# Angular

## Introduction

### 

### What is Angular



### no code

### no code

### CLI Deep Dive



[**https://github.com/angular/angular-cli**](https://github.com/angular/angular-cli)

### Project set up and first app

Need to isntall Node js

Create repo on github



git clone <https://github.com/ron2015schmitt/project-angular.git>

Then install angular using npm:



cd project-angular/

**global** install of angular

npm install -g @angular/cli@latest

create an angular project (ng is the CLI for Angular). This takes a few minutes

ng new my-first-app --no-strict



Now run the angular web server on the project

cd my-first-app

ng serve

eventually, after about a minute, it should finish compiling:

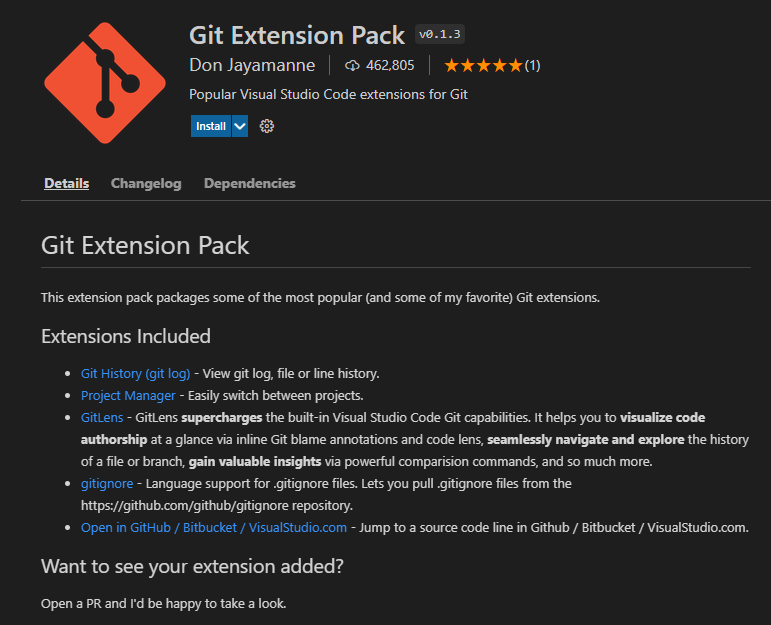


Type <http://localhost:4200/> in chrome and the following web page should appear (Angular is acting as the server)



### Editing the First App

Install VSCode (or other IDE)



Open folder in VSCode. You will see a whole slew of files that angular created



Copy my-first-app folder to my-first-app-chap-7 and open with VSCode

Open a terminal window and build the node\_modules folder

|  |
| --- |
| npm install |

Edit the html file by deleting everything and writing below

|  |
| --- |
| <div >    <h1>Hi this is {{ title }}!</h1>  </div> |
|  |

Now compile and runthe web server

|  |
| --- |
| ng serve |

Type <http://localhost:4200/> in chrome



Variable title used in the html file above is set as shown below



Change the title variable as shown below



And the web page will automatically update



Hit F12 to see source code in browser. The index.html file is the starting point of the app. Notice that the index.html is just a bunch of scripts: angular creates code dynamically. Also notice webpack listed on the left. Angular uses wbepack!





Also notice the use of custom html tag app-root





This is defiend inside the Angular component definition for app



Now change the variable title to name and give it a new value shown below in the ts file



Now change the html file:

|  |
| --- |
| <input type="text">  <p>{{ name }}</p> |
|  |

Result is



Now add FormsMoldule to src\app\app.module.ts



Now add [(ngModel)]=”name” to input which bind the input value to the variable name



Reuslt is as follows and it automatically updates as you type!!



### no code

### no code

### What is TypeScript?

A super script of JavaScript and is compiled to JavaScript



### Optional TypeScript Intro as appendix



### A Basic Project Using Bootstrap for Styling

Copy project and open in vscode

|  |
| --- |
| npm install |

Now install bootstrap 3

npm install --save bootstrap@3

add bootstrap to the project json

|  |
| --- |
| my-first-app\angular.json |
| {    "$schema": "./node\_modules/@angular/cli/lib/config/schema.json",    "version": 1,    "newProjectRoot": "projects",    "projects": {      "my-first-app": {        "projectType": "application",        "schematics": {},        "root": "",        "sourceRoot": "src",        "prefix": "app",        "architect": {          "build": {            "builder": "@angular-devkit/build-angular:browser",            "options": {              "outputPath": "dist/my-first-app",              "index": "src/index.html",              "main": "src/main.ts",              "polyfills": "src/polyfills.ts",              "tsConfig": "tsconfig.app.json",              "assets": [                "src/favicon.ico",                "src/assets"              ],              "styles": [                "node\_modules/bootstrap/dist/css/bootstrap.min.css",                "src/styles.css"              ],              "scripts": []            },            "configurations": {              "production": {                "budgets": [                  {                    "type": "initial",                    "maximumWarning": "2mb",                    "maximumError": "5mb"                  },                  {                    "type": "anyComponentStyle",                    "maximumWarning": "6kb",                    "maximumError": "10kb"                  }                ],                "fileReplacements": [                  {                    "replace": "src/environments/environment.ts",                    "with": "src/environments/environment.prod.ts"                  }                ],                "outputHashing": "all"              },              "development": {                "buildOptimizer": false,                "optimization": false,                "vendorChunk": true,                "extractLicenses": false,                "sourceMap": true,                "namedChunks": true              }            },            "defaultConfiguration": "production"          },          "serve": {            "builder": "@angular-devkit/build-angular:dev-server",            "configurations": {              "production": {                "browserTarget": "my-first-app:build:production"              },              "development": {                "browserTarget": "my-first-app:build:development"              }            },            "defaultConfiguration": "development"          },          "extract-i18n": {            "builder": "@angular-devkit/build-angular:extract-i18n",            "options": {              "browserTarget": "my-first-app:build"            }          },          "test": {            "builder": "@angular-devkit/build-angular:karma",            "options": {              "main": "src/test.ts",              "polyfills": "src/polyfills.ts",              "tsConfig": "tsconfig.spec.json",              "karmaConfig": "karma.conf.js",              "assets": [                "src/favicon.ico",                "src/assets"              ],              "styles": [                "src/styles.css"              ],              "scripts": []            }          }        }      }    },    "defaultProject": "my-first-app"  } |

Remove the FormsModule from app.module.ts



Change app.component.thtml file to





Run ng serve



### About Course Code



<https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655698#questions/8079942>

## The Basics

### Intro – no code

### How an Angular App gets loaded and started

Copy project and open in vscode

|  |
| --- |
| npm install  ng serve |

Add Loading… to the index.html



The first script executed is main.ts



The line

platformBrowserDynamic().bootstrapModule(AppModule)

causes the module to be loaded



Inside app.module.ts the bootsrap array lists all the components used, in this case AppComponent



The app.component.ts file then calls out the html and css files of the component



### Components are important

Angular is based upon components

### Creating a component

Copy project and open in vscode

|  |
| --- |
| npm install  ng serve |

Time to create a component named server

Add a folder and file as shown below, as a sub-folder of app



Enter file contents

|  |
| --- |
| src\app\server\server.component.ts |
| import { Component } from '@angular/core';  @Component({    selector: 'app-server',    templateUrl: './server.component.html',  })  export class ServerComponent {} |

Create a blank file as shown below



### Understanding the role of AppModule and Component Declaration

Continue with chap17 version of code

Now we need to declare our server component in the app.module.ts



### Using Custom Components

|  |
| --- |
| **server.component.html** |
|  |

|  |
| --- |
| **app.component.html** |
|  |



### Creating Components with the CLI & Nesting Components

ng generate component servers



The following files are added



The following lines are added to app.module.ts



Make the following edit

|  |
| --- |
| **app.component.html** |
|  |

Make the followign edit

|  |
| --- |
| **servers.component.html** |
|  |



### Working with Component Templates

You can also define the component templete inline using template instead of templateUrl

|  |
| --- |
| **servers.component.ts** |
|  |

|  |
| --- |
| **http://localhost:4200/** |
|  |

### Working with Component Styles

Define styles in the css files.

|  |
| --- |
| **app.component.css** |
|  |

|  |
| --- |
| **app.component.html** |
|  |

|  |
| --- |
| **http://localhost:4200/** |
|  |

or you can also define the component stle inline using styles instead of styleUrls

|  |
| --- |
| **app.component.ts** |
|  |

|  |
| --- |
| **http://localhost:4200/** |
|  |

### Fully Understanding the Component Selector

using syntax

@Component({

  selector: 'app-servers',

we call out a component directly as

<app-servers>

but we can also put the name in brackets

@Component({

  selector: '[app-servers]',

we then call out using app-servers as an attribute inside another tag, such as

<h3>I am the AppComponent</h3>

<hr>

<!-- <app-servers></app-servers> -->

<div app-servers></div>

<span app-servers></span>

|  |
| --- |
| **servers.component.ts** |
|  |

thirdly we can use dot syntax

@Component({

  selector: '.app-servers',

which defines a class and is called out by

<div class="app-servers"></div>

**Assignment 1: Practicing Components**

Time to practice what you learned about Components. In this assignment, you're going to create, use and style your own components and see practice how you can build up your Angular app with Components.

download basics-assignment-1-start.zip and open in VSCode

npm install

ng serve

ng generate component warning-alert



ng generate component success-alert



|  |
| --- |
| **app.component.html** |
|  |

|  |
| --- |
| **warning-alert.component.css** |
| p {    color: black;    background-color: yellow;    font-weight: 600;    position: fixed;    left: 300px;  } |

|  |
| --- |
| **success-alert.component.css** |
| p {    color: black;    background-color: yellowgreen;    font-weight: 600;    position: fixed;  } |

|  |
| --- |
| **http://localhost:4200/** |
|  |

### [OPTIONAL] Assignment Solution

### What is Databinding?



### String Interpolation

You can write any TypeScript expression in {{ code\_here }} as long as it returns a string or an object that can be converted to a string, ie has a toString() method.

|  |
| --- |
| **server.component.html** |
|  |

|  |
| --- |
| **server.component.html** |
|  |

### Property Binding

we add [disabled]="code\_here" to the button. code\_here is any TypeScript code that return a bool.

|  |
| --- |
| **servers.component.html** |
|  |

|  |
| --- |
| **servers.component.ts** |
|  |

The button will be disabled at first, then enable after two seconds is up:



### Property Binding vs String Interpolation

Here we use both techniques to display allowNewServer on our site

|  |
| --- |
| **servers.component.html** |
|  |



### Event Binding

the syntax for event binding is (event-name)="code-to-execute"

|  |
| --- |
| **servers.component.html** |
|  |

|  |
| --- |
| **servers.component.ts** |
|  |

state after clicking button:



### Bindable Properties and Events

How do you know to which Properties or Events of HTML Elements you may bind? You can basically bind to all Properties and Events - a good idea is to console.log()  the element you're interested in to see which properties and events it offers.

**Important**: For events, you don't bind to onclick but only to click (=> (click)).

The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for YOUR\_ELEMENT properties  or YOUR\_ELEMENT events  should yield nice results.

### Passing and Using Data with Event Binding

Here we add a text field with a label above it and bind it to a variable

|  |
| --- |
| **servers.component.html** |
|  |

|  |
| --- |
| **servers.component.ts** |
|  |

Note that the bound variable gets updated with each keystroke!



### Important: FormsModule is Required for Two-Way-Binding!

Important: For Two-Way-Binding (covered in the next lecture) to work, you need to enable the ngModel  directive. This is done by adding the FormsModule  to the imports[]  array in the AppModule.

You then also need to add the import from @angular/forms  in the app.module.ts file:

import { FormsModule } from '@angular/forms';

### Two-Way-Databinding

add FormsModule to the app:

|  |
| --- |
| **app.module.ts** |
|  |

|  |
| --- |
| **servers.component.html** |
|  |

|  |
| --- |
| **servers.component.ts** |
|  |

### Combining all Forms of Databinding





**Assignment 2: Practicing Databinding**

You learned a lot about Databinding! Time to practice it on your own. In this assignment, you're going to use the different forms of Databinding and see how you may use them in your app.



### Assignment 2 Solution

### Understanding Directives



### Using ngIf to Output Data Conditionally

ngIf is the Angular if directive. We can use this to rewrite our code





### Enhancing ngIf with an Else Condition

No we add an else clause that displays a message before the server is created





### Styling Elements Dynamically with ngStyle

Using ngStyle to dynamically style the

|  |
| --- |
| server.component.ts |
|  |

### Applying CSS Classes Dynamically with ngClass

For ngClass the keys are the CSS class names and the values are the conditions for whether the class should be applied or not.

|  |
| --- |
| server.component.html |
|  |

### Outputting Lists with ngFor

|  |
| --- |
| server.component.ts |
|  |

For ngClass the keys are the CSS class names and the values are the conditions for whether the class should be applied or not.

|  |
| --- |
| server.component.html |
|  |

**Assignment: Practicing Directives**

11 minutes to complete

20,747 student solutions

The last assignment for this course section. Practice what you learned about Directives and use the most common Directives on your own!

### Assignment Solution

### Getting the Index when using ngFor

\*ngFor=”let logItem of log; let i = index”

The vraible i will function as a counter

## Course Project - The Basics

### Project Introduction

### Planning the app



### Creating a New App Correctly

**MUST READ**

In the next lecture, we set up the course project.

Make sure, you do create that app by also adding the --no-strict flag to the ng new command - otherwise you will run into issues later on (**we'll still dive into that "Strict Mode" later** in the course of course, no worries)!

We'll also install the **Bootstrap CSS Framework** and in this course, we use **version 3** of the framework. Install it via npm install --save bootstrap@3  => The @3  is important!

Additionally, when using a project created with Angular CLI 6+ (check via ng v ), you'll have an angular.json  file instead of an .angular-cli.json  file. In that file, you still need to add Bootstrap to the styles[]  array as shown in the next video, but the path should be node\_modules/bootstrap/dist/css/bootstrap.min.css , **NOT** ../node\_modules/bootstrap/dist/css/bootstrap.min.css . **The leading ../  must not be included**.

Also see this lecture: I do show the complete setup process there: <https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/t/lecture/6655614/>

If you're facing any problems, please have a look at this very thorough thread by Jost: <https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/17862130#questions/10444944>

### Setting up the Application

from top level directory in git bash

cd ~/project-angular/

ng new course-app -–no-strict

output



open in VS Code and terminal

npm install --save bootstrap@3

output



open angular.json and add the bootstrap css file:



ng serve



### Creating the Components

ng generate component header

ng generate component recipes

ng generate component recipes/recipe-list

ng generate component recipes/recipe-list/recipe-item

ng generate component recipes/recipe-detail

ng generate component recipes/shopping-list

ng generate component recipes/shopping-list/shopping-edit

### Using the Components

|  |
| --- |
| **app.component.html** |
|  |

|  |
| --- |
| **shopping-list.component.html** |
|  |

|  |
| --- |
| **recipe-list.component.html** |
|  |

|  |
| --- |
| **recipes.component.html** |
|  |

### Adding a Navigation Bar

|  |
| --- |
| **header.component.html** |
|  |

|  |
| --- |
| **header.component.html** |
|  |

### Alternative Non-Collapsable Navigation Bar

The way we added it, the Navbar will collapse on smaller screens. Since we didn't implement a Hamburger menu, that means that there's no way of accessing our links on smaller screens.

You can either add such a menu on your own (see below), or you replace collapse navbar-collapse  with just navbar-default.

Adding a Hamburger Menu:

Alternatively, if you want to make the navigation bar responsive, please replace these lines in header.component.html:

<div class="navbar-header">

<a routerLink="/" class="navbar-brand">Recipe Book</a>

</div>

<div class="collapse navbar-collapse">

with these lines:

<div class="navbar-header">

<button type="button" class="navbar-toggle" (click)="collapsed = !collapsed">

<span class="icon-bar" \*ngFor="let iconBar of [1, 2, 3]"></span>

</button>

<a routerLink="/" class="navbar-brand">Recipe Book</a>

</div>

<div class="navbar-collapse" [class.collapse]="collapsed" (window:resize)="collapsed = true">

and add this line to header.component.ts:

1. collapsed = true;





### Creating a "Recipe" Mode

|  |
| --- |
| **recipe.model.ts** |
| export class Recipe {    public name: string;    public description: string;    public imagePath: string;    constructor(name: string, desc: string, imagePath: string) {      this.name = name;      this.imagePath = imagePath;    }  } |

### Adding Content to the Recipes Components

|  |
| --- |
| **recipe.model.ts** |
|  |

### Outputting a List of Recipes with ngFor

|  |
| --- |
| **recipe-list.component.ts** |
|  |



### Displaying Recipe Details

|  |
| --- |
| **recipe-detail.component.html** |
| <div class="row">    <div class="col-xs-12">      <img src="" alt="" class="img-responsive">    </div>  </div>  <div class="row">    <div class="col-xs-12">      <h1>Recipe Name</h1>    </div>  </div>  <div class="row">    <div class="col-xs-12">      <div class="btn-group">        <button type="button" class="btn btn-primary dropdown-toggle">          Manage Recipe <span class="caret"></span>        </button>        <ul class="dropdown-menu">          <li><a href="#">To Shopping List</a></li>          <li><a href="#">Edit Recipe</a></li>          <li><a href="#">Delet Recipe</a></li>        </ul>      </div>    </div>  </div>  <div class="row">    <div class="col-xs-12">      Description    </div>  </div>  <div class="row">    <div class="col-xs-12">      Ingredients    </div>  </div> |

### Working on the ShoppingListComponent

|  |
| --- |
| **shopping-list.component.html** |
| <div class="row">    <div class="col-xs-10">      <app-shopping-edit></app-shopping-edit>      <hr>      <ul class="list-group">        <a href="#" class="list-group-item" style="cursor: point;"></a>      </ul>    </div>  </div> |

### Creating an "Ingredient" Model

|  |
| --- |
| **ingredient.model.ts** |
| export class Ingredient {    constructor(public name: string, public amount: number) {}  } |

### Creating and Outputting the Shopping List

|  |
| --- |
| **shopping-list.component.html** |
| <div class="row">    <div class="col-xs-10">      <app-shopping-edit></app-shopping-edit>      <hr>      <ul class="list-group">        <a href="#" class="list-group-item" style="cursor: pointer;" \*ngFor="let ingredient of ingredients">        {{ ingredient.name }} ({{ ingredient.amount }})        </a>      </ul>    </div>  </div> |

|  |
| --- |
| **shopping-list.component.ts** |
| import { Component, OnInit } from '@angular/core';  import { Ingredient } from 'src/app/shared/ingredient.model';  @Component({    selector: 'app-shopping-list',    templateUrl: './shopping-list.component.html',    styleUrls: ['./shopping-list.component.css']  })  export class ShoppingListComponent implements OnInit {    ingredients: Ingredient[] = [      new Ingredient('Apples', 5),      new Ingredient('Tomatoes', 10),    ];    constructor() { }    ngOnInit(): void {    }  } |



### Adding a Shopping List Edit Section

|  |
| --- |
| **shopping-edit.component.html** |
| <div class="row">    <div class="col-xs-12">      <form >        <div class="row">          <div class="col-sm-5 form-group">            <label for="name">Name</label>            <input type="text" id="name" class="form-control">          </div>          <div class="col-sm-2 form-group">            <label for="amount">Amount</label>            <input type="number" id="amount" class="form-control">          </div>        </div>        <div class="row">          <div class="col-xs-12"></div>          <button class="btn btn-success" type="submit">Add</button>          <button class="btn btn-danger" type="button">Delete</button>          <button class="btn btn-primary" type="button">Clear</button>        </div>      </form>    </div>  </div> |



### Wrap Up & Next Steps

## Debugging

### Understanding Angular Error Messages

### Debugging Code in the Browser Using Sourcemaps

These sections are out of date compared to curretn Angular/webpack. My nbote sbelow are up to date.

1. download debugging.zip
2. unzip
3. open folder in VSCode
4. npm install
5. ng serve
6. open in browser



1. F12 to open Chrome debugger
2. click button and error occurs.
3. click highlighted link



line 9: servers is not initialized

1. right-click and select “Reveal in Sidebar”



## Course Project - Components & Databinding

### Module Introduction

1. download cmp-databinding-start.zip
2. unzip
3. open folder in VSCode
4. npm install
5. ng serve
6. open in browser



### Splitting Apps into Components

ng generate component cockpit

ng generate component server-element

move code into the new components as follows

|  |
| --- |
| **app.component.ts** |
|  |

|  |
| --- |
| **app. component.html** |
|  |

|  |
| --- |
| **cockpit.component.ts** |
|  |

|  |
| --- |
| **cockpit. component.html** |
|  |

|  |
| --- |
| **server-element.component.ts** |
|  |

|  |
| --- |
| **server-element. component.html** |
|  |

### Property & Event Binding Overview



### Binding to Custom Properties

|  |
| --- |
| **app.component.ts** |
|  |

|  |
| --- |
| **app. component.html** |
|  |

|  |
| --- |
| **server-element.component.ts** |
|  |

|  |
| --- |
| **server-element. component.html** |
|  |



### Assigning an Alias to Custom Properties

add alias as shown below

|  |
| --- |
| **server-element.component.ts** |
|  |

change corresponding element in the html from element to srvElement

|  |
| --- |
| **app. component.html** |
|  |



### Binding to Custom Events

Neither of these bugs produces an error that I could see.

I had an annoying bug. My code was not within the <app-cockpit> tag

      <app-cockpit>

        (serverCreated)="console.log('hello');onServerAdded($event)"

        (bluePrintCreated)="onBlueprintAdded($event)"

      </app-cockpit>

changed to

  <appcockpit

(serverCreated)="onServerAdded($event)"

(bluePrintCreated)="onBlueprintAdded($event)">

</app-cockpit>

A second annoying bug, a typo. bluePrintCreated in the HTML should be blueprintCreated

  <app-cockpit

  (serverCreated)="onServerAdded($event)"

  (blueprintCreated)="onBlueprintAdded($event)">

  </app-cockpit>

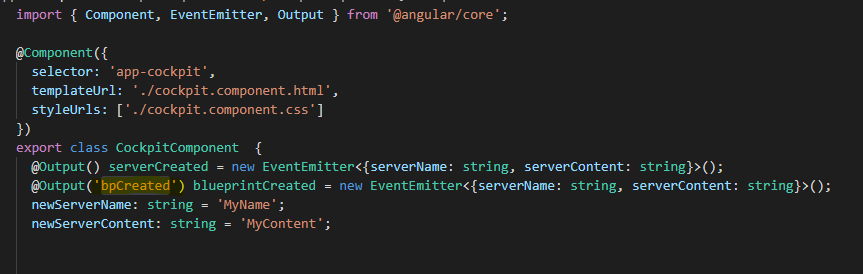
|  |
| --- |
| **app.component.html** |
|  |

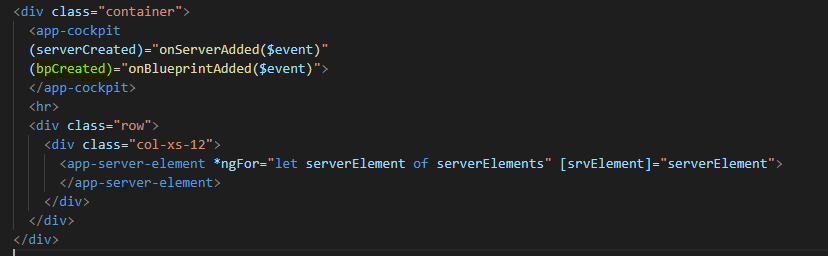
|  |
| --- |
| **app.component.ts** |
|  |

|  |
| --- |
| **cockpit.component.ts** |
| import { Component, EventEmitter, Output } from '@angular/core';  @Component({    selector: 'app-cockpit',    templateUrl: './cockpit.component.html',    styleUrls: ['./cockpit.component.css']  })  export class CockpitComponent  {    @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();    @Output() blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();    newServerName: string = 'MyName';    newServerContent: string = 'MyContent';    onAddServer() {      console.log(`onAddServer; newServerName=${this.newServerName} newServerContent=${this.newServerContent} `);      this.serverCreated.emit({        serverName: this.newServerName,        serverContent: this.newServerContent      });    }    onAddBlueprint() {      this.blueprintCreated.emit({        serverName: this.newServerName,        serverContent: this.newServerContent      });    }  } |

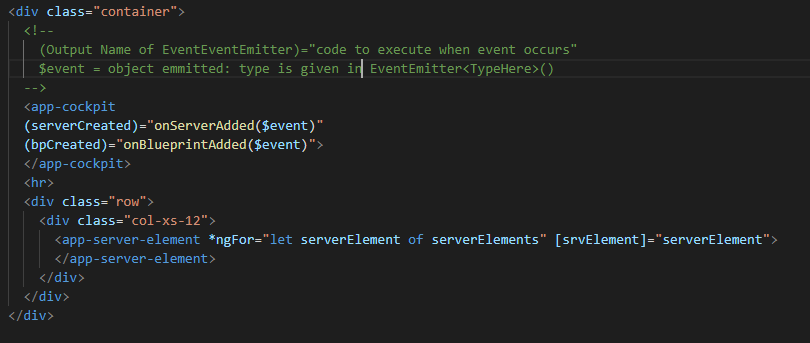
|  |
| --- |
| **cockpit.component.html** |
| <p>Add new Servers or blueprints!</p>  <label>Server Name</label>  <input type="text" class="form-control" [(ngModel)]="newServerName">  <label>Server Content</label>  <input type="text" class="form-control" [(ngModel)]="newServerContent">  <br>  <button class="btn btn-primary" (click)="onAddServer()">Add Server</button>  <button class="btn btn-primary" (click)="onAddBlueprint()">Add Server Blueprint</button> |

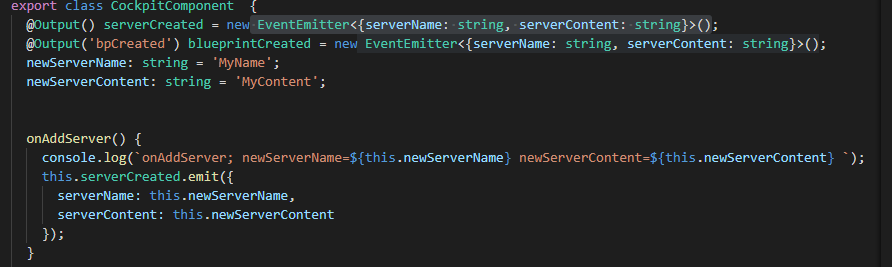
### Assigning an Alias to Custom Events





### Custom Property and Event Binding Summary

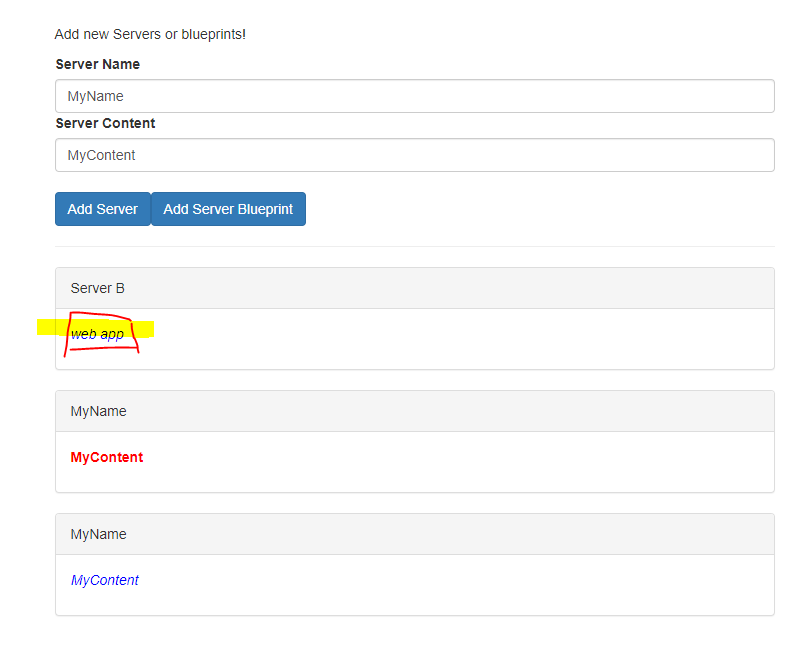




He mentions that there is second method for communication that will be covered later.

### Understanding View Encapsulation

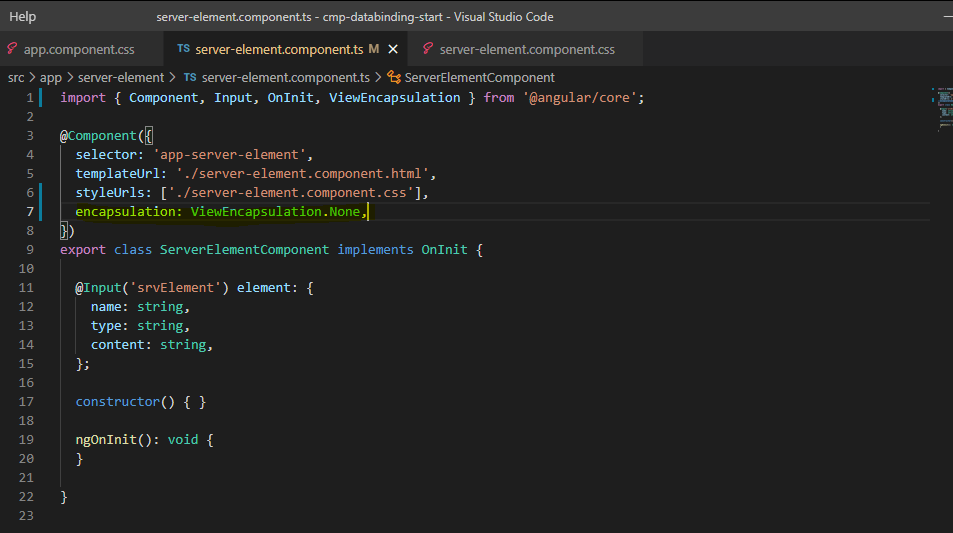
Angular uses different attribute names in the compiled css to give attributes to each component type.





### More on View Encapsulation

You can disable view encapsulation for a components css as follows:



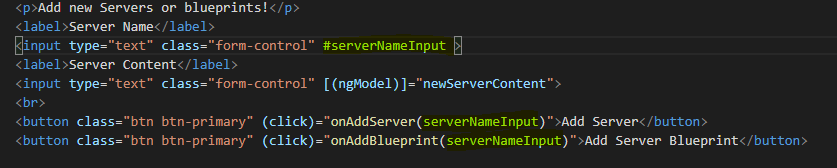
Not sure why you would do this since it makes the css code global!

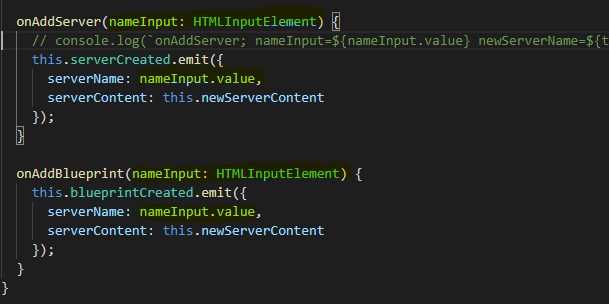
The default value is ViewEncapsulation.Emulated

You can also use ViewEncapsulation.Native, which uses the browsers method of encapsulation. But not all browsers support yet.

### Using Local References in Templates

Here we change from two-way binding to using Angular input for the server name





### @ViewChild() in Angular 8+

In **Angular 8+**, the @ViewChild() syntax which you'll see in the next lecture needs to be changed slightly:

Instead of:

@ViewChild('serverContentInput') serverContentInput: ElementRef;

use

@ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;

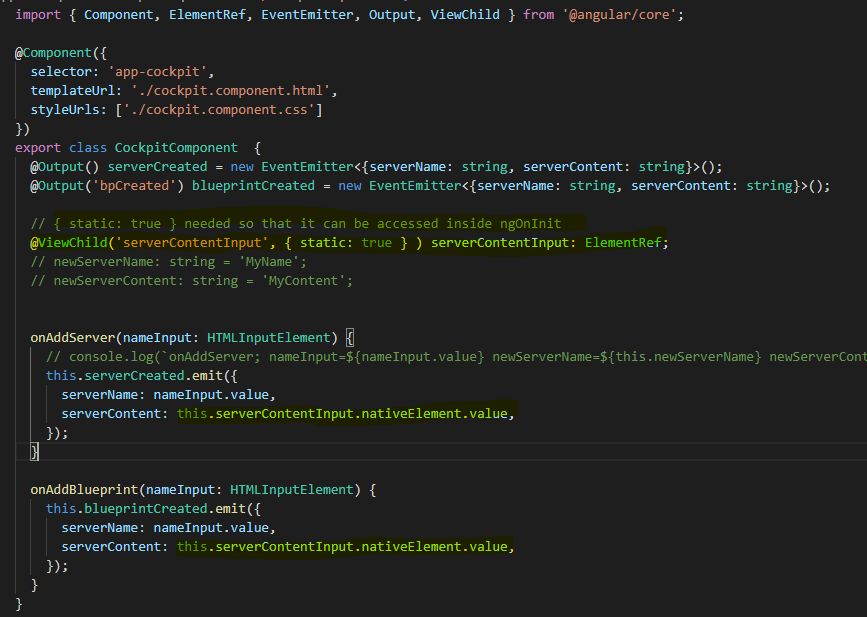
The same change (add { static: true } as a second argument) needs to be applied to ALL usages of @ViewChild() (and also @ContentChild() which you'll learn about later) IF you plan on accessing the selected element inside of ngOnInit().

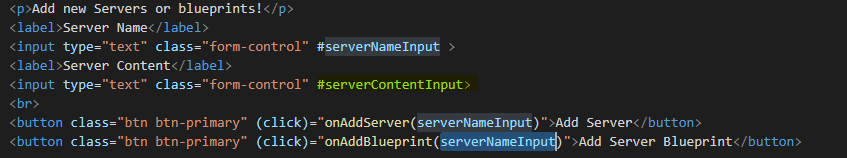
If you DON'T access the selected element in ngOnInit (but anywhere else in your component), set static: false instead!

If you're using Angular 9+, you only need to add { static: true } (if needed) but not { static: false }.

### Getting Access to the Template & DOM with @ViewChild

ViewChild is a second way to access values.





### Projecting Content into Components with ng-content

move

    <p>

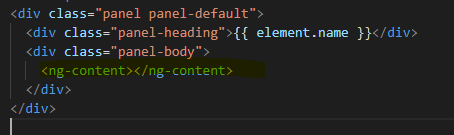
      <strong \*ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>

      <em \*ngIf="element.type === 'blueprint'">{{ element.content }}</em>

    </p>

from server-element.component.html to app.component.html

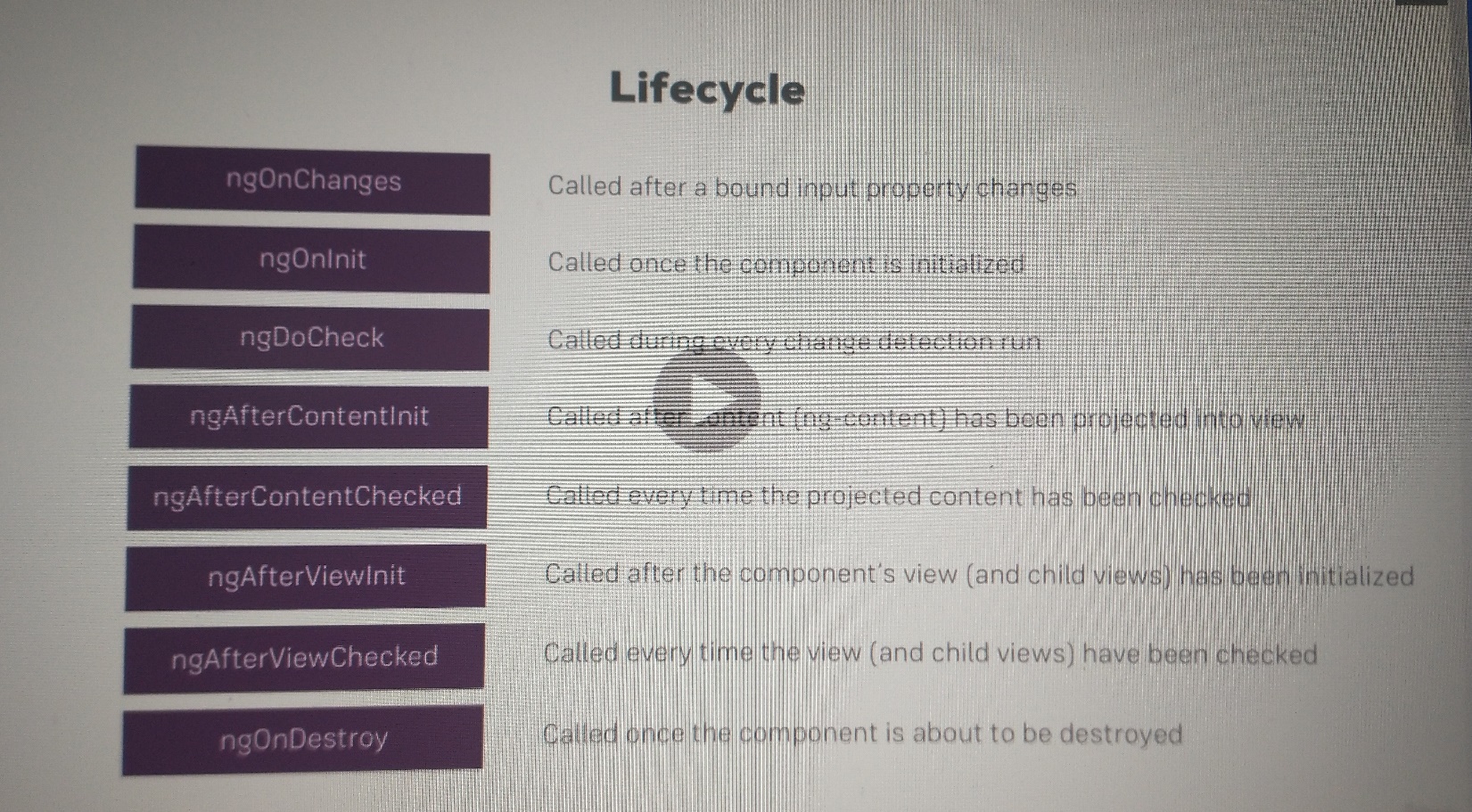
you also need to add <ng-content> so that the other component can use the internals of





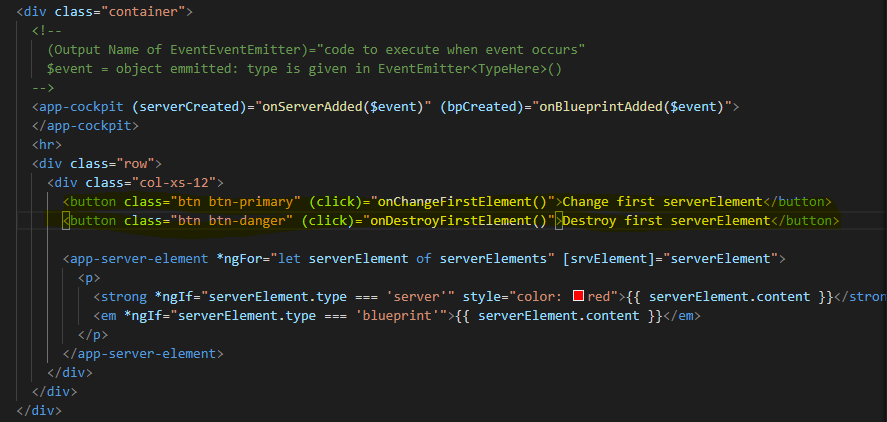
### Understanding the Component Lifecycle

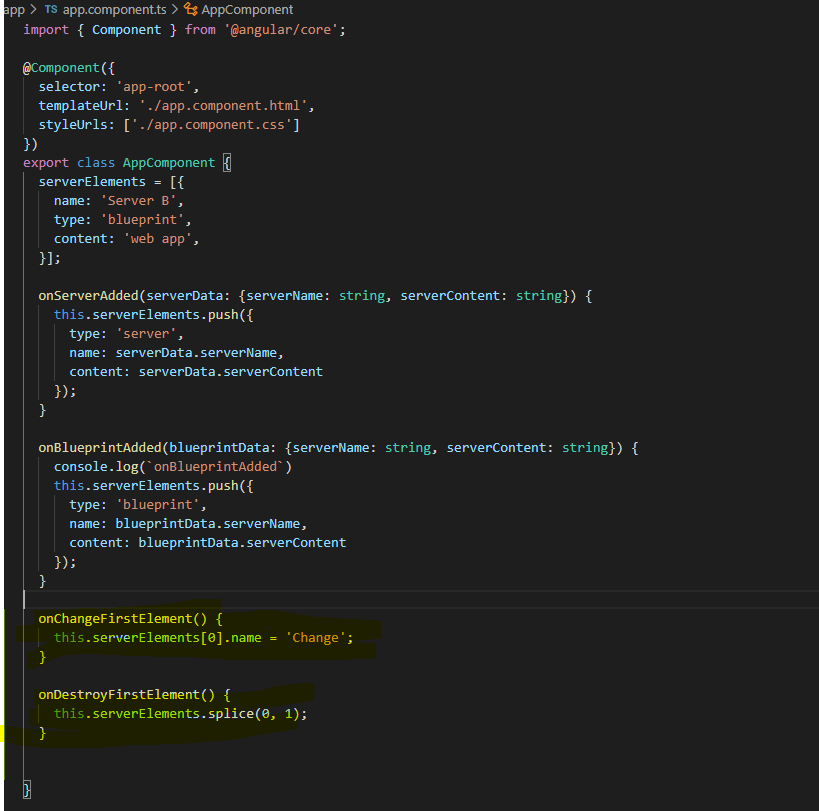
The following summarizes when various component methods are called.



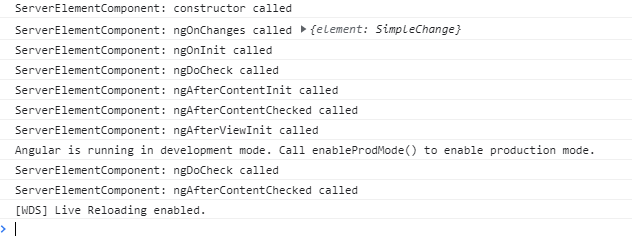
### Seeing Lifecycle Hooks in Action



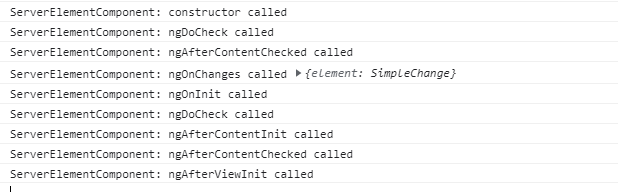




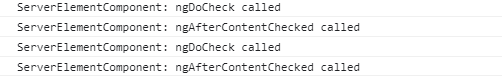
On **startup**



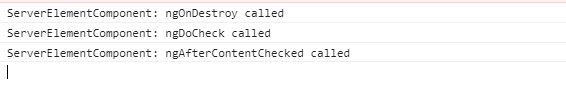
click **Add Server**



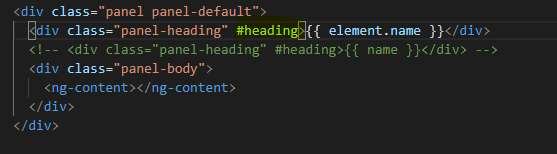
click **Change First ServerElement**

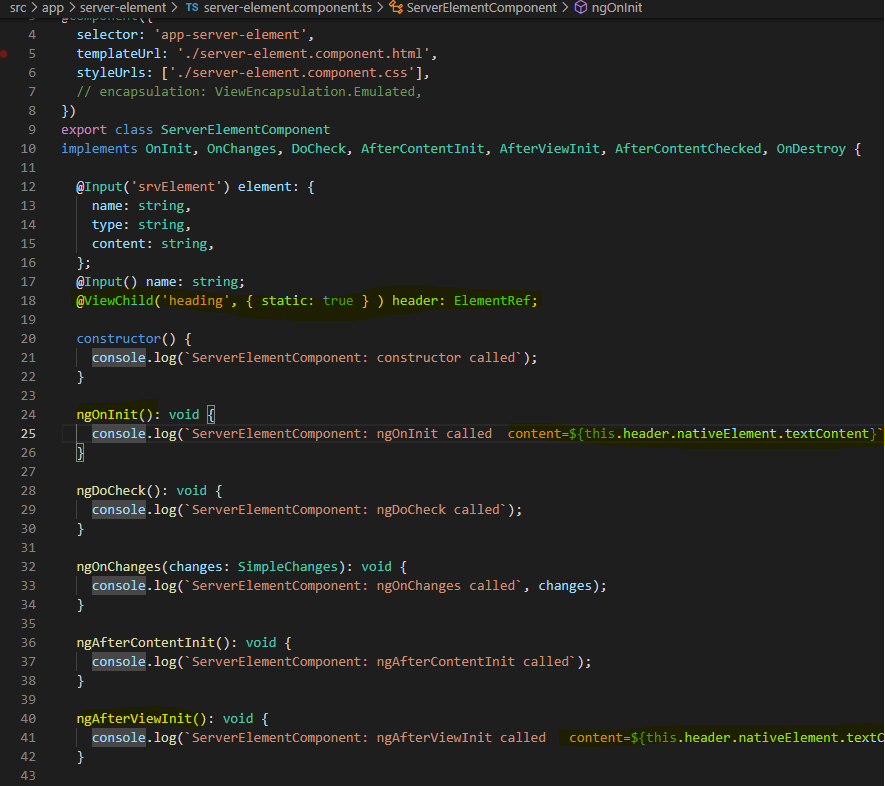


click **Remove First ServerElement**

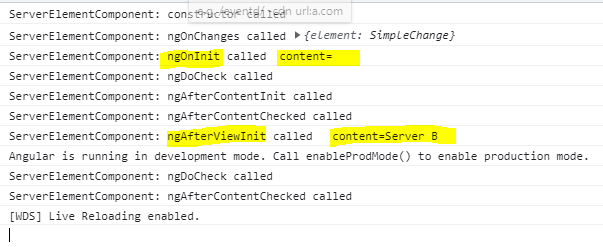


### Lifecycle Hooks and Template Access





note that content is empty on init, but filled after view init:



### @ContentChild() in Angular 8+

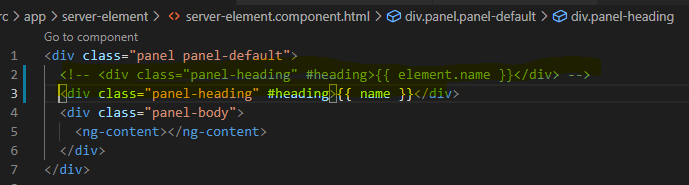
@ContentChild() in Angular 8+

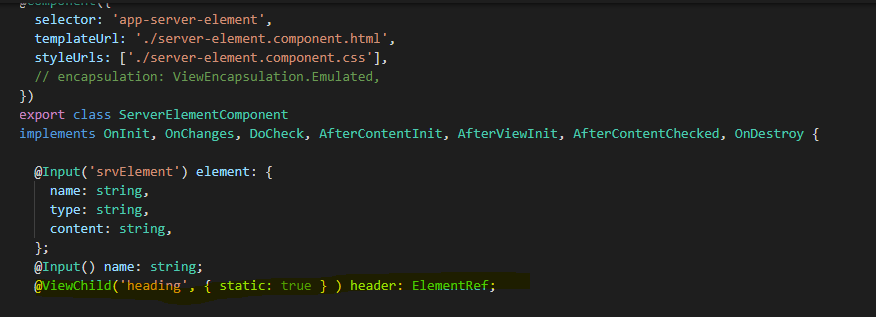
For **ContentChild**, the same adjustments as for **ViewChild** apply: <https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/14865241>

Here I fixed name input referenincg

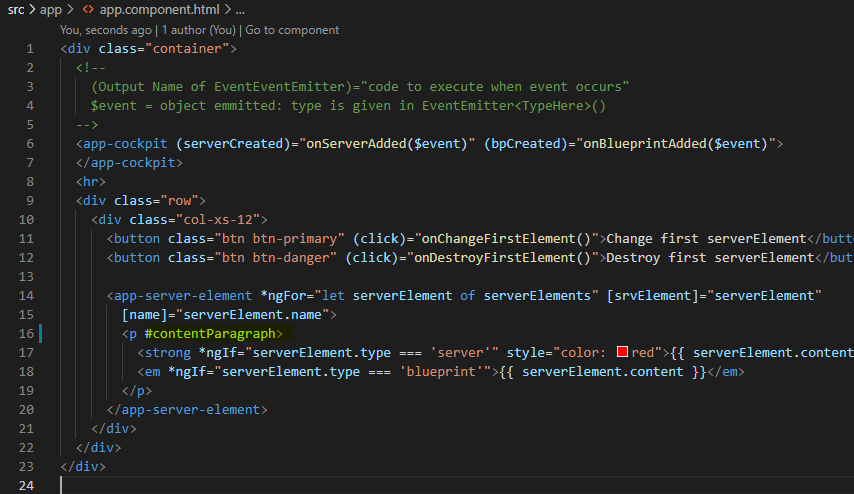
app.component.html



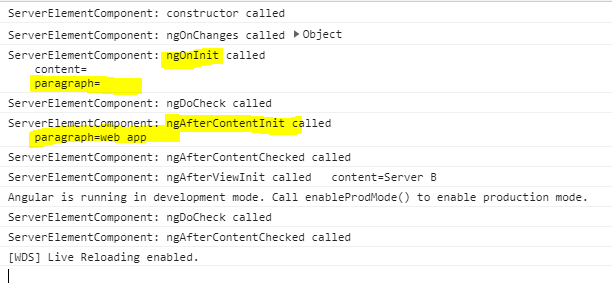




### Getting Access to ng-content with @ContentChild







### Wrap Up

Assignment 4: Practicing Property & Event Binding and View Encapsulation

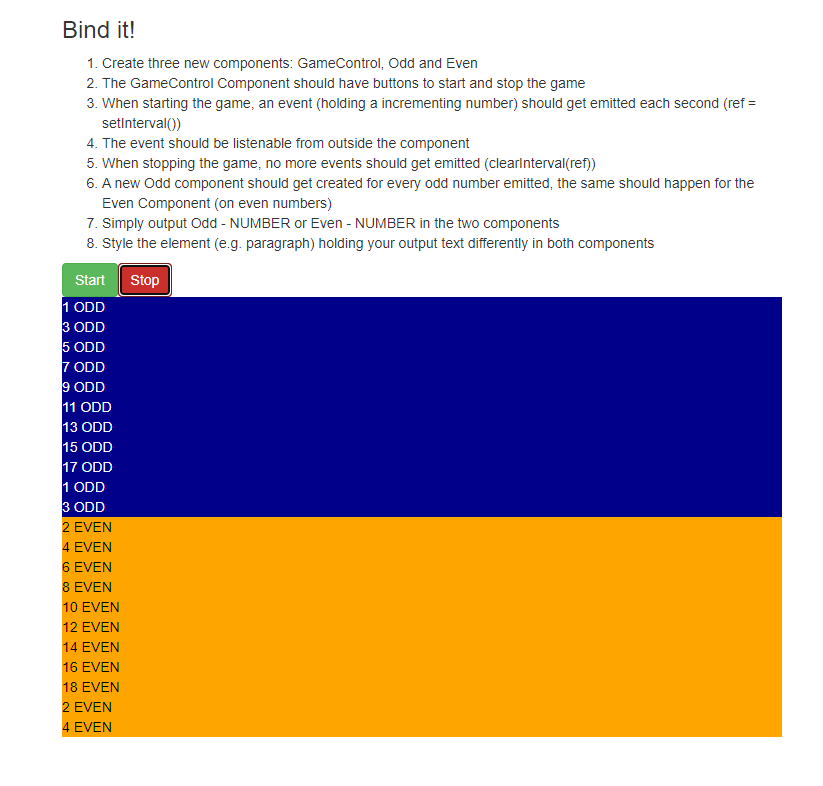
1. Create three new components: GameControl, Odd and Even
2. The GameControl Component should have buttons to start and stop the game
3. When starting the game, an event (holding a incrementing number) should get emitted each second (ref = setInterval())
4. The event should be listenable from outside the component
5. When stopping the game, no more events should get emitted (clearInterval(ref))
6. A new Odd component should get created for every odd number emitted, the same should happen for the Even Component (on even numbers)
7. Simply output Odd - NUMBER or Even - NUMBER in the two components
8. Style the element (e.g. paragraph) holding your output text differently in both compone

### [OPTIONAL] Assignment Solution

ng g c game-control

ng g c odd

ng g c even



|  |
| --- |
| **app.component.html** |
|  |

|  |
| --- |
| **app.component.ts** |
|  |

|  |
| --- |
| **game-control.component.html** |
|  |

|  |
| --- |
| **game-control.component.ts** |
|  |

|  |
| --- |
| **even.component.html** |
|  |

|  |
| --- |
| **even.component.css** |
|  |

|  |
| --- |
| **even.component.ts** |
|  |

|  |
| --- |
| **odd.component.html** |
|  |

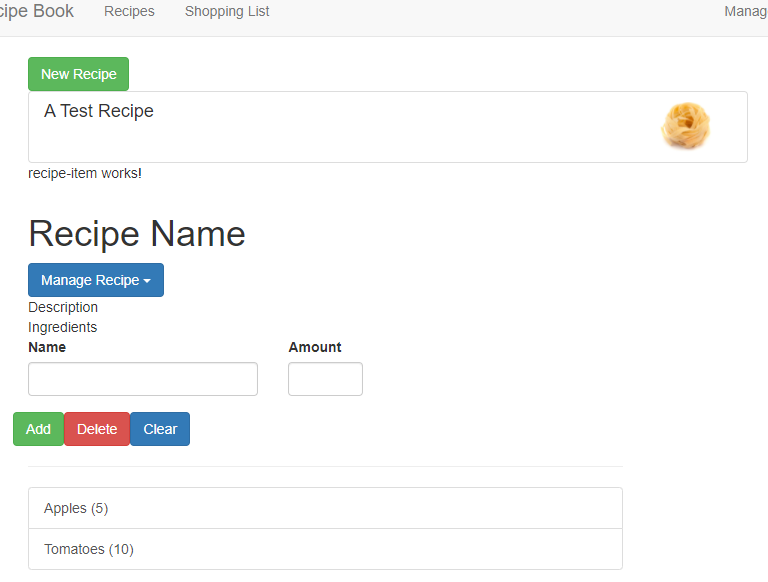
|  |
| --- |
| **odd.component.css** |
|  |

|  |
| --- |
| **odd.component.ts** |
|  |

## Course Project - Components & Databinding

### Introduction

back to our recipe project



### Adding Navigation with Event Binding and ngIf

Here won’t do routing, we just change the visibility of the components. Note that the user can’t bookmark!

add click for the two header items

|  |
| --- |
| **header.component.html** |
|  |

add handler function and Output

|  |
| --- |
| **header. component.ts** |
|  |

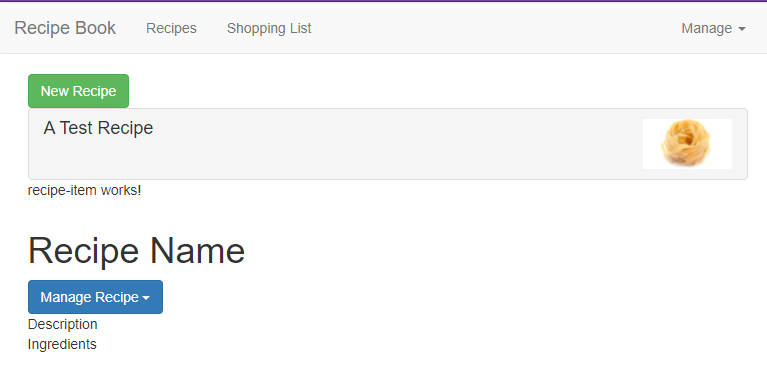
control visibility of components using the selectedFeature

|  |
| --- |
| **app.component.html** |
|  |

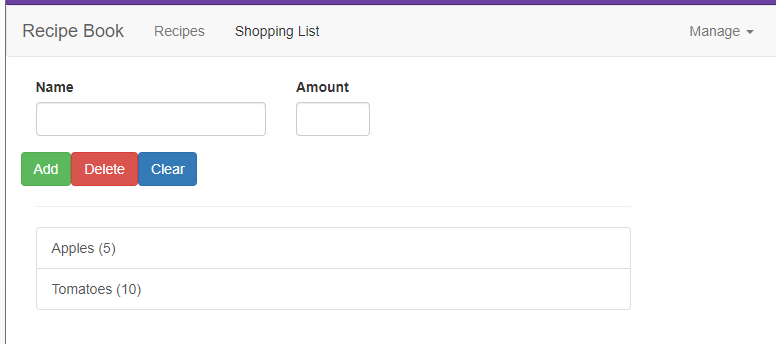
add handler function and state variable

|  |
| --- |
| **app. component.ts** |
|  |

click Recipes we get



click Shopping we get



### Passing Recipe Data with Property Binding

move all recipe html from recipe-list into recipe-item-component

|  |
| --- |
| **recipe-item. component.html** |
|  |

Nbow we add an input for the model data. Note that we are not creating storgae for the mode, that is done in the list.

|  |
| --- |
| **recipe-item. component.ts** |
|  |

now we loop inside the app-recipe-item tag and pass the model object

|  |
| --- |
| **recipe-list. component.html** |
|  |

no changes to ts file:

|  |
| --- |
| **recipe-list. component.ts** |
|  |

### Passing Data with Event and Property Binding (Combined)

### Make sure you have FormsModule added!

### Allowing the User to Add Ingredients to the Shopping List

## Directives Deep Dive

### Module Introduction

### ngFor and ngIf Recap

### ngClass and ngStyle Recap

### Creating a Basic Attribute Directive

### Using the Renderer to build a Better Attribute Directive

### More about the Renderer

### Using HostListener to Listen to Host Events

### Using HostBinding to Bind to Host Properties

### Binding to Directive Properties

### What Happens behind the Scenes on Structural Directives

### Building a Structural Directive

### Understanding ngSwitch

## Course Project – Directives

### Building and Using a Dropdown Directive

### Closing the Dropdown From Anywhere

## Using Services & Dependency Injection

### Module Introduction

### Why would you Need Services?

### Creating a Logging Service

### Injecting the Logging Service into Components

### Creating a Data Service

### Understanding the Hierarchical Injector

### How many Instances of Service Should It Be?

### Injecting Services into Services

### Using Services for Cross-Component Communication

### Services in Angular 6+

Assignment 5: Practicing Services

### [OPTIONAL] Assignment Solution

## Course Project - Services & Dependency Injection

### Introduction

### Setting up the Services

### Managing Recipes in a Recipe Service

### Using a Service for Cross-Component Communication

### Adding the Shopping List Service

### Using Services for Pushing Data from A to B

### Adding Ingredients to Recipes

### Passing Ingredients from Recipes to the Shopping List (via a Service)

### Module Introduction

### Why do we need a Router?

### Understanding the Example Project

### Setting up and Loading Routes

### Navigating with Router Links

### Understanding Navigation Paths

### Styling Active Router Links

### Navigating Programmatically

### Using Relative Paths in Programmatic Navigation

### Passing Parameters to Routes

### Fetching Route Parameters

### Fetching Route Parameters Reactively

### An Important Note about Route Observables

### Passing Query Parameters and Fragments

### Retrieving Query Parameters and Fragments

### Practicing and some Common Gotchas

### Setting up Child (Nested) Routes

### Using Query Parameters - Practice

### Configuring the Handling of Query Parameters

### Redirecting and Wildcard Routes

### Important: Redirection Path Matching

In our example, we didn't encounter any issues when we tried to redirect the user. But that's not always the case when adding redirections.

By default, Angular matches paths by prefix. That means, that the following route will match both /recipes  and just /

{ path: '', redirectTo: '/somewhere-else' }

Actually, Angular will give you an error here, because that's a common gotcha: This route will now **ALWAYS** redirect you! Why?

Since the default matching strategy is "prefix" , Angular checks if the path you entered in the URL does **start with the path** specified in the route. Of course every path starts with ''  (Important: That's no whitespace, it's simply "nothing").

To fix this behavior, you need to change the matching strategy to "full" :

{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }

Now, you only get redirected, if the full path is ''  (so only if you got NO other content in your path in this example).

### Outsourcing the Route Configuration

### An Introduction to Guards

### Protecting Routes with canActivate

### Protecting Child (Nested) Routes with canActivateChild

### Using a Fake Auth Service

### Controlling Navigation with canDeactivate

### Passing Static Data to a Route

### Resolving Dynamic Data with the resolve Guard

### Understanding Location Strategies

### Wrap Up

## Course Project – Routing

### Planning the General Structure

### Setting Up Routes

### Adding Navigation to the App

### Marking Active Routes

### Fixing Page Reload Issues

### Child Routes: Challenge

### Adding Child Routing Together

### Configuring Route Parameters

### Passing Dynamic Parameters to Links

### Styling Active Recipe Items

### Adding Editing Routes

### Retrieving Route Parameters

### Programmatic Navigation to the Edit Page

### One Note about Route Observables

### Project Cleanup

## Understanding Observables

### Module Introduction

Completed thru Chap 84