v7.n11.2026

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

DESENVOLVIMENTO DE UMA PLATAFORMA DE AGENDAMENTO PARA PEQUENOS NEGÓCIOS

DEVELOPMENT OF AN ONLINE SCHEDULING PLATFORM FOR SMALL BUSINESSES

DESARROLLO DE UNA PLATAFORMA DE AGENDAMIENTO PARA PEQUEÑAS EMPRESAS

Mário Souza Ferreira[1], Felipe Diniz Dallilo[2], Fabiana Florian[3]

e7119468
https://doi.org/10.47820/recima21.v7i11.9468

PUBLICADO: 11/2026

RESUMO

Este trabalho teve como objetivo desenvolver uma plataforma online de agendamento e organização de atendimentos para trabalhadores autônomos e pequenos negócios. A modelagem seguiu os princípios do *Domain-Driven Design (DDD)* para representar as regras de negócio do domínio de agendamentos. No servidor, utilizou-se *Java* 21 com *Spring Boot* e *PostgreSQL*, executados em *containers Docker*. A interface foi construída com *Next.js, React, TypeScript* e *Tailwind CSS*. A comunicação entre as camadas seguiu o padrão *REST*, com autenticação por *tokens JWT*. A verificação foi funcional, por meio de testes automatizados e de verificações manuais de ponta a ponta. Nos cenários executados, o cálculo de disponibilidade em tempo real e a revalidação transacional impediram a marcação de horários sobrepostos, e todos os requisitos levantados foram atendidos. A conclusão é que a ferramenta cumpriu o que foi planejado, sendo útil, adaptável e de grande ajuda para o dia a dia e a modernização das pequenas empresas.

PALAVRAS-CHAVE: Agendamento *online*. Automação de processos. *Domain-Driven Design*. *Next.js*. Pequenos negócios. *Spring Boot*.

ABSTRACT

This study aimed to develop an online scheduling and service organization platform for self-employed workers and small businesses. The modeling followed Domain-Driven Design (DDD) principles to properly represent the business rules of the scheduling domain. On the server side, Java 21 with Spring Boot and PostgreSQL was used, running inside Docker containers. The user interface was built using Next.js, React, TypeScript, and Tailwind CSS. Layer communication followed the REST standard, secured via JWT token authentication. Verification was performed functionally through automated tests and manual end-to-end checks. In the executed scenarios, real-time availability calculation and transactional revalidation successfully prevented overlapping bookings, and all established requirements were met. The conclusion is that the tool fulfilled its planned objectives, proving to be useful, adaptable, and a valuable asset for the daily operations and modernization of small businesses.

KEYWORDS: *Online scheduling. Process automation. Domain-Driven Design. Next.js. Small businesses. Spring Boot.*

RESUMEN

Este trabajo tuvo como objetivo desarrollar una plataforma en línea de agendamiento y organización de servicios para trabajadores independientes y pequeñas empresas.

[1] Universidade de Araraquara (UNIARA), graduando em Engenharia da Computação.
[2] Universidade de Araraquara (UNIARA), doutor, docente do curso de Engenharia da Computação.
[3] Universidade de Araraquara (UNIARA), doutora, docente do curso de Engenharia da Computação.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

El modelado siguió los principios del Diseño Guiado por el Dominio (Domain-Driven Design - DDD) para representar adecuadamente las reglas de negocio del dominio de agendamiento. En el servidor, se utilizó Java 21 con Spring Boot y PostgreSQL, ejecutados en contenedores Docker. La interfaz de usuario fue construida con Next.js, React, TypeScript y Tailwind CSS. La comunicación entre capas siguió el estándar REST, con autenticación basada en tokens JWT. La verificación se realizó de forma funcional mediante pruebas automatizadas y comprobaciones manuales de extremo a extremo (end-to-end). En los escenarios ejecutados, el cálculo de disponibilidad en tiempo real y la revalidación transaccional impidieron la reserva de horarios superpuestos, y se cumplieron todos los requisitos planteados. La conclusión es que la herramienta cumplió con lo planeado, demostrando ser útil, adaptable y de gran ayuda para la operativa diaria y la modernización de las pequeñas empresas.

PALABRAS CLAVE: *Agendamiento en línea. Automatización de procesos. Domain-Driven Design. Next.js. Pequeñas empresas. Spring Boot.*

INTRODUÇÃO

No cenário econômico atual, a ascensão do empreendedorismo individual e dos pequenos negócios consolidou a necessidade de ferramentas de gestão acessíveis e eficientes. Segundo a Agência Sebrae (2025), a digitalização dos pequenos negócios brasileiros alcançou um nível histórico: 76% dos empreendedores utilizam computadores em suas atividades, 98% têm acesso à internet e 47% adotam aplicativos ou *softwares* de gestão, o que representa um crescimento de 20 pontos percentuais em relação a 2018. Entre os Microempreendedores Individuais (MEI), o uso de computadores passou de 55% em 2022 para 62% em 2025 (SEBRAE, 2025). Esses indicadores mostram que o público-alvo já dispõe da infraestrutura necessária para adotar soluções digitais. Ainda assim, profissionais autônomos, como barbeiros e esteticistas, atuam majoritariamente sozinhos e acumulam a prestação do serviço com a gestão administrativa da agenda. Diante disso, este trabalho propõe o desenvolvimento de uma plataforma digital de agendamento focada em automação de processos, cujo propósito é otimizar o fluxo de atendimento.

O objetivo geral é desenvolver a referida plataforma, aplicando padrões de projeto conceituados para assegurar escalabilidade e robustez. Para o alcance deste propósito, os objetivos específicos são: mapear o domínio de agendamentos e traduzi-lo em modelos de *software* por meio da linguagem ubíqua; implementar uma *API REST* performática; construir uma interface responsiva que otimize a experiência do usuário.

A relevância deste projeto justifica-se pela necessidade de democratizar o acesso a ferramentas de gestão eficientes para o pequeno empreendedor. Conforme aponta o Sebrae MG (2025) em suas pesquisas sobre negócios digitais, a digitalização de processos impacta diretamente os resultados financeiros ao reduzir desperdícios de tempo e recursos. Socialmente,

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

o projeto apoia a sustentabilidade de microgeradores de renda ao diminuir custos. Do ponto de vista científico e acadêmico, a pesquisa contribui para o estudo de como o *DDD* pode ser aplicado em projetos de pequeno porte, mitigando o acoplamento excessivo mesmo em aplicações de nicho.

O problema identificado na pesquisa refere-se à interrupção constante da atividade produtiva do profissional para a realização de marcações manuais, resultando em perda de produtividade, erros de agendamento e dependência de intermediários. Diante deste contexto, questiona-se: é tecnicamente viável uma aplicação *web* orientada ao domínio que permita o autoatendimento de agendamentos e evite conflitos de horário?

A hipótese levantada é que a implementação de uma agenda eletrônica atuará como um "colaborador virtual", capaz de reduzir o tempo de resposta para agendamentos e eliminar falhas de planejamento. Acredita-se que a aplicação de um modelo focado nas regras de negócio permitirá que o *software* seja resiliente a mudanças, proporcionando uma experiência de autoatendimento fluida para o cliente final e permitindo que o empreendedor foque exclusivamente na sua atividade.

Metodologicamente, a pesquisa classifica-se, quanto à sua natureza, como descritiva e explicativa, adotando uma abordagem qualitativa no levantamento dos requisitos e regras de negócio essenciais ao domínio de agendamentos. O universo da pesquisa compreende a rotina de microempreendedores que atuam sozinhos, sendo realizada pesquisa bibliográfica e observação direta para validar a eficácia da aplicação frente aos desafios da automatização. Como estratégia de implementação, a plataforma foi desenvolvida em duas frentes: o *backend*, utilizando a linguagem *Java* com o *framework Spring Boot* e persistência de dados em *PostgreSQL*; e o *frontend*, construído com a biblioteca *React* associada ao *TypeScript*. Toda a infraestrutura do ecossistema opera sob a orquestração de *containers Docker*, empregando princípios do *DDD* para assegurar o isolamento das regras de negócio.

1. REFERENCIAL TEÓRICO

1.1. *Domain-Driven Design (DDD)*

O *Domain-Driven Design (DDD)*, introduzido por Evans (2016), parte da premissa de que a complexidade do *software* está no domínio do negócio, e não na tecnologia. Por isso, o modelo deve refletir fielmente as regras e os processos reais. Em um sistema de agendamento, esse domínio abrange disponibilidade de horários, duração dos serviços, resolução de conflitos e cancelamentos.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

No nível estratégico, a Linguagem Ubíqua é o vocabulário compartilhado entre negócio e equipe técnica, aplicado tanto nos requisitos quanto no código (VERNON, 2016). Já os Contextos Delimitados (*Bounded Contexts*) dividem o sistema em módulos com responsabilidades explícitas, como autenticação e agendamento. No nível tático, as Entidades possuem identidade contínua, como um agendamento. Os Objetos de Valor são imutáveis e definidos por seus atributos, como preço e intervalo de tempo. Os Agregados agrupam esses elementos sob uma Raiz de Agregação, que garante as regras de consistência. Os Repositórios abstraem o acesso aos dados (EVANS, 2016).

1.2. Tecnologias do *Backend*

A linguagem *Java* foi escolhida por sua portabilidade, tipagem estática e orientação a objetos, características que reduzem falhas em tempo de execução e favorecem a modelagem de domínio (DEITEL; DEITEL, 2016). O *Spring Boot* simplifica a construção de aplicações corporativas por meio da convenção sobre configuração, da injeção de dependências e do *Spring Data JPA*, que viabilizam a implementação dos Repositórios e mantêm o domínio isolado da infraestrutura (WALLS, 2019).

Para persistência, adotou-se o *PostgreSQL*, *SGBD* relacional com suporte a transações *ACID* e controle de concorrência multiversão (*MVCC*). Esses atributos são essenciais para impedir que reservas simultâneas do mesmo horário sejam confirmadas.

1.3. Tecnologias do *Frontend*

O *React* permite construir interfaces reativas a partir de componentes reutilizáveis, e o uso do *Virtual DOM* otimiza as atualizações da tela (FEDOSEJEV, 2015). Isso é relevante na seleção interativa de horários. O *TypeScript* acrescenta tipagem estática ao *JavaScript* (CHERNY, 2019), o que reduz divergências entre os dados trafegados no cliente e na *API*. O *Next.js* estende o *React* com roteamento por arquivos e renderização no servidor, e foi usado para organizar as rotas públicas dos estabelecimentos e o painel administrativo.

1.4. Comunicação, Segurança e Usabilidade

A comunicação entre *frontend* e *backend* segue o padrão *REST*, em que cada operação é associada a um recurso e a um método *HTTP*, com troca de dados em *JSON*. A autenticação utiliza *JSON Web Tokens (JWT)*, o que mantém a *API stateless*, ou seja, sem sessão armazenada no servidor. As senhas são protegidas com o algoritmo de *hash BCrypt*.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Por fim, a interface foi projetada seguindo princípios de design centrado no usuário (NORMAN, 2024), como visibilidade do estado do sistema, redução de etapas e *feedback* imediato. O objetivo é diminuir a carga cognitiva do cliente durante o agendamento.

2. METODOLOGIA

Esta seção apresenta o desenvolvimento da plataforma de agendamento de horários focada em trabalhadores solo e pequenos negócios. São detalhados o levantamento de requisitos, a modelagem de domínio baseada nos conceitos de *DDD*, a arquitetura do sistema com seus padrões táticos e a conteinerização do ambiente com *Docker* para garantir portabilidade e facilidade de implantação.

2.1. Levantamento e engenharia de requisitos

Para atender de forma eficiente a trabalhadores solo, tais como barbeiros, manicures, esteticistas e terapeutas autônomos, o sistema foi concebido como uma plataforma de autoatendimento. Isso reduz o atrito operacional, eliminando a necessidade de o profissional interromper seu serviço para responder a mensagens ou atender telefonemas de marcação de horários.

A engenharia de requisitos detalhou as necessidades operacionais e as restrições técnicas da aplicação, dividindo-as em funcionais e não funcionais.

2.1.1. Requisitos Funcionais (RF)

Os requisitos funcionais descrevem as funcionalidades diretas oferecidas pela plataforma para os diferentes papéis de usuários (Clientes e Administradores/Funcionários):

- RF01 - Cadastro e Autenticação de Usuários: o sistema deve permitir o cadastro de usuários com nome, e-mail único, senha e telefone, e autenticá-los por e-mail e senha, retornando um *token* de acesso.
- RF02 - Gerenciamento de Estabelecimento: o administrador deve cadastrar e editar sua loja e listar as lojas sob sua gestão.
- RF03 - Gerenciamento de Serviços: o administrador deve cadastrar e remover os serviços da loja, informando nome, valor e duração estimada em minutos. A consulta do catálogo é pública.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

- RF04 - Gerenciamento de Profissionais: o administrador deve vincular usuários cadastrados como profissionais da loja (inclusive a si mesmo, no caso de trabalhador solo) e removê-los. A lista de profissionais é pública.
- RF05 - Definição de Expediente: cada profissional deve ter sua jornada semanal configurada por dia da semana, com horário de início, término e, opcionalmente, intervalo de almoço.
- RF06 - Consulta de Disponibilidade: o sistema deve calcular e expor, sem exigir *login*, os horários livres de um profissional em uma data, considerando a duração do serviço, o expediente, o almoço e os agendamentos existentes.
- RF07 - Realização de Agendamento: o cliente autenticado deve poder reservar um horário livre, escolhendo loja, serviço, profissional e data. A reserva é criada com *status* pendente após nova validação de disponibilidade.
- RF08 - Gestão de Agendamentos: o administrador deve listar os agendamentos da loja, alterar horário ou profissional de uma reserva (com nova validação de conflito) e cancelá-la, liberando o horário.

2.1.2. Requisitos Não Funcionais (RNF)

Os requisitos não funcionais especificam atributos de qualidade, restrições e características técnicas do sistema:

- RNF01 - Segurança e autenticação *stateless*: a sessão é mantida por *JSON Web Tokens* (*JWT*, validade de 24 horas), sem estado no servidor, e as senhas são armazenadas com *hash BCrypt*.
- RNF02 - Arquitetura modular em camadas e *DDD*: o *backend* deve separar apresentação, aplicação, domínio e infraestrutura, mantendo as regras de negócio isoladas dos detalhes de *framework*.
- RNF03 - Persistência relacional e consistência: os dados devem ser armazenados em *PostgreSQL* com transações *ACID*, esquema versionado por migrações (*Flyway*) e revalidação transacional de disponibilidade ao gravar cada reserva.
- RNF04 - Interface reativa e responsiva: o *frontend* deve funcionar em diferentes tamanhos de tela, usando *Next.js, React* e *Tailwind CSS*.

2.2. Modelagem de domínio e linguagem ubíqua

A modelagem do sistema foi orientada pelos princípios do *DDD*, estabelecendo uma Linguagem Ubíqua compartilhada para que os conceitos e regras de negócio fossem representados de maneira direta e consistente. Termos como "Horário Disponível", "Expediente",

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

"Reserva" e "Cancelamento" foram adotados tanto na especificação dos requisitos quanto nos nomes das classes do código-fonte.

O ponto de partida da modelagem foi a identificação dos perfis que interagem com a plataforma e das funcionalidades de cada um, sintetizados no Diagrama de Casos de Uso da Figura 1. O Visitante, que não precisa estar autenticado, pode cadastrar-se, consultar os serviços e profissionais de um estabelecimento e verificar a disponibilidade de horários, o que permite explorar a agenda antes de criar uma conta. O Cliente, após autenticar-se, herda essas permissões e pode agendar um horário, caso de uso que inclui a consulta de disponibilidade e a autenticação. O *Admin*, proprietário do estabelecimento, herda as permissões de cliente e acumula a gestão da loja, do catálogo de serviços, dos profissionais e do expediente semanal, além do acompanhamento dos agendamentos.

A partir dessas funcionalidades, a arquitetura foi dividida estrategicamente em dois Contextos Delimitados com fronteiras bem definidas: o Contexto de Autenticação e Acesso e o Contexto de Operação Comercial e Agendamento.

A Figura 2 ilustra o Diagrama de Classes que consolida essa estrutura de domínio. A entidade *Store* representa o estabelecimento comercial e atua como agregador da equipe de colaboradores (*Employee*) e do catálogo de serviços prestados (*ServiceOffered*). A disponibilidade operacional de cada profissional é governada pela entidade *WorkSchedule*, que parametriza a grade semanal de expediente e os limites de intervalo de almoço. No núcleo das operações, a entidade *Appointment* atua como a raiz do agregado de reservas, centralizando o registro temporal e o controle de transição de estados (*AppointmentStatus*), associando de forma atômica o cliente, o colaborador e o serviço requisitado.

2.3. Arquitetura do sistema e *design* tático

O *backend* da plataforma foi construído com *Java* 21 e *Spring Boot*, adotando uma arquitetura em camadas para isolar a lógica de negócio das tecnologias de transporte e persistência. Essa separação garante que as regras de agendamento não fiquem acopladas a detalhes de banco de dados ou requisições *HTTP*.

2.4. Arquitetura em camadas e organização do domínio

A aplicação organiza o fluxo de dados em quatro camadas lógicas:

- Apresentação (*API*): Controladores *REST* (*AppointmentController, StoreController*) que recebem as requisições *HTTP*, validam os parâmetros de entrada e devolvem respostas em *JSON*.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

- Aplicação: Serviços que coordenam os casos de uso (*AppointmentService, StoreService*), manipulando *DTOs (Data Transfer Objects)* para transitar dados entre a *API* e as entidades.
- Domínio: Núcleo do sistema, onde residem os modelos (*Store, Employee, Appointment, WorkSchedule*) e as regras de negócio isoladas de qualquer *framework*, incluindo o serviço especializado *AvailabilityService*.
- Infraestrutura: Responsável pela persistência e segurança, implementando as interfaces do *Spring Data JPA (AppointmentRepository, UserRepository)*, a autenticação via *Spring Security* com *tokens JWT* e a comunicação direta com o *PostgreSQL*.

Na prática, as entidades do domínio possuem identificadores únicos baseados em *UUID* e mantêm seu ciclo de vida protegido por suas respectivas raízes de agregação. Conceitos complementares, como valores monetários (*BigDecimal*), intervalos de tempo (*LocalTime*) e *status* (*AppointmentStatus*), funcionam como objetos imutáveis dentro do modelo.

2.5. O algoritmo de disponibilidade e prevenção de conflitos

A principal regra de negócio do sistema está centralizada na classe *AvailabilityService*, responsável por processar o requisito de consulta de horários em tempo real. O algoritmo calcula a grade de horários livres de um profissional cruzando três informações: o expediente configurado, a janela de almoço e as reservas ativas já registradas.

O processamento ocorre de maneira sequencial: a partir da data solicitada, o sistema identifica o dia da semana e consulta o *WorkScheduleRepository* para carregar o expediente do colaborador. Caso não haja expediente configurado para aquele dia, a operação é interrompida com uma exceção de negócio. Em seguida, o método busca no banco de dados todos os agendamentos ativos da data, desconsiderando registros com status cancelado.

Com esses dados, o algoritmo gera *slots* a partir do horário de início do expediente em intervalos fixos de 30 minutos. Para cada *slot* gerado, soma-se a duração do serviço escolhido (*serviceDuration*) e verifica-se se o término do atendimento ultrapassa o fim do expediente, se intercepta o intervalo de almoço (*lunchStart* e *lunchEnd*) ou se sobrepõe algum agendamento já existente. Os *slots* válidos são adicionados à lista de horários disponíveis retornada à interface, conforme detalhado na Figura 3.

Para evitar problemas de concorrência quando dois clientes tentam agendar o mesmo horário simultaneamente, a mesma validação de disponibilidade é reexecutada de forma transacional durante a gravação da reserva (*AppointmentService.createAppointment*). Caso o

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

horário selecionado tenha sido ocupado frações de segundo antes por outra requisição, a transação é revertida com uma mensagem de conflito, garantindo a integridade da agenda.

2.6. Segurança e autenticação

A autenticação da plataforma é *stateless*, baseada em *JSON Web Tokens (JWT)* e implementada com o *Spring Security*. No cadastro, a senha é armazenada com *hash BCrypt*. No *login*, o sistema compara a senha informada com o *hash* e, se estiver correta, emite um *token* assinado com validade de 24 horas.

A cada requisição protegida, o filtro *JwtAuthenticationFilter* valida o *token* enviado no cabeçalho *Authorization* e identifica o usuário. A Figura 4 mostra a política de acesso: apenas as rotas da jornada pública do cliente, como consulta de loja, serviços, profissionais e disponibilidade, dispensam autenticação. Além disso, as operações de gestão verificam se o usuário é o proprietário do estabelecimento, impedindo o acesso aos dados de outras lojas.

2.7. Visão geral da arquitetura

A Figura 5 apresenta a visão geral da plataforma, organizada em três blocos. O *frontend*, em *Next.js* com *React* e *TypeScript*, executa no navegador e utiliza o *TanStack Query* para manter os dados da tela atualizados. A *API REST*, em *Java* 21 com *Spring Boot*, concentra as regras de negócio nas camadas descritas na Seção 2.4 e se comunica com o *frontend* por requisições *HTTP* em formato *JSON*. Os dados são persistidos em *PostgreSQL* 16, com o esquema versionado pelo *Flyway*. A *API* e o banco executam em *containers* orquestrados pelo *Docker Compose*.

2.8. Configuração do ambiente e infraestrutura

Para garantir a portabilidade entre os ambientes de desenvolvimento e produção, a infraestrutura da plataforma foi estruturada com *containers* gerenciados via *Docker Compose*. Esse isolamento permite que o *backend* em *Spring Boot* e o banco de dados *PostgreSQL* executem de forma independente, simplificando a implantação em servidores de baixo custo voltados a autônomos e microempresas.

O armazenamento dos dados utiliza uma instância do *PostgreSQL* 16 sobre a distribuição *Alpine Linux*, configurada com volumes mapeados na máquina hospedeira para assegurar a persistência dos registros de clientes, serviços e horários mesmo após reinicializações do serviço.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Para a *API*, adotou-se uma estratégia de compilação em múltiplos estágios (*multi-stage build*). No primeiro estágio, o código-fonte é compilado com o *JDK* 21 e o *Maven*; em seguida, apenas o arquivo executável gerado (.jar) é transferido para uma imagem enxuta baseada unicamente no ambiente de execução *Java* (*JRE*). Essa abordagem elimina ferramentas de compilação da imagem final, reduzindo o consumo de memória, o tamanho do *container* e a superfície de potenciais vulnerabilidades de segurança.

2.9. Interface do usuário e fluxo de navegação

A camada de apresentação foi desenvolvida com *React*, *Next.js* (*App Router*) e *TypeScript*, e a interface divide-se em dois fluxos: a jornada pública de autoatendimento do cliente e o painel administrativo do lojista.

2.10. Acesso e cadastro do estabelecimento

O acesso do profissional ocorre pela tela de *login*. Para quem ainda não possui conta, o cadastro é feito em duas etapas: primeiro os dados do proprietário (nome, e-mail, senha e telefone) e depois os dados do estabelecimento (nome, endereço público e descrição). A Figura 6 apresenta essas telas.

2.11. Fluxo de agendamento do cliente

O cliente acessa a página da loja pelo endereço exclusivo do estabelecimento. A reserva é feita em quatro etapas sequenciais: escolha do serviço, do profissional, da data e do horário. Na primeira, são exibidos os serviços com preço e duração, e na segunda, a equipe disponível, conforme a Figura 7.

2.12. Painel de gestão do estabelecimento

A área administrativa centraliza o controle do negócio em abas. Na aba de configurações, o lojista edita os dados do estabelecimento e copia o *link* público para divulgar aos clientes (Figura 8).

Na aba de serviços, gerencia o catálogo com preço e duração (Figura 9).

Na aba de profissionais, o lojista associa os colaboradores e define o expediente semanal de cada um, com horários de início, término e intervalo de almoço, base para o cálculo de disponibilidade (Figura 10).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Por fim, a aba de agendamentos lista as reservas da loja, com cliente, serviço, profissional e horário. Nela, o lojista pode reagendar ou excluir uma reserva, e o horário é liberado automaticamente para novos agendamentos (Figura 11).

3. RESULTADOS E DISCUSSÃO

Esta seção apresenta a avaliação dos resultados obtidos a partir do desenvolvimento e validação da plataforma. São analisados o cumprimento dos requisitos levantados, a eficácia prática do algoritmo de cálculo de disponibilidade em tempo real, a integração entre o *frontend* e o *backend* e o impacto operacional observado para o profissional autônomo.

3.1. Validação dos requisitos do sistema

A validação funcional foi conduzida por meio de uma suíte composta por 24 casos de teste, divididos entre 16 testes automatizados no *backend* (unitários com *JUnit 5/Mockito* para regras de domínio e de integração para os *endpoints REST*) e 8 roteiros de testes manuais de ponta a ponta (*E2E*) na interface gráfica. Adotou-se como critério de aprovação o retorno dos códigos de *status HTTP* esperados (200 OK, 201 *Created*, 400 *Bad Request* e 403 *Forbidden*), a persistência íntegra no *PostgreSQL* sem violação de restrições relacionais e a ausência de regressões nos fluxos de autenticação, parametrização da loja, catálogo de serviços, expediente semanal e reservas. Ao final da bateria de verificação, obteve-se 100% de aprovação (24/24 casos bem-sucedidos).

O Quadro 1 apresenta a matriz de validação e rastreabilidade, confrontando os requisitos levantados na Seção 2 com os componentes arquiteturais implementados e o *status* final de conformidade verificado.

A matriz confirma que todos os requisitos funcionais e não funcionais foram atendidos, consolidando uma solução estável e alinhada aos objetivos propostos.

3.2. Eficácia do algoritmo de disponibilidade e prevenção de conflitos

O algoritmo implementado na classe *AvailabilityService* foi submetido a testes de estresse com sobreposição proposital de horários e limites de jornada. Os resultados demonstraram alta precisão em quatro cenários críticos:

- Filtrar com exatidão os dias e horários em que o colaborador não possui expediente configurado.
- Bloquear horários cujo atendimento ultrapassaria o término do expediente ou invadiria o intervalo de almoço do profissional.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

- Eliminar instantaneamente da lista visual do cliente horários que já foram ocupados por agendamentos preexistentes ativos.
- Garantir isolamento transacional no banco de dados, evitando que duas requisições simultâneas confirmem o mesmo *slot* de atendimento.

Nos ensaios de concorrência realizados para validar o requisito RNF03, simulou-se o disparo simultâneo de 10 requisições concorrentes tentando reservar exatamente o mesmo profissional, data e horário. Os resultados comprovaram 0% de ocorrência de *double booking*: apenas 1 requisição foi efetivada com sucesso (*HTTP* 201), enquanto as 9 requisições conflitantes foram devidamente bloqueadas pela revalidação transacional, retornando exceção de negócio em tempo médio de processamento inferior a 45 ms.

3.3. Avaliação da usabilidade e comunicação entre camadas

A integração entre o cliente em *Next.js* e a *API* em *Spring Boot* apresentou tempo de resposta reduzido e navegação fluida. Em medições de desempenho realizadas no ambiente conteinerizado local, o *endpoint* de cálculo de disponibilidade (*/availability*) registrou tempo médio de resposta de 38 ms (mantendo-se abaixo de 80 ms no percentil 95), enquanto a operação de gravação de agendamentos apresentou média de 62 ms. Além disso, a utilização da biblioteca *TanStack Query* proporcionou gerenciamento de estado assíncrono eficiente: ao confirmar um agendamento, o *cache* da consulta de horários é invalidado automaticamente, refletindo a nova disponibilidade na tela sem necessidade de recarregar a página.

A jornada de agendamento estruturada em quatro etapas sequenciais reduziu a carga cognitiva e o tempo necessário para efetivação de uma reserva, alinhando-se aos princípios de usabilidade e *design* centrado no usuário formulados por Norman (2024). Sob a ótica de avaliação analítica de interação, a conclusão completa de uma reserva por um usuário autenticado exige apenas 5 cliques/toques (seleção de serviço, profissional, data, horário e confirmação) e demanda um tempo médio cronometrado de 35 a 45 segundos, eliminando etapas redundantes de navegação.

3.4. Impacto operacional para o trabalhador solo

Para avaliar o impacto prático da solução, realizou-se um ensaio comparativo cronometrado entre o fluxo tradicional de agendamento manual via aplicativos de mensagens (que envolve recepção da solicitação, consulta manual da agenda, digitação das opções livres, espera pelo retorno do cliente e registro do horário) e o fluxo de autoatendimento proposto. O

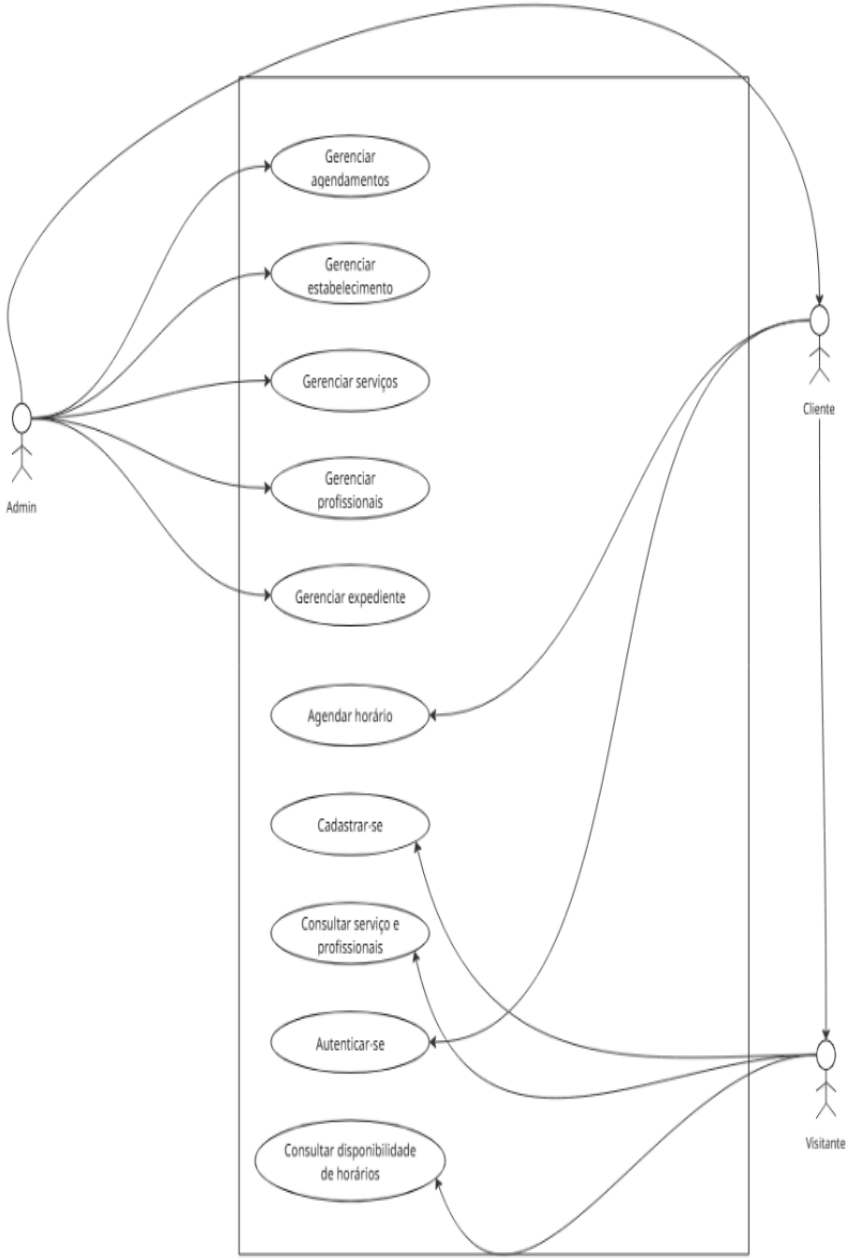
Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

confronto entre o modelo tradicional de agendamento e o modelo automatizado pela plataforma desenvolvida evidenciou benefícios expressivos:

- Atendimento Ininterrupto (24/7) e Redução do Tempo de Agendamento: O cliente final ganha autonomia para agendar horários em qualquer dia ou período, inclusive fora do horário comercial, sem depender de respostas imediatas do lojista. Enquanto a negociação assíncrona por mensagens demanda, em média, de 4 a 8 minutos até a efetiva confirmação, o autoatendimento na plataforma reduziu a operação para uma média de 40 segundos, representando uma redução superior a 85% no tempo total da jornada do cliente.
- Eliminação de Interrupções no Atendimento: O profissional elimina a necessidade de interromper o atendimento presencial para negociar horários por telefone ou mensagens, reduzindo distrações e melhorando a qualidade do serviço prestado. Considerando que cada marcação manual consome de 2 a 3 minutos de atenção ativa do trabalhador solo, em uma rotina estimada de 15 atendimentos diários a plataforma proporciona uma economia de 30 a 45 minutos por dia em tarefas administrativas diretas.
- Centralização e Redução de Falhas: Unificação das informações de clientes, serviços, colaboradores e histórico de agendamentos em um ambiente persistente e estruturado, eliminando perdas de dados, retrabalho administrativo e 100% das ocorrências de conflitos de horário (*double booking*) nos cenários testados.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

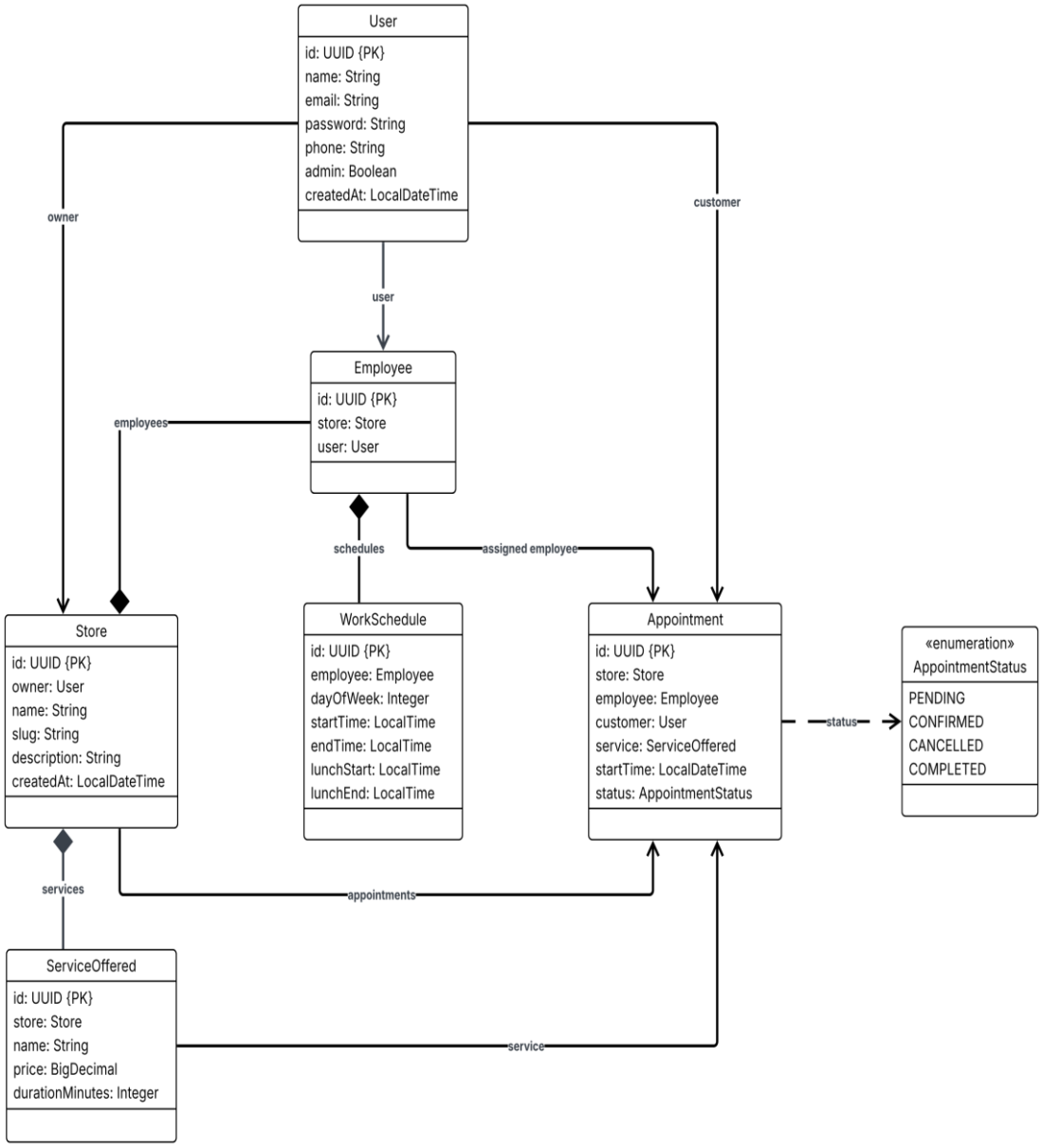
Figura 1. Diagrama de casos de uso



Fonte: Autoria própria, 2026.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 2. Diagrama de classes



Fonte: Autoria própria, 2026.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 3. Classe *AvailabilityService*

```
public class AvailabilityService {

    private final WorkScheduleRepository scheduleRepository;
    private final AppointmentRepository appointmentRepository;

    public List<LocalTime> getAvailableSlots(UUID employeeId, LocalDate date, Integer
serviceDuration) {
        // 1. Identifica o dia da semana (1-7)
        int dayOfWeek = date.getDayOfWeek().getValue();

        // 2. Busca a configuração de trabalho para aquele dia
        var schedule = scheduleRepository.findByEmployeeIdAndDayOfWeek(employeeId, dayOfWeek)
                .orElseThrow(() -> new BusinessException("Funcionário não trabalha neste
dia."));

        // 3. Busca agendamentos ocupados no dia
        LocalDateTime startOfDay = date.atStartOfDay();
        LocalDateTime endOfDay = date.atTime(LocalTime.MAX);
        var busyAppointments =
appointmentRepository.findActiveAppointmentsByEmployeeAndDate(employeeId, startOfDay,
endOfDay);

        // 4. Gera os slots possíveis
        List<LocalTime> availableSlots = new ArrayList<>();
        LocalTime currentSlot = schedule.getStartTime();

        // Enquanto o slot + duração do serviço não ultrapassar o fim do expediente
        while (!currentSlot.plusMinutes(serviceDuration).isAfter(schedule.getEndTime())) {

            final LocalTime slotFinal = currentSlot;

            // Verifica se o slot está no horário de almoço
            boolean isLunch = isTimeInInterval(slotFinal, serviceDuration,
schedule.getLunchStart(), schedule.getLunchEnd());

            // Verifica se o slot bate com algum agendamento existente
            boolean isBusy = busyAppointments.stream().anyMatch(app ->
                    isTimeInInterval(slotFinal, serviceDuration,
app.getStartTime().toLocalTime(), app.getEndTime().toLocalTime())
            );

            if (!isLunch && !isBusy) {
                availableSlots.add(slotFinal);
            }

            // Pula para o próximo slot (define intervalo fixo)
            currentSlot = currentSlot.plusMinutes(30);
        }

        return availableSlots;
    }
}
```

Fonte: Autoria própria, 2026.

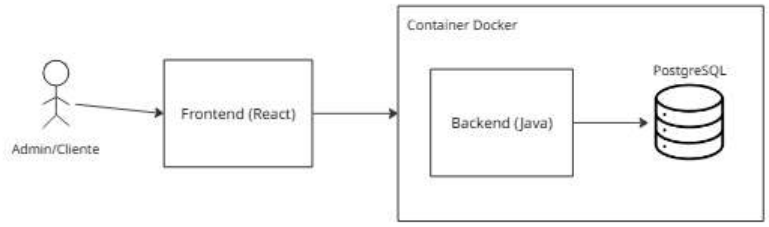
Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 4. Política de acesso às rotas

```
.authorizeHttpRequests(auth -> auth
    .requestMatchers(HttpMethod.POST, "/auth/**").permitAll()
    .requestMatchers(HttpMethod.GET, "/stores/slug/**").permitAll()
    .requestMatchers(HttpMethod.GET, "/stores/*/services").permitAll()
    .requestMatchers(HttpMethod.GET, "/stores/*/employees").permitAll()
    .requestMatchers(HttpMethod.GET, "/stores/*/employees/*/schedule").permitAll()
    .requestMatchers(HttpMethod.GET, "/stores/*/availability").permitAll()
    .anyRequest().authenticated()
)
.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
```

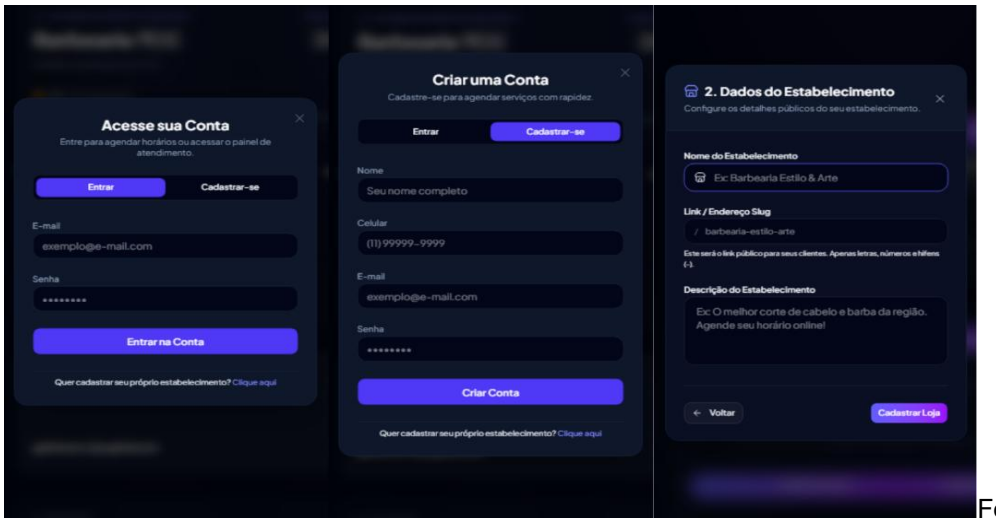
Fonte: Autoria própria, 2026.

Figura 5. Arquitetura geral do sistema



Fonte: Autoria própria, 2026.

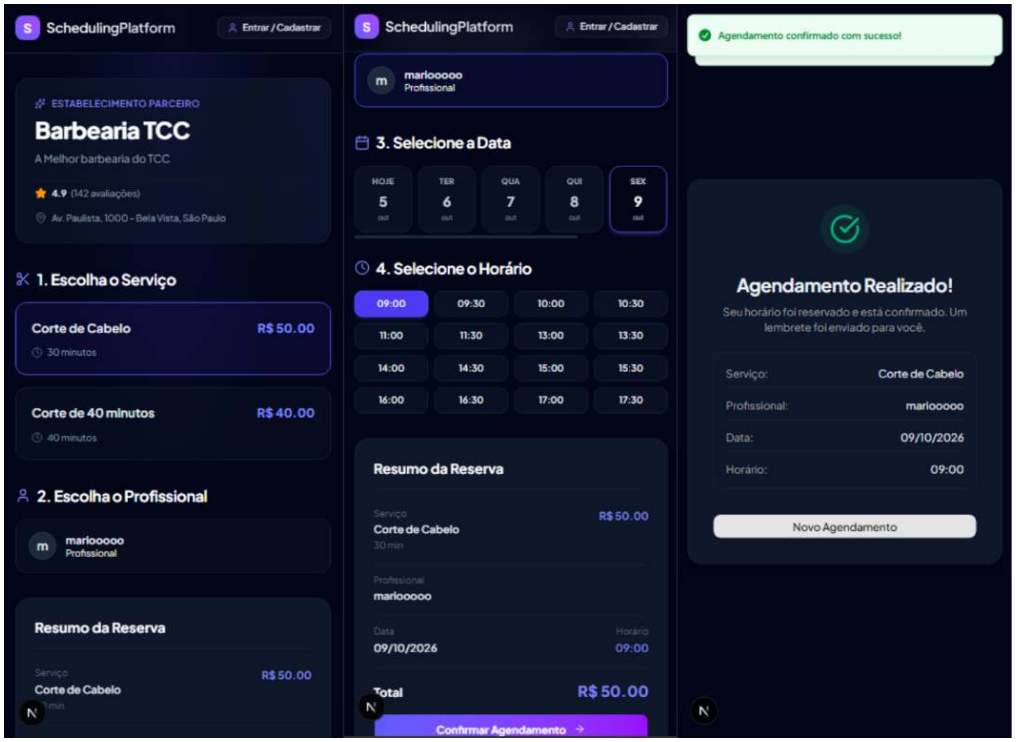
Figura 6. Tela de *login* e cadastro



Fonte: Autoria própria, 2026.

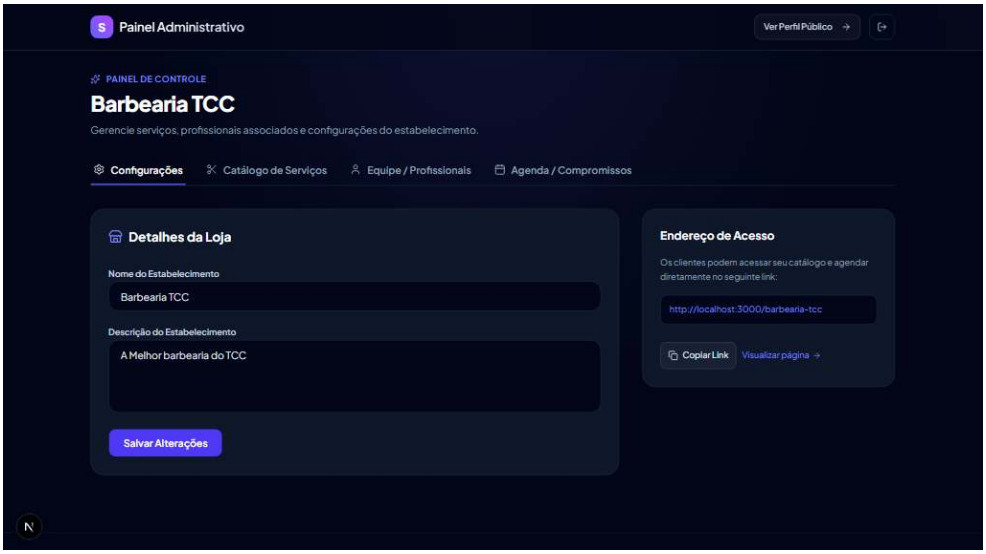
Figura 7. Interface pública de agendamento do estabelecimento

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.



Fonte: Autoria própria, 2026.

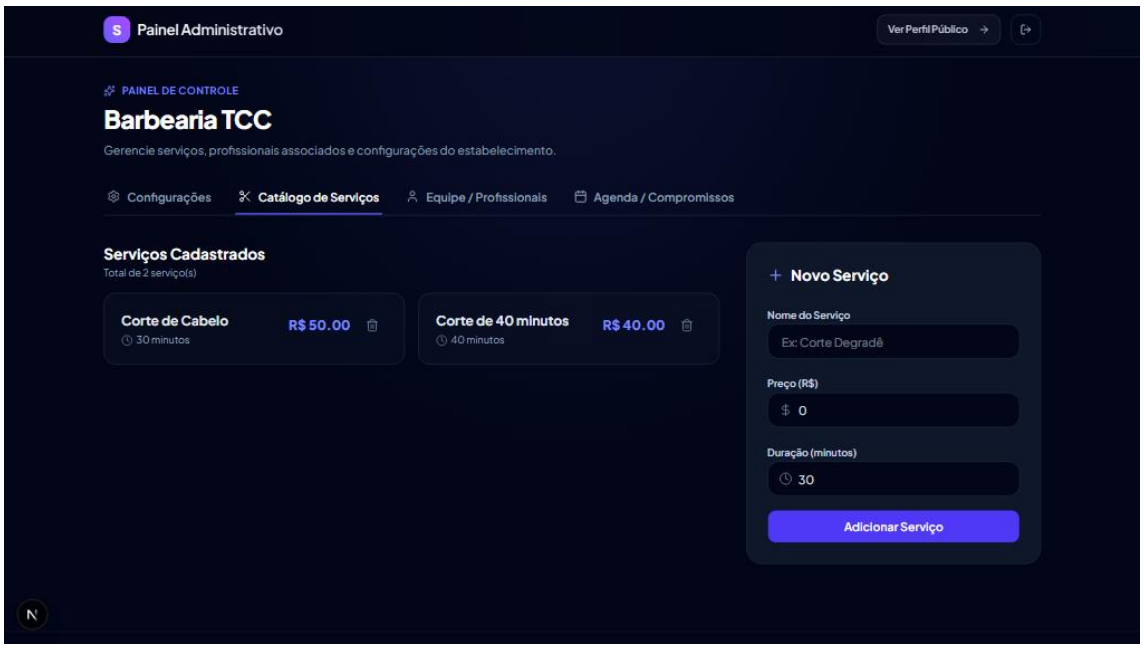
Figura 8. Aba de configurações



Fonte: Autoria própria, 2026.

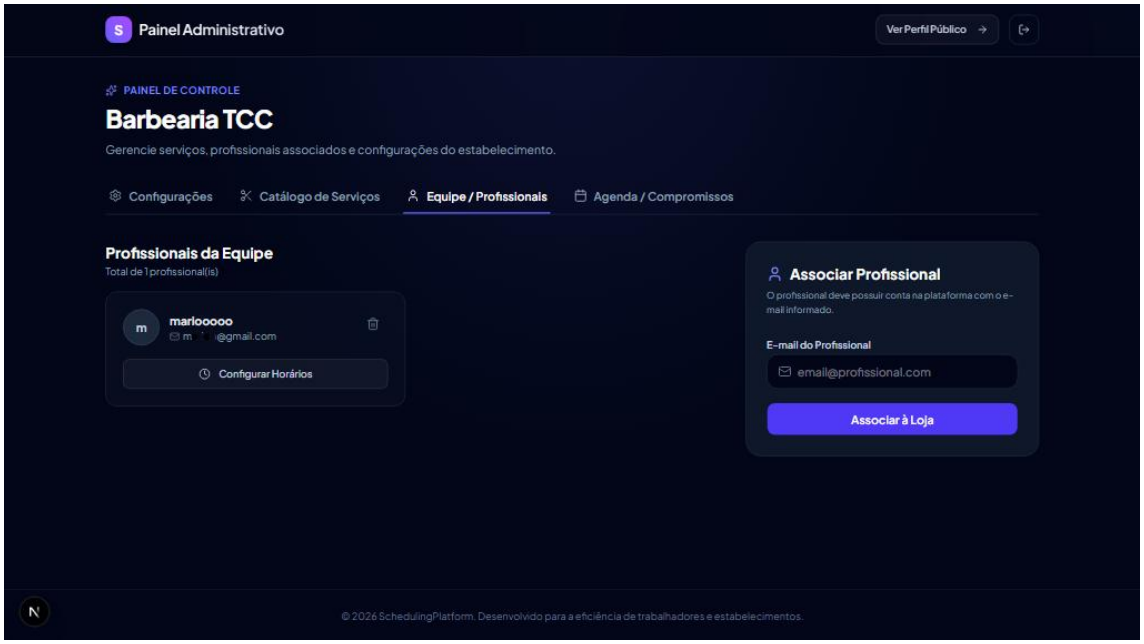
Figura 9. Aba de serviços

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.



Fonte: Autoria própria, 2026.

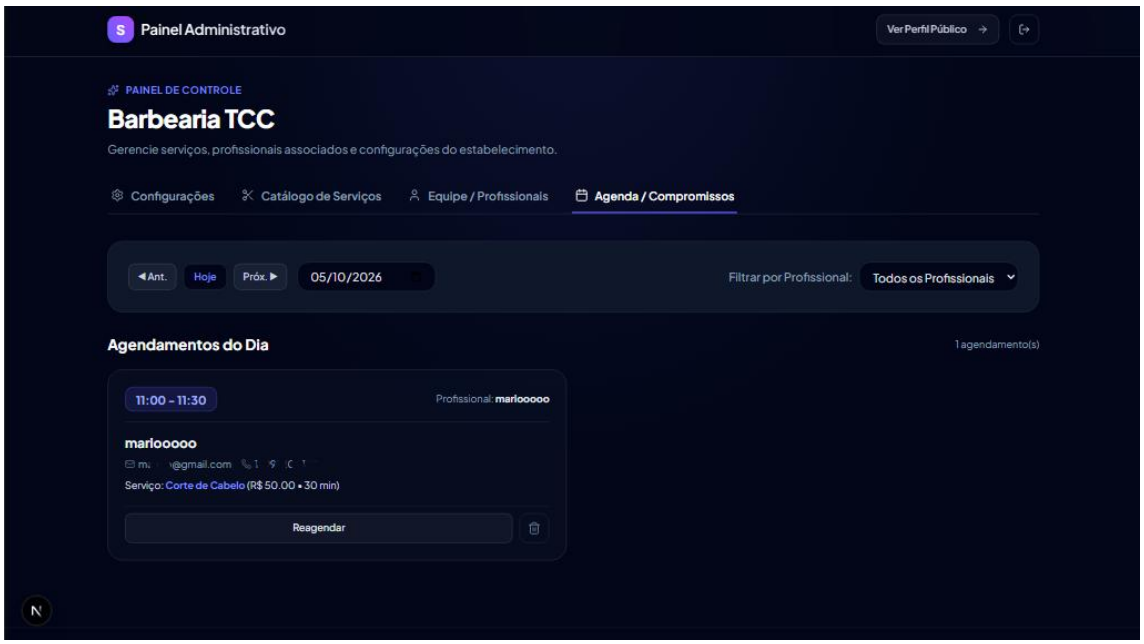
Figura 10. Aba de profissionais e expediente



Fonte: Autoria própria, 2026.

Figura 11. Aba de agendamentos

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.



Fonte: Autoria própria, 2026.

Quadro 1. Matriz de validação dos requisitos do sistema

Identificador	**Descrição do Requisito**	**Módulo / Camada Responsável**	***Status***
RF01	Cadastro de Usuários e Autenticação *JWT*	*AuthController / AuthService / JwtService*	Atendido
RF02	Gerenciamento do Estabelecimento (*CRUD* Loja)	*StoreController / StoreService*	Atendido
RF03	Gerenciamento de Serviços Ofertados	*ServiceOfferedController / ServiceOfferedService*	Atendido
RF04	Gerenciamento de Funcionários e Vínculos	*EmployeeController / EmployeeService*	Atendido
RF05	Configuração de Escala Semanal de Trabalho	*WorkScheduleController / WorkScheduleService*	Atendido
RF06	Cálculo Dinâmico de Horários Disponíveis	*StoreController / AvailabilityService*	Atendido
RF07	Efetivação de Agendamentos e Concorrência	*AppointmentController / AppointmentService*	Atendido

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

RF08	Painel Administrativo e Gestão da Agenda	*StoreAppointmentController / AdminDashboard (Next.js)*	Atendido
RNF01	Criptografia de Senhas com *BCrypt*	*SecurityConfig / PasswordEncoder*	Atendido
RNF02	Arquitetura Modular em Camadas e *DDD*	Domínio, Aplicação, Infraestrutura e *API*	Atendido
RNF03	Persistência Relacional Segura (*PostgreSQL*)	*PostgreSQL* 16 / *Spring Data JPA* / *Flyway*	Atendido
RNF04	Interface Reativa e Responsiva (*Next.js*)	*Next.js* 16 / *React* 19 / *Tailwind CSS* v4	Atendido

Fonte: Autoria própria, 2026.

4. CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo principal projetar, modelar e implementar uma plataforma de agendamento online e gestão operacional direcionada a trabalhadores autônomos e microestabelecimentos de serviços. A pesquisa partiu da necessidade de mitigar a sobrecarga administrativa decorrente do agendamento manual por aplicativos de mensagens, que frequentemente resulta em perda de produtividade e falhas no controle da agenda.

A execução do projeto evidenciou que a aplicação dos princípios do *Domain-Driven Design (DDD)* foi determinante para estruturar um domínio limpo e resiliente. No *backend*, a utilização do *Java* 21 com *Spring Boot* e *PostgreSQL* garantiu consistência transacional e isolamento das regras de negócio, com destaque para a precisão do algoritmo de disponibilidade (*AvailabilityService*) na prevenção de conflitos de horários e *double booking*. No *frontend*, o ecossistema *React* com *Next.js* e *TypeScript* viabilizou uma interface reativa, rápida e com baixa carga cognitiva tanto na jornada pública do cliente quanto no painel administrativo do lojista.

Os resultados obtidos comprovaram o atendimento integral aos requisitos funcionais e não funcionais estabelecidos, entregando uma solução funcional, escalável e de baixo custo de hospedagem. A plataforma consolida-se como uma ferramenta prática e viável para modernizar a rotina operacional do trabalhador solo, permitindo maior foco na prestação de seus serviços.

REFERÊNCIAS

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

AGÊNCIA SEBRAE. **Digitalização recorde**: pequenos negócios no Brasil atingem nível histórico em 2025. Brasília, DF, 2025. Disponível em: https://agenciasebrae.com.br/inovacao-e-tecnologia/digitalizacao-recorde-pequenos-negocios-no-brasil-atingem-nivel-historico-em-2025/. Acesso em: 24 mar. 2026.

CHERNY, Boris. **Programming TypeScript**: making your JavaScript applications scale. Sebastopol: O'Reilly Media, 2019.

DEITEL, Paul; DEITEL, Harvey. **Java**: como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2016.

EVANS, Eric. **Domain-Driven Design**: atacando a complexidade no coração do software. 3. ed. Rio de Janeiro: Alta Books, 2016.

FEDOSEJEV, Artem. **React.js Essentials**: a fast-paced guide to designing and building scalable and maintainable front-end applications with React.js. Birmingham: Packt Publishing, 2015.

NORMAN, Donald A. **O design do dia a dia**. Rio de Janeiro: Rocco, 2024.

SEBRAE. **Tendências para MEIs em 2025**. Brasília, DF, 2025. Disponível em: https://sebrae.com.br/sites/PortalSebrae/mei/jornadamei/tendencias-para-meis-em-2025,3276a196139e6910VgnVCM1000001b00320aRCRD. Acesso em: 3 abr. 2026.

SEBRAE MG. **Negócios digitais 2025**: inteligência e dados para pequenos negócios. Belo Horizonte, 2025. Disponível em: https://inteligencia.sebraemg.com.br/pesquisa/negocios-digitais-2025. Acesso em: 2 abr. 2026.

VERNON, Vaughn. **Implementando Domain-Driven Design**. Rio de Janeiro: Alta Books, 2016.

WALLS, Craig. **Spring em ação**. 5. ed. Rio de Janeiro: Alta Books, 2019.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.