E-BOOK
DO ESTUDANTE

CRUD

Operando!

RESILIA

SUMÁRIO

CRUD

Operando!

Neste *e-book*, mostraremos como criar um CRUD básico utilizando o *Express*. O CRUD é uma **abordagem fundamental utilizada diariamente em aplicativos e *sites* para a criação de aplicativos que permitem ao usuário criar, ler, atualizar e excluir informações**.

Essas operações são *Create*, *Read*, *Update* e *Delete*, e elas nos permitem interagir entre dados que estão armazenados no banco de dados e que são disponibilizados por uma API.

Quando enviamos uma mensagem, estamos criando um novo dado, e para isso realizamos o *Create*. Quando lemos as mensagens enviadas anteriormente, estamos lendo um dado e utilizando o *Read*. Quando atualizamos nossos *status*, utilizamos o *Update* e, quando deletamos uma mensagem, utilizamos o *Delete*.

O *Express* é uma estrutura *web* para Node.js que torna fácil criar rotas e lidar com requisições HTTP. Ou seja, a criação de um CRUD (*Create*, *Read*, *Update*, *Delete*) é uma tarefa comum no desenvolvimento de aplicações *web*. Considerando isso, o objetivo deste *e-book* é explicar como usar essas ferramentas e mostrar os passos a serem seguidos para criar um CRUD básico. Vamos lá?

CONTEXTUALIZANDO

CRUD é a sigla para as operações básicas de banco de dados: *Create*, *Read*, *Update* e *Delete*. Essas operações são as mais comuns em aplicativos *web* e *mobile* que trabalham com banco de dados, e cada letra tem um significado e indica uma operação:

- *Create* (Criar) é a operação usada para adicionar novos registros ao banco de dados.
- *Read* (Ler) é a operação usada para ler ou buscar registros existentes no banco de dados.
- *Update* (Atualizar) é a operação usada para atualizar os dados de um registro existente no banco de dados.
- *Delete* (Excluir) é a operação usada para remover registros do banco de dados.

Essas operações são fundamentais para a maioria dos aplicativos e sistemas *web*, pois permitem ao usuário criar, ler, atualizar e excluir informações. Como exemplo, vamos considerar um aplicativo de *e-commerce*: nós podemos adicionar produtos ao nosso carrinho e, para isso, utilizamos a operação *Create* (C). Já para ver os produtos disponíveis para compra, utilizamos a operação *Read* (R). Quando atualizamos a quantidade de produtos em nosso carrinho, utilizamos a operação *Update* (U). E quando removemos produtos do carrinho, utilizamos a operação Delete (D).

Após saber a contextualização dessas operações, vamos aprender a criar o nosso próprio CRUD?

CRUD BÁSICO

Para criar um CRUD em Javascript, vamos utilizar o *Express*, que é uma estrutura *web* para Node.js que irá tornar fácil a criação das rotas e o trabalho com as requisições HTTP.

Para utilizar o *Express*, precisamos criar uma nova pasta em nosso computador, onde iremos desenvolver nosso projeto. Vamos criar a pasta de nome "crudBasico" e, dentro dela, vamos abrir um terminal para instalar o *Express* usando o NPM. Para isso, precisamos executar o comando abaixo:

```
npm install express
```

Após instalar o *Express* em nossa pasta, que agora se tornou o nosso projeto, vamos então começar a desenvolver e criar nosso primeiro CRUD. Para isso, primeiramente é necessário importar o *Express*:

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

Note que criamos uma constante de nome "express" que estará requerendo a biblioteca instalada de nome "express" através do código "require('express')".

Depois de importado, podemos passar a utilizá-lo através da nossa constante de nome "express" e precisamos então iniciar a instância da nossa API, ou seja: agora precisamos executar o *Express* para que ele crie um novo serviço de API a fim de que possamos configurar e montar as rotas, e consecutivamente criar os métodos do CRUD. Para isso, vamos realizar o seguinte código:

```
const express = require('express')
const app = express()
```

Note que temos agora mais uma constante, de nome “app”, que está recebendo a execução do Express, ou seja, a constante de nome “app” está contendo toda a nossa instância, todo o nosso serviço de API que utilizaremos para criar as nossas rotas.

Em seguida, precisamos utilizar os métodos HTTP (*Hypertext Transfer Protocol*) para criar nossos caminhos, pois eles indicam a ação desejada a ser realizada em uma determinada rota de uma API. Os métodos mais comuns são:

- **GET:** é usado para obter informações do servidor. Ele é usado para ler dados de uma rota específica.
- **POST:** é usado para enviar informações para o servidor. Ele é usado para criar novos registros em uma rota específica.
- **PUT:** é usado para atualizar informações no servidor. Ele é usado para atualizar dados em uma rota específica.
- **DELETE**: é usado para excluir informações do servidor. Ele é usado para excluir dados de uma rota específica.

Cada método tem sua funcionalidade e é usado de acordo com a necessidade de cada operação. Para montar uma rota usando o método GET, que é simbolizado como o *Read* dentro do CRUD, vamos realizar o seguinte código:

```
const express = require('express')
const app = express()

app.get('/items', (req, res) => {
  res.send({ titulo: ”Notebook”, valor: 2.000 })
})
```

Note a chamada da nossa constante app. Com ela, estamos dizendo que iremos criar uma nova rota com o método .get, utilizando o caminho simbolizado por “/items”. Isso quer dizer que, quando o usuário acessar o caminho “/items” dentro da nossa API, ele estará acessando esse método que acabamos de criar.

Logo após, temos uma *arrow function* com dois parâmetros: o req (A requisição) e o res (A resposta). Esses dois métodos nos permitem acessar os dados e as informações vindos da requisição através do req e enviar uma resposta, seja um dado, um arquivo ou somente um texto. Conseguimos enviar essa resposta usando o res.

Dentro da *arrow function* fica todo o código que nós queremos que o usuário possa ler ao acessar essa rota. No nosso caso, estamos enviando um item de título "*Notebook*" e o valor de "2.000", ou seja: quando o usuário acessar o caminho "/items" dentro da nossa API, ele irá receber o dado "`{ titulo: "Notebook", valor: 2.000 }`".

Agora que já configuramos nosso GET, precisamos então configurar nossas outras rotas, as quais vão seguir o mesmo conceito e a mesma sintaxe da anterior, como podemos ver abaixo:

```
const express = require("express");
const app = express();
app.post("/items", (req, res) => {
  res.send("Item criado");
});
app.get("/items", (req, res) => {
  res.send({ titulo: "Notebook", valor: 2.000 });
});
app.put("/items/:id", (req, res) => {
  res.send("Item Atualizado");
});
app.delete("/items/:id", (req, res) => {
  res.send("Item Deletado");
});
```

Note que temos agora a rota "/items" com os métodos *post (Create)*, *get (Read)*, *put (Update)*, *delete (Delete)*. Veja que todos seguem a mesma estrutura — todos contêm o caminho e a *arrow function* (A ação) — e que todos estão enviando uma resposta para o usuário.

Temos também um "/:id" dentro dos métodos *put* e *delete*, e o nome disso é *parâmetro de rota*. Trata-se de um dado que é passado pelo caminho. Quando o usuário for acessar a rota, ele precisará passar esse parâmetro. No nosso caso, queremos receber o id do item para que possamos atualizá-lo ou deletá-lo. Para pegar os valores passados pelo parâmetro de rota, é preciso configurar um *middleware*.

Glossário

Middleware: É um conjunto de funções ou componentes que são executados entre a requisição do usuário e a resposta do servidor (API). Eles são usados para realizar tarefas específicas, tais como autenticação, validação, manipulação de dados, manipulação de erros etc.

Um *middleware* pode ser entendido como uma "ponte" entre a requisição e a resposta, um meio onde as informações são processadas antes de seguirem para o próximo passo. Ele é geralmente escrito em uma linguagem de programação específica, como Node.js, e é usado para processar as requisições e respostas HTTP antes que elas sejam encaminhadas para o controlador ou para as rotas, permitindo, assim, que nós possamos pegar a requisição feita para a API, que vem em formato de texto simples (*plain text*), e transformar esse dado em JSON. Isso é necessário para acessar as informações e pegar o dado que desejamos dentro das rotas. Dessa forma, para transformar todos os dados da requisição em JSON, usamos o seguinte código:

```
app.use(express.json());
```

Note que usamos o “app.use” para pegar todas as requisições, ou seja, todas as vezes que a API for utilizada, ela irá pegar a requisição e irá transformá-la em estrutura JSON, permitindo-nos manipular os dados vindos da requisição nas rotas.

Vamos dar uma olhada em como nosso código ficou depois de acessar os dados vindos da requisição:

```
const express = require("express");
const app = express();
app.post("/items", (req, res) => {
  res.send("O item: " + req.body.titulo + "foi criado com
sucesso!");
});
app.get("/items", (req, res) => {
  res.send({ titulo: "Notebook", valor: 2.000 });
});
app.put("/items/:id", (req, res) => {
  res.send("O item de id: " + req.params.id + "Foi atualizado!!");
});
app.delete("/items/:id", (req, res) => {
  res.send("O item de id: " + req.params.id + "Foi deletado!!");
});
```

Note que agora, dentro do post, nós estamos acessando o “*body*”, que é o conteúdo da requisição, e estamos pegando o título do produto que foi enviado. Se o usuário enviasse o seguinte produto, por exemplo:

```
“{ titulo: ”Monitor”, valor: 1.700 }”
```

A mensagem retornada seria “O item Monitor foi criado com sucesso!”, indicando que ele pegou o "título" do produto que foi passado dentro do conteúdo da requisição. Se olharmos o put e o delete, nós estamos acessando o “params” dentro da request e depois acessando o ID (nosso parâmetro de rota), ou seja, nós estamos pegando o id que foi passado pela rota. Assim, se o usuário entrasse no caminho a seguir:

```
/items/1
```

A mensagem devolvida seria “Item de id: 1 Foi atualizado!”. Já se fosse no método delete, a mensagem devolvida seria “Item de id: 1 Foi deletado!!”.

Note como nós conseguimos pegar os dados que o usuário nos passou, essa é a chave para que possamos desenvolver uma API, pegar os dados passados e os manipular é a chave de uma comunicação limpa entre a API e o front-end (O usuário).

Até aqui, aprendemos a criar um CRUD básico e como podemos pegar os dados passados pelo usuário através do *body* e de parâmetros de rotas. Essas informações são bastante importantes para que possamos construir uma API estruturada e funcional.

Para refletir

- Quais são os métodos HTTP mais comuns? O que cada método faz?
- O que é o CRUD?
- Enquanto usuários, quando usamos as operações do CRUD?
- Qual foi o último *site* ou aplicativo que você utilizou com as quatro operações que compõem o CRUD?

AVANÇANDO COM O CRUD

Até o momento nós somente estávamos programando o nosso CRUD, porém um processo crucial dentro do desenvolvimento de APIs é sempre termos uma aplicação para podermos ir testando e verificando possíveis erros ou bugs. Para isso, utilizamos a aplicação Insomnia.

O Insomnia é uma ferramenta de teste de API popular que permite enviar requisições HTTP para um servidor e analisar as respostas. Ele é uma ótima opção para testar um CRUD construído com Express e Node.js, pois nos permite enviar facilmente requisições usando os métodos HTTP (GET, POST, PUT, DELETE), além de permitir gerenciar parâmetros e cabeçalhos.

Para utilizar o Insomnia e poder testar o nosso CRUD, primeiro precisamos baixar e instalar o aplicativo disponível no *link* abaixo:

Download - Insomnia

Após instalado, precisamos colocar nossa API para funcionar em alguma porta de nossa máquina. Um computador tem diversas portas, ou serviços, e cada uma dessas portas transmite um dado. A porta do *wi-fi*, por exemplo, transmite os nossos dados de conexões, como IP, *gateway*, entre outros.

Podemos entender o funcionamento dessas portas imaginando uma árvore: o tronco dela seria a nossa máquina, enquanto os galhos e as ramificações seriam as portas. Cada galho e cada raiz teriam uma funcionalidade específica, ou seja, totalmente diferente uma da outra. É assim que funciona nosso computador, pois existem diversas portas e cada porta disponibiliza um serviço para que possamos utilizar.

Dentro então do nosso app.js, que atualmente está desta forma:

```
const express = require("express");
const app = express();
app.use(express.json());
app.post("/items", (req, res) => {
  res.send("O item: " + req.body.titulo + "foi criado com sucesso!");
});
app.get("/items", (req, res) => {
  res.send({ titulo: "Notebook", valor: 2.0 });
});
app.put("/items/:id", (req, res) => {
  res.send("O item de id: " + req.params.id + "Foi atualizado!!");
});
app.delete("/items/:id", (req, res) => {
  res.send("O item de id: " + req.params.id + "Foi deletado!!");
});
```

Vamos agora adicionar um ouvinte, ou seja, um recurso que irá permitir que nossa API fique escutando uma porta específica em nosso computador, permitindo, assim, o acesso através do localhost.

Glossário

Localhost: simboliza o IP do computador em que estamos como um host local, todos os serviços serão acessados através do IP do computador na rede ou utilizando o localhost.

Para isso, adicionamos após o Delete o seguinte código:

```
app.listen(3000, () => {
  console.log('Server started on port 3000')
})
```

Note que estamos chamando a nossa constante app que contém toda a nossa instância da

API para que possamos utilizar o método *listen*, que ficará ouvindo uma porta. No exemplo, trata-se da porta 3000, que foi passada como argumento ao método.

Após a numeração da porta, temos uma *arrow function* que será executada toda vez que nossa API estiver ouvindo a porta 3000.

Agora basta salvar o arquivo e executá-lo através do Node, para que finalmente a nossa API funcione. Por isso, vamos então executar o seguinte comando em nosso terminal:

```
node ./app.js
```

Após executar o comando, veremos a seguinte mensagem no console:

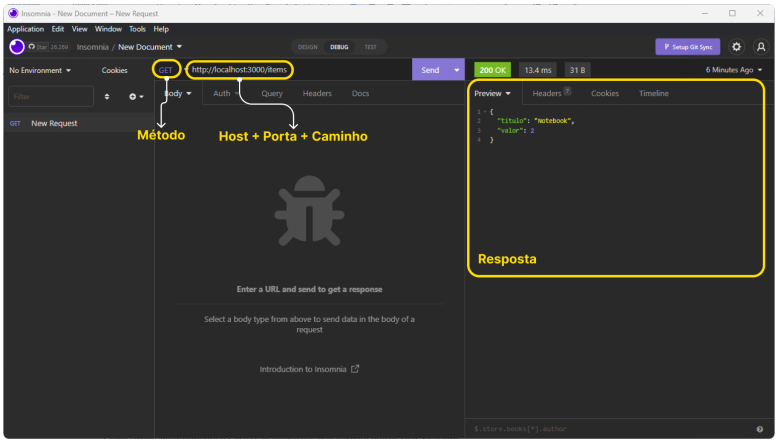
“`Server started on port 3000`”

Sabemos então que a nossa API já está executando e ouvindo as requisições que forem feitas para o link:

`http://localhost:3000`

Em seguida, vamos iniciar o Insomnia e criar uma nova requisição. Após criá-la, precisamos selecionar o método HTTP que desejarmos. Como exemplo, utilizaremos o método GET.

Após selecionado o método, inserimos a URL da rota correspondente que, em nosso caso, será o “http://localhost:3000/items”, ou seja, um *host* (servidor) interno da nossa máquina, e dentro dele acessaremos o caminho “/items” que definimos em nossa API. Realizamos, assim, a requisição. Note que recebemos uma resposta:



Este dado que recebemos é o dado que configuramos dentro do método GET da nossa API:

```
app.get("/items", (req, res) => {
  res.send({ titulo: "Notebook", valor: 2.0 });
});
```

Ou seja, quando nós realizamos a requisição no caminho “/items” com o método GET, foi acionado o nosso método GET dentro da API que já havíamos configurado anteriormente, e ele nos enviou uma resposta que foi o dado que colocamos dentro do método “*send*”. Isso permitiu comunicação com a API através de requisições, e é assim que as aplicações *web* funcionam: estão sempre realizando requisições e, portanto, recebendo e enviando dados

Atividade: Traduzindo

Nesta atividade, você terá que testar todas as rotas da nossa API que criamos anteriormente no caminho “/itens”.

Lembre-se de que, em algumas requisições, será necessário passar dados no *body* da requisição e em parâmetros. Para isso, utilize os links disponíveis na seção “Para ir além”.

Após esta visão geral sobre APIs, pudemos conhecer os métodos do CRUD e entender como podemos utilizá-los nas nossas aplicações. Esse é um assunto muito importante para quem trabalha com desenvolvimento *web*. Por isso, busque se aprofundar nesse assunto e pratique bastante, pois será um diferencial para sua carreira como dev.

Para refletir

- O que é uma porta dentro do nosso computador?
- Qual é a finalidade do Insomnia?
- Por que o CRUD é tão importante para a web?

Para ir além

- Como vimos, CRUD são operações que permitem aos *sites* ler, criar, atualizar e deletar dados de uma API. É por meio dessas operações que as redes sociais, os *sites* de *e-commerce*, entre outros tantos *sites* que conhecemos conseguem disponibilizar uma melhor experiência para seus usuários. Para se aprofundar nos conceitos vistos neste *e-book* sobre a criação de um CRUD básico usando o Node.js com o Express, acesse os *sites* a seguir:

  <https://vsvaibhav2016.medium.com/create-crud-application-in-express-js-9b88a5a94299>

  <https://zellwk.com/blog/crud-express-mongodb>

  <https://www.freecodecamp.org/news/building-a-simple-crud-application-with-express-and-mongodb-63f80f3eb1cd>

  <https://ichi.pro/pt/crie-uma-api-crud-rest-com-node-e-express-js-280895381233712>

  <https://medium.com/baixada-nerd/criando-um-crud-completo-com-nodejs-express-e-mongodb-parte-3-3-b243d14a403c>

NUVEM DE PALAVRAS