✅ Prisma

O que é um ORM?

Mapeamento de objetos relacionais ou seja, mapeai as tabela ou coleções em objetos na linguagem que estiver sendo usada.

Prisma ORM vantagem

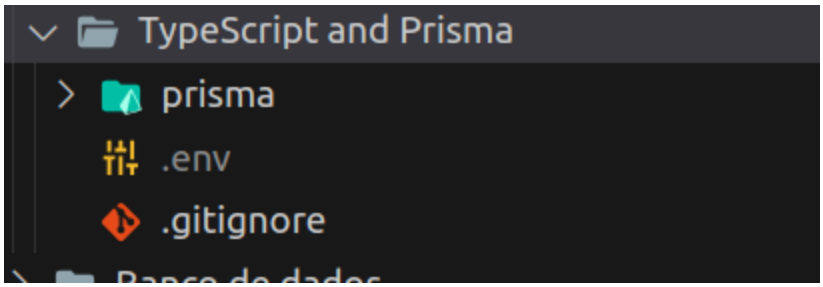
Por conhecer todo o modelo na hora que ele cria o mapeamento em TypeScript ele ira criar todas as tipagens(Get,Put,Post,Delete).

Iniciando com Prisma

Primeiro iremos criar os arquivos do prisma com o seguinte comando:

```
npx init prisma init
```

Após o comando rodar teremos a seguinte estrutura de pastas criadas:



E dentro da pasta `prisma` teremos um arquivo chamado `schema.prisma` que contém nossa conexão com o banco e onde colocaremos nossas tabelas mas antes iremos trocar a nossa conexão do banco e para isso acontecer iremos criar um banco PostegresSQL por meio de Docker e para isso usaremos os seguinte comandos:

```
// Realizarmos o donwload da imagem do Postgres
docker pull postgres
```

```
//Verificar se a imagem foi instalada
docker images
```

```
//Iniciando o container do postgres
docker run -p 5432:5432 -e POSTGRES_PASSWORD=1234 postgres
```

Com o nosso container rodando o banco podemos alterar o nosso `.env` para o seguinte código:

```
DATABASE_URL="postgresql://postgres:1234@localhost:5432/mydb?sch
```

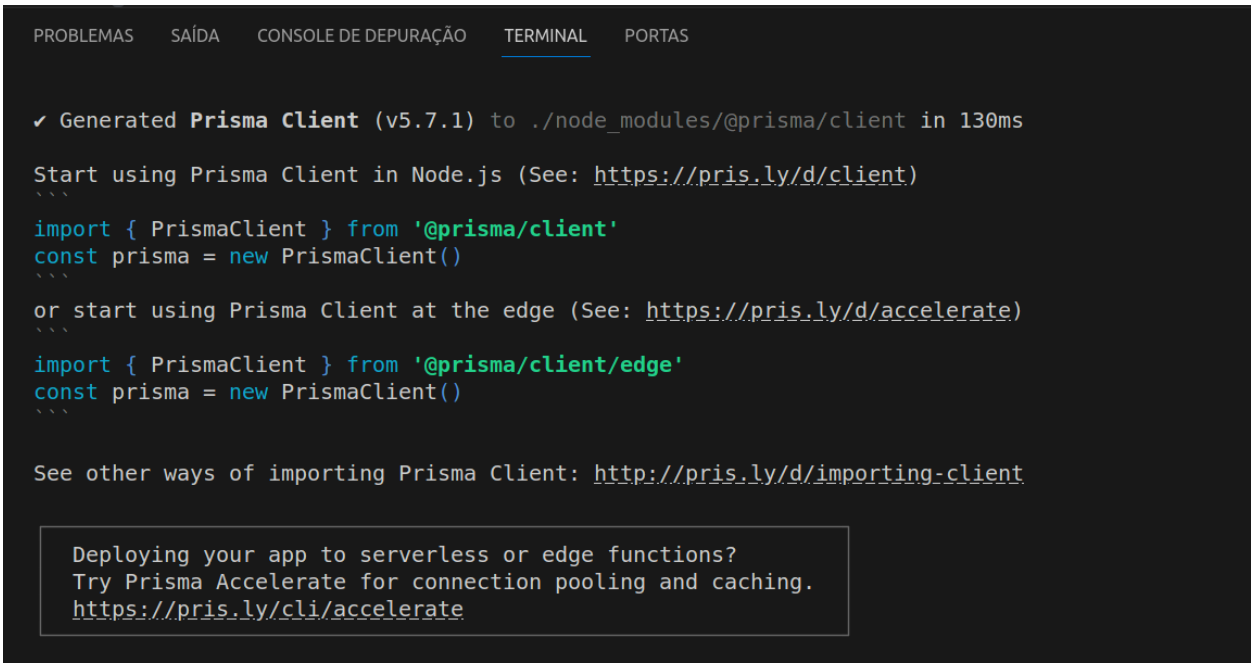
E agora podemos começar a criar nossos modelos no prisma, então começaremos com um simples de produtos:

```
model Produto {
  id Int @id @default(autoincrement()) //Por padrão temos que te
  nome String? // Este campo ele é opcional ou seja não precisa
  Ean String @unique //Ja este campo ele tem que ser unico e dev
}
```

Agora que temos nosso primeiro modelo temos que solicitar ao Prisma que gere este modelo então usaremos o seguinte código:

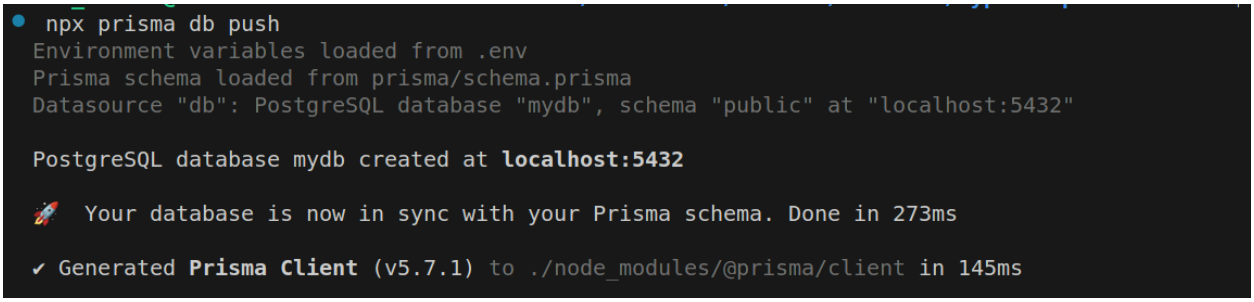
```
npx prisma generate
```

O retorno deve ser algo parecido com isso:



E é aqui que o prisma se destaca ele pegou o model de Produto,ou seja, ele sabe o que é um produto e que possui um Id,um Nome e um Ean e converteu para TypeScript e criou todas as operações e interações com o Banco de dado , se notarmos a imagem a cima, o mesmo até ja disponibiliza o código para chamar o client do Prisma e utilizar as operações e é isso que iremos fazer mas antes temos mais um comando para dar ao prisma afinal não criamos a tabela no nosso banco e para isso:

```
npx prisma db push
```



Agora possuímos nossa tabela criada então iremos criar um aquivo chamado `index.ts` e rodaremos um count na tabela para ver o número de produtos e para isso utilizamos o código abaixo, ou seja, importamos e criamos nossa conexão com o Prisma Client e

em seguida realizamos uma chamada Assíncrona que se auto chama para realizar o count e retornar o valor no console:

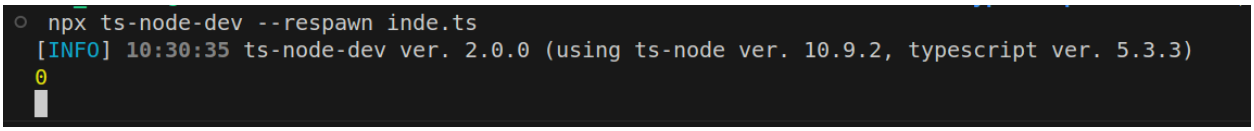
```
import { PrismaClient } from "@prisma/client";
const prisma = new PrismaClient();

(async () => {
  const produto = await prisma.produto.count();
  console.log(produto);
})();
```

Agora podemos dar o seguinte comando para rodarmos nosso `index.ts` :

```
npx ts-node-dev --respawn inde.ts
```

E teremos a seguinte resposta:

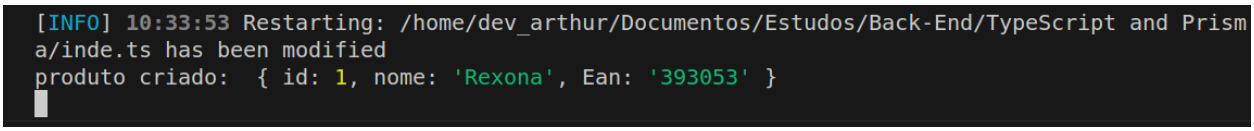


Criando um produto

Agora que ja conectamos e ajustamos tudo vamos criar um produto e inserir o mesmo no banco e para isso utilizaremos as funcionalidades que o Client do Prisma nos oferece:

```
(async () => {
    const produto = await prisma.produto.create({
        data:{
            nome: 'Rexona',
            Ean:'393053'
        }
    });
    console.log("produto criado: ",produto);
  })();
```

E o nosso retorno será:



Antes de seguirmos para o próximo tópico iremos criar mais um produto:

```
(async () => {
  const produto = await prisma.produto.create({
    data: {
      nome: "Nivea Men Black&White",
      Ean: "8795543",
    },
  });
  console.log("produto criado: ", produto);
})();
```

Pesquisando por Produto

Agora temos dois produtos na nossa base de dados e podemos utilizar as ferramentas do Prisma para consulta, ou seja, vamos procurar por um produto onde o Nome do mesmo começa com R:

```
(async () => {
  const produto = await prisma.produto.findMany({
    where: {
      nome: {
        startsWith: "R",
      },
    },
  });
  console.log(produto);
})();
```

Criando relacionamento entre duas tabelas com Prisma

Para isso criaremos um novo Model chamado de Loja e indicaremos que um produto pode possui N lojas e para isso atualizaremos nossos Model das seguintes formas:

```
model Produto {
  id Int @id @default(autoincrement())
  nome String?
  Ean String @unique
  loja Loja[]
}

model Loja {
  id Int @id @default(autoincrement())
  loja String?
  Produto Produto? @relation(fields: [produtoId],references: [id
  produtoId Int?
}
```

E com isso iremos novamente rodas os comandos para gerar o modelo e em seguida as tabelas no banco:

```
npx prisma generate
```

```
npx prisma db push
```

Agora criaremos um único produto que irá possuir uma loja:

```
(async () => {
  const produto = await prisma.produto.create({
    data: {
      nome: "Colagate",
      Ean: "3451",
      loja: {
        create: {
```

```
          loja: "Filial 01",
        },
      },
    },
  });
  console.log(produto);
})();
```

E iremos conferir a nossa tabela de lojas:

```
(async () => {
  const loja = await prisma.loja.findMany({
    where: {
      loja: {
        startsWith: "F",
      },
    },
  });
  console.log(loja);
})();
```

```
[INFO] 11:16:19 Restarting: /home/dev_arthur/Documentos/Estudos/Back-End/TypeScript and Prism
a/index.ts has been modified
[ { id: 1, loja: 'Filial 01', produtoId: 3 } ]
```

E para finalizar iremos procurar por todos os nossos produtos e para isso usaremos um select que retornara o que solicitarmos:

```
(async () => {
  const produto = await prisma.produto.findMany({
    select: {
      id: true,
      nome: true,
      Ean: true,
      loja: true,
```

```
    },
  });
  console.log(produto);
})();
```