


# API RESTful com Node.js, Express.js e Mongoose

CONTEÚDO DEV DICAS JAVASCRIPT TÉCNOLOGIA TUTORIAL

29 DE DEZEMBRO DE 2022

Aprenda a criar uma API RESTful com Node.js, Express.js e Mongoose seguindo este tutorial passo a passo.

Uma API RESTful (Application Programming Interface) é um conjunto de rotinas, protocolos e ferramentas para construir aplicações de software que expõem serviços a outras aplicações. Elas permitem que diferentes sistemas se comuniquem e troquem dados de forma organizada e padronizada.

# NodeJS

Node.js é uma plataforma de desenvolvimento de aplicações baseadas em JavaScript que permite o uso de JavaScript no lado do servidor. Ele é baseado em uma arquitetura de eventos assíncronos, o que o torna leve e eficiente para lidar com ações em tempo real.

# ExpressJS

Express.js é um framework para Node.js que facilita o desenvolvimento de aplicações web. Ele fornece uma série de recursos e funcionalidades para criar rotas, lidar com requisições e respostas, tratar erros e muito mais.

# Mongoose

Mongoose é um módulo para Node.js que fornece uma camada de abstração sobre o MongoDB, um banco de dados NoSQL orientado a documentos. Ele permite que você crie esquemas e modelos para os

# O que vamos aprender?

Neste tutorial, vamos mostrar os primeiros passos para criar uma API RESTful usando Node.js, Express.js e Mongoose. Vamos começar instalando e configurando o ambiente de desenvolvimento, depois vamos criar as rotas e os modelos para a nossa aplicação e, por fim, vamos testar a API usando o Postman.

# Pré-requisitos

Para seguir este tutorial, é preciso ter o seguinte software instalado em seu computador:

- Node.js: você pode baixar a última versão do site oficial (https://nodejs.org/) ou usar um gerenciador de pacotes, como o nvm (https://github.com/nvm-sh/nvm).
- MongoDB: você pode baixar a última versão do site oficial (https://www.mongodb.com/) ou usar um gerenciador de banco de dados, como o MongoDB Atlas (https://www.mongodb.com/cloud/atlas).
- Postman: é uma ferramenta para testar APIs que permite enviar requisições HTTP e ver a resposta do servidor. Você pode baixar o Postman na página de downloads (https://www.postman.com/downloads/).

# Criando o projeto

`npm init`. Você será perguntado algumas informações sobre o projeto, como o nome, a versão e a descrição. Você pode preencher esses campos ou pressionar Enter para usar os valores padrão. Quando terminar, um arquivo `package.json` será criado na pasta do projeto com as informações fornecidas.

Agora, vamos instalar as dependências do projeto. Para isso, use o comando **`npm install express mongoose`** para instalar o Express.js e o Mongoose. Esse comando irá criar uma pasta **`node_modules`** com os módulos instalados e adicionar as dependências ao arquivo **`package.json`**.

# Criando o servidor

Com o projeto criado e as dependências instaladas, vamos criar o arquivo principal da nossa aplicação. Vamos chamar esse arquivo de `server.js`. Nele, vamos importar o Express.js e o Mongoose e configurar o servidor.

No início do arquivo, adicione o seguinte código:

```
const express = require('express');const mongoose = require('mon
```

O código acima importa o **Express.js** e o **Mongoose** e cria uma instância do Express chamada app. Também definimos uma variável port com o número da porta que o servidor irá escutar.

Agora, vamos configurar o **Mongoose** para se conectar ao banco de dados. Para isso, adicione o seguinte código logo abaixo do código anterior:

```
mongoose.connect('mongodb://localhost:27017/minha-api', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'))
db.once('open', function() {
  console.log('Conexão com o banco de dados estabelecida com suc
});
```

O código acima usa a função `connect` do Mongoose para se conectar ao banco de dados. No primeiro parâmetro, passamos a URL de conexão do MongoDB. No segundo parâmetro, passamos algumas opções para configurar a conexão.

Depois, criamos uma variável `db` que representa a conexão com o banco de dados. Usamos o método `on` para registrar um ouvinte de eventos de erro e o método `once` para registrar um ouvinte de eventos de sucesso. Se a conexão for bem-sucedida, a mensagem "Conexão com o banco de dados estabelecida com sucesso!" será exibida no console.

Agora, vamos configurar o Express.js para receber requisições HTTP e retornar respostas. Adicione o seguinte código logo abaixo do código anterior:

```
  res.send('API funcionando corretamente!');
});

app.listen(port, () => {
  console.log(`Servidor iniciado na porta ${port}`);
});
```

O código acima habilita o middleware de suporte a JSON do Express. Isso permite que o servidor receba e envie dados no formato JSON. Depois, criamos uma rota `GET` para a raiz da aplicação que retorna a mensagem “API funcionando corretamente!”. Por fim, usamos o método `listen` do Express para iniciar o servidor na porta especificada.

Salve o arquivo `server.js` e execute o comando **`node server.js`** na linha de comando para iniciar o servidor. Se tudo estiver configurado corretamente, você deverá ver a mensagem “Servidor iniciado na porta 3000” no console.

# Testando a API

Agora que o servidor está rodando, vamos testar a API usando o Postman. Abra o Postman e crie uma nova solicitação `GET` para a URL `http://localhost:3000`. Clique em “Enviar” e verifique se a resposta é “API funcionando corretamente!”.

# Criando os modelos

Com o servidor configurado e funcionando, vamos criar os modelos para os dados da nossa aplicação. Os modelos são esquemas que

Eles são usados pelo Mongoose para realizar operações de CRUD e validar os dados antes de salvá-los no banco de dados.

Vamos criar um modelo para uma coleção de produtos. Crie um novo arquivo chamado `product.js` na pasta do projeto e adicione o seguinte código:

**product.js**

```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

name: {

type: String,

required: true

},

price: {

type:Number,

required: true

},

description: {

type: String

},

createdAt: {
```

```
}

});

const Product = mongoose.model('Product', productSchema);

module.exports = Product;
```

O código acima cria um esquema chamado `productSchema` usando o construtor `Schema` do Mongoose. Nele, definimos os campos `name`, `price` e `description` como strings e o campo `createdAt` como uma data. Também marcamos o campo `name` e o campo `price` como obrigatórios.

Depois, criamos um modelo chamado `Product` usando o método `model` do Mongoose e passando o esquema como parâmetro.

Por fim, exportamos o modelo para que possa ser usado em outros arquivos.

# Criando as rotas

Com o modelo criado, vamos criar as rotas da nossa API. As rotas são os endpoints da API que permitem acessar os recursos da aplicação. Vamos criar rotas para as operações de **CRUD** (Create, Read, Update e Delete) de produtos:

- `**GET /products**`: lista todos os produtos
- `**GET /products/:id**`: exibe um produto específico

- `**DELETE /products/:id**`: exclui um produto

Para criar as rotas, vamos adicionar o seguinte código no final do arquivo `server.js`:

**Server.js**

```
const Product = require('./product');

app.get('/products', (req, res) => {
Product.find((err, products) => {
if (err) return res.status(500).send(err);
res.send(products);
});
});

app.get('/products/:id', (req, res) => {
Product.findById(req.params.id, (err, product) => {
if (err) return res.status(500).send(err);
if (!product) return res.status(404).send('Produto não encontrad
res.send(product);
});
});

app.post('/products', (req, res) => {
const product = new Product(req.body);
product.save((err, newProduct) => {
if (err) return res.status(500).send(err);
res.send(newProduct);
});
});

app.put('/products/:id', (req, res) => {
Product.findByIdAndUpdate(req.params.id, req.body, { new: true }
if (err) return res.status(500).send(err);
if(!product) return res.status(404).send('Produto não encontrado
res.send(product);
});
```

```
Product.findByIdAndDelete(req.params.id, (err, product) => {
if (err) return res.status(500).send(err);
if (!product) return res.status(404).send('Produto não encontrad
res.send(product);
});
});
```

O código acima cria cinco rotas para a coleção de produtos. Cada rota usa os métodos do Mongoose para realizar as operações de CRUD. Por exemplo, a rota `GET /products` usa o método `find` para buscar todos os produtos no banco de dados e retorná-los na resposta. A rota `POST /products` usa o método `save` para criar um novo produto e salvar no banco de dados.

# Testando as Rotas

Agora que as rotas estão criadas, vamos testá-las usando o Postman. Crie uma nova solicitação `POST` para a URL `**http://localhost:3000/products**` e adicione um corpo JSON com os dados do produto. Por exemplo:

```
{
"name": "Produto 1",
"price": 9.99,
"description": "Descrição do produto 1"
}
```

Clique em “Enviar” e verifique se o produto foi criado com sucesso. Em seguida, crie uma nova solicitação `GET` para a URL

Você também pode testar as outras rotas da mesma forma. Por exemplo, para atualizar um produto, crie uma solicitação `PUT` para a URL `http://localhost:3000/products/<id do produto>` e adicione um corpo JSON com os novos dados.

# Código completo

**server.js**

```
const express = require('express');
const mongoose = require('mongoose');
const Product = require('./product');

const app = express();
const port = 3000;

mongoose.connect('mongodb://localhost:27017/minha-api', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'))
db.once('open', function() {
  console.log('Conexão com o banco de dados estabelecida com suc
});

app.use(express.json());

app.get('/', (req, res) => {
  res.send('API funcionando corretamente!');
});

app.get('/products', (req, res) => {
  Product.find((err, products) => {
    if (err) return res.status(500).send(err);
```

```
app.get('/products/:id', (req, res) => {
  Product.findById(req.params.id, (err, product) => {
    if (err) return res.status(500).send(err);
    if (!product) return res.status(404).send('Produto não encon
    res.send(product);
  });
});

app.post('/products', (req, res) => {
  const product = new Product(req.body);
  product.save((err, newProduct) => {
    if (err) return res.status(500).send(err);
    res.send(newProduct);
  });
});

app.put('/products/:id', (req, res) => {
  Product.findByIdAndUpdate(req.params.id, req.body, { new: true
    if (err) return res.status(500).send(err);
    if (!product) return res.status(404).send('Produto não encon
    res.send(product);
});
});

app.delete('/products/:id', (req, res) => {
Product.findByIdAndDelete(req.params.id, (err, product) => {
if (err) return res.status(500).send(err);
if (!product) return res.status(404).send('Produto não encontrad
res.send(product);
});
});

app.listen(port, () => {
console.log(Servidor iniciado na porta ${port});
});
```

Neste tutorial, aprendemos os primeiros passos para criar uma API RESTful usando Node.js, Express.js e Mongoose. Configuramos o ambiente de desenvolvimento, criamos o servidor, criamos os modelos e as rotas da nossa aplicação e testamos a API usando o Postman.

Existem muitas outras coisas que você pode fazer para melhorar e expandir a sua API. Algumas sugestões são:

- Adicionar autenticação e autorização para proteger as rotas;
- Criar mais modelos e rotas para outros recursos da aplicação;
- Usar o MongoDB Atlas ou outro serviço de banco de dados em nuvem em vez de um banco de dados local;
- Implementar paginação e filtragem de resultados para evitar carregar grandes quantidades de dados de uma vez;
- Adicionar validação de dados e tratamento de erros mais detalhado.

Espero que este tutorial tenha sido útil e que você tenha gostado. Se tiver dúvidas ou sugestões, deixe um comentário abaixo.

Até a próxima!

# Api # API RESTFul # Aprender # dica # ExpressJs # Iniciantes # MongoDB # Mongoose # NodeJS # Tutorial

POST ANTERIOR

**Manipulando banco de dados com SQL e MySQL**

**Introdução ao Docker: Criando e gerenciando aplicativos em contêineres**

## Deixe um comentário

O seu endereço de e-mail não será publicado. Campos obrigatórios são marcados com *

Nome *

E-mail *

Site

Adicionar comentário *

Salvar meus dados neste navegador para a próxima vez que eu comentar.

Eu aceito as políticas de privacidade

Publicar comentário

# Posts relacionados


### Meu Setup Minimalista

27 de novembro de 2023

### Guia rápido de como construir um site do zero a publicação

14 de novembro de 2023

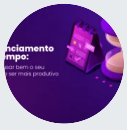

### Dominando o JavaScript: Dicas, Exemplos Práticos e Recursos Essenciais

13 de novembro de 2023

## Mais lidos

6 temas claros incríveis para VSCode


Aprenda os comandos básicos do MongoDB em 5min.



Gerenciamento de tempo: como usar bem o seu tempo e ser mais produtivo

Aplicativo: o que é, como funciona e para que serve?