MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ADENILSON FERREIRA RAMOS
MÁGNO SILLAS NUNES RAMOS GOMES
NICOLY LANA LOURENÇO CARVALHO
RICAELLE NASCIMENTO TEIXEIRA PONTES

PROJETO DE PROGRAMAÇÃO WEB:
GO BARBERSHOP

GARANHUNS

2026

ADENILSON FERREIRA RAMOS
MÁGNO SILLAS NUNES RAMOS GOMES
NICOLY LANA LOURENÇO CARVALHO
RICAELLE NASCIMENTO TEIXEIRA PONTES

PROJETO DE PROGRAMAÇÃO WEB:
GO BARBERSHOP

Trabalho apresentado ao Professor Igor Medeiros Vanderlei como parte avaliativa da disciplina de Programação Web, do curso de Bacharelado em Ciência da Computação, da Universidade Federal do Agreste de Pernambuco (UFAPE), desenvolvido pelos alunos Adenilson Ferreira Ramos, Mágno Sillas Nunes Ramos Gomes, Nicoly Lana Lourenço Carvalho, Ricaelle Nascimento Teixeira Pontes.

GARANHUNS

2026

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento de sistemas de informação voltados para a gestão de pequenos e médios negócios tem se tornado cada vez mais relevante, especialmente em setores que operam com processos manuais ou pouco informatizados. As barbearias, apesar de sua crescente profissionalização, ainda enfrentam dificuldades relacionadas ao controle de agendamentos, gestão financeira, cálculo de comissões e controle de estoque, o que impacta diretamente a eficiência operacional e a tomada de decisão gerencial.

Neste contexto, o projeto Go BarberShop propõe o desenvolvimento de um sistema de gestão empresarial focado nas necessidades específicas de barbearias, com ênfase na integração entre os módulos operacionais e financeiros. O sistema é implementado na forma de uma API backend, desenvolvida em Java com o framework Spring Boot, adotando uma arquitetura monolítica e seguindo padrões amplamente aceitos no desenvolvimento de software corporativo.

Além de atender a uma necessidade prática de mercado, o Go BarberShop tem como propósito consolidar conhecimentos relacionados à engenharia de software, arquitetura de sistemas, persistência de dados, segurança, testes automatizados e gestão de projetos. Dessa forma, o projeto cumpre simultaneamente um papel técnico, acadêmico e profissional.

2. CONTEXTUALIZAÇÃO E CARACTERIZAÇÃO DO PROJETO

O Go BarberShop caracteriza-se como um sistema de informação corporativo orientado a serviços, cujo núcleo funcional está concentrado em uma API RESTful. Essa API atua como camada intermediária entre aplicações clientes (como sistemas web ou mobile) e o banco de dados, sendo responsável por processar requisições, aplicar regras de negócio e garantir a integridade dos dados.

A opção por uma arquitetura monolítica foi tomada de forma consciente e estratégica. Em projetos de curta duração e com equipes reduzidas, a arquitetura monolítica apresenta vantagens significativas, como menor complexidade de configuração, facilidade de depuração, simplicidade no controle transacional e redução de custos cognitivos durante o desenvolvimento. Todos os módulos do sistema: Agendamento, Usuários, Vendas, Estoque e Financeiro, compartilham o mesmo código-base, o mesmo banco de dados e o mesmo ciclo de vida da aplicação.

O sistema foi concebido com foco prioritário no desenvolvimento da camada backend, responsável pela implementação da lógica de negócio, persistência de dados e exposição da API REST. Embora a interface gráfica do sistema esteja em fase de desenvolvimento em paralelo, o escopo inicial do projeto concentrou-se na construção de uma API, capaz de atender às necessidades funcionais. Essa abordagem permitiu maior aprofundamento na modelagem de dados, na implementação das regras de negócio e na garantia da qualidade da aplicação, estando alinhada aos objetivos acadêmicos da disciplina e possibilitando a integração futura com o frontend em desenvolvimento.

3. OBJETIVOS DO PROJETO

3.1 OBJETIVO GERAL

O objetivo geral do projeto é projetar, desenvolver e documentar uma API backend, capaz de suportar as principais operações administrativas e financeiras de uma barbearia, respeitando os requisitos acadêmicos e o prazo estabelecido.

3.2 OBJETIVOS ESPECÍFICOS

De forma mais detalhada, o projeto busca atingir os seguintes objetivos específicos:

- Desenvolver uma arquitetura backend organizada em camadas, promovendo separação de responsabilidades e manutenibilidade do código;
- Implementar operações CRUD completas para todas as entidades do domínio do sistema;
- Projetar uma modelagem de dados relacional consistente, garantindo integridade referencial e coerência entre os módulos;
- Implementar regras de negócio complexas, com destaque para o cálculo automatizado de comissões de barbeiros;
- Garantir controle transacional nas operações críticas, como vendas e atualização de estoque;
- Implementar autenticação baseada em tokens JWT, garantindo acesso seguro e stateless à API;
- Desenvolver testes unitários e de integração para validação do comportamento do sistema;
- Documentar a API de forma clara, utilizando ferramentas adequadas, facilitando seu consumo por aplicações cliente.

Os critérios de sucesso do projeto estão diretamente relacionados à entrega de uma API funcional, testada, documentada e aderente às especificações definidas no planejamento inicial do grupo.

4. INFORMAÇÕES TÉCNICAS

Esta sessão apresenta as informações técnicas referentes ao projeto, descrevendo as tecnologias, ferramentas e recursos utilizados na implementação do sistema. A identificação desses elementos é fundamental para a compreensão da base tecnológica adotada, como também para possibilitar a reprodução, manutenção e evolução futura da aplicação. As ferramentas e versões adotadas foram definidas de acordo com as configurações presentes no repositório do projeto.

As principais tecnologias e recursos utilizados são listados a seguir:

- Linguagem de Programação: Java 17 (Long Term Support – LTS).

- Framework Backend: Spring Boot versão 2.7.2.
- Módulos do Spring Framework Utilizados:
    - Spring Web, para desenvolvimento da API REST;
    - Spring Data JPA, para persistência e acesso aos dados;
    - Spring Security, para controle de autenticação e segurança;
    - Spring Boot Test, para suporte à realização de testes automatizados.
- Autenticação: JSON Web Token (JWT).
- Banco de Dados: PostgreSQL.
- Mapeamento Objeto-Relacional (ORM): JPA com Hibernate.
- Gerenciamento de Dependências: Maven, por meio do arquivo pom.xml.
- Controle de Versão: Git, com repositório remoto.
- Ambiente de Desenvolvimento (IDE):
    - IntelliJ IDEA;
    - Visual Studio Code.
- Testes Automatizados: Testes unitários e de integração utilizando JUnit e Spring Boot Test.
- Empacotamento da Aplicação: Arquivo JAR executável.
- Containerização: Docker, com uso de Dockerfile e docker-compose.
- Tipo de Aplicação: API REST backend monolítica, com frontend em fase de desenvolvimento.

5. ESTUDO DE VIABILIDADE

5.1 VIABILIDADE TÉCNICA

A viabilidade técnica do projeto foi avaliada considerando-se as tecnologias disponíveis, a experiência da equipe e a complexidade das funcionalidades propostas. A escolha de Java 17 e Spring Boot 2.7.2 garante estabilidade, desempenho e ampla documentação, fatores essenciais para um projeto acadêmico com prazo restrito.

A arquitetura facilita a implementação de transações que envolvem múltiplos módulos, como vendas que impactam estoque e financeiro simultaneamente. A utilização de Spring Data JPA abstrai grande parte da complexidade do acesso a dados, permitindo que o foco do desenvolvimento esteja na lógica de negócio.

A modelagem de dados foi identificada como um dos principais desafios técnicos do projeto. Entidades como vendas, itens de venda, receitas, despesas e comissões exigem relacionamentos bem definidos e regras claras de consistência. Para lidar com essa complexidade, a lógica de negócio foi centralizada na camada de serviços, evitando acoplamento excessivo entre controllers e repositórios.

5.2 VIABILIDADE OPERACIONAL

Sob a perspectiva operacional, o projeto é viável por concentrar-se exclusivamente no backend. A exclusão do frontend reduz significativamente o escopo e permite maior dedicação à implementação correta das regras de negócio.

O tamanho da equipe é compatível com a arquitetura adotada, e a divisão do trabalho em módulos facilita a paralelização das atividades. Além disso, as funcionalidades implementadas correspondem às necessidades essenciais de um ERP, garantindo relevância prática ao sistema desenvolvido.

5.3 VIABILIDADE ECONÔMICA (ACADÊMICA)

Em um contexto acadêmico, a viabilidade econômica está associada ao retorno educacional. O Go BarberShop apresenta alto retorno em termos de aprendizado, pois envolve conceitos avançados de backend, persistência de dados, segurança, testes e arquitetura de software.

O esforço exigido é compensado pela entrega de um projeto completo, funcional e alinhado às práticas do mercado, agregando valor significativo ao portfólio acadêmico e profissional dos envolvidos.

6. ARQUITETURA DO SISTEMA

A arquitetura do Go BarberShop foi planejada e implementada com base em uma arquitetura monolítica em camadas, adotando práticas consolidadas no desenvolvimento de aplicações corporativas com Spring Boot. Essa organização não se limita a uma decisão teórica, mas reflete diretamente a estrutura real do projeto, tanto na disposição dos pacotes quanto na separação funcional das responsabilidades.

O sistema encontra-se organizado em módulos bem definidos, contemplando os seguintes domínios principais: Usuários, Agendamento, Vendas, Estoque e Financeiro. Embora todos os módulos compartilhem o mesmo código-base e banco de dados, cada um possui responsabilidades claramente delimitadas, evitando acoplamento excessivo e facilitando a manutenção do sistema.

6.1 CAMADA DE CONTROLLERS (API REST)

A camada de controllers é responsável por expor os endpoints REST do sistema. Cada controller representa um contexto funcional específico do sistema, como, por exemplo:

- Controller de usuários, responsável pelo cadastro, atualização e consulta de usuários do sistema (clientes e barbeiros);
- Controller de agendamentos, responsável pela criação, consulta e cancelamento de horários de atendimento;
- Controller de vendas, que gerencia o registro de vendas realizadas na barbearia;
- Controller financeiro, que centraliza operações relacionadas a receitas, despesas e comissões.

Esses controllers seguem os princípios REST, utilizando os verbos HTTP adequados para cada operação (POST para criação, GET para consulta, PUT para atualização e DELETE para remoção). As respostas retornam códigos HTTP condizentes com o resultado da operação, como sucesso, erro de validação ou recurso não encontrado.

Um aspecto relevante é que os controllers foram implementados de forma enxuta, atuando apenas como intermediários entre o cliente da API e a camada de serviços. Nenhuma regra de negócio crítica foi implementada diretamente nos controllers, o que contribui para maior clareza arquitetural e facilita a escrita de testes automatizados.

6.2 CAMADA DE SERVICES (LÓGICA DE NEGÓCIO)

A camada de services constitui o núcleo funcional do sistema, concentrando todas as regras de negócio do sistema. É nessa camada que ocorrem as decisões mais importantes relacionadas ao funcionamento do ERP.

Entre as principais funcionalidades implementadas nos services, destacam-se:

- A validação de disponibilidade de horários no momento da criação de um agendamento;
- O processamento completo de uma venda, incluindo o registro da venda, associação dos itens vendidos e atualização do estoque;
- A geração de registros financeiros, como receitas provenientes de vendas e despesas operacionais;
- O cálculo de comissões dos barbeiros, funcionalidade central do projeto, que considera dados provenientes de vendas e agendamentos.

O cálculo de comissões foi implementado de forma explícita e isolada em um serviço específico, responsável por aplicar as regras definidas (por exemplo, percentual de comissão sobre serviços ou produtos vendidos). Essa lógica envolve múltiplas entidades e depende da correta integração entre os módulos de agendamento, vendas e financeiro, o que evidencia a complexidade técnica do projeto.

Além disso, operações críticas utilizam controle transacional por meio da anotação @Transactional, garantindo que processos compostos, como uma venda que afeta estoque e financeiro simultaneamente, sejam executados de forma atômica, evitando inconsistências nos dados.

6.3 CAMADA DE REPOSITORIES (PERSISTÊNCIA)

A camada de persistência foi implementada utilizando Spring Data JPA, permitindo uma abstração eficiente do acesso ao banco de dados PostgreSQL. Cada entidade principal do sistema possui um repositório associado, responsável por realizar operações de leitura e escrita no banco de dados.

Os repositórios são utilizados exclusivamente pela camada de services, garantindo que o acesso aos dados não seja feito diretamente pelos controllers. Essa decisão reforça o princípio de separação de responsabilidades e contribui para a organização do código.

Além das operações CRUD básicas, alguns repositórios possuem consultas específicas, necessárias para atender requisitos de negócio, como a obtenção de vendas por período ou a consulta de dados financeiros para geração de relatórios.

6.4 CAMADA DE MODEL (ENTIDADES DO DOMÍNIO)

As entidades do Go BarberShop foram modeladas para refletir de forma fiel a realidade operacional de uma barbearia. Entre as principais entidades implementadas estão:

- Usuário (representando clientes e barbeiros);
- Agendamento, associado a um usuário, um serviço e um horário específico;
- Produto e estoque, responsáveis por representar os itens comercializados;
- Venda e item de venda, que registram transações comerciais;
- Receita, despesa e comissão, que compõem o módulo financeiro do sistema.

Os relacionamentos entre essas entidades foram definidos para garantir integridade referencial e permitindo rastreabilidade das informações. Por exemplo, uma comissão pode ser associada a uma venda específica, que por sua vez está relacionada a um agendamento e a um barbeiro.

Essa modelagem consistente foi fundamental para viabilizar a lógica financeira do sistema e para permitir futuras expansões do projeto.

6.5 DTOs E TRATAMENTO DE EXCEÇÕES

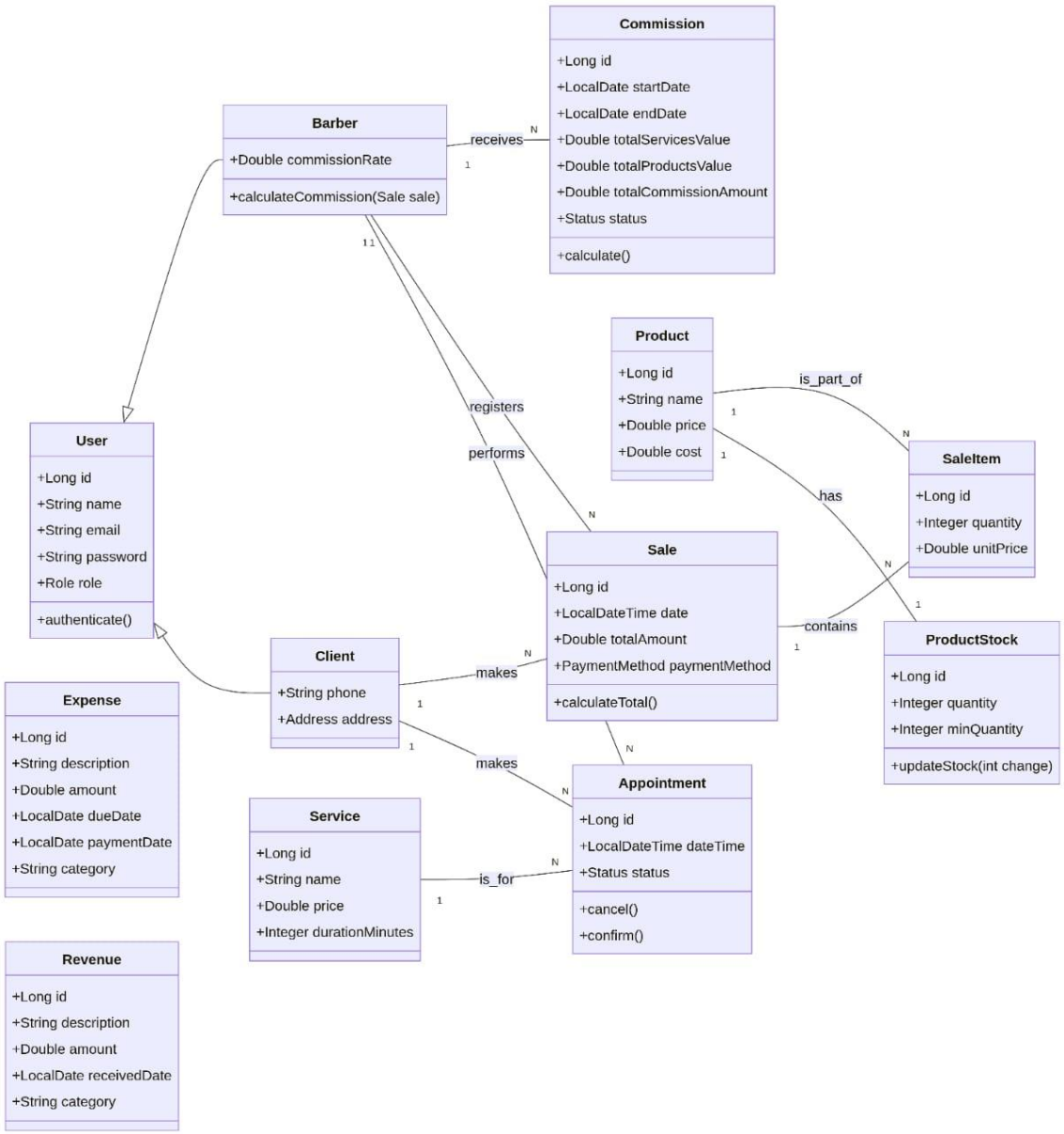
O uso de DTOs (Data Transfer Objects) permitiu controlar quais informações são expostas pela API, evitando o envio direto das entidades do banco de dados. Essa abordagem aumenta a segurança, reduz o acoplamento e facilita alterações futuras na estrutura interna do sistema.

O projeto também implementa um tratamento centralizado de exceções, garantindo respostas padronizadas para erros comuns, como dados inválidos, recursos inexistentes ou falhas internas. Isso melhora significativamente a experiência de quem consome a API e reforça o caráter profissional da implementação.

7. DIAGRAMA DE CLASSES

O diagrama apresentado representa a estrutura básica do sistema Go BarberShop, evidenciando as principais classes, seus atributos e os relacionamentos existentes entre elas. Esse diagrama tem como finalidade facilitar a compreensão da organização do sistema e da forma como os diferentes módulos se conectam para atender às funcionalidades propostas.

Por meio do diagrama, é possível visualizar as entidades centrais do projeto, como usuários, clientes, barbeiros, agendamentos, vendas, produtos, estoque e registros financeiros. As relações entre essas classes demonstram o fluxo de informações dentro do sistema, especialmente no que se refere à integração entre os módulos de agendamento, vendas e financeiro.



8. TESTES E GARANTIA DA QUALIDADE

A qualidade do Go BarberShop foi tratada como um aspecto essencial do projeto, principalmente em função da complexidade das regras de negócio implementadas. Para isso,

foram desenvolvidos testes unitários e testes de integração, garantindo maior confiabilidade ao sistema.

8.1 TESTES UNITÁRIOS

Os testes unitários foram aplicados principalmente na camada de services, onde se concentram as regras mais críticas. Esses testes validam cenários específicos, como:

- Cálculo correto de comissões para diferentes valores de venda;
- Validação de regras de criação de agendamentos;
- Processamento adequado de vendas e atualização de estoque.

Ao testar essas funcionalidades de forma isolada, foi possível garantir que cada regra de negócio se comportasse conforme esperado, independentemente da camada de apresentação ou persistência.

8.2 TESTES DE INTEGRAÇÃO

Os testes de integração validam o comportamento do sistema de ponta a ponta, simulando requisições reais aos endpoints REST. Esses testes garantem que controllers, services, repositórios e banco de dados funcionem corretamente em conjunto.

No contexto do sistema, os testes de integração foram fundamentais para validar fluxos completos, como o registro de uma venda ou a geração de dados financeiros, assegurando que o sistema estivesse pronto para uso em um cenário real.

9. CONSIDERAÇÕES FINAIS

O desenvolvimento do Go BarberShop resultou em uma aplicação backend consistente, funcional e tecnicamente bem estruturada. O projeto conseguiu integrar múltiplos módulos: agendamento, vendas, estoque e financeiro, em um único sistema coeso, demonstrando domínio prático de conceitos avançados de engenharia de software.

A implementação da lógica de comissões, aliada ao controle financeiro e ao uso de transações, representa um dos pontos mais relevantes do projeto, evidenciando a capacidade de lidar com regras de negócio complexas e interdependentes. A arquitetura em camadas, aliada ao uso de tecnologias consolidadas, contribuiu para a clareza do código e para a facilidade de manutenção.

Do ponto de vista acadêmico, o projeto cumpriu integralmente seus objetivos, oferecendo uma experiência prática abrangente no desenvolvimento de APIs RESTful, persistência de dados, segurança e testes automatizados. Além disso, o Go BarberShop apresenta potencial de evolução, podendo futuramente incorporar uma interface gráfica, relatórios mais avançados ou integrações externas.