



# ► 1. Componentes y comunicación asíncrona

## 🔍 Objetivos

- Estructurar los componentes de una aplicación de forma sistemática
- Delegar el problema asíncrono a los controladores de nivel superior
- Aplicar el patrón container/presenter con detección de cambios mejorada

## 📚 Referencia

### Presentational components with Angular - Angular inDepth

Presentational components are literally the user interface of our Angular application. They serve two purposes: Present application state to the user Change application state triggered by user interaction To communicate with

🔗 <https://indepth.dev/posts/1066/presentational-components-with-angular>



### Angular Change Detection and the OnPush Strategy

You have started using Angular for all of your favorite projects. You know what Angular has to offer, and how you can leverage it to build amazing web apps. But, there are certain things about Angular, and knowing them can

🔗 <https://www.toptal.com/angular/angular-change-detection>



### 🚀 A Comprehensive Guide to Angular onPush Change Detection Strategy

By default Angular uses the `ChangeDetectionStrategy.Default` change detection strategy. The default strategy doesn't assume anything about the application, therefore every time something changes in our application, as a result of various

🔗 <https://netbasal.com/a-comprehensive-guide-to-angular-onpush-change-detection-strategy-5bac493074a4>



### Ivy preview

⚠ <https://alexzuza.github.io/angular-9-ivy-change-detection-preview/>

### How to turn on Angular Debug Tools to measure change detection cycle time

To turn this feature on, just make a small change in your `main.ts` file as below  
`platformBrowserDynamic().bootstrapModule(AppModule).then((module) => { const appRef = module.injector.get(ApplicationRef); const appComponent =`

🔗 <https://phamhuuhien.medium.com/how-to-turn-on-angular-debug-tools-to-measure-change-detection-cycle-time-dc6d87405302>



## Atomic Design

Hey there! I wrote a book called Atomic Design that dives into this topic in more detail, which you can buy as an ebook.

👉 <https://bradfrost.com/blog/post/atomic-web-design/>



## ▶ 0 - Dos problemas a evitar: mega componentes y espera de datos asíncronos

brick icon Los componentes son los bloques de construcción principales de Angular. Cuanto mejores sean, mejor será nuestra aplicación.

### cloud and rain icon Riesgo

- Grandes componentes con mucho HTML en sus plantillas y mucho TypeScript en sus controladores
- Falta de datos iniciales que obliguen a múltiples `*ngIf` o inicializaciones tediosas

### rainbow icon Solución

- Dividir los componentes en otros más pequeños
- Aislar la responsabilidad de presentación de datos los asíncronos

## ▶ 1 - Tipos de componentes y responsabilidades

### target icon Separamos responsabilidades para :

- Evitar grandes componentes
- Fomentar la reusabilidad
- Aislarn cambios y consecuencias
- Facilidad de pruebas específicas

### person icon Smart, Parent or Container

bar chart icon Se encargan de obtener, manipular y guardar los datos de la aplicación



Envían información en propiedades `[props]="data"`



Se suscriben a cambios en eventos `(event)="method()"`

### Incluyen dependencias con servicios

### Esperan por los datos asíncronos

#### ▼ Características

- Al contener o acceder a la lógica de negocio se crean en módulos funcionales
- Normalmente asociados a rutas cargadas en modo *lazy*.
- Pueden residir en su propia librería
- Pueden ser nombrados con el tipos específicos *Page*, *Widget*



### Dumb, Child, Presenter

| Se encargan de presentar la información e interactuar con el usuario



Reciben datos vía `@Input() props`



Emiten eventos vía `@Output() event`

### No contienen dependencias con servicios

### Reciben datos ya preparadas, sin asincronismo

#### ▼ Características

- No suelen requerir constructor ni *ngOnInit*
- Publican sus propias interfaces o modelos de datos.
- Si resuelven un problema concreto de negocio se declaran sin exportarse
- Si son genéricos se declaran y exportan en módulos o librerías de infraestructura.

El patrón container/presenter resuelve ambos problemas.

Las plantillas se reparten en numerosos componentes presentacionales

El componente container se ocupa de obtener los datos e invocar a los presentacionales cuando estos llegan



## 2 - Estrategias de detección de cambios

Angular destaca por su capacidad de repintar las vistas y reflejar el estado actual del modelo de datos.. Pero, ¿cómo lo hace? ¿a qué coste?



### Default

Detección automática.



Transparente para el **programador**



Su **ejecución** se lanza ante eventos de usuario, threads asíncronos y ajax ⇒ demasiadas veces



**Evaluación** por valor, comparando cada propiedad de cada objeto u array ⇒ costosa



### OnPush

Detección semiautomática.



Su **ejecución** se lanza sólo ante eventos de usuario ⇒ menos veces



**Evaluación** por referencia, algo es igual si apunta a la misma dirección ⇒ ligera



El **programador** debe ser consciente de cuándo y porqué se lanza

Se recomienda configurarlo por defecto para todo el workspace en el `angular.json`

```
"schematics": {  
  "@schematics/angular:component": {  
    "change-detection": "OnPush"  
  }  
}
```



💡 Como regla general es mejor exponer objetos que tipos primitivos (**avoid primitive obsession**).

🧙 Es auto mágico si el componente **contenedor** usar el `pipe async` antes de enviar los `[inputs]` a los **presentadores**.

僬 El control por referencia puede implicar usar alguna técnica de **inmutabilidad** y enviar **clones** para cada cambio.

🐟 Aunque en http no es necesario porque siempre que recibimos datos estos son frescos...

•• Como último recurso queda usar el `ChangeDetectorRef` para marcar el componente para su detección.

## ▶ 3 - Atomic Design

Sistema de diseño jerárquico basado en la composición del mundo natural



### Domain Components



#### Componentes de ruta (PAGES) o de funcionalidad completa (WIDGETS)

Son los 🧑únicos componentes inteligentes con responsabilidad de llamada a **lógica de negocio** y accesos a datos.

```
# 📁 Módulo (en su propia librería) ya creado durante la estructura general del proyecto
nx g library home --importPath=@ab/home --prefix=ab --routing --lazy
  --parentModule='apps\www\src\app\core\core-routing.module.ts' --tags='domain, page'
# 📄 Componente página
nx g c home --project=domain-home --flat --skipTests=false --skipSelector --type=Page
# 🏠 Servicio de acceso a datos
nx g s home --project=domain-home --flat
```



🚫 No deben contener HTML standard en su plantilla

🧪 Deberían tener pruebas unitarias de su clase controladora

📄 Se sugiere usar el sufijo `Page` en lugar de `Component`

🔍 Sin selector css (los invoca el router, no otro programador)

✨ Otro uso es para **funcionalidades sin página** propia (`search-box`, `shopping-cart`, `user-anonymous`, ) prueba con el sufijo `Feature` o `Widget`



## Componentes de negocio (ORGANISMS)

Son **componentes presentacionales** con responsabilidad de presentación y potencialmente transformación y validación de entrada.

```
# Se crean de forma provada en el módulo de la ruta que los consume  
# Presentacional de categorías  
ng g c categories --project=domain-home  
# Suelen ser formularios, listados...
```

- ⚠️ ↴ Reciben datos síncronos vía `@Input` y emiten eventos mediante `@Output`.
- 💡 Pueden necesitar pruebas unitarias de código para posibles validaciones o transformaciones.
- 💡 Si la lógica se complica debe llevarse a servicios especializados de validación y transformación



## Shared Componentes

```
nx g library ui --directory=shared --importPath=@ab/ui --prefix=ab-ui --tags='shared, ui'
```



## Plantillas de páginas (TEMPLATES)

Componentes con **placeholders** para incrustar otros componentes aportando consistencia y reutilizando *layouts*

- ⚠️ 📦 Hacen uso extenso de la directiva `ng-content`.
- 💡 No suelen necesitar pruebas específicas

```
# Algunos ejemplos típicos...  
nx g c templates/box --project=shared-ui --export=true --type=Template  
nx g c templates/card --project=shared-ui --export=true --type=Template  
nx g c templates/modal --project=shared-ui --export=true --type=Template  
nx g c templates/page --project=shared-ui --export=true --type=Template  
nx g c templates/panel --project=shared-ui --export=true --type=Template  
nx g c templates/section --project=shared-ui --export=true --type=Template
```



## Componentes de bloques (MOLECULES)

Similares a los organismos, pero más  **abstractos**.

 No están adaptados a un problema de negocio concreto; resuelven **problemas comunes** a cualquier aplicación.

```
# Son exportados desde los módulos compartidos en dónde se crean
nx g c components/breadcrumb --project=shared-ui --export=true
nx g c components/menu --project=shared-ui --export=true
nx g c components/message --project=shared-ui --export=true
nx g c components/notification --project=shared-ui --export=true
nx g c components/tabs --project=shared-ui --export=true
```

## Componentes de elementos (ATOMS)

Nivel mínimo de componentización. Encapsulan elementos de HTML estándar o de algún framework CSS. De esta forma hacen que la aplicación sea menos dependiente de terceros.

```
# No siempre son exportados desde los módulos compartidos en dónde se crean
# Pues algunos son poco útiles fuera de un bloque mayor
nx g c components/header --project=shared-ui --export=true
```



Aportan homogeneidad UX reescribiendo elementos básicos del HTML. (Title, Link...)



TODOS (átomos, moléculas, organismos y plantillas) presentan al usuario estructuras de datos y las ofrecen al programador en forma de modelos (`interface`, `type`)

## Práctica

### Shared UI Lib

```
// ✨ Header atom component
export interface Header {
  title: string;
  subtitle: string;
  heroClass: string;
}
<header class="hero mt-4 mb-4 {{header.heroClass}} ">
  <div class="hero-body">
    <p class="title"> {{ header.title }} </p>
    <p class="subtitle"> {{ header.subtitle }} </p>
```

```

        </div>
    </header>
    export class HeaderComponent {
        @Input() header: Header = { title: '', subtitle: '', heroClass: '' };
        constructor() {}
    }

    // 📄 Card template component
    export interface Card {
        title: string;
        description?: string;
        link?: string;
        href?: string;
    }
<article class="card mb-4">
    <header class="card-header">
        <p class="card-header-title">
            <a *ngIf="card.link" [routerLink]=[card.link]>{{ card.title }}</a>
            <a *ngIf="card.href" [href]=card.href>{{ card.title }}</a>
            <span *ngIf="!card.link && !card.href">{{ card.title }}</span>
        </p>
    </header>
    <main class="card-content">
        <div *ngIf="!card.description" class="content">
            {{ card.description }}
        </div>
        <div *ngIf="!card.description" class="content">
            <ng-content select=".content">
                </ng-content>
            </div>
        </main>
        <footer class="card-footer">
            <ng-content select=".footer-content">
                </ng-content>
            </footer>
    </article>
    export class CardTemplate {
        @Input() card: Card = { title: '', description: '' };
        constructor() {}
    }

```

## 🗄️ Domain 📜 Home Lib

```

    // 🌱 CategoryList organism component
    export interface Category {
        id: string;
        name: string;
        description: string;
    }
<section class="section mt-4 mb-4">
    <ab-ui-header [header]=""></ab-ui-header>
    <section class="mt-4 mb-4 columns is-multiline">
        <ab-ui-card *ngFor="let category of categories"
            [card]="getCardFrom(category)"
            class="column is-one-quarter-desktop is-half-tablet">
        </ab-ui-card>
    </section>
</section>
    export class CategoryList{
        @Input() categories: Category[] = [];
        header = {
            heroClass: 'is-danger',
            title: 'Resources for Angular developers',
            subtitle: 'Coming soon...',
        };
        getCardFrom(category: Category) {

```

```

        return {
          title: category.name,
          description: category.description,
          link: '/category/' + category.id,
        };
      }
    }

// 📄 Home page component
<main>
  <ab-ui-header [header]="header"></ab-ui-header>
  <ab-category-list
    *ngIf="categories$ | async as categories"
    [categories]="categories">
  </ab-category-list>
</main>
export class HomePage {
  categories$ = this.service.getCategories$();
  header = {
    heroClass: 'is-primary',
    title: 'The home of the Angular Builders',
    subtitle: 'A site to help you build great applications with Angular',
  };
  constructor(private service: HomeService) {}
}

// 🚀 Home Service
@Injectable({
  providedIn: 'root',
})
export class HomeService {
  private readonly categoriesUrl = `https://api-angular-builders.herokuapp.com/v1/categories`;
  constructor(
    private http: HttpClient
  ) {}
  getCategories$() {
    return this.http
      .get<{ data: Category[] }>(this.categoriesUrl)
      .pipe(map((result) => result.data));
  }
}

```

## ☰ Domain 📚 Category Lib

```

// 🌱 Resource list organism component
export interface Category {
  id: string;
  name: string;
  description: string;
}
export interface Resource {
  id: string;
  name: string;
  description: string;
}
<section class="section mt-4 mb-4">
  <ab-ui-header [header]="getHeader()"></ab-ui-header>
  <section class="columns is-multiline">
    <ab-ui-card *ngFor="let resource of resources"
      [card]="getCardFrom(resource)"
      class="column is-one-quarter-desktop is-half-tablet">
    </ab-ui-card>
  </section>
</section>

export class ResourceList {

```

```

@Input() categoryName = '';
@Input() resources: Resource[] = [];

private header = {
  heroClass: 'is-warning',
  title: `...`,
  subtitle: ' No resources yet ',
};

getHeader() {
  const header = { ...this.header };
  if (this.categoryName) {
    header.title = `List of resources for ${this.categoryName}`;
  }
  if (this.resources.length) {
    header.subtitle = `Found ${this.resources.length} resources`;
  }
  return header;
}
getCardFrom(resource: Resource) {
  return {
    title: resource.name,
    description: resource.description,
    link: '/resource/' + resource.id,
  };
}
}

// Category page component
<main>
  <ab-ui-header *ngIf="header$ | async as header" [header]="header"></ab-ui-header>
  <ab-resource-list *ngIf="categoryResources$ | async as categoryResources"
    [resources]="categoryResources.resources"
    [categoryName]="categoryResources.category.name">
  </ab-resource-list>
</main>
export class CategoryPage implements OnInit {
  private header = {
    heroClass: 'is-primary',
    title: 'Category',
    subtitle: 'loading...',
  };

  header$ = new BehaviorSubject<Header>(this.header);
  categoryResources$!: Observable<{
    category: Category;
    resources: Resource[];
  }>;

  constructor(
    private route: ActivatedRoute,
    private service: CategoryService
  ) {}

  ngOnInit(): void {
    const categoryId = this.route.snapshot.params.id;
    this.header$.next({ ...this.header, title: categoryId });
    const category$ = this.service.getCategoryById$(categoryId);
    const resources$ = this.service.getResourcesByCategoryId$(categoryId);
    this.categoryResources$ = forkJoin({
      category: category$,
      resources: resources$,
    }).pipe(
      tap(result) =>
        this.header$.next({
          ...this.header,
          title: result.category.name,
          subtitle: result.category.description,
        })
    )
  }
}

```

```

        )
    );
}

// 🎉 Category Service
export class CategoryService {
    private readonly categoriesUrl = `https://api-angularbuilders.herokuapp.com/v1/categories`;
    constructor(private http: HttpClient) {}
    getCategoryById$(categoryId: string) {
        return this.http
            .get<{ data: Category }>(`${this.categoriesUrl}/${categoryId}`)
            .pipe(map((result) => result.data));
    }
    getResourcesByCategoryId$(categoryId: string) {
        return this.http
            .get<{ data: Resource[] }>(
                `${this.categoriesUrl}/${categoryId}/resources`
            )
            .pipe(map((result) => result.data));
    }
}

```

## 💻 Apps 💻 WWW

```

<nav class="navbar" role="navigation" aria-label="main navigation">
    <div class="navbar-brand">
        <a class="navbar-item" href="/"> <strong class="ml-4 is-size-2">{{ title }}</strong> </a>
    </div>
</nav>
<main>
    <router-outlet></router-outlet>
</main>
<footer class="footer">
    <div class="content has-text-centered">
        <p> by <a href="https://twitter.com/albertobasalo">By Alberto Basalo</a>.</p>
    </div>
</footer>
export class AppComponent {
    title = 'Angular.Builders'
}

```

## 💡 Alberto Basalo