



Mundo Tech

# PROGRAMAÇÃO BACK-END MONGODB

# SUMÁRIO

|  |    |
|--|----|
| Mas, antes de começar.....                                 | 3  |
| <br>   |    |
| MongoDB .....  | 3  |
| Instalação e configuração do SGBD MongoDB na nuvem .....   | 3  |
| Conhecendo o MongoDB Atlas.....                            | 4  |
| Pacote Mongoose.....                                       | 11 |
| Como conectar o MongoDB ao Node.js usando o Mongoose ..... | 12 |
| Criação de um esquema utilizando o Mongoose .....          | 15 |
| Estilo arquitetural REST .....                             | 16 |
| Fundamentos da API REST .....                              | 18 |
| GET.....   | 19 |
| POST .....   | 20 |
| PUT .....  | 21 |
| DELETE .....   | 21 |
| Definição de URI .....                                     | 22 |
| Teste de API utilizando a ferramenta Postman.....          | 23 |
| <br>   |    |
| Referências.....   | 25 |



## MAS, ANTES DE COMEÇAR...

Sua equipe já criou o esquema de banco de dados, utilizando o Mongoose, que permitirá a validação dos campos relacionados à coleção de usuários, cujos dados serão persistidos no banco de dados MongoDB na nuvem. Para conferir esses arquivos você deve retornar ao moodle e baixar a pasta: **api-nodejs\_incompleto** com os arquivos iniciais.

## MONGODB

O MongoDB foi desenvolvido para unir características dos bancos de dados relacionais e NoSQL. Como resultado dessa mistura de abordagens, o MongoDB apresenta ganhos de agilidade com o uso de esquemas flexíveis e facilita a escalabilidade horizontal, recursos pouco explorados por outros SGBDs (Sistemas Gerenciadores de Banco de Dados), como MySQL e PostgreSQL. O MongoDB Atlas, que você conhecerá neste módulo, é um serviço global de banco de dados em nuvem criado e executado pela equipe por trás do MongoDB. Na sequência, compartilharemos informações sobre a tecnologia REST (*Representational State Transfer*), com o objetivo de fornecer uma compreensão básica dos principais fundamentos de API (*Application Programming Interface*), REST e protocolo HTTP. Por fim, você verá mais detalhes sobre a ferramenta Postman, usada para desenvolver, testar, compartilhar e documentar APIs. Inicie agora seus estudos! Vamos para mais uma jornada de aprendizados na área de desenvolvimento de sistemas!

## Instalação e configuração do SGBD MongoDB na nuvem

Existem vários bancos de dados disponíveis no mercado, dos quais os mais populares entre os desenvolvedores são MySQL e MongoDB. Nesta seção, iremos focar no MongoDB. O MongoDB é um banco de dados relativamente recente. Sua primeira versão foi lançada em 2009 e seu uso vem se expandindo muito desde então. O MongoDB é um banco de dados NoSQL de código aberto que trabalha com o modelo de dados orientado a documentos. O uso do MongoDB tem crescido, o que levou a uma expansão conjunta da linguagem JavaScript, utilizada por esse SGBD (MONGODB, 2020).

Como você já identificou, o Node.js é o *framework* JavaScript mais popular quando se trata de desenvolvimento de aplicativos de alta velocidade. Os profissionais do Node.js geralmente optam por um banco de dados NoSQL que pode acompanhar a velocidade do Node.js enquanto mantém o desempenho do aplicativo. O MongoDB é perfeito para esse tipo de requisito, pois possui um ciclo de desenvolvimento muito rápido e um desempenho bastante eficiente.

E para facilitar nosso trabalho de instalação e configuração durante o acesso ao banco de dados, iremos utilizar em nossos projetos a versão do MongoDB na nuvem: o MongoDB Atlas.

## Conhecendo o MongoDB Atlas

O MongoDB Atlas é um serviço global de banco de dados em nuvem desenvolvido pelo MongoDB. Você tem a opção de escolher o provedor de serviços em nuvem. A plataforma MongoDB Atlas lidará com a complexidade de implantação, gerenciamento e recuperação de suas implantações. Com o MongoDB Atlas, é possível aproveitar a flexibilidade e a escalabilidade de um banco de dados de documentos, com a facilidade e automação de um serviço totalmente gerenciado em sua nuvem preferida (MONGODB, 2021).

Ele funciona com serviços de hospedagem como AWS, Azure e Google Cloud para ajudar os usuários a provisionar, manter e proteger novos bancos de dados para seus aplicativos. Assim, usaremos seu nível de serviço gratuito para hospedar bancos de dados MongoDB para nossas aplicações durante nossos exemplos.

Primeiramente, você terá que fazer uma conta gratuita. Para isso, acesse o seguinte site: <https://www.mongodb.com/atlas/database>. Clique no botão “Try Free”. Preencha os campos conforme os dados forem sendo solicitados ou utilize uma conta do Google para realizar o cadastro.

Na tela de preferências, escolha as opções que mais se adequem à sua realidade e clique no botão “Finish”, conforme imagem apresentada.

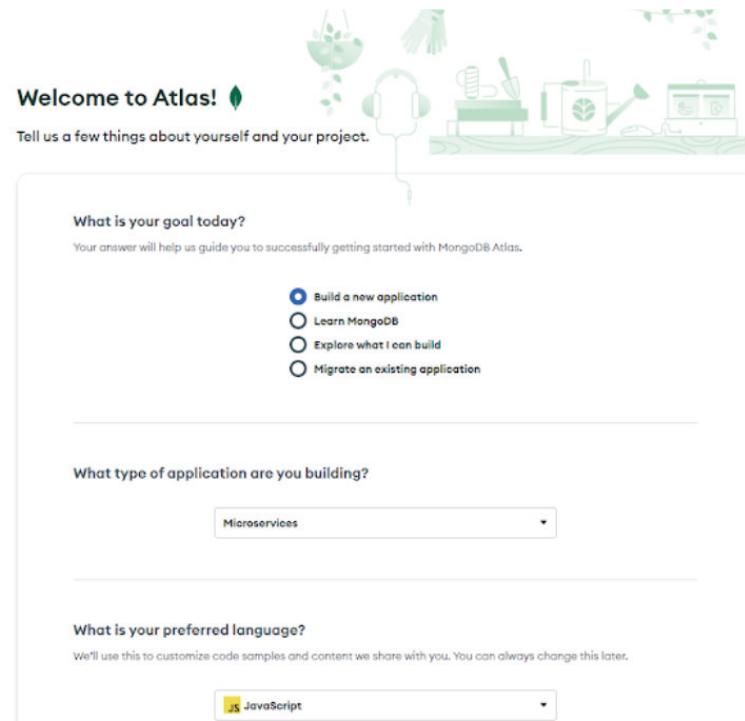


Figura 1 - Tela de preferência do MongoDB Atlas

Fonte: do Autor (2022)

Na próxima tela, você deverá escolher a forma de disponibilização do banco de dados na nuvem. Para isso, escolha a opção “Shared”, que é uma versão “Free”, assim não será necessário realizar nenhum pagamento. Clique no botão “Create”, conforme identificado na imagem.

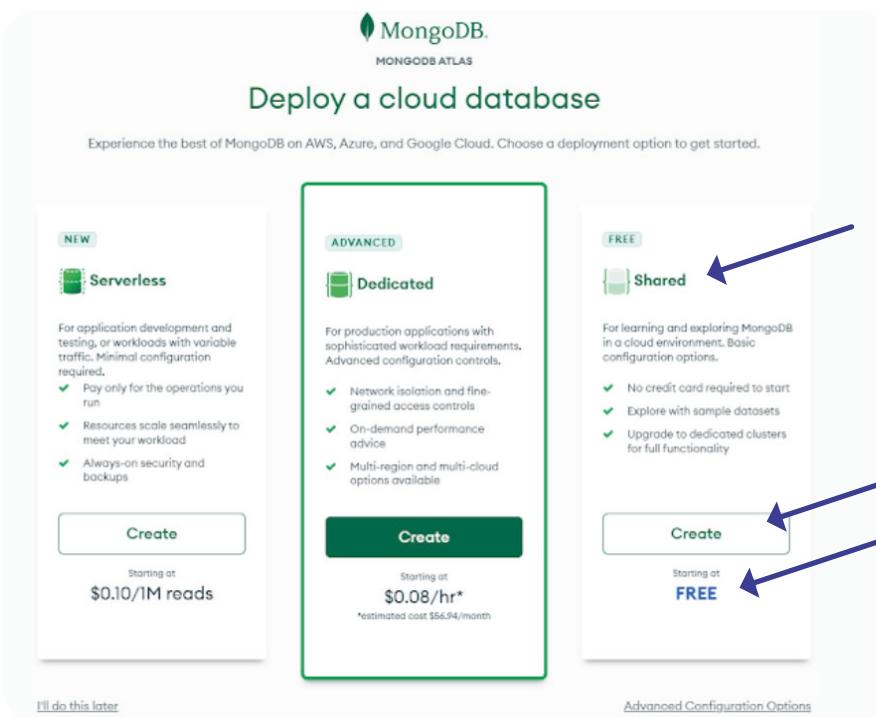


Figura 2 - Tela de escolha do processo de implantação do banco de dados

Fonte: do Autor (2022)

O próximo passo é criar um cluster. Na tela seguinte, certifique-se de que a opção “Free Shared” esteja selecionada para o tipo de implantação do banco de dados, pois isso determinará que seja gratuito e só o usaremos para um projeto pessoal. Em seguida, você deverá definir qual provedor de nuvem será utilizado. Você tem a opção de escolher AWS, Google Cloud ou Azure. Então, na seção “Cloud Provider & Region”, deixaremos a opção padrão sugerida, por exemplo, “AWS”. Usaremos a região “Sao Paulo (sa-east-1)”, pois ela possui a tag de região recomendada, além de ser de nível gratuito. Veja as configurações selecionadas na imagem a seguir.

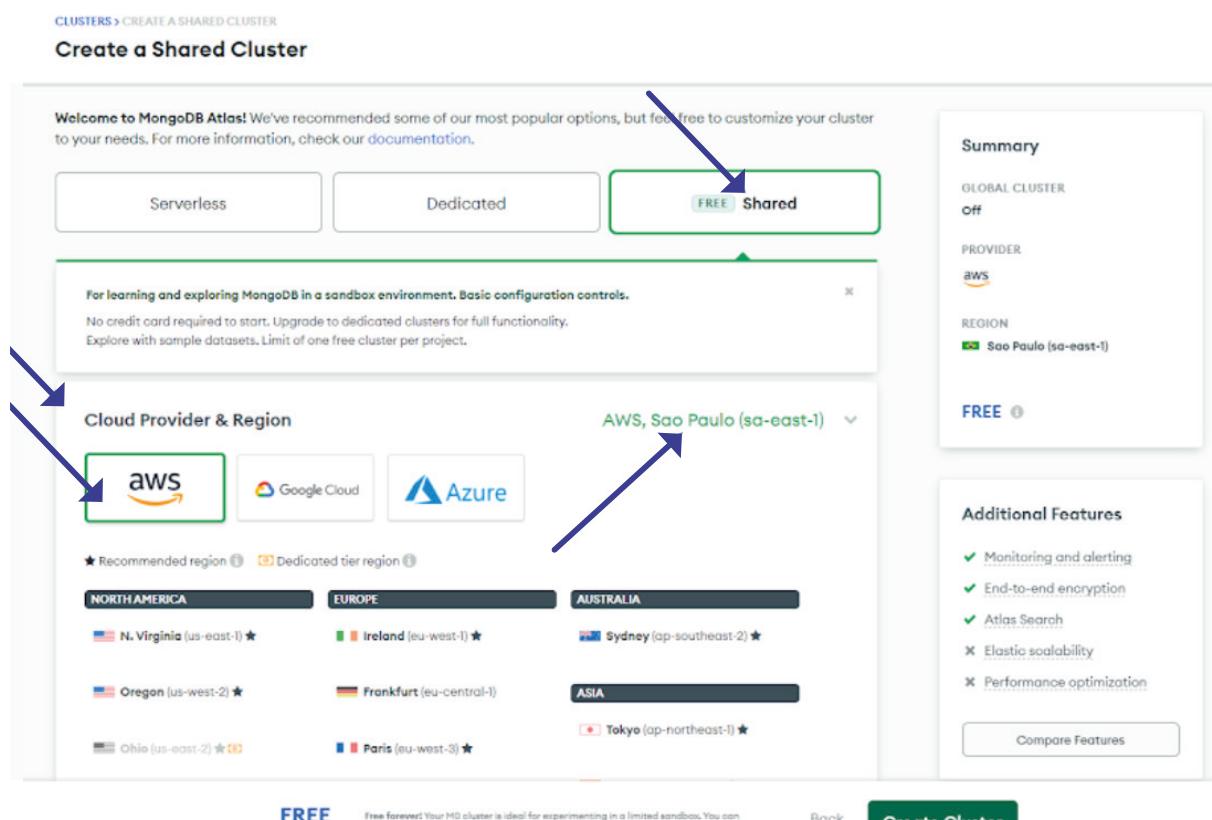


Figura 3 - Tela de escolha da região do cluster a ser implantado

Fonte: Fonte: do Autor (2022)

Na seção “Cluster Tier”, certifique-se de que a opção “M0 Sandbox”, que indica o nível de gratuidade “Free forever”, esteja selecionada. Você não precisa alterar as informações localizadas na seção “Additional Settings”. Entretanto, sinta-se à vontade para alterar o nome do cluster, se necessário. Feito esses procedimentos, clique no botão “Create Cluster”, conforme mostra a imagem.

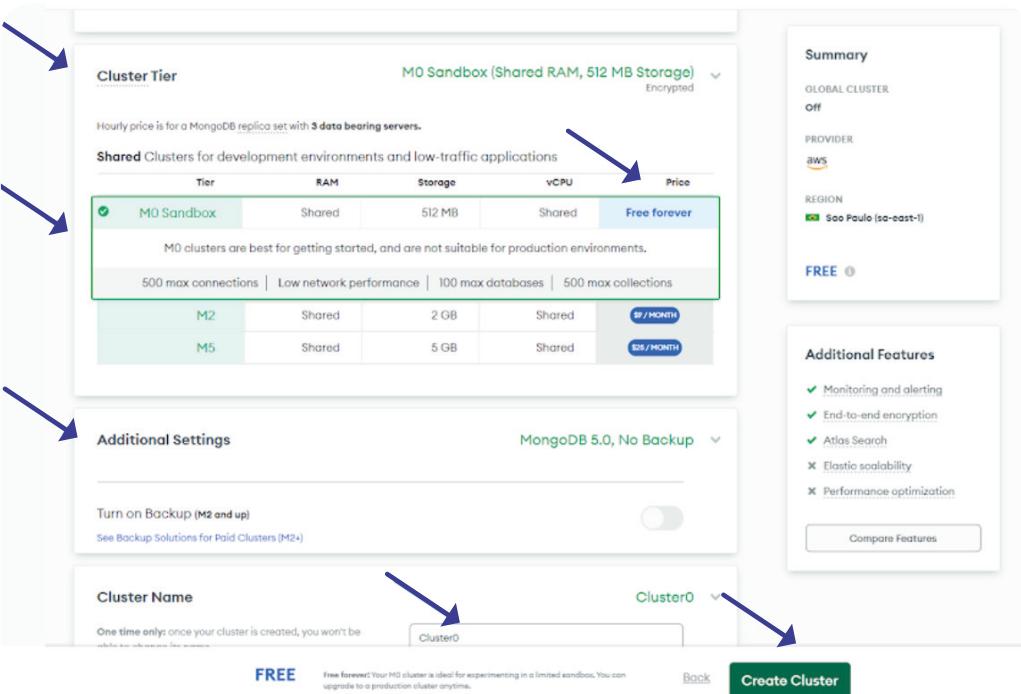


Figura 4 - Tela de definição da camada do cluster

Fonte: do Autor (2022)

Este processo pode levar de 3 a 5 minutos para ser finalizado. Enquanto isso, aguarde até que o cluster seja criado. Uma vez feito isso, você deve acessar o menu “Database Access” e clicar no botão “Add New Database User” para adicionar um novo usuário, que irá ter acesso ao banco de dados remotamente pela aplicação Node.js.

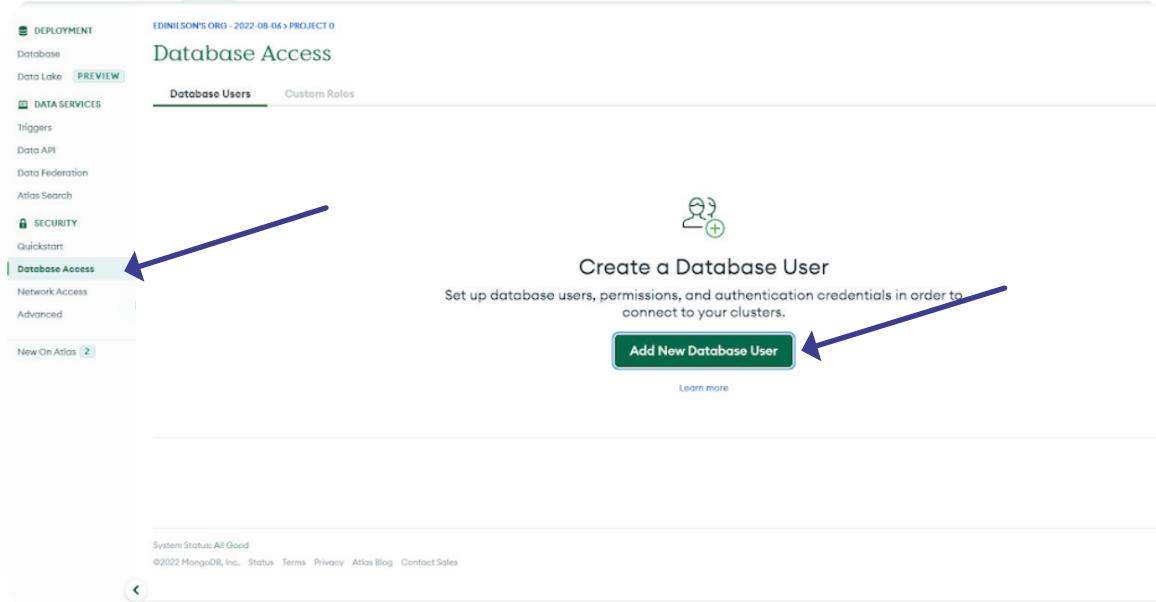


Figura 5 - Menu para gerenciamento de acessos

Fonte: do Autor (2022)

Na tela a ser exibida, na seção “Password Authentication”, informe um nome de usuário e uma senha de sua escolha. As demais configurações podem ser mantidas conforme o padrão sugerido. Posteriormente, clique no botão “Add User”, localizado na parte inferior da página.

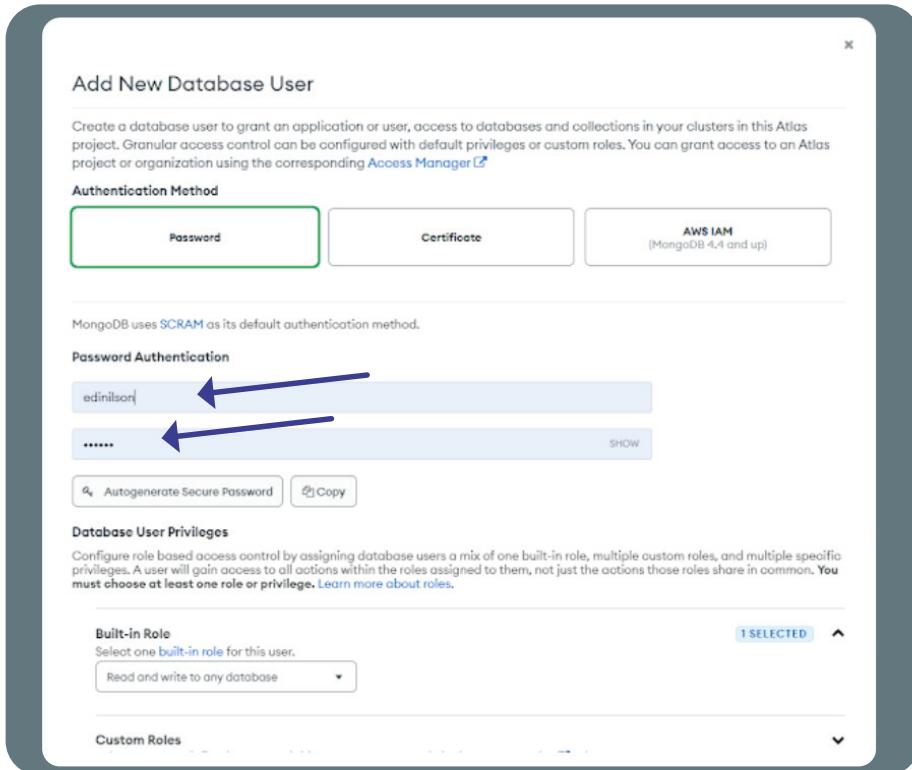


Figura 6 - Tela para adição de novo usuário

Fonte: do Autor (2022)

Agora, você precisa configurar o acesso externo pelas aplicações ao nosso banco de dados hospedado na nuvem. Neste caso, acesse o menu “Network Access” e clique no botão “Add IP Address”.

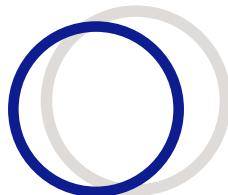


Figura 7 - Menu de gerenciamento de rede

Fonte: do Autor (2022)

Na próxima tela de adição do IP, clique no botão “ALLOW ACCESS FROM ANYWHERE”. Assim, tornaremos o banco de dados acessível de qualquer lugar. Isso definirá o valor de entrada da lista de permissões para “0.0.0.0/0”. Lembre-se de que esta opção deve ser utilizada somente durante os testes de conexão e nunca deve ser utilizada para um ambiente em produção. O MongoDB informará por e-mail que este processo é muito perigoso e que oferece riscos de acesso não autorizado aos dados. Posteriormente, clique no botão “Confirm”.

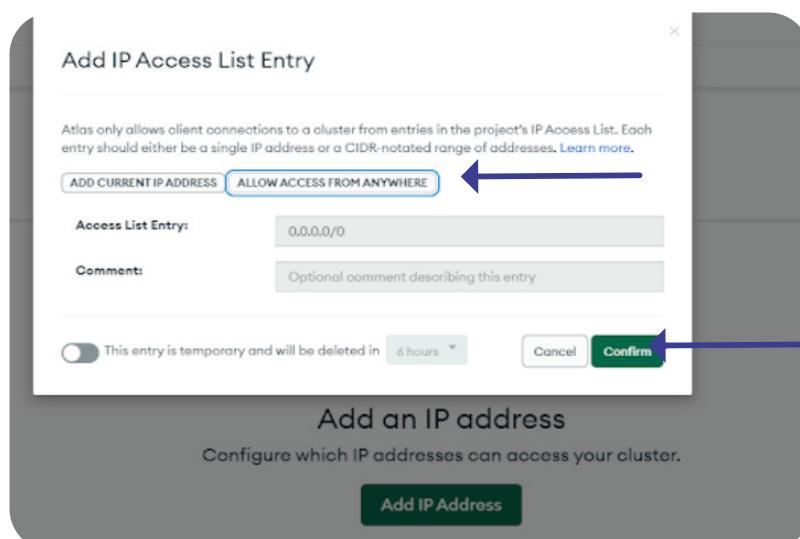


Figura 8 - Tela de adição de IPs

Fonte: do Autor (2022)

O último passo é obter a *string* de conexão do banco de dados para ser utilizado nas aplicações Node.js. Para isso, clique no menu “Database” e, depois, na opção “Connect”.

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with options like Project 0, DEPLOYMENT, Database (which is selected and highlighted with a blue arrow), DATA SERVICES, SECURITY, and New On Atlas. The main area is titled "Database Deployments" and shows a cluster named "Cluster0". At the top of this section, there are four buttons: "Connect" (highlighted with a blue arrow), "View Monitoring", "Browse Collections", and "...". Below these buttons, there's a section for "More Storage for \$9/mo" and some connection statistics. Further down, there's a "Upgrade" button and a table with cluster details. At the bottom of the main content area, there's a footer with links to System Status, ©2022 MongoDB, Inc., Status, Terms, Privacy, Atlas Blog, and Contact Sales.

Figura 9 - Tela de acesso ao gerenciamento do cluster

Fonte: do Autor (2022)

Este menu de acesso oferece três opções para se conectar ao cluster do banco de dados a partir do seu aplicativo. Neste caso, devemos selecionar “Connect your application”. Esta opção lhe dará uma *string* de conexão e exemplos de como usá-la.

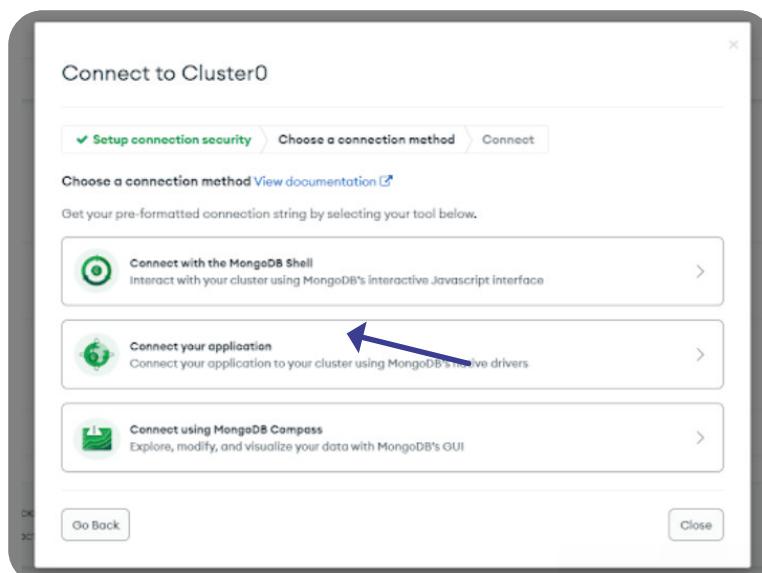


Figura 10 - Tela de acesso à *string* de conexão ao cluster

Fonte: do Autor (2022)

A opção de driver deve ser definida como “Node.js”, e a versão deve ser “4.1 or later”. Na sequência, você precisará copiar a *string* de conexão apresentada e armazená-la em um local seguro, pois iremos utilizá-la posteriormente em um exemplo prático. Veja um exemplo de como uma *string* de conexão se parece: “mongodb+srv://ednilson:<password>@cluster0.7rvrjsn.mongodb.net/?retryWrites=true&w=majority”, devendo a expressão “<password>” ser substituída pela senha que você definiu para o seu usuário criado durante o cadastro na tela “Database Access”.

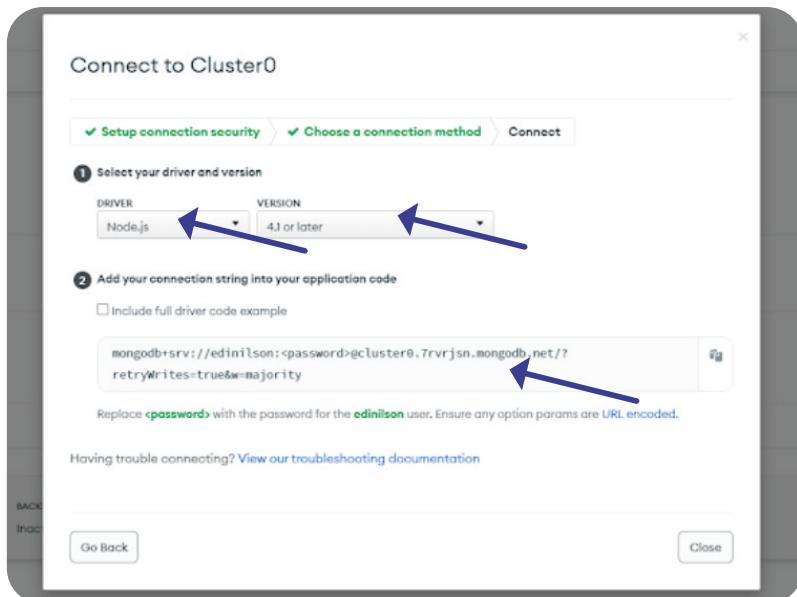


Figura 11 - Tela de definição da *string* de conexão

Fonte: do Autor (2022)

## Pacote Mongoose

O NoSQL trouxe flexibilidade ao mundo tabular dos bancos de dados. O MongoDB, em particular, tornou-se uma excelente opção para armazenar documentos JSON (JavaScript Object Notation) não estruturados. Os dados começam como JSON na interface do usuário e passam por pouquíssimas transformações para serem armazenados, de modo que obtemos benefícios com o aumento do desempenho e a diminuição do tempo de processamento.

Mas, NoSQL não significa uma completa falta de estrutura. Você ainda precisa validar e converter seus dados antes de armazená-los e ainda pode precisar aplicar alguma lógica de negócios a eles. Recebeu dados e precisa validá-los para garantir que o que recebeu é o que precisa? Também pode ser que você precise processar os dados de alguma forma antes de salvá-los. É aqui que o Mongoose entra em ação.

Mongoose é um pacote NPM para aplicativos Node.js. Ele permite definir esquemas para que nossos dados se encaixem, além de abstrair o acesso ao MongoDB. Dessa forma, é possível garantir que todos os documentos salvos compartilhem uma estrutura e contenham as propriedades necessárias.

O pacote “mongoose” conecta seus clusters ou coleções do MongoDB com seu aplicativo Node.js. Ele permite que você crie esquemas para seus documentos. O Mongoose fornece muitas funcionalidades ao criar e trabalhar com esquemas (MONGOOSEJS, 2022). Nesta seção, veremos um exemplo de como conectar uma instância do MongoDB com um aplicativo Node.js.

## Como conectar o MongoDB ao Node.js usando o Mongoose

Como você viu, o MongoDB é um dos bancos de dados NoSQL mais usados atualmente no mundo dos desenvolvedores. Os bancos de dados NoSQL permitem que os desenvolvedores enviem e recuperem dados como documentos JSON, em vez de objetos SQL. Para trabalhar com o MongoDB em um aplicativo Node.js, é possível usar o Mongoose.

Para isso, desenvolveremos uma aplicação Node.js simples para se conectar ao cluster do MongoDB Atlas antes que o Node.js execute o servidor na porta “8080”. Veja, a seguir, um exemplo de uma aplicação que realiza a conexão com o MongoDB Atlas, a versão do MongoDB na nuvem utilizando o pacote “mongoose”.



```
//Importação de pacotes/módulos para uso da aplicação
var express = require("express"); //Importação do pacote express
var app = express(); //Inicialização da aplicação 'app' pelo pacote express

//Importação do pacote mongoose
var mongoose = require("mongoose");

//Configuração da conexão com o MongoDB no serviço cloud MongoDB Atlas
const uri =
  "mongodb+srv://ednilson:ed2021@cluster0.puysq.mongodb.net/?retryWrites=true&w=majority";
```

```

//Validação da configuração da conexão com o MongoDB
mongoose
  .connect(uri, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("A conexão com o MongoDB foi realizada com sucesso!");
  })
  .catch((err) => {
    console.log(err);
  });
}

//Rota para exibir uma mensagem ao usuário
app.get("/", (req, res) => {
  res.send("Olá mundo! Esta é a página inicial da nossa aplicação.");
});

//A porta é uma variável de ambiente
const porta = process.env.PORT || 8080;
app.listen(porta, () =>
  console.log("Servidor inicializado na porta: " + porta)
);

```

Usando o método “connect” que o “mongoose” nos fornece, estamos primeiro passando a *string* de conexão armazenada na variável “uri”. Posteriormente, passamos alguns parâmetros auxiliares, como “useNewUrlParser” e “useUnifiedTopology”. O método “connect” retornará uma promessa (*promise*) do JavaScript, e precisaremos lidar com essa promessa. Para isso, é inserido um “.then” após o método, e o comando “console.log” é acionado com uma mensagem de sucesso no caso de que tudo corra bem. Também, inserimos um “.catch”, responsável por detectar quaisquer erros se não pudermos nos conectar ao cluster do MongoDB Atlas.

Para testar este exemplo, você precisará criar um projeto na plataforma CodeSandbox e instalar as dependências: "express" e "mongoose". Posteriormente, cole o código-fonte apresentado anteriormente e substitua o valor da variável "uri" pelo valor da sua *string* de conexão, obtida em procedimentos anteriores, para ter acesso ao seu banco de dados no MongoDB Atlas. Assim, ao executar sua aplicação, caso a sua configuração com o MongoDB Atlas tenha sido realizada com sucesso, será exibida a seguinte mensagem no "Terminal": "A conexão com o MongoDB foi realizada com sucesso!", conforme mostra a imagem seguinte.

The screenshot shows a CodeSandbox environment with the following components:

- Left Panel (Files):** Shows the project structure with files: README.md, index.js, package.json, and yarn.lock. A callout box labeled "Dependências instaladas" points to the package.json file, which lists "express" and "mongoose" as dependencies.
- Code Editor (index.js):** Displays the Node.js code for the application. A callout box labeled "URI de conexão com o MongoDB Atlas (altere o usuário e senha pelos seus dados de acesso)" highlights the line `const uri = "mongodb+srv://ednilson:ed2021@cluster0.puysq.mongodb.net/?retryWrites=true&w=majority";`.
- Output Terminal:** Shows the command `yarn start` being run, followed by logs indicating the application is starting and connecting to MongoDB. A callout box labeled "Mensagem confirmando que a conexão com o MongoDB foi realizada com sucesso" points to the line "A conexão com o MongoDB foi realizada com sucesso!".
- Preview Area:** Shows the application's UI with the message "Olá mundo! Esta é a página inicial da nossa aplicação." A callout box labeled "Mensagem de saudação exibida pela rota principal de aplicação" points to this message.
- Bottom Status Bar:** Shows the status bar with icons for Share, Fork, and Create Sandbox.

Figura 12 - Exemplo de conexão com Mongoose

Fonte: do Autor (2022)

Caso sejam exibidos erros no “Terminal”, você deverá conferir se a sintaxe da sua *string* de conexão está correta ou se deverá refazer os procedimentos de configuração da sua conta do MongoDB Atlas.

## Criação de um esquema utilizando o Mongoose

MongoDB é um banco de dados sem esquema. Ao trabalhar com Node.js, é possível usar o “mongoose” para definir um esquema para uma coleção do MongoDB. Um esquema do Mongoose define a forma dos documentos dentro de uma coleção específica. Acompanhe, a seguir, um esquema utilizando o mongoose.



```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;

var UsuarioEsquema = new Schema({
  nome: String,
  login: String,
  senha: String
});

module.exports = mongoose.model("Usuario", UsuarioEsquema);
```

Este arquivo, nomeado, por exemplo, como “usuario.js”, está dentro de uma pasta “modelo”, localizada em um projeto desenvolvido na plataforma CodeSandbox. Destacamos que, para este projeto, também há a necessidade de instalarmos o pacote “mongoose”. Na sequência, é possível observar que, neste arquivo, temos a importação do pacote “mongoose” para podermos utilizar todas as funcionalidades que ele nos oferece. Posteriormente, chamamos o construtor “Schema” para criar uma instância chamada de “UsuarioEsquema” deste esquema. A instância do esquema, agora, é usada para definir um esquema contendo os seguintes atributos: “nome”, “login” e “senha”. Todos esses atributos são do tipo “String”.

A maioria desses tipos de dados será familiar para os desenvolvedores. Se você não estiver familiarizado com os tipos de dados, basta definir o tipo de dados que você armazenará. Por exemplo, um nome será “String” porque é texto. Um preço ou idade será um “Number” porque é um valor numérico. Um valor lógico é um valor verdadeiro ou falso, podendo ser definido pelo tipo “Boolean”.

Posteriormente, você vai precisar exportar este módulo para ser utilizado em outros projetos, por isso, deve utilizar o comando “module.exports”. O primeiro parâmetro do comando “mongoose.model” é o nome da coleção que conterá os documentos que, neste caso, é “Usuario”. O segundo parâmetro é o esquema que definimos anteriormente, nomeado como “UsuarioEsquema”.

Agora, este modelo está pronto para ser importado e usado para criar dados seguindo nosso esquema na coleção “Usuario”. Este é um padrão de desenvolvimento comum quando se usa Node.js e MongoDB. Para importar este esquema em algum projeto desenvolvido em Node.js, utilize a seguinte instrução:

```
var Usuario = require("./modelo/usuario");
```

## Estilo arquitetural REST

Vivemos em um mundo moderno e altamente conectado que compartilha uma enorme quantidade de dados a cada segundo por meio de navegadores, servidores, softwares e aplicativos. Para que todos esses sistemas se comuniquem, temos uma ferramenta que é a peça-chave para integrar toda essa complexidade: as APIs (*Application Programming Interface*, ou Interface de Programação de Aplicação).

Uma API é um contrato (ou especificação) prometido pelo software, o qual ele honrará caso outro software queira interagir com ele para realizar operações de negócios. A API permite que dois ou mais aplicativos de software conversem entre si por meio de uma interface de computação bem definida.

Os aplicativos móveis podem ser um dos exemplos muito familiares de APIs. Os aplicativos móveis geralmente contêm todas as partes relacionadas à interface do usuário do aplicativo.

Todos os dados em tempo real exibidos nos aplicativos móveis são buscados por meio das APIs, por exemplo. Todas as informações meteorológicas, e-mails, pontuações de jogos, transmissões ao vivo e muitos outros dados em tempo real são obtidos de APIs implantadas no servidor.

Da mesma forma, muitos sites modernos buscam os dados sob demanda (que são buscados do servidor exatamente quando necessário). Esse tipo de interação acontece por meio de uma API.



Os aplicativos de software são desenvolvidos em partes. Para evitar desenvolver um software várias vezes em lugares diferentes, ele é desenvolvido como um componente reutilizável. Esse componente reutilizável é então exposto a outros componentes e aplicativos por meio de uma interface, ou seja, uma API.

A API ajuda a tornar o componente padrão, reutilizável, de fácil compreensão pelos usuários. A abstração ajuda a expor apenas informações mínimas relevantes para outras entidades e protege a lógica de negócios para realizar uma ação. No caso de uma API, ela permite interações entre sistemas, seguindo um conjunto de padrões e protocolos para compartilhar funcionalidades, informações e dados.

Fazer APIs reutilizáveis não só beneficia os usuários, mas também facilita a vida do desenvolvedor. O escopo definido da API ajuda no design, teste, construção, gerenciamento e controle de versão do componente.

Existem diferentes padrões de API. Atualmente, o principal padrão utilizado é o denominado REST (*Representational State Transfer*), que será descrito nos tópicos a seguir. Além desse padrão, há também um importante e crescente uso de APIs em outros padrões, especialmente o GraphQL e o gRPC. A escolha por um desses padrões dependerá de vários fatores e deve resultar de um trabalho de análise técnica e arquitetural (RUBY, 2007; SAUDATE, 2014).

## Fundamentos da API REST

O REST foi projetado para aproveitar os protocolos HTTP existentes quando usados para APIs da web. É muito flexível, pois não está vinculado a recursos ou métodos e tem a capacidade de lidar com diferentes chamadas e formatos de dados. Como a API REST não é restrita a um formato XML como SOAP (*Simple Object Access Protocol*), ela pode retornar vários outros formatos, dependendo do que for necessário. Se um serviço aderir a esse estilo, ele será considerado um aplicativo “RESTful”. REST permite que componentes acessem e gerenciem funções dentro de outro aplicativo (RUBY, 2007; SAUDATE, 2014).

As diretrizes REST sugerem o uso de um método HTTP específico, em um tipo específico de chamada feita ao servidor, em que pode retornar esses dados em um formato XML (*Extensible Markup Language*) ou JSON (*JavaScript Object Notation*), por exemplo. Dessa forma, é necessário encontrar um método HTTP adequado para a ação executada pela API (RUBY, 2007; SAUDATE, 2014). Confira na imagem a seguir os principais métodos HTTP.

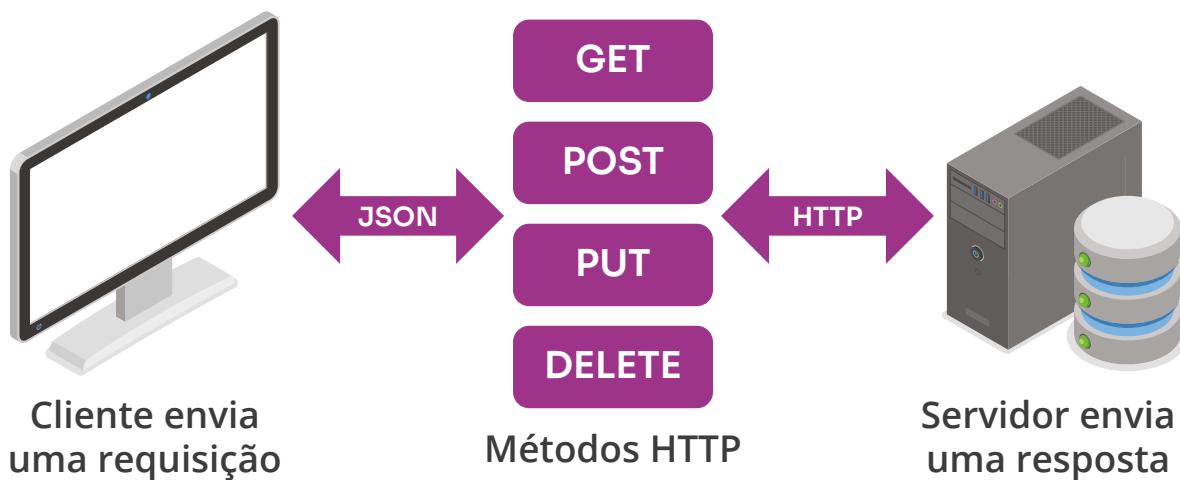


Figura 13 - Representação da API REST

Fonte: do Autor (2022)

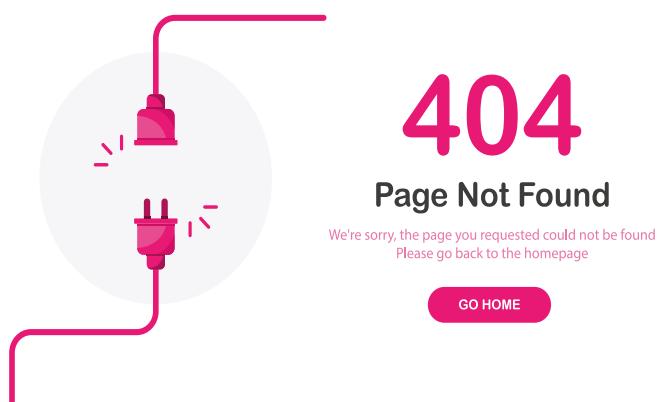
Assim, as APIs REST são responsáveis por permitirem que você desenvolva todos os tipos de aplicativos da web com todas as operações possíveis de CRUD (*Create, Read, Update, Delete*, isto é, Criar, Recuperar, Atualizar, Excluir). A partir dessas informações, destacamos que iremos focar apenas nos principais métodos HTTP: GET, POST, PUT e DELETE. Acompanhe, na sequência, suas principais definições.

## GET

Use solicitações GET para apenas recuperar informações de recursos. Como as solicitações GET não alteram o estado do recurso, eles são considerados métodos seguros.

Além disso, quando as APIs GET fazem várias solicitações idênticas, elas devem produzir sempre o mesmo resultado até que outra API (POST ou PUT) altere o estado do recurso no servidor, prevalecendo o conceito da idempotência. Veja os códigos de resposta da API GET (RUBY, 2007; SAUDATE, 2014):

- › Para qualquer API HTTP GET, se o recurso for encontrado no servidor, ele deverá retornar o código de resposta HTTP 200 (OK) junto com o corpo da resposta, que geralmente é conteúdo XML ou JSON (devido à sua natureza independente de plataforma);
- › Caso o recurso não seja encontrado no servidor, a API deve retornar o código de resposta HTTP 404 (*NOT FOUND*);
- › Da mesma forma, se for determinado que a própria solicitação GET não foi formada corretamente, o servidor retornará o código de resposta HTTP 400 (*BAD REQUEST*).

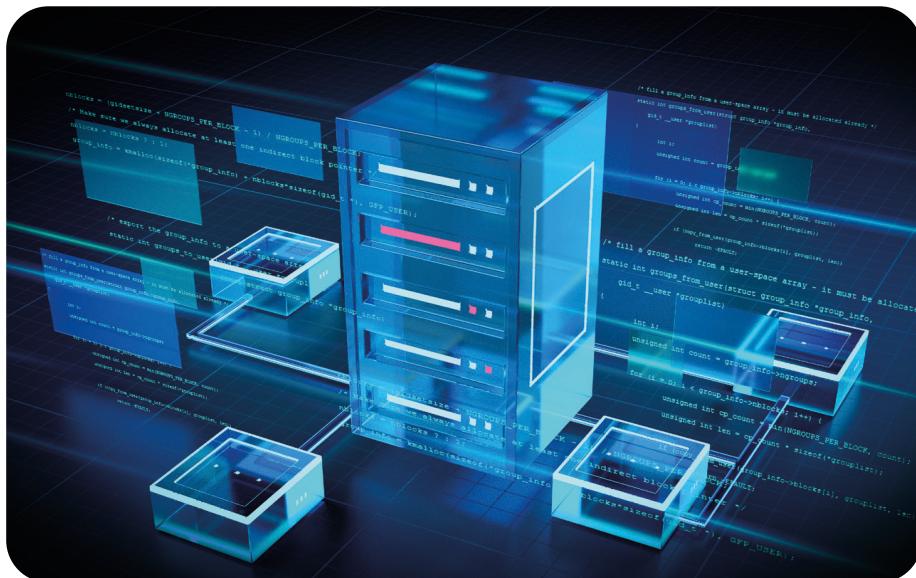


Confira alguns exemplos de URIs (*Uniform Resource Identifier*) utilizando GET:

- › HTTP GET `http://www.site.com.br/api/usuarios/meuperfil`: obter perfil de usuário logado;
- › HTTP GET `http://www.site.com.br/api/usuarios/{nomeusuario}/perfil`: obter perfil de usuário por nome de usuário;
- › HTTP GET `http://www.site.com.br/api/usuarios/{nomeusuario}/postagens`: obter postagens criadas pelo usuário;
- › HTTP GET `http://www.site.com.br/api/usuarios/{nomeusuario}/albuns`: obter álbuns criados pelo usuário;
- › HTTP GET `http://www.site.com.br/api/usuarios/verificarNomeUsuario`: verificar se o nome de usuário está disponível para registro.

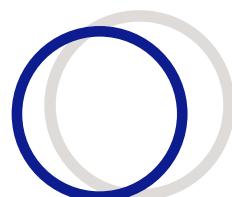
## POST

Use o método POST para criar novos recursos subordinados, por exemplo, quando um arquivo está subordinado a um diretório que o contém ou quando um registro está subordinado a uma tabela de banco de dados.



Falando estritamente sobre REST, os métodos POST são usados para criar um novo recurso na coleção de recursos. Observe que invocar duas solicitações POST idênticas resultará em dois recursos diferentes contendo as mesmas informações. Confira os códigos de resposta da API para um método POST (RUBY, 2007; SAUDATE, 2014):

- › Idealmente, se um recurso foi criado no servidor de origem, a resposta deve ser o código de resposta HTTP 201 (*CREATED*) e conter uma entidade que descreva o status da solicitação e se refira ao novo recurso e um cabeçalho de localização;
- › Muitas vezes, a ação executada pelo método POST pode não resultar em um recurso que possa ser identificado por um URI. Nesse caso, o código de resposta HTTP 200 (OK) ou 204 (NO CONTENT) é o status de resposta apropriado.



Confira alguns exemplos de URIs utilizando POST:

- › HTTP POST `http://www.site.com.br/api/usuarios`: criar um novo usuário;
- › HTTP POST `http://www.site.com.br/api/postagens`: criar uma nova postagem;
- › HTTP POST `http://www.site.com.br/api/postagens/{postagemId}/comentarios`: criar um novo comentário para postagem com identificador igual ao valor de “postagemId”.

## PUT

Use o método PUT principalmente para atualizar um recurso existente. A diferença entre os métodos POST e PUT pode ser observada nos URIs de solicitação. As solicitações POST são feitas em coleções de recursos, enquanto as solicitações PUT são feitas em um único recurso. Veja os códigos de resposta para o método PUT (RUBY, 2007; SAUDATE, 2014):

- › Se um novo recurso foi criado pela API PUT, o servidor de origem deve informar, por meio da resposta, o código de resposta HTTP 201 (*CREATED*).
- › Se um recurso existente for modificado, os códigos de resposta 200 (OK) ou 204 (*NO CONTENT*) devem ser enviados para indicar a conclusão bem-sucedida da solicitação.

Confira alguns exemplos de URIs utilizando PUT:

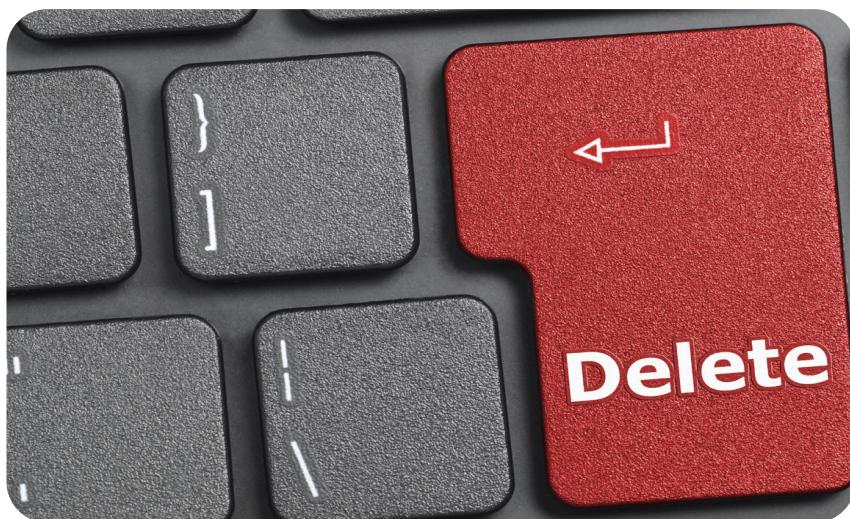
- › HTTP PUT `http://www.site.com.br/api/usuarios/{nomeusuario}`: atualizar um usuário específico pelo nome de usuário (“`nomeusuario`”);
- › HTTP PUT `http://www.site.com.br/api/postagens/{id}`: atualizar uma postagem específica pelo identificador (“`id`”);
- › HTTP PUT `http://www.site.com.br/api/postagens/{postagemId}/comentarios/{id}`: atualizar um comentário específico por identificador (`id`) se pertencer à postagem com identificador igual ao valor de “`postagemId`”.

## DELETE

Os métodos DELETE excluem os recursos. Se você excluir um recurso, ele será removido da coleção de recursos. Veja os códigos de resposta para um método DELETE (RUBY, 2007; SAUDATE, 2014):

- › Uma resposta bem-sucedida de solicitações DELETE deve ser uma resposta de código HTTP 200 (OK) se a resposta incluir uma entidade descrevendo o status;

- › O status deve ser 202 (*ACCEPTED*) se a ação foi enfileirada;
- › O status deve ser 204 (*NO CONTENT*) se a ação foi executada, mas a resposta não incluiu uma entidade;
- › Chamar repetidamente a API **DELETE** nesse recurso não alterará o resultado. No entanto, chamar **DELETE** em um recurso uma segunda vez retornará um 404 (*NOT FOUND*), pois esse recurso já foi removido.



Confira alguns exemplos de URLs utilizando **DELETE**:

- › **DELETE** `http://www.site.com.br/api/usuarios/{nomeusuario}`: excluir usuário pelo nome de usuário (“`nomeusuario`”);
- › **DELETE** `http://www.site.com.br/api/postagens/{id}`: excluir postagem pelo identificador (“`id`”);
- › **DELETE** `http://www.site.com.br/api/postagens/{postagemId}/comentarios/{id}`: excluir comentário por identificador (“`id`”) se pertencer a postagem com identificador = “`postagemId`”.

## Definição de URI

Um URI (*Uniform Resource Identifier*) é uma cadeia de caracteres usada para identificar um recurso na web. Em palavras simples, o URI em um serviço da web é representado por um hiperlink com acesso a um recurso, sendo o único meio de trocas de representações entre clientes e servidores.

O cliente usa um URI para localizar os recursos na web, enviar uma solicitação ao servidor e ler a resposta. Por exemplo, no quadro a seguir, é exibido o URI de acesso ao recurso para gerenciamento de usuários na web.

| Método | URL                | Descrição                                       |
|--------|--------------------|---|
| GET    | /api/usuarios      | Obter todos os usuários.                        |
| GET    | /api/usuarios/{id} | Obter um usuário pelo "id" (identificador).     |
| POST   | /api/usuarios      | Criar um novo usuário.                          |
| PUT    | /api/usuarios/{id} | Atualizar um usuário pelo "id" (identificador). |
| DELETE | /api/usuarios/{id} | Excluir um usuário pelo "id" (identificador).   |

## Teste de API utilizando a ferramenta Postman

Postman é uma das ferramentas de teste de software mais populares, usada para testes de API (*Application Programming Interface*). Com a ajuda dessa ferramenta, os desenvolvedores podem criar, testar, compartilhar e documentar APIs facilmente. Dessa forma, ela é usada para testes de *back-end*, onde inserimos o URL, enviamos a solicitação ao servidor e recebemos a resposta de volta do servidor (POSTMAN, 2022).

Esta seção o ajudará a entender por que o Postman é tão famoso e o que o torna único quando comparado a outras ferramentas de teste de API. Postman é uma plataforma independente de API de teste de software para construir, testar, projetar, modificar e documentar APIs. É uma interface gráfica de usuário simples para enviar e visualizar solicitações e respostas HTTP.

Ao usar o Postman para fins de teste, não é necessário escrever nenhum código de rede do cliente HTTP. Em vez disso, construímos suítes de teste chamadas de coleções e deixamos o Postman interagir com a API.

Nessa ferramenta, quase todas as funcionalidades que qualquer desenvolvedor pode precisar estão incorporadas. Essa ferramenta tem a capacidade de fazer vários tipos de solicitações HTTP, como GET, POST, PUT, DELETE, fundamento para uso no estilo arquitetural REST, e converter a API em código para linguagens como JavaScript e Python.

O Postman é baseado em uma ampla gama de ferramentas extremamente fáceis de usar. A seguir, estão os motivos pelos quais o Postman é usado (POSTMAN, 2022):

- **Acessibilidade:** pode-se usá-lo em qualquer plataforma depois de instalá-lo, basta fazer login na conta;
- **Uso de coleções:** o Postman permite aos usuários construir coleções para suas chamadas API. Cada conjunto pode criar várias solicitações e subpastas. Isso ajudará a organizar os conjuntos de teste;

- › **Teste de automação:** os testes podem ser realizados em várias repetições ou iterações usando o Collection Runner ou Newman, o que economiza tempo para testes repetidos;
- › **Criação de ambientes:** o design de vários ambientes resulta em menos replicação de testes, pois cada um pode usar a mesma coleção, mas para uma configuração diferente;
- › **Depuração:** para depurar os testes com eficácia, o console ajuda a rastrear quais dados estão sendo recuperados;
- › **Colaboração:** você pode importar ou exportar coleções e ambientes para aprimorar o compartilhamento de arquivos entre os membros da equipe.



### Dica

Quer saber mais detalhes de como utilizar a ferramenta Postman? Acesse o link <https://www.devmedia.com.br/testando-apis-web-com-o-postman/37264> para mais informações.



Neste estudo, você viu como configurar o Mongoose e como estabelecer uma conexão com o banco de dados MongoDB Atlas. O Mongoose pode ser usado para conectar o MongoDB e o MongoDB Atlas ao seu aplicativo Node.js. O MongoDB Atlas é um serviço de banco de dados baseado em nuvem criado e mantido pelo MongoDB. Você viu também que REST não é uma arquitetura; em vez disso, é um conjunto de restrições que cria um estilo de arquitetura de software, que pode ser usado para construir aplicativos distribuídos. Como complemento, você viu que uma API, como qualquer outra interface, permite interações. Em outras palavras, oferece aos desenvolvedores a oportunidade de construir e projetar produtos e serviços que se comunicarão com outros produtos e serviços. Por fim, você descobriu que Postman é usado com frequência pelos desenvolvedores e engenheiros de automação para garantir que o serviço esteja funcionando junto com a versão de compilação de uma API que está sendo implantada. Esperamos que você tenha achado este material útil. Reforçamos a importância de você continuar seus estudos!

## REFERÊNCIAS

MONGODB. **Introduction to MongoDB**. MongoDB, Inc., New York (NY, USA), 2020? Disponível em: <https://www.mongodb.com/docs/manual/introduction>. Acesso em: 25 jul. 2022.

MONGODB. **MongoDB Atlas Tutorial**. MongoDB, Inc., New York (NY, USA), 2021? Disponível em: <https://www.mongodb.com/basics/mongodb-atlas-tutorial>. Acesso em: 01 ago. 2022.

MONGOOSEJS. **Getting Started**. Mongoose ODM, 2022. Disponível em: <https://mongoosejs.com/docs/index.html>. Acesso em: 30 jul. 2022.

POSTMAN. **Introduction**. Postman HQ, San Francisco (CA, USA), 5 jul. 2022. Disponível em: <https://learning.postman.com/docs/getting-started/introduction>. Acesso em: 05 ago. 2022.

RUBY, S.; RICHARDSON, L. **Restful web services**. Sebastopol (CA, USA): O'Reilly Media, 2007.

SAUDATE, A. **REST**: construa APIs inteligentes de maneira simples. São Paulo (SP): Casa do Código, 2014.



