Objetivos:

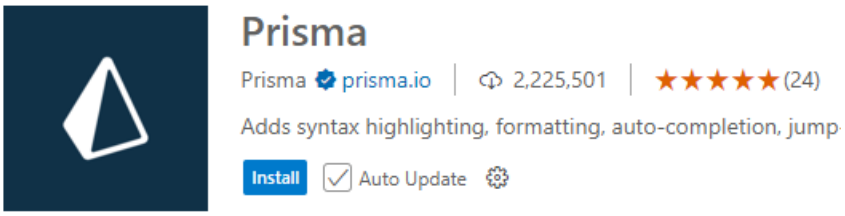
I. Aplicação prática com Prisma
II. Criar e aplicar a migração usando Prisma;
III. Relacionamentos usando Prisma.

i. Aplicação prática com Prisma

O objetivo é criar uma aplicação Node.js, Express, TypeScript e Prisma para fazer o CRUD nas tabelas users e spents do SBGB PostgreSQL.

Antes de tudo:

Instale a extensão do Prisma ORM no VS Code para facilitar o autocompletar:



Para inicialização do projeto, crie uma pasta com o nome de ListaTarefas.

Instale as seguintes dependências:

```
npm init -y
npm install express prisma @prisma/client dotenv
npm install -D typescript ts-node-dev @types/node @types/express
```

Crie o arquivo de configuração do TS:

```
npx tsc –init
```

Crie um BD no PostgreSQL e coloque os parâmetros de conexão no arquivo **.env**. Variáveis declaradas no .env, automaticamente estarão disponíveis para o Prisma:

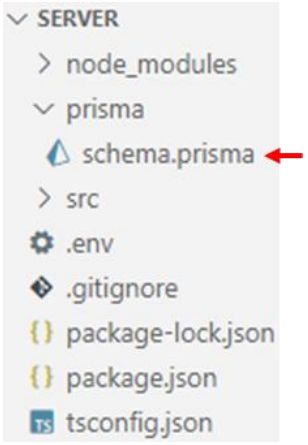
PORT = 3101 -- Porta de comunicação que roda o PostgreSQL
USER=postgres -- Padrão utilizado pelo postgre
HOST=localhost
DATABASE=**todo_db** -- Nome do Banco de Dados
PASSWORD=123
SGBDPORT=5432

Sintaxe → DATABASE_URL=postgresql://${USER}:${PASSWORD}@${HOST}:${SGBDPORT}/${DATABASE}

Arquivo .env:

```
DATABASE_URL="postgresql://postgres:123@localhost:5432/todo_db"
```

Passos para criar o projeto usando Prisma.



Passo 1 – Inicialização do Prima:

```
npx prisma init --datasource-provider postgresql
```

O comando cria a pasta `prisma` com o arquivo de configuração do esquema `schema.prisma`:

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

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

Adicione as seguintes linhas no código

```
model Task {
  id          Int      @id
@default(autoincrement())
  title       String
  description String?
  done        Boolean  @default(false)
  createdAt   DateTime @default(now())
}
```

A URL de conexão com o SGBD está na variável `DATABASE_URL` do `.env`.

A pasta `prisma`: usada para armazenar todos os arquivos relacionados ao Prisma, como o `schema.prisma` e as configurações de migração.

Arquivo `schema.prisma`: é o coração do modelo de dados no Prisma. Nele, definimos as tabelas, campos, relacionamentos e outras informações sobre a estrutura do nosso BD.

Passo 2 – Gerar as migrações e sincronizar o BD:

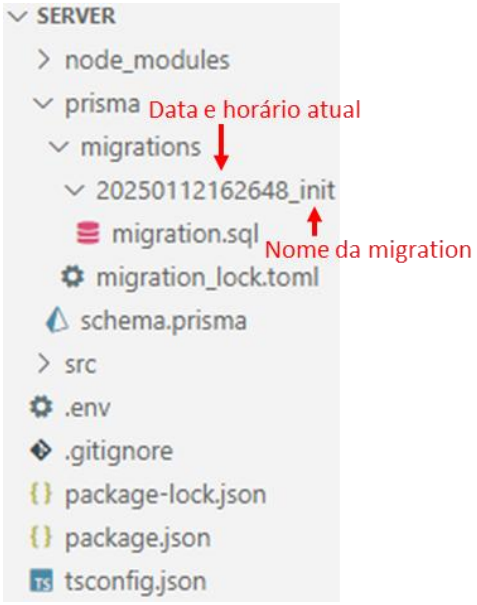
O comando a seguir é utilizado para criar a primeira migração no nosso projeto Prisma e aplicar as alterações definidas no seu schema.prisma diretamente ao BD.

```
npx prisma migrate dev --name init
```

Sobre o comando:

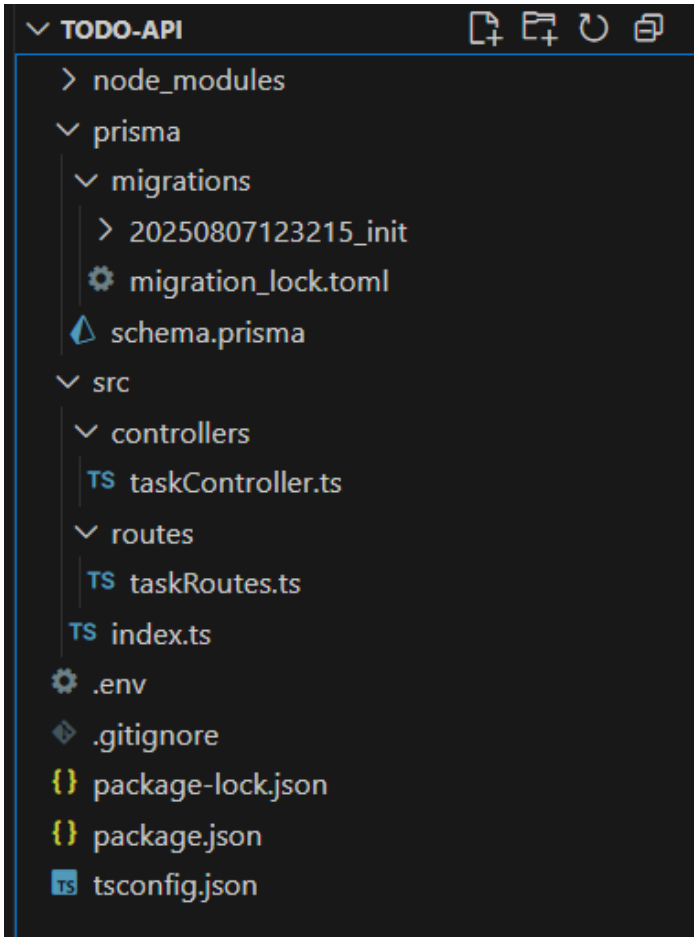
- `prisma migrate dev`: comando usado para criar e aplicar migrações. O parâmetro `dev` indica que a migração será aplicada diretamente ao BD, sem a necessidade de um estágio intermediário;
- `--name init`: este parâmetro define o nome da migração como "init". A migração recebe como nome a data e horário atual + o nome usado para identificar a migração.

O comando também cria o esquema das tabelas no BD do SGBD e a tabela `_prisma_migrations` para armazenar os dados das migrações.



O comando também cria o esquema das tabelas no BD do SGBD e a tabela `_prisma_migrations` para armazenar os dados das migrações.

Abaixo tem-se uma sugestão de estrutura de pastas e arquivos:



Edite o arquivo **tsconfig.json** para garantir que o código será compilado corretamente. Certifique-se de configurar as seguintes propriedades:

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "module": "commonjs",
    "target": "ES6",
    "types": [],
    "sourceMap": true,
    "declaration": true,
    "declarationMap": true,

    "noUncheckedIndexedAccess": true,
    "exactOptionalPropertyTypes": true,

    "strict": true,
    "jsx": "react-jsx",
    "verbatimModuleSyntax": false,
    "isolatedModules": true,
    "noUncheckedSideEffectImports": true,
    "moduleDetection": "force",
    "skipLibCheck": true,
    "esModuleInterop": true,
    "moduleResolution": "node",
  }
}
```

A propriedade **scripts** do arquivo **package.json** deverá ter as seguintes propriedades para executar o projeto:

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "ts-node-dev --respawn src/index.ts",
    "seed": "ts-node prisma/seed.ts"
  },
```

A seguir tem-se o código dos arquivos do projeto.

Vamos criar o servidor com Express. Para isso Vamos criar o arquivo index.ts na raiz da pasta src e colocaremos os seguintes códigos:

src/index.ts

```
import express from "express";
import dotenv from "dotenv";
import { taskRoutes } from "./routes/taskRoutes";

dotenv.config();
const app = express();
app.use(express.json());

app.get("/", (req, res) => {
  res.send("API To-Do List está rodando! Acesse /task para usar.");
});

app.use("/tasks", taskRoutes);

app.listen(3000, () => {
  console.log("Servidor rodando em http://localhost:3000");
});
```

src/controllers/taskController.ts

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

const prisma = new PrismaClient();

export const getTasks = async (req: Request, res: Response) => {
  const tasks = await prisma.task.findMany();
  console.log("Tarefas encontradas:", tasks); // debug
  res.json(tasks);
};

export const createTask = async (req: Request, res: Response) => {
  const { title, description } = req.body;
  const task = await prisma.task.create({
```

```
    data: { title, description },
  });
  res.status(201).json(task);
};

export const updateTask = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { title, description, done } = req.body;
  const task = await prisma.task.update({
    where: { id: Number(id) },
    data: { title, description, done },
  });
  res.json(task);
};

export const deleteTask = async (req: Request, res: Response) => {
  const { id } = req.params;
  await prisma.task.delete({ where: { id: Number(id) } });
  res.status(204).send();
};
```

src/routes/taskRoutes.ts

```
import express from "express";
import {
  getTasks,
  createTask,
  updateTask,
  deleteTask
} from "../controllers/taskController";

const router = express.Router();

router.get("/", getTasks);
router.post("/", createTask);
router.put("/:id", updateTask);
router.delete("/:id", deleteTask);
export const taskRoutes = router;
```

Antes de executar o nosso projeto, vamos abastecer o pgAdmin:

1. Abra o pgAdmin 4
2. Vá até o banco todo_db ➜ Schemas ➜ Tables ➜ Task
3. Clique com o botão direito ➜ View/Edit Data ➜ All Rows
4. Preencha um registro diretamente:
    - title: "Estudar Prisma"
    - description: "Aprender o básico do ORM Prisma"
    - done: false (ou true)
    - createdAt: clique e ele preencherá automaticamente

Não esqueça de salvar os dados inseridos.

Execute a aplicação utilizando o seguinte comando:

```
npm run dev
```

Acesse no navegador:

```
http://localhost:3000/tasks
```