Documentação Detalhada da API-NODEJS

1. Introdução

Esta documentação detalha o funcionamento da API-NODEJS, construída com Node.js, Express e Prisma Client para interagir com um banco de dados MongoDB. A API permite operações CRUD (Create, Read, Update, Delete) no modelo `User`.

---

2. Tecnologias Utilizadas

- **Node.js**: Ambiente de execução JavaScript server-side.
- **Express**: Framework web para criação de rotas e gerenciamento de requisições HTTP.
- **Prisma Client**: ORM para facilitar a interação com o MongoDB.
- **MongoDB**: Banco de dados NoSQL.
- **ECMAScript Modules**: Permite usar `import` e `export` no Node.js.

---

3. Estrutura do Projeto

```
API-NODEJS/
├─ node_modules/
├─ prisma/
│  └─ schema.prisma
├─ server.js
├─ package.json
├─ package-lock.json
└─ .env
```

- `server.js`: Contém todas as rotas e inicialização do servidor.
- `prisma/schema.prisma`: Define o modelo `User` e a conexão com o MongoDB.
- `.env`: Contém a URL de conexão com o MongoDB.

---

4. Schema Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

model User {
  id    String @id @default(auto()) @map("_id") @db.ObjectId
```

```
  email String @unique
  name  String
  age   String
}
```

- `id`: Chave primária do usuário, gerada automaticamente como `ObjectId` do MongoDB. - `email`: Campo único, impedindo duplicações. - `name` e `age`: Informações do usuário.

---

5. Configuração do Servidor (`server.js`)

1. **Importações**

```
import express from "express";
import { PrismaClient } from "@prisma/client";
```

2. `express`: Para criar o servidor HTTP.
3. `PrismaClient`: Para interação com o MongoDB.

4. **Inicialização**

```
const app = express();
const prisma = new PrismaClient();
app.use(express.json());
app.listen(3000);
```

5. `app.use(express.json())` habilita leitura de JSON no corpo das requisições.
6. Porta padrão: 3000.

---

6. Rotas

GET /main

```
app.get("/main", async (req, res) => {
  const users = await prisma.user.findMany();
  res.status(200).json(users);
});
```

- Retorna todos os usuários cadastrados.

POST /main

```
app.post("/main", async (req, res) => {
  try {
```

```
    const newUser = await prisma.user.create({
      data: {
        email: req.body.email,
        name: req.body.name,
        age: req.body.age,
      },
    });
    res.status(201).json(newUser);
  } catch (error) {
    if (error.code === "P2002") {
      return res.status(400).json({ error: "E-mail já cadastrado." });
    }
    res.status(500).json({ error: "Erro ao criar usuário." });
  }
});
```

- Cria um novo usuário, tratando e-mails duplicados com `P2002`.

PUT /main/:id

```
app.put("/main/:id", async (req, res) => {
  try {
    const updatedUser = await prisma.user.update({
      where: { id: req.params.id },
      data: { email: req.body.email, name: req.body.name, age:
req.body.age },
    });
    res.status(200).json(updatedUser);
  } catch (error) {
    if (error.code === "P2002") return res.status(400).json({ error: "E-mail
já está em uso." });
    if (error.code === "P2025") return res.status(404).json({ error:
"Usuário não encontrado." });
    res.status(500).json({ error: "Erro ao atualizar usuário." });
  }
});
```

- Atualiza informações de um usuário específico, tratando duplicação de e-mail e usuário não encontrado.

DELETE /main/:id

```
app.delete("/main/:id", async (req, res) => {
  try {
    await prisma.user.delete({ where: { id: req.params.id } });
    res.status(200).json({ message: "Usuário deletado com sucesso." });
  } catch (error) {
    if (error.code === "P2025") return res.status(404).json({ error:
```

```
"Usuário não encontrado." });
    res.status(500).json({ error: "Erro ao deletar usuário." });
  }
});
```

- Remove um usuário pelo ID, com tratamento para ID inexistente.

---

7. Tratamento de Erros

- `P2002`: Violação de campo único (ex: e-mail duplicado).
- `P2025`: Registro não encontrado (ex: atualizar/deletar usuário inexistente).
- Outros erros: Retornam `500 Internal Server Error`.

---

8. Conclusão

Esta API serve como base para aplicações CRUD com MongoDB usando Prisma Client e Express. Ela inclui tratamento de erros, validação de campos únicos e endpoints completos para gerenciamento do modelo `User`.