Output Property Binding

[How to pass data between components in angular](https://stackoverflow.com/questions/50444406/how-to-pass-data-between-components-in-angular)

[Angular 4 pass data between 2 not related components](https://stackoverflow.com/questions/44414226/angular-4-pass-data-between-2-not-related-components)

Greetings Earthlings… welcome back to codewithsrini.

This lecture will primarily be talking about output property binding or in other words how to pass data between components in Angular.

In my previous lecture called “Routing and Navigation” I left out a minor implementation detail regarding the login mechanism, where in, if a user successfully logs in, the login menu doesn’t change at all. Ideally the menu should change itself to “Logout”.

Well, now we will implement that change in this lecture.

In our app.component template, I want to create a custom event for our login selector like “change” that we can bind a method in our app.component

<login (change)="onLoginEventFired()"></login>

In our app component, let’s create method called onLoginEventFired()

We are not there yet, got to work on login component to get this piece working.

Let’s open up our login component class file.

Firstly, on the top I am gonna import Output decorator function from angular/core.

login.component.ts

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

then, I am gonna declare a field here. Remember the name of the field should be the exact same name of an event we wanted to raise.

So in this case, “change” and we should decorate this field with @Output decorator.

@Output() change;

And we should initialize this field to an EventEmitter class;

@Output() change = new EventEmitter();

Now you can see a compilation error here, we need to add this class on top for import.

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

Finally, in our login method, after successful login, we want to raise an event, we use EventEmitter for that.

So this.change and here we have a method called “emit” and here we gonna publish our event. That basically means, notify others or other components that something has happened.

Now let’s test it.

So I am gonna do a fake login, if you notice in the console, we got that “Login Event Fired” message which is actually a method in app.component class.

The next step is, pass an event data to our app.component.

In our login component, when raising an event, we can pass some data optionally and this value will be available to all subscribers of this event.

Now we can pass an object that will contain user details, assuming that we did a successful login.

So, (login.component.ts)

this.change.emit({ user: 'codewithsrini', usertype: 'internal', role: 'admin'});

Back in app.component.ts

onLoginEventFired(eventArgs) {

console.log('Login Event Fired', eventArgs);

}

Finally the last step, in our app component html

App.component.html

<login (change)="onLoginEventFired($event)"></login>

Here the $event represents the actual javascript object with a property called new value.

Also in the app component, since this is an object, we gonna have an argument called eventArgs.

Now if we look at console.log in our browser, we can capture the value passed from our login component.

Now, to make this as a strict type checking, we can create an interface called LoggedInUser and with this we get compile time checking. This will make our code strongly typed.

On top of our **login.component.ts**

export interface LoggedInUserEventArgs {

user: string;

usertype: string;

role: string

};

And in our method;

onLoginEventFired(eventArgs: LoggedInUserEventArgs) {

console.log('Login Event Fired', eventArgs);

}

And in our app component class, we gonna add isUserLogged and user with type as interface LoggedInUserEventArgs

isUserLogged = false;

user: LoggedInUserEventArgs;

In our template, let’s modify our code to show or hide login / logout with user name.

<li \*ngIf="!isUserLogged"class="nav-item">

      <a href="#" class="nav-link">Login</a>

    </li>

<li \*ngIf="isUserLogged"class="nav-item">

      <a href="#" class="nav-link">Logout Logout ({{user.user}}) </a>

    </li>

The lesson we learnt here is.

Using output decorator class, we can subscribe to any operation published from the login component.

In our case, we are publishing a successful login and passing user details to our app component. Both are different component and app.component is a parent of login component.

This is how we pass data from one component to another component.

And finally, we need to alias our output decorator function, this is make our code hard to break. So if in the future we change this field name, our Output method still works.

Thank You for watching this lecture, if you like it, please give it a like and if you have any suggestions or comments do write them below and finally subscribe to my channel to get latest updates.