



Ryan Luca

PRODUTOS NATURAIS

BACKEND - Node.js
Express



SUMÁRIO

1 Objetivo do Projeto

2 Stack do projeto

3 Criação do projeto

4 Pacotes instalados

5 Configurando o Servidor Express

6 Integração com Supabase

7 Rotas da API

8 CRUD

9 Postman

10 Estrutura Final do Projeto

11 Conclusão



OBJETIVO DO PROJETO

- 
- O objetivo foi criar um Back-End completo usando Node.js e Express.
 - A API representa um comércio (no seu caso, uma loja de produtos naturais a granel).
 - A API permite que outros sistemas (como um front-end) enviem requisições para consultar, criar, atualizar ou excluir produtos
 - Essa API se comunica com um banco de dados externo (Supabase).
- 



STACK DO PROJETO

> Node.js:

Plataforma que executa JavaScript no back-end.

> Express:

Framework que facilita criar rotas e servidores HTTP.

> Supabase:

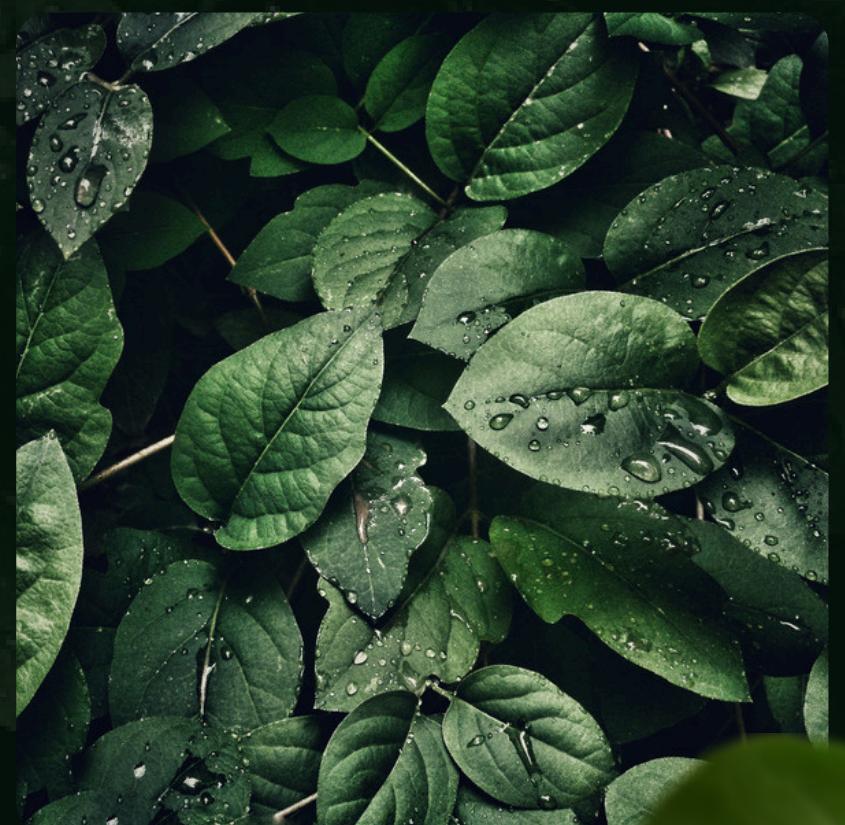
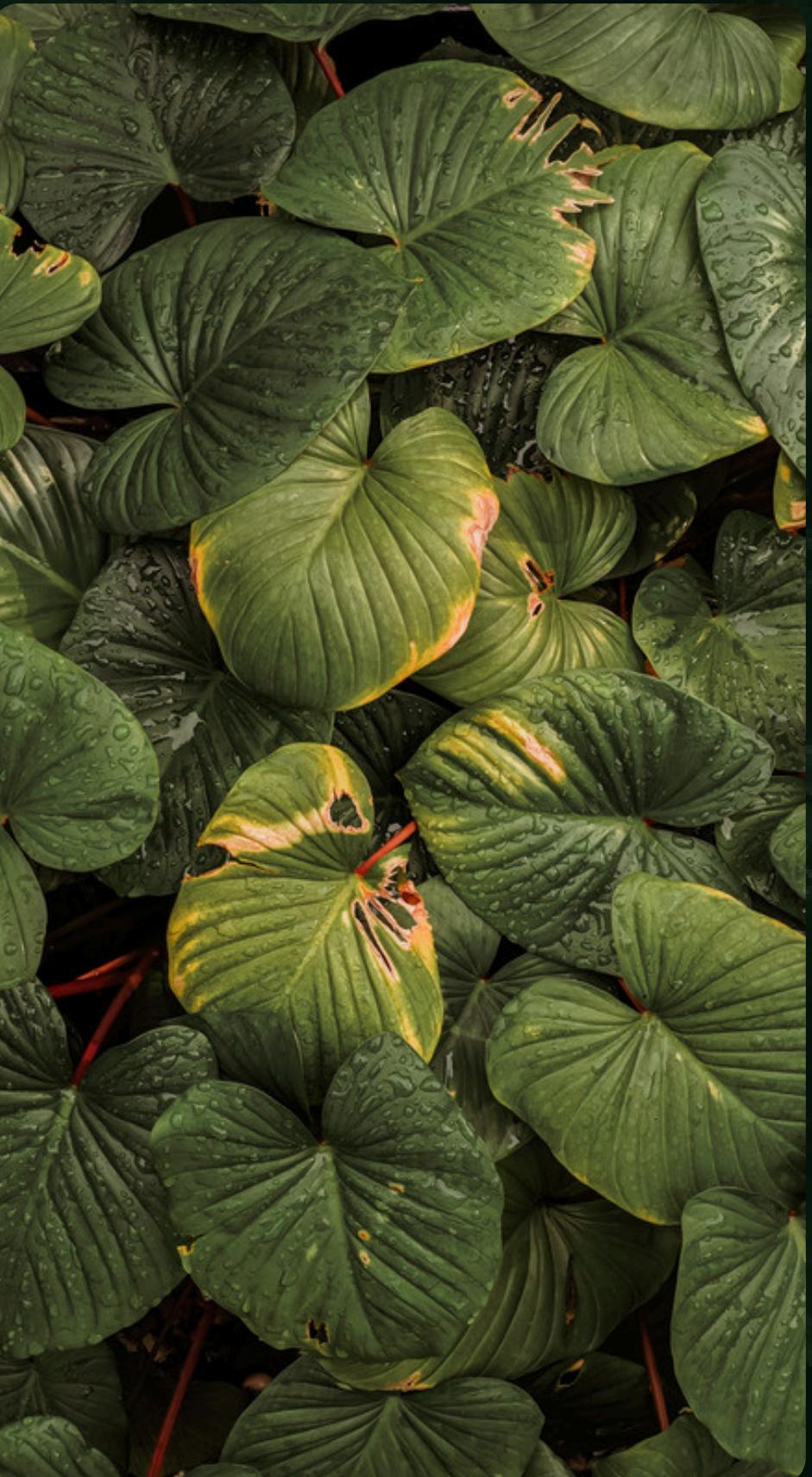
Banco de dados online que armazena os produtos.

> Nodemon:

Ferramenta que reinicia o servidor automaticamente sempre que alteramos o código

> JavaScript:

Linguagem usada no projeto.



CRIAÇÃO DO PROJETO

Nesta etapa criamos a pasta do projeto e iniciamos o Node dentro dela usando:

npm init -y

Depois montamos uma estrutura organizada de pastas, separando:

- **Rotas**
- **Controladores**
- **Configuração**
- **Arquivos principais**

Isso ajuda a manter o projeto limpo e profissional.



PACOTES INSTALADOS

express

cria o servidor e rotas.

nodemon

facilita o desenvolvimento.

@supabase/supabase-js

conecta o projeto ao Supabase.

Também atualizamos o package.json para facilitar rodar o servidor:

npm run dev





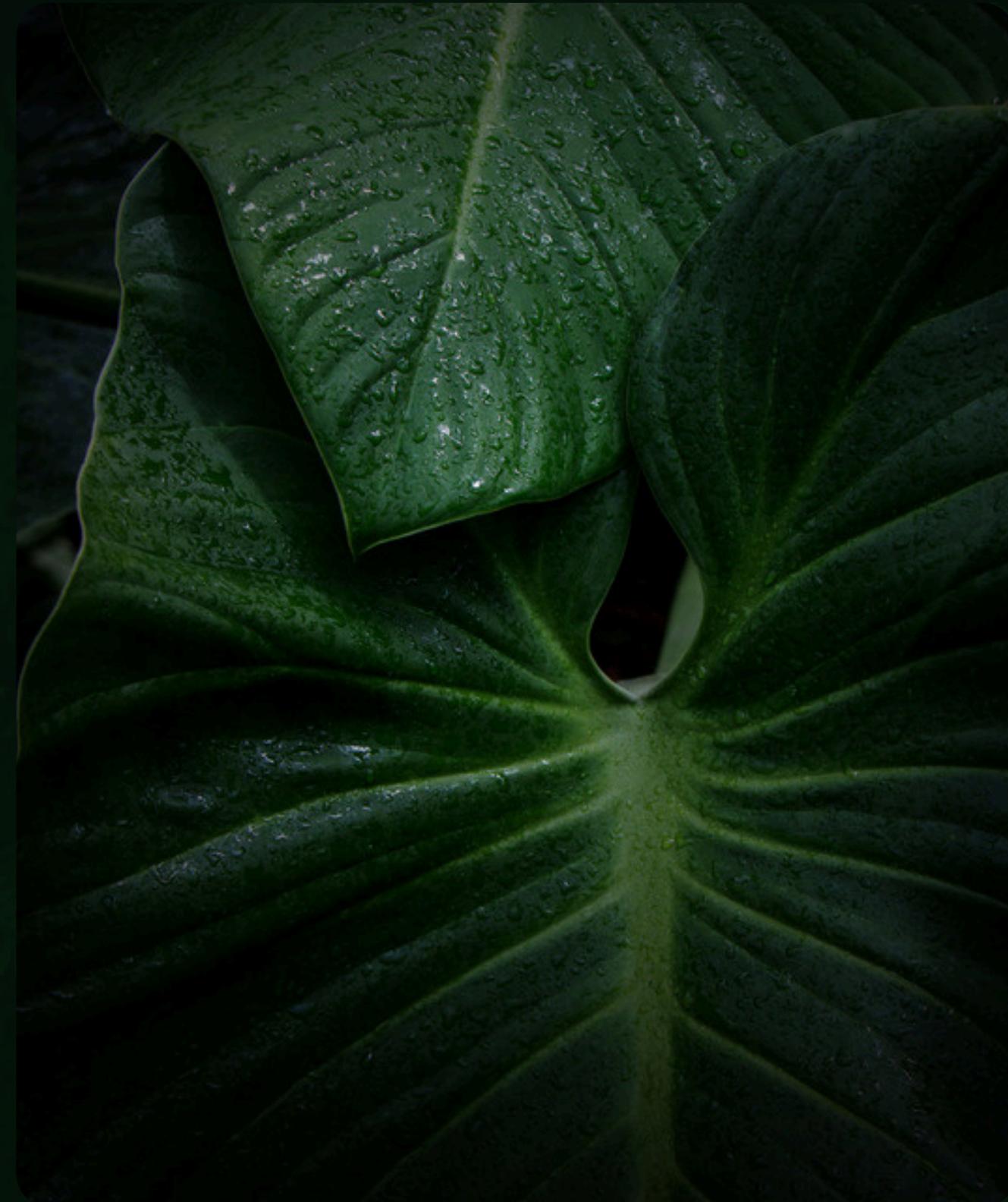
CONFIGURANDO O SERVIDOR EXPRESS



Criamos o arquivo server.js, que:

- Importa o Express.
- Inicia o servidor.
- Habilita o uso de JSON.
- Importa os arquivos de rotas.
- Faz o servidor rodar na porta 3000.

Esse arquivo é como o "centro de comando" da aplicação.



INTEGRAÇÃO COM SUPABASE

Criamos um cliente de conexão com o Supabase:

- Colocamos a URL do projeto.
- Colocamos a chave de acesso.
- Exportamos para usar em outras partes.

Também criamos no banco uma tabela chamada “produtos” com campos:

- id
- nome
- preco
- imagem

Essa tabela representa os dados que a API controla.



ROTAS DA API

As rotas são os “caminhos” que o usuário pode acessar.

No caso dos produtos, criamos:

- **GET** /produtos – lista todos os produtos.
- **GET** /produtos/:id – busca um produto específico.
- **POST** /produtos – cria um novo produto.
- **PUT** /produtos/:id – atualiza um produto.
- **DELETE** /produtos/:id – exclui um produto.

Cada rota chama uma função do controlador.



CRUD

lógica das ações:

- **Receber a requisição.**
- **Enviar para o Supabase.**
- **Retornar uma resposta ao usuário.**

Por exemplo:

- **Criar produto (POST)**

Recebe um JSON do usuário e envia para o banco:

```
supabase.from("produtos").insert([req.body]);
```

- **Listar (GET)**

```
supabase.from("produtos").select("*");
```

Temos também atualizar e deletar.

Essas funções fazem toda a comunicação com o banco.



Postman

Agora testamos as rotas para garantir que tudo funciona.

Exemplo:

- Envio um **POST** → cria um novo produto.
- Mando **GET** → vejo se o produto apareceu na lista.
- Faço **PUT** → atualizo dados.
- Faço **DELETE** → ele remove do banco.

Se tudo funciona, a API está pronta.



ESTRUTURA FINAL DO PROJETO

Mostramos o resultado final:

```
/Backend
  └── server.js
  └── /src
    └── routes/
    └── controllers/
      └── supabaseClient.js
```

Isso garante:

- Organização
- Separação de responsabilidades
- Manutenção mais fácil



Conclusão

E aqui fechamos a apresentação mostrando:

- A API RESTful foi criada com sucesso.
- Está conectada ao banco.
- Possui CRUD completo.
- Está estruturada de forma profissional.
- Pode ser integrada ao Front-End quando necessário.