

# App - Estoque

## Objetivos do projeto

- Criação do Banco de Dados em SQLite; ✓
- Criação da API; ✓
- Criação do Front-End em React; ✗

## Criação do Banco de Dados em SQLite

Foi iniciado o projeto com intenção de estruturar os arquivos em um modelo Client/Server, por ser uma estrutura mais simples eu poderia focar nas minhas áreas de dificuldade, como Front-end.

## Dependências instaladas

- SQLite3 - Para criação e manutenção do banco de dados.
- Knex - Utilizada para facilitar/mitigar o trabalho da criação de Queries para o banco, e pensando no caso de um dia esse projeto "precisar" mudar sua estrutura do banco, talvez para MySQL ou outro.

## Desenvolvimento

Criado pasta contendo arquivo *produtos.js* e *knex.js*, dentro do diretório `APP-ESTOQUE/server/db` .

O arquivo *knex.js* informa o nome do arquivo que vamos usar para o banco de dados e também o tipo de conexão.

Recursos que me ajudaram a entender melhor sobre a ferramenta foram: <http://knexjs.org/>, <https://devhints.io/knex>, <https://dev.to/easybuoy/setting-up-a-node-api-with-postgres-and-knex-588f>.

O arquivo *produtos.js* utiliza diretamente as configurações do arquivo *knex.js* para criarmos as funções de inserção, exclusão, edição de registros no banco de dados, no final é exportado cada função criada para usarmos no nosso arquivo de comunicação.

```

1  const knex = require("./knex");
2
3  function cadastrarProduto(produto) {
4    return knex("produtos").insert(produto);
5  };
6
7  function listarTodosProdutos() {
8    return knex("produtos").select("*");
9  };
10
11 function deletarProduto(id) {
12   return knex("produtos").where("id", id).del();
13 };
14
15 function editarProduto(id, produto) {
16   return knex("produtos").where("id", id).update(produto);
17 }
18
19 module.exports = {
20   cadastrarProduto,
21   listarTodosProdutos,
22   deletarProduto,
23   editarProduto
24 }

```

Já no diretório `APP-ESTOQUE/server` foi criado um arquivo chamado *produtos.sqlite3*, este arquivo vai conter o código dos dados da nossa tabela produtos, composta pelos campos *id*, *nome*, *preco* e *produto*.

Utilizei o programa DB Browser para SQLite, para criar a tabela apontando o caminho do arquivo *produtos.sqlite3*.

E depois dentro do arquivo de conexão da api estruturei os comandos referentes a comunicação com banco de dados.

# Criação da API

## Dependências instaladas

- Body-Parser - Middleware de comunicação que ajuda a receber as requisições de formulários.
- Cors - Tive um problema ao fazer o envio do método POST e vi em um artigo que ele ajudaria na solução de alguns problemas de comunicação na hora de compartilhar recursos do navegador.

["https://www.treinaweb.com.br/blog/o-que-e-cors-e-como-resolver-os-principais-erros"](https://www.treinaweb.com.br/blog/o-que-e-cors-e-como-resolver-os-principais-erros)


- Express - Framework para auxiliar a criação da api com Node.
- Nodemon - Dependência que facilita o trabalho na hora de desenvolver, por evitar ter que ficar reiniciando o servidor a cada mudança.

## Desenvolvimento

Nunca havia criado uma API, então precisei primeiro me inteirar sobre o assunto, como funcionava a comunicação e para que servia.

Fontes que me ajudaram a entender melhor variam desde artigos, como <https://stackabuse.com/building-a-rest-api-with-node-and-express/>, <https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291>, e também vídeos no youtube passo a passo de como a estrutura de uma API funcionava <https://www.youtube.com/watch?v=cr3pX6fSUpc>.

Dentro do arquivo `server.js` importei os módulos necessários e criei os blocos de comunicação para cada função criada anteriormente no arquivo de configuração do banco de dados.



```
1  const bodyParser = require('body-parser');
2  const express = require('express');
3  const app = express();
4  const db = require("./db/produtos");
5
6  app.use(bodyParser.urlencoded({ extended: false }));
7  app.use(bodyParser.json());
8
9  app.post("/produtos", async (req, res) => {
10     const results = await db.cadastrarProduto(req.body);
11     res.status(201).json({ id: results[0] });
12 });
13
14 app.get("/produtos", async (req, res) => {
15     const produtos = await db.listarTodosProdutos();
16     res.status(200).json({ produtos });
17 });
18
19 app.patch("/produtos/:id", async (req, res) => {
20     const id = await db.editarProduto(req.params.id, req.body);
21     res.status(200).json({ id });
22 });
23
24 app.delete("/produtos/:id", async (req, res) => {
25     await db.deletarProduto(req.params.id);
26     res.status(200).json({ success: true });
27 });
28
29 //app.get('/', (req, res) => { res.status(200).json({ success: true }); });
30
31 app.listen(3001, ()=>{ console.log('Servidor rodando na porta 3001'); });
```

## Criação do Front-end em REACT

Foi a parte que subestimei, criei apenas uma tela dentro do arquivo principal *app.js* e foquei em realizar a comunicação entre o front-end e API.

Realizei alguns testes, e li alguns artigos sobre o assunto, consegui depois de ver um erro sobre o CORS que o front estava conseguindo alcançar a requisição, mas ela não retornava com os dados, resolvi alguns erros de sintaxe e por já estar quase no limite do tempo, usei o *client/App.js* como tela principal para tentar pelo menos exibir os dados do array retornado pela API.

### Dependências instaladas

- React-Router - Comecei a pesquisar sobre a atualização da tabela utilizando o *react-router-dom*, a estrutura havia ficado confusa demais, então decidi voltar e focar em tentar exibir os dados dos itens na tela.

Algumas das fontes que utilizei para estudar:

<https://medium.com/collabcode/roteamento-no-react-com-os-poderes-do-react-router-v4-fbc191b9937d>

<https://medium.com/@henrique.weiand/react-react-router-dom-navegação-entre-telas-plano-de-aula-xii-880b91e4bd9c>

<https://reactrouter.com/web/guides/quick-start>

- Axios - Para realizar a comunicação essa foi a ferramenta que escolhi para manipular os pedidos e respostas.
- Bootstrap - Apesar de não ter dado foco ao front-end, uma das bibliotecas que usaria, acabei instalando para já preparar as telas, por que facilita na criação de padrões de layout, mas não cheguei a utilizar.

Quando consegui identificar o problema com o CORS, tive outro problema também com *.map()* ao tentar buscar os registros do array criado, consegui achar alguns projetos no github em Typescript e com base neles achar uma forma correta de trazer os dados do array dentro do console.