

# PROGRAMAÇÃO WEBII

1

# Tratamento de erros

Biblioteca ngx-toastr

# O que iremos aprender?

- Vamos aprender como melhorar a interface do listar produtos para o usuário.
- Atualmente a nossa aplicação está buscando os nossos dados no nosso backend, porém se o backend parar e atualizarmos o navegador, não teremos dados e não teremos erro, não saberemos o que está acontecendo.

# 0 que iremos aprender?

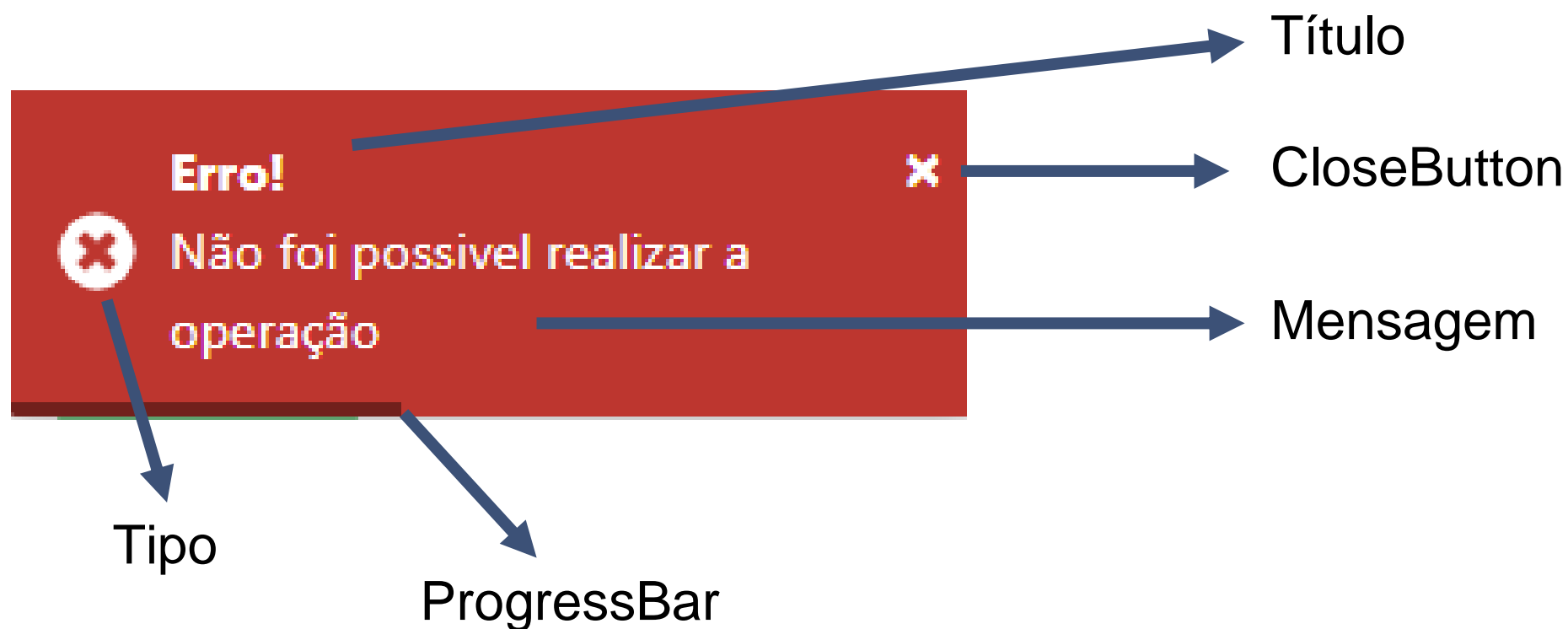
- Lembram do Observable?
- Segue site para lembrarmos:
- <https://angular.io/guide/rx-library>

## Biblioteca ngx-toastr

- Para melhorar a interação com o usuário, vamos fazer uma instalação com a biblioteca ngx-toastr.
- Essa biblioteca irá permitir que enviemos aos usuários mensagens para que ele possa ter uma noção do que possa estar acontecendo com a nossa aplicação.

# Biblioteca ngx-toastr

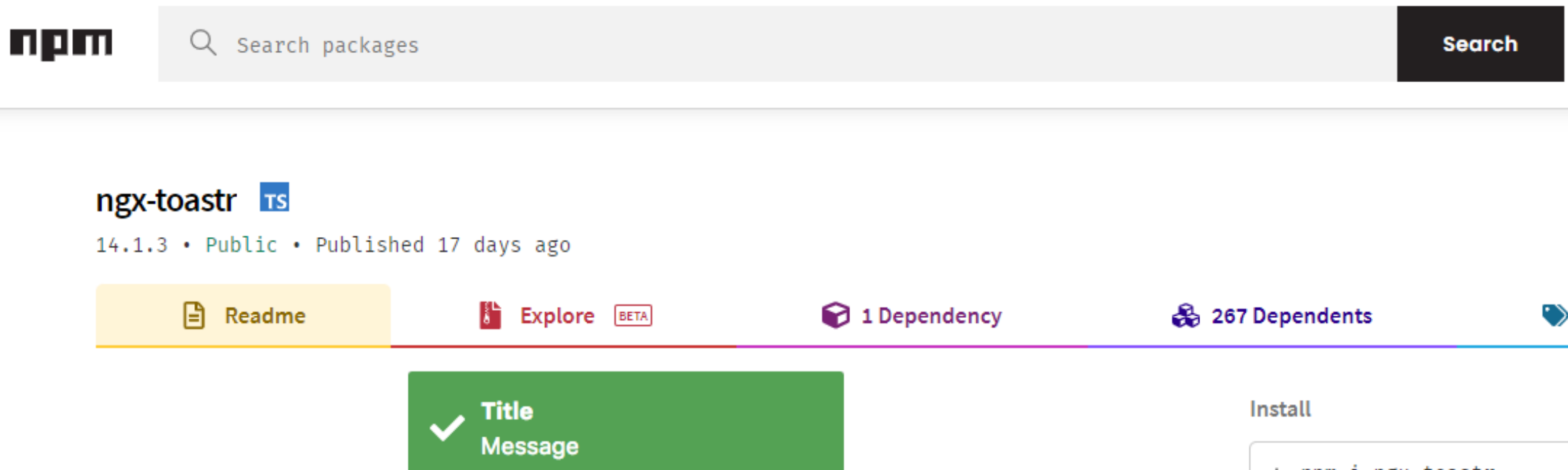
- Qual a mensagem iremos exibir na aplicação?



# Biblioteca ngx-toastr

- Para fazer a instalação dela, é muito simples. Vamos acessar o site:

► <https://www.npmjs.com/package/ngx-toastr>



The screenshot shows the npm website interface. At the top, there's a search bar with the npm logo on the left, a search input field containing "Search packages", and a "Search" button on the right. Below the search bar, the package "ngx-toastr" is displayed with a "TS" tag. Underneath the package name, it says "14.1.3 • Public • Published 17 days ago". A horizontal navigation bar contains several links: "Readme" (highlighted in yellow), "Explore" (with a "BETA" badge), "1 Dependency", and "267 Dependents". Below this bar, there's a green box with a checkmark and the text "Title Message". To the right of this box, there's an "Install" button and a partially visible input field containing "npm i ngx-toastr".

# Biblioteca ngx-toastr

■ Estando no site, acessar o github, conforme segue:

**npm**

Search packages

Search

Sign Up

**ngx-toastr** TS

14.2.1 • Public • Published 6 days ago

Readme

Explore BETA

1 Dependency

275 Dependents

105 Versions

Keywords

ng2 ngx ngx-toastr toastr-ng2 angular angular2 typescript alert toast  
toastr angulartoastr notifications

Install

```
> npm i ngx-toastr
```

Repository

[github.com/scttcper/ngx-toastr](https://github.com/scttcper/ngx-toastr)



# Biblioteca ngx-toastr

- Vamos copiar o comando para instalar, descendo um pouco com o mouse terá o comando abaixo:

```
npm install ngx-toastr --save
```

## Install

---

```
npm install ngx-toastr --save
```

# Biblioteca ngx-toastr

- Agora vamos instalar o @angular/animations para que nossa aplicação possa ter animações. Seria a linha abaixo do install.

```
npm install @angular/animations --save
```

## Install

```
npm install ngx-toastr --save
```

@angular/animations package is a required dependency for the default toast

```
npm install @angular/animations --save
```

# Biblioteca ngx-toastr

- Agora vamos fazer a importação do css do ngx-toastr
- Descendo na documentação, temos a linha que devemos colar no angular.json para realizar a importação.
- Vamos copiar a linha abaixo:

▶ `"node_modules/ngx-toastr/toastr.css"`

- If you are using angular-cli you can add it to your angular.json

```
"styles": [  
  "styles.scss",  
  "node_modules/ngx-toastr/toastr.css" // try adding '../' if you're using angular  
]
```

## Biblioteca ngx-toastr

- Vamos procurar o arquivo angular.json, por styles e antes do src/styles.css, vamos colar a linha, inserindo uma virgula no final.

```
"styles": [  
  "node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "node_modules/font-awesome/css/font-awesome.min.css",  
  "node_modules/ngx-toastr/toastr.css",  
  "src/styles.css"  
],
```

## Biblioteca ngx-toastr

- Seguindo a documentação, vamos para o Step 2, inserindo BrowserAnimationsModule e ToastrModule
- Vamos copiar as duas linhas abaixo:

```
@NgModule({  
  imports: [  
    CommonModule,  
    BrowserAnimationsModule, // required animations module  
    ToastrModule.forRoot(), // ToastrModule added
```

# Biblioteca ngx-toastr

- Vamos na aplicação e vamos abrir o app.module.ts, procurar por imports e vamos colar abaixo do HttpClientModule:

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule,  
  SharedModule,  
  HttpClientModule,  
  BrowserAnimationsModule, // required animations module  
  ToastrModule.forRoot(), // ToastrModule added  
],
```

## Biblioteca ngx-toastr

- Apresentou erro porque precisamos inserir o import no início do código. Para isso, vamos copiar os imports:

```
import { CommonModule } from '@angular/common';  
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';  
import { ToastrModule } from 'ngx-toastr';
```

# Biblioteca ngx-toastr

- Vamos inserir abaixo dos imports e os erros desaparecerão.

```
import {HttpClientModule} from '@angular/common/http';  
  
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';  
import { ToastrModule } from 'ngx-toastr';
```



## Biblioteca ngx-toastr

- Acabamos de fazer as mudanças estruturais, podemos ir no prompt e rodar o comando para executar a página.
  - `ng serve -o <ENTER>`

## Biblioteca ngx-toastr

- Vamos abrir o arquivo produtos.service.ts
- Vamos criar um método chamado `exibirMensagem()` com retorno de `void`

```
✓ buscarTodos(): Observable<IProduto[]> {  
  |   return this.http.get<IProduto[]>(this.URL);  
  | }  
  |  
✓ exibirMensagem():void {  
  |  
  | }  
  | }  
}
```

## Biblioteca ngx-toastr

- O que iremos precisar para exibir a mensagem?
- Vamos ter um titulo da mensagem, do tipo string (Titulo:string)
- Vamos ter a mensagem que também é do tipo string (Mensagem: string)
- Tipo do erro que também é do tipo string (Tipo: string)

```
exibirMensagem(titulo: string, mensagem: string, tipo: string):void {  
  
  
  
  
  
  
  
  
  
}
```

# Biblioteca ngx-toastr

- Agora já podemos chamar o toastr dentro do método `exibirMensagem()`, mas antes precisamos fazer a importação do toastr.
- Voltando na documentação, no USE, precisaremos criar a variável no construtor, conforme a documentação. Vamos copiar e colar:

## Use

```
import { ToastrService } from 'ngx-toastr';

@Component({...})
export class YourComponent {
  constructor(private toastr: ToastrService) {}

  showSuccess() {
    this.toastr.success('Hello world!', 'Toastr fun!');
  }
}
```

## Biblioteca ngx-toastr

- Vamos no produtos.service.ts e colar após o HttpClient inserir uma vírgula e colar.

```
constructor(private http: HttpClient, private toastr: ToastrService) {}
```

# Biblioteca ngx-toastr

■ <CTRL><espaço> após o ToastrService para importar a biblioteca:

```
9   export class ProductService {
10     private url: string = 'http://localhost:8080/products';
11     ✚
12     constructor(private http: HttpClient, private toastr: ToastrService) {}
13
14     get from "ngx-toastr" class ToastrService ✚ ToastrService node_modules/ngx-toastr/toastr/toastr.service
15     {} > {
16         {this.url};
17     }
18 }
19 }
```

## Biblioteca ngx-toastr

- Pronto, agora já podemos utilizar ele dentro do método criado.
- Dentro do `exibirMensagem()`, digitar:
- `toast` (selecionar o `toast.show()` para exibir a mensagem. Dentro do `show`, vamos passar a mensagem, o título e um objeto com a propriedade `closeButton` e `progressBar` e o tipo de mensagem que será exibida:

`show(mensagem, titulo, {closeButton: true, progressBar: true}, tipo)`

- `closeButton: true` -> exibe o botão de fechar
- `progressBar : true` -> Será uma barra de progresso

## Biblioteca ngx-toastr

- Ficará assim. O método de `exibirMensagem` está pronto.

```
exibirMensagem(titulo: string, mensagem: string, tipo: string):void {  
    this.toastr.show(mensagem, titulo, {closeButton:true, progressBar: true}, tipo);  
}
```



## Biblioteca ngx-toastr

- O próximo passo é criarmos uma mensagem genérica para exibir erros.
- Acima do método `exibirMensagem()`, vamos criar um método `exibirErro() : void { }`

```
exibirErro(): void {  
  
}  
exibirMensagem(titulo: string, mensagem: string, tipo: string):void {  
    this.toastr.show(mensagem, titulo, {closeButton:true, progressBar: true}, tipo);  
}
```

## Biblioteca ngx-toastr

- O `exibirErro()` irá receber um `e`, do tipo `any`. Daqui a pouco vcs irão entender porque aqui irá receber um `e` do tipo `any`.

```
exibirErro(e: any): void {  
  
}
```

# Biblioteca ngx-toastr

## ■ Dentro do método:

- ▶ Digitar `exibirMensagem` e selecionar o que está sendo exibido, e vamos passar os parâmetros: título, mensagem e tipo de erro:

```
exibirErro(e: any): void {  
  this.exibirMensagem('Erro!!!', 'Não foi possível realizar a operação', 'toast-error');  
}
```

# Biblioteca ngx-toastr

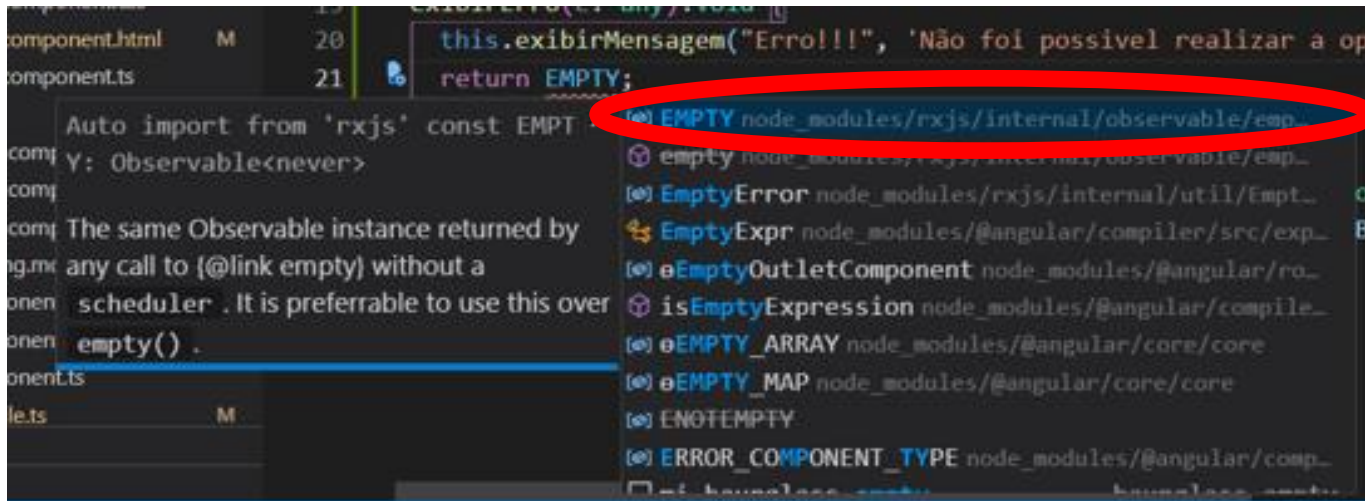
- Todos os dados foram inseridos de acordo com a documentação.
- Se vocês acessarem a documentação, encontrarão essas informações.

## iconClasses defaults

```
iconClasses = {  
  error: 'toast-error',  
  info: 'toast-info',  
  success: 'toast-success',  
  warning: 'toast-warning',  
};
```

# Biblioteca ngx-toastr

- Para que não dê erro, no `exibirErro`, precisaremos dar um `return EMPTY` (Ao digitar o `EMPTY`, é possível fazer a importação dele). Irá ser importado do lado do `Observable`.
- Por que precisamos fazer essa importação? Para que ele não dê erro no momento que chamar o `exibirErro()` e ficar com o `toast` carregando.



```
import { EMPTY, Observable } from 'rxjs';
import { ToastrService } from 'ngx-toastr';
```

```
exibirErro(e: any): void {
    this.exibirMensagem('Erro!!!', 'Não foi possível realizar a operação', 'toast-error');
    return EMPTY;
}
```

## Biblioteca ngx-toastr

- Estamos com erro no return, vamos corrigir. O tipo não é void, é Observable do tipo any. Porque será exibido de acordo com a nossa necessidade. Precisamos que fique observando até que o termino da execução termine e retorne um vazio para não dar erro na nossa aplicação.

```
exibirErro(e: any): Observable<any> {  
  this.exibirMensagem('Erro!!!', 'Não foi possível realizar a operação', 'toast-error');  
  return EMPTY;  
}
```

## Biblioteca ngx-toastr

- Agora já podemos fazer a nossa mensagem de erro aparecer quando não conseguir fazer o retorno aqui do nosso get:

```
buscarTodos(): Observable<IProduto[]> {  
  return this.http.get<IProduto[]>(this.URL);  
}
```

## Biblioteca ngx-toastr

- Para isso, vamos colocar aqui .pipe (.pipe é um método que temos dentro do Observable que irá permitir que façamos a concatenação de vários métodos.

```
buscarTodos(): Observable<IProduto[]> {  
    return this.http.get<IProduto[]>(this.URL).pipe(  
    );  
}
```



## Biblioteca ngx-toastr

- O primeiro método é o map e teremos que fazer a importação dele.

```
buscarTodos(): Observable<IProduto[]> {  
  return this.http.get<IProduto[]>(this.URL).pipe(  
    map()  
  );  
}
```

## Biblioteca ngx-toastr

- Para importar, vamos escrever manualmente. Vamos dar um <ENTER> após o EMPTY, Observable.
- Vamos digitar `import {map} from` e vamos chamar nossa biblioteca 'rxjs', mas nós vamos pegar um operators dela, vamos pegar uma das operações que a biblioteca nos fornece.

```
import {map} from 'rxjs/operators';  
import { ToastrService } from 'ngx-toastr';
```

## Biblioteca ngx-toastr

- O map vai poder mapear, conseguimos fazer uma manipulação dos dados de retorno que teremos aqui.
- Vamos colocar no método map() que o retorno que vou receber eu quero que retorne ele mesmo. Isso parece até estranho ter que fazer isso, mas estamos fazendo isso porque quando o Observable receber a resposta do nosso get ele terá que fazer alguma coisa. O map irá pegar o retorno e vai enviar como resposta do nosso método buscarTodos() para o nosso listar.produto.component.ts

```
buscarTodos(): Observable<IProduto[]>{  
  return this.http.get<IProduto[]>(this.URL).pipe(  
    map(retorno => retorno)  
  )  
}
```

## Biblioteca ngx-toastr

- Por que não fizemos mais nada aqui? Porque não vamos fazer nenhum tratamento. Caso você precisasse fazer algum tratamento das informações que vem do nosso banco de dados, da nossa API vocês poderiam fazer aqui.
- Para funcionar, vamos colocar uma vírgula após o map, antes do último parênteses ) e utilizar um outro operator aqui, que é o `catchError`.

```
buscarTodos(): Observable<IProduto[]>{  
  return this.http.get<IProduto[]>(this.URL).pipe(  
    map(retorno => retorno),  
    catchError()  
  )  
}
```

# Biblioteca ngx-toastr

- Ao começar a digitar, já irá aparecer para importar o `catchError`, ficando assim:

```
import {map, catchError} from 'rxjs/operators';  
import { ToastrService } from 'ngx-toastr';
```

## Biblioteca ngx-toastr

- Esse método será responsável por pegar os erros que aconteceu na nossa aplicação. Vamos utilizar após o map. Dentro do catchError, vou passar o meu erro e vou passar para o `exibirErro(erro)`. Colocar o `;` após o fechamento da chave `}`:

```
buscarTodos(): Observable<IProduto[]>{  
  return this.http.get<IProduto[]>(this.URL).pipe(  
    map(retorno => retorno),  
    catchError(erro => this.exibirErro(erro)))  
};
```

## Biblioteca ngx-toastr

- Por que colocamos no `exibirErro()` o tipo `any`? Porque não sei como será essa resposta de erro que virá da nossa biblioteca, por isso colocamos o tipo `any`, assim qualquer coisa que virá ele irá aceitar na nossa mensagem.

## Biblioteca ngx-toastr

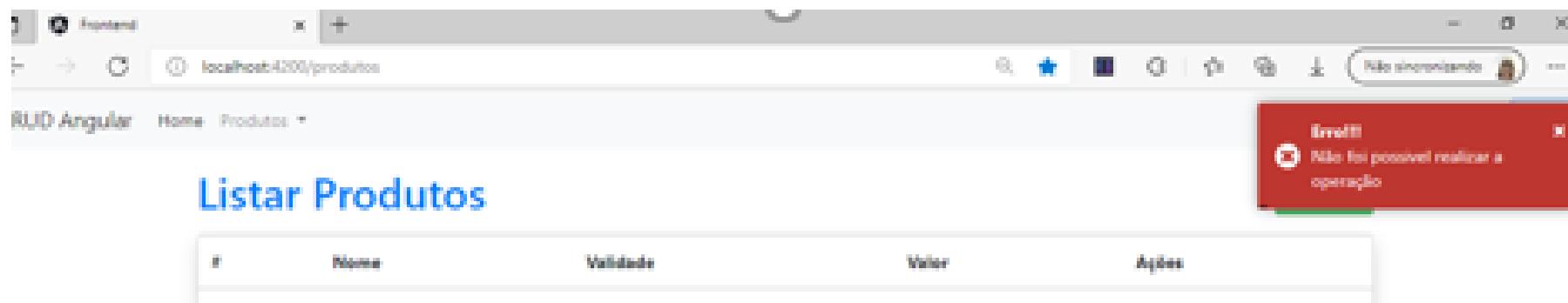
E agora, automaticamente, caso o get não consiga ter uma resposta do nosso servidor, da nossa api, o Observable ao receber uma mensagem de erro vai acionar automaticamente o nosso catchError().

```
buscarTodos(): Observable<IProduto[]> {  
  return this.http.get<IProduto[]>(this.URL).pipe(  
    map(retorno => retorno),  
    catchError(erro => this.exibirErro(erro))  
  );  
}  
  
exibirErro(e: any): Observable<any> {  
  this.exibirMensagem('Erro!!!', 'Não foi possível realizar a operação', 'toast-error');  
  return EMPTY;  
}
```



# Biblioteca ngx-toastr

- Agora podemos ir na nossa aplicação. Estamos com o servidor frontend rodando e o servidor backend parado. Ao listar produtos, irá exibir a mensagem:





# Biblioteca ngx-toastr

- Se voltarmos a subir o servidor backend, ficará normal.
  - ▶ `npm start <ENTER>`
- Clica no botão atualizar e a listagem será exibida.

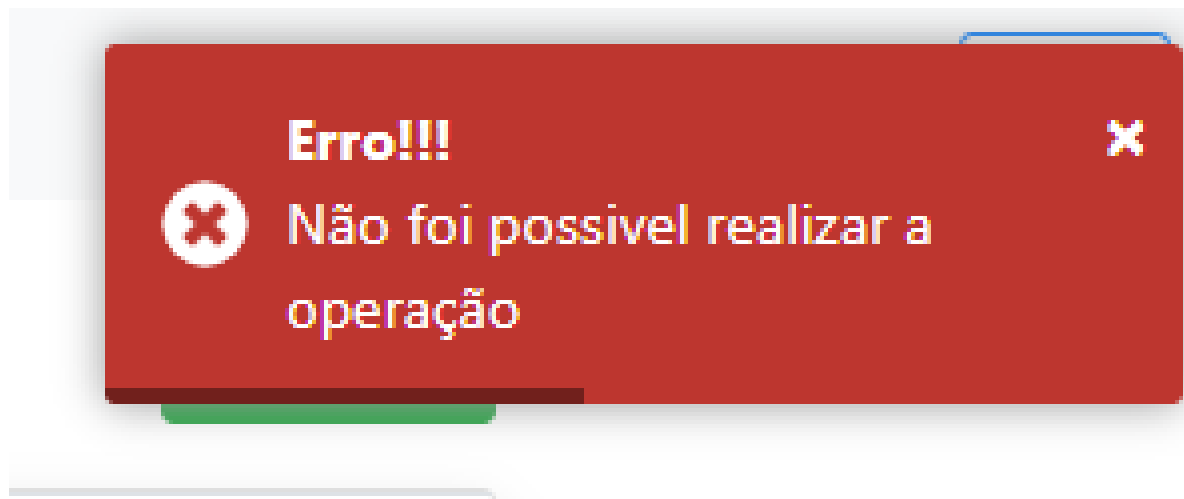
## Listar Produtos

Cadastrar

#	Nome	Validade	Valor	Ações
1	Curso de Angular	31/12/2021	R\$ 550,54	 

## Biblioteca ngx-toastr

- Pare o servidor novamente, clique no botão atualizar e a mensagem de erro será exibida novamente. Demora um tempinho porque ele tem o tempo de resposta.



# Biblioteca ngx-toastr

- Indo na ferramenta do desenvolvedor, podemos também ver o erro:

```
Angular Contexto de JavaScript: top nt mode. Call core.js:28072  
enableProdmode() to enable production mode.  
  
SyntaxError: Unexpected token C in JSON at position 0 app.js:59  
    at JSON.parse (<anonymous>)  
    at window.onload (app.js:59)  
  
[WDS] Live Reloading enabled. index.js:52  
  
✖ Failed to load resource: :3000/produtos:1  
net::ERR_CONNECTION_REFUSED  
>
```

