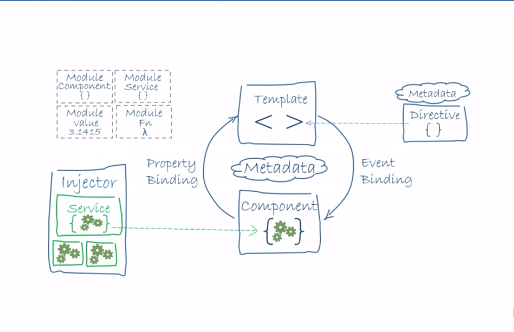
Angular

O Framework possui 8 blocos principais:

1. Componentes
2. Diretivas
3. Roteamento
4. Serviços
5. Template
6. Metadata
7. Data Binding
8. Injeção / Dependência



* **Componentes**: (view), ela encapsula:
  + Template
  + Metadata (processamento das classes)
  + Dado a ser mostrado na tela (Data Binding) (uma div por exemplo)

Quando criamos aplicação em Angular, criamos aplicações que são orientadas à componentes. Temos então: pai/mãe (que vai ser o componente raiz) e partir dele, deriva-se novos componentes.

Sendo que cada componente também pode ter um componente filho (como uma árvore binária)

O componente pode interagir com o back-end em inúmeras linguagens, como: NodeJS, Java, .NET, Ruby, Python... Quem faz o intermédio, entre o componente e o back-end é o **“Serviço”** ele pode ser injetado em outras classes

Por mais que o Angular trabalhe como SPA (Single Page Application), isso não impede que tenha várias abas dentro da mesma aplicação, o responsável pela navegação entre as páginas é o **Router.**

Além disso, temos uma Diretiva, a qual é responsável por modificar elementos DOM (Document Object Model) e/ou seu comportamento, exibir uma div, modificar valores, etc.

Utilizando Angular CLI para criar novos componentes:

$ ng generate componente <name>

Ou:

$ ng g c <name>

Para inicializar o projeto:

$ npm start

**TYPESCRIPT**

Para trabalhar com ecma script, precisaremos de um transpiller... um dos mais conhecidos é o babel.js.  
mas também é possível usar o ‘tsc’ no console para transpillar para js

$ tsc <nome-arquivo.ts>

Hoje em dia, temos os decorators para fazer essa parte. Depois procurar saber melhor

As variáveis podem ser tipadas, ou seja, definir o tipo primitivo dela, assim como C e Java. para isso: funciona dessa forma:

Var n1: string = “teste string recebida”;

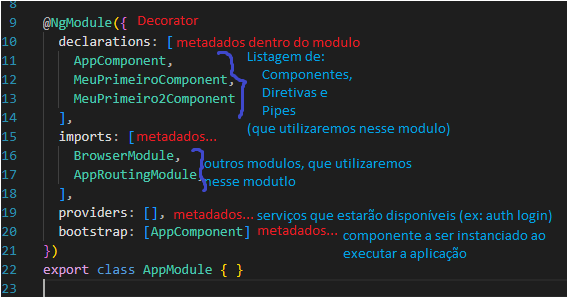
 n1 = 4 (causará erro, pois só aceita tipo string)

var n2: any = “outro teste”;

n2 = 5 (não causará erro, pois a n2 aceita qualquer tipo)

**Módulo**

Por que separar em módulos? Organizar melhor o código, escolher o que cada modulo expõe, etc.



Para gerar um modulo pelo CLI:

$ng generate modules <nome>

Ou

$ng g m <nome>

a

**Templates**

Com os templates é possível utilizar os dados dinamicamente. No component.ts, precisa declarar os atributos (de forma tipada) que utilizará no html. Por Exemplo:

Para fazer a interpolação nessa linha:

<p>Lista de cursos do portal: {{ nomePortal }}</p>

O atributo nomePortal já precisa estar declarado na classe do componente, e sua declaração é feita dessa forma:

export class CursosComponent {

  nomePortal: string;

  constructor() {

    this.nomePortal = 'https://loiane.training';

  }

}

**Serviços (injeção de dependências)**

$ng g s <nome do serviço>

A injeção de dependência serve para que o angular automaticamente forneça uma instância de uma classe de serviço, ao invés de usar um HttpRequest, isso é feito através de um decorator: @Injectable(); esse decorator fica acima da declaração da classe. Para usá-lo, seguem os passos:

* Precisa receber a classe como argumento do método construtor:

  constructor(private cursosService: CursosService) {

* (após isso, consigo utilizar um método dessa classe dentro do bloco do constructor):

    this.cursos = this.cursosService.getCursos();

* E também, preciso passar como um provider no “index” do modulo:

  providers: [

    CursosService

  ]

* Não esquecer de importar:

import { CursosService } from './cursos.service';

Plugins / Extensões:

Angular Snippets

Auto Import

**Novo projeto: Data Binding**

Data Binding -> associar informações que estão no componente para o template, e vice-versa

No angular existem 4 formas

**Interpolação:** “ {{ valor }} “

**Property Binding:** “ [propriedade]= “valor”; ”

**Event Listener:** “ (evento) = “handler”; “ -> (oposto do property binding)

**2 way database:** “[(ngModel)] = “propriedade” “ -> mantem os 2 componentes atualizados ao mesmo tempo.

**Class e Style Binding**

Para pegar o valor de um combo (um select) precisa criar uma variável local dentro do template, e para isso, precisa “escutar” quando o valor muda. Nesse caso:

<select name="" #classe (change)="0">

Class Binding pode ser usado com property Binding, mas também é possível usar a interpolação.

1. Class Binding com property binding:

<div class="alert" role="alert" [class.alert-success]="classe.value == 'alert-success'">Sucesso</div>

1. Class binding com interpolação:

<div class="alert {{ classe.value }}" role="alert" >Texto colorido conforme valor do combobox</div>

Event Binding

No property Binding, usamos colchetes para indicar qual a propriedade a ser alterada. No caso do evento, usamos parênteses. Ex:

<button (click)="botaoClicado()" class="btn btn-primary">Me Clique!</button><br><br>

**Anotações para se lembrar:**

Eventos:

onclick == (click)

onmouseover == (mouseover)

onmouseout == (mouseout)

Variáveis locais em elementos HTML:

Basta usar o ‘#’ e indicar o nome dela. Ex:



Inclusive no exemplo acima, usa-se a ‘#’ para atribuir um id, e utiliza o EVENTO onchange como == (change)

Diferença entre **Propriedade** (que via acima, no property-binding) e **Atributo**.

Uma tag, exemplo: <p id=’aoba’>Teste</p>

Possui um **atributo:** **ID**, de descrição ‘aoba’, que por sua vez, também possui uma propriedade de mesmas características, e a maioria dos atributos também geram propriedades de mesmo nome.   
 Algumas excessões são: o **atributo: for**. Neste caso, a propriedade gerada, é de nome htmlFor, pelo que vi, é uma das únicas excessões, pesquisar futuramente caso precise.

Então, no resumo, o atributo, é uma indicação do html. Quando isso é convertido pelo DOM, é Gerado uma propriedade. Que é como se fosse o “”objeto”” real. Até por isso, quando eu faço algo como:

<input id=’teste’ value=’TESTANDO’></input>

O input gerado já vem com o texto “TESTANDO” nele. Mas se eu deletar, e digitar novas coisas por cima, o Atributo value continuar sendo “testando”, mas a propriedade value será alterada para o valor que eu digitar.

Two Way Data-Binding

O angular possui uma propriedade chamada ngModel, ela é a representação de uma entidade, essa entidade pode ser um campo simples, como uma string, ou um objeto. (entidades essas, estão no arquivo .ts)

Além dessa propriedade, também possui um evento: ngModelChange, que é para quando tal campo for atualizado

Basicamente, deixa os valores atualizados entre template e componente

NgModel é uma diretiva, que pertence ao FormsModule

Input-Modules

Para passar um componente entre modulos, precisa antes no arquivo component.ts declarar que vai receber um input de tal nome, e precisa importar a biblioteca Input. Fica dessa forma:

@Input() nome: string = '';

@Input é um decorator

No caso de nomes diferentes para essa variável. Por exemplo, renomeando a variavel acima para nomeCurso, mas ainda pasando o parâmetro como nome la no html do template anterior. Esse decorator pode receber um argumento, o qual possibilita, expor externamente a variavel com outro nome

@Input('nome') nomeCurso: string = '';

Ex: externamente -> nome, internamente -> nomeCurso

Output modules

Ele possui decorators similares ao input (precisa importar também)

E tem um detalhe em um emissor de eventos. Declarado dessa forma no modulo

  @Output() mudouValor = new EventEmitter();

E para utilizar, utilizamos o nome da variavel.emit(), e pode se emitir varios tipos de dados, incluindo objetos, como está no exemplo a seguir:

    this.mudouValor.emit({novoValor: this.valor})

isso é executado dentro de uma função

Ciclo de Vida de Componentes:

Existe uma lista de lifecycle hooks (ou eventos disponiveis durante o ciclo de vida de um componente), link:

<https://angular.io/guide/lifecycle-hooks#:~:text=kinds%20of%20operations.-,HOOK%20METHOD,-PURPOSE>

ao mudar o valor de um componente, são disparados em sequencia:

ngOnChanges (mais usado)

ngDoCheck

ngAfterContentChecked

ngAfterViewChecked

Existe também o ngif

Veremos melhor no futuro

Acesso ao DOM / Template com ViewChild

  @ViewChild('campoInput')

  campoValorInput!: HTMLElement;

Utilizando o decorator @ViewChild, e passando como parametro a variavel local dentro do component.html é possivel visualizar o input diretamente, sem uso de html, etc.

Declarando a variável como HTMLElement foi possível visualizar no console.log qual o tipo do elemento correto

    console.log(this.campoValorInput)

 que nesse caso é ElementRef

Dessa forma é possivel modificar apenas o html.

Módulo de **ANGULAR CLI** (ferramenta de linha de comando)

Instalação Angular: (requer Node.js > v4)

$npm install -g angular-cli

Criação de novo projeto:

Opção 1:

$ng new <nomeProjeto>

Opção 2:

$mkdir <nomeProjeto>

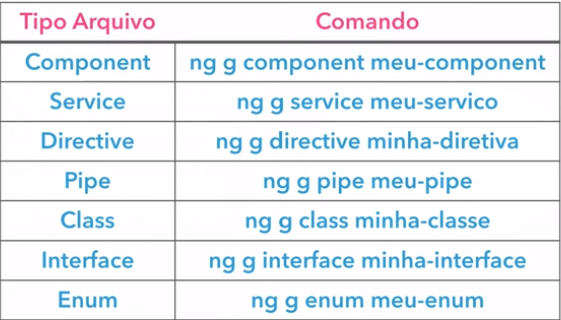
$ng init

Rodar o projeto:

$npm start

Generate:

$ng generate <classe>

**Classes**: 

Criar novo componente:

$ng generate component <nome>

$ng g c <nome> (aliases)

Criar novo serviço:

$ng generate service <nome>

$ng g s <nome> (alias)

Comandos ng lint, ng test and ng e2e

$ng lint: escaneia o projeto, e verifica se está de acordo com as boas práticas AND se há erros

$ng test

os arquivos .spec.ts seguem o padrão do Jasmine do JS, um framework? De testes unitários (Interessante procurar saber como o Jasmine funciona, para realizar os testes futuros)

$ng e2e

End to end, para testar a integração com outros apps

**Gerando o Build de Desenvolvimento (Deploy)**

4 opções para build de dev:

$ng build –target=development –environment=dev

$ng build –dev -e=dev

$ng build –dev

$ng build

Para facilitar, rodar apenas o ultimo.

O build será gerado dentro da pasta ./dist/

Buildando para dev, o codigo não é ofuscado, nem minificado, então é possível debugar, ler nomes de variáveis, fazer uma rodada de testes manuais.

3 opções para build de produção:

$ng build –target=production –environment=prod

$ng build –prod –env=prod

$ng build –prod

Obs: para rodar a versão buildada (como teste) é necessario utilizar uma ferramenta do node, chamada http-server, para instalá-lo:

$npm install http-server -g

E aí para rodar a versão teste: mude para a pasta, com $cd dist, e rodar:

$http-server

Mais fácil para testar codigos simples, já que não precisa de um servidr apache.

Para instalar bibliotecas externas:

Basicamente rodar:

$npm install –save bootstrap@next (ou procurar um codigo equivalente no npm lib)

E depois adicionar no campo de scripts



**Diretivas**

Diretivas Estruturais: interagem com a view e modificam a estrutura do DOM, exemplos: \*ngFor, ou \*ngIf

Diretivas de Atributo: interagem com o elemento que foi aplicado, exemplo: ng-class, ou ng-style

**Diretiva \*ngIf**

Tem o mesmo comportamento da estrutura condicional if da programação.

Não possui else