Introduction

Hello friends! In this guide, we're going to learn about template reference.
There are multiple ways to interact between components, such as `@Input` and `@Output`. But what if we have to call the child function in the parent component?
In this scenario, we could create a new function in the parent component, or we could create a service file with a common function and use it in the parent and child function. Today we'll learn another method known as *template reference* so you can avoid creating a different function in the parent component or a different service file.

Creating Components

We need to create two components: a child component and a parent component. It is a good practice to have a different folder for each file. So I'm going to create all the components in `components` folder.

```
1
ng g c /components/parent --skipTests=true
```

console

```
1
ng g c /components/child --skipTests=true
```

console
In the above snippet, I've used the short form of the `generate`, or `g`, and the short form of the `component`,or `c`. Notice that `--skipTests=true`, lthough we're not going to write and unit test case, so I've used that to skip the file of the testing.
So we are ready to go. Let's get dive into template reference.

Make Function in Child Component

Now it's time to make a function in the child component so we can call it in the parent component by using the reference of the child component.
child.component.ts

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


@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.scss']
})
export class ChildComponent implements OnInit {
```

```
  constructor() { }

  ngOnInit() {
  }

  hello(){
    console.log('hello from child component');
  }

}
```

I've made a `hello()` function. Once the function is called, we'll get a message on the console.

Get Reference of Child Component

We've successfully made a function in the child component. Now it's time to put the child component in the parent component.
parent.component.html

```
<app-child #child></app-child>
```

Noticed that I've written `#child` under the tag of `app-child`. In Angular, to refer to any component, we need to put `#` with a string. I've written `#child` in our case so we can use it to get the reference of the child component.

Use of @ViewChild Decorator

The `@ViewChild()` decorator is one of the most useful decorators in Angular. It is used to get the reference of the component. The syntax of `@ViewChild()` is :

```
@ViewChild('referenceVariable') childVariable:ChildComponent
```

It accepts a reference variable, which is `#child` in our case. We've written this in the parent component under the child tag. `childVariable` can be anything—it's just a variable name that we will use to access a variable or function of the child component. To make a typed secure, we should always write the type of the variable by using a colon (:).

Use of ngAfterViewInit() Angular Lifecycle Hooks

`ngAfterViewInit` is one of the Angular lifecycle hooks. We can use the `@ViewChild` variable into it. We can't access the `@ViewChild` variable in `ngOnInit`, `ngOnChanges`, or `ngDoCheck` because it can be accessed only after our view is initialized.

Call Child Function in Parent Component

parent.component.ts

```
import { Component, OnInit, ViewChild, ElementRef } from
'@angular/core';
import { ChildComponent } from '../child/child.component';

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.scss']
})
export class ParentComponent implements OnInit {

  @ViewChild('child') child: ChildComponent;

  constructor() { }

  ngOnInit() {
  }

  ngAfterViewInit(){
    this.child.hello();
  }

}
```

In the above snippet, we've used the `@ViewChild` decorator to get the reference of the child element. We used the child element reference variable I've called `hello()` function in `ngAfterViewInit()`, which we've defined in the child component.

Output:

```
hello from child component
```

console

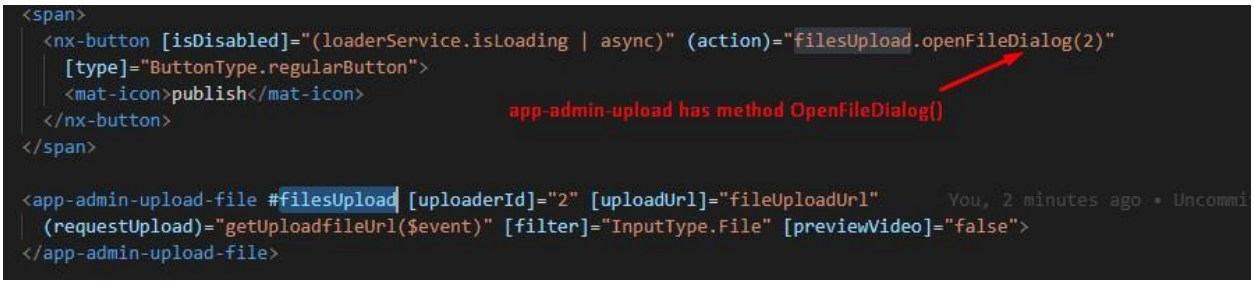
You'll get the same output at the console.

Conclusion

Template reference is one of the best ways to minimize code. It saves a line of code and gives you the same result with minimum time. Lines of code do matter if you talk about productivity.
You can read more about template reference here.

Call child method in parent html file

In this case the child method is openFileDialog()




```
<span>
  <nx-button [isDisabled]="(loaderService.isLoading | async)" (action)="filesUpload.openFileDialog(2)"
    [type]="ButtonType.regularButton">
    <mat-icon>publish</mat-icon>
  </nx-button>
</span>

<app-admin-upload-file #filesUpload [uploaderId]="2" [uploadUrl]="fileUploadUrl"
  (requestUpload)="getUploadfileUrl($event)" [filter]="InputType.File" [previewVideo]="false">
</app-admin-upload-file>
```

Another use of template variable

A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a web component (Read more at Angular.io). That means you can easily access the variable anywhere in the template.

You declare a reference variable by using the hash symbol (#). The **#firstNameInput** declares a **firstNameInput** variable on an `<input>` element.

```
<input type="text" #firstNameInput><input type="text" #lastNameInput>
```

After that, you can access the variable anywhere inside the template. For example, I pass the variable as a parameter on an event.

```
<button (click)="show(lastNameInput)">Show</button>
```

Remember that the lastNameInput belongs to HTMLInputElement type.

```
show(lastName: HTMLInputElement){
    console.log(lastName.value);
}
```

Usually, the reference variable can only be accessed inside the template. However, you can use **ViewChild** decorator to reference it inside your component.

```
import {ViewChild, ElementRef} from '@angular/core';// Reference
firstNameInput variable inside Component
@ViewChild('firstNameInput') nameInputRef: ElementRef;
```

After that, you can use ***this.nameInputRef*** anywhere inside your Component.

```
show(lastName: HTMLInputElement){
  this.fullName = this.nameInputRef.nativeElement.value + ' ' +
lastName.value;
}
```