Nested Components

Mohamed Mukthar Ahmed

Nested Components

Outline

- What are Nested Components?
- Sharing Data
- Sending Data to Child Component
- Configuring Parent Component
- Watching for @Input Changes
- Sending Data to Parent Component
  - Configure Child Component
  - Configure Child Template
  - Configure Parent Component
  - Configure Parent Template
- Fixing Property … has no initializer Error
- Q&A

Mohamed Mukthar Ahmed

What are Nested Components?

- The Angular framework allows us to use a component within another component and when we do so then it is called Angular **Nested Components**.
- The outside component is called the **parent component** and the inner component is called the **child component**.
- Consider the following hierarchy.

```
<parent-component>
  <child-component></child-component>
</parent-component>
```

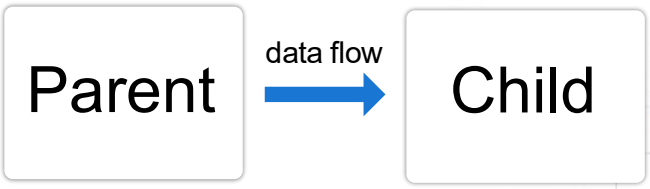
Mohamed Mukthar Ahmed

Sharing Data

- Sharing data between **child** and **parent** components.
- A common pattern in Angular is sharing data between a parent component and one or more child components.
- Implement with the **@Input()** and **@Output()** decorators.
- **@Input()** and **@Output()** give a child component a way to communicate with its parent component.

Mohamed Mukthar Ahmed

@Input



Mohamed Mukthar Ahmed

Sending data to a child component

- The **@Input()** decorator in a child component signifies that the property can receive its value from its parent component.
- To use the **@Input()** decorator in a child component class,
  - first **import Input** and then
  - decorate the property with **@Input()**

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

export class ChildComponent {
  @Input() item = '';
}
```

Mohamed Mukthar Ahmed

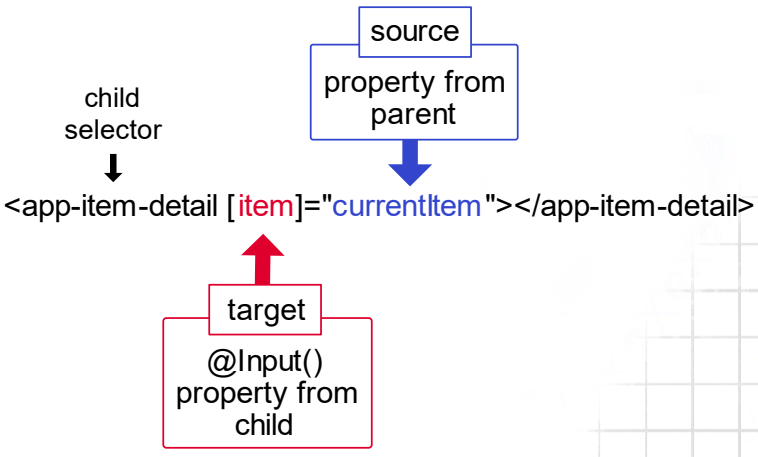
Configuring the parent component

- The next step is to **bind the property** in the parent component's template.
  - Use the child's **selector**, here **<app-item-detail>,** as a directive within the parent component template.
  - Use **property binding** to bind the `item` property in the child to the `currentItem` property of the parent.
- The parent component template:

```
<app-item-detail
     [item]="currentItem">
</app-item-detail>
```

Mohamed Mukthar Ahmed

169



Mohamed Mukthar Ahmed

170

Watching for @Input() changes

- To watch for changes on an **@Input()** property, use **OnChanges**, one of Angular's lifecycle hooks.
- See the **OnChanges** section of the Lifecycle Hooks guide for more details.
- Refer: https://angular.io/guide/lifecycle-hooks

Mohamed Mukthar Ahmed

171

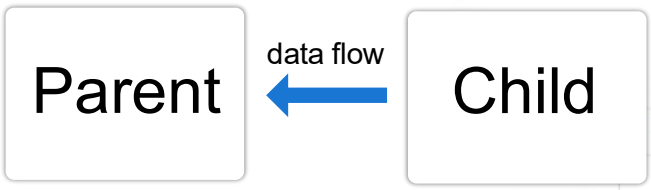
Sending data to a parent component

- The **@Output()** decorator in a child component lets data flow from the child to the parent.
- **@Output()** marks a property in a child component as a **doorway** through which data can travel from the child to the parent.
- The child component uses the **@Output()** property to **raise an event** to notify the parent of the change.
- To raise an event, an **@Output()** must have the type of `**EventEmitter**`, which is a class in **@angular/core**

Mohamed Mukthar Ahmed

172

@Output



Mohamed Mukthar Ahmed

Sending data to a parent component

- Example
- In this example let's set up an **@Output()** in a child component that pushes data from an **HTML** **<input>** to an array in the parent component.

Mohamed Mukthar Ahmed

Configuring the child component

- This example features an **<input>** where a user can enter a value and click a **<button>** that raises an event.
- The **EventEmitter** then relays the data to the parent component.
- Steps:
  - Import **Output** and **EventEmitter** in the child component class
  - In the component class, decorate a property with **@Output()**
  - Create an **addNewItem()** method in the same component class

Mohamed Mukthar Ahmed

Configuring the child component

```
import { Output, EventEmitter } from
'@angular/core';

 export class ItemOutputComponent {
     @Output() newItemEvent =
                new EventEmitter<string>();

     addNewItem(value: string) {
       this.newItemEvent.emit(value);
     }
 }
```

- The **addNewItem()** function uses the **@Output(), newItemEvent**, to **raise an event** with the value the user types into the **<input>.**

Mohamed Mukthar Ahmed

Configuring the child's template

- The child's template has two controls.
- The first is an **HTML <input>** with a `**template reference variable**`, **#newItem**.
- The value property of the **#newItem** variable stores what the user types into the **<input>**

```
<label for="item-input">
      Add an item:</label>
<input type="text" id="item-input"
      #newItem>
<button type="button"
      (click)="addNewItem(newItem.value)">
   Add to parent's list
</button>
```

Mohamed Mukthar Ahmed

Configuring the parent component

- The AppComponent in this example features a list of **items** in an array and a method for adding more items to the array.

```
export class AppComponent {
  items = ['item1', 'item2', 'item3'];

  addItem(newItem: string) {
    this.items.push(newItem);
  }
}
```

- The **addItem()** method takes an argument in the form of a string and then adds that string to the items array.

Mohamed Mukthar Ahmed

Configuring the parent's template

- In the parent's template, bind the parent's method to the child's event.
- Put the child **selector**, here **<app-item-output>,** within the parent component's template.

```
<app-item-output
     (newItemEvent)="addItem($event)" >
</app-item-output>
```

- The event binding, (newItemEvent)='addItem($event)', connects the event in the child, **newItemEvent**, to the method in the parent, **addItem().**
- The $event contains the data that the user types into the <input> in the child template UI.

Mohamed Mukthar Ahmed

179

Configuring the parent's template

- To see the **@Output()** working, add the following to the parent's template:

```
<ul>
  <li *ngFor="let item of items">
      {{ item }}
  </li>
</ul>
```

Mohamed Mukthar Ahmed

180

Fixing Property '…' has no initializer

- If you are using latest version of **Angular**, you might have encountered

  Property '...' has no initializer ERROR

- This is because of **Strict Class Initialization flag** introduced in **TypeScript 2.7** version.

- The **flag** is enabled by **default**.

- To fix **Property '...' has no initializer** Error, we can use the following method:

Mohamed Mukthar Ahmed

181

Fixing Property '…' has no initializer

- To fix **Property '...' has no initializer Error**, we can use the following method:
  - Disable strictPropertyInitialization flag
  - Adding undefined type to the property
  - Add definite assignment assertion to property
  - Add initializer to property
  - Assignment in the Constructor

Mohamed Mukthar Ahmed

182

Mohamed Mukthar Ahmed

183