**Curso Angular**

# O que é Angular?

Angular é um framework JS desenvolvido pelo Google para criação de aplicações Web SPA (Single Page Applications) baseada em **componentes.**

# Angular CLI – Como instalar o CLI

Para **começar** é necessário instalar o CLI ou Angular CLI, isso requer que o node.js já esteja instalado na máquina. Para verificar a existência do node e sua versão, basta executar o comando **node -v.** Caso não tenha o node.js, simplesmente faça o download.

Sendo assim, execute o comando **npm install -g angular-cli**

## Criação de um projeto

Para criar um projeto, utilizando o CLI, vá até a pasta desejada e execute o comando **ng new “nome\_projeto”**

Após o projeto ser **criado**, execute um **ng serve** para abrir o mesmo em localhost.

## Criação de Classes

Após ter criado um projeto, caso deseje criar um componente ou serviço, basta estar dentro do projeto criado anteriormente e executar **ng generate (g) e o nome da classe,** como **component (c),** para criar um componente. Caso a Classe seja um Service, substitua o **c** por **s.**

## Escolhendo o Pré-Processador de CSS

A primeira maneira é ao gerar um novo projeto. Segue um exemplo:



A segunda maneira é modificando um projeto já existente. Segue um exemplo:

Texto

Descrição gerada automaticamente

## **3 Comandos** importantes da ferramenta do Angular CLI

**ng lint:** irá realizar uma verificação por boas práticas do style guide, como algum caractere faltante.

**ng test:** executa testes unitários.

**ng e2e:** Testes E2E tem o objetivo de testar a interação com aplicações web tal como um usuário faria, e o framework **Protractor** foi desenvolvido para viabilizar isso.

## Package.json

Dentro das **dependências** estão as bibliotecas necessárias para executar o projeto. A maioria das bibliotecas são do próprio Angular.

Já nas **devDependencies** contem frameworks que necessitamos para executar o projetor enquanto em desenvolvimento. Como por exemplos frameworks de testes.

# Gerando Build de Produção

**Build de Dev =>** Comando: **ng build**

**Build de Produção=>** Comando **ng build --prod**

**TypeScript:** equivalente ao JS, porém com acréscimo de funcionalidades. O browser não entende estas novas funcionalidades e por isso o TS é compilado para o JavaScript.

**Árvore de Componentes:** No Angular o usuário começa a aplicação com um componente principal. O nome dado pelo Angular para este componente principal é **APP Component.**

Assim, a partir deste comp. principal são referenciados outros componentes. Segue um exemplo na imagem abaixo:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

# Conceitos Essenciais

**Inicialização da APP**

O primeiro arquivo que será chamado para iniciar a aplicação é o **main.ts.**

Na sequência, este main.ts irá chamar o **módulo da aplicação**, nomeado como **appModule**. Dizemos que um aplicação angular é **modularizada**, pois os componentes são organizados dentro de módulos. Estes módulos podem oferecer alguns poderes como por exemplo criar determinados componentes que estão visíveis apenas dentro do módulo.

Por padrão, quando é criada uma aplicação já existe um módulo, no caso o appModule, que será chamado para inicializar a aplicação. **Dentro** do appModule há um atributo chamado **bootstrap,** que irá apontar para o componente appComponent (criado por padrão).

A partir do **appComponent**, toda a árvore de componentes será chamada, contendo referências como Header, Content entre outros.

Diagrama

Descrição gerada automaticamente

# O que é um componente?

É um trecho de código que representa um componente visual da sua tela. Este componente visual possui um HTML, CSS e TS.

Sendo assim, irá existir por via de regra 3 arquivos padrão: [**home.component.css**](http://home.component.css)**;** [**home.component.html**](http://home.component.html)**;** [**home.component.ts**](http://home.component.ts)**.**

Para se tornar possível utilizar este componente, é criado uma tag personalizada, como por exemplo <app-home></app-home>. Esta é a forma para **referenciar** todo o código HTML, CSS e toda a lógica que foram criados dentro deste component.

Visão dentro de um TypeScript:

Texto

Descrição gerada automaticamente

# Organização usando Módulo

Dentro de uma aplicação, a organização é feita por componentes. Estes componentes por sua vez, estão organizados dentro de módulos.

É possível organizar os componentes dentro de um único módulo ou organizá-los em diversos módulos.

Segue um exemplo na imagem abaixo:

Uma imagem contendo relógio, bola, jogador, monitor

Descrição gerada automaticamente

Neste caso da imagem, o único componente que pode ficar visível apenas dentro do módulo, é o H, pois o H é um componente que está referenciando G, que por sua vez está dentro do **mesmo** Módulo.

# Anatomia do Módulo

Dentro do arquivo AppModule, irá conter 5 atributos, o objetivo destes 5 atributos é permitir a configuração do Module. Seguem os mesmos:

* **Declarations:** dentro de Declarations você deve declarar todos os componentes, Diretivas e Pipes que fazem parte deste determinado Módulo.
* **Exports:** todos os Componentes, Diretivas e Pipes que vão ficar visíveis fora do módulo, são colocados em Exports.
* **Imports:** dentro de Imports é possível importar outros módulos. Este módulo pode ser um módulo da sua própria aplicação ou um módulo externo, que importa uma dependência, como por exemplo uma biblioteca de recursos visuais.
* **Providers**: nos Providers são declarados os Services. **Serviços** são responsáveis por fazer a comunicação dos **componentes** com o **servidor** (backend), organizando a parte lógica.
* **BootStrap**: é o local em que é definido o principal componente (AppComponent) que será carregado no Módulo.

Segue um exemplo de imagem:

Interface gráfica do usuário

Descrição gerada automaticamente

Na prática (app.module.ts):

Texto

Descrição gerada automaticamente

Outro exemplo por imagem:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

# Elementos do Angular - Parte 1

## Componente Angular

É formado por via de regra por três propriedades, seguem as mesmas novamente: [*home.component.css*](http://home.component.css)*;* [*home.component.html*](http://home.component.html)*; home.component.ts;*

O componente, neste caso o **“home”** irá criar uma tag personalizada, como **“app-home”.** Esta tag personalizada é inserida dentro do **app.component.html**, tornando possível que todo código **HTML e CSS** criado dentro do component “**home”**, seja exibido na tela.

## Diretiva de Atributo

É focada em dois objetivos, alterar a **aparência (CSS) e/ou comportamento (TS)** de um elemento, componente ou outra diretiva.

Exemplo: colocar uma diretiva dentro de um determinado botão e esta diretiva faz com que seja feita uma chamada no backend, alterando assim o **comportamento** executando uma ação.

Exemplo de mudança de aparência:

Texto

Descrição gerada automaticamente

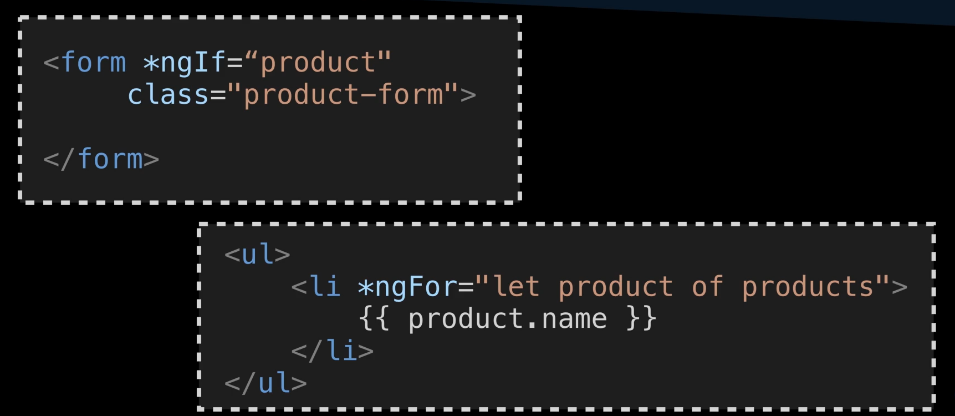
Texto

Descrição gerada automaticamente

## Diretiva Estrutural

Altera o layout/estrutura da página (HTML), **adicionando ou removendo** elementos da DOM (Document Object Model).

Exemplo:



## Property Binding

Data Binding é associar informações que estão no @component (TS) para o template e vice-versa. Existem algumas formas de fazer isso, uma delas é o Property Binding. Uma característica do Property Binding é a utilização dos colchetes “ [ ] ”.

Segue um exemplo:

Interface gráfica do usuário

Descrição gerada automaticamente

## Event Binding (Binding de Evento)

É relacionar um evento do HTML, para dentro de um método que está dentro de componente associado ao HTML. Uma característica do Event Binding é utilização dos parênteses “ ( ) ”.

Segue um exemplo:

Interface gráfica do usuário

Descrição gerada automaticamente

## One Way Data Binding

No exemplo dado abaixo, é possível visualizar que o sentido é dado do componente para o HTML. Sendo assim, qualquer alteração é refletida no HTML.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

## Two Way Data Binding

Já no caso do Two Way Data Binding, caso a alteração seja feita no componente, o HTML irá se alterar. Caso a alteração seja feita no HTML, o componente também irá se alterar/atualizar. Ou seja, ambos fazem esta atualização.

Exemplo:

Diagrama

Descrição gerada automaticamente

## Router

Durante a navegação dentro da aplicação, existe um mapeamento entre a rota e o componente. Sendo assim, o componente selecionado a partir da rota, será colocado no Router Outlet e será exibido.

Segue um exemplo:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Exemplo no código:

Texto

Descrição gerada automaticamente

## Pipes

Os Pipes são os processamentos que são realizados nas **variáveis**. Ou seja, para os dados que são comumente transformados. É possível por exemplo, realizar a formatação de um dado, caso o mesmo não esteja no formato desejado.

Exemplo:



É possível também realizar uma cadeia de Pipes. Na imagem abaixo é possível visualizar que é passado um primeiro Pipe (fullDate) seguido de um segundo Pipe (uppercase). Segue o exemplo:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Alguns outros exemplos existentes:

**Date:** Formata um valor de data de acordo com as regras locais.

**UpperCase:** Transforma todos caracteres do texto para maiúsculo.

**LowerCase:** Transforma todos caracteres do texto para minúsculo.

# Elementos do Angular - Parte 2

## Observables

**Conceito de programação Reativa:** conceito de que o código só será chamado de forma reativa, ou seja, somente quando há a necessidade de uma ação externa para assim, o código ser executado.

## Padrão Observer

* Padrão orientado a Evento

**Subject:** tem o objetivo de monitorar e detectar Eventos determinados pelos Observers.

Sendo assim, os **Observers** mandam requisições para o **Subject**, na qual passam Eventos específicos para serem monitorados pelo próprio Subject e assim que o determinado Evento for detectado/acontecer, o **Subject notifica** o **Observer** responsável pela solicitação.

Assim que o **Observer** for notificado é executado uma função especifica de acordo com o Evento que foi gerado. Segue uma imagem:

Diagrama

Descrição gerada automaticamente

Nesta imagem a seguir é possível visualizar os antecessores do **Observer.** Segue a imagem:

Interface gráfica do usuário

Descrição gerada automaticamente

Exemplo no código:

Texto

Descrição gerada automaticamente

# Elementos do Angular - Parte 3

## Services

São classes que tem como principal objetivo **organizar** e **compartilhar** métodos e dados entre **componentes**. É necessário ser cuidadoso para possuir Services com lógicas que tenham coerência de estarem juntos.

Interface gráfica do usuário, Diagrama

Descrição gerada automaticamente

Exemplo no código:

Texto

Descrição gerada automaticamente

A criação do ProductService vai ser gerenciada pelo Angular e será possível injetar esta classe nos **componentes e diretivas.**

## Injeção de Dependência

A Injeção de Dependência é um padrão no qual a classe recebe as dependências de uma fonte externa ao invés de criar por conta própria.

Exemplo na prática:

Texto

Descrição gerada automaticamente

O root é responsável por fazer a injeção de dependência. Quando é declarado o root, isso significa que só existirá uma única instancia em toda a aplicação, já que o root está associado ao AppModule

**Texto

Descrição gerada automaticamente**

Os Services são singletons dentro do escopo de um injector, ou seja, possuem uma única instancia.

**Diagrama

Descrição gerada automaticamente**

**JSON** é um acrônimo para "JavaScript Object Notation", é um formato leve para troca de dados computacionais.

**O JSON Server** lê um arquivo JSON e cria uma API baseada neste JSON

**API:** são uma forma de integrar sistemas e oferece alguns pontos positivos.

Exemplo prático:

Imagine um site de hotel que possui um mapa do Google, no qual o visitante da página possa ver a melhor forma de chegar ao local, analisando se o serviço atende as condições requisitadas. Idealize que seu potencial cliente realizou todo esse procedimento sem precisar sair da sua página

As APIs proporcionam essa integração de sistema com o Google Maps, otimizando a experiência do usuário em uma página ou aplicativo.

## Reusando Componentes – Input Properties

# Serviços (Services) e Injeção de dependência (DI)

**Serviços** são responsáveis por fazer a comunicação dos **componentes** com o **servidor** (backend), organizando a parte lógica.

**Injeção de dependência** é um padrão na qual uma classe solicita dependências de fontes externas. Ou seja, A DI visa separar a implementação de serviços dos componentes.

# Life Cycle Hooks (Ciclo de vida de um Componente)

Uma imagem contendo Texto

Descrição gerada automaticamente