LUIS FERNANDO SCALVI DOS SANTOS

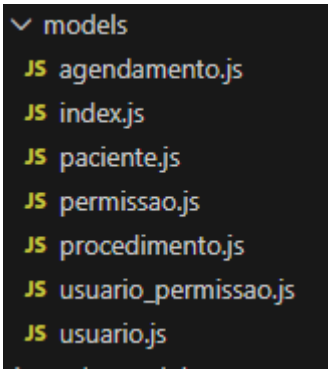
PROJETO INTEGRADOR

BACK-END

O backend do sistema foi desenvolvido em **Node.js** utilizando **Express**, seguindo uma arquitetura organizada em camadas para facilitar a manutenção e a compreensão do código. A ideia principal foi separar as responsabilidades em módulos distintos: **models**, **repositories**, **services** e **controllers**.

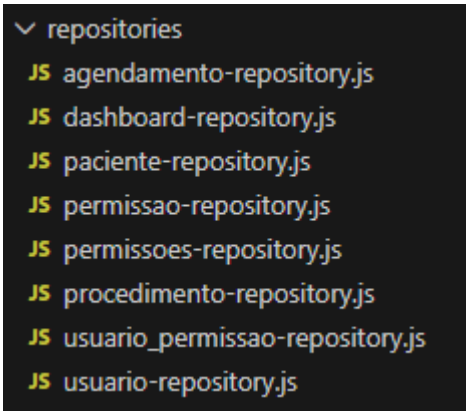
Models

Os models representam as entidades do banco de dados e foram implementados com o Sequelize, que é um ORM para Node.js. Cada model corresponde a uma tabela, como Usuario, Agendamento, Procedimento e Permissao, definindo os campos, tipos de dados e restrições. Isso torna o acesso ao banco mais seguro e estruturado, além de simplificar operações de leitura e escrita.



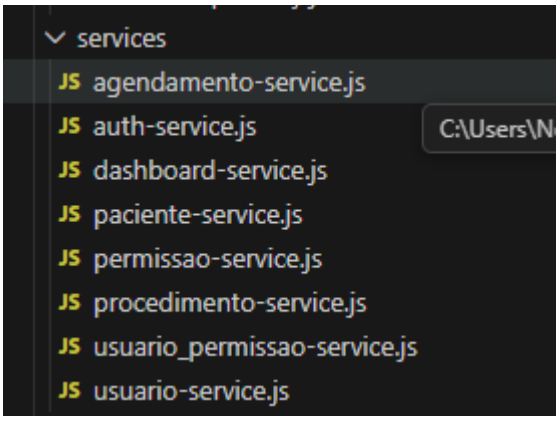
Repositories

Concentra todas as operações diretas com o banco, como consultas, inserções, atualizações e exclusões. Com isso, mantemos a lógica de acesso aos dados isolada, o que facilita testes e eventuais mudanças na estrutura do banco sem impactar outras partes da aplicação.



Services

São responsáveis pela lógica de negócio da aplicação. Eles recebem os dados das rotas, aplicam validações, regras e cálculos, e interagem com os repositories para manipular as informações no banco. Por exemplo, o serviço de agendamentos calcula o faturamento mensal e filtra os agendamentos por status, garantindo que apenas informações corretas sejam enviadas ao frontend.



Controller

Por fim, os controllers lidam com as requisições HTTP, chamam os serviços correspondentes e retornam as respostas ao cliente. Dessa forma, o backend fica organizado e desacoplado: os controllers cuidam da comunicação com o frontend, os services da lógica de negócio e os repositories do acesso aos dados

Essa estrutura em camadas deixa o código mais limpo e fácil de manter, permitindo que novas funcionalidades sejam adicionadas de maneira segura e sem prejudicar outras partes do sistema.

Autenticação e Permissões

Para garantir que apenas usuários autorizados tenham acesso ao sistema, o backend implementa um mecanismo de **autenticação baseado em JWT (JSON Web Token)**, utilizando a biblioteca **Passport.js**. O processo é dividido em duas etapas principais: login e validação de token nas rotas protegidas.

No **login**, o usuário envia seu email e senha, que são processados pela **Local Strategy** do Passport. A senha enviada é comparada com a senha armazenada no banco, utilizando **bcrypt** para garantir que os dados estejam criptografados e seguros. Se a combinação for correta, o servidor gera um **token JWT** assinado com uma chave secreta (`your-secret-key`) e envia de volta ao cliente. Esse token é necessário para acessar qualquer rota protegida.

Para cada requisição a rotas protegidas, a **JWT Strategy** verifica a validade do token. Se o token for inválido, expirado ou ausente, o servidor retorna um erro 401 (Unauthorized), garantindo que usuários não autenticados não consigam acessar os dados.

Além da autenticação, o sistema possui um **controle de permissões** baseado em descrições associadas a cada usuário. Cada permissão é cadastrada no banco e vinculada aos usuários. O middleware de verificação de permissões consulta o banco para conferir se o usuário tem a permissão necessária antes de permitir o acesso à rota. Se o usuário não tiver a permissão exigida, a resposta é 403 (Forbidden).

Essa abordagem permite que diferentes níveis de acesso sejam facilmente gerenciados. Por exemplo, apenas usuários com a permissão `"VIZUALIZAR_DASH"` podem acessar a dashboard, enquanto outras funções, como cadastro ou edição de procedimentos, podem exigir permissões específicas.

FRONT-END

O frontend do sistema foi desenvolvido utilizando **React**, aproveitando seus conceitos de componentes e gerenciamento de estado para criar uma interface interativa e responsiva. Todo o layout segue um padrão moderno, com **Material-UI** para componentes visuais, garantindo consistência nos elementos como botões, tabelas, cards e modais.

A estrutura do frontend foi organizada de forma a facilitar a manutenção e a escalabilidade do código. Cada tela do sistema é representada por um componente separado, com responsabilidades bem definidas: alguns componentes lidam com **telas principais** como login, dashboard, agenda e procedimentos, enquanto outros cuidam de **componentes menores**, como tabelas, cards de estatísticas e formulários.

O frontend se comunica com o backend por meio de requisições **HTTP**, utilizando a biblioteca **Axios**, garantindo que todos os dados sejam carregados dinamicamente e respeitando as permissões de cada usuário. O gerenciamento de **autenticação e autorização** é feito através de tokens JWT, que são armazenados no navegador e enviados nas requisições às rotas protegidas.

Além disso, o sistema inclui **feedback visual em tempo real** para ações do usuário, como mensagens de erro, confirmações de cadastro ou login e estados de carregamento, tornando a interface mais amigável e intuitiva. Essa abordagem garante que o usuário esteja sempre ciente do que está acontecendo, sem depender apenas de alertas ou redirecionamentos bruscos.

Tela de Login e Cadastro

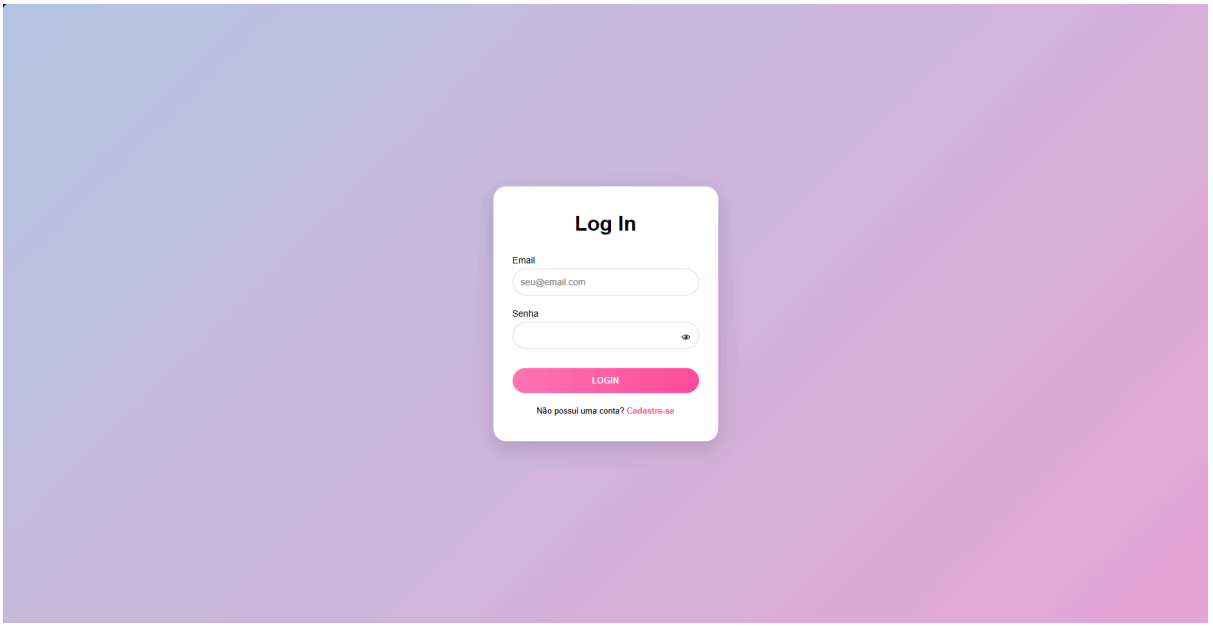
A tela de login foi desenvolvida para oferecer ao usuário uma experiência simples e segura ao acessar o sistema. Nela, o usuário informa seu email e senha, e o formulário envia essas informações para o backend via uma requisição POST para a rota /login.

Para garantir a segurança da senha, o campo permite alternar a visualização da senha, mas os dados enviados permanecem criptografados no backend. Ao enviar o login, se os dados estiverem corretos, o backend retorna um token JWT, que é armazenado no localStorage do navegador. Esse token é fundamental para que o usuário consiga acessar rotas protegidas no frontend, como a dashboard ou o gerenciamento de agendamentos.

Além disso, imediatamente após o login, o frontend faz uma segunda requisição para obter as permissões do usuário **(/usuario_permissao/usuario/:email)**. Essas permissões são armazenadas no localStorage e utilizadas para determinar quais áreas do sistema o usuário pode acessar. Por exemplo, apenas usuários com a permissão **"VIZUALIZAR_DASH"** podem acessar a dashboard; caso contrário, o usuário é redirecionado para a agenda.

A tela também inclui um modal de cadastro, que permite criar novos usuários sem sair da página de login. O cadastro envia uma requisição POST para /novoUsuario, criando o usuário no banco de dados. Mensagens de sucesso ou erro são exibidas dinamicamente, oferecendo feedback imediato ao usuário.

Do ponto de vista de desenvolvimento, a lógica do login foi organizada em funções separadas **(enviaLogin e enviarCadastro)**, e o estado dos campos de entrada foi gerenciado com React Hooks (useState). Essa abordagem facilita a manutenção e a escalabilidade, permitindo, por exemplo, adicionar novos campos ou validações sem impactar o fluxo principal.



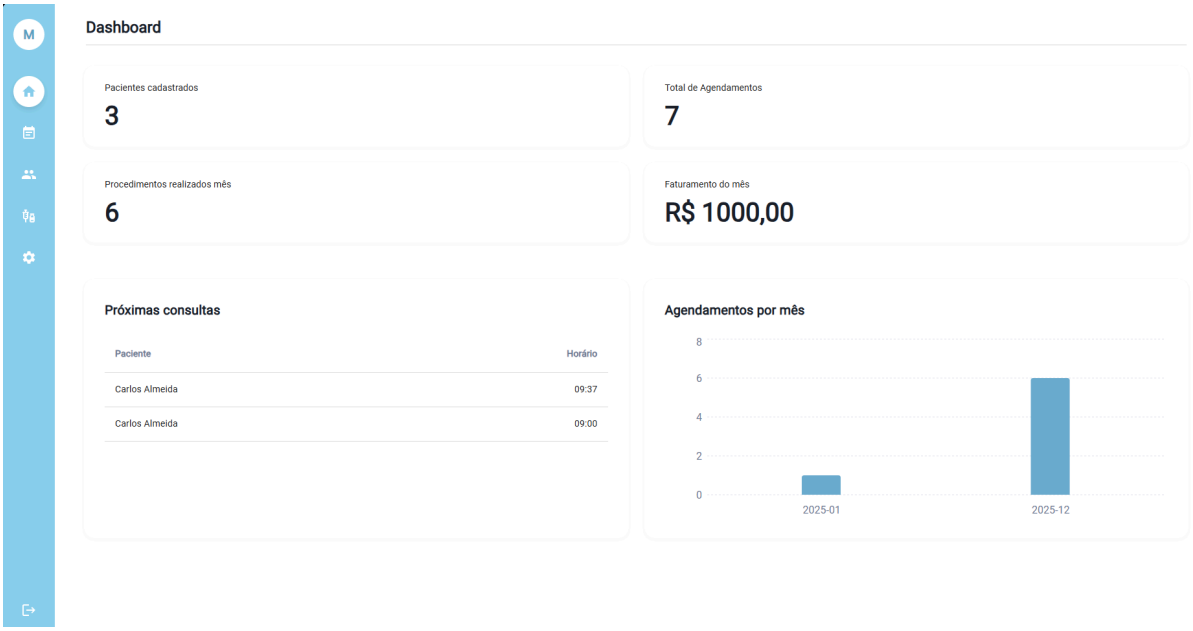
Dashboard

O conteúdo principal do Dashboard foi desenvolvido para fornecer ao usuário uma visão geral clara e objetiva do sistema, consolidando informações essenciais sobre pacientes, agendamentos e procedimentos em um único espaço. Assim que o componente é carregado, uma requisição é enviada ao backend utilizando Axios, juntamente com o token JWT armazenado no localStorage, garantindo que apenas usuários autenticados consigam acessar os dados. Enquanto os dados não são carregados, um indicador de carregamento (CircularProgress) é exibido, mantendo o usuário informado sobre o estado da aplicação e evitando telas em branco ou inconsistentes. Em caso de falha na requisição, uma mensagem de erro é apresentada, oferecendo feedback imediato e amigável ao usuário.

O conteúdo do Dashboard é organizado em cards e seções, facilitando a visualização rápida das informações mais relevantes. Os cards superiores apresentam métricas de destaque, como o total de pacientes cadastrados, o número de agendamentos realizados, procedimentos do mês e o faturamento mensal. Cada card utiliza componentes do Material-UI para apresentar os dados de forma clara e visualmente agradável, com tipografia diferenciada e destaque numérico. Essa organização permite que o usuário compreenda rapidamente o desempenho e a movimentação do sistema sem precisar navegar por múltiplas telas.

Abaixo dos cards, encontram-se seções mais detalhadas: a primeira exibe as próximas consultas em uma tabela, mostrando o nome do paciente e o horário agendado. A tabela é construída com Material-UI, permitindo alinhamento consistente e estilização padronizada. A segunda seção apresenta um gráfico de barras que demonstra a quantidade de agendamentos por mês, utilizando a biblioteca Recharts. Esse gráfico fornece uma visão histórica e comparativa do fluxo de atendimentos, permitindo identificar tendências e planejar ações futuras.

Todo o conteúdo principal é reativo e integrado com a Sidebar e o sistema de permissões. A exibição dos dados depende da autenticação e do token JWT válido, garantindo segurança no acesso às informações. A combinação de cards, tabelas e gráficos proporciona uma experiência de uso intuitiva, visualmente organizada e funcional, permitindo ao usuário tomar decisões rápidas com base em dados consolidados. Além disso, a estrutura modular do código facilita manutenção e futuras expansões, como a inclusão de novos indicadores ou tipos de visualizações, sem impactar o restante do sistema.



Slide Bar

A Sidebar é um componente essencial do frontend, responsável por fornecer a navegação principal do sistema e permitir que o usuário acesse todas as seções de forma rápida e intuitiva. Ela faz parte do componente Dashboard, mas sua lógica se integra diretamente com o roteamento definido em App.jsx, garantindo controle de permissões e navegação condicional. Visualmente, a Sidebar apresenta um avatar com a letra "M", representando a identidade do sistema e servindo como ponto de referência para o usuário. Logo abaixo, encontram-se os ícones de navegação, cada um direcionando para uma seção específica: Dashboard, Agenda, Pacientes, Procedimentos e Configurações. A navegação é realizada através da função **navigate** do React Router, permitindo que o roteamento ocorra sem recarregar a página. O item ativo é destacado visualmente, facilitando a orientação do usuário sobre a seção atualmente exibida.

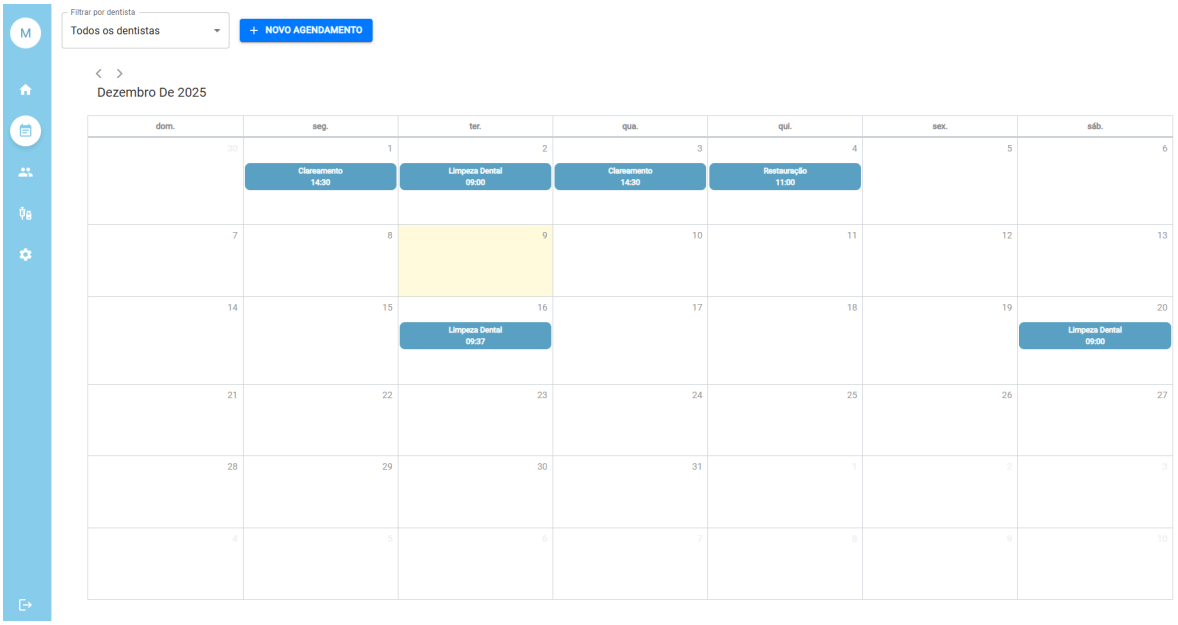
Além da navegação, a Sidebar inclui um botão de logout, que remove o token JWT armazenado no localStorage e redireciona o usuário para a tela de login, garantindo segurança ao invalidar o acesso assim que o usuário decide sair. O fluxo de navegação está integrado com as permissões do usuário, definidas no backend e armazenadas no localStorage. O App.jsx verifica se o usuário está autenticado (isLoggedIn) e se possui permissão para visualizar o Dashboard **(canSeeDashboard)**. Usuários sem a permissão "VIZUALIZAR_DASH" são redirecionados para a tela de Agenda, mesmo que tentem acessar a rota do Dashboard, garantindo que a Sidebar funcione em harmonia com as regras de acesso do sistema.

Agenda

A tela de Agenda foi desenvolvida para permitir que os usuários visualizem, criem, editem e excluam agendamentos de forma intuitiva. Utiliza o FullCalendar para exibir um calendário mensal interativo, onde é possível clicar em datas específicas para ver os compromissos do dia. Ao carregar a página, são feitas requisições ao backend para buscar pacientes, procedimentos, dentistas e agendamentos, sempre utilizando o token JWT para garantir que apenas usuários autenticados tenham acesso aos dados.

A interface mantém estados para controlar listas de eventos, filtros por dentista e modais de visualização e edição. O filtro por dentista facilita a organização, mostrando apenas os agendamentos relacionados a um profissional específico. Os modais permitem visualizar detalhes do agendamento e realizar ações como edição ou exclusão, enquanto outro modal possibilita a criação de novos compromissos com seleção de paciente, procedimento, dentista, data, hora e observações.

A tela também respeita o sistema de permissões do usuário, exibindo o acesso ao Dashboard apenas para quem possui autorização. A sidebar garante navegação rápida para outras telas, mantendo a consistência visual da aplicação. No geral, a Agenda combina clareza, interatividade e segurança, facilitando o gerenciamento eficiente dos compromissos da clínica.



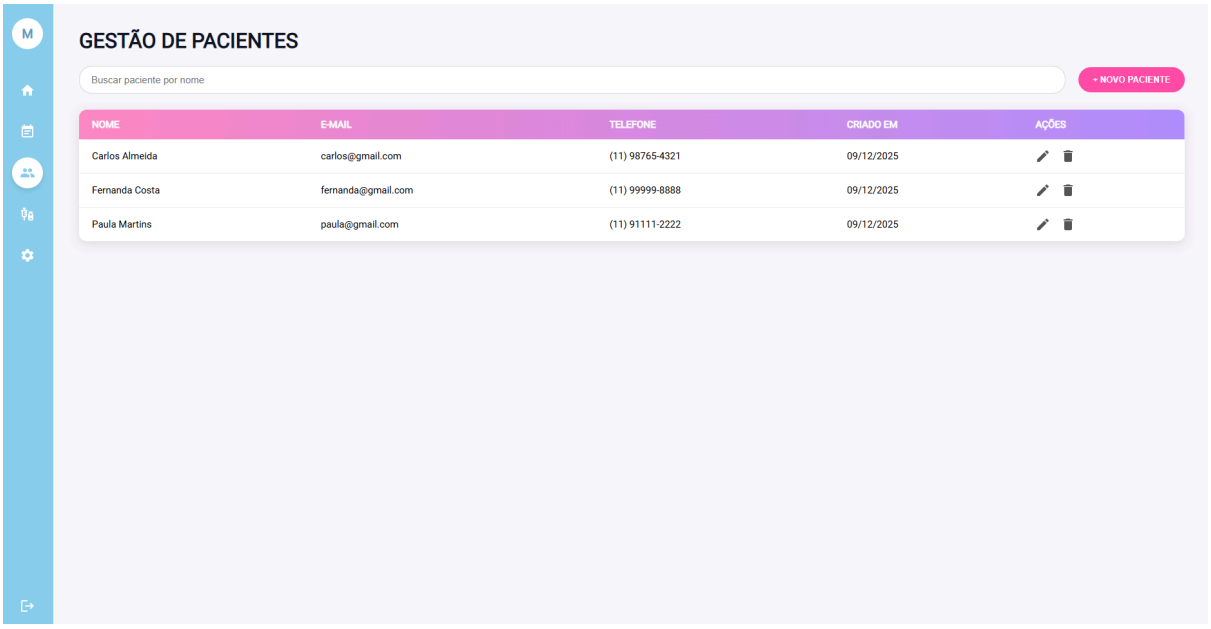
Gestão de Pacientes

A tela de **Pacientes** foi desenvolvida para possibilitar o gerenciamento completo dos dados dos pacientes da clínica, integrando funcionalidades de cadastro, edição, exclusão e busca. Ao carregar a página, a aplicação realiza requisições ao backend para buscar todos os pacientes cadastrados, utilizando o token JWT armazenado no localStorage, garantindo que apenas usuários autenticados possam acessar as informações.

O layout da tela segue a mesma estrutura das outras páginas, com uma **sidebar** de navegação que mantém consistência visual e permite acesso rápido a outras áreas do sistema, como Dashboard, Agenda e Procedimentos. A tela apresenta uma tabela com os pacientes, mostrando informações como nome, e-mail, telefone e data de criação do cadastro. Para facilitar a busca, há um campo que permite filtrar pacientes pelo nome ou CPF, tornando a navegação e localização de registros mais ágil.

A tela também oferece modais interativos para criar novos pacientes e editar informações existentes. Todos os formulários utilizam campos controlados pelo estado do React, permitindo atualização instantânea da interface e validações simples, como formatação de telefone. A exclusão de pacientes é realizada mediante confirmação do usuário, garantindo segurança e evitando remoções acidentais.

O sistema ainda trata erros de permissão, exibindo mensagens claras quando o usuário não possui autorização para acessar os dados, além de mensagens de alerta para falhas de rede ou erros no backend. Dessa forma, a tela de Pacientes combina organização, usabilidade e segurança, proporcionando à equipe da clínica uma ferramenta eficiente para gerenciar informações essenciais dos pacientes.



Gestão de Procedimentos

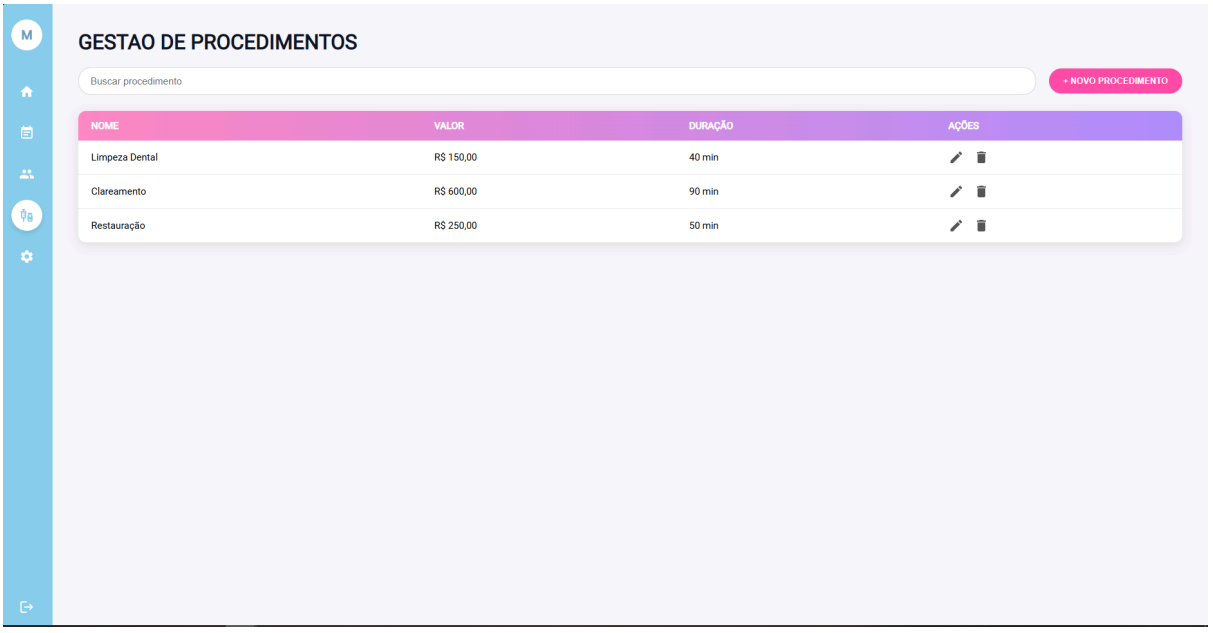
A tela de **Procedimentos** foi projetada para gerenciar todos os serviços oferecidos pela clínica, permitindo cadastrar, editar, excluir e buscar procedimentos de forma eficiente. Assim como nas outras páginas do sistema, a tela utiliza uma **sidebar** de navegação consistente, oferecendo acesso rápido ao Dashboard, Agenda, Pacientes e Configurações, mantendo a uniformidade da interface.

Ao carregar a página, é realizada uma requisição ao backend para obter a lista completa de procedimentos, utilizando o token JWT do usuário autenticado, garantindo segurança no acesso aos dados. Caso o usuário não possua permissão adequada, mensagens claras de erro são exibidas, indicando restrições de acesso ou falhas no carregamento.

A tabela principal exibe os procedimentos cadastrados, mostrando o nome, preço base formatado em reais e duração em minutos. Para facilitar a localização de registros, um campo de busca permite filtrar procedimentos pelo nome.

A interface inclui **modais interativos** para criar novos procedimentos e editar os existentes. Os formulários são controlados pelo estado do React, garantindo atualização imediata da interface e validações básicas de formato e tipo, como conversão de preço e duração para valores numéricos. A exclusão de procedimentos requer confirmação do usuário, prevenindo remoções acidentais.

Dessa forma, a tela de Procedimentos integra usabilidade, segurança e consistência visual, fornecendo à equipe da clínica uma ferramenta completa para gerenciar os serviços disponíveis com agilidade e precisão.



LINK GITHUB : https://github.com/luisscalvi-estudo/Prog22

FRONT-END

1. Estrutura Geral do Projeto

O projeto presente no arquivo ZIP refere-se ao frontend de uma aplicação desenvolvida com React, configurada através do Vite, que é uma ferramenta moderna de construção voltada para desempenho e rápido carregamento durante o desenvolvimento. Observando a estrutura de diretórios, fica claro que o projeto foi organizado de forma profissional, contendo arquivos essenciais como *package.json*, *vite.config.js*, *index.html* e diversos componentes React distribuídos na pasta principal. A presença de dependências como Material UI, FullCalendar, Axios e React Router indica que a aplicação possui interface moderna, comunicação com API externa e mecanismos de navegação bem definidos.

2. Configuração Base do Projeto

A raiz do projeto contém arquivos fundamentais para o funcionamento do ambiente de desenvolvimento. O arquivo *index.html* serve como ponto inicial de carregamento da aplicação React, enquanto o *vite.config.js* controla ajustes de build, otimização e processamento de módulos. O *eslint.config.js* mostra que o projeto utiliza ESLint para manter padronização de código, o que contribui para evitar erros e manter legibilidade. Além disso, arquivos como *.prettierrc* indicam o uso de Prettier para formatação automática, reforçando a preocupação com qualidade e consistência.

3. Dependências e Tecnologias Utilizadas

O *package.json* demonstra claramente as tecnologias centrais utilizadas na construção do frontend. Entre elas, destaca-se o React como biblioteca principal da interface, acompanhado por Material UI, responsável pelos componentes estilizados e responsivos que estruturam botões, formulários, diálogos, tabelas e elementos visuais. Outro ponto essencial é o uso do FullCalendar, que aparece através de múltiplos módulos e permite a exibição de agendas e agendamentos de forma interativa. Adicionalmente, observa-se o Axios, que garante comunicação simplificada com o backend por meio de requisições HTTP. Essas dependências, combinadas, formam um ecossistema robusto que oferece tanto visual moderno quanto integração fluida com o servidor.

4. Arquitetura dos Componentes e Organização da Interface

Embora o repositório analisado contenha uma quantidade extensa de arquivos dentro da pasta *node_modules*, a estrutura do projeto indica que os componentes da aplicação estão organizados na pasta principal do frontend. A interface foi construída com base em módulos reutilizáveis, seguindo boas práticas do React. Cada componente representa uma parte da interface e utiliza hooks modernos, como useState e useEffect, além de recursos visuais do Material UI. A presença de FullCalendar sugere que a aplicação possui uma tela de agenda dinâmica, com eventos carregados a partir da API, o que reforça a ideia de se tratar de um sistema voltado para cadastros, agendamentos e administração de informações.

5. Navegação e Roteamento da Aplicação

O projeto faz uso do React Router (evidenciado nas dependências internas), o que permite a existência de múltiplas telas acessadas por URL, como páginas de login, dashboard, painel de pacientes e demais seções administrativas. Cada rota da aplicação carrega componentes específicos, que se comunicam com o backend utilizando Axios. Esse tipo de arquitetura permite que o frontend seja totalmente desacoplado do servidor, funcionando como cliente REST que busca dados conforme necessidade.

6. Comunicação com o Backend

Um elemento importante na análise é a integração por meio de Axios, que provavelmente faz requisições para endpoints do backend previamente analisado. Com isso, o frontend é capaz de autenticar usuários, carregar dados do dashboard, listar pacientes, exibir agendamentos e realizar operações CRUD. A estrutura encontrada no ZIP demonstra que as requisições estão organizadas em pontos específicos dos componentes, seguindo o padrão de chamadas assíncronas dentro de hooks. O uso dessa biblioteca contribui para uma comunicação eficiente, simples e padronizada entre cliente e servidor.

7. Construção Visual com Material UI

A forte presença do Material UI nos módulos instalados indica que grande parte da interface foi construída com componentes pré-formatados e personalizáveis, como Cards, Buttons, TextFields, Grids e tabelas responsivas. Essa biblioteca proporciona ao sistema uma estética moderna e coerente, além de garantir responsividade nativa, permitindo que a aplicação funcione corretamente em diferentes tamanhos

de tela. O uso de ícones do pacote *@mui/icons-material* reforça a consistência visual do sistema e facilita a identificação das ações realizadas pelo usuário.

8. Funcionalidades Avançadas com FullCalendar

A inclusão de módulos como *@fullcalendar/react*, *@fullcalendar/daygrid* e *@fullcalendar/interaction* mostra que o ambiente possui uma tela orientada ao gerenciamento de agendamentos em calendário. O usuário provavelmente interage com eventos, visualiza horários disponíveis e acessa detalhes específicos sobre cada agendamento. Esse recurso contribui significativamente para a usabilidade do sistema, transformando o calendário em uma ferramenta visual poderosa e que complementa a lógica administrativa presente no backend.

9. Processo de Build e Execução

Como o sistema foi criado com Vite, o processo de desenvolvimento ocorre de maneira rápida e otimizada, com recarregamento instantâneo de tela. Ao executar o comando *npm run dev* ou *yarn dev*, o servidor local é iniciado automaticamente. Para produção, o Vite compila os arquivos em uma versão otimizada e minimizada, garantindo menor tamanho final e carregamento mais eficiente para o usuário final.

10. Conclusão Geral

O frontend disponibilizado no arquivo ZIP demonstra organização, uso de tecnologias modernas e aderência às práticas recomendadas do ecossistema React. A integração eficiente entre Material UI, FullCalendar e Axios indica que se trata de uma aplicação visualmente estruturada, responsiva e preparada para comunicação constante com o backend. A escolha pelo Vite como ferramenta de desenvolvimento otimiza o desempenho e facilita a escalabilidade, enquanto o uso de ESLint e Prettier assegura qualidade no código. De maneira geral, o projeto se apresenta como um frontend sólido, profissional e bem arquitetado, apto para ser utilizado em ambiente acadêmico ou como base em projetos reais.

LINK GIT: https://github.com/luisscalvi-estudo/Prog22