Início › Back-end

4 anos atrás

Passo a passo para criar um CRUD com NodeJS do zero

Neste artigo pretendo guiar vocês na criação de um CRUD com Node JS! Essa é a base para muitos, se não todos, os programadores back-ends das empresas.

O conhecimento que você terá ao concluir o conteúdo deste artigo permitirá que você crie suas próprias API REST! Seja para criar um CRUD de casas para alugar, ou até mesmo de quartos disponíveis em um hostel.

Você pode adaptar esse conhecimento e, com os ajustes certos, aplicá-lo a qualquer situação.

Deixe a sua imaginação livre! Me envie sua ideia através dos comentários deste artigo ou mencione meu user nas redes sociais, vou adorar conhecer sua criação! 🥲

Conteúdo [ocultar]

Antes de iniciar o CRUD

Antes de partirmos para o código, farei uma breve introdução para que você entenda o que é Node JS, para que serve e como funciona.

As ferramentas necessárias para concluir essa API REST com Node JS são:

- NPM ou preferencialmente Yarn;
- Visual Studio Code, esta IDLE torna o desenvolvimento mais rápido;
- Node JS, verifique se precisa dar o Update no Node JS antes.

O que é NODE JS?

Muitos acreditam que Node JS é uma linguagem de programação, acontece que, geralmente, é referida como linguagem. No entanto, Node JS é uma plataforma para back-end que aceita código JavaScript. Isto é, Node JS é uma maneira de usar JavaScript no back-end.

Fato curioso sobre Node JS: ele foi construído em cima da V8, que é o Engine, ou seja, o motor por trás do Google Chrome. Isso torna o Node JS muito rápido, além de permitir que você reutilize todo conhecimento de JavaScript (adquirido para front-end) também no back-end.

Características do Node JS

Uma das principais características do Node é sua arquitetura, chamada de Event-loop. Totalmente baseada em eventos e, na maioria das vezes, rotas.

Call Stack é uma pilha de eventos, esses eventos podem ser uma função disparada pelo código, poi isso o event-loop fica monitorando para que toda vez que uma nova função é disparada, ele a executará.

Node é single-thread. Isso significa que ele executa em apenas uma thread do nosso processador.

Outra característica do Node é que ele tem uma arquitetura chamada Non-blocking I/O, que significa Output e Input não bloqueante. Isto quer dizer que se você fizer uma requisição de listagem de página, por exemplo, você pode retornar a listagem em partes, sem precisar listar tudo de uma única vez, e ainda assim continuar a comunicação entre back-end e front-end.

Diferente de outras linguagens de back-end como por exemplo PHP, que perde a comunicação quando a requisição é enviada em forma de JSON, XML ou HTML etc.

Com Node, o front e o back mantém uma comunicação aberta, permitindo que o front faça mais requisições ao back ininterruptamente. Um bom exemplo de uso são os Chats, que acontecem em real-time graças ao WebSockets.

Gerenciadores de Pacotes

Gerenciadores de pacotes, basicamente, são utilizados para instalar, remover e configurar dependências e bibliotecas de terceiros em nossos projetos.

Imagine que você queira implementar uma integração com meios de pagamento em seu projeto, você precisará fazer isto utilizando um dos mais populares package

manager que existem: NPM e Yarn.

NPM versus Yarn

Apesar de servirem para as mesmas finalidades, o Yarn tem algumas vantagens sobre o NPM!

Mesmo o NPM sendo mais velho, o Yarn se adiantou e implementou funcionalidades mais avançadas, como por exemplo o Yarn workspaces, utilizado em múltiplos projetos que geralmente contam com as mesmas dependências, permitindo compartilhamento de dependências, isso evita que o desenvolvedor fique sempre instalando uma por uma.

Até aqui você já entendeu o que é Node JS, então antes de sair codando, vamos rever alguns conceitos de API REST, que é o que vamos criar logo depois desta introdução.

API REST

Os benefícios deste modelo de API REST é que podemos servir múltiplos clientes com o mesmo back-end, ou seja, um único código fornecido para Web Mobile ou até mesmo uma API pública.

É importante entender o fluxo de requisição e resposta, não vou entrar em detalhes, mas basicamente acontece nesta sequência:

- Requisição é feita por um cliente;
- Resposta retornada através de uma estrutura de dados (ex: JSON);
- Cliente recebe a resposta e processa o resultado.

Estas respostas utilizam métodos HTTP, que são:

- **GET** http://minhaapi.com/*users* ➔ Buscar alguma informação no back-end
- **POST** http://minhaapi.com/*users* ➔ Criar alguma informação no back-end
- **PUT** http://minhaapi.com/*users*/1 ➔ Editar alguma informação no back-end

- **DELETE** http://minhaapi.com/*users*/1 ➔ Deletar alguma informação no back-end

Negrito ➔ Método HTTP
Itálico ➔ Recurso/Rota
Número ➔ Parâmetro

É importante também que você entenda sobre HTTP codes, que são os códigos HTTPs retornados de uma requisição, vejamos alguns exemplos mais comuns:

- 1xx: HTTP codes iniciados em 1 são informativos:
- 102: PROCESSING.

- 2xx: HTTP codes iniciados com 2 são de sucesso:
- 200: SUCCESS;
- 201: CREATED.

- 3xx: HTTP codes iniciados em 3 são de redirecionamento:
- 301: MOVED PERMANENTLY;
- 302: MOVED.

- 4xx: HTTP codes iniciados em 4 são de erros do cliente:
- 400: BAD REQUEST;
- 401: UNAUTHORIZED;
- 404: NOT FOUND.

- 5xx: HTTP codes iniciados em 5 são erros do servidor:
- 500: INTERNAL SERVER ERROR.

Vamos entender melhor sobre estes métodos, recursos/rotas e parâmetros na prática, aguenta só um pouquinho.

Benefícios de API REST

Os benefícios deste modelo de API REST é que podemos servir múltiplos clientes com o mesmo back-end, ou seja, um único código fornecido para Web Mobile ou até mesmo uma API pública.

Agora sim, até aqui você já sabe o mínimo necessário para começar a criar sua primeira API REST com CRUD em Node JS, então vamos para o código!

Você pode começar criando uma pasta para armazenar este código.

No terminal execute:

```
❯ mkdir nodegeek
```

Acesse a pasta via terminal:

```
❯ cd nodegeek
```

Ainda no terminal, execute:

```
❯ yarn init -y
```

Este comando simplesmente cria um arquivo chamado package.json.

Agora você pode acessar o arquivo via terminal executando o comando code (abrirá o VScode) na pasta raiz do projeto, execute:

```
❯ code
```

Agora com o arquivo package.json aberto no seu VScode, você terá algo parecido com isso:

```
{
"name": "nodegeek",
"version": "1.0.0",
"main": "index.js",
"license": "MIT"
}
```

Neste arquivo ficará armazenado a referência de todas as dependências que você instalar via NPM ou Yarn.

Abra o terminal do VScode, experimente usar as teclas de atalho CTRL + ‘, se isso não funcionar verifique quais sãos as teclas de atalho teclando CTRL + P, escreva “>Preferences: Open Keyboard Shortcuts”, e veja quais são as teclas de atalho para o seu VScode.

Com o terminal do VScode aberto, execute:

```
❯ yarn add express
```

Note que o seu arquivo package.json foi alterado, agora está assim:

```
{
"name": "nodegeek",
"version": "1.0.0",
"main": "index.js",
"license": "MIT",
"dependencies": {
"express": "^4.17.1"
}
}
```

Agora crie um arquivo e nomeie de index.js, este arquivo conterá todo nosso código.

Então é hora de importar o express para começar a utilizá-lo.

```
const express = require('express');
```
// importa o express

```
const server = express();
```
// cria uma variável chamada server que chama a função express

```
server.listen(3000);
```
// faz com que o servidor seja executado na porta 3000 do seu localhost:3000

Salve o código teclando CTRL + S, ou salve manualmente.

Agora vamos criar a primeira rota da aplicação, com o método GET.

```
const express = require('express');
```
// importa o express

```
const server = express();
```
// cria uma variável chamada server que chama a função express

```
server.get('/teste', () => {
console.log('teste');
})
```
// Cria a rota /teste com o método GET, o console.log retornará no terminal 'teste' caso tenha executado com sucesso.

```
server.listen(3000);
```
// faz com que o servidor seja executado na porta 3000 do seu localhost:3000

Salve o código teclando CTRL + S, ou manualmente, e agora para testar se a rota está funcionando, execute no terminal:

```
› node index.js
```

Isso iniciará o servidor na porta 3000 do seu localhost.

Perceba que o terminal ficará executando sem retornar nada, então abra seu navegador, pode ser o Google e acesse: http://localhost:3000/geeks

Note que navegador fica carregando, porém não retorna nada, então volte ao seu terminal e perceba que o terminal respondeu 'teste' conforme solicitado no console.log('teste'); que colocamos dentro da função.

Se você chegou até aqui, é sinal que sua primeira rota foi criada com sucesso, então continuemos...

Adicionar os parâmetros

Agora vamos adicionar dois parâmetros na nossa função, dessa forma:

```
server.get('/geeks, (req, res) => {
console.log('teste');
})
```

req ➔ representa todos os dados da requisição.

res ➔ todas as informações necessárias para informar uma resposta para o front-end.

Então vamos testar, exclua o console.log('teste'); e adicione **return res.json({ message: 'Hello world' });**

Desta forma, retornaremos para o front-end "Hello world".

Salve o código novamente, pare o servidor teclando CTRL + C, inicie o servidor de novo e acesse a rota. Perceba o que navegador retorna um json com o texto Hello world na tela.

o Código deve estar assim:

```
const express = require('express');
const server = express();

server.get('/geeks, (

req, res) => {
return res.json( { message: 'Hello world' });
})

server.listen(3000);
```

Instale o Insomnia

Antes de continuar e criar rotas do tipo POST, PUT e DELETE, vamos precisar instalar um software chamado Insomnia.O Insomnia não funciona em sistema Windows de 32bits. Se for preciso, você precisará baixar e usar o Postman.

Os dois funcionam igual, mas vamos usar o Insomnia porque ele é mais bonito! 😀

Depois de fazer o download do Insomnia, abra-o e note que do lado esquerdo ficará listado as requisições.

Clique no ícone de +(plus) em seguida “New Request” para criar uma nova requisição.

Dê um nome para ela, exemplo: “Listagem de geeks” e em seguida clique em CREATE.

No campo superior, adicione o Link com a seguinte rota, aquela criada anteriormente: http://localhost:3000/geeks

Clique em Send, agora você recebeu a mesma resposta que obteve no navegador, a mensagem “Hello world”. Perceba também que o HTTP code é 200 de SUCCESS.

Legal! Até aqui nós já criamos e testamos nossa rota no browser e também em um software.

Então, antes de criarmos outras rotas, vamos instalar uma dependência que manterá nosso servidor rodando e, desta forma, não vamos precisar encerrar e reiniciar o servidor toda vez que alterarmos o código.

Pare o servidor teclando CTRL + C.

Abra o terminal e execute:

```
❯ yarn add nodemon -D
```

nodemon reiniciará sozinho o servidor toda vez que você salvar o código

Agora abra o arquivo package.json e perceba que o nodemon foi adicionado como dependência de desenvolvimento. Isto porque quando a aplicação for para produção, não precisaremos mais ficar monitorando alterações no código.

Para iniciar o servidor com nodemon precisaremos criar uma propriedade chamada scripts para iniciar o servidor.

```
{                                   // yarn dev para iniciar o servidor
"name": "nodegeek",
"version": "1.0.0",
"main": "index.js",
"license": "MIT",
"scripts": {
"dev": "nodemon index.js"

},
"dependencies": {
"express": "^4.17.1"
},
"devDependencies": {
"nodemon": "^2.0.1"
}
}
```

Agora para iniciar o servidor com nodemon, abra o terminal e rode:

```
› yarn dev
```

e deverá ter uma resposta mais ou menos assim:

```
yarn run v1.21.1
warning ../package.json: No license field
$ nodemon index.js
[nodemon] 2.0.1
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js, mjs, json
[nodemon] starting 'node index.js'
```

Agora vamos criar o CRUD

Vamos começar criando uma rota para listar todos os geeks.

Agora precisamos criar um rota com o método GET para listar todos os Geeks na rota /geeks que criamos.

No código adicione o seguinte trecho:

```
server.get('/geeks', (req, res) => {  // rota para listar todos os geeks
return res.json(geeks);
})
```

Agora vamos criar uma rota para criar Geeks e logo depois vou mostrar como seu código deve estar até aqui.

Para criar, você deve lembrar-se, usaremos o método POST, vamos lá:

```
server.post('/geeks', checkUserExists, (req, res) => {
const { name } = req.body; // assim esperamos buscar o name
informado dentro do body da requisição
geeks.push(name);
return res.json(geeks); // retorna a informação da variável geeks
})
```

Agora seu código deve estar mais ou menos assim:

```
const express = require('express');                                    //
const server = express();
server.use(express.json()); // faz com que o express entenda JSON
const geeks = [];
```

As informações ficaram armazenadas dentro deste array []

```
server.get('/geeks', (req, res) => { // rota para listar todos os geeks
return res.json(geeks);
})

server.get('/geeks/:index', checkUserInArray, (req, res) => {
return res.json(req.user);
})
```

```
server.post('/geeks', (req, res) => { ; // assim esperamos buscar o name
const { name } = req.body
```

informado dentro do body da requisição

```
geeks.push(name);

return res.json(geeks);  // retorna a informação da variável geeks

})

server.listen(3000);
```

Se seu código estiver igual ao meu, como apresentado acima, então você já deve estar conseguindo Listar e Criar Geeks, legal né? 😀

Antes de testar todas as rotas, vamos agora incluir as ações de Editar e Deletar, usando os métodos PUT e DELETE.

Para adicionar o método PUT, vamos adicionar o seguinte trecho de código:

```
server.put('/geeks/:index', (req, res) => {
const { index } = req.params; // recupera o index com os dados
const { name } = req.body;

geeks[index] = name;  // sobrepõe o index obtido na rota de acordo com o
```

novo valor

```
return res.json(geeks); ; // retorna novamente os geeks atualizados após o
})
```

update

Agora vamos excluir, método DELETE, vamos adicionar o seguinte trecho de código:

```
server.delete('/geeks/:index', (req, res) => {  // recupera o index com
const { index } = req.params;
```

os dados

```
geeks.splice(index, 1);  // percorre o vetor até o index selecionado e deleta
```

uma posição no array

```
return res.send();  // retorna os dados após exclusão
});
```

CRUD realizado com sucesso!

WOW! Parabéns! Até agora você usou todos os métodos que vimos no começo deste artigo.

Mas antes de partir para os middlewares, que será o bônus deste artigo, vamos voltar ao Insomnia para testar todas as rotas? Vamos lá!

Vamos agora listar os geeks de acordo com sua posição no array, lembrando que em JavaScript a contagem começa em 0.

Com o botão direito, clique sobre a request existente e duplique-a.

Renomeie a duplicada para Listar todos os Geeks, e a anterior nomeie Listar um Geek.

Listar todos os geeks – irá listar todos os Geeks no array.
Rota: http://localhost:3000/geeks

Listagem de um Geek – irá listar um Geek selecionado por você de acordo com sua posição no array.
Rota: http://localhost:3000/geeks/0

Note que estou buscando o Geek na posição 0 do array.

Agora vamos criar uma request para criar Geeks pelo Insomnia.

Crie uma nova requisição com o método POST, com o corpo em JSON.
Na rota, adicione: http://localhost:3000/geeks

Agora, no corpo do JSON, adicione:

```
{
"name": "Luke"
}
```

Isso criará um novo Geek com o nome Luke

Em seguida, clique em Send.

Se tudo der certo, você receberá um HTTP code 200, confirmando a criação do Geek “Luke”

Legal, né? Agora vamos editar o Geek “Luke”.

Com o método PUT, vamos duplicar a requisição Criar Geek.

Agora, mude o método de POST para PUT, e adicione o parâmetro /4 para editar o Geek que ocupa a posição 4 no array, correspondente ao “Luke” que criamos anteriormente.

A rota deve ficar assim: http://localhost:3000/geeks/4

E o corpo do JSON, adicione:

```
{
"name": "Luke Skywalker"
}
```

Agora, liste novamente os Geeks e perceba que o Geek na posição 4 do array foi alterado de Luke para Luke Skywalker. Demais, né?

Agora, vamos excluir um Geek da listagem, usando o método DELETE.

Crie uma nova requisição com o método DELETE, e no corpo selecione a opção “No body”, agora na rota adicione:
Rota: http://localhost:3000/geeks/4
Isso excluirá o Geek que ocupa posição 4 no array, isto é, Luke Skywalker.

Se você listar novamente os Geeks, perceberá que Luke Skywalker não será listado. Se isso acontecer, então você terá feito tudo corretamente.

Vamos ao bônus: middlewares

Os middlewares são basicamente uma função que recebe os parâmetros, req, res, entre outros, e executa uma função na aplicação, manipulando os dados da requisição de alguma forma.

Podemos criar middlewares locais e middlewares globais, vejamos exemplos.

Middlewares globais

```
server.use((req, res, next) => { // server.use cria o middleware global
console.time('Request') ; // marca o início da requisição
console.log(Método: ${req.method}; URL: ${req.url};) ; // retorna qual o método e url foi chamada

next(); // função que chama as próximas ações

console.log('Finalizou') ; // será chamado após a requisição ser concluída

console.timeEnd('Request'); // marca o fim da requisição
});
```

Este middleware é GLOBAL, isto é, será chamado em todas as rotas.

Middlewares locais

Agora, com middlewares LOCAIS é um pouco diferente. Digamos que queremos criar dois middlewares, um para checar se um index já existe no array, e outro para checar se a propriedade name foi passada corretamente.

O código deve ser assim:

```
function checkGeekExists(req, res, next) {                    //
if (!req.body.name) {
return res.status(400).json({ error: 'geek name is required' });
```

middleware local que irá checar se a propriedade name foi informada corretamente,
// caso negativo, irá retornar um erro 400 – BAD REQUEST

```
} // se o nome for informado corretamente, a função next()
return next();
```

chama as próximas ações

```
} //
function checkGeekInArray(req, res, next) {
const geek = geeks[req.params.index];
if (!geek) {
return res.status(400).json({ error: 'geek does not exists' });
}
```

checa se o Geek existe no array, caso negativo informa que o index não existe no array

```
req.geek = geek;

return next();
}

server.post('/geeks', checkGeekExists, (req, res) => { // assim
const { name } = req.body;
```

esperamos buscar se a propriedade name foi informada corretamente

```
geeks.push(name); // retorna a informação da variável geeks
return res.json(geeks);
```

})

Podemos passar mais de um middleware por vez, desta forma:

```
server.put('/geeks/:index', checkGeekInArray, checkGeekExists,
(req, res) => {
const { index } = req.params;
```

// recupera o index com os dados

```
const { name } = req.body; // sobrepõe/edita o index obtido na rota de
geeks[index] = name;
```

acordo com o novo valor

```
return res.json(geeks); // retorna novamente os geeks atualizados após o
});
```

update

Agora se você quiser testar para confirmar o funcionamento, volte ao Insomnia. Abra a requisição de Criar Geek e altere a propriedade NAME, experimente isso:

```
{
"name1": "Princesa Leia"
}
```

Perceba que “name1” está escrito errado. Clique em Send.

Se tudo der certo, você receberá um HTTP 400 – BAD REQUEST com a seguinte mensagem:

```
{
"error": "geek name is required"
}
```

Agora imagine que queremos listar um Geek inexistente no array. Abra a requisição de Listagem de um Geek. Por exemplo o array de posição 8, inexistente.

Rota: http://localhost:3000/geeks/8

Clique em Send. Se tudo der certo, você receberá um HTTP – 400 BAD REQUEST, com a mensagem:

```
{
"error": "geek does not exists"
}
```

WOW! Chegamos ao fim. Isso mesmo! Se você chegou até aqui é sinal de que você conseguiu criar seu primeiro CRUD em Node JS. Sensacional, né?

O código para correção está no meu GitHub!

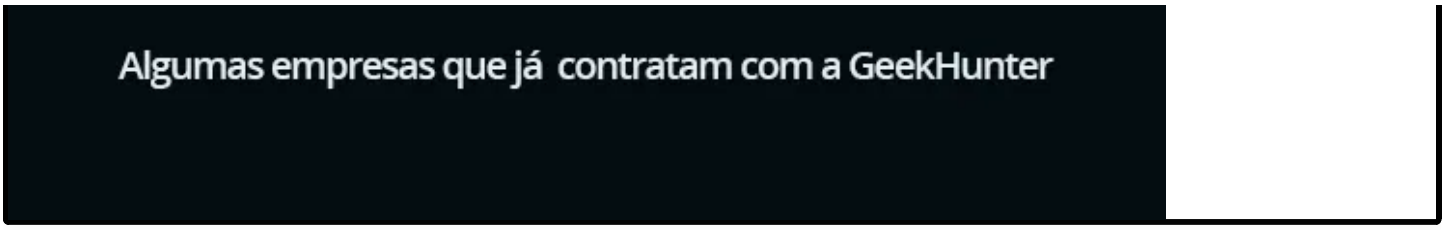
Espero que tenha gostado e até a próxima!

Brendon Guedes

Apaixonado por programação e tecnologias inovadoras. Entusiasta de *fintechs* e de qualquer coisa que facilite a vida. Trabalhando com javascript, react, react native e node.js







Categorias

Carreira de programador

Back-end

Full stack

Front-end

Big Data

Mobile

DevOps

Gestão de TI

Scrum Master

Leituras Recomendadas

BACK-END FULL STACK

Por que diabos usar o Node.js? Uma justificativa passo a passo

BACK-END FULL STACK

Sua API não é RESTful: Entenda por quê

BACK-END

Automatizando Testes com Python, Selenium e Behave

Quer receber conteúdos incríveis como esses?

Assine nossa newsletter para ficar por dentro de todas as novidades do universo de TI e carreiras tech.

Nome *

Email *

Inscreva-se