

AULA 1

O Angular 19, lançado em novembro de 2024, pelo google consolidou a maior transformação da história do framework: a transição definitiva para um modelo de reatividade baseada em Signals e o fim da dependência obrigatória de módulos (NgModules).

os pilares fundamentais desta versão:

- Standalone Components como Padrão
  - Reatividade Nativa com Signals (Estáveis)
  - Zoneless"
  - Novas Funcionalidades de Template

| Nome         | Angular "Antigo" (< v17)                  | Angular Moderno (v19)             |
|--------------|---|-----------------------------------|
| Organização  | Baseada em NgModules                      | Standalone (Módulos opcionais)    |
| Estado       | Variáveis comuns + Zone.js                | Signals (Granular e performático) |
| Sintaxe HTML | Diretivas estruturais (*ngIf, *ngFor)     | Control Flow (@if, @for, @switch) |
| Async        | RxJS (muito complexo para estado simples) | Signals + Resource API            |

## INSTALAR O ANGULAR CLI

Comando para saber se tem o angular instalado: `ng version`

```
$ ng version
```



Angular CLI : 21.1.4  
Node.js : 24.13.1  
Package Manager : npm 11.5.2  
Operating System : win32 x64

Em caso de não ter o Angular no terminal digitar : `npm install -g @angular/cli` esse comando será

executado apenas uma vez pois está instalando o angular de forma global em seu sistema operacional.

## Comandos

`npm install -g @angular/cli`

**Apenas uma vez** (ou quando quiser atualizar a versão do Angular no seu PC)

`ng new nome-do-projeto`

**Sempre** que for iniciar um projeto novo do zero.

`ng serve`

**Sempre** que for abrir um projeto já criado para começar a trabalhar nele.

`npm install -g @angular/cli@latest`

para atualizar a versão ,vai substituir a versão antiga pela mais nova globalmente

## PASSO 2- CRIAR UM PROJETO

No terminal se não tiver criado a pasta do projeto digite: `ng new meu-projeto`

Caso já esteja dentro da pasta que será criado o projeto digite: `ng new exemplo0 --directory ./`  
(O . diz para instalar dentro da estrutura atual)

A seguinte estrutura será apresentada, perguntando qual folha de estilo deseja utilizar

```
○ $ ng new .
? Which stylesheet system would you like to use?
> CSS      [ https://developer.mozilla.org/docs/Web/CSS ]]
  Tailwind CSS [ https://tailwindcss.com ]]
  Sass (SCSS)  [ https://sass-lang.com/documentation/syntax#scss ]]
  Sass (Indented) [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]]
  Less        [ http://lesscss.org ]]

↑↓ navigate • ↵ select
```

Depois vai perguntar se Deseja ativar a renderização do lado do servidor (SSR) e a geração de sites estáticos (SSG/pré-renderização)?

### SSR (Server-Side Rendering)

Normalmente, o Angular carrega uma página em branco e o navegador do usuário "monta" o site. Com o SSR, o servidor entrega a página já "montada".

- **Vantagem:** O site carrega mais rápido para o usuário e é melhor para o **SEO** (aparecer no Google).
- **Desvantagem:** Deixa o desenvolvimento um pouco mais complexo para quem está aprendendo agora.
- 

### SSG (Static Site Generation)

Transforma seu site em arquivos estáticos (HTML puro) antes mesmo de você publicá-lo. É excelente para blogs ou sites que não mudam o conteúdo todo.

Escolha **N** pois vamos criar uma SPA (Single Page Application) e depois criamos manualmente as configurações necessárias.

O CLI agora oferece a opção de configurar automaticamente arquivos de contexto para ferramentas de IA (como Cursor, Copilot ou Gemini), ajudando essas IAs a entenderem melhor a estrutura do seu

projeto.

Escolha **None** ( e quando estiver em seus projetos pode escolher a que mais lhe convém)

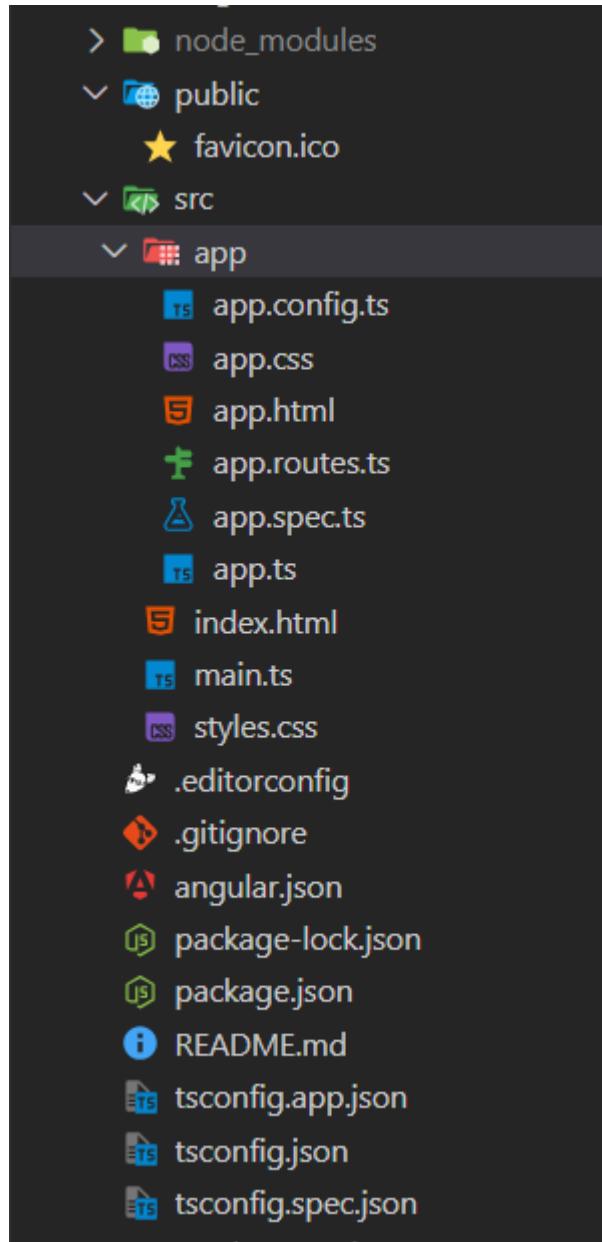
```
○ $ ng new exemplo0 --directory ./
  ✓ Which stylesheet system would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS
  ✓ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
  ? Which AI tools do you want to configure with Angular best practices? https://angular.dev/ai/develop-with-ai
    > None
    ○ Agents.md      [ https://agents.md/ ]
    ○ Claude         [ https://docs.anthropic.com/en/docs/clause-code/memory ]
    ○ Cursor          [ https://docs.cursor.com/en/context/rules ]
    ○ Gemini          [ https://ai.google.dev/gemini-api/docs ]
    ○ GitHub Copilot [ https://code.visualstudio.com/docs/copilot/copilot-customization ]
    ○ JetBrains AI    [ https://www.jetbrains.com/help/junie/customize-guidelines.html ]
```

↑↓ navigate • space select • a all • i invert • ⌘ submit

Projeto criado com sucesso.

```
● $ ng new exemplo0 --directory ./
  ✓ Which stylesheet system would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS
  ✓ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
  ✓ Which AI tools do you want to configure with Angular best practices? https://angular.dev/ai/develop-with-ai None
  CREATE angular.json (1982 bytes)
  CREATE package.json (955 bytes)
  CREATE README.md (1520 bytes)
  CREATE tsconfig.json (990 bytes)
  CREATE .editorconfig (331 bytes)
  CREATE .gitignore (666 bytes)
  CREATE .vscode/extensions.json (134 bytes)
  CREATE .vscode/launch.json (490 bytes)
  CREATE .vscode/mcp.json (188 bytes)
  CREATE .vscode/tasks.json (1020 bytes)
  CREATE src/app/app.spec.ts (698 bytes)
  CREATE src/app/app.ts (302 bytes)
  CREATE src/app/app.css (0 bytes)
  CREATE src/app/app.html (20446 bytes)
  CREATE src/main.ts (228 bytes)
  CREATE src/app/app.config.ts (324 bytes)
  CREATE src/app/app.routes.ts (80 bytes)
  CREATE tsconfig.app.json (444 bytes)
  CREATE tsconfig.spec.json (456 bytes)
  CREATE public/favicon.ico (15886 bytes)
  CREATE src/index.html (307 bytes)
  CREATE src/styles.css (81 bytes)
  ✓ Packages installed successfully.
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
  Successfully initialized git.
```

Estrutura de Pasta



### Pastas Principais

**node\_modules/**: É o "depósito" do projeto. Aqui ficam guardadas todas as bibliotecas e códigos que o Angular baixou para funcionar. **Dica:** Você nunca mexe aqui manualmente.

**public/**: Contém arquivos que serão servidos exatamente como são, como o ícone do site (favicon.ico).

**src/**: **O coração do projeto.** É aqui que você vai passar 95% do seu tempo escrevendo código.

### **Arquivos dentro da src/**

**app/**: É onde ficam os seus componentes (o HTML, CSS e a lógica que criamos anteriormente).

**index.html**: O arquivo principal. O Angular "injeta" seus componentes dentro deste arquivo. Você

raramente precisará editá-lo.

**main.ts**: É o motor de arranque. Ele diz ao navegador: "Ei, comece a rodar o Angular a partir do componente principal!".

**styles.css**: Onde você coloca o CSS **global**. O que for escrito aqui vale para o site inteiro.

### Pasta src/app

**app.ts**: É o "cérebro". Aqui você escreve a lógica em TypeScript (como as funções do contador e os **Signals**). Antigamente se chamava app.component.ts.

**app.html**: É o "corpo". Aqui você escreve o HTML, definindo o que vai aparecer visualmente (botões, textos, imagens).

**app.css**: É a "roupa". Aqui você coloca os estilos que afetam **apenas** este componente, garantindo que o design dele não estrague o resto do site.

**app.config.ts**: É o "manual de regras". Ele configura como o app deve se comportar globalmente, como as rotas e os provedores de dados. Substitui boa parte do que era feito no antigo app.module.ts.

**app.routes.ts**: É o "mapa". Aqui você define as rotas do site (ex: qual componente deve aparecer quando o usuário acessar /contato).

**app.spec.ts**: É o "inspetor". Este arquivo serve para escrever **testes automatizados** para garantir que seu código funciona. Iniciantes podem ignorá-lo por enquanto.

### Arquivos de Configuração

Arquivos que dizem como o projeto deve se comportar:

- **angular.json**: O manual de instruções do Angular CLI. Define como o projeto é compilado e onde estão os arquivos principais.
- **package.json**: A lista de compras. Contém o nome do projeto e todas as bibliotecas que ele precisa para rodar.
- **tsconfig...json**: Arquivos de configuração do **TypeScript**. Eles dizem ao tradutor como transformar seu código TypeScript em JavaScript puro para o navegador entender.
- **.gitignore**: Diz ao Git (sistema de controle de versão) quais pastas ele deve ignorar (como a pesada node\_modules).
- **.editorconfig**: Ajuda a manter o padrão de escrita (espaços, abas) entre diferentes editores de código.

Para executar o projeto e já abrir o navegador automaticamente digite no terminal:

```
ng serve --open
```

### Limpando o projeto

#### 1. Apagar imagens do public

#### 2. Limpar o Template (app.html)

O Angular vem com uma página de boas-vindas cheia de links e CSS interno. Vamos apagar tudo. Colocar um `<h1>Projeto Inicializado! </h1>` para teste

#### 3. Limpar a Lógica (app.ts)

Vamos remover o título padrão e deixar a classe pronta para receber nossos Signals.

#### 4. Limpar o Estilo Global (src/styles.css)

Às vezes o Angular ou o navegador aplicam margens padrão que atrapalham o layout dos alunos.

**Signal** é a tecnologia que torna o Angular 19 tão rápido.

Antigamente: O Angular tinha que "adivinhar" o que mudou na página inteira toda vez que o usuário clicava em algo. Com Signals: O framework sabe exatamente qual pequena parte da tela precisa ser atualizada, economizando processamento do computador ou celular do usuário.

### **Exemplo:**

```
<h1>Bem-vindo ao projeto: {{ title() }}</h1>
```

### **Decorators**

O conceito: A Analogia do "Selo de Qualidade"

Imagine que você tem uma caixa de papelão comum (uma Classe). Se você colar um selo de "Frágil" nela, você não mudou a caixa, mas deu a ela instruções especiais: "esta caixa deve ser manuseada com cuidado". No Angular, um **Decorator** é esse "selo". Ele é uma função marcada com o símbolo @ que adiciona metadados (instruções extras) a uma classe, sem que precisemos mudar o código interno dela.

### **Exemplo:**

```
@Component({ // <-- Aqui começa o "Selo" ou Decorator
}) export class App { ... } // A classe é apenas uma caixa vazia; o Decorator é que diz que ela é um Componente.
```

### **Outros Decorators**

**@Directive**: Cria comportamentos personalizados para tags HTML.

**@Pipe**: Transforma a forma como um dado aparece (ex: formatar moeda).

**@Input / @Output**: Permitem que um componente "converse" com outro pai ou filho.

Dica:

" sempre que vocês virem um @ no código, pensem: '**Estou dando superpoderes para o que vem logo abaixo**'. O @Component transforma uma classe em uma parte da interface do site, o @Injectable a transforma em um serviço, e assim por diante."

### **Tipos de Binding**

**Interpolação** {{ }} Exibe um dado do TS no meio do texto HTML.

**Property Binding** [prop] Define o valor de um atributo HTML (como src, disabled, href).

**Event Binding** (click) Faz o HTML avisar o TS que o usuário clicou ou interagiu.

**Two-Way Binding** [(ngModel)] Sincronização total entre o campo de digitação e a variável.

Material criado por Prof. Wellington Cidade Silva Tenório