Projeto Pronto Afeto

Trabalho de Conclusão de Curso

Rebeca Vitória de Aguiar Coutinho

Frederico Jorge Ribeiro Barboza

Orientador

Instituto Federal da Bahia – IFBA

Curso de Análise e Desenvolvimento de Sistemas

Campus Salvador

Salvador, Bahia, Brasil

Fevereiro 2025

SUMÁRIO

1. Visão Geral

1.1 Declaração do Problema

O cuidado assistido, seja em ambiente domiciliar ou institucional, exige interação constante entre cuidadores, familiares e equipes administrativas. No entanto, a ausência de ferramentas digitais integradas e intuitivas compromete a eficiência dessa comunicação, sobretudo quando o acompanhamento ocorre a distância. Essa carência tecnológica resulta em fragmentação de informações, falta de transparência e baixa confiabilidade na gestão dos cuidados prestados.

O projeto propõe-se a mitigar essas limitações por meio da criação de uma camada de apresentação web que unifique os principais fluxos de interação entre os três perfis de usuários. Embora a lógica de negócio e o armazenamento de dados já estejam estruturados na API REST desenvolvida por Moreira (2024), a ausência de uma interface gráfica limita o alcance prático e a adoção do sistema por usuários não técnicos. Sem essa camada visual, cuidadores enfrentam dificuldades para registrar atividades ou atualizar prontuários, clientes têm restrições no acompanhamento das rotinas de cuidado e funcionários carecem de ferramentas que facilitem o monitoramento e a administração do serviço.

Em síntese, a falta de uma interface funcional e acessível representa a principal limitação do sistema existente, afetando diretamente sua usabilidade, transparência operacional e eficiência no gerenciamento do cuidado. O desenvolvimento do front-end proposto busca atender às necessidades identificadas, oferecendo uma experiência de uso intuitiva e validada por papéis, capaz de traduzir a complexidade da API em fluxos claros e responsivos.

De forma mais específica, a aplicação visa:

- Melhorar a clareza e acessibilidade das operações, por meio de telas de cadastro, autenticação e gerenciamento de propostas com validações visuais e feedbacks imediatos;

- Promover maior transparência, permitindo que clientes acompanhem de forma imediata o andamento dos cuidados e que cuidadores mantenham registros organizados e historicamente acessíveis;

- Otimizar a comunicação entre usuários, por meio de notificações visuais e estados de status que indiquem mudanças relevantes, como aprovações, atualizações de prontuário e encerramento de serviços;

- Apoiar a gestão administrativa, oferecendo aos funcionários ferramentas visuais de controle de cadastros, monitoramento de propostas e visualização de métricas de desempenho.

Dessa forma, o problema central abordado neste projeto é a inexistência de uma interface unificada e funcional que permita a utilização plena da API *Pronto Afeto*, comprometendo o potencial do sistema de promover um acompanhamento de cuidados mais confiável, ágil e acessível.

1.2 Proposta de Solução de Software

A solução proposta consiste no desenvolvimento de uma aplicação web front-end responsiva que funcione como a camada de apresentação da API REST Pronto Afeto, permitindo a interação fluida entre os perfis de Cuidador, Cliente e Funcionário. A aplicação foi planejada para traduzir a complexidade da lógica de negócio em interfaces visuais organizadas, acessíveis e adaptadas ao contexto de uso de cada perfil, promovendo uma experiência clara, responsiva e orientada a papéis.

O sistema será implementado utilizando o framework Next.js, aliado ao TypeScript e à biblioteca de componentes shadcn/ui. Essa combinação tecnológica possibilita a criação de uma aplicação modular, tipada e escalável, além de garantir consistência visual e acessibilidade. O Next.js oferece uma estrutura robusta para aplicações React modernas, com suporte nativo a diferentes estratégias de renderização, incluindo o Server-Side Rendering (SSR), que permite o pré-carregamento das páginas no servidor, reduzindo o tempo de renderização inicial e melhorando o desempenho percebido pelo usuário (VERCEL, 2025).

A utilização do TypeScript contribui para maior segurança no desenvolvimento, por meio de tipagem estática, detecção antecipada de erros e melhor organização estrutural do código, favorecendo a manutenção do sistema (MICROSOFT, 2025).

O uso do SSR no Next.js é especialmente relevante para aplicações que demandam melhor performance inicial e otimização para mecanismos de busca, pois o conteúdo é entregue já processado ao cliente, reduzindo o tempo até a primeira renderização significativa (VERCEL, 2025; PRATA, 2025).

A aplicação atuará em comunicação direta com a API REST, responsável por fornecer os dados e executar as operações de negócio. Essa integração permitirá que a aplicação de apresentação consuma endpoints voltados para autenticação, cadastro de cuidadores e clientes, gerenciamento de propostas, registro de cuidados, emissão de notificações e acompanhamento administrativo. A comunicação seguirá o padrão HTTP/HTTPS com formato JSON, garantindo compatibilidade e segurança na troca de informações.

Do ponto de vista funcional, o sistema será dividido em três interfaces principais, cada uma projetada para atender às demandas específicas de um perfil de usuário:

- Cuidador: interface voltada ao registro de atividades, acompanhamento de tratamentos e atualização de prontuários;

- Cliente (ou familiar): interface voltada ao acompanhamento das rotinas de cuidado, com acesso a históricos, notificações e status de execução;

- Funcionário (administrador): interface voltada à gestão do sistema, incluindo controle de cadastros, aprovação de propostas, visualização de métricas e auditoria de ações.

A proposta de solução prioriza a clareza dos fluxos de navegação, a redução do retrabalho e o reforço da transparência nas interações, permitindo que cada usuário tenha acesso apenas às funcionalidades pertinentes ao seu papel. Além disso, a estrutura modular adotada facilita futuras expansões do sistema, como a integração de novos módulos ou dispositivos, preservando a coesão visual e a estabilidade da aplicação.

Em síntese, a aplicação web proposta busca operacionalizar as funcionalidades já disponíveis na API Pronto Afeto, transformando-as em uma experiência interativa, segura e eficiente. Essa abordagem garante não apenas a utilização prática do sistema, mas também a consolidação de uma plataforma tecnológica capaz de apoiar cuidadores e familiares na gestão do cuidado assistido de forma mais integrada e confiável.

1.3 Tecnologias Adotadas

O desenvolvimento do front-end utiliza um conjunto de tecnologias modernas e consolidadas no ecossistema JavaScript, selecionadas com base em critérios de desempenho, acessibilidade, manutenibilidade e padronização visual. As principais tecnologias adotadas são descritas a seguir.

1. **Next.js:**
   O Next.js é o framework principal utilizado na construção da aplicação. Baseado em React, ele fornece uma estrutura completa para o desenvolvimento de interfaces dinâmicas e escaláveis. Seu suporte a Server-Side Rendering (SSR) e Static Site Generation (SSG) permite que as páginas sejam renderizadas no servidor antes de serem enviadas ao navegador, reduzindo o tempo de carregamento inicial, otimizando o desempenho e melhorando a indexação em mecanismos de busca.
   Além disso, o framework disponibiliza recursos como rotas dinâmicas, otimização de imagens, cache inteligente e integração simplificada com APIs externas. No contexto deste projeto, o Next.js atua como base estrutural do front-end, responsável por consumir a API REST Pronto Afeto API e centralizar as interações entre os perfis de Cuidador, Cliente e Funcionário (VERCEL, 2025).

2. **TypeScript:**
   O TypeScript adiciona tipagem estática ao JavaScript, tornando o código mais seguro e previsível. Sua adoção visa aumentar a robustez, legibilidade e confiabilidade do projeto, prevenindo erros durante o desenvolvimento e facilitando a manutenção e evolução do sistema. O uso de interfaces e tipos definidos favorece a escalabilidade e reduz inconsistências em aplicações de grande porte (MICROSOFT, 2025).

3. **Tailwind CSS:**
   O Tailwind CSS é o framework utilitário de estilização utilizado na construção da camada visual. Ele permite a criação de interfaces modernas e responsivas a partir de classes utilitárias, eliminando a necessidade de folhas de estilo extensas. Essa

abordagem favorece a consistência visual, reduz o acoplamento entre estilo e lógica e otimiza o desempenho, uma vez que apenas as classes efetivamente utilizadas são incluídas no build final (TAILWIND LABS, 2025).

4. **shadcn/ui:**
   A biblioteca shadcn/ui foi adotada para padronizar os componentes de interface e garantir uma identidade visual coesa em toda a aplicação. Construída sobre o Tailwind CSS e baseada em componentes acessíveis da Radix UI, ela oferece uma ampla gama de elementos prontos, como botões, modais, formulários e tabelas, que podem ser facilmente personalizados para manter a coerência estética e funcional da aplicação (SHADCN, 2025).

5. **Zod:**
   O Zod é a biblioteca utilizada para validação de esquemas e tipagem em tempo de execução, assegurando a integridade dos dados manipulados no front-end. Ele é aplicado na validação de formulários e na comunicação com a API, garantindo que as informações enviadas e recebidas estejam no formato esperado antes do processamento. Essa prática reduz falhas lógicas e melhora a confiabilidade da aplicação (MCKENZIE, 2025).

6. **Framer Motion:**

   A biblioteca Motion, anteriormente conhecida como Framer Motion, é utilizada para a implementação de animações e transições no front-end da aplicação. Integrada de forma nativa ao ecossistema React, ela permite a criação de interações visuais fluidas e declarativas, contribuindo para uma experiência do usuário mais intuitiva e agradável. Por meio de animações de entrada, saída e feedback visual em ações do usuário, o Motion auxilia na comunicação do estado da interface, reduzindo a sensação de latência e aumentando a usabilidade do sistema.

   No contexto do projeto Pronto Afeto, a biblioteca é aplicada para enriquecer a navegação entre páginas, destacar mudanças de estado em componentes e melhorar a percepção de continuidade nas interações dos perfis de Cuidador, Cliente e Funcionário, sem comprometer o desempenho da aplicação (FRAMER, 2025).

Em conjunto, essas tecnologias fornecem uma base consistente para o desenvolvimento da aplicação Pronto Afeto. O front-end atua como camada de apresentação do sistema, comunicando-se diretamente com a API REST Pronto Afeto, e traduzindo suas funcionalidades em uma experiência visual fluida e acessível para os diferentes perfis de usuários.

1.4 Trabalhos Relacionados

A revisão de trabalhos relacionados permite compreender o estado atual das soluções voltadas ao cuidado assistido e identificar as lacunas que justificam o desenvolvimento da aplicação Pronto Afeto. Nesta seção, são destacados dois estudos internacionais que abordam o uso de tecnologias digitais em contextos de cuidado remoto, além do projeto Pronto Afeto API (MOREIRA, 2024), que constitui a base lógica e de persistência deste trabalho.

O estudo de Queirós et al. (2018), intitulado *Remote Care Technology: A Systematic Review of Reviews and Meta-Analyses*, apresenta uma revisão sistemática sobre o uso de tecnologias digitais aplicadas ao cuidado de pacientes com doenças crônicas. Os autores destacam que, embora exista um crescimento expressivo de plataformas voltadas ao monitoramento remoto, muitas delas ainda carecem de interfaces centradas no usuário e de integração eficiente entre cuidadores, familiares e profissionais de saúde. O estudo conclui que o êxito dessas soluções depende não apenas da automação dos processos, mas também da usabilidade e acessibilidade das interfaces, fatores essenciais para a adesão dos usuários. Esses apontamentos reforçam a motivação deste projeto, que busca traduzir os fluxos de cuidado assistido em uma camada de apresentação clara, responsiva e intuitiva, construída sobre uma base de dados e lógica de negócio previamente estruturadas.

Já o trabalho de Sriram et al. (2022), *Carers using assistive technology in dementia care at home*, investiga o impacto do uso de tecnologias assistivas por cuidadores de pessoas com demência em ambientes domiciliares. O estudo evidencia benefícios como a redução do estresse do cuidador e o aumento da segurança do paciente, mas também ressalta limitações associadas à complexidade de uso e à falta de padronização das interfaces. Essa observação reforça a necessidade de soluções que priorizem a simplicidade de interação e a personalização das funcionalidades conforme o perfil do usuário — princípios adotados no desenvolvimento da interface Pronto Afeto, que oferece fluxos específicos para cuidadores, clientes e funcionários administrativos.

Complementarmente, o trabalho de Moreira (2024), intitulado *Projeto Pronto Afeto API*, apresenta a camada de persistência e de negócio do sistema de cuidado assistido. Desenvolvida em arquitetura REST, essa API estrutura os processos de autenticação, registro de usuários, gerenciamento de propostas, controle de cuidados e armazenamento das informações sensíveis do sistema. Embora a solução entregue a base lógica e os contratos necessários para o funcionamento do sistema, o próprio autor reconhece a ausência de uma camada de apresentação capaz de disponibilizar essas funcionalidades de forma acessível e orientada ao uso cotidiano. O presente trabalho surge, portanto, como continuação direta dessa proposta, fornecendo a interface visual que operacionaliza e amplia o potencial de aplicação prática da API desenvolvida.

Em síntese, enquanto os estudos de Queirós et al. (2018) e Sriram et al. (2022) demonstram a relevância social e tecnológica do cuidado assistido, o TCC de Moreira (2024) oferece a infraestrutura técnica sobre a qual esta pesquisa se apoia. A contribuição deste projeto reside na implementação da camada front-end do sistema Pronto Afeto, responsável por traduzir a lógica de negócio existente em uma experiência de uso acessível, responsiva e validada por papéis, consolidando uma solução completa e integrada para o gerenciamento digital do cuidado assistido.

2. Requisitos

2.1 Requisitos Funcionais

Nº	Requisito Funcional	Critérios de Aceite	Observação
RF01	Permitir o cadastro de cuidador com dados pessoais, profissionais e documentos obrigatórios.	Validar campos obrigatórios (nome, CPF, e-mail, telefone, endereço, senha); exibir checkbox obrigatório de aceite dos termos; apresentar mensagens de erro ou sucesso após envio.	Status inicial “Pendente de Aprovação” é definido pela API.
RF02	Permitir login do cuidador utilizando e-mail e senha.	Validar formato dos campos; exibir mensagens informativas em caso de erro; redirecionar para o painel do cuidador após autenticação.	Autenticação e token são tratados pela API.
RF03	Permitir atualização do perfil do cuidador autenticado.	Exibir tela de edição com validação de campos obrigatórios; permitir salvar alterações com feedback visual; bloquear envio se houver campos inválidos.	Persistência e autorização são geridas pela API.
RF04	Permitir que o cuidador visualize e gerencie propostas de serviço recebidas.	Exibir lista de propostas vinculadas ao cuidador; permitir aceitar ou recusar proposta com confirmação visual; atualizar status exibido após resposta.	Atualização real e vínculo das propostas são controlados pela API.
RF05	Permitir o registro de observações e atualizações sobre cuidados ativos.	Exibir campo de texto para observações; validar preenchimento antes do envio; atualizar a lista de registros após submissão.	Controle de prontuários e histórico é da API.

RF06	Permitir cadastro de cliente com dados pessoais, e-mail e senha.	Validar formato de CPF, e-mail e senha; exibir confirmação visual após cadastro; mostrar mensagem de erro em caso de duplicidade.	Verificação de unicidade e envio de e-mail são feitos pela API.
RF07	Permitir login do cliente autenticado.	Validar campos antes do envio; exibir mensagem apropriada em caso de erro; redirecionar para o painel principal após sucesso.	Autenticação é realizada pela API.
RF08	Permitir cadastro de cuidados vinculados ao cliente autenticado.	Validar campos obrigatórios (tipo de cuidado, endereço, descrição); exibir confirmação visual após envio; atualizar lista local.	Associação ao cliente e persistência ocorrem na API.
RF09	Permitir envio de proposta de serviço.	Validar campos antes do envio; exibir mensagem de confirmação ou erro; atualizar a lista de propostas do cliente.	Notificações e status “Aguardando Aprovação” são definidos pela API.
RF10	Permitir acompanhamento das propostas enviadas.	Exibir histórico de propostas com status atualizados; permitir atualização automática ou manual da lista.	Dados e status vêm da API.
RF11	Permitir visualização de cuidados e prontuários.	Exibir histórico cronológico; permitir visualização de detalhes; ocultar dados sensíveis conforme configuração.	Controle de acesso e sigilo pertencem à API.
RF12	Exibir notificações sobre atualizações em propostas e cuidados.	Exibir notificações push no navegador; registrar histórico de notificações; alertar em tempo de execução.	Envio de e-mails e origem das notificações são da API.
RF13	Permitir que o cliente avalie o cuidador após conclusão do serviço.	Disponibilizar formulário com nota e comentário; validar campos antes do envio; exibir mensagem de confirmação.	Cálculo da média e restrição de elegibilidade são da API.

RF14	Permitir atualização de informações pessoais do cliente autenticado.	Validar formato de e-mail, telefone e endereço; exibir confirmação após salvar alterações.	Persistência e unicidade são da API.
RF15	Permitir cancelamento de proposta ou cuidado.	Exibir botão de cancelamento com mensagem de confirmação; atualizar interface após resposta da operação.	Regras de cancelamento e verificação de status são da API.
RF16	Permitir login do funcionário na área administrativa.	Validar campos antes do envio; redirecionar para o painel após autenticação.	Controle de perfis e permissões é feito pela API.
RF17	Permitir aprovação de cadastros de cuidadores pendentes.	Exibir lista de cuidadores pendentes; permitir aprovação com confirmação visual; exibir feedback de sucesso.	Atualização de status e envio de notificação são feitos pela API.
RF18	Permitir gerenciamento e alteração de status de propostas.	Exibir tabela de propostas com filtros e colunas de status; permitir alteração com justificativa; exibir mensagem de confirmação.	Persistência das alterações e regras de negócio são da API.
RF19	Permitir visualização do histórico de cuidadores e suas atividades.	Exibir lista com filtros por status, cliente e data; exibir detalhes sob demanda.	Controle de acesso e dados são fornecidos pela API.
RF20	Permitir rejeição de cadastros de cuidadores.	Exibir botão de rejeição com campo para justificativa; atualizar status visualmente após ação.	Registro de data/hora e notificação são da API.
RF21	Permitir cadastro de novos funcionários administrativos.	Exibir formulário para inserção dos dados; validar campos obrigatórios; exibir mensagem de confirmação.	Criação de credenciais e permissões são da API.

2.2 Requisitos Não-Funcionais

Nº	Requisito Não Funcional	Descrição
RNF01	Desempenho de Carregamento	A aplicação deve exibir conteúdo interativo inicial em menos de 2 segundos em conexões de banda larga comum, para as páginas principais de Cliente, Cuidador e Funcionário, utilizando renderização no servidor (SSR) para reduzir o tempo de resposta percebido pelo usuário.
RNF02	Responsividade	A interface deve ser totalmente responsiva, adaptando-se a diferentes resoluções (desktop, tablet e dispositivos móveis), sem perda de funcionalidade ou necessidade de zoom manual.
RNF03	Acessibilidade	As páginas e componentes devem seguir boas práticas de acessibilidade Web responsiva (uso adequado de rótulos, hierarquia semântica e foco navegável), permitindo navegação por teclado e leitura assistida em elementos essenciais como formulários e botões de ação.
RNF04	Consistência Visual	A aplicação deve manter padronização visual em toda a interface por meio do uso de componentes reutilizáveis construídos com Tailwind CSS e shadcn/ui, garantindo que botões, formulários, tabelas e modais mantenham o mesmo estilo, espaçamento e tipografia.
RNF05	Segurança de Sessão	O front-end deve garantir que apenas usuários autenticados tenham acesso às áreas restritas, ocultando funcionalidades e rotas protegidas quando não houver sessão válida retornada pela API.
RNF06	Tratamento de Erros	A aplicação deve exibir mensagens claras e compreensíveis para erros de validação de formulário, falhas de autenticação e recusas de acesso, sem expor detalhes técnicos internos da API para o usuário final.
RNF07	Validação de Dados no Cliente	Todos os formulários críticos (cadastro, login, propostas, registros de cuidado) devem realizar validação no cliente utilizando esquemas definidos com Zod, impedindo o envio de dados em formato inválido para a API.
RNF08	Usabilidade	As telas devem apresentar fluxos lineares e guiados, com feedback visual imediato após cada ação do usuário (ex.: confirmação de envio, estado “carregando”, sucesso ou falha), reduzindo ambiguidade nas operações de cadastro, proposta ou atualização de prontuário.

RNF09	Manutenibilidade do Código	O código do front-end deve ser modular e escrito em TypeScript, utilizando componentes reutilizáveis e tipados, de forma a facilitar manutenção, evolução e correção de falhas sem impacto em outras áreas da interface.
RNF10	Testabilidade	Os principais componentes e fluxos de uso (autenticação, cadastro, envio de proposta, registro de prontuário) devem possuir testes automatizados com Jest e Testing Library, permitindo a verificação de comportamento sem interação manual.
RNF11	Compatibilidade de Navegador	A aplicação deve ser compatível com as versões suportadas dos navegadores modernos (Chrome, Edge e Firefox), garantindo a execução correta das telas e interações sem a necessidade de plugins adicionais.
RNF12	Proteção de Dados Sensíveis	Campos que exibem informações sensíveis de saúde, dados pessoais de Cliente e dados profissionais de Cuidador devem ser renderizados apenas para perfis autorizados, evitando a exposição visual desses dados em telas públicas ou compartilháveis (ex.: telas sem login).

3. Design

3.1 Projeto UML

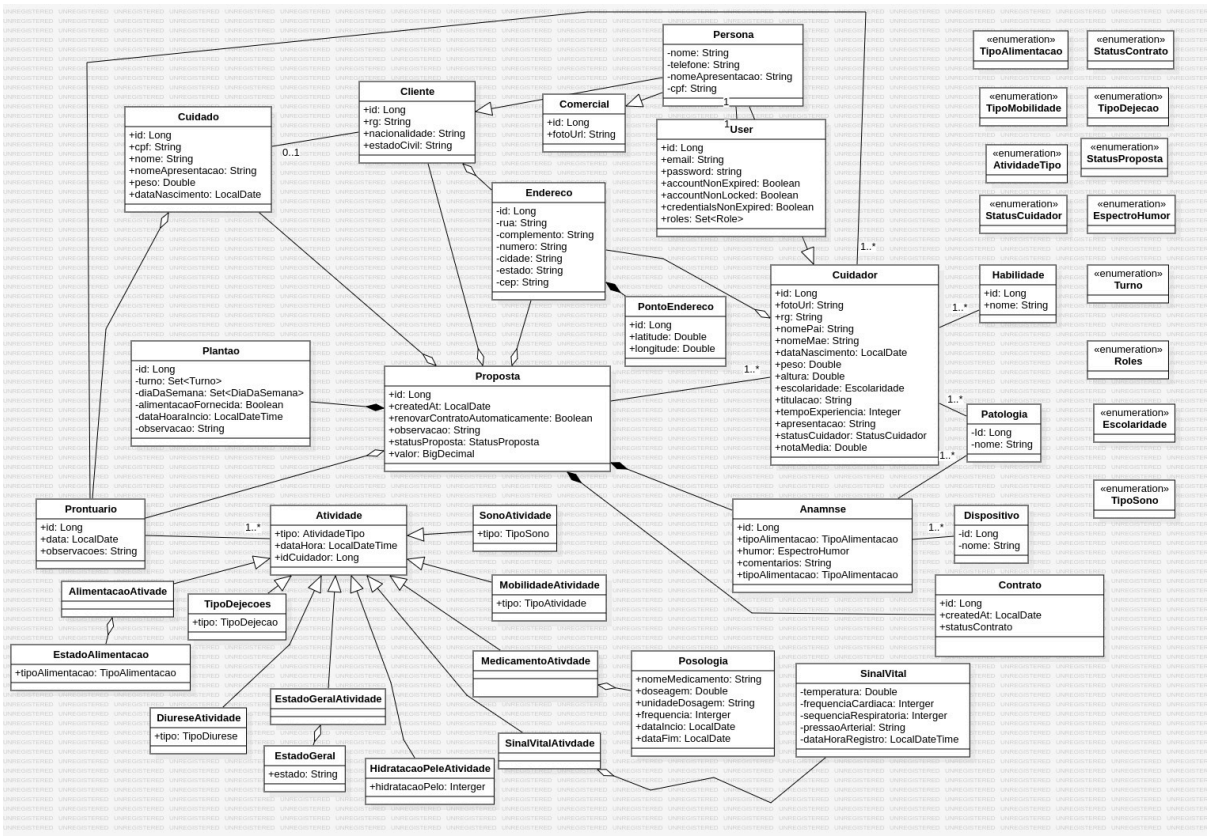

Figura 1: Diagrama de classes

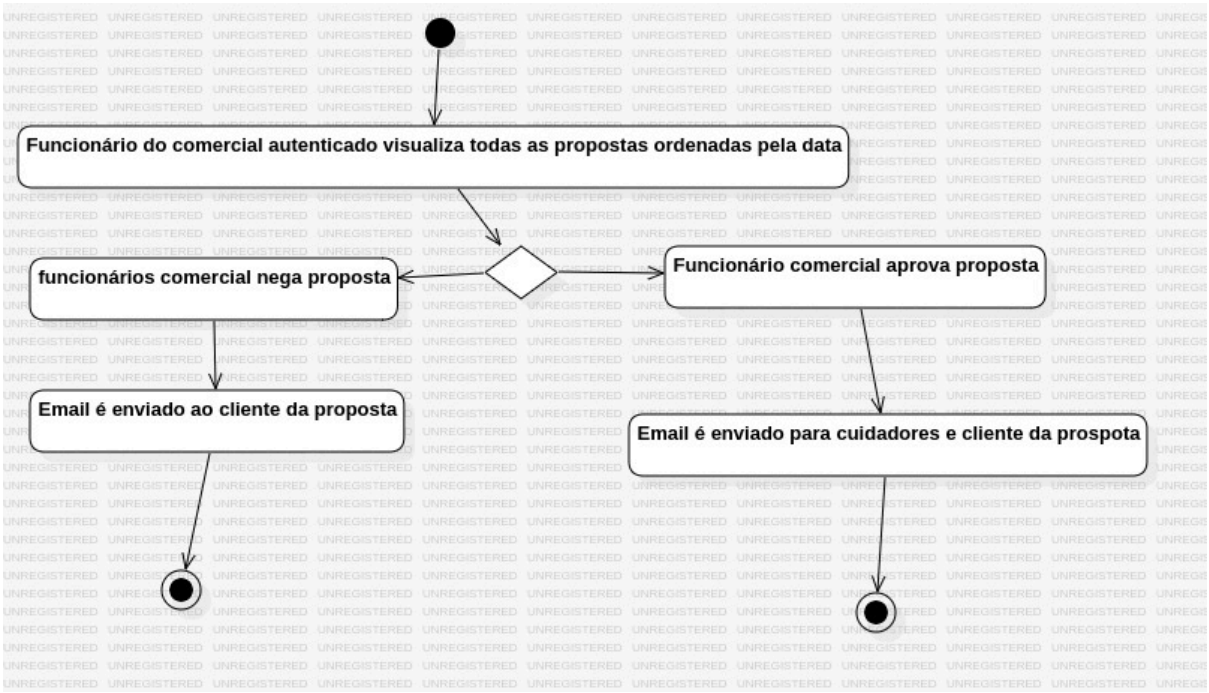

Figura 2: Diagrama de Atividades Comercial

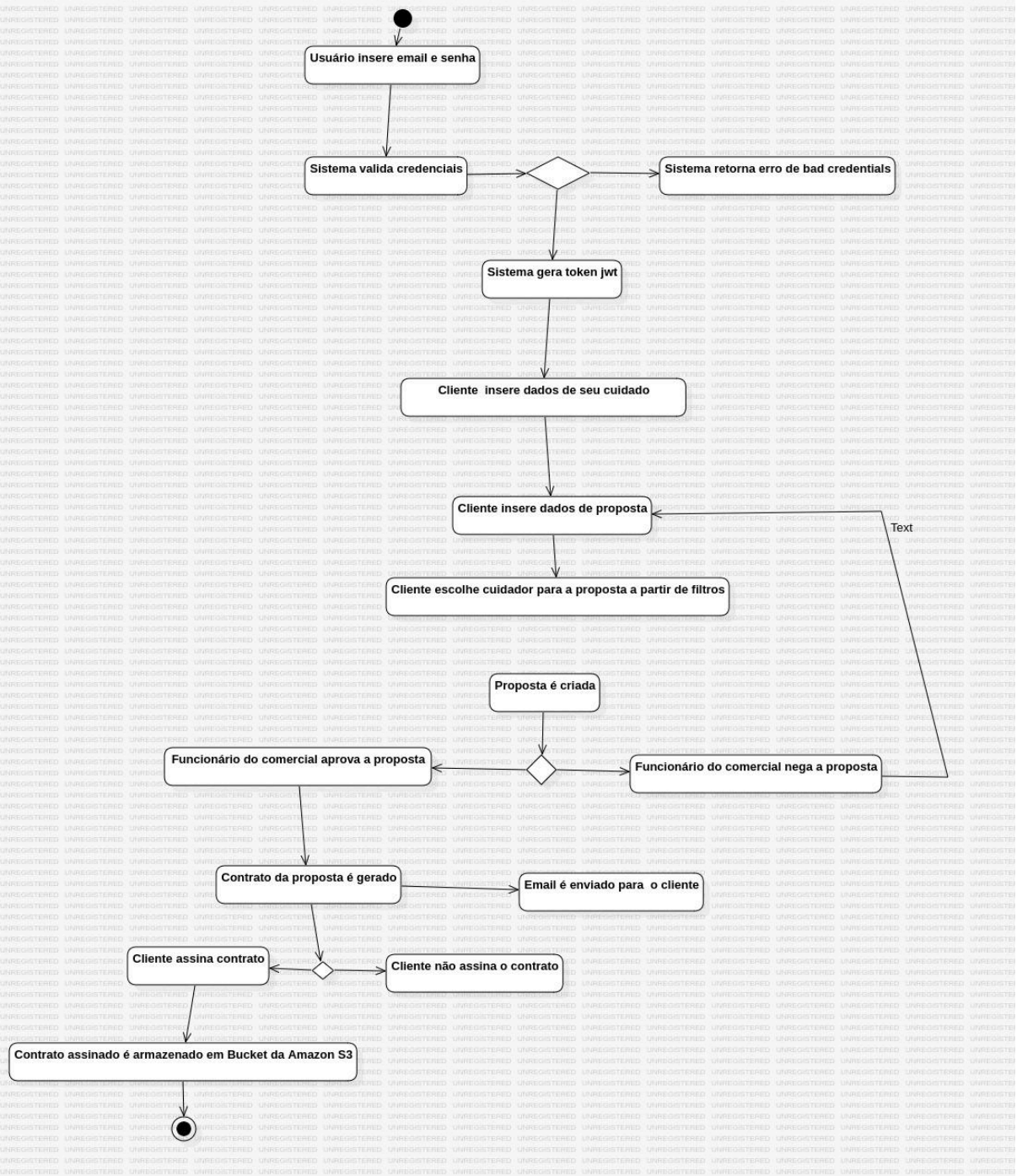

Figura 3: Diagrama de Atividades Cliente

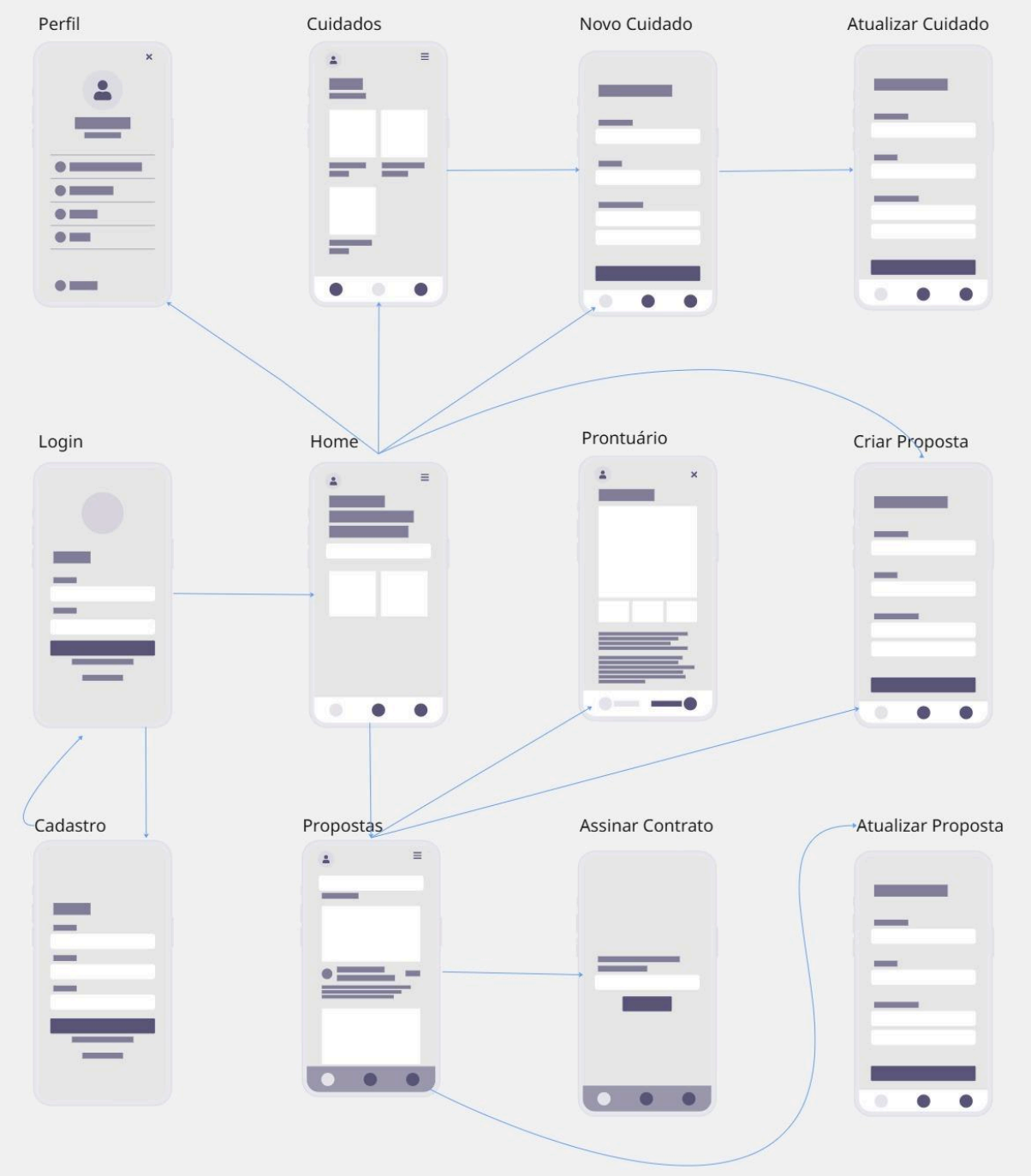

Figura 4: Diagrama de Navegação Cliente

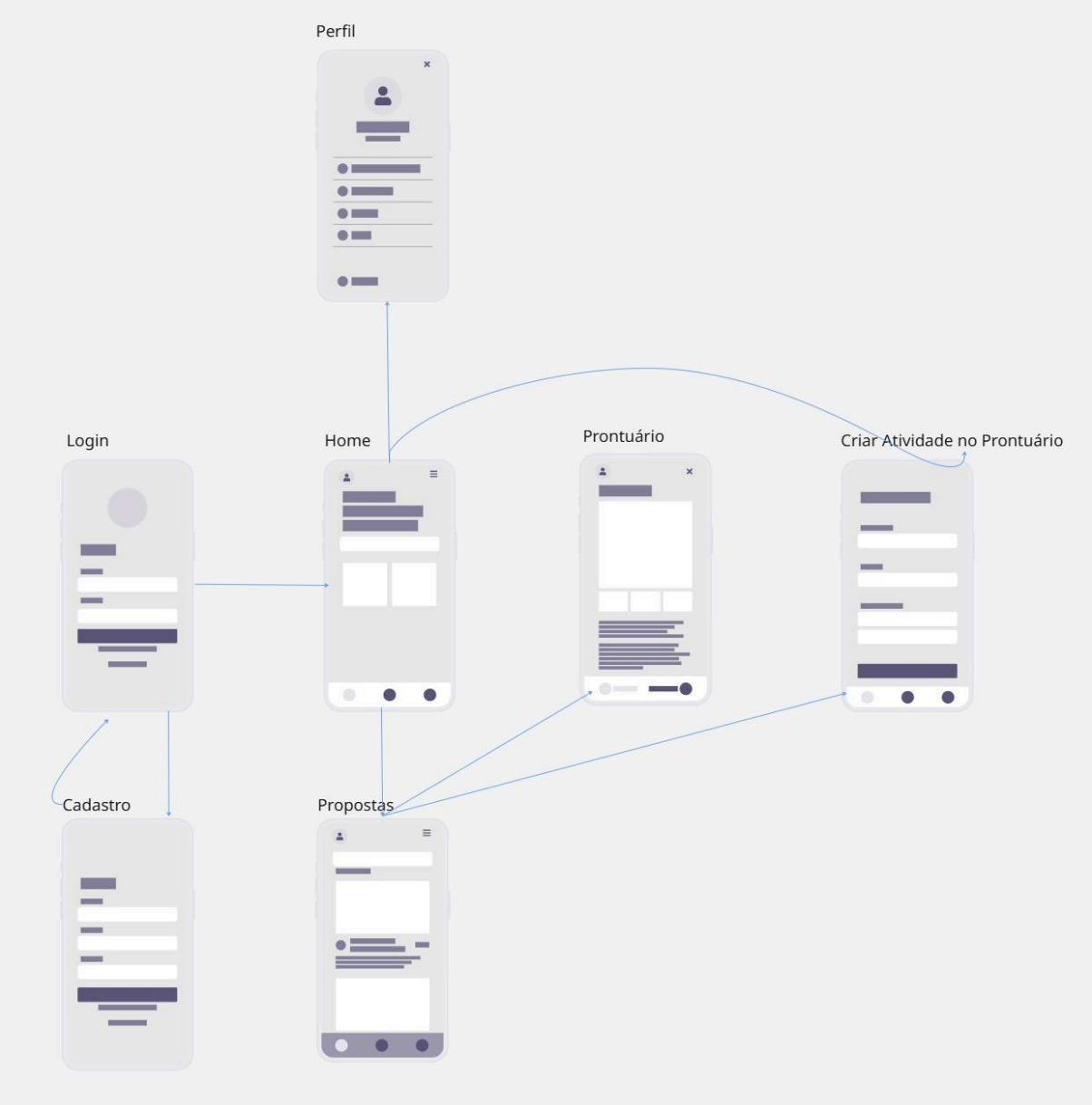

Figura 5: Diagrama de Navegação Cuidador

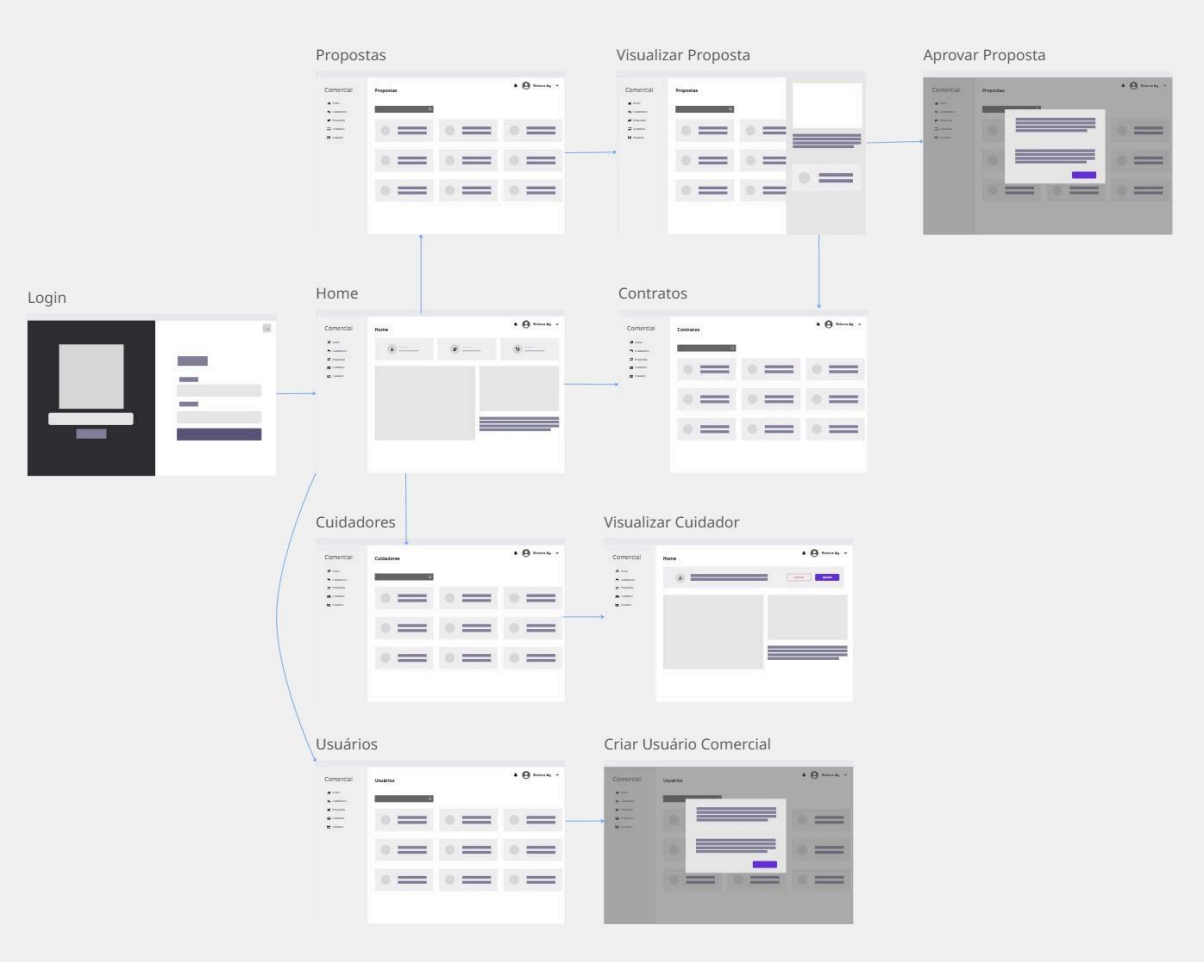

Figura 6: Diagrama de Navegação Funcionário

3.2 Visão Arquitetural

O sistema Pronto Afeto organiza-se em três aplicações front-end independentes, voltadas aos perfis Cliente, Cuidador e Funcionário, que consomem uma única API REST responsável pelas regras de negócio e pela persistência dos dados. Cada front-end possui ciclo próprio de implantação, o que permite evoluir funcionalidades, interface e fluxo de navegação de forma autônoma para cada perfil de uso, reduzindo o acoplamento entre as interfaces e facilitando a manutenção do ecossistema como um todo. Essa opção dialoga com abordagens contemporâneas de arquitetura de front-end modular, que recomendam a decomposição da interface em unidades menores e coesas em vez de um único front-end monolítico (FRONT-END ARCHITECTURE, 2024; FOWLER, 2019).

A Pronto Afeto API constitui a camada central de backend da solução e foi desenvolvida por Moreira (2024) como parte do Trabalho de Conclusão de Curso intitulado *Projeto Pronto Afeto API*. Essa API é responsável por concentrar as regras de negócio, a validação de dados e a orquestração dos casos de uso do sistema. Sua adoção como serviço intermediário entre as aplicações de apresentação e o banco de dados permite a separação clara de responsabilidades, promovendo baixo acoplamento entre as interfaces e o domínio da aplicação.

A comunicação entre as três aplicações de apresentação e a Pronto Afeto API ocorre por meio de contratos HTTP com troca de dados em formato JSON. Esses contratos são documentados e versionados, definindo recursos, parâmetros e estruturas de resposta de forma explícita, o que possibilita a inclusão de novos clientes e a evolução dos existentes com impacto controlado. A API assume o papel de fachada única para o domínio de negócio, centralizando o acesso às operações e garantindo consistência no tratamento das regras e validações.

No nível de estilo arquitetural, o ecossistema adota um modelo em camadas. A camada de apresentação é composta pelas três aplicações front-end, responsáveis pela interação com usuários de perfis distintos. A camada de negócio corresponde à Pronto Afeto API, que implementa autenticação, autorização e processamento das operações do sistema. A camada de persistência é materializada por um banco de dados relacional, responsável pelo armazenamento consistente de entidades como clientes, cuidadores, contratos, propostas e prontuários. Essa separação em camadas busca reduzir o acoplamento entre preocupações técnicas e de negócio e favorecer a evolução incremental dos componentes, em linha com discussões sobre modularização e segmentação de responsabilidades em arquiteturas de front-end e back-end. A Figura 7, Diagrama de Containers do Ecossistema Pronto Afeto, ilustra essa organização em alto nível e os fluxos de comunicação entre os principais componentes.

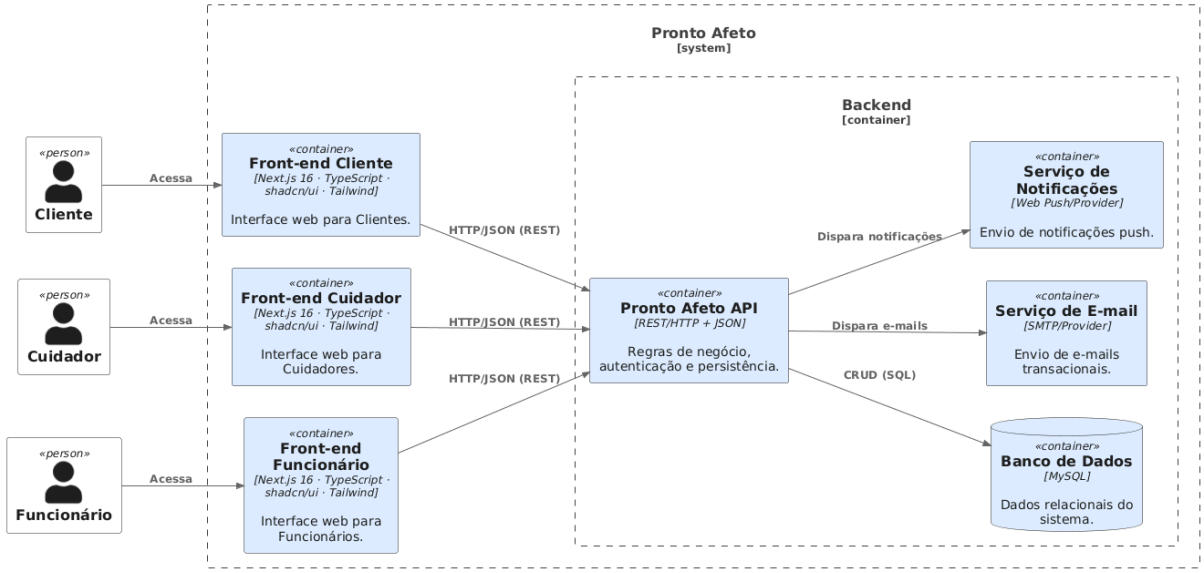

Figura 7: Diagrama de Containers do Ecossistema Pronto Afeto

3.3 Organização das Aplicações Front-end e Estrutura do Código

Esta seção aprofunda a visão arquitetural apresentada anteriormente, descrevendo como o front-end do Pronto Afeto foi organizado em três aplicações independentes e como o código de cada uma delas é estruturado em torno de domínios de negócio e módulos compartilhados. A intenção é explicitar decisões que favorecem modularidade, coesão e facilidade de evolução, em consonância com discussões recentes sobre arquitetura de front-end modular, que recomendam decompor interfaces extensas em unidades menores e mais focadas em vez de concentrar tudo em um único aplicativo monolítico (FRONT-END ARCHITECTURE, 2024; FOWLER, 2019).

Cada aplicação front-end do ecossistema Pronto Afeto é implementada com Next.js e organizada de modo a refletir os contextos de uso do respectivo perfil de usuário. A estrutura de pastas privilegia agrupamentos por domínio de negócio, de forma que fluxos relacionados, como cadastro de clientes, gestão de propostas ou acompanhamento de cuidados, se mantenham encapsulados em módulos próprios. Em paralelo, recursos compartilhados entre diferentes partes da interface são concentrados em módulos comuns, que funcionam como uma base reutilizável. Esse arranjo segue a orientação de organizar aplicações React modernas com foco em features, aproximando o código da linguagem do domínio e facilitando sua evolução ao longo do tempo (PRATA, 2025; FRONT-END ARCHITECTURE, 2024).

Além disso, busca-se manter um padrão estrutural uniforme entre as três aplicações front-end. A mesma lógica de organização por domínios de negócio e de uso de módulos compartilhados é replicada nos contextos de Cliente, Cuidador e Funcionário. Essa padronização reduz o custo de transição entre projetos, facilita o compartilhamento de práticas dentro da equipe e contribui para que o ecossistema evolua de maneira mais previsível e controlada, aspecto frequentemente destacado em propostas atuais de arquitetura de front-end modular.

3.3.1 Separação em três aplicações especializadas

A principal decisão de alto nível na organização do front-end do Pronto Afeto foi a opção por três aplicações independentes, uma para cada perfil de uso: Cliente, Cuidador e Funcionário. Em vez de concentrar todos os fluxos em um único front-end, cada aplicação aborda um recorte específico do domínio e expõe apenas as funcionalidades relevantes para o seu público alvo. Essa separação permite adequar linguagem, navegação e prioridades de interface a cada contexto, tornando a experiência mais alinhada às necessidades de quem contrata o serviço, de quem presta o cuidado e de quem realiza a gestão interna.

Do ponto de vista técnico e operacional, a divisão em três aplicações proporciona ciclos independentes de desenvolvimento e implantação. Atualizações em uma aplicação, como a introdução de um novo fluxo para cuidadores, podem ser planejadas e implantadas sem que isso implique necessariamente uma nova versão das interfaces de Cliente ou Funcionário. Esse isolamento reduz o risco de regressões cruzadas, facilita a gestão de releases e permite que cada aplicação avance em ritmos diferentes, o que é especialmente útil em cenários em que determinados perfis exigem mudanças mais frequentes. Estudos sobre arquitetura de front-end modular destacam justamente esses benefícios de decompor a interface em módulos ou aplicações menores, com fronteiras bem definidas e acoplamento reduzido (FRONT-END ARCHITECTURE, 2024).

Essa escolha também dialoga com a discussão sobre micro front-ends, que propõe a divisão de soluções de interface em partes independentes, cada uma responsável por uma área do domínio ou por um conjunto de funcionalidades (FOWLER, 2019). No entanto, para o contexto deste trabalho, optou-se por uma solução mais simples e adequada. Em vez de compor múltiplos micro front-ends dentro de um único shell, o Pronto Afeto adota três aplicações completas, cada uma consumindo a mesma API REST central. Dessa forma, o sistema se beneficia do princípio de decomposição e independência defendido por Fowler

(2019), sem incorporar a complexidade adicional de orquestração e integração típica de cenários de micro front-ends em larga escala.

Em síntese, a separação em três aplicações especializadas permite alinhar melhor os fluxos de interface aos perfis de usuário, isolar ciclos de entrega e, ao mesmo tempo, manter uma arquitetura coerente com tendências atuais de modularização do front-end. Essa decisão serve de base para a organização interna do código descrita nas próximas subseções, em que se detalham os subdomínios de negócio e os módulos compartilhados que compõem cada uma dessas aplicações.

3.3.2 Organização interna por domínios de negócio

No interior de cada aplicação front-end do Pronto Afeto, a estrutura do código foi organizada a partir de domínios de negócio, e não apenas por tipo técnico de arquivo. Em vez de concentrar todos os componentes em pastas globais, como `components` ou `services`, foram definidos submódulos que refletem capacidades específicas do sistema, como cadastro e gestão de clientes, acompanhamento de cuidados ou tratamento de propostas. Cada submódulo reúne os artefatos necessários para aquele recorte do domínio, aproximando a estrutura do código da forma como o problema é descrito pelos usuários e pela equipe.

Na aplicação Cliente, por exemplo, essa organização é representada por subdomínios como `client`, `care` e `proposal`, além de um conjunto de módulos compartilhados (`shared`). Cada subdomínio agrupa páginas e containers de interface, actions responsáveis por orquestrar chamadas à API, schemas de validação e tipos TypeScript associados àquele contexto. Os módulos compartilhados concentram componentes visuais reutilizáveis, hooks utilitários, serviços de acesso à API e tipos globais que dão suporte aos diferentes subdomínios. A Figura 8 apresenta, em nível de módulos, essa organização interna da aplicação Cliente, destacando as relações entre os subdomínios e os recursos compartilhados.

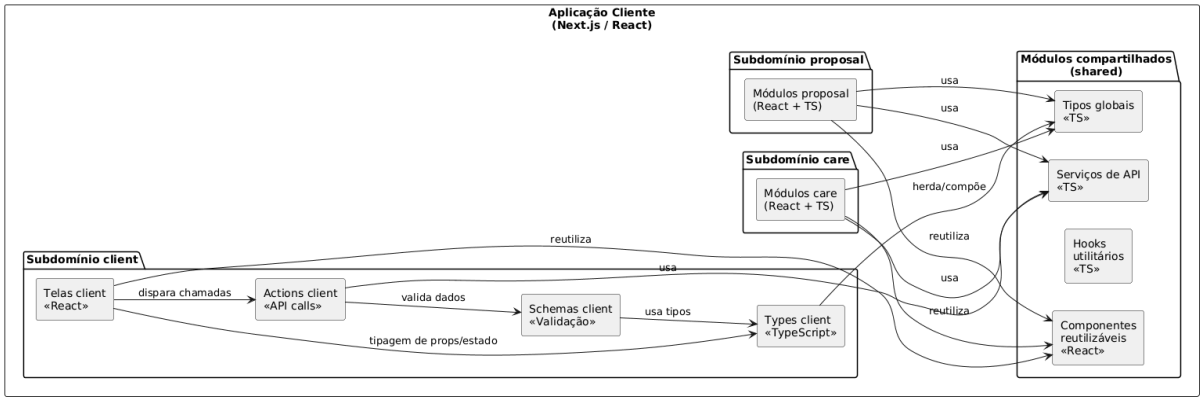

Figura 8: Estrutura de módulos da aplicação Cliente organizada por subdomínios de negócio

Essa abordagem corresponde a uma estrutura orientada a features, em que componentes, serviços e modelos são agrupados em torno de funcionalidades específicas, e não espalhados por camadas técnicas genéricas. A literatura recente sobre organização de

aplicações React aponta que esse tipo de arranjo aumenta a coesão interna de cada módulo, facilita a localização de funcionalidades e torna mais previsível a evolução da base de código em sistemas que tendem a crescer (PRATA, 2025). Estudos sobre arquitetura de front-end modular também ressaltam que o alinhamento entre a estrutura do código e o modelo de domínio contribui para reduzir acoplamento e melhorar a capacidade de evolução do sistema como um todo (FRONT-END ARCHITECTURE, 2024).

3.3.3 Módulos compartilhados e padronização entre aplicações

Além dos subdomínios específicos de cada aplicação, o front-end do Pronto Afeto conta com um conjunto de módulos compartilhados que concentram recursos transversais, utilizados em mais de um fluxo de negócio. Nesses módulos são reunidos componentes visuais reutilizáveis, hooks utilitários, serviços de acesso à API, funções auxiliares e tipos globais, que dão suporte às diferentes partes da interface. A ideia é evitar a duplicação de soluções semelhantes em cada subdomínio, oferecendo uma "caixa de ferramentas" comum a ser consumida pelos módulos de domínio, como ilustrado na Figura 8 para o caso da aplicação Cliente.

A dependência entre essas camadas é intencionalmente assimétrica: subdomínios de negócio dependem dos módulos compartilhados, mas os módulos compartilhados não dependem dos subdomínios. Essa restrição ajuda a manter uma direção clara de dependências e reduz o risco de acoplamentos cíclicos, que dificultariam a evolução do código. Na prática, isso significa que alterações em componentes ou serviços genéricos podem ser propagadas de forma controlada para os subdomínios consumidores, enquanto mudanças em um fluxo específico não obrigam a reestruturar o restante da base. Esse tipo de organização reforça o papel dos módulos compartilhados como infraestrutura de suporte, e não como repositório arbitrário de código reaproveitado.

Outro aspecto importante é que a mesma estratégia de separação entre subdomínios e módulos compartilhados é replicada nas três aplicações front-end: Cliente, Cuidador e Funcionário. Com isso, padrões de nomeação de pastas, localização de componentes, forma de declarar serviços de API e modelos de dados tendem a ser consistentes em todo o ecossistema. Essa padronização reduz o custo de transição entre projetos, facilita o trabalho colaborativo da equipe e contribui para que o front-end evolua de forma mais previsível, em linha com recomendações de estruturas orientadas a features que enfatizam a reutilização controlada de componentes e serviços em aplicações React de médio e grande porte (PRATA, 2025; FRONT-END ARCHITECTURE, 2024).

3.3 Modelo de Banco de Dados

O diagrama de banco de dados da Pronto Afeto representa um modelo relacional que sustenta as operações de contratação, acompanhamento de cuidados e registro clínico. As tabelas foram organizadas em grupos de entidades que refletem os principais processos do sistema, permitindo que a API forneça dados consistentes para os três front-ends web.

1. **Clientes e cuidadores**
   A tabela `clientes` guarda os dados pessoais e de endereço das pessoas que contratam o serviço. A tabela `cuidadores` registra informações sobre quem presta

o cuidado, incluindo experiência, habilidades e histórico de avaliação. A tabela de associação `clientes_cuidados` vincula clientes e cuidadores, registrando quais profissionais atendem cada pessoa.

2. **Contratos e propostas**
   A tabela `propostas` registra as propostas de prestação de serviço, contendo valores, datas de início e observações financeiras. Quando uma proposta é formalizada, seus dados são consolidados na tabela `contratos`. As tabelas `contrato_periodos` e `contrato_hora` detalham intervalos de tempo e horários acordados, enquanto `contrato_cuida` vincula cada contrato aos cuidadores responsáveis.

3. **Avaliações e prontuários**
   A tabela `avaliacoes` armazena avaliações feitas sobre os cuidadores, com campos para comunicação, empatia, pontualidade e outros critérios. Os atendimentos realizados são descritos em `prontuarios`, que registra os cuidados prestados para cada cliente. A tabela `prontuarios_cuida` estabelece a ligação entre cada prontuário e o cuidador que executou o atendimento.

4. **Habilidades e patologias**
   A tabela `habilidade` define competências específicas que um cuidador pode possuir. A relação entre cuidadores e essas competências é mantida em `cuidador_habilidades`. Já a tabela `patologias` descreve condições clínicas associadas às anamneses, e `cuidador_patologias` indica quais patologias cada cuidador está habituado a manejar.

5. **Endereços e localização**
   A tabela `enderecos` concentra as informações de localização de clientes e cuidadores. A tabela `ponto_endereco` registra coordenadas geográficas, permitindo o mapeamento espacial dos atendimentos e facilitando a busca por cuidadores próximos.

6. **Segurança e usuários**
   As tabelas `users` e `user_role` implementam o controle de autenticação e autorização. `users` armazena credenciais e dados de acesso, enquanto `user_role` define papéis e níveis de permissão, garantindo que cada perfil utilize apenas as funcionalidades previstas.

7. **Outras entidades de apoio**
   A tabela `anamnese` registra o estado de saúde e o histórico do cliente. A tabela `dispositivos` guarda informações sobre equipamentos utilizados no cuidado e `plantoes` organiza turnos e períodos de trabalho dos cuidadores.

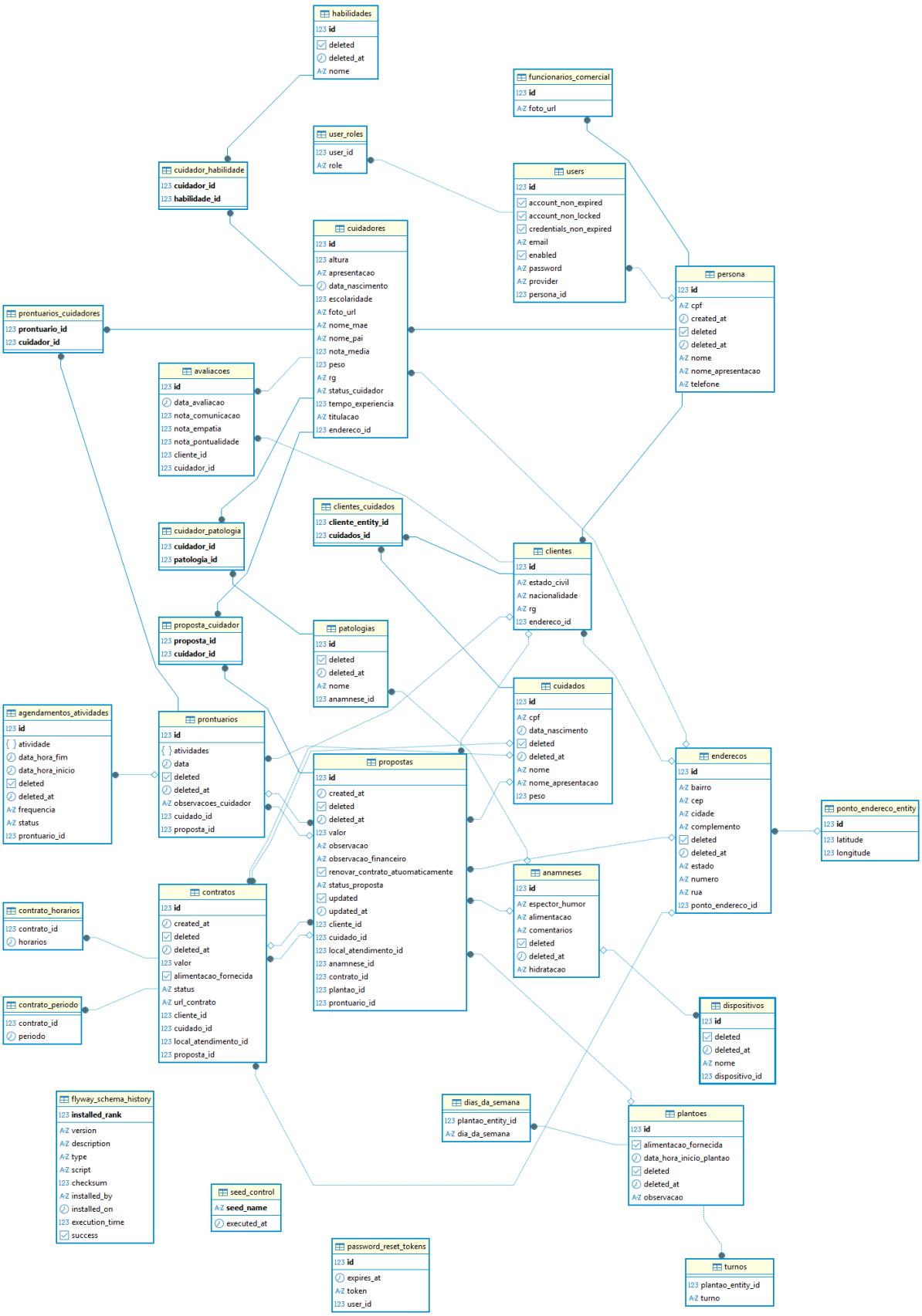

Figura 9: Modelo de Banco de dados

Esse esquema relacional foi estruturado para manter integridade referencial e facilitar consultas pela API, que por sua vez abastece as interfaces do cliente, cuidador e funcionário com dados consistentes e atualizados.

4. Implantação

4.1 Projeto de Implantação

O ponto de entrada da solução Pronto Afeto é o front-end web, desenvolvido em Next.js e acessado por navegadores em diferentes formatos de tela, como smartphones, tablets e computadores. Essa camada de interface é responsável por renderizar as páginas, aplicar as regras de navegação e oferecer uma experiência responsiva para os perfis envolvidos no sistema, consumindo os serviços da API por meio de chamadas HTTPS.

O fluxo de funcionamento em produção pode ser descrito nas etapas abaixo:

- Acesso ao sistema: a pessoa usuária acessa o endereço do Pronto Afeto em um navegador web no dispositivo de sua preferência. O Nginx recebe essa conexão HTTPS e entrega os arquivos do front-end (páginas, scripts e estilos), iniciando o carregamento da interface no cliente.

- Carregamento da interface: após o primeiro acesso, o front-end é carregado no navegador, exibindo os componentes visuais, formulários e menus. A partir desse momento, a maior parte das interações ocorre no próprio navegador, que envia apenas os dados necessários para a API, reduzindo tráfego e melhorando o tempo de resposta.

- Comunicação com a API: sempre que a pessoa usuária realiza ações como cadastro, edição ou consulta, o front-end dispara requisições HTTPS para a API REST implementada em Java com Spring Boot. A API recebe essas chamadas, executa as regras de negócio, valida as informações e coordena o acesso aos demais serviços.

- Persistência e arquivos: durante o processamento, a API consulta ou atualiza o banco de dados PostgreSQL, responsável por manter as informações estruturadas do sistema. Arquivos binários, como fotos de perfil ou documentos enviados, são gravados em um bucket Amazon S3, liberando o banco relacional para lidar apenas com dados transacionais.

- Retorno ao navegador: após o processamento, a API devolve as respostas em formato JSON. O front-end interpreta esses dados e atualiza a interface em tempo

real, exibindo mensagens de sucesso ou erro, recarregando tabelas e ajustando os componentes visuais, o que proporciona uma navegação contínua e responsiva.

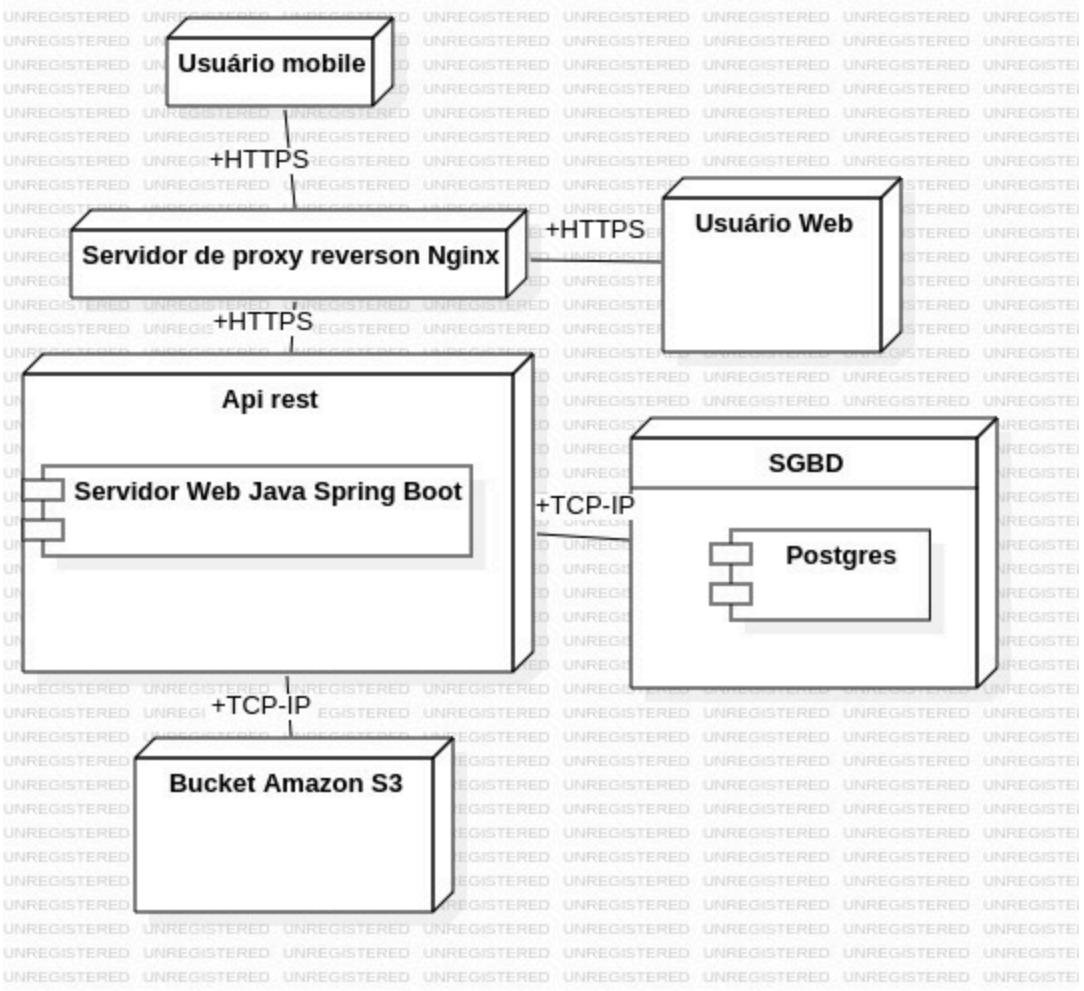

Figura 10: Diagrama de implantação

A arquitetura de implantação do Pronto Afeto, representada no diagrama de deployment, favorece escalabilidade, segurança e organização das responsabilidades. O uso combinado de Nginx, API REST com Spring Boot, banco PostgreSQL e armazenamento em Amazon S3, aliado a um front-end web responsivo, garante um fluxo de comunicação padronizado em HTTPS e protege os dados manipulados durante toda a interação com a plataforma.

5. Manual do Usuário

Nesta seção será descrito de forma prática e objetiva, como utilizar as interfaces do Pronto Afeto, detalhando as telas, suas funções, campos, ações disponíveis e navegação entre páginas. O foco é orientar o usuário final e também registrar o comportamento do front-end desenvolvido em Next.js.

O sistema Pronto Afeto é composto por três aplicações independentes, cada uma com seu público e conjunto de telas:

1. **Aplicação Cliente:**
   Voltada ao cliente que busca cuidadores, acompanha propostas, contratos e informações relacionadas ao atendimento.

2. **Aplicação Cuidador:**
   Voltada ao cuidador, com foco em perfil profissional, propostas recebidas, contratos, agenda e acompanhamento do trabalho.

3. **Aplicação Comercial/Operação**:
   Voltada ao gerenciamento operacional, cadastros, validações, moderação, acompanhamento de contratos e suporte ao ecossistema.

5.1 Aplicação Cliente

5.1.1 Tela de Login e Cadastro

Responsável por permitir o acesso seguro à plataforma e o registro de novos clientes, validando dados pessoais, e-mail e senha para criar a identidade do usuário no sistema.

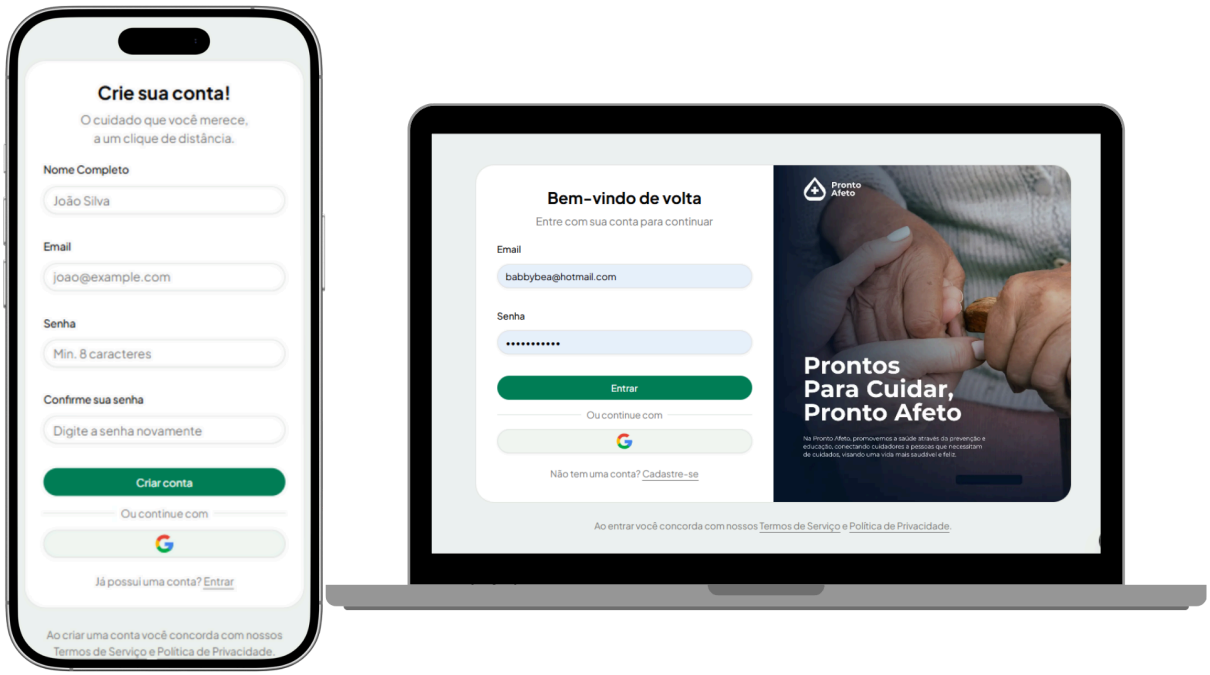

Figura 11: Telas de Cadastro e Login da Aplicação Cliente

5.1.2 Tela Home

Funciona como o painel principal do cliente, oferecendo uma visão geral das funcionalidades do sistema e permitindo uma navegação rápida e intuitiva do ecossistema cliente.

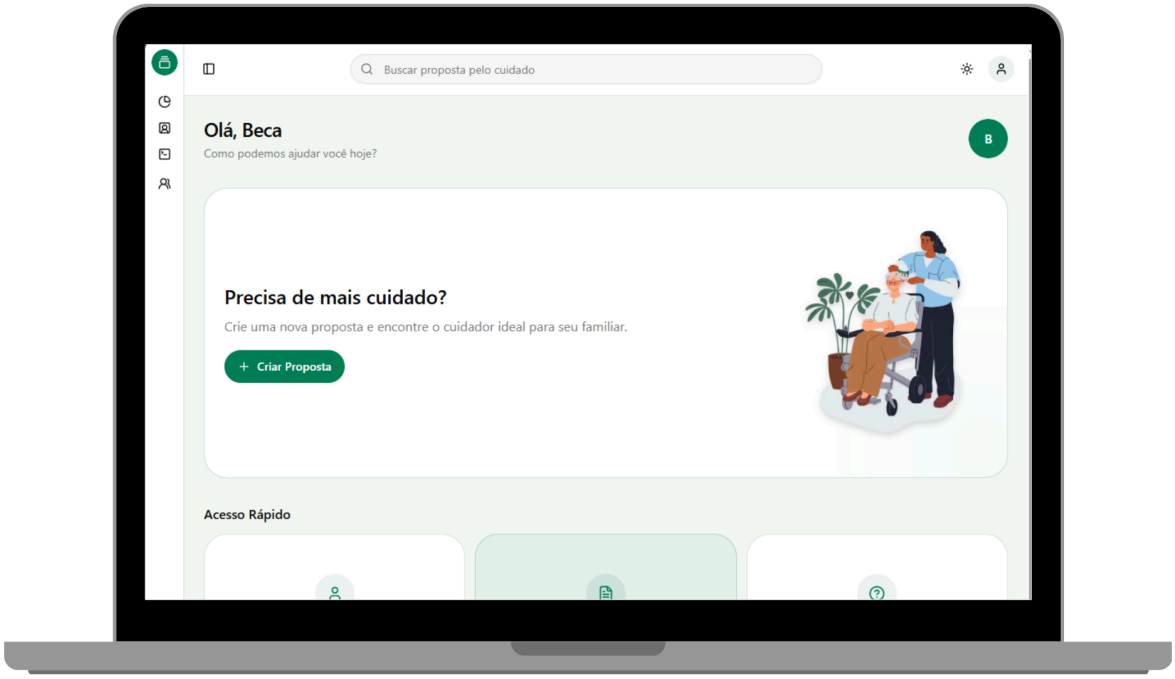

Figura 12: Tela de Início da Aplicação Cliente

5.1.3 Tela de Gerenciamento de Cuidados

Destina-se ao acompanhamento direto dos dependentes do cliente.

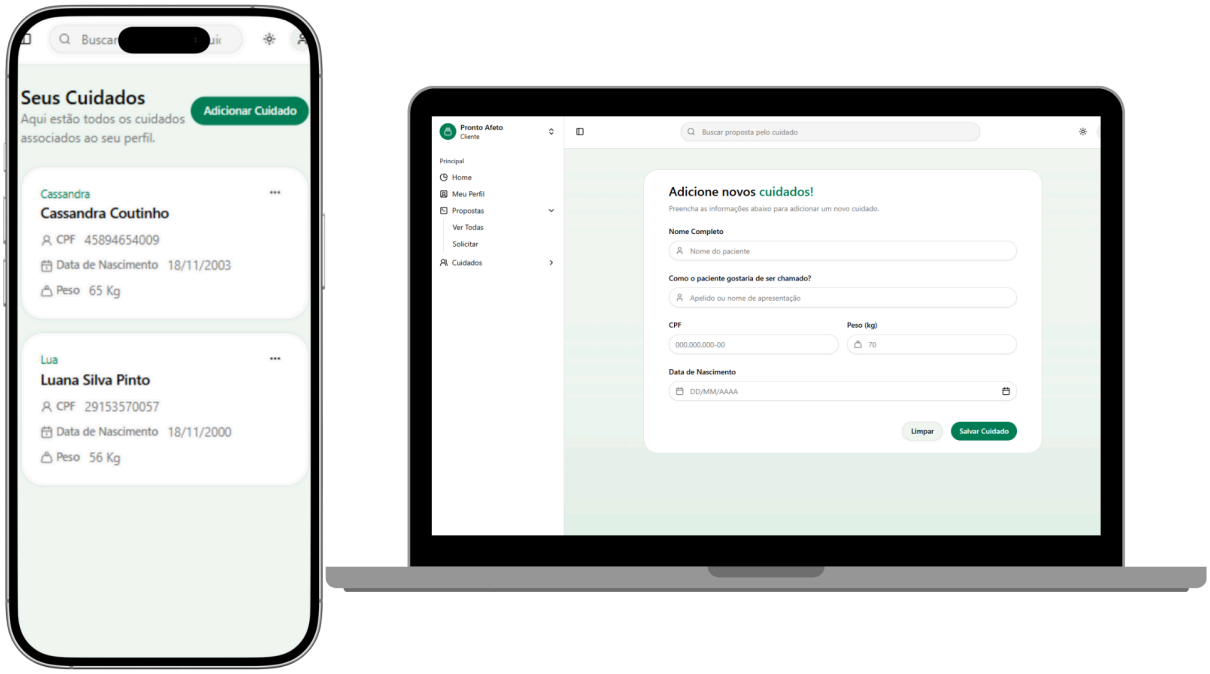

Figura 13: Telas de Gerenciamento de dependentes da Aplicação Cliente

5.1.4 Tela de Solicitação de Propostas

Permite ao cliente criar e enviar pedidos de orçamento ou serviço para cuidadores, detalhando as necessidades de atendimento para que os profissionais possam responder com propostas.

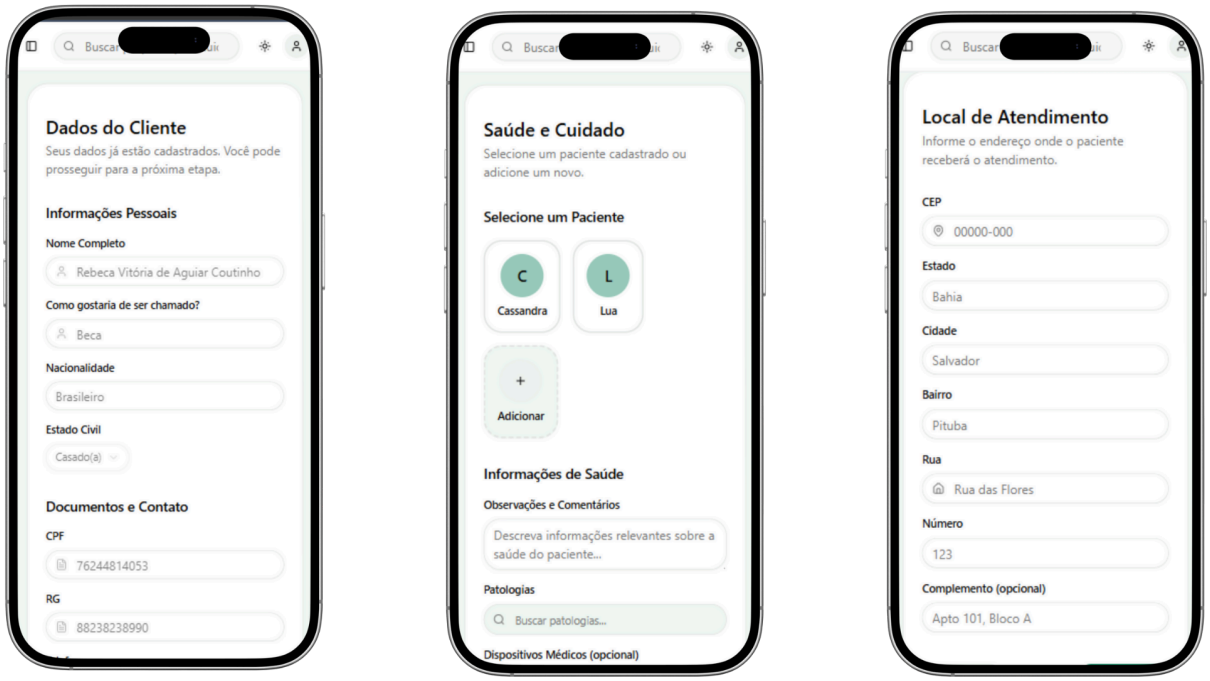

Figura 14: Telas Solicitação de Proposta da etapa 1 até a 3

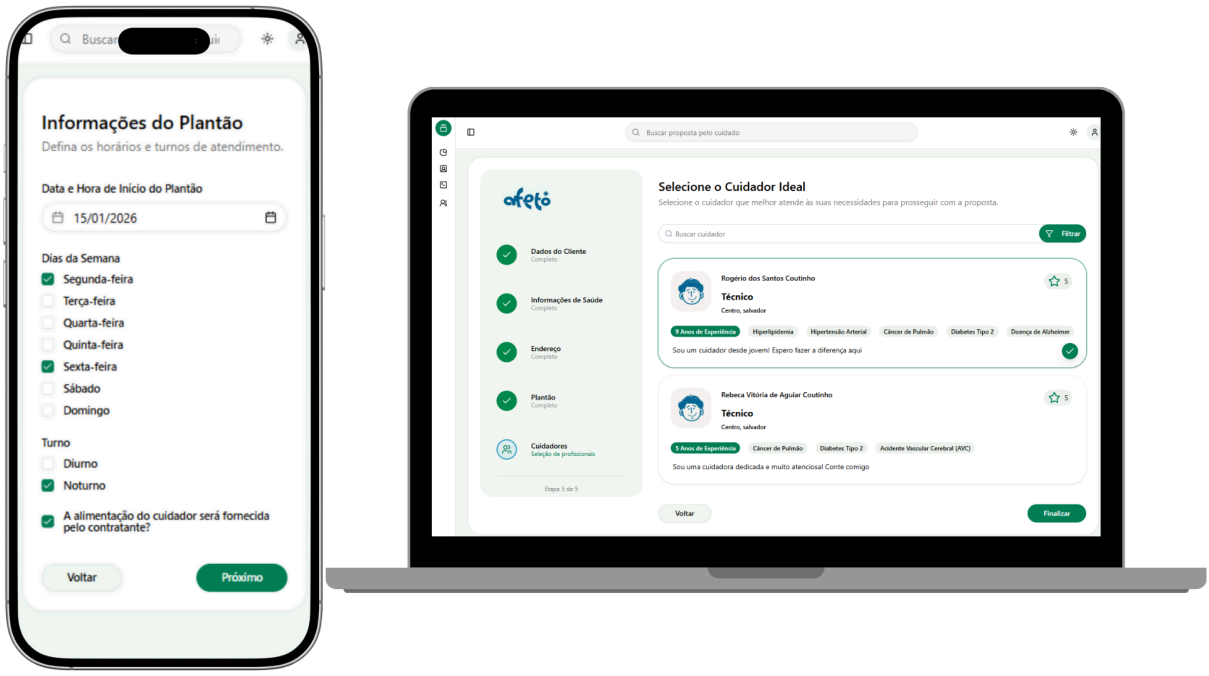

Figura 15: Telas Solicitação de Proposta da etapa 4 até a 5

Figura 16: Tela de confirmação de solicitação da proposta

5.1.5 Tela de Gerenciamento de Propostas

Focada no controle das negociações, onde o cliente pode visualizar a lista de propostas recebidas, aceitar ou recusar orçamentos e monitorar o status de cada transação até a formalização do contrato.

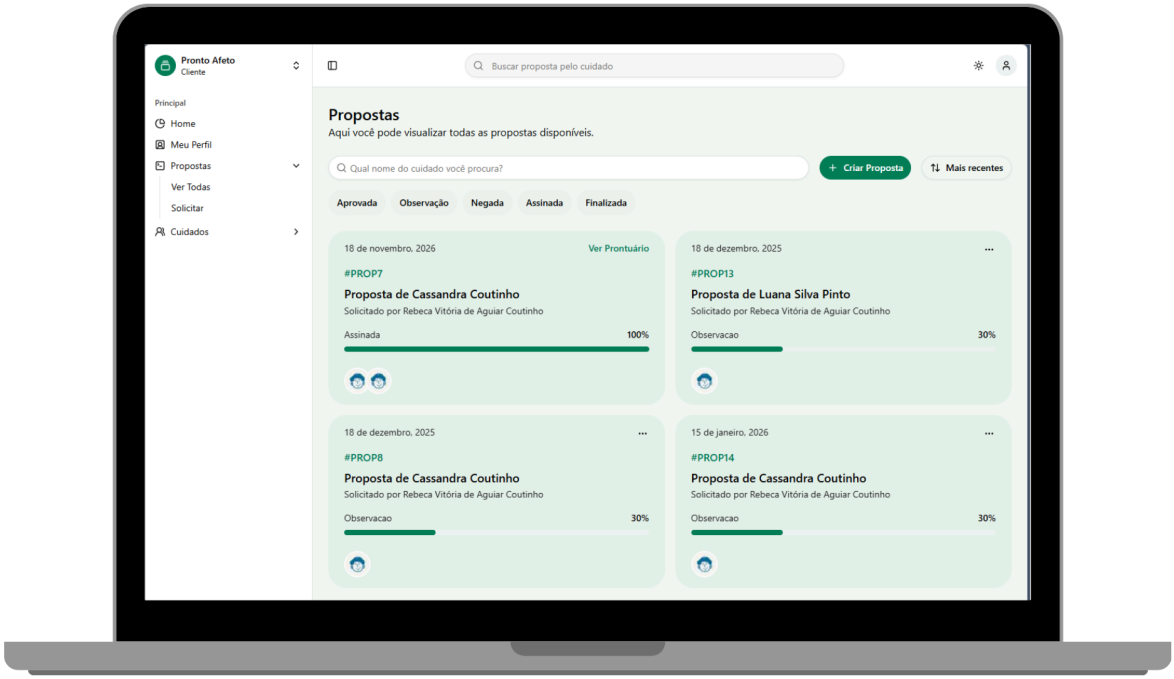

Figura 17: Tela de Acompanhamento de Propostas da Aplicação Cliente

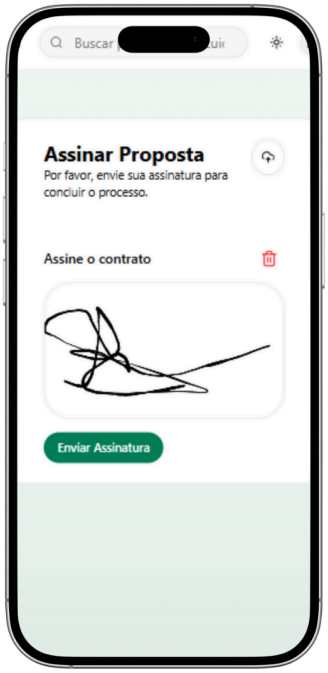

Figura 18: Tela de Assinatura de Contrato da Aplicação Cliente

5.1.6 Tela de Prontuário e Avaliação

Espaço dedicado à transparência do serviço, onde o cliente consulta os registros diários feitos pelo cuidador e, após a conclusão do contrato, realiza a avaliação do profissional com notas e comentários.

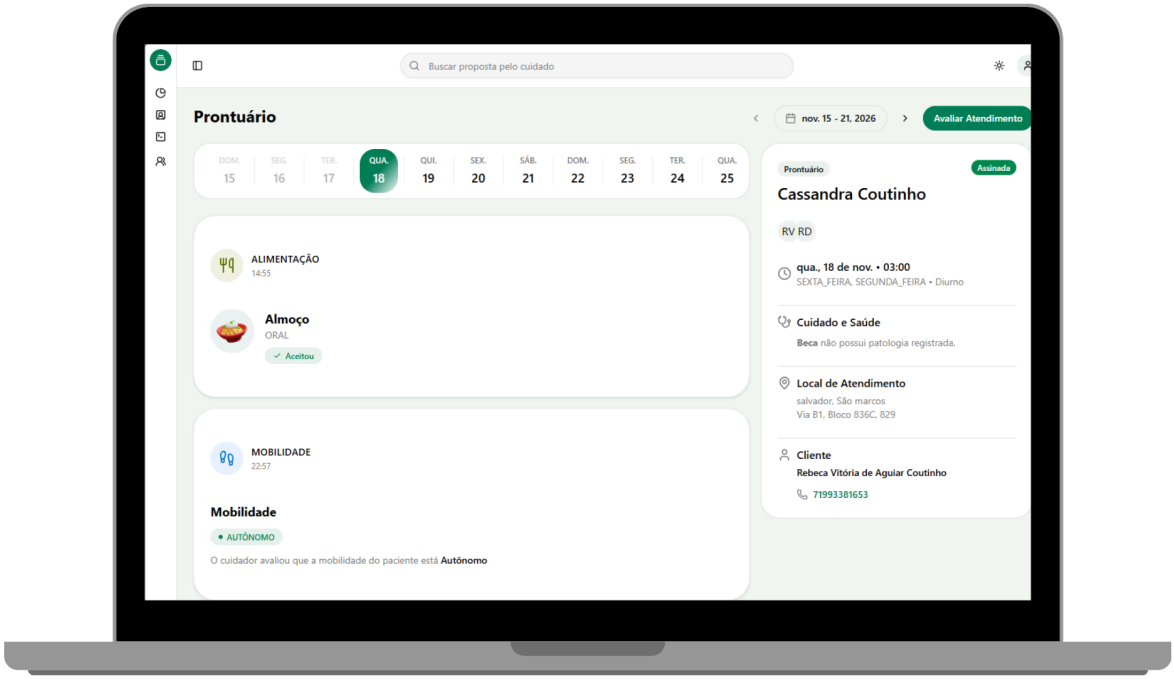

Figura 19: Tela de Acompanhamento de Prontuário

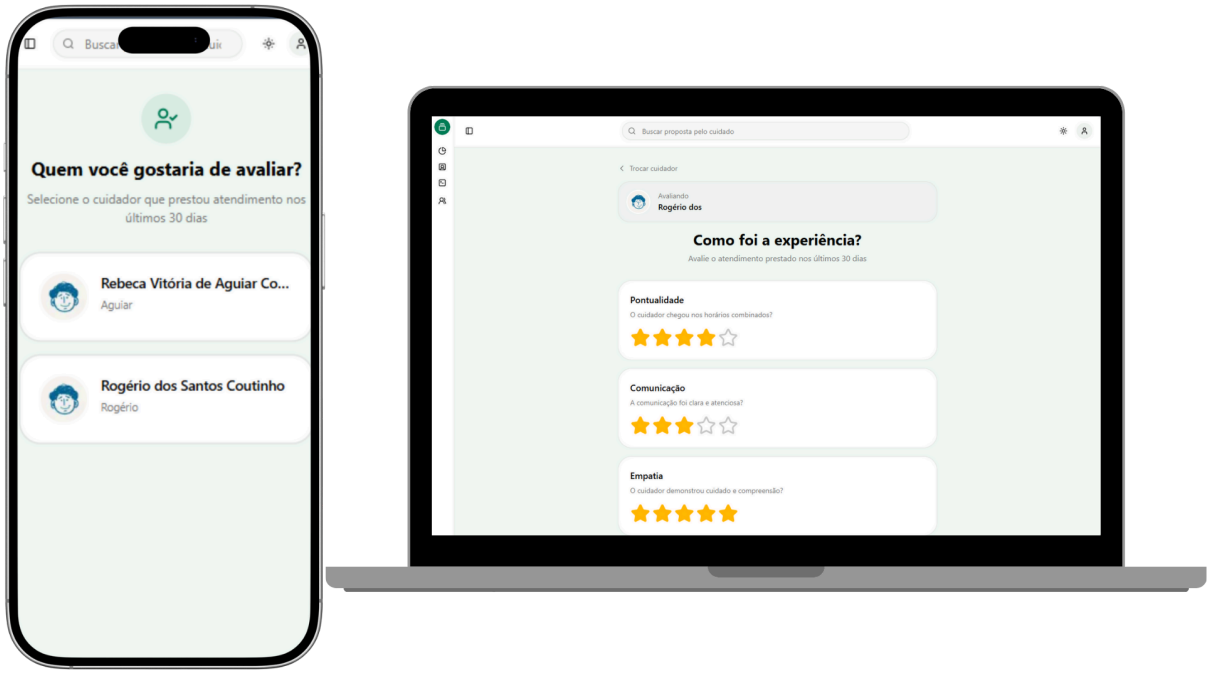

Figura 20: Telas de Avaliação de Cuidador

5.1.7 Tela de Perfil

Permite a gestão dos dados cadastrais do cliente, possibilitando a atualização de informações pessoais, de contato e de endereço diretamente na interface.

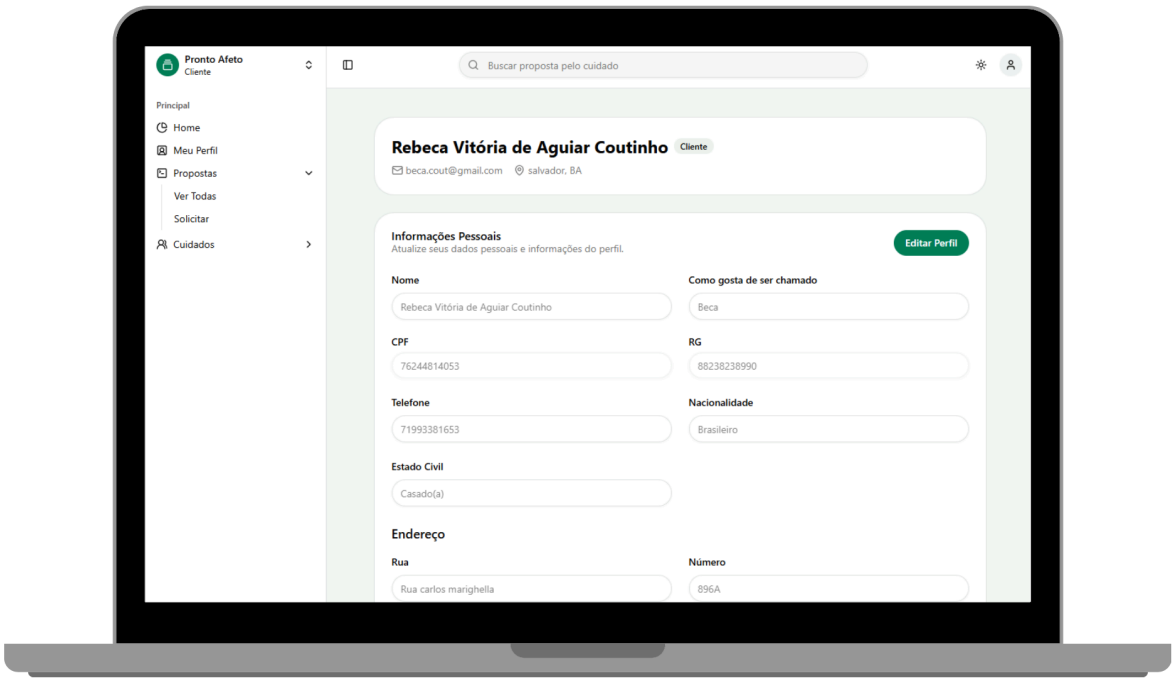

Figura 21: Tela de Perfil

5.1.8 Tela de Ajuda

Oferece suporte ao usuário com orientações sobre o uso da plataforma, FAQs ou canais de comunicação para resolução de dúvidas técnicas ou operacionais.

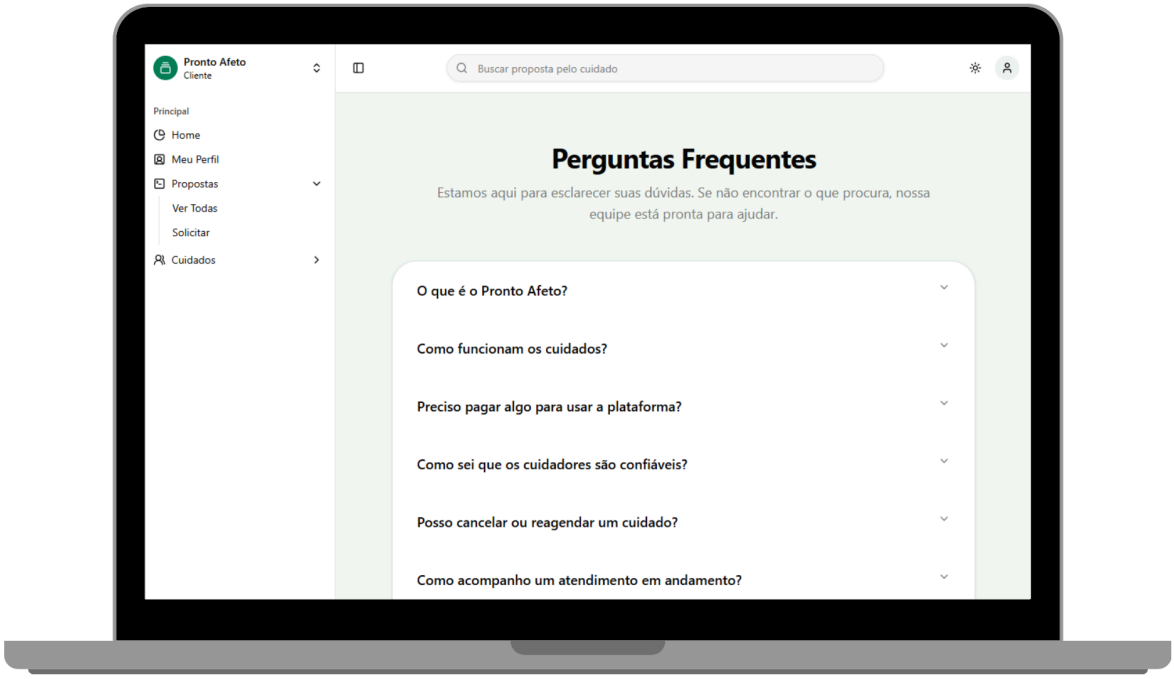

Figura 22: Tela de Ajuda

5.2 Aplicação Comercial

5.2.1 Tela de Acompanhamento de Propostas

Destina-se ao monitoramento e moderação das propostas de serviço enviadas no ecossistema. Permite que a equipe operacional valide as negociações em curso, garantindo que os termos estejam de acordo com as diretrizes da plataforma antes da formalização.

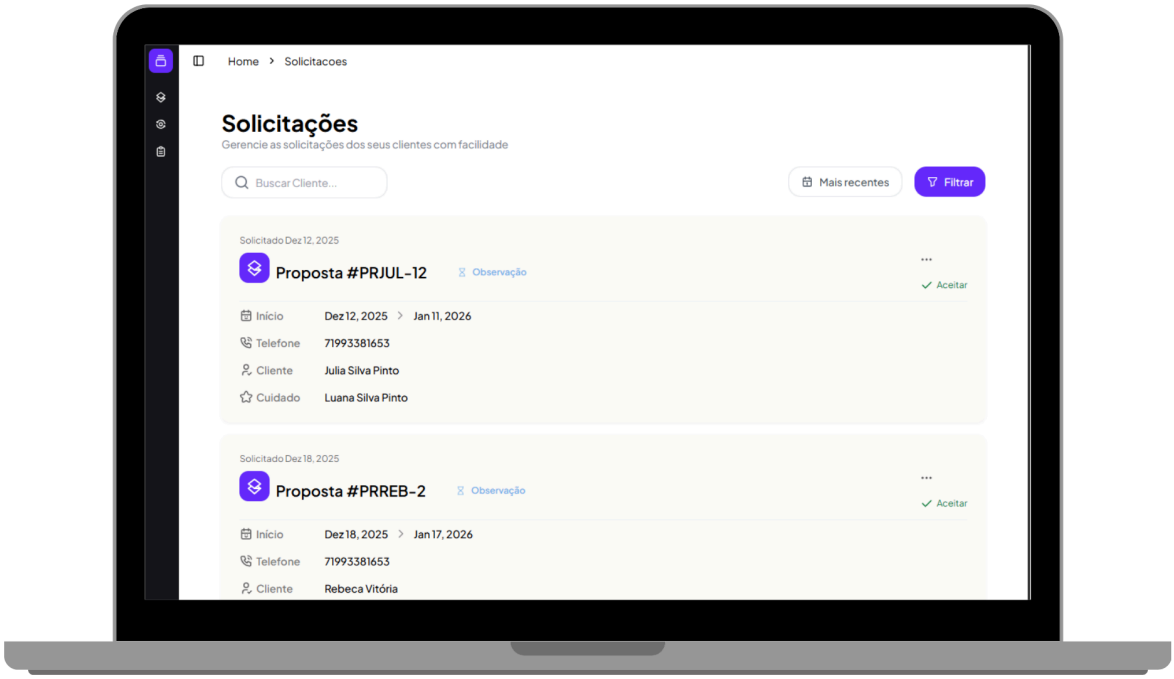

Figura 23: Tela de Gerenciamento de Solicitações de Propostas

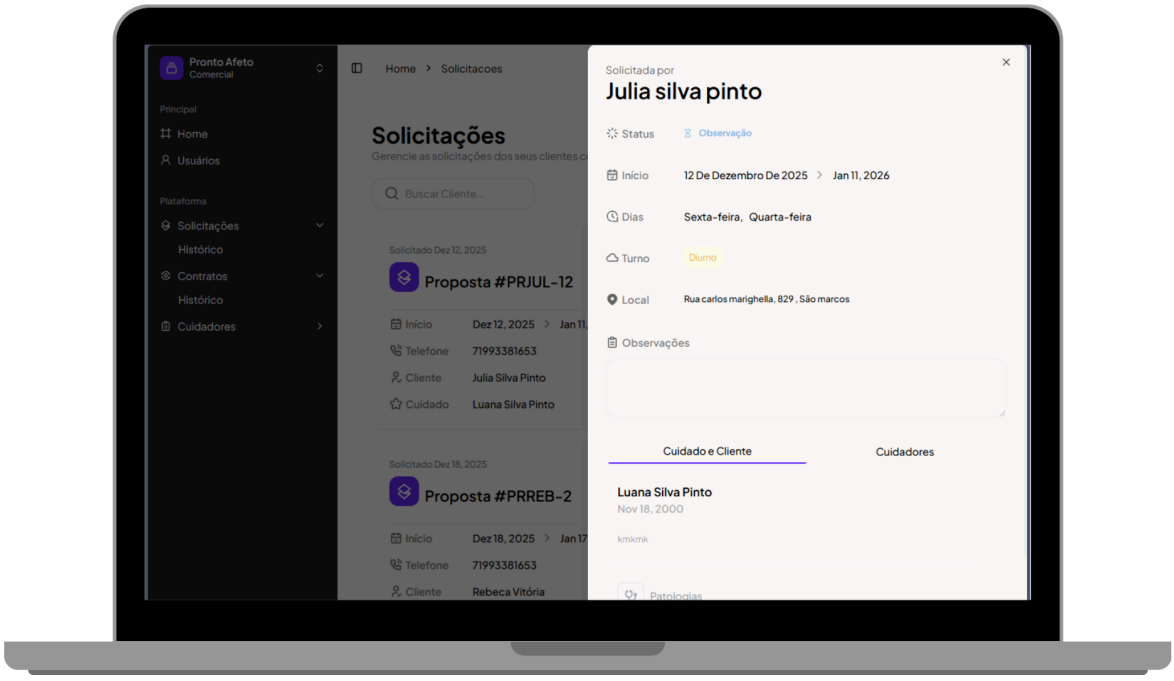

Figura 24: Tela de Visualização de Solicitação de Proposta

5.2.2 Tela de Consulta de Contratos

Centraliza o acesso a todos os contratos firmados entre clientes e cuidadores. O objetivo é permitir a auditoria, o acompanhamento de vigências e a resolução de eventuais disputas ou dúvidas contratuais, servindo como base histórica da operação.

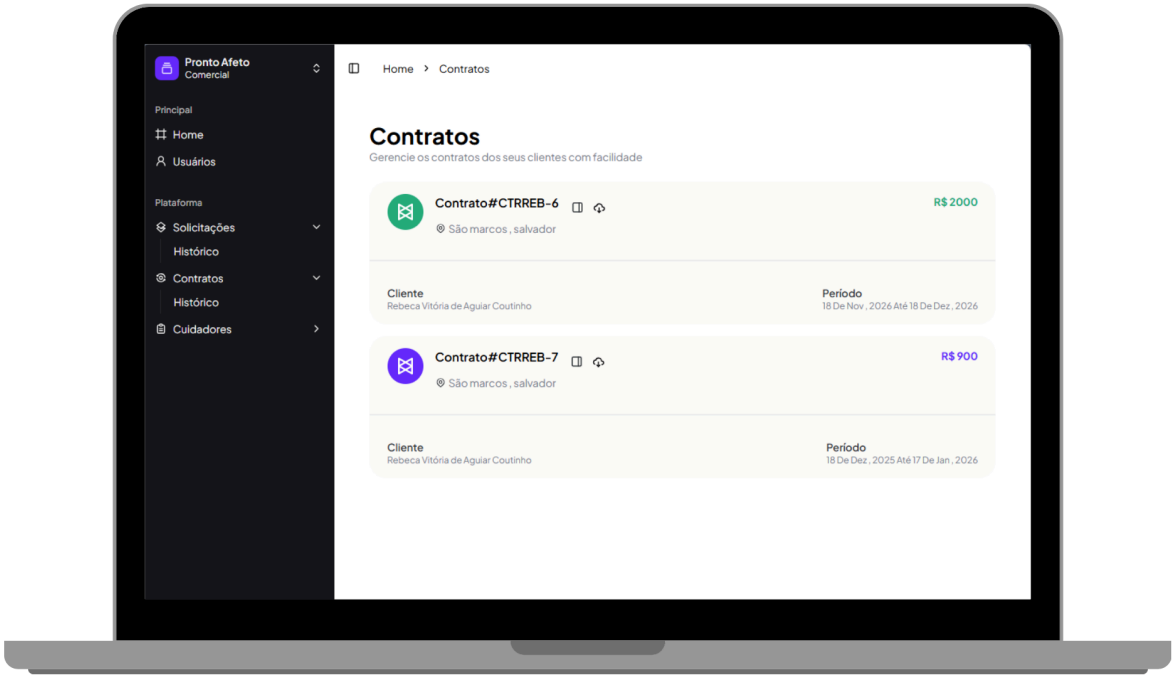

Figura 25: Tela de Visualização de Solicitação de Proposta

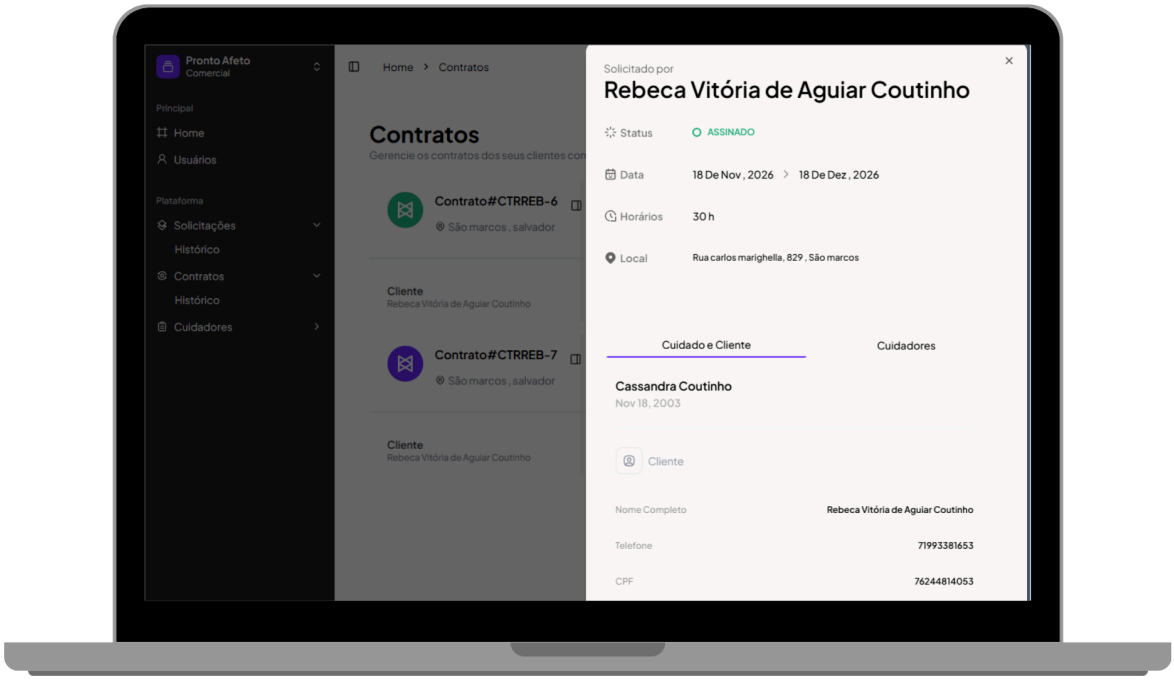

Figura 26: Tela de Visualização de Contratos

5.2.3 Tela de Gerenciamento de Usuários

Focada na administração dos usuários da aplicação comercial. Permite visualizar dados cadastrais, status da conta e histórico de atividades, sendo essencial para o suporte técnico e a manutenção da base de usuários do sistema.

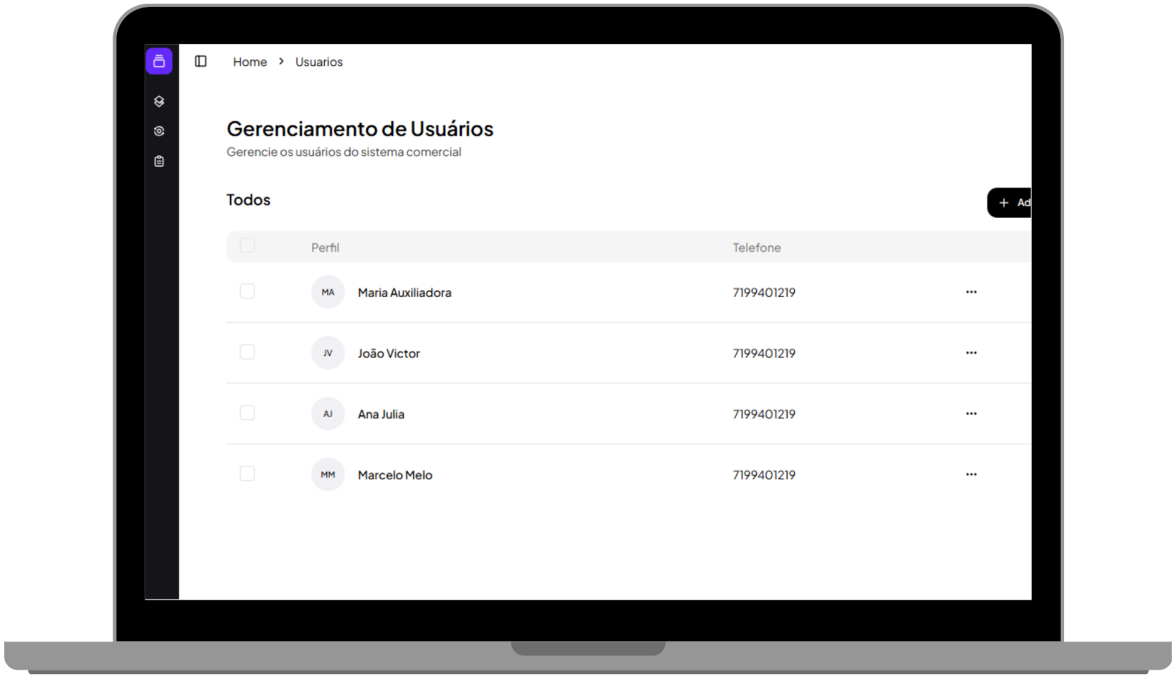

Figura 27: Tela de Visualização de Gerenciamento de Usuários Comerciais

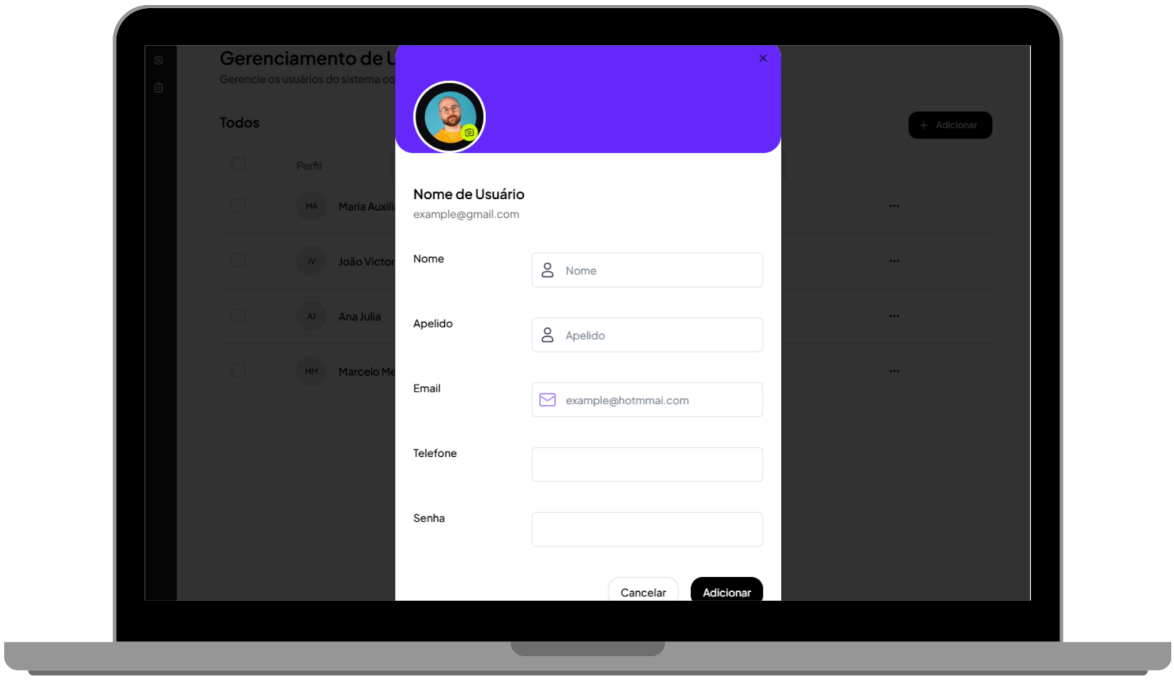

Figura 28: Tela de Cadastro de Usuário Comercial

5.2.4 Tela de Gerenciamento de Cuidadores

Voltada especificamente para o controle dos profissionais da plataforma. Seu objetivo é gerenciar o processo de validação de documentos, aprovação de perfis e monitoramento do desempenho dos cuidadores, assegurando a qualidade e a segurança dos serviços prestados.

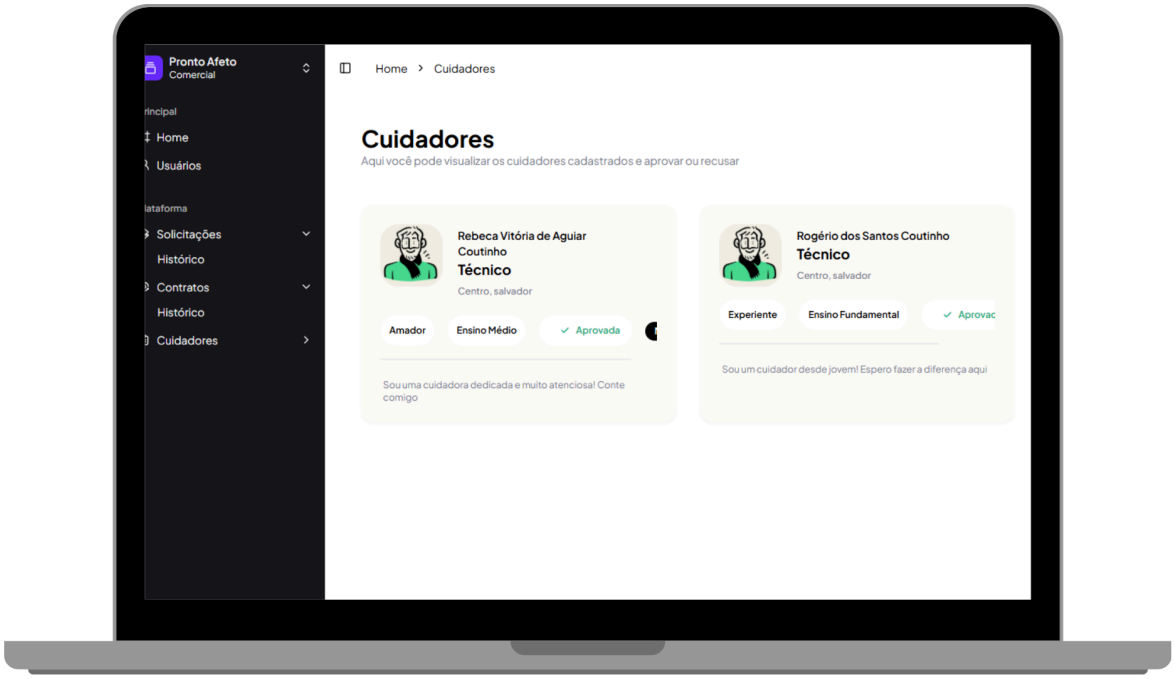

Figura 29: Tela de Gerenciamento de Cuidadores

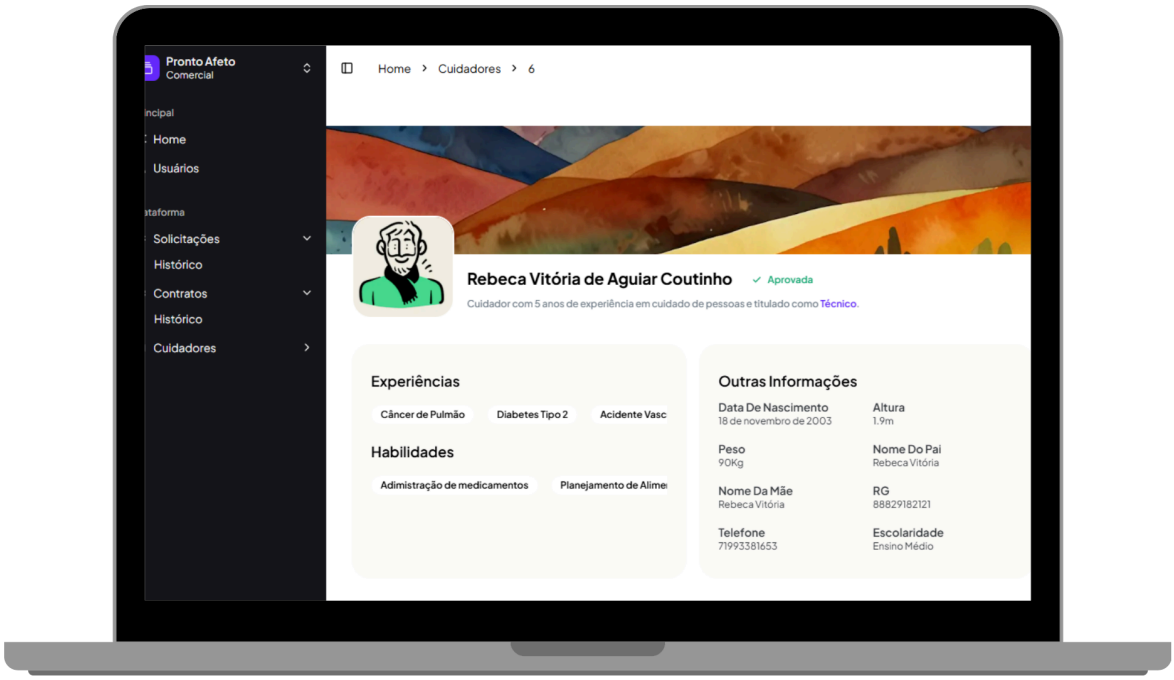

Figura 30: Tela de Visualização de Cuidador

5.3 Aplicação Cuidador

5.3.1 Tela de Login e Cadastro

Responsável por permitir o acesso seguro ao sistema e o registro de novos profissionais. No cadastro, o cuidador insere dados pessoais, profissionais e documentos obrigatórios,

que ficam com status inicial "Pendente de Aprovação" até a validação pela equipe administrativa.

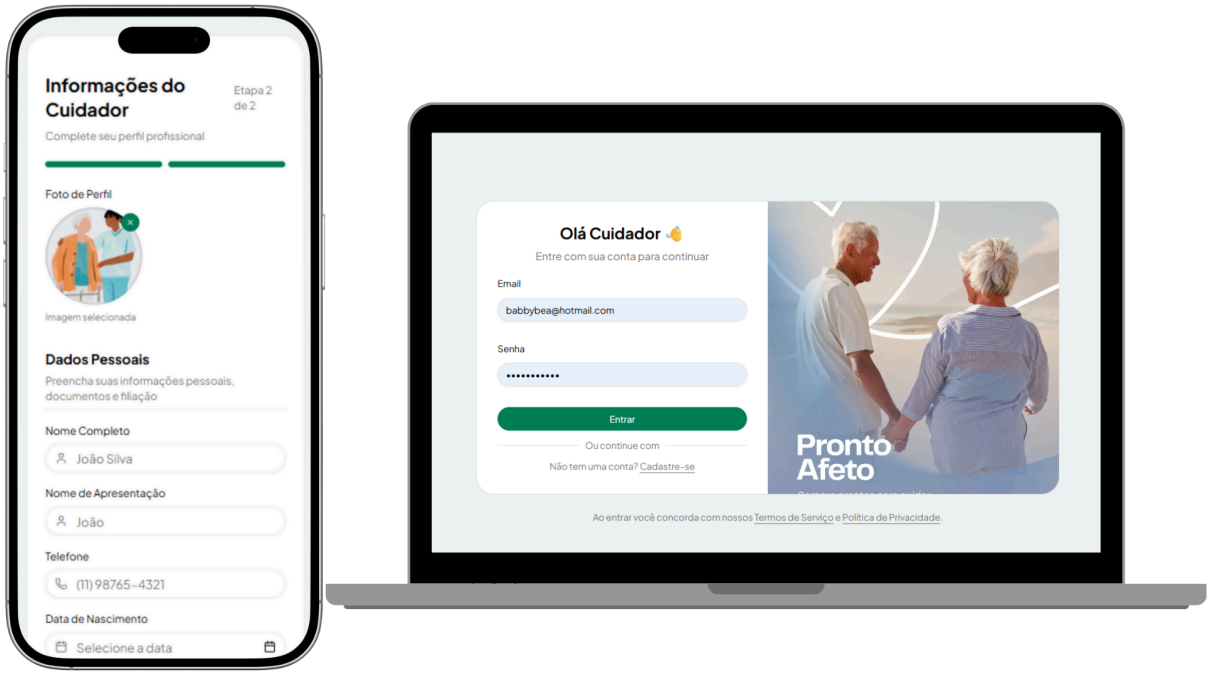

Figura 31: Tela de Login e Cadastro da Aplicação Cuidador

5.3.2 Tela de Home e Perfil

Funciona como o painel central do profissional, permitindo a gestão de sua identidade na plataforma. Nela, o cuidador visualiza o resumo de suas atividades e pode atualizar suas informações cadastrais e profissionais para manter seu perfil atrativo e atualizado para os clientes.

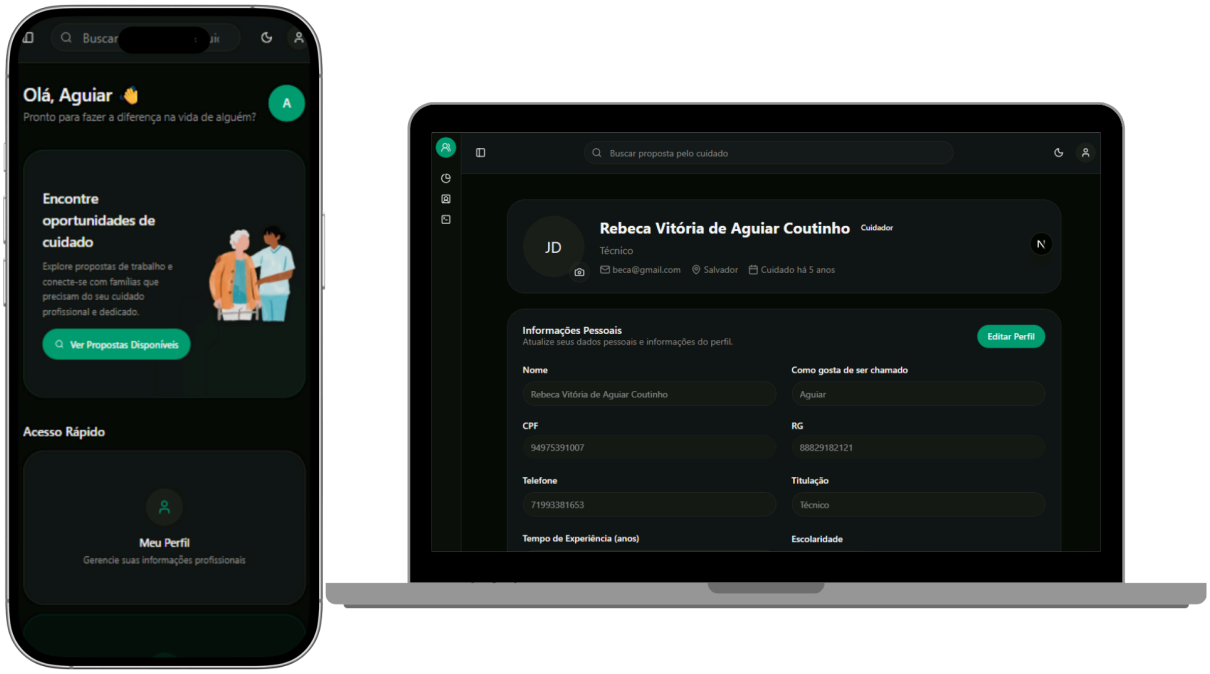

Figura 32: Tela de Início e Perfil da Aplicação Cuidador

5.3.3 Tela de Acompanhamento de Propostas

Destina-se ao gerenciamento das oportunidades de trabalho recebidas. O cuidador pode visualizar uma lista de propostas de serviço enviadas por clientes, tendo a opção de aceitá-las ou recusá-las, o que atualiza o status da negociação em tempo real no sistema.

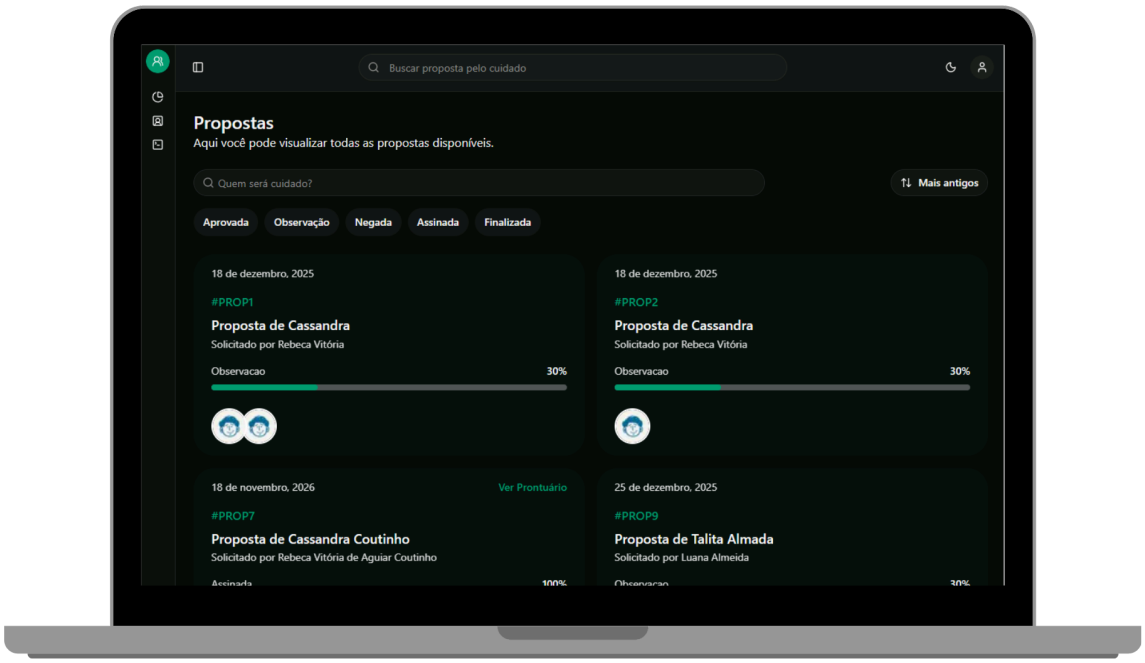

Figura 33: Tela de Acompanhamento de Propostas da Aplicação Cuidador

5.3.4 Tela de Gerenciamento de Prontuários

Focada no registro cotidiano do cuidado assistido. Permite que o cuidador insira observações, atualizações e registros detalhados sobre os cuidados ativos, garantindo que o histórico do paciente seja mantido de forma organizada, transparente e acessível para o acompanhamento do cliente.

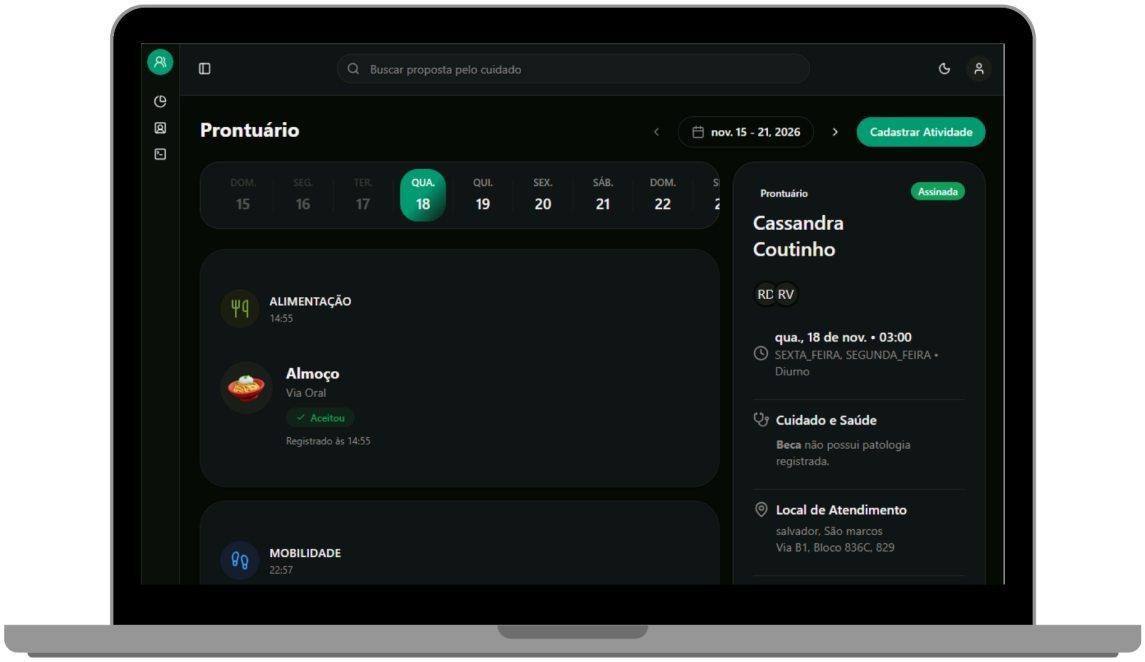

Figura 34: Tela de Acompanhamento do Prontuário da Aplicação Cuidador

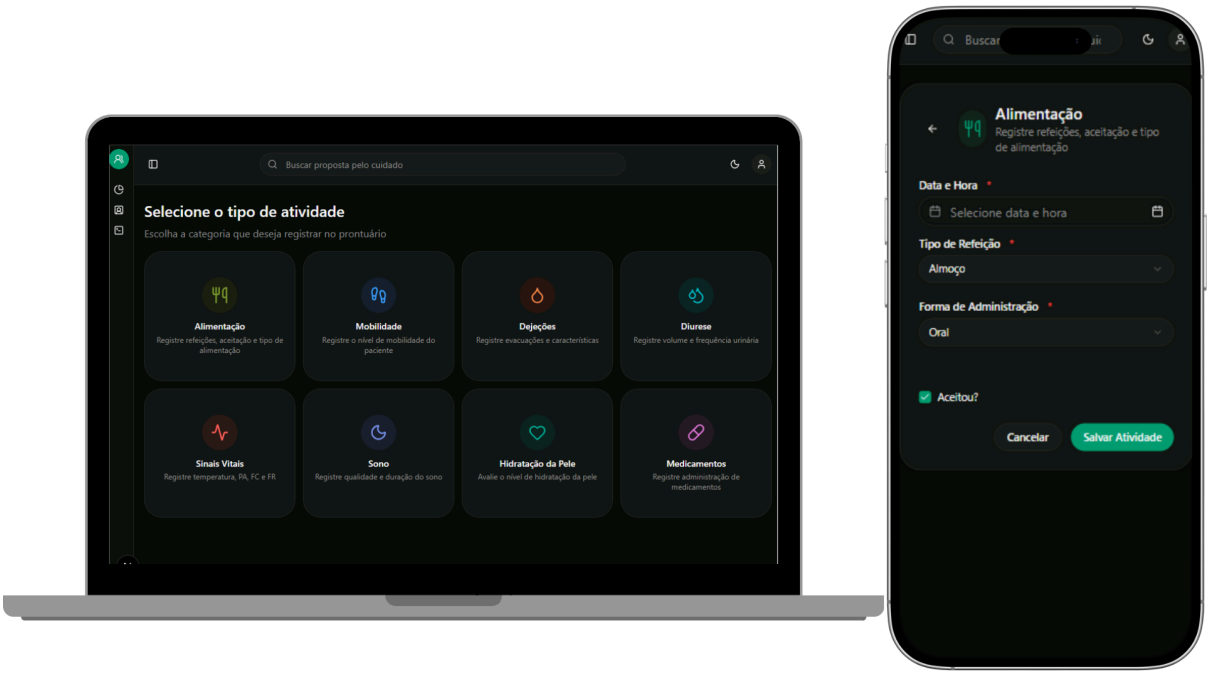

Figura 35: Tela de Registro de Atividades da Aplicação Cuidador

Agradecimentos

Quero agradecer, primeiro de tudo, aos meus pais, Alessandra e Rogério. Vocês me apoiaram em cada escolha e estiveram do meu lado durante todos esses anos de estudos no IFBA. Sem o suporte de vocês, eu não teria chegado até aqui. Esta conquista é nossa.

Aos professores que não só me ensinaram, mas acreditaram em mim e me ajudaram a trilhar meu caminho:

Ao professor Antonio Carlos, meu muito obrigado por ter aberto as portas para as atividades docentes; essa experiência foi um diferencial enorme na minha jornada. Ao Professor Manoel e meu Orientador Frederico , agradeço pela confiança e pela oportunidade de participar do projeto de pesquisa, algo que me fez crescer demais. À professora Flávia, agradeço de coração pela oportunidade de estágio, que foi um dos grandes passos no mercado de trabalho.

Olhando para trás, vejo que esse ciclo de quatro anos no curso de Análise e Desenvolvimento de Sistemas foi muito mais do que apenas estudar. Foi um tempo de amadurecimento, tanto profissional quanto pessoal. Sinto que encerro essa fase pronta para o que vem pela frente, sabendo que cada um de vocês fez parte da construção da pessoa e da profissional que sou hoje.

Referências

MOREIRA, Vinícius Rodrigues dos Santos. *Projeto Pronto Afeto API*. 2024. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) — Instituto Federal da Bahia, Salvador. Disponível em: https://www.ads.ifba.edu.br/item1249.

VERCEL. *Next.js Documentation*. Disponível em: https://nextjs.org/docs. Acesso em: 29 out. 2025.

MICROSOFT. *TypeScript Documentation*. Disponível em: https://www.typescriptlang.org/docs/. Acesso em: 1 nov. 2025.

SHADCN. *shadcn/ui Documentation*. Disponível em: https://ui.shadcn.com/docs. Acesso em: 1 nov. 2025.

TAILWIND LABS. *Tailwind CSS Documentation*. Disponível em: https://tailwindcss.com/docs. Acesso em: 1 nov. 2025.

FRAMER. *Motion for React: documentation*. Disponível em: https://motion.dev/. Acesso em: 10 jan. 2025.

COLIN MCKENZIE. *Zod Documentation*. Disponível em: https://zod.dev/. Acesso em: 1 nov. 2025.

QUEIRÓS, A. et al. *Remote Care Technology: A Systematic Review of Technologies for Chronic Disease Management*. MDPI Technologies, v. 6, n. 1, 2018.

SRIRAM, V.; RICHARDSON, A.; MCCANN, J. *Carers using assistive technology in dementia care at home: An integrative literature review*. BMC Geriatrics, v. 22, n. 1, 2022.

ZUBRYTSKYI, Anton. *Front-end architecture: in-depth analysis, best practices, and insights*. ELITEX, 09 abr. 2024. Disponível em: https://elitex.systems/blog/front-end-architecture-in-depth-analysis\. Acesso em: 14 dez. 2025.

PRATA, Ramon. *How to structure a React App in 2025 (SPA, SSR or Native).* Medium, 27 ago. 2025. Disponível em: https://ramonprata.medium.com/how-to-structure-a-react-app-in-2025-spa-ssr-or-native-10d8de7a245a\. Acesso em: 14 dez. 2025.

JACKSON, Cam. *Micro frontends*. Martin Fowler, 19 jun. 2019. Disponível em: https://martinfowler.com/articles/micro-frontends.html. Acesso em: 14 dez. 2025.