

O nosso objetivo

- Criar uma API nos padrões RESTful, que terá um **CRUD**;
- Os endpoints serão criados com os verbos HTTP que correspondem a ação do mesmo;
- As respostas serão baseadas em JSON, retornando também o status correto;
- Aplicaremos validações simples, para simular o 'mundo real';

Crie uma API RESTful com Node.js e MongoDB | CRUD com Node, Express e Mongoose

Uma API (Interface de Programação de Aplicações) é um conjunto de definições e protocolos que permite a comunicação entre diferentes softwares.

Ela define como diferentes componentes de software devem interagir uns com os outros. Em essência, uma API especifica como as partes de um sistema de software devem se comunicar, permitindo que desenvolvedores integrem funcionalidades de um aplicativo em outro.

Aqui estão alguns pontos-chave sobre APIs:

Padrões de Comunicação: Uma API define um conjunto de padrões e regras para comunicação entre diferentes sistemas. Isso pode incluir o formato de dados a ser usado (como JSON ou XML), os métodos de requisição (como GET, POST, PUT, DELETE), autenticação e autorização, entre outros.

Abstração: Uma API fornece uma camada de abstração sobre a implementação interna de um software. Isso permite que os desenvolvedores interajam com o software usando uma interface consistente, sem precisar entender todos os detalhes internos de como ele funciona.

Reutilização de Funcionalidades: APIs permitem que os desenvolvedores reutilizem funcionalidades existentes de outros aplicativos em seus próprios projetos. Isso é especialmente útil em um mundo onde muitos serviços e aplicativos estão interconectados, permitindo que os desenvolvedores construam sobre o trabalho de outros e acelerem o desenvolvimento.

Integração de Sistemas: APIs são amplamente utilizadas para integrar sistemas e aplicativos diferentes. Por exemplo, uma API de pagamento pode ser usada para integrar um sistema de comércio eletrônico com um provedor de serviços de pagamento, permitindo que os clientes façam compras online de forma segura.

Desenvolvimento de Plataformas: Empresas frequentemente disponibilizam APIs para seus produtos e serviços como uma forma de permitir que terceiros desenvolvam aplicativos e extensões que se integrem com sua plataforma. Isso pode levar à criação de um ecossistema rico de aplicativos e serviços em torno de uma plataforma principal.

Em resumo, uma API é uma ferramenta fundamental para permitir a comunicação e integração entre diferentes sistemas e aplicativos de software, proporcionando uma maneira padronizada e eficiente para que eles interajam uns com os outros.

JSON (JavaScript Object Notation) é um formato de dados leve e de fácil leitura utilizado para troca de informações entre sistemas computacionais. Ele é baseado em um subconjunto da linguagem de programação JavaScript, mas é independente de plataforma e pode ser utilizado em várias linguagens de programação.

A estrutura básica do JSON consiste em pares de chave-valor, onde uma chave é uma string que identifica um dado específico e o valor pode ser qualquer tipo de dado válido, como um número, uma string, um booleano, um array ou até mesmo outro objeto JSON. Os pares de chave-valor são separados por vírgulas e os objetos JSON são delimitados por chaves {}.

Aqui está um exemplo simples de um objeto JSON:

```
json
Copiar código
{
  "nome": "João",
  "idade": 30,
  "cidade": "São Paulo",
  "casado": false,
  "filhos": ["Ana", "Pedro"]
}
```

Neste exemplo, temos um objeto com cinco pares de chave-valor:

"nome": "João": A chave é "nome" e o valor é "João", uma string.
"idade": 30: A chave é "idade" e o valor é 30, um número.
"cidade": "São Paulo": A chave é "cidade" e o valor é "São Paulo", uma string.
"casado": false: A chave é "casado" e o valor é false, um booleano.
"filhos": ["Ana", "Pedro"]: A chave é "filhos" e o valor é um array contendo duas strings: "Ana" e "Pedro".

JSON é amplamente utilizado na comunicação entre sistemas distribuídos, especialmente em aplicações web, onde é comumente utilizado para enviar e receber dados entre o cliente e o servidor. Ele é fácil de ler e escrever para humanos, além de ser facilmente interpretável por máquinas.

O Express é um framework web para Node.js, projetado para criar aplicativos da web e APIs de forma rápida e fácil. **É um dos frameworks mais populares para Node.js** e é conhecido por sua simplicidade, flexibilidade e extensibilidade.

Aqui estão algumas características-chave do Express:

Roteamento: O Express simplifica o roteamento de URLs para manipular solicitações HTTP, permitindo definir rotas para diferentes URLs e métodos HTTP.

Middlewares: O Express utiliza um sistema de middlewares que permite executar funções **em uma solicitação antes que ela alcance seu destino final**. Isso é útil para realizar tarefas como análise de corpo da solicitação, autenticação, autorização, e muito mais.

Templates: Embora o Express não venha com um mecanismo de template padrão, ele é flexível o suficiente para trabalhar com vários mecanismos de template populares, como Handlebars, EJS, Pug (antigo Jade) e outros.

Gestão de Requisições e Respostas: O Express facilita a manipulação de requisições e respostas HTTP, permitindo acessar os parâmetros da requisição, o corpo da requisição, definir cabeçalhos de resposta e enviar respostas para o cliente.

Integração com o ecossistema Node.js: O Express é compatível com o vasto ecossistema de módulos Node.js, o que significa que você pode facilmente integrar outros módulos e bibliotecas ao seu aplicativo Express.

Node.js é um ambiente de tempo de execução (runtime) de código aberto, construído sobre o motor JavaScript V8 do Google Chrome. Ele permite que os desenvolvedores executem JavaScript no lado do servidor, em vez de apenas no navegador da web.

Aqui estão algumas características importantes do Node.js:

JavaScript no Servidor: Node.js permite que os desenvolvedores usem **JavaScript** tanto no lado do cliente quanto no lado do servidor. Isso proporciona uma experiência

de desenvolvimento unificada e simplificada para equipes que trabalham em ambas as partes de um aplicativo web.

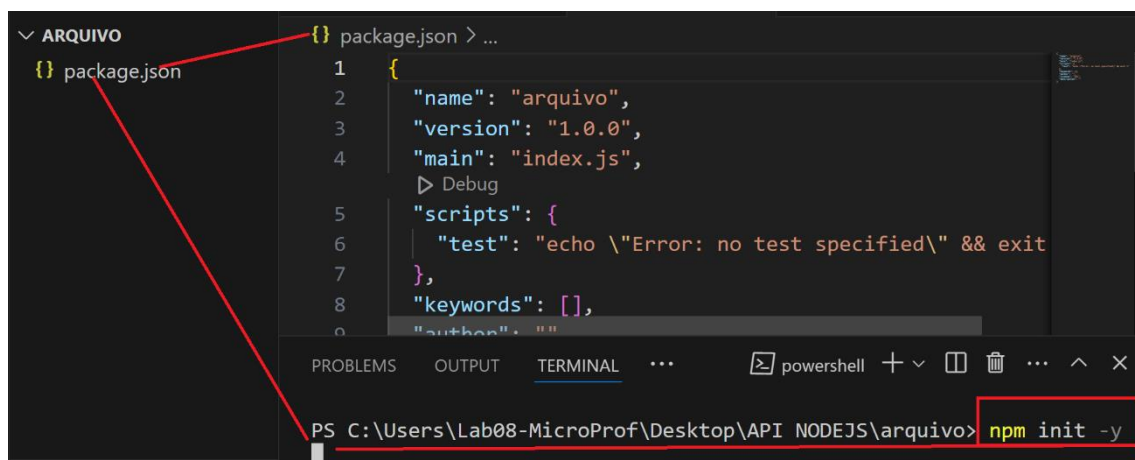
Assíncrono e Orientado a Eventos: Node.js é conhecido por seu modelo de E/S (entrada e saída) não bloqueante e orientado a eventos. Isso significa que, em vez de aguardar operações de E/S (como leitura de arquivos ou chamadas de banco de dados) serem concluídas, **o Node.js continua executando outras tarefas**. Quando uma operação de E/S é concluída, **o Node.js chama uma função de retorno de chamada (callback)** para lidar com o resultado. Esse modelo assíncrono é eficiente para lidar com muitas conexões simultâneas e operações intensivas de E/S.

Módulos: Node.js possui um sistema de módulos que permite aos desenvolvedores organizar seu código em arquivos separados e reutilizáveis. O sistema de módulos do Node.js segue o padrão CommonJS, que simplifica a importação e exportação de funcionalidades entre arquivos.

Ecosistema de Pacotes (npm): Node.js é acompanhado pelo npm (Node Package Manager), um gerenciador de pacotes que permite aos desenvolvedores instalar, compartilhar e gerenciar dependências de projetos de forma eficiente. O npm é o maior ecossistema de pacotes de software do mundo, com milhões de pacotes disponíveis para uso.

Versatilidade: Node.js é altamente versátil e pode ser usado para uma variedade de aplicações, incluindo servidores web, APIs, aplicativos de linha de comando, aplicativos de desktop, ferramentas de automação, jogos e muito mais. Sua eficiência e escalabilidade o tornam uma escolha popular para muitos tipos de projetos.

Em resumo, Node.js é uma plataforma poderosa e popular para desenvolvimento de software, permitindo que os desenvolvedores criem aplicações de alto desempenho e escaláveis usando JavaScript tanto no lado do cliente quanto no lado do servidor.



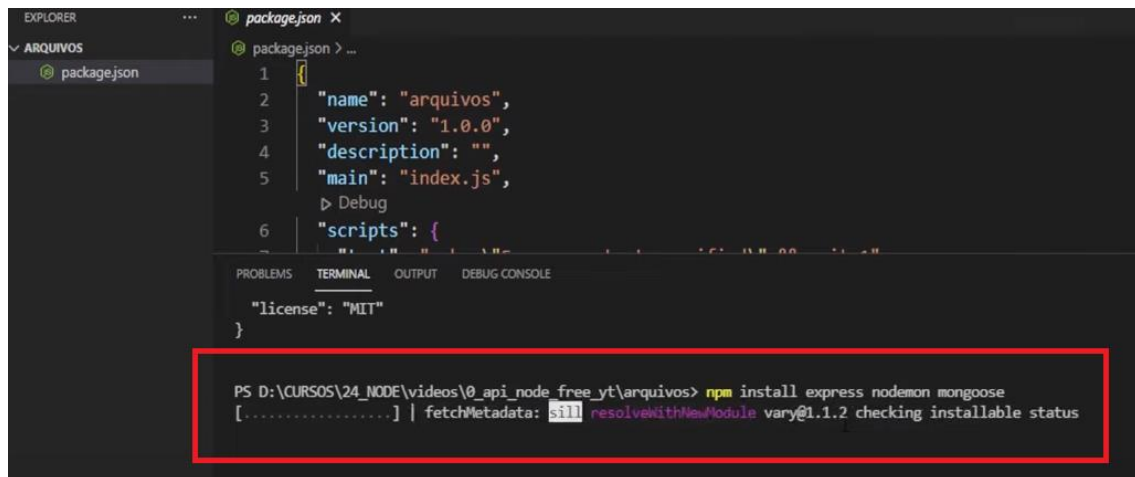
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a file named 'package.json' under a folder named 'ARQUIVO'. A red arrow points from this file to the main editor. The main editor displays the content of 'package.json', which is a JSON object with the following structure:

```
{
  "name": "arquivo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit"
  },
  "keywords": [],
  "author": ""
}
```

Below the editor, the TERMINAL panel is open, showing a PowerShell prompt. The command 'npm init -y' has been entered and is highlighted with a red box. The prompt shows the current directory as 'C:\Users\Lab08-MicroProf\Desktop\API NODEJS\arquivo'.

Criar a pasta APINODEJS digitar npm init -y

Agora vamos puxar os pacotes adicionais

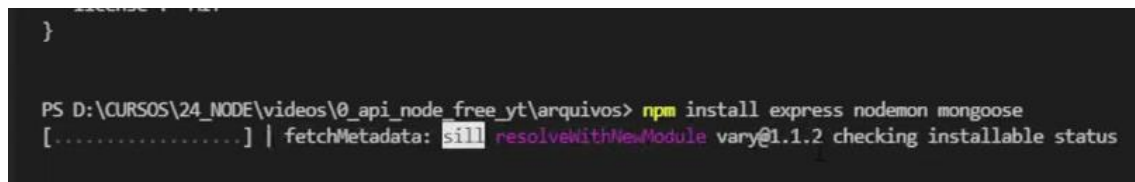


The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows a file named 'package.json'. The main editor displays the content of 'package.json' with the following JSON structure:

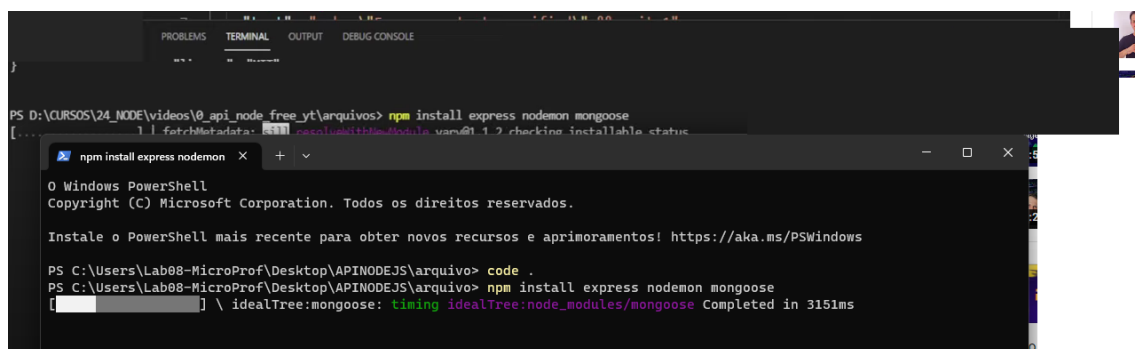
```
{
  "name": "arquivos",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "license": "MIT"
}
```

The TERMINAL panel at the bottom shows the command `npm install express nodemon mongoose` being executed. The output includes the message `[.....] | fetchMetadata: sill resolveWithNewModule vary@1.1.2 checking installable status`. A red rectangle highlights the terminal output.

npm install express nodemon mongose



This screenshot shows a terminal window with the command `npm install express nodemon mongoose` being executed. The output shows the progress of the installation, including the message `[.....] | fetchMetadata: sill resolveWithNewModule vary@1.1.2 checking installable status`.



This screenshot shows the Visual Studio Code interface with the terminal panel open. The command `npm install express nodemon mongoose` has been executed. The output shows the progress of the installation, including the message `[.....] | fetchMetadata: sill resolveWithNewModule vary@1.1.2 checking installable status`. A new window titled 'npm install express nodemon' is also visible in the background.

Adicionar o Pacote

Index.js

```
// config inicial chamar o express vai procurar o módulo
const express = require('express')
const app = express() // Inicializar as apps

//forma de ler JSON UTILIZAR MIDDLEWARES
app.use( //criando o MIDDLEWARES
  express.urlencoded({
    extended: true,
  }),
)

app.use(express.json())

//rota inicial GET pegar algo so servidor endpoint
app.get('/', (req, res) => {

  //mostrar requisição mostrar a resposta que vai ser JSON
  res.json({ message: 'Oi Express meu nome é Thiago'})
})

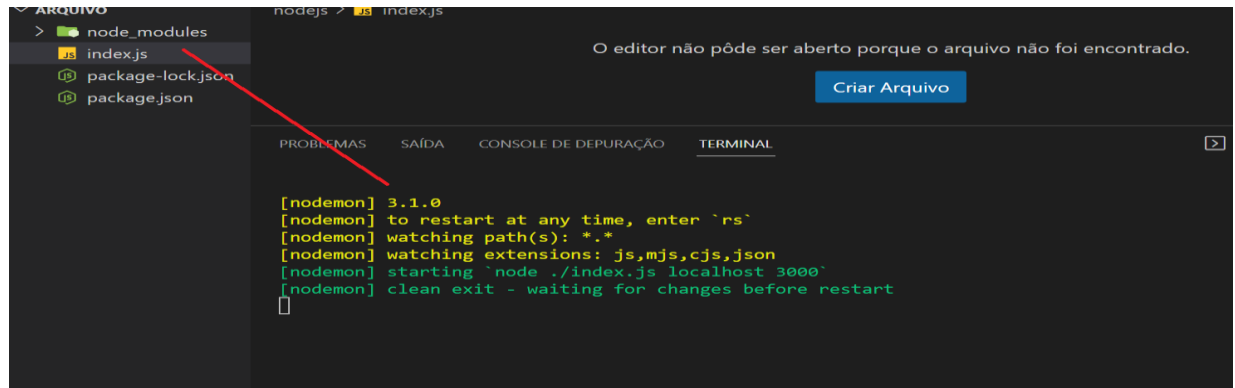
//entregar a porta
app.listen(3000)
```

Package.JSON

```
{
  "name": "arquivo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "nodemon ./index.js localhost 3001"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.19.2",
```

```
"mongoose": "^8.3.1",  
"nodemon": "^3.1.0"  
}  
}
```

Digite NPM START



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

require('express'): require é uma **função** do Node.js usada para carregar módulos. Quando você passa o argumento 'express' para require, está solicitando ao Node.js que **carregue o módulo chamado 'express'**. Em outras palavras, está importando o framework Express.js para o seu aplicativo Node.js.

```
const app = express() // Inicializar as apps
```

O comando const app = express() é usado para criar uma instância de um aplicativo Express. Vamos quebrar isso em detalhes:

express(): Este é um método fornecido pelo framework Express.js que cria uma nova instância de um aplicativo Express. Quando chamado, ele retorna um novo objeto de aplicativo Express que pode ser configurado para lidar com solicitações HTTP.

const app = ...: Aqui, estamos atribuindo a instância recém-criada do aplicativo Express a uma constante chamada app. Isso significa que podemos usar a

variável app para configurar todas as rotas, middlewares e comportamentos do nosso aplicativo Express.

Depois de executar essa linha de código, app se torna o ponto central do seu aplicativo Express. Você pode usar app para adicionar rotas, definir middlewares, configurar manipuladores de erro, configurar parâmetros de aplicativo e muito mais. Em resumo, app é onde você configura toda a lógica do seu servidor Express.

```
app.use( //criando o MIDDLEWARES
  express.urlencoded({
    extended: true,
  }),
)
```

O comando app.use() é usado para adicionar middlewares ao aplicativo Express. **Middlewares são funções que têm acesso tanto ao objeto de solicitação (request) quanto ao objeto de resposta (response)** em um ciclo de solicitação-resposta HTTP. Eles podem executar código, fazer modificações nas solicitações ou respostas, encerrar o ciclo de solicitação-resposta ou chamar o próximo middleware na pilha.

Aqui está o que está acontecendo neste comando específico:

express.urlencoded(): **Este é um middleware integrado do Express.js projetado para analisar os corpos das solicitações que são codificados em URL** (application/x-www-form-urlencoded). Isso geralmente é usado para lidar com dados de formulários HTML enviados através de solicitações POST.

{ extended: true }: Este é um parâmetro de configuração opcional passado para o middleware express.urlencoded(). Quando definido como true, ele permite que o middleware analise os dados codificados em URL com qualquer formato, incluindo objetos complexos. Se definido como false, o middleware irá interpretar esses dados apenas como strings ou arrays.

app.use(): Este método do Express é usado para adicionar middlewares ao **pipeline de requisição do aplicativo**. O middleware express.urlencoded() é passado como um argumento para app.use(), o que significa que será aplicado a todas as rotas e todas as solicitações que o aplicativo recebe.

Resumindo, este comando específico configura o aplicativo Express para usar o middleware express.urlencoded() para analisar os dados codificados em URL nos corpos das solicitações, permitindo assim que o aplicativo lide facilmente com dados de formulários enviados através de solicitações POST.


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

`express.json()`: Este é um middleware integrado do Express.js projetado para analisar os corpos das solicitações que são formatados como JSON (`application/json`). Quando uma solicitação contém dados no formato JSON, este **middleware** converte esses dados em um objeto JavaScript acessível através da propriedade `req.body` da solicitação (`req` é o objeto de solicitação passado para cada manipulador de solicitação Express).

`app.use()`: Este método do Express é usado para adicionar middlewares ao pipeline de requisição do aplicativo. O middleware `express.json()` é passado como um argumento para `app.use()`, o que significa que será aplicado a todas as rotas e todas as solicitações que o aplicativo recebe.

```
app.get('/', (req, res) => {  
  
  //mostrar requisição mostrar a resposta que vai ser JSON  
  res.json({ message: 'Oi Express'})  
})
```

O comando que você forneceu define uma rota inicial (ou raiz) do tipo GET em um aplicativo Express. Vamos analisá-lo em detalhes:

1. `app.get('/', (req, res) => { ... })`: Este comando utiliza o método `app.get()` do Express para definir uma rota do tipo GET. A primeira parte (`'/'`) especifica o caminho da rota, neste caso, é a rota raiz, o que significa que será acessada quando o cliente visitar a URL base do servidor. A segunda parte é uma função de retorno de chamada (callback) que será executada quando uma solicitação GET for feita para esta rota.
2. `(req, res) => { ... }`: Esta é a função de retorno de chamada (callback) que será chamada quando a rota for acessada. Ela recebe dois parâmetros:
 - `req`: É o objeto de solicitação (request) que contém informações sobre a solicitação HTTP feita pelo cliente, como parâmetros de consulta, cabeçalhos e corpo da solicitação.
 - `res`: É o objeto de resposta (response) que é usado para enviar uma resposta de volta ao cliente.
3. `res.json({ message: 'Oi Express'})`: Dentro da função de retorno de chamada, este comando envia uma resposta JSON de volta ao cliente. Ele usa o método `res.json()` para enviar um objeto JSON como resposta. Neste caso, o objeto JSON é `{ message: 'Oi Express'}`. Isso significa que quando esta rota

for acessada, o cliente receberá um objeto JSON com a chave message contendo o valor 'Oi Express'.

Resumindo, este código define uma rota inicial do tipo GET no aplicativo Express que responde com um objeto JSON contendo a mensagem 'Oi Express' sempre que o cliente acessar a URL raiz do servidor.

```
app.listen(3000)
```

O comando `app.listen(3000)` é usado para iniciar o servidor HTTP Express e fazer com que ele comece a ouvir as solicitações HTTP na porta especificada. Vamos entender melhor:

`app.listen(port)`: Este método é utilizado para iniciar o servidor Express. Ele começa a ouvir as solicitações HTTP na porta especificada pelo argumento `port`. Quando uma solicitação chega a esta porta, o servidor Express irá lidar com ela de acordo com as rotas e middlewares configurados.

3000: No exemplo fornecido, 3000 é o número da porta onde o servidor Express começará a ouvir as solicitações. Portanto, o servidor Express estará disponível para lidar com as solicitações HTTP na porta 3000.

Assim, após a execução deste comando, o servidor Express estará em execução e pronto para receber solicitações HTTP na porta especificada (3000, neste caso). Por exemplo, se você acessar `http://localhost:3000` em um navegador da web, o servidor Express responderá às solicitações enviadas para essa URL.

Index

Chamar na Postman o Json

<https://www.postman.com/downloads/>

Professor Thiago Aula1 Redes Servidor IPPS Chamar na Postman o Json

