

# Component Communication

6. Content Projection - Pass “html” from parent to child

# Content Projection

- With content projection I can pass from parent component to child component an entire template to process and display

## Parent

```
@Component({
  selector: 'app-root',
  template: `
    <academeez-child>
      <h1>hello from parent</h1>
      <p>I'm passing entire template here</p>
    </academeez-child>
  `
})
```

## Child

```
@Component({
  selector: 'academeez-child',
  template: `
    <ng-content></ng-content>
  `
})
```

- By placing the **<ng-content>** tag, the Renderer will replace that tag with a projected content from the parent component
- The **<ng-content>** can get a select attribute which you can use to create multiple slots

## Parent

```
@Component({
  selector: 'app-root',
  template: `
    <academeez-child>
      <h1>hello from parent</h1>
      <p>I'm passing entire template here</p>
    </academeez-child>
  `
})
```

## Child

```
@Component({
  selector: 'academeez-child',
  template: `
    <ng-content select="h1"><ng-content>
    <ng-content select="p"><ng-content>
  `
})
```

# AfterContentInit

- Angular provides a lifecycle hook **AfterContentInit** which will be called once when projected content finished initialising.
- You can change properties of the class in this hook

```
export class ChildComponent implements AfterContentInit {  
    /**  
     * Called once  
     * when projected content finished initializing  
     */  
    ngAfterContentInit() {  
  
    }  
}
```

# AfterContentChecked

- There are cases when you want to perform some action based on projected content change. Angular provides you with **AfterContentChecked** for that
- This will run every change detection
- You can modify class properties in this hook

```
export class ChildComponent implements AfterContentChecked {  
  /**  
   * called every change detection  
   * perform logic when projected content is changing  
   */  
  ngAfterContentChecked() {  
  }  
}
```

# Use case of content projection

- Component in angular needs to be reusable
- It needs to support plenty of use cases
- We often use **ng-content** as a way to configure how our component look while adapting it to different use cases.

- We often take component like login form or register form and tend to duplicate them along different application
- Perhaps we can make a more generic login for all our apps?
- Perhaps we can customise it to be more generic and more fitting for all the apps using content projection
- **ng-content** can be used in the child to display template items passed between the component or directive tags
- **AfterContent\*** hooks can be used to know when the content projected is initialised or changed
- @angular/material often use **ng-content** patterns along with **@ContentChild** to create generic components.

# Thank You

Next Lesson: 7. @ContentChild