INSTITUTO FEDERAL
Rondônia

MINISTÉRIO DA EDUCAÇÃO
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia

Campus Vilhena

Coordenação do Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

LUCAS FERNANDES DA SILVA

Sistema de gerenciamento de barbearia - *BeardhairApp*

VILHENA - RO

2025

LUCAS FERNANDES DA SILVA

Sistema de gerenciamento de barbearia - *BeardhairApp*

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Mestre Marco Antônio Augusto de Andrade.

VILHENA - RO

2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Silva, Lucas Fernandes da.
Sistema de gerenciamento de barbearia: BeardHairApp / Lucas Fernandes da Silva. - Vilhena, 2025.
24 f. : il.

Orientador(a): Prof. Me. Marco Antonio Augusto de Andrade.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena, 2025.

1. Sistema de gestão. 2. Barbearia. 3. Kotlin. 4. Node.js. 5. API REST. I. Andrade, Marco Antonio Augusto de (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

LUCAS FERNANDES DA SILVA

Sistema de gerenciamento de barbearia - *BeardhairApp*

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Mestre Marco Antônio Augusto de Andrade.

Aprovado em 18/11/2025 pela banca examinadora.

gov.br Documento assinado digitalmente
ROBERTO SIMPLICIO GUIMARAES
Data: 03/12/2025 00:23:18-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Roberto Simplicio Guimaraes

Examinador interno

gov.br Documento assinado digitalmente
GILBERTO PEREIRA DA SILVA
Data: 03/12/2025 12:42:07-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Gilberto Pereira da Silva

Examinador interno

gov.br Documento assinado digitalmente
MARCO ANTONIO AUGUSTO DE ANDRADE
Data: 01/12/2025 20:27:23-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Marco Antonio Augusto de Andrade

Orientador

Resumo

Este artigo apresenta o desenvolvimento de um sistema voltado à digitalização do agendamento e da gestão financeira da barbearia The Beard Hair, localizada em Vilhena (RO). O projeto surgiu da necessidade de eliminar falhas operacionais causadas pelo uso de uma agenda física. O sistema foi desenvolvido em Kotlin, seguindo a arquitetura MVVM e integrando-se a uma API desenvolvida em Node.js. Entre as principais funcionalidades estão o gerenciamento de usuários, clientes, serviços e agendamentos, relatórios financeiros e indicadores de desempenho. A solução proposta visa otimizar o atendimento, aumentar a confiabilidade das informações e contribuir para a modernização da gestão de pequenos empreendimentos do setor de beleza.

Palavras-chave: Sistema de gestão; Barbearia; Kotlin; Node.js; API REST.

Abstract

This article presents the development of a system aimed at digitizing the scheduling and financial management of The Beard Hair barbershop, located in Vilhena (RO), Brazil. The project arose from the need to eliminate operational failures caused by the use of a physical appointment book. The system was developed in Kotlin, following the MVVM architecture and integrating with an API developed in Node.js. Key functionalities include user, client, service, and appointment management, financial reports, and performance indicators. The proposed solution aims to optimize customer service, increase the reliability of information, and contribute to the modernization of the management of small businesses in the beauty sector.

Keywords: Management system; Barber shop; Kotlin; Node.js; REST API.

1. Introdução

As barbearias de pequeno porte têm apresentado crescimento nos últimos anos. Estima-se que cerca de 18.278 novos pequenos negócios voltados às atividades econômicas de cabeleireiros e beleza foram abertos no primeiro trimestre de 2025 (BRASIL 2025). Esse segmento, composto em grande parte por microempreendedores individuais (MEIs), enfrenta desafios característicos de gestão, especialmente no controle de agendamentos e no acompanhamento financeiro. Em cidades de médio porte, como Vilhena - RO, observa-se que muitos estabelecimentos ainda utilizam métodos manuais, como agendas físicas ou planilhas eletrônicas, para organizar os atendimentos e registrar informações financeiras. Essa prática, embora comum entre pequenos negócios, aumenta a probabilidade de falhas operacionais e dificulta a análise sistematizada do desempenho do empreendimento.

A barbearia *The Beard Hair*, localizada em Vilhena, reflete bem essa realidade. O proprietário utiliza uma agenda física para registrar os atendimentos, o que frequentemente ocasiona confusões, sobreposição de horários e falhas no controle financeiro. Essa falta de integração entre os meios de gestão impede a geração de relatórios precisos de faturamento, dificultando a análise de desempenho e a tomada de decisões estratégicas sobre o negócio.

A ausência de um sistema unificado de agendamentos e controle financeiro compromete não apenas a organização do trabalho, mas também a experiência do cliente, uma vez que aumenta as chances de esquecimentos ou duplicidade de marcações. Além disso, o acompanhamento manual das finanças torna o processo mais suscetível a erros, impossibilitando a visualização de indicadores essenciais, como faturamento mensal, serviços mais solicitados e clientes mais recorrentes.

Diante desse cenário, este trabalho tem como objetivo desenvolver um sistema completo de gestão para a barbearia *The Beard Hair*, integrando em uma única plataforma o agendamento de horários, o controle de clientes e serviços, e a geração de relatórios financeiros automatizados. O sistema visa otimizar o tempo do barbeiro, reduzir erros de marcação e oferecer maior confiabilidade nas informações, tornando o processo de atendimento mais ágil e profissional.

De forma complementar, foram definidos os seguintes objetivos específicos:

- Analisar soluções existentes voltadas à gestão de barbearias, identificando suas principais funcionalidades e limitações.
- Compreender o funcionamento operacional da barbearia *The Beard Hair*.
- Realizar o levantamento dos requisitos da aplicação proposta.
- Modelar o banco de dados de acordo com os requisitos levantados.
- Implementar e testar a aplicação, aplicando testes automatizados para garantir a integridade das funcionalidades.
- Efetuar o *deploy* em ambiente de produção, assegurando a disponibilidade e o funcionamento contínuo do *software*.

A justificativa deste trabalho baseia-se tanto na relevância prática quanto na acadêmica. Do ponto de vista prático, o sistema atende a uma necessidade real de digitalização e automação de processos em pequenos negócios locais, possibilitando

maior controle e eficiência na gestão da barbearia. Do ponto de vista acadêmico, o projeto representa a aplicação concreta de conceitos de Engenharia de *Software*, arquitetura em camadas, DevOps e desenvolvimento *mobile*, consolidando o aprendizado obtido ao longo do curso.

O artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica sobre os conceitos e tecnologias utilizadas; a Seção 3 descreve a metodologia de desenvolvimento adotada; a Seção 4 detalha o processo de construção do sistema; a Seção 5 apresenta os resultados obtidos e discussões; e, por fim, a Seção 6 traz as conclusões e sugestões para trabalhos futuros.

2. Fundamentação Teórica

A gestão eficiente de microempresas é um fator essencial para garantir a sustentabilidade e o crescimento desses empreendimentos. No contexto das barbearias, atividades como o controle de agendamentos e a administração financeira demandam organização e agilidade, uma vez que envolvem o relacionamento direto com os clientes e o acompanhamento das receitas diárias. Segundo Polato (2023), muitos microempreendedores ainda recorrem a métodos manuais de gestão, o que aumenta a probabilidade de erros operacionais e reduz a capacidade de análise gerencial. Na barbearia *The Beard Hair*, localizada em Vilhena, o proprietário realiza seus agendamentos por meio de uma agenda física, complementada pelo uso do WhatsApp para comunicação com os clientes. As marcações e cancelamentos são registrados manualmente, o que frequentemente gera conflitos de horários, esquecimentos e duplicidades de agendamento. Somado a isso, o controle financeiro é feito de forma manual, exigindo o somatório individual dos atendimentos para obtenção dos indicadores de faturamento, o que se torna trabalhoso e propenso a falhas.

Essas dificuldades refletem um cenário comum entre microempreendedores individuais (MEIs), que enfrentam limitações em relação ao uso de ferramentas tecnológicas de gestão. Conforme Queiroz Junior e Terencio (2024), a adoção de *softwares* modernos é uma estratégia essencial para otimizar os processos administrativos e melhorar o desempenho operacional de pequenos negócios. Nesse sentido, a automatização de processos, especialmente em áreas como o agendamento e o controle financeiro, surge como uma solução que oferece agilidade, centralização de informações e redução de erros. Outro ponto relevante é que o uso de sistemas informatizados contribui para a tomada de decisão mais assertiva, uma vez que os dados ficam organizados e facilmente acessíveis.

No mercado de aplicativos para barbearias e salões de beleza, existem diversas soluções que oferecem funcionalidades como agendamento de clientes, cadastro de serviços e controle financeiro, exemplos que incluem Trinks[1] e BarberApp[2]. Embora muitos desses aplicativos possuam funcionalidades similares às implementadas no presente sistema, o desenvolvimento da solução para a barbearia *The Beard Hair* possui um foco acadêmico e prático, permitindo a aplicação de conceitos de arquitetura de *software*, metodologias ágeis e integração de tecnologias modernas, como Node.js,

[1] Disponível em: https://www.trinks.com/. Acesso em: 6 out. 2025.
[2] Disponível em: https://www.barberapp.co.uk/. Acesso em: 6 out. 2025.

MongoDB e Kotlin com Jetpack Compose. O sistema foi projetado considerando as particularidades operacionais de microempreendedores individuais, o que torna a implementação mais próxima da realidade de pequenos estabelecimentos.

Observando o cenário brasileiro, houve um crescimento significativo no número de microempreendedores individuais (MEIs). No primeiro trimestre de 2025, foram abertos 1,4 milhão de pequenos negócios, dos quais os MEIs corresponderam a aproximadamente 78% do total, evidenciando o aumento de pequenos empreendimentos no país e a crescente necessidade de organização e eficiência na gestão dessas empresas (BRASIL, 2025). Esse crescimento reforça a importância de iniciativas que promovam o uso da tecnologia como meio de fortalecimento da gestão e da competitividade no mercado.

3. Metodologia / Materiais e métodos

No desenvolvimento do sistema, foi adotada a metodologia *Kanban*, por sua praticidade e adequação a projetos individuais. Segundo Jurowitz e Silva (2024), o *Kanban* tem ganhado popularidade por sua abordagem visual e flexível, oferecendo uma alternativa eficiente às metodologias tradicionais de gerenciamento de projetos e permitindo uma adaptação mais ágil às mudanças de requisitos e condições do desenvolvimento. No GitLab, as atividades foram organizadas em três colunas principais: “A Fazer”, “Em andamento” e “Concluído”, o que permitiu visualizar o progresso das tarefas e gerenciar as prioridades de forma prática e intuitiva. Essa escolha mostrou-se adequada devido à natureza individual do projeto, dispensando as cerimônias e papeis formais de metodologias como o *Scrum*. O uso do *Kanban* contribuiu para uma gestão leve, visual e contínua das atividades, promovendo maior autonomia e organização pessoal durante o desenvolvimento do sistema.

A escolha das tecnologias utilizadas no desenvolvimento foi orientada pela busca por simplicidade, desempenho e integração eficiente entre os componentes. A arquitetura adotada segue o modelo cliente-servidor, em que o aplicativo *mobile*, desenvolvido em Kotlin com Jetpack Compose, consome os dados da *Application Programming Interface* (API) *RESTful* implementada em Node.js, por meio de requisições HTTP (*Hypertext Transfer Protocol*).

O Node.js foi adotado na camada de *back-end* por sua arquitetura orientada a eventos e pela capacidade de lidar com múltiplas requisições de forma eficiente e escalável, conforme destacado por Stein Junior (2019). A API foi organizada em camadas internas, segundo Melo e Silva (2024), essa separação favorece a implementação e futuras manutenções. De acordo com Teixeira Filho (2025), Cada camada possui uma responsabilidade específica: os *controllers* recebem as requisições do cliente, neste caso do aplicativo, e repassam os dados para processamento; os *services* contêm as regras de negócio e realizam validações; e os *repositories* são responsáveis pela comunicação com o banco de dados, executando operações de armazenamento, consulta, atualização e exclusão de dados.

Para o armazenamento de dados, optou-se pelo MongoDB, um banco de dados NoSQL orientado a documentos, amplamente reconhecido por sua flexibilidade e desempenho. De acordo com Souza e Oliveira (2019), as vantagens do MongoDB se

dão na facilidade de consulta e escalabilidade, devido à sua estrutura baseada em objetos orientados a documentos. Essa abordagem permite que os dados sejam armazenados de forma mais natural, reduzindo a complexidade de modelagem e facilitando futuras alterações na estrutura da aplicação.

A API está hospedada no *cluster* Kubernetes do Laboratório de Fábricas de *Software* (FSlab) do Instituto Federal de Rondônia (IFRO) – *Campus* Vilhena, que serve como ambiente institucional para experimentação e implantação de sistemas desenvolvidos em projetos acadêmicos. Segundo SILVA (2023), o Kubernetes oferece recursos avançados para garantir alta disponibilidade, escalabilidade e resiliência de aplicações. Sua utilização permite orquestrar os contêineres da aplicação, facilitando a implantação de atualizações e o balanceamento de carga, assegurando que o sistema permaneça estável e acessível mesmo em cenários de alta demanda. Esses recursos tornam o Kubernetes uma ferramenta eficiente para gerenciar aplicações e automatizar operações, possibilitando maior eficiência operacional.

O aplicativo foi desenvolvido utilizando o Android Studio, ambiente integrado de desenvolvimento (IDE) oficial para o sistema operacional Android. Por meio do Jetpack Compose, foi possível adotar um padrão de desenvolvimento moderno e declarativo, que facilita a criação de interfaces dinâmicas e reativas. Segundo Carvalho (2025), o Compose representa uma evolução significativa em relação aos métodos tradicionais de construção de interfaces no Android, pois resulta em um código intuitivo e de fácil manutenção.

O projeto foi estruturado segundo o padrão MVVM (*Model-View-ViewModel*), realizando a separação entre lógica do negócio, interface do usuário e controle de estado. Conforme explica Cardoso (2025), Nesse modelo, a *view* corresponde à interface onde o usuário realiza as interações; o *model* é responsável pelo gerenciamento dos dados; e o *viewmodel* atua processando as informações enviadas pela *view* e realizando a integração com a API.

Durante a construção do sistema, foram realizadas etapas de prototipagem no Figma, que serviram para definir o escopo visual e os requisitos funcionais antes da implementação. Essa etapa inicial auxiliou na identificação dos componentes de interface, fluxos de navegação e elementos de interação necessários para garantir uma boa experiência do usuário (*UX*).

Por fim, o sistema foi projetado para ser adaptável, permitindo sua aplicação não apenas na barbearia *The Beard Hair*, mas também em outros empreendimentos de pequeno porte que enfrentam desafios semelhantes de gestão. Dessa forma, o desenvolvimento contribui não apenas como um estudo de caso técnico, mas também como uma iniciativa prática de transformação digital voltada a pequenos negócios locais.

4. Desenvolvimento / Implementação

O desenvolvimento do sistema seguiu uma abordagem incremental e planejada, iniciando pela modelagem visual das interfaces no Figma, que serviu como base para a definição dos requisitos e atributos necessários. A partir desse planejamento, as etapas

seguintes envolvem a implementação da base de dados, construção da *API*, implantação no ambiente de produção e, por fim, o desenvolvimento do aplicativo *mobile* em Kotlin.

O sistema foi construído com base no modelo cliente-servidor, composto por três camadas principais:

1. *Front-end*: aplicativo Android desenvolvido em Kotlin com Jetpack Compose.
2. *Back-end*: API RESTful desenvolvida em Node.js utilizando o *framework* Express.js.
3. Banco de dados: MongoDB, hospedado no MongoDB Atlas, serviço de banco de dados gerenciado na nuvem.

4.1. Padrões e boas práticas adotadas

Durante o desenvolvimento, foram aplicadas boas práticas que contribuíram para a organização e qualidade do código, incluindo o GitFlow, um modelo de versionamento que define uma estrutura padronizada de *branches* para o controle do ciclo de vida do software. Esse modelo propõe uma divisão entre os *branches master*, responsável pela versão estável em produção; *develop*, destinado ao ambiente de desenvolvimento e *feature*, utilizado para o desenvolvimento de novas funcionalidades de forma isolada. Conforme explicam Marques e Silva (2020), o GitFlow favorece a organização do desenvolvimento em equipe, a rastreabilidade das alterações e a manutenção da infraestrutura do projeto sem interferir na versão em produção.

Optou-se por manter a nomenclatura tradicional (*master*, *develop* e *feature*) porque, ao criar o repositório no GitLab, a *branch* principal foi gerada automaticamente como *master*, não havendo necessidade de renomeação. Portanto, manteve-se a estrutura original do projeto e o fluxo clássico do GitFlow.

Outras boas práticas foram adotadas ao longo da implementação, como a documentação da API com o Swagger, que permitiu descrever de forma clara os *endpoints* e parâmetros das rotas, facilitando seu consumo; a implementação de JWT (*JSON Web Token*) para autenticação de usuários, assegurando o acesso seguro às rotas protegidas; e o uso de variáveis de ambiente em arquivos *.env*, garantindo a proteção de informações sensíveis, como chaves e URLs (*Uniform Resource Locator*) do banco de dados.

4.2. Modelagem no Figma

A primeira etapa consistiu na criação do protótipo de interface no Figma, ferramenta utilizada para planejar o fluxo de navegação e o design das telas do aplicativo, permitindo visualizar como o usuário interagiria com o sistema e definindo os elementos essenciais, como campos de entrada, botões de ação e informações exibidas em cada tela. As principais telas projetadas incluíram a tela de *login* e recuperação de senha, usuários, agendamentos, serviços e relatórios financeiros, conforme ilustrado na Figura 1. Com base nesse planejamento, foram identificados os atributos necessários para cada entidade, orientando a modelagem do banco de dados, criação das rotas da *API* e desenvolvimento do aplicativo.

4.3. Estruturação do banco de dados

Após o levantamento de requisitos no Figma, iniciou-se a modelagem do banco de dados MongoDB, refletindo diretamente as necessidades identificadas nas telas do aplicativo. A Figura 2 apresenta a estrutura das coleções utilizadas no banco de dados MongoDB, destacando seus principais atributos e a forma como os dados são organizados. Essa representação demonstra a estrutura flexível do modelo não relacional adotado, permitindo o armazenamento dinâmico das informações.

4.4. Desenvolvimento da API

Com o banco estruturado, iniciou-se a implementação da API em Node.js, utilizando o *framework* Express.js e organizada em camadas internas, que proporciona organização, clareza e facilidade de manutenção. A estrutura geral da API pode ser visualizada na Figura 3, que ilustram respectivamente as camadas *controller, service e repository*. Na camada *controller*, as requisições do cliente são recebidas e direcionadas para a camada *service*; na camada *service*, é aplicada a lógica de negócio, incluindo validação de dados com a biblioteca Zod, garantindo que apenas informações consistentes e corretas sejam processadas; e na camada *repository*, ocorre a comunicação direta com o banco de dados. A API foi validada por meio de uma suíte de testes automatizados desenvolvida em Jest, abrangendo testes unitários e de integração responsáveis por verificar regras de negócio, validações, respostas das rotas e fluxos internos entre *controller, service* e *repository*. Como resultado, os testes alcançaram elevada cobertura, conforme o relatório gerado automaticamente pelo Jest Figura 5. A API foi documentada utilizando o Swagger, conforme exemplo apresentado na Figura 6 permitindo a geração de uma interface visual com todos os *endpoints*, parâmetros e exemplos de requisições, facilitando o consumo e entendimento das rotas implementadas.

4.5. *Deploy* e Integração Contínua (CI/CD – *Continuous Integration / Continuous Deployment*)

Após a conclusão dos testes e validações, a API foi implantada no *cluster* Kubernetes do FSlab do IFRO – *Campus* Vilhena, garantindo padronização do processo de *deploy*, isolamento entre serviços e facilidade de escala futura. A configuração atual utiliza uma única réplica do contêiner, suficiente para o escopo acadêmico do projeto, mas não caracteriza alta disponibilidade ou escalabilidade horizontal. Recursos como Horizontal Pod Autoscaler (HPA), múltiplas réplicas, Ingress Controller e políticas avançadas de rolling update não foram empregados nesta versão, mantendo a arquitetura simples e compatível com as necessidades do projeto.

Para assegurar a qualidade e a confiabilidade do desenvolvimento, foi configurado um *pipeline* de CI/CD no GitLab, responsável por automatizar as etapas de *build*, *test* e *deploy*. O *pipeline* executa testes unitários e de integração com Jest a cada *merge request*, bloqueando automaticamente a fusão de código em caso de falha nos testes e, após aprovação, realiza o *merge* nas *branches develop* ou *master*, gerando *build* e *deploy* automáticos no Kubernetes.

Essa automação reduziu o tempo de entrega de novas versões e eliminou erros manuais durante o processo de publicação, aumentando a confiabilidade do sistema em produção. Além disso, o uso combinado do GitLab CI/CD e do Kubernetes permite que novas versões do sistema sejam publicadas de forma automatizada, garantindo um processo contínuo de integração e entrega. Essa abordagem possibilita uma resposta mais ágil a mudanças, facilitando a implementação de melhorias e correções, mantendo o processo de atualização consistente dentro do escopo atual da arquitetura. Também reduz a dependência de intervenções manuais, o que contribui para um ciclo de desenvolvimento mais estável e previsível.

Embora o *software* seja utilizado atualmente apenas pelo barbeiro e não ofereça interação direta com clientes, sua arquitetura permite a evolução para novas funcionalidades no futuro, sem exigir alterações estruturais no *back-end.* A abordagem adotada proporciona uma base organizada e modular, facilitando a incorporação de novos recursos conforme a necessidade. O sistema apresenta uma estrutura estável e compatível com práticas modernas de desenvolvimento, mantendo boa manutenibilidade e permitindo expansão gradual dentro do escopo proposto.

4.6. Desenvolvimento do Aplicativo

Com o *back-end* em funcionamento, iniciou-se o desenvolvimento do aplicativo Android em Kotlin, utilizando o Jetpack Compose para a criação das interfaces de forma moderna e declarativa. Essa abordagem proporcionou maior agilidade na construção das telas e reduziu a quantidade de código necessária, permitindo uma experiência de desenvolvimento mais eficiente e intuitiva. A integração com a API foi realizada por meio da biblioteca Retrofit, permitindo requisições HTTP de maneira simples, padronizada e eficiente. A estrutura do aplicativo segue o padrão MVVM, conforme ilustrado na Figura 4, garantindo a separação clara entre a lógica de negócio e a interface gráfica, facilitando a manutenção. Cada *viewmodel* encapsula a lógica de negócio correspondente a sua tela, realizando requisições à API, tratando respostas e erros, e mantendo o estado atualizado da interface. A integração com a API inclui tratamento de erros, verificação de respostas e manipulação de exceções para garantir que o aplicativo permaneça estável mesmo em caso de falhas de rede ou dados inconsistentes.

Figura 1. Protótipo desenvolvido no Figma

Fonte: Elaborada pelo autor.

A figura ilustra o protótipo das principais telas do aplicativo, incluindo *login*, listagem de agendamentos e relatórios financeiros.

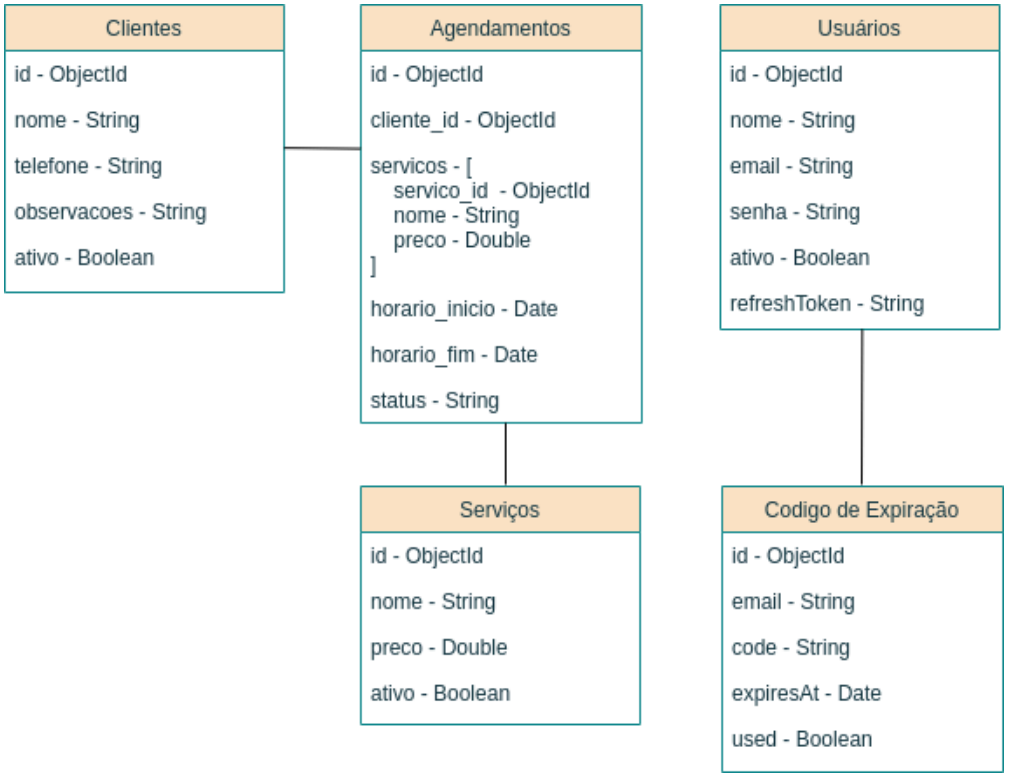


Figura 2. Estrutura das coleções do banco de dados MongoDB

Fonte: Elaborada pelo autor.

A figura apresenta as coleções utilizadas pela aplicação, incluindo seus principais atributos e as relações lógicas entre elas. Apesar do MongoDB ser um banco de dados não relacional, essas relações são representadas por meio de referências de identificadores (*ObjectId*) ou pela incorporação de documentos. A coleção Agendamentos contém atributos como cliente_id, que faz referência à coleção Clientes, e servicos que consiste em uma lista incorporada com informações sobre os serviços realizados. A coleção Usuários possui relação lógica com a coleção Código de Expiração, responsável pelos códigos temporários de redefinição de senha.

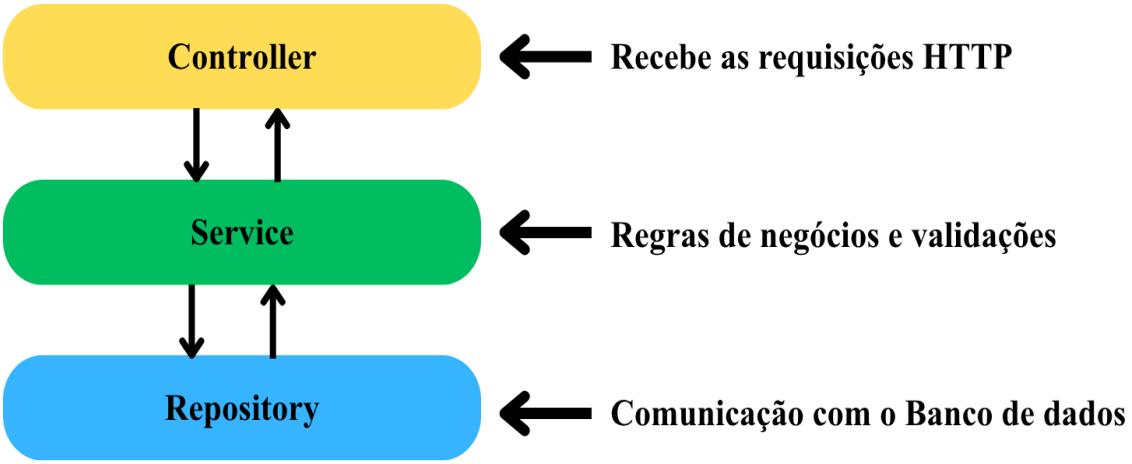


Figura 3. Camadas da API

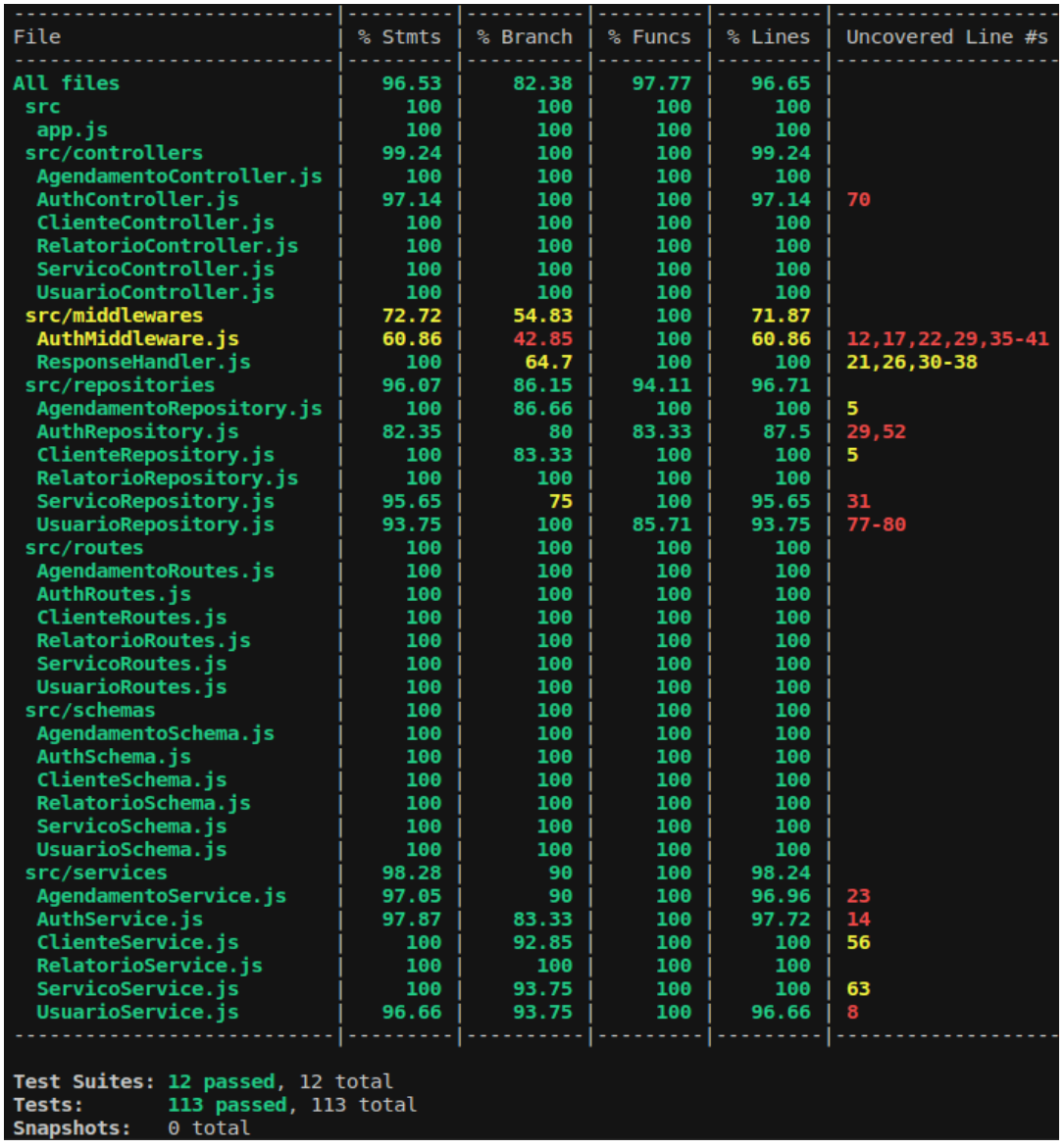
Fonte: Elaborada pelo autor.

A figura ilustra a organização interna da API, estruturada segundo o modelo em camadas. Nesse fluxo, o cliente (aplicativo) realiza uma requisição HTTP para a API. Essa requisição é recebida pelo *controller*, responsável por interpretá-la e encaminhá-la ao *service*, onde são executadas as regras de negócio e validações necessárias. Em seguida, o *service* aciona o *repository*, camada responsável pela comunicação com o banco de dados. Após o processamento, o resultado da operação é retornado ao cliente.

Figura 4. Padrão de arquitetura MVVM aplicado ao aplicativo Android

Fonte: Elaborada pelo autor.

A figura representa a estrutura do modelo MVVM, utilizada na organização interna do aplicativo. A camada *view* é responsável pela interface com o usuário, exibindo informações e capturando as interações. Esses dados são repassados ao *viewmodel*, que realiza o controle de estado da interface e a comunicação com a *API*. Em seguida, o *model* define as estruturas de dados utilizadas tanto nas requisições quanto nas respostas da *API*, garantindo consistência e segurança de tipos, uma vez que o Kotlin é uma linguagem fortemente tipada.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	96.53	82.38	97.77	96.65	
src	100	100	100	100	
app.js	100	100	100	100	
src/controllers	99.24	100	100	99.24	
AgendamentoController.js	100	100	100	100	
AuthController.js	97.14	100	100	97.14	70
ClienteController.js	100	100	100	100	
RelatorioController.js	100	100	100	100	
ServicoController.js	100	100	100	100	
UsuarioController.js	100	100	100	100	
src/middlewares	72.72	54.83	100	71.87	
AuthMiddleware.js	60.86	42.85	100	60.86	12,17,22,29,35-41
ResponseHandler.js	100	64.7	100	100	21,26,30-38
src/repositories	96.07	86.15	94.11	96.71	
AgendamentoRepository.js	100	86.66	100	100	5
AuthRepository.js	82.35	80	83.33	87.5	29,52
ClienteRepository.js	100	83.33	100	100	5
RelatorioRepository.js	100	100	100	100	
ServicoRepository.js	95.65	75	100	95.65	31
UsuarioRepository.js	93.75	100	85.71	93.75	77-80
src/routes	100	100	100	100	
AgendamentoRoutes.js	100	100	100	100	
AuthRoutes.js	100	100	100	100	
ClienteRoutes.js	100	100	100	100	
RelatorioRoutes.js	100	100	100	100	
ServicoRoutes.js	100	100	100	100	
UsuarioRoutes.js	100	100	100	100	
src/schemas	100	100	100	100	
AgendamentoSchema.js	100	100	100	100	
AuthSchema.js	100	100	100	100	
ClienteSchema.js	100	100	100	100	
RelatorioSchema.js	100	100	100	100	
ServicoSchema.js	100	100	100	100	
UsuarioSchema.js	100	100	100	100	
src/services	98.28	90	100	98.24	
AgendamentoService.js	97.05	90	100	96.96	23
AuthService.js	97.87	83.33	100	97.72	14
ClienteService.js	100	92.85	100	100	56
RelatorioService.js	100	100	100	100	
ServicoService.js	100	93.75	100	100	63
UsuarioService.js	96.66	93.75	100	96.66	8

Test Suites: 12 passed, 12 total
Tests: 113 passed, 113 total
Snapshots: 0 total

Figura 5. Cobertura de testes da API

Fonte: Elaborada pelo autor.

A Figura 5 apresenta o relatório gerado durante a execução da suíte de testes. Esses resultados comprovam a efetividade da automação implementada, garantindo estabilidade do código e reduzindo a chance de falhas regressivas no ambiente de produção.

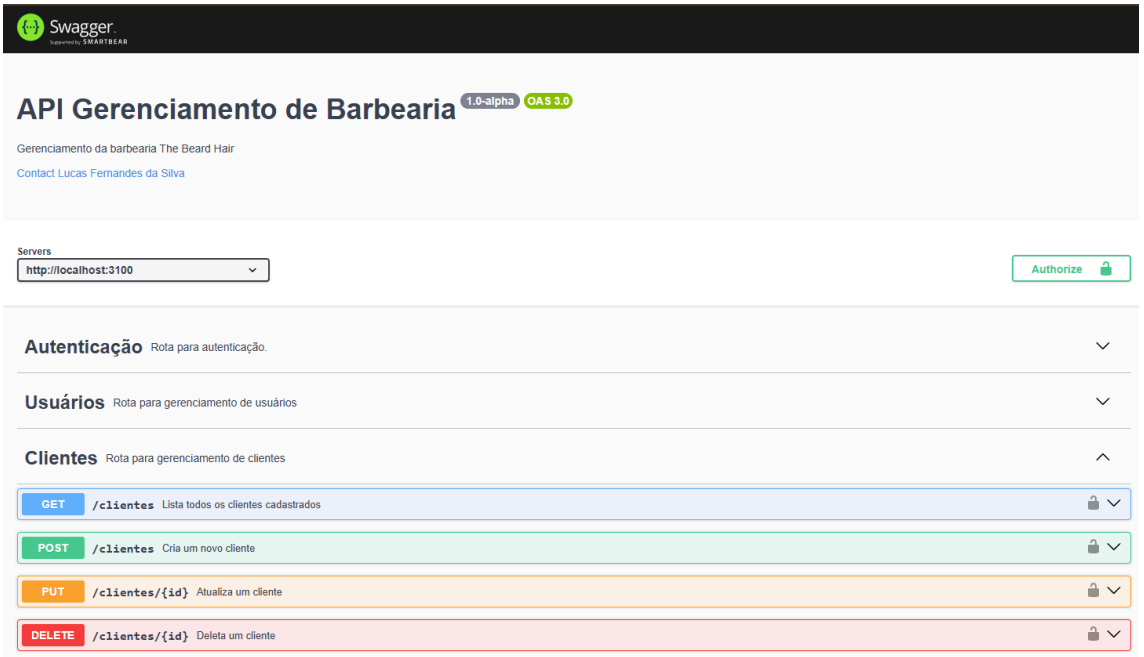


Figura 6. Documentação da API utilizando Swagger

Fonte: Elaborada pelo autor.

A figura demonstra a interface gerada pelo Swagger, que apresenta os *endpoints* da API, métodos disponíveis, parâmetros e exemplos de requisições, facilitando o entendimento e consumo.

5. Resultados e Discussões

5.1. Regras de Negócio

As regras de negócio do sistema foram definidas de forma a garantir a integridade dos dados e a consistência das operações. O sistema é de uso exclusivo do barbeiro ou administrador, não havendo acesso para clientes externos. Assim, todas as ações, como cadastros, edições e consultas, são realizadas por um único perfil com permissão total de acesso. Entre as principais regras de negócio estabelecidas, destacam-se:

- Política de inativação: registros de usuários, clientes, serviços e agendamentos não são excluídos do banco de dados, mas marcados como inativos, garantindo o histórico de informações.
- Controle de disponibilidade: o sistema valida automaticamente conflitos de horários, impedindo o agendamento de dois atendimentos no mesmo período.
- Integridade de dados: o cadastro de clientes e serviços é validado para evitar duplicidade de registros com nome e telefone ou nome e valor idênticos.
- Regra de faturamento: apenas os agendamentos com *status* “atendido” são computados no relatório financeiro, impedindo que valores referentes a serviços ainda não executados sejam considerados no cálculo do faturamento. Essa validação assegura maior precisão nos relatórios e reflete fielmente o desempenho financeiro real da barbearia.

5.2. Funcionalidades Implementadas

O sistema desenvolvido apresenta um conjunto de funcionalidades voltadas ao gerenciamento completo de uma barbearia, atendendo tanto às necessidades administrativas quanto operacionais. Dentre as principais funções, destacam-se:

- RF01 – Autenticação e recuperação de senha: garante acesso seguro e restrito ao sistema.
- RF02 – Gerenciamento de usuários: permite o cadastro, listagem, edição, inativação lógica de registros.
- RF03 – Gerenciamento de clientes: possibilita registrar dados pessoais, realizar buscas por nome ou telefone, editar informações e inativar clientes sem perda de histórico.
- RF04 – Gerenciamento de serviços: viabiliza o controle de serviços oferecidos, seus valores e disponibilidade, impedindo duplicidades e o uso de serviços inativos em novos agendamentos.
- RF05 – Controle de agendamentos: permite cadastrar, visualizar, editar ou desmarcar atendimentos; aplicar validações automáticas de conflito de horários.
- RF06 – Relatórios e indicadores: gera relatórios financeiros e operacionais com métricas como faturamento total, número de clientes atendidos, serviço mais executado e ticket médio no período, considerando apenas os agendamentos com *status* atendido.

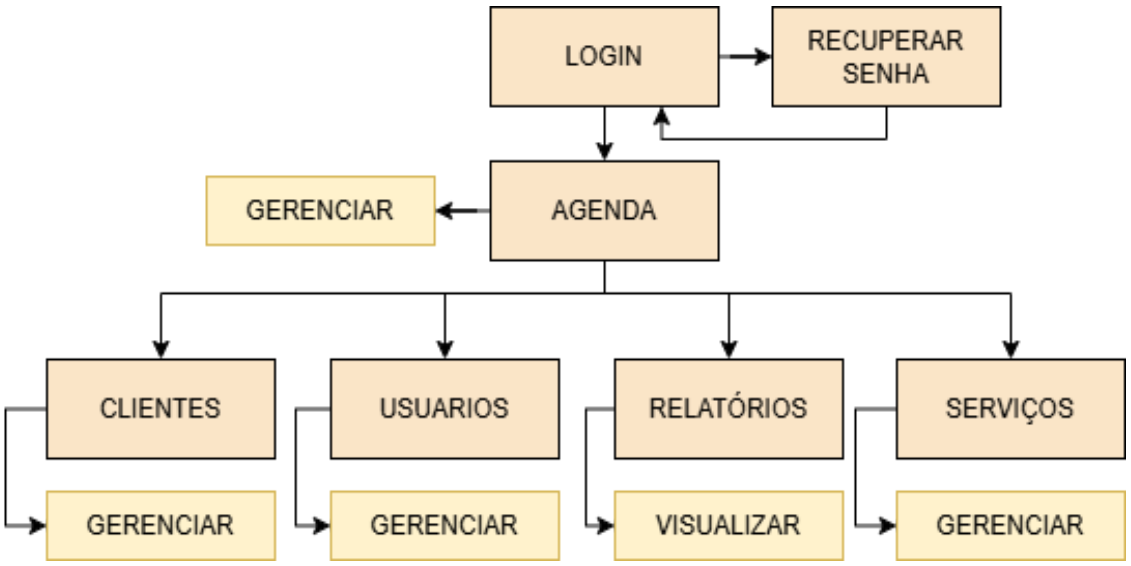


Figura 7. Fluxo de Navegação do Usuário no Aplicativo

Fonte: Elaborada pelo autor.

A Figura apresenta o fluxo de navegação do usuário no aplicativo, representando as principais etapas e caminhos possíveis dentro do sistema. O acesso ao aplicativo inicia pela tela de *login*, onde o usuário insere suas credenciais. Caso tenha esquecido a senha, pode utilizar a opção recuperar senha, que o direciona para o processo de redefinição por e-mail. Após o *login* bem-sucedido, o usuário é direcionado para a tela principal, que oferece acesso à agenda e ao módulo de gerenciamento. Na agenda, é possível visualizar, criar e editar agendamentos, permitindo o controle dos horários de atendimento. O módulo de gerenciamento dá acesso às seções de clientes,

usuários, relatórios e serviços. Em clientes, o usuário pode cadastrar, editar e desativar registros. Na seção de usuários, é possível gerenciar as credenciais de acesso ao sistema. Em relatórios, o usuário pode visualizar informações financeiras e operacionais, enquanto em serviços é possível gerenciar os tipos de atendimento e valores correspondentes.

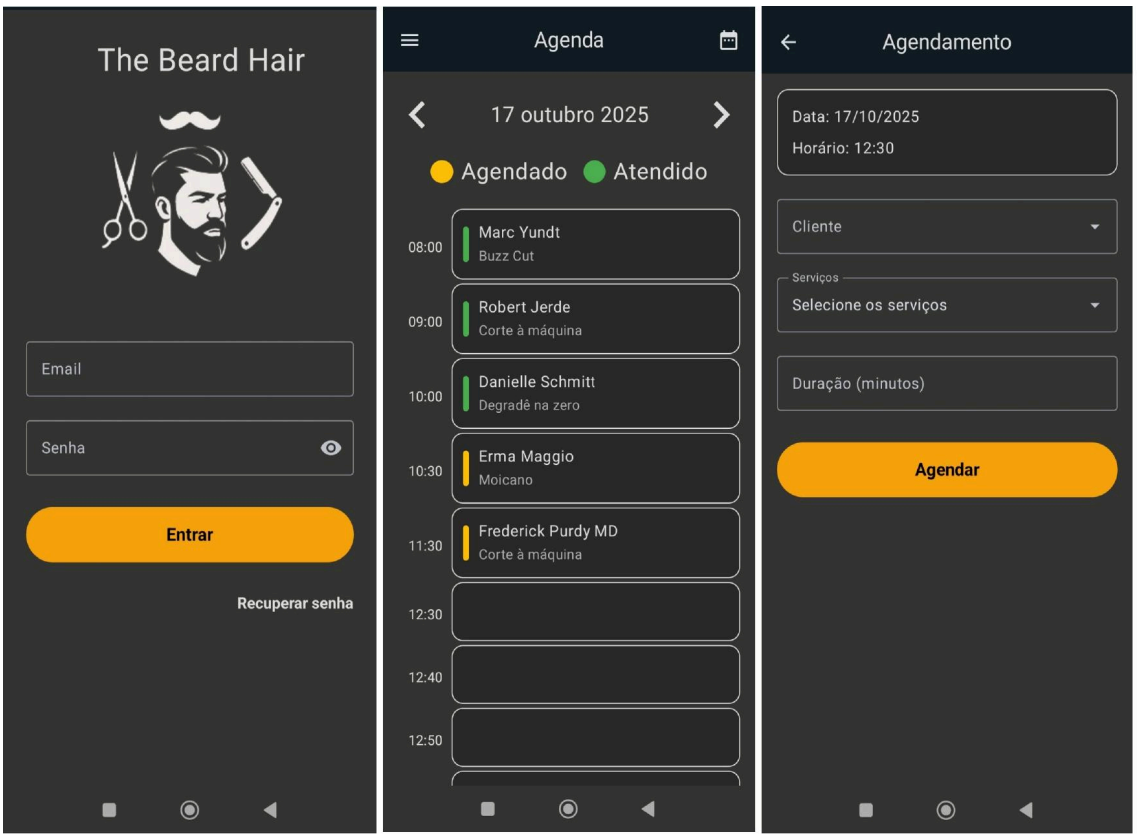


Figura 8. Interface de autenticação e gerenciamento de agendamentos

Fonte: Elaborada pelo autor.

A figura apresenta as principais telas do sistema, que compõem o fluxo de acesso e gerenciamento dos agendamentos. Na tela de *login*, o usuário realiza sua autenticação para garantir o acesso seguro às funcionalidades. Em seguida, a tela de agenda exibe os clientes agendados, permitindo ao barbeiro visualizar os horários disponíveis e os atendimentos do dia. A terceira tela mostra o formulário de agendamento, onde são inseridas as informações necessárias para registrar um novo atendimento, como o cliente, o serviço e a duração. Esse conjunto de telas representa o processo central do sistema, desde o acesso inicial até o registro efetivo de um agendamento.

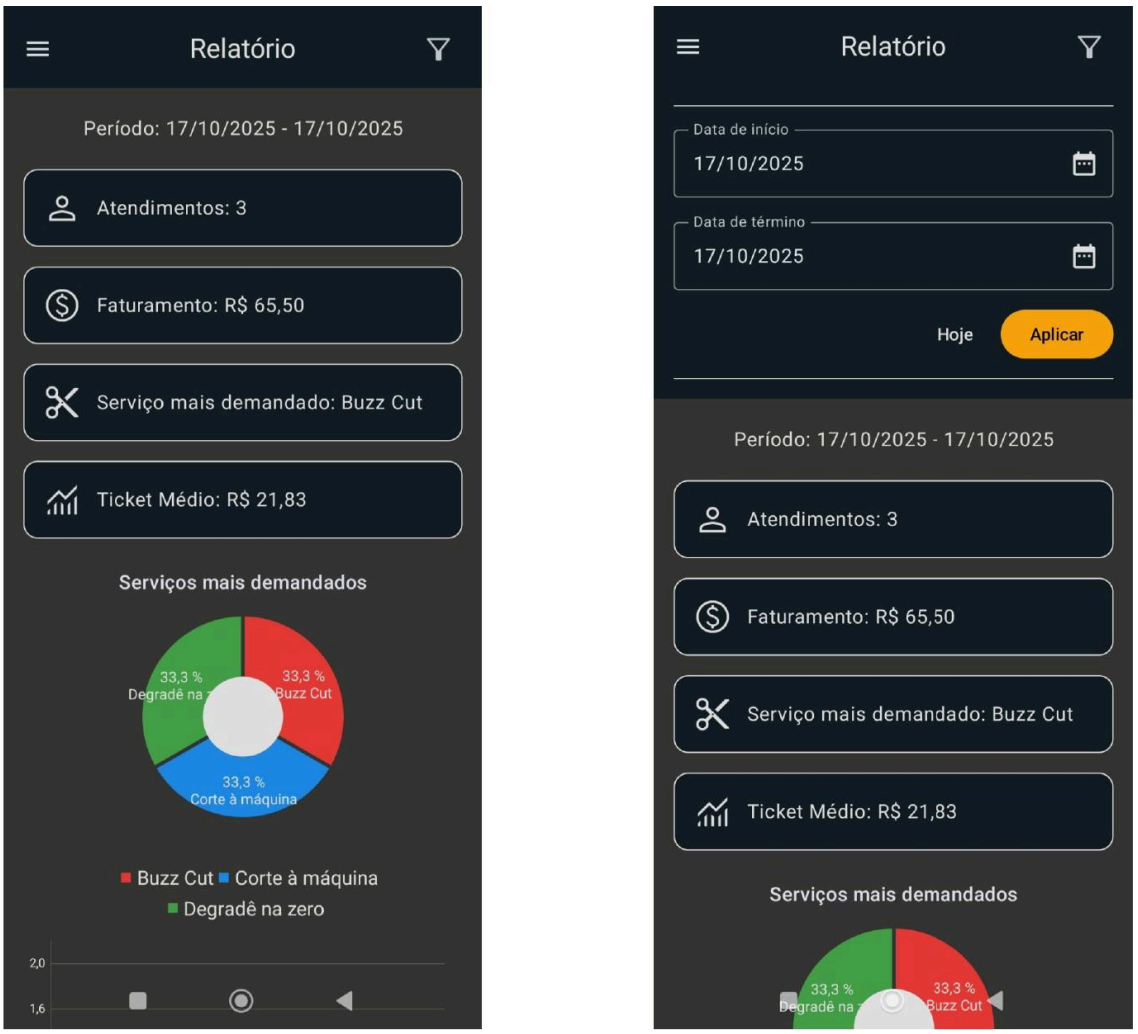


Figura 9. Geração de relatório financeiro e indicadores

Fonte: Elaborada pelo autor.

A tela de relatório, ao ser aberta, exibe por padrão as informações do dia atual. Para visualizar dados de um intervalo específico, o usuário deve clicar no ícone de filtro e selecionar as datas de início e término do relatório.

5.3. Testes Realizados

Após a validação automatizada da API, apresentada na Seção 4.4, o aplicativo Android foi testado manualmente pelo autor, garantindo que a experiência do usuário e os fluxos de interface funcionassem conforme o esperado. Esse processo também permitiu avaliar a integração do app com a API em situações reais de uso. Entre os cenários de teste considerados, destacam-se:

- Autenticação de usuário: foram feitas tentativas de *login* com credenciais válidas e inválidas, a fim de garantir que apenas usuários autorizados tenham acesso ao sistema.
- Agendamento: testou-se a criação de novos agendamentos, verificando a possibilidade de marcar horários já ocupados, garantindo o funcionamento correto da lógica de disponibilidade.

- Relatórios financeiros: validação dos cálculos dos indicadores, comparando os resultados obtidos com os agendamentos marcados como atendidos na agenda.
- Persistência de dados: verificação do salvamento e atualização correta das informações no banco de dados após operações de criação e edição.

Durante essa etapa, foi identificado um erro lógico na geração dos relatórios financeiros. O problema ocorria porque o cálculo do faturamento utilizava o valor atual do serviço cadastrado no banco de dados, sem considerar possíveis reajustes de preço ao longo do tempo. Por exemplo, se um corte de cabelo custava R$50,00 e posteriormente foi reajustado para R$60,00, o relatório somava os valores atualizados, distorcendo o histórico de faturamento.

A solução implementada consistiu em armazenar no documento de agendamento não apenas o identificador (ID) do serviço, mas também o valor praticado na data do atendimento, garantindo fidelidade histórica das informações financeiras e maior precisão nos indicadores de desempenho apresentados ao usuário.

5.4. *Deploy* e Integração Contínua

A API foi implantada utilizando integração e entrega contínuas (CI/CD) através do GitLab, o que automatizou as etapas de *build, test* e *deploy*. Essa configuração foi definida no arquivo *.gitlab-ci.yml*, cuja elaboração exigiu um esforço significativo, principalmente pela complexidade inicial de integração entre as etapas de *build* e *deploy*.

5.5. Comparação com Expectativas

O desenvolvimento alcançou os objetivos principais definidos no início do projeto, entregando uma aplicação funcional e de fácil utilização. A proposta inicial previa a criação de um sistema que facilitasse o controle de agendamentos e o acompanhamento de indicadores financeiros, metas que foram atendidas. As funcionalidades entregues correspondem ao essencial para o uso real de uma barbearia. Entre as ideias para futuras versões, destacam-se:

- Implementação de controle de pacotes (ex.: pacotes de cortes e barbas com preço promocional);
- Controle de produtos, permitindo o registro e a venda de itens como pomadas e cremes;
- Sistema de perfis e permissões, diferenciando o acesso de administradores e colaboradores.

5.6. Análise Crítica e Impactos

O projeto trouxe benefícios diretos tanto do ponto de vista técnico quanto prático. No aspecto técnico, proporcionou ao autor uma experiência completa envolvendo integração entre *back-end*, *front-end* e DevOps, aplicando na prática conceitos estudados durante a graduação. O desenvolvimento em Kotlin, apesar de desafiador inicialmente, resultou em uma aplicação funcional.

Do ponto de vista prático, o impacto para o usuário (barbeiro) é significativo. O sistema elimina a necessidade de agenda física, reduz riscos de esquecimento e

duplicidade de horários e possibilita o acompanhamento remoto da agenda e dos resultados financeiros. Os indicadores oferecidos auxiliam na tomada de decisões estratégicas, como ajustar preços, criar promoções ou oferecer descontos a clientes fieis, fortalecendo a gestão do negócio.

Para garantir objetividade na avaliação da solução, foram considerados indicadores de qualidade baseados na ISO/IEC 25010. Em relação à confiabilidade, a API obteve cobertura de testes automatizados de 96,65% das linhas de código foram executadas pelos testes, 97,77% das funções foram validadas, 96,53% das instruções (*statements*) foram cobertas e 82,38% dos caminhos condicionais (*branches*) foram exercitados Quanto ao desempenho, a medição realizada via Postman apontou tempos médios de resposta de aproximadamente 812 ms para autenticação, 225 ms para criação de agendamentos e 138 ms para listagem de serviços, valores aceitáveis para o contexto do projeto e para o ambiente executado em *cluster* remoto.

Nos testes manuais conduzidos no aplicativo *mobile*, realizados pelo próprio autor em cenários reais de uso, não foram identificados conflitos de horário, falhas de persistência ou inconsistências na comunicação com a API, reforçando a correção funcional dos fluxos principais. Em ambiente Kubernetes, a API permaneceu estável, com o *pod* executando de forma contínua por 47 horas sem qualquer reinicialização inesperada, caracterizando uma disponibilidade operacional próxima de 100% no intervalo analisado.

Como análise reflexiva, uma possível melhoria seria o uso do Flutter, permitindo compatibilidade multiplataforma (Android e iOS) e ampliando o alcance da aplicação. Essa decisão técnica poderia aumentar a adoção do sistema sem exigir grandes alterações no *back-end* já desenvolvido.

Em síntese, o projeto cumpriu seu papel de propor uma solução tecnológica aplicável a um problema real, reforçando o aprendizado prático e evidenciando a relevância da engenharia de *software* como ferramenta de transformação digital para pequenos negócios.

6. Conclusão

O desenvolvimento deste projeto teve como objetivo criar um sistema para digitalizar o processo de agendamento e controle financeiro da barbearia *The Beard Hair*, com potencial de replicação em outras microempresas que operam sob modelo semelhante de atendimento agendado. A solução proposta visou centralizar informações e automatizar tarefas que anteriormente eram realizadas de forma manual, contribuindo para maior organização, confiabilidade e agilidade no gerenciamento das atividades do estabelecimento.

Com base na implementação realizada, constatou-se que o sistema atendeu aos objetivos propostos, integrando de forma eficiente o *back-end*, desenvolvido em Node.js e implantado em um ambiente Kubernetes, com o aplicativo *mobile* construído em Kotlin. Essa integração resultou em uma estrutura funcional e bem organizada, permitindo a evolução gradual do sistema e a incorporação de novas funcionalidades no futuro, sem necessidade de alterações estruturais significativas. Entre os principais recursos desenvolvidos estão o gerenciamento de clientes, serviços, usuários e

agendamentos, além da geração de relatórios financeiros com indicadores de desempenho relevantes para a tomada de decisão.

Por questões de tempo, não foram realizados testes diretos com o usuário final, uma vez que o projeto não foi submetido ao Comitê de Ética em Pesquisa (CEP). No entanto, as validações internas realizadas durante o desenvolvimento indicaram o funcionamento adequado das funcionalidades e coerência com os requisitos definidos.

A experiência proporcionou um aprendizado significativo, principalmente por envolver a construção completa de uma solução tecnológica, englobando o desenvolvimento do *back-end*, *front-end* e a aplicação de práticas de DevOps para integração e entrega contínua. Esse processo favoreceu o aprimoramento técnico e a compreensão prática das etapas envolvidas no ciclo de vida de um *software* moderno.

Como trabalhos futuros, destaca-se a possibilidade de migração do aplicativo para o *framework* Flutter, permitindo compatibilidade multiplataforma, e a implementação de novas funcionalidades, como:

- Controle de pacotes de serviços: permitir que o usuário cadastre e gerencie pacotes promocionais compostos por múltiplos serviços.
- Gerenciamento de produtos: possibilitar o controle de estoque e a venda de produtos comercializados na barbearia.
- Interface *web*: desenvolver uma versão *web* do sistema, facilitando o uso em computadores, especialmente em casos em que seja necessário designar um funcionário para funções administrativas, como recepcionista ou secretário.
- Integração com sistemas de pagamento: automatizar o processo de recebimento financeiro, permitindo transações por meio de plataformas digitais.

Tais aprimoramentos poderiam ampliar o alcance e a utilidade do sistema, tornando-o ainda mais completo e adaptável a diferentes contextos de pequenos empreendimentos.

Além da contribuição técnica, o projeto também possui importância social, pois demonstra como soluções acessíveis e personalizadas podem promover a transformação digital em pequenos negócios. A digitalização de processos operacionais, antes realizados de forma manual, contribui não apenas para a eficiência do estabelecimento, mas também para o fortalecimento da economia local e a valorização do empreendedorismo. O aplicativo desenvolvido reforça o papel da tecnologia como otimizadora de trabalho e meio de inclusão no contexto das microempresas brasileiras.

Conclui-se, portanto, que o projeto desenvolvido representa uma contribuição relevante para a informatização de microempresas, especialmente no setor de serviços. A proposta demonstra o potencial da tecnologia para otimizar o tempo de atendimento, reduzir falhas humanas e fortalecer a relação entre profissional e cliente, promovendo maior eficiência e profissionalismo no ambiente de trabalho.

7. Referências

QUEIROZ JUNIOR, Francisco Antônio de; TERÊNCIO, Yago Henrique Silva. Fastbarber: aplicativo para organização e gerenciamento de tarefas, 2024. Trabalho de conclusão de curso. (Curso Superior de Tecnologia de Análise e Desenvolvimento

de Sistemas) - Faculdade de Tecnologia de Franca “Dr. Thomaz Novelinho”, Franca, 2024. Disponível em: https://ric.cps.sp.gov.br/handle/123456789/23027. Acesso em: 22 out. 2025.

JUROWITZ, Nathâne de Aguiar Alves. Aplicando kanban para um gerenciamento de projetos eficiente, 2024. Trabalho de conclusão de curso (Curso Superior de Tecnologia em Gestão Empresarial) - Faculdade de Tecnologia de Praia Grande, Praia Grande, 2024. Disponível em: https://ric.cps.sp.gov.br/handle/123456789/29612. Acesso em: 22 out. 2025.

STEIN JUNIOR, Antonio Ricardo Medronha. Arquitetura Rest API e desenvolvimento de uma Aplicação Web Service. Revista Alcides Maya, Porto Alegre, RS, 2019. v. 1, n.1,p.1-59. Disponível em: https://raam.alcidesmaya.com.br/index.php/projetos/article/view/97. Acesso em: 22 out. 2025.

SOUZA, Elaine Calasans; OLIVEIRA, Marcus Rogério de. COMPARATIVO ENTRE OS BANCOS DE DADOS MYSQL E MONGODB: quando o MongoDB é indicado para o desenvolvimento de uma aplicação. Revista Interface Tecnológica, Taquaritinga, SP, v. 16, n. 2, p. 38–48, 2019. DOI: 10.31510/infa.v16i2.664. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/664. Acesso em: 21 out. 2025.

BRASIL. Brasil registra abertura de 1,4 milhão de pequenos negócios no primeiro trimestre do ano. Governo do Brasil, 2025. Disponível em: https://www.gov.br/secom/pt-br/assuntos/noticias/2025/04/brasil-registra-abertura-de-1-4-milhao-de-pequenos-negocios-no-primeiro-trimestre-do-ano. Acesso em: 13 out. 2025.

CARVALHO, Davi Israel Abtibol. Desenvolvimento Android Moderno com Kotlin e Jetpack Compose: Um Guia Prático Baseado no App PlainTextKotlin. 2025. Trabalho de conclusão de curso (Curso de Engenharia da Computação) - Universidade Federal do Amazonas - UFAM, Manaus - AM, 2025. Disponível em: http://riu.ufam.edu.br/handle/prefix/8925. Acesso em: 22 out. 2025.

MELO, Mateus Souza de; SILVA, Edvaldo de Oliveira da. DATAW: API Web para anonimização de dados. 2024. Trabalho de conclusão de curso (Curso de Sistemas de Informação) - Associação Propagadora Esdeva Centro Universitário Academia – UniAcademia, Juiz de Fora - MG, 2024. Disponível em: https://seer.uniacademia.edu.br/index.php/cesi/article/view/4199/3299. Acesso em: 22 out. 2025.

SILVA, Kaique Rierickson Torres. Implementação e Orquestração Automatizada de Clusteres Kubernetes com GitOps: Um Estudo de Caso. 2023. Trabalho de conclusão de curso (Bacharel em Engenharia de Software) - Instituto Federal de Pernambuco Campus Belo Jardim, Belo Jardim, PB, 2023. Disponível em: https://repositorio.ifpe.edu.br/xmlui/handle/123456789/1056. Acesso em: 22 out. 2025.

POLATO, Matheus Willian. AGON: plataforma de agendamento e controle online de horários. 2023. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de

Informação) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Bauru, 2023. Disponível em: http://hdl.handle.net/11449/239498. Acesso em: 22 out. 2025.

CARDOSO, Rodrigo de Castro. Aplicativo móvel para registro e compartilhamento de mídias usando mapa interativo. 2025. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Universidade Federal de Uberlândia – UFU Faculdade de Computação, Uberlândia, 2025. Disponível em: https://repositorio.ufu.br/handle/123456789/45983. Acesso em: 10 nov. 2025.

TEIXEIRA FILHO, Francisco Chagas. Desenvolvimento de um Backend para Aplicação de Adoção de Animais para o Grupo Unidos pelas Patinhas - Pau dos Ferros/RN. 2025. Trabalho de Conclusão de Curso (Bacharelado Interdisciplinar em Ciência e Tecnologia) - Universidade Federal do Semi-árido - UFERSA, Pau dos Ferros, RN, 2025. Disponível em: https://repositorio.ufersa.edu.br/server/api/core/bitstreams/fa2c7680-85dc-40a1-8cfd-05789d2ef60a/content. Acesso em: 10 nov. 2025.

MARQUES, Fernando Souza, Silva, Leandro Cesar. Desenvolvimento de Aplicativo PWA como instrumento didático na aprendizagem de língua estrangeira. 2020. Trabalho de conclusão de curso. (Curso Superior de Tecnologia de Análise e Desenvolvimento de Sistemas) - Faculdade de Tecnologia de Franca “Dr. Thomaz Novelinho”, Franca, SP, 2020. Disponível em: http://ric.cps.sp.gov.br/handle/123456789/7049. Acesso em: 10 nov. 2025.