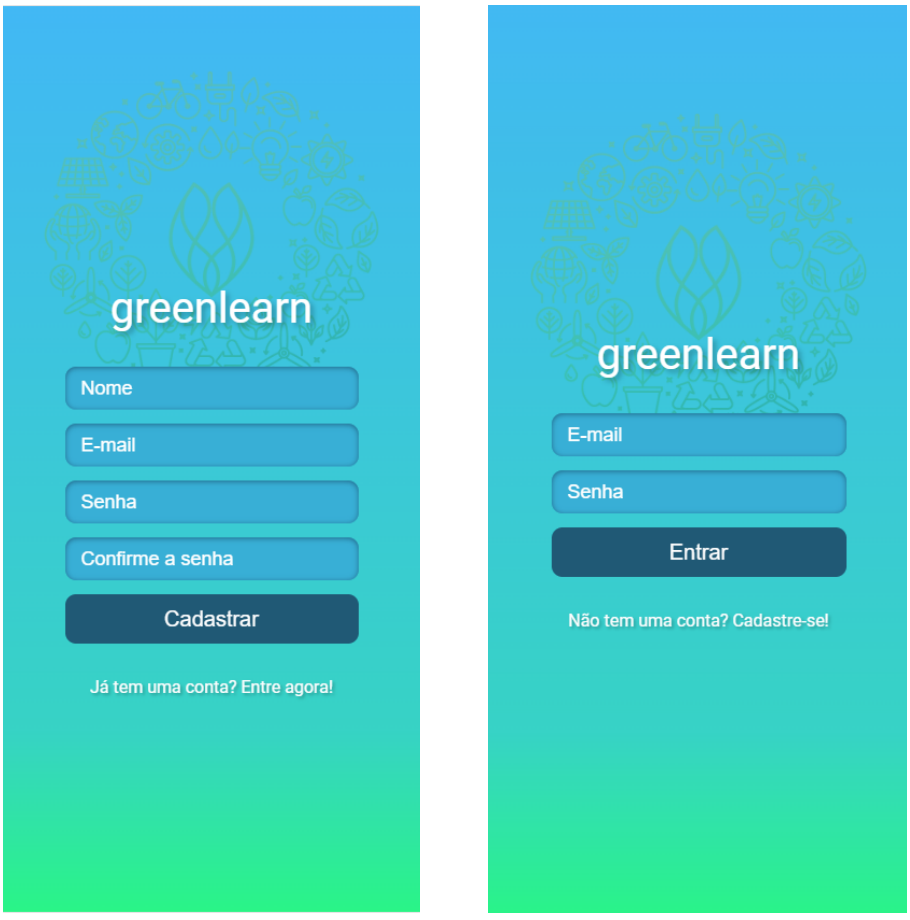
O arquivo *index.js* carrega o código responsável por renderizar as páginas no *DOM* pré definido no *index.html* da pasta *public*. A definição das rotas existentes e a renderização condicional das páginas baseada na rota acessada ocorre no arquivo *App.js*.

5.3.2 Interface e navegação

Para a criação do *layout* da aplicação, um esboço foi construído usando a ferramenta de *design figma*, na qual foram testados padrões de cor e distribuição de elementos. A partir do esboço, a aplicação começou a ser construída utilizando o conceito de *mobile first*.

Grande parte das empresas ainda desenvolve aplicações utilizando um processo que assume a tela *desktop* como principal, fazendo adaptações posteriores para o *mobile*. Entretanto, com a grande expansão do uso dos *smartphones* e a importância que tomaram no desempenho de funções cotidianas, passaram a assumir o papel, de forma prática, de primeira tela. O *mobile first* é o conceito do desenvolvimento que se baseia em tratar o *mobile* como primeira tela, desenvolvendo aplicações a partir de telas menores e, posteriormente, construindo a responsividade para que se adapte a telas maiores, melhorando a experência do usuário em termos de performance, usabilidade e acesso ao conteúdo em *smartphones*.

As Figuras 5.4-a e 5.4-b mostram, respectivamente, as telas de cadastro e *login*.

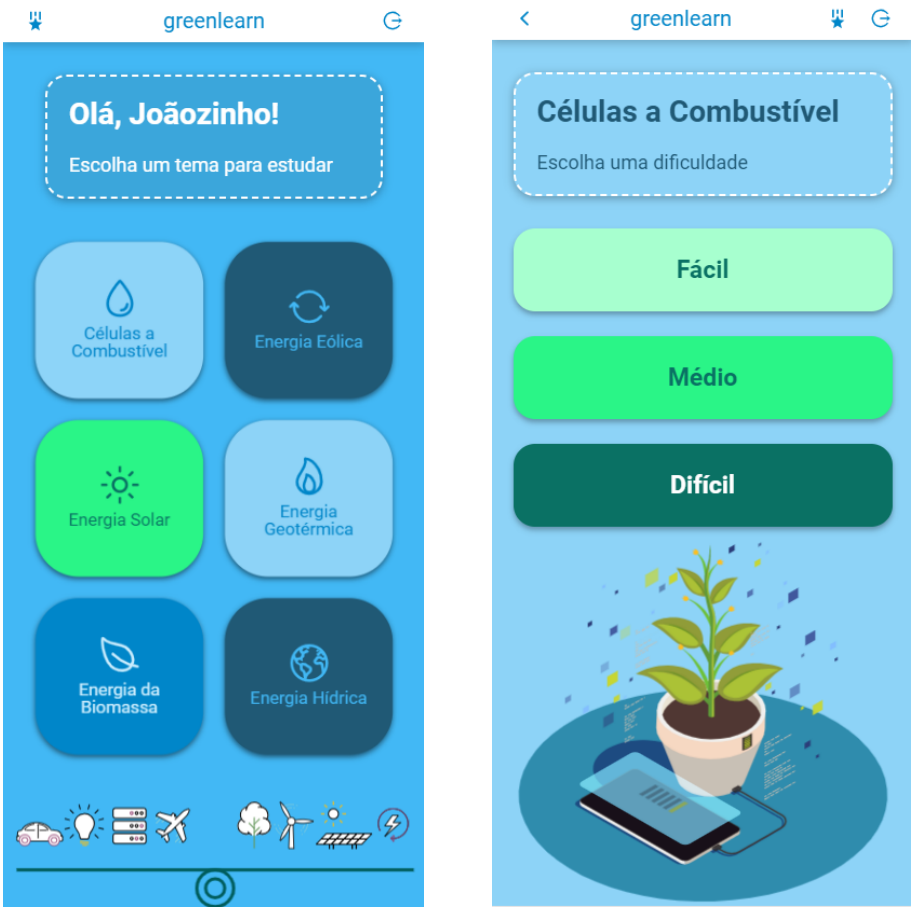


(a) Tela de cadastro do *greenlearn* (b) Tela de login do *greenlearn*

Figura 5.4 – Captura de tela das páginas de cadastro e *login* do projeto *greenlearn*. Fonte: Acervo do autor.

Todos os dados inseridos pelo usuário nos campos de *input* dos formulários são armazenados em variáveis de estado e, uma vez que o botão é pressionado, o corpo da requisição é montado com base no estado das variáveis no momento em que tal ação ocorre e uma requisição do tipo *post* é disparada para a API.

Após realizar o cadastro e o login, o usuário é redirecionado para a página de categorias, onde pode escolher um tema para estudar. Ao selecionar uma categoria, é enviado para a tela de níveis, onde é possível escolher o nível de dificuldade das perguntas que irá responder. Após o *login*, todas as páginas possuem um *header*, isto é, um menu superior de opções. O menu contem o logotipo da aplicação, que, ao ser clicado, redireciona de volta para a tela de categorias; um ícone de saída, que realiza o *logout* do usuário; um ícone de estrela, que leva ao *ranking* de usuários; e, exceto na tela de categorias, um botão de retorno, que leva o usuário de volta à última página acessada. As Figuras 5.5-a e 5.5-b mostram, respectivamente, as telas de categorias e de níveis.

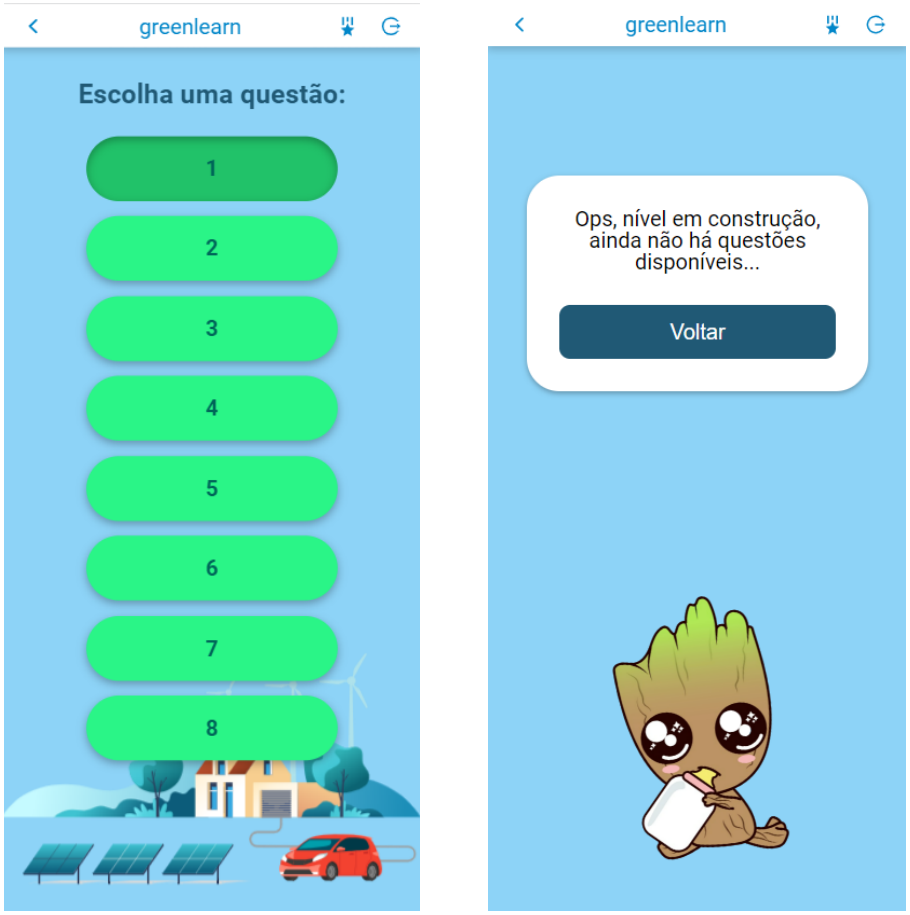


(a) Tela de categorias (b) Tela de níveis

Figura 5.5 – Captura de tela das páginas de categorias (*home*) e níveis do projeto *greenlearn*. Fonte: Acervo do autor.

Ao selecionar um nível, uma requisição é enviada à API para obter todas as questões correspondentes a categoria e ao nível selecionados. Uma vez que a requisição é

respondida, caso haja perguntas cadastradas para aquele nível, a renderização das perguntas é feita na forma de botões. A estilização do botão é feita com base no progresso do usuário. Caso o usuário ainda não tenha respondido corretamente a pergunta, é apresentado na forma de um botão não apertado. Caso a pergunta já tenha sido respondida pelo usuário, o botão aparece como pressionado. Tal comportamento pode ser observado na Figura 5.6-a. Em casos onde ainda não existem perguntas cadastradas na categoria para o nível em questão, uma mensagem informando que o nível está em construção, como mostra a Figura 5.6-b.



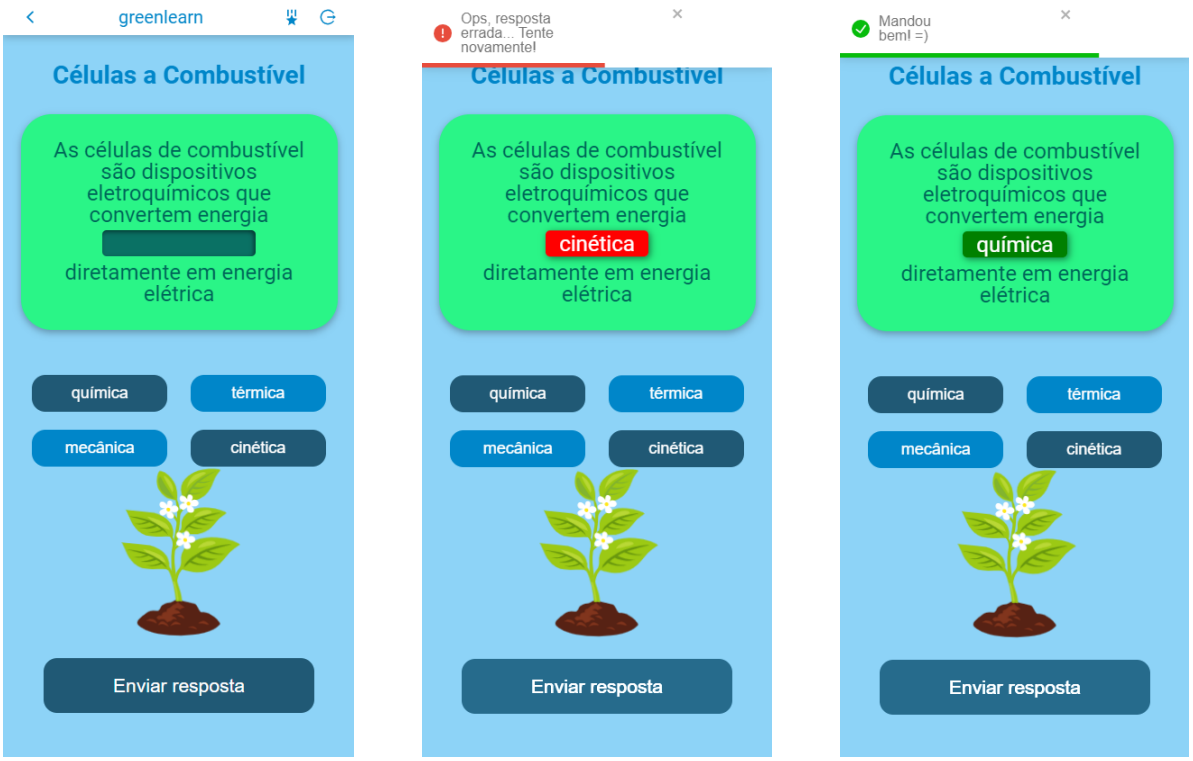
(a) Tela de perguntas (b) Tela de nível em construção

Figura 5.6 – Captura de tela das páginas de perguntas e de nível em construção do projeto *greenlearn*. Fonte: Acervo do autor.

A página da pergunta é renderizada quando o usuário seleciona qual pergunta deseja responder. Como mostra a Figura 5.7-a, a tela apresenta ao usuário uma frase contendo uma lacuna, que ocupa o lugar de uma palavra ou conjunto de palavras da frase em questão. Logo abaixo, quatro botões disponibilizam opções para preencher a lacuna, três deles estão errados e um completa corretamente a frase.

Ao clicar em uma opção, a lacuna é substituída pelo item escolhido. Clicando em enviar, ocorre a verificação da resposta e, caso esteja incorreta, a lacuna preenchida se

torna vermelha e um aviso é disparado na tela, informando ao usuário que a pergunta não foi respondida corretamente e solicitando que tente novamente, como pode ser observado na Figura 5.7-b. Caso a resposta esteja correta, a lacuna assume um tom mais intenso de verde e um *card* aparece na tela, informando que o usuário respondeu de forma correta a pergunta, como mostra a Figura 5.7-c. Em seguida, ocorre o redirecionamento para a página de perguntas, para que seja possível selecionar a próxima pergunta que se deseja responder.



(a) Tela de perguntas (b) Envio de resposta incorreta (c) Envio de resposta correta

Figura 5.7 – Captura de tela da página de perguntas e suas variações de acordo com o envio de respostas. Fonte: Aplicação desenvolvida pelo autor.

Ao clicar no ícone de estrela no menu superior, o usuário é redirecionado para a tela de *ranking*, que mostra sua colocação e pontuação, assim como dos cinco usuários mais bem colocados. Se o usuário não estiver entre as cinco melhores posições, um sexto *card* aparece, mostrando seus dados, como na Figura 5.8-a. Se estiver entre os cinco primeiros colocados, o sexto *card* não é renderizado, como na Figura 5.8-b. O *card* pertencente ao usuário pode ser identificado através de uma borda amarela que o destaca dos demais.

(a) Tela de ranking - usuário abaixo da quinta posição

(b) Tela de ranking - usuário entre os cinco primeiros colocados

Figura 5.8 – Captura de tela do ranking do projeto *greenlearn*. Fonte: Acervo do autor.

Quando um usuário administrador realiza *login*, a página inicial se apresenta com um *layout* ligeiramnte diferente. Um menu inferior é introduzido, dispondo de duas opções: gerenciar usuários ou adicionar conteúdo. O menu inferior pode ser observado na Figura 5.9-a.

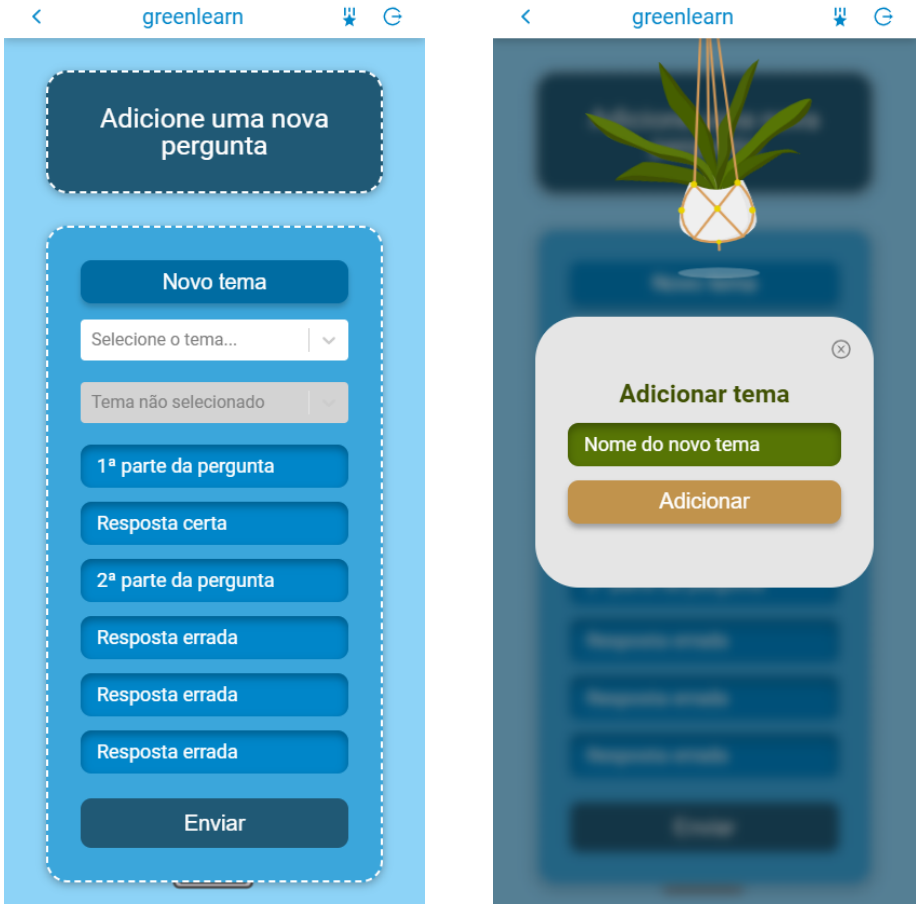
Ao clicar em gerenciar usuários, o administrador é redirecionado para uma página onde estão listados todos os usuários cadastrados na aplicação, ordenados alfabeticamente. Ao lado do nome do usuário, existe um botão que pode tomar duas formas: verde, com o texto "tornar adm", se o usuário não for administrador; ou "revogar adm", caso o usuário seja um administrador. Clicar nos botões realiza alteração das permissões do acesso do usuário escolhido, transformando-o em administrador ou em usuário comum. A página de gerenciamento de usuário pode ser vista na Figura 5.9-b.

(a) Tela inicial do administrador

(b) Tela de gerenciamento de usuários

Figura 5.9 – Captura de tela do menu de administrador e da página de usuários do *greenlearn*. Fonte: Acervo do autor.

Na página inicial, caso opte por clicar no botão de adicinar conteúdo, o administrador é redirecionado para uma página contendo uma sequência de *inputs* e botões, como mostra a Figura 5.10-a. Os *inputs* dizem respeito aos dados da nova pergunta a ser inserida, e o usuário deve selecionar a categoria, o nível, digitar a primeira parte da pergunta, referente ao texto que antecede a lacuna, a resposta certa, que preenche corretamente a lacuna, a segunda parte da pergunta (texto posterior a lacuna) e as três respostas erradas. Com os campos preenchidos, basta clicar no botão enviar e a pergunta será adicionada à aplicação. Caso o usuário deseje inserir uma nova categoria, pode clicar no botão "novo tema" e o *modal* mostrado na Figura 5.10-b será aberto, contendo o *input* para a inserção do título da categoria e o botão de envio.



(a) Tela adição de perguntas (b) Modal de adição de categorias

Figura 5.10 – Captura de tela das opções de adição de conteúdo do *greenlearn*. Fonte: Acervo do autor.

Vale ressaltar que animações foram adicionadas ao projeto, pois desempenham um papel importante ao construir uma experiência mais envolvente e agradável para os usuários. Além disso, têm diversas funções, como fornecer *feedback* visual imediato, orientar os usuários pelo *layout*, destacar informações relevantes, criar transições suaves entre as telas e adicionar elementos estéticos, que, dado o público alvo, são relevantes para a captura da atenção e interesse.

O *deploy* do *frontend* da aplicação foi feito utilizando a ferramenta *vercel* e pode ser acessado através da URL <https://greenlearn.vercel.app>.

6 RESULTADOS

6.1 TESTES EM AMBIENTE DE DESENVOLVIMENTO

Ao longo do desenvolvimento da aplicação, apenas testes manuais foram realizados. Testes automatizados não foram desenvolvidos por questão de gestão de tempo, haja vista a demanda causada pela escrita dos testes automatizados em comparação com o tempo para o desenvolvimento da aplicação e escrita do relatório. Por tanto, foi definido como prioridade a utilização carga horária de trabalho no desenvolvimento de funcionalidades que foram para ambiente de produção.

Os testes manuais foram realizados ao longo do desenvolvimento das funcionalidades, sendo de extrema importância para a identificação de erros, falhas e comportamentos inesperados em diferentes cenários, garantindo que problemas de usabilidade pudessem ser contornados.

A estruturação dos testes aconteceu, ao longo do desenvolvimento do *backend*, através da criação de requisições salvas em *collections* do *Thunderclient*, uma extensão do *Visual Studio Code (VS Code)* que permite fazer solicitações HTTP e visualizar os testes da API diretamente no editor de código. À medida em que as requisições são salvas na *collection*, basta rodá-la novamente para que todos os testes anteriormente realizados fossem feitos novamente, garantindo o bom funcionamento da aplicação com uma boa otimização de tempo.

6.2 TESTES EM AMBIENTE DE PRODUÇÃO

A testagem em ambiente de produção aconteceu de duas formas: as funcionalidades básicas passaram por um teste de aceitação contextualizado e o lançamento das funcionalidades seguiu os padrões de um teste de *rollout* gradual.

6.2.1 Teste de aceitação

O teste de aceitação tem como objetivo verificar se a aplicação atende aos requisitos e expectativas do usuário final, garantindo que ela funcione conforme o esperado e seja utilizável em diferentes cenários. Durante sua execução, um pequeno grupo controlado de usuários se responsabiliza pela verificação de diversas funcionalidades da aplicação, como

a interação com os elementos da tela, a validação de entradas e a saída correta de informações, registro de dados e demais elementos que compõem as funcionalidades oferecidas, além de determinar se a aplicação cumpre seu propósito.

Para a realização do teste de aceitação com usuários controlados, foi desenvolvida uma versão paralela e simplificada da aplicação, mantendo suas funcionalidades principais. Trata-se de uma adaptação de *layout* e banco de dados do *greenlearn* feita no momento em que as funcionalidades básicas (cadastro, login e resposta de perguntas) foram finalizadas. O objetivo é avaliar as principais funcionalidades em um ambiente próximo ao autor, de forma a conseguir *feedbacks* acessíveis de usuários orgânicos, que vejam a aplicação com um interesse real e similar ao do público-alvo do *greenlearn* teriam: a fixação e obtenção de novos conhecimentos.

A aplicação foi adaptada para ser inserida no contexto da disciplina de Projeto Conceitual de Sistemas Espaciais, do curso de Engenharia Aeroespacial da Universidade Federal de Santa Maria, e foi utilizada como ferramenta de estudo por seis alunos, incluindo o autor, que, realizaram uma prova da disciplina no dia seguinte. A aplicação foi nomeada *spacestudy* e as capturas de tela de sua interface estão apresentadas na Figura 6.1

(a) Tela de cadastro (b) Tela de login (c) Tela de categorias

(d) Tela de temas (e) Tela de perguntas (f) Tela da pergunta

Figura 6.1 – Capturas de tela da aplicação *spacestudy*. Fonte: Acervo do autor.

Para a realização do teste, foi construído um formulário contendo perguntas capazes de coletar informações relevantes sobre a performance da aplicação e sobre a experiência do usuário. Com o objetivo de preservar a autenticidade dos dados, o formulário foi enviado para cinco dos seis alunos que utilizaram a aplicação na disciplina de Projeto Conceitual de Sistemas Espaciais, excluindo o autor do artigo. Adicionalmente, o formulário foi projetado para garantir o anonimato das respostas, não sendo necessário fornecer qualquer identificação por parte dos participantes. Essas medidas foram adotadas visando assegurar a integridade e a imparcialidade das informações coletadas. No final, foram adicionadas algumas perguntas para entender a viabilidade de implementação de uma ferramenta similar para estudo em outras disciplinas do curso de Engenharia Aeroespacial na UFSM. Abaixo estão as perguntas realizadas:

1. De 0 à 10, qual o seu nível de satisfação geral com a aplicação?
2. De 0 à 10, quanto você acredita que a aplicação contribuiu positivamente para o seu desempenho na prova?
3. Para além da prova, de 0 à 10, em que medida você acredita que a aplicação ajudou a consolidar seus conhecimentos sobre os temas nela abordados?
4. De 0 à 10, como você avalia a experiência de usabilidade da aplicação?
5. Na sua opinião, quais são os pontos fortes da aplicação no processo de aprendizado e reforço de conteúdo?
6. Em que aspectos a aplicação poderia ser melhorada para tornar o estudo mais eficiente?
7. Você encontrou dificuldades ao utilizar a aplicação? Se sim, quais?
8. Você acredita que a aplicação possui potencial para ser implementada no estudo de outras disciplinas e conteúdos do curso de Engenharia Aeroespacial na UFSM? Se sim, quais conteúdos e/ou disciplinas você acha que seriam mais interessantes de serem abordados por meio dessa aplicação?
9. Por favor, compartilhe quaisquer comentários, sugestões, opiniões ou ideias adicionais (opcional).

Os usuários expressaram um alto nível de satisfação geral com a aplicação, atribuindo notas altas em relação à sua experiência e relatando que o *greenlearn* contribuiu positivamente para seu desempenho na prova e ajudou a consolidar outros conhecimentos relevantes para a disciplina, como pode ser visto no gráfico da Figura 6.2, que indica a média das respostas obtidas para as quatro primeiras perguntas.

A usabilidade da aplicação foi elogiada, com destaque para a forma como o conteúdo foi apresentado por meio de questões, facilitando a revisão e a fixação dos conceitos, e a facilidade e praticidade de uso. Além disso, os usuários sugeriram algumas melhorias para tornar o estudo mais eficiente, como o aumento do banco de questões, a inclusão de justificativas para as respostas e a inclusão de referências ou sugestões de estudos adicionais para validar os conhecimentos adquiridos.

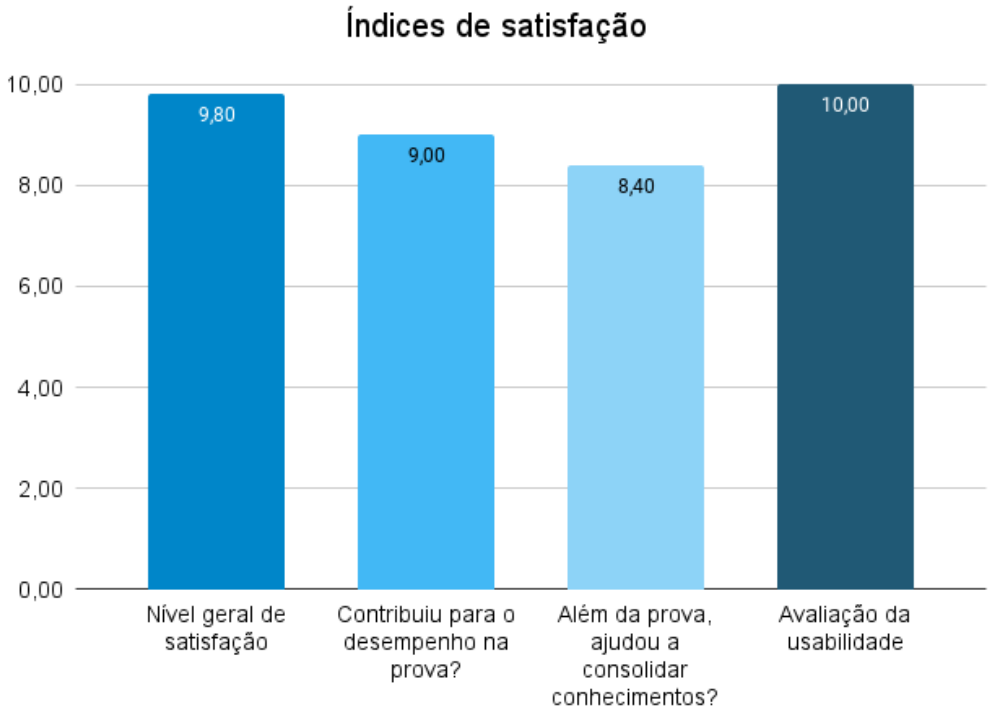


Figura 6.2 – Resultado da pesquisa de satisfação do teste de aceitação. Fonte: Acervo do autor.

É relevante mencionar que nenhum dos usuários relatou dificuldades ao utilizar o aplicativo, demonstrando sua eficiência e facilidade de uso e todos concordaram que o *greenlearn* possui potencial para ser implementado em outras disciplinas e conteúdos do curso de Engenharia Aeroespacial na UFSM. Em relação aos conteúdos e disciplinas mais interessantes de serem abordados pela aplicação, surgiram os temas propulsão, aerodinâmica, desempenho, ciência dos materiais, materiais aeroespaciais, projeto conceitual de aeronaves, sistemas de aeronaves, e as disciplinas de física e química. O gráfico da Figura 6.3 mostra os tópicos que surgiram, assim como a proporção que eles representam em relação ao total de itens sugeridos.

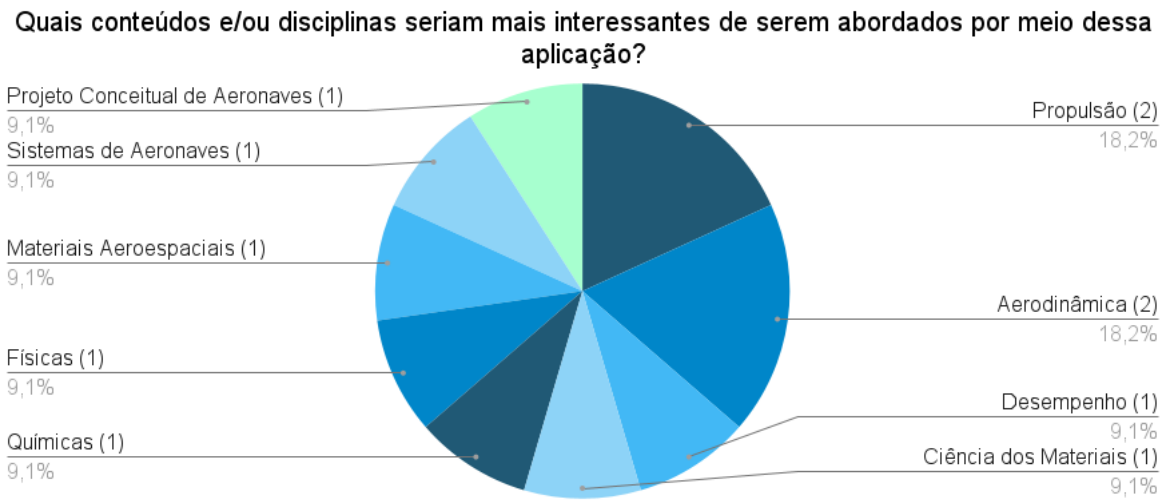


Figura 6.3 – Sugestões de disciplinas por parte dos participantes do teste de aceitação. Fonte: Acervo do autor.

É interessante ressaltar que um dos usuários observou que o *layout* do aplicativo *spacestudy*, com cores mais escuras, relacionadas ao tema espacial, transmitia uma sensação de cansaço. O *spacestudy* não passou pelo processo cuidadoso de estudo e criação de identidade visual desenvolvido para o *greenlearn*. Portanto, a informação fornecida na avaliação destaca a importância de considerar a experiência do usuário e a psicologia das cores na concepção de aplicativos, a fim de garantir que o usuário disponha de um ambiente visualmente adequado ao propósito da aplicação e a sua expectativa de tempo de uso.

6.2.2 Teste de *rollout* gradual

O teste de *rollout* gradual, também conhecido como *rollout* progressivo, é uma estratégia de implementação de *software* em que as mudanças ou atualizações são liberadas gradualmente para um subconjunto de usuários antes de serem disponibilizadas para o público geral. Nesse tipo de teste, a nova versão é lançada para uma pequena porcentagem de usuários, e o comportamento e a estabilidade são observados antes de expandir a disponibilidade para um número maior de usuários.

O objetivo do teste de rollout gradual é minimizar os riscos associados a mudanças significativas no *software*. Ao lançar a nova versão apenas para um grupo limitado, é possível identificar e corrigir problemas e falhas de maneira mais rápida e eficiente, antes que afetem um público maior. Isso também permite que o desenvolvedor colete avaliações valiosas dos usuários iniciais, que podem ser usadas para fazer ajustes e melhorias antes de uma implantação em larga escala. Para o *greenlearn*, os usuários de teste foram a equipe do projeto Energizando o Futuro e alguns alunos e professores do curso de Engenharia

Aeroespacial, que se voluntariaram a participar do processo.

No momento em que as primeiras funcionalidades foram concluídas, a aplicação foi disponibilizada para que os participantes pudessem testá-la e trazer opiniões até o autor sobre a identificação de falhas (*bugs*) e necessidades de implementação de novas ferramentas.

Ao longo do processo, opiniões interessantes e capazes de validar o fluxo jornada de usuário planejado surgiram, como a necessidade de diferentes permissões de usuário para que os membros do projeto pudessem adicionar novos conteúdos de forma independente, sem a necessidade de um desenvolvedor. Ademais, foram identificadas pequenas lacunas, como problemas relacionados ao funcionamento do botão de voltar. Em vez de retornar à página anterior dentro do fluxo de uso da aplicação, o botão redirecionava para a última página visitada pelo usuário. Essa inconsistência resultava em uma experiência não tão fluida e contínua em certos casos, mas pôde ser percebida e corrigida de forma rápida e sem prejuízos à experiência do usuário final.

Um dos pontos mais recentes identificados pelo teste de *roullout* gradual foi a necessidade de criação de uma ferramenta de edição de conteúdo, uma vez que erros podem ser cometidos no momento de inserção de novas perguntas. A ferramenta, portanto, foi adicionada às sugestões levantadas pelos usuários do teste de aceitação como melhorias futuras a serem adicionadas ao projeto.

7 CONCLUSÃO

Através do presente trabalho, foi possível realizar o desenvolvimento e análise da aplicação *greenlearn*, um aplicativo educacional que tem como objetivo complementar o ensino de sustentabilidade e energias renováveis no âmbito do projeto de extensão "Energizando o Futuro". Ao longo do processo de desenvolvimento de produto, foram realizadas pesquisas de mercado, estudos de interface e experiência do usuário e implementação da aplicação em ambiente controlado, metrificando através de testes de desenvolvimento e de aceitação. Os resultados obtidos revelaram uma alta satisfação dos usuários com a aplicação, destacando sua contribuição positiva no desempenho acadêmico, consolidação de conhecimentos e usabilidade.

As informações obtidas a partir da avaliação dos usuários forneceram importantes diretrizes para o aprimoramento da aplicação, além de revelar seu potencial para ser implementada em outros contextos educacionais, ampliando seu escopo para mais áreas, como disciplinas e conteúdos do curso de Engenharia Aeroespacial na Universidade Federal de Santa Maria.

Em suma, o desenvolvimento e análise da aplicação *greenlearn* demonstraram sua eficácia como ferramenta educacional. Através do engajamento dos usuários e da manutenção contínua de conteúdo, a aplicação possui o potencial de impactar positivamente o processo de aprendizagem e fortalecer o conhecimento, contribuindo para o aprimoramento do ensino nas temáticas abordadas.

REFERÊNCIAS BIBLIOGRÁFICAS

APPS, B. of. **Education App Market Statistics & Trends**. 2022. <https://www.businessofapps.com/data/education-app-market/#:~:text=Education%20apps%20generated%20%247%20billion,CAGR%20of%208.9%25%20until%202030>. Acessado em 03 de julho de 2023.

AUSUBEL, D. P. **The Psychology of Meaningful Verbal Learning**. New York: Grune Stratton, 1963.

Axios. **Axios v0.27.2 Documentation**. 2022. Acesso em 22 ago 2022. Disponível em: <https://axios-http.com/ptbr/docs/intro>.

BIRREN, F. **Color Psychology and Color Therapy**. Secaucus, NJ: Citadel Press, 1978.

COTTER, R. Exploring concepts and applications of benchmarking. **Leading and Managing; v.3 n.1 p.1-8; Autumn 1997**, Australian Council for Educational Leaders, Penrith, NSW, Australia, v. 3, n. 1, p. 18, 1997. Disponível em: <https://search.informit.org/doi/10.3316/aeipt.83163>.

DWECK, C. S. **Mindset: The new psychology of success**. New York: Random House, 2006.

Facebook Open Source. **React v17.0.2 Documentation**. 2022. Acesso em 22 ago 2022. Disponível em: <https://reactjs.org/docs/getting-started.html>.

GARRETT, J. J. **The Elements of User Experience**. Berkeley, CA: New Riders, 2002.

HASSENZAHL, M.; TRACTINSKY, N. User experience - a research agenda. **Behaviour & Information Technology**, Taylor & Francis, v. 25, n. 2, p. 91–97, 2006.

HATTIE, J. **Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement**. Abingdon, UK: Routledge, 2009.

ITTEN, J. **The Elements of Color**. New York: John Wiley & Sons, 1970.

KOTLER, P. **Administração de Marketing**. São Paulo: Prentice Hall, 2000.

MCNAIR, C. J.; SCRIVEN, K. H. Benchmarking: A tool for continuous improvement. **Management Decision**, Emerald Publishing, v. 40, n. 8, p. 747–762, 2002.

NIELSEN, J.; NORMAN, D. **Usability Engineering**. [S.l.]: Elsevier Science, 1993.

NORMAN, D. **The Design of Everyday Things: Revised and Expanded Edition**. [S.l.]: Basic Books, 2013.

NOVAK, J. D.; GOWIN, D. B. **Learning How to Learn**. New York: Cambridge University Press, 1984.

Open JS Foundation. **Node.js v16.17.0 documentation**. 2021. Acesso em 22 ago 2022. Disponível em: <https://nodejs.org/dist/latest-v16.x/docs/api/>.

REEVE, J. Why teachers adopt a controlling motivating style toward students and how they can become more autonomy supportive. **Educational Psychologist**, v. 44, n. 3, p. 159–175, 2009.

SCHNEIDERMAN, B. **Designing the User Interface: Strategies for Effective Human-Computer Interaction**. [S.l.]: Pearson, 1998.

STYLED COMPONENTS. **Styled Components v5.3.5 Documentation**. 2022. Acesso em 22 ago 2022. Disponível em: <https://styled-components.com/docs>.

The PostgreSQL Global Development Group. **PostgreSQL 14.5 Documentation**. 2021. Acesso em 22 ago 2022. Disponível em: <https://www.postgresql.org/files/documentation/pdf/14/postgresql-14-A4.pdf>.