Fixi: Plataforma *Web* Inteligente para Conexão de Usuários e Prestadores de Serviços

Rafaela Inês Jung[1], **Alexandre Perin de Souza**[1], **Robson Costa**[1]

[1]Instituto Federal de Santa Catarina (IFSC)
Rua Heitor Villa Lobos, 225, 88506-400 - Lages - SC - Brasil

{rafaela.i}@aluno.ifsc.edu.br

{robson.costa, alexandre.perin}@ifsc.edu.br

Abstract. *This work presents the development of Fixi, a web platform designed to streamline the connection between clients and service providers in a fast and organized manner. The identified problem was the difficulty users face in finding reliable professionals suited to their needs. The proposed solution consists of an accessible platform that centralizes essential information about service providers and enables service scheduling. Its main differential is the use of artificial intelligence, through the Groq API, which automatically suggests the most appropriate service category based on the user's problem description. The evaluations conducted with users indicated high acceptance, highlighting both the platform's ease of use and the usefulness of its recommendations.*

Resumo. *Este trabalho apresenta o desenvolvimento da plataforma Fixi, uma aplicação web criada para facilitar o encontro entre clientes e prestadores de serviços de forma rápida e organizada. O problema identificado foi a dificuldade dos usuários em encontrar profissionais confiáveis e alinhados às suas necessidades. A solução proposta consiste em uma plataforma acessível que reúne informações relevantes dos prestadores e permite o agendamento de serviços. Seu diferencial é o uso de inteligência artificial, por meio da API Groq, capaz de sugerir automaticamente a categoria mais adequada conforme a descrição do problema. As avaliações realizadas com usuários indicaram alta aceitação e destacaram tanto a facilidade de uso quanto a utilidade das recomendações.*

1. Introdução

O avanço das tecnologias digitais e a ampla disseminação da internet têm transformado a forma como consumidores buscam e contratam serviços. A transformação digital tem alterado profundamente a relação entre empresas e consumidores, oferecendo experiências mais ágeis, personalizadas e convenientes (Dragicevic, 2022; Souza e Ferreira, 2022). O comércio eletrônico no Brasil, por exemplo, movimentou R$ 262 bilhões em 2022, evidenciando sua relevância para a economia nacional (NielsenIQ | Ebit, 2023). Em 2023, esse setor continuou em expansão, alcançando R$ 196 bilhões em faturamento (Ministério do Desenvolvimento, 2024), impulsionado pelo aumento do número de consumidores digitais, que já somam mais de 87 milhões de brasileiros (Edrone, 2024).

Além disso, em 2024, 93,6% dos domicílios no país possuíam acesso à internet, totalizando 74,9 milhões de lares conectados, o que reforça a dimensão e o potencial de alcance desse mercado (IBGE, 2024). Muitas vezes, entretanto, os consumidores ainda

dependem de indicações informais de profissionais ou realizam buscas dispersas em diferentes plataformas, o que resulta em processos ineficientes e demorados.

No Brasil, usuários enfrentam dificuldades ao buscar prestadores de serviços confiáveis, seja pela ausência de informações claras, pela falta de mecanismos de avaliação ou pela dificuldade em encontrar profissionais adequados às suas necessidades. Plataformas já existentes apresentam limitações quanto ao escopo de atuação ou carecem de recursos avançados de recomendação, o que restringe sua eficácia. Diante desse cenário, surge o problema central deste trabalho: como criar uma plataforma web que organize e disponibilize informações de prestadores de serviços, garantindo facilidade de uso e incorporando Inteligência Artificial como apoio à recomendação de profissionais?

Diante desse cenário, soluções baseadas em plataformas *web* apresentam-se como alternativas viáveis para organizar e disponibilizar informações sobre prestadores de serviços. Essas plataformas podem centralizar dados relevantes, como nome, serviços oferecidos, descrições e número de contato, permitindo que o usuário tenha acesso rápido e estruturado às informações necessárias para entrar em contato diretamente com o prestador (Iliadis et al., 2023).

A proposta deste trabalho consiste no desenvolvimento de uma plataforma *web* que integra *front-end* em *React*, *back-end* em *Java* e banco de dados *MySQL*, permitindo que usuários encontrem prestadores de serviços de forma organizada e rápida. A plataforma contará com o suporte de uma Inteligência Artificial (IA) para auxiliar na identificação da categoria de profissional mais adequada a partir da descrição do problema apresentada pelo usuário (Saadatmand et al., 2019). O objetivo geral é desenvolver uma plataforma acessível, que apresente informações relevantes dos prestadores, como nome, serviços oferecidos, possibilidade de agendamento, descrição e contato. Para atingir esse objetivo, foram definidos os seguintes objetivos específicos:

- Desenvolver o *front-end* em *React*, incluindo telas para cadastro de prestadores e envio de descrições de problemas pelos clientes;
- Implementar o *back-end* em *Java*, gerenciando a lógica da plataforma e a integração com o banco de dados *MySQL*;
- Estruturar o banco de dados *MySQL* para armazenar informações de prestadores e categorias de serviços;
- Integrar a IA de apoio para sugerir a categoria de prestador mais adequada a partir da descrição do problema;
- Avaliar a aplicação quanto à usabilidade, clareza da apresentação das informações e eficácia da recomendação de profissionais.

Sob o aspecto metodológico, este trabalho foi dividido em seis etapas. A primeira etapa consistiu em uma pesquisa bibliográfica sobre desenvolvimento de aplicações *web*, bancos de dados e sistemas de recomendação de serviços, utilizando livros, artigos científicos e relatórios técnicos. A segunda etapa envolveu o estudo e a seleção das ferramentas empregadas para o desenvolvimento, definindo-se o uso do *React* para o *front-end*, do Java para o *back-end*, do *MySQL* para o banco de dados e de *APIs* de Inteligência Artificial para a classificação de profissionais.

A terceira etapa correspondeu ao desenvolvimento do *back-end* em Java, com o auxílio do *Maven* para gerenciamento de dependências, e à modelagem e implementação

do banco de dados em *MySQL*. A quarta etapa contemplou a implementação do *front-end* em *React*, com a criação de componentes reutilizáveis e a construção das telas de cadastro de prestadores e de envio de problemas pelos clientes. A quinta etapa foi dedicada à integração da aplicação *web* com as *APIs* de Inteligência Artificial, utilizando técnicas de classificação supervisionada, além da realização de testes funcionais e da avaliação da ferramenta junto a usuários. Por fim, a sexta etapa apresentou as conclusões do estudo, relacionando os resultados obtidos com os objetivos propostos.

A abordagem metodológica adotada caracteriza-se como pesquisa de natureza aplicada, pois propõe uma solução tecnológica concreta para um problema real enfrentado por consumidores: a dificuldade de identificar e localizar prestadores de serviços de maneira eficiente. Quanto à forma de abordagem, trata-se de uma pesquisa qualitativa, que permitirá analisar a percepção dos usuários quanto à funcionalidade, usabilidade e clareza da apresentação das informações da aplicação. Em relação aos objetivos, classifica-se como uma pesquisa exploratória, visto que busca aprofundar conhecimentos sobre desenvolvimento de aplicações *web* integradas a recursos de Inteligência Artificial para recomendação de prestadores. Por fim, quanto aos procedimentos técnicos, trata-se de uma pesquisa bibliográfica e experimental.

Além desta introdução, este trabalho estará estruturado da seguinte forma: a seção dois apresenta o referencial teórico, abordando conceitos fundamentais para o entendimento da pesquisa; a seção três detalha o desenvolvimento da aplicação; a seção quatro descreve a avaliação da plataforma; e, por fim, a seção cinco traz as conclusões do trabalho.

2. Referencial Teórico

Esta seção está dividida em três partes. A primeira parte apresenta conceitos fundamentais sobre desenvolvimento de aplicações *web*. A segunda parte explana sobre bancos de dados. Por fim, a terceira parte apresenta trabalhos relacionados ao uso de IA para suporte à identificação de profissionais, bem como aplicações *web* similares voltadas à organização e disponibilização de informações de prestadores de serviços.

2.1. Desenvolvimento de Aplicações *Web*

As aplicações *web* constituem sistemas de software que possibilitam a interação entre utilizadores e serviços através de navegadores, baseando-se no modelo arquitetural cliente-servidor. Nesse modelo, o cliente, frequentemente desenvolvido com tecnologias de *front-end* como *React*, *Angular* ou *Vue.js*, é responsável pela apresentação da interface gráfica e pela experiência do utilizador. Do lado do servidor, aplicações construídas com linguagens como *Java*, *Python* ou *Node.js* executam a lógica de negócio, o processamento de dados e a comunicação com sistemas de persistência, como bancos de dados (Li e Zhang, 2021).

Essa separação de responsabilidades entre *front-end* e *back-end* promove benefícios significativos em termos de escalabilidade, manutenibilidade e segurança, uma vez que permite a evolução independente de cada camada e a especialização de desenvolvedores (Fielding, 2000). O desenvolvimento *web* contemporâneo adota ainda princípios de modularidade, reutilização de código e boas práticas de engenharia de software, tais como

a definição de *APIs RESTful*, que facilitam a interoperabilidade entre sistemas heterogêneos, e a implementação de mecanismos robustos de tratamento de exceções (Richardson et al., 2013).

Além disso, o uso de *frameworks* e bibliotecas modernas, como *React*, permite a construção de interfaces dinâmicas, componentizadas e responsivas, contribuindo para uma melhor performance e usabilidade (Li e Zhang, 2021). Tais tecnologias são frequentemente integradas em ambientes de desenvolvimento que adotam práticas de Integração e Entrega Contínuas (CI/CD), testes automatizados e conteinerização, reforçando a qualidade e a confiabilidade das aplicações (Shahin et al., 2017).

2.2. Bancos de Dados e Gerenciamento de Informações

O armazenamento eficiente de informações é fundamental em aplicações que conectam usuários a prestadores de serviços. Bancos de dados relacionais, como o *MySQL*, possibilitam a organização de dados em tabelas estruturadas, com relações claras entre usuários, prestadores e categorias de serviços (Connolly e Begg, 2021). Uma boa modelagem inclui atributos como nome do prestador, serviços oferecidos, descrição detalhada e informações de contato, permitindo consultas rápidas e consistentes.

Além disso, práticas de normalização e criação de índices são essenciais para melhorar o desempenho da aplicação e evitar redundâncias. Em sistemas que podem ter grande volume de dados, essas técnicas garantem que o acesso às informações seja ágil e confiável, favorecendo a experiência do usuário e a escalabilidade do sistema (Elmasri e Navathe, 2020).

2.3. Inteligência Artificial para Recomendação de Serviços

A Inteligência Artificial (IA) é uma área da ciência da computação que busca desenvolver sistemas capazes de realizar tarefas que tradicionalmente exigiriam inteligência humana, como classificar, aprender padrões e apoiar na tomada de decisões (Russell e Norvig, 2021). Em aplicações *web* voltadas à conexão de usuários com prestadores de serviços, a IA pode ser utilizada para analisar as descrições dos problemas fornecidas pelos usuários e recomendar a categoria de profissional mais adequada.

Algoritmos de recomendação podem variar desde regras simples baseadas em palavras-chave até modelos de aprendizado de máquina que analisam o histórico de interações, padrões de serviço e comportamento do usuário (Ricci et al., 2015). Essa tecnologia não substitui a decisão final do usuário, mas atua como suporte para agilizar o processo de identificação do prestador correto, aumentando a eficiência e a precisão da plataforma (Roy e Dutta, 2022).

2.4. Trabalhos Similares

Nesta seção, são descritos os trabalhos mais relevantes que apresentam objetivos semelhantes aos deste estudo. A pesquisa foi realizada nas plataformas *Google*, *Google Scholar* e *Semantic Scholar*, utilizando os seguintes termos de busca: "Aplicações *Web*", "Plataformas de Conexão de Serviços", "Recomendação de Profissionais", "Inteligência Artificial para Recomendação"e "Sistemas de Apoio ao Usuário".

Com base nos resultados obtidos, foram selecionados cinco trabalhos considerados análogos, segundo os seguintes critérios: compatibilidade com o tema, atualidade,

foco na recomendação de serviços e contribuição prática para usuários e prestadores. Para cada trabalho selecionado, o mesmo foi lido e analisado em detalhes, considerando aspectos técnicos, usabilidade e a integração de IA para suporte à decisão.

Silva (2025) apresenta a plataforma ConecteVidas, voltada para conectar profissionais de saúde a pacientes que necessitam de atenção domiciliar. O objetivo é otimizar o acesso e a eficiência dos serviços, promovendo interações mais humanizadas. A metodologia incluiu questionários com 80 participantes e testes de usabilidade com 10 usuários, que avaliaram a aplicação de forma altamente positiva (nota média 4,6/5). O ponto forte do trabalho está na abordagem prática, unindo requisitos levantados junto ao público-alvo com um protótipo funcional validado em testes. Entretanto, as limitações geográficas e o ambiente controlado de avaliação restringem a generalização dos resultados, além de não haver uma implementação em larga escala em cenários reais de saúde.

Souza et al. (2021) propõe o Conecta *Jobs*, um *e-marketplace* que aproxima prestadores de serviços e clientes por meio de uma plataforma *web*. A ferramenta permite busca por filtros, contratação e avaliação de profissionais, funcionando também como estratégia de marketing pessoal. O sistema foi desenvolvido em *PHP* com *Bootstrap* e *MySQL*, utilizando *UML* para modelagem, e passou por testes de usabilidade que demonstraram boa aceitação. O trabalho contribui ao explorar a democratização da tecnologia como apoio a trabalhadores autônomos. No entanto, limita-se a um protótipo em fase beta, com validação em público reduzido, o que restringe a análise de sua escalabilidade e impacto no mercado real.

Silva (2016) desenvolve uma rede social voltada para reparos residenciais, permitindo que prestadores de serviços anunciem e que usuários avaliem com notas e comentários, criando um histórico de reputação. O projeto utilizou métodos ágeis (*Extreme Programming e Kanban*) e aplicou modelagem *UML* para estruturar o sistema. O ponto forte é a proposta inovadora de combinar redes sociais com um sistema de reputação, oferecendo mais confiança na contratação. Contudo, a limitação está no escopo restrito a reparos residenciais e na ausência de testes em maior escala, o que impede a validação de sua aceitação e desempenho em ambientes reais.

Oliveira et al. (2023) apresenta uma plataforma *web* para contratação de serviços com foco no cenário pós-pandemia, visando apoiar pequenas e médias empresas na divulgação e atração de clientes. O sistema foi desenvolvido com *Angular* 16 (*front-end*), *.NET* 7 em *C#* (*back-end*) e banco de dados *SQL Server*, incluindo funcionalidades como geolocalização, *chat* em tempo real (*SignalR*), integração de pagamentos (Mercado Pago), autenticação social (*Google e Facebook*) e relatórios gráficos (*Chart.js*). O trabalho se destaca pela robustez tecnológica e diversidade de recursos implementados. Entretanto, não há evidências de validação empírica junto a usuários reais, o que limita a avaliação de sua usabilidade e impacto mercadológico.

Silva (2021) propõe um sistema para contratação de profissionais de TI, abrangendo modalidades de CLT, estágio e *freelancer*. A plataforma busca suprir a carência de mão de obra na área de tecnologia, oferecendo funcionalidades para cadastro de contratantes e colaboradores, criação de projetos e avaliação de desempenho. O desenvolvimento utilizou *HTML*, *CSS*, *JavaScript*, *PHP*, *MySQL* e *Bootstrap*, além de diagramas *UML* para estruturar casos de uso e interações. O diferencial do trabalho é a especialização no

setor de TI, que carece de soluções direcionadas. Todavia, o estudo se restringe a uma proposta inicial com prototipagem, sem testes de larga escala para verificar sua aceitação por profissionais e empresas.

Quadro 1. Trabalhos similares.

Autor(es)	IA	Plataforma	Objetivos	Tecnologias Utilizadas
Silva (2025)	Não	ConecteVidas: Plataforma para conectar profissionais de saúde a pacientes	Melhorar acesso e eficiência no atendimento domiciliar em saúde	*Strapi* (*back-end*), arquitetura *MVC*, questionários e testes de usabilidade
Souza et al. (2021)	Não	Conecta *Jobs*: *E-marketplace* para prestadores de serviços	Intermediar prestadores de serviços e clientes com avaliações e filtros	*PHP*, *MySQL*, *Bootstrap*, *UML*
Silva (2016)	Não	Rede Social para reparos residenciais	Criar rede social de anúncios e avaliações de serviços de reparos	*Kanban*, *Extreme Programming (XP)*, *UML*
Oliveira et al. (2023)	Não	Plataforma *web* para contratação de serviços (pós-pandemia)	Apoiar pequenos e médios empresários na divulgação e contratação de serviços	*Angular 16*, *.NET 7 (C#)*, *SQL Server*, *SignalR*, Mercado Pago, *Chart.js*
Silva (2021)	Não	Sistema para contratação de profissionais de TI	Suprir carência de mão de obra em TI, oferecendo opções CLT, estágio e freelancer	*HTML*, *CSS*, *JavaScript*, *PHP*, *MySQL*, *Bootstrap*, *UML*
TCC - Proposta	Sim	Plataforma *web* para conectar prestadores de serviços a clientes	Facilitar a busca, contratação e avaliação de prestadores em diversas áreas	*React* (*front-end*), *Java* (*back-end*), *MySQL* (banco de dados)

A análise crítica dessas iniciativas vistas no Quadro 1 demonstra que, embora relevantes, elas apresentam limitações quanto à diversidade de áreas atendidas e validação empírica. Nesse contexto, a proposta do presente trabalho busca preencher essa lacuna ao adotar tecnologias modernas e integrar um módulo de Inteligência Artificial para recomendação de prestadores, unindo aplicabilidade prática, foco em experiência do usuário e potencial de expansão para múltiplos segmentos de serviços (Vicente e Burnay, 2024).

3. Desenvolvimento

Nesta seção, são descritas, de forma estruturada e detalhada, as etapas empreendidas para o desenvolvimento da plataforma *web* de conexão entre usuários e prestadores de serviços. As subseções apresentam a abordagem metodológica adotada, os requisitos funcionais e não funcionais da plataforma, a modelagem da arquitetura da aplicação, os protótipos da interface de usuário, bem como a organização do *software* e o conjunto de tecnologias empregadas ao longo do processo de implementação, incluindo *front-end* em *React*, *back-end* em *Java*, banco de dados *MySQL* e suporte de Inteligência Artificial (IA) para recomendação de profissionais.

A Subseção 3.1 apresenta a metodologia de desenvolvimento adotada e suas adaptações ao contexto do projeto. A Subseção 3.2 descreve o levantamento e a especificação dos requisitos de *software*. Na Subseção 3.3, são expostos os aspectos relacionados à

arquitetura da plataforma, incluindo sua estrutura em camadas e os fluxos de processamento. A Subseção 3.4 aborda a modelagem do banco de dados, destacando as entidades e relacionamentos definidos. Em seguida, a Subseção 3.5 apresenta os protótipos de interface elaborados com base nos princípios de usabilidade e design centrado no usuário. A Subseção 3.6 detalha as principais tecnologias e ferramentas utilizadas durante a implementação. Por fim, a Subseção 3.7 discute a aplicação dos recursos de Inteligência Artificial integrados à plataforma, especialmente no suporte à recomendação e avaliação automatizada dos prestadores de serviços.

3.1. Metodologia de Desenvolvimento

Durante o desenvolvimento deste trabalho, adotou-se uma adaptação da metodologia ágil *Scrum* (Figura 1), ajustada para o contexto em que apenas uma desenvolvedora esteve envolvida em todas as etapas do projeto. Essa escolha possibilitou maior flexibilidade e entregas incrementais, com revisões contínuas ao longo do processo (Schwaber e Sutherland, 2020).

O desenvolvimento foi organizado em *sprints* curtos, nos quais eram priorizadas tarefas específicas a partir de um *backlog* previamente definido. No início de cada ciclo, eram estabelecidas as funcionalidades a serem implementadas, como o cadastro de usuários, o gerenciamento de prestadores, os agendamentos e a recomendação via Inteligência Artificial. A cada entrega parcial, realizava-se uma revisão individual equivalente a uma *Sprint Review*, com o objetivo de validar o funcionamento do incremento produzido. Além disso, foram registradas reflexões ao final de cada *sprint*, similares a uma *Retrospective*, que permitiram identificar dificuldades e propor melhorias para os ciclos seguintes.

A utilização desse modelo adaptado trouxe como principais vantagens a flexibilidade para incorporar mudanças de requisitos e a entrega contínua de valor ao projeto, permitindo avaliar a evolução da plataforma de forma iterativa. Como limitação, destacou-se a ausência de uma equipe completa, o que concentrou todas as responsabilidades na desenvolvedora, reduzindo as interações colaborativas típicas do *Scrum*. Ainda assim, a metodologia mostrou-se eficaz para manter a organização do trabalho e assegurar entregas incrementais alinhadas aos objetivos do estudo.

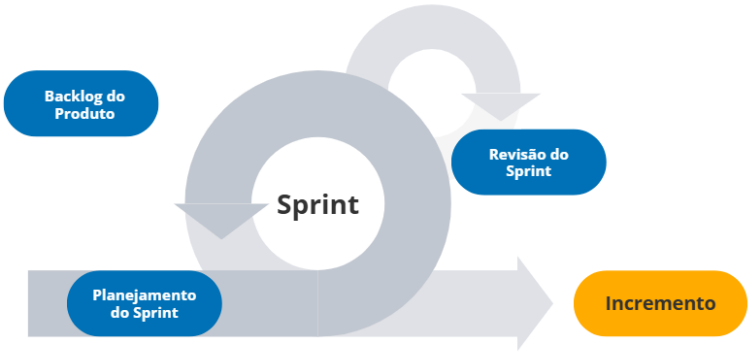

Figura 1. Metodologia *Scrum* adaptado para um desenvolvedor — etapas do processo de desenvolvimento.

3.2. Requisitos

A definição dos requisitos desta plataforma foi realizada a partir da combinação de diferentes técnicas de elicitação. Inicialmente, foram analisados trabalhos acadêmicos e relatórios técnicos relacionados a sistemas de recomendação e plataformas de serviços. Além disso, foram conduzidas conversas exploratórias e observações de experiências de usuários em plataformas semelhantes, o que possibilitou identificar expectativas, problemas recorrentes e funcionalidades desejadas. Dessa forma, os requisitos aqui apresentados refletem tanto boas práticas documentadas na literatura quanto necessidades reais identificadas no contexto do projeto.

Os requisitos são tradicionalmente classificados em duas categorias principais: funcionais e não funcionais (Wiegers e Beatty, 2013). Requisitos funcionais descrevem "declarações de funções ou serviços que o sistema deve fornecer"(Sommerville, 2011), enquanto requisitos não funcionais representam "restrições às funções ou serviços oferecidos pelo sistema"(Sommerville, 2011).

3.2.1. Requisitos Funcionais

Os requisitos funcionais descrevem as funcionalidades essenciais que a plataforma deve oferecer para cumprir seus objetivos:

- RF01 - Cadastro de prestadores de serviços: A plataforma deve permitir que prestadores registrem seu nome, serviços oferecidos e número de contato;
- RF02 - Visualização de prestadores pelos usuários: O usuário deve conseguir visualizar uma lista de prestadores de serviços cadastrados, com todas as informações relevantes;
- RF03 - Envio de descrição de problema pelo usuário: O usuário poderá enviar uma descrição breve do problema que deseja resolver para a Inteligência Artificial integrada;
- RF04 - Recomendação de categoria de prestador via IA: A plataforma deve sugerir automaticamente a categoria de prestador mais adequada para o problema descrito, utilizando um modelo de Inteligência Artificial;
- RF05 - Consulta de prestadores por categoria: O usuário poderá filtrar prestadores por categoria recomendada pela IA ou por interesse próprio;
- RF06 - Agendamento: A plataforma deve permitir o agendamento do serviço;
- RF07 - Informações: Ao solicitar um agendamento, o usuário poderá enviar informações como uma breve descrição do seu problema para informar o prestador de serviço, assim como poderá enviar (de forma opcional) o valor que está disposto a pagar;
- RF08 - Segurança no cadastro: A plataforma deve validar campos obrigatórios no cadastro de prestadores e evitar entradas inválidas ou vazias;
- RF09 - Avaliação pelo Cliente: O cliente poderá realizar avaliações do serviço prestado, incluindo uma nota de 0 a 5 e um breve texto explicativo;
- RF10 - Avaliação pelo Prestador: O prestador poderá avaliar o usuário, incluindo uma nota de 0 a 5 e um breve texto explicativo;
- RF11 - Avaliação pela Plataforma: A plataforma será responsável por avaliar os prestadores de serviço cadastrados seguindo alguns critérios (Tempo de Plataforma, Taxa de Aceitação, Taxa de Cancelamento e Avaliação por IA).

3.2.2. Requisitos Não Funcionais

Os requisitos não funcionais estabelecem critérios de qualidade, desempenho e restrições técnicas para a plataforma:

- RNF01 - Usabilidade: A interface deve ser intuitiva, com botões, menus e campos claramente identificáveis, proporcionando fácil navegação para todos os usuários;
- RNF02 - Desempenho: A aplicação deve apresentar respostas rápidas, garantindo que os usuários visualizem a lista de prestadores sem atrasos perceptíveis;
- RNF03 - Escalabilidade: A arquitetura deve permitir inclusão de novos prestadores e expansão de categorias sem grandes alterações na aplicação;
- RNF04 - Portabilidade: A plataforma deve funcionar em navegadores modernos (*Chrome*, *Firefox*, *Edge*) sem necessidade de *plugins* adicionais;
- RNF05 - Manutenibilidade: O código deve ser organizado em camadas (*front-end*, *back-end* e banco de dados), facilitando futuras alterações e atualizações da plataforma.

A elaboração dos requisitos desta plataforma foi conduzida em conformidade com as melhores práticas estabelecidas pela ISO/IEC/IEEE 29148 (2018), que recomenda que "a especificação de requisitos do sistema (*SyRS*) deve incluir, entre outros itens: introdução, descrição geral, requisitos funcionais, requisitos não funcionais, interfaces externas, restrições de projeto e critérios de aceitação".

Ao adotar essa abordagem estruturada, assegurou-se que as funcionalidades essenciais, as restrições técnicas e as expectativas dos usuários estivessem claramente documentadas, promovendo a rastreabilidade e a qualidade do produto final. Assim, os requisitos aqui definidos constituem a base para as próximas etapas de arquitetura, modelagem do banco de dados, implementação do *front-end* e *back-end*, e integração da Inteligência Artificial para recomendação de prestadores de serviços, alinhando o desenvolvimento do projeto aos padrões internacionais de engenharia de *software* e às necessidades reais dos usuários em termos de organização, acessibilidade e eficiência na identificação de profissionais.

3.2.3. Diagrama de Casos de Uso

O diagrama de casos de uso apresenta a interação entre os atores (usuário e prestador) e os principais casos de uso da plataforma *Fixi* (Figura 2). Observa-se que a Inteligência Artificial (IA) está disponível exclusivamente para o usuário, auxiliando na recomendação da categoria de prestador mais adequada. No diagrama, cada ator possui acesso apenas às funcionalidades que lhe são pertinentes:

- **Usuário**: realiza cadastro, busca prestadores, filtra resultados, consulta perfis, solicita agendamentos, avalia prestadores e utiliza a IA para recomendação de categorias.
- **Prestador**: cadastra-se, atualiza seu perfil, gerencia solicitações de agendamento (aceitar ou recusar), avalia clientes e mantém disponibilidade para contato.

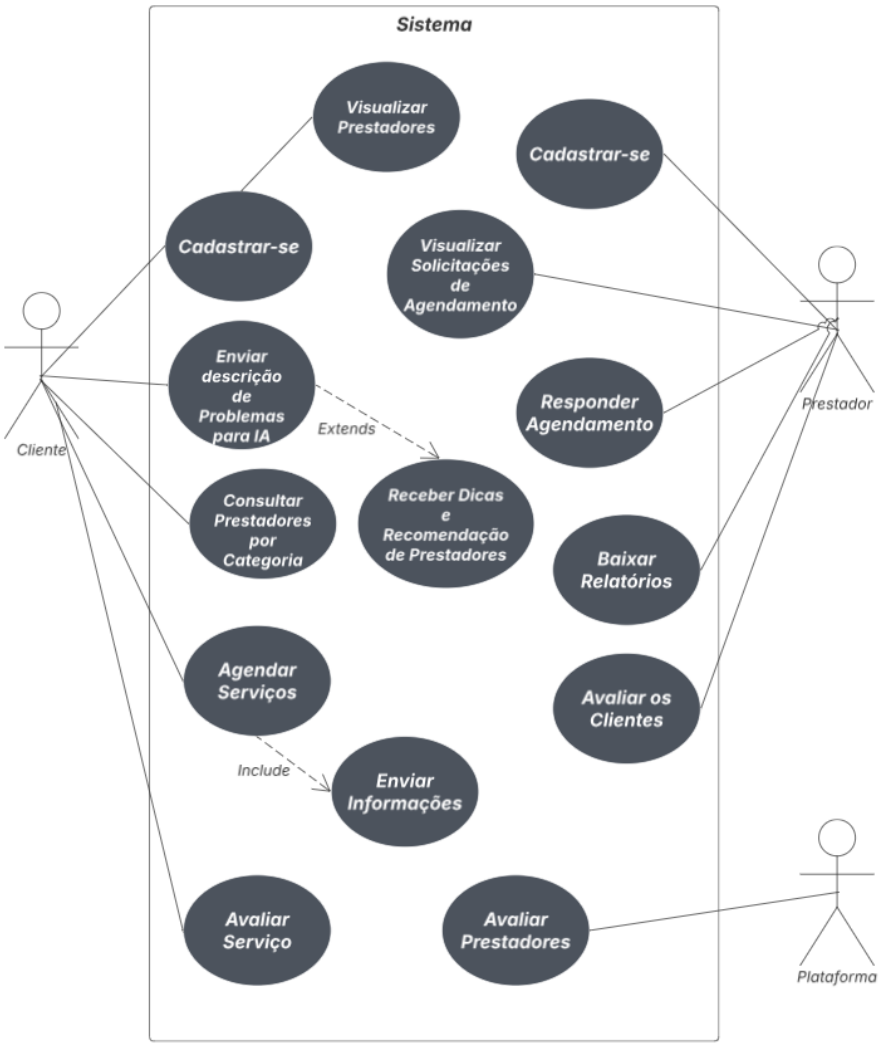

Figura 2. Diagrama de casos de uso da aplicação *Fixi*.

3.3. Arquitetura

A arquitetura da plataforma *web Fixi* foi projetada para garantir modularidade, escalabilidade e desempenho, além de demonstrar a integração com recursos de Inteligência Artificial (IA). A plataforma conecta usuários e prestadores de serviços de forma organizada, adotando uma abordagem em camadas.

3.3.1. Visão Geral e Estilo Arquitetural

A plataforma adota uma arquitetura em três camadas: apresentação (*front-end*), processamento (*back-end*) e armazenamento de dados. Essa organização promove a separação de responsabilidades, facilita a manutenção, amplia a escalabilidade e permite a evolução modular. A comunicação entre as camadas é realizada via *APIs REST*, com dados no formato *JSON*, assegurando interoperabilidade e integração com outras plataformas.

3.3.2. Arquitetura, Tecnologias e Fluxos da Plataforma

A plataforma *Fixi* foi estruturada com base em tecnologias consolidadas, escolhidas por sua robustez, escalabilidade e aderência às boas práticas de Engenharia de Software:

- *Front-end (React + MUI Material)*: Responsável pela interface interativa e responsiva, construída com base em componentes reutilizáveis, assegurando consistência visual e boa experiência do usuário;
- *Back-end (Java com Spring Boot e JPA/Hibernate)*: Centraliza a lógica da plataforma, processa requisições do *front-end*, integra-se ao banco de dados e ao módulo de IA, além de gerenciar o fluxo de agendamentos;
- Banco de Dados *(MySQL)*: Armazena informações sobre usuários, prestadores, categorias de serviços, agendamentos e avaliações, assegurando integridade e consistência por meio de chaves primárias e estrangeiras;
- *Módulo de IA*: Acessado pelo *back-end* via *API REST*, recebe como entrada a descrição textual fornecida pelo usuário e retorna a categoria de prestador mais adequada, apoiando a tomada de decisão;
- *APIs REST + Swagger*: Padronizam a comunicação entre os módulos e são documentadas de forma automática, facilitando a integração e a manutenção futura;
- *Segurança (JWT + Criptografia)*: Contempla autenticação baseada em *JSON Web Tokens*, criptografia de senhas e conformidade com a *LGPD*, garantindo proteção de dados sensíveis e privacidade de usuários e prestadores.

O funcionamento inicia-se com a interação do usuário no *front-end*, que envia dados ao *back-end*. Este processa as informações, consulta o banco de dados e, quando necessário, aciona o módulo de IA. O resultado consolidado retorna ao usuário em formato *JSON*, de forma clara e organizada. A Figura 3 ilustra esse fluxo de funcionamento.

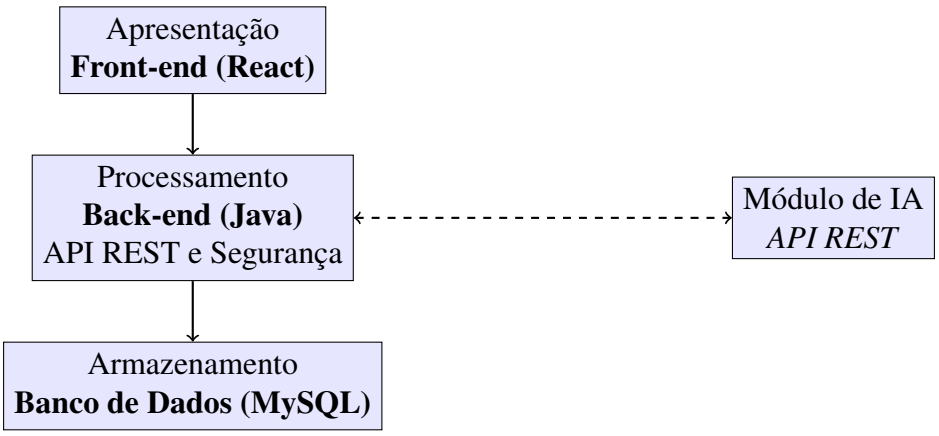

Figura 3. Arquitetura em três camadas da plataforma *Fixi*.

3.3.3. Diagramas de Atividade

Para ilustrar os principais fluxos da plataforma *Fixi*, foram elaborados dois diagramas de atividade. O primeiro descreve o processo de cadastro e busca de prestadores com apoio da Inteligência Artificial, enquanto o segundo representa o processo de agendamento de serviços.

O diagrama de atividade apresentado na Figura 4 mostra como o usuário descreve o problema e recebe da plataforma uma lista de prestadores recomendados pela IA. Já o diagrama da Figura 5 apresenta o fluxo de agendamento, no qual o prestador pode aceitar ou recusar a solicitação, e o usuário é informado sobre o status final do serviço.

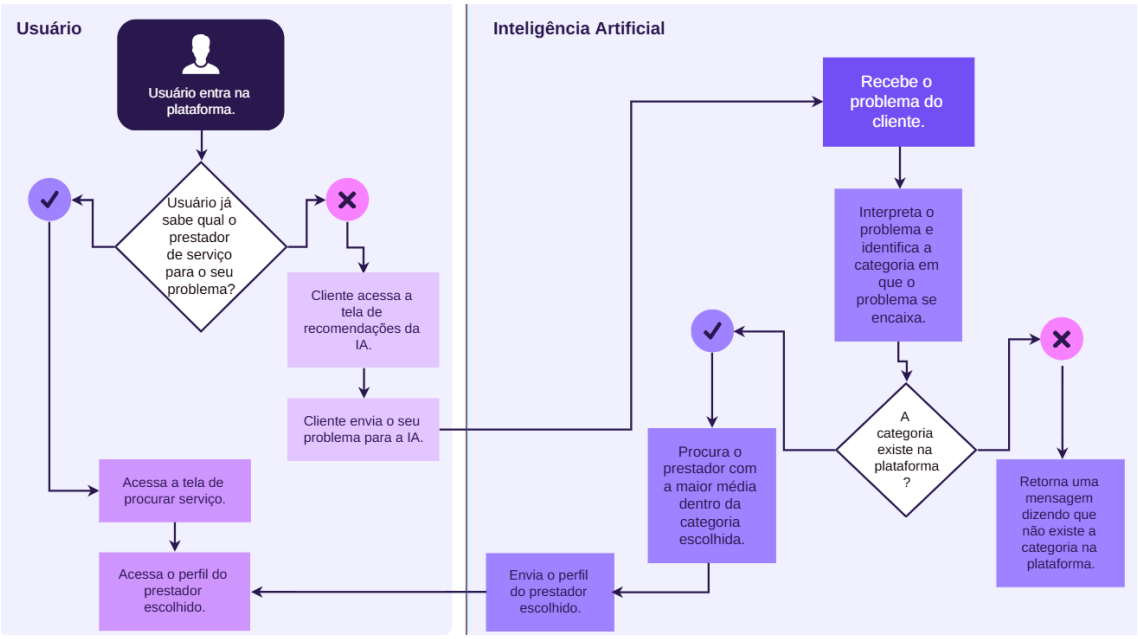

Figura 4. Diagrama de atividade que exemplifica a utilização da IA pelo cliente para receber recomendação de prestador de serviço.

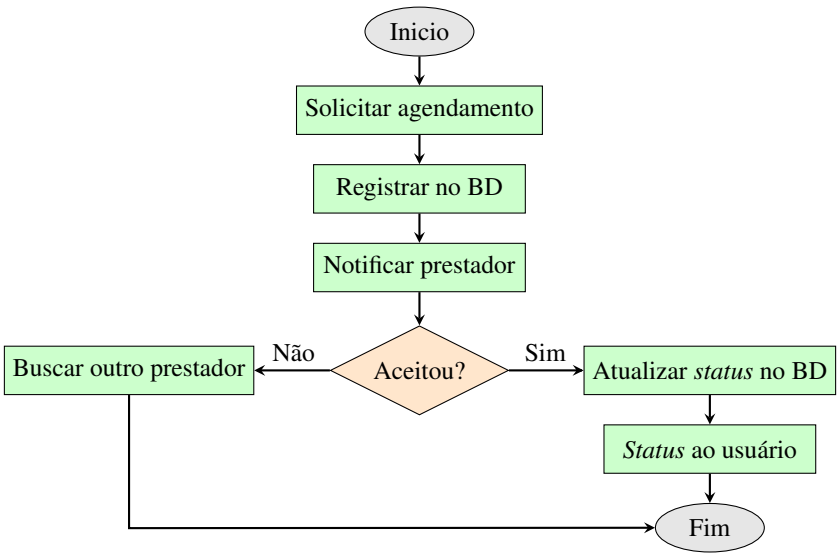

Figura 5. Diagrama de atividade do processo de agendamento de serviços.

Com esses diagramas, o funcionamento da plataforma é representado de forma mais clara e estruturada, permitindo visualizar como os atores interagem com a plataforma desde o cadastro até a finalização do agendamento.

3.4. Modelagem de Banco de Dados

O banco de dados constitui um dos pilares fundamentais da plataforma *Fixi*, pois é responsável pelo armazenamento, organização e recuperação das informações referentes a clientes, prestadores de serviços, categorias, agendamentos, conversas, mensagens e avaliações. Como destacado por Watt e Eng (2015), "as bases de dados relacionais proporcionam uma forma estruturada e eficiente de armazenar, recuperar e gerir grandes volumes de dados". Para garantir eficiência, integridade e consistência, foi adotada uma modelagem relacional que estabelece vínculos claros entre as entidades da plataforma (Figura 6).

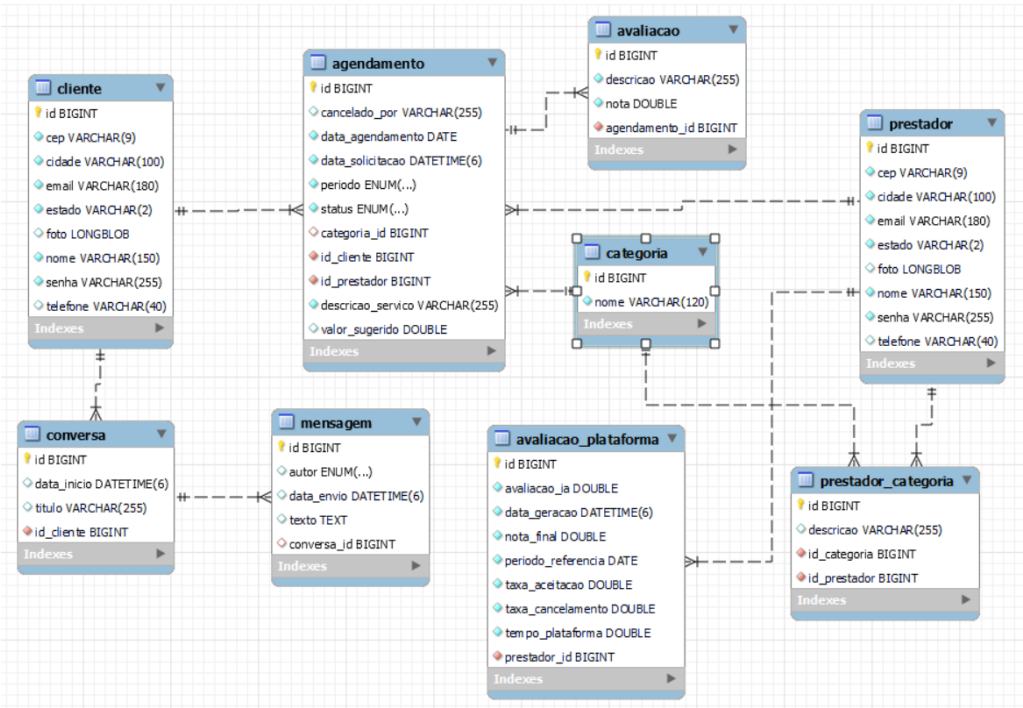

Figura 6. Modelagem do Banco de Dados da plataforma *Fixi*.

As principais entidades incluem Cliente, que representa os usuários da plataforma; Prestador, referente aos profissionais que oferecem serviços; Categoria, que classifica os tipos de serviços disponíveis; e Agendamento, que registra as solicitações realizadas pelos clientes. A tabela Prestador_Categoria foi implementada como uma associação N:N, permitindo que um prestador esteja vinculado a múltiplas categorias e, simultaneamente, que cada categoria seja atendida por diversos prestadores.

Além disso, para possibilitar interações dentro da plataforma, foram incluídas as tabelas Conversa e Mensagem, que armazenam os diálogos entre clientes e prestadores. No que se refere à qualidade e desempenho dos prestadores, a plataforma dispõe de duas formas de avaliação: a tabela Avaliação, vinculada a cada agendamento e composta por descrições e notas fornecidas pelos clientes, e a tabela Avaliação_Plataforma, que concentra as métricas calculadas automaticamente pela plataforma. Esta última contempla critérios como tempo de plataforma, taxa de aceitação, taxa de cancelamento e avaliação de comentários pela inteligência artificial, todos ponderados por seus respectivos pesos para gerar uma nota final mensal de desempenho.

A estrutura relacional escolhida garante integridade referencial por meio do uso de chaves primárias e estrangeiras, além de otimizar consultas essenciais, como a busca de prestadores por categoria, disponibilidade ou desempenho. Essa organização fortalece a escalabilidade do *Fixi* e assegura a manutenção de dados consistentes, permitindo a geração de relatórios de evolução e rankings de prestadores ao longo do tempo.

3.5. Interface Gráfica

Durante o desenvolvimento deste projeto, foram elaboradas interfaces que priorizam a clareza visual, a organização das informações e a simplicidade na interação com o usuário. As telas foram projetadas com base nos princípios do design centrado no usuário, visando proporcionar uma experiência intuitiva, agradável e funcional. Ressalta-se que os nomes, imagens e demais informações apresentadas nas interfaces são fictícios, sendo utilizados exclusivamente para fins de demonstração e ilustração do sistema.

Nesta subseção, são apresentadas as principais interfaces gráficas finais da plataforma, resultantes do processo iterativo de concepção e aprimoramento da interface. Essas telas representam o fluxo essencial de uso da plataforma, evidenciando as funcionalidades centrais que possibilitam a interação entre cliente e prestador.

As demais interfaces, de caráter complementar — relacionadas a funcionalidades de suporte e acompanhamento, como histórico de agendamentos, gestão de solicitações e relatórios — podem ser consultadas no apêndice A.2.

3.5.1. Interface Gráfica 1

A primeira interface (Figura 7) demonstra a tela inicial do usuário com a plataforma *Fixi*. Nela, é exibida uma breve descrição do que é a plataforma.

Figura 7. Exemplo da interface inicial da plataforma. É a primeira tela em que o cliente/prestador tem contato.

3.5.2. Interface Gráfica 2

A interface gráfica apresentada na Figura 8 corresponde à página inicial do cliente, na qual é possível visualizar os agendamentos realizados. Na barra superior de navegação, o

usuário pode acessar as demais funcionalidades da plataforma.

Essa interface serviu como modelo de referência para o desenvolvimento das demais telas, mantendo consistência visual e padronização de elementos. A tela inicial do prestador de serviços segue o mesmo estilo e estrutura, garantindo uniformidade na experiência do usuário em ambos os perfis.

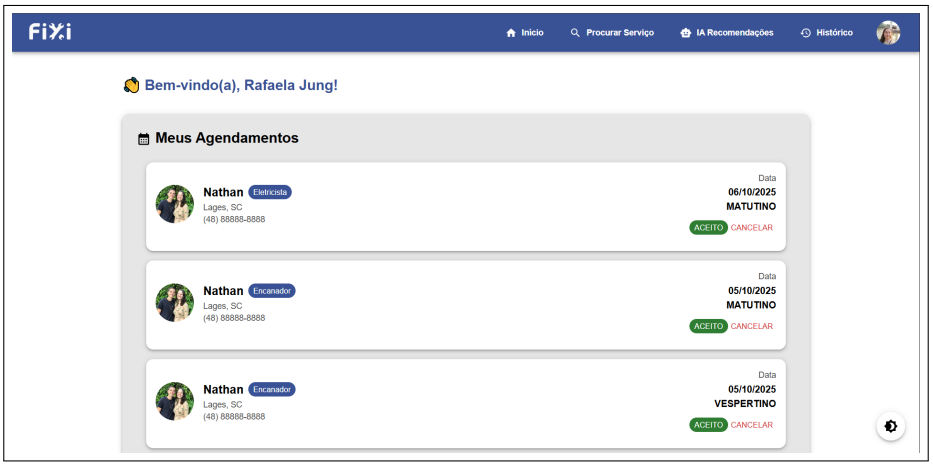

Figura 8. Interface gráfica inicial do cliente após logar na plataforma.

3.5.3. Interface Gráfica 3

A interface apresentada na Figura 9 exibe a página onde o cliente pode fazer a sua procura pelo prestador. Ele pode usar o filtro de categorias ou pesquisar por nome de prestador ou categoria. Nessa tela, ele consegue acessar a tela de perfil do prestador que ele deseja contratar, clicando no botão "visualizar perfil".

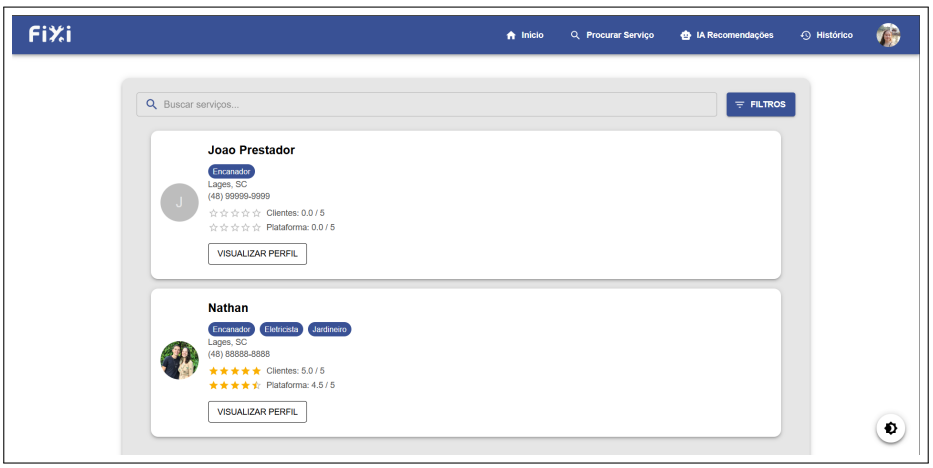

Figura 9. Interface gráfica onde pode ser realizada a busca por prestadores de serviço por parte do cliente.

3.5.4. Interface Gráfica 4

A interface apresentada na Figura 10 exibe o perfil do prestador de serviço, mostrando suas avaliações e as informações cadastradas desse prestador. O cliente pode visualizar as datas de agendamento disponíveis clicando no botão "realizar agendamento".

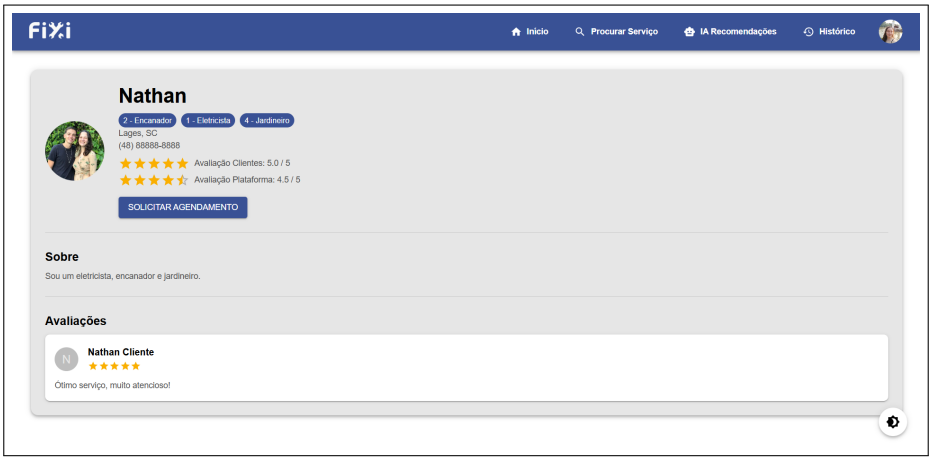

Figura 10. Interface gráfica onde o cliente pode visualizar o perfil do prestador e solicitar o agendamento de determinado serviço.

3.6. Tecnologias Utilizadas

No desenvolvimento deste trabalho, foram empregadas tecnologias modernas e consolidadas, selecionadas por sua eficiência, escalabilidade e confiabilidade. Cada tecnologia adotada desempenha um papel específico dentro da arquitetura proposta, garantindo integração adequada entre os módulos, usabilidade da interface e robustez no processamento de dados. A seguir, detalham-se as principais tecnologias utilizadas.

3.6.1. *React*

O *React* é uma biblioteca *JavaScript* desenvolvida pelo *Facebook* para criação de interfaces de usuário baseadas em componentes reutilizáveis (Team, 2025). Sua finalidade é facilitar o desenvolvimento de aplicações *web* interativas, dinâmicas e responsivas, promovendo a modularidade e a manutenção do código. Na plataforma *Fixi*, o *React* foi utilizado para implementar a interface de usuário, incluindo as telas de cadastro de prestadores, envio de descrições de problemas e visualização das informações, proporcionando uma experiência fluida e intuitiva (React Team, 2025). A camada de apresentação foi reforçada com a biblioteca *MUI Material*, baseada em *Material Design*, para garantir consistência visual, responsividade e boas práticas de usabilidade.

3.6.2. *MUI Material*

O *MUI Material* é uma biblioteca de componentes para *React*, desenvolvida com base nas diretrizes de *Material Design* do *Google* (MUI Team, 2025). Sua finalidade é acelerar o desenvolvimento de interfaces modernas e responsivas, garantindo consistência

visual e boas práticas de usabilidade. Na plataforma *Fixi*, o *MUI Material* foi utilizado na construção do *front-end*, oferecendo componentes prontos, como botões, caixas de diálogo e barras de navegação, que foram personalizados para proporcionar uma experiência agradável e consistente ao usuário.

3.6.3. *Java*

O *Java* é uma linguagem de programação orientada a objetos, multiplataforma e amplamente empregada no desenvolvimento de sistemas corporativos (Oracle, 2025). Sua finalidade é fornecer robustez, segurança e portabilidade, possibilitando o desenvolvimento de aplicações escaláveis e de alto desempenho. Na plataforma *Fixi*, o *Java* foi utilizado no *back-end*, responsável pelo processamento das solicitações, pela lógica de negócio e pela integração com o banco de dados *MySQL* e com os módulos de Inteligência Artificial (Oracle, 2025).

3.6.4. *Spring Boot*

O *Spring Boot* é um *framework* baseado no ecossistema *Spring*, amplamente utilizado para simplificar a criação e configuração de aplicações em *Java* (Spring, 2025). Sua principal finalidade é reduzir a complexidade do desenvolvimento ao oferecer configurações automáticas, servidores embarcados e integração facilitada com bibliotecas do ecossistema *Spring*, eliminando grande parte da configuração manual. Na plataforma *Fixi*, o *Spring Boot* foi utilizado no desenvolvimento do *back-end*, possibilitando a criação de serviços robustos, modulares e de fácil manutenção, além de garantir maior escalabilidade e integração fluida com o banco de dados *MySQL* e os módulos de inteligência artificial.

3.6.5. *JPA / Hibernate*

A *Java Persistence API* (JPA) é uma especificação para o mapeamento objeto-relacional (ORM), que permite que classes em *Java* sejam associadas a tabelas em bancos de dados relacionais. O *Hibernate* é a implementação mais popular dessa especificação (Keith e Schincariol, 2018). Sua finalidade é abstrair consultas *SQL* complexas e simplificar o acesso a dados, promovendo independência entre a camada de negócio e a de persistência. Na plataforma *Fixi*, a *JPA* com *Hibernate* foi utilizada para gerenciar a persistência no *MySQL*, assegurando a integridade dos dados e agilidade na manipulação de entidades como clientes, prestadores, categorias, agendamentos e avaliações da plataforma.

3.6.6. *JWT*

O *JSON Web Token* (*JWT*) é um padrão aberto utilizado para autenticação e troca de informações de forma segura entre sistemas distribuídos (Jones et al., 2015). Baseado em um *token* assinado digitalmente, o *JWT* garante que apenas usuários devidamente autenticados possam acessar recursos restritos, preservando confidencialidade e integridade. Na

plataforma *Fixi*, o *JWT* foi adotado como mecanismo de autenticação, permitindo que clientes e prestadores mantenham sessões seguras e que o acesso a recursos seja controlado de acordo com as permissões do usuário.

3.6.7. *APIs REST*

As *APIs REST* (*Representational State Transfer*) constituem um estilo arquitetural para construção de serviços *web*, baseado em operações HTTP (*GET*, *POST*, *PUT*, *DELETE*) (Oracle, 2025). Sua finalidade é padronizar a comunicação entre sistemas distintos, assegurando interoperabilidade, independência entre cliente e servidor e suporte a múltiplos formatos de dados, como *JSON* e *XML*. Na plataforma *Fixi*, as *APIs REST* foram utilizadas para integrar o *front-end* em *React*, o *back-end* em *Java*, o banco de dados *MySQL* e o módulo de Inteligência Artificial.

3.6.8. *MySQL*

O *MySQL* é um sistema de gerenciamento de banco de dados relacional (SGBDR) que organiza informações em tabelas estruturadas, permitindo consultas eficientes e consistentes (Oracle, 2025). Sua finalidade é armazenar, recuperar e manipular dados de forma segura e confiável, garantindo integridade referencial por meio de chaves primárias e estrangeiras. Na plataforma *Fixi*, o *MySQL* foi utilizado para manter os registros de usuários, prestadores, categorias de serviços e agendamentos, otimizando a organização e a consistência dos dados.

3.6.9. *Swagger*

O *Swagger* é um conjunto de ferramentas para design, documentação e teste de *APIs RESTful* (SmartBear, 2025). Sua finalidade é padronizar a descrição de *endpoints*, parâmetros e respostas, permitindo maior clareza no uso das interfaces de comunicação. Na plataforma *Fixi*, o *Swagger* foi empregado para documentar a *API* construída em *Java*, facilitando a integração com o *front-end* em *React*, assegurando transparência nas interações entre os módulos do sistema e permitindo a realização de testes eficientes durante o desenvolvimento.

3.6.10. *ViaCep*

O *ViaCep* é uma *API* pública e gratuita que fornece informações de endereços a partir do *CEP* (Código de Endereçamento Postal) brasileiro (via, 2025). Sua finalidade é simplificar a obtenção de dados como rua, bairro, cidade e estado, reduzindo erros de digitação e aumentando a praticidade no preenchimento de formulários. Na plataforma *Fixi*, o *ViaCep* foi utilizado para automatizar a busca e preenchimento dos dados de endereço de clientes e prestadores, garantindo agilidade no processo de cadastro e consistência das informações armazenadas no banco de dados.

3.6.11. *iText/OpenPDF*

O *iText/OpenPDF* é uma biblioteca de código aberto para a linguagem Java que permite a criação e manipulação de documentos em formato *PDF* (OpenPDF Project, 2025). Ela fornece recursos para inserir textos, tabelas, imagens e elementos gráficos, permitindo a geração de relatórios dinâmicos e personalizados. Na plataforma *Fixi*, o *iText/OpenPDF* foi utilizado para a construção de relatórios de avaliações dos prestadores, possibilitando que os usuários exportem informações organizadas em um formato padronizado e amplamente aceito. A adoção dessa tecnologia contribuiu para aumentar a profissionalização da plataforma e oferecer uma funcionalidade essencial para prestadores que necessitam de documentos de fácil compartilhamento e armazenamento.

3.6.12. Inteligência Artificial

A Inteligência Artificial (IA) refere-se ao desenvolvimento de sistemas capazes de executar tarefas que tradicionalmente exigiriam inteligência humana, como classificação de dados, análise de padrões e apoio à tomada de decisão (Russell e Norvig, 2021; Mecheri et al., 2023). Sua finalidade é aumentar a eficiência de sistemas ao automatizar tarefas complexas e oferecer recomendações inteligentes aos usuários.

Na plataforma *Fixi*, a tecnologia de IA da *Groq*, especializada em arquiteturas de processamento de alto desempenho, foi integrada ao *back-end* para acelerar tarefas de inferência de modelos com baixa latência (Inc., 2025). Esse módulo desempenha duas funções principais:

- **Classificação de descrições**: A IA analisa os textos fornecidos pelos usuários e sugere automaticamente a categoria de prestador mais adequada, otimizando o processo de busca e recomendação de profissionais;
- **Avaliação de comentários**: A IA processa as avaliações textuais recebidas pelos prestadores e gera uma nota normalizada em uma escala de 0 a 5. Essa métrica compõe, junto a outros critérios da plataforma, o cálculo de desempenho mensal dos prestadores.

Esse uso combinado da IA melhora tanto a eficácia da recomendação de serviços quanto a objetividade na avaliação dos prestadores, assegurando maior escalabilidade e padronização nos processos da aplicação (Inc., 2025).

3.7. Implementação da Plataforma

Nesta subseção, detalha-se o processo de desenvolvimento da plataforma, com apresentação de trechos de código e imagens que evidenciam sua implementação prática.

3.7.1. Arquitetura Geral

A plataforma foi estruturada em camadas bem definidas, garantindo modularidade, clareza e facilidade de manutenção. No *back-end* os principais pacotes são:

- *controller*: expõe os *endpoints REST*, recebendo requisições do *front-end* e retornando respostas em formato *JSON*.

- *service*: concentra as regras de negócio, incluindo validações, envio de *e-mails* automáticos e integrações externas.
- *repository*: implementa a comunicação com o banco de dados por meio do *Spring Data JPA*.
- *dto*: realiza a transferência de dados entre as camadas, evitando o acoplamento direto das entidades.
- *model*: contém as entidades anotadas com `@Entity`, representando a modelagem do banco.
- *config*: reúne as classes de configuração, como segurança, autenticação via *JWT* e parâmetros de *CORS*.
- *scheduler*: implementa rotinas automatizadas de atualização, como a expiração de agendamentos.

No *front-end*, desenvolvido em *React* com *TypeScript*, a organização seguiu a seguinte estrutura:

- *components*: componentes reutilizáveis, como diálogos, botões, formulários e cabeçalhos.
- *assets*: recursos estáticos (ícones, imagens e fontes).
- *contexts*: gerenciamento de estados globais com *Context API*, incluindo dados do usuário e autenticação.
- *pages*: telas principais (*login*, cadastro, *home*, agendamentos, avaliações).
- *services*: abstração das chamadas HTTP ao *back-end*, utilizando *fetch* e *Axios*.
- *styles*: arquivos de estilização, garantindo consistência visual.

3.7.2. Cadastro, *Login* e Autenticação

A plataforma estabelece dois perfis de usuários: **Cliente** e **Prestador**. O Cliente representa o usuário que busca contratar um dos serviços disponíveis, enquanto o Prestador é o profissional responsável por oferecer esses serviços e pode ser contratado por meio da plataforma.

Atualmente, estão disponíveis 14 categorias de serviços pré-definidos: eletricista, encanador, pedreiro, jardineiro, cozinheiro privado, babá, motorista, *dog walker*, faxineiro, professor particular, manicure/pedicure, assistente virtual, fotógrafo e consultor de TI.

A autenticação dos usuários é realizada utilizando o padrão *JSON Web Token (JWT)*, o que assegura a proteção das rotas privadas e impede acessos não autorizados, garantindo maior segurança durante a navegação.

No *front-end*, o contexto global de autenticação é controlado pelo `UserContext`, responsável por armazenar as informações do usuário autenticado e pelo gerenciamento do *token*. Já no *back-end*, cada requisição protegida passa por uma validação do *token JWT*, assegurando que apenas usuários devidamente autenticados possam acessar os recursos da aplicação.

A Figura 11 apresenta a *dialog* de *login*, utilizada tanto por clientes quanto por prestadores.

Figura 11. *Dialog* de *login* que aparece após o botão de *Login* ser clicado.

O trecho de código a seguir (Implementação 1) demonstra o componente `PrivateRoute`, responsável por proteger rotas no *front-end* com base no papel do usuário e na validade do *token*.

```
const PrivateRoute: React.FC<PrivateRouteProps> = ({ children, role })
    => {
  const token = localStorage.getItem("token");
  if (!token) {
    return <Navigate to="/main" replace />;
  }
  try {
    const decoded = jwtDecode<CustomJwtPayload>(token);
    const currentTime = Date.now() / 1000;
    if (decoded.exp && decoded.exp < currentTime) {
      localStorage.removeItem("token");
      return <Navigate to="/main" replace />;
    }
};
```

Implementação 1. Trecho de código do componente `PrivateRoute` para proteção de rotas no *front-end*.

O trecho de código a seguir (Implementação 2) apresenta a definição das rotas principais da plataforma, onde o componente `PrivateRoute` é aplicado para restringir o acesso conforme o papel do usuário autenticado (*Cliente* ou *Prestador*).

```
    <Routes>
      <Route path="/main" element={<MainPage />} />
      <Route
        path="/home/cliente"
        element={
          <PrivateRoute role="CLIENTE">
```

```
            <HomeCliente />
          </PrivateRoute>
        }
      />
    {...}
      <Route path="*" element={<Navigate to="/main" replace />} />
    </Routes>
```

Implementação 2. Trecho de códgio para definição de rotas no *front-end* com proteção baseada em papéis.

Por fim, o trecho de código a seguir (Implementação 3) demonstra a implementação do filtro `JwtFilter` no *back-end*, responsável por interceptar as requisições e validar o *token JWT* antes da execução dos *endpoints* protegidos. Caso o *token* esteja ausente ou inválido, a requisição é rejeitada com o código de status HTTP 401 (`Unauthorized`).

```
    public void doFilter(ServletRequest request, ServletResponse
        response, FilterChain chain)throws IOException,
        ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        String authHeader = req.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            res.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }
        String token = authHeader.substring(7);
        try {
            JwtUtil.validateToken(token);
            chain.doFilter(request, response);
        }
        {...}
    }
```

Implementação 3. Trecho de código que demonstra a aplicação do filtro para validação de JWT no *back-end*.

3.7.3. Termo de Consentimento

Durante o processo de cadastro, é obrigatória a leitura e a aceitação do termo de consentimento, em conformidade com a *Lei Geral de Proteção de Dados* (*LGPD*).

Caso o usuário não concorde com os termos, o registro não poderá ser concluído. O documento esclarece como os dados são utilizados, incluindo o compartilhamento de informações de contato entre clientes e prestadores, bem como as responsabilidades da plataforma.

Ao aceitar o termo, o cliente autoriza que seus dados sejam visíveis aos prestadores, e o mesmo ocorre no sentido inverso. Entretanto, os prestadores têm a maioria de suas informações visíveis aos clientes (descrição, nome, telefone, cidade e categorias), enquanto os dados dos clientes só se tornam acessíveis aos prestadores após a solicitação de um agendamento. Não há visualização de informações entre clientes.

A interface gráfica do sistema (Figura 12) apresenta um *dialog* de cadastro, no qual o usuário deve optar por aceitar ou recusar o termo de consentimento.

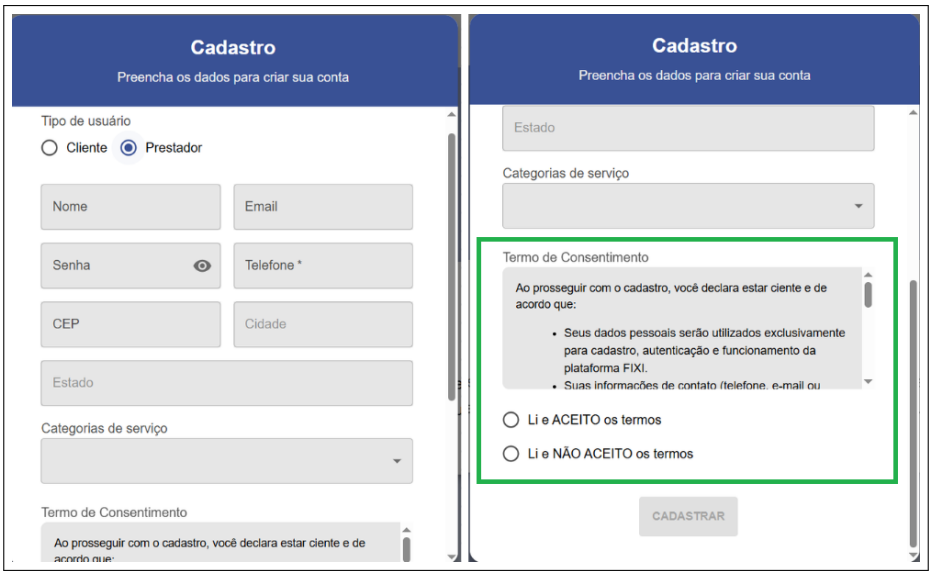

Figura 12. *Dialog* de cadastro de prestadores onde é possível visualizar o termo de consentimento.

3.7.4. Integração com Serviços Externos

Essas integrações ampliam as funcionalidades e a confiabilidade da plataforma, permitindo a automação de tarefas e respostas inteligentes aos usuários.

- **ViaCEP**: utilizado no *front-end* para preenchimento automático de cidade e estado a partir do CEP. Isso reduz erros e torna o cadastro mais ágil.
- **Groq API (IA)**: no *back-end*, a classe `GroqService` conecta-se a um modelo de linguagem para responder dúvidas de clientes e auxiliar na recomendação de prestadores. O serviço utiliza *WebClient* para consumir a *API* e gerar respostas inteligentes a partir de *prompts*.

O código a seguir (Implementação 4) demonstra como ocorre a integração do *ViaCep*.

```
1  const buscarCep = async (valor: string) => {
2    const cepLimpo = valor.replace(/\D/g, "");
3    setCep(valor);
4    {...}
5    if (cepLimpo.length === 8) {
6      try {
7        const resp = await
             fetch(`https://viacep.com.br/ws/${cepLimpo}/json/`);
8        const data = await resp.json();
9        if (!data.erro) {
10         setCidade(data.localidade || "");
11         setEstado(data.uf || "");
12       }
13       {...}
```

```
        }
    }
};
```

Implementação 4. Trecho de código que demonstra a utilização da *API ViaCEP* para preenchimento de dados automáticos (cidade e estado).

O próximo trecho de código (Implementação 5) apresenta a integração com a IA *Groq*, responsável por recomendar prestadores de serviços aos clientes. O *prompt* utilizado para orientar o comportamento da IA pode ser consultado no Apêndice A.1.

```
public String gerarResposta(String mensagemCliente) {
        try {
        {...}
            List<PrestadorCategoria> prestadorCategorias =
                    prestadorCategoriaRepository.findByCategoriaId
                    (categoria.getId());
            if (prestadorCategorias.isEmpty()) {
                return " No momento nao ha prestadores cadastrados na
                    categoria " + categoriaNome +
                        "Por favor, tente novamente mais tarde ou
                            escolha outro servico disponivel na
                            plataforma FIXI.";
            }
            Prestador melhorPrestador = medias.entrySet().stream()
                    .max(Map.Entry.comparingByValue())
                    .map(Map.Entry::getKey)
                    .orElse(prestadorCategorias.get(0).getPrestador());

            double melhorMedia = medias.getOrDefault(melhorPrestador,
                0.0);
            String listaPrestadores = prestadorCategorias.stream()
                    .map(pc -> String.format("%s (%s)      [Ver
                        perfil](http://localhost:3000/prestador/%d)",
                            pc.getPrestador().getNome(),
                            categoriaNome,
                            pc.getPrestador().getId()
                    ))
                    .collect(Collectors.joining("\n"));
                {...}
```

Implementação 5. Trecho de código que demonstra a utilização da IA *Groq* para recomendação de prestadores.

A interface gráfica apresentada (Figura 13) exibe o *chat* disponível para o cliente se comunicar com a Inteligência Artificial.

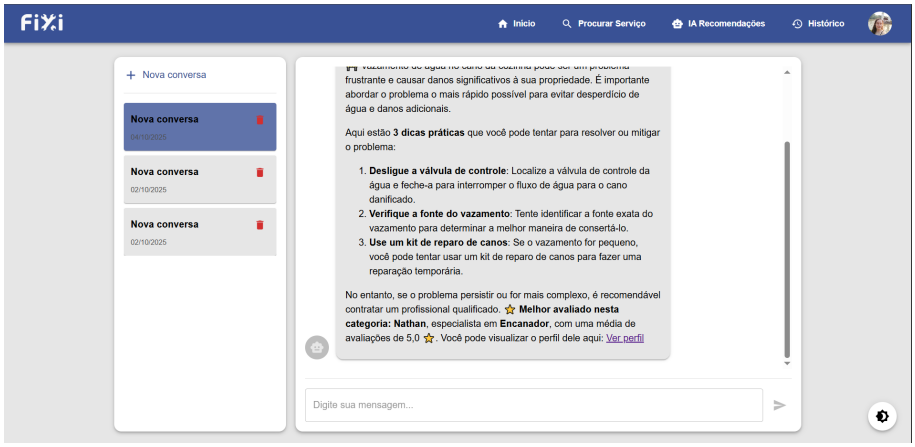

Figura 13. Interface gráfica onde o cliente pode se comunicar com a IA para pedir recomendações de serviços.

O código completo da integração com a IA *Groq* para a recomendação de prestadores de serviço com base no problema indicado pelo usuário está disponível no Apêndice A.3.

3.7.5. Agendamentos

O fluxo de agendamentos é central para a plataforma. O cliente solicita um serviço a um prestador, que pode aceitar ou recusar. O `AgendamentoService` gerencia este processo, garantindo:

- Validação de disponibilidade do prestador.
- Registro do status (*pendente*, *aceito*, *recusado*, *expirado*).
- Atualizações automáticas em caso de não resposta.

O código a seguir (Implementação 6) demonstra o método `aceitarAgendamento`, presente no `AgendamentoService`.

```
1   {...}
2   public void aceitarAgendamento(Long prestadorId, Long
        agendamentoId) {
3   Agendamento agendamento =
        agendamentoRepository.findById(agendamentoId)
4       .orElseThrow(() -> new RuntimeException("Agendamento nao
            encontrado"));
5   if (!agendamento.getPrestador().getId().equals(prestadorId)) {
6   throw new RuntimeException("Prestador nao autorizado para esse
        agendamento.");
7   }
8       agendamento.setStatus(StatusAgendamento.ACEITO);
9       agendamentoRepository.save(agendamento);
10      Cliente cliente = agendamento.getCliente();
11      Prestador prestador = agendamento.getPrestador();
12      enviarEmailAceiteCliente(cliente, prestador, agendamento);
13      enviarEmailAceitePrestador(prestador, cliente, agendamento);
14  }
```

```
15      {...}
16      }
```

Implementação 6. Trecho de código que demonstra a função que é chamada quando o prestador aceita o agendamento.

A interface gráfica apresentada na Figura 14, exibe o *dialog* onde o cliente pode realizar o agendamento. As datas disponíveis para agendamento correspondem a um intervalo de sete dias, sendo o dia atual considerado como o primeiro dia desse período.

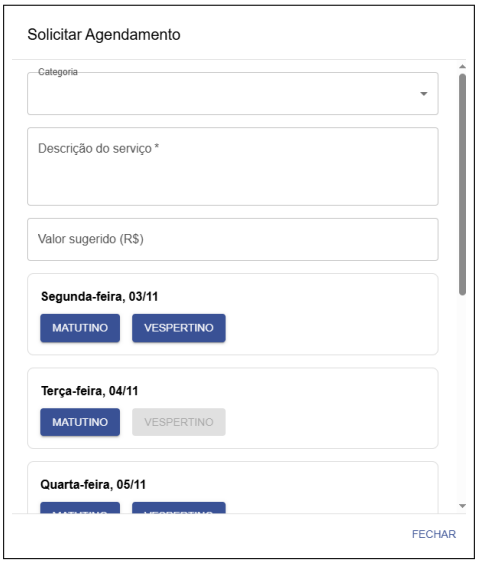

Figura 14. *Dialog* de solicitação de agendamento ao prestador escolhido.

O código que demonstra o `AgendamentoService` do *back-end* e do *front-end* está disponível no Apêndice A.3.

3.7.6. Envio de *E-mail*s Automáticos

Para garantir a comunicação entre as partes, a plataforma envia *e-mails* em diferentes situações:

- Cancelamento (cliente ou prestador).
- Expiração (não confirmação dentro do prazo).
- Aceite do agendamento (contendo dados de contato para que cliente e prestador possam se comunicar diretamente).

Foram implementados métodos específicos, como `enviarEmailAceiteCliente` (Implementação 7).

```
1  public void enviarEmailAceiteCliente(Cliente cliente, Prestador
       prestador, Agendamento agendamento) {
2          {...}
3          try {
4              MimeMessage msg = mailSender.createMimeMessage();
5              var helper = new MimeMessageHelper(msg, true, "UTF-8");
```

```
6               helper.setFrom(from);
7               helper.setTo(cliente.getEmail());
8               helper.setSubject("Agendamento aceito pelo prestador");
9               helper.setText(html, true);
10              mailSender.send(msg);
11              System.out.println("Email de aceite enviado para cliente");
12          }
13          {...}
14      }
```

Implementação 7. Trecho de código que exemplifica o envio de *e-mail* quando o agendamento é aceito.

3.7.7. Avaliações e Reputação

Após a execução do serviço, o cliente pode avaliar o prestador, gerando uma reputação visível em seu perfil. Assim como o prestador de serviço também poderá realizar uma avaliação sobre o cliente, seguindo os mesmos moldes da avaliação feita pelo cliente. Essas avaliações só ficarão disponíveis no perfil dos seus respectivos, após os dois (prestador e cliente) efetuarem suas avaliações, para evitar viés.

A interface gráfica da Figura 15 exibe o *dialog* de avaliação, disponibilizado ao cliente somente após a data do agendamento aceito, permitindo que ele registre sua experiência com o prestador. Esse *dialog* está disponível na tela de histórico de agendamentos.

Figura 15. *Dialog* de avaliação, onde o cliente avalia o prestador e o serviço.

Além dessas avaliações diretas, a própria plataforma realiza uma análise automatizada de desempenho, composta por critérios objetivos e subjetivos. Essa avaliação periódica considera os seguintes indicadores:

- Tempo de Plataforma: mede o tempo de permanência do prestador na *Fixi*, representando sua experiência e engajamento com a plataforma.
- Taxa de Aceitação: reflete o nível de comprometimento e disponibilidade do prestador em aceitar os agendamentos solicitados pelos clientes.
- Taxa de Cancelamento: avalia a confiabilidade e responsabilidade do prestador, penalizando aqueles que cancelam serviços com frequência.
- Avaliação por Inteligência Artificial: a IA *Groq* realiza uma análise de sentimento dos comentários deixados pelos clientes após a conclusão dos serviços. A plataforma interpreta as avaliações textuais e atribui uma nota entre 0 e 5, considerando aspectos como satisfação, cordialidade, qualidade e confiabilidade. Esse

critério complementa as métricas quantitativas com uma análise qualitativa baseada em linguagem natural, proporcionando uma visão mais ampla do desempenho do prestador.

Após o cálculo individual de cada métrica, o método `calcularNotaFinal()` gera uma nota global de desempenho para cada prestador. Essa nota reflete o equilíbrio entre experiência, comprometimento, confiabilidade e satisfação dos clientes, servindo como base para os relatórios mensais e para o algoritmo de recomendação dos prestadores mais bem avaliados na plataforma.

Essas métricas são calculadas automaticamente a cada 60 dias, seguindo práticas consolidadas no mercado, como as utilizadas por plataformas de reputação profissional, a exemplo do Mercado Livre (Implementação 8).

```
public void iniciar() {
        Runnable tarefa = () -> {
            System.out.println(" Calculando notas mensais dos
               prestadores...");
            avaliacaoService.calcularNotasMensais();
        };
        scheduler.scheduleAtFixedRate(tarefa, 0, 60, TimeUnit.DAYS);
    }
```

Implementação 8. Trecho de código que demonstra a periodização da avaliação da plataforma para o prestador.

O trecho a seguir (Implementação 9) ilustra a função responsável pelo cálculo das métricas de avaliação do prestador.

```
{...}
    private Double calcularTaxaCancelamento(Prestador prestador) {
        long total = agendamentoRepository.countByPrestador(prestador);
        if (total == 0) return 5.0;
        long cancelados =
           agendamentoRepository.countByPrestadorAndStatus(prestador,
           StatusAgendamento.CANCELADO);
        double taxaCancel = cancelados / (double) total;
        return (1 - taxaCancel) * 5.0;
    }
    {...}
```

Implementação 9. Trecho de código que exemplifica um dos cálculos feitos pela plataforma para compor a nota do prestador.

O *prompt* utilizado para orientar o comportamento da IA para avaliação dos comentários dos clientes referentes ao prestador pode ser encontrado no Apêndice A.1. E o código completo dos cálculos realizados pela plataforma está disponível no Apêndice A.3.

3.7.8. Relatórios em PDF

A plataforma oferece a funcionalidade de exportar relatórios em formato PDF contendo as avaliações recebidas pelos prestadores de serviços. Para a geração desses documentos, foi empregada a biblioteca `iText/OpenPDF`, que possibilita a criação de arquivos com

tabelas, textos e formatações personalizadas (Implementação 10). Esse recurso fornece aos usuários uma forma de visualizar, analisar e arquivar os dados de desempenho, contribuindo para um acompanhamento mais estruturado da evolução do prestador ao longo do tempo. Um dos relatórios gerados pode ser visualizado na Figura 16.

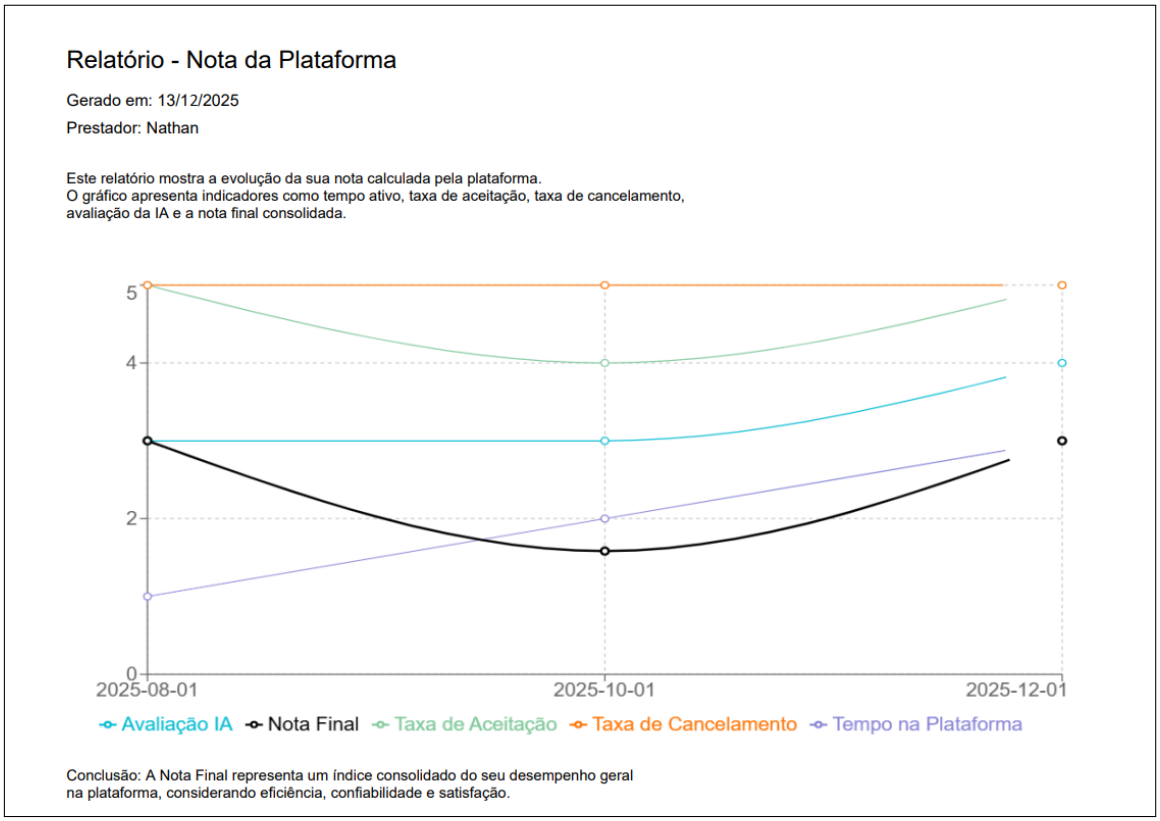

Figura 16. Relatório gerado pela plataforma, que mostra as notas que a plataforma gerou para determinado prestador.

```
try {
  {...}
  const root = createRoot(container);
  root.render(grafico);
  await new Promise((resolve) => setTimeout(resolve, 1200));
  const canvas = await html2canvas(container);
  const imgData = canvas.toDataURL("image/png");
  const pdf = new jsPDF("landscape");
  pdf.text("Relatorio - Desempenho Geral", 15, 15);
  pdf.addImage(imgData, "PNG", 15, 30, 260, 120);
  autoTable(pdf, {
    startY: 160,
    head: [["Periodo", "Nota Final (IA)", "Media Clientes"]],
    body: tabelaResumo.map(
      (linha: { periodo: string; notaIA: string; mediaClientes:
         string }) => [
        linha.periodo,
        linha.notaIA,
        linha.mediaClientes,
      ]
```

```
      ),
    });
    pdf.save("desempenho-geral.pdf");
    root.unmount();
    container.remove();
  }
```

Implementação 10. Trecho de código para a geração de relatórios para auxílio do prestador (Desempenho Geral do Prestador).

O código completo da geração dos três relatórios está disponível no apêndice A.3.

3.7.9. Automatizações

Foi implementada a anotação `@Scheduled` para monitorar periodicamente os agendamentos pendentes (Implementação 11). Caso o prestador não responda dentro do prazo estipulado, a plataforma altera automaticamente o status para **expirado** e envia notificações por *e-mail* às partes envolvidas. Caso um prestador não responda até o horário limite, a plataforma altera o status para **expirado** automaticamente e envia notificações por *e-mail* ao cliente e ao prestador.

```
    @Scheduled(cron = "0 0 * * * *")
    @Transactional
    public void atualizarAgendamentosExpirados() {
        LocalDateTime hoje = LocalDateTime.now();
        List<Agendamento> agendamentos =
            agendamentoRepository.findAll();
        for (Agendamento ag : agendamentos) {
            LocalDate dataAg = ag.getDataAgendamento();
            LocalDateTime limite;
            if (ag.getPeriodo() == Periodo.MATUTINO) {
                limite = dataAg.atTime(12, 0);
            } else {
                limite = dataAg.atTime(18, 0);
            }
            if (ag.getStatus() == StatusAgendamento.PENDENTE &&
                hoje.isAfter(limite)) {
                ag.setStatus(StatusAgendamento.EXPIRADO);
                agendamentoRepository.save(ag);

                enviarEmailExpiradoCliente(ag.getCliente(), ag);
                enviarEmailExpiradoPrestador(ag.getPrestador(), ag);
            }
        }
    }
```

Implementação 11. Trecho de código que demonstra a periodização para expirar a solicitação de agendamentos

4. Avaliação

Nesta seção, é descrito o processo de avaliação da plataforma *Fixi* e os resultados obtidos. Para a realização da avaliação, foram criados dois formulários utilizando o *Google Forms*

(um para prestadores de serviços e outro para clientes) devido à sua facilidade de uso e de compartilhamento pela *internet*. Os formulários foram encaminhados para um amplo grupo de pessoas, uma vez que a plataforma se encaixa em vários nichos e não em apenas um determinado grupo. As perguntas gerais (que estão presentes nos dois formulários) são:

1. O sistema está adequado (atende) às suas necessidades?
2. O sistema possui características (interfaces intuitivas, vocabulário conhecido pelo usuário etc) que permitem que ele seja aprendido com facilidade?
3. O sistema é atrativo, ou seja, as interfaces estão bem projetadas, os componentes de menu são fáceis de localizar e as cores são adequadas?
4. As funcionalidades que você acessou e utilizou funcionaram de maneira correta?
5. Informe o que você mudaria ou aquilo que você não gostou no sistema.
6. Informe o que você gostou no sistema.
7. Qual nota geral você daria para o sistema?

As perguntas específicas para os prestadores de serviços:

1. A página de perfil do prestador contém informações suficientes sobre seus serviços?
2. Você considera o painel de avaliações da plataforma útil para acompanhar seu desempenho?

A pergunta específica para os clientes:

1. A recomendação de prestadores feita pela Inteligência Artificial foi adequada ao serviço solicitado?

As perguntas 5 e 6 são abertas, a pergunta 7 segue uma escala de 0 a 10, e as demais perguntas utilizam cinco alternativas baseadas no modelo de escala de atitudes proposto por *Likert* (Likert, 1932), sendo elas:

1. Concordo totalmente;
2. Concordo;
3. Não sei responder;
4. Discordo;
5. Discordo totalmente;

4.1. Análise de Resultados

Nessa subseção, serão analisados os resultados obtidos em cada uma das perguntas que aparecem nos formulários e as conclusões que podem ser feitas a partir das respostas. O questionário de clientes foi respondido por 14 pessoas e o de prestadores de serviços, por 12 pessoas.

A primeira questão avaliou se o sistema atende às necessidades dos usuários. Os resultados podem ser observados nas Figuras 17 e 18, que apresentam, respectivamente, as respostas dos clientes e dos prestadores.

Figura 17. Respostas do formulário do cliente referente a 1ª pergunta geral.

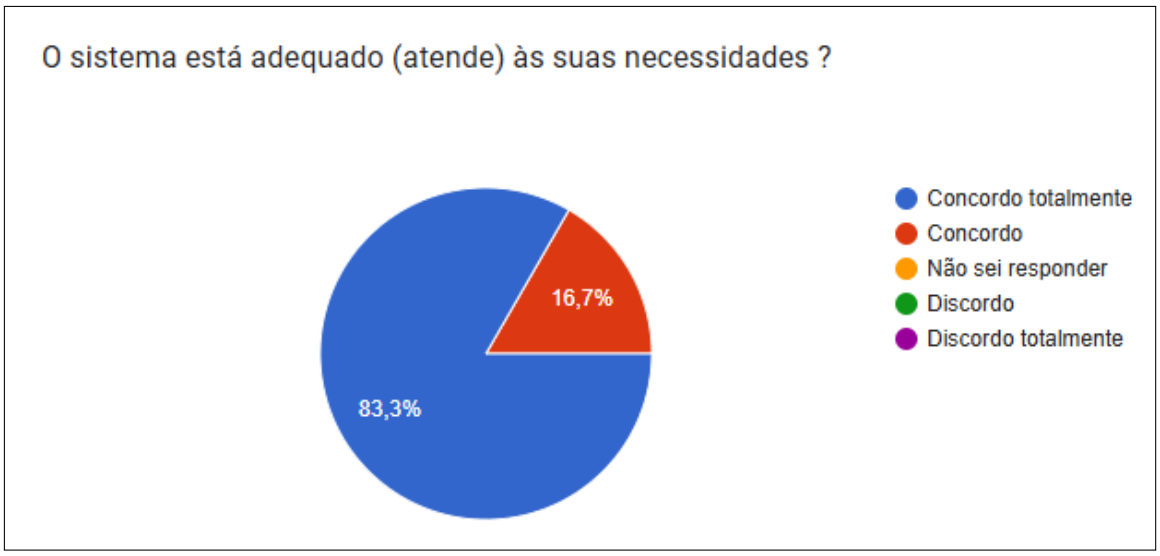

Figura 18. Respostas do formulário do prestador de serviço referente a 1ª pergunta geral.

As questões 2 e 3 questionaram a intuitividade da plataforma e o quanto ela é visualmente atrativa. Os resultados encontram-se na Figura 19 para o público cliente e na Figura 20 para os prestadores de serviço.

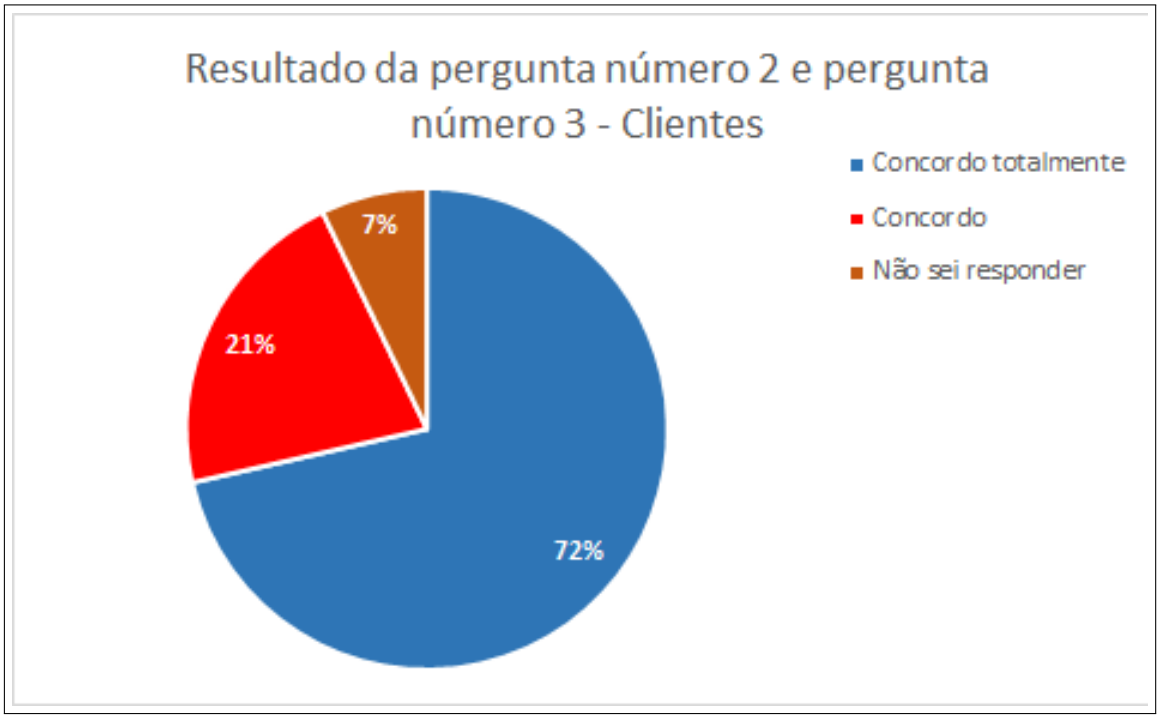

Figura 19. Respostas do formulário do cliente referente a 2ª e 3ª pergunta geral.

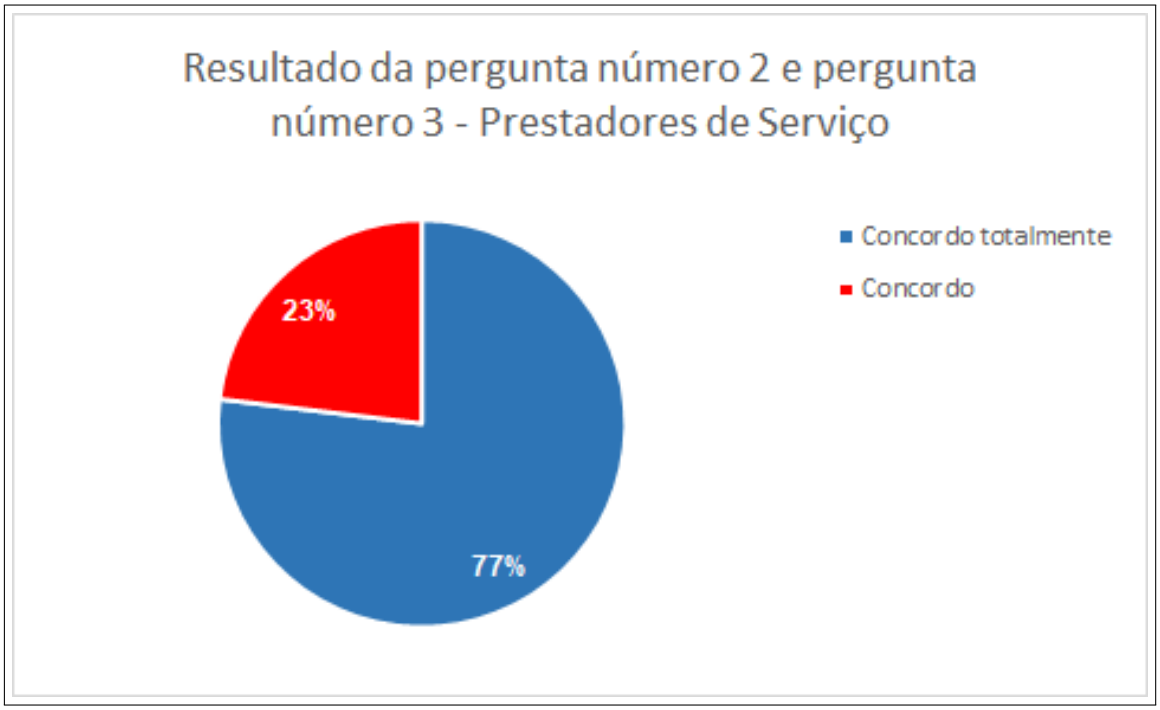

Figura 20. Respostas do formulário do prestador de serviço referente a 2ª e 3ª pergunta geral.

A questão 4 teve como foco as funcionalidades testadas pelos avaliadores. As Figuras 21 e 22 apresentam, de forma comparativa, a percepção de cada grupo.

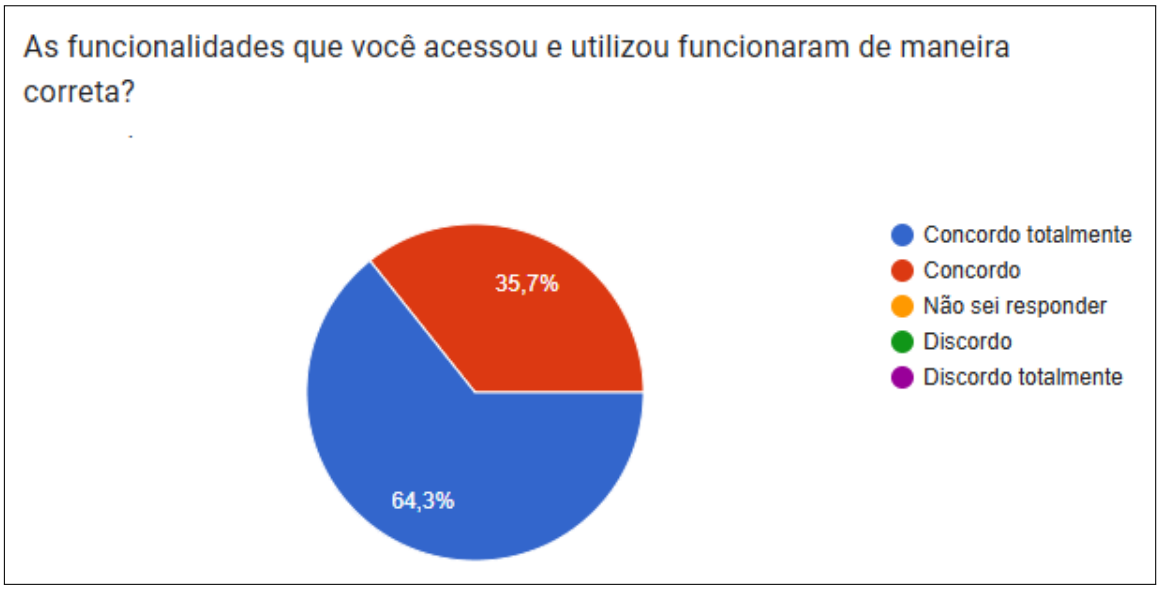

Figura 21. Respostas do formulário do cliente referente a 4ª pergunta geral.

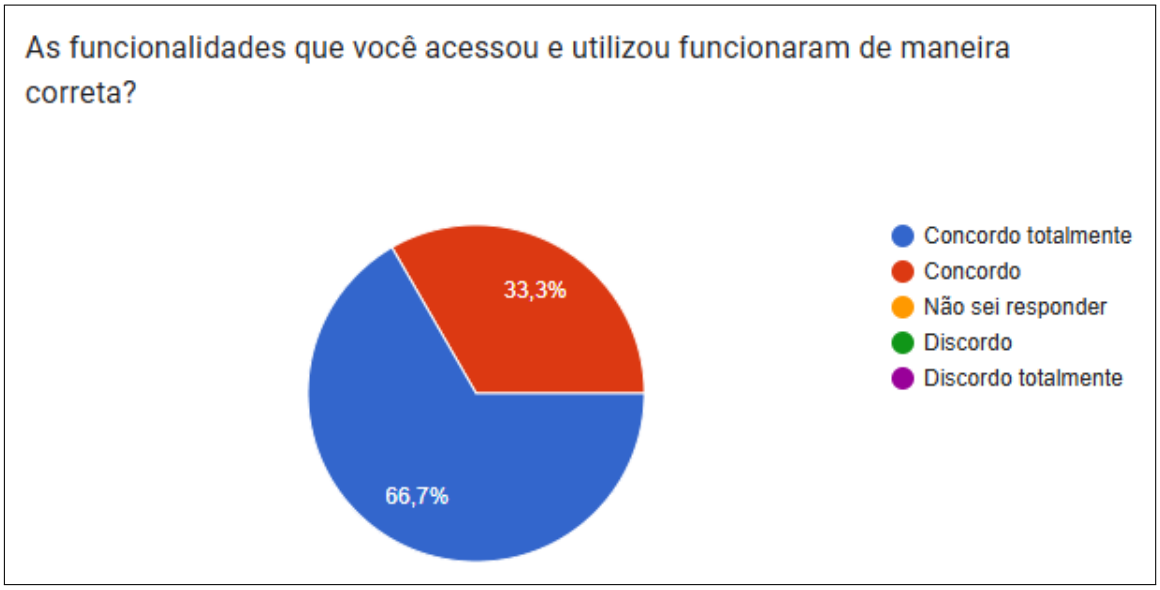

Figura 22. Respostas do formulário do prestador de serviço referente a 4ª pergunta geral.

A questão 5 é uma das questões abertas que aborda o que os avaliadores mudariam na plataforma ou o que não gostaram. Tanto nas respostas dos prestadores de serviço quanto nas respostas dos clientes, observou-se uma similaridade nas respostas e, por isso, optou-se por apresentar apenas um gráfico de nuvem de palavras para a questão 5 (Figura 23) e para a questão 6 também (Figura 24), já que se refere ao que os avaliadores gostaram na plataforma.

Figura 23. Gráfico de nuvem de palavras referente às respostas da questão geral 5 dos clientes e prestadores de serviço.

Figura 24. Gráfico de nuvem de palavras referente às respostas da questão geral 6 dos clientes e prestadores de serviço.

A questão específica para os clientes refere-se à utilização da inteligência artificial na recomendação de prestadores de serviço. A figura 25 demonstra as respostas obtidas no formulário dos clientes.

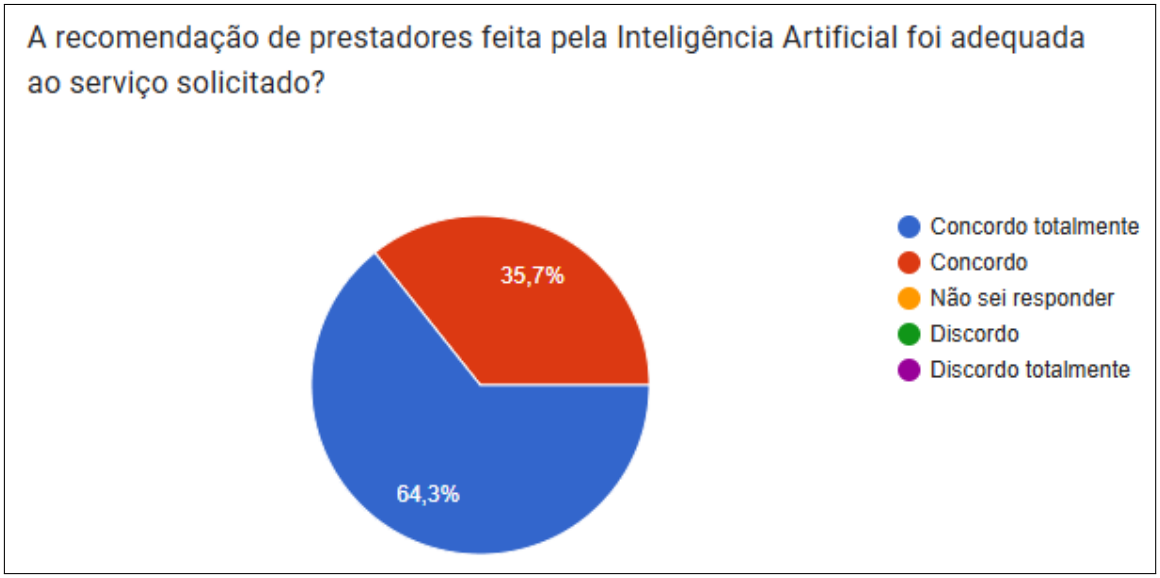

Figura 25. Respostas do formulário dos clientes referente à pergunta específica sobre a IA.

As questões específicas dos prestadores de serviços referem-se ao cadastro. A primeira delas questiona se o prestador consegue colocar informações suficientes em seu perfil, e a segunda questiona se o painel de avaliações, que contém os relatórios, foi útil para o prestador. A Figura 26 demonstra as respostas obtidas na questão sobre as informações e a Figura 27 demonstra sobre o painel de avaliações.

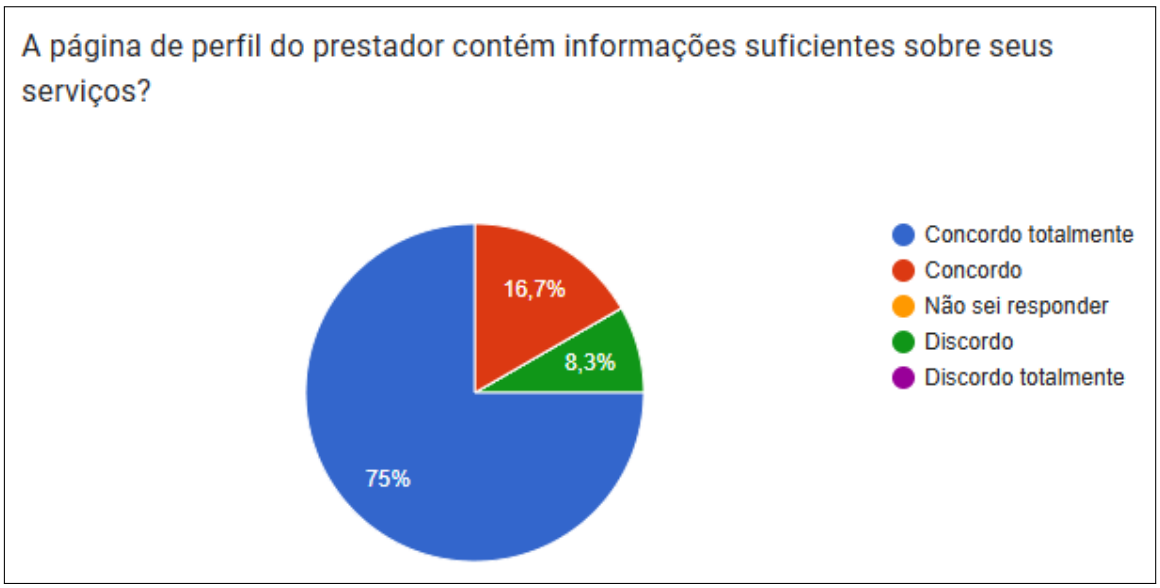

Figura 26. Respostas do formulário dos prestadores de serviço referente à pergunta específica sobre o perfil.

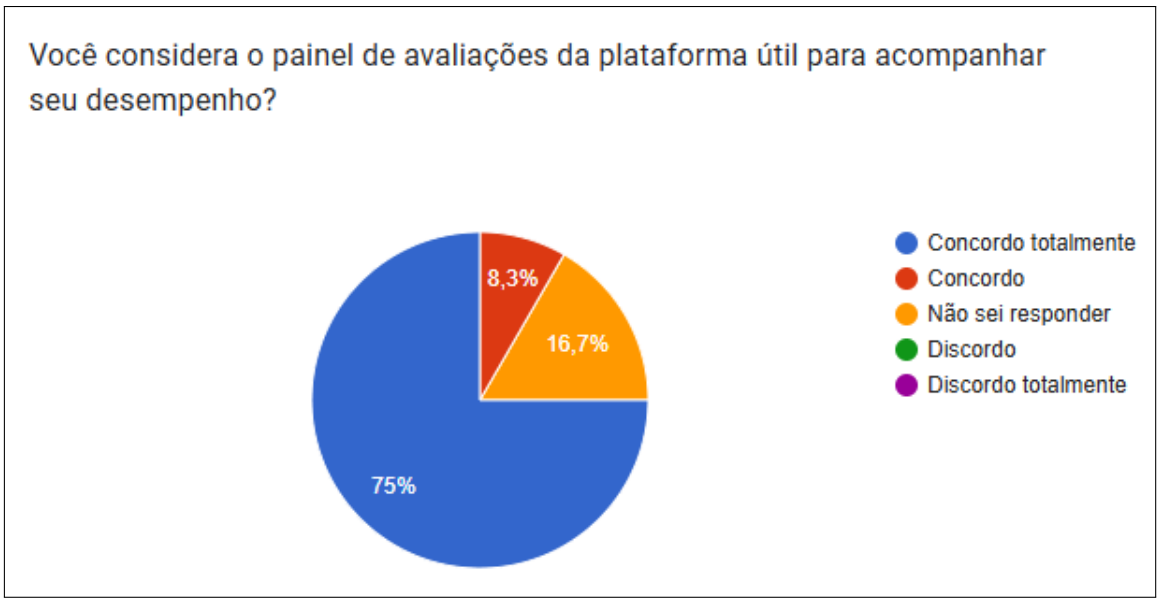

Figura 27. Respostas do formulário dos prestadores de serviço referente à pergunta específica sobre o painel de avaliações.

Por fim, a última pergunta solicitou uma avaliação geral da plataforma, em escala de 0 a 10. As Figuras 28 e 29 sintetizam essa análise, permitindo observar a nota atribuída por clientes e prestadores.

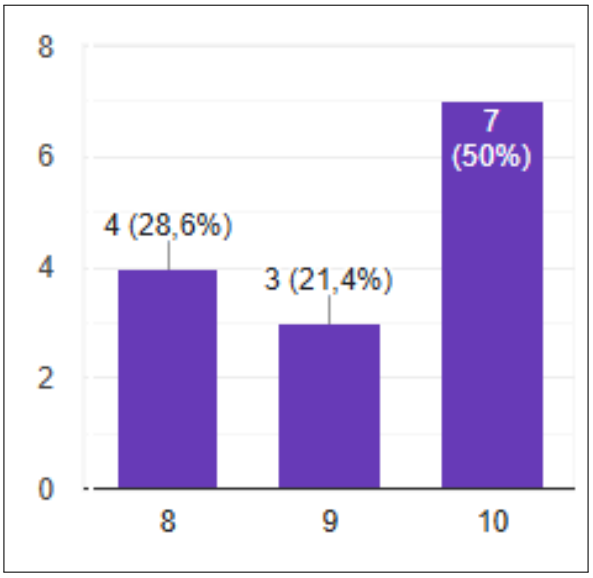

Figura 28. Avaliação da plataforma feita pelos clientes.

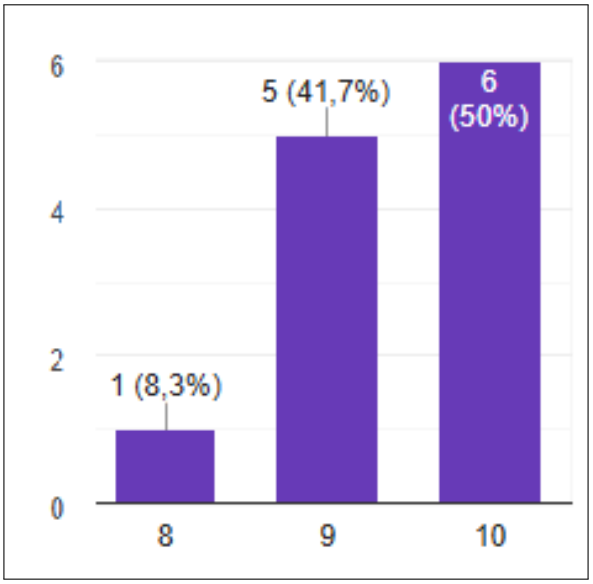

Figura 29. Avaliação da plataforma feita pelos prestadores de serviço.

De modo geral, os resultados obtidos nas avaliações indicam que a plataforma foi bem recebida tanto por clientes quanto por prestadores. A maioria dos participantes considerou o sistema adequado às suas necessidades, intuitivo e atrativo visualmente. As respostas abertas reforçaram a percepção de facilidade de uso e destacaram o diferencial proporcionado pela inteligência artificial nas recomendações. As médias das notas finais atribuídas pelos avaliadores demonstram um alto grau de satisfação, confirmando o alcance dos objetivos propostos neste trabalho.

5. Conclusão

Este trabalho apresentou o desenvolvimento de uma plataforma *web* com o objetivo de auxiliar clientes a encontrarem prestadores de serviços de forma organizada, segura e ágil.

Dessa forma, os usuários podem recorrer à plataforma sempre que necessitarem contratar um profissional, contando ainda com o suporte de uma inteligência artificial que os auxilia na resolução de dúvidas e na recomendação de prestadores confiáveis. Por outro lado, os prestadores podem utilizar a plataforma como meio de divulgação de seus serviços e de captação de novos clientes. Além disso, o sistema oferece relatórios de desempenho, que podem ser baixados, permitindo o acompanhamento da atuação dos prestadores ao longo do tempo.

Após a avaliação do sistema, observou-se que a plataforma possui potencial de uso cotidiano, apresentando uma interface intuitiva e objetiva, o que facilita sua utilização. O uso da inteligência artificial, diferencial deste projeto, mostrou-se eficaz tanto na recomendação de prestadores, quando analisada pela perspectiva do cliente, quanto na análise automatizada de comentários, quando vista do ponto de vista dos prestadores de serviço.

Os objetivos propostos no início deste trabalho foram alcançados, conforme evidenciado pelas avaliações realizadas. Foi desenvolvida uma plataforma acessível, capaz de apresentar informações relevantes sobre os prestadores, possibilitando o agendamento de serviços e a interação entre clientes e profissionais. As principais tecnologias utilizadas foram *React*, *Java*, *MySQL* e o uso de inteligência artificial por meio da *API Groq*.

Durante o desenvolvimento, algumas limitações foram identificadas. O uso de *APIs* externas implica custos adicionais de operação e dependência de serviços de terceiros. Além disso, o uso de inteligência artificial pode gerar erros inesperados, decorrentes de respostas probabilísticas e da necessidade de ajustes constantes do modelo.

Como trabalhos futuros, sugere-se a implementação de novas funcionalidades, como um chat interno para facilitar a comunicação entre clientes e prestadores, a possibilidade de inserir imagens e certificados nos perfis dos profissionais e nas avaliações realizadas, o desenvolvimento de uma interface de suporte interativo para novos usuários, a ampliação da gama de categorias disponíveis, bem como a disponibilidade do *Fixi* para dispositivos *mobile*. Também se recomenda aprimorar o design da interface gráfica, visando tornar a experiência do usuário ainda mais agradável e eficiente.

Referências

(2025). Viacep - webservice cep e ibge gratuito. https://viacep.com.br/. Acesso em: 24 set. 2025.

Connolly, T. e Begg, C. (2021). *Database Systems: A Practical Approach to Design, Implementation and Management*. Pearson, 6th edition.

Dragicevic, K. (2022). Trends in modern web development: Ci/cd, containers and beyond. *Journal of Systems and Software*, 183:111–125.

Edrone (2024). Dados do e-commerce no brasil: faturamento, consumidores e tendências. Acesso em: 12 set. 2025.

Elmasri, R. e Navathe, S. B. (2020). *Fundamentals of Database Systems*. Pearson, 7th edition.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

IBGE (2024). Internet chega a 74,9 milhões de domicílios do país em 2024. Acesso em: 12 set. 2025.

Iliadis, L., Maglogiannis, I., e Papadopoulos, H. (2023). *AI Applications for Web and Data Science*. Springer.

Inc., G. (2025). Groq: Infraestrutura de inferência de ia de alto desempenho. Acesso em: 8 set. 2025.

ISO/IEC/IEEE 29148 (2018). ISO/IEC/IEEE 29148:2018 – Systems and Software Engineering — Life Cycle Processes — Requirements Engineering. https://ieeexplore.ieee.org/servlet/opac?punumber=8559684. IEEE Standard 29148-2018. IEEE Xplore punumber: 8559684.

Jones, M., Bradley, J., e Sakimura, N. (2015). Json web token (jwt) - rfc 7519. Acessado em 24 set. 2025.

Keith, M. e Schincariol, M. (2018). *Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs*. Apress.

Li, W. e Zhang, Y. (2021). Design and implementation of a web application based on react and spring boot. In *2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1806–1810.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Mecheri, K., Chaari, W. L., Chatti, S., e Abraham, A. (2023). Deep learning based web service recommendation methods. *Journal of Intelligent & Fuzzy Systems*, 45(6):7795–7808.

Ministério do Desenvolvimento, Indústria, C. e. S. M. (2024). E-commerce no brasil cresce 4,8% e alcança r$ 196 bi em 2023. Acesso em: 12 set. 2025.

MUI Team (2025). Mui material documentation. Acessado em 24 set. 2025.

NielsenIQ | Ebit (2023). Webshoppers 47: Relatório sobre o comércio eletrônico brasileiro em 2022. Acesso em: 12 set. 2025.

Oliveira, A. V. M., Oliveira, M. R. B., e de Paiva, L. F. R. (2023). Plataforma web para contratação de serviços. *Revista Acadêmica da Universidade de Uberaba*, pages 1–12. Artigo acadêmico.

OpenPDF Project (2025). Openpdf documentation. https://github.com/LibrePDF/OpenPDF. Acesso em: 01 out. 2025.

Oracle (2025). Building restful web services with java. https://www.oracle.com/java/technologies/restful-web-services.html. Acesso em: 2025-09-07.

Oracle (2025). The java™ tutorials. https://docs.oracle.com/javase/tutorial/. Acesso em: 2025-09-07.

Oracle (2025). Mysql documentation. https://dev.mysql.com/doc/. Acesso em: 2025-09-07.

React Team (2025). Optimizing performance. https://reactjs.org/docs/optimizing-performance.html. Acesso em: 2025-09-07.

Ricci, F., Rokach, L., e Shapira, B., editors (2015). *Recommender Systems Handbook*. Springer, 2nd edition.

Richardson, L., Amundsen, M., e Ruby, S. (2013). *RESTful Web APIs: Services for a Changing World*. O'Reilly Media.

Roy, D. e Dutta, M. (2022). A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1):1–45.

Russell, S. J. e Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition.

Saadatmand, M., Crnkovic, I., e Carlson, J. (2019). Modern web development practices: A multivocal literature review. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 123–134.

Schwaber, K. e Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Acesso em: 12 set. 2025.

Shahin, M., Liang, P., e Babar, M. A. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Silva, A. M. (2025). Conectevidas: Desenvolvimento de uma aplicação para conectar profissionais de saúde a pacientes que necessitam de atenção domiciliar. Trabalho de conclusão de curso (graduação em redes de computadores), Universidade Federal do Ceará, Campus de Quixadá.

Silva, D. (2016). Desenvolvimento de uma rede social na Área de reparos residenciais. Trabalho de conclusão de curso (bacharelado em ciência da computação), Universidade Tecnológica Federal do Paraná.

Silva, R. J. R. (2021). Sistema para contratação de profissionais de ti. Trabalho de conclusão de curso (bacharelado em sistemas de informação), Universidade Federal de Uberlândia.

SmartBear (2025). Swagger: Api design and documentation. https://swagger.io/. Acesso em: 2025-09-07.

Sommerville, I. (2011). *Software Engineering*. Addison-Wesley, 9th edition.

Souza, A. N., Né, E. S. P., e Caravieri, F. P. M. (2021). Conecta jobs: E-marketplace para prestadores de serviços. In *V Simpósio de Tecnologia da FATEC Jales*, pages 1–10, Jales, SP. Fatec Jales. ISSN 2595-2323.

Souza, R. e Ferreira, L. (2022). Digital transformation and consumer behavior in emerging economies. *Revista Brasileira de Gestão e Inovação*, 9(2):45–62.

Spring (2025). Spring boot documentation. Acessado em 24 set. 2025.

Team, R. (2025). React – a javascript library for building user interfaces. https://reactjs.org/. Acesso em: 2025-09-07.

Vicente, P. N. e Burnay, C. D. (2024). Recommender systems and over-the-top services: A systematic review study (2010–2022). *Applied System Innovation*, 7(3):80.

Watt, A. e Eng, N. (2015). *Database Design*. BCcampus, 2nd edition. Acesso em: 8 set. 2025.

Wiegers, K. E. e Beatty, J. (2013). *Software Requirements, 3rd Edition*. Microsoft Press.

A. Apêndice

A.1. *Prompts* de comando para a IA da *Groq*

```
String prompt = """
    Voce e uma IA de suporte para o aplicativo de servicos domesticos FIXI.
    So pode responder perguntas relacionadas a servicos da plataforma.
    Profissionais disponiveis na plataforma (use apenas estes para recomendar):
    %s
    O cliente perguntou: "%s"
    Monte uma resposta em portugues, seguindo este formato:
    Texto introdutorio explicando o problema.
     Liste 3 dicas praticas que o cliente pode tentar resolver ou mitigar o problema.
     No final, recomende o melhor avaliado da categoria, incluindo nome, especialidade e link do perfil.
     Sempre inclua o link no formato Markdown: [Ver perfil](URL).
    Use Markdown para formatar em negrito e listas numeradas.
    %s
""".formatted(listaPrestadores, mensagemCliente, destaque);

// Chamada a IA
Map<String, Object> body = Map.of(
        "model", "llama-3.3-70b-versatile",
        "messages", new Object[]{
                Map.of("role", "system", "content", "Voce e um assistente de servicos domesticos da plataforma FIXI."),
                Map.of("role", "user", "content", prompt)
        },
        "temperature", 0.4
);
Map<String, Object> response = webClient.post()
        .uri("/chat/completions")
        .bodyValue(body)
        .retrieve()
        .bodyToMono(Map.class)
        .block();

if (response == null || response.get("choices") == null) {
    return "Nao foi possivel obter resposta da IA no momento.";
}

var choices = (List<Map<String, Object>>) response.get("choices");
var message = (Map<String, Object>) choices.get(0).get("message");
return (String) message.get("content");

} catch (Exception e) {
    e.printStackTrace();
    return " Erro ao consultar a IA.";
}
```

Implementação 12. *Prompt* de resposta ao cliente quando ele solicita ajuda à IA integrada na plataforma.

```java
public Double avaliarComentariosPrestador(List<String> comentarios) {
        if (comentarios == null || comentarios.isEmpty()) {
            return 0.0;
        }

        String prompt = """
        Voce e um avaliador imparcial.
        Analise os seguintes comentarios de clientes sobre um
           prestador:

        %s

        Com base neles, atribua uma nota unica de 0 a 5 que represente
           a satisfacao media geral.
        Retorne apenas o numero, sem texto extra.
        """.formatted(String.join("\n", comentarios));

        try {
            Map<String, Object> body = Map.of(
                    "model", "llama-3.3-70b-versatile",
                    "messages", new Object[]{
                            Map.of("role", "system", "content", "Voce
                               e um avaliador de qualidade de
                               prestadores."),
                            Map.of("role", "user", "content", prompt)
                    },
                    "temperature", 0.0
            );

            Map<String, Object> response = webClient.post()
                    .uri("/chat/completions")
                    .bodyValue(body)
                    .retrieve()
                    .bodyToMono(Map.class)
                    .block();

            if (response == null || response.get("choices") == null) {
                return 0.0;
            }

            var choices = (java.util.List<Map<String, Object>>)
               response.get("choices");
            var message = (Map<String, Object>)
               choices.get(0).get("message");
            String content = ((String) message.get("content")).trim();

            // Extrai numero 0 a 5 da resposta
            return parseNota(content);

        }
    }
```

Implementação 13. *Prompt* para avaliação de comentários pela IA

A.2. Interfaces Gráficas

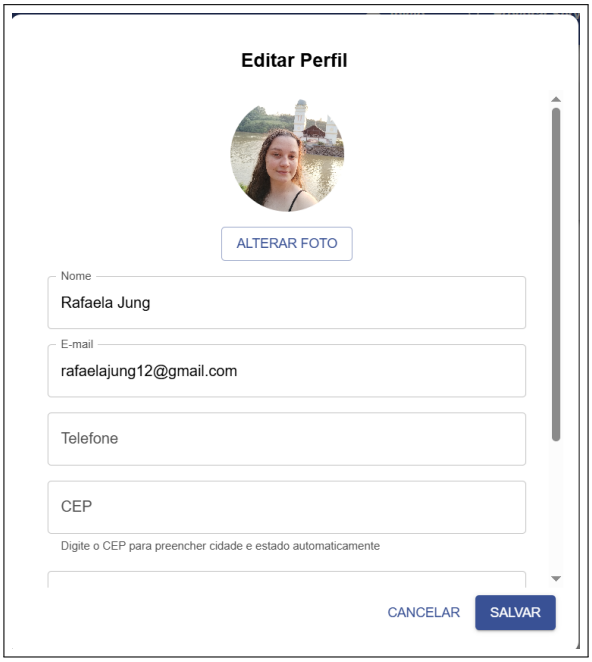

Figura 30. *Dialog* responsável pela edição do perfil do cliente.

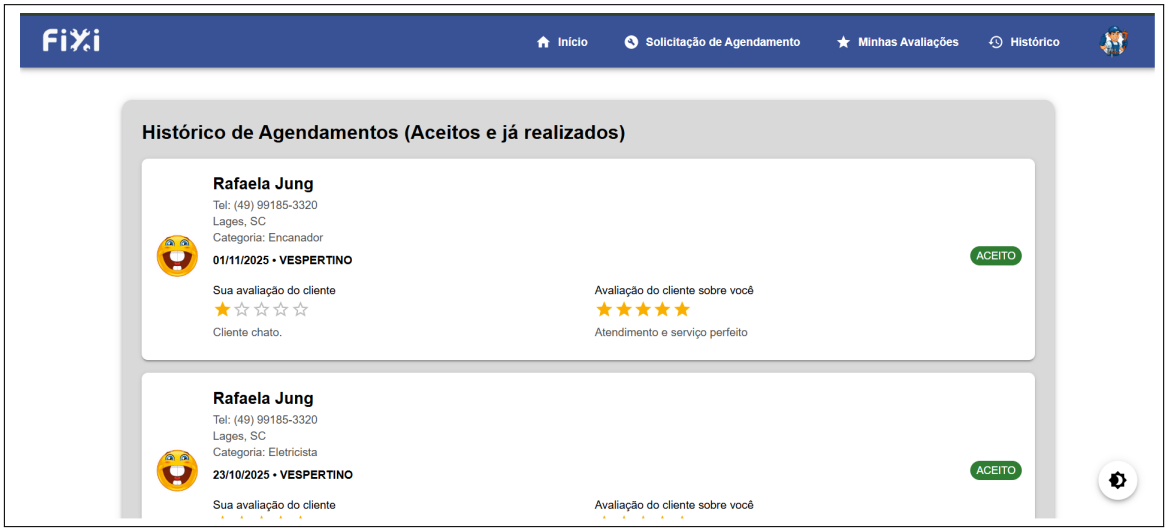

Figura 31. Interface onde o prestador pode visualizar seu histórico de agendamentos concluídos.

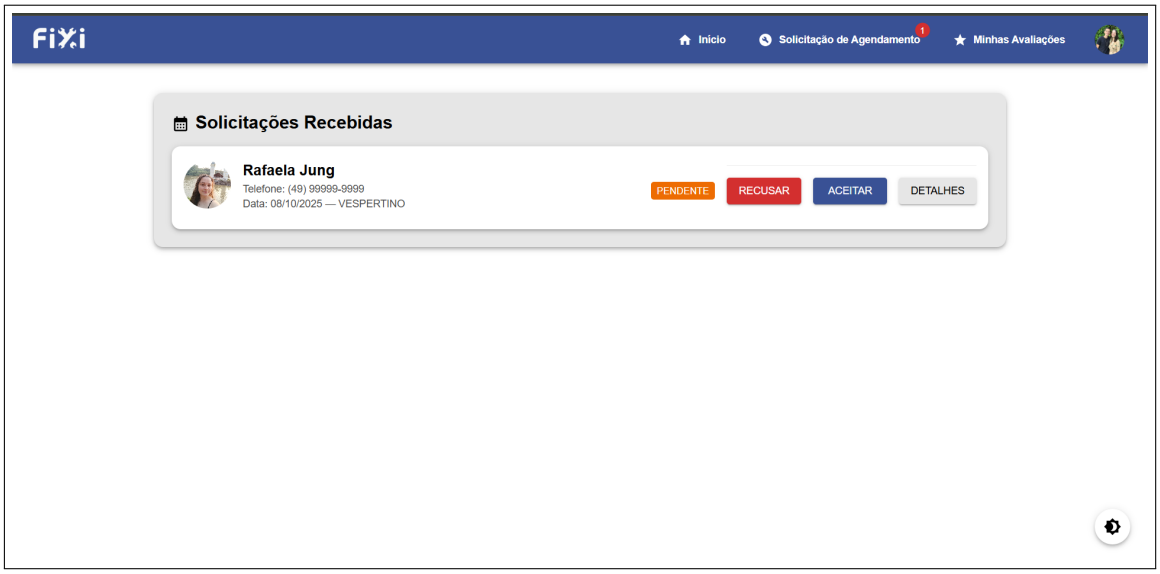

Figura 32. Interface onde o prestador pode visualizar suas solicitações de serviço.

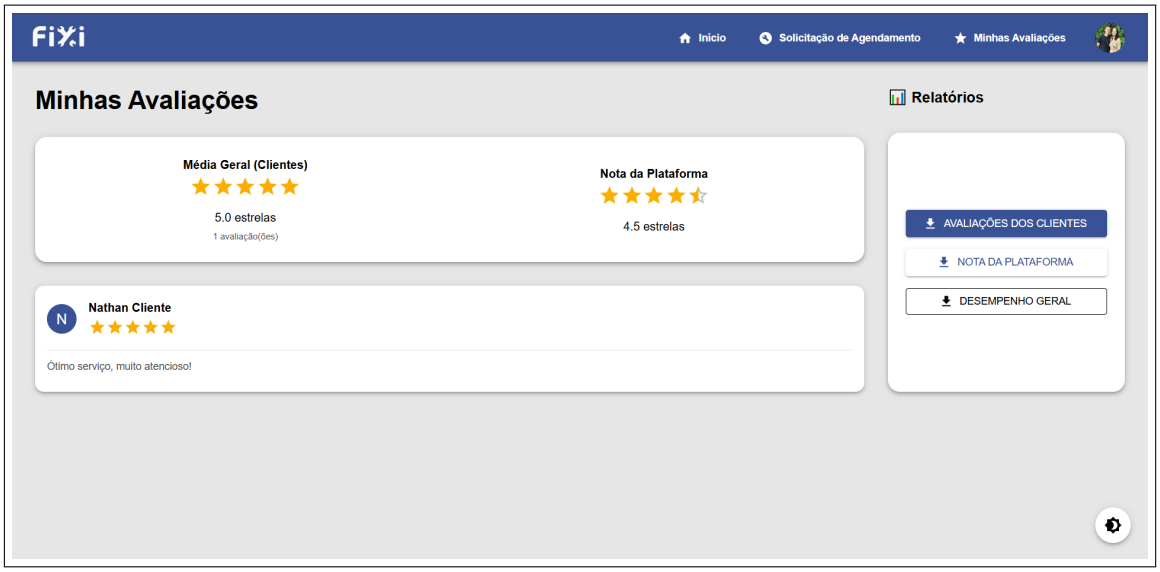

Figura 33. Interface onde o prestador visualiza as suas avaliações e pode realizar o *download* de relatórios.

A.3. Códigos Complementares

```
{...}
public String gerarResposta(String mensagemCliente) {
        try {
            // Classifica a categoria
            String categoriaNome =
                classificarCategoria(mensagemCliente).trim();
            // Fora do escopo
            if ("FORA_DO_ESCOPO".equalsIgnoreCase(categoriaNome)) {
                return "Nao posso fornecer informacoes que nao sejam
                    relacionadas aos servicos da plataforma FIXI. "
```

```
                    + "Se precisar de ajuda com servicos
                        domesticos, estou a disposicao para
                        recomendar um profissional qualificado.";
            }

            // Busca a categoria
            Categoria categoria =
                categoriaRepository.findByNome(categoriaNome);
            if (categoria == null) {
                return "Nao encontrei a categoria " + categoriaNome +
                    " na plataforma FIXI.";
            }
            // Busca os prestadores da categoria
            List<PrestadorCategoria> prestadorCategorias =
                    prestadorCategoriaRepository.findByCategoriaId
                    (categoria.getId());

            if (prestadorCategorias.isEmpty()) {
                return "No momento nao ha prestadores cadastrados na
                    categoria " + categoriaNome +
                         ". Por favor, tente novamente mais tarde ou
                            escolha outro servico disponivel na
                            plataforma FIXI.";
            }
            //Calcula a media de avaliacoes de cada prestador
            Map<Prestador, Double> medias = new HashMap<>();

            for (PrestadorCategoria pc : prestadorCategorias) {
                Prestador prestador = pc.getPrestador();
                // busca agendamentos do prestador que possuem
                    avaliacao
                List<Agendamento> agendamentos =
                    agendamentoRepository.findHistoricoByClienteId
                (prestador.getId());

                double somaNotas = 0.0;
                int totalNotas = 0;
                for (Agendamento ag : agendamentos) {
                    if (ag.getAvaliacao() != null &&
                        ag.getAvaliacao().getNota() != null) {
                        somaNotas += ag.getAvaliacao().getNota();
                        totalNotas++;
                    }
                }
                double media = totalNotas > 0 ? somaNotas / totalNotas
                    : 0.0;
                medias.put(prestador, media);
            }

            //Seleciona o prestador com maior media
            Prestador melhorPrestador = medias.entrySet().stream()
                    .max(Map.Entry.comparingByValue())
                    .map(Map.Entry::getKey)
                    .orElse(prestadorCategorias.get(0).getPrestador());

```

```
53              double melhorMedia = medias.getOrDefault(melhorPrestador,
                    0.0);
54
55              // Monta lista de prestadores
56              String listaPrestadores = prestadorCategorias.stream()
57                      .map(pc -> String.format("%s (%s)     [Ver
                            perfil](http://localhost:3000/prestador/%d)",
58                              pc.getPrestador().getNome(),
59                              categoriaNome,
60                              pc.getPrestador().getId()
61                      ))
62                      .collect(Collectors.joining("\n"));
63
64              // Destaque do melhor avaliado
65              String destaque = """
66                  Melhor avaliado nesta categoria: %s
67                  Media de avaliacoes: %.1f
68                  [Ver perfil](http://localhost:3000/prestador/%d)
69                  """.formatted(melhorPrestador.getNome(), melhorMedia,
                        melhorPrestador.getId());
70
71              // Prompt final
72              String prompt = """
73                  Voce e uma IA de suporte para o aplicativo de servicos
                        domesticos FIXI.
74                  So pode responder perguntas relacionadas a servicos da
                        plataforma.
75
76                  Profissionais disponiveis na plataforma (use apenas
                        estes para recomendar):
77                  %s
78
79                  O cliente perguntou: "%s"
80
81                  Monte uma resposta em portugues, seguindo este formato:
82                  Texto introdutorio explicando o problema.
83                  Liste 3 dicas praticas que o cliente pode tentar
                        resolver ou mitigar o problema.
84                  No final, recomende o **melhor avaliado** da
                        categoria, incluindo nome, especialidade e link do
                        perfil.
85
86                  Sempre inclua o link no formato Markdown: [Ver
                        perfil](URL).
87                  Use Markdown para formatar em **negrito** e listas
                        numeradas.
88
89                  %s
90                  """.formatted(listaPrestadores, mensagemCliente,
                        destaque);
91
92              // Chamada a IA
93              Map<String, Object> body = Map.of(
94                      "model", "llama-3.3-70b-versatile",
95                      "messages", new Object[]{
```

```
                                Map.of("role", "system", "content", "Voce
                                    e um assistente de servicos domesticos
                                    da plataforma FIXI."),
                                Map.of("role", "user", "content", prompt)
                        },
                        "temperature", 0.4
                );

                Map<String, Object> response = webClient.post()
                        .uri("/chat/completions")
                        .bodyValue(body)
                        .retrieve()
                        .bodyToMono(Map.class)
                        .block();

                if (response == null || response.get("choices") == null) {
                    return "Nao foi possivel obter resposta da IA no
                        momento.";
                }

                var choices = (List<Map<String, Object>>)
                    response.get("choices");
                var message = (Map<String, Object>)
                    choices.get(0).get("message");
                return (String) message.get("content");

            } catch (Exception e) {
                e.printStackTrace();
                return "Erro ao consultar a IA.";
            }
        }
{...}
```

Implementação 14. Trecho de código onde a IA classifica o prestador que melhor se enquadra no problema descrito pelo usuário.

```
{...}
public void cancelarAgendamentoCliente(Long agendamentoId, Long
    clienteId) {
        cancelarAgendamento(agendamentoId, clienteId, false);
        Cliente cliente =
            clienteRepository.findById(clienteId).orElse(null);
        Agendamento agendamento =
            agendamentoRepository.findById(agendamentoId).orElse(null);

        if (cliente != null && agendamento != null) {
            Prestador prestador = agendamento.getPrestador();
            if (prestador != null) {
                System.out.println("Cliente cancelou     enviar e-mail
                    para o prestador");
                enviarEmailPrestador(prestador, agendamento);
            }
        } else {
            System.out.println("Nao enviou e-mail porque cliente ou
                agendamento veio null");
        }
```

```
16      }
17  public void cancelarAgendamentoPrestador(Long agendamentoId, Long
        prestadorId) {
18          cancelarAgendamento(agendamentoId, prestadorId, true);
19
20          Prestador prestador =
                prestadorRepository.findById(prestadorId).orElse(null);
21          Agendamento agendamento =
                agendamentoRepository.findById(agendamentoId).orElse(null);
22
23          if (prestador != null && agendamento != null) {
24              Cliente cliente = agendamento.getCliente();
25              if (cliente != null) {
26                  System.out.println("Prestador cancelou     enviar
                        e-mail para o cliente");
27                  enviarEmailCliente(cliente, agendamento);
28              }
29          } else {
30              System.out.println("Nao enviou e-mail porque prestador ou
                    agendamento veio null");
31          }
32      }
33  private void cancelarAgendamento(Long agendamentoId, Long usuarioId,
        boolean isPrestador) {
34          Agendamento ag = agendamentoRepository.findById(agendamentoId)
35                  .orElseThrow(() -> new RuntimeException("Agendamento
                        nao encontrado"));
36
37          if (isPrestador &&
                !ag.getPrestador().getId().equals(usuarioId)) {
38              throw new RuntimeException("Prestador nao autorizado a
                    cancelar este agendamento");
39          }
40          if (!isPrestador &&
                !ag.getCliente().getId().equals(usuarioId)) {
41              throw new RuntimeException("Cliente nao autorizado a
                    cancelar este agendamento");
42          }
43
44          ag.setStatus(StatusAgendamento.CANCELADO);
45          ag.setCanceladoPor(isPrestador ? "PRESTADOR" : "CLIENTE");
46
47          agendamentoRepository.save(ag);
48      }
49  public AgendamentoRespostaDTO solicitarAgendamento(
50              Long prestadorId,
51              Long clienteId,
52              Long idCategoria,
53              LocalDate data,
54              Periodo periodo,
55              String descricaoServico,
56              Double valorSugerido
57      ) {
58          var prestador = prestadorRepository.findById(prestadorId)
59                  .orElseThrow(() -> new RuntimeException("Prestador nao
                        encontrado"));
```

```
60          var cliente = clienteRepository.findById(clienteId)
61                  .orElseThrow(() -> new RuntimeException("Cliente nao
                        encontrado"));
62
63          var categoria = categoriaRepository.findById(idCategoria)
64                  .orElseThrow(() -> new RuntimeException("Categoria nao
                        encontrada"));
65
66          if (categoria == null) {
67              throw new RuntimeException("Categoria nao encontrada");
68          }
69
70          // verifica se o prestador possui a categoria
71          boolean prestadorPossuiCategoria =
                prestador.getCategorias().stream()
72                  .anyMatch(pc ->
                        pc.getCategoria().getId().equals(idCategoria));
73          if (!prestadorPossuiCategoria) {
74              throw new RuntimeException("Prestador nao atende a
                    categoria selecionada.");
75          }
76
77          // verifica disponibilidade
78          boolean prestadorOcupado = agendamentoRepository.
79              existsByPrestadorIdAndDataAgendamentoAndPeriodoAndStatusIn(
80                  prestadorId,
81                  data,
82                  periodo,
83                  List.of(StatusAgendamento.PENDENTE,
                        StatusAgendamento.ACEITO)
84          );
85          if (prestadorOcupado) {
86              throw new RuntimeException("Prestador ja possui
                    agendamento nesse periodo.");
87          }
88
89          Agendamento ag = new Agendamento();
90          ag.setPrestador(prestador);
91          ag.setCliente(cliente);
92          ag.setCategoria(categoria);
93          ag.setDataAgendamento(data);
94          ag.setPeriodo(periodo);
95          ag.setStatus(StatusAgendamento.PENDENTE);
96          ag.setDataSolicitacao(LocalDateTime.now());
97
98          //novos campos
99          ag.setDescricaoServico(descricaoServico);
100         ag.setValorSugerido(valorSugerido);
101
102         var salvo = agendamentoRepository.save(ag);
103
104         return toDTO(salvo);
105     }
106 public void aceitarAgendamento(Long prestadorId, Long agendamentoId) {
107         Agendamento agendamento =
                agendamentoRepository.findById(agendamentoId)
```

```
                .orElseThrow(() -> new RuntimeException("Agendamento
                    nao encontrado"));

        if (!agendamento.getPrestador().getId().equals(prestadorId)) {
            throw new RuntimeException("Prestador nao autorizado para
                esse agendamento.");
        }

        agendamento.setStatus(StatusAgendamento.ACEITO);
        agendamentoRepository.save(agendamento);

        Cliente cliente = agendamento.getCliente();
        Prestador prestador = agendamento.getPrestador();

        enviarEmailAceiteCliente(cliente, prestador, agendamento);
        enviarEmailAceitePrestador(prestador, cliente, agendamento);
    }
public void recusarAgendamento(Long prestadorId, Long agendamentoId) {
        Agendamento agendamento =
            agendamentoRepository.findById(agendamentoId)
                .orElseThrow(() -> new RuntimeException("Agendamento
                    nao encontrado"));

        if (!agendamento.getPrestador().getId().equals(prestadorId)) {
            throw new RuntimeException("Prestador nao autorizado para
                esse agendamento.");
        }

        agendamento.setStatus(StatusAgendamento.NEGADO);
        agendamentoRepository.save(agendamento);

        enviarEmailAgendamentoRecusadoCliente(agendamento.getCliente(),
            agendamento);
    }
{...}
```

Implementação 15. Trecho de código onde é possível visualizar as principais funções do `AgendamentoService` do *back-end*.

```
{...}
    export async function solicitarAgendamento(
      clienteId: number,
      prestadorId: number,
      idCategoria: number,
      data: string,
      periodo: Periodo,
      descricaoServico: string,
      valorSugerido?: number | null
    ) {
      const { data: response } = await
          api.post<AgendamentoRespostaDTO>(
        `/prestadores/${prestadorId}/agendamentos`,
        null,
        {
          params: {
            clienteId,
```

```
            idCategoria,
            data,
            periodo,
            descricaoServico,
            valorSugerido,
          },
        }
      );
      return response;
    }
    export async function aceitarAgendamentoPrestador(prestadorId:
        number, agendamentoId: number) {
      const { data } = await api.put(
        `/prestadores/${prestadorId}/agendamentos/
            ${agendamentoId}/aceitar`
      );
      return data.mensagem;
    }
    export async function recusarAgendamentoPrestador(
      prestadorId: number,
      agendamentoId: number
    ) {
      const { data } = await api.put(
        `/prestadores/${prestadorId}/agendamentos
            /${agendamentoId}/recusar`
      );
      return data;
    }
    export async function cancelarAgendamentoPrestador(
      idAgendamento: number,
      prestadorId: number
    ) {
      const { data } = await api.put(
        `/prestadores/${prestadorId}/agendamentos
            /${idAgendamento}/cancelar`
      );
      return data;
    }
{...}
```

Implementação 16. Trecho de código onde é possível visualizar as principais funções do `AgendamentoService` do *front-end*.

```
{...}
public void calcularNotasMensais() {
        List<Prestador> prestadores = prestadorRepository.findAll();
        LocalDate periodo = LocalDate.now().withDayOfMonth(1);

        for (Prestador prestador : prestadores) {
            AvaliacaoPlataforma avaliacao = new AvaliacaoPlataforma();
            avaliacao.setPrestador(prestador);

            avaliacao.setTempoPlataforma
                (calcularTempoPlataforma(prestador));
            avaliacao.setTaxaAceitacao
                (calcularTaxaAceitacao(prestador));
```

```
            avaliacao.setTaxaCancelamento
                (calcularTaxaCancelamento(prestador));
            avaliacao.setAvaliacaoIa
                (calcularAvaliacaoIa(prestador));

            avaliacao.calcularNotaFinal();
            avaliacao.setPeriodoReferencia(periodo);

            avaliacaoRepository.save(avaliacao);
        }
    }
 private Double calcularTempoPlataforma(Prestador prestador) {
        if (prestador.getDataCadastro() == null) return 0.0;

        long mesesNaPlataforma =
            java.time.temporal.ChronoUnit.MONTHS.between(
                prestador.getDataCadastro(), LocalDate.now()
        );

        return Math.min((mesesNaPlataforma / 12.0) * 5.0, 5.0);
    }
private Double calcularTaxaAceitacao(Prestador prestador) {
        long total = agendamentoRepository.countByPrestador(prestador);
        if (total == 0) return 0.0;

        long aceitos =
            agendamentoRepository.countByPrestadorAndStatus(prestador,
            StatusAgendamento.ACEITO);
        return (aceitos / (double) total) * 5.0;
    }
private Double calcularTaxaCancelamento(Prestador prestador) {
        long total = agendamentoRepository.countByPrestador(prestador);
        if (total == 0) return 5.0;

        long cancelados =
            agendamentoRepository.countByPrestadorAndStatus
        (prestador, StatusAgendamento.CANCELADO);
        double taxaCancel = cancelados / (double) total;
        return (1 - taxaCancel) * 5.0;
    }
private Double calcularAvaliacaoIa(Prestador prestador) {
        List<String> comentarios =
            agendamentoRepository.findComentariosByPrestador
        (prestador.getId());
        return groqService.avaliarComentariosPrestador(comentarios);
    }
{...}
```

Implementação 17. Trecho de código onde é possível visualizar os cálculos realizados para a geração da nota da plataforma de cada prestador.

```
 const handleDownloadAvaliacoes = async () => {
  try {
    if (!user?.id) {
      alert("Usuario nao encontrado");
      return;
```

```
    }
    const response = await
        fetch(`http://localhost:8080/avaliacoes/${user.id}/download`, {
      method: "GET",
    });
    if (!response.ok) {
      throw new Error("Erro ao baixar avaliacoes");
    }
    const blob = await response.blob();
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "avaliacoes-clientes.pdf";
    document.body.appendChild(a);
    a.click();
    a.remove();
    window.URL.revokeObjectURL(url);
  } catch (err) {
    console.error(err);
    alert("Erro ao baixar avaliacoes");
  }
};
  const handleDownloadNotaPlataforma = async () => {
    try {
      if (!user?.id) {
        alert("Usuario nao encontrado");
        return;
      }
      const dados = await buscarAvaliacoesPlataforma(user.id);
      // cria container oculto
      const container = document.createElement("div");
      container.style.width = "800px";
      container.style.height = "400px";
      container.style.position = "absolute";
      container.style.top = "-9999px";
      document.body.appendChild(container);
      const grafico = (
        <ResponsiveContainer width={800} height={400}>
          <LineChart data={dados}>
            <CartesianGrid strokeDasharray="3 3" />
            <XAxis dataKey="periodoReferencia" />
            <YAxis domain={[0, 5]} />
            <Tooltip />
            <Legend />
            <Line type="monotone" dataKey="tempoPlataforma"
                stroke="#8884d8" name="Tempo na Plataforma" />
            <Line type="monotone" dataKey="taxaAceitacao"
                stroke="#82ca9d" name="Taxa de Aceitacao" />
            <Line type="monotone" dataKey="taxaCancelamento"
                stroke="#ff7300" name="Taxa de Cancelamento" />
            <Line type="monotone" dataKey="avaliacaoIa"
                stroke="#00bcd4" name="Avaliacao IA" />
            <Line type="monotone" dataKey="notaFinal" stroke="#000"
                strokeWidth={2} name="Nota Final" />
          </LineChart>
        </ResponsiveContainer>
```

```
      );
      const root = createRoot(container);
      root.render(grafico);
      await new Promise((resolve) => setTimeout(resolve, 1200));
      const canvas = await html2canvas(container);
      const imgData = canvas.toDataURL("image/png");
      const pdf = new jsPDF("landscape");
      const hoje = new Date().toLocaleDateString("pt-BR");
      pdf.setFontSize(18);
      pdf.text("Relatorio - Nota da Plataforma", 15, 15);
      pdf.setFontSize(12);
      pdf.text(`Gerado em: ${hoje}`, 15, 25);
      pdf.text(`Prestador: ${user?.nome}`, 15, 32);
      pdf.setFontSize(11);
      pdf.text(
        "Este relatorio mostra a evolucao da sua nota calculada pela plataforma.\n" +
        "O grafico apresenta indicadores como tempo ativo, taxa de aceitacao, taxa de cancelamento,\n" +
        "avaliacao da IA e a nota final consolidada.",
        15,
        45,
        { maxWidth: 260 }
      );
      pdf.addImage(imgData, "PNG", 15, 70, 260, 120);
      pdf.text(
        "Conclusao: A Nota Final representa um indice consolidado do seu desempenho geral\nna plataforma, considerando eficiencia, confiabilidade e satisfacao.",
        15,
        200,
        { maxWidth: 260 }
      );
      pdf.save("nota-plataforma.pdf");
      root.unmount();
      container.remove();
    } catch (err) {
      console.error(err);
      alert("Erro ao gerar PDF da Nota da Plataforma");
    }
  };
  const handleDownloadDesempenhoGeral = async () => {
    try {
      if (!user?.id) {
        alert("Usuario nao encontrado");
        return;
      }
      const dados = await buscarDesempenhoGeral(user.id);
      const dadosIA = dados.avaliacoesPlataforma;
      const clientesPorMes: Record<string, number[]> = {};
      dados.avaliacoesClientes.forEach((av: { data: string; nota: number }) => {
        const mes = av.data?.substring(0, 7) || "2025-09";
        if (!clientesPorMes[mes]) clientesPorMes[mes] = [];
        clientesPorMes[mes].push(av.nota);
      });
```

```
      const dadosClientes = Object.entries(clientesPorMes).map(([mes,
          notas]) => ({
        periodoReferencia: mes + "-01",
        mediaClientes: notas.reduce((a, b) => a + b, 0) / notas.length,
      }));
      const tabelaResumo: LinhaResumo[] = dadosIA.map((ia: any) => {
        const clientes = dadosClientes.find((c) => c.periodoReferencia
            === ia.periodoReferencia);
        return {
          periodo: ia.periodoReferencia,
          notaIA: ia.notaFinal.toFixed(2),
          mediaClientes: clientes ? clientes.mediaClientes.toFixed(2)
              : "-",
        };
      });
      const container = document.createElement("div");
      container.style.width = "800px";
      container.style.height = "400px";
      container.style.position = "absolute";
      container.style.top = "-9999px";
      document.body.appendChild(container);
      const grafico = (
        <ResponsiveContainer width={800} height={400}>
          <LineChart>
            <CartesianGrid strokeDasharray="3 3" />
            <XAxis dataKey="periodoReferencia" />
            <YAxis domain={[0, 5]} />
            <Tooltip />
            <Legend />
            <Line
              type="monotone"
              dataKey="notaFinal"
              stroke="#000"
              strokeWidth={3}
              name="Nota Final (IA)"
              data={dadosIA}
            />
            <Line
              type="monotone"
              dataKey="mediaClientes"
              stroke="#82ca9d"
              strokeWidth={3}
              name="Media dos Clientes"
              data={dadosClientes}
            />
          </LineChart>
        </ResponsiveContainer>
      );
      const root = createRoot(container);
      root.render(grafico);
      await new Promise((resolve) => setTimeout(resolve, 1200));
      const canvas = await html2canvas(container);
      const imgData = canvas.toDataURL("image/png");
      const pdf = new jsPDF("landscape");
      const hoje = new Date().toLocaleDateString("pt-BR");
      pdf.setFontSize(18);
```

```
      pdf.text("Relatorio - Desempenho Geral", 15, 15);
      pdf.setFontSize(12);
      pdf.text(`Gerado em: ${hoje}`, 15, 25);
      pdf.text(`Prestador: ${user?.nome}`, 15, 32);
      pdf.setFontSize(11);
      pdf.text(
        "Este relatorio compara a Nota Final atribuida pela IA com a media das notas recebidas\n" +
        "dos clientes ao longo do tempo. O objetivo e mostrar convergencias ou diferencas\n" +
        "entre a avaliacao automatica da plataforma e a percepcao real dos clientes.",
        15,
        45,
        { maxWidth: 260 }
      );
      pdf.addImage(imgData, "PNG", 15, 70, 260, 120);
      autoTable(pdf, {
        startY: 200,
        head: [["Periodo", "Nota Final (IA)", "Media Clientes"]],
        body: tabelaResumo.map((linha: LinhaResumo) => [
          linha.periodo,
          linha.notaIA,
          linha.mediaClientes,
        ]),
      });
      const yAfterTable = (pdf as any).lastAutoTable.finalY + 10;
      pdf.text(
        "Legenda: Nota Final (IA)     avaliacao automatica da plataforma.\n" +
        "Media Clientes     percepcao dos clientes reais em cada periodo.",
        15,
        yAfterTable
      );
      const ultimaNotaIA = tabelaResumo[tabelaResumo.length - 1]?.notaIA;
      pdf.text(
        `Conclusao: Sua ultima nota da IA foi ${ultimaNotaIA}, ` +
        `${parseFloat(ultimaNotaIA) >= 4 ? "indicando bom desempenho" : "mostrando que ha pontos a melhorar"}.`,
        15,
        yAfterTable + 20
      );
      pdf.save("desempenho-geral.pdf");
      root.unmount();
      container.remove();
    } catch (err) {
      console.error(err);
      alert("Erro ao gerar PDF do Desempenho Geral");
    }
  };
```

Implementação 18. Trecho do código do *front-end* que faz a montagem dos relatórios que são disponibilizados para *download*