

Introdução ao Sequelize – Mapeamento Objeto Relacional

→ <https://sequelize.org/>



Sequelize

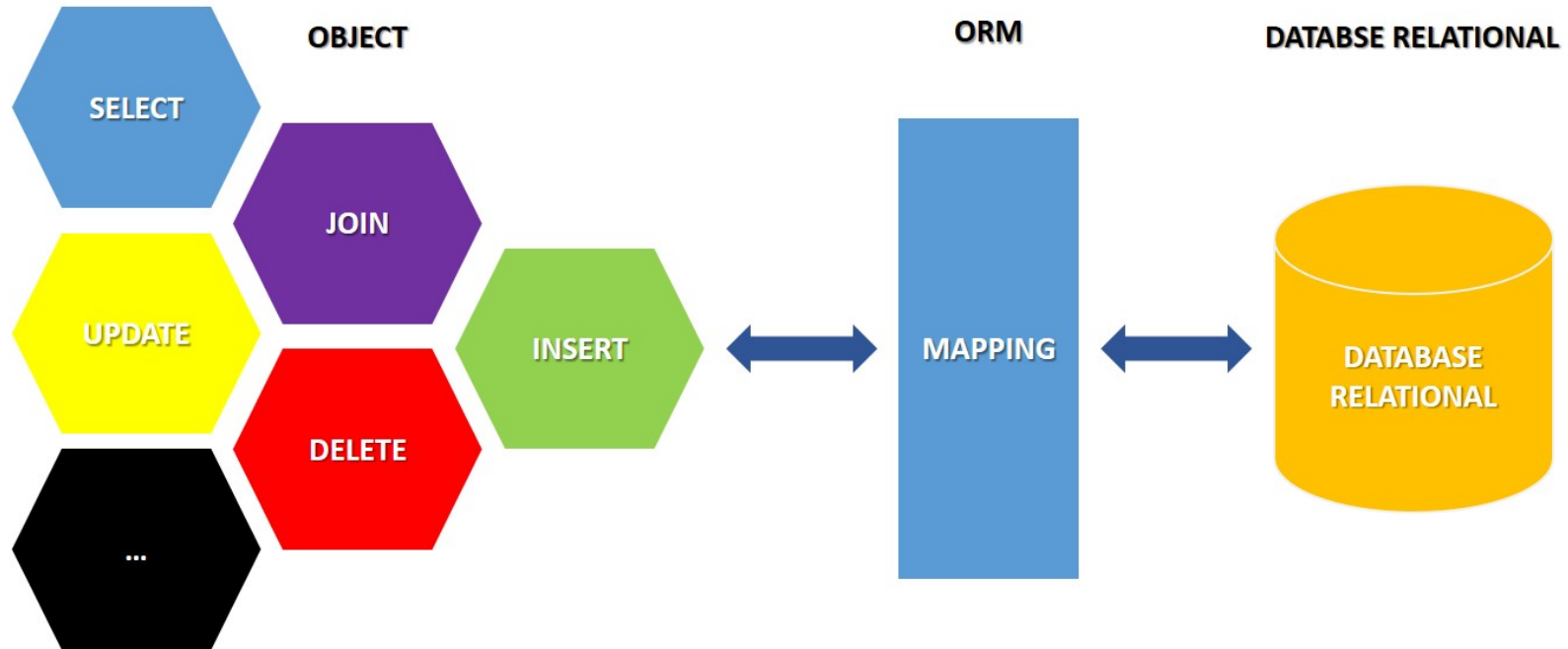
Este material não é um manual nem tão pouco um guia de construção de projetos usando NodeJs e Sequelize.

Trata-se de um apanhado de informações relacionadas a ORM, NodeJs, Express e MVC.

Portanto este material não oferecerá passo a passo para a construção de nenhum projeto WEB usando NodeJs.

Definição

O Sequelize, segundo o site oficial, é um ORM para Node.js baseado em Promises, para os bancos *PostgreSQL, MySQL, MariaDB, SQLite e MS SQL Server*.



ORM

OBJECT RELATIONAL MAPPING

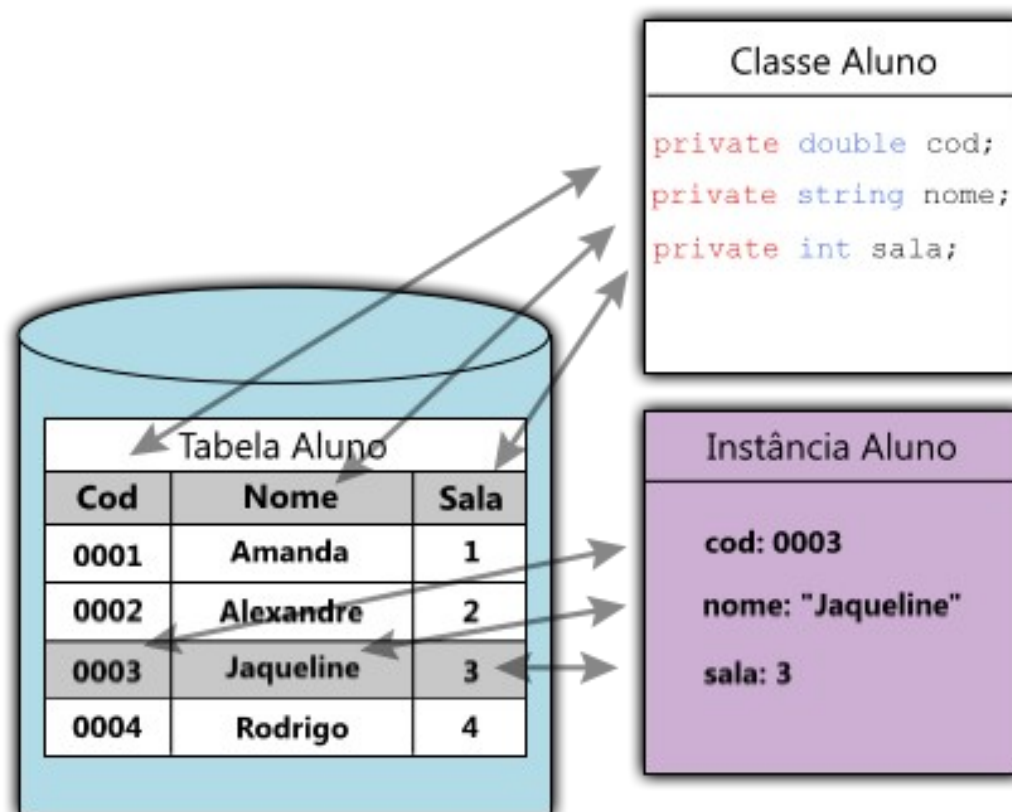
ORM – Object Relational Mapping

O ORM (*Object Relational Mapping*) ou *Mapeamento Objeto Relacional* trata-se de uma técnica na programação que consiste em fazer a ponte entre *modelo relacional* (banco de dados) e objetos (*aplicação*), através de frameworks capazes de converter dados entre sistemas *utilizando linguagens de programação orientada a objetos*.

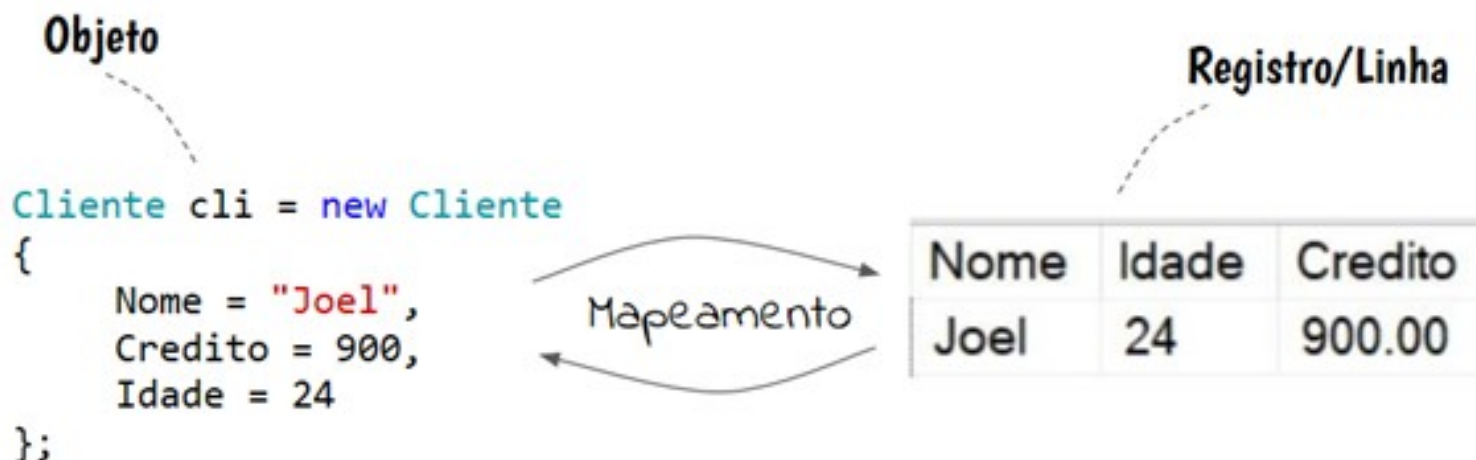
Conversão de dados



Modelo relacional para ORM



Objeto recuperado em JSON



Exemplo Json recuperado do banco

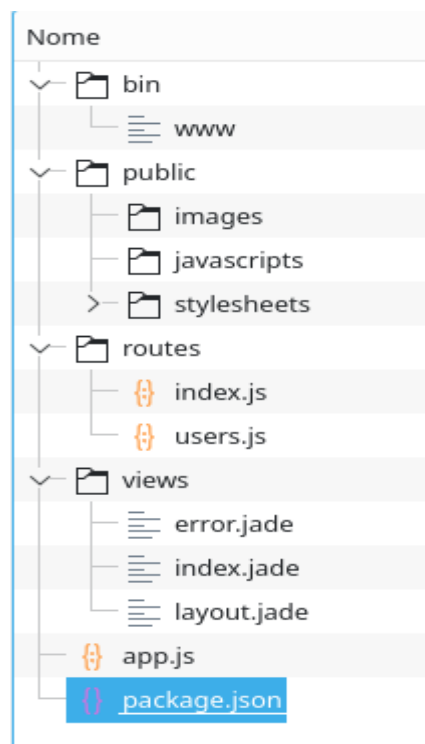
```
{
  "bairro": "Centro",
  "cidade": "Alto Porã",
  "logradouro": "Rua São Francisco 24",
  "estado_info": {
    "area_km2": "248.221,996",
    "codigo_ibge": "35",
    "nome": "São Paulo"
  },
  "cep": "14480970",
  "estado": "SP"
}
```

Geradores de aplicativos: Express e Sequelize

Gerador do express

* gerar aplicativo com express

`npx express-generator -g`



```
create : public/
create : public/javascripts/
create : public/images/
create : public/stylesheets/
create : public/stylesheets/style.css
create : routes/
create : routes/index.js
create : routes/users.js
create : views/
create : views/error.jade
create : views/index.jade
create : views/layout.jade
create : app.js
create : package.json
create : bin/
create : bin/www
```

install dependencies:

```
$ npm install
```

run the app:

```
$ DEBUG=express:* npm start
```

```
kdeneto@Linux-Dell:~/Geral/temp/express$
```

Gerador do sequelize

* gerar aplicativo com sequelize

```
npm init -y
```

```
npm i --save-dev sequelize-cli
```

```
npx sequelize-cli init
```

Instalação, configuração e principais comandos

Instalação pacotes básicos

* instalar express

```
npm i --save express
```

* instalar nodemon

```
npm i --save nodemon
```

* instalar mysql

```
npm i --save mysql2
```

Instalação pacotes básicos

- * criar o package.json do projeto

```
npm init -y
```

- * instalar sequelize no projeto

```
npm install sequelize express
```

- * instalar sequelize dependência de desenvolvimento

```
npm install --save-dev sequelize-cli
```


Comandos básicos

* criar um projeto vazio

`npx sequelize-cli init`

→ config, models, migrations, seeders

** depois de configurado criar o banco de dados vazio

`npx sequelize-cli db:create`

* criar a primeira migration

`npx sequelize-cli migration:create --name=criar-tabela`

Comandos básicos

* executar a migrate

```
npx sequelize-cli db:migrate
```

* desfazer a execução da última migration

```
npx sequelize-cli migrate:undo
```

Configurações – server.js

JS app copy.js ×

JS app copy.js > ...

```
1  const express = require('express')
2  const http = require('http')
3
4  require('./src/database/indexDb.js');
5
6  // instanciar o express
7  const app = express();
8
9  app.use(express.json())
10
11
12  //configurar a porta e url para execução do aplicativo
13  app.set('url', 'http://localhost:')
14  app.set('porta', 3001);
15
16
17
18
19
20  module.exports = app;
21
22  |
```

Configurações

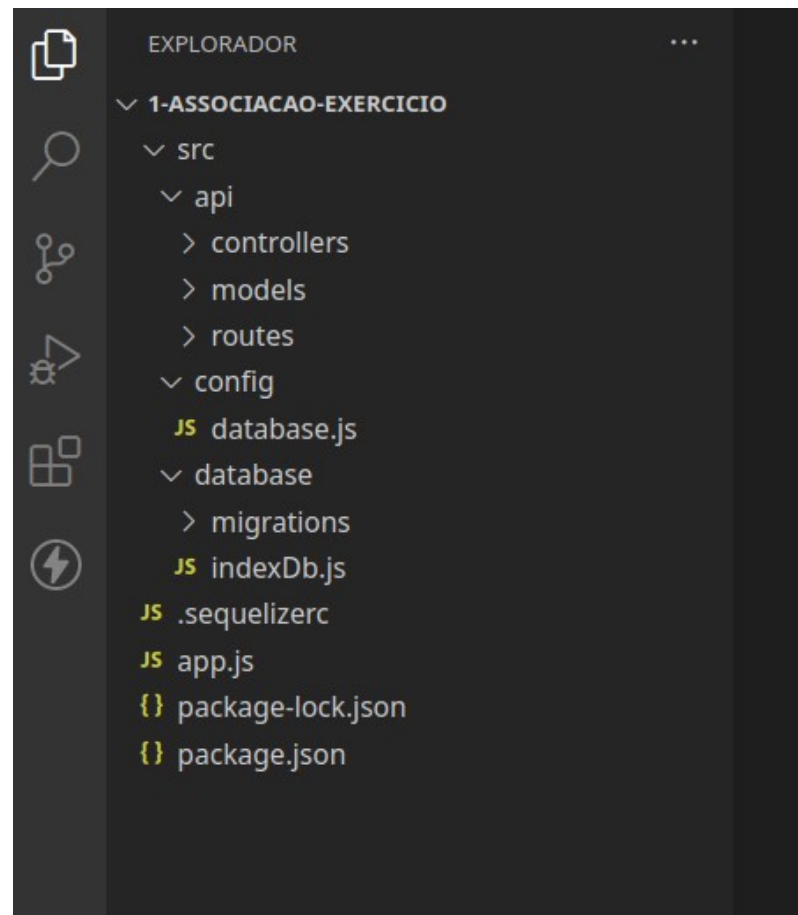
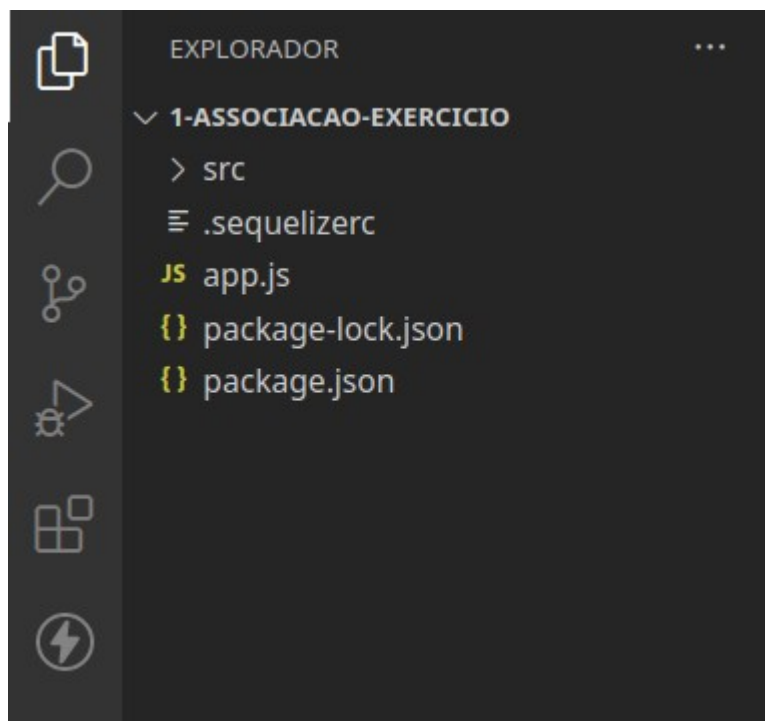
* criar na raiz do projeto o arquivo .sequelizerc

JS .sequelizerc X

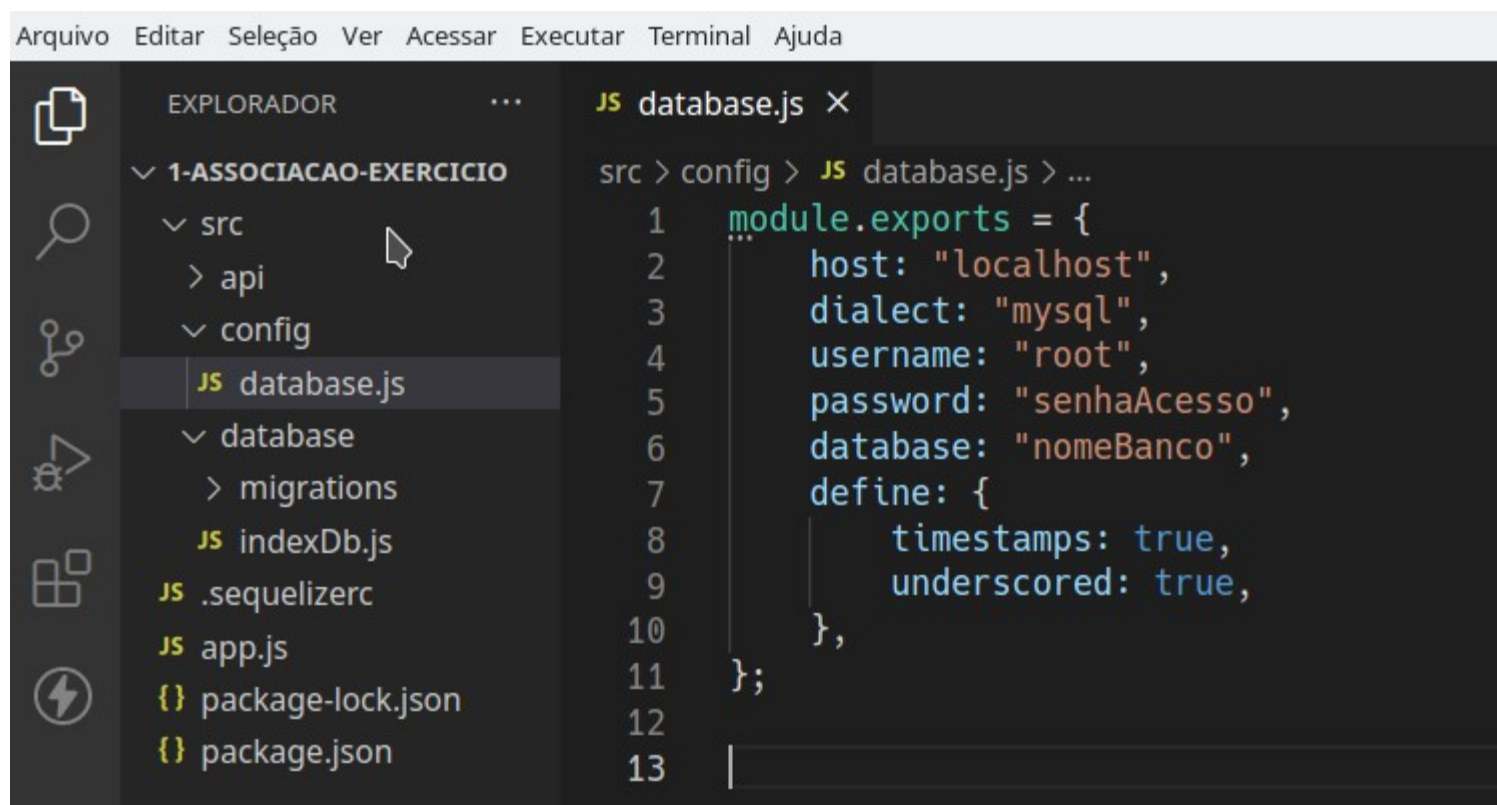
JS .sequelizerc > ...

```
1  const path = require('path');
2
3  module.exports = {
4    'config': path.resolve(__dirname, 'src', 'config', 'database.js'),
5    'migrations-path': path.resolve(__dirname, 'src', 'database', 'migrations'),
6  };
7
8
```

Configurações - estrutura de pastas



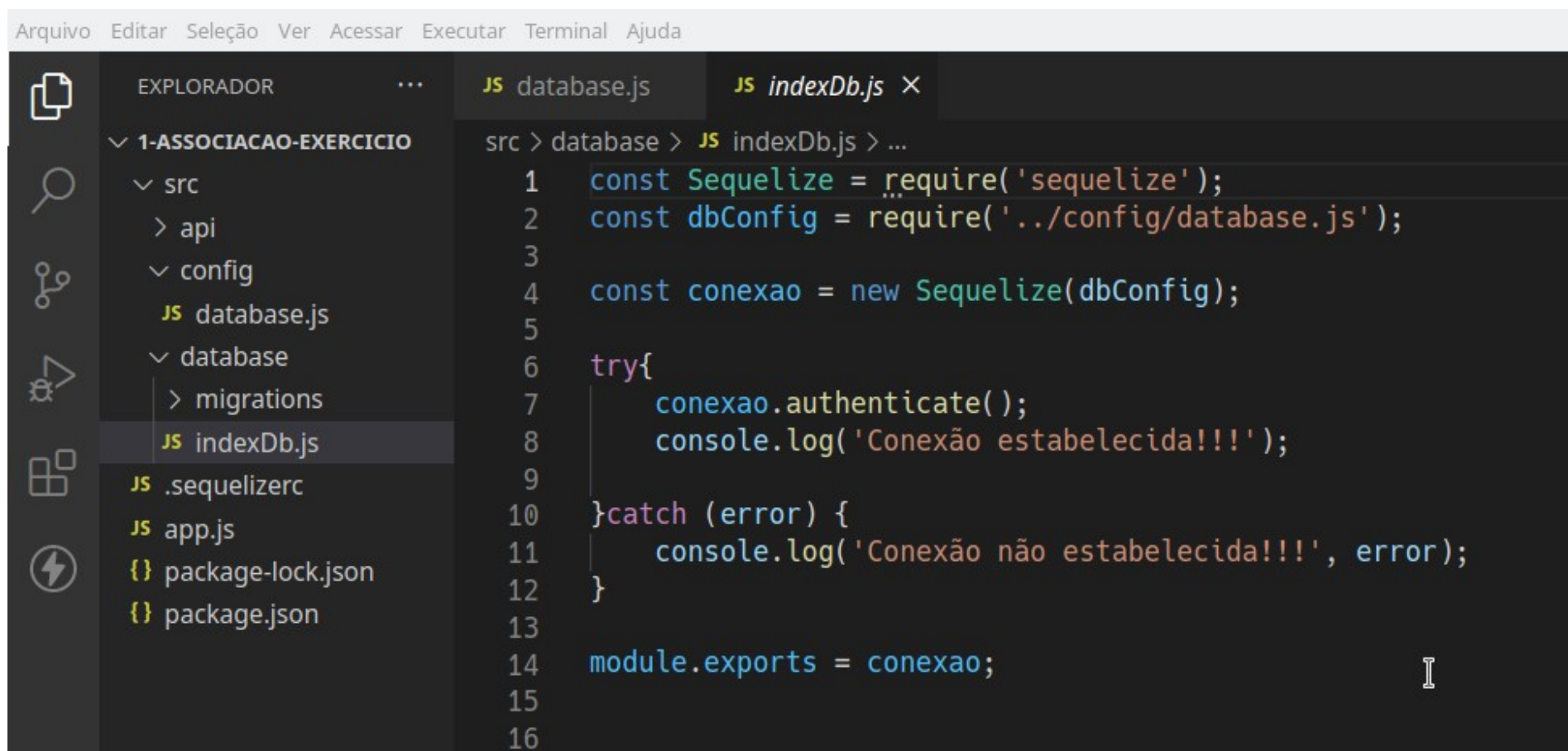
Configurações – Banco de Dados



The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure under the folder '1-ASSOCIACAO-EXERCICIO'. The 'config' folder is expanded, and 'database.js' is selected. The main editor area shows the content of 'database.js', which is a JavaScript file defining Sequelize database configuration. The code is as follows:

```
src > config > JS database.js > ...
1  module.exports = {
2      ...
3      host: "localhost",
4      dialect: "mysql",
5      username: "root",
6      password: "senhaAcesso",
7      database: "nomeBanco",
8      define: {
9          timestamps: true,
10         underscored: true,
11     },
12 };
13
```

Configurações – Conexão – indexDb.js



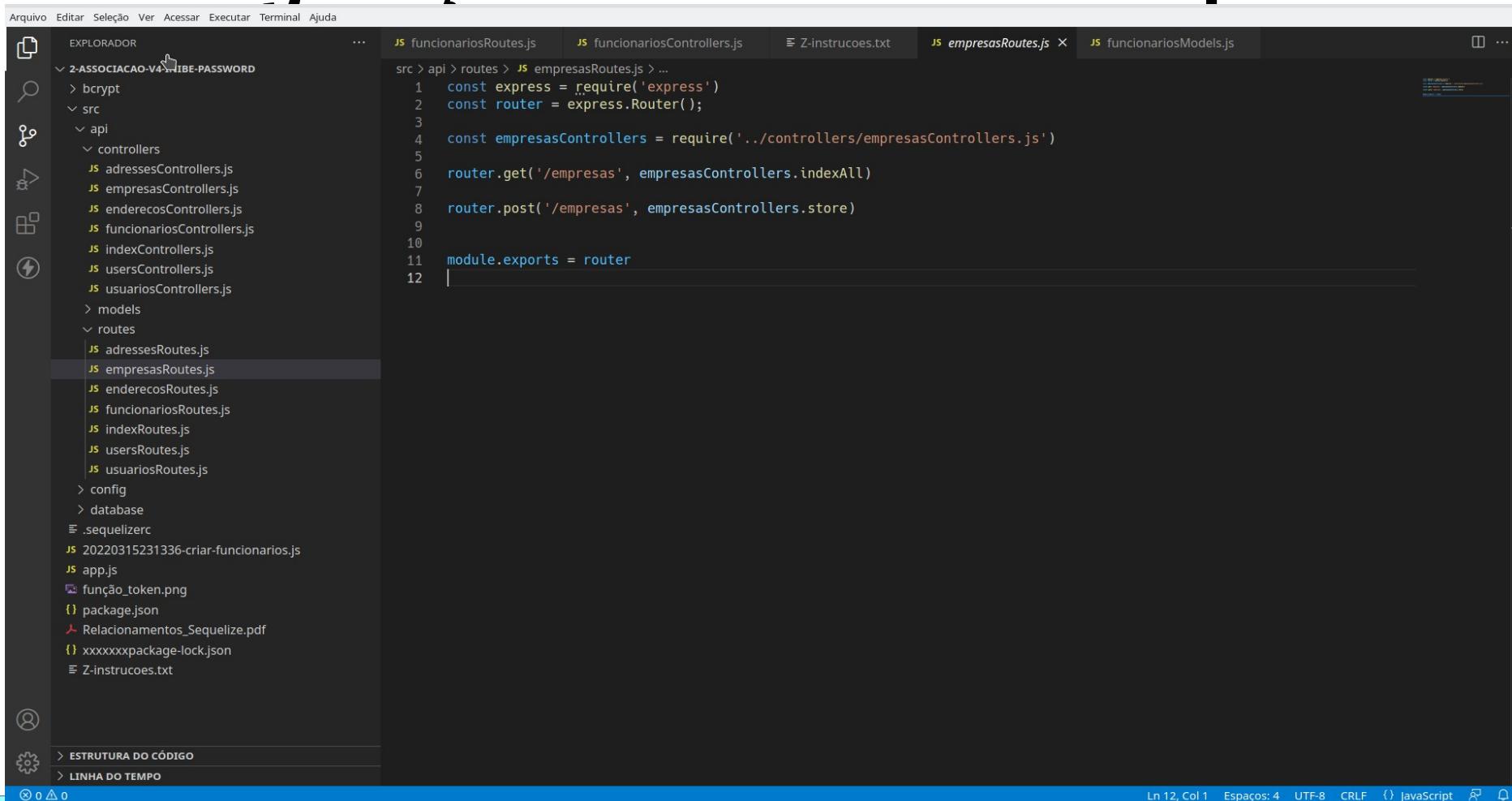
```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda

EXPLORADOR  ...  JS database.js  JS indexDb.js X

1-ASSOCIACAO-EXERCICIO
  src
    > api
  config
    JS database.js
  database
    > migrations
    JS indexDb.js
  JS .sequelizerc
  JS app.js
  {} package-lock.json
  {} package.json

src > database > JS indexDb.js > ...
1  const Sequelize = require('sequelize');
2  const dbConfig = require('../config/database.js');
3
4  const conexao = new Sequelize(dbConfig);
5
6  try{
7    conexao.authenticate();
8    console.log('Conexão estabelecida!!!');
9
10 }catch (error) {
11   console.log('Conexão não estabelecida!!!', error);
12 }
13
14 module.exports = conexao;
```

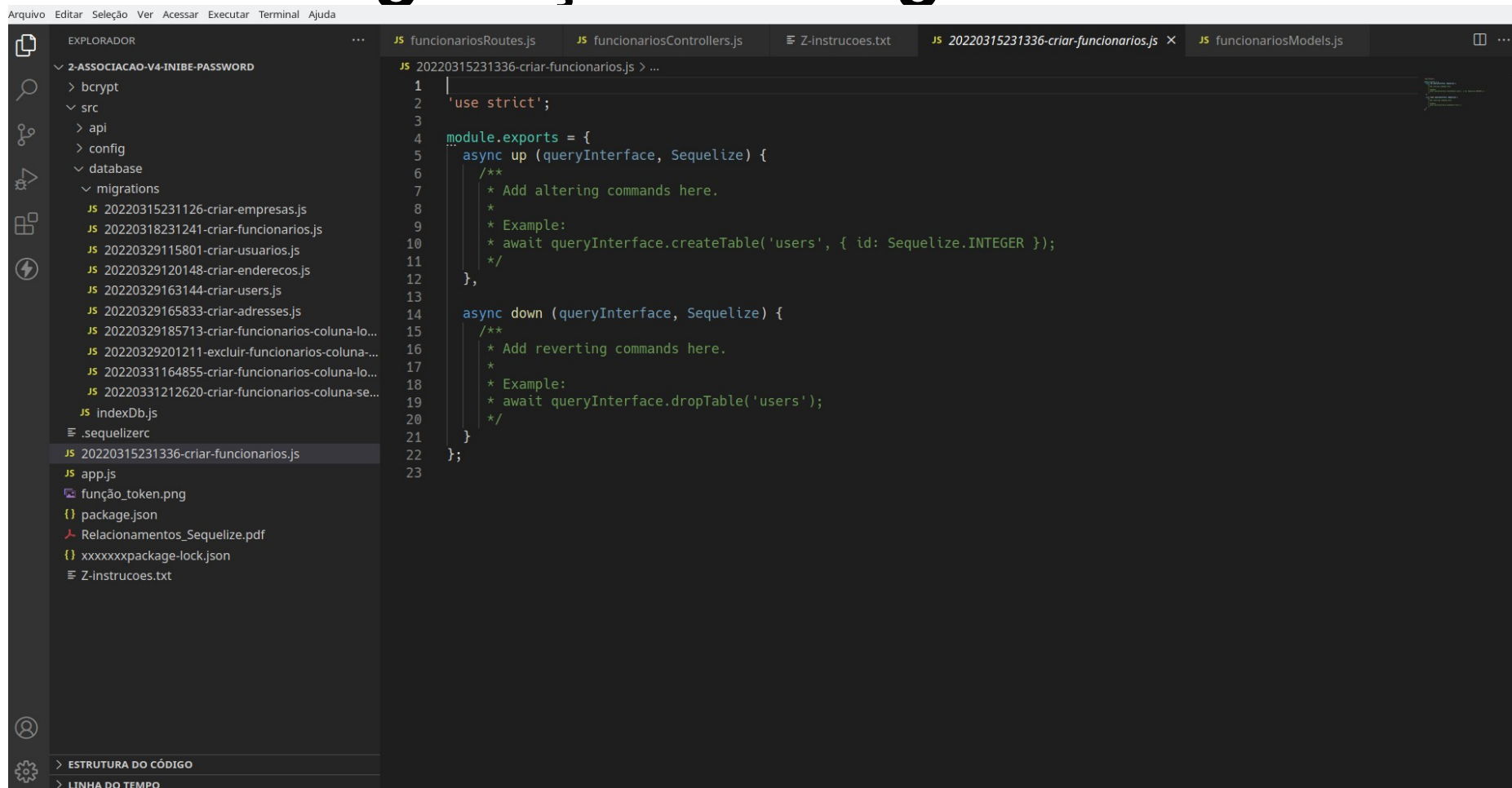

Configurações – Routes de empresas



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'api', 'controllers', 'models', and 'routes'. The 'routes' directory contains several route files, including 'empresasRoutes.js'. The code editor shows the content of 'empresasRoutes.js'.

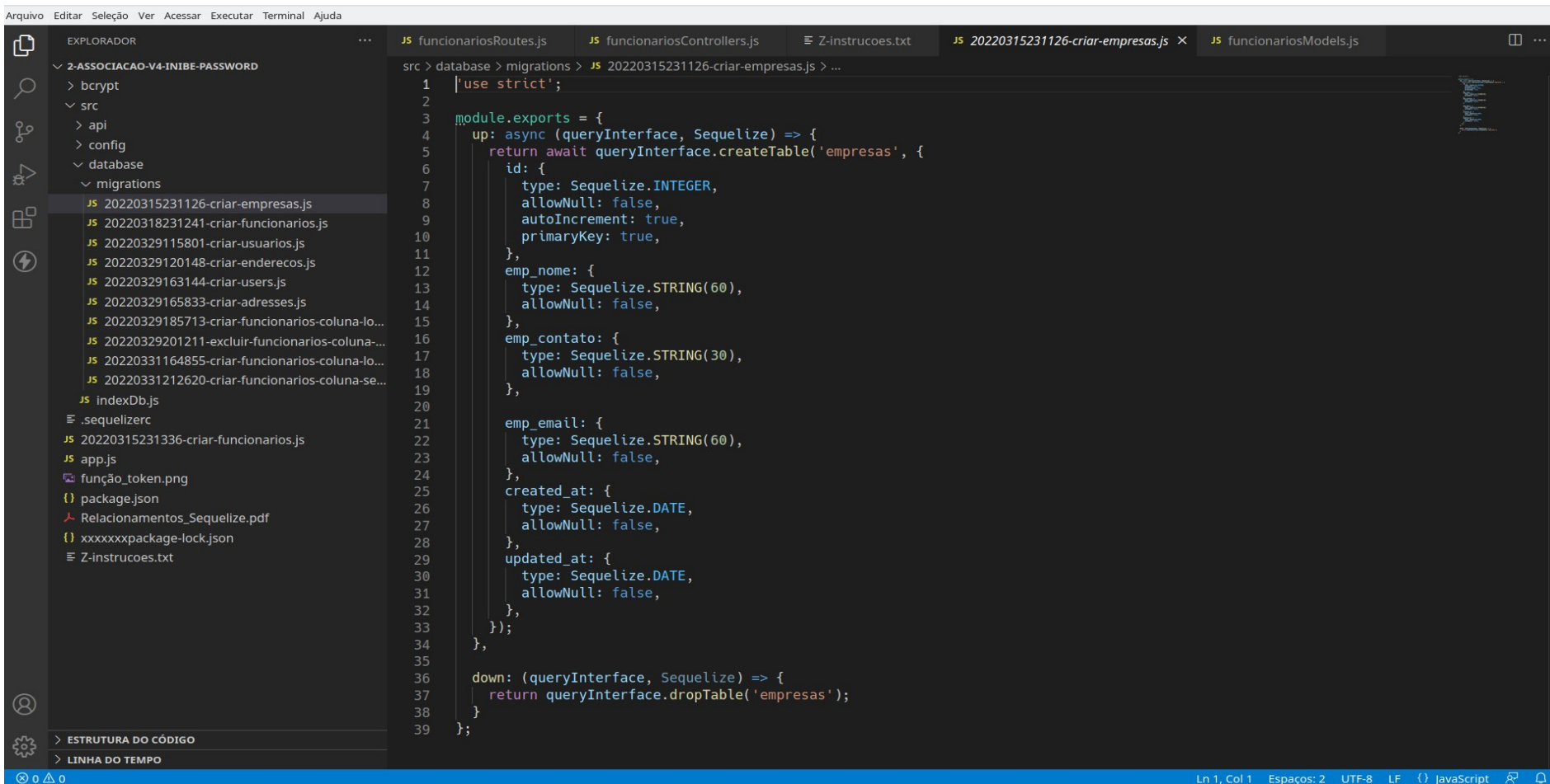
```
src > api > routes > JS empresasRoutes.js > ...
1  const express = require('express')
2  const router = express.Router();
3
4  const empresasControllers = require('../controllers/empresasControllers.js')
5
6  router.get('/empresas', empresasControllers.indexAll)
7
8  router.post('/empresas', empresasControllers.store)
9
10
11 module.exports = router
12
```


Configurações – Migrate inicial



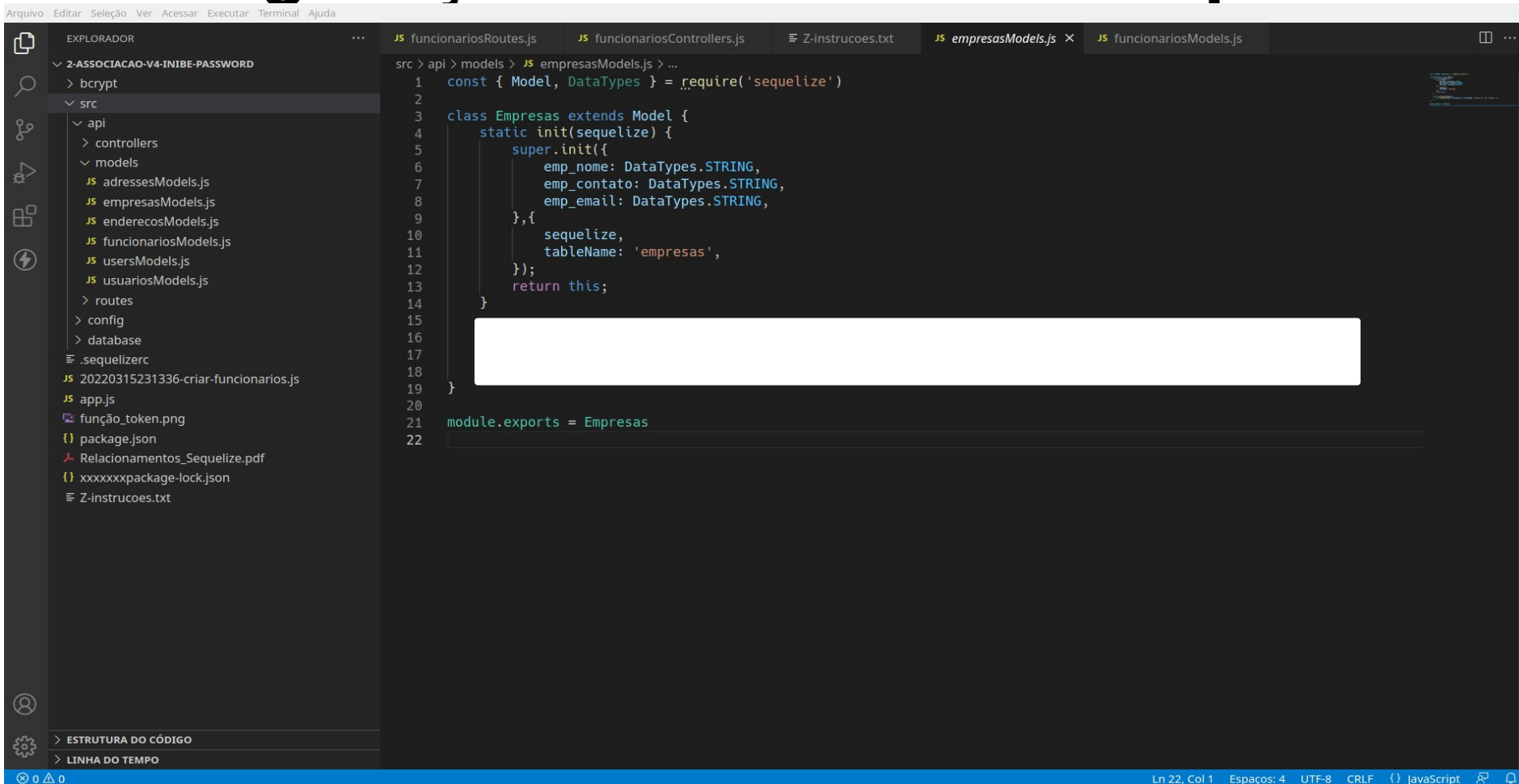
```
JS 20220315231336-criar-funcionarios.js > ...
1 |
2 'use strict';
3
4 module.exports = {
5   async up (queryInterface, Sequelize) {
6     /**
7      * Add altering commands here.
8      *
9      * Example:
10      * await queryInterface.createTable('users', { id: Sequelize.INTEGER });
11      */
12   },
13
14   async down (queryInterface, Sequelize) {
15     /**
16      * Add reverting commands here.
17      *
18      * Example:
19      * await queryInterface.dropTable('users');
20      */
21   }
22 };
23
```

Configurações – Migrate alterada



```
src > database > migrations > JS 20220315231126-criar-empresas.js > ...
1  'use strict';
2
3  module.exports = {
4    up: async (queryInterface, Sequelize) => {
5      return await queryInterface.createTable('empresas', {
6        id: {
7          type: Sequelize.INTEGER,
8          allowNull: false,
9          autoIncrement: true,
10         primaryKey: true,
11       },
12       emp_nome: {
13         type: Sequelize.STRING(60),
14         allowNull: false,
15       },
16       emp_contato: {
17         type: Sequelize.STRING(30),
18         allowNull: false,
19       },
20       emp_email: {
21         type: Sequelize.STRING(60),
22         allowNull: false,
23       },
24       created_at: {
25         type: Sequelize.DATE,
26         allowNull: false,
27       },
28       updated_at: {
29         type: Sequelize.DATE,
30         allowNull: false,
31       },
32     },
33   });
34 },
35
36   down: (queryInterface, Sequelize) => {
37     return queryInterface.dropTable('empresas');
38   }
39 };
```

Configurações – Models de empresas



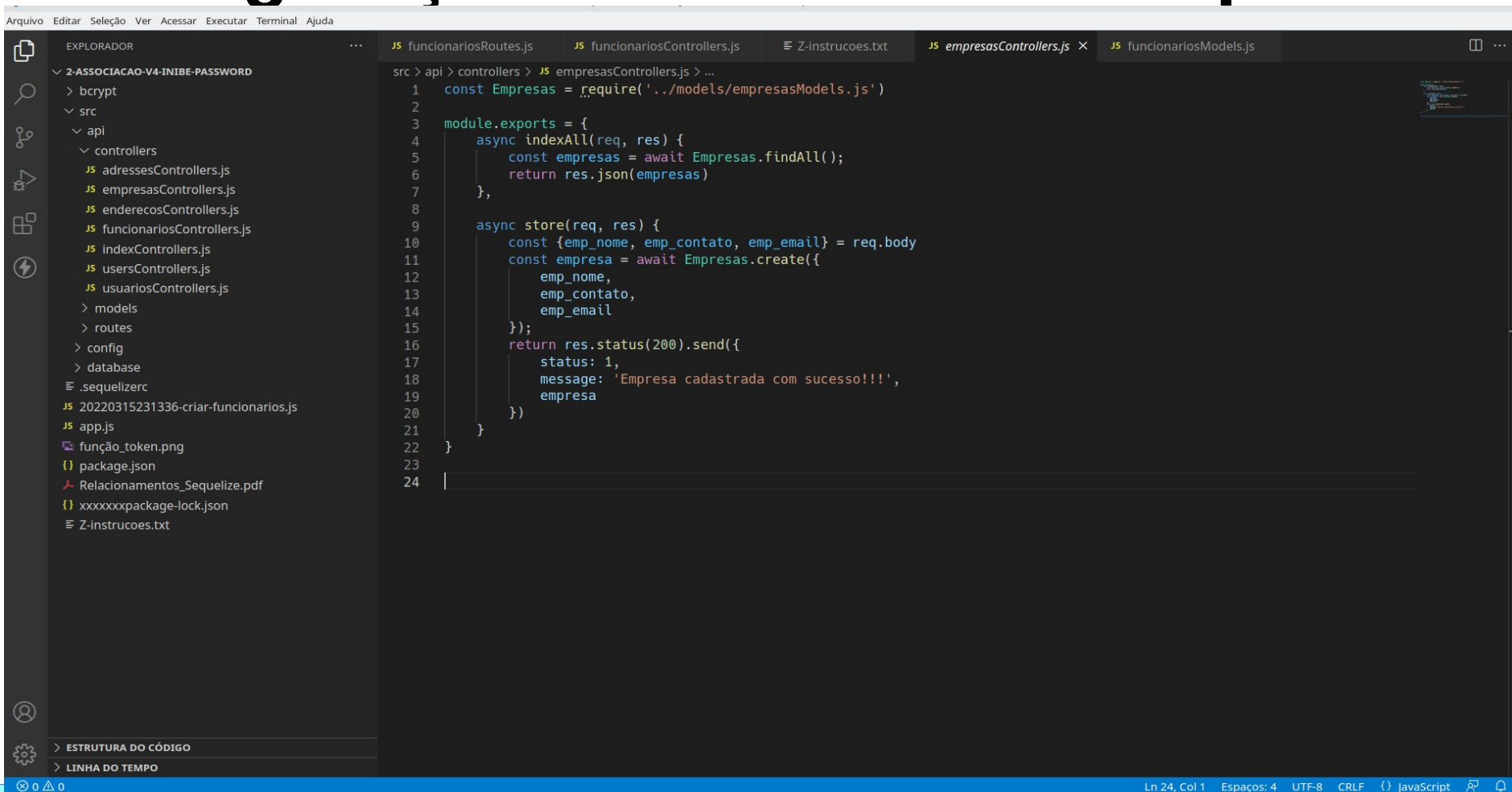
The screenshot shows a code editor with the following structure:

- EXPLORADOR (Left Sidebar):**
 - 2-ASSOCIACAO-V4-INIBE-PASSWORD
 - bcrypt
 - src
 - api
 - controllers
 - models
 - adressesModels.js
 - empresasModels.js
 - enderecosModels.js
 - funcionariosModels.js
 - usersModels.js
 - usuariosModels.js
 - routes
 - config
 - database
 - .sequelizerc
 - 20220315231336-criar-funcionarios.js
 - app.js
 - função_token.png
 - package.json
 - Relacionamentos_Sequelize.pdf
 - xxxxxxpackage-lock.json
 - Z-instrucoes.txt
- funcionariosRoutes.js**
- funcionariosControllers.js**
- Z-instrucoes.txt**
- empresasModels.js** (Active)
- funcionariosModels.js**

Code in empresasModels.js:

```
1  src > api > models > JS empresasModels.js > ...
2  const { Model, DataTypes } = require('sequelize')
3
4  class Empresas extends Model {
5    static init(sequelize) {
6      super.init({
7        emp_nome: DataTypes.STRING,
8        emp_contato: DataTypes.STRING,
9        emp_email: DataTypes.STRING,
10      }, {
11        sequelize,
12        tableName: 'empresas',
13      });
14      return this;
15    }
16  }
17
18  module.exports = Empresas
19
20
21
22
```

Configurações – Controllers empresas



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'src' and 'controllers', and files like 'empresasControllers.js'. The code editor shows the content of 'empresasControllers.js'.

```
src > api > controllers > JS empresasControllers.js > ...
1  const Empresas = require('../models/empresasModels.js')
2
3  module.exports = {
4    async indexAll(req, res) {
5      const empresas = await Empresas.findAll();
6      return res.json(empresas)
7    },
8
9    async store(req, res) {
10     const {emp_nome, emp_contato, emp_email} = req.body
11     const empresa = await Empresas.create({
12       emp_nome,
13       emp_contato,
14       emp_email
15     });
16     return res.status(200).send({
17       status: 1,
18       message: 'Empresa cadastrada com sucesso!!!',
19       empresa
20     })
21   }
22 }
23
24
```