2-Change

Detección del cambio en Angular

```
# 1. Estrategias de detección del cambio

# 2. Técnicas OnPush

# 3. Optimización
```

```
As a: customer,
  I want: to see a shopping cart page
  so that: i can browse the list of products in my basket

As a: customer,
  I want: to pick a product
  so that: I can add units to my basket

As a: customer,
  I want: to remove a product from my basket
  so that: I can take less units

As a: customer,
  I want: to see always counters of my basket
  so that: I can know what I'm buying
```

1. Estrategias de detección del cambio

```
ng g m cart --project=shop --module=app.module.ts --routing --route=cart
```

apps\shop\src\app\app.component.html

```
<nav>
  <a [routerLink]="['/cart']">Basket: {{ 0 }} items</a>
</nav>
```

```
ng g c cart/item-picker --project=shop
ng g c cart/basket-list --project=shop
```

```
ng g s basket --project=shop
```

1.1 Default

```
import { ChangeDetectionStrategy } from '@angular/core';
changeDetection: ChangeDetectionStrategy.Default
```

Con las estrategias por defecto

> Las cosas funcionan como se espera.

Se actualiza la vista con:

1 - Datos asíncronos recibidos desde el API

2 - Procesos en Background

3 - Interacción del usuario

Los cambios se detectan siempre por comparación de valores.

> Aunque demasiadas veces

y por si fuera poco

> Con demasiado coste cada vez

1.2 OnPush

```
import { ChangeDetectionStrategy } from '@angular/core';
changeDetection: ChangeDetectionStrategy.OnPush
```

Al usar la detección OnPush en el contenedor:

> Las llamadas se reducen pero...

siempre hay un pero

> Las datos muestran incoherencias o no se muestran

---

2 Técnicas OnPush

2.1 Async

apps\shop\src\app\cart\cart.component.ts

```
export class CartComponent implements OnInit {
  public products$: Observable<Product[]>;
  public basket: Array<BasketItem> = [];

  constructor(private http: HttpClient, private basketService: BasketService) {}

  ngOnInit() {
    this.products$ = this.http.get<Product[]>('./assets/data/products.json');
  }
}
```

apps\shop\src\app\cart\cart.component.html

```
<div *ngIf="products$ | async as products">
  <ab-shop-item-picker [products]="products"
                       (addItem)="onAddItem($event)"></ab-shop-item-picker>
  <ab-shop-basket-list [basket]="basket"
                       (removeItem)="onRemoveItem($event)"></ab-shop-basket-list>
</div>
```

> Al menos ya tenemos productos, pero ...

otro pero

> Las vistas siguen mostrando incoherencias

1 - Los recepción de productos funciona pues el `pipe async` llama por su cuenta al *cdr*

2 - Pero la lista de productos no se muestra porque no se detectan sus cambios, aunque ocurren.

2.2 Inmutable

¿Qué le ocurre al array?

> Que aunque su contenido cambia

--

> Su referencia es siempre la misma

--

> ¡Tenemos que evitar eso!

--

> El problema es que el componente `CartComponent` con la estrategia *OnPush* ya no detecta cambios internos en un array. Sólo se refresca ante cambios en las referencias. Para forzarlos debemos clonar

> los objetos.

apps\shop\src\app\cart\cart.component.ts

```
  public onAddItem(item: BasketItem) {
    const itemIndex = this.getIndexofItem(item);
    if (itemIndex !== -1) {
      this.basket[itemIndex].units += item.units;
      this.basket = [...this.basket];
    } else {
      // this.basket.push(item);
      this.basket = [...this.basket, item];
    }
    this.onBasketChange();
  }
  public onRemoveItem(item: BasketItem) {
    const itemIndex = this.getIndexofItem(item);
    if (itemIndex !== -1) {
      //this.basket.splice(itemIndex, 1);
      this.basket = this.basket.filter(i => i.product._id !== item.product._id);
    }
    this.onBasketChange();
  }
```

2.3 DetectChanges

¿Qué le ocurre al contador de la barra de navegación? ¿Porqué al borrar en *background* la lista no lo refleja?

> Que aunque su contenido cambia.

--

> La detección no se lanza porque nada lo provoca

--

> No hay evento de usuario, ni `@Output()`

--

> El problema es que un componente con la estrategia *OnPush* sigue sin detectar cambios asíncronos. O usamos el `pipe async` o en situaciones límite, tendremos que lanzar el proceso de detección de cambios de forma manual.

apps\shop\src\app\cart\cart.component.ts

```
  private autoBackGroundRemover() {
    setTimeout(() => {
```

```
        this.onRemoveItem(this.basket[0]);
        this.cdr.detectChanges();
      }, 1000);
    }
```

```
import { ChangeDetectorRef } from '@angular/core';

export class AppComponent implements OnInit {
  public title = 'shop';
  public basketUnits = 0;
  public basket = [];
  constructor(
    private basketService: BasketService,
    private cdr: ChangeDetectorRef
  ) {}
  ngOnInit(): void {
    this.basketService.units$.subscribe({
      next: units => {
        this.basketUnits = units;
        this.cdr.detectChanges();
      }
    });
    this.basketService.basket$.subscribe({
      next: basket => {
        this.basket = basket;
        this.cdr.detectChanges();
      }
    });
  }
```

Nadie mejor que quien lo desarrolla para saber cuando algo cambia.

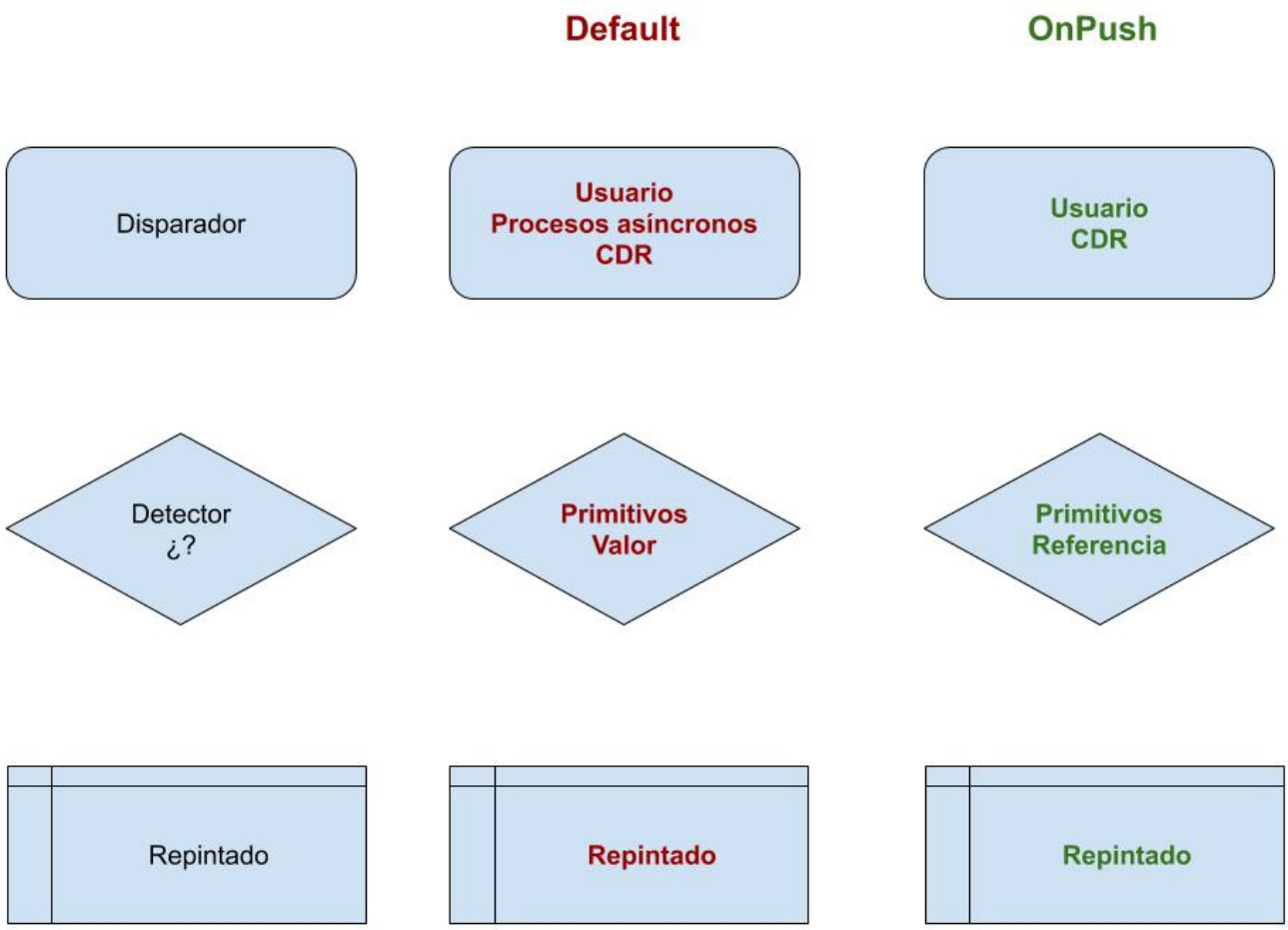
Pero ojo...

> Forzar la detección de cambios, no implica detección

> Hay que usarlo en combinación con el clonado.

3 Optimización

Change Detection



3.1 OnPush es más ligero

- Se lanza menos veces
- Sólo comprueba referencias, no valores

3.2 Async, CDR y clone detectan los cambios

- `Async`**:** para que las respuestas desde observables sean limpias
- ***CDR*****:** cuando el cambio venga de procesos asíncronos pero no observables
- ***Clonado*****:** para que se detecten cambios en las referencias

---

> **Blog de apoyo:** Detección del cambio en Angular
>
> > By Alberto Basalo

> Next:

Componentes dinámicos, directivas y pipes

Plantillas de contenido dinámico

Atributos custom con Directivas

Funciones de transformación con Pipes