NuTryo - Arquitetura Frontend

Índice

Visão Geral

O frontend do NuTryo é uma aplicação React desenvolvida em TypeScript que gerencia o controle nutricional diário do usuário. A aplicação utiliza uma arquitetura baseada em componentes com separação clara entre lógica de apresentação (componentes), lógica de negócio (controladores) e utilitários.

Tecnologias Principais

- **React 18** com TypeScript
- **Vite** como bundler
- **CSS Modules** para estilização
- **localStorage** para cache de sessão

Estrutura de Diretórios

```
frontend/
├── public/                    # Recursos estáticos
│   ├── favicon/              # Ícones do site
│   ├── icon/                 # Ícones da interface
│   └── logo/                 # Logotipos da aplicação
│
├── src/
│   ├── main.tsx              # Ponto de entrada da aplicação
│   ├── App.tsx               # Componente raiz
│   │
│   ├── components/           # Componentes React
│   │   ├── auth/            # Autenticação
│   │   │   ├── auth.tsx     # Container de autenticação
│   │   │   ├── login.tsx    # Formulário de login
│   │   │   └── registro.tsx # Formulário de registro
│   │   │
│   │   ├── calendario/      # Sistema de calendário
│   │   │   ├── calendario.tsx
│   │   │   └── dia.tsx
│   │   │
│   │   ├── janela/          # Janela principal de refeições
│   │   │   ├── janela.tsx
```

```
│   │   │   ├── aba.tsx
│   │   │   ├── alimentos/   # Gestão de alimentos
│   │   │   └── refeicoes/   # Gestão de refeições
│   │   │       ├── refeicoes.tsx  # Lista de refeições
│   │   │       └── refeicao.tsx   # Edição de refeição
│   │   │
│   │   └── sidebar/         # Barra lateral
│   │       ├── sidebar.tsx
│   │       └── usuario.tsx
│   │
│   ├── controllers/         # Lógica de negócio
│   │   ├── auth/
│   │   │   ├── authController.ts       # Gerenciamento de sessão
│   │   │   ├── loginController.ts      # Lógica de login
│   │   │   └── registerController.ts   # Lógica de registro
│   │   │
│   │   ├── calendario/
│   │   │   └── calendarioController.ts # Lógica do calendário
│   │   │
│   │   └── refeicoes/
│   │       └── refeicoesController.ts  # Lógica de refeições
│   │
│   ├── utils/               # Utilitários
│   │   ├── connection.ts    # Configuração de conexão
│   │   ├── nutryoFetch.ts   # Gerenciamento de dados
│   │   ├── diaObjeto.ts     # Manipulação de objetos de dia
│   │   └── buscarAlimento.ts
│   │
│   └── styles/              # Estilos CSS
│       ├── reset.css
│       ├── document.css
│       ├── auth/
│       ├── calendario/
│       ├── janela/
│       └── sidebar/
│
├── index.html               # HTML principal
├── vite.config.ts          # Configuração Vite
├── tsconfig.json           # Configuração TypeScript
└── package.json            # Dependências
```

Arquitetura da Aplicação

Hierarquia de Componentes

```
App (raiz)
├── Auth (se não autenticado)
│   ├── Login
│   └── Registro
│
└── Main (se autenticado)
```

```
    ├── Sidebar
    │   └── Usuario
    ├── Calendario
    │   └── Dia (múltiplos)
    └── Janela
        ├── Aba (múltiplas)
        ├── Refeicoes (lista)
        │   └── Refeicao (item)
        └── Refeicao (edição)
            └── Alimento (múltiplos)
```

Padrões Arquiteturais

1. **Separação de Responsabilidades**
   - **Componentes**: Apresentação e interação com usuário
   - **Controladores**: Lógica de negócio e estado compartilhado
   - **Utilitários**: Funções auxiliares e gerenciamento de dados
2. **Estado Local vs Global**
   - Estado local gerenciado com `useState` nos componentes
   - Estado compartilhado através de controladores estáticos
   - Cache persistente via localStorage
3. **Comunicação**
   - Parent-to-Child via props
   - Child-to-Parent via callbacks
   - Dados globais via controladores singleton

Componentes Principais

1. App.tsx

Responsabilidade: Componente raiz que gerencia autenticação e renderização condicional.

Estados:

- `authenticated` : Controla se usuário está logado
- `loading` : Controla estado de carregamento
- `dataDisplay` : Data selecionada no calendário

Lógica:

1. Verifica sessão ao montar usando `authController.verificarSessao()`
2. Renderiza `Auth` se não autenticado
3. Renderiza sistema principal se autenticado

Correlações:

- Importa: `Auth` , `Sidebar` , `Calendario` , `Janela` , `CalendarioController` , `authController`
- Propaga: callback `onAuthenticated` para Auth, `onLogout` para Sidebar
- Sincroniza: estado `dataDisplay` entre Calendario e Janela

```
// Fluxo de autenticação
useEffect(() => {
  async function verificarAuth() {
    const temSessao = await authController.verificarSessao();
    setAuthenticated(temSessao);
    setLoading(false);
  }
  verificarAuth();
}, [])
```

---

2. Auth (auth.tsx)

Responsabilidade: Container que alterna entre Login e Registro.

Estados:

- `authMode` : 'login' | 'register'

Lógica:

1. Renderiza Login ou Registro baseado em `authMode`
2. Permite alternância entre modos
3. Notifica App quando autenticação é bem-sucedida

Correlações:

- Recebe: `onAuthenticated` callback do App
- Renderiza: `Login` ou `Registro`
- Propaga: callbacks de navegação entre modos

---

3. Calendario (calendario.tsx)

Responsabilidade: Exibe calendário mensal e permite seleção de dias.

Estados:

- `dias` : Array de objetos de dia do mês
- `diaSelecionado` : Dia atualmente selecionado (dia/mês/ano)

Lógica Principal:

1. **Geração do Calendário**:

```
const [dias, setDias] = useState(() => {
  const dataAtual = new Date();
  return CalendarioController.gerarArrayDias(
    dataAtual.getMonth(),
    dataAtual.getFullYear()
  );
});
```

2. **Navegação entre Meses**:

```
function navegarMes(direcao: "anterior" | "proximo") {
  const novosDias = CalendarioController.navegarMes(direcao);
  setDias(novosDias);
}
```

3. **Seleção de Dia**:
    - Atualiza `CalendarioController.dataSelecionada`
    - Notifica Janela via `setDataDisplay`
    - Gera objeto de dia via `diaObjeto.gerarDia()`
    - Busca refeições do dia via `NutryoFetch`

Correlações:

- Depende: `CalendarioController` (lógica de calendário)
- Depende: `NutryoFetch` (buscar refeições)
- Depende: `diaObjeto` (criar objeto de dia)
- Recebe: `setDataDisplay` callback do App
- Renderiza: múltiplos componentes `Dia`

---

4. Janela (janela.tsx)

Responsabilidade: Gerencia visualização e edição de refeições do dia.

Estados:

- `refeicoes` : Array de refeições do dia
- `abas` : Array de abas abertas (lista + edições)

Lógica Principal:

1. **Carregamento de Refeições**:

```
useEffect(() => {
  async function fetchRefeicoes() {
    const res = await NutryoFetch.retornaRefeicoesDoDia(
      CalendarioController.dataSelecionada
    );
    const normalized = (res ?? []).map((r, idx) => ({
      ...r,
      id: idx + 1,
      uid: r._id ?? `local-${Date.now()}-${idx}`
    }));
    setRefeicoes(normalized);
  }
  fetchRefeicoes();
}, [CalendarioController.dataSelecionada])
```

2. **Sistema de Abas**:
    - Aba fixa (id=0): Lista de refeições
    - Abas dinâmicas: Edição de refeições específicas

- Sincronização de títulos com tipo de refeição

3. **Gerenciamento de Refeições**:
   - Adicionar: Cria nova refeição local
   - Remover: Apaga via `diaObjeto.apagarRefeicao()`
   - Editar: Abre aba de edição específica

Correlações:

- Depende: `NutryoFetch` (buscar refeições)
- Depende: `CalendarioController` (data selecionada)
- Depende: `diaObjeto` (manipular refeições)
- Renderiza: `Aba`, `Refeicoes` (lista), `Refeicao` (edição)
- Recebe: `dataDisplay` do App

---

5. Refeicao (refeicao.tsx)

Responsabilidade: Edição detalhada de uma refeição específica.

Estados:

- `listaAberta`: Controla dropdown de tipo
- `alimentos`: Array de alimentos da refeição

Lógica Principal:

1. **Carregamento de Alimentos**:

```
useEffect(() => {
  if (refeicao?._id) {
    const alimentosBuscados = NutryoFetch.retornaAlimentosDaRefeicao(
      CalendarioController.dataSelecionada,
      refeicao._id
    );
    const normalized = (alimentosBuscados || []).map((a, idx) => ({
      ...a,
      id: idx + 1,
      uid: a._id ?? `local-alimento-${Date.now()}-${idx}`,
      peso: Math.trunc(Number(a.peso)) || 0,
      // ... normaliza valores numéricos
    }));
    setAlimentos(normalized);
  }
}, [refeicao?._id]);
```

2. **Gerenciamento de Alimentos**:
   - Adicionar: Cria novo alimento local
   - Remover: Apaga via `diaObjeto.apagarAlimento()`
   - Atualizar: Modifica nome/peso de alimento

3. **Alteração de Tipo**:
   - Notifica parent via callback `onTipoChange`

- Atualiza título da aba correspondente

Correlações:

- Depende: `NutryoFetch` (buscar alimentos)
- Depende: `CalendarioController` (data selecionada)
- Depende: `diaObjeto` (manipular alimentos)
- Recebe: `refeicao` (dados) e `onTipoChange` (callback)
- Renderiza: múltiplos componentes `Alimento`

---

6. Sidebar (sidebar.tsx)

Responsabilidade: Navegação e acesso a funcionalidades.

Estados:

- `mostrarUsuario` : Controla exibição da janela de usuário

Funcionalidades:

- Logo da aplicação
- Perfil do usuário
- Estatísticas (placeholder)
- Link para GitHub

Correlações:

- Recebe: `onLogout` callback do App
- Renderiza: `Usuario` (quando ativado)

Controladores

1. authController.ts

Responsabilidade: Gerenciamento de autenticação e sessão.

Propriedades Estáticas:

- `email` : Email do usuário
- `nome` : Nome do usuário

Métodos:

```
// Verifica se existe sessão válida no cache
static async verificarSessao(): Promise<boolean>

// Salva sessão no localStorage
static async definirSessao(email, senha, nome): Promise<void>

// Remove sessão do localStorage
static removerSessao(): void
```

Fluxo:

1. Busca credenciais do localStorage

2. Valida com backend via `LoginController.login()`
3. Inicializa dados via `NutryoFetch.iniciar()`

Correlações:

- Usa: `LoginController` (validação)
- Usa: `NutryoFetch` (inicialização de dados)
- Usado por: `App.tsx` (verificação de sessão)

---

2. calendarioController.ts

Responsabilidade: Lógica de calendário e navegação entre datas.

Propriedades Estáticas:

```
static mesAtual: number           // Mês atual (0-11)
static anoAtual: number           // Ano atual
static diaAtual: number           // Dia atual (1-31)
static mesSelecionado: number     // Mês selecionado
static anoSelecionado: number     // Ano selecionado
static diaSelecionado: number     // Dia selecionado
static dataSelecionada: string    // Formato: "dia-mês-ano"
static meses: string[]            // Nomes dos meses
```

Métodos:

1. **navegarMes(direcao)**:
    - Ajusta `mesSelecionado` e `anoSelecionado`
    - Retorna novo array de dias
    - Atualiza `mesStringAtual`
2. **gerarArrayDias(mes, ano)**:
    - Calcula dias do mês anterior, atual e seguinte
    - Retorna array de 42 dias (6 semanas)
    - Cada dia tem: `{dia, index, tipo}`

Algoritmo de Geração:

```
// Exemplo de estrutura retornada:
[
  {dia: 29, index: 0, tipo: "mesAnterior"},
  {dia: 30, index: 1, tipo: "mesAnterior"},
  {dia: 1,  index: 2, tipo: "mesAtual"},
  // ... dias do mês atual ...
  {dia: 31, index: 30, tipo: "mesAtual"},
  {dia: 1,  index: 31, tipo: "mesSeguinte"},
  // ...
]
```

Correlações:

- Usado por: `Calendario.tsx` (geração e navegação)
- Usado por: `Janela.tsx` (data selecionada)
- Usado por: `Refeicao.tsx` (data selecionada)

Utilitários

1. nutryoFetch.ts

Responsabilidade: Camada de abstração para comunicação com backend.

Propriedades Estáticas:

```
static objects: any[]     // Cache de objetos de dias
static metas: any         // Metas do usuário
static username: string   // Nome do usuário
static email: string      // Email do usuário
```

Métodos Principais:

1. **iniciar(email, nome)**:

```
static async iniciar(email: string, nome?: string) {
  NutryoFetch.username = nome ?? '';
  NutryoFetch.email = email;
  return await NutryoFetch.fetchDias(email);
}
```

   - Inicializa dados do usuário
   - Busca todos os dias do backend

2. **fetchDias(email)**:

```
static async fetchDias(email: string) {
  const resposta = await fetch(`${backend}/refeicoes/${email}`);
  const data = await resposta.json();
  NutryoFetch.objects = data;
  diaObjeto.diasSalvos = data;
  return data;
}
```

   - Busca dias do backend
   - Atualiza cache local e `diaObjeto`

3. **retornaRefeicoesDoDia(data)**:

   - Busca refeições de um dia específico no cache
   - Formato da data: "dia-mês-ano"
   - Retorna array de refeições ou undefined

4. **retornaRefeicao(data, refeicaoId)**:

   - Retorna refeição específica de um dia

- Usado para navegação em abas

5. **retornaAlimentosDaRefeicao(data, refeicaoId)**:
   - Retorna array de alimentos de uma refeição
   - Usado para popular lista de alimentos

Correlações:

- Depende: `connection.ts` (URL do backend)
- Depende: `diaObjeto` (sincronização de cache)
- Usado por: `authController`, `Calendario`, `Janela`, `Refeicao`

---

2. diaObjeto.ts

Responsabilidade: Manipulação de objetos de dia e sincronização com backend.

Propriedades Estáticas:

```
static diasSalvos: any[]    // Dias já salvos no backend
static dia: any             // Dia atualmente sendo editado
static refeicoes: any[]     // Refeições do dia atual
static alimentos: object[]  // Alimentos sendo manipulados
static usuario: string      // Email do usuário
```

Estrutura de Objeto de Dia:

```
{
  id: "dia-mês-ano",
  _usuario: "email@exemplo.com",
  refeicoes: [
    {
      _id: "1",
      tipo: "Café da Manhã",
      alimentos: [
        {
          _id: "1",
          alimento: "Pão",
          peso: 50,
          calorias: 150,
          proteinas: 5,
          carboidratos: 30,
          gorduras: 2
        }
      ]
    }
  ]
}
```

Métodos Principais:

1. **gerarDia(data, usuario, corpo)**:

- Cria objeto de dia com estrutura padrão
- Armazena em `diaObjeto.dia`

2. **gerarRefeicao(id, tipo, alimentos)**:
   - Cria objeto de refeição
   - Atualiza dia local via `atualizarDia()`
3. **gerarAlimento(...params)**:
   - Cria objeto de alimento
   - Atualiza refeição via `atualizarDia()`
4. **atualizarDia(campo, local, objeto)**:
   - Atualiza refeição ou alimento no dia local
   - Chama `postarOuEditar()` para sincronizar
5. **postarOuEditar()**:
   - Verifica se dia existe no backend
   - Chama `postarDiaBanco()` ou `editarDiaBanco()`
6. **postarDiaBanco()**:

```
static async postarDiaBanco() {
  await fetch(`${backend}/refeicoes`, {
    method: "POST",
    body: JSON.stringify(diaObjeto.dia)
  });
  await NutryoFetch.fetchDias(diaObjeto.usuario);
}
```

7. **editarDiaBanco()**:

```
static async editarDiaBanco() {
  await fetch(`${backend}/refeicoes/${email}/${diaObjeto.dia.id}`, {
    method: "PUT",
    body: JSON.stringify(diaObjeto.dia)
  });
  await NutryoFetch.fetchDias(diaObjeto.usuario);
}
```

8. **apagarAlimento(refeicao, alimento)**:
   - Remove alimento do array
   - Reindexada IDs para manter ordem
   - Sincroniza com backend
9. **apagarRefeicao(refeicao)**:
   - Remove refeição do array
   - Reindexada IDs
   - Sincroniza com backend
10. **editarTipoRefeicao(idRefeicao, novoTipo)**:

    - Atualiza tipo da refeição
    - Sincroniza com backend

Correlações:

- Depende: `connection.ts` (URL backend)
- Depende: `NutryoFetch` (refetch após mudanças)
- Depende: `CalendarioController` (data selecionada)
- Usado por: `Calendario`, `Janela`, `Refeicao`

Fluxo de Dados

1. Fluxo de Autenticação

```
App.tsx
  ↓ useEffect (mount)
  ↓ verificarSessao()
  ↓
authController.verificarSessao()
  ↓ localStorage.getItem("sessaoNutryo")
  ↓ LoginController.login(email, senha)
  ↓ fetch → backend
  ↓ NutryoFetch.iniciar(email, nome)
  ↓ fetchDias(email)
  ↓ diaObjeto.diasSalvos = data
  ↓
App.tsx
  ↓ setAuthenticated(true)
  ↓ renderiza <main>
```

2. Fluxo de Seleção de Dia

```
Calendario.tsx
  ↓ onClick(dia)
  ↓ selecionarDia(dia)
  ↓
CalendarioController.dataSelecionada = "dia-mês-ano"
  ↓
setDataDisplay("dia/mês/ano")  → App.tsx → Janela.tsx
  ↓
diaObjeto.gerarDia(data, usuario, corpo)
  ↓ NutryoFetch.retornaRefeicoesDoDia(data)
  ↓
Janela.tsx
  ↓ useEffect [CalendarioController.dataSelecionada]
  ↓ fetchRefeicoes()
  ↓ setRefeicoes(normalized)
```

3. Fluxo de Adição de Refeição

```
Janela.tsx
  ↓ onClick(+)
  ↓ handleAddRefeicao()
  ↓ setRefeicoes([...prev, novaRefeicao])
  ↓ render <Refeicoes>
  ↓
Usuário clica em editar
  ↓ abrirAbaEdicao(id)
  ↓ setAbas([...abas, novaAba])
  ↓ render <Refeicao>
  ↓
Usuário altera tipo
  ↓ onTipoChange(id, tipo)
  ↓ handleTipoChange()
  ↓ setRefeicoes (atualiza tipo)
  ↓ useEffect sincroniza títulos das abas
```

4. Fluxo de Adição de Alimento

```
Refeicao.tsx
  ↓ onClick(+)
  ↓ handleAddAlimento()
  ↓ setAlimentos([...prev, novoAlimento])
  ↓ render <Alimento>
  ↓
Alimento.tsx
  ↓ usuário edita nome/peso
  ↓ onBlur()
  ↓ handleSalvar()
  ↓
diaObjeto.gerarAlimento(refeicao, id, nome, peso, ...)
  ↓ atualizarDia("alimento", local, objeto)
  ↓ postarOuEditar()
  ↓
  ├→ postarDiaBanco()  (se novo dia)
  └↳ editarDiaBanco()  (se dia existente)
     ↓ fetch PUT → backend
     ↓ NutryoFetch.fetchDias() (refetch)
     ↓ diaObjeto.diasSalvos = data
```

5. Fluxo de Remoção

```
Refeicao.tsx / Janela.tsx
  ↓ onClick(trash)
  ↓ handleRemoveAlimento() / handleRemoveRefeicao()
  ↓
diaObjeto.apagarAlimento() / apagarRefeicao()
  ↓ filtra array
  ↓ reindexada IDs
  ↓ postarOuEditar()
```

```
  ↓ editarDiaBanco()
  ↓ fetch PUT → backend
  ↓ NutryoFetch.fetchDias()
  ↓
Componente
  ↓ setAlimentos / setRefeicoes (filtered)
```

Correlações entre Arquivos

Mapa de Dependências

```
App.tsx
├─ imports: Auth, Sidebar, Calendario, Janela
├─ imports: CalendarioController, authController
└─ provides: callbacks para child components

Auth.tsx
├─ imports: Login, Registro
└─ provides: callbacks de navegação

Calendario.tsx
├─ imports: Dia, CalendarioController, NutryoFetch, diaObjeto
├─ uses: CalendarioController (geração de dias, navegação)
├─ uses: NutryoFetch (verificar refeições)
└─ uses: diaObjeto (gerar objeto de dia)

Janela.tsx
├─ imports: Aba, Refeicoes, Refeicao
├─ imports: CalendarioController, NutryoFetch, diaObjeto
├─ uses: CalendarioController (data selecionada)
├─ uses: NutryoFetch (buscar refeições)
└─ uses: diaObjeto (apagar refeição)

Refeicao.tsx
├─ imports: Alimento
├─ imports: CalendarioController, NutryoFetch, diaObjeto
├─ uses: CalendarioController (data selecionada)
├─ uses: NutryoFetch (buscar alimentos)
└─ uses: diaObjeto (manipular alimentos)

authController.ts
├─ imports: LoginController, NutryoFetch
├─ uses: LoginController (validação)
└─ uses: NutryoFetch (inicialização)

CalendarioController.ts
└─ standalone (sem dependências)

NutryoFetch.ts
├─ imports: connection, diaObjeto
├─ uses: connection (URL backend)
```

```
└─ uses: diaObjeto (sincronização)

diaObjeto.ts
├─ imports: connection, NutryoFetch, CalendarioController
├─ uses: connection (URL backend)
├─ uses: NutryoFetch (refetch)
└─ uses: CalendarioController (data selecionada)
```

Ciclo de Vida dos Dados

```
Backend (MongoDB)
    ↕ HTTP (GET/POST/PUT)
NutryoFetch (cache)
    ↕ sincronização
diaObjeto (manipulação)
    ↕ comandos
Componentes (UI)
    ↕ eventos de usuário
diaObjeto (atualização)
    ↕ POST/PUT
Backend (persistência)
```

Especificações das Lógicas

1. Sistema de IDs

Refeições:

- IDs sequenciais: 1, 2, 3...
- Reindexação após remoção
- UID para React keys: `local-${Date.now()}-${idx}`

Alimentos:

- IDs sequenciais por refeição: 1, 2, 3...
- Reindexação após remoção
- UID para React keys: `local-alimento-${Date.now()}-${idx}`

2. Sistema de Abas

Aba Fixa (id=0):

- Sempre presente
- Mostra lista de refeições
- Não pode ser fechada

Abas Dinâmicas:

- Uma por refeição em edição
- Título sincronizado com tipo da refeição
- Podem ser fechadas
- Ao fechar, retorna para aba fixa

3. Normalização de Dados

Números:

```
const peso = Math.trunc(Number(a.peso)) || 0;
const calorias = Math.trunc(Number(a.calorias)) || 0;
```

- Truncamento de decimais
- Default para 0 se inválido

Arrays:

```
const normalized = (array || []).map((item, idx) => ({
  ...item,
  id: idx + 1,
  uid: item._id ?? `local-${Date.now()}-${idx}`
}));
```

- Sempre garante array válido
- IDs sequenciais começando em 1
- UIDs únicos para keys do React

4. Sincronização com Backend

Estratégia:

1. Atualização local imediata (optimistic UI)
2. Envio para backend (POST/PUT)
3. Refetch completo para garantir consistência
4. Atualização de cache local

Endpoints:

- `GET /refeicoes/:email` - Buscar todos os dias
- `POST /refeicoes` - Criar novo dia
- `PUT /refeicoes/:email/:diaId` - Atualizar dia

Resumo de Funcionalidades

Módulo de Autenticação

- Login com email/senha
- Registro de novo usuário
- Verificação de sessão via localStorage
- Logout com limpeza de cache

Módulo de Calendário

- Visualização mensal
- Navegação entre meses/anos
- Seleção de dia
- Indicador visual de dias com refeições

Módulo de Refeições

- Listagem de refeições do dia
- Adição/remoção de refeições
- Edição de tipo de refeição
- Sistema de abas para múltiplas edições

Módulo de Alimentos

- Adição/remoção de alimentos
- Edição de nome e peso
- Cálculo automático de macros (planejado)
- Busca de alimentos (planejado)

Persistência

- Sincronização automática com backend
- Cache local via NutryoFetch
- Refetch após mutações
- localStorage para sessão

Documento gerado em: 24/11/2025
Versão da aplicação: Frontend React + TypeScript
Autor: Análise arquitetural do sistema NuTryo