

**Documentação Técnica do Front-end**  
**Gerenciador de Produtos**  
**Projeto teste**

[Sobre o Projeto](#)

[Objetivo da Documentação](#)

[Stacks do Projeto](#)

[Angular](#)

[Bootstrap](#)

[Bootstrap Icons](#)

[ngx-toastr](#)

[ESLint](#)

[Ngx-translate](#)

[Ngx-translate/http-loader](#)

[Ngx-mask](#)

[Firebase](#)

[Estrutura e Organização Inicial](#)

[Estrutura de Pastas](#)

[App](#)

[Components](#)

[Interfaces](#)

[Services](#)

[Shared](#)

[Views](#)

[Modals](#)

[Assets](#)

[Images](#)

[Icons](#)

[I18n](#)

[Uso de Temas e Estilos](#)

[Custom-theme](#)

[Style.scss](#)

[Assets e Compressões](#)

[Componentes, Layout e Responsividade](#)

[Bootstrap](#)

[Grid](#)

[Ajustes de Responsividade](#)

[Quebra de Linha no HTML](#)

[Código e Lógica](#)

[Interfaces \(Contrato de Dados\)](#)

[Tipagem de Dados](#)

[Endpoints](#)

[Repetição de Código](#)

[Rotas no Angular 20](#)

[Environments, Run e Build](#)

[Arquivo environments](#)

[Rodando o Projeto](#)

[Build para ambiente](#)

[ESLint](#)

[Ngx-Toastr](#)

[Configuração do Toastr](#)

[Como Usar o Serviço de Toast](#)

Serviço de Modals

Componente Base das Modals

Inputs de Dados

Comportamento

Modal Service

Método principal

Fluxo

Padrão de Uso

Exemplo visual

i18n

Componentes reutilizáveis

Common Button

Common Spinner

Common table

Configuração da Tabela

Uso da tabela

## Sobre o Projeto

Esse projeto é um web app em Angular focado em gerenciamento de produtos, implementando um CRUD completo (listar, visualizar/detalhar, criar e editar) consumindo uma Fake API de produtos (com campos como id, title, price, description, category, image). Ele tem uma UI moderna e simples (com telas como listagem e formulário de detalhes do produto), suporte a i18n via ngx-translate (ex.: pt-BR/en-US), componentes e serviços usando injeção via inject(), além de organização típica de SPA com rotas (ex.: rotas de auth vs área principal) e build/deploy pensado para ambientes (dev/qa/stg/prod) e publicação via Firebase.

## Objetivo da Documentação

Esta documentação tem como objetivo detalhar a implementação da camada front-end, com foco em:

- Padrões de código adotados;
- Estrutura do tema e estilos globais;
- Organização e roteamento de páginas;
- Serviços reutilizáveis (modais, toasts, etc.);
- Boas práticas e decisões arquiteturais.

Ela serve como referência para desenvolvedores atuais e futuros, garantindo consistência na manutenção e evolução da interface do sistema.

## Stacks do Projeto

O front-end do projeto foi desenvolvido utilizando angular 20 como principal framework, aliado a um conjunto de bibliotecas e ferramentas modernas que garantem performance, consistência visual e qualidade de código. A seguir, são destacadas as principais tecnologias empregadas.

### Angular

Framework principal para construção da interface, adotando os recursos mais recentes como Standalone Components e melhorias no roteamento e modularização.

## **Bootstrap**

Utilizado como base para estruturação responsiva e grid layout. A estilização visual, no entanto, é customizada com SCSS para manter a identidade visual do projeto.

## **Bootstrap Icons**

O projeto usa Bootstrap Icons como biblioteca de ícones para a interface. Ela fornece um conjunto grande de ícones em SVG/fonte, fáceis de aplicar com classes (ou importação).

## **ngx-toastr**

Biblioteca utilizada para exibição de mensagens do tipo toast, garantindo feedbacks visuais claros e não intrusivos ao usuário.

## **ESLint**

Utilizados para padronização de código, com regras específicas para angular e integração com TypeScript. Garante legibilidade, manutenção e evita erros comuns.

## **Ngx-translate**

Biblioteca de internacionalização de textos do sistema.

## **Ngx-translate/http-loader**

É um utilitário para angular usado junto ao ngx-translate. Ele carrega arquivos de tradução (JSON) via HTTP, facilitando a internacionalização. Assim, permite gerenciar múltiplos idiomas de forma dinâmica na aplicação.

## **Ngx-mask**

O ngx-mask é uma biblioteca para angular que aplica máscaras a campos de entrada. Ela permite formatar dados como CPF, CNPJ, telefone, CEP e datas de forma automática. Assim, garante padronização e melhora a usabilidade em formulários.

## **Firebase**

O projeto usa o Firebase Hosting para publicar a aplicação na internet de forma simples e segura. O código gerado (build) é enviado para o Firebase, que distribui o

conteúdo via CDN, garantindo carregamento rápido, e oferece HTTPS automático e configuração fácil de rotas/redirecionamentos (útil para apps SPA).

Essas tecnologias foram escolhidas por sua maturidade, performance e ampla adoção na comunidade, garantindo escalabilidade e facilidade de manutenção no longo prazo.

## Estrutura e Organização Inicial

### Estrutura de Pastas

Foi pensado na estrutura do projeto antes do início da codificação, a estrutura apresentada irá separar cada função em pastas específicas. Foi adotado dessa forma para melhor organização e manutenções futuras. (Pode sofrer alterações).

<b>app/</b> └── <b>components/</b> └── <b>interfaces/</b> └── <b>services/</b> └── <b>shared/</b> └── <b>views/</b> └── <b>modals/</b> └── <b>endpoints/</b>	<b>assets/</b> └── <b>img/</b> └── <b>Icons/</b> └── <b>i18n/</b>
---	--

### App

Pasta pai do sistema, nela conterá todas as outras pastas referentes à features do sistema, como páginas, componentes e outros.

### Components

Pasta onde se encontra todos os componentes reutilizáveis do sistema, como cards, gráficos e outros. Os componentes são feitos pensando em reuso no sistema inteiro se necessário. É separado em uma pasta específica onde cada componente tem seu html, scss e ts. Contendo lógica e seus próprios estilos. Eles também têm seus próprios módulos, ou seja, só será chamado e carregado o que for necessário para o funcionamento deles.

## Interfaces

Pasta onde se encontram todas as interfaces do sistema, geralmente organizadas por pastas. Essas interfaces são a identificação dos tipos de dados que vai ser consumido dentro do sistema, é utilizada no sistema todos, tipando variáveis, chamadas http e outros.

## Services

Pasta que contém os serviços de consumo de dados, usado para armazenar chamadas http no sistema bem como tratamento específico de lógicas dentro do sistema, dependendo da regra pode ser utilizado métodos dentro do sistema, mediante a injeção via construtor.

## Shared

Pasta onde contém arquivos compartilhados, diferente da pasta componentes, é geralmente utilizada para compartilhar arquivos únicos, não componentes com módulos. Um bom exemplo é scss de efeitos.

## Views

Pasta que contém todas as páginas de fato do sistema, ou seja, os componentes que possuem rotas e que serão renderizados para o usuário.

## Modals

Pasta que contém todas as modais do sistema, bem como o componente modal container que é responsável pela atribuição dinâmica de um componente personalizado para a modal.

## Assets

Pasta que guarda conteúdos estáticos do sistema, como imagens, documentos e outros.

## Images

pasta para guardar imagens do sistema, ao guardar uma imagem nessa pasta, deixo o conselho de passar a mesma em algum sistema de compressão antes, para ajudar no desempenho e carregamento delas.

## Icons

Pasta que guarda ícones personalizados do sistema, foi usado o bootstrap icons, porém, caso possua um ícone personalizado, será aqui que se encontrará o mesmo.

## I18n

Pasta que contém todos os arquivos de tradução do sistema, ou seja, caso tenha um novo idioma no sistema, é aqui que vão estar todos os textos traduzidos.

## Uso de Temas e Estilos

### Custom-theme

No projeto existe o arquivo de tema global, nele deve conter apenas as variáveis de tema, ou seja, nenhum outro scss que não tenha relação com o tema do projeto.

As variáveis consistem em um nome e a cor, elas são usadas nos arquivos de estilos por todo o sistema, para usar, basta importar o custom-theme.

### Style.scss

Arquivo de estilos globais, muito importante, nele contém os estilos principais em todo o sistema. Deve ser adicionado estilo nesse arquivo somente se necessário, pois ele tem efeito no sistema todo, então deve ser usado com cautela.

### Assets e Compressões

No sistema tem a pasta de assets, nela contém todos os arquivos do projeto, é importante salientar a importância da otimização dos arquivos nele contido. Sempre que possível, otimizar os arquivos que irão para essa pasta, por mais leves ou pesados que eles sejam originalmente, pois quanto mais otimizado melhor. Um exemplo é o uso de imagens, antes de adicioná-las no projeto, usar um compressor de imagens, para reduzir seu tamanho. Também é interessante, se possível, diminuir as dimensões para uma próxima a ser usada no sistema, como por exemplo, se uma imagem vai ter no máximo 150px no sistema, não é recomendada e nem interessante em adicionar ela em 4k.

## Componentes, Layout e Responsividade

### Bootstrap

No sistema é adotado o uso do bootstrap, para facilitar a estrutura HTML bem como facilitar o uso de alguns componentes que ele fornece.

## Grid

Usando o bootstrap, é apresentado o sistema de grid, que está fortemente seguido dentro do projeto, sistema esse que fornece uma estrutura html mais robusta que ajuda a criar linhas e colunas responsivas.

Padrão de classes	Descrição
.container	container com espaçamentos laterais (width: 1200px)
.container-fluid	container sem espaçamentos laterais (width: 100%)
.row	Div única, deve ser usada como uma div pai que terá de 1 a 12 divs filhas em colunas
.col	Div única, deve ser usada como uma div pai que terá de 1 a 12 divs filhas

O padrão mostrado acima na tabela apresenta a estrutura das páginas HTML do projeto. Novas páginas devem seguir esse padrão. Exemplo:

```
<div class="container-fluid">
  <div class="row">
    <div class="col">
      <h1>Page title</h1>
    </div>
  </div>

  <div class="row">
    <div class="col">
      <p>First container</p>
    </div>
    <div class="col">
      <p>Second container</p>
    </div>
  </div>
</div>
```

## Ajustes de Responsividade

No projeto, apesar do uso do bootstrap facilitar a responsividade, caso necessário, usar média query para efetuar ajustes de responsividade. Importante sempre que finalizar uma página, componentes e afins. Analisar o mobile e demais telas se necessário.

```
@media only screen and (max-width: 574px) {  
  .btn-actions-box {  
    text-align: center;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
  };  
}
```

## Quebra de Linha no HTML

Sempre usar quebra de linha para elementos que estiverem ficando muito grandes, ou seja, vamos supor que você tenha uma lista, não é muito interessante deixar todos os elementos em uma linha somente, então entra a quebra de linha e indentação, esse padrão deve ser seguido em todos os elementos html

```
<div class="row">  
  <div class="col-sm-12 text-end mt-3">  
    <div class="form-group btn-actions-box">  
      <app-common-button class="action-btn" [buttonText]="'Cancelar'" [type]="" [buttonStyle]="" (clicked)="onCancel()"></app-common-button>  
      <app-common-button  
        class="action-btn"  
        [buttonText]={(productId ? 'BUTTONS.EDIT_PRODUCT' : 'BUTTONS.NEW_PRODUCT') | translate}  
        [type]=""[submit]"  
        [buttonStyle]=""primary""  
        [disableButton]=""isSubmitting"  
      >  
      </app-common-button>  
    </div>  
  </div>  
</div>
```

Note no exemplo acima, o elemento em vermelho mostra um elemento muito extenso e de difícil entendimento, já no verde, com o uso correto da quebra de linha, fica muito mais claro e de melhor leitura. Esse padrão deve ser adotado em todo o sistema.

## Código e Lógica

### Interfaces (Contrato de Dados)

O sistema adota o uso forte de interfaces para a tipagem de dados que deve ser respeitado, toda a variável, função, chamadas http e afins. Devem ter seu tipo de dado declarado. As interfaces estão concentradas na pasta de interface, que podem estender outras e assim por diante.

### Tipagem de Dados

Forte tipagem de dados no sistema, todo e qualquer item que tenha fluxo de dados, deve ser passado o seu tipo, seja ele variáveis (mesmo já sendo iniciadas), funções (caso não retorne nada, ou seja, função somente de processamento, atribuir void),

chamadas http (importante atribuir o tipo de todas as requisições, o sistema deve saber o que é consumido e enviado) e outros.

## Endpoints

Será adotado o uso de arquivos endpoints.ts, cuja finalidade é centralizar os caminhos dos endpoints em um único local. Isso permite que esses caminhos sejam reutilizados em todo o sistema, promovendo uma melhor organização e facilitando a manutenção.

Por exemplo, considere um endpoint como /api/v2/user, utilizado em 50 métodos distintos no sistema. Caso futuramente a API seja atualizada para a versão v3, seria necessário alterar manualmente os 50 métodos. Com os arquivos centralizados de endpoints, essa mudança pode ser feita de forma simples e rápida em um único local, garantindo maior consistência e clareza no código.

## Repetição de Código

Evitar ao máximo códigos duplicados, sempre verificar os arquivos, caso tenha códigos duplicados, pensar em lógica de reutilização de arquivos ou métodos, se for arquivos estáticos como SCSS e afins, pode se criar um arquivo novo com a lógica a ser compartilhada dentro da pasta Shared.

Caso seja funções com códigos complexos e lógicas específicas que serão utilizadas no sistema, talvez adicionar no utils.service.ts e deixar disponível para o consumo através de injeção de dependência nos componentes necessários.

Caso tenha transformações de estilo através de lógica, pode adotar o uso de pipes, que receberam o item que terá a lógica e retornará o que for necessário. Conclusão, evitar sempre a repetição de código, sempre explorar alternativas com base na função desejada.

## Rotas no Angular 20

No Angular 20 é usado standalone components nativamente no projeto, com isso, o gerenciamento de rotas muda um pouco, pois agora tem arquivo separado para isso e dependendo de onde ele estiver a lógica muda.

Para arquivos pai, geralmente views/pages do sistema, a lógica fica centralizada dentro do arquivo app.routes.ts Foi usado lazy load em todas as rotas criadas, bem como as rotas filhas também, as rotas também ganharam títulos para cada uma:

```

export const routes: Routes = [
  {
    path: '',
    redirectTo: 'produtos',
    pathMatch: 'full',
  },
  // -----
  {
    path: 'produtos',
    title: 'Produtos',
    loadChildren: () => import('./views/products/products.routes').then(r => r.productsRoutes),
  },
  // -----
];

```

Para rotas filhas, cada módulo terá seu próprio arquivo de rotas, essa organização será mantida para todas as futuras rotas do sistema:

```

3  export const routes: Routes = [
  // ...
11  {
12    path: 'produtos',
13    title: 'Produtos',
14    loadChildren: () => import('./views/products/products.routes').then(r => r.productsRoutes),
15  },
16  // ...
18];

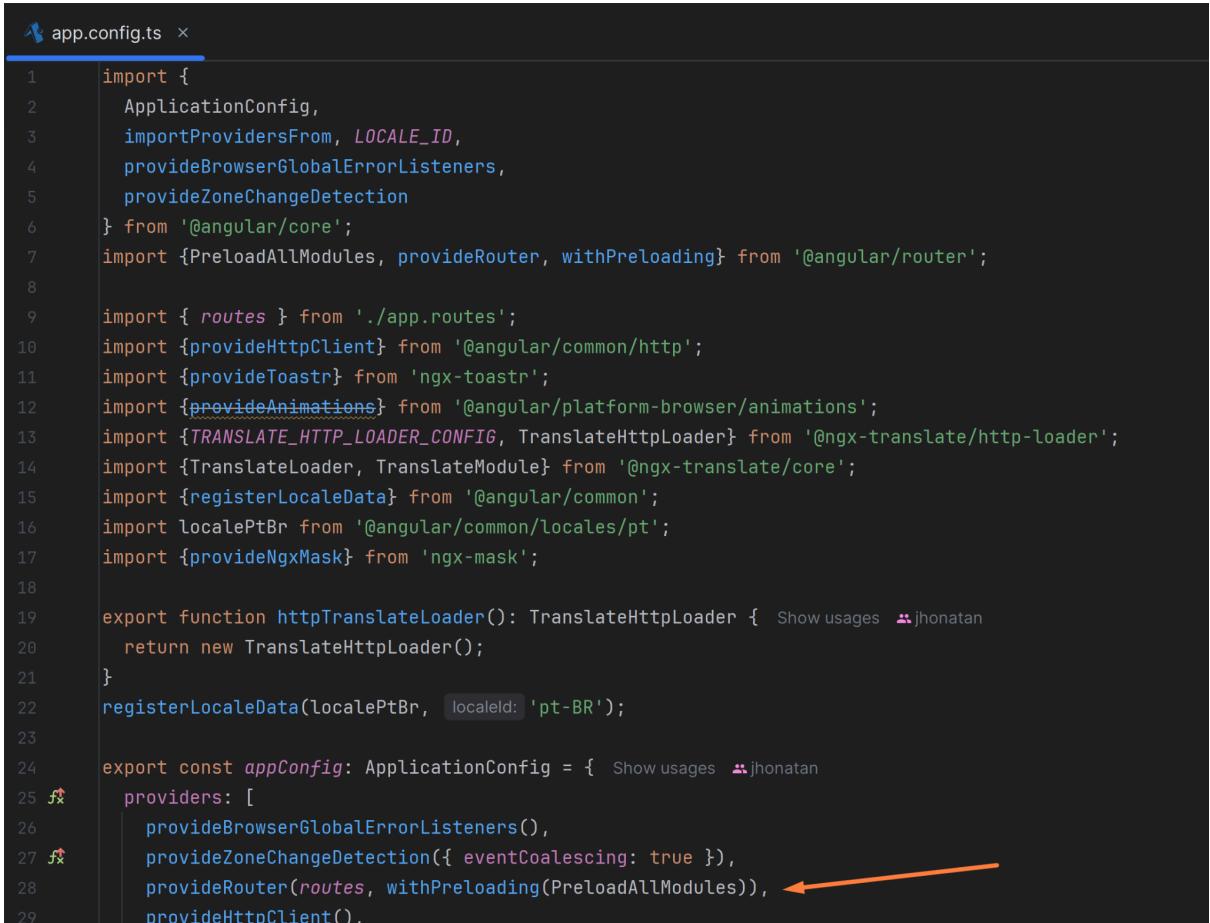
```

```

products.routes.ts
1  import {Routes} from '@angular/router';
2
3  export const productsRoutes: Routes = [
  // ...
4  {
5    path: '',
6    title: 'Produtos',
7    loadComponent: () => import('./products.page').then(c => c.ProductsPage),
8    pathMatch: 'full',
9  },
10  {
11    path: 'novo',
12    title: 'Produtos - Cadastro',
13    loadComponent: () => import('./product-detail/product-detail.page').then(c => c.ProductDetailPage),
14  },
15  {
16    path: ':id',
17    title: 'Produtos - Detalhes',
18    loadComponent: () => import('./product-detail/product-detail.page').then(c => c.ProductDetailPage),
19  },
20];

```

Foi adicionado a estratégia de pré-carregamento de todos os módulos, isso ajuda a ter um desempenho maior na navegação em todo o sistema:



```
app.config.ts ×
1 import {
2   ApplicationConfig,
3   importProvidersFrom, LOCALE_ID,
4   provideBrowserGlobalErrorListeners,
5   provideZoneChangeDetection
6 } from '@angular/core';
7 import {PreloadAllModules, provideRouter, withPreloading} from '@angular/router';
8
9 import { routes } from './app.routes';
10 import {provideHttpClient} from '@angular/common/http';
11 import {provideToastr} from 'ngx-toastr';
12 import {provideAnimations} from '@angular/platform-browser/animations';
13 import {TRANSLATE_HTTP_LOADER_CONFIG, TranslateHttpLoader} from '@ngx-translate/http-loader';
14 import {TranslateLoader, TranslateModule} from '@ngx-translate/core';
15 import {registerLocaleData} from '@angular/common';
16 import localePtBr from '@angular/common/locales/pt';
17 import {provideNgxMask} from 'ngx-mask';
18
19 export function httpTranslateLoader(): TranslateHttpLoader { Show usages ↗.jhonatan
20   return new TranslateHttpLoader();
21 }
22 registerLocaleData(localePtBr, { localeId: 'pt-BR' });
23
24 export const appConfig: ApplicationConfig = { Show usages ↗.jhonatan
25   providers: [
26     provideBrowserGlobalErrorListeners(),
27     provideZoneChangeDetection({ eventCoalescing: true }),
28     provideRouter(routes, withPreloading(PreloadAllModules)), ←
29     provideHttpClient(),
30   ],
31 }
```

## Environments, Run e Build

### Arquivo environments

O arquivo **environment.ts** contém as variáveis de ambiente que serão usadas pela aplicação Angular. Cada ambiente tem seu próprio arquivo de environments, local, dev, qa, staging e prod:



```
ts environment.ts ×
1 import {EnvironmentInterface} from './environments-interface/environment.interface';
2
3 export const environment: EnvironmentInterface = {
4   production: false,
5   apiUrl: 'https://fakestoreapi.com'
6 }
```

No arquivo angular.json, cada ambiente tem sua configuração, e na hora do build para os ambientes, ele faz a substituição do arquivo environment para o ambiente do build correspondente:

```
angular.json ×
5 "projects": {
6   "starian-test": {
16     "architect": {
17       "build": {
18         "configurations": {
19           "production": {
20             "optimization": true,
21             "aot": true,
22             "extractLicenses": true,
23             "outputHashing": "all",
24             "sourceMap": {
25               "scripts": false,
26               "styles": false,
27               "vendor": false
28             },
29             "budgets": [
30               {
31                 "type": "initial",
32                 "maximumWarning": "2mb",
33                 "maximumError": "5mb"
34               },
35               {
36                 "type": "anyComponentStyle",
37                 "maximumWarning": "8KB",
38                 "maximumError": "16KB"
39               }
40             ],
41             "fileReplacements": [
42               {
43                 "replace": "src/environments/environment.ts",
44                 "with": "src/environments/environment.prod.ts"
45               }
46             ]
47           }
48         }
49       }
50     }
51   }
52 }
```

## Rodando o Projeto

Similar a environments, há um comando para rodar cada ambiente puxando as configurações de cada ambiente, exemplo, DEV é focada em build e refresh, ja PROD é focada em performance absoluta, os comandos para rodar cada um foram definidos no arquivo package.json:

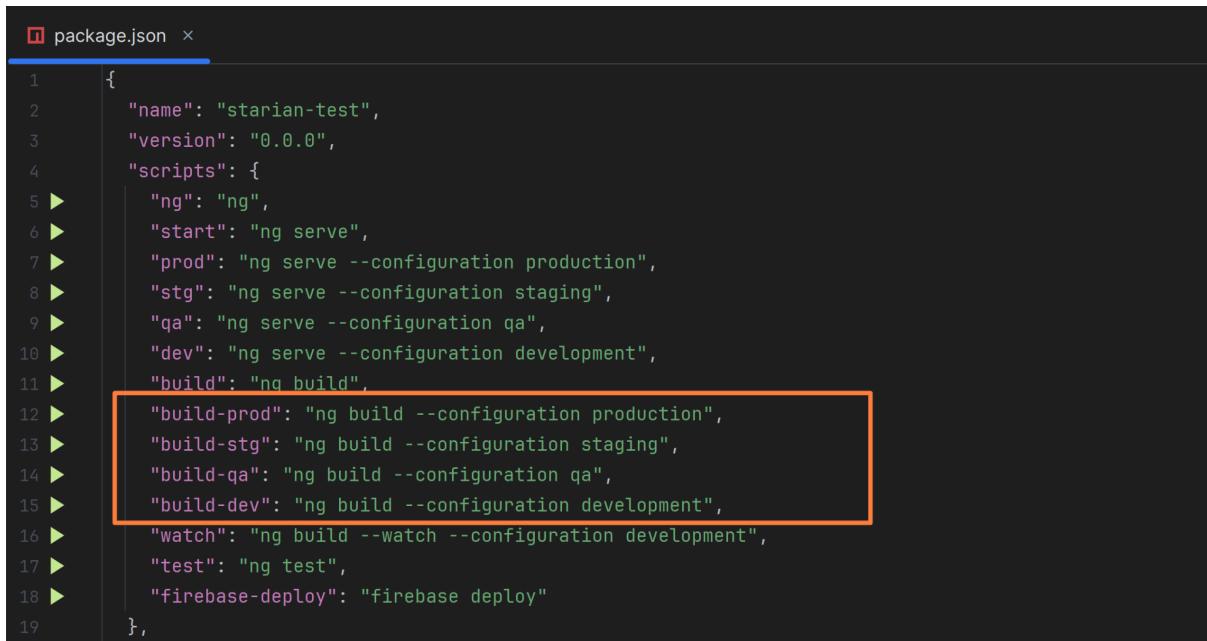


```
1  {
2    "name": "starian-test",
3    "version": "0.0.0",
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "prod": "ng serve --configuration production",
8      "stg": "ng serve --configuration staging",
9      "qa": "ng serve --configuration qa",
10     "dev": "ng serve --configuration development",
11   }
12 }
```

Para executar algum comando, basta colocar no terminal o comando **npm run dev**, ou o ambiente que deseja emular e rodar.

### Build para ambiente

Similar a environments e os comandos run, tem um comando para cada build de ambiente:

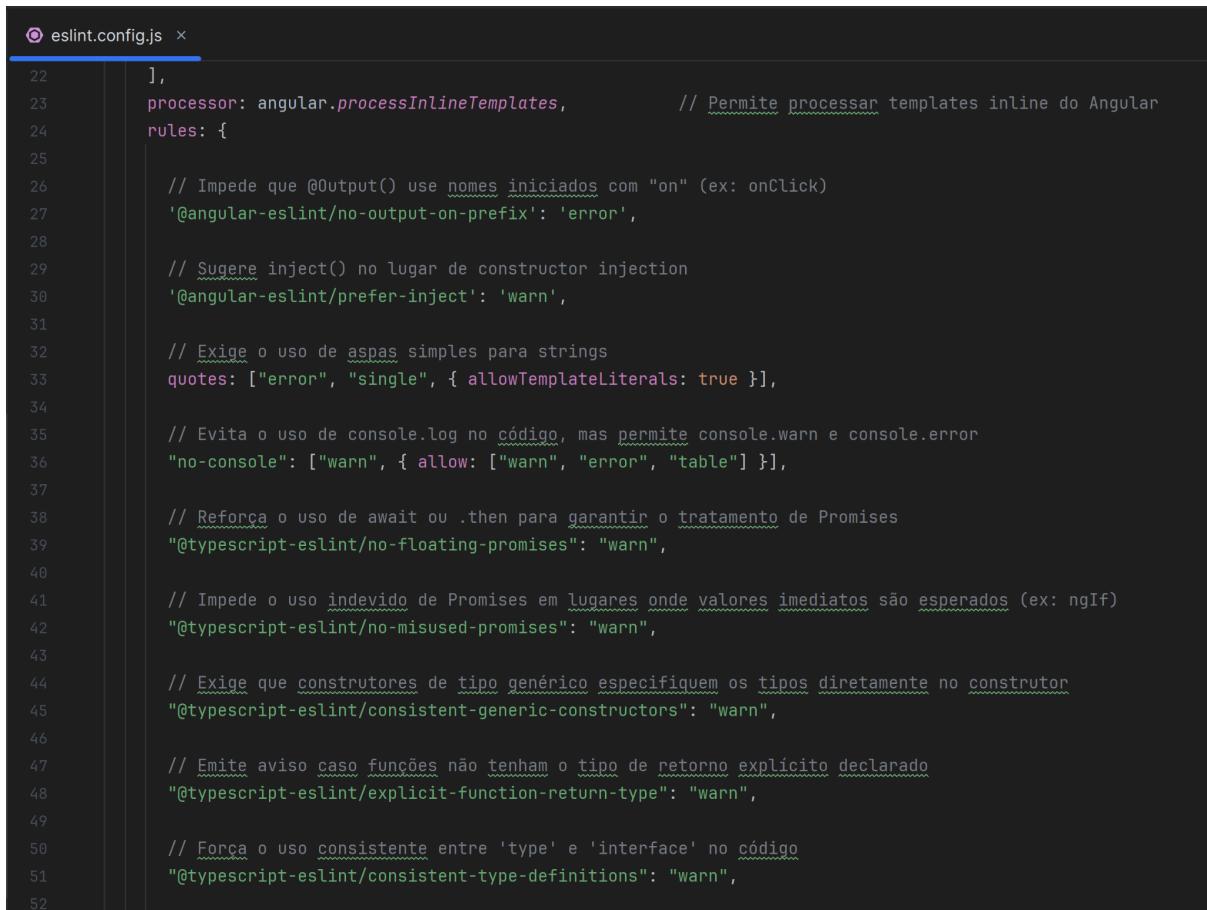


```
1  {
2    "name": "starian-test",
3    "version": "0.0.0",
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "prod": "ng serve --configuration production",
8      "stg": "ng serve --configuration staging",
9      "qa": "ng serve --configuration qa",
10     "dev": "ng serve --configuration development",
11     "build": "ng build",
12     "build-prod": "ng build --configuration production",
13     "build-stg": "ng build --configuration staging",
14     "build-qa": "ng build --configuration qa",
15     "build-dev": "ng build --configuration development",
16     "watch": "ng build --watch --configuration development",
17     "test": "ng test",
18     "firebase-deploy": "firebase deploy"
19   }
20 }
```

Por exemplo, se eu quiser rodar o build para PROD, eu usaria o comando **npm run build-prod**, e seria feito o empacotamento para o ambiente, no projeto atual é hospedado no firebase, ao rodar o comando de build, é criado uma pasta dist, depois é só rodar o comando de deploy, que no projeto atual é o **npm run firebase-deploy**

## ESLint

Este projeto conta com regras de ESLint para o projeto frontend. Todas as regras estão no arquivo `eslint.config.js`, todas as regras nesse arquivo tem um comentário em cima explicando para que serve:



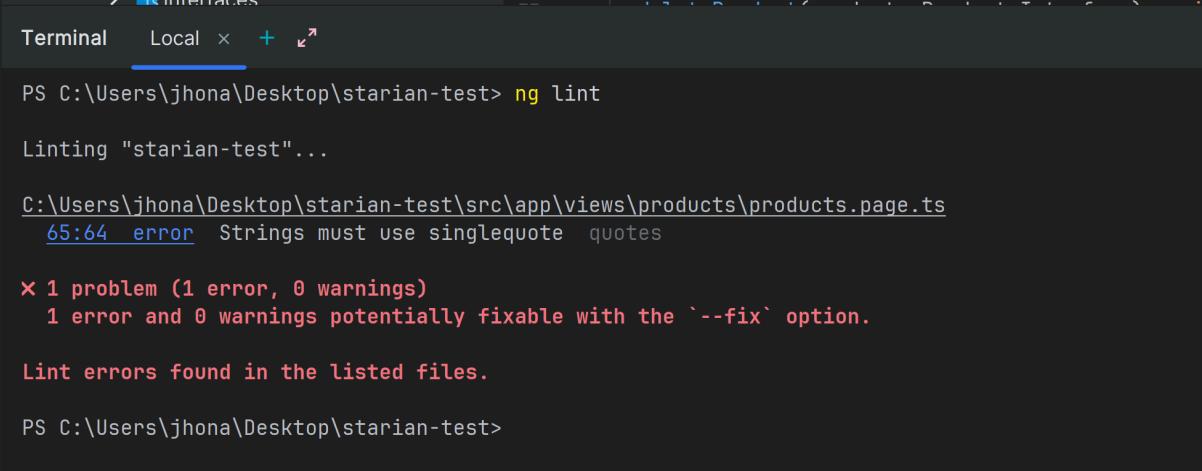
```
eslint.config.js ×
22  ],
23  processor: angular.processInlineTemplates,           // Permite processar templates inline do Angular
24  rules: {
25
26    // Impede que @Output() use nomes iniciados com "on" (ex: onClick)
27    '@angular-eslint/no-output-on-prefix': 'error',
28
29    // Sugere inject() no lugar de constructor injection
30    '@angular-eslint/prefer-inject': 'warn',
31
32    // Exige o uso de aspas simples para strings
33    quotes: ["error", "single", { allowTemplateLiterals: true }],
34
35    // Evita o uso de console.log no código, mas permite console.warn e console.error
36    "no-console": ["warn", { allow: ["warn", "error", "table"] }],
37
38    // Reforça o uso de await ou .then para garantir o tratamento de Promises
39    '@typescript-eslint/no-floating-promises': "warn",
40
41    // Impede o uso indevido de Promises em lugares onde valores imediatos são esperados (ex: ngIf)
42    '@typescript-eslint/no-misused-promises': "warn",
43
44    // Exige que construtores de tipo genérico especifiquem os tipos diretamente no construtor
45    '@typescript-eslint/consistent-generic-constructors': "warn",
46
47    // Emite aviso caso funções não tenham o tipo de retorno explícito declarado
48    '@typescript-eslint/explicit-function-return-type': "warn",
49
50    // Força o uso consistente entre 'type' e 'interface' no código
51    '@typescript-eslint/consistent-type-definitions': "warn",
52 }
```

Para rodar o lint e checar se está dentro dos padrões, é só habilitar na IDE o lint para inspeção, também é possível rodar via comando, no terminal rode o comando `ng lint`, se tiver tudo certo ele vai voltar uma mensagem que passou corretamente no lint:



```
Terminal Local × + ↵
PS C:\Users\jhona\Desktop\starian-test> ng lint
Linting "starian-test"...
All files pass linting. ↗
PS C:\Users\jhona\Desktop\starian-test>
```

Caso de algum erro de lint, no terminal vai informar qual o erro e um link direto para o arquivo fora dos padrões, exemplo:



```
Terminal Local + ↗
PS C:\Users\jhona\Desktop\starian-test> ng lint
Linting "starian-test"...
C:\Users\jhona\Desktop\starian-test\src\app\views\products\products.page.ts
65:64 error Strings must use singlequote quotes
✖ 1 problem (1 error, 0 warnings)
1 error and 0 warnings potentially fixable with the `--fix` option.

Lint errors found in the listed files.
PS C:\Users\jhona\Desktop\starian-test>
```

No exemplo coloquei propositalmente um texto em aspas duplas, ao invés de aspas simples. Para ajustar, é só clicar no link e ajustar para o padrão estipulado no lint.

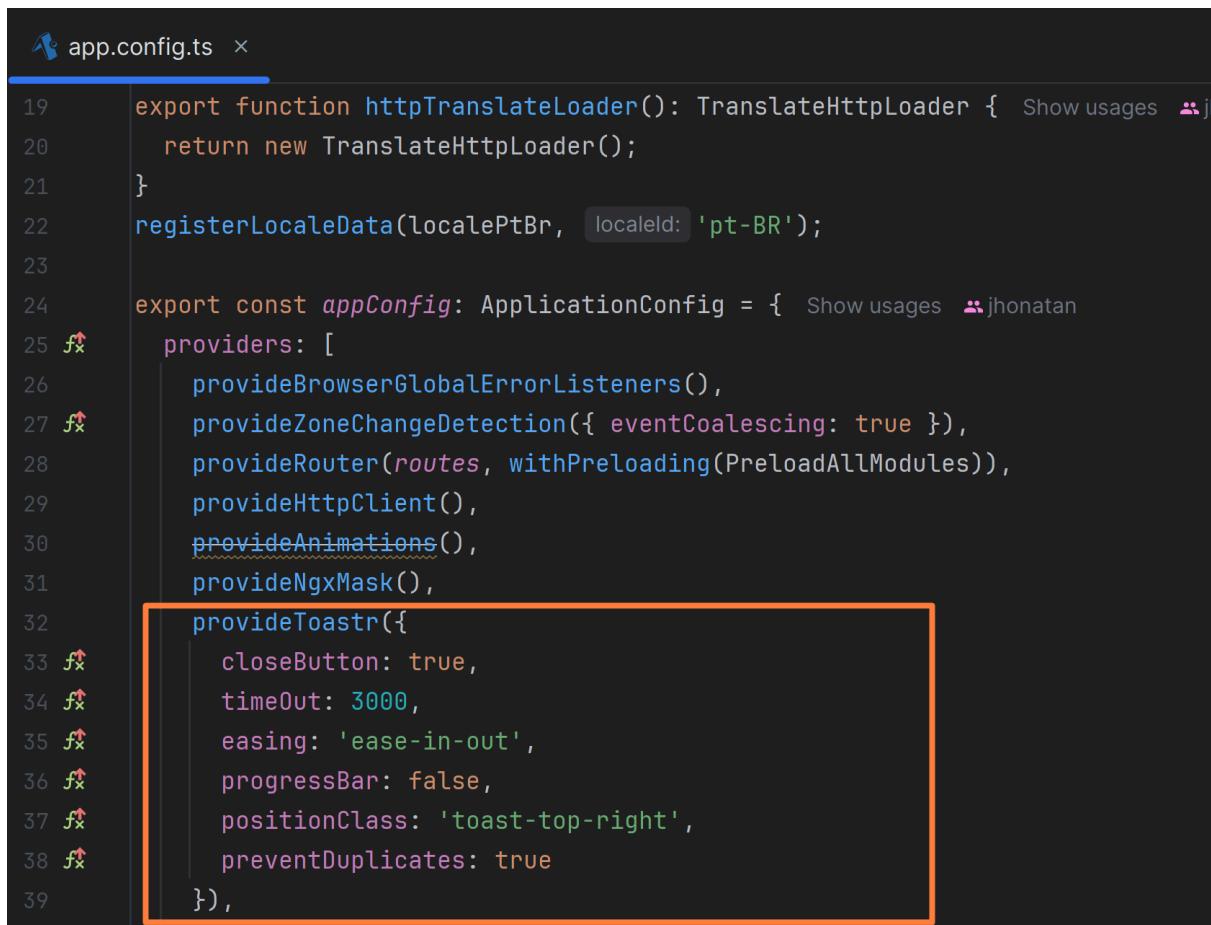
## Ngx-Toastr

Foi adicionado o ngx-toastr para agir como serviço de toast no sistema, essa biblioteca é bem comum, leve e está atualizada com o angular 20, dando suporte a standalones componentes.

```
package.json ×
20      "prettier": {
23          "overrides": [
24              {
28                  |
29                  |
30              ]
31          },
32          "private": true,
33          "dependencies": {
34              "@angular/animations": "^20.3.15", ←
35              "@angular/common": "^20.3.0", |
36              "@angular/compiler": "^20.3.0",
37              "@angular/core": "^20.3.0",
38              "@angular/forms": "^20.3.0",
39              "@angular/platform-browser": "^20.3.0",
40              "@angular/router": "^20.3.0",
41              "@ngx-translate/core": "^17.0.0",
42              "@ngx-translate/http-loader": "^17.0.0",
43              "bootstrap": "^5.3.8",
44              "bootstrap-icons": "^1.13.1",
45              "firebase": "^12.6.0",
46              "ngx-mask": "^20.0.3",
47              "ngx-toastr": "^19.1.0", ←
48              "rxjs": "~7.8.0",
49              "tslib": "^2.3.0",
50              "zone.js": "~0.15.0"
51          },

```

## Configuração do Toastr



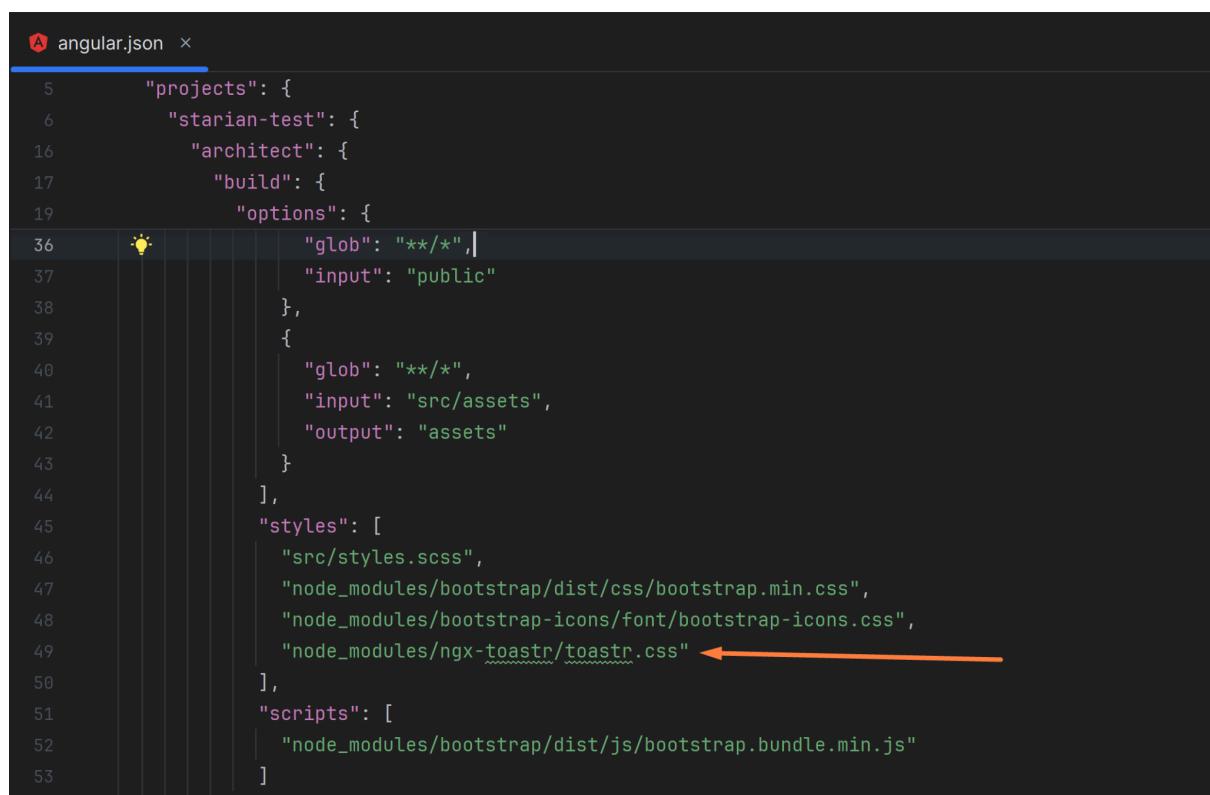
```
app.config.ts ×

19  export function httpTranslateLoader(): TranslateHttpLoader { Show usages ↗ j
20      return new TranslateHttpLoader();
21  }
22  registerLocaleData(localePtBr, localeId: 'pt-BR');
23
24  export const appConfig: ApplicationConfig = { Show usages ↗ jhonatan
25  providers: [
26      provideBrowserGlobalErrorListeners(),
27      provideZoneChangeDetection({ eventCoalescing: true }),
28      provideRouter(routes, withPreloading(PreloadAllModules)),
29      provideHttpClient(),
30      provideAnimations(),
31      provideNgxMask(),
32      provideToastr({
33          closeButton: true,
34          timeOut: 3000,
35          easing: 'ease-in-out',
36          progressBar: false,
37          positionClass: 'toast-top-right',
38          preventDuplicates: true
39      }),

```

- **closeButton: true:** Exibe um botão de fechar no toast.
- **timeOut: 3000:** Tempo de exibição do toast em milissegundos (3 segundos neste caso).
- **easing: 'ease-in-out':** Efeito de transição do toast ao aparecer/desaparecer.
- **progressBar true:** Exibe uma barra de progresso para indicar a duração do toast.
- **positionClass 'toast-top-right':** Define a posição do toast na tela. Pode ser: toast-top-right, toast-bottom-right, toast-top-left, etc.
- **preventDuplicates: true:** Impede a exibição de toasts duplicados com a mesma mensagem.

Foi adicionado o path dos estilos da toast no arquivo angular.json:

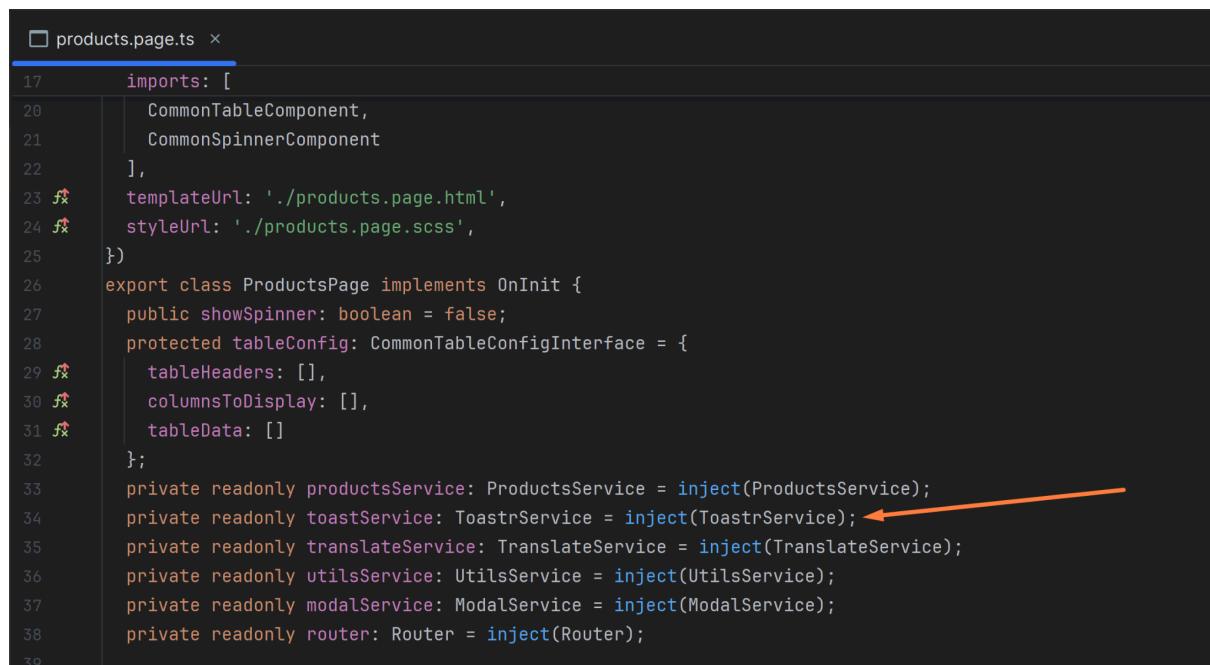


```
angular.json
5     "projects": {
6         "starian-test": {
16             "architect": {
17                 "build": {
19                     "options": {
36                         "glob": "**/*",
37                         "input": "public"
38                     },
39                     {
40                         "glob": "**/*",
41                         "input": "src/assets",
42                         "output": "assets"
43                     }
44                 ],
45                 "styles": [
46                     "src/styles.scss",
47                     "node_modules/bootstrap/dist/css/bootstrap.min.css",
48                     "node_modules/bootstrap-icons/font/bootstrap-icons.css",
49                     "node_modules/ngx-toastr/toastr.css" ←
50                 ],
51                 "scripts": [
52                     "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
53                 ]

```

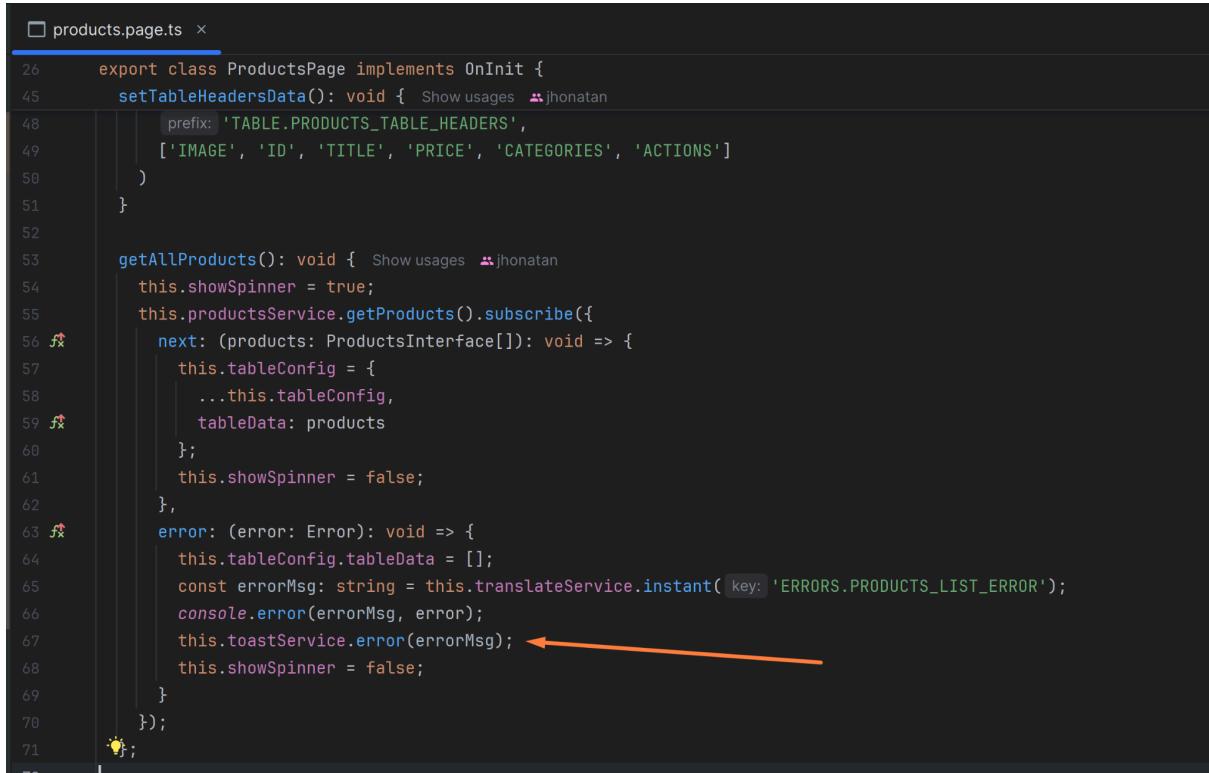
## Como Usar o Serviço de Toast

Para usar o serviço de toast é bem simples, basta injetar a dependência do serviço no componente que deseja utilizar, para a injeção de dependência, você pode adicionar via constructor ou do modo atual via inject, que aí não precisa declarar constructor:



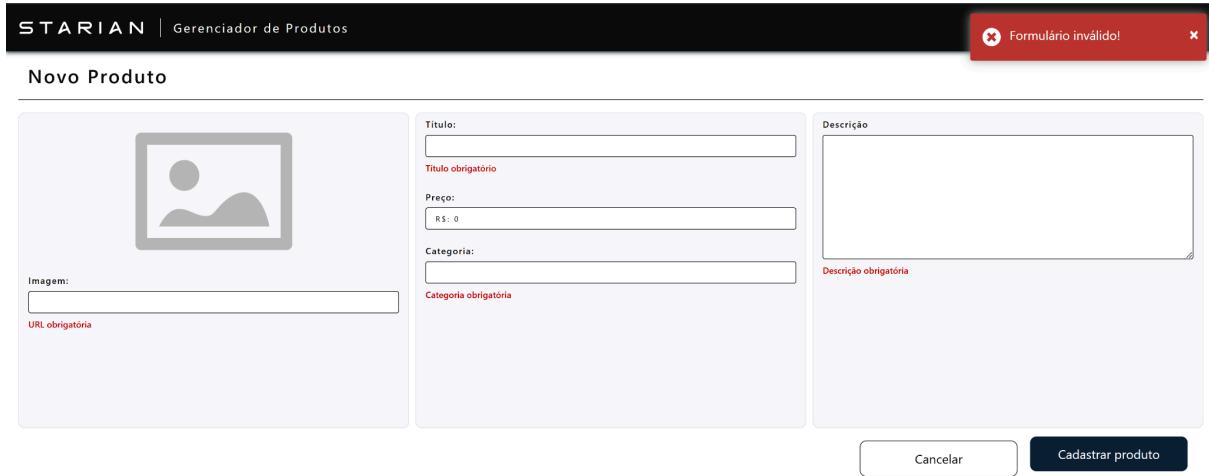
```
products.page.ts
17     imports: [
20         CommonModule,
21         CommonSpinnerComponent
22     ],
23     templateUrl: './products.page.html',
24     styleUrls: ['./products.page.scss'],
25   })
26   export class ProductsPage implements OnInit {
27     public showSpinner: boolean = false;
28     protected tableConfig: CommonTableConfigInterface = {
29       tableHeaders: [],
30       columnsToDisplay: [],
31       tableData: []
32     };
33     private readonly productService: ProductsService = inject(ProductsService);
34     private readonly toastService: ToastrService = inject(ToastrService); ←
35     private readonly translateService: TranslateService = inject(TranslateService);
36     private readonly utilsService: UtilsService = inject(UtilsService);
37     private readonly modalService: ModalService = inject(ModalService);
38     private readonly router: Router = inject(Router);
39   }
```

Para utilizar, basta chamar o método desejado, que pode ser sucess, error, warn ou info, dessa forma:



```
products.page.ts
26  export class ProductsPage implements OnInit {
45    setTableHeadersData(): void { Show usages ↵jhonatan
48      prefix: 'TABLE.PRODUCTS_TABLE_HEADERS',
49      ['IMAGE', 'ID', 'TITLE', 'PRICE', 'CATEGORIES', 'ACTIONS']
50    }
51  }
52
53  getAllProducts(): void { Show usages ↵jhonatan
54    this.showSpinner = true;
55    this.productsService.getProducts().subscribe({
56      next: (products: ProductsInterface[]): void => {
57        this.tableConfig = {
58          ...this.tableConfig,
59          tableData: products
60        };
61        this.showSpinner = false;
62      },
63      error: (error: Error): void => {
64        this.tableConfig.tableData = [];
65        const errorMsg: string = this.translateService.instant( key: 'ERRORS.PRODUCTS_LIST_ERROR');
66        console.error(errorMsg, error);
67        this.toastService.error(errorMsg); ←
68        this.showSpinner = false;
69      }
70    });
71  }
72
```

É suportado título e mensagem, no exemplo acima, eu passo somente a mensagem, ficando dessa forma:



STARIAN | Gerenciador de Produtos

Novo Produto

Formulário inválido!

Cancelar Cadastrar produto

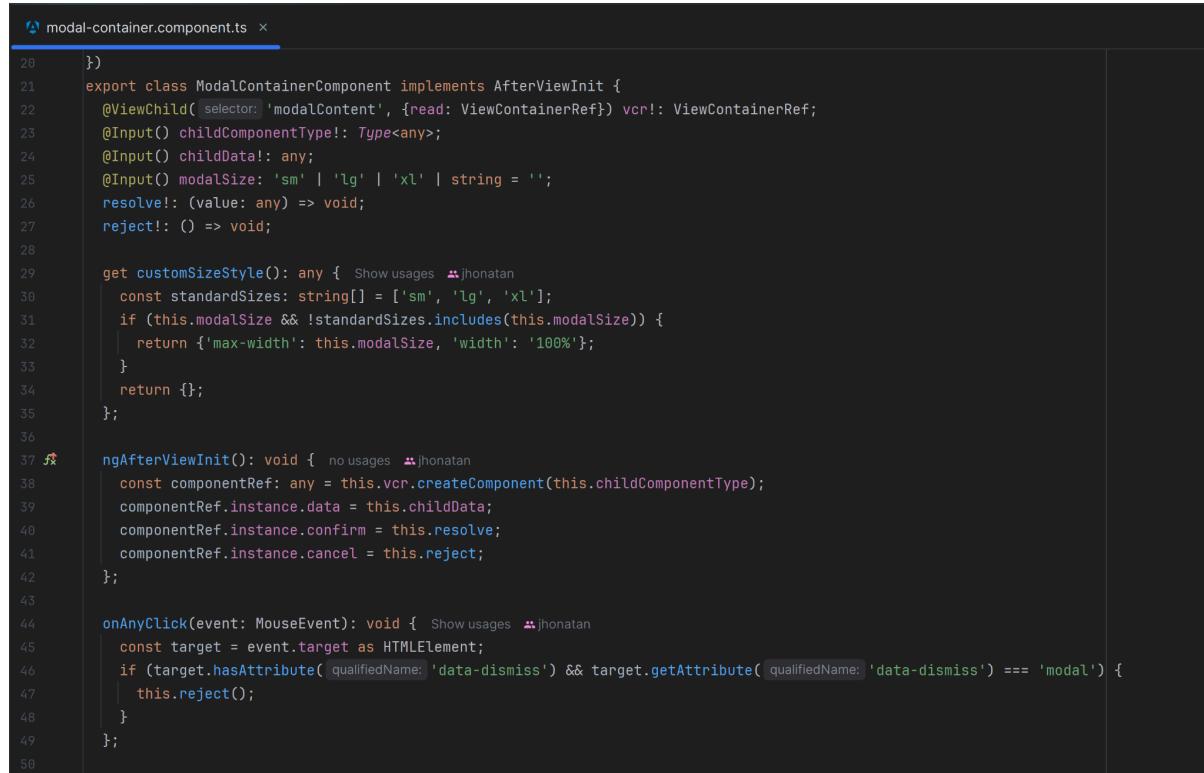
## Serviço de Modals

Foi criado um módulo que fornece um serviço de modal genérico e reutilizável. A abordagem adotada permite abrir dinamicamente componentes como modais,

encapsulando-os em um container (ModalContainerComponent) e manipulando a lógica de exibição via ModalService.

## Componente Base das Modals

Componente responsável criar e renderizar dinamicamente o componente filho passada via Input.



```
1  import { Component, ViewContainerRef, Input, AfterViewInit } from '@angular/core';
2  import { ModalContent } from './modal-content';
3
4  @Component({
5    selector: 'app-modal',
6    template: `
7      <div>
8        <ng-template #modalContent></ng-template>
9      </div>
10     <div class="modal-backdrop" (click)="onAnyClick($event)">
```

- Encapsular o conteúdo com a estrutura de modal do Bootstrap 5.
- Trata eventos de clique no backdrop e/ou botões com data-dismiss="modal" para fechamento.
- Expõem métodos de resolve() e reject() para comunicação com o chamador.

## Inputs de Dados

- **childComponentType:** Tipo do componente a ser instanciado dinamicamente.
- **childData:** Dados opcionais a serem injetados no componente modal.
- **ModalSize:** Tamanho da modal

## Comportamento

- **resolve:** Encaminhar a confirmação da ação (ex: "Confirmar").
- **reject:** Fechar o modal silenciosamente, sem disparar promessas.

## Modal Service

Serviço que:

1. Cria instâncias de ModalContainerComponent dinamicamente com createComponent;
2. Injeta o componente modal desejado como filho do container;
3. Anexa e remove o modal do DOM diretamente (document.body);
4. Expõe o método open(), retornando uma Promise, resolvida apenas se o usuário confirmar.

```
modal.service.ts
4  @Injectable({ Show usages ↵ Jhonatan Silva da Costa
5    providedIn: 'root'
6  })
7  export class ModalService {
8
9    private modalStack: ComponentRef<ModalContainerComponent>[] = [];
10
11   constructor( no usages ↵ Jhonatan Silva da Costa
12     private appRef: ApplicationRef,
13     private envInjector: EnvironmentInjector
14   ) {}
15
16   open<T>(component: Type<T>, data?: any, size?: 'sm' | 'lg' | 'xl' | string): Promise<any> { Show usages ↵ Jhonatan Silva da Costa
17     return new Promise((resolve: any): void => {
18       const modalRef: any = createComponent(ModalContainerComponent, {
19         environmentInjector: this.envInjector
20       });
21       this.appRef.attachView(modalRef.hostView);
22       const domElem = (modalRef.hostView as EmbeddedViewRef<any>).rootNodes[0] as HTMLElement;
23       document.body.appendChild(domElem);
24       this.modalStack.push(modalRef);
25       modalRef.instance.childComponentType = component;
26       modalRef.instance.childData = data || {};
27       modalRef.instance.modalSize = size || '';
28       modalRef.instance.resolve = (result: any): void => {
29         resolve(result);
30         this.close();
31       };
32       modalRef.instance.reject = (): void => {
33         this.close();
34       };
35     });
36   }
37
38   close(): void { Show usages ↵ Jhonatan Silva da Costa
39     const modalRef: any = this.modalStack.pop();
40     if (modalRef) {
41       this.appRef.detachView(modalRef.hostView);
42       modalRef.destroy();
43     }
44   }
45 }
```

### Método principal

open(component: Type, data?: any): Promise

### Fluxo

1. Cria ModalContainerComponent.
2. Injeta o componente filho.
3. Resolve a Promise somente quando confirm() for chamado no modal.

4. Em caso de cancelamento ou fechamento, apenas fecha a modal sem rejeitar a Promise.
5. É possível abrir uma modal sobre a outra.

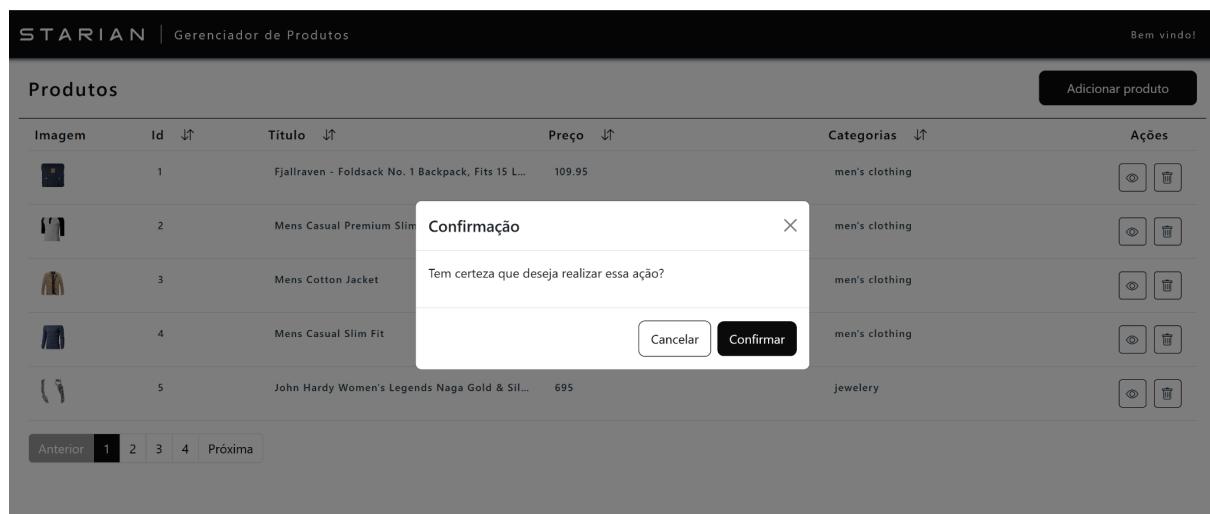
## Padrão de Uso

```
deleteProduct(product: ProductsInterface): void { Show usages ↵jhonatan
  void this.modalService.open(ModalConfirmation).then(): void => {
    this.productsService.deleteProduct(product.id).subscribe({
      next: (): void => {
        this.getAllProducts();
        this.toastService.success(this.translateService.instant( key: 'MODALS.CONFIRM_DELETE'));
      },
      error: (error: Error): void => {
        const errorMsg: string = this.translateService.instant( key: 'ERRORS.PRODUCT_DELETE_ERROR');
        console.error(errorMsg, error);
      }
    });
  });
}
```

## Importante

Como o cancelamento/fechamento não chama resolve() nem reject(), o .then() é disparado somente com a ação de confirmação. O .catch() é opcional e não necessário, pois o reject() foi removido para evitar exceções não tratadas.

## Exemplo visual

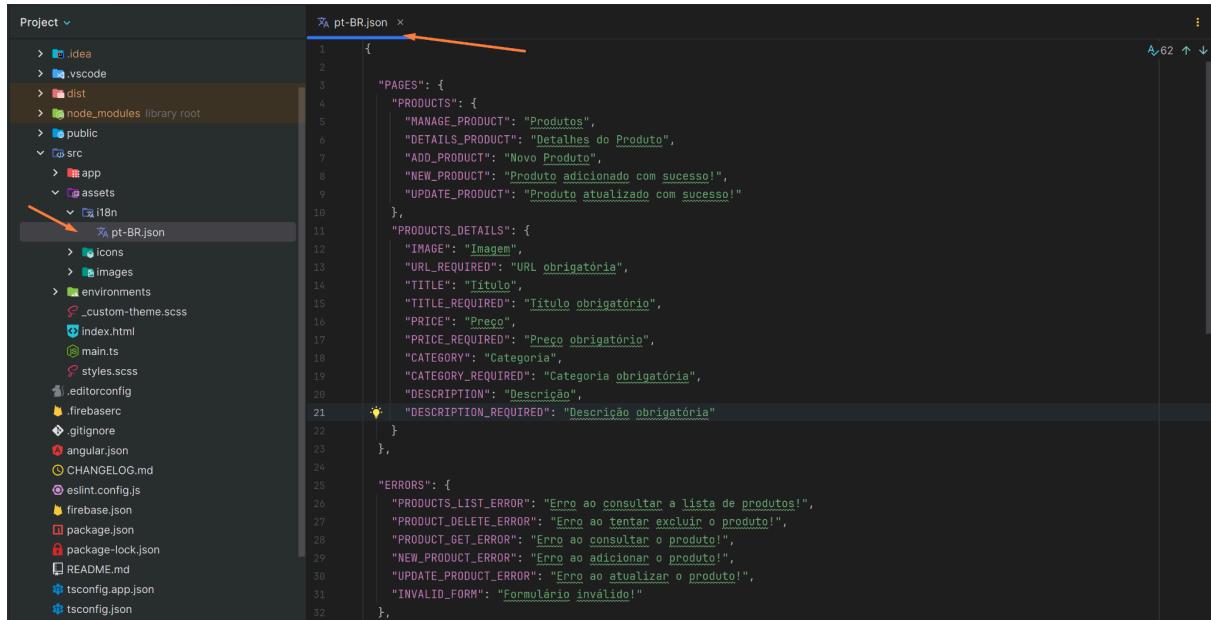


## i18n

No projeto é usado i18n para a setorização total dos textos do projeto em um arquivo json, os arquivos de traduções estão na pasta assets/i18n, por padrão tem

um arquivo somente, o pt-BR.json, mas nada impede de adicionar mais idiomas, exemplo en-US.json.

Com essa boa prática todos os textos estão setorizada por arquivos, facilitando a manutenção e atribuições para outras línguas, em melhorias futuras, exemplo do arquivo:



```
Project ▾
  > .idea
  > .vscode
  > dist
  > node_modules library root
  > public
  > src
    > app
      > assets
        > i18n
          pt-BR.json
    > icons
    > images
  > environments
    _custom-theme.scss
  > index.html
  > main.ts
  > styles.scss
  > .editorconfig
  > .firebaserc
  > .gitignore
  > angular.json
  > CHANGELOG.md
  > eslint.config.js
  > firebase.json
  > package.json
  > package-lock.json
  > README.md
  > tsconfig.app.json
  > tsconfig.json
```

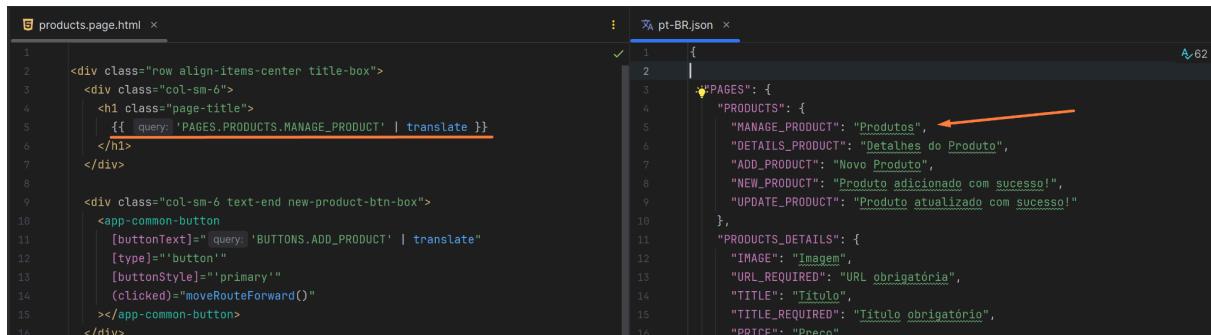
```
pt-BR.json
1  {
2
3   "PAGES": {
4     "PRODUCTS": {
5       "MANAGE_PRODUCT": "Produtos",
6       "DETAILS_PRODUCT": "Detalhes do Produto",
7       "ADD_PRODUCT": "Novo Produto",
8       "NEW_PRODUCT": "Produto adicionado com sucesso!",
9       "UPDATE_PRODUCT": "Produto atualizado com sucesso!"
10    },
11    "PRODUCTS_DETAILS": {
12      "IMAGE": "Imagen",
13      "URL_REQUIRED": "URL obrigatória",
14      "TITLE": "Título",
15      "TITLE_REQUIRED": "Título obrigatório",
16      "PRICE": "Preço",
17      "PRICE_REQUIRED": "Preço obrigatório",
18      "CATEGORY": "Categoria",
19      "CATEGORY_REQUIRED": "Categoria obrigatória",
20      "DESCRIPTION": "Descrição",
21      "DESCRIPTION_REQUIRED": "Descrição obrigatória"
22    }
23  },
24
25  "ERRORS": {
26    "PRODUCTS_LIST_ERROR": "Erro ao consultar a lista de produtos!",
27    "PRODUCT_DELETE_ERROR": "Erro ao tentar excluir o produto!",
28    "PRODUCT_GET_ERROR": "Erro ao consultar o produto",
29    "NEW_PRODUCT_ERROR": "Erro ao adicionar o produto",
30    "UPDATE_PRODUCT_ERROR": "Erro ao atualizar o produto",
31    "INVALID_FORM": "Formulário inválido!"
32  }
},
```

## Uso do recurso

Para usar, basta adicionar um parâmetro que serve com chave de localização, e seu valor, o valor é o texto que vai aparecer para o usuário.

No arquivo foi separado por setores, têm PAGES, que é o setor onde tem textos referente às páginas do sistema, também tem o setor ERRORS, que traz todos os textos de feedback de erro para o usuário, e assim por diante.

Uma vez feitos os textos desejados, é usado a Pipe e/ou o service do Translate, exemplo:



```
products.page.html
1
2   <div class="row align-items-center title-box">
3     <div class="col-sm-6">
4       <h1 class="page-title">
5         {{ query: 'PAGES.PRODUCTS.MANAGE_PRODUCT' | translate }}
6       </h1>
7     </div>
8
9     <div class="col-sm-6 text-end new-product-btn-box">
10       <app-common-button
11         [buttonText]="'BUTTONS.ADD_PRODUCT' | translate"
12         [type]="'button'"
13         [buttonStyle]="'primary'"
14         (clicked)="moveRouteForward()"
15       ></app-common-button>
16     </div>
17   </div>
```

```
pt-BR.json
1  {
2
3   "PAGES": {
4     "PRODUCTS": {
5       "MANAGE_PRODUCT": "Produtos", ←
6       "DETAILS_PRODUCT": "Detalhes do Produto",
7       "ADD_PRODUCT": "Novo Produto",
8       "NEW_PRODUCT": "Produto adicionado com sucesso!",
9       "UPDATE_PRODUCT": "Produto atualizado com sucesso!"
10    },
11    "PRODUCTS_DETAILS": {
12      "IMAGE": "Imagen",
13      "URL_REQUIRED": "URL obrigatória",
14      "TITLE": "Título",
15      "TITLE_REQUIRED": "Título obrigatório",
16      "PRICE": "Preço",
17    }
18  },
19
20  "ERRORS": {
21    "PRODUCTS_LIST_ERROR": "Erro ao consultar a lista de produtos!",
22    "PRODUCT_DELETE_ERROR": "Erro ao tentar excluir o produto!",
23    "PRODUCT_GET_ERROR": "Erro ao consultar o produto",
24    "NEW_PRODUCT_ERROR": "Erro ao adicionar o produto",
25    "UPDATE_PRODUCT_ERROR": "Erro ao atualizar o produto",
26    "INVALID_FORM": "Formulário inválido!"
27  }
},
```

Note no exemplo, foi passado o caminho das chaves até o texto desejado, depois é importado a pipe translate ao lado do texto.

Para usar em arquivo .TS é um pouco diferente, primeiro deve ser injetado a dependencia, seja via constructor ou via inject (padrão mais moderno), exemplo:

```
products.page.ts
22 ],
23 templateUrl: './products.page.html',
24 styleUrls: ['./products.page.scss'],
25 )
26 export class ProductsPage implements OnInit {
27   public showSpinner: boolean = false;
28   protected tableConfig: CommonTableConfigInterface = {
29     tableHeaders: [],
30     columnsToDisplay: [],
31     tableData: []
32   };
33   private readonly productService: ProductService = inject(ProductService);
34   private readonly toastService: ToastrService = inject(ToastrService);
35   private readonly translateService: TranslateService = inject(TranslateService); ←
36   private readonly utilsService: UtilsService = inject(UtilsService);
37   private readonly modalService: ModalService = inject(ModalService);
38   private readonly router: Router = inject(Router);
39 
```

Uma vez injetado, você deve chamar o service e consumir o método instant, passando o caminho da chave da tradução:

```
products.page.ts
26 export class ProductsPage implements OnInit {
27   setTableHeadersData(): void { Show usages ↗,jonatan
28   }
29
30   getAllProducts(): void { Show usages ↗,jonatan
31   this.showSpinner = true;
32   this.productService.getProducts().subscribe({
33     next: (products: ProductsInterface[]): void => {
34       this.tableConfig = {
35         ...this.tableConfig,
36         tableData: products
37       };
38       this.showSpinner = false;
39     },
40     error: (error: Error): void => {
41       this.tableConfig.tableData = [];
42       const errorMsg: string = this.translateService.instant('ERRORS.PRODUCTS_LIST_ERROR');
43       console.error(errorMsg, error);
44       this.toastService.error(errorMsg);
45       this.showSpinner = false;
46     }
47   });
48 }
49 
```

```
pt-BR.json
3 "PAGES": {
4   "PRODUCTS_DETAILS": {
5     "TITLE_REQUIRED": "Título obrigatório",
6     "PRICE": "Preço",
7     "PRICE_REQUIRED": "Preço obrigatório",
8     "CATEGORY": "Categoria",
9     "CATEGORY_REQUIRED": "Categoria obrigatória",
10    "DESCRIPTION": "Descrição",
11    "DESCRIPTION_REQUIRED": "Descrição obrigatória"
12  }
13
14 "ERRORS": {
15   "PRODUCTS_LIST_ERROR": "Erro ao consultar a lista de produtos!",
16   "PRODUCT_DELETE_ERROR": "Erro ao tentar excluir o produto!",
17   "PRODUCT_GET_ERROR": "Erro ao consultar o produto!",
18   "NEW_PRODUCT_ERROR": "Erro ao adicionar o produto!",
19   "UPDATE_PRODUCT_ERROR": "Erro ao atualizar o produto!",
20   "INVALID_FORM": "Formulário inválido!"
21 }
22
23 "MODALS": {
24   "CONFIRM_TITLE": "Confirmação"
25 }
```

O sistema ja conta com um sistema de isAppReady para impedir qualquer fluxo de iniciar o app antes da tradução, essa configuração fica no arquivo .TS principal, o app.ts, nele também é feito a adição da tradução inicial a ser usada:

```
Project ▾
  > endpoints
  > interfaces
  > modals
  > services
  > shared
  > views
    ↘ app.config.ts
    ↗ app.html
    ↗ app.routes.ts
  ↗ app.ts
  > assets
  > environments
    ↘ _custom-theme.scss
  ↗ index.html
  ↗ main.ts
  ↗ styles.scss
  .editorconfig
  .firebaserc
  .gitignore
  angular.json
  CHANGELOG.md
  eslint.config.js
  firebase.json
  package.json
  package-lock.json
  README.md
```

```
app.ts x
1  > import ...
2
3  @Component({
4    selector: 'app-root',
5    imports: [
6      RouterOutlet,
7      HeaderComponent
8    ],
9    templateUrl: './app.html',
10   })
11  export class App implements OnInit {
12    protected isAppReady: boolean = false;
13    protected isAuthUrl: boolean = false;
14    private readonly router: Router = inject(Router);
15    private readonly translate: TranslateService = inject(TranslateService);
16
17    ngOnInit(): void {
18      this.translate.addLangs(['pt-BR', 'en-US']);
19      this.translate.use(lang: 'pt-BR');
20
21      this.router.events.pipe(
22        filter((event: any): any => event instanceof NavigationEnd)
23        .subscribe((event: NavigationEnd): void => {
24          this.isAuthUrl = event.urlAfterRedirects.includes(searchString: '/auth');
25          this.isAppReady = true;
26        });
27      }
28    }
29  }
```

Caso um dia o app seja apresentado em inglês inicialmente, é aqui que você define a linguagem.

A configuração de onde ele pega os arquivos e os providers necessários estão dentro do arquivo de configuração, que no angular 20 fica em app.config.ts:

```
app.config.ts x
24  export const appConfig: ApplicationConfig = { Show usages jhonatan
25    providers: [
26      {
27        easing: 'ease-in-out',
28        progressBar: false,
29        positionClass: 'toast-top-right',
30        preventDuplicates: true
31      },
32      importProvidersFrom([TranslateModule.forRoot({
33        loader: {
34          provide: TranslateLoader,
35          useFactory: httpTranslateLoader
36        },
37      }]),
38      {
39        provide: LOCALE_ID,
40        useValue: 'pt-BR'
41      },
42      {
43        provide: TRANSLATE_HTTP_LOADER_CONFIG,
44        useValue: {
45          prefix: '/assets/i18n/',
46          suffix: '.json',
47        },
48      },
49    ]
50  };
```

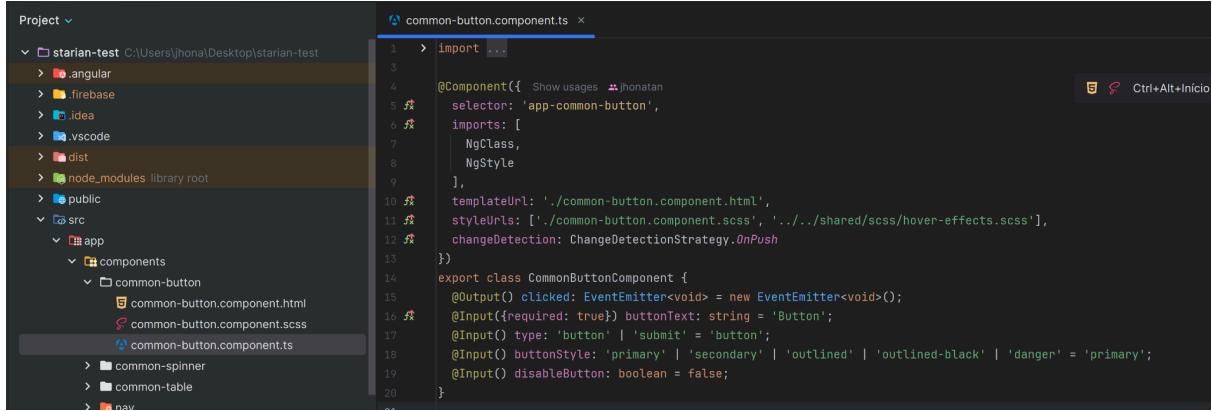
Uma vez configurado, não precisa mais mexer nessa configuração, porque aqui ele seta o idioma inicial no navegador e sistema e também passa a configuração de onde estão a pasta desses arquivos .json.

## Componentes reutilizáveis

Componentes reutilizáveis estão localizados na pasta components dentro do sistema, eles têm configurações de somente renderização, métodos e passagem de dados é responsabilidade do componente pai.

### Common Button

Botão dinâmico do sistema, tem configurações e estilos e pode ser usado no sistema inteiro, a ação de clique passa um void para o componente pai, que pode receber esse evento, através do EventEmitter e redirecionar para o que desejar.



The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The project structure includes folders for angular, firebase, idea, vscode, dist, node\_modules, public, src, and app. Under the app folder, there is a components folder containing common-button, common-spinner, common-table, and nav. The code editor displays the content of common-button.component.ts:

```
Project v
  starian-test C:\Users\jhonat\Desktop\starian-test
    .angular
    .firebase
    .idea
    .vscode
    dist
    node_modules library root
    public
    src
      app
        components
          common-button
            common-button.component.html
            common-button.component.scss
            common-button.component.ts
          common-spinner
          common-table
          nav

common-button.component.ts
1 > import ...
2
3
4 @Component({
  selector: 'app-common-button',
  imports: [
    NgClass,
    NgStyle
  ],
  templateUrl: './common-button.component.html',
  styleUrls: ['./common-button.component.scss', '../../../../../shared/scss/hover-effects.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Para usar basta chamar o botão onde desejado e efetuar o importe dele, o sistema é standalone components, então vai ser aplicado o import somente no componente que estará fazendo uso do botão, ou seja, esse componente só carrega onde é realmente usado.

```
products.page.html ×

1 <div class="row align-items-center title-box">
2   <div class="col-sm-6">
3     <h1 class="page-title">
4       {{ query: 'PAGES.PRODUCTS.MANAGE_PRODUCT' | translate }}
5     </h1>
6   </div>
7
8
9   <div class="col-sm-6 text-end new-product-btn-box">
10    <app-common-button
11      [buttonText]={{ query: 'BUTTONS.ADD_PRODUCT' | translate }}
12      [type]="'button'"
13      [buttonStyle]="'primary'"
14      (clicked)="moveRouteForward()"
15    ></app-common-button>
16  </div>
17</div>
```

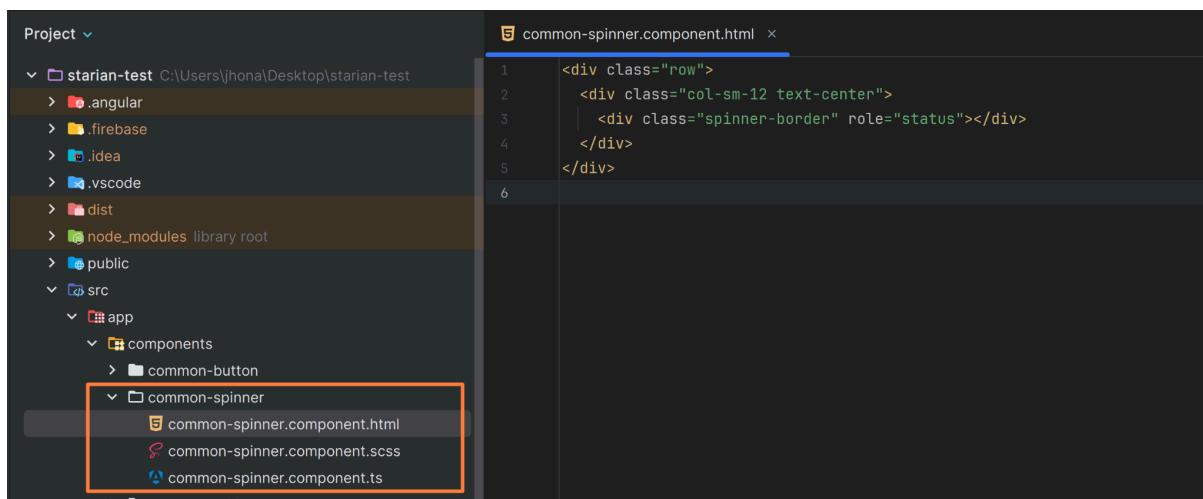
Exemplo do importe no componente que está utilizando esse componente:

```
products.page.html products.page.ts ×

1 import {Component, inject, OnInit} from '@angular/core';
2 import {ProductsService} from '../../../../../services/products/products.service';
3 import {ProductsInterface} from '../../../../../interfaces/products.products.interface';
4 import {ToastrService} from 'ngx-toastr';
5 import {TranslatePipe, TranslateService} from '@ngx-translate/core';
6 import {CommonButtonComponent} from '../../../../../components/common-button/common-button.component';
7 import {CommonTableComponent} from '../../../../../components/common-table/common-table.component';
8 import {CommonTableConfigInterface} from '../../../../../interfaces/common-table.common-table.interface';
9 import {UtilsService} from '../../../../../services/utils/utils.service';
10 import {ModalService} from '../../../../../services/modal/modal.service';
11 import {ModalConfirmation} from '../../../../../modals/modal-confirmation/modal-confirmation';
12 import {CommonSpinnerComponent} from '../../../../../components/common-spinner/common-spinner.component';
13 import {Router} from '@angular/router';
14
15 @Component({
16   selector: 'app-products',
17   imports: [
18     TranslatePipe,
19     CommonButtonComponent, ←
20     CommonTableComponent,
21     CommonSpinnerComponent
22   ],
23 })
```

## Common Spinner

Componente de spinner simples, usado para carregamento pelo sistema todo



Spinner simples somente com classes do bootstrap Para uso é só chamar o componente e fazer o importe onde deseja ser utilizado:

The screenshot shows a code editor with a dark theme. The template part of a component is displayed, featuring an `<app-common-spinner></app-common-spinner>` tag. An orange arrow points from the text above to this specific tag.

```
<div class="row">
  <div class="col-sm-12">
    @if (showSpinner) {
      <app-common-spinner></app-common-spinner> ←
    } @else {
      <app-common-table
        [tableConfig]="tableConfig"
        (detailsClicked)="updateProduct($event)"
        (deleteClicked)="deleteProduct($event)"
      ></app-common-table>
    }
  </div>
</div>
|
```

## Common table

Componente mais complexo, é uma tabela totalmente dinâmica, que espera receber as colunas a serem exibidas, o headers da tabela e um any array, com essa configuração, ela pode ser utilizada em todo o sistema com qualquer tipo de array de objeto, a tabela se molda e já aplica paginação, tratamento de ellipse caso o texto seja grande, ações de clique caso possua a coluna actions, e também sorting.

lembrando que paginação e sorting é feito no frontend, logo espera receber o array completo, e não via paginação no backend. Melhorias futuras podem ser aplicadas para sort e paginação passarem para o backend, mas nesse projeto, é feito no frontend.

## Configuração da Tabela

Tabela tem uma interface de configuração chamada common-table.interface.ts

```

1 export interface CommonTableConfigInterface { Show usages ↗ jhonatan
2   columnsToDisplay: string[];
3   tableHeaders: string[];
4   tableData: any[];
5 }
6

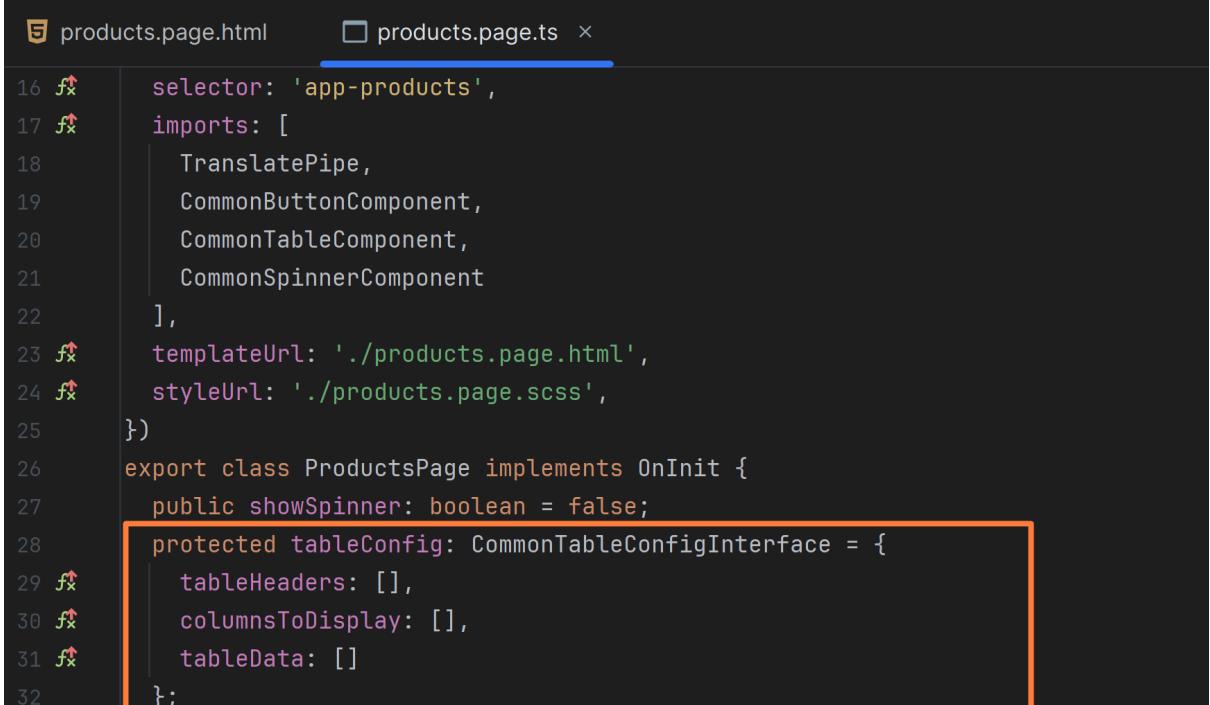
```

- **columnsToDisplay:** É a chave do array de objetos a ser mostrado de fato, exemplo: title
- **tableHeaders:** É os headers da tabela, pois no exemplo de produtos, eu quero mostrar a coluna title, mas o header deve ser Título, nesse projeto é ainda mais evidente a setorização, pois o título é dinâmico via i18n. (mais detalhes logo mais abaixo no tópico i18n no header da tabela)
- **tableData:** É o array de objetos que é renderizado na tela, no cenário atual, o componente pai busca o array de objetos, e passa esses dados para essa variável que alimenta a tabela.

## Uso da tabela

Para usar a tabela, como dito no tópico acima, você vai precisar de 3 itens de configurações, as colunas a serem exibidas, os headers da tabela e o array de objetos da tabela.

Iniciamos criando uma variável que conterá a interface de configuração, no exemplo é iniciada vazia e atribuído os valores no init.



```
products.page.html      products.page.ts ×

16  selector: 'app-products',
17  imports: [
18    TranslatePipe,
19    CommonButtonComponent,
20    CommonTableComponent,
21    CommonSpinnerComponent
22  ],
23  templateUrl: './products.page.html',
24  styleUrls: ['./products.page.scss'],
25)
26 export class ProductsPage implements OnInit {
27   public showSpinner: boolean = false;
28   protected tableConfig: CommonTableConfigInterface = {
29     tableHeaders: [],
30     columnsToDisplay: [],
31     tableData: []
32   };

```

Na página de produtos tem um método somente para atribuir os headers da tabela:

```

products.page.ts
26  export class ProductsPage implements OnInit {
27    protected tableConfig: CommonTableConfigInterface = {
28      columnsToDisplay: [],
29      tableData: []
30    };
31
32    private readonly productService: ProductService = inject(ProductService);
33    private readonly toastService: ToastrService = inject(ToastrService);
34    private readonly translateService: TranslateService = inject(TranslateService);
35    private readonly utilsService: UtilsService = inject(UtilsService);
36    private readonly modalService: ModalService = inject(ModalService);
37    private readonly router: Router = inject(Router);
38
39
40    ngOnInit(): void { no usages ↵jhonatan
41      this.setTableHeadersData();
42      this.getAllProducts(); ←
43    };
44
45    setTableHeadersData(): void { Show usages ↵jhonatan
46      this.tableConfig.columnsToDisplay = ['image', 'id', 'title', 'price', 'category', 'actions']
47      this.tableConfig.tableHeaders = this.utilsService.buildAndTranslateTableHeaders(
48        prefix: 'TABLE.PRODUCTS_TABLE_HEADERS',
49        ['IMAGE', 'ID', 'TITLE', 'PRICE', 'CATEGORIES', 'ACTIONS']
50      )
51    }
52

```

columnsToDisplay recebe as keys do array de objeto que quero mostrar na tabela, já os headers recebe o retorno de um método chamado buildAndTranslateTableHeaders(), que traduz o campo com o i18n e retorna o header traduzido, seja ele pt-BR, en-US ou outro.

esse método fica no utils.service.ts:

```

utils.service.ts
1  > import ...
3
4  @Injectable({ Show usages ↵jhonatan
5    providedIn: 'root',
6  })
7  export class UtilsService {
8
9    private readonly translateService: TranslateService = inject(TranslateService);
10
11    buildAndTranslateTableHeaders(prefix: string, stateKeys: string[]): string[] { Show usages ↵jhonatan
12      return stateKeys.map((key: string) :any => this.translateService.instant(`key: ${prefix}.${key}`));
13    }
14
15

```

O prefix é o path até o objeto i18n com as traduções, e o stateKeys é as chaves da tabela, exemplo:

```

products.page.html
products.page.ts
utils.service.ts
pt-BR.json

products.page.html
products.page.ts
utils.service.ts
pt-BR.json

26  export class ProductsPage implements OnInit {
27    protected tableConfig: CommonTableConfigInterface = {
28      columnsToDisplay: [],
29      tableData: []
30    };
31
32    private readonly productService: ProductService = inject(ProductService);
33    private readonly toastService: ToastrService = inject(ToastrService);
34    private readonly translateService: TranslateService = inject(TranslateService);
35    private readonly utilsService: UtilsService = inject(UtilsService);
36    private readonly modalService: ModalService = inject(ModalService);
37    private readonly router: Router = inject(Router);
38
39    ngOnInit(): void { no usages ↵jhonatan
40      this.setTableHeadersData();
41      this.getAllProducts();
42    }
43
44    setTableHeadersData(): void { Show usages ↵jhonatan
45      this.tableConfig.columnsToDisplay = ['image', 'id', 'title', 'price', 'categories'];
46      this.tableConfig.tableHeaders = this.utilsService.buildAndTranslateTableHeader(
47        [
48          ...'TABLE.PRODUCTS_TABLE_HEADERS',
49          ['IMAGE', 'ID', 'TITLE', 'PRICE', 'CATEGORIES', 'ACTIONS']
50        ]
51      );
52
53      getAllProducts(): void { Show usages ↵jhonatan
54        this.showSpinner = true;
55        this.productService.getProducts().subscribe(() => {
56          this.showSpinner = false;
57        });
58      }
59    }
60  }

33
34  "MODALS": {
35    "CONFIRM_TITLE": "Confirmação",
36    "CONFIRM": "Tem certeza que deseja realizar essa ação?",
37    "CONFIRM_DELETE": "Produto excluído com sucesso!"
38  },
39
40  "BUTTONS": {
41    "ADD_PRODUCT": "Adicionar produto",
42    "CONFIRM": "Confirmar",
43    "CANCEL": "Cancelar",
44    "EDIT_PRODUCT": "Editar produto",
45    "NEW_PRODUCT": "Cadastrar produto"
46  },
47
48  "TABLE": {
49    "PRODUCTS_TABLE_HEADERS": {
50      "IMAGE": "Imagem",
51      "ID": "Id",
52      "TITLE": "Título",
53      "PRICE": "Preço",
54      "CATEGORIES": "Categorias",
55      "ACTIONS": "Ações"
56    }
57  }
58
59

```

Mantendo o padrão acima mostrado, você consegue traduzir quaisquer campos do header da tabela e ainda mostrar somente as colunas desejadas, pois é setorizado as responsabilidades.

Com essa configuração feita, basta chamar e importar o método da tabela no componente desejado e passar os valores para ela:

```

products.page.html
products.page.ts
utils.service.ts
pt-BR.json

products.page.html
products.page.ts
utils.service.ts
pt-BR.json

2  <div class="row align-items-center title-box">
3    </div>
4
5    <div class="col-sm-6 text-end new-product-btn-box">
6      <app-common-button
7        [buttonText]="'query: 'BUTTONS.ADD_PRODUCT' | translate"
8        [type]="'button'"
9        [buttonStyle]="'primary'"
10       (clicked)="moveRouteForward()"
11     ></app-common-button>
12   </div>
13
14 <div class="row">
15   <div class="col-sm-12">
16     @if (showSpinner) {
17       <app-common-spinner></app-common-spinner>
18     } @else {
19       <app-common-table
20         [tableConfig]="tableConfig"
21         (detailsClicked)="updateProduct($event)"
22         (deleteClicked)="deleteProduct($event)"
23       ></app-common-table>
24     }
25   </div>
26 </div>
27
28
29
30
31

```

Como mostrado na imagem acima, tem coluna de ações na tabela, com 2 métodos disponíveis, o detail e o delete, você consegue captar o clique e receber o item clicado, e redirecionar para a lógica que precisar.