academeez

@ViewChild

Getting items from the template to the component class

@ViewChild

academeez

- With the @ViewChild decorator we can request template item in the component class

```
@Component({
  selector: 'app-root',
  template: `
    <h1 #greeting>hello</h1>
    <input type="text" ngModel />
    <academeez-child></academeez-child>
  `,
})
export class AppComponent {
  @ViewChild('greeting')
  h1: ElementRef;

  @ViewChild(NgModel)
  inputModel: NgModel;

  @ViewChild(ChildComponent)
  child: ChildComponent;
}
```

What we can grab

- With @ViewChild we can grab:
    - **ElementRef**
    - **Component / Directive**
    - **TemplateRef**
    - **ViewContainerRef**

ElementRef

academeez

- **ElementRef** is an angular wrapper around native elements
- On the browser it will be a wrapper around DOM Elements

```
@Component({
  selector: 'app-root',
  template: `
    <h1 #greeting>hello</h1>
  `,
})
export class AppComponent{
  @ViewChild('greeting')
  h1: ElementRef;
}
```

TemplateRef

academeez

- **TemplateRef** represents a template which can be turned to a view
- You create it with **ng-template** or implicitly create it with template directive (example ***ngIf, *ngFor**)
- You can use **@ViewChild** to grab it from the component template

```
@Component({
  selector: 'app-root',
  template: `
    <ng-template #sampleTemplate>
      <h1>This is not shown</h1>
      <p>We can access this using ViewChild</p>
    </ng-template>
  `,
})
export class AppComponent{
  @ViewChild('sampleTemplate')
  sampleTemplate: TemplateRef<any>;
}
```

*ngTemplateOutlet

academeez

- After defining a TemplateRef you can create a view from the template with the directive ***ngTemplateOutlet**

```
@Component({
  selector: 'app-root',
  template: `
    <ng-template #sampleTemplate let-message>
      <h1>This is shown with ngTemplateOutlet {{message}}</h1>
    </ng-template>
    <div *ngTemplateOutlet="sampleTemplate; context: {$implicit: 'hello'}">
    </div>
  `,
})
```

ViewContainerRef

academeez

- Container where one or more views can be attached to a component
- ViewContainerRef can create a host view by creating a component
- Can create embedded views with **TemplateRef**

```
@Component({
  selector: 'app-root',
  template: `
    <!-- I can dynamically create components here
    I can embed TemplateRef view here -->
    <div #container>
    </div>
  `,
})
export class AppComponent{
  @ViewChild('container', {read: ViewContainerRef})
  container: ViewContainerRef;
}
```

ViewContainerRef + TemplateRef EX

- Using **@ViewChild** grab a **ViewContainerRef** and a **TemplateRef** from the component template
- In you component create a button where each click will create the **TemplateRef** in the **ViewContainerRef**

@ViewChild - Component

academeez

- You can use **@ViewChild** to grab a component instance
- Placing the component class in **@ViewChild** will search for the first occurrence of that component
- Or you can place on the component a TRV to grab an exact occurrence of the component

```
@Component({
  selector: 'app-root',
  template: `
    <academeez-child></academeez-child>
    <academeez-child #second></academeez-child>
  `,
})
export class AppComponent{
  @ViewChild(ChildComponent)
  first: ChildComponent;

  @ViewChild('second')
  second: ChildComponent;
}
```

@ViewChild - directive

- You can use **@ViewChild** to grab a directive
- You need to specify the **exportAs** or the **read**

```
@Component({
  selector: 'app-root',
  template: `
    <input ngModel #first="ngModel" ↗
    <input ngModel #second ↗
  `,
})
export class AppComponent{
  @ViewChild('first')
  first: NgModel;

  @ViewChild('second', {read: NgModel})
  second: NgModel;

  @ViewChild(NgModel)
  firstAgain: NgModel;
}
```

AfterViewInit

academeez

- The **@ViewChild** properties will be available in this lifecycle hook
- Called once
- You do init things that require **@ViewChild** members
- Don’t change class properties synchronously

```
export class AppComponent implements AfterViewInit{
  /**
   * called once
   * @ViewChild members will be populated
   * do init stuff the require the @ViewChild
   */
  ngAfterViewInit() {

  }
}
```

AfterViewChecked

academeez

- **@ViewChild** properties can change and we might want to react to those changes
- Example **@ViewChild** captured a component and we need to perform an action when a property of the component is changed
- **AfterViewChecked** is triggered every change detection after the **@ViewChild** is updated

```
export class AppComponent implements AfterViewChecked{
  /**
   * used to trigger an action based on @ViewChild change
   * called every cd
   * Do not alter properties sync
   */
  ngAfterViewChecked() {
  }
}
```

@ViewChildren

academeez

- Similar to **@ViewChild** only based on the selector will look for a query list of all the matches

```
@Component({
  selector: 'app-root',
  template: `
    <input ngModel ↗
    <input ngModel ↗
  `,
})
export class AppComponent {
  @ViewChildren(NgModel)
  ngModels: QueryList<NgModel>;
}
```

Summary

academeez

- With **@ViewChild** we can grab items from out template
- We use the directive class or component class, or we use a TRV to reference items from the template
- TemplateRef can help us dynamically create views
- ViewContainerRef is a container where we can create views
- **AfterViewInit** and **AfterViewChecked** lifecycle hooks can help us react in our code for the view properties initialising and changing

academeez

Thank You

Next Lesson: @ContentChild