**Sobre angular**

**Módulos –** Limitador de contexto – Módulo é segregação de responsabilidade

**Componente** – Dá vida ao Módulo – Expressar o contato do módulo – Blocos que tem um sentido sozinho, podem ser únicos ou um conjunto de vários outros componentes.

* **- Reutilização de trechos de código**
* **- Isolamento de contexto**
* **- Legibilidade do Código**
* **- Redução de código**
* **- Padronização do Projeto**

**Metadados –** Definições que iremos informar ao angular e que a nossa classe é um componente por meio do decorator - **@Component**

**Classe** – POO



[ngModel] (com colchetes – OneWay DataBind – Só vai exibir, não vai alterar nosso atributo) – Relação de input com uma variável

([ngModel]) (com parêntese e colchetes - TwoWay DataBind – Lê e altera o nosso Atributo)

Implements OnInit

Quer mesclar HTML com um campo do seu atributo? Coloque entre colchetes

Ex:  <td><img [src]="course.imageUrl"></td>

@Input() – Deixa nossa variável elegível pra receber valor de um componente externo (de outro componente)

@Injectable torna a classe elegível para injeção de independência

Não é legal criar na classe serviço, variáveis que mudem de valor, apenas métodos ou variáveis estáticas

Começa com \_ o nome da variável? Quer dizer que ela só pode ser usada naquela classe

* **Pipes** – alterar a forma como será mostrada alguma coisa
* **Shared** – Pasta que contém componentes pequenos, mais genéricos
* **Core** – Pasta que contém componentes maiores, que não são tão genéricos
* **Decorator** – Define uma classe como componente, module, pipe, servisse e seus **metadados** que determina como o mesmo deverá ser processado

**Arquitetura de Componentes**

**Componente** – Fala da interface visual

Os componentes devem ser **independentes**, **substituíveis** e **modulares** que auxiliem no **gerenciamento da complexidade** e **encorajem** a **reutilização**.

**Benefícios**:

Escalabilidade

Manutenção

Performance

**Serviços** – O que o usuário não vê mas faz a lógica... A parte mais pesada

Responsáveis por **organizar** e **compartilhar** lógica de negócios.

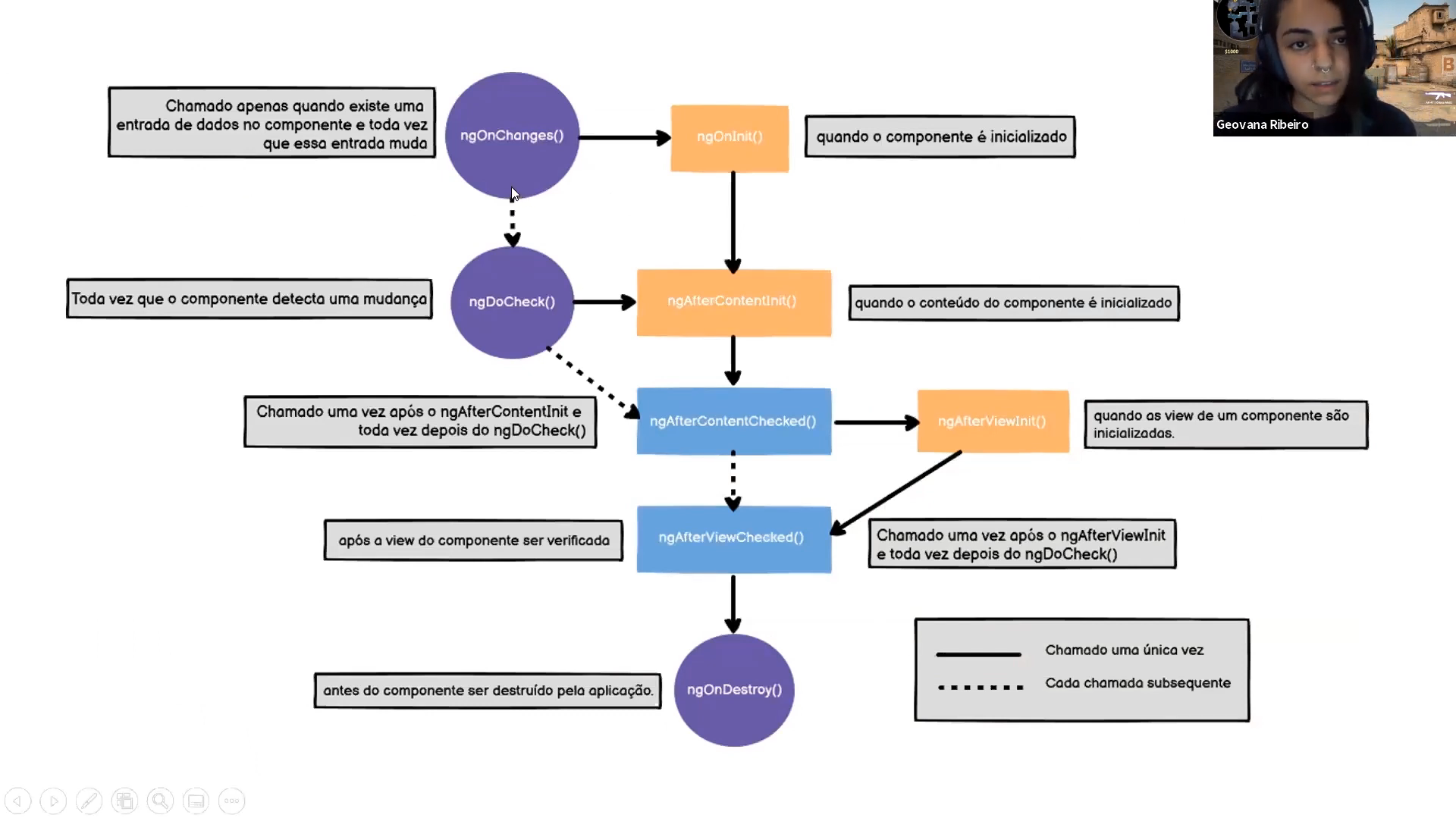
**Reutilizáveis** entre diferentes componentes de uma aplicação

**Mandatórios** para uma arquitetura modular e reutilizável.

**Injeção de Dependência** – **Todo serviço é uma dependência** que precisa ser instanciada dentro do componente para ser utilizada dentro do mesmo.

**Ciclo de vida do componente**

**Todo componente possui seu ciclo de vida (lifecycle hooks)**



**Construtor**

Deve ser utilizado apenas para inicializar serviços injetados

**ngOnInit**

Deve ser utilizado para todo tipo de lógica que o componente precisa executar após ter sido criado.

**Data Bindind** – Associa informações do Componente para o template

**Interpolation** – {{ }} – associa informação do componente para o template (HTML)

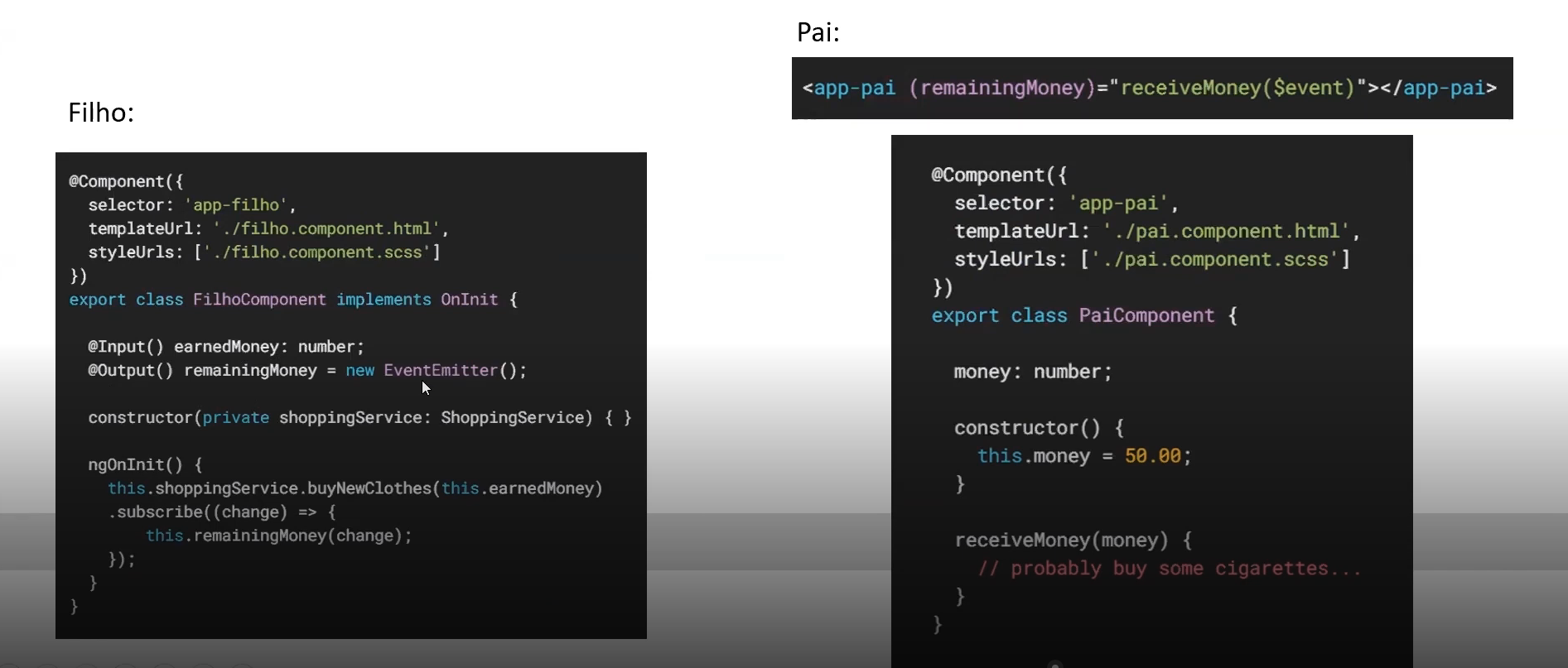
**Property Binding** – [ ] = “valor” – Associa informação do componente para propriedades do template (HTML)

**Event Binding** – (evento) = handler – Associa informações do template (HTML) para o componente

**Two-Way Data Binding:** [(ngModel)] = “propriedade” – Associa informação entre ambos, ou seja, mantém ambos atualizados (Componente e template)

**@Input** -> Principal maneira de compartilhar dados do pai para filho



**@Output** –e **EventMitter** >. Do filho pro Pai, contrário do de cima 

(tem que ter o .emmit dps de remainingMoney)

**Componentes inteligentes e apresentacionais**

**Apresentacional**

Parecido com funções puras

Se preocupam apenas com a interface do usuário

Não fica responsáveis por recuperar dados ou lidar com logica de negocio

Não causam efeitos colaterais na aplicação

Recebem dados via @Input e emite eventos via @Output

**Inteligentes**

Parecidos com funções impuras

Causam efeitos colaterais

Contém toda a lógica de negócio

Ficam responsáveis por repassar dados para os componentes apresentacionais

São internamente compostos por componentes apresentacionais

**Node modules** – Responsável por armazenar as dependências e pacotes do projeto

**Src** – Diretório da aplicação onde nós criamos os nossos módulos, componentes, serviços

**BrowserListrc** – Usado pelo sistema de build para ajustar saída css e js para suportar os navegadores listados

**EditorConfig** – configurações do editor de código

**Angular.json** – Arquivo de config do angular

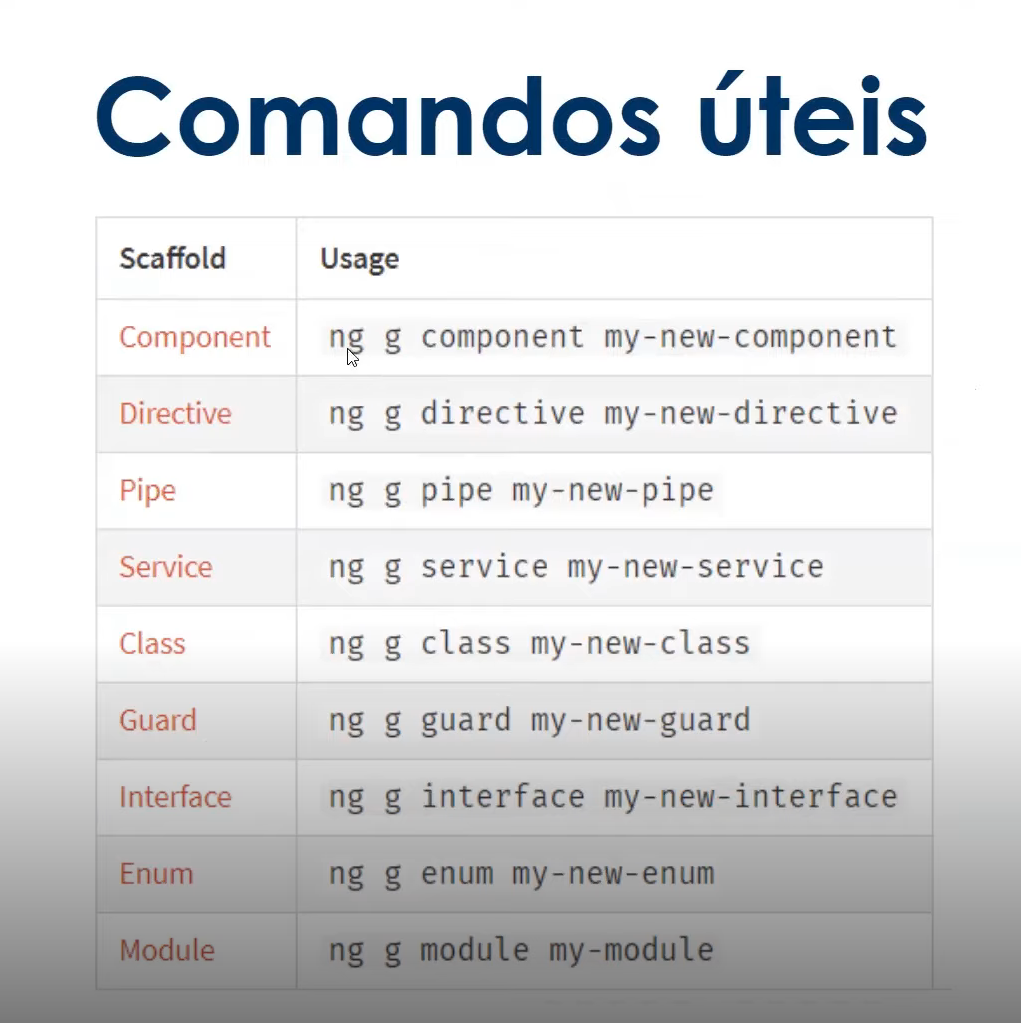
**Karma** – Teste unitário

**Package-lock –** lista de dependências e sub

**Package.json** – Configurações das dependências e scripts do NPM

**README.md**

**Comandos uteis**

****

**Design Modular**

**Core module** – Define serviços Singleton, componentes de instancia única, configuração e exportação de quaisquer módulos de terceiros necessários no modulo principal.

**Shared module –** Contém componentes/pipes/diretivas comuns e também exporta módulos do Angular usados com frequência (CommonsModule)

**Feature module –** Organiza um conjunto de recursos da aplicação num módulo de funcionalidade

**Library** – Possui código que pode ser reutilizado entre diferentes aplicações

**Angular Element** – Recurso do angular para criar web componentes, padrão da web para definir novos elementos HTML de uma maneira independente de estrutura e agnóstica de frameworks.

**SMACSS**

É uma arquitetura modular e escalável para CSS, dividida em 5 camadas

**Base** – Estilização de seletores e pseudo-classes, além de resets

**Layout** – Principais componentes como cabeçalho, rodapé, entre outros

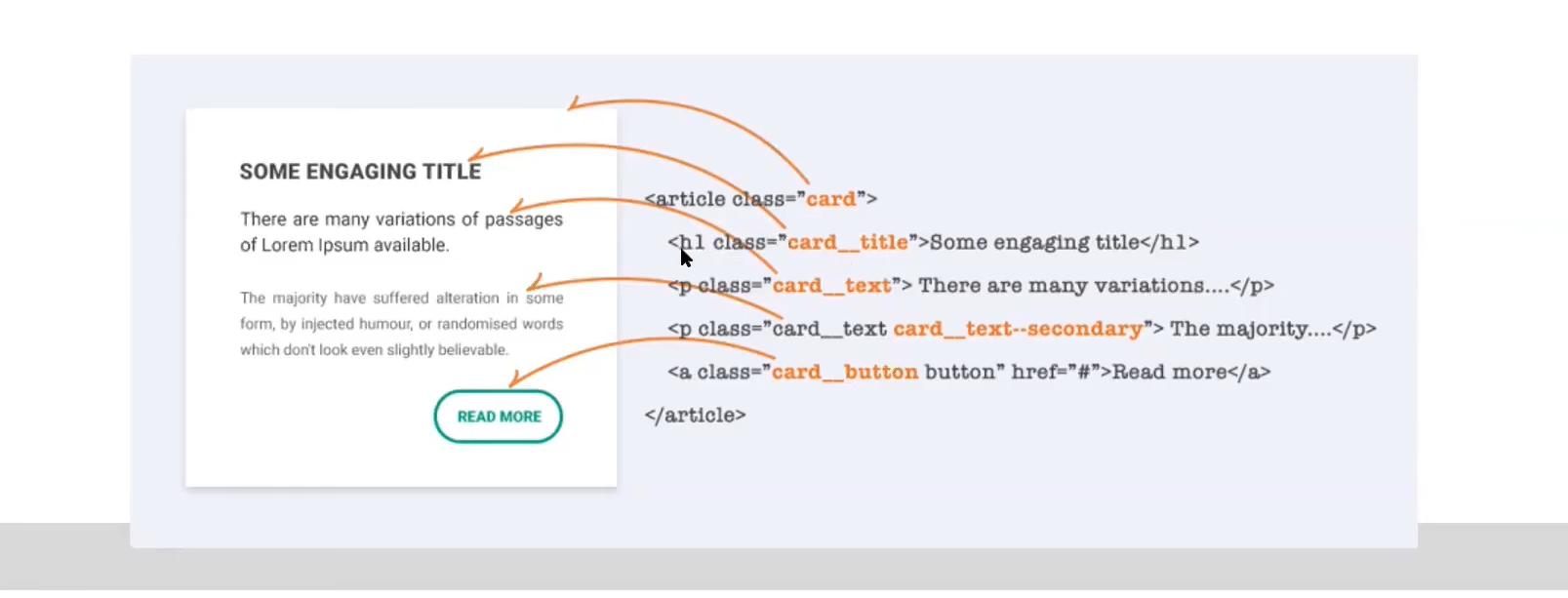
**Module** – Componentes reutilizados como botões e ícones

**State** – Todo elemento que será modificado ou terá alguma alteração no seu estado inicial

**Theme –** Temas específicos para uma mesma aplicação

**BEM CSS – Block, element e Modifier**

Serve para definir nomenclatura de nomes para classes CSS



**OOCSS – CSS Orientado a Objeto,** identifica um padrão visual que pode se repetir no projeto e o agrupa em classes, tornado os reutilizáveis.

**Diretivas**

Fornecem meios para que possamos manipular o DOM ou estender as funcionalidades do elemento.

**Diretivas de atributos** – Alteram a aparência ou o comportamento de um elemento, componente ou outra diretiva, como por exemplo, NgClass e NgStyle.

**Diretivas Estruturais** – Modificam o layout adicionando ou removendo elemento do DOM, como por exemplo, NgIf e NgFor

**Custom** – Customizadas, tudo que eu preciso fazer e não tem ainda

Logica para o template = []

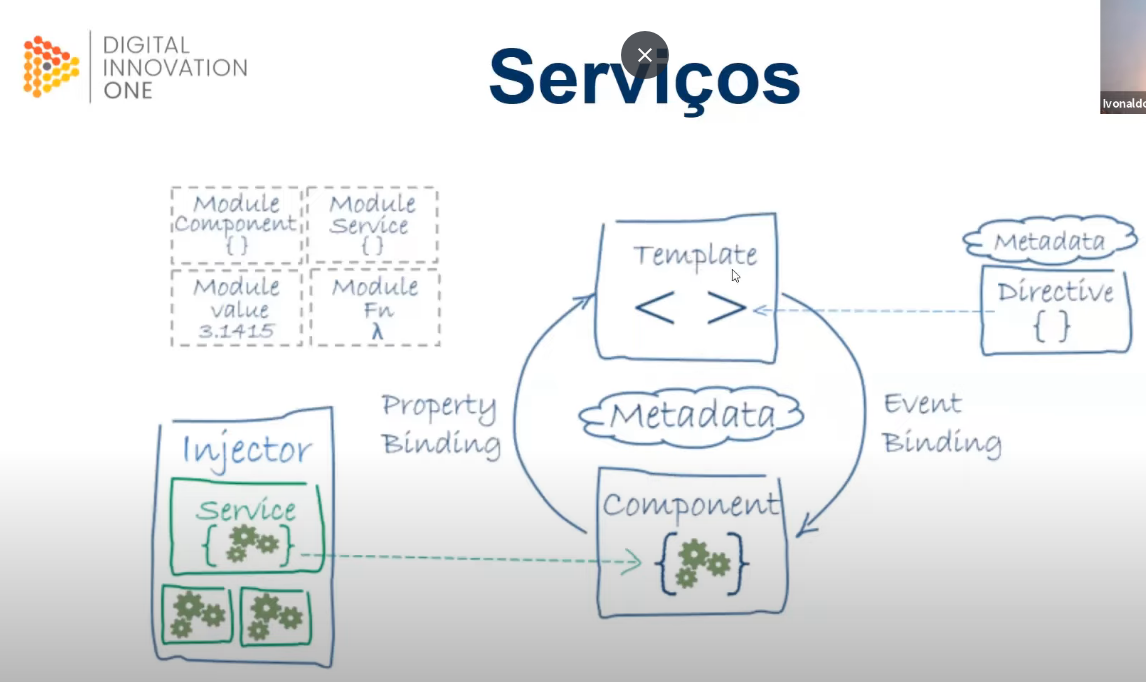
Template para logica = ( )

Ng generate directive – ng g d

**Serviços**

Classe feito pra deixar sua lógica de dados (Regras de negócio) encapsulada.

Se você escreve tudo em componente, você não poderá reutilizar sua lógica e terá que recodificar sempre que for usar em outro componente. Além disso a manutenção de seus componentes será difícil, pois você terá que manter duas cópias do mesmo código



**Injeção de dependência (@Injectable)**

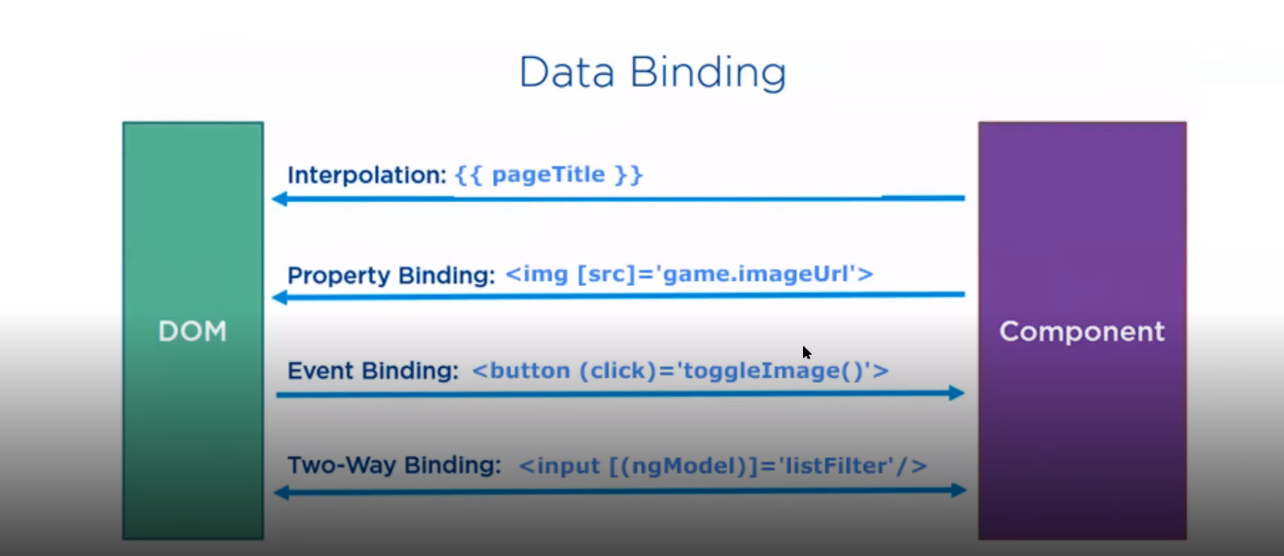
Fornecer aos componentes quais serviços ela pode usar

**Data Binding**

Componente pra View – **Interpolação** {{ valor }} ou **Property Binding** [propriedade] = “valor”

View para Componente – **Event Binding** (evento)

Ambos pra Ambos – **Two-Way Binding** [(ngModel)]= “propriedade”-



**ngOnInit** – Quando é inicializado (única vez)

**ngOnChanges** – Uma vez na criação do componentes e sempre que houve alteração em uma de suas propriedades de entrada

**NgDoCheck** – Cada ciclo de alterações

**ngAfterContentInit** – Dps que conteúdo externo é inserido

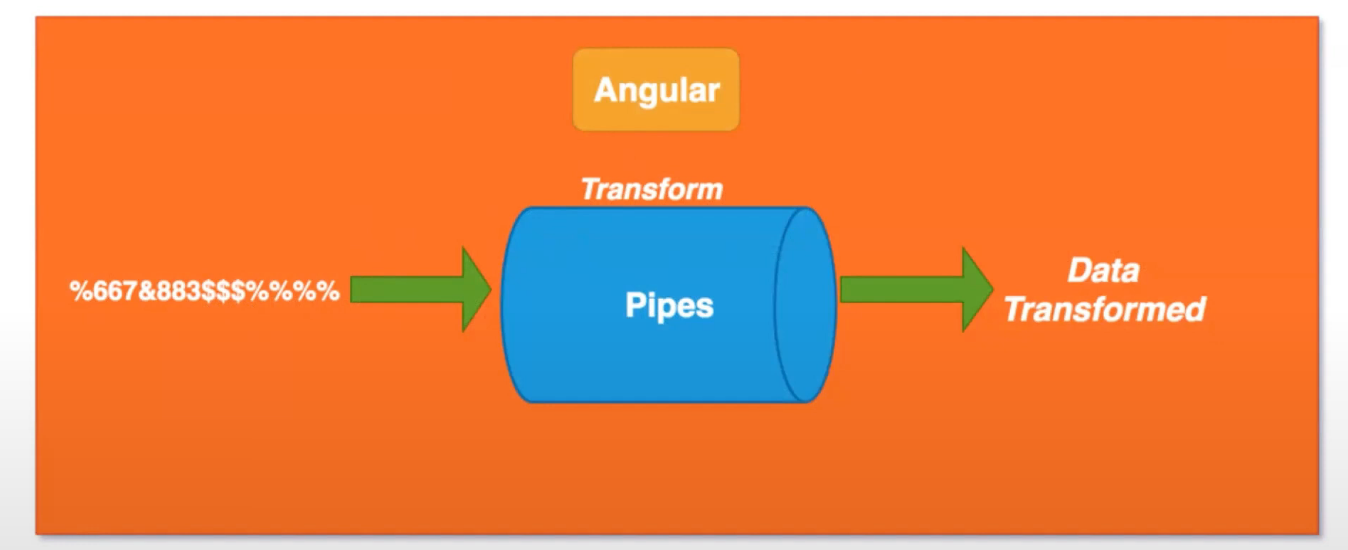
**ngAfterContentChecked** – Dps que conteúdo externo é validado

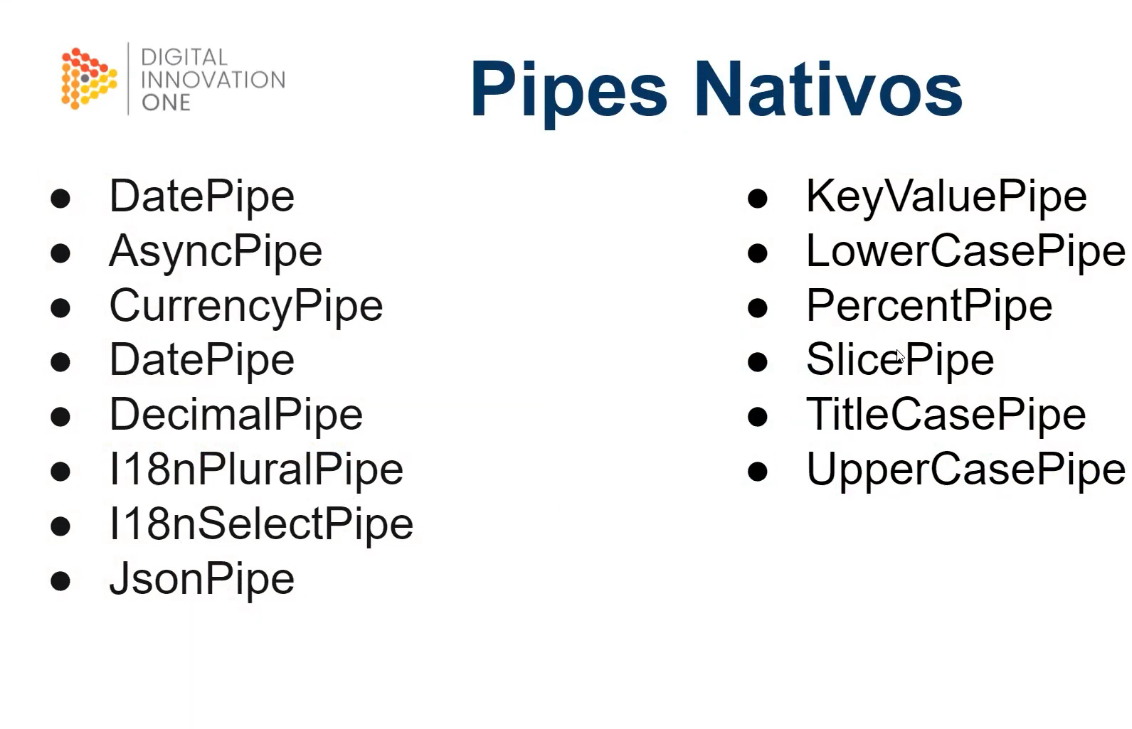
**ngAfterViewInit** - Logo após o conteúdo do próprio componente e de seus filhos ser inicializado



**Pipes**

Funções simples que você pode usar em expressões nos templates para aceitar um valor de entrada e retornar um valor transformado.



**Ex:** 

Pipe puro – Só executa se tiver alguma alteração no valor de entrada

Pipe Impuro – Executa para cada detecção de alteração