



# API REST com Node e TypeScript

Henrique Augusto Maltauro
Matheus Henrique de Barros Gobo

# Henrique Augusto Maltauro

***Formação Acadêmica***

- ***Graduação (2016 - 2018) - Unipar:*** Tecnologia em Análise e Desenvolvimento de Sistemas
- ***Pós-Graduação (2021 - Atualmente) - Unipar:*** Especialização em Desenvolvimento de Aplicações Para WEB e Mobile

***Experiência Profissional***

- ***Inside Sistemas (2016 - 2016):*** Estágio Suporte
- ***InterSystem Sistemas (2016 - 2017):*** Desenvolvedor Desktop
- ***Junsoft Sistemas (2017 - 2020):*** Desenvolvedor Desktop
- ***Maxicon Sistemas (2020 - 2020):*** Desenvolvedor Desktop
- ***Inside Sistemas (2021 - 2021):*** Desenvolvedor Mobile
- ***Metadados Sistemas de RH (2022 - Atualmente):*** Desenvolvedor Full Stack

# Matheus Henrique de Barros Gobo

***Formação Acadêmica***

- ***Graduação (2016 - 2018) - Unipar:*** Tecnologia em Análise e Desenvolvimento de Sistemas
- ***Pós-Graduação (2021 - Atualmente) - Unipar:*** Especialização em Desenvolvimento de Aplicações Para WEB e Mobile

***Experiência Profissional***

- ***Junsoft Sistemas (2017 - 2019):*** Desenvolvedor Desktop
- ***Maxicon Sistemas (2019 - 2022):*** Desenvolvedor Desktop
- ***Körber Supply Chain (2022 - Atualmente):*** Desenvolvedor Desktop/Mobile

# API

# API

Application Programming Interface

Interface de Programação de Aplicação

O objetivo é permitir que dois softwares, totalmente distintos, possam se comunicar através de um conjunto de definições e protocolos

# API

Nesse contexto, nós conseguimos fazer uma relação do objetivo com a sigla API

Onde, a aplicação vai ser qualquer software com funcionalidades distintas, e a interface vai ser imaginada como um contrato entre duas aplicações

# REST

# REST

Representational State Transfer

Transferência Representacional de Estado

É uma arquitetura de software que define um conjunto de regras a serem usadas para a criação de um serviço WEB, no nosso caso, uma API

# REST

A ideia do REST é padronizar a comunicação na internet em geral de forma abstrata, utilizando os métodos do protocolo HTTP

Sempre que vocês virem algum software definido como RESTfull, simplesmente quer dizer que aquele software utiliza a arquitetura REST

# HTTP

# HTTP

Hypertext Transfer Protocol

Protocolo de Transferência de Hipertexto

É o protocolo padrão de comunicação na internet

Toda a estrutura de comunicação do HTTP é feita através de requisição e resposta

# HTTP

Os métodos HTTP vai receber uma requisição (request), definindo que eles façam alguma ação na API

E vão retornar uma resposta (response), dizendo se a requisição foi executada com sucesso ou não, e até retornando dados que foram solicitados na requisição

# HTTP

**GET**

**POST**

**PUT**

**DELETE**

**PATCH**

**CONNECT**

**OPTIONS**

**TRACE**

**HEAD**

# HTTP

GET

- Requisita retorno de um recurso específico

POST

- Requisita uma adição de um recurso novo

# HTTP

## PUT

- Requisita uma atualização de todo o conteúdo de um recurso específico

## DELETE

- Requisita uma remoção de recurso específico

# Node

# Node

O Node é um ambiente de execução JavaScript server-side

Ele é baseado no interpretador de JavaScript V8, que é desenvolvido pela Google, e utilizado por navegadores como o Chromium e variantes do Chromium como o Chrome, Opera, Vivaldi, Brave, Microsoft Edge, etc

# Node

Isso permite que com o Node crie aplicações JavaScript para rodar como uma aplicação standalone em uma máquina, não dependendo de um navegadores para a execução

# Node

O principal motivo de sua utilização é a sua alta capacidade de escala, arquitetura, flexibilidade e baixo custo

Toda a execução do Node é realizada através de linha de comando, com os comandos `node`, `npm` e `npx`

# Node

`node`

- Ele serve para executar algum arquivo JavaScript
- No nosso caso, como vamos construir a nossa aplicação em TypeScript, não faremos uso desse comando

# Node

npm

- Node Package Manager
- Gerenciador de Pacotes do Node
- Ele vai instalar, desinstalar e atualizar pacotes na nossa aplicação Node
- Permite gerenciar toda a nossa aplicação Node

# Node

npx

- Node Package eXecutor
- Executor de Pacotes do Node
- Ele vai, da mesma forma que o npm, baixar algum pacote, e executar alguma ação, criando pastas e/ou arquivos padronizados de acordo com o pacote que foi baixado

# Node

```
npm init -y
```

- Inicializar a estrutura básica de uma aplicação Node, criando o arquivo package.json

# Node

## package.json

- É um arquivo de configuração utilizar para
  - Definir configurações gerais do aplicação como, nome do projeto, versão, autor, licença, etc
  - Definir scripts automatizados para executarmos a aplicação
  - Definir os pacotes utilizados pela aplicação

# Node

JavaScript

- É uma linguagem de programação interpretada, de script e fracamente tipada, ou seja, nós não definimos tipos para as variáveis
- É através da interpretação do JavaScript que os navegadores executam os códigos

# Node

`npm run <script>`

- Executa um script definido no `package.json`

# Node

TypeScript

- É uma linguagem de programação, melhor definida como um super conjunto do JavaScript
- A principal diferença do JavaScript para o TypeScript, é a tipagem de variáveis
- Tudo o que funciona no JavaScript, funciona da mesma forma no TypeScript

# Node

TypeScript

- Porém algumas funcionalidades do TypeScript não funcionam no JavaScript
- Como o Node foi construído para funcionar com JavaScript, essas funcionalidades únicas do TypeScript precisam ser convertidas em JavaScript

# Node

## Dependências de Desenvolvimento

- Como dito anteriormente, o arquivo `package.json` armazena as definições de pacotes utilizados na aplicação
- Porém, alguns desses pacotes serão utilizados apenas para o desenvolvimento da aplicação

# Node

## Dependências de Desenvolvimento

- Sendo assim nos definimos esses pacotes como dependências de desenvolvimento
- Esses pacotes não irão para o build final da aplicação, que é o que vai estar sendo executado lá no servidor

# Node

## typescript

- Para trabalharmos com TypeScript no Node, nós precisamos instalar o pacote `typescript`
- Como o build final da aplicação vai ser em JavaScript, o pacote `typescript` é incluído na aplicação como dependência de desenvolvimento
- `npm i typescript -D`

# Node

## tsconfig.json

- Uma vez instalado o pacote `typescript` nós precisamos gerar o arquivo `tsconfig.json`, que vai ser responsável por definir as configurações de execução do TypeScript
- `npx tsc --init`

# Node

## ts-node-dev

- Esse pacote converte o código TypeScript em JavaScript em tempo de execução
- Ou seja, ele serve para executarmos a nossa aplicação enquanto estamos desenvolvendo ela
- Como é apenas para o processo de desenvolvimento, ele é incluído como dependência de desenvolvimento
- `npm i ts-node-dev -D`

# Node

express

- Esse pacote disponibiliza um framework, que fornece os recurso mínimos para construir um serviço WEB em Node, no nosso caso uma API em Node
- `npm i express`

# Node

## @types

- Como estamos trabalhando em TypeScript, e os pacotes do Node foram construídos inicialmente para JavaScript, nós precisamos de alguns pacotes do escopo @types, que vão permitir que utilizemos algumas definições próprias do TypeScript para esses pacotes

# Node

## @types/express

- Esse pacote permite que utilizemos definições próprias do TypeScript para o express
- Como essa definições são apenas para o processo de desenvolvimento da aplicação, ele é incluído como dependência de desenvolvimento
- `npm i @types/express -D`

# Node

## Pasta  src

- Na raiz do projeto, vamos criar uma pasta chamada src, que vai armazenar todos os arquivos principais da nossa aplicação

# Node

server.ts

- Dentro da pasta src, vamos criar um arquivo chamado server.ts
- Vai ser o arquivo responsável por inicializar o nosso servidor

# Node

app.ts

- Dentro da pasta src, vamos criar um arquivo chamado app.ts
- Por convenção nós separamos o arquivo server.ts do app.ts, para diferenciarmos ambientes de execução
- Por exemplo, se formos realizar testes automatizados, nós precisamos do app.ts, mas não precisamos iniciar o servidor, que é a função do server.ts

# JSON

# JSON

JavaScript Object Notation

É um objeto JavaScript, baseada em texto, sem schema, baseada em pares de chave-valores

Extremamente leve, e por causa disso, quase todos os serviços WEB fazem uso dele

# JSON

```
{
    “id”: “6341c9ea434105629cdaaeea”,
    “name”: “Nome da Pessoa”,
    “age”: 23
}
```

# JSON


```
[
    {
        “id”: “6341c9ea434105629cdaaeea”,
        “name”: “Nome da Pessoa 1”,
        “age”: 23
    },
    {
        “id”: “2432dsf424s2324434tfg233”,
        “name”: “Nome da Pessoa 2”,
        “age”: 18
    }
]
```

# Insomnia

# Insomnia

O Insomnia é um software que permite simularmos as requisições de qualquer API

Isso nos permite validarmos se a nossa API está funcionando da maneira esperada

# Node

typeorm

- Esse pacote disponibiliza uma ferramenta para manipular banco de dados através de um mapeamento de objetos
- `npm i typeorm`

# MongoDB

# MongoDB

É um banco de dados orientado a documentos, que salva os dados em documentos semelhantes a um JSON

```
{
    “id”: “6341c9ea434105629cdaaeea”,
    “name”: “Nome da Pessoa”,
    “age”: 23
}
```

# Node

## mongodb

- Esse pacote permite utilizar o MongoDB em uma aplicação Node
- `npm i mongodb`

# Node

`@types/mongodb`

- Esse pacote permite que utilizemos definições próprias do TypeScript para o `mongodb`
- Como essa definições são apenas para o processo de desenvolvimento da aplicação, ele é incluído como dependência de desenvolvimento
- `npm i @types/mongodb -D`

# Node

## Pasta  database

- Dentro da pasta src, vamos criar uma pasta chamada database, que vai armazenar todas as configurações de banco de dados da nossa aplicação
- Dentro desta pasta vamos criar um arquivo chamado AppDataSource.ts

# Node

## dotenv

- Esse pacote permite utilizarmos variáveis de ambiente em uma aplicação Node
- `npm i dotenv`

# Node

.env

- Dentro da raiz do projeto vamos criar um arquivo chamado .env
- Esse arquivo é onde vamos definir as nossas variáveis de ambiente para a nossa aplicação Node

# Node

## Pasta  entities

- Dentro da pasta src, vamos criar uma pasta chamada entities, que vai armazenar todas as nossas entidades
- Dentro desta pasta vamos criar um arquivo chamado Person.ts

# Node

## Pasta  controllers

- Dentro da pasta src, vamos criar uma pasta chamada controllers, que vai armazenar todos os nossos controladores
- Dentro desta pasta vamos criar um arquivo chamado PersonController.ts

# Node

## Pasta  routes

- Dentro da pasta src, vamos criar uma pasta chamada routes, que vai armazenar todas as nossas rotas
- Dentro desta pasta vamos criar um arquivo chamado PersonRoutes.ts

# Node

## Pasta  repositories

- Dentro da pasta src, vamos criar uma pasta chamada repositories, que vai armazenar todos os nossos repositórios
- Dentro desta pasta vamos criar um arquivo chamado PersonRepository.ts

# Node

## express-async-errors

- Esse pacote permite que o pacote express utilize erros de forma assíncrona, ou seja, a nossa aplicação não vai ser interrompida quando tiver algum erro
- npm i express-async-errors

# Node

yup

- Esse pacote permite validarmos as informações que estaremos recebendo no corpo das nossas requisições
- `npm i yup`