



13/8/2019

# Treinamento Angular 8

COTI INFORMATICA



[WWW.COTIINFORMATICA.COM.BR](http://WWW.COTIINFORMATICA.COM.BR)

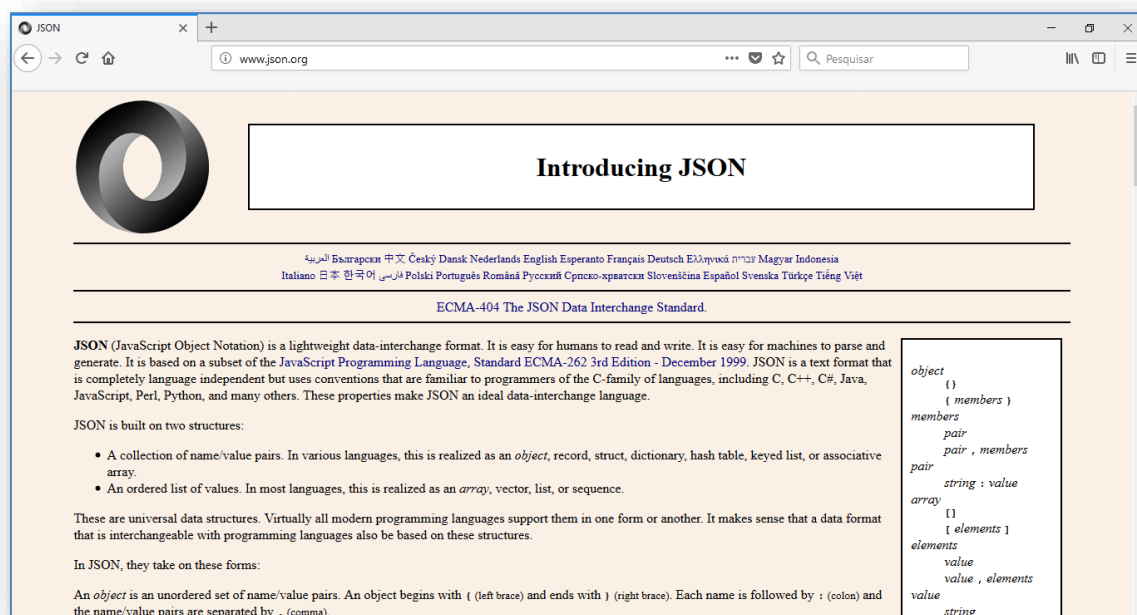
AV RIO BRANCO, 185 – SALA 308, CENTRO, RIO DE JANEIRO - RJ

## Sumário

|   |    |
|---|----|
| JSON .....  | 2  |
| Sintaxe no JSON.....                                | 2  |
| JSON x XML.....                                     | 6  |
| JSON no Java .....                                  | 7  |
| O que é o JSON Server.....                          | 8  |
| O Que É REST? .....                                 | 8  |
| Por Que Precisamos de uma API REST Falsa? .....     | 8  |
| Injectable() .....                                  | 9  |
| Hierarquia do injetor e instâncias de serviço ..... | 10 |
| HttpClient .....                                    | 11 |
| instalação .....                                    | 12 |
| Por que escrever um link de serviço .....           | 12 |
| Tratamento de erros.....                            | 12 |
| Obtendo o link dos detalhes do erro .....           | 13 |
| Observables e operadores link.....                  | 13 |
| Enviando dados para o link do servidor .....        | 14 |
| Adicionando cabeçalhos.....                         | 14 |
| Input().....  | 14 |
| Notas de uso.....                                   | 15 |
| Lista De Status HTTP.....                           | 16 |
| NOVO PROJETO: .....                                 | 19 |
| NOVO PROJETO .....                                  | 33 |
| NOVO PROJETO .....                                  | 43 |
| NOVO PROJETO .....                                  | 50 |

# JSON

<http://www.json.org/>



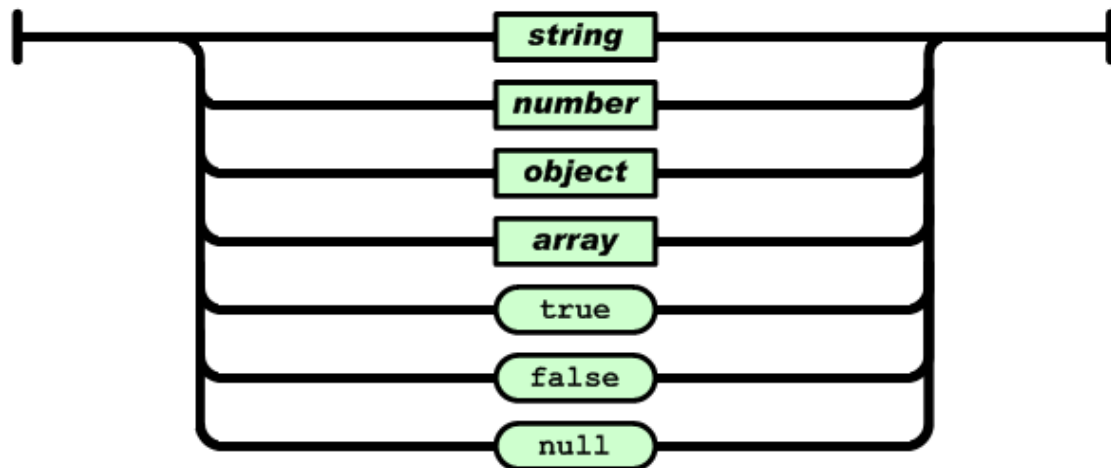
**JSON (JavaScript Object Notation) é um modelo para armazenamento e transmissão de informações no formato texto.** Apesar de muito simples, tem sido bastante utilizado por aplicações Web devido a sua capacidade de estruturar informações de uma forma bem mais **compacta** do que a conseguida pelo modelo XML, tornando mais rápido o parsing dessas informações. Isto explica o **fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados.**

## Sintaxe no JSON

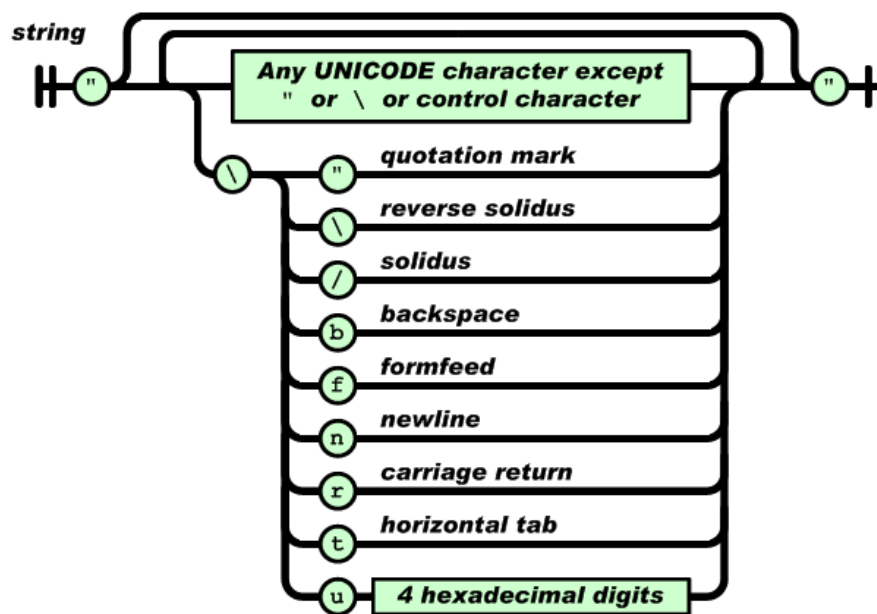
A ideia utilizada pelo **JSON para representar informações é simples: para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado.** Esta sintaxe é derivada da forma utilizada pelo JavaScript para representar informações.

Um valor pode ser uma string entre aspas duplas, ou um número, ou verdadeiro ou falso ou nulo, ou um objeto ou uma matriz. Essas estruturas podem ser aninhadas.

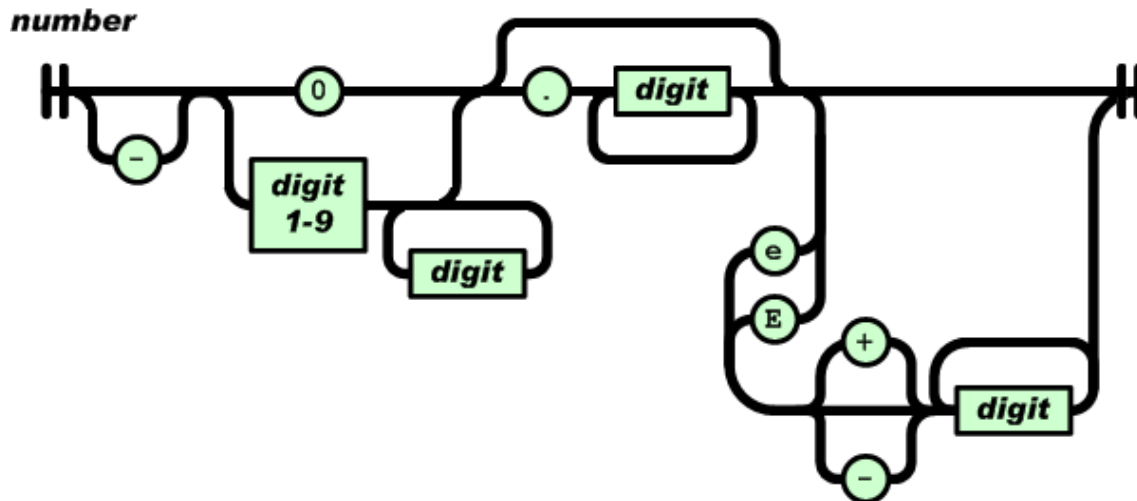
**value**



Uma string é uma seqüência de zero ou mais caracteres Unicode, agrupados em aspas duplas, usando escapes de barra invertida. Um caractere é representado como uma única cadeia de caracteres. Uma string é muito parecida com uma string C ou Java.



Um número é muito parecido com um número C ou Java, exceto que os formatos octal e hexadecimal não são usados.



Representando o ano de 2018

**"ano": 2018**

Um par nome/valor deve ser representado pelo nome entre aspas duplas, seguido de dois pontos, seguido do valor. Os valores podem possuir apenas 3 **tipos básicos**: numérico (inteiro ou real), booleano e string. As Listagens 2, 3, 4 e 5 apresentam exemplos. Observe que os valores do tipo string devem ser representados entre aspas.

Representando um número real

**"altura": 1.72**

Representando uma string

**"site": "www.devmedia.com.br"**

Representando um número negativo

**"temperatura": -2**

Representando um valor booleano

**"casado": true**

A partir dos tipos básicos, é possível construir **tipos complexos**: array e objeto. Os arrays são delimitados por colchetes, com seus elementos separados entre vírgulas. As listagens 6 e 7 mostram exemplos.

Array de Strings

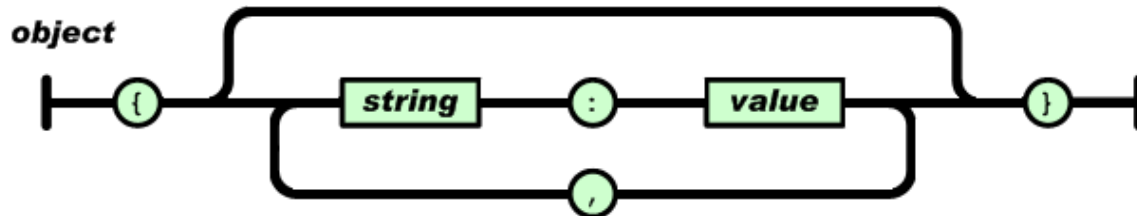
**`["RJ", "SP", "MG", "ES"]`**

#### Matriz de Inteiros

**`[  
 [1,5],  
 [-1,9],  
 [1000,0]  
]`**

Os objetos são especificados entre chaves e podem ser compostos por múltiplos pares nome/valor, por arrays e também por outros objetos. Desta forma, um objeto JSON pode representar, virtualmente, qualquer tipo de informação! O exemplo da Listagem 8 mostra a representação dos dados de um filme.

Um objeto é um conjunto não ordenado de pares nome / valor. Um objeto começa com {(chave esquerda) e termina com} (chave direita). Cada nome é seguido por: (dois pontos) e os pares nome / valor são separados por (vírgula).

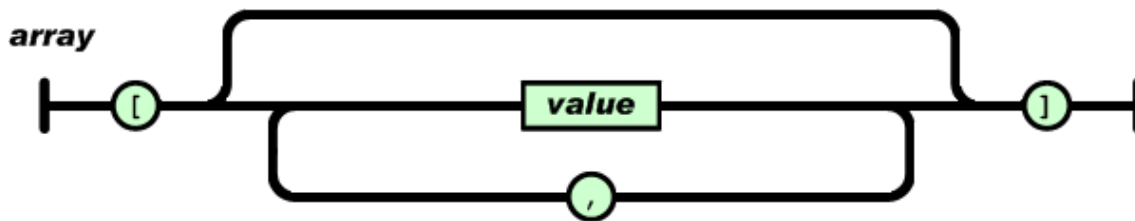


#### Listagem 8: Objeto

**`{  
 "titulo": "JSON x XML",  
 "resumo": "o duelo de dois modelos de representação de informações",  
 "ano": 2012,  
 "genero": ["aventura", "ação", "ficção"]  
}`**

Uma matriz é uma coleção ordenada de valores. Um array começa com [(colchete esquerdo) e termina com] (colchete direito). Os valores são separados por (vírgula).

#### Listagem 9: Array de objetos



```
[
  {
    "titulo": "JSON x XML",
    "resumo": "o duelo de dois modelos de representação de informações",
    "ano": 2012,
    "genero": ["aventura", "ação", "ficção"]
  },
  {
    "titulo": "JSON James",
    "resumo": "a história de uma lenda do velho oeste",
    "ano": 2012,
    "genero": ["western"]
  }
]
```

A palavra-chave "null" deve ser utilizada para a representação de valores nulos.

#### Representando um valor nulo

**"site":null**

## JSON x XML

Podemos entender o **JSON** como uma espécie de "concorrente" da **XML** na área de troca de informações.

#### **Semelhanças:**

- Os dois modelos representam informações no formato texto.
- Ambos possuem natureza auto-descritiva (ou seja, basta "bater o olho" em um arquivo JSON ou em um arquivo XML para entender o seu significado).
- Ambos são capazes de representar informação complexa, difícil de representar no formato tabular. Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivalorados, arrays, dados ausentes, etc.
- Ambos podem ser utilizados para transportar informações em aplicações [AJAX](#).
- Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.

- Ambos são independentes de linguagem. Dados representados em XML e **JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas.**

#### Diferenças:

- JSON não é uma linguagem de marcação. Não possui tag de abertura e muito menos de fechamento!
- JSON representa as informações de forma mais compacta.
- JSON não permite a execução de instruções de processamento, algo possível em XML.
- JSON é tipicamente destinado para a troca de informações, enquanto XML possui mais aplicações. Por exemplo: nos dias atuais existem bancos de dados inteiros armazenados em XML e estruturados em SGBD's XML nativo.

#### JSON no Java

Ao acessar a home page oficial do JSON ([www.json.org](http://www.json.org)) você verá que existem parsers disponíveis para quase todas as linguagens: [Delphi](#), [PHP](#), [Java](#), Matlab, C++, C#, etc.

No XML a coisa é bem mais fácil, pois existem duas API's básicas para o parsing de informações: SAX e DOM. Ambas já são instaladas junto com o Java e, daí, basta utilizá-las. Adicionalmente, o princípio de funcionamento das API's SAX e DOM é bastante conhecido pelos desenvolvedores: DOM importa o documento todo para a memória, criando uma árvore, enquanto o SAX percorre o arquivo sequencialmente disparando eventos, sem realizar a importação de informações para a memória.

O JSON parece ainda não ter atingido esse grau maturidade. Várias pessoas/empresas implementaram os seus próprios parsers, com diferentes princípios de funcionamento. Como ainda não há um padrão, fica difícil decidir qual utilizar!

Para converter um objeto Java para JSON, utiliza-se o método toJson. **E para fazer ao contrário, ou seja, atribuir o conteúdo de um objeto Java a partir de uma string Json, utiliza-se fromJson.**



## O que é o JSON Server

Ter uma API pronta para começar o desenvolvimento do front-end da sua aplicação as vezes pode ser um problema. Com isso te apresento o JSON Server.

JSON Server é uma biblioteca capaz de criar uma API Fake em 30 segundos e sem precisar escrever nenhuma linha de código.

## O Que É REST?

REST vem de REpresentational State Transfer. É um estilo de arquitetura para projetar aplicações conectadas, usando os simples verbos HTTP para permitir a comunicação entre as máquinas. Assim, ao invés de usar uma URL para manipular informação do usuário, REST enviar uma requisição HTTP, como GET, POST, DELETE, etc., para uma URL manipular os dados.

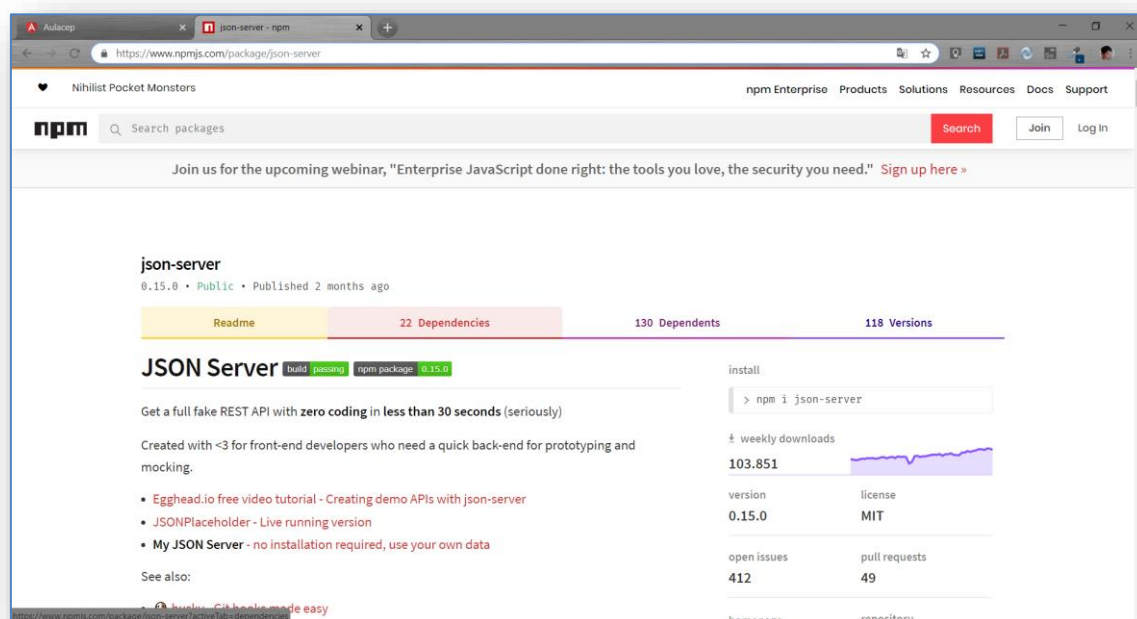
## Por Que Precisamos de uma API REST Falsa?

APIs REST formam o backend de aplicações web e móveis. Ao criar aplicações, algumas vezes não temos APIs REST prontas para uso no durante o desenvolvimento. Para ver um app web ou móvel em ação, precisamos de um servidor que retorne dados JSON falsos.

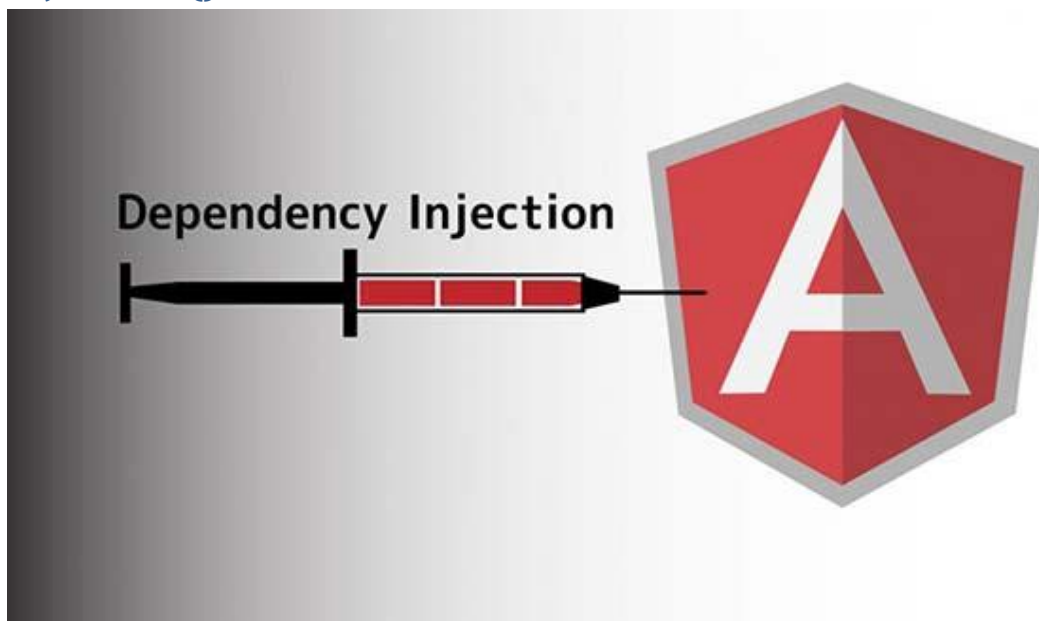
É aí que uma API REST falsa entra em ação. `json-server` provê a funcionalidade para configurar um servidor de API REST falso com mínimo de esforço.

---

<https://www.npmjs.com/package/json-server>



## Injectable()



A injeção de dependência (DI) é um importante padrão de projeto de aplicativo. A Angular possui sua própria estrutura de DI, que é normalmente usada no projeto de aplicações angulares para aumentar sua eficiência e modularidade.

Dependências são serviços ou objetos que uma classe precisa para executar sua função. DI é um padrão de codificação em que uma classe pede dependências de fontes externas em vez de criá-las.

Em Angular, a estrutura de DI fornece dependências declaradas para uma classe quando essa classe é instanciada. Este guia explica como o DI funciona em Angular e como você o utiliza para tornar seus aplicativos flexíveis, eficientes e robustos, além de testáveis e de fácil manutenção.

A estrutura DI permite que você forneça dados para um componente de uma classe de serviço injetável, definida em seu próprio arquivo. Para demonstrar, criaremos uma classe de serviço injetável que forneça uma lista de heróis e registre essa classe como um provedor desse serviço.

Ter várias classes no mesmo arquivo pode ser confuso. Geralmente, recomendamos que você defina componentes e serviços em arquivos separados.

Se você combinar um componente e um serviço no mesmo arquivo, é importante definir primeiro o serviço e depois o componente. Se você definir o componente antes do serviço, obterá um erro de referência nulo em tempo de execução.

É possível definir o componente primeiro com a ajuda do `forwardRef()` método, conforme explicado neste post .

Você também pode usar referências avançadas para interromper dependências circulares. Veja um exemplo no DI Cookbook .

O decorador o marca como um serviço que pode ser injetado, mas o Angular não consegue injetá-lo em lugar algum até que você configure um injetor de dependência Angular com um provedor desse serviço. `@Injectable()`

O injetor é responsável por criar instâncias de serviço e injetá-las em classes como `HeroListComponent`. Você raramente cria um injetor angular sozinho. A Angular cria injetores para você enquanto executa o aplicativo, começando com o injetor de raiz que ele cria durante o processo de bootstrap .

Um provedor informa ao injetor como criar o serviço . Você deve configurar um injetor com um provedor antes que o injetor possa criar um serviço (ou fornecer qualquer outro tipo de dependência).

Um provedor pode ser a própria classe de serviço, para que o injetor possa usar `new` para criar uma instância. Você também pode definir mais de uma classe para fornecer o mesmo serviço de maneiras diferentes e configurar diferentes injetores com provedores diferentes.

## Hierarquia do injetor e instâncias de serviço

Os serviços são singletons no escopo de um injetor . Ou seja, há no máximo uma instância de um serviço em um determinado injetor.

Existe apenas um injetor de raiz para um aplicativo. Fornecer `UserService` no nível root ou `AppModule` significa que é registrado com o injetor de raiz. Há apenas uma `UserService` instância em todo o aplicativo e cada classe que injeta `UserService` obtém essa instância de serviço, a menos que você configure outro provedor com um `child injector` .

O DI angular possui um sistema de injeção hierárquica , o que significa que os injetores aninhados podem criar suas próprias instâncias de serviço. Angular cria regularmente injetores aninhados. Sempre que Angular cria uma nova instância de um componente providers especificado , também cria um novo injetor secundário para essa instância. Da mesma forma, quando um novo `NgModule` é carregado com preguiça em tempo de execução, o Angular pode criar um injetor para ele com seus próprios provedores. `@Component()`

Módulos filho e injetores de componentes são independentes uns dos outros e criam suas próprias instâncias separadas dos serviços fornecidos. Quando o Angular destrói um NgModule ou uma instância do componente, ele também destrói o injetor e as instâncias de serviço desse injetor.

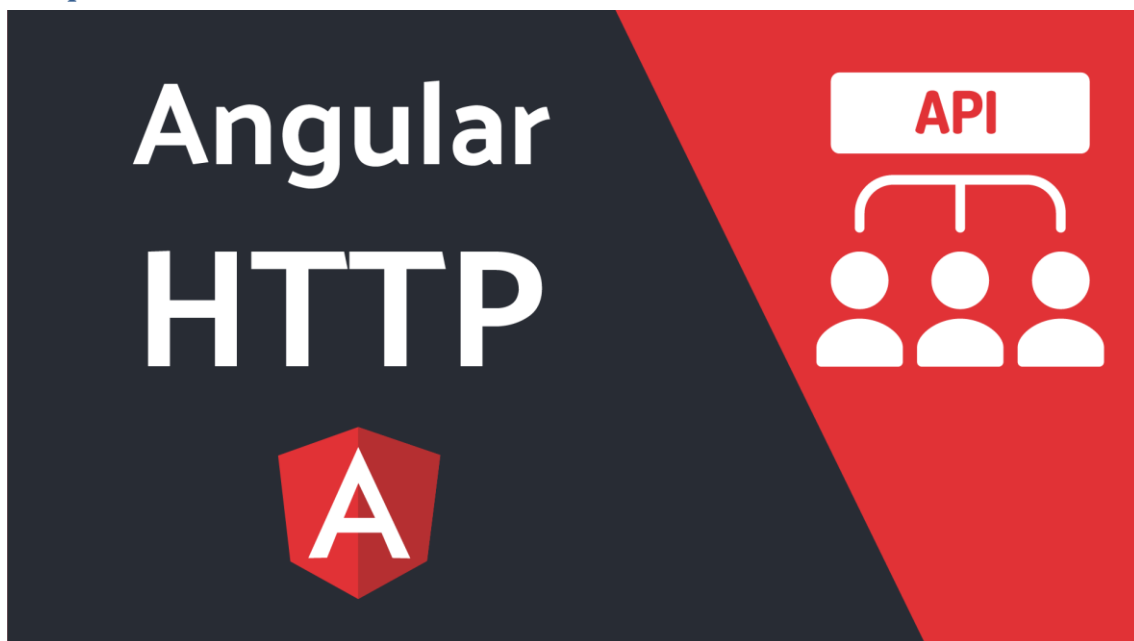
Graças à herança do injetor, você ainda pode injetar serviços em todo o aplicativo nesses componentes. O injetor de um componente é um filho do injetor de seu componente pai e herda de todos os injetores ancestrais até o injetor de raiz do aplicativo. Angular pode injetar um serviço fornecido por qualquer injetor nessa linhagem.

Por exemplo, o Angular pode injetar HeroListComponent tanto

o HeroService fornecido HeroComponent quanto

o UserService fornecido AppModule.

## HttpClient



A maioria dos aplicativos front-end se comunica com os serviços de backend pelo protocolo HTTP. Navegadores modernos suportam duas APIs diferentes para fazer solicitações HTTP: a XMLHttpRequest interface e a fetch() API.

O HttpClient oferece uma API HTTP de cliente simplificada para aplicativos Angulares que se apóia na interface exposta pelos navegadores. Os benefícios adicionais de incluir recursos de testabilidade, objetos de solicitação e resposta

digitados, interceptação de solicitação e resposta, apis e tratamento simplificado de erros.

@angular/common/httpXMLHttpRequestHttpClientObservable

## instalação

Antes de poder usar o HttpClient, você precisa importar o Angular HttpClientModule. A maioria dos aplicativos faz isso na raiz AppModule.

## Por que escrever um link de serviço

Este exemplo é tão simples que é tentador escrever o Http.get() interior do próprio componente e pular o serviço.

No entanto, o acesso a dados raramente fica tão simples. Você normalmente pós-processa os dados, adiciona manipulação de erros e talvez alguma lógica de repetição para lidar com a conectividade intermitente.

O componente rapidamente se torna confuso com minúcias de acesso a dados. O componente se torna mais difícil de entender, mais difícil de testar, e a lógica de acesso a dados não pode ser reutilizada ou padronizada.

É por isso que é uma prática recomendada separar a apresentação de dados do acesso a dados encapsulando o acesso a dados em um serviço separado e delegando a esse serviço no componente, mesmo em casos simples como este.

## Tratamento de erros

O que acontece se a solicitação falhar no servidor ou se uma conexão de rede ruim impedir que ela atinja o servidor? HttpClient retornará um objeto de erro em vez de uma resposta bem-sucedida.

Você poderia manipular no componente adicionando um segundo retorno de chamada ao .subscribe():

## Obtendo o link dos detalhes do erro

Detectar que um erro ocorreu é uma coisa. Interpretar esse erro e compor uma resposta amigável é um pouco mais envolvente.

Dois tipos de erros podem ocorrer. O back-end do servidor pode rejeitar a solicitação, retornando uma resposta HTTP com um código de status, como 404 ou 500. Essas são respostas de erro .

Ou algo pode dar errado no lado do cliente, como um erro de rede que impede que a solicitação seja concluída com êxito ou que uma exceção seja lançada em um operador RxJS. Esses erros produzem `ErrorEvent` objetos JavaScript .

A `HttpClient` captura de ambos os tipos de erros no seu `HttpErrorResponse` e você pode inspecionar essa resposta para descobrir o que realmente aconteceu.

Inspeção, interpretação e resolução de erros é algo que você deseja fazer no serviço , não no componente .

## Observables e operadores link

As seções anteriores deste guia se referiam a RxJS `Observable` e operadores como `catchError` e `retry`. Você encontrará mais artefatos RxJS enquanto continua abaixo.

RxJS é uma biblioteca para composição de código assíncrono e baseado em retorno de chamada em um estilo funcional e reativo . Muitas APIs angulares, incluindo `HttpClient`, produzem e consomem RxJS `Observables`.

O próprio RxJS está fora do escopo deste guia. Você encontrará muitos recursos de aprendizado na web. Enquanto você pode conviver com um mínimo de conhecimento RxJS, você vai querer aumentar suas habilidades RxJS ao longo do tempo, a fim de usar de forma eficaz.

Se você estiver acompanhando esses snippets de código, observe que deve importar os símbolos observáveis e de operador RxJS que aparecem nesses snippets. Essas `ConfigService` importações são típicas.

## Enviando dados para o link do servidor

Além de buscar dados do servidor, HttpClient suporta solicitações mutantes, isto é, enviando dados para o servidor com outros métodos HTTP, como PUT, POST e DELETE.

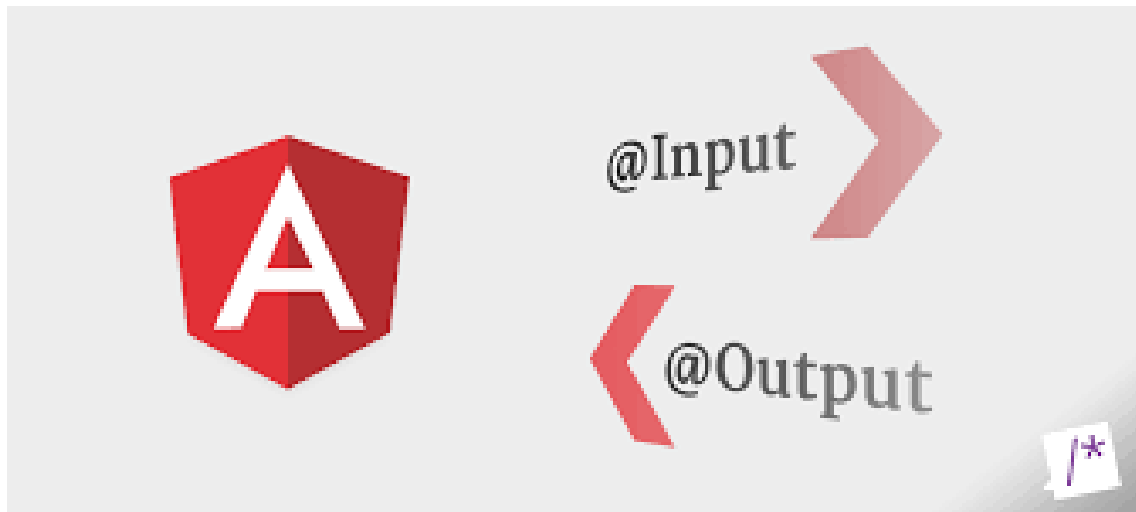
O aplicativo de amostra deste guia inclui uma versão simplificada do exemplo "Tour of Heroes", que busca heróis e permite que os usuários os adicionem, excluam e atualizem.

As seções a seguir extraem os métodos da amostra HeroesService.

## Adicionando cabeçalhos

Muitos servidores exigem cabeçalhos extras para operações de salvaguarda. Por exemplo, eles podem exigir um cabeçalho "Content-Type" para declarar explicitamente o tipo MIME do corpo da solicitação. Ou talvez o servidor precise de um token de autorização.

## Input()



Decorador que marca um campo de classe como uma propriedade de entrada e fornece metadados de configuração. A propriedade input está vinculada a uma propriedade DOM no modelo. Durante a detecção de alterações, o Angular atualiza automaticamente a propriedade de dados com o valor da propriedade DOM.

## Notas de uso

Você pode fornecer um nome opcional para usar em modelos quando o componente é instanciado, que é mapeado para o nome da propriedade associada. Por padrão, o nome original da propriedade associada é usado para ligação de entrada.

O exemplo a seguir cria um componente com duas propriedades de entrada, uma das quais recebe um nome de ligação especial.

```
@Component({
  selector: 'bank-account',
  template: `
    Bank Name: {{bankName}}
    Account Id: {{id}}
  `
})
class BankAccount {
  // This property is bound using its original name.
  @Input() bankName: string;
  // this property value is bound to a different property name
  // when this component is instantiated in a template.
  @Input('account-id') id: string;

  // this property is not bound, and is not automatically
  updated by Angular
  normalizedBankName: string;
}

@Component({
  selector: 'app',
  template: `
    <bank-account bankName="RBC" account-id="4747"></bank-
    account>
  `
})
class App {}
```





## Lista De Status HTTP

- 1 Lista de códigos de status HTTP
- 2 1xx Informativa
  - 2.1 100 Continuar
  - 2.2 101 Mudando protocolos
  - 2.3 102 Processamento (WebDAV) (RFC 2518)
  - 2.4 122 Pedido-URI muito longo
- 3 200 Sucesso
  - 3.1 201 Criado
  - 3.2 202 Aceito
  - 3.3 203 não-autorizado (desde HTTP/1.1)
  - 3.4 204 Nenhum conteúdo
  - 3.5 205 Reset
  - 3.6 206 Conteúdo parcial
  - 3.7 207-Status Multi (WebDAV) (RFC 4918)
- 4 3xx Redirecionamento
  - 4.1 300 Múltipla escolha
  - 4.2 301 Movido
  - 4.3 302 Encontrado
  - 4.4 303 Consulte Outros

- 4.5 304 Não modificado
- 4.6 305 Use Proxy (desde HTTP/1.1)
- 4.7 306 Proxy Switch
- 4.8 307 Redirecionamento temporário (desde HTTP/1.1)
- 4.9 308 Redirecionamento permanente (RFC 7538[2])
- 5 4xx Erro de cliente
  - 5.1 400 Requisição inválida
  - 5.2 401 Não autorizado
  - 5.3 402 Pagamento necessário
  - 5.4 403 Proibido
  - 5.5 404 Não encontrado
  - 5.6 405 Método não permitido
  - 5.7 406 Não Aceitável
  - 5.8 407 Autenticação de proxy necessária
  - 5.9 408 Tempo de requisição esgotou (Timeout)
  - 5.10 409 Conflito geral
  - 5.11 410 Gone
  - 5.12 411 comprimento necessário
  - 5.13 412 Pré-condição falhou
  - 5.14 413 Entidade de solicitação muito grande
  - 5.15 414 Pedido-URI Too Long
  - 5.16 415 Tipo de mídia não suportado
  - 5.17 416 Solicitada de Faixa Não Satisfatória
  - 5.18 417 Falha na expectativa
  - 5.19 418 Eu sou um bule de chá
  - 5.20 422 Entidade improcessável (WebDAV) (RFC 4918)
  - 5.21 423 Fechado (WebDAV) (RFC 4918)
  - 5.22 424 Falha de Dependência (WebDAV) (RFC 4918)
  - 5.23 425 coleção não ordenada (RFC 3648)
  - 5.24 426 Upgrade Obrigatório (RFC 2817)

- 5.25 450 bloqueados pelo Controle de Pais do Windows
- 5.26 499 cliente fechou Pedido (utilizado em ERPs/VPsA)
- 6 5xx outros erros

6.1 500 Erro interno do servidor (Internal Server Error)

6.2 501 Não implementado (Not implemented)

6.3 502 Bad Gateway

6.4 503 Serviço indisponível (Service Unavailable)

6.5 504 Gateway Time-Out

6.6 505 HTTP Version not support

## CRIANDO NOVO PROJETO:

Digitar o comando para criação do projeto:

***"ng new nome\_do\_projeto"***

Digitar "y" para a criação das rotas.

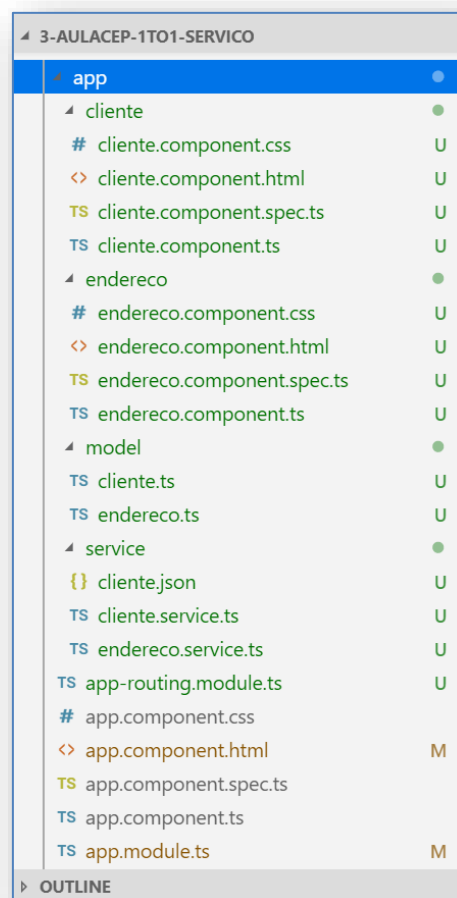
Selecionar "CSS" para o estilo.

Aguardar criar o projeto..

Projeto criado...

## NOVO PROJETO:

Estrutura do projeto depois de finalizado:



Criar o diretório “model”.

## cliente.ts

```
import { Endereco } from './endereco';

export class Cliente {

    id: number;
    nome: string;
    email: string;

    //RELACIONAMENTO ONE TO ONE
    //CLIENTE TEM UM ENDEREÇO
    endereco: Endereco;

    constructor(id?: number, nome?: string, email?: string) {
        this.id = id;
        this.nome = nome;
        this.email = email;
    }
}
```

---

## endereco.ts

```
import { Cliente } from './cliente';

export class Endereco {

    codigo: number;
    logradouro: string;
    bairro: string;
    cidade: string;
    estado: string;
    cep: string;

    //RELACIONAMENTO ENDEREÇO É DE UM CLIENTE
    //BIDIRECIONAL, ONDE CLIENTE TEM ENDERECO E ENDERECO TEM
    CLIENTE
    cliente: Cliente;

    constructor(codigo?: number, logradouro?: string, bairro?:
string, cidade?: string, estado?: string, cep?: string) {
        this.codigo = codigo;
        this.logradouro = logradouro;
    }
}
```

```

        this.bairro = bairro;
        this.cidade = cidade;
        this.estado = estado;
        this.cep = cep;
    }
}

```

---

Criar o diretorio "service"

## cliente.json

```

{
  "cliente": [
    {
      "id": 1,
      "nome": "belem",
      "email": "belem@gmail.com",
      "imagem":
"https://randomuser.me/api/portraits/lego/1.jpg",
      "endereco": [
        {
          "codigo": 10,
          "logradouro": "av rio branco",
          "bairro": "centro",
          "cidade": "rio de janeiro",
          "estado": "RJ",
          "cep": "20040007"
        }
      ]
    },
    {
      "id": 2,
      "nome": "bebel",
      "email": "bebel@gmail.com",
      "imagem":
"https://randomuser.me/api/portraits/lego/4.jpg",
      "endereco": [
        {
          "codigo": 10,
          "logradouro": "rua das casas",
          "bairro": "recreio",
          "cidade": "rio de janeiro",
          "estado": "RJ",
          "cep": "20040007"
        }
      ]
    }
  ]
}

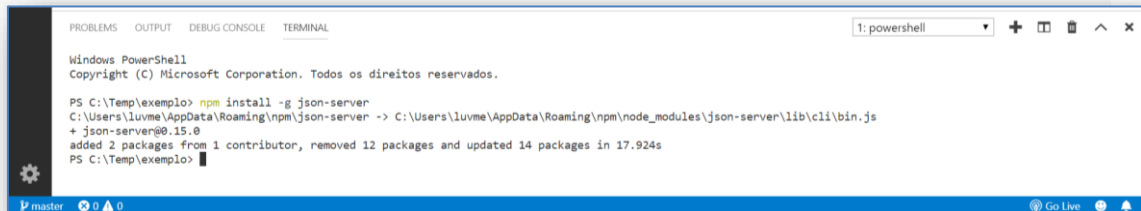
```

```
}  
]  
}
```

Para instalar o Json-server.

Digitar no terminal.

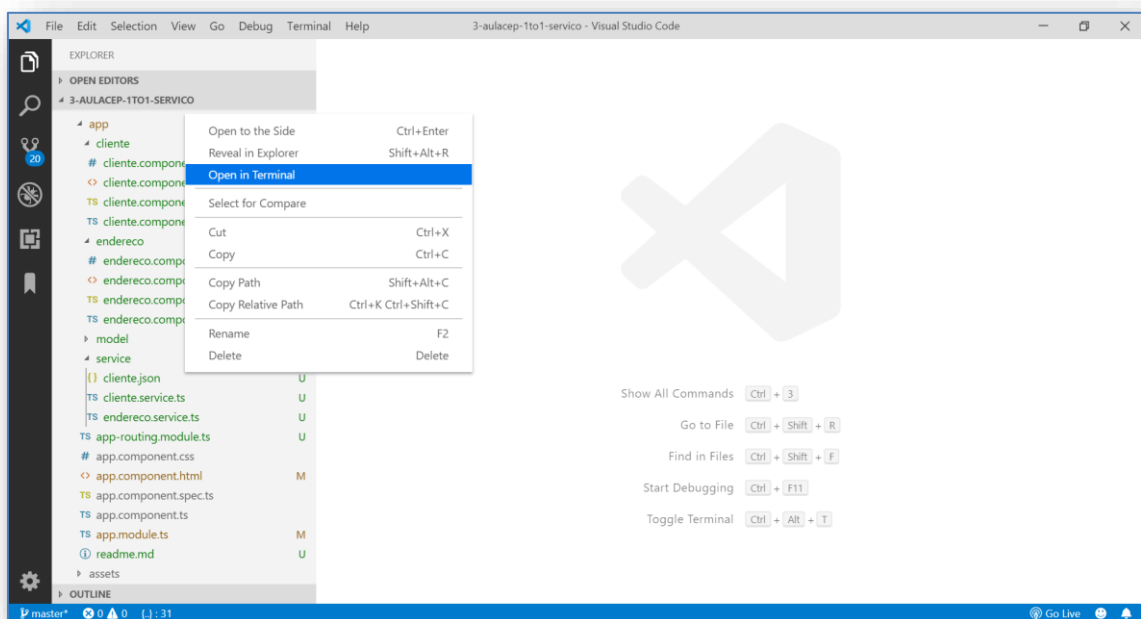
**“npm install -g json-server”**



```
Windows PowerShell  
Copyright (C) Microsoft Corporation. Todos os direitos reservados.  
  
PS C:\Temp\exemplo> npm install -g json-server  
C:\Users\luvre\AppData\Roaming\npm\json-server -> C:\Users\luvre\AppData\Roaming\npm\node_modules\json-server\lib\cli\bin.js  
+ json-server@0.15.0  
+ json-server@0.15.0  
added 2 packages from 1 contributor, removed 12 packages and updated 14 packages in 17.924s  
PS C:\Temp\exemplo>
```

Para iniciar o json-server.

Clicar no arquivo “cliente.json” com o botão direito -> Open in terminal



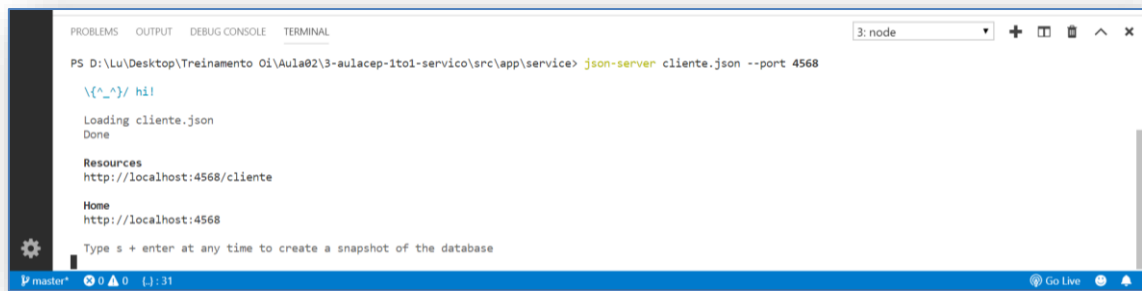
O Terminal será aberto dentro do diretório onde se encontra o arquivo.

Digitar no terminal:

**“json-server cliente.json”** (json-server [nome do comando] [arquivo.json])

ou **“json-server cliente.json –port 4568”** (para mudar a porte, se já estiver em uso)

**“json-server cliente.json”**



```
PS D:\Lu\Desktop\Treinamento 01\Aula02\3-aulacep-1to1-servico\src\app\service> json-server cliente.json --port 4568

\{^_^}/ hi!

Loading cliente.json
Done

Resources
http://localhost:4568/cliente

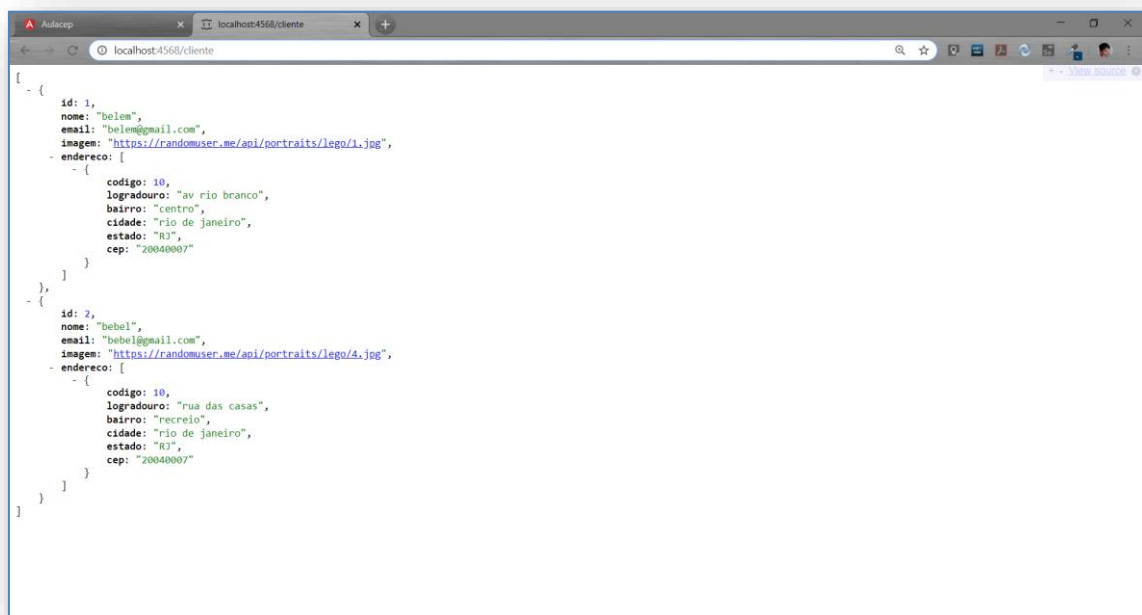
Home
http://localhost:4568

Type s + enter at any time to create a snapshot of the database
```

Clicar no link gerado pelo json-server

<http://localhost:4568/cliente>

Criou o serviço de cliente com todos os dados do arquivo



```
[
  {
    id: 1,
    nome: "belem",
    email: "belem@gmail.com",
    imagen: "https://randomuser.me/api/portraits/lego/1.jpg",
    endereco: [
      {
        codigo: 10,
        logradouro: "av rio branco",
        bairro: "centro",
        cidade: "rio de janeiro",
        estado: "RJ",
        cep: "20040007"
      }
    ]
  },
  {
    id: 2,
    nome: "bebel",
    email: "bebel@gmail.com",
    imagen: "https://randomuser.me/api/portraits/lego/4.jpg",
    endereco: [
      {
        codigo: 10,
        logradouro: "rua das casas",
        bairro: "recreio",
        cidade: "rio de janeiro",
        estado: "RJ",
        cep: "20040007"
      }
    ]
  }
]
```

## cliente.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

//CRIAR A CONTANTE COM A URL DO SERVIDOR CRIADO
const CLIENTES = "http://localhost:4568/cliente";

@Injectable() //INJEÇÃO DE DEPENDENCIA, JA ESTÁ ALOCADO NA
MEMORIA
export class ClienteService {
```



```

//NO CONSTRUTOR PASSANDO A CLASSE DE CONEXÃO A INTERNET
constructor(private http: HttpClient) {

}

//METODO PARA BUSCAR TODOS PELO WEBSERVICE
public findAll() {
    return this.http.get<any>(`${CLIENTES}`);
}
}

```

---

## endereco.service.ts

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

//CRIAR A CONTANTE COM A URL DO SERVIDOR DOS CORREIOS
const URL = "https://api.postmon.com.br/v1/cep";

@Injectable() //INJEÇÃO DE DEPENDENCIA
export class EnderecoService{

    //CONSTRUTOR HABILITANDO A CONEXÃO COM A INTERNET
    constructor(private http: HttpClient){

    }

    //METODO DE BUSCAR O CEP, VIA INTERNET (GET), PASSANDO A URL
    E O CEP
    public buscaCep(vcep: string){
        return this.http.get<any>(`${URL}/${vcep}`);
    }
}

```

---

## Instalar o Material Desing Bootstrap

Digitar no terminal:

**"npm i npm-registry-client"**

**"ng add angular-bootstrap-md"**

## app.module.ts

```
import { ClienteService } from './service/cliente.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { MDBBootstrapModule } from 'angular-bootstrap-md';
import { ClienteComponent } from './cliente/cliente.component';
import { EnderecoComponent } from
'./endereco/endereco.component';
import { HttpClientModule } from '@angular/common/http';
import { EnderecoService } from './service/endereco.service';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    ClienteComponent, //COMPONENTE CLIENTE ADICIONADO
    EnderecoComponent //COMPONENTE ENDERECO ADICIONADO
  ],
  imports: [
    BrowserModule,
    MDBBootstrapModule.forRoot(), //MATERIAL DESING
    BOOTSTRAP
    HttpClientModule, //ACESSO A INTERNET
    FormsModule //HABILITA O USO DE FORMULARIO
  ],
  //SERVIÇOS QUE IREMOS UTILIZAR
  providers: [EnderecoService, ClienteService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---

## app.component.html

```
<!-- CABEÇALHO QUE IRÁ APARECER EM TODOS OS COMPONENTES -->
<h1 class="bg-dark text-white p-3 text-center">Projeto
Serviços</h1>
<!-- CRIAMOS UM LINHA PARA COLOCAR OS COMPONENTES LADO A LADO --
>
<div class="row">
  <!-- DIVIDINDO NA METADE DA TELA -->
```

```

<div class="col-md-6">
  <!-- PASSANDO O COMPONENTE DE ENDEREÇO -->
  <app-endereco></app-endereco>
</div>
<div class="col-md-6">
  <app-cliente></app-cliente>
</div>
</div>

```

---

## endereco.component.ts

```

import { Component, OnInit } from '@angular/core';
import { Endereco } from '../model/endereco';
import { EnderecoService } from '../service/endereco.service';

@Component({
  selector: 'app-endereco',
  templateUrl: './endereco.component.html',
  styleUrls: ['./endereco.component.css']
})
export class EnderecoComponent implements OnInit {

  //CRIA A VARIÁVEL DE ENDEREÇO
  endereco: Endereco;
  msg = ""; //VARIÁVEL DE MSG JÁ INICIADO EM BRANCO

  //NO CONSTRUTOR DA ACESSO AO SERVIÇO DE ENEDEREÇO
  constructor(public service: EnderecoService) {
    //DA ESPAÇO DE MEMÓRIA PARA ENDEREÇO
    this.endereco = new Endereco();
  }

  ngOnInit() {

  }

  //METODO DO BOTÃO DE BUSCAR O CEP
  public mostraCep(){
    //SE O CEP FOR DIFERENTE DE NULL
    if(this.endereco.cep !== null){
      //VAI NO SERVIÇO E BUSCA O CEP
      this.service.buscaCep(this.endereco.cep).subscribe(res => {
        //RETORNANDO O ENDEREÇO
        this.endereco = res;
        //MOSTRA NO CONSOLE A MSG
        console.log("CEP Encontrado!")
      })
    }
  }
}

```

```

    });
  }else{ //SE NAO
    this.msg = "CEP nao encontrado";
    console.log("CEP não encontrado");
  }
}
}
}

```

---

## endereco.component.html

```

<div class="container-fluid">

  <h2> Endereço API Postmon</h2>
  <br>
  <!-- FORMATANDO UM CARD (CARTÃO) PARA O FORMULARIO, COM
  LARGURA DE 10 COLUNAS E CENTRALIZADO-->
  <md-card class="col-md-10 offset-md-1">
    <!-- CRIANDO O CORPO DO CARD -->
    <md-card-body>
      <!-- TITULO DO CARD -->
      <md-card-title>
        <h4>Busca CEP</h4>
      </md-card-title>
      <!-- CRIA UMA LINHA PARA POR O CAMPO E O BOTÃO -->
      <div class="row">
        <!-- CAMPO DE INPUT COM LARGURA DE 8 COLUNAS -->
        <div class="col-md-8">
          <input mdbInput type="number"
placeholder="Cep" [(ngModel)]="endereco.cep" name="cep"
class="form-control">
        </div>
        <!-- CAMPO DO BOTÃO TERÁ LARGURA DE 4 COLUNAS --
        <div class="col-md-4">
          <!-- BOTÃO DE COR AZUL E TAMANHO SMALL -->
          <button (click)="mostraCep()" mdbBtn
color="primary" size="sm">Buscar CEP</button>
        </div>
        <!-- CAMPOS DE ENDEREÇO QUE SERÁ RESGATADO -->
        <div class="col-md-12">
          <input mdbInput type="text"
name="logradouro" [(ngModel)]="endereco.logradouro"
placeholder="Logradouro" class="form-control mb-1" />

```

```

        <input mdbInput type="text" name="bairro"
[(ngModel)]="endereco.bairro" placeholder="Bairro" class="form-
control mb-1"/>
        <input mdbInput type="text" name="cidade"
[(ngModel)]="endereco.cidade" placeholder="Cidade" class="form-
control mb-1"/>
        <input mdbInput type="text" name="estado"
[(ngModel)]="endereco.estado" placeholder="Estado" class="form-
control mb-1"/>
    </div>
</div>
</mdb-card-body>
</mdb-card>

</div>

```

---

## cliente.component.ts

```

import { Component, OnInit } from '@angular/core';
import { ClienteService } from '../service/cliente.service';
import { Cliente } from '../model/cliente';

@Component({
  selector: 'app-cliente',
  templateUrl: './cliente.component.html',
  styleUrls: ['./cliente.component.css']
})
export class ClienteComponent implements OnInit {

  //CRIANDO AS VARIÁVEIS DE CLIENTE E DA LISTA, INICIANDO O
  VETOR
  cliente: Cliente;
  clientes: Cliente[] = [];

  //CONSTRUTOR ACESSANDO O SERVIÇO DO CLIENTE
  constructor(public service: ClienteService) {
    //ESPAÇO DE MEMÓRIA PARA O CLIENTE
    this.cliente = new Cliente();
  }

  //AO INICIAR O COMPONENTE SERÁ CARREGADA A LISTA
  ngOnInit() {
    this.listarTodos();
  }
}

```

```

//METODO PARA LISTAR
public listarTodos() {
    //IRÁ NO SERVIÇO BUSCAR TODOS E ENCHER A LISTA CLIENTES
    return this.service.findAll().subscribe(res => {
        this.clientes = res;
    })
}
}

```

## cliente.component.html

```

<div class="container-fluid">

    <h2>Cliente Json-Server</h2>

    <!-- ATRAVES DA DIRETIVA NGFOR FAZEMOS UMAVARREDIRA NA LISTA
    DE CLIENTES -->
    <div *ngFor="let item of clientes">
        <!-- FORMATANDO UM CARD (CARTÃO) PARA SER PREENCHIDO COM
        A LISTA DE CLIENTES, QUE OCUPARÁ METADE DA TELA (6 COLUNAS) E
        TERÁ UMA MARGEM INFERIOR DE 30PX -->
        <md-card class="col-md-6 mb-3">
            <!-- CORPO DO CARD -->
            <md-card-body>
                <!-- TITULO DO CARD -->
                <md-card-title>
                    <!-- RESGATANDO O NOME -->
                    Nome: <h4>{{item.nome}}</h4>
                </md-card-title>
                <!-- IMAGEM DA LISTA, QUANDO PASSAR O MOUSE
                SOBRE A IMAGEM APARECERÁ O NOME DO CLIENTE (TITLE) E ESTILO
                FLUIDO, COM BORDA ARREDONDADA E SOMBRA -->
                <img src={{item.imagem}} title={{item.nome}}
                class="img-fluid rounded-circle shadow">
                <md-card-text class="mt-5">
                    <!-- MOSTAR EMAIL -->
                    <h5>Email: {{item.email}}</h5>
                    <!-- FAZ UM FOR DENTRO DE CLIENTE TARZENDO
                    OS CAMPOS DE ENDEREÇO -->
                    <div *ngFor="let linha of item.endereco">
                        <p>Logradouro: {{linha.logradouro}}</p>
                        <p>Bairro: {{linha.bairro}}</p>
                        <p>Cidade: {{linha.cidade}}</p>
                        <p>Estado: {{linha.estado}}</p>
                        <p>CEP: {{linha.cep}}</p>
                    </div>
                </md-card-text>
            </md-card-body>
        </md-card>
    </div>

```

```

        </div>
      </mdb-card-text>
    </mdb-card-body>
  </mdb-card>
</div>

```

```

<!-- MOSTRA A LISTA DE CLIENTES EM FORMATO JSON -->
<pre>
{{clientes | json}}
</pre>

```

```

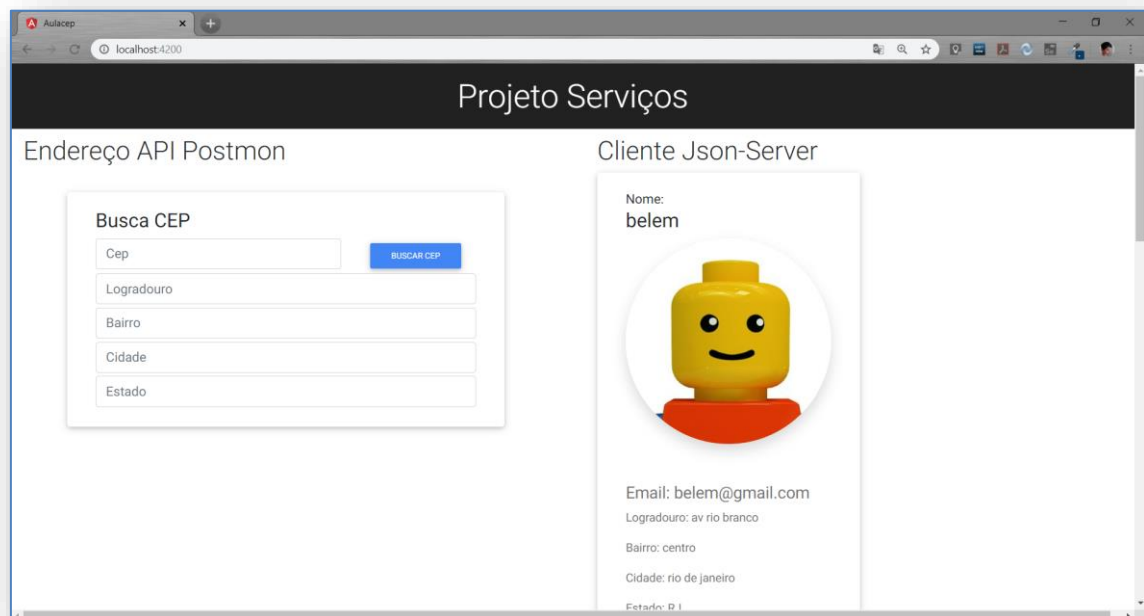
</div>

```

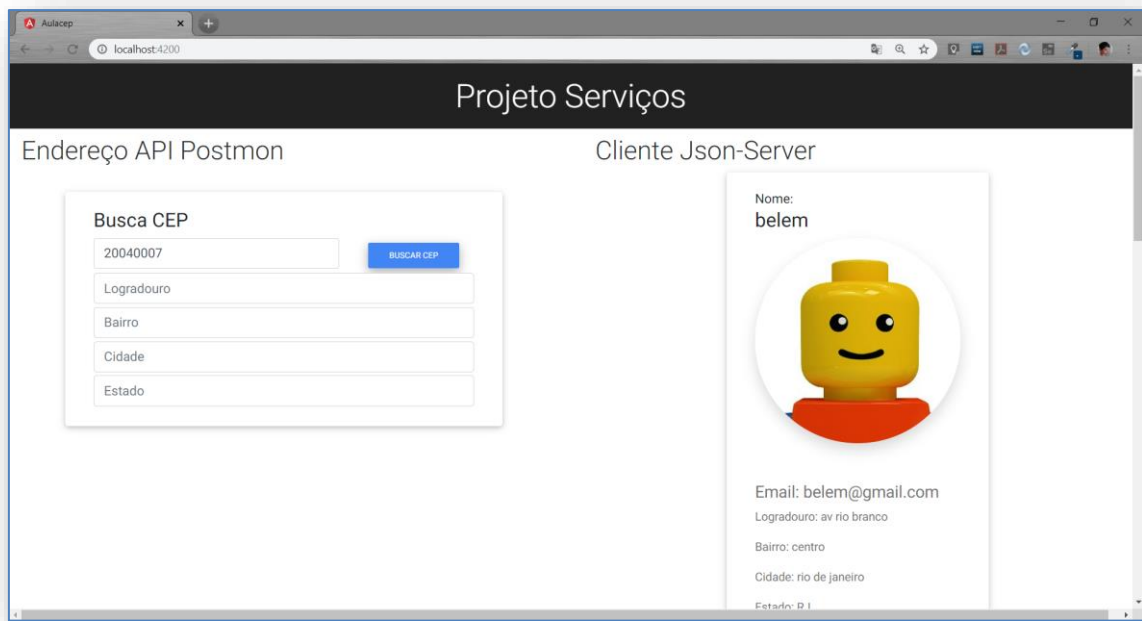
## Rodando o projeto.

Digitar no terminal:

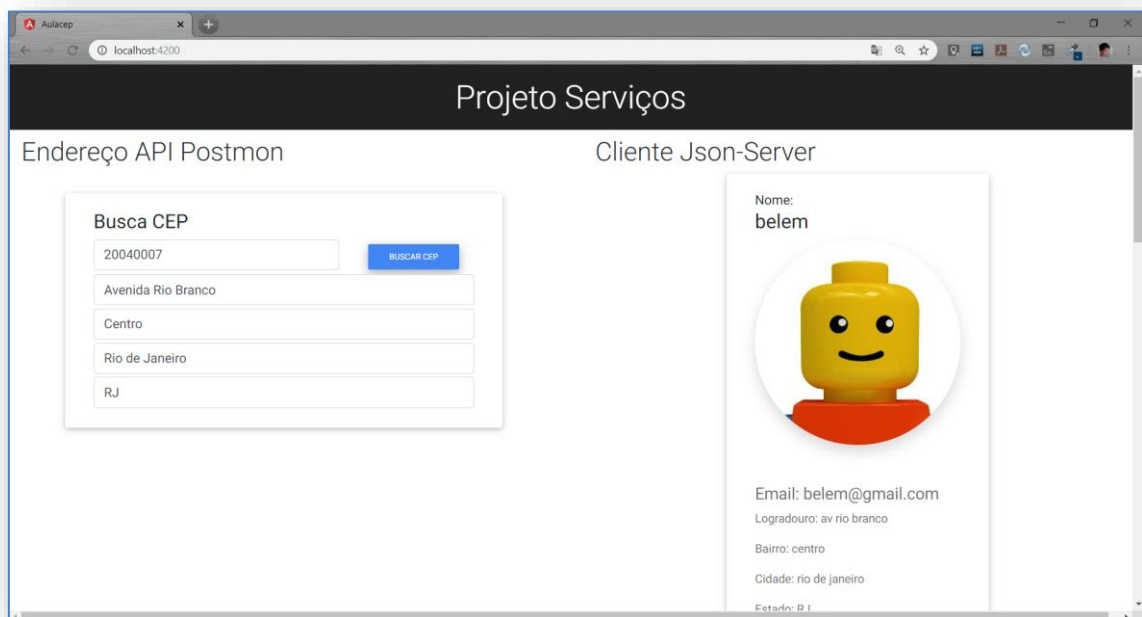
“ng s -o”



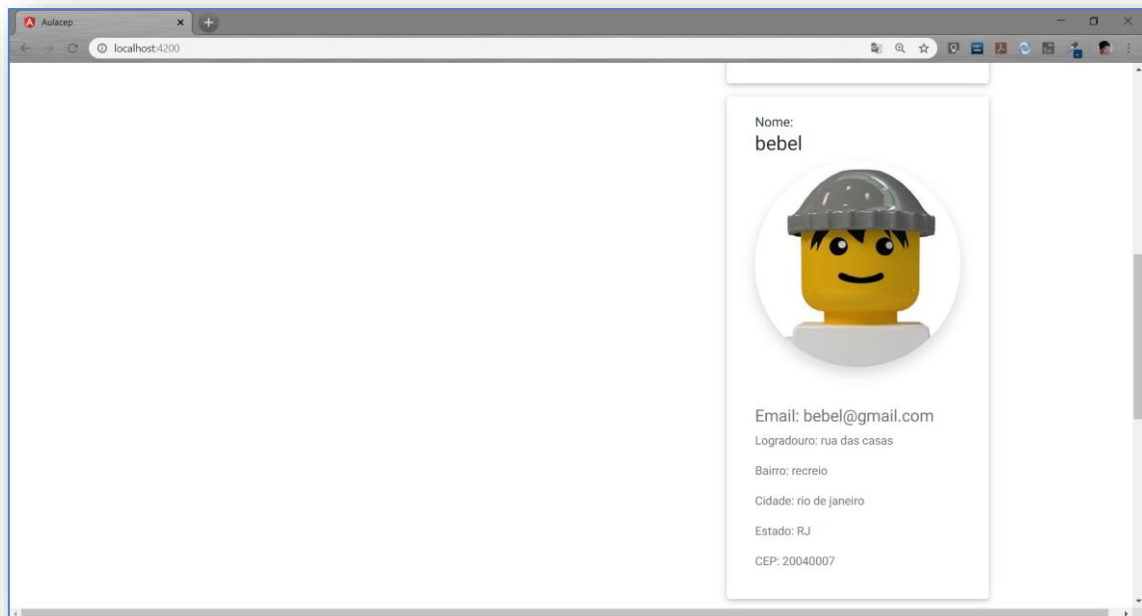
Digitar o cep e clicar no botão



Clicando no botão, trouxe os dados do servidor

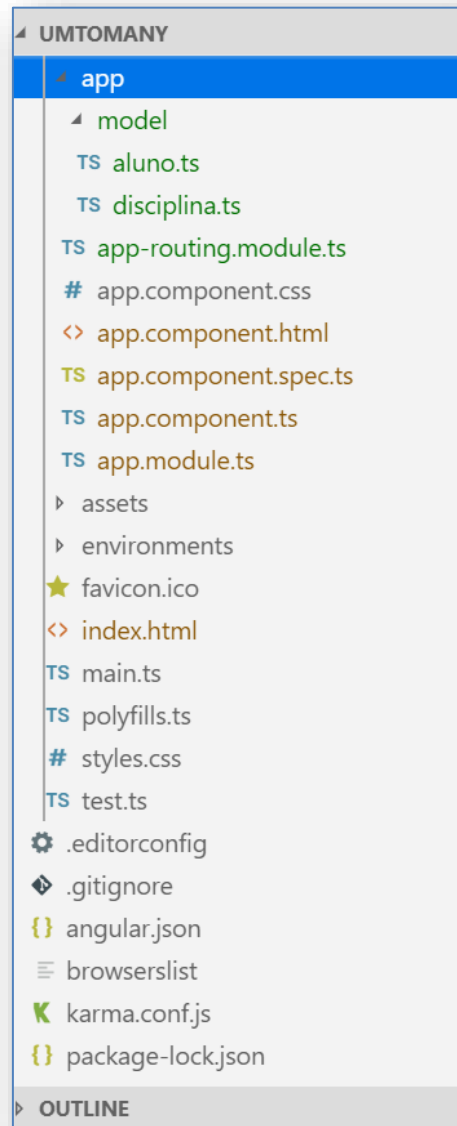






## NOVO PROJETO

Estrutura do projeto depois de finalizado:



## aluno.ts

```
import { Disciplina } from './disciplina';

export class Aluno {

    id: number;
    nomeAluno: string;

    //RELACIONAMENTO UM PARA MUITOS. O ALUNO TEM MUITAS DISCIPLINAS
    disciplinas: Disciplina[] = [];

    constructor(id?: number, nomeAluno?: string) {
        this.id = id;
        this.nomeAluno = nomeAluno;
    }

    //METODO ADICIONAR DISCIPLINAS
    public adicionar(disciplina: Disciplina): Aluno {
        if (!this.disciplinas) { //SE A LISTA NOA EXISTE, DÁ ESPAÇO DE MEMORIA PARA ELA
            this.disciplinas = [];
        }
        //ADICIONA DISCIPLINA ATRAVES DO PUSH NAS LISTA
        this.disciplinas.push(disciplina);
        // RETORNA A LISTA
        return this;
    }
}
```

---

## disciplina.ts

```
export class Disciplina {

    idDisciplina: number;
    nomeDisciplina: string;
    nota1: number;
    nota2: number;
    media: number;
    situacao: string;

    //METODO PARA CALCULA A MEDIA
    public gerarMedia(): Disciplina {
```

```

        this.media = (this.nota1 + this.nota2) / 2;
        return this;
    }

    //METODO DE CALCULAR A SITUAÇÃO
    public gerarSituacao(): Disciplina {
        this.situacao = (this.media >= 8) ? "aprovado" :
"reprovado";
        return this;
    }
}

```

---

## app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    FormsModule, //HABILITANDO O USO DO FORMULARIO
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

---

## app.component.ts

```

import { Component } from '@angular/core';
import { Aluno } from './model/aluno';

```

```

import { Disciplina } from '../model/disciplina';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  //LISTA DE ALUNOS
  listaAluno = [{
    id: 10,
    nomeAluno: "pedro"
  }, {
    id: 11,
    nomeAluno: "lu"
  }, {
    id: 12,
    nomeAluno: "rafa"
  }
  ] as Aluno[];

  // LISTA DE DISCIPLINA
  listaDisciplina = [{
    idDisciplina: 1000,
    nomeDisciplina: "java",
    nota1: 0,
    nota2: 0,
    media: 0,
    situacao: ""
  },
  {
    idDisciplina: 1001,
    nomeDisciplina: "angular",
    nota1: 0,
    nota2: 0,
    media: 0,
    situacao: ""
  }
  ] as Disciplina[];

  //CRIANDO AS VARIÁVEIS
  aluno: Aluno;
  alunos: Aluno[] = [];
  disciplinas: Disciplina[] = []

  //INICIALIZANDO NO CONSTRUTOR
  constructor() {

```

```

        this.disciplinas = [] //ESPAÇO PARA A LISTA DE
DISCIPLINAS
        this.aluno = new Aluno(); //ESPAÇO PARA O ALUNO
        this.aluno.disciplinas = []; //ESPAÇO PARA AS
DISCIPLINAS PRELACIONADAS COM ALUNO
        this.alunos = []; //ESPAÇO PARA A LISTA DE ALUNOS
    }

    //METODO PARA ALOCAR O ALUNO, PASSANDO ALUNOREF COMO
PARAMETRO
    public alocarAluno(alunoref: Aluno) {
        this.aluno = alunoref;
    }

    //METODO DE ADICIONAR E CALCULAR AS DISCIPLINAS
    public adicionarDisciplina(disciplina: Disciplina) {
        //CALCULAR A MEDIA
        disciplina.media = (+disciplina.nota1 +
+disciplina.nota2) / 2;
        //SE A MEDIA FOR MAIOR QUE 7 A SITUAÇÃO SERÁ APROVADO
        if (+disciplina.media >= 7) {
            disciplina.situacao = "aprovado";
        } else { //SENÃO REPROVADO
            disciplina.situacao = "reprovado";
        }
        //ADICIONA A DISCIPLINA AO VETOR
        this.disciplinas.push(disciplina);
    }

    //ALOCA A DISCIPLINA
    public alocarDisciplina(disciplina: Disciplina) {
        this.adicionarDisciplina(disciplina);
        //atribuo a lista ao Aluno
        this.aluno.disciplinas = this.disciplinas;
    }

    //ADICIONA O ALUNO AO VETOR E ZERA
    public adicionarGrupoAlunos() {
        this.alunos.push(this.aluno); //armazeno
        this.aluno = new Aluno(); //zero
        this.aluno.disciplinas = []; //zero
        this.disciplinas = [];
    }
}

```

## app.component.html

```
<div class="container-fluid">

  <H2 class="jumbotron text-center bg-light">
    Relacionamento OneToMany
  </H2>

  <div class="row">
    <div class="col-md-4">
      <h3 class="text-center">Alunos <mdb-icon fas
icon="user-plus"></mdb-icon>
      </h3>
      <mdb-card *ngFor="let item of listaAluno" class="mb-
3 border-info shadow">
        <mdb-card-body>
          Id:
          <input mdbInput type="number" #id
[(ngModel)]="item.id" title='digite o Id'
            class="form-control mb-2" />
          Nome:
          <input mdbInput type="text" #nome
[(ngModel)]="item.nomeAluno" title='digite o nome'
            class="form-control mb-2" />
          <button (click)="alocarAluno(item)" mdbBtn
color='primary' size='sm'>Alocar o Aluno</button>
        </mdb-card-body>
      </mdb-card>
    </div>

    <div class="col-md-4">
      <h3 class="text-center">Disciplinas <mdb-icon fas
icon="book-open"></mdb-icon>
      </h3>
      <mdb-card *ngFor="let linha of listaDisciplina"
class="mb-3 border-success shadow">
        <mdb-card-body>
          IdDisciplina:
          <input type="number" #idDisciplina
[(ngModel)]="linha.idDisciplina" mdbInput class="form-control"/>
          NomeDisciplina:
```

```

        <input type="text" #nomeDisciplina
[(ngModel)]= "linha.nomeDisciplina" mdbInput class="form-
control"/>
        Nota1:
        <input type="number" #nota1
[(ngModel)]= "linha.nota1" mdbInput class="form-control"/>
        Nota2:
        <input type="number" #nota2
[(ngModel)]= "linha.nota2" mdbInput class="form-control"/>
        Media:
        <input type="number" #media
[(ngModel)]= "linha.media" mdbInput class="form-control"/>
        Situacao:
        <input type="text" #media
[(ngModel)]= "linha.situacao" mdbInput class="form-control"/>
        <button (click)="alocarDisciplina(linha)"
mdbBtn size='sm' color='success' class="mt-3">Alocar
Disciplina</button>
    </mdb-card-body>
</mdb-card>
    <button (click)="adicionarGrupoAlunos()" mdbBtn
size='sm' color='secondary'>Adicionar Aluno ao Grupo</button>
</div>

<div class="col-md-4">
    <h3 class="text-center">Saída Aluno <mdb-icon fas
icon="user-graduate"></mdb-icon></h3>
    <b>Aluno:</b> {{aluno | json}}
    <br />
    <b>Disciplinas:</b>
    {{aluno.disciplinas | json}}

    <h3 class="text-center mt-4">Relatorio Grupo <mdb-
icon fas icon="users"></mdb-icon></h3>
    <pre>
    {{alunos | json}}
    </pre>
    </div>
</div>
</div>

```



Rodando o projeto.

Digitar no terminal:

"ng s -o"

<http://localhost:4200/>

The screenshot shows a web browser at localhost:4200 displaying a web application titled "Relacionamento OneToMany". The interface is divided into three main sections: "Alunos" (Students), "Disciplinas" (Subjects), and "Saída Aluno" (Student Output). The "Alunos" section has two forms for adding students. The first form has "Id:" set to 10 and "Nome:" set to "pedro", with a blue "ALOCAR O ALUNO" button. The second form has "Id:" set to 11 and "Nome:" set to "lu", with a blue "ALOCAR O ALUNO" button. The "Disciplinas" section has a form for adding subjects. It has "IdDisciplina:" set to 1000, "NomeDisciplina:" set to "java", "Nota1:" set to 0, "Nota2:" set to 0, "Media:" set to 0, and "Situacao:" set to an empty field. There is a green "ALOCAR DISCIPLINA" button. The "Saída Aluno" section shows the output of the system. It displays "Aluno: { 'disciplinas': [] }" and "Disciplinas: []". Below this, there is a "Relatorio Grupo" button with a group icon.

Clicando em um aluno

The screenshot shows the same web application interface as before, but with updated data in the "Saída Aluno" section. The "Aluno" object now contains the selected student: "Aluno: { 'id': 10, 'nomeAluno': 'pedro' }". The "Disciplinas" array remains empty: "Disciplinas: []". The "Relatorio Grupo" button is still present.

Digitando as notas da disciplina e clicando em alocar disciplina

Relacionamento OneToMany

**Alunos** 👤

Id: 10  
Nome: pedro  
ALOCAR O ALUNO

Id: 11  
Nome: lu  
ALOCAR O ALUNO

**Disciplinas** 📖

IdDisciplina: 1000  
NomeDisciplina: java  
Nota1: 10  
Nota2: 8  
Media: 9  
Situacao: aprovado  
ALOCAR DISCIPLINA

**Saída Aluno** 👤

Aluno: { "id": 10, "nomeAluno": "pedro", "disciplinas": [ { "idDisciplina": 1000, "nomeDisciplina": "java", "nota1": 10, "nota2": 8, "media": 9, "situacao": "aprovado" } ] }

Disciplinas: [ { "idDisciplina": 1000, "nomeDisciplina": "java", "nota1": 10, "nota2": 8, "media": 9, "situacao": "aprovado" } ]

**Relatorio Grupo** 👤

[ ]

Clicando em adicionar aluno ao grupo

Id: 11  
Nome: lu  
ALOCAR O ALUNO

Id: 12  
Nome: rafa  
ALOCAR O ALUNO

Media: 9  
Situacao: aprovado  
ALOCAR DISCIPLINA

IdDisciplina: 1001  
NomeDisciplina: angular  
Nota1: 0  
Nota2: 0  
Media: 0  
Situacao:  
ALOCAR DISCIPLINA

ADICIONAR ALUNO AO GRUPO

## Resultado

Relacionamento OneToMany

Alunos 👤

Id: 10

Nome: pedro

ALOCAR O ALUNO

Disciplinas 📖

IdDisciplina: 1000

NomeDisciplina: java

Nota1: 10

Nota2: 8

Media: 9

Situacao: aprovado

ALOCAR DISCIPLINA

Saída Aluno 👤

Aluno: { "disciplinas": [] }

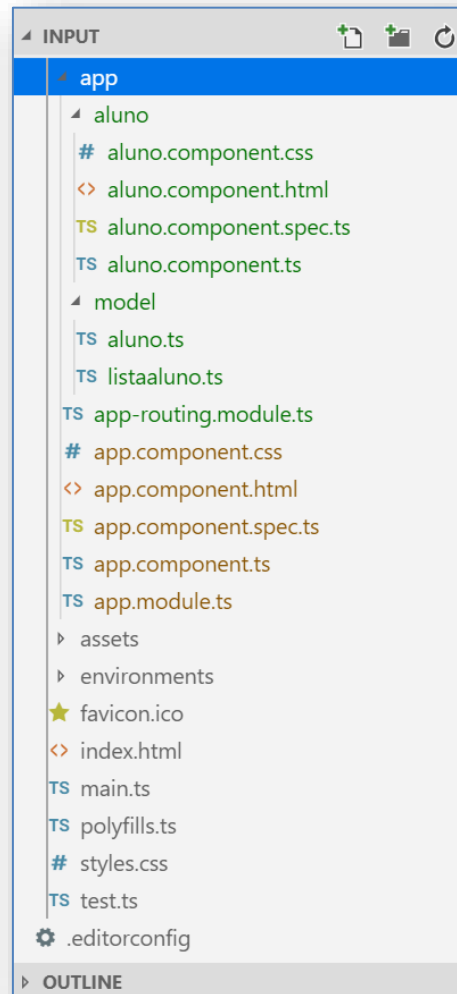
Disciplinas: []

Relatorio Grupo 👤

```
[
  {
    "id": 10,
    "nomeAluno": "pedro",
    "disciplinas": [
      {
        "idDisciplina": 1000,
        "nomeDisciplina": "java",
        "nota1": 10,
        "nota2": 8,
        "media": 9,
        "situacao": "aprovado"
      }
    ]
  }
]
```

## NOVO PROJETO

Estrutura do projeto depois de finalizado:



### aluno.ts

```
export class Aluno {  
  
  id: number;  
  nome: string;  
  email: string;  
  
  constructor(id?: number, nome?: string, email?: string) {  
    this.id = id;  
    this.nome = nome;  
    this.email = email;  
  }  
}
```

## listaaluno.ts

```
import { Aluno } from './aluno';

export class ListaAluno {

    idlista: number;
    turma: string;
    alunos: Array<Aluno>;

    construtor(idlista?: number, turma?: string) {
        this.idlista = idlista;
        this.turma = turma;
    }

}
```

---

## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AlunoComponent } from './aluno/aluno.component';
import { FormsModule } from "@angular/forms";

@NgModule({
  declarations: [
    AppComponent,
    AlunoComponent
  ],
  imports: [
    FormsModule,
    BrowserModule,
    AppRoutingModuleModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---

## app.component.ts

```
import { ListaAluno } from '../model/listaaluno';
import { Component } from '@angular/core';
import { Aluno } from '../model/aluno';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'input';

  //VETOR COM A LISTA DE ALUNOS
  listaAluno = [
    {
      idlista: 2000,
      turma: "java developer",
      alunos: [
        {
          id: 10,
          nome: "lu",
          email: "lu@gmail.com"
        },
        {
          id: 11,
          nome: "eu",
          email: "eu@gmail.com"
        },
        {
          id: 12,
          nome: "biel",
          email: "biel@gmail.com"
        }
      ]
    }
  ] as ListaAluno[];

  //CRIANDO A VARIÁVEL
  umaluno: Aluno;

  construtor() {
    this.umaluno = new Aluno();
  }

  //RECEBE OS DADOS DE ALUNO
```

```

    public associar(recebealuno: Aluno) {
        this.umaluno = recebealuno;
    }
}

```

---

## app.component.html

```

<h2>Projeto Input</h2>

<div *ngFor="let item of listaAluno" class="painel">
    <div class="card">
        <h3>Disciplina:</h3>
        IdLista: <br />
        <input type="number" name="idlista"
[(ngModel)]="item.idlista" />
        <br />
        Turma:<br />
        <input type="text" name="turma" [(ngModel)]="item.turma"
/>
    </div>
    <div class="card" *ngFor="let escolhe of item.alunos">
        <h3>Aluno:</h3>
        Id:<br />
        <input type="number" name="id" [(ngModel)]="escolhe.id"
/>
        <br />
        Nome:<br />
        <input type="text" name="nome"
[(ngModel)]="escolhe.nome" />
        <br />
        Email:<br />
        <input type="email" name="email"
[(ngModel)]="escolhe.email" />
        <button (click)="associar(escolhe)">Escolha</button>
    </div>
</div>

<!-- RECEBERÁ OS DADOS DO ALUNO CLICADO -->
<app-aluno [outroAluno]="umaluno"></app-aluno>

```

---

## app.component.css

```
h2{
  text-align: center;
  padding: 20px;
  background-color: #ccc;
  text-transform: uppercase;
}
.card{
  border: 1px solid #ccc;
  border-radius: 10px;
  padding: 10px;
  width: 200px;
  margin-bottom: 10px;
  margin-left: 50px;
}
.painel{
  float: left;
  width: 500px;
}
```

---



## aluno.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Aluno } from '../model/aluno';

@Component({
  selector: 'app-aluno',
  templateUrl: './aluno.component.html',
  styleUrls: ['./aluno.component.css']
})
export class AlunoComponent implements OnInit {

  //INPUT COM A VARIÁVEL QUE RECEBA OS DADOS
  @Input() outroAluno: Aluno;

  constructor() {

  }

  ngOnInit() {
  }
}
```

---

## aluno.component.html

```
<h2>Detalhes do aluno: </h2>
<br/>
<div *ngIf="outroAluno">
  {{outroAluno.id |
json}},{{outroAluno.nome}},{{outroAluno.email}}
</div>
```

---

Rodando o projeto:

Digitando no terminal:

**"ng s -o"**

<http://localhost:4200/>

The screenshot shows a web browser window with the address bar at `localhost:4200`. The page title is "PROJETO INPUT". The main content area is titled "Detalhes do aluno:". On the left side, there are three form sections:

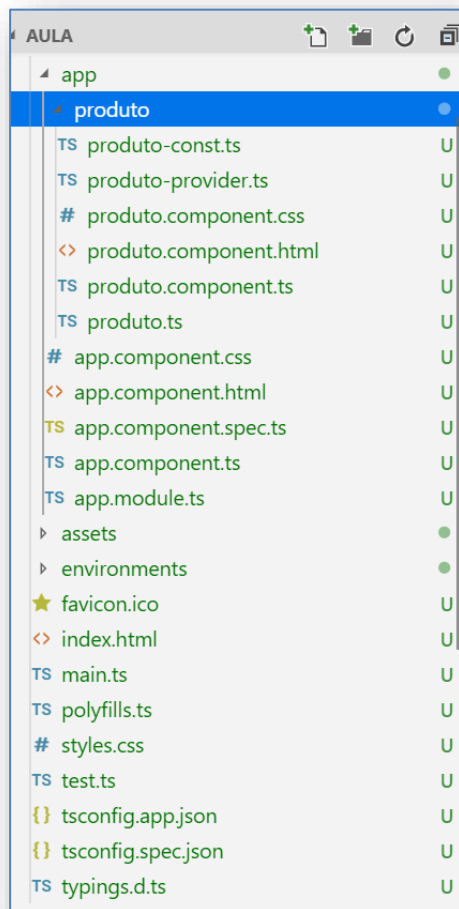
- Disciplina:** Contains three input fields: "IdLista:" with value "2000", "Turma:" with value "java developer", and an empty "Escolha" button.
- Aluno:** Contains four input fields: "Id:" with value "10", "Nome:" with value "lu", "Email:" with value "lu@gmail.com", and an "Escolha" button.
- Aluno:** Contains four input fields: "Id:" with value "11", "Nome:" with value "eu", "Email:" with value "eu@gmail.com", and an "Escolha" button.

Clicando em escolher um aluno

The screenshot shows the same web browser window after clicking the "Escolha" button. The "Detalhes do aluno:" section now displays the email address "10.lu.lu@gmail.com". The form sections on the left remain unchanged.

## NOVO PROJETO

Estrutura do projeto depois de finalizado:



## produto.ts

```
export class Produto {

    codigo: number;
    nome: string;
    preco: number;

    //SOBRECARGA DE CONSTRUTORES
    constructor()

    constructor(codigo: number, nome: string, preco: number)

    constructor(codigo?: number, nome?: string, preco?: number) {
        this.codigo = codigo;
        this.nome = nome;
        this.preco = preco;
    }

    get $preco() {
        return this.preco;
    }

    set $preco(preco: number) {
        this.preco = preco;
    }

}
```

---

## produto-const.ts

```
export class ProdutoConstante {  
  
    public static PRODUTOS = [  
        {  
            codigo: 1,  
            nome: 'redbull blue',  
            preco: 6.5  
        },  
        {  
            codigo: 2,  
            nome: 'redbull lilas',  
            preco: 14  
        },  
        {  
            codigo: 3,  
            nome: 'redbull red',  
            preco: 12  
        },  
        {  
            codigo: 4,  
            nome: 'redbull 475',  
            preco: 22  
        },  
        {  
            codigo: 10,  
            nome: 'monster',  
            preco: 3  
        }  
    ];  
  
}
```

---

## produto.provider.ts

```
import { ProdutoConstante } from './produto-const';
import { Injectable } from '@angular/core';
import { Produto } from './produto';

@Injectable()
export class ProdutoService {

    private produtos = ProdutoConstante.PRODUTOS;

    getProdutoByCode(codigo): object {
        return this.produtos.filter(obj => {
            return obj.codigo == codigo;
        })[0];
    }

    getProdutoAll() {
        return this.produtos;
    }

    //ocultar o preco
    mapProdutos() {
        return this.produtos.map(item => {
            //delete item.preco;
            return item;
        })
    }

    getProdutosByText(searchText): Produto[] {
        if (searchText == null) { return []; }
        this.produtos = ProdutoConstante.PRODUTOS;
        return this.produtos.filter(item => {
            return item.nome.indexOf(searchText) > -1;
        }) as Produto[];
    }
}
```

```
}
```

---

## produto.component.ts

```
import { ProdutoService } from './produto-provider';
import { ProdutoConstante } from './produto-const';
import { Component } from '@angular/core';
import { Produto } from './produto';

@Component({
  selector: 'app-produto',
  templateUrl: './produto.component.html',
  styleUrls: ['./produto.component.css']
})
export class ProdutoComponent {
  searchText: string;
  produtoMap: Array<any>;
  produto: Produto;
  produtos: Produto[];

  constructor(private service: ProdutoService) {
    //lista
    this.produtoMap = this.service.mapProdutos();
  }

  changed() {
    //lista
    this.produtos =
this.service.getProdutosByText(this.searchText);
  }
}
```

---

## app.component.html

```
<app-produto></app-produto>
```

---

## app.module.ts

```
import { ProdutoComponent } from './produto/produto.component';
import { ProdutoService } from './produto/produto-provider';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
    ProdutoComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [ProdutoService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---



# index.html

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Projeto5</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootst
rap.min.css">
  <script type="text/javascript"
src="https://code.jquery.com/jquery-3.1.0.js"></script>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.m
in.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstra
p.min.js"></script>

  <script type="text/javascript"
src="https://cdn.datatables.net/1.10.12/js/jquery.dataTables.min
.js"></script>
  <script type="text/javascript"
src="https://cdn.datatables.net/1.10.12/js/dataTables.bootstrap.
min.js"></script>
</head>

<body>
  <app-root></app-root>

  <script type="text/javascript">
    $(document).ready(function () {
      $('#tabela').dataTable({
        "lengthMenu": [[4, 8, 12, 50, -1], [4, 8, 12,
50, "Todos"]],
```

```

        "language": {
            "url":
"cdn.datatables.net/plugins/9dcbeed42ad/i18n/Portuguese.json"
        }
    });
});
</script>

```

```
</body>
```

```
</html>
```

---

## produto.component.html

```

<div class="container">
  <div class="well">
    Projeto Produtos Angular4
  </div>

  <!-- COLUNA LADO ESQUERDO -->
  <div class="col-md-6 col-xs-12">
    <div class="panel panel-primary">
      <div class="panel-body">
        <b>Digite o produto</b>
        <input type="text" [(ngModel)]="searchText"
(keyup)="changed()" name="searchText"
placeholder="Buscar ..." class="form-
control" />
        <br>
        <ul *ngFor="let item of produtos">
          <li> {{item.codigo}}, {{item.nome}}
,{{item.preco}} </li>
        </ul>
      </div>
    </div>
  </div>

  <!-- COLUNA DA DIREITA -->

```

```

<div class="col-md-6 col-xs-12">
  <div class="panel panel-info">
    <div class="panel-heading">
      <div class="panel-title">
        Lista de Produtos em Json
      </div>
    </div>
    <div class="panel-body">
      <div *ngFor="let prod of produtoMap">
        <span>{{prod | json}}</span>
      </div>
    </div> <!-- FECHA PANEL-BODY -->
  </div> <!-- FECHA PANEL -->
</div>
<!--FECHA COL-MD -->

```

```

<table class="table table-hover" id="tabela">
  <thead>
    <tr>
      <th>Codigo</th>
      <th>Nome</th>
      <th>Preço</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let prod of produtoMap">
      <td>{{prod.codigo}}</td>
      <td>{{prod.nome}}</td>
      <td>{{prod.preco}}</td>
    </tr>
  </tbody>
</table>

```

```

</div>

```

---

## Para rodar o projeto:

No terminal digitar: `ng serve -o`. Mostrando formato web (desktop)

### Projeto Produtos Angular4

Digite o produto:

Lista de Produtos em Json

```
{ "codigo": 1, "nome": "redbull blue", "preco": 6.5 }  
{ "codigo": 2, "nome": "redbull lilas", "preco": 14 }  
{ "codigo": 3, "nome": "redbull red", "preco": 12 }  
{ "codigo": 4, "nome": "redbull 475", "preco": 22 }  
{ "codigo": 5, "nome": "monster", "preco": 3 }
```

Lista de Produtos com DataTable

Show  entries Search:

| Codigo | Nome          | Preço |
|--------|---------------|-------|
| 1      | redbull blue  | 6.5   |
| 2      | redbull lilas | 14    |
| 3      | redbull red   | 12    |
| 4      | redbull 475   | 22    |

Showing 1 to 4 of 5 entries

Previous **1** 2 Next

Digitando um produto (red) para pesquisar...

### Projeto Produtos Angular4

Digite o produto:

Lista de Produtos em Json

```
{ "codigo": 1, "nome": "redbull blue", "preco": 6.5 }  
{ "codigo": 2, "nome": "redbull lilas", "preco": 14 }  
{ "codigo": 3, "nome": "redbull red", "preco": 12 }  
{ "codigo": 4, "nome": "redbull 475", "preco": 22 }  
{ "codigo": 5, "nome": "monster", "preco": 3 }
```

Lista de Produtos com DataTable

Show  entries Search:

| Codigo | Nome          | Preço |
|--------|---------------|-------|
| 1      | redbull blue  | 6.5   |
| 2      | redbull lilas | 14    |
| 3      | redbull red   | 12    |
| 4      | redbull 475   | 22    |

Showing 1 to 4 of 5 entries

Previous **1** 2 Next

Na lista com Datatable pesquisando “mo”...

### Projeto Produtos Angular4

**Digite o produto:**

**Lista de Produtos em Json**

```
{ "codigo": 1, "nome": "redbull blue", "preco": 6.5 }  
{ "codigo": 2, "nome": "redbull illas", "preco": 14 }  
{ "codigo": 3, "nome": "redbull red", "preco": 12 }  
{ "codigo": 4, "nome": "redbull 475", "preco": 22 }  
{ "codigo": 5, "nome": "monster", "preco": 3 }
```

**Lista de Produtos com DataTable**

Show Todos ▼ **entries**

Search:

| Codigo | Nome    | Preço |
|--------|---------|-------|
| 5      | monster | 3     |

Showing 1 to 1 of 1 entries (filtered from 5 total entries)

Previous **1** Next