Rabobank

Angular Advanced
03 – Content Projection

Peter Kassenaar
info@kassenaar.com

1. Content projection

2. Smart components / View Components

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

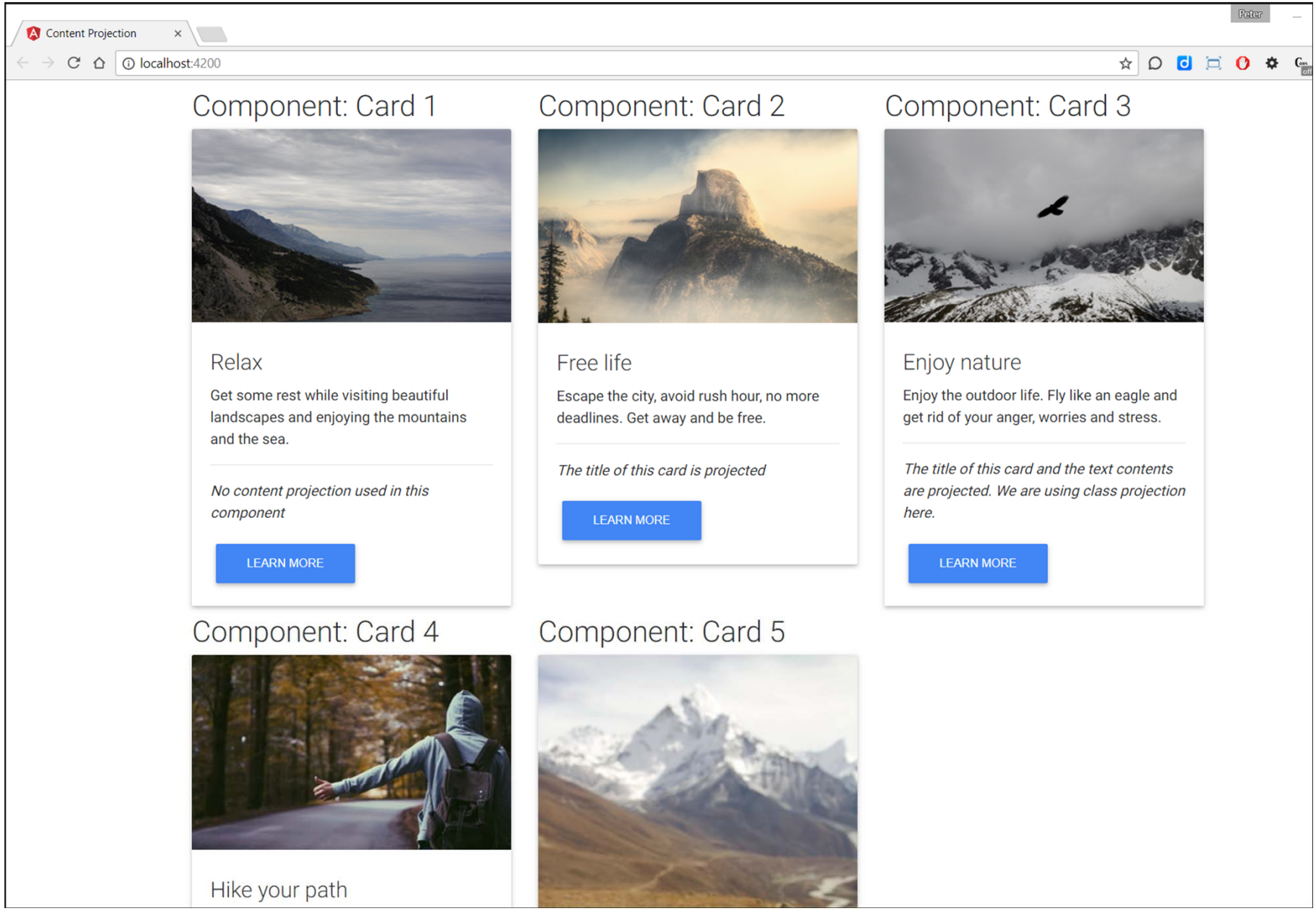
https://dev-academy.com/angular-architecture-best-practices/

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



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>
   <img class="card-photo"
      src="assets/img/photo-04.jpg">
    …

</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

Showing default values in ng-content blocks

- Showing a default value is NOT out-of-the-box behavior
- We have to create it ourselves

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

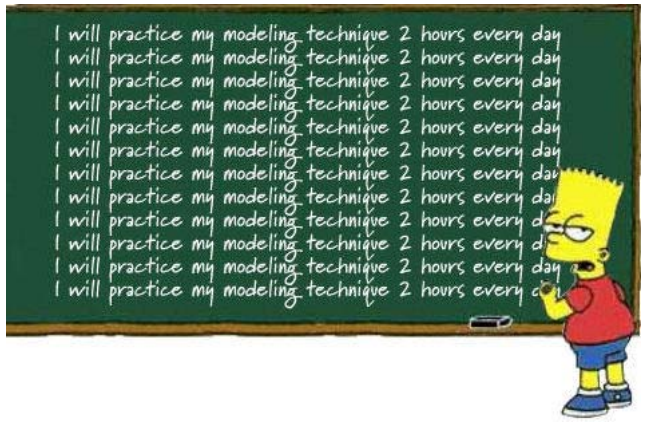
```
<button class="my-button">
    <ng-content select="img"></ng-content>
    <!-- wrap the ng-content block with a local default template variable -->
    <span #ref><ng-content></ng-content></span>
    <!--    Check if there is content available,
            otherwise show default text-->
    <ng-container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

You'll call/use the button as follows:

```
        <!--        first instance, WITH text-->
        <app-my-button>
            <img src="assets/login.png" alt="">
            Login
        </app-my-button>
        <!--        Second instance, WITHOUT text (so showing the default value)-->
        <app-my-button>
            <img src="assets/save.png" alt="">
        </app-my-button>
```

Workshop

- 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



Workshop

- 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