**Arquitetura de componentes e a gestão da complexidade no front-end**

**Componente:**

* são os vários elementos que formam o site como um todo
* se refere mais ao visual (front-end)

**Arquitetura de Componentes**

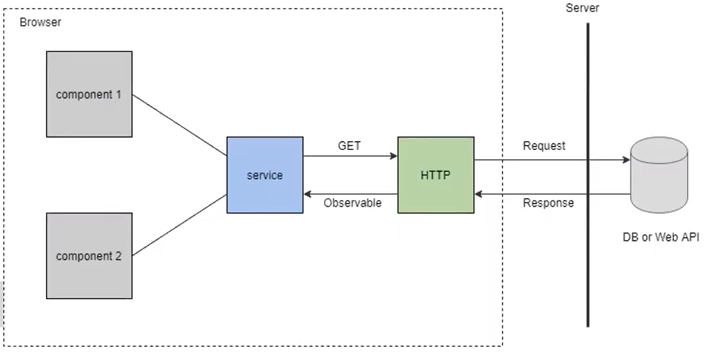
Baseia-se na construção de **componentes independentes, substituíveis** e **modulares** que auxiliem no **gerenciamento de complexidade** e **encorajem a reutilização**

**seus benefícios:**

* escalabilidade
* manutenção
* performance

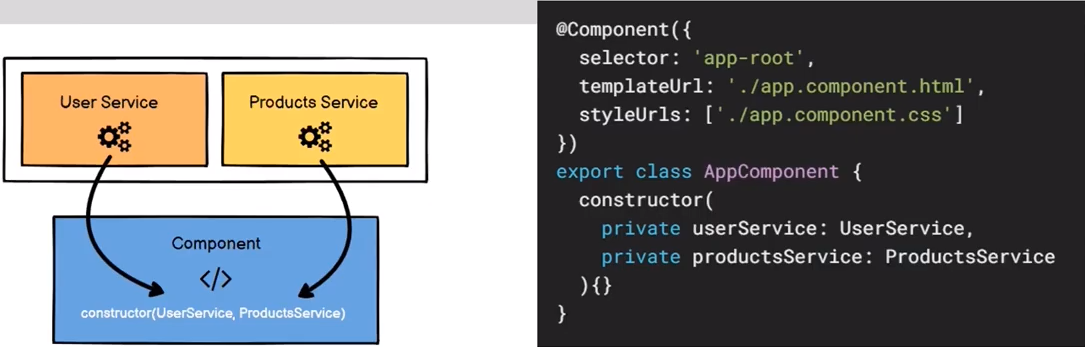
**Serviços:**

* responsáveis por **organizar e compartilhar** lógica de negócios
* **reutilizáveis** entre diferentes componentes de uma aplicação
* **mandatórios** para uma **arquitetura modular** e **reutilizável**



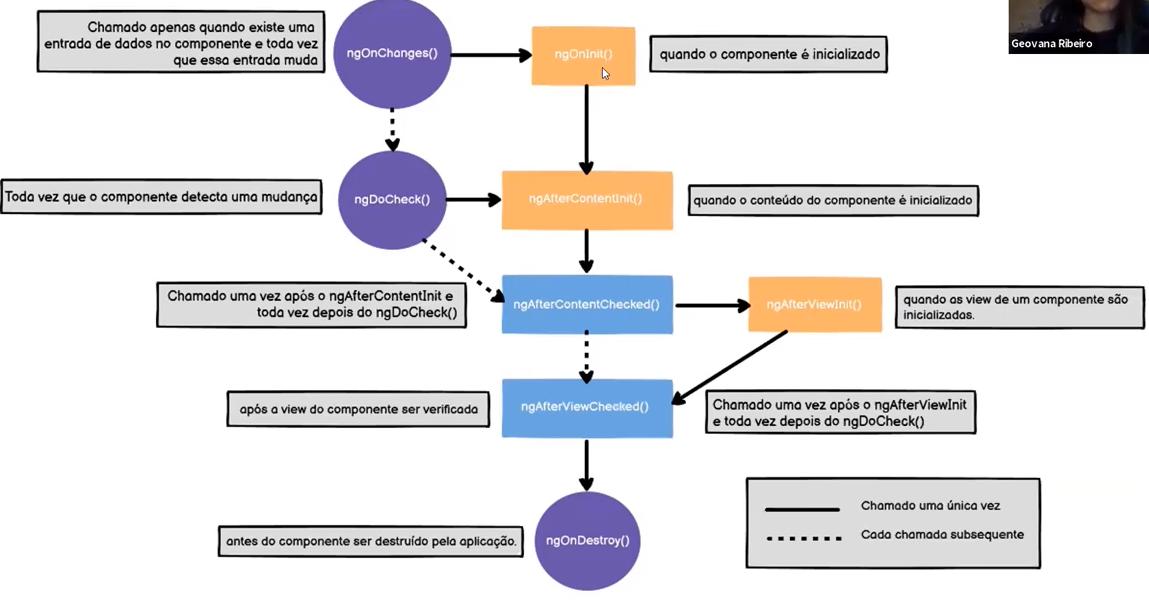
**Injeção de independência:**

* todo serviço é uma **dependência** que precisa ser **instanciada dentro do componente** para ser utilizada pelo menos
* no angular, o componente pede para aplicação quais dependências ele precisa e então as injeta dentro de si



**Ciclo de vida do Componente**

* todo componente possui seu **ciclo de vida** (chamado de **lifecycle hooks**), que começa assim que o Angular o instancializa na aplicação
* através dos ciclos de vida é possível executar diferentes lógicas nos vários estágios de um componente
* **ngOnInit e ngOnDestroy** é um dos principais lifecycles



**Constructor vs ngOnInit**

* **Constructor**
  + deve ser utilizado apenas para inicializar serviços injetados via DI (injeção de independência)
* **ngOnInit**
  + deve ser utilizado para todo tipo de lógica que o componente precisar executar após ter sido criado

**Data Biding**

A forma como associamos informações que estão no componente para o template e vice-versa

* **String Interpolation: {{ valor }}**
  + associa informação do componente para o template (HTML)
* **Property binding: [propriedade]=”valor”**
  + associa informação do componente para propriedades do template (HTML)
* **Event binding: (evento)=”handler”**
  + associa informação do template (HTML) para o componente
* **Two-way data bidint: [(ngModel)]=”propriedade”**
  + associa informação entre ambos, ou seja, mantém ambos atualizados (componente e template (HTML)).

**@Input()**

* **pai => filho:** principal maneira de compartilhar dados do pai para filho

**@Output() e EventEmitter**

* **filho => pai:** principal maneira de compartilhar dados do filho para o pai

**Componentes Apresentacionais**

* parecido com funções puras
* se preocupam apenas com a interface do usuário
* não ficam responsáveis por recuperar dados ou lidar com lógica de negócio
* não causam efeitos colaterais na aplicação
* recebem dados via @Input e emite eventos via @Output

**Componentes Inteligentes**

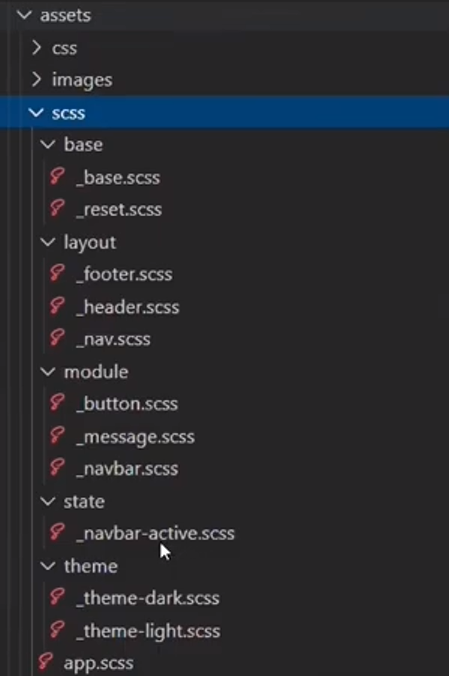
* parecido com funções impuras
* contém toda a lógica de negócio
* são internamente compostos por componentes apresentacionais
* ficam responsáveis por repassar os dados para os componentes apresentacionais apresentarem ao usuário final

**Design Modular**

* divisão da aplicação web em módulos de recursos que representam diferentes funcionalidades de negócios
* **Core Module:** define serviços singleton, componentes de instância única, configuração e exportação de quaisquer módulos de terceiros necessários no módulo principal (App Module)
* **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 reutilizável entre diferentes aplicações
* **Angular elemento:** 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

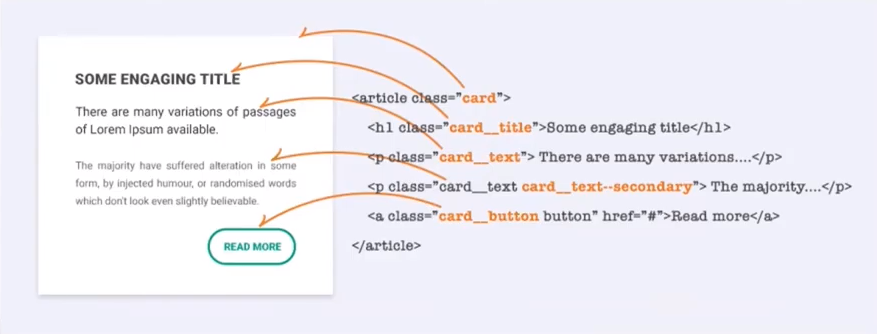
**SMACSS**

* é uma arquitetura modular e escalável para CSS, dividida em 5 camadas, sendo elas:
* **Base:** estilização de seletores e pseudo-classes, além de resets
* **Layout:** principais componentes como cabeçalho, rodapé, entre outros
* **Module:** componentes reutilizáveis, 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. não é necessário utilizar essa pasta



**BEM CSS**

* a sigla BEM significa **block, element, modifier (bloco, elemento e modificador)**, sendo uma metodologia que segue esses conceitos para definir uma nomenclatura de nomes para classes CSS



**OOCSS**

* o OOCSS (**CSS orientado à objeto)** é uma metodologia que identifica um **padrão visual** que pode se repetir no projeto e o agrupa em classes, tornando-os reutilizáveis

