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

avanade

# Estão todos escutando?

avanade

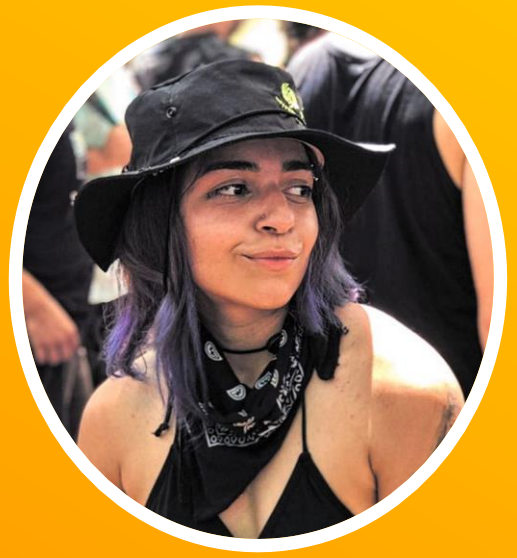
# Agenda

1. >_ whoami
   Um pouquinho sobre Ravena
2. Introdução
   Príncipios da arquitetura de componentes
3. Gestão de complexidade no front-end
   Organizando a estrutura do projeto
4. Indo além | Dúvidas
   Expandindo o conhecimento e bate papo

avanade

# >_ whoami

- 2 anos de Avanade;
- Fez sua primeira página web com 12 anos e nunca mais parou;
- Atualmente alocada num projeto de arquitetura de front-end;
- Escrever, ler, degustar músicas e pular uns muros nas horas vagas <3
- Pode me chamar de Ravena (=



**Geovana Silva Ribeiro**

Anls
Front-End Dev

avanade

# Introdução

Príncipios de arquitetura de componentes

avanade

# O que é um componente?

Header Top

Brasil - Portuguese | Pesquisar | Central de mídia | Contato | COMPARTILHAR:

Header

avanade

Nav

SOLUÇÕES | INDÚSTRIAS | TECNOLOGIAS E CAPACIDADES | ESTUDOS DE CASO | PENSAMENTO | CARREIRAS | SOBRE A AVANADE

Carousel

# Experiência do ambiente de trabalho

Redefina o ambiente de trabalho como um criador de valor comercial sustentável.

Saiba mais

Card

VÍDEO

## Uma ótima experiência no local de trabalho muda tudo

REPRODUZIR O VÍDEO

Card

GUIA PRÁTICO

## O Guia do CIO para experiência no local de trabalho

LEIA MAIS

Card

PESQUISA

## Agregando valor empresarial à experiência do funcionário

LEIA MAIS

## Arquitetura de componentes

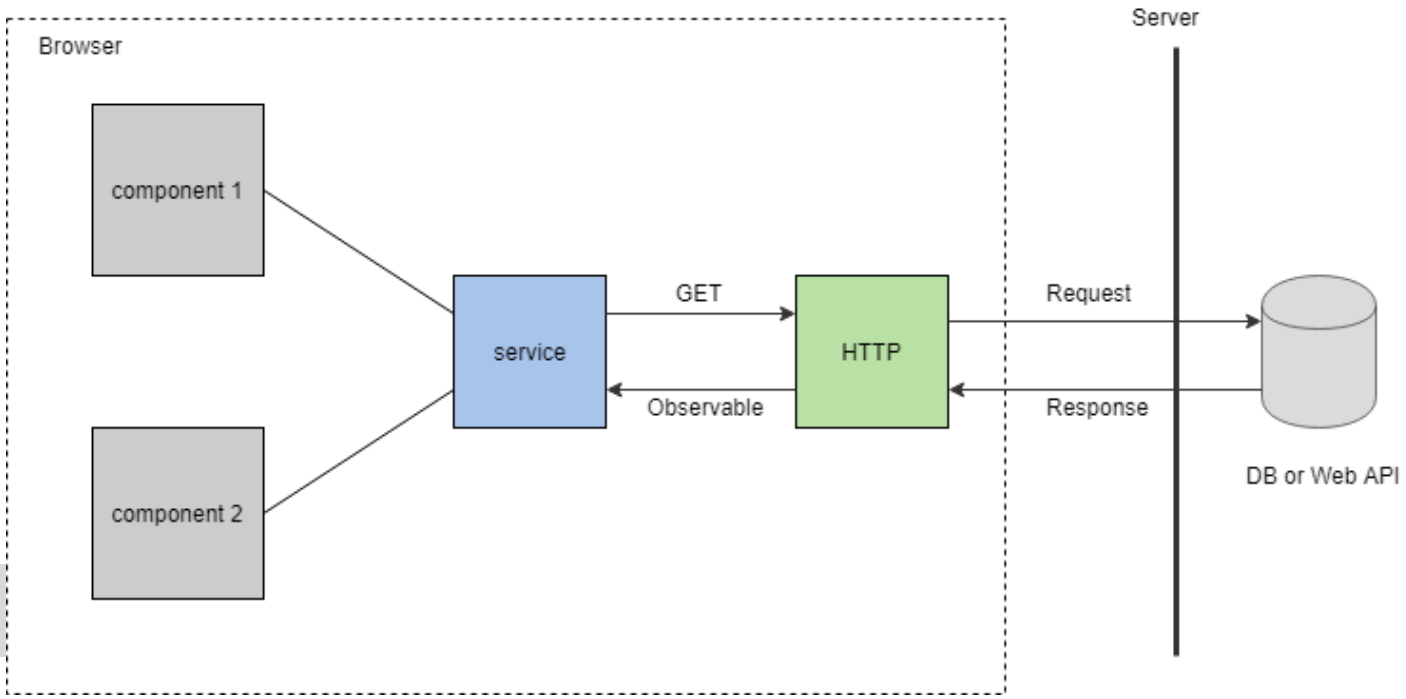
A arquitetura de componentes baseia-se na construção de **componentes independentes**, **substituíveis** e **modulares** que auxiliem no **gerencianciamento da complexidade** e **encorajem** a **reutilização**.

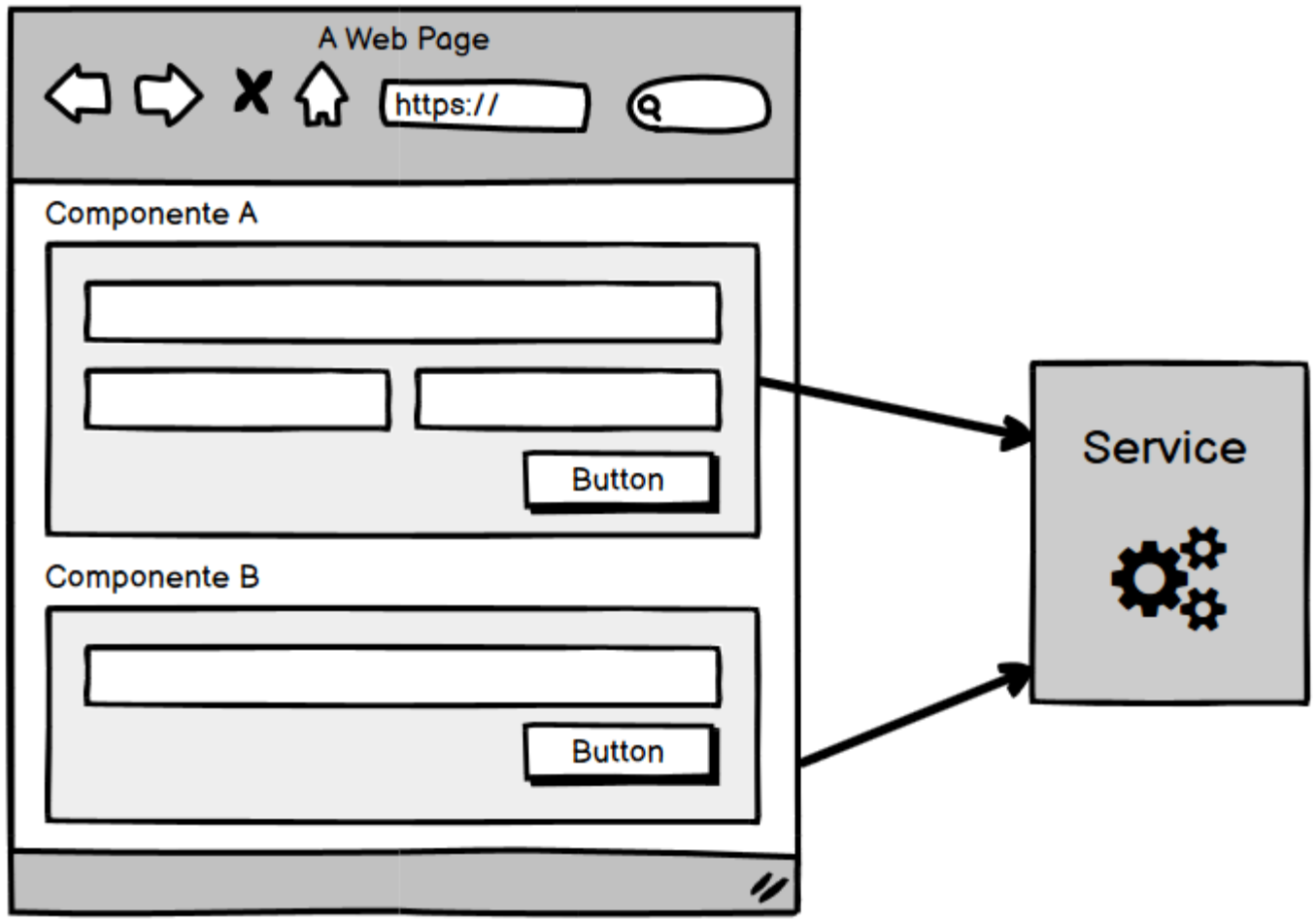
Seus benefícios incluem:

- Escalabilidade
- Manutenção
- Performance

## Serviços

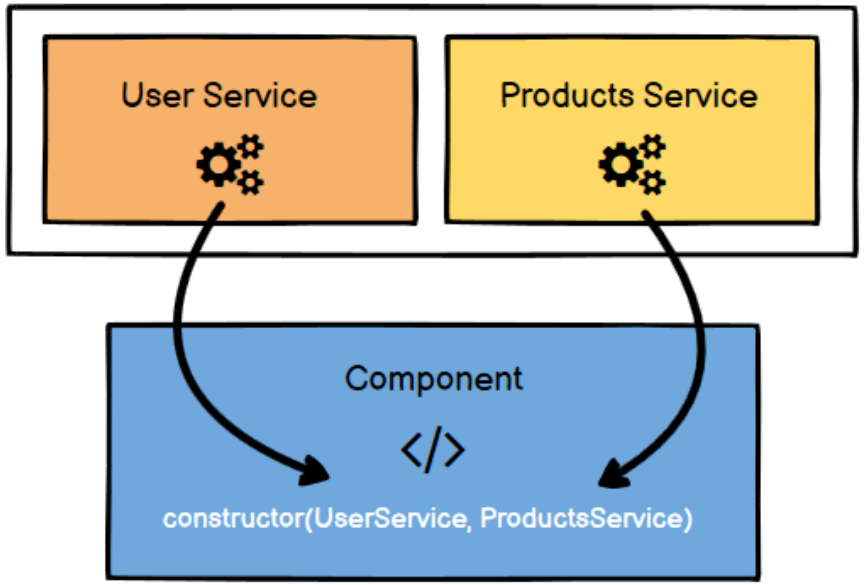
- Responsáveis por **organizar** e **compartilhar** lógica de negócios
- **Reutilizáveis** entre diferentes componentes de um aplicação
- **Mandatorios** para uma **arquitetura modular** e **reutilizável**




A Web Page
https://
Componente A
Button
Componente B
Button
Service

## Injeção de dependência

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

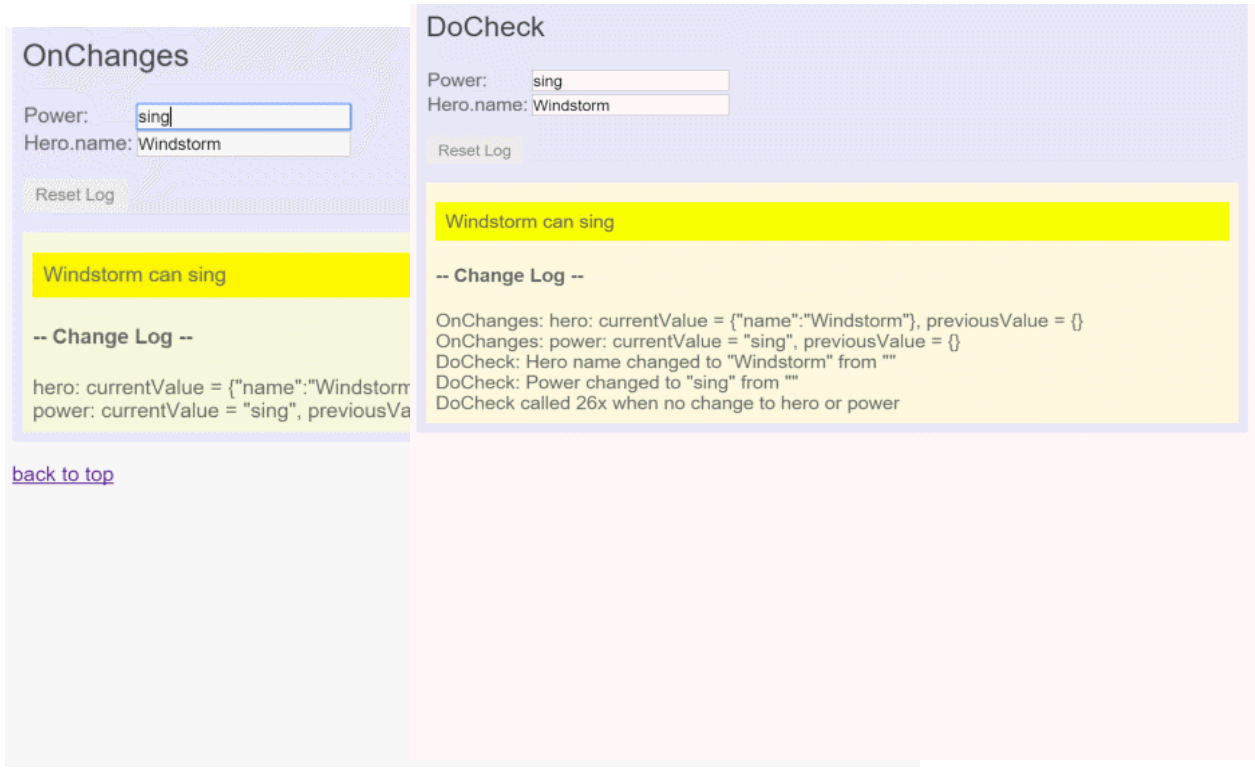



```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(
    private userService: UserService,
    private productsService: ProductsService
  ){}
}
```

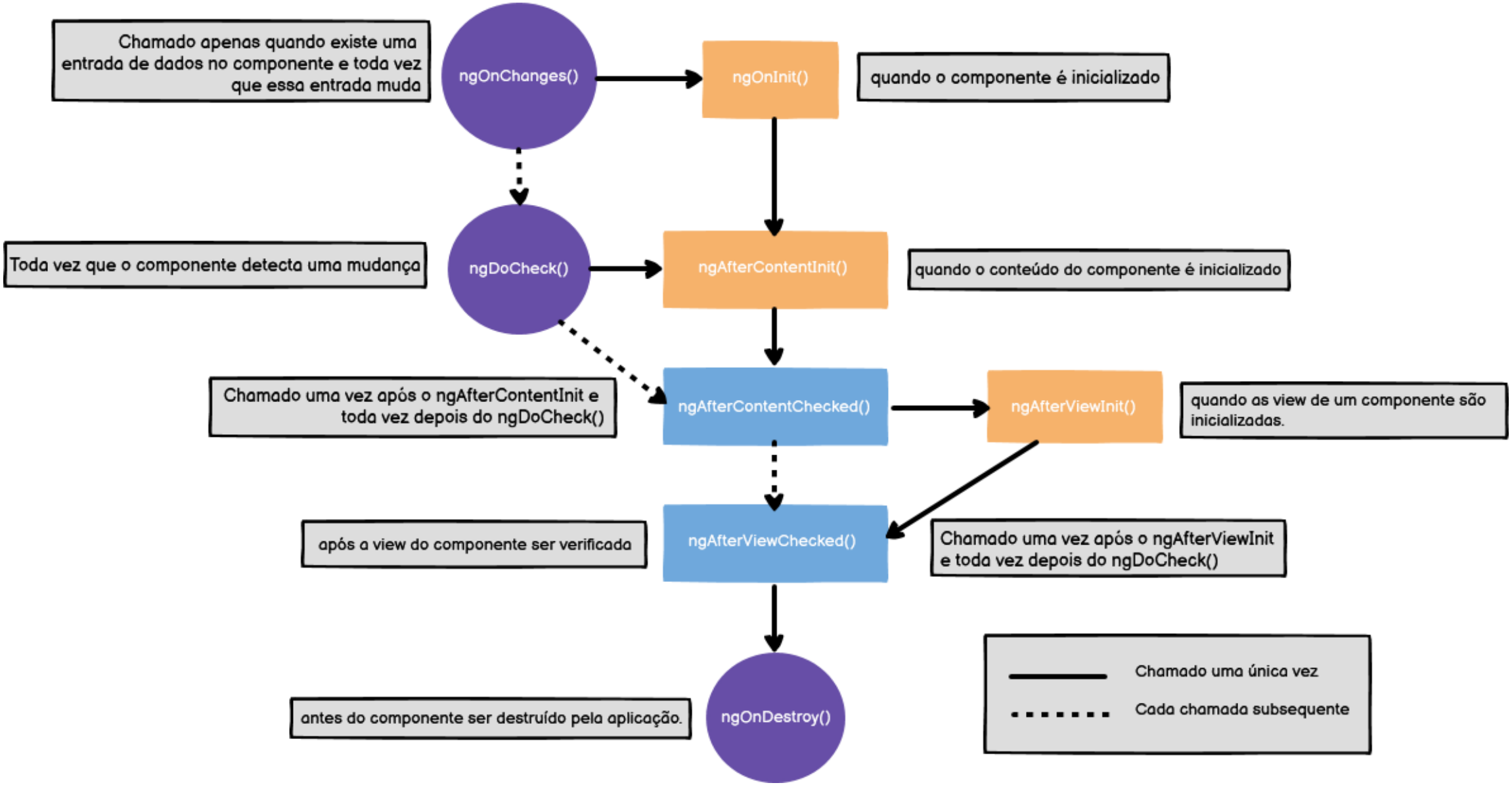
## Ciclo de vida do Componente

Todo componente possui seu **ciclo de vida** (normalmente chamado de **lifecycle hooks**), que começa assim que o Angular o instancializa na aplicação e através deles é possível executar diferentes lógicas nos vários estágios de um componente.

```
@Component({
  ...
})
export class AppComponent implements OnInit, OnDestroy {

  constructor() {}

  ngOnInit() {
    console.log(`Component it's created! \o/`);
  }

  ngOnDestroy() {
    console.log(`Component has destroyed! =(`);
  }
}
```




**Atenção:** Use com sabedoria para não comprometer a performance de sua aplicação!


Chamado apenas quando existe uma entrada de dados no componente e toda vez que essa entrada muda
ngOnChanges()
ngOnInit()
quando o componente é inicializado
Toda vez que o componente detecta uma mudança
ngDoCheck()
ngAfterContentInit()
quando o conteúdo do componente é inicializado
Chamado uma vez após o ngAfterContentInit e toda vez depois do ngDoCheck()
ngAfterContentChecked()
ngAfterViewInit()
quando as view de um componente são inicializadas.
após a view do componente ser verificada
ngAfterViewChecked()
Chamado uma vez após o ngAfterViewInit e toda vez depois do ngDoCheck()
antes do componente ser destruído pela aplicação.
ngOnDestroy()
Chamado uma única vez
Cada chamada subsequente

## Constructor vs ngOnInit

### Constructor

- Deve ser utilizado apenas para inicializar serviços injetados via DI (injeção de dependência)

### ngOnInit

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

## Data Binding

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 Binding: [(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:

Pai:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-pai',
  templateUrl: './pai.component.html',
  styleUrls: ['./pai.component.scss']
})
export class PaiComponent {

    money: number;

    constructor() {
        this.money = 50.00;
    }
}
```

```
<app-filho [earnedMoney]="money"></app-filho>
```

Filho:

```
@Component({
  selector: 'app-filho',
  templateUrl: './filho.component.html',
  styleUrls: ['./filho.component.scss']
})
export class FilhoComponent {

    @Input() earnedMoney: number;

    constructor(private shoppingService: ShoppingService) {}

    ngOnInit(){
        this.shoppingService.buyNewClothes(this.earnedMoney);
    }
}
```

## @Output() e EventEmitter

**Filho → Pai:**

Principal maneira de compartilhar dados do filho para o pai:

Filho:

```
@Component({
  selector: 'app-filho',
  templateUrl: './filho.component.html',
  styleUrls: ['./filho.component.scss']
})
export class FilhoComponent implements OnInit {

  @Input() earnedMoney: number;
  @Output() remainingMoney = new EventEmitter();

  constructor(private shoppingService: ShoppingService) { }

  ngOnInit() {
    this.shoppingService.buyNewClothes(this.earnedMoney)
    .subscribe((change) => {
        this.remainingMoney(change);
    });
  }
}
```

Pai:

```
<app-pai (remainingMoney)="receiveMoney($event)"></app-pai>
```

```
@Component({
  selector: 'app-pai',
  templateUrl: './pai.component.html',
  styleUrls: ['./pai.component.scss']
})
export class PaiComponent {

  money: number;

  constructor() {
    this.money = 50.00;
  }

  receiveMoney(money) {
    // probably buy some cigarettes...
  }
}
```

# Gestão de complexidade no front-end

## Organizando a estrutura do projeto

avanade

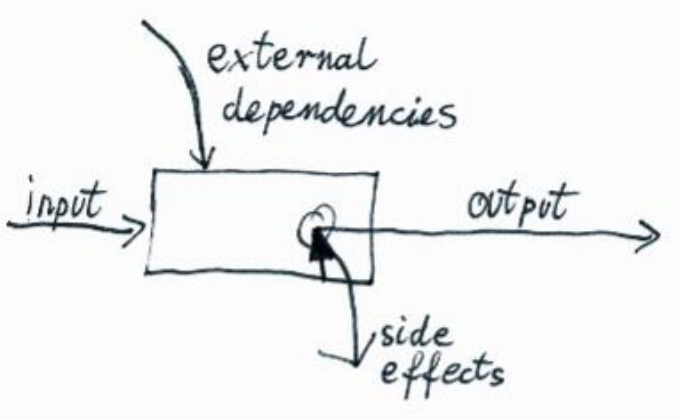
# Componentes inteligentes e apresentacionais

## 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 apresentaram ao usuário final




## Pai

```
@Component({
  selector: 'app-view-todos',
  templateUrl: './view-todos.component.html',
  styleUrls: ['./view-todos.component.css']
})
export class ViewTodosComponent implements OnInit {
  list: Observable;

  constructor(
    private http: HttpClient,
    private todoService: TodoService
  ) {}

    ngOnInit() {
        this.list = this.todoService.getTodos()
        .subscribe((data) => {
            ...
        });
    }
}
```

## Filho:

```
@Component({
  selector: 'app-todos-list',
  templateUrl: './todos-list.component.html',
  styleUrls: ['./todos-list.component.css']
})
export class TodosListComponent implements OnInit {
  @Input() list: any[];
  constructor() { }

  ngOnInit() {}

}
```

```
<app-todos-list [list]="list | async"></app-todos-list>
```

## Design Modular

Dividisão 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.

## Design Modular

Para ter em mente!

Library: possui código que pode ser reutilizável entre diferentes aplicações.

Angular Element: recurso do angular para criar web components, padrão da web para definir novos elementos HTML de uma maneira independente de estrutura e agnóstica de frameworks.

# Indo além

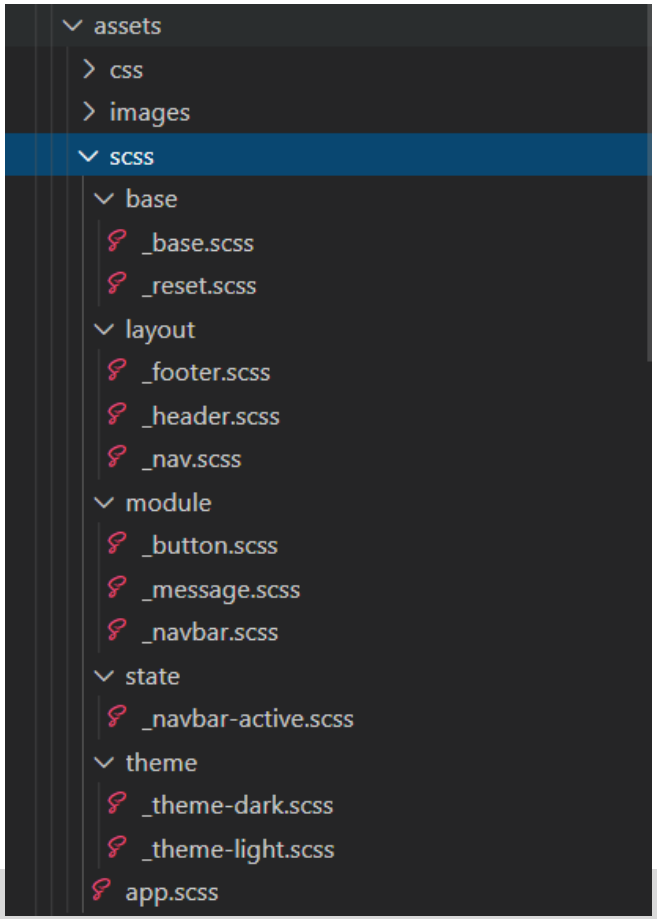
## Expandindo o conhecimento

avanade

## SMACSS

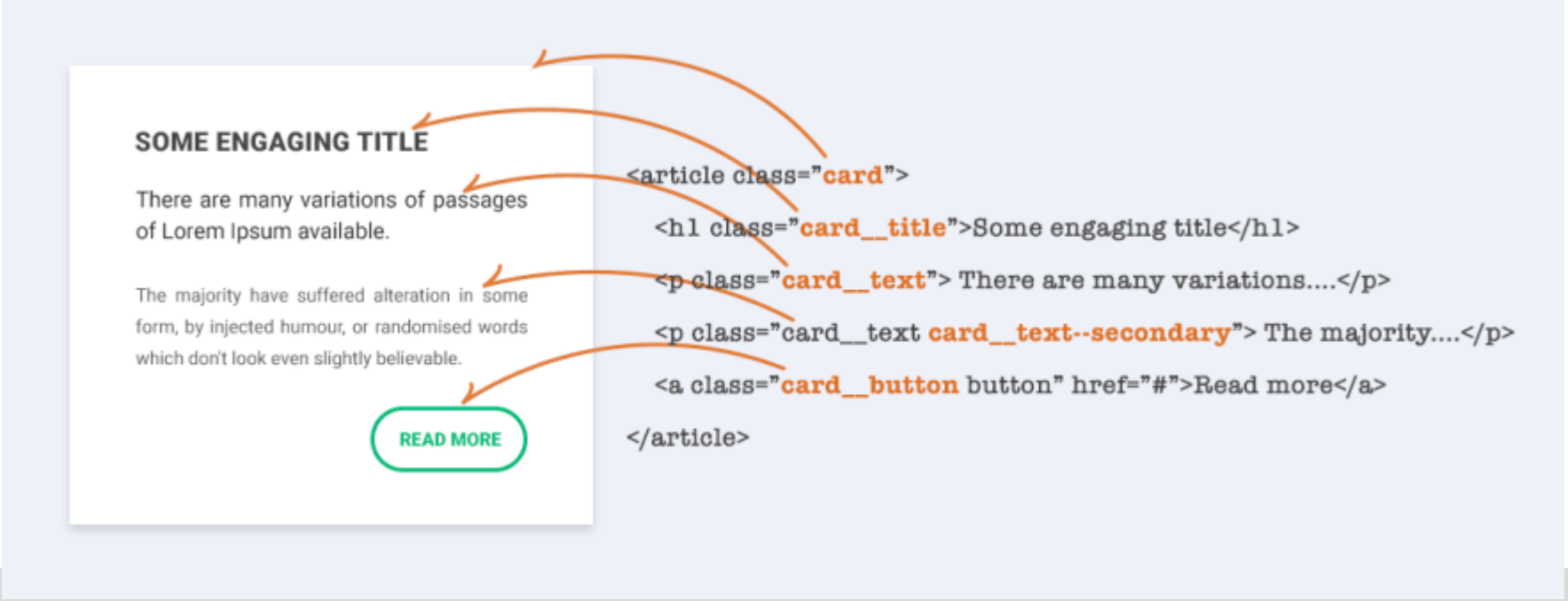
SMACSS é uma arquitetura modular e escalável para CSS, dividida em 5 camadas. Sendo elas:

- 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.



## 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.


```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>
```

- Padrões e Boas Práticas em Angular (Que te ajudarão a escalar)
- Angular 8 — Interação entre Componentes
- Angular Elements – Introdução
- Acessibilidade: O guia para uma web universal
- OCSS, SMACSS, BEM, E DRY CSS: afinal, como escrever CSS?
- Smart Components vs Presentational Components

## Se desafie! (Opcional)

- Aplique a estrutura **SMACSS** num projeto de sua escolha
- **Refatore seu código HTML** utilizando a metodologia **BEM**
- Identifique **padrões visuais** na sua aplicação e as **agrupe em classes**. E **refatore seu projeto** utilizando essas classes.
- Identifique quais componentes do seu projeto são **apresentacionais** e quais podem ser **inteligentes.**
- Utilize os decorators **@Input** e **@Output** para aplicar este conceito.

Não vale pontos =(

# Dúvidas?

avanade

# Muito Obrigada!

avanade