UNiSA
Universidade de Santo Amaro

Curso de graduação tecnológica em Analise e Desenvolvimento de Sistemas

Projeto integrador Programação Web

Lucas da Silva Damaso

Damaso Barber

Polo Educacional de Embu Guaçu – SP 2025

Lucas da Silva Damaso

Damaso Barber

Trabalho de Projeto Integrador apresentado como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, no Curso de Graduação Tecnológica em Análise e Desenvolvimento de Sistemas da Universidade Santo Amaro – UNISA.

Luis Fernando dos Santos Pires

Link do Repositorio: https://github.com/luucdamaso/damasoBarber

Polo Educacional de Embu Guaçu – SP 2025

RESUMO

O presente trabalho apresenta o desenvolvimento de um sistema web completo para gestão de barbearias, denominado Damaso Barber. O sistema foi desenvolvido utilizando tecnologias modernas de programação web, incluindo React no frontend e Node.js com Express no backend, integrando-se a um banco de dados MySQL através do Prisma ORM. O objetivo principal do projeto é fornecer uma solução digital eficiente para o gerenciamento de agendamentos, clientes e serviços em estabelecimentos de barbearia. A aplicação implementa funcionalidades essenciais como autenticação de usuários, cadastro e gerenciamento de clientes, catálogo de serviços, sistema de agendamentos e dashboard administrativo. A arquitetura adotada segue o padrão de aplicações de página única (SPA - Single Page Application), proporcionando uma experiência de usuário fluida e responsiva. O desenvolvimento seguiu práticas modernas de engenharia de software, incluindo separação de responsabilidades, componentização e uso de APIs RESTful para comunicação entre frontend e backend. Os resultados demonstram a viabilidade e eficácia da solução proposta para a digitalização de processos em estabelecimentos do setor de serviços de beleza e cuidados pessoais.

Palavra-chave: Desenvolvimento Web. Sistema de Gestão. React. Node.js. Barbearia.

ABSTRACT

This paper presents the development of a complete web system for barbershop management, called Damaso Barber. The system was developed using modern web programming technologies, including React on the frontend and Node.js with Express on the backend, integrating with a MySQL database through Prisma ORM. The main objective of the project is to provide an efficient digital solution for managing appointments, clients and services in barbershop establishments. The application implements essential functionalities such as user authentication, client registration and management, service catalog, appointment system and administrative dashboard. The adopted architecture follows the Single Page Application (SPA) pattern, providing a fluid and responsive user experience. The development followed modern software engineering practices, including separation of concerns, componentization and use of RESTful APIs for communication between frontend and

backend. The results demonstrate the feasibility and effectiveness of the proposed solution for the digitalization of processes in establishments in the beauty and personal care services sector.

Keywords: Web Development. Management System. React. Node.js. Barbershop.

---

1. INTRODUÇÃO

A transformação digital tem revolucionado diversos setores da economia, e o segmento de serviços de beleza e cuidados pessoais não é exceção. Barbearias, que tradicionalmente operavam com métodos manuais de agendamento e controle de clientes, enfrentam desafios crescentes relacionados à organização, eficiência operacional e experiência do cliente. Neste contexto, a implementação de sistemas de gestão digital surge como uma necessidade estratégica para estabelecimentos que buscam competitividade e excelência no atendimento.

O desenvolvimento de aplicações web modernas tem sido facilitado pelo surgimento de tecnologias robustas e frameworks eficientes que permitem a criação de sistemas complexos com maior produtividade e qualidade. JavaScript, que inicialmente era utilizado apenas para adicionar interatividade a páginas web, evoluiu para se tornar uma linguagem completa para desenvolvimento full-stack, possibilitando a construção tanto do frontend quanto do backend de aplicações com uma única linguagem de programação.

O projeto Damaso Barber foi concebido para atender às necessidades específicas de gestão de barbearias, oferecendo funcionalidades que abrangem desde o cadastro de clientes até o controle de agendamentos e serviços. A escolha das tecnologias utilizadas baseou-se em critérios de modernidade, escalabilidade, facilidade de manutenção e disponibilidade de recursos da comunidade de desenvolvedores.

1.1 Contextualização do Problema

Estabelecimentos de barbearia frequentemente enfrentam dificuldades no gerenciamento de agendamentos, controle de clientes e organização de serviços. Métodos tradicionais baseados em agendas físicas ou planilhas eletrônicas apresentam limitações significativas, incluindo falta de integração entre informações, dificuldade de acesso remoto, ausência de histórico organizado e propensão a erros humanos. Além disso, a experiência do cliente pode ser comprometida por processos

manuais que não oferecem praticidade e agilidade.

A digitalização desses processos através de um sistema web integrado representa uma oportunidade de melhorar significativamente a eficiência operacional do estabelecimento, reduzir custos com retrabalho, minimizar conflitos de agendamento e proporcionar uma experiência superior aos clientes. Um sistema bem projetado também permite a geração de relatórios e análises que auxiliam na tomada de decisões gerenciais.

1.2 Justificativa

A relevância deste projeto manifesta-se em múltiplas dimensões. Do ponto de vista acadêmico, o desenvolvimento do sistema Damaso Barber proporciona a aplicação prática de conceitos fundamentais de programação web, arquitetura de software, banco de dados e engenharia de requisitos, consolidando conhecimentos adquiridos ao longo do curso de Análise e Desenvolvimento de Sistemas.

Do ponto de vista profissional, o projeto demonstra competências essenciais para o mercado de trabalho atual, incluindo domínio de tecnologias modernas amplamente utilizadas na indústria, capacidade de desenvolver soluções completas (full-stack) e habilidade de traduzir requisitos de negócio em funcionalidades técnicas.

Do ponto de vista social e econômico, o sistema desenvolvido tem potencial de beneficiar pequenos e médios empreendedores do setor de serviços, democratizando o acesso a ferramentas de gestão que anteriormente eram acessíveis apenas a grandes empresas. A solução proposta pode contribuir para a profissionalização do setor e para a melhoria da qualidade dos serviços prestados.

1.3 Delimitação do Escopo

O sistema Damaso Barber foi desenvolvido como uma aplicação web full-stack, abrangendo as seguintes funcionalidades principais: autenticação e autorização de usuários, cadastro e gerenciamento de clientes, cadastro e gerenciamento de serviços oferecidos, sistema de agendamentos com controle de horários e dashboard administrativo para visualização de informações consolidadas.

O projeto utiliza tecnologias modernas e consolidadas no mercado, incluindo React para o frontend, Node.js com Express para o backend, MySQL como sistema gerenciador de banco de dados e Prisma como ORM (Object-Relational Mapping). A arquitetura adotada segue o padrão de Single Page Application (SPA), com

comunicação entre frontend e backend através de APIs RESTful.

---

2. OBJETIVOS

2.1 Objetivo Geral

Desenvolver um sistema web completo para gestão de barbearias, integrando funcionalidades de cadastro de clientes, gerenciamento de serviços e controle de agendamentos, utilizando tecnologias modernas de programação web para proporcionar uma solução eficiente, escalável e de fácil utilização.

2.2 Objetivos Específicos

- Implementar um sistema de autenticação seguro utilizando JSON Web Tokens (JWT) e criptografia de senhas com bcrypt, garantindo a proteção de dados sensíveis dos usuários.
- Desenvolver uma interface de usuário intuitiva e responsiva utilizando React e bibliotecas de componentes modernos (Radix UI), proporcionando uma experiência de usuário consistente em diferentes dispositivos.
- Criar uma API RESTful robusta utilizando Node.js e Express, seguindo boas práticas de desenvolvimento backend e padrões de arquitetura de software.
- Implementar um modelo de dados relacional eficiente utilizando MySQL e Prisma ORM, garantindo integridade referencial e otimização de consultas.
- Desenvolver funcionalidades completas de CRUD (Create, Read, Update, Delete) para clientes, serviços e agendamentos, permitindo a gestão completa das operações da barbearia.
- Aplicar conceitos de componentização e reutilização de código no frontend, promovendo manutenibilidade e escalabilidade da aplicação.
- Integrar frontend e backend através de requisições HTTP assíncronas, utilizando a biblioteca Axios para comunicação eficiente entre as camadas da aplicação.

3. FUNDAMENTAÇÃO TEÓRICA

3.1 Desenvolvimento Web Moderno

O desenvolvimento web moderno caracteriza-se pela utilização de tecnologias e práticas que permitem a criação de aplicações complexas, escaláveis e com alta qualidade de experiência do usuário. Segundo a Mozilla Developer Network (MDN), o desenvolvimento web contemporâneo envolve a integração de múltiplas tecnologias, incluindo HTML para estruturação de conteúdo, CSS para estilização e JavaScript para interatividade e lógica de aplicação.

A evolução das aplicações web trouxe o conceito de Single Page Applications (SPA), onde a aplicação carrega uma única página HTML e atualiza dinamicamente o conteúdo conforme o usuário interage com a interface, sem necessidade de recarregar a página completa. Este paradigma proporciona uma experiência mais fluida e responsiva, semelhante a aplicações desktop ou móveis nativas.

O desenvolvimento full-stack refere-se à capacidade de trabalhar tanto no frontend (interface do usuário) quanto no backend (servidor e lógica de negócio) de uma aplicação. A University of Helsinki, através do curso Full Stack Open, define o desenvolvimento full-stack moderno como a criação de aplicações web utilizando JavaScript em toda a stack, desde a interface do usuário até o servidor e banco de dados.

3.2 Node.js e Programação Backend

Node.js é um ambiente de execução JavaScript de código aberto e multiplataforma que permite aos desenvolvedores criar aplicações de servidor utilizando JavaScript. Conforme a documentação oficial do Node.js, o ambiente executa o motor JavaScript V8, o núcleo do Google Chrome, fora do navegador, proporcionando alto desempenho e eficiência.

Uma das características fundamentais do Node.js é sua arquitetura assíncrona e

orientada a eventos. Uma aplicação Node.js executa em um único processo, sem criar uma nova thread para cada requisição. O Node.js fornece um conjunto de primitivas de I/O assíncronas em sua biblioteca padrão que impedem o bloqueio do código JavaScript. Quando o Node.js executa uma operação de I/O, como ler da rede, acessar um banco de dados ou o sistema de arquivos, em vez de bloquear a thread e desperdiçar ciclos de CPU esperando, o Node.js retoma as operações quando a resposta retorna.

Esta arquitetura permite que o Node.js lide com milhares de conexões simultâneas com um único servidor sem introduzir o ônus de gerenciar a concorrência de threads, que poderia ser uma fonte significativa de bugs. O Node.js tem uma vantagem única porque milhões de desenvolvedores frontend que escrevem JavaScript para o navegador agora podem escrever o código do lado do servidor sem a necessidade de aprender uma linguagem completamente diferente.

O Express é um framework web minimalista e flexível para Node.js que fornece um conjunto robusto de recursos para aplicações web e móveis. Ele facilita a criação de APIs RESTful, gerenciamento de rotas, middlewares para processamento de requisições e integração com diversos sistemas de template e bancos de dados.

3.3 React e Desenvolvimento Frontend

React é uma biblioteca JavaScript para construção de interfaces de usuário, desenvolvida e mantida pelo Facebook (Meta). Segundo a documentação da Mozilla Developer Network, o objetivo principal do React é minimizar os bugs que ocorrem quando desenvolvedores estão construindo interfaces de usuário, através do uso de componentes - peças de código autocontidas e lógicas que descrevem uma porção da interface do usuário.

No React, um componente é um módulo reutilizável que renderiza uma parte da aplicação geral. Os componentes podem ser grandes ou pequenos, mas geralmente são claramente definidos e servem a um único propósito óbvio. Os componentes podem ser compostos juntos para criar uma interface completa, e o React abstrai grande parte do trabalho de renderização, permitindo que os desenvolvedores se concentrem no design da interface.

O React utiliza JSX (JavaScript XML), uma extensão de sintaxe para JavaScript que permite escrever código semelhante a HTML dentro do JavaScript. Os componentes React retornam expressões JSX que definem o que o navegador renderiza no DOM

(Document Object Model). Esta abordagem declarativa torna o código mais legível e facilita o desenvolvimento de interfaces complexas.

O React implementa um conceito chamado Virtual DOM, que é uma representação em memória do DOM real. Quando o estado de um componente muda, o React primeiro atualiza o Virtual DOM, compara com a versão anterior e calcula a forma mais eficiente de atualizar o DOM real, minimizando operações custosas e melhorando o desempenho da aplicação.

3.4 Arquitetura de Aplicações Full-Stack

A arquitetura de aplicações web modernas geralmente segue o padrão cliente- servidor, onde o frontend (cliente) e o backend (servidor) são desenvolvidos como componentes separados que se comunicam através de APIs. Esta separação de responsabilidades traz diversos benefícios, incluindo escalabilidade independente, facilidade de manutenção e possibilidade de múltiplos clientes (web, mobile, desktop) consumirem a mesma API.

As APIs RESTful (Representational State Transfer) são um padrão arquitetural para criação de serviços web que utilizam os métodos HTTP (GET, POST, PUT, DELETE) para realizar operações sobre recursos. REST é um estilo de arquitetura que define um conjunto de restrições e propriedades baseadas em HTTP, proporcionando uma interface uniforme entre componentes e facilitando a escalabilidade e manutenibilidade do sistema.

A comunicação entre frontend e backend em aplicações modernas geralmente utiliza o formato JSON (JavaScript Object Notation) para troca de dados. JSON é um formato leve de intercâmbio de dados que é fácil para humanos lerem e escreverem, e fácil para máquinas parsearem e gerarem. A utilização de JSON permite uma integração natural com JavaScript, tanto no cliente quanto no servidor.

3.5 Bancos de Dados e ORM

Bancos de dados relacionais organizam dados em tabelas com linhas e colunas, estabelecendo relações entre diferentes entidades através de chaves primárias e estrangeiras. MySQL é um dos sistemas de gerenciamento de banco de dados relacionais mais populares, conhecido por sua confiabilidade, desempenho e facilidade de uso.

ORM (Object-Relational Mapping) é uma técnica de programação que permite aos

desenvolvedores interagir com bancos de dados relacionais utilizando paradigmas de programação orientada a objetos. Prisma é um ORM moderno para Node.js e TypeScript que facilita o acesso a bancos de dados, fornecendo uma interface type-safe, migrações automáticas e um cliente de consulta intuitivo.

O Prisma utiliza um schema declarativo onde os desenvolvedores definem seus modelos de dados, e o ORM gera automaticamente o cliente de banco de dados, as migrações e a documentação. Esta abordagem reduz significativamente o código boilerplate e minimiza erros relacionados a consultas SQL manuais.

3.6 Autenticação e Segurança

A autenticação é o processo de verificar a identidade de um usuário, enquanto a autorização determina quais recursos e operações um usuário autenticado pode acessar. Em aplicações web modernas, JSON Web Tokens (JWT) são amplamente utilizados para implementar autenticação stateless.

JWT é um padrão aberto (RFC 7519) que define uma maneira compacta e autocontida de transmitir informações entre partes como um objeto JSON. Estas informações podem ser verificadas e confiadas porque são assinadas digitalmente. JWTs podem ser assinados usando um segredo (com o algoritmo HMAC) ou um par de chaves pública/privada usando RSA ou ECDSA.

A segurança de senhas é implementada através de funções de hash criptográficas, como bcrypt, que transformam a senha original em uma string irreversível. Bcrypt incorpora um salt (valor aleatório) automaticamente e permite configurar o custo computacional, tornando ataques de força bruta impraticáveis mesmo com hardware moderno.

---

4. METODOLOGIA

4.1 Tipo de Pesquisa

Este projeto caracteriza-se como uma pesquisa aplicada de natureza qualitativa, com desenvolvimento de um produto tecnológico. A abordagem metodológica combina pesquisa bibliográfica para fundamentação teórica e desenvolvimento experimental para implementação do sistema.

4.2 Tecnologias Utilizados

O sistema Damaso Barber foi desenvolvido utilizando as seguintes tecnologias e ferramentas:

Frontend:

- React 19.0.0: Biblioteca JavaScript para construção de interfaces de usuário
- TypeScript 5.6.3: Superset tipado do JavaScript para maior segurança e produtividade
- Vite 7.1.7: Build tool moderna para desenvolvimento rápido
- Wouter 3.3.5: Biblioteca de roteamento leve para React
- Radix UI: Conjunto de componentes acessíveis e customizáveis
- Tailwind CSS 4.1.14: Framework CSS utility-first para estilização
- Axios: Cliente HTTP para requisições à API
- React Hook Form 7.64.0: Biblioteca para gerenciamento de formulários
- Zod 4.1.12: Biblioteca de validação de schemas TypeScript-first

Backend:

- Node.js: Ambiente de execução JavaScript no servidor
- Express 4.18.2: Framework web para Node.js
- Prisma 5.22.0: ORM moderno para Node.js e TypeScript
- MySQL: Sistema gerenciador de banco de dados relacional
- JSON Web Token 9.0.0: Implementação de JWT para autenticação
- Bcrypt 5.1.0: Biblioteca para hashing de senhas
- CORS 2.8.5: Middleware para habilitar CORS (Cross-Origin Resource Sharing)
- Dotenv 16.0.0: Carregamento de variáveis de ambiente

Ferramentas de Desenvolvimento:

- Git: Sistema de controle de versão
- Visual Studio Code: Editor de código
- Nodemon 2.0.22: Utilitário para reinicialização automática do servidor durante desenvolvimento

- Prettier: Formatador de código

4.3 Arquitetura do Sistema

O sistema foi desenvolvido seguindo uma arquitetura em camadas, com separação clara entre frontend, backend e banco de dados. A arquitetura adotada pode ser descrita da seguinte forma:

Camada de Apresentação (Frontend): Responsável pela interface do usuário e experiência de interação. Implementada como uma Single Page Application (SPA) utilizando React, com componentes reutilizáveis e gerenciamento de estado através de Context API. A comunicação com o backend ocorre através de requisições HTTP assíncronas.

Camada de Aplicação (Backend): Responsável pela lógica de negócio, processamento de requisições e comunicação com o banco de dados. Implementada utilizando Node.js e Express, seguindo o padrão MVC (Model-View-Controller) adaptado para APIs RESTful, onde os controllers processam requisições, os services contêm a lógica de negócio e os models representam as entidades do domínio.

Camada de Dados: Responsável pelo armazenamento persistente de informações. Utiliza MySQL como sistema gerenciador de banco de dados, com acesso intermediado pelo Prisma ORM, que fornece uma camada de abstração type-safe e facilita operações de consulta e manipulação de dados.

4.4 Modelagem de Dados

O modelo de dados do sistema foi projetado para atender aos requisitos de gestão de barbearias, incluindo as seguintes entidades principais:

User (Usuário): Representa os usuários do sistema com capacidade de autenticação. Atributos incluem identificador único (UUID), email, nome, senha criptografada, papel (role) para controle de acesso e data de criação.

Client (Cliente): Representa os clientes da barbearia. Atributos incluem identificador único, nome, telefone, email opcional, observações e data de cadastro. Possui relacionamento um-para-muitos com agendamentos.

Service (ServiÇo): Representa os serviços oferecidos pela barbearia. Atributos incluem identificador único, nome do serviço, preço, duração em minutos e data de cadastro.

Booking (Agendamento): Representa os agendamentos realizados. Atributos incluem identificador único, referência ao cliente, data/hora de início, data/hora de término, serviço solicitado, status (agendado, cancelado, concluído) e data de criação. Possui relacionamento muitos-para-um com clientes.

4.5 Implementação de Funcionalidades

Autenticação e Autorização: Implementada utilizando JWT para geração de tokens de acesso após login bem-sucedido. As senhas são criptografadas utilizando bcrypt antes de serem armazenadas no banco de dados. Middlewares de autenticação verificam a validade do token em rotas protegidas, e middlewares de autorização verificam o papel do usuário para acesso a funcionalidades administrativas.

Gerenciamento de Clientes: Funcionalidades completas de CRUD permitindo cadastro de novos clientes com validação de dados, listagem de clientes com opções de busca e filtro, visualização de detalhes e histórico de agendamentos, edição de informações cadastrais e exclusão de registros com verificação de dependências.

Gerenciamento de Serviços: Sistema de catálogo de serviços com cadastro incluindo nome, preço e duração, listagem de todos os serviços disponíveis, edição de informações e valores, e exclusão de serviços não utilizados em agendamentos.

Sistema de Agendamentos: Funcionalidade central do sistema permitindo criação de novos agendamentos associando cliente e serviço, visualização de agenda com filtros por data e status, edição de agendamentos existentes, cancelamento de agendamentos e marcação de agendamentos como concluídos.

Dashboard Administrativo: Interface consolidada apresentando visão geral das operações, incluindo estatísticas de agendamentos, lista de próximos agendamentos e indicadores de desempenho.

4.6 Processo de Desenvolvimento

O desenvolvimento seguiu uma abordagem iterativa e incremental, com as seguintes etapas:

1. **Levantamento de Requisitos:** Identificação das necessidades do sistema através de análise do domínio de barbearias e definição de funcionalidades essenciais.

2. **Modelagem de Dados:** Criação do schema do banco de dados utilizando Prisma

Schema Language, definindo entidades, atributos e relacionamentos.

3. **Desenvolvimento do Backend:** Implementação da API RESTful com endpoints para todas as operações necessárias, incluindo autenticação, CRUD de entidades e lógica de negócio.

4. **Desenvolvimento do Frontend:** Criação de componentes React para interface do usuário, implementação de rotas e navegação, integração com a API backend e implementação de gerenciamento de estado.

5. **Testes e Validação:** Testes funcionais de todas as funcionalidades implementadas, validação de segurança e autenticação, e testes de integração entre frontend e backend.

6. **Documentação:** Elaboração de documentação técnica do código, criação de documentação de usuário e preparação de material acadêmico (presente artigo).

---

5. RESULTADOS E DISCUSSÃO

5.1 Sistema Desenvolvido

O sistema Damaso Barber foi desenvolvido com sucesso, implementando todas as funcionalidades planejadas. A aplicação apresenta uma interface moderna e intuitiva, com navegação fluida entre as diferentes seções do sistema. A arquitetura adotada demonstrou-se adequada para os requisitos do projeto, proporcionando separação clara de responsabilidades e facilitando a manutenção e evolução do código.

Funcionalidades Implementados

Sistema de Autenticação: O sistema de autenticação implementado utilizando JWT e bcrypt demonstrou-se seguro e eficiente. Usuários podem realizar login fornecendo email e senha, recebendo um token de acesso que é armazenado no cliente e enviado em requisições subsequentes. O middleware de autenticação valida o token e extrai informações do usuário, permitindo controle de acesso granular.

Gestão de Clientes: A funcionalidade de gestão de clientes permite o cadastro completo de informações, incluindo dados de contato e observações relevantes. A interface de listagem apresenta todos os clientes cadastrados com opções de busca e filtro,

facilitando a localização de registros específicos. A visualização de detalhes de um cliente inclui seu histórico de agendamentos, proporcionando uma visão completa do relacionamento com o estabelecimento.

Catálogo de Serviços: O sistema de gerenciamento de serviços permite a manutenção de um catálogo atualizado dos serviços oferecidos pela barbearia. Cada serviço inclui informações de nome, preço e duração estimada, facilitando o planejamento de agendamentos e a comunicação de valores aos clientes.

Sistema de Agendamentos: A funcionalidade de agendamentos constitui o núcleo do sistema, permitindo o registro de horários marcados associando clientes aos serviços solicitados. O sistema controla o status de cada agendamento (agendado, cancelado, concluído), permitindo acompanhamento completo do ciclo de vida de cada atendimento. A interface de visualização de agenda facilita a organização da rotina do estabelecimento.

Dashboard Administrativo: O dashboard proporciona uma visão consolidada das operações, apresentando informações relevantes de forma organizada e acessível. Esta funcionalidade auxilia na tomada de decisões gerenciais e no acompanhamento do desempenho do estabelecimento.

5.2 Arquitetura e Tecnologias

A escolha de React para o frontend demonstrou-se acertada, proporcionando uma experiência de desenvolvimento produtiva através da componentização e reutilização de código. A biblioteca de componentes Radix UI contribuiu significativamente para a criação de uma interface acessível e consistente, enquanto o Tailwind CSS facilitou a estilização responsiva.

No backend, a combinação de Node.js com Express proporcionou um ambiente robusto e eficiente para implementação da API RESTful. O Prisma ORM simplificou significativamente a interação com o banco de dados MySQL, eliminando a necessidade de escrever queries SQL manualmente e proporcionando type-safety através da integração com TypeScript.

A arquitetura de Single Page Application adotada no frontend, com comunicação assíncrona com o backend através de APIs RESTful, proporcionou uma experiência de usuário fluida e responsiva, sem recarregamentos de página e com feedback imediato para as ações do usuário.

5.3 Desafios e Soluões

Durante o desenvolvimento, diversos desafios técnicos foram encontrados e superados. A implementação de autenticação segura exigiu compreensão aprofundada de conceitos de criptografia e gerenciamento de tokens. A solução adotada utilizando JWT demonstrou-se adequada, proporcionando autenticação stateless e escalável.

O gerenciamento de estado no frontend, especialmente para manter informações de autenticação e dados do usuário logado, foi implementado utilizando Context API do React, evitando prop drilling e centralizando a lógica de autenticação. Esta abordagem facilitou o controle de acesso a rotas protegidas e a personalização da interface baseada no usuário autenticado.

A validação de dados tanto no frontend quanto no backend foi implementada utilizando bibliotecas especializadas (Zod no frontend e validações customizadas no backend), garantindo integridade e consistência das informações armazenadas no banco de dados.

5.4 ContribuiÇões do Projeto

Este projeto contribui de múltiplas formas para o conhecimento e prática em desenvolvimento de software. Do ponto de vista técnico, demonstra a aplicação prática de tecnologias modernas e amplamente utilizadas na indústria, consolidando conhecimentos em programação full-stack, arquitetura de software e engenharia de requisitos.

Do ponto de vista acadêmico, o projeto integra conhecimentos de diversas disciplinas do curso, incluindo lógica de programação, estruturas de dados, banco de dados, engenharia de software e programação web, demonstrando a interdisciplinaridade necessária para o desenvolvimento de sistemas completos.

Do ponto de vista prático, o sistema desenvolvido tem potencial de aplicação real em estabelecimentos de barbearia, contribuindo para a digitalização e profissionalização do setor. A solução pode ser adaptada e expandida para atender necessidades específicas de diferentes estabelecimentos.

---

6. CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema Damaso Barber alcançou os objetivos propostos, resultando em uma aplicação web completa e funcional para gestão de barbearias. O projeto demonstrou a viabilidade de utilizar tecnologias modernas de programação web para criar soluções eficientes e escaláveis para problemas reais do setor de serviços.

A experiência de desenvolvimento proporcionou aprendizado significativo em múltiplas áreas do desenvolvimento de software, incluindo arquitetura de aplicações, programação frontend e backend, modelagem de dados, segurança de aplicações e boas práticas de engenharia de software. A integração de conhecimentos teóricos com aplicação prática consolidou competências essenciais para a atuação profissional na área de tecnologia da informação.

O sistema desenvolvido apresenta uma base sólida que pode ser expandida com funcionalidades adicionais, como sistema de notificações, integração com sistemas de pagamento, relatórios analíticos avançados, aplicativo móvel complementar e integração com redes sociais para divulgação e agendamento.

As tecnologias utilizadas demonstraram-se maduras e adequadas para o desenvolvimento de aplicações web modernas. A comunidade ativa e a vasta documentação disponível facilitaram o processo de desenvolvimento e resolução de problemas. A experiência adquirida com estas tecnologias é diretamente aplicável a diversos outros contextos e domínios de aplicação.

Este projeto representa não apenas um produto tecnológico, mas também um marco no processo de formação acadêmica e profissional, demonstrando capacidade de conceber, planejar, desenvolver e documentar soluções de software completas que atendem a necessidades reais de usuários e organizações.

REFERÊNCIAS

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine, 2000.

FULL STACK OPEN. **Deep Dive Into Modern Web Development**. University of Helsinki, 2024. Disponível em: https://fullstackopen.com/en/. Acesso em: 29 nov. 2024.

MOZILLA DEVELOPER NETWORK. **Getting started with React - Learn web development**. Mozilla Foundation, 2024. Disponível em:

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started. Acesso em: 29 nov. 2024.

NODE.JS FOUNDATION. **Introduction to Node.js**. OpenJS Foundation, 2024. Disponível em: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs. Acesso em: 29 nov. 2024.

PRISMA. **Prisma Documentation**. Prisma Data, Inc., 2024. Disponível em: https://www.prisma.io/docs. Acesso em: 29 nov. 2024.

REACT. **React Documentation**. Meta Platforms, Inc., 2024. Disponível em: https://react.dev/. Acesso em: 29 nov. 2024.

SOMMERVILLE, Ian. **Engenharia de Software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

TANENBAUM, Andrew S.; WETHERALL, David J. **Redes de Computadores**. 5. ed. São Paulo: Pearson Prentice Hall, 2011.