

**UNIVERSIDADE DE SALVADOR  
CAMPUS PROFESSOR BARROS**

**VITOR PIO VIEIRA, RA: 1272220376**

**MARCOS RIBEIRO, RA: 12723116626**

**HERICLES SANDER DOS SANTOS CONCEIÇÃO, RA: 1272226965**

**PEDRO HENRIQUE NASCIMENTO LUZ, RA: 12722122396**

**EDUARDO GONZAGA LIMA, RA: 1272220388**

**APLICAÇÃO SIMULANDO A CAPTAÇÃO DE DADOS DE VENDA**

**SALVADOR, BAHIA**

**2023**

**VITOR PIO VIEIRA, RA: 1272220376**

**MARCOS RIBEIRO, RA: 12723116626**

**HERICLES SANDER DOS SANTOS CONCEIÇÃO, RA: 1272226965**

**PEDRO HENRIQUE NASCIMENTO LUZ, RA: 12722122396**

**EDUARDO GONZAGA LIMA, RA: 1272220388**

## **APLICAÇÃO SIMULANDO A CAPTAÇÃO DE DADOS DE VENDA**

Este projeto tem como objetivo realização criar uma aplicação para simulação de uma captação de dados em um comercio de vendas, em uma rede de lojas. Com objetivo de obtenção de nota para a AV3.

**SALVADOR, BAHIA**

**2023**

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>04</b>
<b>2 METODOLOGIA.....</b>	<b>05</b>
<b>3 INSTRUÇÕES PARA INSTALAÇÃO E EXECUÇÃO DA APLICAÇÃO.....</b>	<b>06</b>
<b>4 REQUERIMENTOS.....</b>	<b>12</b>
<b>5 JUSTIFICATIVA DAS TECNOLOGIAS ESCOLHIDAS.....</b>	<b>13</b>
<b>6 APRESENTAÇÃO E DETALHAMENTOS.....</b>	<b>14</b>
<b>7 CONSIDERAÇÕES FINAIS.....</b>	<b>15</b>
<b>8 BIBLIOGRAFIA.....</b>	<b>16</b>

## 1 - INTRODUÇÃO

Neste projeto trouxemos uma aplicação para facilitar o cenário do varejo, com a finalidade de uma gestão mais eficiente de suas vendas, que é algo crucial para o sucesso de qualquer rede de lojas. Nesse contexto, apresentamos nossa aplicação inovadora que simula a captação de dados de vendas para uma rede de lojas, proporcionando um controle abrangente por meio de funcionalidades robustas e uma arquitetura flexível.

A aplicação foi desenvolvida para atender às necessidades essenciais de gerenciamento, envolvendo aspectos cruciais como clientes, estoque, vendas e a geração de relatórios estatísticos perspicazes. Com a capacidade de operar em Java, Python ou Javascript, oferecemos uma solução adaptável às preferências e habilidades da equipe de desenvolvimento.

Para assegurar uma experiência rica desde o início, a aplicação é pré-configurada com um conjunto inicial de 10 produtos cadastrados e 5 clientes. Essa base sólida permite uma rápida implementação e avaliação das funcionalidades propostas. A flexibilidade é uma marca registrada do projeto, oferecendo à equipe a liberdade de escolher entre a implementação utilizando sockets, RPC, API ou Filas de Mensagens. Além disso, garantimos a solidez do banco de dados ao optar por uma abordagem relacional, fundamental para a integridade e consistência dos dados.

Detalhando as principais funcionalidades, destacamos a gestão de clientes, permitindo operações de CRUD (Criar, Ler, Atualizar, Deletar) de forma intuitiva. Da mesma forma, o gerenciamento de vendas oferece um controle preciso do estoque, possibilitando a recepção eficiente de pedidos de compra. A geração de relatórios estatísticos representa o diferencial do nosso projeto, proporcionando insights valiosos para a tomada de decisões estratégicas. Desde relatórios de produtos mais vendidos até análises personalizadas por cliente, a aplicação fornece uma visão abrangente do desempenho do negócio. Com relatórios de consumo médio do cliente e alertas sobre produtos com baixo estoque, garantimos uma gestão proativa e orientada por dados.

Nossa aplicação visa otimizar a eficiência operacional e aprimorar a tomada de decisões, contribuindo significativamente para o sucesso da rede de lojas. Estamos empolgados em apresentar esta solução, esperando que ela atenda e supere as expectativas, proporcionando uma experiência de gestão de vendas excepcional.

## 2 - METODOLOGIA

Com a identificação dos requisitos funcionais e não funcionais da aplicação, com foco nas funcionalidades de gerenciamento de cliente, vendas, estoque e geração de relatórios, esse era um dos requisitos para a nossa aplicação, em seguida pensamos na estrutura de projeto, onde utilizamos o comando `npm init` para criar um arquivo `package.json` e definir as configurações do projeto, e organizando a estrutura do projeto, criando diretórios para cada módulo (cliente, vendas, estoque, relatórios). Criamos as dependências necessárias com `npm install express nodemon -save` e configuramos o script start no arquivo `package.json` para iniciar a aplicação usando o Nodemon.

Em seguida foi desenvolvimento do módulo cliente: Criamos as rotas e controladores utilizando o Express para o CRUD de clientes, após os testes de operações com ferramentas como o Postman integrado o módulo com o banco de dados relacional. Também foi desenvolvido um módulo de vendas e estoque, com as funções de implementar as funcionalidades de CRUD para vendas e estoque, de integrar as operações com a manipulação do banco de dados relacional e garantia da consistência nas transações de compra e atualização de estoque.

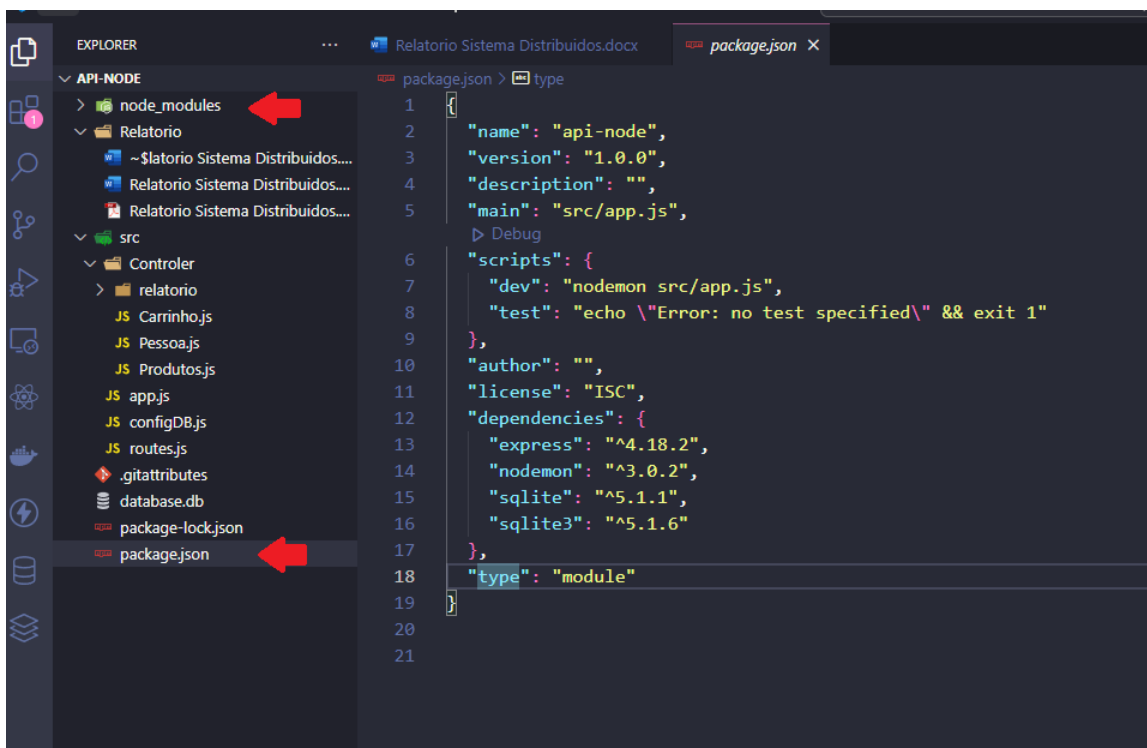
Na parte de desenvolvimento de relatórios, foi implementado as lógicas para gerar relatórios estatísticos e utilizado consultas SQL para obter dados relevantes do banco de dados. E os testes unitários da integração gera testes para cada módulo utilizando frameworks como Mocha ou Jest, e realizando testes de integração para garantir o funcionamento harmonioso entre os diferentes componentes.

### 3 - INSTRUÇÕES PARA INSTALAÇÃO E EXECUÇÃO DA APLICAÇÃO.

Nesse tópico iremos abordar um passo a passo para que consiga rodar a aplicação de forma correta, será necessário seguir, para que não haja erro na execução da aplicação e funcione perfeitamente.

O nosso primeiro passo iremos certificar se que o Node.js (e o npm) estão instalados corretamente em seu computador, você pode selecionar abrir o Windows Powershell ou o Prompt, digitar `node -v` para conferir se está instalando e qual a sua versão do que está usando. Caso não possua ou esteja desatualizada, faça o download e instale o Node.js, a versão utilizada foi a 20.10.0, para realizar o download e instalação, basta acessar o site oficial do Node.js (<https://nodejs.org/en>), baixar e instalar.

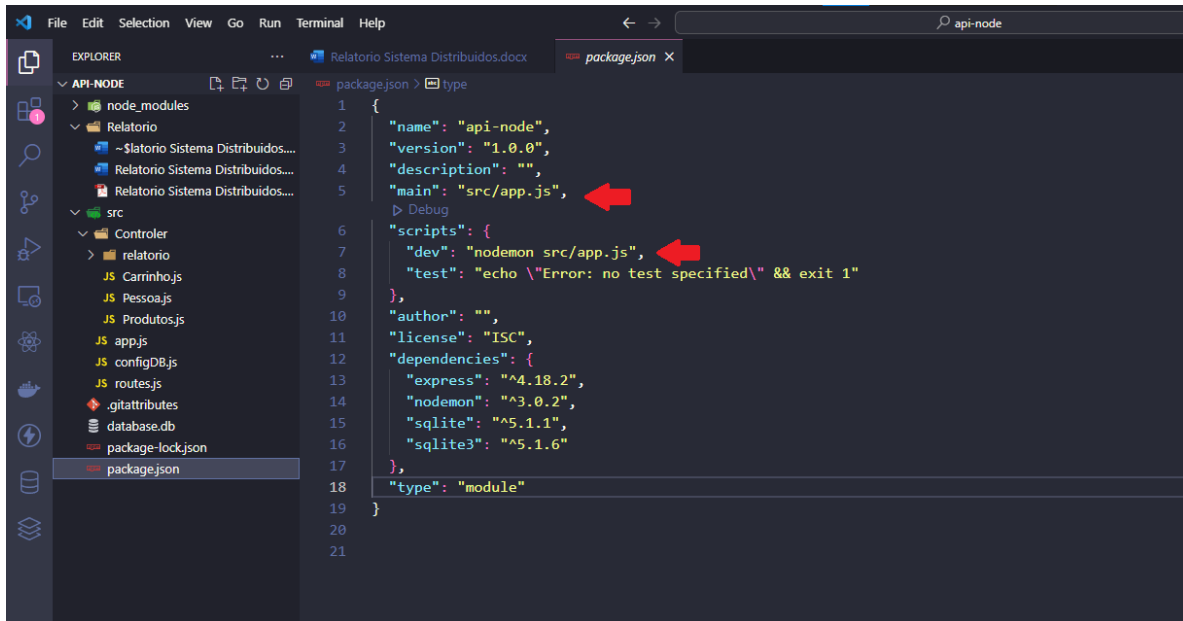
Primeiramente, é necessário a criação de uma pasta com o nome do projeto (ProjetoA3) e depois a instalação do **node\_modules** e o JSON com o comando **npm init**.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer panel displays a project structure under the name 'API-NODE'. The 'node\_modules' folder is highlighted with a red arrow. Below it, there's a 'Relatorio' folder containing several files: 'Carrinho.js', 'Pessoa.js', 'Produtos.js', 'app.js', 'configDB.js', 'routes.js', '.gitattributes', 'database.db', 'package-lock.json', and 'package.json'. The 'package.json' file is also highlighted with a red arrow. On the right, the package.json file is open in the editor, showing the following content:

```
1 {
2   "name": "api-node",
3   "version": "1.0.0",
4   "description": "",
5   "main": "src/app.js",
6   "scripts": {
7     "dev": "nodemon src/app.js",
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.18.2",
14    "nodemon": "^3.0.2",
15    "sqlite": "^5.1.1",
16    "sqlite3": "^5.1.6"
17  },
18  "type": "module"
19 }
```

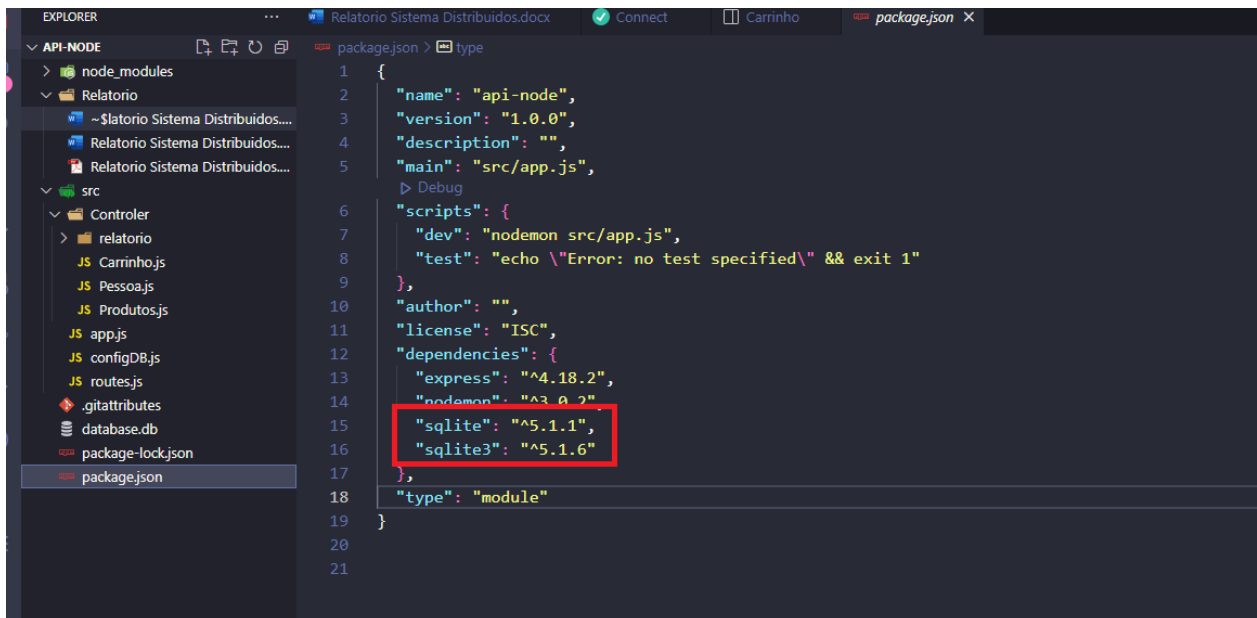
Vamos instalar o nodemon através do prompt com o comando *npm install nodemon* e em seguida colocar o main como “src/app.js” e depois o “dev” : “nodemon src/app.js” para atualizar o servidor sempre que o código for alterado.



The screenshot shows the VS Code interface with the Explorer view on the left displaying the project structure. The main editor shows the package.json file. Two red arrows point to the 'main' field (line 5) and the 'dev' script (line 7).

```
1 {
2   "name": "api-node",
3   "version": "1.0.0",
4   "description": "",
5   "main": "src/app.js",
6   "scripts": {
7     "dev": "nodemon src/app.js",
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.18.2",
14    "nodemon": "^3.0.2",
15    "sqlite": "^5.1.1",
16    "sqlite3": "^5.1.6"
17  },
18  "type": "module"
19 }
20
21
```

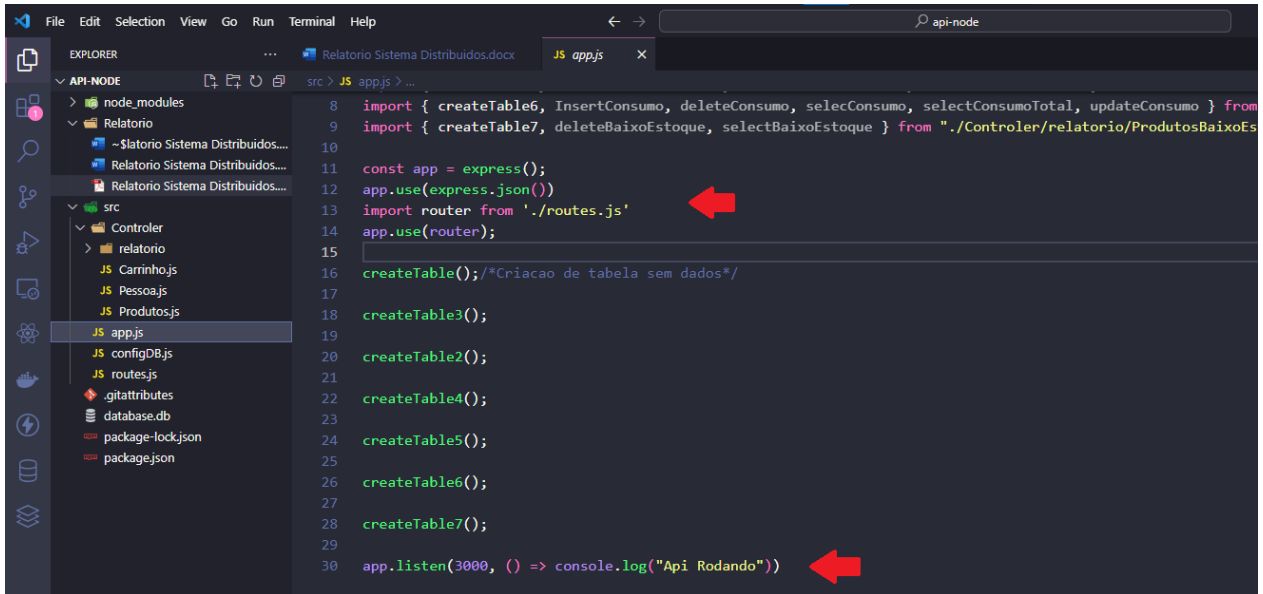
Depois disso vamos instalar o banco de dados sqLite com o comando *npm intall sqlite* e *npm install sqlite3*.



The screenshot shows the VS Code interface with the Explorer view on the left. The main editor shows the package.json file. A red box highlights the 'sqlite' and 'sqlite3' dependencies in the 'dependencies' object (lines 15 and 16).

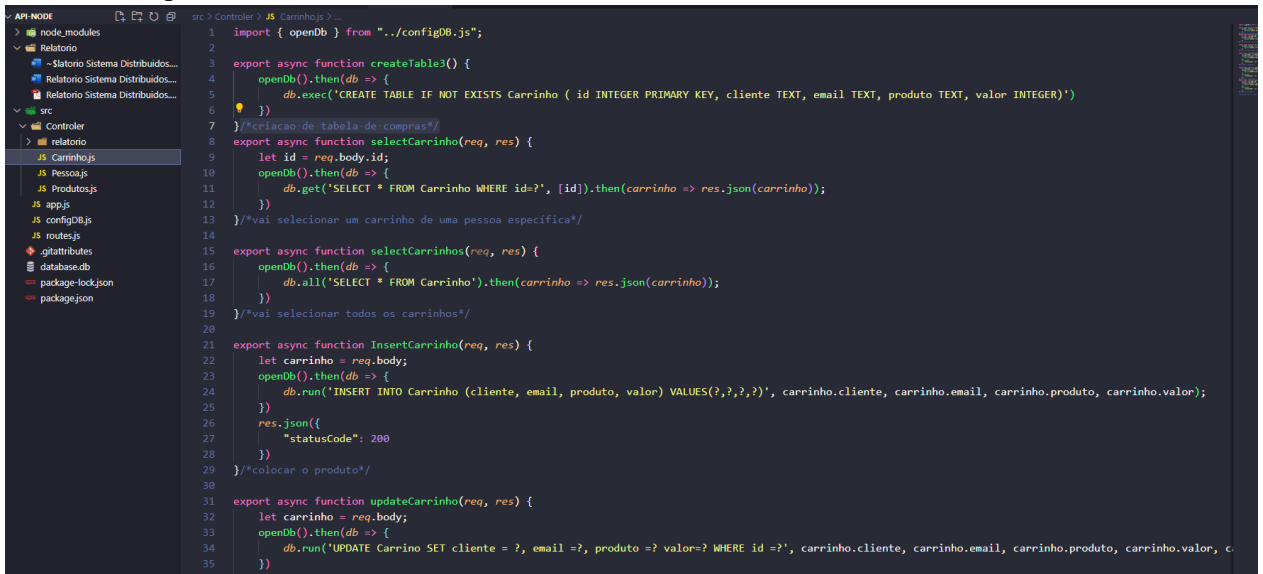
```
1 {
2   "name": "api-node",
3   "version": "1.0.0",
4   "description": "",
5   "main": "src/app.js",
6   "scripts": {
7     "dev": "nodemon src/app.js",
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.18.2",
14    "nodemon": "^3.0.2",
15    "sqlite": "^5.1.1",
16    "sqlite3": "^5.1.6"
17  },
18  "type": "module"
19 }
20
21
```

Em seguida vamos instalar a biblioteca **express** através do comando **npm install express**, e depois vamos fazer um require criando uma rota e em seguida criaremos uma porta 3000 como ouvinte.



```
8 import { createTable6, InsertConsumo, deleteConsumo, selecConsumo, selectConsumoTotal, updateConsumo } from
9 import { createTable7, deleteBaixoEstoque, selectBaixoEstoque } from "../Controler/relatorio/ProdutosBaixoEs
10
11 const app = express();
12 app.use(express.json())
13 import router from './routes.js'
14 app.use(router);
15
16 createTable(); /*Criação de tabela sem dados*/
17
18 createTable3();
19
20 createTable2();
21
22 createTable4();
23
24 createTable5();
25
26 createTable6();
27
28 createTable7();
29
30 app.listen(3000, () => console.log("Api Rodando"))
```

Em seguida vamos criar os códigos SQL para criarmos as tabelas dentro do arquivo **Carrinho.js**, **Pessoa.js**, **Produtos.js** na pasta **controler**, com os verbos **create**, **select**, **Insert**, **update** e **delete** e o **createTable** para criar a tabela mesmo estando vazia.



```
1 import { openDb } from "../configDB.js";
2
3 export async function createTable3() {
4   openDb().then(db => {
5     db.exec('CREATE TABLE IF NOT EXISTS Carrinho ( id INTEGER PRIMARY KEY, cliente TEXT, email TEXT, produto TEXT, valor INTEGER)')
6   })
7 } /*criação de tabela de compras*/
8 export async function selectCarrinho(req, res) {
9   let id = req.body.id;
10  openDb().then(db => {
11    db.get('SELECT * FROM Carrinho WHERE id=?', [id]).then(carrinho => res.json(carrinho));
12  })
13 } /*vai selecionar um carrinho de uma pessoa especifica*/
14
15 export async function selectCarrinhos(req, res) {
16   openDb().then(db => {
17     db.all('SELECT * FROM Carrinho').then(carrinho => res.json(carrinho));
18   })
19 } /*vai selecionar todos os carrinhos*/
20
21 export async function InsertCarrinho(req, res) {
22   let carrinho = req.body;
23   openDb().then(db => {
24     db.run('INSERT INTO Carrinho (cliente, email, produto, valor) VALUES(?,?,?,?)', carrinho.cliente, carrinho.email, carrinho.produto, carrinho.valor);
25   })
26   res.json({
27     "statusCode": 200
28   })
29 } /*colocar o produto*/
30
31 export async function updateCarrinho(req, res) {
32   let carrinho = req.body;
33   openDb().then(db => {
34     db.run('UPDATE Carrinho SET cliente = ?, email =?, produto =? valor=? WHERE id =?', carrinho.cliente, carrinho.email, carrinho.produto, carrinho.valor, c
35   })
36 }
```

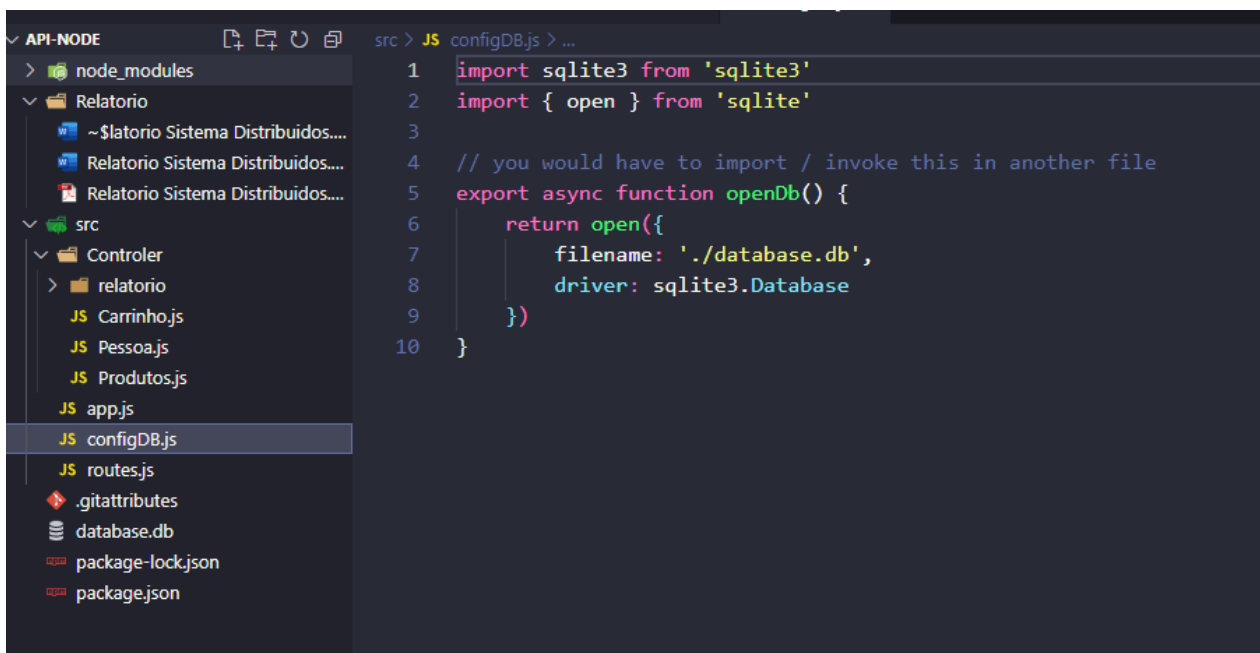


Em seguida vamos fazer o import dos arquivos *Pessoa.js*, *Produto.js*, *Carrinho.js* e instanciar o *createTable()* para criar a tabela, e depois fazer o *npm start dev* para iniciar o servidor apenas uma vez, por que o *nodemon* ja esta em execução.



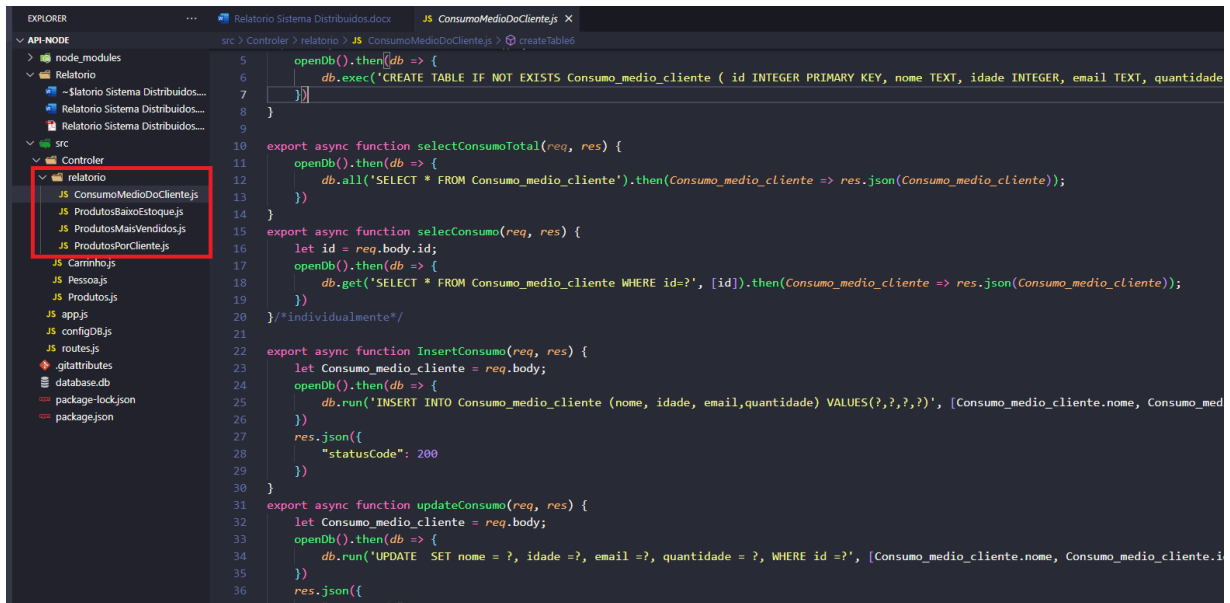
```
1 //import { openDb } from './configDB.js';
2 import express from 'express';
3 import { InsertPessoa, createTable, updatePessoa, selectPessoas, selectPessoa, deletePessoa } from './Controller/Pessoa.js';
4 import { InsertProduto, createTable2, updateProduto, selectProdutos, selectProduto, deleteProduto } from './Controller/Produto.js';
5 import { InsertCarrinho, createTable3, updateCarrinho, selectCarrinhos, selectCarrinho, deleteCarrinho } from './Controller/Carrinho.js';
6 import { createTable4, selectMaisVendido, selectMaisVendidos, deleteMaisVendido } from './Controller/relatorio/ProdutosMaisVendidos.js';
7 import { createTable5, InsertProdutoCliente, deleteProdutoCliente, selectProdutoCliente, selectProdutosClientes, updateProdutoCliente } from './Controller/relato';
8 import { createTable6, InsertConsumo, deleteConsumo, selectConsumo, selectConsumoTotal, updateConsumo } from './Controller/relatorio/ConsumoMedioDoCliente.js';
9 import { createTable7, deleteBaixoEstoque, selectBaixoEstoque } from './Controller/relatorio/ProdutosBaixoEstoque.js';
10
11 const app = express();
12 app.use(express.json());
13 import router from './routes.js'
14 app.use(router);
15
16 createTable(); /*Criação de tabela sem dados*/
17
18 createTable3();
19
20 createTable2();
21
22 createTable4();
23
24 createTable5();
25
26 createTable6();
27
28 createTable7();
29
30 app.listen(3000, () => console.log("Api Rodando"))
```

Vamos configurar o banco de dados em uma arquivo chamado *configDB.js*.



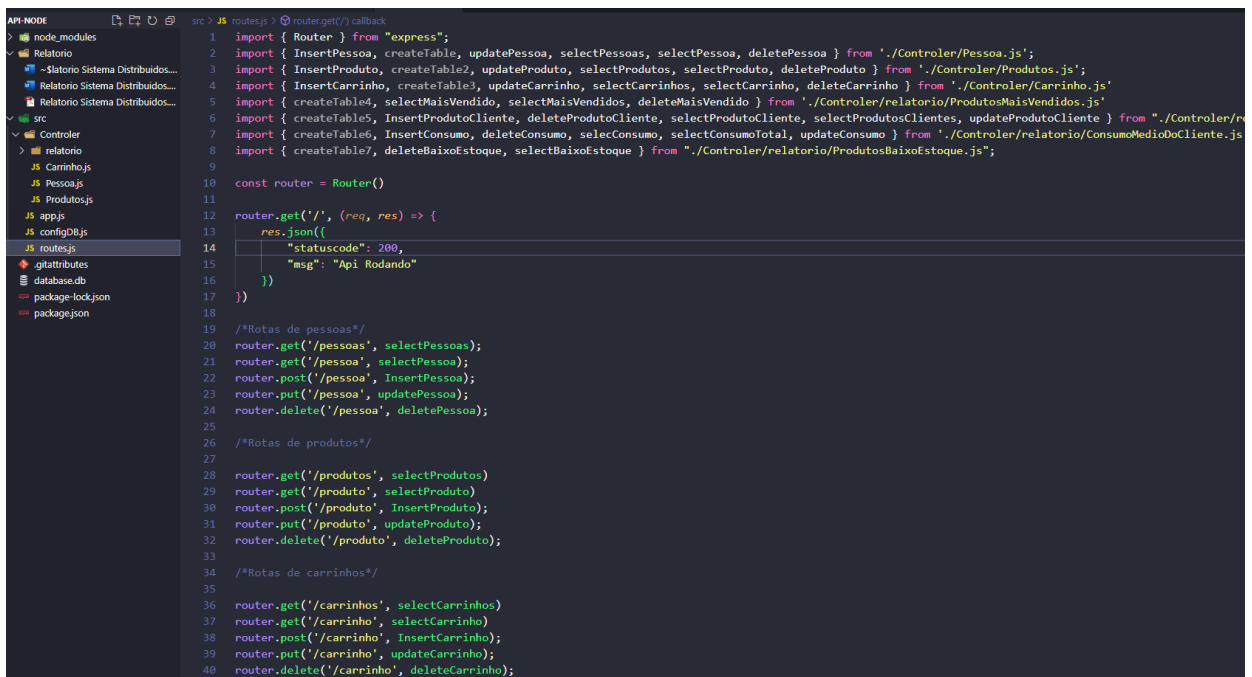
```
1 import sqlite3 from 'sqlite3'
2 import { open } from 'sqlite'
3
4 // you would have to import / invoke this in another file
5 export async function openDb() {
6   return open({
7     filename: './database.db',
8     driver: sqlite3.Database
9   })
10 }
```

Em seguida vamos criar as tabelas restantes em uma pasta chamada *relatorio* e depois fazer o import no *app.js* instanciando o *createTable*.



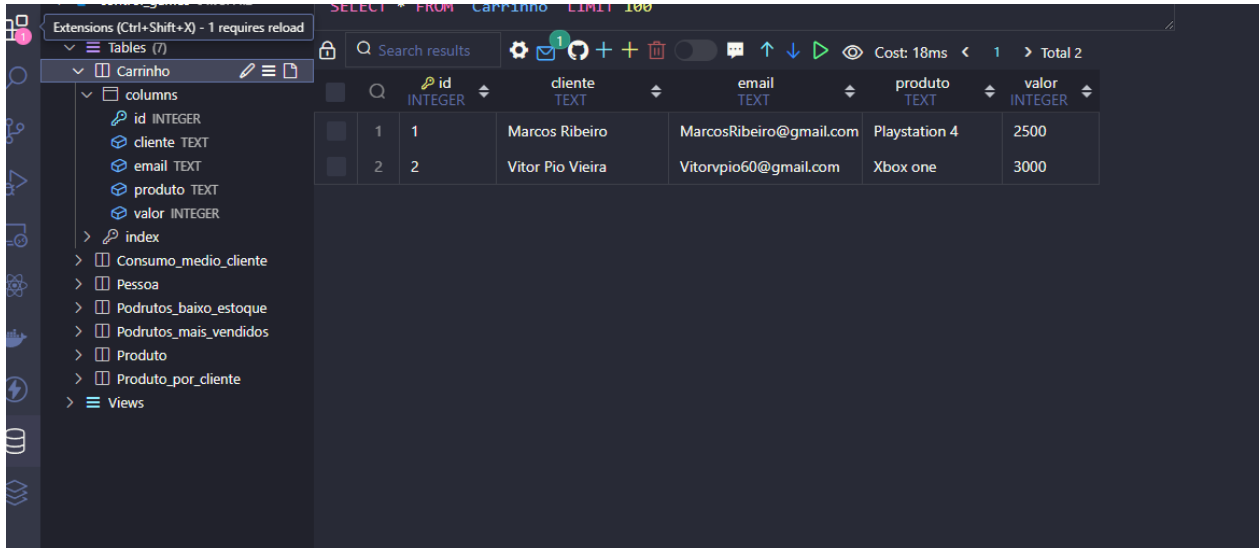
```
5 openDb().then(db => {
6   db.exec('CREATE TABLE IF NOT EXISTS Consumo_medio_cliente ( id INTEGER PRIMARY KEY, nome TEXT, idade INTEGER, email TEXT, quantidade
7   ));
8 }
9
10 export async function selectConsumoTotal(req, res) {
11   openDb().then(db => {
12     db.all('SELECT * FROM Consumo_medio_cliente').then(Consumo_medio_cliente => res.json(Consumo_medio_cliente));
13   })
14 }
15 export async function selectConsumo(req, res) {
16   let id = req.body.id;
17   openDb().then(db => {
18     db.get('SELECT * FROM Consumo_medio_cliente WHERE id=?', [id]).then(Consumo_medio_cliente => res.json(Consumo_medio_cliente));
19   })
20 } /*individualmente*/
21
22 export async function InsertConsumo(req, res) {
23   let Consumo_medio_cliente = req.body;
24   openDb().then(db => {
25     db.run('INSERT INTO Consumo_medio_cliente (nome, idade, email, quantidade) VALUES(?,?,?,?)', [Consumo_medio_cliente.nome, Consumo_med
26   ])
27   res.json({
28     "statusCode": 200
29   })
30 }
31 export async function updateConsumo(req, res) {
32   let Consumo_medio_cliente = req.body;
33   openDb().then(db => {
34     db.run('UPDATE SET nome = ?, idade = ?, email = ?, quantidade = ?, WHERE id = ?', [Consumo_medio_cliente.nome, Consumo_medio_cliente.i
35   ])
36   res.json({
37     "statusCode": 200
38   })
39 }
```

Em seguida vamos criar as rotas em um arquivo chamado *routes.js* com o *post*, *delete*, *post*, *put*, e *get*.

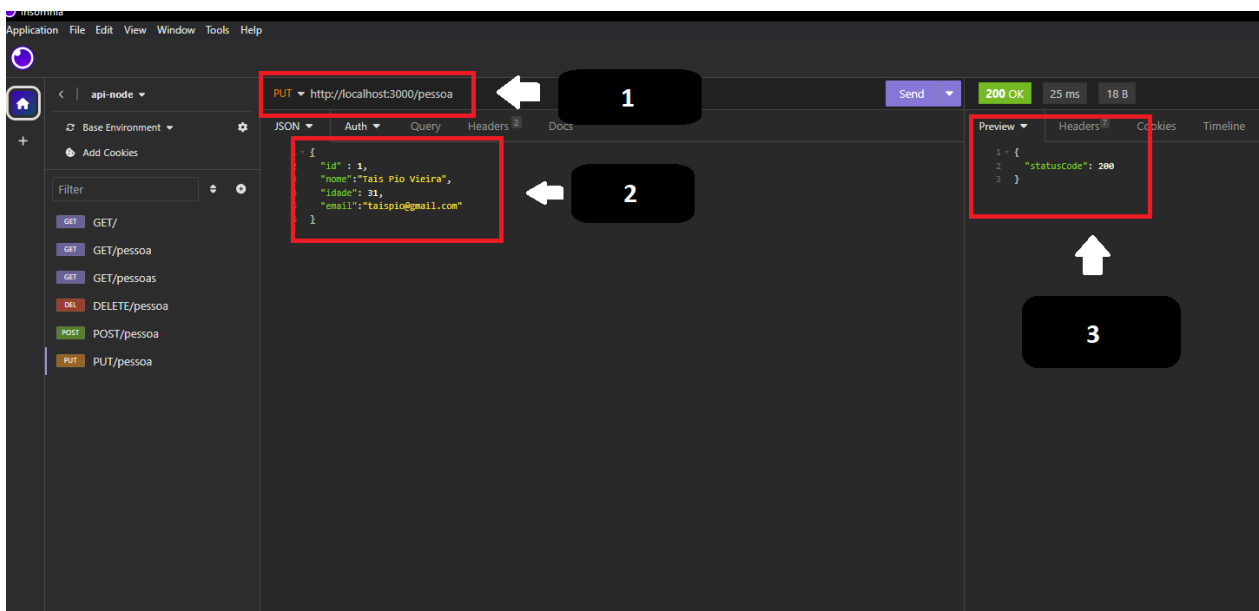


```
1 import { Router } from "express";
2 import { InsertPessoa, createTable, updatePessoa, selectPessoas, selectPessoa, deletePessoa } from './Controler/Pessoa.js';
3 import { InsertProduto, createTable2, updateProduto, selectProdutos, selectProduto, deleteProduto } from './Controler/Produtos.js';
4 import { InsertCarrinho, createTable3, updateCarrinho, selectCarrinhos, selectCarrinho, deleteCarrinho } from './Controler/Carrinho.js'
5 import { createTable4, selectMaisVendido, selectMaisVendidos, deleteMaisVendido } from './Controler/relatorio/ProdutosMaisVendidos.js'
6 import { createTable5, InsertProdutoCliente, deleteProdutoCliente, selectProdutoCliente, selectProdutosClientes, updateProdutoCliente } from './Controler/relatorio/ProdutosPorCliente.js'
7 import { createTable6, InsertConsumo, deleteConsumo, selectConsumo, selectConsumoTotal, updateConsumo } from './Controler/relatorio/ConsumoMedioDoCliente.js'
8 import { createTable7, deleteBaixoEstoque, selectBaixoEstoque } from './Controler/relatorio/ProdutosBaixoEstoque.js';
9
10 const router = Router()
11
12 router.get('/', (req, res) => {
13   res.json({
14     "statusCode": 200,
15     "msg": "Api Rodando"
16   })
17 })
18
19 /*Rotas de pessoas*/
20 router.get('/pessoas', selectPessoas);
21 router.get('/pessoa', selectPessoa);
22 router.post('/pessoa', InsertPessoa);
23 router.put('/pessoa', updatePessoa);
24 router.delete('/pessoa', deletePessoa);
25
26 /*Rotas de produtos*/
27
28 router.get('/produtos', selectProdutos)
29 router.get('/produto', selectProduto)
30 router.post('/produto', InsertProduto);
31 router.put('/produto', updateProduto);
32 router.delete('/produto', deleteProduto);
33
34 /*Rotas de carrinhos*/
35
36 router.get('/carrinhos', selectCarrinhos)
37 router.get('/carrinho', selectCarrinho)
38 router.post('/carrinho', InsertCarrinho);
39 router.put('/carrinho', updateCarrinho);
40 router.delete('/carrinho', deleteCarrinho);
```

Depois disso vamos adicionar o arquivo *database.db* e visualizar as tabelas com um plugin do *vsCode* chamado *mySQL*.



Agora utilize ferramentas como *Postman* ou *Insomnia* para testar as rotas da API:



- 1: Definir a rota e o metodo desejado.
- 2: Utilizar o formato JSON para manipular as informações e depois clicar em Send.
- 3: Mensagem 200 indicando que a operação foi bem feita

## **4 - REQUERIMENTOS DE SOFTWARES NECESSÁRIOS PARA EXECUÇÃO DA APLICAÇÃO**

Para a execução da API é necessário a instalação de algumas bibliotecas como: express "4.18.2", sqlite: 5.1.1", sqlite3: "5.1.6", nodemon: "3.0.2" e o nodeJS lts.

- 4.1** A biblioteca express vai ajudar a facilitar na criação de rotas, possibilita no tratamento de exceções, permite a integração de vários sistemas templates e gerencia as requisições HTTP.
- 4.2** O banco de dados relacional sqlite vai ajudar no gerenciamento das tabelas criadas.
- 4.3** O nodemon é uma ferramenta que fornece a capacidade de rodar o código sempre que tiver alguma atualização.
- 4.4** O nodeJs é a linguagem de programação responsável pela construção da API.

## 5 - JUSTIFICATIVA DAS TECNOLOGIAS ESCOLHIDAS

O Node.js é uma plataforma de computação de código aberto baseada em eventos para a construção de aplicações web e de rede.

- Node.js foi escolhido para o projeto de gestão de vendas por ser uma plataforma adequada para o desenvolvimento de aplicações web. Ele é rápido, escalável e fácil de usar.

O Express é um framework web para Node.js que fornece um conjunto de middlewares e ferramentas para facilitar o desenvolvimento de aplicações web.

- O Express foi escolhido para o projeto de gestão de vendas por ser um framework popular e bem documentado. Ele fornece uma base sólida para o desenvolvimento de aplicações web, incluindo recursos como roteamento, middlewares e validação.

O SQLite3 é um banco de dados relacional embutido que é usado para armazenar dados em aplicações web e de rede.

- O SQLite3 foi escolhido para o projeto de gestão de vendas por ser um banco de dados rápido, eficiente e fácil de usar e tbm por criar tabelas independentes.

O Insomnia é uma ferramenta de teste de API que permite testar e depurar APIs de forma fácil e rápida.

- O Insomnia foi escolhido para o projeto de gestão de vendas por ser uma ferramenta popular e bem documentada. Ele fornece uma variedade de recursos que ajudam a testar e depurar APIs, incluindo suporte para diferentes protocolos, autenticação e geração de relatórios.

As tecnologias mencionadas acima foram usadas por oferecem uma combinação de recursos robustos, escaláveis e eficientes.

As tecnologias escolhidas são adequadas para atender às necessidades do projeto, que são:

- Atender às necessidades essenciais de gerenciamento, envolvendo aspectos cruciais como clientes, estoque, vendas e a geração de relatórios estatísticos perspicazes.
- Oferecer uma solução adaptável às preferências e habilidades da equipe de desenvolvimento.
- Garantir a solidez do banco de dados ao optar por uma abordagem relacional, fundamental para a integridade e consistência dos dados.

## **6 - APRESENTAÇÃO E DETALHAMENTO SOBRE A ARQUITETURA, ESTRATÉGIA E ALGORÍTIMO UTILIZADO**

Nos algoritmos, cada operação CRUD (Create, Read, Update, Delete) é implementada com instruções SQL básicas. Por exemplo, a criação de um novo item envolve uma instrução INSERT INTO na tabela correspondente. Os relatórios estatísticos envolvem consultas SQL mais complexas para extrair informações específicas do banco de dados. Por exemplo, o relatório de produtos mais vendidos pode exigir uma consulta que envolva operações de agrupamento e ordenação.

A lógica de roteamento e manipulação de requisições os algoritmos que controlam como as solicitações HTTP são tratadas, desde a validação até a execução da lógica de negócios, são implementados nos controladores e nas rotas. Aqui, utilizamos a flexibilidade do JavaScript para criar lógicas assíncronas quando necessário.

Essa estratégia e arquitetura proporcionam uma aplicação modular, escalável e fácil de entender. O uso de SQLite simplifica a configuração do banco de dados, enquanto o Express facilita a criação de uma API RESTful para operações CRUD e geração de relatórios estatísticos. Escolhemos a arquitetura API(Application Programming Interfaces) que permite a interação de uma aplicação com outra por meio da requisição e resposta. Quando a AP1 solicita um dado da AP2, a AP2 envia a informação por meio de uma resposta. Roteamento e controladores utilizamos o framework Express para facilitar o roteamento e a criação de controladores. Cada recurso (clientes, vendas, etc.) tem seu próprio conjunto de rotas e controladores para lidar com as operações CRUD. O SQLite foi escolhido como o banco de dados relacional, por ser leve, fácil de configurar e suficientemente poderoso para uma aplicação de pequena a média escala, e por fim o npm para gerenciamento de dependências da aplicação. Isso permite que outras pessoas reproduzam facilmente o ambiente de desenvolvimento e instalem as dependências necessárias.

A estratégia utilizada nessa aplicação de vendas é que essa arquitetura compõe um único software baseado em um conjunto de serviços autônomos e simples, além disso, ele permite que, diferentes sistemas se comuniquem entre si e o código seja reutilizado várias vezes e também no controle de funcionalidades específicas permitindo um sistema mais seguro.

## **7- CONSIDERAÇÕES FINAIS**

Desenvolvemos essa API com base na arquitetura REST aprendidas em aula com o professor Wellington Lacerda e Adailton Cerqueira para mostrar o que aprendemos durante o semestre que começou no início de Agosto e finalizou no início de Dezembro para a disciplina de Sistemas distribuídos para o terceiro e quarto semestre. Essa aplicação de certa forma contribuiu para o nosso aprendizado sobre as questões abordadas durante as aulas e também a realização da tarefa em equipe

## 8-BIBLIOGRAFIA

Npm: <https://www.npmjs.com>

Red hat: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>

Hostinger: <https://www.hostinger.com.br/tutoriais/api-restful>

Totvs: <https://www.totvs.com/blog/developers/>

Amazon: <https://aws.amazon.com/what-is/api/>