



# Angular Advanced Component Design Patterns



Peter Kassenaar  
[info@kassenaar.com](mailto:info@kassenaar.com)

# Angular Design Patterns

*1. Content projection*

*2. Smart components / View  
Components*

# But wait! There is more (like, *a lot* more)

**Dev Academy**  
Previously Angular Academy

YOUTUBE   WS-TUESDAY   [COURSES](#)

DevAcademy Slack Community Learn with hundreds of devs worldwide! [JOIN NOW](#)

Join **+6,119** subscribers  
that receive latest  
knowledge and tips  
[Subscribe](#)

  
**Bartosz Pietrucha**  
Dev Academy founder,  
Master of Science in CS,  
Software consultant  
[@bartosz\\_io](#)

## Angular Architecture Patterns and Best Practices (that help to scale)

 Bartosz Pietrucha • 2 Jul 2019 • [#Angular](#) [#Architecture](#)



Share

[Twitter](#)  
[Facebook](#)  
[Reddit](#)

Building scalable software is a challenging task. When we think about scalability in front-end applications, we can think of increasing complexity, more and more business rules, a growing amount of data loaded into the application and large teams often distributed around the world. In order to deal with mentioned factors to maintain a high quality of delivery and prevent technical debt, robust and well-grounded architecture is necessary. Angular itself is a quite opinionated framework, forcing developers to do things *the proper way*, yet there are a lot of places where things can go wrong. In this article, I will present

<https://dev-academy.com/angular-architecture-best-practices/>

# 1. What is Content Projection

- Re-use of content *inside* of components
- Often used when you create components *to be used* by others
- Simple use:
  - *Attribute binding* to pass data into components `[prop]="data"`
  - *Event binding* to get data out of components  
`(event)="handler ()"`


# Examples ../130-content-projection

Content Projection

localhost:4200

☆ 🔍 📄 📱 📺 ⚙️ 🌐

Component: Card 1




Relax

Get some rest while visiting beautiful landscapes and enjoying the mountains and the sea.

No content projection used in this component

LEARN MORE

Component: Card 2




Free life

Escape the city, avoid rush hour, no more deadlines. Get away and be free.

The title of this card is projected

LEARN MORE

Component: Card 3




Enjoy nature

Enjoy the outdoor life. Fly like an eagle and get rid of your anger, worries and stress.

The title of this card and the text contents are projected. We are using class projection here.


LEARN MORE

Component: Card 4



Hike your path

Component: Card 5



**Goal:**

# Creating reusable components

**Component Composition** instead of  
Component Inheritance

# 1. Simple content projection:

```
<my-component>  
  I want to re-use this text  
</my-component>
```

In the parent component:

```
<app-card2>  
  Free life  
</app-card2>
```

In the child component:

```
<h4 class="card-title">  
  <ng-content></ng-content>  
</h4>
```

## 2. Content projection based on CSS-class

In the parent component:

```
<app-card3>  
<p class="card-text">Enjoy...</p>  
</app-card3>
```

In the child component:

```
<ng-content  
  select=".card-text">  
</ng-content>
```



### 3. Content projection based element selector

In the parent component:

```
<app-card4>
  
  ...
</app-card4>
```

In the child component:

```
<ng-content
  select="img.card-photo">
</ng-content>
```

## 3a. Alternative : Content projection based on attribute

In the parent component:

```
<app-cardXYZ>  
  <div myAttribute>...</div>  
  ...  
</app-card4>
```

In the child component:

```
<ng-content  
  select="[myAttribute]">  
</ng-content>
```

## 4. Based on custom component

- Create an extra component (here: `<app-newsletter>`)
- Use content projection based on element selector
- Extra: submit events from nested component back to parent

```
<div>
  <label>
    <input type="checkbox" (change)="onChange($event.target.checked)">
    Subscribe to newsletter!
  </label>
</div>
```

```
export class NewsletterComponent {

  @Output() checked: EventEmitter<boolean> = new EventEmitter<boolean>();

  onChange(value: boolean) {
    this.checked.emit(value)
  }
}
```

In parent Component:

```
<app-card5>  
  ...  
  <app-newsletter (checked)="onChecked($event)">  
  </app-newsletter>  
  ...  
</app-card5>
```

...handle event on parent component class

```
<!-- nested component, projected from parent component -->  
<ng-content select="app-newsletter"></ng-content>
```

# Verdict

- Use Content Projection, to present **VIEW** information inside the component
  - Again: *mostly* used on redistributable components
- Use `@Input()` and `@Output()` decorators for **logic** of the component

# Previously: showing default values in ng-content blocks

- Showing a default value is NOT out-of-the-box behavior
- We have to create it ourselves
- (In modern Angular it IS possible: `<ng-content>default value</ng-content>`)

Content projection with default values. A bit cumbersome, but it works.

```
<button class="my-button">
  <ng-content select="img"></ng-content>
  <!-- wrap the ng-content block with a local default template variable -->
  <span #ref><ng-content></ng-content></span>
  <!-- Check if there is content available,
        otherwise show default text-->
  <ng-container *ngIf="!ref.hasChildNodes()">
    Button text here
  </ng-container>
</button>
```

You'll call/use the button as follows:

```
<!-- first instance, WITH text-->
<app-my-button>
  
  Login
</app-my-button>
<!-- Second instance, WITHOUT text (so showing the default value)-->
<app-my-button>
  
</app-my-button>
```

# More info on Modern `<ng-content>`

[← Components](#)

[Anatomy of components](#)

[Selectors](#)

[Styling](#)

[Accepting data with input properties](#)

[Custom events with outputs](#)

[Content projection with ng-content](#)

[Host elements](#)

[Lifecycle](#)

[Referencing component children with queries](#)

[Using DOM APIs](#)

[Inheritance](#)

[Programmatically rendering components](#)

[Advanced configuration](#)

[Custom Elements](#)

In-depth Guides > Components

Content projection with ng-content

★ **TIP:** This guide assumes you've already read the [Essentials Guide](#). Read that first if you're new to Angular.

You often need to create components that act as containers for different types of content. For example, you may want to create a custom card component:

```
@Component({
  selector: 'custom-card',
  template: '<div class="card-shadow"> <!-- card content goes here --> </div>',
})
export class CustomCard { /* ... */ }
```

You can use the `<ng-content>` element as a placeholder to mark where content should go:

```
@Component({
  selector: 'custom-card',
  template: '<div class="card-shadow"> <ng-content></ng-content> </div>',
})
```

On this page

[Multiple content placeholders](#)

[Fallback content](#)

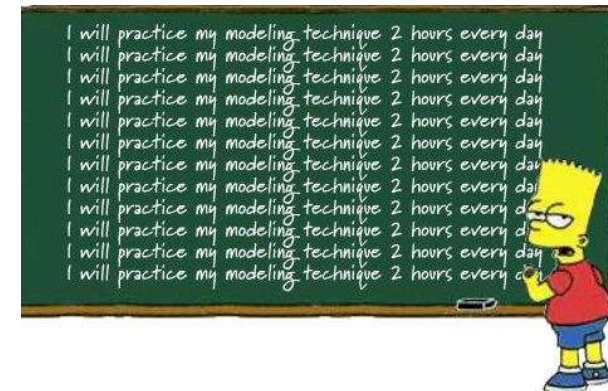
[Aliasing content for projection](#)

[↑ Back to the top](#)

<https://angular.dev/guide/components/content-projection>

# Workshop #1

- Create a new, custom button component (`<my-button>`)
- The contents of that button are:
  - **General** attributes: background red, 180x65px, 2px border solid black
  - An **icon** (save, login, profile, etc)
  - **Text** ('save', 'login', 'profile', etc)
- Each component instance should have **the same** general layout, but **different contents** that is *projected* inside the button component
  - Text
  - Icon
- Example `../130-content-projection`





# Workshop #2

- Open `../130-content-projection` for examples, or use your own app
- Create a new component
- Use `<ng-content>` to project content from outside into the component.
- Add an `<ng-content>` class selector. Use it from the outside component.
- Add an `<ng-content>` element selector. Use it from the outside component.
- Create another component and nest it inside your child component. Use the element selector to project it. Optional: Propagate events up

