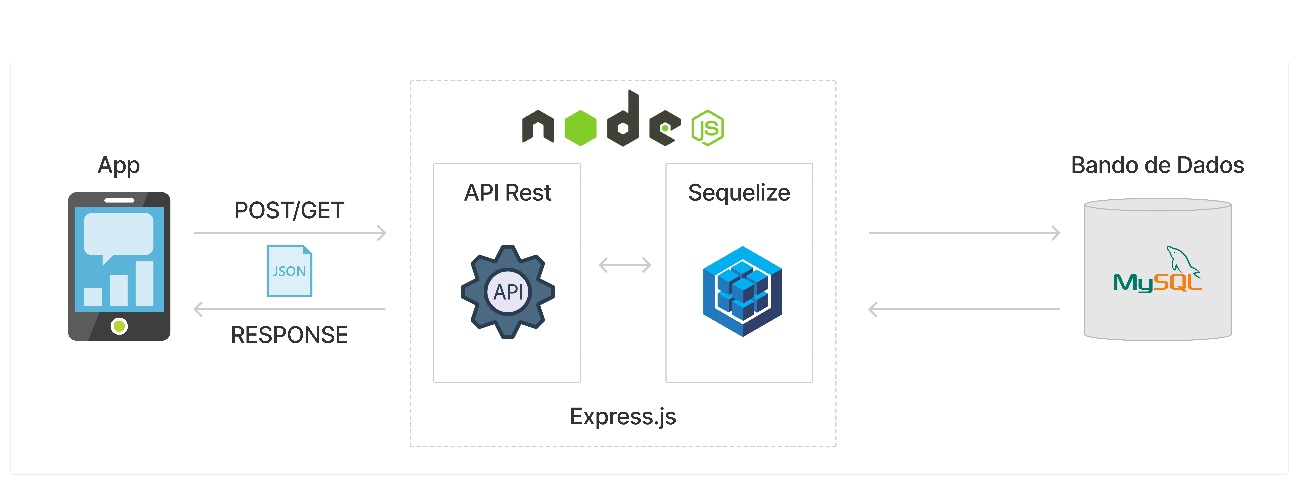
**Usando Sequelize com node e express**

Em nosso projeto utilizamos o Sequelize que é um padrão ORM (Object Relacional Mapper - Mapeamento Objeto-Relacional) baseado em promises para Node.js. Este modelo tem como finalidade abstrair os comandos de operações de SQL.

Utilizamos a linguagem JavaScript para se conectar, passar instruções para o banco e realizar as operações de CRUD. Através do Sequelize, o JavaScript vai ser responsável por traduzir as instruções para o SQL, para o banco, e fazer as operações necessárias, deixando mais fácil o acesso e a manipulação aos dados.



**Ferramentas que precisam ser instaladas antes de iniciar o projeto:**

- MySQL installer v8.0.31 – Banco utilizado para guardar os dados de nossa API;

- MySql Workbench v8.0.31 – Consulta ao banco de dados;

- NodeJS v18.12.0

- Postman v10 – Ferramenta que serve para enviar e receber requisições HTTP para testar APIs;

- VScode

**Instalação de dependências das bibliotecas:**

Node.js:

* npm init (iniciar aplicação node – cria a package.json com esqueleto base).

Express:

* npm install express (Para subir o servidor local e gerenciar as rotas criadas)

Nodemon:

* npm install -g nodemon (Nodemon serve para ficar escutando automaticamente todas as alterações que fazemos nos arquivos na aplicação)

Mysql2:

* npm instal -- save mysql2

Sequelize:

* npm install sequelize (Sequelize como dependência da nossa aplicação);
* npm install –save-dev sequelize-cli (Para trabalhar com Sequelize na linha de comando)
* npx sequelize-cli init (Inicia com todas as pastas necessárias para conexão com o banco: config, migrações, modelos e seeders.). Em seguida passamos todas estas pastas para dentro da pasta onde estar nosso projeto.

Criamos um arquivo .sequelizerc para estruturar nossa aplicação. Dentro do arquivo, inserimos o seguinte código:

const path = require('path');

module.exports = {

'config': path.resolve('./api/config', 'config.json'),

'models-path': path.resolve('./api/models'),

'seeders-path': path.resolve('./api/seeders'),

'migrations-path': path.resolve('./api/migrations')

}

Este o comando acima, avisa ao Sequelize para qual diretório foi movido pastas.

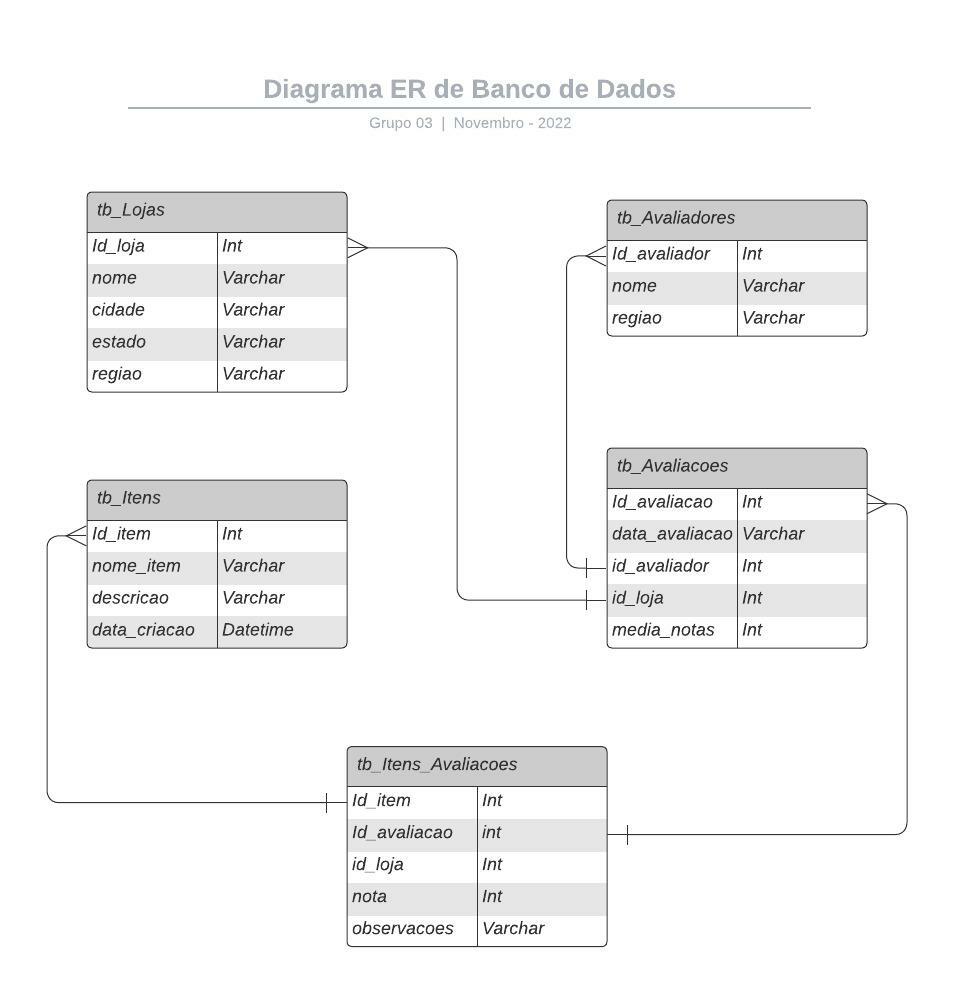
Executamos o comando npx sequelize-cli init  para criar nossa estrutura do projeto (pastas de configurações para o config, models, migrations e seeders).

A pasta **config**: Terá o arquivo config.js que vai ficar a Url ou dados de conexão com o banco de dados.

Pasta migrations e pasta models: terá a nossas migrations e models. Os models são os arquivos de criação da tabela e as migrations são alterações nessas tabelas. Por exemplo, o model cria a tabela e a migration cria a chave estrangeira: foreigns keys.

**Criando as tabelas**

Usaremos o seguinte MER (Modelo de Entidade Relacional):



Começamos criando as tabelas principais “tb\_lojas” e td\_Avaliadores através do comando sequelize-cli direto no terminal, passando o nome do modelo que criamos e os atributos com os seus respectivos dados. Em seguida, criamos as demais tabelas com base nas tabelas principais.

msql> create database if not exists projetofinal4;

msql> show databases;

**Migração**

Usamos o Sequelize para fazer as migrações do banco, para gerenciar as alterações do nosso esquema e guardar essas alterações através do comando chamado *migrate.*

* *Npx sequelized projetofinal4:migrate (criação da migrate)*
* *mysql> use projetofinal4 (verificar se a tabela foi criada)*
* *mysql>show table (visualizar tabela criada)*

**Conexão**

sequelize-auto -o "./models" -d PROJETOFINAL -h localhost -u root -p 3306 -x @SENHA -e mysql (Comando para conectar a API ao banco de dado).