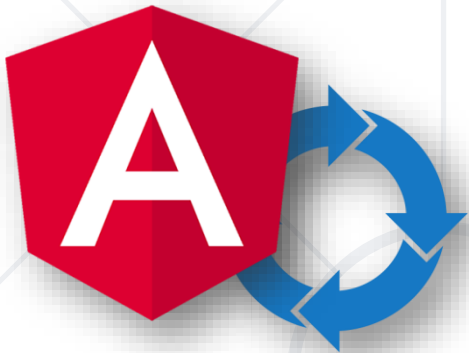# Components and Data Binding

The Building Blocks of Our Application

SoftUni Team
Technical Trainers

Software University

SoftUni

Software University
https://softuni.bg

sli.do

#angular

Table of Contents

1. Components Basic Idea

2. Creating Components

3. Bootstrapping & Modules

4. Data Bindings & Templates

5. Lifecycle Hooks

6. Component Interaction

Components: Basic Idea

The Main Building Block

The Idea Behind Components

- A component controls **part** of the screen (the view)

- You define **application logic** into the component

- Each component has its **own** HTML/CSS template

```
import { Component } from '@angular/core';
```
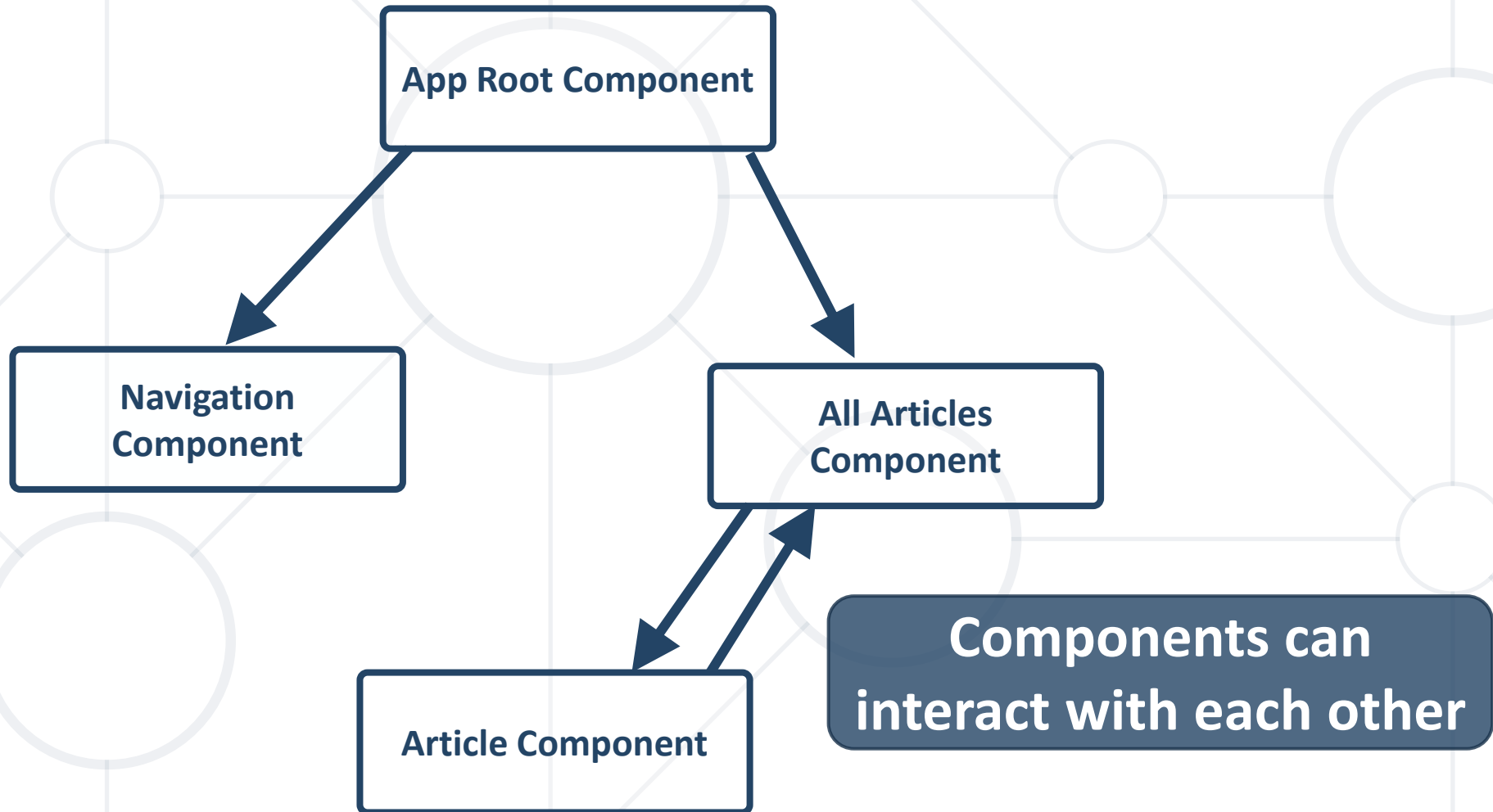
```
@Component({
    selector: 'app-root',
    template: `<h1>{{title}}</h1>`,
    styles: [ `h1 {
    background-color: red;}` ]
})
```

Unique html template and styling

```
export class AppComponent { title = 'App Title'; }
```

The Idea Behind Components

App Root Component

Navigation Component

All Articles Component

Article Component

Components can interact with each other

Creating Components

And Their Unique Templates

Creating Components Manually

- To create a component, we need the **Component** decorator

```
import { Component } from '@angular/core';
```

- It provides **metadata** and tells **Angular** that we are creating a **Component** and not an **ordinary** class

```
@Component({
    selector: 'app-home',
    template: '<h1>Home View</h1>'
})
```

> We call it whilist adding '@' in front and pass in metadata

Creating Components Manually

- Component Metadata

    - **selector** - the component's **HTML** selector

      ```
      selector: 'app-home'
      ```

    - **template** or **templateUrl** - the component's template

      ```
      templateUrl: 'Path to template'
      ```

    - **styles** or **styleUrls** - unique styles for the **current** component

      ```
      styleUrls: 'Array of paths'
      ```

    - **providers** - list of providers that can be **injected** using DI

Creating Components Manually

- After the **creation** of a **component,** we need to **add** it in the **declarations** array at the **app module**

- **NgModules** help **organize** an application **into cohesive blocks** of functionality

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ]
})
```

Creating Components with the CLI

- We can use the Angular **CLI** to **generate** a **new** component

```
ng generate component home
```

- The CLI **creates** a new folder **src/app/home/**

- The CLI directly **imports** the component in the **app module**

Bootstrapping

Starting the Application

Bootstrapping an Application

- An NgModule class describes how the application parts fit together

- Every application has at least one NgModule – the root module

```
platformBrowserDynamic().bootstrapModule(AppModule)
```

- It is used to bootstrap (launch) the application

- Usually it is called AppModule, but it is not necessary

The Initial Module

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
```

```
@NgModule({
    declarations: [ AppComponent ],
    imports: [ BrowserModule ],
    providers: [],
    bootstrap: [ AppComponent ]
})
```

The @NgModule tells Angular how to compile and launch the app

```
export class AppModule { }
```

14

Initial Module Properties

- The **declarations** array

  - Only declarables – (**components**, **directives** and **pipes**)

- The **imports** array

  - Only @**NgModule** classes – integrated (**HttpClientModule**, **BrowserModule**) or **custom** made

Initial Module Properties

- The **providers** array

  - Register **service** providers and **inject** them into components

- The **bootstrap** array

  - The **root** component – used to **launch** the application

- Inserting a **bootstrapped** component usually **triggers** a **cascade** of component creation

Templates & Data Bindings Overview

- A **template** is a form of **HTML** that **tells** Angular how to **render** the component

  - **render** array **properties** using ***ngFor** repeater

  - **render nested** properties of an object

  - **condition** statements using ***ngIf**

  - **attach** events and **handle** them in the component

- They can be both **inline** or in a **separate** file

Render an Array Using *NgFor

```
export class GamesComponent {
    games : Game[];
    constructor() {
        this.games = [ // Array of games ]
    }
}
```

```
<h1>Games List</h1>
    <p>Pick a game to Buy</p>
<ul>
    <li *ngFor="let game of games">
        {{game.title}}
    </li>
</ul>
```

The '*' symbol is required in front

Conditional Statements Using *NgIf

```html
<h1>Games List</h1>
<p>Pick a game to Buy</p>
<ul>
  <li *ngFor="let game of games">
   <div>
      {{game.title}}
   </div>
   <span *ngIf="game.price >= 100">
      Price: {{game.price}}
   </span>
  </li>
</ul>
```

Attach Events

```html
<button (click)="showContent($event)">Show Content</button>
```

```typescript
export class GamesComponent {
    public games: Game[];
    showContent: boolean;

    constructor() {
      this.games = [ // Array of games ]
    }


    showAdditionalContent($event) {
      this.showContent = true;
    }
}
```

Binding Attributes

- Binding attributes

```
export class GamesComponent {
    imgUrl: string;
    constructor() {
     this.imgUrl = "a url to an image"
    }
}
```

```
<img [attr.src]="imgUrl" />
```

The name of the property in the component

Binding CSS Classes or Specific Class Name

- Binding classes

```
<div [class]="badCurly">Bad curly</div>
```

- You can bind to a specific class name

```
<div [class.special]="isSpecial">
 The class binding is special
</div>

<div class="special"[class.special]="!isSpecial">
 This one is not so special
</div>
```

Toggle class "special" on/off

Binding Styles or Styles with Units

- Binding styles

```
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'" >
  Save
</button>
```

- Or styles with units

```
<button [style.font-size.em]="isSpecial ? 3 : 1">
  Big
</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50">
  Small
</button>
```

Reference and Null-safe Operator

- Reference other elements

```
<input #phone placeholder="phone number">
<button (click)="callPhone(phone.value)">Call</button>
```

Phone refers to the input element

- You can also use the null-safe operator

```
<div>The current hero's name is {{game?.title}}</div>
<div>The null hero's name is {{game && game.name}}</div>
```

Template Expressions

- The text **between** the curly brackets is **evaluated** to a string

  ```
  <p>The sum of two + two + four is {{2 + 2 + 4}}</p>
  ```

- Template expressions are **not pure** JavaScript

- You **cannot** use these:

  - Assignments (**=**, **+=**, **-=**, **...**)

  - The **new** operator

  - **Multiple** expressions

  - **Increment** or **decrement** operations (**++** or **--**)

  - **Bitwise** operators

Types of Data Binding

- There are **three types** of data binding

  - From data-source to view

    ```
    {{expression}}
    [target]="expression"
    bind-target="expression"
    ```
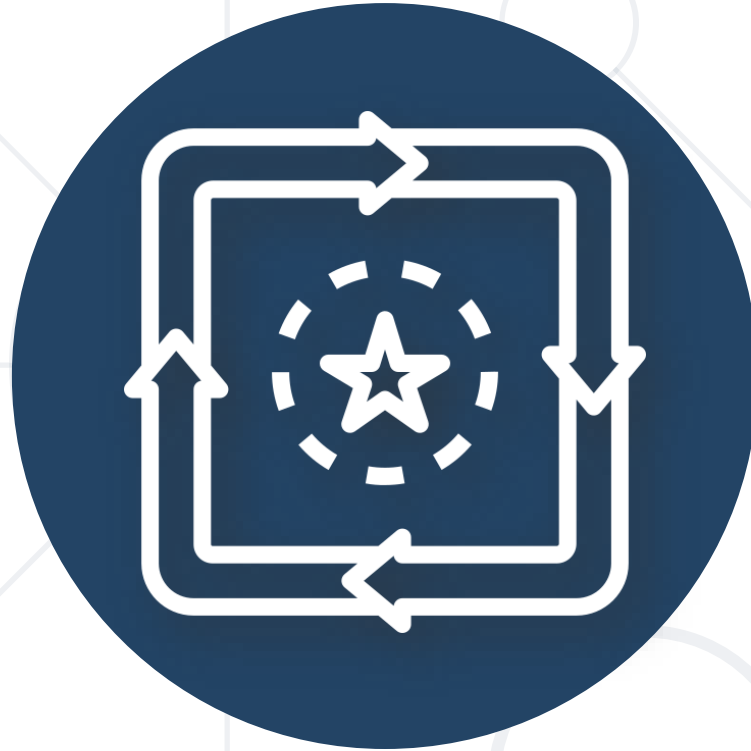
  - From view to data-source

    ```
    (target)="statement"
    on-target="statement"
    ```

  - Two-way

    ```
    [(ngModel)]="expression"
    bindon-target="expression"
    ```

    FormsModule needed

27

Lifecycle Hooks

Intersect Through the Loop

Lifecycle Overview

- A component has a lifecycle **managed** by Angular

- Angular offers lifecycle **hooks** that provide **control** over life moments of a component

- Directive and component instances have a **lifecycle** as Angular **creates**, **updates** and **destroys** them

NgOnInit and NgOnDestroy Example

```
import { Component, OnInit, OnDestroy } from '@angular/core';

@Component({..})
export class GamesComponent implements OnInit, OnDestroy {
  games: Game[];

  ngOnInit() {
    console.log('CREATED');
  }


  ngOnDestroy() {
    console.log('DELETED');
  }
}
```

Called **shortly** after creation

Used for **cleanup**

Other Lifecycle Hooks

- All lifecycle hooks

    - **ngOnChanges**() - when data is changed

    - **ngDoCheck**() - detect your own changes

    - **ngAfterContentInit**() - when external content is received

    - **ngAfterContentChecked**() - when external content is checked

    - **ngAfterViewInit**() - when the views and child views are created

    - **ngAfterViewChecked**() - when the above are checked

    - More at: https://angular.io/guide/lifecycle-hooks

Component Interaction

Passing Data in Between

From Parent to Child

```
import { Component, Input } from '@angular/core';
import { Game } from '../games/game';
```

```
@Component({
  selector: 'game',
  template: `
  <li><div>{{game.title | uppercase}}
  <span *ngIf="game.price >= 100">-> Price: {{game.price}}</span>
  </div></li>`
})
```

```
export class GameComponent {
  @Input('gameProp') game : Game;
}
```

The **prop** will come from **parent**

From Parent to Child

```
<h1>Games List</h1>
 <p>Pick a game to Buy</p>
   <ul>
      <game *ngFor="let game of games"
        [gameProp]="game">
      </game>
   </ul>
<button (click)="showAdditionalContent()">Show Image</button>
```

Render the child into the parent template and pass the needed prop

Component Interaction

- In order to pass data from **child** to **parent** component we need the **Output** decorator and an **Event Emitter**

```
import { Output, EventEmitter } from '@angular/core';
export class GameComponent {
 @Input('gameProp') game : Game;
 @Output() onReacted = new EventEmitter<boolean>();

 react(isLiked : boolean) {
  this.onReacted.emit(isLiked);
 }
}
```

The parent will receive the event

Component Interaction

- The Parent component handles the event

```html
<game *ngFor="let game of games="[gameProp]"="game"
      (onReacted)="onReacted($event)">
</game>
```

```typescript
export class GamesComponent {
  games: Game[];
  likes: number;
  dislikes : number;
  onReacted(isLiked: boolean) {
    isLiked ? this.likes++ : this.dislikes++;
  }
}
```

Summary

- **Each component has its own template**

```
@Component({ selector: 'app', template:
`<h1>{{title}}</h1`})
```

- **There are three types of data binding**
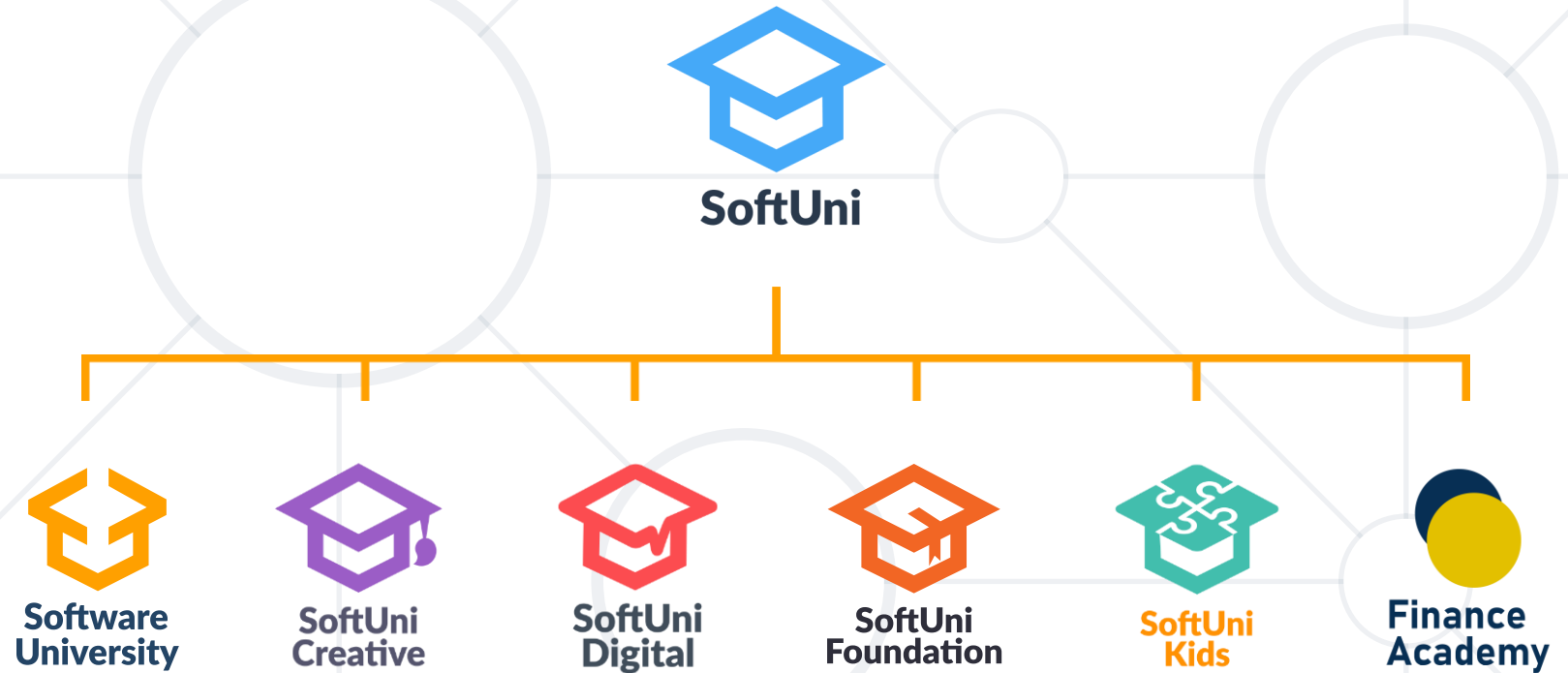
- **We can intersect the lifecycle of a component**

```
ngOnInit() { this.data = // Retrieve data }
```

- **Components can interact with each other**

```
@Output() fromChild = new EventEmitter<boolen>();
```

37

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**

- Unauthorized copy, reproduction or use is illegal

- © SoftUni – https://softuni.org

- © Software University – https://softuni.bg