Documentação Completa da API-NODEJS

1. Introdução

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

Esta versão vai além do código, explicando conceitos como o que é uma API, ORM, estrutura do projeto e detalhando cada linha do código.

---

2. O que é uma API?

Uma **API (Application Programming Interface)** é um conjunto de regras e protocolos que permite que diferentes sistemas se comuniquem. No contexto web, uma API geralmente recebe requisições HTTP (GET, POST, PUT, DELETE) e retorna respostas em formatos como JSON.

Exemplo: Neste projeto, a API permite que aplicações frontend, como React ou mobile apps, enviem dados de usuários, consultem informações existentes ou atualizem registros.

---

3. Tecnologias Utilizadas

- **Node.js**: Ambiente de execução para JavaScript no servidor.
- **Express**: Framework web que simplifica criação de rotas e gerenciamento de requisições HTTP.
- **Prisma Client**: ORM (Object-Relational Mapping) que facilita o acesso a bancos de dados. Ele traduz comandos JavaScript em consultas MongoDB.
- **MongoDB**: Banco de dados NoSQL orientado a documentos, armazenando dados em JSON-like BSON.
- **ECMAScript Modules**: Permite usar `import` e `export` no Node.js.

3.1 O que é um ORM?

Um **ORM (Object-Relational Mapping)** é uma ferramenta que permite manipular dados de um banco de dados usando objetos de uma linguagem de programação, sem precisar escrever consultas SQL/MongoDB manualmente. No caso do Prisma Client, ele funciona como um ORM para MongoDB, transformando chamadas como `prisma.user.create()` em comandos que o MongoDB entende.

---

4. Estrutura do Projeto

```
API-NODEJS/
├─ node_modules/           # Dependências do projeto
├─ prisma/                 # Configurações do Prisma
│  └─ schema.prisma        # Modelo de dados e conexão com o MongoDB
```

```
├─ server.js               # Código principal da API, contendo rotas e
servidor
├─ package.json            # Configuração do projeto Node.js
├─ package-lock.json       # Controle de versões das dependências
└─ .env                    # Variáveis de ambiente (ex: DATABASE_URL)
```

---

5. Prisma Schema

Arquivo `prisma/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
}
```

- **generator client**: Gera o Prisma Client para uso no código. - **datasource db**: Define o banco de dados (MongoDB) e a URL de conexão. - **model User**: Estrutura dos dados do usuário: - `id`: Chave primária, gerada automaticamente pelo MongoDB. - `email`: Único, impede duplicações. - `name` e `age`: Dados adicionais do usuário.

---

6. Servidor Express (`server.js`)

6.1 Importações

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

- `express`: Permite criar servidor e rotas HTTP. - `PrismaClient`: Permite interagir com o banco de dados MongoDB.

6.2 Inicialização do Servidor

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

- `express.json()`: Habilita o parse automático de JSON nas requisições. - Porta padrão: 3000. - `PrismaClient` cria uma instância para consultar e modificar o banco.

---

7. Rotas Detalhadas

7.1 GET /main

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

- Retorna todos os usuários cadastrados no MongoDB. - `findMany()`: Retorna todos os registros da coleção `User`.

7.2 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. - Trata erros de e-mail duplicado (`P2002`). - `201 Created` indica que o recurso foi criado com sucesso.

7.3 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 usuário existente. - Trata duplicação de e-mail e usuário não encontrado (`P2025`).

7.4 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." });
  }
});
```

- Deleta um usuário pelo ID. - Retorna mensagem de sucesso ou erro.

---

8. 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`.

---

9. Conclusão

Esta documentação detalha não só o funcionamento do código, mas também conceitos fundamentais: - O que é uma API e como ela permite comunicação entre sistemas. - O que é um ORM (Prisma) e como

ele facilita interação com o MongoDB. - Estrutura completa do projeto e funções de cada trecho do código. - Como tratar erros comuns do Prisma.

A API serve como base sólida para aplicações CRUD com MongoDB, podendo ser expandida para funcionalidades mais avançadas.