05

Faça como eu fiz: Encerramento

Agora vamos alterar o LivrosService para usar o modelo Livro.

O primeiro passo é criar um construtor que recebe o modelo injetado através do decorator @InjectModel do pacote @nestjs/sequelize.

```
constructor(
    @InjectModel(Livro)
    private livroModel: typeof Livro
) {}
```

COPIAR CÓDIGO

O array de livros não será mais necessário, pois agora vamos ler do banco de dados, portanto, podemos excluí-lo.

```
livros: Livro[] =
[
```

```
        new Livro("LIV01", "Livro TDD e BDD
        new Livro("LIV02", "Livro Iniciando
        new Livro("LIV03", "Inteligência ar
    ];
```

COPIAR CÓDIGO

No método obterTodos vamos retornar this.livroModel.findAll(). Este método devolve uma Promise de Livros, logo, também vamos alterar o tipo de retorno do método e torná-lo assíncrono, através da palavra chave async.

```
  async obterTodos(): Promise<Livro[]> {
      return this.livroModel.findAll();
  }
```

COPIAR CÓDIGO

Nos demais métodos também vamos tornar async e retornar uma promise.

No obterUm usamos o método findByPk ao invés do findAll, ele irá retornar apenas 1 livro, conforme o valor passado no parâmetro id.

```
async obterUm(id: number): Promise<Livr
    return this.livroModel.findByPk(id)
}
```

COPIAR CÓDIGO

No criar usamos o método create.

```
async criar(produto: Livro) {
    this.livroModel.create(Livro);
}
```

COPIAR CÓDIGO

No método alterar, utilizaremos o método update do Sequelize, ele retorna uma promise que contém um array com dois valores, o número de linhas atualizado e o array de livros atualizados, então vamos ter que ajustar o retorno do nosso método.

```
async alterar(livro: Livro): Promise<[n
    return this.livroModel.update(livro
        where: {
```

```
                id: livro.id
            }
        });
    }
```

COPIAR CÓDIGO

No método apagar, vamos primeiro carregar o livro e em seguida chamar o método destroy.

```
  async apagar(id: number) {
      const livro: Livro = await this.obt
      livro.destroy();
  }
```

COPIAR CÓDIGO

Nosso serviço está sem erros, mas como ele teve os tipos de retorno dos métodos alterados será necessário refatorar o nosso controller. Basicamente o que precisaremos fazer é tornar os métodos assíncronos e ajustar os tipos de retorno de acordo com o service.

A versão final do controller ficará assim:

```
import { Body, Controller, Delete, Get,
import { Livro } from './livro.model';
import { LivrosService } from "./livros


@Controller('livros')
export class LivrosController {
    constructor(private livrosService:

    }


    @Get()
    async obterTodos(): Promise<Livro[]
        return this.livrosService.obter
    }


    @Get(':id')
    async obterUm(@Param() params): Pro
        return this.livrosService.obter
    }


    @Post()
    async criar(@Body() livro: Livro) {
```

```
        this.livrosService.criar(livro)
    }


    @Put()
    async alterar(@Body() livro: Livro)
        return this.livrosService.alter
    }


    @Delete(':id')
    async apagar(@Param() params) {
        this.livrosService.apagar(param
    }
}
```

COPIAR CÓDIGO

Agora precisamos pedir explicitamente ao Sequelize que sincronize o banco de dados quando a aplicação subir.

Na classe AppService vamos criar um método sincronizaBancoDados e chamá-lo no construtor.

```
constructor(private sequelize: Sequeliz
    this.sincronizaBancoDados();
}

async sincronizaBancoDados() {
    await this.sequelize.sync();
}
```

COPIAR CÓDIGO

Configurações

A aplicação está funcionando em nosso ambiente, mas e se quisermos utilizá-la em outro ambiente, com configurações de banco de dados diferentes? Bom, aí teríamos que alterar essas configurações no arquivo app.module.ts, isso funciona, mas não é uma boa prática manter valores de configurações fixas no código fonte, as boas práticas dizem para manteremos isso em um arquivo específico de configuração. Em aplicações Node é comum utilizar um arquivo chamado .env para guardar essas configurações e é isso que vamos fazer. O Nest disponibiliza um

pacote para cuidar das configurações, vamos instalá-lo com o seguinte comando:

```
npm i --save @nestjs/config
```

COPIAR CÓDIGO

Com a dependência instalada, vamos adicionar `ConfigModule.forRoot()` no array imports do decorator `@Module` do arquivo `app.module.ts` .

Na pasta raíz do projeto, a mesma onde está o arquivo package.json, por exemplo, vamos criar um arquivo chamado .env para guardar as nossas configurações. Neste arquivo usamos o padrão CHAVE=VALOR, vamos passar para lá algumas configurações de banco de dados.

- USUARIO_BANCO_DADOS=root
- SENHA_BANCO_DADOS=root

Agora nos pontos onde estão fixos o usuário e a senha de acesso ao banco de dados, vamos usar

process.env.USUARIO_BANCO_DADOS e process.env.SENHA_BANCO_DADOS.

A configuração do SequelizeModule ficará assim:

```
SequelizeModule.forRoot({
    dialect: 'mysql', // dialeto do ban
    host: 'localhost', // endereço do s
    port: 3307, // porta do banco de da
    username: process.env.USUARIO_BANCO
    password: process.env.SENHA_BANCO_D
    database: 'livraria', // nome do ba
    models: [ Livro ],
}),
```

COPIAR CÓDIGO

Pronto, agora ficou mais fácil utilizar a nossa aplicação em ambientes diferentes, basta ajustar as configurações no arquivo .env.