Documentação de API para controle de inventário da loja X

APIs (Interfaces de Programação de Aplicação) referem-se a softwares com determinadas funções, cuja interface interliga dois serviços de comunicação, solicitações do usuário (cliente) e respostas a ele (servidor). Mais especificamente será utilizada uma API do tipo REST, pois o cliente enviará dados ao servidor e este executará funções internas para retorná-los ao cliente.

APIs REST possuem um conjunto de funções essenciais para o controle de banco de dados desde o básico até o avançado, como o GET, POST, PUT, PATCH e DELETE. Os clientes podem utilizar essas funções para acessar dados no servidor através do HTTP, protocolos para comunicação entre eles, assim como URLs digitadas no navegador para visitar um site qualquer.

A seguinte documentação apresenta informações de como foi desenvolvida a estrutura de uma API para atender uma contratação de serviço da loja X para administrar e controlar o inventário desta, de modo que o administrador possa consultar, adicionar, atualizar de modo parcial e total e deletar os produtos do inventário. Cada produto possuirá um código único, nome, sua descrição, preço, quantidade em estoque e prazo de validade.

A estrutura do projeto, pastas, arquivos e processos, organizada de maneira eficiente facilita o desenvolvimento e a manutenção posterior da API. Abaixo está a estrutura deste projeto com algumas informações importantes de pastas e arquivos utilizados para o desenvolvimento da API:

/adminInventory >Pasta raiz

|--- config >Pasta para a configuração do banco de dados

|--- inventory\_db.js >Arquivo para conexão ao banco de dados

|--- controllers >Pasta para configurar os controladores das transações  
 |--- transactionsController.js >Arquivo para manipular as requisições  
|--- routes >Pasta das rotas da API  
 |--- transactions.js >Arquivo que contém as rotas das transações  
|--- .env >Arquivo de variáveis ambiente  
|--- server.js >Arquivo com códigos para o servidor  
|--- package.json >Arquivo de configuração do npm  
|--- package-lock.json >Arquivo que lista módulos do node

Será utilizado a linguagem de programação em JavaScript, para isso deve ser necessário está instalado na máquina o Node.js e também o seu arquivo de gerenciador de pacotes, o npm que permite instalar bibliotecas, frameworks e outras ferramentas para o projeto e que são disponibilizados para uso público.

Para iniciar o projeto primeiro deve-se instalar as dependências que incluem as bibliotecas que serão utilizadas durante os processos de análise de dados no servidor para responder ao cliente de maneira fluida e eficiente, o código abaixo refere-se às instalações.

npm init -y

npm install express cors body-parser dotenv mysql2

npm install nodemon –save-dev

Essas dependências são: express, cors, body-parser, dotenv, nodemon e mysql2, uma framework para criar servidores web e APIs, um middleware que permite requisições externas ao servidor, outro middleware que analisa o corpo das requisições HTTP que podem ser acessadas em notação de objeto (.json), um módulo responsável por carregar as variáveis de ambiente, uma ferramenta de desenvolvimento que reinicia o servidor toda vez que um arquivo é modificado e um pacote que oferece recursos melhorados de autenticação à banco de dados, respectivamente.

Também deve ser criado o banco de dados, dados organizados e armazenados digitalmente em um sistema relacional com linhas e tabelas que poderão ser consultados, armazenados, modificados ou deletados, exatamente apropriado para a administração de um inventário de qualquer empresa.

Um SGBD (Sistema de Gerenciamento de Banco de Dados) compatível com as instalações feitas anteriormente é o MySQL que foi projetado e otimizado para aplicações web e pode ser executado em qualquer plataforma, possui segurança, é fácil integração, além de contar com opções de ajustes, backups e recuperações de dados. O MySQL trabalha com banco de dados relacionais casando com o projeto proposto de criação de API para controle de inventário.

Para armazenar os produtos do inventário da loja X foi criado um banco de dados nomeado de **inventory\_db** e uma tabela chamada **products** no MySQL, com algumas características já citadas anteriormente, código único sendo uma chave primária para consultas específicas e interligações futuras com outras tabelas, nome, descrição, categoria, preço, estoque e data de validade de cada produto. Abaixo está o código de criação e inserção dos dados dos produtos fornecidos pelo responsável do inventário:

CREATE DATABASE inventory\_db;

USE inventory\_db;

CREATE TABLE products (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255),

description TEXT,

category VARCHAR(255),

price DECIMAL(10,2),

stock INT,

expiry\_date DATE

);

INSERT INTO `products` VALUES

(1, 'Creme de avelã', 'Creme de leite com avelã', 'Creme', 4.99, 20, 20250625),

(2, 'Creme com passas', 'Creme de leite com passas', 'Creme', 4.99, 15, 20250601),

(3, 'Sorvete de bacuri', 'Sorvete sabor bacuri', 'Sorvete', 11.99, 8, 20241212),

(4, 'Sorvete de cajá', 'Sorvete sabor cajá', 'Sorvete', 11.99, 7, 20250114),

(5, 'Picolé de chocolate', 'Picolé sabor chocolate', 'Picolé', 3.99, 18, 20241030),

(6, 'Picolé de abóbora', 'Picolé sabor abóbora', 'Picolé', 3.99, 15, 20241028);

Com o banco de dados criado, agora devemos adicionar as variáveis de ambiente no arquivo .env, na raíz do projeto, que serão carregadas pelo dotenv e utilizadas na configuração da conexão com o banco de dados, ou seja, o .env permite que as variáveis sejam carregadas no process.env para uso do código e caso o banco de dados não conecte-se ao servidor, o erro pode está nos dados das variáveis de ambiente. Esses dados são sigilosos, pois possuem user e password do responsável pelo inventário da loja. A seguir o código:

DB\_HOST = 0.0.0.0

DB\_USER = your\_username

DB\_PASS = your\_password

DB\_NAME = inventory\_db

Antes de conectarmos ao banco de dados, precisamos iniciar o servidor. No arquivo server.js utilizaremos algumas dependências instaladas anteriormente, o express, cors, body-parser e dotenv. O express é responsável pela criação e gerenciamento do servidor da API, o dotenv vai gerenciar as variáveis de ambiente, cors vai permitir requisições externas e o body-parser analisa o corpo das requisições e permitir que o servidor manipule os dados enviados pelo cliente.

Devemos importá-las e armazená-las em variáveis específicas e carregar as variáveis de ambiente do arquivo .env para que o código process.env possa permitir chamar as variáveis em qualquer parte do código inteiro.

const express = require('express');

const dotenv = require('dotenv');

const cors = require('cors');

const bodyParser = require('body-parser');

dotenv.config();

Depois inicializar a variável express em uma função para chama-la também quando quisermos em outro lugar no código, usaremos esta função para usar o cors para habilitar todas as rotas da aplicação e o body-parser para analisar as requisições em formato JSON enviadas pelo cliente.

const app = express();

app.use(cors());

app.use(bodyParser.json());

Então definiremos a rota inicial para teste do servidor usando a aplicação com uma requisição e uma resposta juntamente com uma porta específica ou padrão para o servidor rodar.

app.get('/',(req, res) => {

res.send('Servidor OK!');

});

const PORT = process.envPORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor online na porta ${PORT}.`);

});

Para que o código em server.js execute devemos adicioná-lo no arquivo package.json juntamente com o nodemon, então quando fizermos alterações no código, o servidor sempre irá iniciar automaticamente.

"scripts": {

"start": "nodemon server.js"

},

Com o comando **npm start** no terminal a seguinte mensagem deve aparecer: “Servidor online na porta 3000.” Caso ocorra os erros: **ReferenceError** algo pode estar escrito de maneira incorreta, alguma variável, e **TypeError** alguma função pode também estar escrita de maneira errada.

Agora depois de o servidor está rodando iniciaremos a conexão com o banco de dados na pasta config, arquivo inventory\_db.js com a importação da biblioteca mysql2, armazenando-a em uma variável, e o código para criar a conexão do servidor com o banco de dados criado anteriormente com os produtos já inseridos no MySQL, junto com o endereço do servidor do banco de dados, o nome e a senha do usuário responsável pelo banco e o nome deste.

const mysql = require('mysql2');

const inventory\_db = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASS,

database: process.env.DB\_NAME

});

Para iniciar a conexão, o seguinte código irá pegar a variável criada anteriormente e conectá-la apresentando duas respostas, um caso dê erro e outra de sucesso, e para podermos usá-la em outros arquivos exportamo-la.

inventory\_db.connect((err) => {

if (err) {

console.error('Erro ao conectar ao banco de dados:', err);

return;

}

console.log('Conectado ao banco de dados Inventory.')

});

module.exports = inventory\_db;

Voltando no arquivo server.js adicionaremos o código para importar e utilizar a conexão com o banco de dados quando o servidor iniciar.

const inventory\_db = require('./config/inventory\_db');

Inserindo novamente o comando **npm start** no terminal a seguinte mensagem deve aparecer: “Servidor online na porta 3000. Conectado ao banco de dados Inventory.” Caso ocorra os erros: **ReferenceError** algo pode estar escrito de maneira incorreta, alguma variável, **TypeError** alguma função pode também estar escrita de maneira errada, **code: 'ETIMEDOUT'** o endereço host está incorreto, **code: 'ER\_ACCESS\_DENIED\_ERROR'** o nome ou a senha do usuário está incorreta, e **code: 'ER\_BAD\_DB\_ERROR'** o banco de dados não foi encontrado, o nome está incorreto.

Falta as trasações e seus controllers e os teste no insomia

Referencias

06.08.2024

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

<https://kinsta.com/pt/base-de-conhecimento/npm-gerenciador-pacotes-node/>

<https://fabiojanio.medium.com/nodejs-express-cors-consign-autoload-bodyparser-e-compression-na-pr%C3%A1tica-fbfc12e46ff4>