

NuTryo - Arquitetura Backend & API

Índice

1. [Visão Geral](#)
 2. [Estrutura de Diretórios](#)
 3. [Arquitetura dos Serviços](#)
 4. [Backend - Serviço Principal](#)
 5. [API - Serviço de Alimentos](#)
 6. [Modelos de Dados](#)
 7. [Rotas e Endpoints](#)
 8. [Fluxo de Dados](#)
 9. [Deploy e Containerização](#)
-

Visão Geral

O NuTryo utiliza uma **arquitetura de microserviços** com dois serviços Node.js independentes:

Backend (Porta 3001)

- **Responsabilidade:** Gerenciamento de usuários, refeições e metas nutricionais
- **Banco de Dados:** MongoDB Atlas (mongoose)
- **Tecnologias:** Node.js, Express, TypeScript, Mongoose
- **URL Produção:** `https://nutryo2.onrender.com`

API (Porta 3002)

- **Responsabilidade:** Fornecimento de dados nutricionais de alimentos
- **Fonte de Dados:** Arquivo Excel (XLSX) carregado em memória
- **Tecnologias:** Node.js, Express, TypeScript, XLSX
- **URL Produção:** `https://nutryo2-1.onrender.com`

Vantagens da Arquitetura

1. **Separação de Responsabilidades:** Backend gerencia estado do usuário, API fornece dados estáticos
 2. **Escalabilidade:** Serviços podem ser escalados independentemente
 3. **Manutenibilidade:** Mudanças em um serviço não afetam o outro
 4. **Performance:** API sem banco de dados é extremamente rápida
 5. **Deploy Independente:** Cada serviço tem seu próprio ciclo de deploy
-

Estrutura de Diretórios

Backend

```
backend/
├── src/
│   ├── app.ts           # Configuração Express + CORS
│   ├── server.ts        # Inicialização do servidor
│   └──
│       ├── config/
│       └── dbConnect.ts  # Conexão MongoDB
```

```

| | | controllers/
| | |   |─ alimentoController.ts # (Não utilizado - migrado para API)
| | |   |─ metasController.ts   # Lógica de metas nutricionais
| | |   └─ refeicoesController.ts # Lógica de refeições
| | |
| | | models/
| | |   |─ metas.ts              # Schema Mongoose de metas
| | |   |─ refeicoes.ts         # Schema Mongoose de refeições
| | |   └─ usuario.ts           # Schema Mongoose de usuário
| | |
| | | routes/
| | |   |─ index.ts              # Agregador de rotas
| | |   |─ authRoutes.ts        # Rotas de autenticação
| | |   |─ metasRoutes.ts       # Rotas de metas
| | |   └─ RefeicoesRouter.ts   # Rotas de refeições
| | |
| | | utils/
| | |   └─ autenticar.ts         # (Planejado) Middleware de auth
| | |
| | | views/
| | |   └─ redirect.html         # Página de redirecionamento
| | |
| |─ tsconfig.json              # Configuração TypeScript
|─ Dockerfile.backend          # Containerização

```

API

```

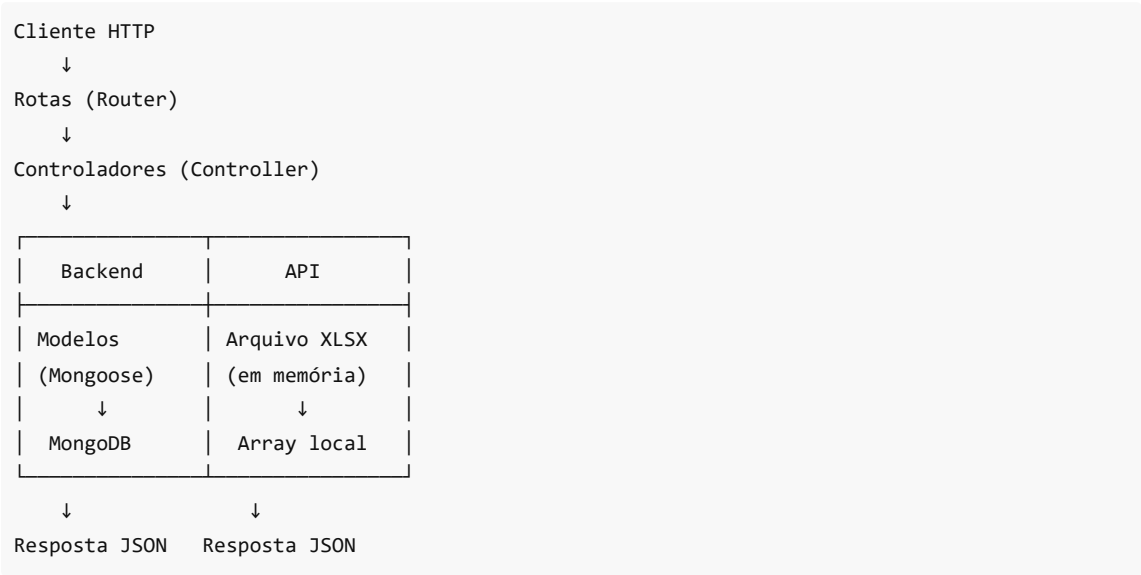
api/
|─ src/
| |─ app.ts                    # Configuração Express + CORS
| |─ server.ts                 # Inicialização do servidor
| |
| |─ config/
| |   └─ dbConnect.ts         # (Não utilizado)
| |
| |─ controllers/
| |   └─ alimentoController.ts # Lógica de busca de alimentos
| |
| |─ data/
| |   └─ alimentos.xlsx       # Tabela nutricional (600+ alimentos)
| |
| |─ routes/
| |   |─ index.ts              # Agregador de rotas
| |   └─ alimentosRoutes.ts   # Rotas de alimentos
| |
| |─ views/
| |   └─ index.html            # Página de status da API
| |
|─ tsconfig.json              # Configuração TypeScript
|─ Dockerfile.api             # Containerização

```

Arquitetura dos Serviços

Padrão MVC (Model-View-Controller)

Ambos os serviços seguem o padrão MVC adaptado:



Camadas

1. **Rotas:** Definem endpoints e mapeiam para controladores
2. **Controladores:** Lógica de negócio e validação
3. **Modelos** (Backend): Schemas Mongoose com validação
4. **Dados** (API): Arquivo Excel carregado na inicialização

Backend - Serviço Principal

Arquivos Principais

1. server.ts

```
import express from "express";
import "dotenv/config.js";
import app from './app.js';

const PORT = 3001;

app.listen(PORT, () => {
  console.log('Servidor rodando na porta 3001');
})
```

Responsabilidades:

- Importa configuração do Express (app.ts)
- Define porta do servidor

- Inicia servidor HTTP

2. app.ts

```
import express from "express";
import connect from '../config/dbConnect.js';
import routes from "../routes/index.js";
import cors from 'cors';
import { fileURLToPath } from 'url';
import { dirname, join } from 'path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

// Cria a conexão
const connection = await connect();

// Eventos de conexão
connection.on("error", (error) => {
  console.log("Erro de conexão " + error);
});

connection.once("open", () => {
  console.log("Conexão com o banco feita com sucesso");
});

const app = express();

// Configuração de CORS
const allowedOrigins = [
  'http://localhost:3000', // desenvolvimento local
  'https://nutryo2-w5pq.onrender.com', // frontend deploy
  'https://nutryo2.onrender.com' // backend deploy
];

app.use(cors({
  origin: (origin, callback) => {
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      console.warn(`Bloqueado por CORS: ${origin}`);
      callback(new Error('CORS não permitido pelo servidor'));
    }
  },
  credentials: true
}));

// Rotas
routes(app);

// Rota raiz - Renderiza página de redirecionamento
```

```
app.get('/', (req, res) => {
  res.sendFile(join(__dirname, 'views', 'redirect.html'));
});

export default app;
```

Responsabilidades:

- Conecta ao MongoDB via `dbConnect.ts`
- Configura CORS para permitir frontend e outros serviços
- Registra rotas da aplicação
- Serve página de redirecionamento na rota raiz
- Exporta app configurado

Configuração CORS:

- Permite requisições do frontend local e em produção
- Permite requisições do próprio backend (para testes)
- Bloqueia origens não autorizadas

3. config/dbConnect.ts

```
import mongoose from "mongoose";

async function connect() {
  mongoose.connect(process.env.DB_CONNECTION_STRING as string);
  return mongoose.connection;
}

export default connect;
```

Responsabilidades:

- Estabelece conexão com MongoDB Atlas
- Usa string de conexão do arquivo `.env`
- Retorna objeto de conexão para listeners

Variáveis de Ambiente:

```
DB_CONNECTION_STRING=mongodb+srv://usuario:senha@cluster.mongodb.net/nutryo
```

Controladores

1. metasController.ts

```
import Metas from "../models/metاس.js";

class MetasController {
  // Listar metas de um usuário
  static async listarMetas(req, res) {
    const email = req.params.email;
```

```

    const metas = await Metas.find({ _usuario: email });
    res.json(metas);
  }

  // Criar novas metas
  static async criarMetas(req, res) {
    const novaMeta = req.body;
    const metaCriada = await Metas.create(novaMeta);
    res.status(201).json(metaCriada);
  }

  // Atualizar metas existentes
  static async atualizarMetas(req, res) {
    const email = req.params.email;
    const metasAtualizadas = req.body;

    await Metas.findOneAndUpdate(
      { _usuario: email },
      metasAtualizadas,
      { new: true }
    );

    res.json({ message: "Metas atualizadas" });
  }
}

export default MetasController;

```

Endpoints Gerenciados:

- GET /metas/:email - Buscar metas do usuário
- POST /metas - Criar novas metas
- PUT /metas/:email - Atualizar metas existentes

Lógica:

- Usa modelo Mongoose `Metas`
- Filtra por `_usuario` (email)
- Retorna array (GET) ou objeto criado/atualizado

2. refeicoesController.ts

```

import Refeicoes from "../models/refeicoes.js";

class RefeicoesController {
  // Listar todas as refeições de um usuário
  static async listar(req, res) {
    const email = req.params.email;
    const refeicoes = await Refeicoes.find({ _usuario: email });
    res.json(refeicoes);
  }
}

```

```

// Criar nova refeição (dia completo)
static async criar(req, res) {
  const novaRefeicao = req.body;
  const refeicaoCriada = await Refeicoes.create(novaRefeicao);
  res.status(201).json(refeicaoCriada);
}

// Atualizar refeição existente
static async atualizar(req, res) {
  const { email, id } = req.params;
  const refeicaoAtualizada = req.body;

  await Refeicoes.findOneAndUpdate(
    { _usuario: email, id: id },
    refeicaoAtualizada,
    { new: true }
  );

  res.json({ message: "Refeição atualizada" });
}
}

export default RefeicoesController;

```

Endpoints Gerenciados:

- GET /refeicoes/:email - Buscar todos os dias do usuário
- POST /refeicoes - Criar novo dia com refeições
- PUT /refeicoes/:email/:id - Atualizar dia específico

Estrutura de Dados:

```

{
  "id": "25-11-2025",
  "_usuario": "user@email.com",
  "refeicoes": [
    {
      "_id": "1",
      "tipo": "Café da Manhã",
      "alimentos": [
        {
          "_id": "1",
          "alimento": "Pão",
          "peso": 50,
          "calorias": 150,
          "proteinas": 5,
          "carboidratos": 30,
          "gorduras": 2
        }
      ]
    }
  ]
}

```

```
]
}
```

Modelos Mongoose

1. models/metast.ts

```
import mongoose from "mongoose";

const metasSchema = new mongoose.Schema({
  _usuario: { type: String, required: true },
  metaCalorias: { type: Number, default: 2000 },
  metaProteinas: { type: Number, default: 150 },
  metaCarboidratos: { type: Number, default: 250 },
  metaGorduras: { type: Number, default: 65 }
}, { versionKey: false });

const Metas = mongoose.model("metas", metasSchema);

export default Metas;
```

Campos:

- `_usuario` : Email do usuário (obrigatório)
- `metaCalorias` : Meta diária de calorias (padrão: 2000 kcal)
- `metaProteinas` : Meta diária de proteínas (padrão: 150g)
- `metaCarboidratos` : Meta diária de carboidratos (padrão: 250g)
- `metaGorduras` : Meta diária de gorduras (padrão: 65g)

Collection MongoDB: `metas`

2. models/refeicoes.ts

```
import mongoose from "mongoose";

const refeicoesSchema = new mongoose.Schema({
  id: { type: String, required: true },
  _usuario: { type: String, required: true },
  refeicoes: [
    {
      _id: String,
      tipo: String,
      alimentos: [
        {
          _id: String,
          alimento: String,
          peso: Number,
          calorias: Number,
          proteinas: Number,
          carboidratos: Number,

```

```

        gorduras: Number
      }
    ]
  }
], { versionKey: false });

const Refeicoes = mongoose.model("refeicoes", refeicoesSchema);

export default Refeicoes;

```

Estrutura Hierárquica:

```

Documento (1 por dia)
├─ id: "dia-mês-ano"
├─ _usuario: "email"
└─ refeicoes: Array
    └─ Refeição
        ├── _id: identificador único
        ├── tipo: "Café da Manhã" | "Almoço" | "Jantar" | etc
        └─ alimentos: Array
            └─ Alimento
                ├── _id: identificador único
                ├── alimento: nome
                ├── peso: gramas
                └─ macros: calorias, proteínas, carboidratos, gorduras

```

Collection MongoDB: `refeicoes`

Rotas

`routes/index.ts`

```

import express from "express";
import authRoutes from "./authRoutes.js";
import metasRoutes from "./metasRoutes.js";
import refeicoesRoutes from "./RefeicoesRouter.js";

const routes = (app) => {
  app.route("/").get((req, res) => res.status(200).send("API NuTryo"));

  app.use(express.json());
  app.use(authRoutes);
  app.use(metasRoutes);
  app.use(refeicoesRoutes);
}

export default routes;

```

Responsabilidades:

- Agrega todas as rotas do sistema
- Configura middleware `express.json()`
- Rota raiz de teste

routes/metasesRoutes.ts

```
import express from "express";
import MetasController from "../controllers/metasesController.js";

const router = express.Router();

router.get("/metases/:email", MetasController.listarMetases);
router.post("/metases", MetasController.criarMetases);
router.put("/metases/:email", MetasController.atualizarMetases);

export default router;
```

routes/RefeicoesRouter.ts

```
import express from "express";
import RefeicoesController from "../controllers/refeicoesController.js";

const router = express.Router();

router.get("/refeicoes/:email", RefeicoesController.listar);
router.post("/refeicoes", RefeicoesController.criar);
router.put("/refeicoes/:email/:id", RefeicoesController.atualizar);

export default router;
```

API - Serviço de Alimentos

Arquivos Principais

1. server.ts

```
import express from "express";
import "dotenv/config.js";
import app from './app.js';

const PORT = 3002;

app.listen(PORT, () => {
  console.log('Servidor rodando na porta 3002');
})
```

Diferenças do Backend:

- Porta diferente (3002)
 - Sem necessidade de variáveis de ambiente
-

2. app.ts

```
import express from "express";
import routes from "../routes/index.js";
import cors from 'cors';
import { fileURLToPath } from 'url';
import { dirname, join } from 'path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const app = express();

// Configuração de CORS (mesma do backend)
const allowedOrigins = [
  'http://localhost:3000',
  'https://nutryo2-w5pq.onrender.com',
  'https://nutryo2.onrender.com'
];

app.use(cors({
  origin: (origin, callback) => {
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      console.warn(`Bloqueado por CORS: ${origin}`);
      callback(new Error('CORS não permitido pelo servidor'));
    }
  },
  credentials: true
}));

// Rotas
routes(app);

// Rota raiz - Página de status da API
app.get('/', (req, res) => {
  res.sendFile(join(__dirname, 'views', 'index.html'));
});

export default app;
```

Diferenças do Backend:

- Sem conexão MongoDB
 - Sem listeners de conexão
 - Página de status ao invés de redirecionamento
-

Controladores

controllers/alimentoController.ts

```
import { Request, Response } from "express";
import XLSX from "xlsx";

interface Alimento {
  id: number;
  nome: string;
  calorias?: string;
  proteínas?: string;
  lipídios?: string;
  colesterol?: string;
  carboidrato?: string;
}

export default class AlimentoController {
  private static alimentos: Alimento[] = AlimentoController.carregarExcel();

  // Carregar Excel na memória (executado uma vez na inicialização)
  private static carregarExcel(): Alimento[] {
    const workbook = XLSX.readFile("api/src/data/alimentos.xlsx");
    const sheetName = workbook.SheetNames[0];
    const sheet = workbook.Sheets[sheetName];
    const data: Alimento[] = XLSX.utils.sheet_to_json(sheet);
    return data;
  }

  // Listar todos os alimentos
  static listar(req: Request, res: Response) {
    res.json(AlimentoController.alimentos);
  }

  // Busca alimento por ID
  static buscarId(req: Request, res: Response) {
    const idParam = req.params.id;
    const id = Number(idParam);

    if (isNaN(id)) {
      return res.status(400).json({ erro: "ID inválido" });
    }

    const alimento = AlimentoController.alimentos.find(a => a.id === id);

    if (!alimento) {
      return res.status(404).json({ erro: "Alimento não encontrado" });
    }

    res.json(alimento);
  }
}
```

```

// Busca fuzzy aproximada
static buscarNome(req: Request, res: Response) {
  const termo = ((req.query.nome as string) || "").toLowerCase();

  // Função de normalização
  function normalizar(str: string) {
    return str
      .normalize("NFD") // remove acentos
      .replace(/[\u0300-\u036f]/g, "")
      .replace(/,/g, " ") // vírgulas viram espaço
      .replace(/\s+/g, " ") // múltiplos espaços viram um
      .trim()
      .toLowerCase();
  }

  const palavrasBusca = normalizar(termo).split(" ");

  const resultados = AlimentoController.alimentos.filter(alimento => {
    const palavrasNome = normalizar(alimento.nome).split(" ");
    // Verifica se todas as palavras do termo estão no nome
    return palavrasBusca.every(palavraBusca =>
      palavrasNome.some(palavraNome => palavraNome.startsWith(palavraBusca))
    );
  });

  res.json(resultados);
}
}

```

Estratégia de Dados:

1. **Carregamento Único:** Excel carregado na inicialização do servidor
2. **Cache em Memória:** Array estático mantido durante toda execução
3. **Performance:** Busca em array JavaScript (muito rápido)
4. **Sem Persistência:** Dados são read-only

Algoritmo de Busca Fuzzy:

1. Normaliza termo de busca (remove acentos, vírgulas, espaços duplos)
2. Divide termo em palavras
3. Normaliza nome de cada alimento
4. Verifica se TODAS as palavras do termo aparecem no nome
5. Usa `startsWith` para match parcial (ex: "arr" encontra "arroz")

Exemplos de Busca:

- `?nome=arr` → encontra "Arroz", "Arroz integral", "Arroz doce"
- `?nome=arroz int` → encontra "Arroz integral"
- `?nome=leite,desnatado` → encontra "Leite desnatado"

Rotas

routes/alimentosRoutes.ts

```
import express from "express";
import AlimentoController from "../controllers/alimentoController.js";

const router = express.Router();

router.get("/alimentos", AlimentoController.listar);
router.get("/alimentos/buscar", AlimentoController.buscarNome);
router.get("/alimentos/:id", AlimentoController.buscarId);

export default router;
```

Endpoints:

- GET /alimentos - Lista todos os alimentos (600+)
- GET /alimentos/buscar?nome={termo} - Busca fuzzy por nome
- GET /alimentos/:id - Busca por ID numérico

Modelos de Dados

Backend - MongoDB

Collection: metas

```
{
  "_id": ObjectId("..."),
  "_usuario": "user@email.com",
  "metaCalorias": 2000,
  "metaProteinas": 150,
  "metaCarboidratos": 250,
  "metaGorduras": 65
}
```

Collection: refeicoes

```
{
  "_id": ObjectId("..."),
  "id": "25-11-2025",
  "_usuario": "user@email.com",
  "refeicoes": [
    {
      "_id": "1",
      "tipo": "Café da Manhã",
      "alimentos": [
        {
          "_id": "1",
          "alimento": "Pão francês",
          "peso": 50,

```

```
      "calorias": 150,
      "proteinas": 5,
      "carboidratos": 30,
      "gorduras": 2
    },
    {
      "_id": "2",
      "alimento": "Manteiga",
      "peso": 10,
      "calorias": 75,
      "proteinas": 0,
      "carboidratos": 0,
      "gorduras": 8
    }
  ]
},
{
  "_id": "2",
  "tipo": "Almoço",
  "alimentos": []
}
]
```

API - Arquivo XLSX

Estrutura do Excel (alimentos.xlsx)

| id | nome | calorias | proteinas | lipidios | colesterol | carboidrato |
|-----|--------------|----------|-----------|----------|------------|-------------|
| 1 | Arroz branco | 130 | 2.5 | 0.2 | 0 | 28 |
| 2 | Feijão preto | 77 | 4.5 | 0.5 | 0 | 14 |
| ... | ... | ... | ... | ... | ... | ... |

Observações:

- Valores são para 100g de alimento
- Frontend calcula proporcionalmente ao peso consumido
- 600+ alimentos cadastrados
- Fonte: Tabela TACO (Tabela Brasileira de Composição de Alimentos)

Rotas e Endpoints

Backend (<https://nutryo2.onrender.com>)

Autenticação

| Método | Endpoint | Descrição | Body/Params |
|--------|-------------|------------------|------------------|
| POST | /auth/login | Login de usuário | { email, senha } |

| | | | |
|------|----------------|--------------------------|------------------------|
| POST | /auth/register | Registro de novo usuário | { email, senha, nome } |
|------|----------------|--------------------------|------------------------|

Metas

| Método | Endpoint | Descrição | Body/Params |
|--------|---------------|-------------------------|----------------------------------|
| GET | /metas/:email | Buscar metas do usuário | email via params |
| POST | /metas | Criar novas metas | { _usuario, metaCalorias, ... } |
| PUT | /metas/:email | Atualizar metas | email via params, metas via body |

Refeições

| Método | Endpoint | Descrição | Body/Params |
|--------|-----------------------|---------------------------------|-------------------------------------|
| GET | /refeicoes/:email | Buscar todos os dias do usuário | email via params |
| POST | /refeicoes | Criar novo dia | { id, _usuario, refeicoes } |
| PUT | /refeicoes/:email/:id | Atualizar dia específico | email e id via params, dia via body |

API (<https://nutryo2-1.onrender.com>)

Alimentos

| Método | Endpoint | Descrição | Query/Params |
|--------|-------------------|---------------------------|---------------|
| GET | /alimentos | Listar todos os alimentos | - |
| GET | /alimentos/buscar | Busca fuzzy por nome | ?nome={termo} |
| GET | /alimentos/:id | Buscar por ID | id via params |

Exemplos de Requisição:

```
# Listar todos
GET https://nutryo2-1.onrender.com/alimentos

# Buscar "arroz"
GET https://nutryo2-1.onrender.com/alimentos/buscar?nome=arroz

# Buscar ID específico
GET https://nutryo2-1.onrender.com/alimentos/123
```

Fluxo de Dados

1. Autenticação

```
Frontend
↓ POST /auth/login { email, senha }
Backend
```

```
↓ verifica credenciais (planejado: bcrypt)
↓ retorna { ok: true, nome: "..."}
Frontend
↓ salva em localStorage
↓ inicializa NutryoFetch
```

2. Carregamento de Dados Inicial

```
Frontend (após login)
↓ GET /refeicoes/:email
Backend
↓ Refeicoes.find({ _usuario: email })
MongoDB
↓ retorna array de dias
Backend
↓ res.json(dias)
Frontend
↓ NutryoFetch.objects = dias
↓ diaObjeto.diasSalvos = dias
```

3. Busca de Alimento

```
Frontend
↓ GET /alimentos/buscar?nome=arroz
API
↓ AlimentoController.buscarNome()
↓ normaliza termo e busca em array
↓ res.json(resultados)
Frontend
↓ exibe lista de resultados
```

4. Adicionar Alimento à Refeição

```
Frontend
↓ usuário seleciona alimento e define peso
↓ diaObjeto.gerarAlimento(...)
↓ diaObjeto.atualizarDia()
↓ diaObjeto.postarOuEditar()
↓
↳ Dia novo: POST /refeicoes
↳ Dia existe: PUT /refeicoes/:email/:id
↓
Backend
↓ Refeicoes.findOneAndUpdate()
MongoDB
↓ atualiza documento
Backend
↓ res.json({ message: "..."})
Frontend
```

```
↓ NutryoFetch.fetchDias() (refetch)
↓ atualiza cache local
```

5. Atualização de Metas

```
Frontend (JanelaEstatisticas)
  ↓ usuário edita meta
  ↓ onBlur do input
  ↓ PUT /metas/:email
  ↓ body: { metaCalorias: 2200, ... }
Backend
  ↓ MetasController.atualizarMetas()
  ↓ Metas.findOneAndUpdate({ _usuario: email }, ...)
MongoDB
  ↓ atualiza documento
Backend
  ↓ res.json({ message: "..." })
Frontend
  ↓ atualiza estado local
```

Deploy e Containerização

Dockerfiles

Dockerfile.backend

```
FROM node:24.3-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

# Compila TypeScript e copia views
RUN npm run build:back:unix

EXPOSE 3001

CMD ["node", "backend/dist/server.js"]
```

Processo de Build:

1. Instala dependências
2. Copia código fonte
3. Compila TypeScript (`tsc -p backend/tsconfig.json`)
4. Copia pasta `views` para `dist/views`
5. Inicia servidor Node

Dockerfile.api

```
FROM node:24.3-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

# Compila TypeScript e copia views + data
RUN npm run clearapi:unix && npm run build:api:unix

EXPOSE 3002

CMD ["node", "api/dist/server.js"]
```

Processo de Build:

1. Instala dependências
2. Copia código fonte (incluindo `alimentos.xlsx`)
3. Limpa dist anterior
4. Compila TypeScript
5. Copia `views` e `data` para dist
6. Inicia servidor Node

Scripts de Build (package.json)

```
{
  "scripts": {
    "build:back:unix": "tsc -p backend/tsconfig.json && npm run copy:views:unix",
    "build:api:unix": "tsc -p api/tsconfig.json && npm run copy:views:api:unix",

    "copy:views:unix": "mkdir -p backend/dist/views && cp -r backend/src/views/* backend/dist/views/",
    "copy:views:api:unix": "mkdir -p api/dist/views && cp -r api/src/views/* api/dist/views/",

    "clearapi:unix": "shx rm -rf api/dist/*",
    "clearback:unix": "shx rm -rf backend/dist/*"
  }
}
```

Importante:

- Scripts Unix usam comandos shell (mkdir, cp, rm)
 - Scripts Windows usam PowerShell
 - Docker sempre usa versão Unix
-

Deploy no Render.com

Backend

Configurações:

- **Type:** Web Service
- **Build Command:** `docker build -f Dockerfile.backend -t nutryo-backend .`
- **Start Command:** Definido no Dockerfile
- **Port:** 3001
- **Environment Variables:**

```
DB_CONNECTION_STRING=mongodb+srv://...
```

API

Configurações:

- **Type:** Web Service
- **Build Command:** `docker build -f Dockerfile.api -t nutryo-api .`
- **Start Command:** Definido no Dockerfile
- **Port:** 3002
- **Environment Variables:** Nenhuma necessária

Monitoramento

Health Checks

Backend:

```
curl https://nutryo2.onrender.com/  
# Retorna página HTML de redirecionamento
```

API:

```
curl https://nutryo2-1.onrender.com/  
# Retorna página HTML de status
```

Endpoints de Teste:

```
# Backend - Listar refeições  
curl https://nutryo2.onrender.com/refeicoes/test@email.com  
  
# API - Listar alimentos  
curl https://nutryo2-1.onrender.com/alimentos
```

Resumo de Funcionalidades

Backend

- ✓ Autenticação de usuários
- ✓ CRUD completo de refeições
- ✓ CRUD completo de metas nutricionais
- ✓ Sincronização com MongoDB Atlas
- ✓ CORS configurado para frontend
- ✓ Página de redirecionamento na rota raiz

API

- ✓ Listagem de 600+ alimentos
- ✓ Busca fuzzy por nome
- ✓ Busca por ID
- ✓ Dados em memória (performance)
- ✓ CORS configurado
- ✓ Página de status na rota raiz

Melhorias Futuras

Backend:

- ☐ Implementar middleware de autenticação JWT
- ☐ Hash de senhas com bcrypt
- ☐ Validação de dados com Joi/Yup
- ☐ Rate limiting
- ☐ Logging estruturado
- ☐ Testes automatizados

API:

- ☐ Cache de resultados de busca
- ☐ Paginação de resultados
- ☐ Filtros adicionais (por macronutriente)
- ☐ Versionamento da API
- ☐ Documentação Swagger/OpenAPI

Documento gerado em: 26/11/2025

Versão dos serviços: Backend + API microservices

Autor: Análise arquitetural do sistema NuTryo