| Comandos Prisma!

> “***Uma linguagem não faz seu código ser bom, programadores bons fazem seu código ser bom.***” – Samuel Custódio

Comandos

Inicia um projeto Node

```
npm init -y
```

Comandos Prisma

Instala o framework Express para Node.Js em seu projeto (O Express é um framework web para Node.js que simplifica o desenvolvimento de aplicativos web e APIs.)

```
npm i express
```

Instala TypeScript como dependência de desenvolvimento (São pacotes necessários apenas durante o processo de desenvolvimento, como ferramentas de teste, linters, e compiladores (como o TypeScript).)

```
npm i -D typescript
```

Instala o CORS (Cross-Origin Resource Sharing é uma maneira de permitir que seu site ou API receba pedidos de outros sites)

```
npm i cors
```

Instala ts-node-dev como dependência de desenvolvimento (`ts-node-dev` compila automaticamente seus arquivos TypeScript para JavaScript e reinicia o servidor sempre que detecta mudanças nos arquivos `.ts`)

```
npm i -D ts-node-dev
```

Instala o Prisma como uma dependência de desenvolvimento, uma ferramenta ORM moderna para banco de dados, simplificando o acesso e a manipulação de dados em aplicações Node.js.

```
npm i -D prisma
```

Instala o `@prisma/client`, que é uma biblioteca gerada automaticamente pelo Prisma com base no seu modelo de dados. Ela fornece um cliente de banco de dados para acesso seguro e tipo seguro aos seus dados no banco de dados configurado com Prisma.

```
npm i @prisma/client
```

Inicializa um projeto Prisma com suporte ao banco de dados PostgreSQL.

```
npx prisma init --datasou
```

Cria e aplica uma nova migração no banco de dados, permitindo que você versione e aplique alterações no esquema do banco de dados de forma controlada e automatizada usando o Prisma.

@types São pacotes que fornecem definições de tipos TypeScript para bibliotecas JavaScript. Isso permite que desenvolvedores que estão escrevendo código em TypeScript tenham acesso aos tipos corretos.

```
npm i -D @types/express
npm i -D @types/node
npm i -D @types/cors
```

Instala o pacote dotenv (Ajuda a configurar variáveis de ambiente específicas para diferentes ambientes (desenvolvimento, teste, produção) de forma organizada e segura)

```
npm i dotenv
```

Criar o arquivo .env e inserir o DATABASE_URL e PORT

Criar o arquivo .gitignore e inserir as seguintes informações:

```
/node_modules
.env
/package-lock.json
/yarn.lock
/dist
```

- `/node_modules`:
  - **Motivo**: Este diretório contém todas as dependências do seu projeto instaladas

```
npx prisma migrate dev --
```

Visualizar as models e registros usando o Prisma

```
npx prisma studio
```

Roda a aplicação

```
npm run dev
```

pelo npm ou yarn. Normalmente, é muito grande e pode ser facilmente reconstruído usando os comandos de instalação (`npm install` ou `yarn install`). Incluir `node_modules` no Git resultaria em um repositório muito grande e difícil de gerenciar.

- `.env`:
  - **Motivo**: O arquivo `.env` geralmente contém variáveis de ambiente sensíveis, como chaves de API, senhas, tokens de acesso, etc. Essas informações não devem ser versionadas, pois podem comprometer a segurança do projeto se expostas.
- `/package-lock.json` e `/yarn.lock`:
  - **Motivo**: Esses arquivos são gerados automaticamente pelo npm e yarn para garantir que as instalações subsequentes usem exatamente as mesmas versões das dependências. Eles são específicos para o ambiente local de desenvolvimento e não devem ser versionados para evitar conflitos e problemas de compatibilidade entre diferentes ambientes de desenvolvimento.
- `/dist`:
  - **Motivo**: Este diretório geralmente contém o código compilado ou os arquivos de distribuição do seu projeto. Eles são gerados a partir do código-fonte e podem ser reconstruídos a qualquer momento. Incluir `dist` no Git pode resultar em

duplicação de arquivos e tornar o repositório maior do que o necessário.

Inserir no package.json

```
"scripts": {
    "dev": "ts-node-dev --respawn
  },
```

Inicializa um arquivo de configuração tsconfig.json em projetos TypeScript.

```
npx tsc --init
```

Alterar as linhas

```
29 - "rootDir": "./src"
58 - "outDir": "./dist",
```

Inserir

```
"exclude": ["node_modules"],
"include": ["./src/**/*.ts"]
```

- Exclude: O diretório `node_modules` geralmente contém todas as dependências instaladas do seu projeto, como bibliotecas e frameworks. Incluí-lo na seção `"exclude"` evita que o TypeScript tente compilar esses arquivos, o que seria desnecessário e pode resultar em erros ou compilações demoradas.
- Include: Especifica que todos os arquivos TypeScript (`*.ts`) dentro do diretório `src` e

seus subdiretórios devem ser compilados pelo TypeScript.

Converte seus arquivos TypeScript em JavaScript, colocando os arquivos resultantes no diretório especificado (`dist` neste exemplo).

```
npx tsc
```