Descrição do Projeto

Este projeto foi realizado como parte do estudo do curso de formação em Node.js da Rocketseat. Embora tenha sido desenvolvido seguindo o conteúdo do curso, este documento irá detalhar todas as tecnologias utilizadas. Isso permitirá refazer todo o conteúdo sem a necessidade de consultar novamente o material do curso.

Detalhamento do Projeto

Inicializamos o projeto com `npm init -y`, gerando o arquivo `package.json`. Em seguida, instalamos as dependências de desenvolvimento `Typescript`, `@types/node`, `ts-node`, `eslint` e `@rocketseat/eslint-config` utilizando o comando `npm install --save-dev`. As dependências do projeto, `knex`, `sqlite3`, `zod`, `fastify` e `dotenv`, foram instaladas com o comando `npm install`.

Criamos a estrutura de pastas do projeto começando por `src` e `database`. Dentro da pasta `source`, criamos os arquivos `server.ts` e `database.ts`:

- `database.ts` será nosso arquivo de configuração do banco de dados.
- Utilizaremos o banco de dados sqlite3.

Inicializamos o arquivo de configuração do Knex com o comando `npx knex init -x ts`, que gerará o arquivo `knexfile.ts`. Nele, teremos a configuração do banco de dados. Movemos o objeto de configuração deste arquivo para o arquivo `database.ts`.

Para criar nossas migrations com TypeScript, adicionamos a seguinte configuração ao arquivo `database.ts`:

```
{
  "migration": {
    "directory": "caminho do diretório desejado ex: database/migration",
    "extension": ".ts"
  }
}
```

Também no arquivo `database.ts`, encontraremos a configuração do cliente do banco de dados e o caminho onde o banco será criado. É importante observar que a pasta onde o arquivo `.db` será

criado precisa existir previamente, pois o Knex não a criará automaticamente quando passamos apenas o caminho do arquivo.

```
{
  "client": "aqui escolhemos nosso banco de dados. Recomendo ler a documentação do Knex caso se
  "connection": {
    "filename": "aqui a pasta onde queremos que o arquivo de db seja criado precisa ser criada
  }
}
```

Criando tabela em migration do Knex

Se estivermos usando outra biblioteca como o Prisma, não faremos modificações dentro do arquivo de migration. No entanto, com Knex, precisaremos criar nossas próprias tabelas.

```
import type { Knex } from 'knex'

export async function up(knex: Knex): Promise<void> {
  await knex.schema.createTable('transactions', (table) => {
    table.uuid('id').primary()
    table.text('title').notNullable()
    table.decimal('amount', 10, 2).notNullable()
    table.timestamp('created_at').defaultTo(knex.fn.now()).notNullable()
  })
}

export async function down(knex: Knex): Promise<void> {
  await knex.schema.dropTable('transactions')
}
```

Nossas migrations deverão sempre conter um método `up` e um método `down`. Após definir a estrutura da tabela, rodamos o comando `npx knex migrate:latest` para criar a tabela no banco de dados.

Caso precisemos realizar mudanças na migration, como adição de colunas ou alteração de dados, rodamos o comando `npx knex migrate:rollback`, realizamos as modificações dentro da migration e, em seguida, executamos o comando `npx knex migrate:latest` novamente.

Após a configuração das migrations e do banco de dados, criamos os arquivos `.env` e `.env.example`. As variáveis de ambiente contidas no arquivo `.env` não serão adicionadas ao repositório Git. Portanto, criamos o arquivo `.env.example` e passamos todas as variáveis de ambiente necessárias.

No arquivo `.env.example`, deixamos as variáveis sensíveis sem valor, apenas com a declaração da variável. Certificamo-nos de adicionar o arquivo `.env` ao arquivo `.gitignore`, juntamente com a pasta `node_modules/`.

Após a criação das variáveis de ambiente no arquivo `.env`, criamos um diretório `env` dentro de `src` e dentro desse diretório o arquivo `index.ts`. Este arquivo será responsável por validar as variáveis e fazer a conexão delas com outros arquivos.

```
import 'dotenv/config'
import { z } from 'zod'

const envSchema = z.object({
  NODE_ENV: z.enum(['development', 'production', 'test']).default('production'),
  DATABASE_CLIENT: z.string(),
  DATABASE_URL: z.string(),
  MIGRATIONS_URL: z.string(),
  MIGRATIONS_EXT: z.string(),
  PORT: z.number().default(3000),
  HOST: z.enum(['localhost', '127.0.0.1']).default('localhost'),
})

export const _env = envSchema.safeParse(process.env)

if (!_env.success) {
  console.log('Alguma variável de ambiente não foi encontrada. ', _env.error.format())
  throw new Error('Alguma variável de ambiente não foi encontrada. ')
}

export const env = _env.data
```

Para acessar essas variáveis em nossos arquivos `server.ts` e `database.ts`, realizamos o import: `import { env } from './env'`.

Após essas configurações estamos prontos para cirar querys em nossas tabelas, criar usuários e etc...