Angular Component Communication

Overview

- Communicating with a Template
    - ViewChild and ViewChildren
    - Communicating with a Child Component
    - Communicating with a Parent Component
- Communicating Through a Service
    - Communicating Through a State Management Service
    - Communicating Through Service Notifications
- Communicating Using the Router

Component <-> Template

- View updates when data changed
- React to user changes
- Ask an element to set a property or perform a task
- Check form or control state

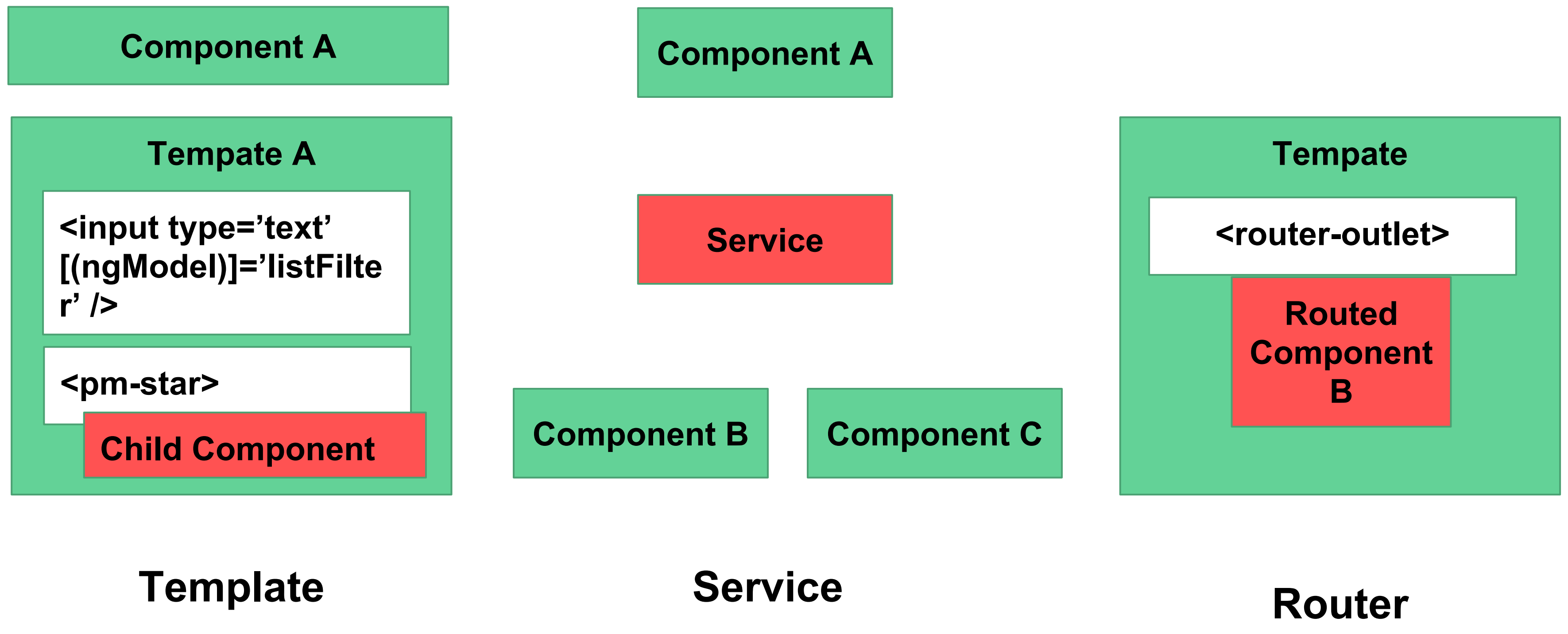
Component <-> Service <-> Component

- Retain state
- Share data
- Send notifications

Component <-> Router <-> Component

- Pass required data
- Pass optional data

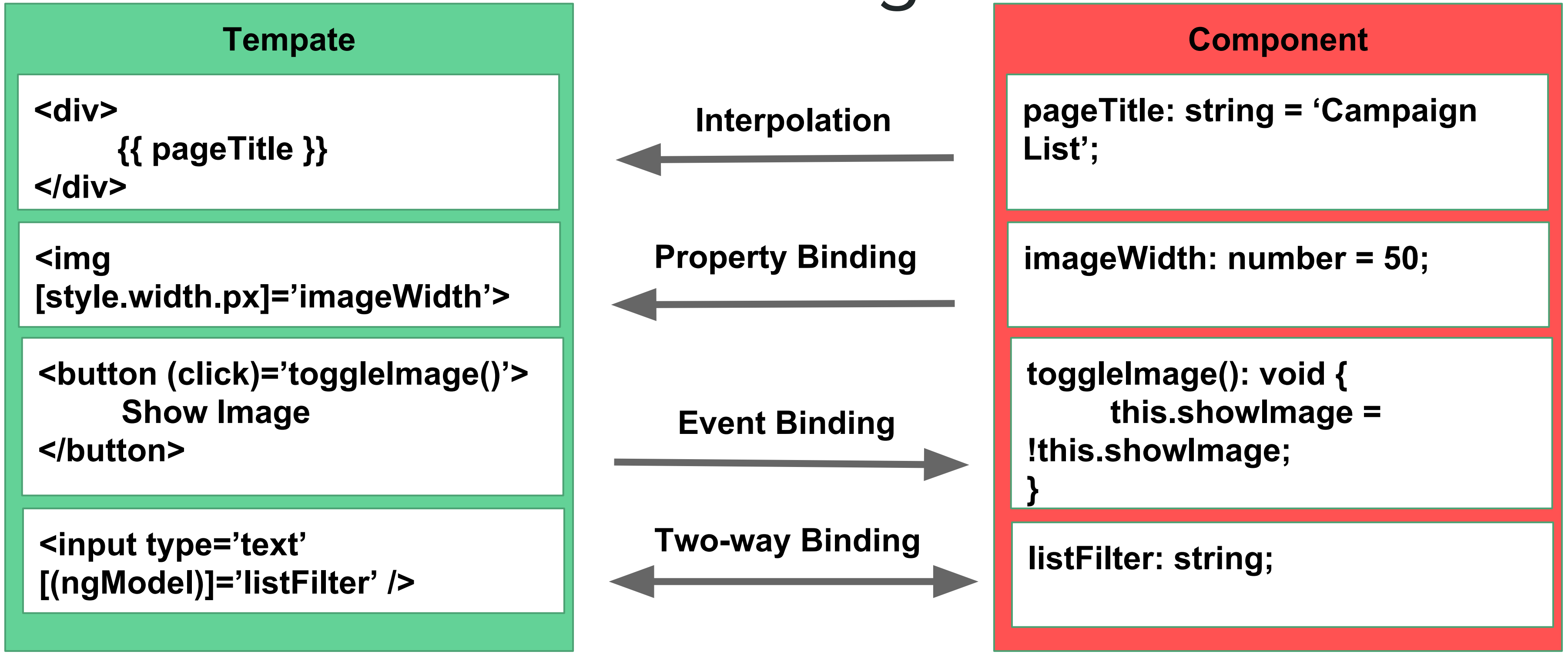
Component Communication



Communicating with a Template

- Binding and Structural Directives
- Two-way Binding, the Long Way
- Getters and Setters

Binding



```
<div>
        {{ pageTitle }}
</div>
```

```
pageTitle: string = 'Campaign
List';
```

```
<img
[style.width.px]='imageWidth'>
```

```
imageWidth: number = 50;
```

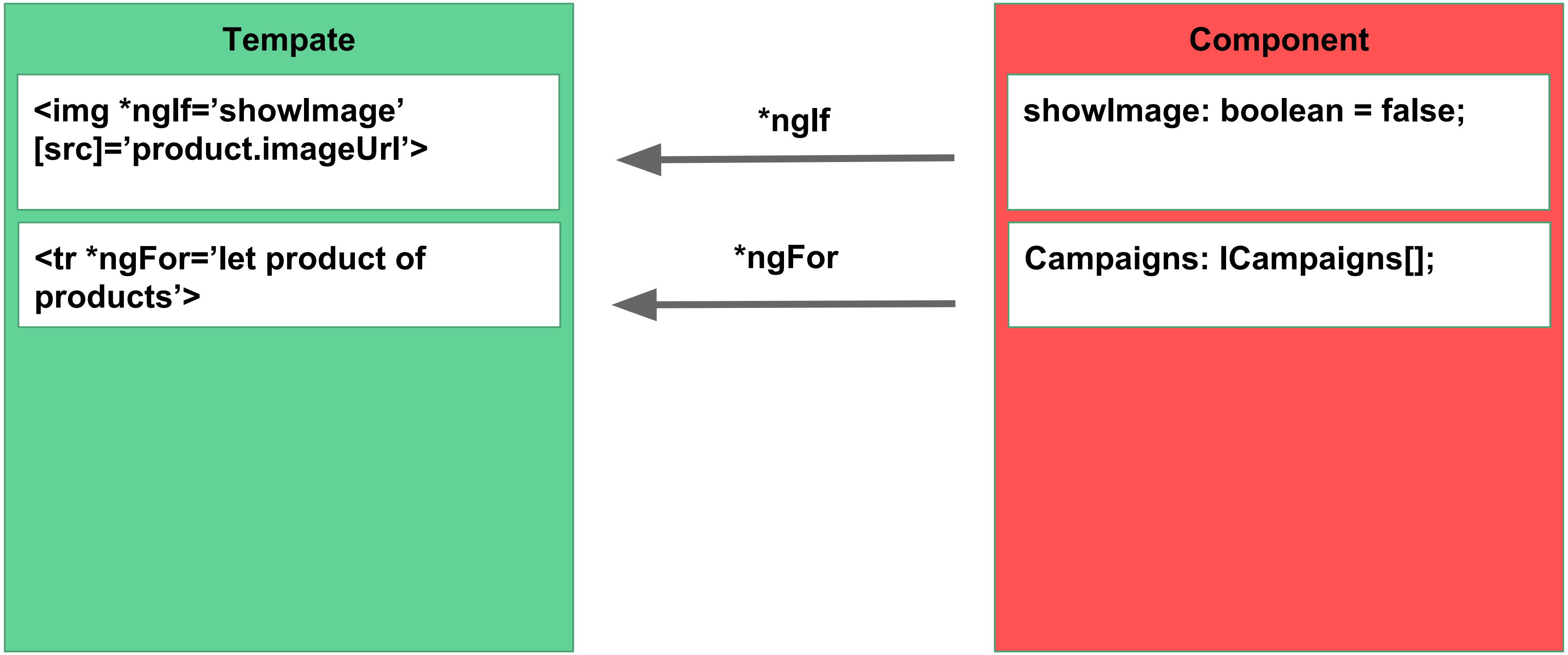
```
<button (click)='toggleImage()'>
        Show Image
</button>
```

```
toggleImage(): void {
        this.showImage =
!this.showImage;
}
```

```
<input type='text'
[(ngModel)]='listFilter' />
```

```
listFilter: string;
```

Structural Directives



```
<img *ngIf='showImage'
[src]='product.imageUrl'>
```

```
<tr *ngFor='let product of
products'>
```

```
showImage: boolean = false;
```

```
Campaigns: ICampaigns[];
```

Notifying the Component of User Changes

- Two-way binding, the long way
- Getter and setter
- valueChanges observable

Two-way Binding, the Long Way

```
<input type='text' [(ngModel)]='listFilter' />
```

```
<input type='text' [ngModel]='listFilter'
(ngModelChange)='listFilter=$event' />
```

```
<input type='text' [ngModel]='listFilter'
(ngModelChange)='onFilterChange($event)' />
```

Two-way Binding, the Long Way

```
<input type='text' [ngModel]='listFilter'
(ngModelChange)='onFilterChange($event)' />
```

Plus:

- Notifies the component when the user changes the value
- Allows any logic in the component method
- Caught in the template

Caveats:

- No two-way binding
- Caught in the template
- Uncommon syntax

Getter and Setter

```
private _listFilter: string;
get listFilter(): string {
      return this._listFilter;
}
```

```
set listFilter(value: string) {
      this._listFilter = value;
}
```

Plus:

- **Notifies the component when the user changes the value**
- **Allows any logic in the setter**
- **Caught in the component class**

Caveats:

- **One line of code becomes 7**

ViewChild and ViewChildren

- ViewChild
- ViewChildren
- ViewChild and Angular Forms
    - valueChanges Observable
- ViewChild and ngIf

Getting a Reference

DOM

```
let divElement = document.getElementById(‘divElementId’);
```

Decorator

```
@ViewChild(‘divElementVar’) divElementRef;
```

ViewChild

Directive

```
@ViewChild(NgModel) filterInput: NgModel;
```

```
<input type=’text’ [(ngModel)]=’listFilter’ />
```

Custom Directive / Child Component

```
@ViewChild(StarComponent) star: StarComponent;
```

```
<pm-star [rating]=’product.starRating’></pm-star>
```

Template Reference Variable

```
@ViewChild(‘divElementVar’) divElementRef: ElementRef;
```

```
<div #divElementVar>{{pageTitle}}</div>
```

Considerations When Using nativeElement

- Using nativeElement -> directly accessing the DOM
- Tightly coupled to the browser
- May not be able to use server-side rendering or web workers
- Can pose a security threat, especially if accessing innerHtml

ViewChildren

```
@ViewChildren(‘divElementVar’)
divElementRefs: QueryList<ElementRef>;
```

Differences:

- Returns a QueryList of element or directive references
- Tracks changes in the DOM

```
this.divElementRefs.changes.subscribe(() => {
    // Code here
})
```

ViewChildren

Directive

```
@ViewChildren(NgModel) inputs: QueryList<NgModel>;
```

Custom Directive / Child Component

```
@ViewChildren(StarComponent) stars: QueryList<StarComponent>;
```

Template Reference Variable

```
@ViewChildren(‘divElementVar’) divElementRefs: QueryList<ElementRef>;
```

Template Reference Variables

```
@ViewChildren(‘filterElement, nameElement’) divElementRefs:
QueryList<ElementRef>;
```

ViewChild and Angular Forms

Tempate

```
<input type='text'
[(ngModel)]='listFilter' />
```

Component

```
@ViewChild(NgModel) filterInput:
NgModel;
```

```
this.filterInput.valueChanges.subscribe
(() => this.performFilter(this.listFilter));
```

Angular Forms

Tempate-Driven

- **Angular creates the form data structures**
- **Based on info in the template**
- **Access reference with ViewChild**

Reactive

- **We create the form data structures**
- **Defined in the component class**
- **No need for ViewChild**

```
this.filterInput.valueChanges.subscribe(
    () => this.performFilter(this.listFilter)
);
```

Template-Driven Forms / No Form

```
<form (ngSubmit)='saveProduct()'>
```

```
<input type='text' [(ngModel)]='listFilter' />
```

ViewChild/ViewChildren: Html Element

```
@ViewChild(‘divElementVar’)
divElementRefs: ElementRef;
```

Plus:

- Provides a nativeElement property
- Access any Html element properties
- Call any Html elment methods

Caveats:

- ViewChild reference not reliably available until AfterViewInit
- ViewChild reference not available if the element is not in the DOM
- Does not work with server-side rendering or web workers
- Could cause a security concern, especially with innerHtml

ViewChild/ViewChildren: Directive

```
@ViewChild(NgModel) filterInput: NgModel;
```

Plus:

- Provides reference to the directive's data structures
- Access any properties

Caveats:

- ViewChild reference not reliably available until AfterViewInit
- ViewChild reference not available if the element is not in the DOM
- NgForm and NgModel data structures are read-only

Subscribe to the valueChanges Observable

```
@ViewChild(NgModel) filterInput: NgModel;
```

```
this.filterInput.valueChanges.subscribe(
      () => this.performFilter(this.listFilter)
);
```

Plus:

- Favor this technique if using other NgModel information

Caveats:

- Watch out for ngIf
- Reference not reliably available until AfterViewInit

5.1

Communicating with a Child Component

- **Child Components**
- **Parent to Child Communication**
- **Input Property**
- **Watching for Changes**
    - **Getter and Setter**
    - **OnChanges Lifecycle Hook**
- **Template Reference Variable**
- **ViewChild Decorator**

Defining Child Components

When?:

- When the piece performs a specific task that we want to encapsulate
- When the piece is sufficiently complex such that we want to build and test it as a separate component
- When the piece could be reused within a component or in multiple components.

When would not?:

- if it is easier to maintain the component as one unit.

Parent to Child

- @Input() decorator
- Getter/Setter
- OnChanges

Getter/Setter:

- Favor to only react to changes to specific properties

OnChanges:

- Favor to react to any input property changes
- Favor if current and prior values

Input: Passing Data to the Child

Parent Template

```
<pm-criteria
[displayDetail]='includeDetail'>
</pm-criteria>
```

Child Component

```
@Input() displayDetail: boolean;
```

Parent Component

```
includeDetail: boolean = true;
```

Changes to an Input Property

Parent Template

```
<pm-criteria
[displayDetail]='includeDetail'>
</pm-criteria>
```

Child Component

```
@Input() displayDetail: boolean;
```

Parent Component

```
includeDetail: boolean = true;
```

```
includeDetail: boolean = false;
```

Child Template

```
<div *ngIf='displayDetail'>
        <h3>Filtered by:
{{listFilter}}</h3>
</div>
```

Watching for Changes to an Input Property

Child Component

```
Private _hitCount: number;
get hitCount(): number {
        Return this._hitCount;
}
@Input()
set hitCount(value: number) {
        this._hitCount = value;
}
```

Getter and Setter

Child Component

```
@Input() hitCount: number;
```

```
ngOnChanges(changes:
SimpleChanges){
}
```

OnChanges Lifecycle Hook

Parent to Child

- Template Reference Variable
- @ViewChild

ViewChild:

- Use from the parent's class

Template Reference Variable:

- Use from the parent's template

Template Reference Variable: Referencing a Child Component

Parent Template

```
<pm-criteria #filterCrieria
[displayDetail]='includeDetail'>
</pm-criteria>
```

```
{{ filterCriteria.listFilter }}
```

```
{{ filterCriteria.clear() }}
```

Child Component

```
@Input() displayDetail: boolean;
```

```
listFilter: string;
```

```
clear(): void {

}
```

ViewChild: Referencing a Child Component

Parent Template

```
<pm-criteria
[displayDetail]='includeDetail'>
</pm-criteria>
```

Parent Component

```
Export class ProductListComponent
        Implements OnInit, AfterViewInit
```

```
@ViewChild(CriteriaComponent)
filterComponent: CriteriaComponent;
```

```
ngAfterViewInit(): void {
        this.filterComponent.clear();
}
```

Child Component

```
@Input() displayDetail: boolean;
```

```
listFilter: string;
```

```
clear(): void {

}
```

Communicating with a Parent Component

- **Child to Parent Communication**
- **Output Propeties**

Event notification

- **When a child component needs to communicate with its parent, emit an event using an output property.**
- **the child needs to notify the parent of an action and optionally pass along some data.**

Output: Notifying the Parent

Parent Template

```
<pm-criteria  [displayDetail]='includeDetail'
        (valueChange)='onValueChnage($event)'
>
        </pm-criteria>
```

Parent Component

```
onValueChange(value: string): void {
        this.performFilter(value);
}
```

Child Component

```
@Output()
valueChange:EventEmitter<string>;
```

```
this.valueChange.emit(value);
```

Demo

- **Product-list.component**
- **criteria.component**