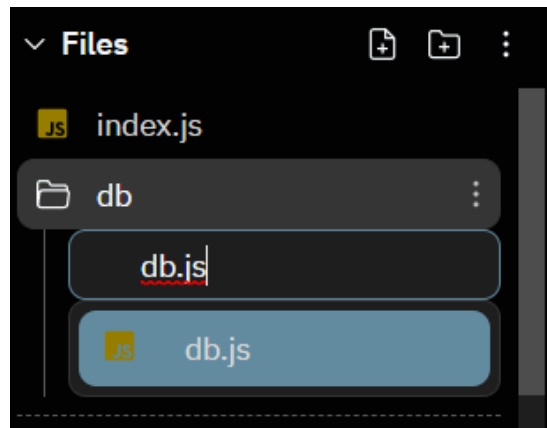


Criando o banco de dados e o model da API

Após a criação do servidor e conexão da nossa API com o Postman, vamos criar o banco da nossa API.

Crie uma pasta chamada 'db' e dentro dela crie um arquivo chamado **db.js**



Vamos criar o banco de dados chamado 'mercado.sqlite'.

O código de criação é o mesmo já utilizado nas aulas do módulo III.

```
db > JS db.js > ...  
1 // BIBLIOTECAS/MODULOS UTILIZADOS  
2 const Sequelize = require('sequelize');  
3 //CRIANDO A CONFIGURAÇÃO DO BANCO DE DADOS  
4 const sequelize = new Sequelize({  
5   dialect: 'sqlite',  
6   storage: './mercado.sqlite'  
7 })  
8 //TRATANDO POSSÍVEIS ERROS E AUTENTICANDO NO BANCO  
9 try {  
10   sequelize.authenticate();  
11   console.log("Banco de dados conectado com sucesso!");  
12 }  
13 catch (erro) {  
14   console.log("Erro ao conectar ao banco",erro);  
15 }  
16 module.exports = sequelize;  
17
```

```
// BIBLIOTECAS/MODULOS UTILIZADOS
const Sequelize = require('sequelize');
//CRIANDO A CONFIGURAÇÃO DO BANCO DE DADOS
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: './mercado.sqlite'
})
//TRATANDO POSSÍVEIS ERROS E AUTENTICANDO NO BANCO
try {
  sequelize.authenticate();
  console.log("Banco de dados conectado com sucesso!");
}
catch (erro) {
  console.log("Erro ao conectar ao banco",erro);
}
module.exports = sequelize;
```

No arquivo index.js, vamos criar nossa biblioteca de utilização do arquivo db.js e nosso sincronismo com o banco de dados.

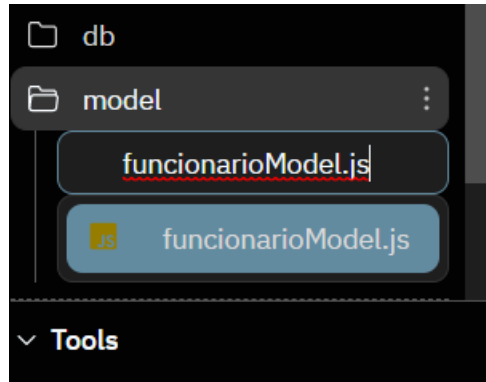
```
3
4 //BIBLIOTECAS/MODULOS UTILIZADOS
5 const database = require("../db/db");
6
7 //SINCRONISMO COM O BANCO DE DADOS
8 try {
9   database.sync().then(() => {
10
11   })
12 }
13 catch(erro) {
14   console.log("Houve uma falha ao sincronizar com o banco de dados. ", erro);
15 };
16 |
```

```
//BIBLIOTECAS/MODULOS UTILIZADOS
const database = require("../db/db");
//SINCRONISMO COM O BANCO DE DADOS
try {
  database.sync().then(() => {
  })
}
catch(erro) {
  console.log("Houve uma falha ao sincronizar com o banco de dados. ", erro);
```

```
};
```

Crie agora a pasta 'model' e dentro dela o arquivo **funcionarioModel.js**.

O arquivo **funcionarioModel.js** será responsável pela estrutura da nossa tabela "funcionário".



Dentro do arquivo **funcionarioModel.js**, vamos criar a estrutura da nossa tabela "funcionários".

```
const Sequelize = require('sequelize');
const database = require('../db/db');

const Funcionario = database.define('funcionario', {
  matricula: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  nome: {
    type: Sequelize.STRING,
    allowNull: false,
  },
  endereco: {
    type: Sequelize.STRING,
    allowNull: false
  },
  telefone: {
    type: Sequelize.STRING,
```

```
    allowNull: false
  },
  email: {
    type: Sequelize.STRING
  },
  nascimento: {
    type: Sequelize.DATE,
    allowNull: false
  }
}, {database, modelName: 'funcionario', tableName: 'funcionarios'})
module.exports = Funcionario;
```

Agora no **index.js**, inclua o módulo **funcionarioModel.js**.

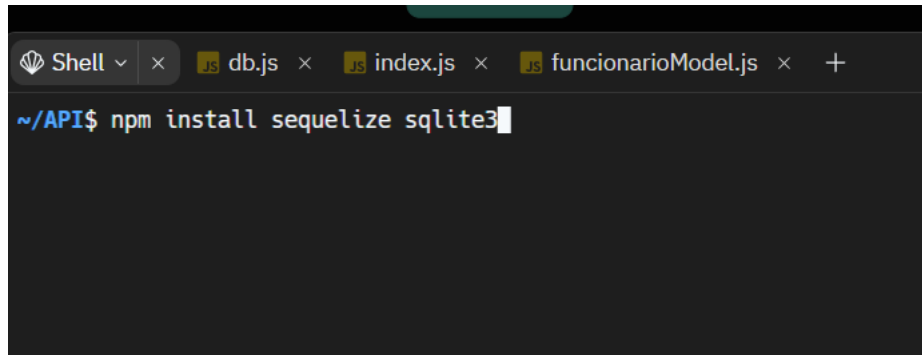
```
const Funcionario = require("../model/funcionarioModel");
```

O arquivo index.js completo é mostrado abaixo.

```
const express = require("express");
const app = express();
//BIBLIOTECAS/MODULOS UTILIZADOS
const database = require("../db/db");
const Funcionario = require("../model/funcionarioModel");
//SINCRONISMO COM O BANCO DE DADOS
try {
  database.sync().then(() => {
  })
}
catch(erro) {
  console.log("Houve uma falha ao sincronizar com o banco de dados. ", erro);
};
app.get("/", (req, res) => {
  //return res.send("Olá Mundo!");
  return res.json({message: "Olá Mundo!"});
})
app.listen(3000);
```

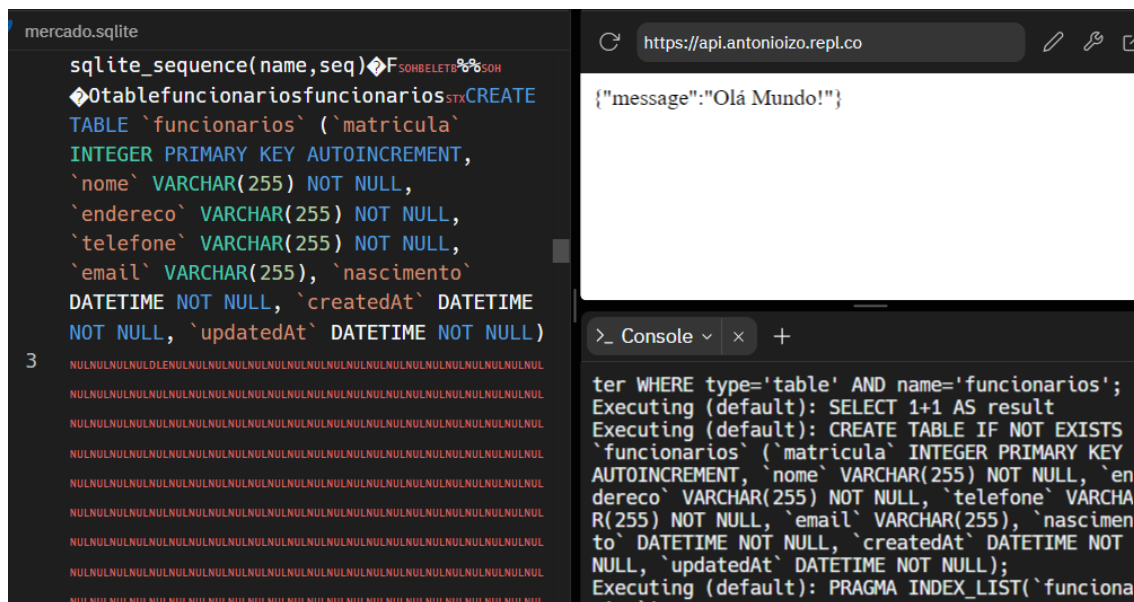
Vamos instalar nossos módulos pelo *shell* *sqlite* e *sequelize* com o comando **npm**.

Npm install sequelize sqlite3



```
~/API$ npm install sequelize sqlite3
```

Ao executar nosso API, teremos a criação do banco de dados e da nossa tabela “funcionários”.



```
mercado.sqlite
sqlite_sequence(name,seq)
CREATE TABLE `funcionarios` (`matricula` INTEGER PRIMARY KEY AUTOINCREMENT, `nome` VARCHAR(255) NOT NULL, `endereco` VARCHAR(255) NOT NULL, `telefone` VARCHAR(255) NOT NULL, `email` VARCHAR(255), `nascimento` DATETIME NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)
```

```
https://api.antoniozo.repl.co
{"message":"Olá Mundo!"}
```

```
_ Console
ter WHERE type='table' AND name='funcionarios';
Executing (default): SELECT 1+1 AS result
Executing (default): CREATE TABLE IF NOT EXISTS `funcionarios` (`matricula` INTEGER PRIMARY KEY AUTOINCREMENT, `nome` VARCHAR(255) NOT NULL, `endereco` VARCHAR(255) NOT NULL, `telefone` VARCHAR(255) NOT NULL, `email` VARCHAR(255), `nascimento` DATETIME NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
Executing (default): PRAGMA INDEX_LIST('funcionarios')
```

Na próxima aula, você verá o processo de criação do *controller*, o *create* e o *read*.

Até lá...