



Mundo Tech

PROGRAMAÇÃO BACK-END FUNDAMENTOS BÁSICOS DE BACK-END

SUMÁRIO

Fundamentos básicos de <i>back-end</i>	3
Introdução à plataforma Node.js	5
Configuração do ambiente de desenvolvimento utilizando a plataforma CodeSandbox	7
Introdução ao <i>framework</i> Express.js	14
Conhecendo o Express.js	15
Roteamento no Express	19
Middleware no Express.js	25
Referências.....	28



O desenvolvimento de um *back-end* envolve a criação de um aplicativo para permitir que os usuários interajam com o *front-end* de forma dinâmica e realizem algumas ações reais que produzam resultados. Uma estrutura de *back-end* é usada para criar esses aplicativos com o objetivo de fornecer funcionalidades específicas ao site. Escolher o *framework* de *back-end* certo para seu desenvolvimento JavaScript é crucial para garantir o sucesso do aplicativo geral. Por isso, neste estudo, serão abordados os conceitos da plataforma de aplicação Node.js e o *framework* Express.js. Como complemento, será apresentada a configuração do ambiente de desenvolvimento de projetos por meio da plataforma CodeSandbox. Por fim, você verá algumas dicas de como utilizar a ferramenta online Postman, responsável por realizar testes de API (*Application Programming Interface*). Agora é a hora de adquirir novos conhecimentos sobre essas tecnologias muito utilizadas no mercado! Bons estudos!

FUNDAMENTOS BÁSICOS DE BACK-END

O desenvolvimento *front-end* refere-se ao processo de criação do lado interativo ou interface de um site. Um *framework* que se utiliza para desenvolver o *front-end* de uma aplicação web é chamado de *framework front-end*. JavaScript, CSS e HTML são algumas das ferramentas mais populares para desenvolvimento *front-end*. Além disso, servem de recursos para os *frameworks front-end*, assim como React ou Angular. O trabalho de um desenvolvedor *front-end* é um trabalho de design, projetado para que os usuários possam usar bem o sistema.

Já o *back-end*, também conhecido como lado do servidor, é o lado que o usuário final não pode ver ou interagir. É a parte de codificação para dar ao seu site uma funcionalidade dinâmica. Envolve criar um banco de dados para seu aplicativo e anexá-lo ao *front-end* a fim de garantir que o usuário possa realmente interagir com o aplicativo ou executar uma ação.

Dessa forma, o desenvolvedor *back-end* é responsável pelo lado do servidor, a lógica do programa, o design de dados, o gerenciamento de dados do sistema e o projeto de arquitetura do sistema. São utilizadas algumas linguagens de desenvolvimento, tais como JavaScript, Python, Java, C#, PHP e Ruby. O trabalho do desenvolvedor *back-end* requer entendimento e aprendizado de novas tecnologias o tempo todo. Como regularmente há novas tecnologias ou atualizações de vários softwares, há uma boa lógica a ser pensada para projetar e resolver problemas de maneira funcional. Veja um comparativo entre *back-end* e *front-end* na imagem seguinte.

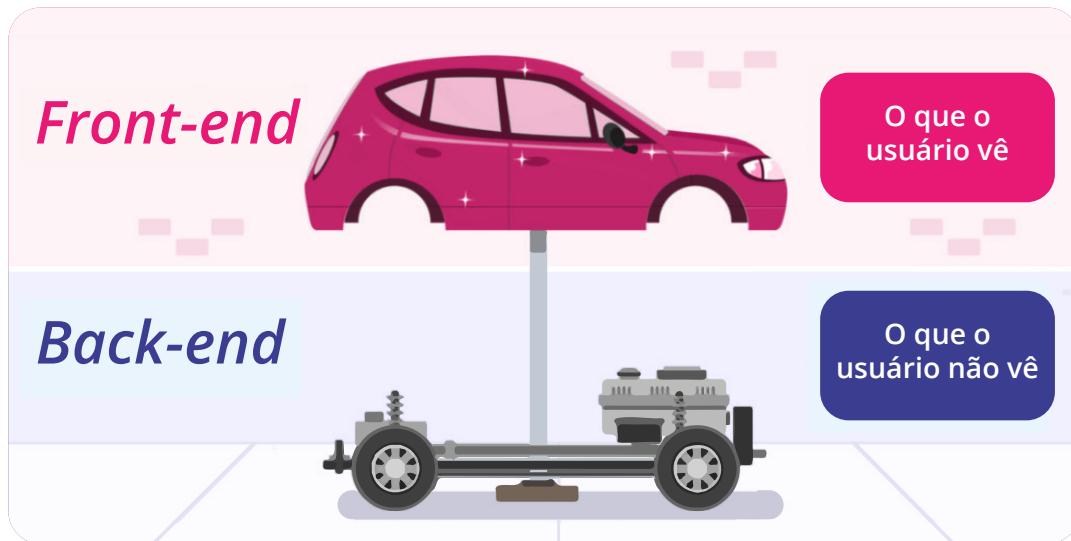


Figura 1 - Comparativo entre back-end e front-end

Fonte: adaptada de Clark (2021)

Como você pode observar na imagem apresentada, o desenvolvimento *front-end* e *back-end* é uma segregação de responsabilidades do lado do cliente e do lado do servidor. Comparado a um carro, veremos o *front-end* como um visual que pode ser sentido como cores, rodas de liga leve, luzes diversas, portas, volante, console, algo que podemos ver ou sentir. Ao contrário, o *back-end* indica o que não podemos ver, mas sabemos que pode funcionar, como o sistema de motor, o sistema de transmissão, o sistema de combustível, o sistema de condução automática etc.

Por exemplo, quando você preenche um formulário em uma página da web, está interagindo com o *front-end*. Essas informações são então processadas pelo código de *back-end* e as informações necessárias são enviadas de volta ao usuário em um formato legível pelo navegador. Veja um exemplo:

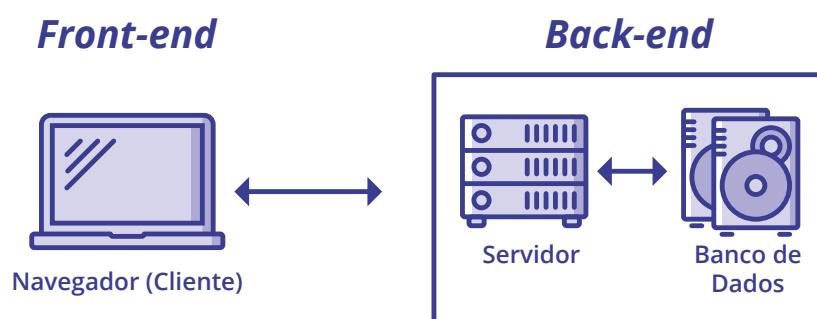


Figura 2 - Exemplo de interação entre back-end e front-end

Fonte: adaptado de Fernandes (2020)

Dessa forma, perceba que um *framework* que é usado para criar o código de *back-end*, incluindo o banco de dados, é chamado de *framework de back-end*. Eles são usados para construir aplicações web dinâmicas. Graças a uma ampla variedade de *frameworks*, o JavaScript agora está sendo usado para desenvolver aplicativos web *front-end* e *back-end*.



Dica

Os bancos de dados normalmente armazenam agregações de registros de dados ou arquivos que contêm informações, como transações de vendas, dados de clientes, informações financeiras e de produtos. Os Sistemas de Gerenciamento de Bancos de Dados (SGBDs) são constituídos por um conjunto de softwares com diferentes funcionalidades, que se complementam para oferecer serviços de manipulação de bancos de dados. Um SGBD é uma coleção de softwares que permitem ao usuário criar e manter um banco de dados. O SGBD, portanto, é um sistema de software de uso geral que facilita o processo de definição, construção, manipulação e compartilhamento de dados. Oferece interfaces para criação, manipulação e compartilhamento de bancos de dados, ou seja, recursos que facilitam a experiência do usuário (ELMASRI e NAVATHE, 2011).



Introdução à plataforma Node.js

Esta seção é um guia de introdução ao Node.js, o ambiente *runtime* (tempo de execução) JavaScript do lado do servidor. O Node.js é usado principalmente para criar servidores web. O Node.js é de código aberto e, desde sua introdução, em 2009, tornou-se muito popular e agora desempenha um papel significativo no cenário de desenvolvimento web. Assim, o Node.js executa o motor (interpretador) JavaScript, conhecido como V8, o núcleo do Google Chrome, fora do navegador. Isto permite ao Node.js se beneficiar das enormes melhorias de desempenho e da compilação em tempo real que o V8 oferece. Graças a isso, o código JavaScript executado na plataforma Node.js apresenta grande performance (NODEJS, 2022).

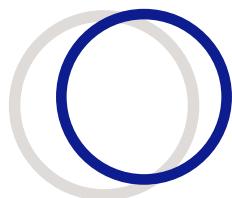


Um aplicativo Node.js é executado por um único processo, sem criar um novo thread para cada solicitação, diferente de outras linguagens de programação, como o Java. Assim, o Node.js fornece um conjunto de primitivas de entrada/saída (E/S) assíncronas em sua biblioteca padrão, que evitará o bloqueio do código JavaScript. O termo assíncrono se refere a todas as funções em JavaScript que são processadas em segundo plano sem bloquear nenhuma outra solicitação. Além disso, as bibliotecas no Node.js são escritas usando paradigmas de não bloqueio, tornando um comportamento de bloqueio uma exceção ao invés do normal (NODEJS, 2022).

Quando precisa realizar uma operação de E/S, como leitura da rede, acessar um banco de dados ou o sistema de arquivos, em vez de bloquear o thread, o Node.js retomará as operações quando a resposta voltar, em vez de desperdiçar os ciclos de espera do processador. Isto permite ao Node.js lidar com milhares de conexões simultâneas com um único servidor sem introduzir restrições de gerenciamento dos threads simultâneos. Caso isso não fosse implementado, haveria uma grande fonte de problemas.

O Node.js tem uma vantagem exclusiva, pois milhões de desenvolvedores de *front-end* que desenvolviam suas aplicações utilizando a linguagem JavaScript para o navegador agora são capazes de executar o código do lado do servidor e o código do lado do cliente sem a necessidade de aprender uma linguagem completamente diferente.

Além disso, o Node.js tem um grande número de bibliotecas. O gerenciador de pacotes mais conhecido e utilizado pelo Node.js é o NPM (Node Package Manager). A critério de curiosidade, há também um outro gerenciador de pacotes concorrente, conhecido como Yarn. Assim, o NPM, com sua estrutura simples, ajudou o ecossistema do Node.js a se expandir, e agora a plataforma do NPM hospeda milhares de pacotes de código aberto que você pode usar livremente.





Dica

O NPM é um repositório online para a publicação de projetos Node.js de código aberto. Além disso, ele é um utilitário de linha de comando para interagir com o referido repositório que auxilia na instalação de pacotes, gerenciamento de versões e gerenciamento de dependências. Uma infinidade de bibliotecas e aplicativos Node.js são publicados no NPM (GASPAR, 2021). Esses pacotes podem ser procurados no seguinte site:



<https://www.npmjs.com>



Configuração do ambiente de desenvolvimento utilizando a plataforma CodeSandbox

O IDE (Ambiente de Desenvolvimento Integrado) deste módulo será a plataforma CodeSandbox (<https://codesandbox.io/>), um ambiente de desenvolvimento integrado online, fácil de usar e compartilhável, que está em alta com os desenvolvedores de JavaScript e Node.js.

Os IDEs online aproveitaram os recursos de ferramentas baseadas em nuvem, crescendo em poder nos últimos anos. O ambiente CodeSandbox é uma das opções mais populares nesse espaço, e seu uso vem aumentando recentemente. O CodeSandbox está ganhando popularidade na codificação social por sua facilidade de uso, suporte tecnológico simplificado e estrutura de custo razoável. Além disso, o CodeSandbox é um projeto de código aberto (HASHDORK, 2021).

O CodeSandbox é mais conhecido como um ambiente para ativar e compartilhar rapidamente aplicativos JavaScript de *front-end*. Isso soa semelhante ao CodePen, visto em módulos anteriores, mas, na verdade, o CodeSandbox é um projeto mais ambicioso, com suporte *full-stack* quase comparável a um IDE online completo, embora apenas para JavaScript.

Agora, vamos entender os ambientes Sandbox. Eles determinam como o CodeSandbox hospeda seu projeto. Na plataforma, se você selecionar “Home” e, depois, “New Sandbox” você verá a opção do ambiente “static” (com um ícone do HTML5). Isso significa que é o Sandbox que está sendo executado como um aplicativo de *front-end* simples, como o JSFiddle ou o CodePen. Entretanto, para nossos projetos, precisamos criar um ambiente baseado em Node.js. Para a nossa felicidade, o CodeSandbox suporta a execução de aplicativos baseados em Node.js. Ao executar aplicativos Node.js, o CodeSandbox usa um contêiner do Docker com a imagem oficial do Node.js. Além de permitir acesso ao *runtime* do Node.js e aos scripts no arquivo “package.json”, o ambiente Node.js permite o acesso a um terminal de linha de comando.



Dica

Um arquivo “package.json” é um arquivo JSON localizado no diretório raiz do seu projeto. Este arquivo contém informações importantes sobre o projeto. Ele contém metadados legíveis sobre o projeto (como o nome e a descrição do projeto), bem como metadados funcionais, como o número da versão do pacote e uma lista de dependências exigidas pelo aplicativo.



Você pode utilizar sua conta do GitHub ou do Google para acessar esta plataforma de forma gratuita. Assim, não há necessidade de instalar nada em seu computador. Tudo pode ser feito diretamente do navegador. Isso vai facilitar todo o seu trabalho! Após realizar a autenticação na plataforma, vá até a opção “Home”, depois selecione “New Sandbox” e, posteriormente, “Create Sandbox” e, nas opções sugeridas, escolha o ambiente pré-definido “Node HTTP Server”, conforme apresentado na imagem seguinte.

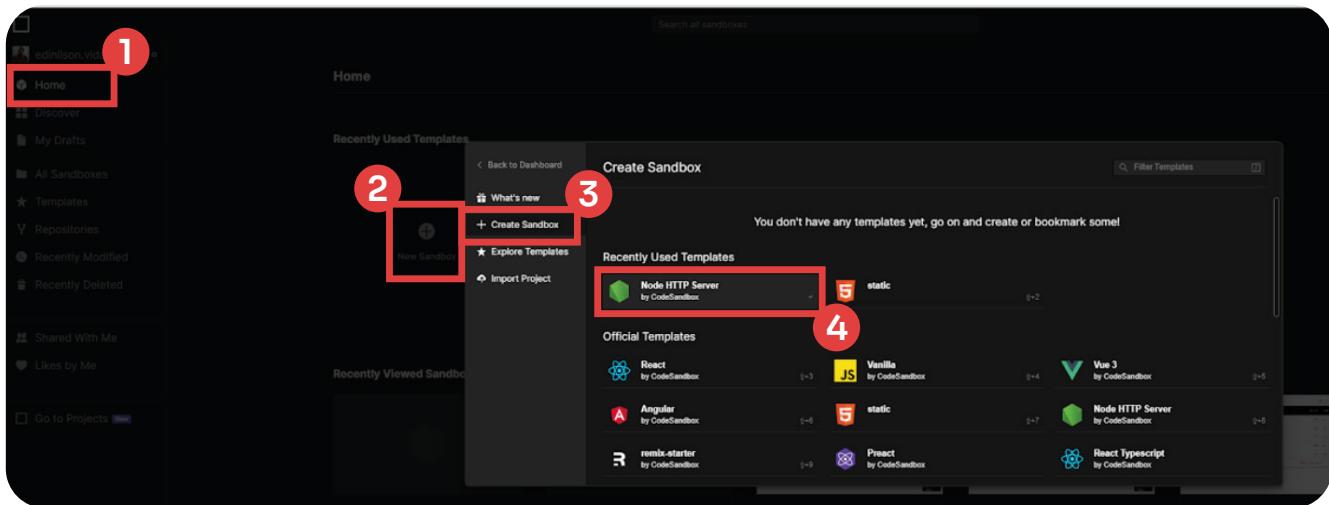


Figura 3 - Configuração de um ambiente na plataforma CodeSandbox

Fonte: do Autor (2022)

Após a realização da etapa anterior, o seguinte exemplo de projeto é criado. Os arquivos “package.json” e “index.js” são o que você esperaria de um módulo HTTP (*Hypertext Transfer Protocol*) simples com uma resposta “Hello World”, conforme o ambiente padrão criado e apresentado na imagem seguinte.

```

js index.js x
1 var http = require("http");
2
3 //create a server object:
4 http
5 .createServer(function(req, res) {
6   res.write("Hello World!"); //write a response to the client
7   res.end(); //end the response
8 })
9 .listen(8080); //the server object listens on port 8080
10

```

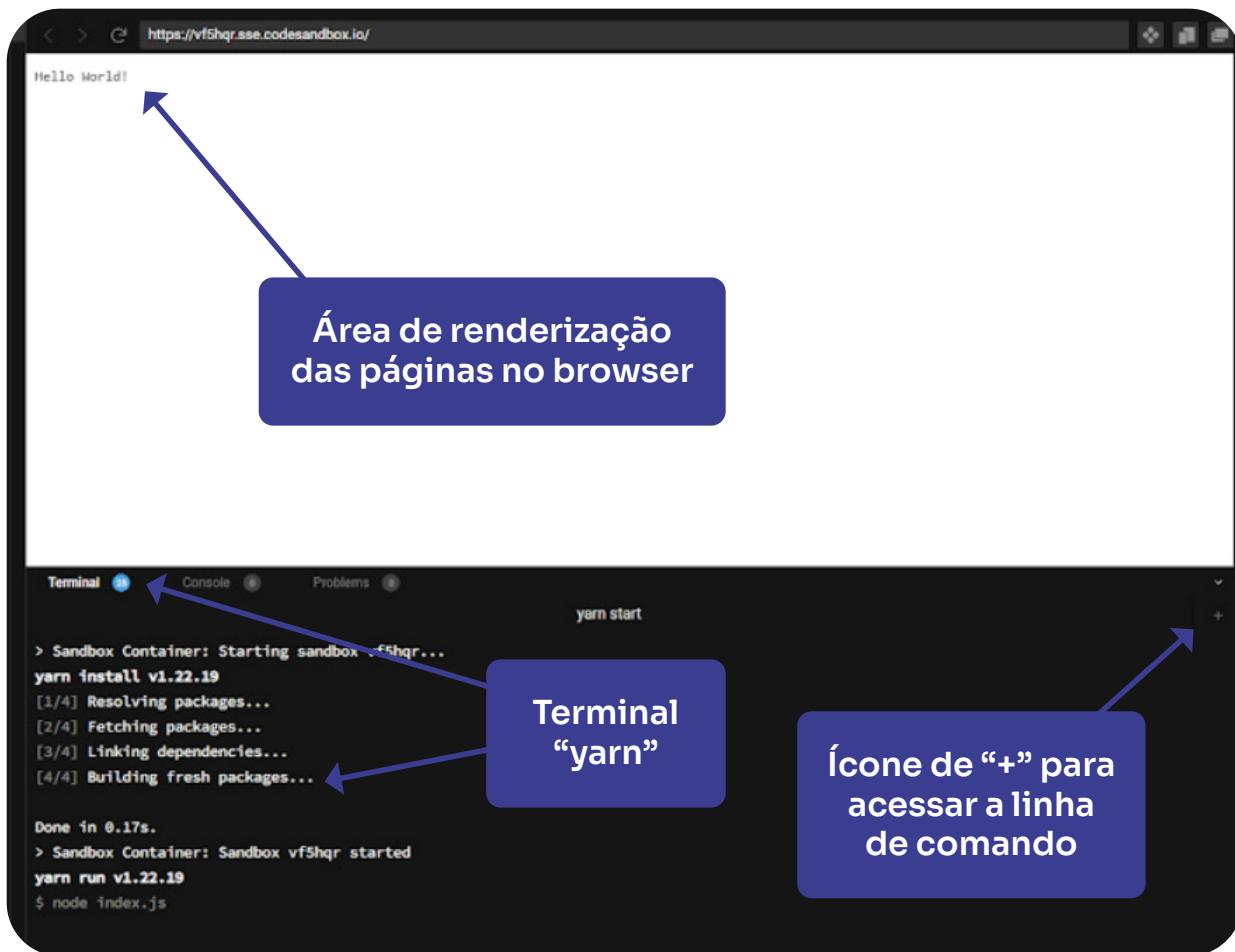
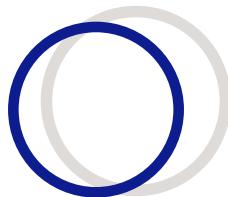


Figura 4 - Principais áreas da plataforma CodeSandbox

Fonte: do Autor (2022)

Confira um outro exemplo de aplicação desenvolvida utilizando o Node.js. O exemplo mais comum do Node.js é a criação de um servidor web. Após a criação do projeto na plataforma SandBox, apague o conteúdo do código-fonte de exemplo do arquivo “index.js” e insira o código apresentado na sequência:



```
const http = require("http");

const hostname = "127.0.0.1";
const port = "4000";

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader("Content-Type", "text/plain");
    res.end("Oi mundo!");
});

server.listen(port, hostname, () => {
    console.log("Servidor está rodando no endereço: " + hostname +
    " e na porta: " + port);
});
```



Não se esqueça de que todas as alterações realizadas nos arquivos criados na plataforma CodeSandbox precisam ser salvas antes utilizando o comando Ctrl + S. Caso contrário, as mudanças não serão refletidas.

Este código inclui primeiro o módulo “http” do Node.js. Assim, esta biblioteca padrão do Node.js é incrível, pois inclui um suporte de primeira classe para rede. O método “createServer()” do módulo “http” é responsável por criar um novo servidor HTTP.

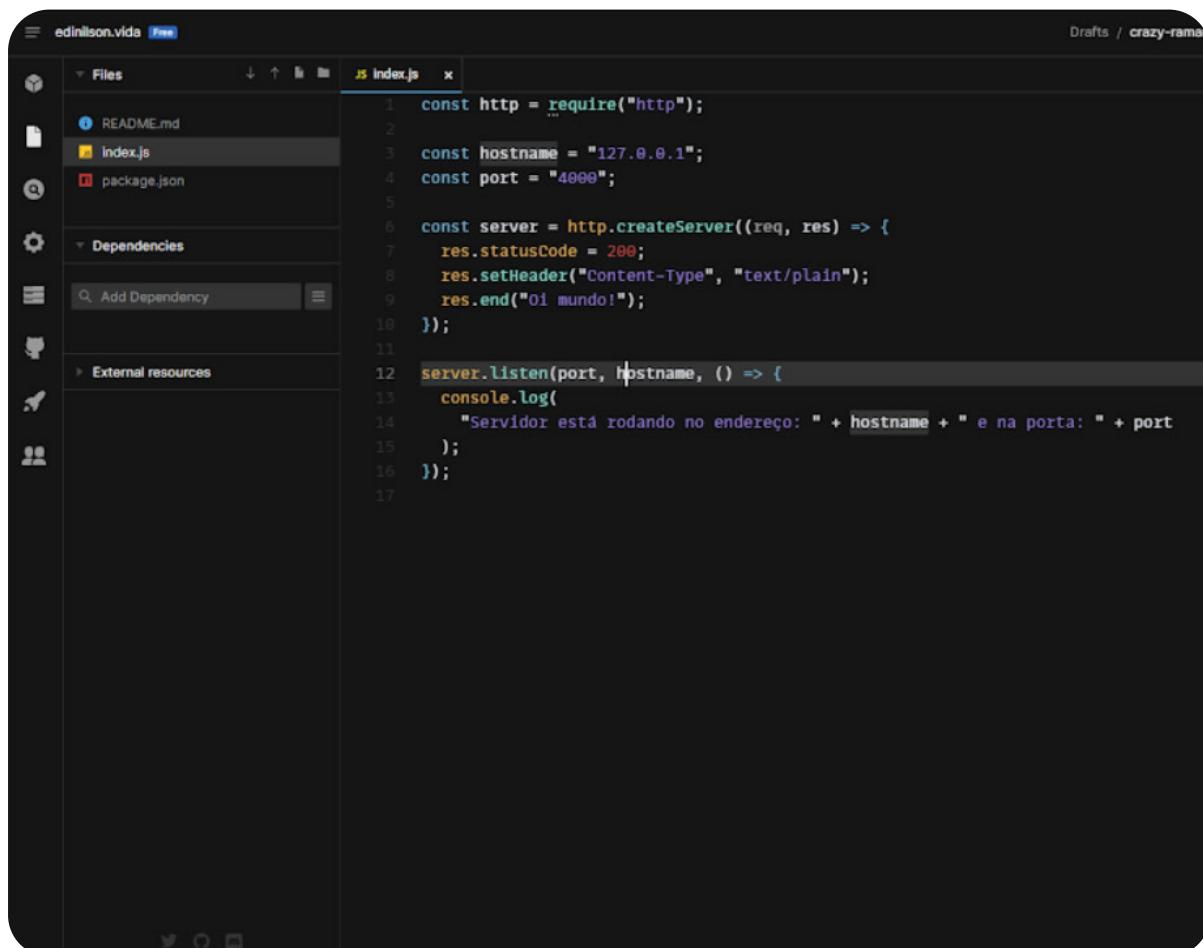
O servidor é configurado para escutar (executar) na porta 4000 e no “hostname” especificados, que no caso é “127.0.0.1”. Quando o servidor está pronto, a função para exibir a mensagem é chamada e, posteriormente, aparece que o servidor está rodando.

Sempre que uma nova solicitação é recebida, o evento da solicitação é chamado, fornecendo dois objetos: uma solicitação (“req”, de *request*) e uma resposta (“res”, de *response*). Esses dois objetos são essenciais para lidar com a chamada HTTP.

O primeiro fornece os detalhes da solicitação. Neste exemplo simples, isto não é usado, mas você poderia acessar os cabeçalhos de solicitação e os dados da solicitação. O segundo é usado para retornar os dados para o chamador.

Neste caso, com “res.statusCode = 200”, definimos a propriedade do código de status para 200, para indicar uma resposta bem-sucedida. Definimos o cabeçalho “Content-Type” com “res.setHeader(‘Content-Type’, ‘text/plain’)” e terminamos fechando a resposta, acrescentando o conteúdo como um argumento para finalizar utilizando: “res.end(“Oi mundo!”);”.

Confira, na imagem seguinte, um exemplo de um projeto criado utilizando a plataforma CodeSandbox, conforme os passos apresentados anteriormente. Observe que, após salvar as alterações, a mensagem “Oi mundo!” é exibida na lateral direita da plataforma.



The screenshot shows the CodeSandbox interface with a dark theme. On the left, there's a sidebar with icons for Files, Dependencies, and External resources. The main area shows the code for `index.js`:

```
const http = require("http");
const hostname = "127.0.0.1";
const port = "4000";

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Oi mundo!");
});

server.listen(port, hostname, () => {
  console.log(
    "Servidor está rodando no endereço: " + hostname + " e na porta: " + port
);
});
```

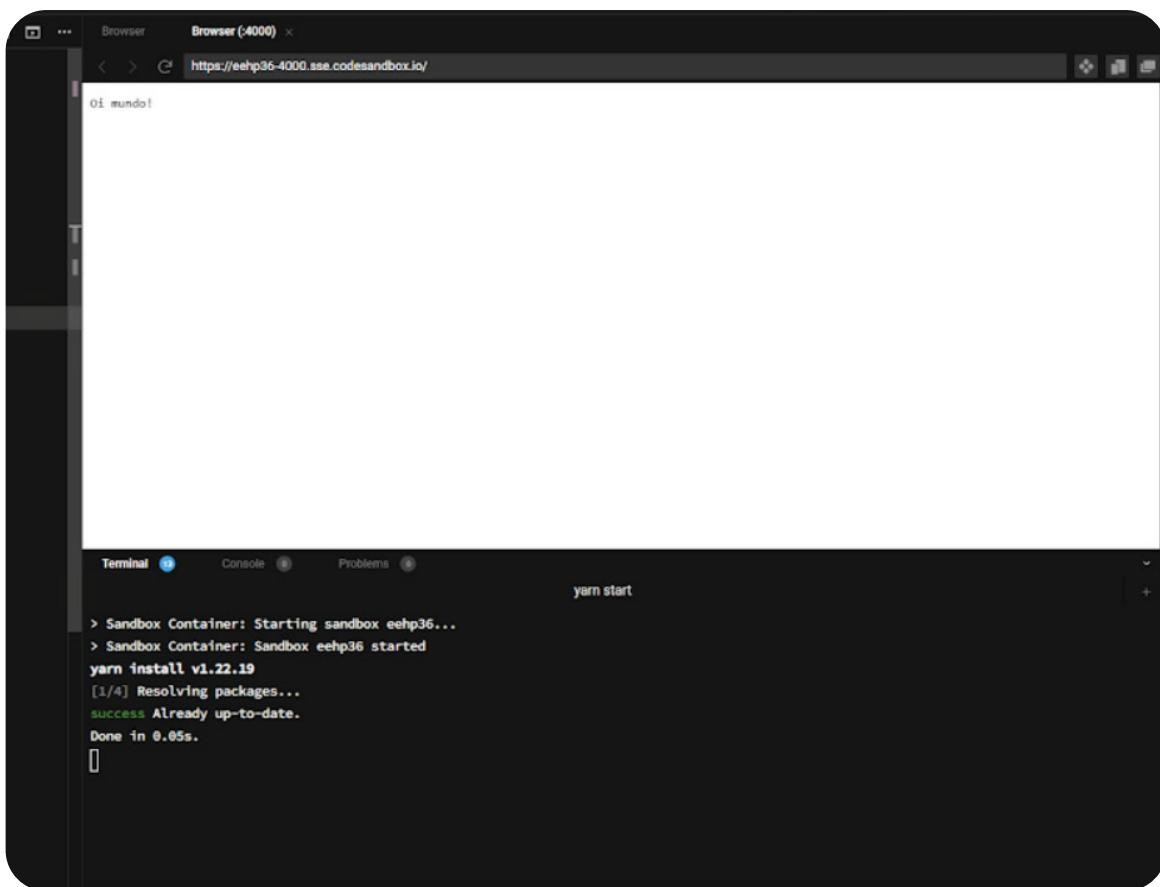
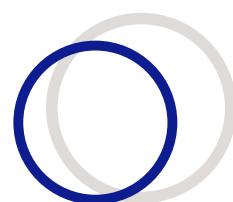


Figura 5 - Exemplo de um projeto de servidor utilizando a Plataforma CodeSandbox

Fonte: do Autor (2022)

Uma outra forma de executar este código é a seguinte: salve o código-fonte no arquivo “index.js” em um projeto na plataforma CodeSandbox, conforme já foi explicado anteriormente. Posteriormente, para executá-lo, clique na opção “+”, localizada na parte inferior do canto direito da aba “Terminal”. Na sequência, digite o comando “node index.js” e aguarde o processamento. Veja que a mensagem “Olá mundo” foi renderizada no browser e a mensagem “Servidor está rodando no endereço: 127.0.0.1 e na porta: 4000” foi exibida na área “Terminal”. Ao executar um programa no “Terminal”, você pode fechá-lo com o comando “Ctrl + C”. Todo esse processo é exibido na imagem seguinte.



```

const http = require("http");
const hostname = "127.0.0.1";
const port = 4000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Olá mundo!");
});

server.listen(port, hostname, () => {
  console.log(`Servidor está rodando no endereço: ${hostname} e na porta: ${port}`);
});

```

Terminal

```

sandbox@sse-sandbox-eepg36:/sandbox$ node index.js
Servidor está rodando no endereço: 127.0.0.1 e na porta: 4000
Olá mundo!

```

Figura 6 - Exibição de mensagens de interação

Fonte: do Autor (2022)

Introdução ao *framework Express.js*

Node.js é uma plataforma de baixo nível, e, para tornar as coisas mais fáceis e interessantes para os desenvolvedores, milhares de bibliotecas foram construídas sobre o Node.js.

Muitas delas foram estabelecidas ao longo do tempo como opções populares. Uma delas é considerada muito relevante, sobre a qual iremos aprender, que é o *framework* Express.js, lançado no mercado em maio de 2010. O Express.js é um dos *frameworks* mais simples, porém poderoso, para criar um servidor web. Sua abordagem minimalista, focada nas características essenciais de um servidor, é a chave para seu sucesso (EXPRESSJS, s. d.).

Express.js é um *framework* rápido e leve usado principalmente para desenvolvimento de aplicativos da web e desenvolvedores Node.js em todo o mundo. O Express.js fornece todos os recursos de aplicativos da web em complemento com os recursos do Node.js.

O Express.js suporta JavaScript, que é uma linguagem amplamente usada, muito fácil de aprender e também suportada em todos os lugares. Portanto, se você já conhece JavaScript, será muito fácil desenvolver aplicações usando Express.js. Esse *framework* fornece um roteamento simples para solicitações feitas por clientes. Ele também fornece um *middleware* responsável por tomar decisões para dar as respostas corretas para as solicitações feitas pelo cliente (EXPRESSJS, s. d.).





Dica

Middleware significa, literalmente, qualquer coisa que você coloca no meio de uma camada do software e outra. O *middleware* representa funções executadas durante o ciclo de vida de uma solicitação ao servidor Express.js. Cada *middleware* tem acesso às requisições e respostas ao HTTP (*Hypertext Transfer Protocol*) para cada rota (ou caminho) em que está anexado. O próprio Express.js é responsável pelas funções de *middleware*. Além disso, o *middleware* pode encerrar a solicitação HTTP ou passá-la para outra função de *middleware* usando o comando “`next()`”. Esse “encadeamento” de *middleware* permite compartimentar código e criar *middleware* reutilizável.



O desempenho do Node.js e a facilidade de codificação usando o Express.js são os recursos mais populares admirados pelos desenvolvedores de aplicativos da web. Com a ajuda do Express.js, você pode criar facilmente diferentes tipos de aplicativos da web em um curto período de tempo. Esta seção dará a você uma visão geral dos fundamentos do Express.js que o ajudarão a começar a usá-lo.

Conhecendo o Express.js

Em termos gerais, o Express.js é um *framework* de aplicativo da web construído sobre o Node.js. Ele fornece uma interface mínima com todas as ferramentas necessárias para construir uma aplicação web. O Express.js adiciona flexibilidade a um aplicativo através de uma enorme variedade de módulos disponíveis no “npm” ou “yarn”, os quais você pode conectar diretamente ao Express.js, conforme a necessidade. O Express.js ajuda no gerenciamento fácil do fluxo de dados entre o servidor e as rotas nos aplicativos do lado do servidor.

O Express.js é o principal responsável por lidar com a parte de *back-end* na *stack* de tecnologia (ou pilha de tecnologia) MEAN. Mean Stack é a pilha de software JavaScript de código aberto que é muito usada para criar sites dinâmicos e aplicativos da web no mercado. Neste caso, a sigla MEAN significa M de MongoDB (banco de dados não relacional), E de Express.js (*framework back-end*), A de Angular (*framework front-end*) e N de Node.js (plataforma de aplicação). Veja a definição desses componentes na imagem seguinte.

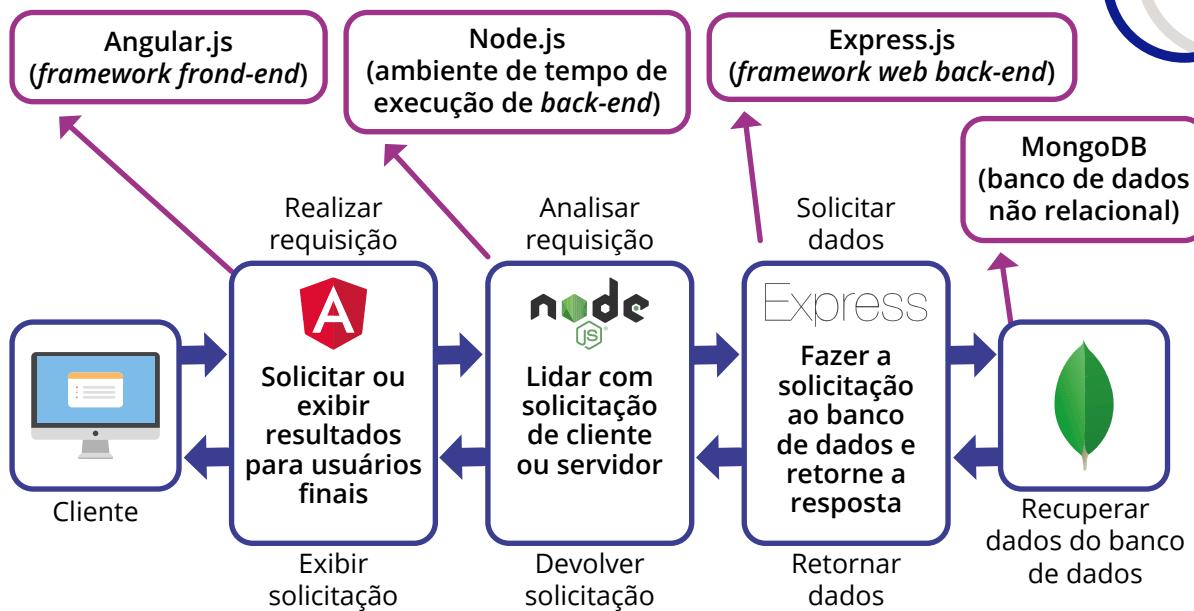


Figura 7 - Representação dos componentes que fazem parte da MEAN stack

Fonte: adaptada de Singla/Net Solutions (2019)

Assim, o Express.js é a parte de *back-end* do MEAN responsável por gerenciar roteamento, sessões, solicitações HTTP, tratamento de erros etc. Dito isso, vamos agora avançar e descobrir os vários recursos desse *framework*. Abaixo, apresentamos alguns dos recursos mais importantes do Express.js (EXPRESSJS, s. d.):

- › Acelera o ritmo de desenvolvimento de um aplicativo da web;
- › Ajuda na criação de aplicativos móveis e da web de tipos de página única (*Single Page Applications – SPA*), várias páginas e híbridos;
- › Segue a arquitetura *Model-View-Controller* (Modelo-Visão-Controlador – MVC);
- › Facilita o processo de integração com bancos de dados, como MongoDB, Redis e MySQL;
- › Define um *middleware* de tratamento de erros;
- › Ajuda a simplificar as etapas de configuração e personalização do aplicativo.

Essas foram algumas das principais vantagens do uso do Express.js no desenvolvimento de aplicativos da web. Agora, veremos como instalar o Express.js utilizando a plataforma CodeSandbox. Para ver isso em ação, crie um novo projeto usando o modelo oficial do Node.js, chamado de “Node HTTP Server”, conforme passos já apresentados anteriormente.

Para instalar o Express.js em um projeto criado na plataforma CodeSandbox, primeiro, você precisa acessar a área “Dependencies” e digitar o nome do pacote a ser instalado, que, neste caso, é “express”. Serão apresentadas algumas sugestões de pacotes, então clique na primeira opção: “express” e aguarde o processo de instalação. Veja detalhes na imagem seguinte.

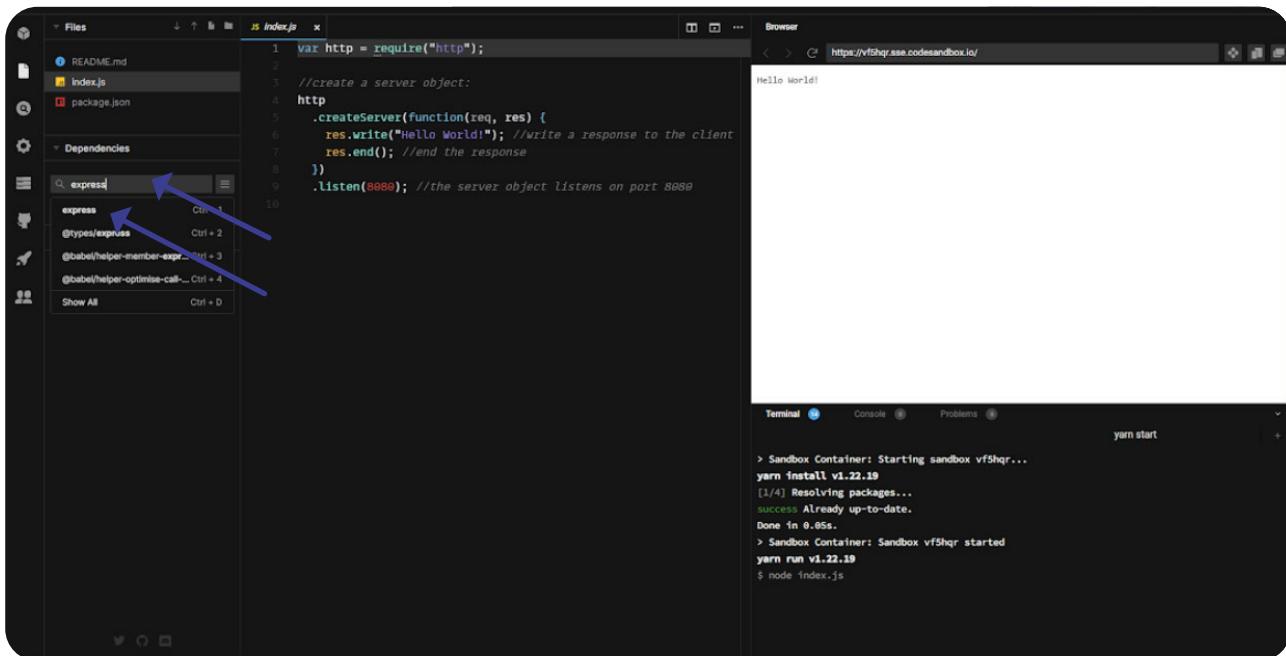


Figura 8 - Processo de instalação de dependências

Fonte: do Autor (2022)

Assim que o processo de instalação finalizar, observe que o módulo “express” foi adicionado ao seu projeto utilizando a versão “4.18.1”, conforme apresentado na imagem seguinte. Essa versão irá variar no momento da instalação, pois esse pacote sofre atualizações constantemente.

The screenshot shows the CodeSandbox interface. On the left, the 'Files' sidebar displays a file tree with 'index.js' selected. The code editor shows the following JavaScript code:

```

1 var http = require("http");
2
3 //create a server object:
4 http
5   .createServer(function(req, res) {
6     res.write("Hello World!"); //write a response to the client
7     res.end(); //end the response
8   })
9   .listen(8080); //the server object listens on port 8080
10

```

A blue arrow points from the word 'express' in the code to the 'Dependencies' section of the sidebar, which lists 'express' version 4.18.1. To the right, the 'Browser' window shows the output 'Hello World!' and the 'Terminal' window shows the command 'yarn start' being run.

Figura 9 - Demonstração de uma dependência instalada

Fonte: do Autor (2022)

Caso haja algum problema com o servidor, recomendamos que as opções “Restart Sandbox” e “Restart Server”, localizadas no menu “Server Control Panel”, sejam acionadas. Dessa forma, o que você pode fazer no CodeSandbox quando o servidor de desenvolvimento ficar em um estado inconsistente é reiniciá-lo, clicando em “Restart Sandbox”, que é mais rápido. Além, disso, há um processo mais demorado, que é o de reiniciar o contêiner usando o comando “Restart Server”. Veja, na imagem seguinte, onde essas opções estão localizadas.

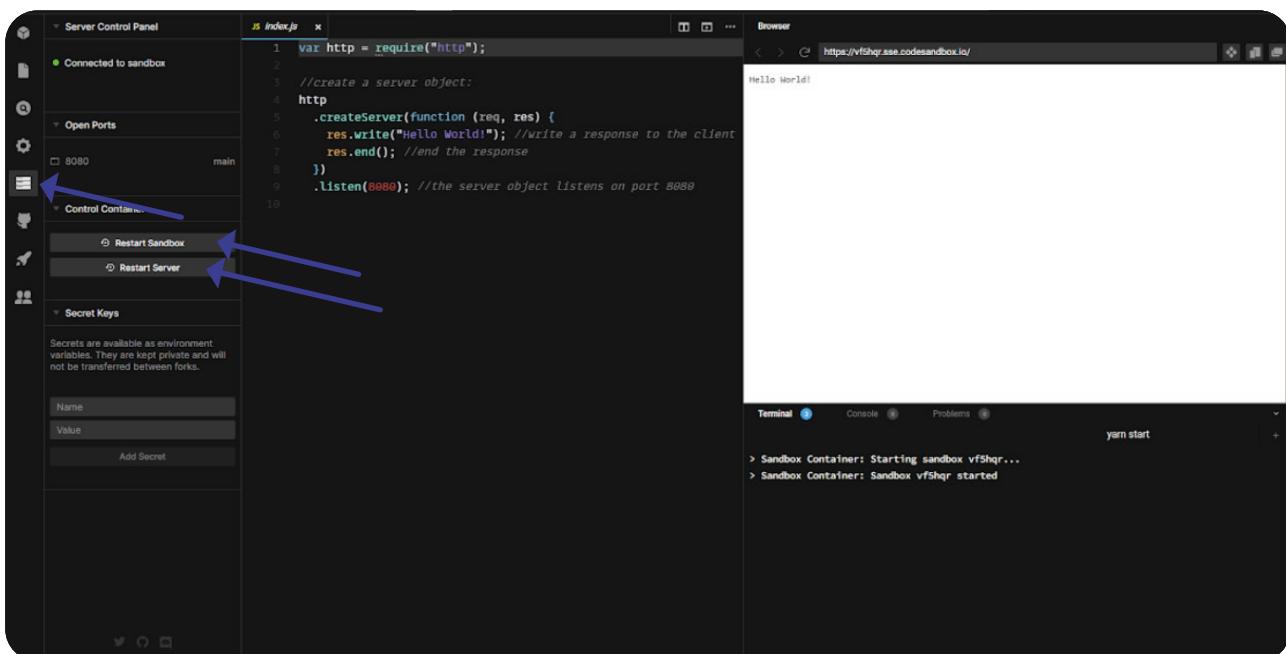


Figura 10 - Painel de gerenciamento do servidor

Fonte: do Autor (2022)

Outra informação importante: o servidor da plataforma CodeSandbox entra em hibernação automaticamente após alguns minutos de inatividade, conforme apresentado na imagem seguinte.

The screenshot shows the CodeSandbox interface. On the left, the 'Server Control Panel' sidebar indicates 'Sandbox hibernated'. The main area displays the 'index.js' code for an Express.js application. The terminal window at the bottom shows the command 'yarn start' was run, and the output includes logs for starting the sandbox and running the application. The browser window on the right shows the URL <https://vfshqr.sse.codesandbox.io/> with the message 'Seja bem-vindo à rota principal (/)!'. A blue arrow points from the 'Sandbox hibernated' status in the sidebar to the message in the browser window.

```

    1 var express = require("express");
  2 const app = express();
  3 app.use(express.json());
  4
  5 app.get("/", function (req, res) {
  6   console.log("Requisição GET recebida para a rota URI /");
  7   res.send("Seja bem-vindo à rota principal (/)!");
  8 });
  9
 10 app.post("/adicionar", function (req, res) {
 11   console.log("Requisição POST recebida para a rota URI /adicionar");
 12   res.send("Seja bem-vindo à rota de adicionar (/adicionar)!");
 13 });
 14
 15 app.put("/atualizar", function (req, res) {
 16   console.log("Requisição PUT recebida para a rota URI /atualizar");
 17   res.send("Seja bem-vindo à rota de atualizar (/atualizar)!");
 18 });
 19
 20 app.delete("/deletar", function (req, res) {
 21   console.log("Requisição DELETE recebida para a rota URI /deletar");
 22   res.send("Seja bem-vindo à rota de deletar (/deletar)!");
 23 });
 24
 25 app.get("/Listar", function (req, res) {
 26   console.log("Requisição GET recebida para a rota URI /Listar");
 27   res.send("Seja bem-vindo à rota de listagem (/Listar)!");
 28 });
 29
 30 //A porta é uma variável de ambiente
 31 const porta = process.env.PORT || 8080;
 32 app.listen(porta, () =>
 33   console.log(`Servidor inicializado na porta: ${porta}`));
 34 );
 35
  
```

Figura 11 - Mensagem de hibernação do servidor

Fonte: do Autor (2022)

Caso identifique a indisponibilidade do servidor, basta reiniciar a aplicação pelo menu “Server Control Panel”, conforme foi explicado anteriormente, ou simplesmente atualize a página do browser.

Agora que você já terminou o processo de instalação, vamos começar a utilizar o Express.js. Na sequência, apresentaremos os conceitos fundamentais da estrutura Express. O primeiro tópico que abordaremos nos fundamentos do Express.js é roteamento (*routing*).

Roteamento no Express

Roteamento refere-se ao processo de determinar um comportamento específico de um aplicativo. Ele é usado para definir como um aplicativo deve responder a uma solicitação de cliente para uma determinada rota, caminho ou URI (*Uniform Resource Identifier*) junto com um método de solicitação HTTP específico. Cada rota pode conter mais de uma função de manipulador, que é executada quando o usuário procura uma rota específica. Abaixo, segue a estrutura do roteamento utilizada no Express.js (EXPRESSJS, s. d.):

```
app.METHOD(PATH, HANDLER)
```

Confira a definição de cada componente da expressão:

- › “app” é apenas uma instância do módulo Express.js. Você pode usar qualquer variável de sua escolha.
- › “METHOD” é um método de solicitação HTTP, como GET, POST, PUT, DELETE etc.
- › “PATH” é a rota para o servidor para uma página da web específica.
- › “HANDLER’ (manipulador) é a função de retorno de chamada que é executada quando a rota correspondente é encontrada.

A seguir, temos um exemplo mostrando o uso de alguns dos principais métodos HTTP.

```
var express = require("express");
const app = express();
app.use(express.json());

app.get("/", function (req, res) {
    console.log("Requisição GET recebida para a rota URI /");
    res.send("Seja bem-vindo à rota principal (/)!");
});

app.post("/adicionar", function (req, res) {
    console.log("Requisição POST recebida para a rota URI /adicionar");
    res.send("Seja bem-vindo à rota de adicionar (/adicionar)!");
});

app.put("/atualizar", function (req, res) {
    console.log("Requisição PUT recebida para a rota URI /atualizar");
    res.send("Seja bem-vindo à rota de atualizar (/atualizar)!");
});
```

```

app.delete("/deletar", function (req, res) {
    console.log("Requisição DELETE recebida para a rota URI /deletar");
    res.send("Seja bem-vindo à rota de deletar (/deletar)!");
});

app.get("/listar", function (req, res) {
    console.log("Requisição GET recebida para a rota URI /listar");
    res.send("Seja bem-vindo à rota de listagem (/listar)!");
});

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

```

Dessa forma, existem quatro métodos HTTP principais que podem ser fornecidos na solicitação. Esses métodos ajudam a especificar a operação solicitada pelo usuário e são utilizados no estilo arquitetural REST (*Representational State Transfer*). Veja, na sequência, quais são estes métodos acompanhados de suas explicações (RUBY, 2007; SAUDATE, 2014):

- **Solicitação HTTP GET:** Usamos o método GET para ler ou recuperar um recurso. Uma mensagem de sucesso GET retorna uma resposta contendo as informações solicitadas. Em um exemplo de uma aplicação, usamos o método GET para recuperar o clima atual para uma cidade específica.
- **Solicitação HTTP POST:** Usamos o método POST para criar um novo recurso. Uma solicitação POST requer um corpo (*body*) no qual você define os dados da entidade a ser criada. Em um exemplo de uma aplicação meteorológica, poderíamos usar um método POST para adicionar dados meteorológicos sobre uma nova cidade.
- **Solicitação HTTP PUT:** Usamos o método PUT para modificar um recurso. O método PUT atualiza todo o recurso com dados que são passados na carga do corpo da solicitação HTTP. Se não houver recurso que corresponda à solicitação, ele criará um novo recurso. Em nosso aplicativo meteorológico, podemos usar o método PUT para atualizar todos os dados climáticos sobre uma cidade específica.

› **Solicitação HTTP DELETE:** Usamos o método DELETE para excluir um recurso. Em nosso aplicativo de clima, poderíamos usar o método DELETE para excluir uma cidade que não queiramos mais rastrear, por algum motivo.

As informações relacionadas sobre a construção de APIs utilizando o estilo arquitetural REST e que tem relação com os métodos HTTP serão apresentadas com mais detalhes posteriormente. Na sequência, veja os procedimentos básicos para realizar testes de rotas criadas no exemplo anterior baseado no Express.js utilizando a ferramenta Postman.

Após inicializar a aplicação apresentada anteriormente utilizando o módulo “express” na plataforma CodeSandbox, iremos realizar o teste utilizando a ferramenta online Postman. Postman é um aplicativo de desktop e web que permite fazer solicitações a uma API a partir de uma interface gráfica de usuário. Antes de começar a usar o Postman, você precisará ter o acesso à ferramenta. Ter uma conta gratuita do Postman permite sincronizar e fazer backup do seu trabalho para que você possa acessá-lo em diferentes máquinas. Com uma conta do Postman, você também pode colaborar com outras pessoas em seus projetos de API. Antes de se inscrever em uma conta do Postman, acesse o aplicativo da web pelo seguinte link: <https://www.postman.com>. O Postman solicita que você faça login ou cadastre-se, então selecione um dos dois botões “Sign Up For Free” localizados na página. Você pode se inscrever usando um endereço de e-mail ou sua conta do Google.

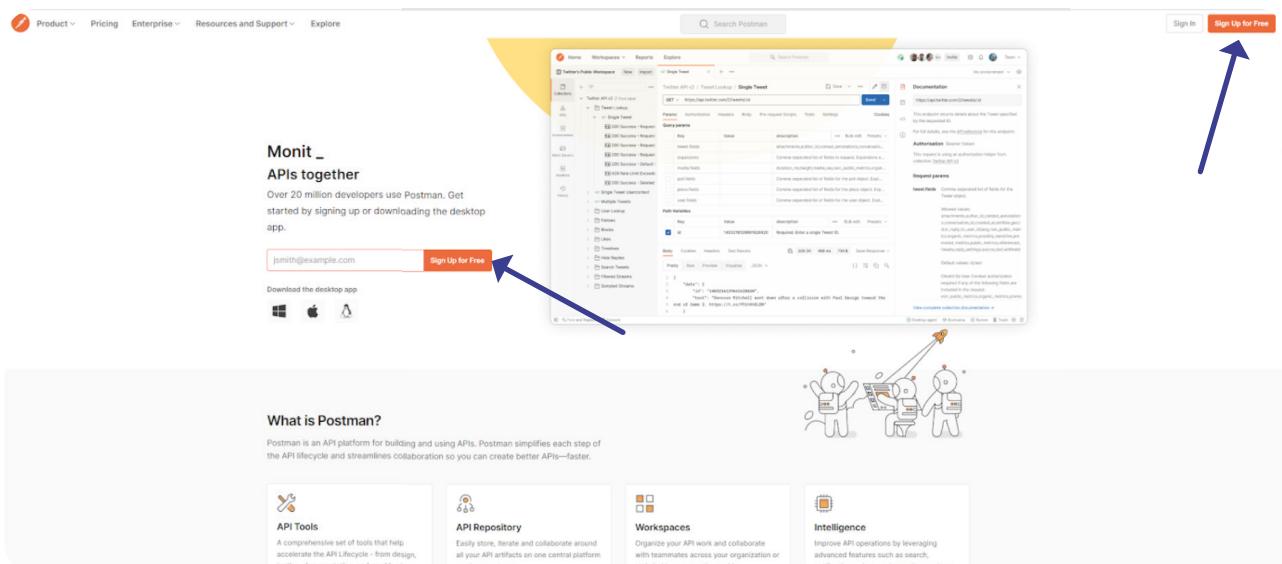
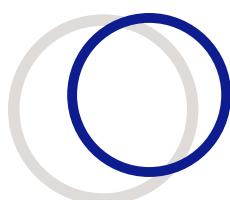


Figura 12 - Criação de conta na plataforma Postman

Fonte: do Autor (2022)



Como você irá perceber, o processo de cadastro é bem simples. Siga os passos apresentados em tela para finalizar o cadastro e ter acesso à plataforma. Após realizar a autenticação na plataforma, caso você não esteja na área de testes, você deverá acessar o menu “Home” e, depois, a área “My Workspace” para iniciar os testes, conforme mostra a imagem seguinte.

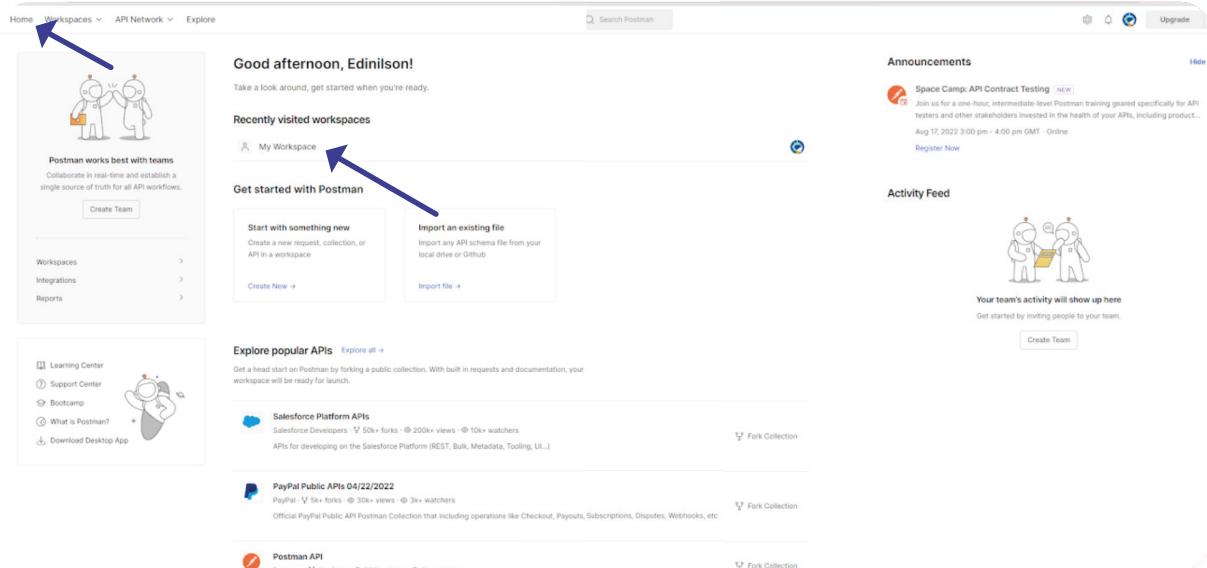


Figura 13 - Acesso à área “My Workspace”

Fonte: do Autor (2022)

Ao acessar a página inicial da área “My Workspace”, clique no botão “+” para abrir uma guia para inserir os URIs de testes, bem como definir quais serão os métodos HTTP utilizados para acessar a nossa aplicação Express, conforme apresentado na imagem.

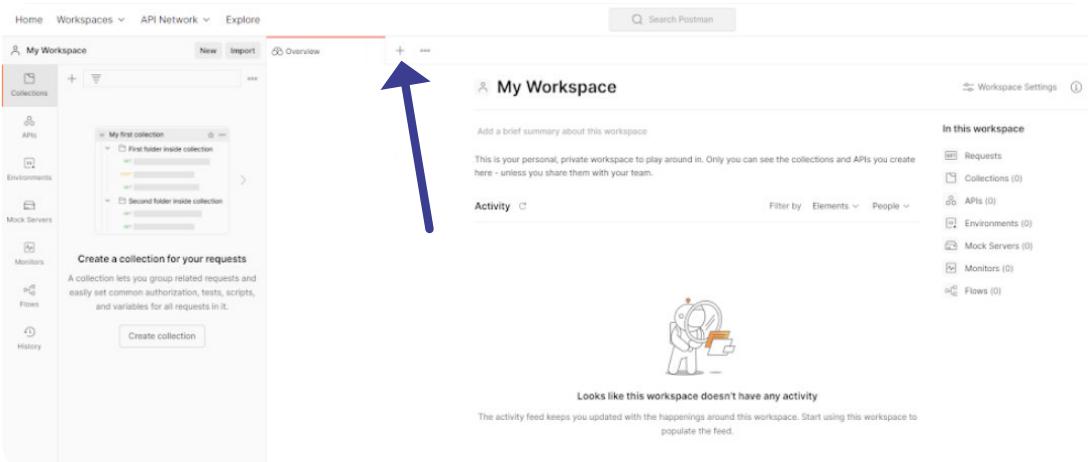


Figura 14 - Criação de uma guia para realização de testes

Fonte: do Autor (2022)

Inicialmente, para realizar os testes dos métodos HTTP, você precisa descobrir qual é o link de acesso à aplicação hospedada na plataforma CodeSandbox. Para isso, você deve acessar o projeto criado nessa plataforma e copiar o link exibido na seção do browser, conforme apresentado na imagem seguinte.

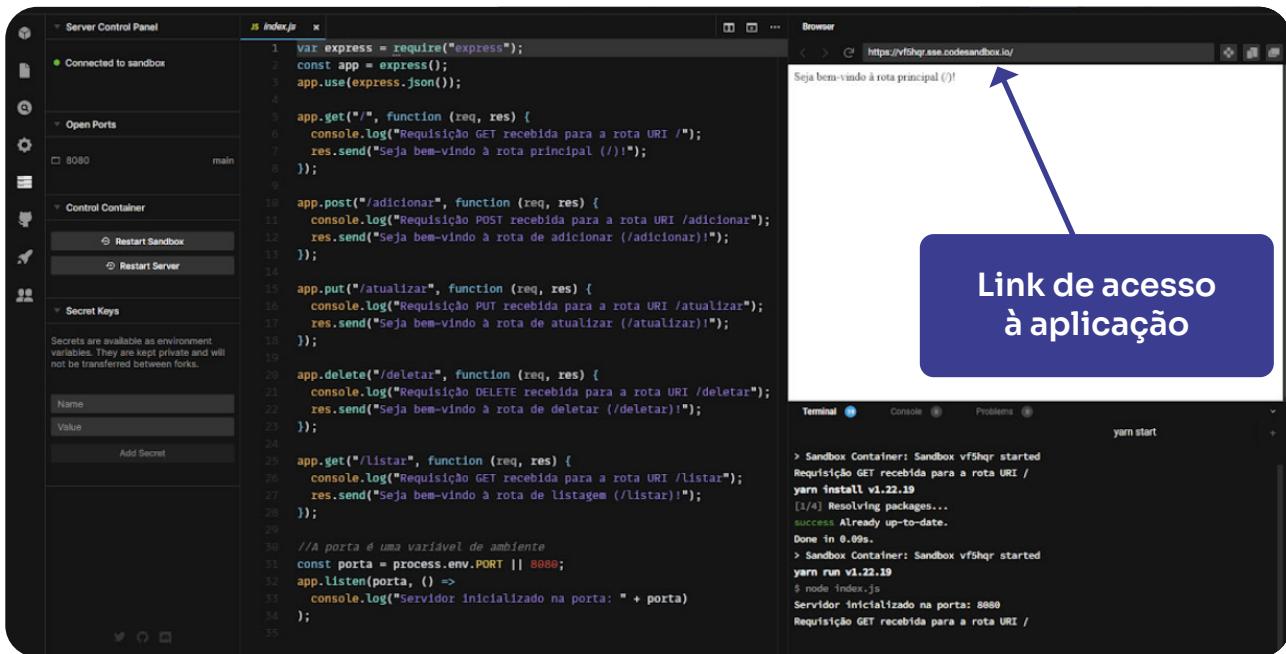


Figura 15 - Captura do link de acesso à aplicação

Fonte: do Autor (2022)

No exemplo criado, temos o seguinte link: <https://vf5hqr.sse.codesandbox.io>. Perceba que este link irá variar de projeto para projeto. Se acessarmos nossa aplicação de outro navegador, veremos a seguinte saudação da nossa aplicação na página inicial: "Seja bem-vindo à rota principal (/)!". Faça este teste: cole este link em outra aba do seu navegador e veja o resultado!

Agora que temos o link de acesso à nossa aplicação, podemos finalmente iniciar alguns testes na ferramenta Postman. No campo "Enter request URL", insira o link de acesso do projeto criado na plataforma CodeSandbox. Entretanto, você deve acrescentar o URI "/adicionar" no final do link, de forma idêntica a como está definido na rota "app.POST" no projeto criado na plataforma CodeSanbox. Veja um exemplo: <https://vf5hqr.sse.codesandbox.io/adicionar>. E no método HTTP, escolha POST e clique no botão "Send". Aguarde a execução da requisição e veja na área "Body" a resposta enviada pelo servidor. A imagem a seguir apresenta o processo a ser executado.

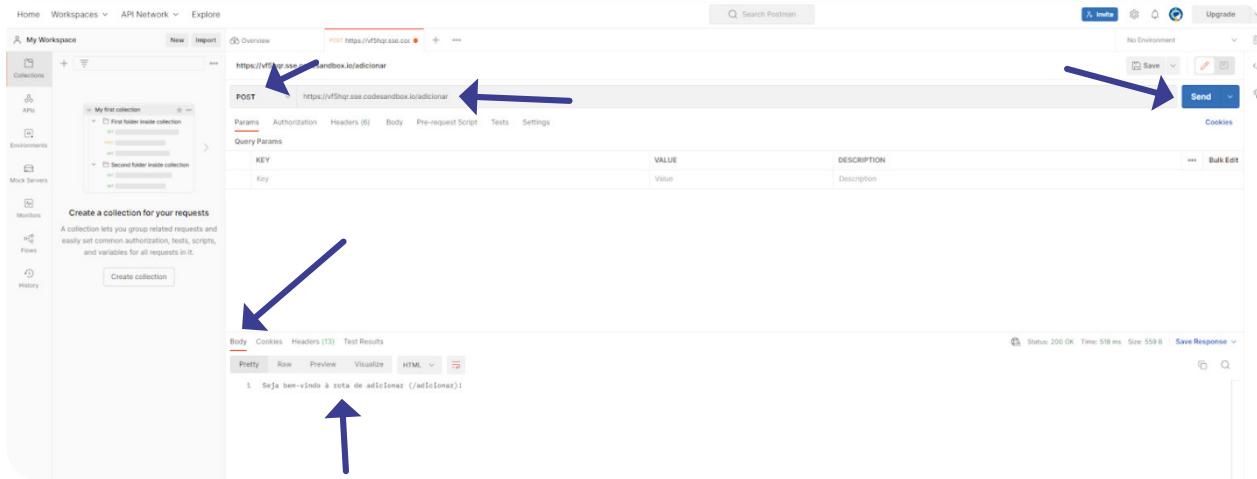


Figura 16 - Teste de rota utilizando o método HTTP POST

Fonte: do Autor (2022)

Caso deseje testar as outras rotas, refaça o procedimento apenas alterando o URI e o método HTTP na área de testes do Postman. Veja um exemplo:

- › Para o URL <https://vf5hqr.sse.codesandbox.io/atualizar>, escolha PUT para o método HTTP.
- › Para o URL <https://vf5hqr.sse.codesandbox.io/deletar>, escolha DELETE para o método HTTP.
- › Para o URL <https://vf5hqr.sse.codesandbox.io/listar>, escolha GET para o método HTTP.

Na próxima seção deste estudo, apresentaremos a você os *middlewares* no Express.js.

Middleware no Express.js

No Express.js, as funções de *middleware* são as funções que têm acesso aos objetos de solicitação e resposta junto com a próxima função presente no ciclo de solicitação-resposta do aplicativo. Essas funções são capazes de realizar as tarefas listadas abaixo (EXPRESSJS, s. d.):

- › Execução de qualquer código;
- › Modificação da solicitação e dos objetos de resposta;
- › Encerramento do ciclo de solicitação-resposta de aplicativos;
- › Chamamento do próximo *middleware* presente no ciclo.

Observe que, caso a função atual não finalize o ciclo de solicitação-resposta, ela deve invocar a função “next()” para passar o controle para a próxima função de *middleware* disponível. Caso contrário, a solicitação ficará incompleta. Agora, para entender sobre as funções de *middleware*, apresentamos um pequeno exemplo que captura a data atual, concatena com uma mensagem personalizada e a exibe ao usuário:

```
var express = require("express");
const app = express();

var requisicaoData = function (req, res, next) {
    req.requisicaoData = new Date().toISOString().slice(0, 10);
    next();
};

app.use(requisicaoData);

app.get("/", function (req, res) {
    var mensagemResposta = "Olá pessoal!";
    mensagemResposta +=
        " Esta mensagem foi gerada na data: " + req.requisicaoData;
    res.send(mensagemResposta);
});

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

Ao executar este exemplo na plataforma CodeSandbox, obteremos o seguinte resultado, conforme apresentado na imagem seguinte:

The screenshot shows the CodeSandbox environment. On the left, the 'Files' sidebar displays a project structure with files like README.md, package.json, and yarn.lock. The main area shows the code for 'index.js':

```
1 var express = require("express");
2 const app = express();
3
4 var requisicaoData = Function (req, res, next) {
5   req.requisicaoData = new Date().toISOString().slice(0, 10);
6   next();
7 };
8
9 app.use(requisicaoData);
10
11 app.get("/", function (req, res) {
12   var mensagemResposta = "Olá pessoal!";
13   mensagemResposta +=
14     " Esta mensagem foi gerada na data: " + req.requisicaoData;
15   res.send(mensagemResposta);
16 });
17
18 //A porta é uma variável do ambiente
19 const porta = process.env.PORT || 8080;
20 app.listen(porta, () =>
21   console.log("Servidor inicializado na porta: " + porta)
22 );
```

To the right, the 'Browser' window shows the output of the application at <https://vfshqr.sse.codesandbox.io/>. The page displays the message: "Olá pessoal! Esta mensagem foi gerada na data: 2022-08-05". Below the browser window, the 'Terminal' shows logs indicating the sandbox container started successfully.

Figura 17 - Exemplo de um *middleware*

Fonte: do Autor (2022)

Como você pôde observar, os *middlewares* acessam os objetos de solicitação e resposta HTTP e podem encerrar a solicitação HTTP ou encaminhá-la para processamento adicional para outra função de *middleware*. As funções de *middleware* são anexadas a um ou mais manipuladores de rota em um aplicativo Express.js e são executadas em sequência desde o momento em que uma solicitação HTTP é recebida pelo aplicativo até que uma resposta HTTP seja enviada de volta ao chamador.

Durante seus estudos, você pode acompanhar que o Express.js é um *framework* de aplicativo da web gratuito e de código aberto para Node.js. Além disso, é usado para projetar e construir aplicativos da web de forma rápida e fácil. Como o Express.js requer apenas JavaScript, ele facilita para que programadores e desenvolvedores construam aplicativos web e API sem grandes esforços.

Por fim, você viu os procedimentos básicos para realizar testes de rotas a partir dos métodos HTTP utilizando a ferramenta online Postman. Mais um conteúdo se encerra por aqui, entretanto, você deve acessar outros recursos para continuar sua rotina de estudos! Avance para adquirir novos aprendizados!

REFERÊNCIAS

CLARK, J. **Backend meaning and everything you need to know about it.** Back4App Blog, 28 fev. 2021. Disponível em: <https://blog.back4app.com/?s=backend+meaning>. Acesso em: 28 ago. 2022.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados.** 6. ed. São Paulo (SP): Pearson, 2011.

EXPRESSJS. **Express:** framework web rápido, flexível e minimalista para Node.js. Fundação Node.js, s. d. Disponível em: <http://expressjs.com/pt-br>. Acesso em: 20 jul. 2022.

FERNANDES, H. M. **What is a frontend developer and what does it do?** TechBlog, 16 ago. 2020. Disponível em: <https://marquesfernandes.com/en/technology/what-is-a-frontend-developer-and-what-does-he-do/>. Acesso em: 10 ago. 2022.

GASPAR, L. **Aprenda o que é NPM e como utilizá-lo.** Blog HostGator, 19 fev. 2021. Disponível em: <https://www.hostgator.com.br/blog/o-que-e-npm>. Acesso em: 30 jul. 2022.

HASHDORK. **CodePen vs CodeSandbox vs StackBlitz.** Blog HashDork, 5 out. 2021. Disponível em: <https://hashdork.com/pt/codepen-codesandbox-stackblitz/>. Acesso em: 31 jul. 2022.

NODEJS. **Introduction to Node.js.** OpenJS Foundation, s. d. Disponível em: <https://nodejs.dev/learn/introduction-to-nodejs>. Acesso em: 26 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.

SINGLA, L./NET SOLUTIONS. **MEAN Stack and Startups:** Are they Made for Each Other? DZone - Web Dev Zone Tutorial, 27 fev. 2019. Disponível em: <https://dzone.com/articles/mean-stack-amp-startups-are-they-made-for-each-oth>. Acesso em: 10 ago. 2022.



