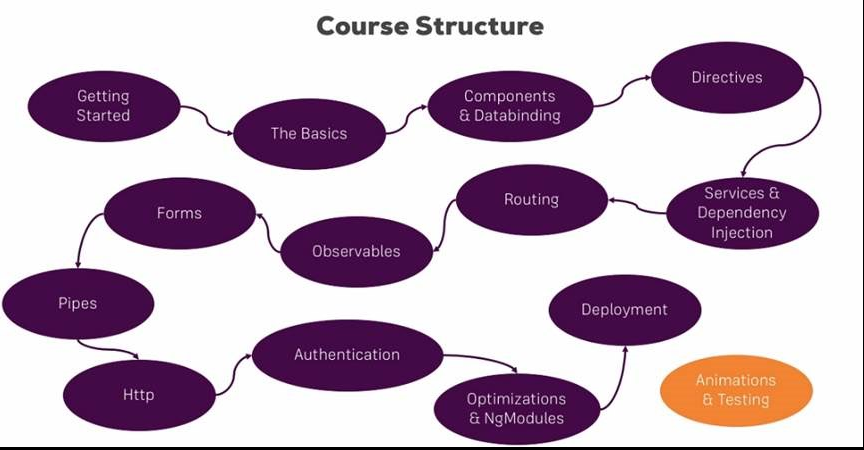
My node git credentials

Username- hemanthkaravalla



Specifically you will learn:

* + Which architecture Angular uses
  + How to use TypeScript to write Angular applications
  + All about directives and components, including the creation of custom directives/ components
  + How databinding works
  + All about routing and handling navigation
  + What Pipes are and how to use them
  + How to access the Web (e.g. RESTful servers)
  + What dependency injection is and how to use it
  + How to use Modules in Angular
  + How to optimize your (bigger) Angular Application
  + We will build a major project in this course
  + and much more!

Course started on may 22nd,should complete around june 18 may june 30th.

Start date -May 22nd

End date -June 18 th

Duration - 30 days

**1 week- 6:30 hrs. -revision on Thursday**

Basics, components and data binding, Directives, services, dependency injection, Type Script introduction, Course Roundup, Angular changes and new features.

**2 week-6:30 hrs. -revision on Thursday**

Routing, observables, forms **.**

**3rd week-6.30 hr -revision on thursday**

Pipes, Making HTTP requests, authentication and route protecting in angular app, using angular modules and optimizing apps, deploying and http client, Custom project and work flow setup.

**4th week-6.30 hr -revision on Thursday**

Working with NGRX in our project, Angular Universal, Angular Animations, Adding office capabilities with service workers, A basic introduction to unit testing in Angular apps

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| topic | No of lectures | days | Time required  (30 hrs) | Examples practice time | Things to revise |
|  |  | |  | | --- | | **1 week- 300min -6:30 hrs**  Basics,components and data binding,Directives,services,dependency injection, Type Script introduction, Course Roundup, Angular changes and new features. | |  |  | -Typess of data binding and syntaxes. |
| Getting started | 10 |  | 36 min |  |  |
| The basics | 33 | Wednesday | 1.53 hrs |  |  |
| Course project basics | 18 | Thursday | 1.03 hrs |  |  |
| debugging | 3 |  | 12.09min |  |  |
| Components and data binding deep dive | 20 | Friday | 1.24 hrs |  |  |
| Course project component and data binding | 6 | Saturday | 31 min |  |  |
| Directives deep dive | 11 | Saturday | 47 min |  |  |
| Project directives | 1 | Sunday | 6.25min |  |  |
| Using services and dependency injection | 12 | Sunday | 43 min |  |  |
| Course project services and dependency injection | 8 | sunday | 30.48 min |  |  |
|  |  | |  | | --- | | 2nd week - 6 :30HRS  Routing,observables,forms | |  |  |  |
| Changing pages with routing | 31 | Wednesday, Thursday | 2.18 hrs |  |  |
| Course project Routing | 15 | Friday | 45min |  |  |
| Understanding observables | 11 | Saturday | 38min |  |  |
| Course project observables | 1 | Saturday | 4 min |  |  |
| Handling forms in angular app | 34 | Sunday | 2.02 hrs |  |  |
| Course project forms | 22 | Monday and Tuesday | 1.14 hrs |  |  |
|  |  | |  | | --- | | 3 rd week - 6 :30HRS  Pipes,Making HTTP requests,authentication and route protecting in angular app,using angular modules and optimizing apps,deploying and http client | |  |  |  |
| Using pipes to transform output | 10 | Wednesday | 37 min |  |  |
| Making HTTP request | 14 | Thursday | 40 min |  |  |
| Course project Http | 5 | Thursday | 21 min |  |  |
| Authentication and route protecting in angular app | 15 | Friday | 45 min |  |  |
| Using angular modules and optimizing apps | 25 | Saturday | 1.27 hrs |  |  |
| Deploying an angular app | 4 | Sunday | 11.49 min |  |  |
| **Bonous**-The http Client | 14 | sunday | 52min |  |  |
|  |  | |  | | --- | | 4TH WEEK – 6 .30HRS | |  |  |  |
| Working with NGRX in our project | 49 | Wednesday,thrusday,Friday | 3.41 hrs |  |  |
| Angular Universal | 9 | Saturday | 27 min |  |  |
| Angular Animations | 12 | Saturday | 39.22 min |  |  |
| Adding office capabilities with service workers | 5 | Sunday | 27 min |  |  |
| A basic introduction to unit testing in Angular apps | 11 | Sunday | 45.19 |  |  |
|  |  | Topics remaining adjusted in every week |  |  |  |
| Angular changes and new features | 3 | 1 WEEK -Monday | 35 |  |  |
| Course Roundup | 2 | 1 WEEK -Monday | 2 min |  |  |
| Custom project and work flow setup | 13 | 3 RD WEEK -Monday | 51 min |  |  |
| Type Script introduction | 7 | 1 –WEEK –Tuesday and overall recall | 26 min |  |  |

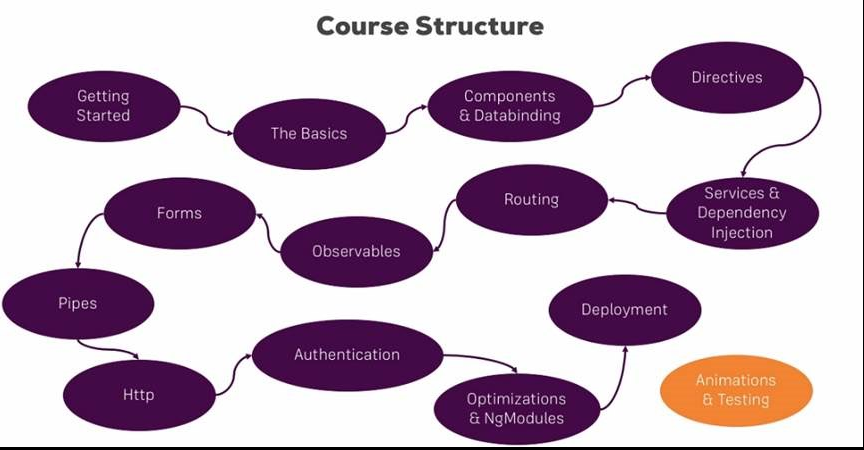
Introduction

* 1. Angular Introduction
  2. Introduction to SPA framework
  3. Node Installation
  4. AngularJS vs Angular Comparison

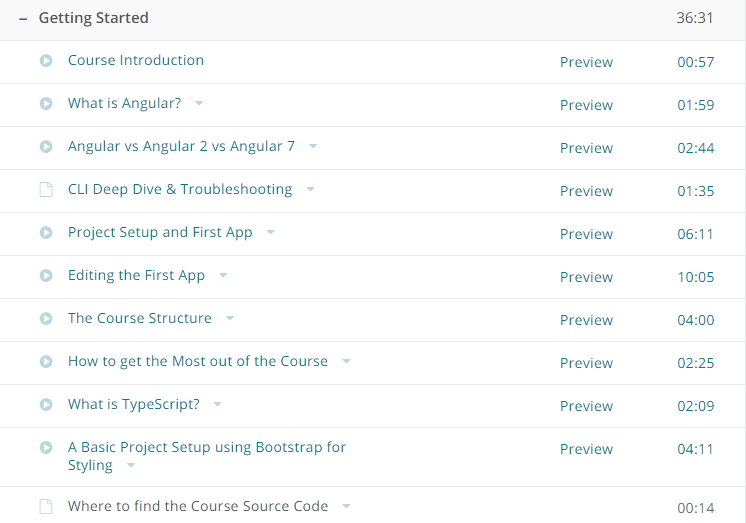
1. TypeScript
   1. Introduction to Type Script
   2. TypeScript compiler
   3. TypeScript Data Types
   4. TypeScript Statements
   5. TypeScript functions and Classes
   6. Modules and Namespaces
   7. Decorators
   8. Generics
   9. Example codes
2. Angular Project setup
   1. Introduction to Angular/CLI
   2. Angular Project Structure
3. Module and Component
   1. Module
      1. Module Introduction
      2. Module Decorator
      3. Import and Export of Modules
      4. Example
   2. Component
      1. Component Introduction
      2. Component Decorator
      3. Template
      4. Typescript class
      5. Example
4. Databinding
   1. String Interpolation
   2. Property
   3. Event
   4. Two Way Data Binding
5. Component Communication
   1. Introduction to Component Hierarchy
   2. @Input Decorator
   3. @Output Decorator
   4. Event Emitter
6. Dependency Injection
   1. Introduction to DI
   2. Introduction to Services
   3. Component communication using Services
7. Directives
   1. Introduction to Directives
   2. Built-in-Directives
   3. Attribute Directives
   4. Structural Directives
8. Custom Directives
   1. @Directive decorator
   2. Custom Attribute Directives
   3. Custom Structural Directives
   4. @Input and @Output usage
   5. @HostListner
   6. Example Code
9. Lifecycle Hooks
   1. Component Lifecycle Hooks Overview
   2. Lifecycle hooks Sequence
10. Pipes
    1. Pipes introduction
    2. Built-In-Pipes
    3. Custom Pipes
    4. Angular-pipes
    5. Custom pipes Example
11. Forms
    1. Introduction to Forms
    2. NgForms
    3. Reactive Forms
       1. Forms Controls
       2. Form Group
       3. Form Array
       4. Validators

12.4 Example Code

1. Pages with Routing
   1. Introduction to Routing
   2. Setting up and Loading Routing
   3. Navigating with Router Links
   4. Working with Route Parameters
   5. Router Outlet
2. Observables
   1. Introduction to Observables
   2. Reactive Programing
3. Http
   1. Introduction to Http
   2. CURD Operations
   3. GET
   4. POST
4. HttpClient
5. Build and Optimization
   1. ng build
   2. ng build vs ng serve
   3. AOT
   4. JIT
   5. Build Optimizer
6. Debugging
   1. Understanding angular messages
   2. Debugging code using Augury
7. Angular Best Practices



36min -



-Angular is a framework which allows you to create reactive, single page applications

Files you need to have a knowledge while learning Angular js

1. **app.component.html**
2. **app.component.ts**
3. **app.module.ts**
4. **index.html**
5. **main.ts**

**app component.html**

**<div class="container">**

**<div class="row">**

**<div class="col-xs-12">**

**<h3>I'm in the AppComponent!</h3>**

**<hr>**

**<app-servers></app-servers>**

**<!--<div app-servers></div>-->**

**<!--<div class="app-servers"></div>-->**

**</div>**

**</div>**

**</div>**

1. **app.component.ts**

**import { Component } from '@angular/core';**

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

// styleUrls: ['./app.component.css']

styles: [`

h3 {

color: dodgerblue;

}

`]

})

export class AppComponent {

}

**app.module.ts**

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

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

import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { ServerComponent } from './server/server.component';

import { ServersComponent } from './servers/servers.component';

@NgModule({

declarations: [

AppComponent,

ServerComponent,

ServersComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**index.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>MyFirstApp</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html>

Main.ts file

**import { enableProdMode } from '@angular/core';**

**import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';**

**import { AppModule } from './app/app.module';**

**import { environment } from './environments/environment';**

**if (environment.production) {**

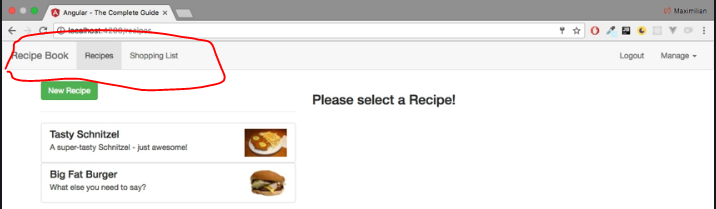
**enableProdMode();**

**}**

**platformBrowserDynamic().bootstrapModule(AppModule)**

**.catch(err => console.log(err));**

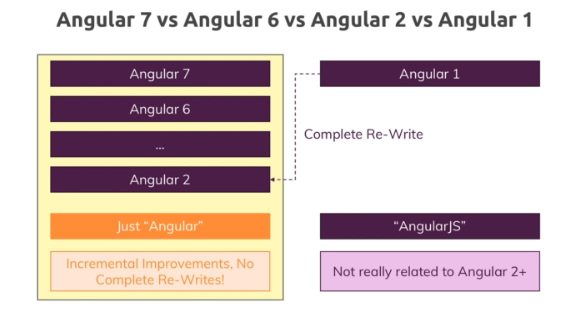
1)app



you can see that we seem to visit different pages,but in the end, our page never changes.

* It's only one HTML file and a bunch of JavaScript code we got from the server and everything which you see here, every change, is rendered in the browser.
* JavaScript is much faster than having to reach out to a server for every page change and for every new,piece of data you want to display. Therefore, this approach allows you to create web applications, which look and feel almost like mobile applications; very fast!
* Well, JavaScript changes the DOM, changes whatever is displayed here (in the browser), by changing the HTML code during runtime,(so to say). That is why you never see the refresh icon on the top-left spin; because we're only changing

->the currently-loaded page. You can even see that if you inspect the source code of a page like this.That is the HTML fileand as you can see, it doesn't seem to contain the content you are seeing on this page.We only have one single HTML element which doesn't seem to be a built-in one (a native one), but that's Angular doing its job.



-complete re-write of angular1

- When it came to using browser side javascript to reach rendered a Dom to update the dom at runtime and,therefore provide highly interactive user experiences without reloading the page angular or was a complete

-angular 2 and it later versions started in 2016

-

Creating first project in Angular

npm i -g @angular/cli

ng new projectname ex-ng new my-app

Basic project setup using Bootstrap for styling

Downloading npm install –save bootstrap@3

but to be able to use it, we also need to make Angular aware of this styling package we want to use and we do that in one of the config files

(the most important one actually) the angular.json file. This configures the CLI project.

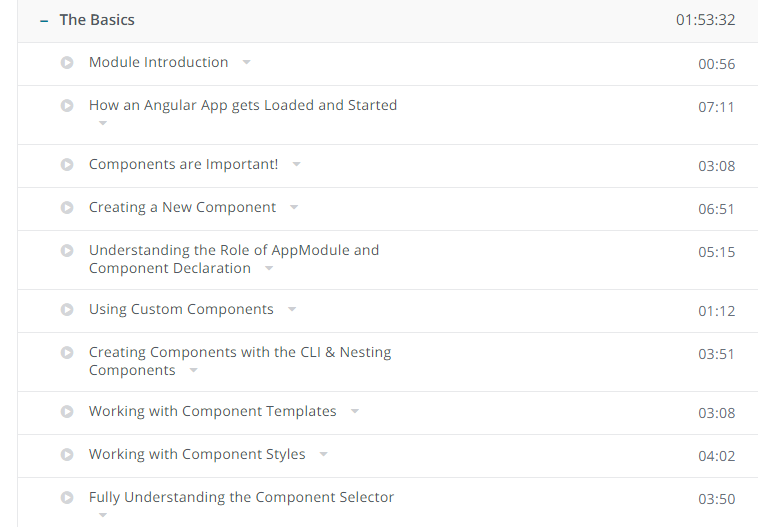


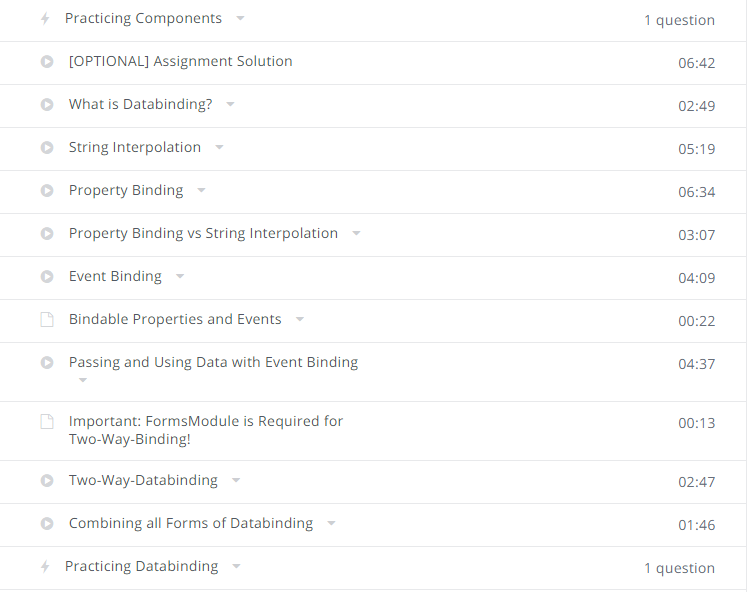
-styles.css is a file you can use to define some global styles you want to use application-wide.

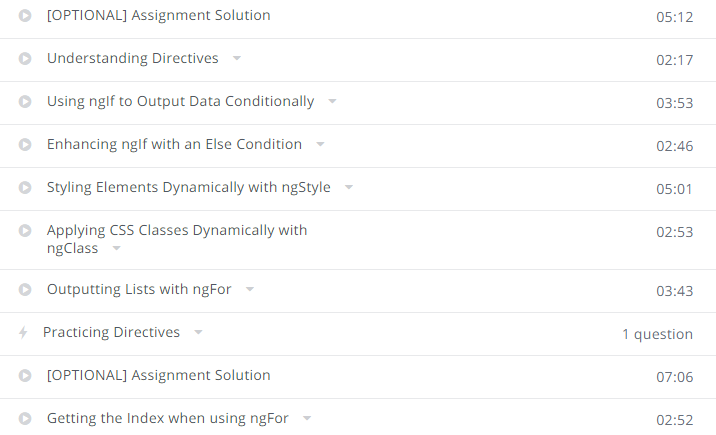
-But if you access the Elements tab in the Developer Tools, you should see that in the head section there

are two styles imports and the first one should be Bootstrap v3.3.7.

Now, this is the import you need there and with that, everything we do in the course will look correct.







Basics

Module Introduction

-how angular app gets started

-By the end of this section, you will understand how you can build your basic Angular application, what it then does and what you need to change to reach a different result

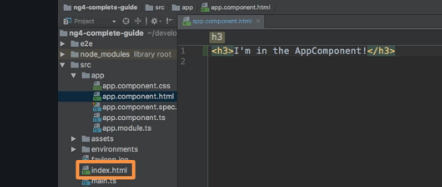
How angular app get started and loaded

-this is where our development

server hosted by the CLI or spun up by the CLI will host our Angular application.

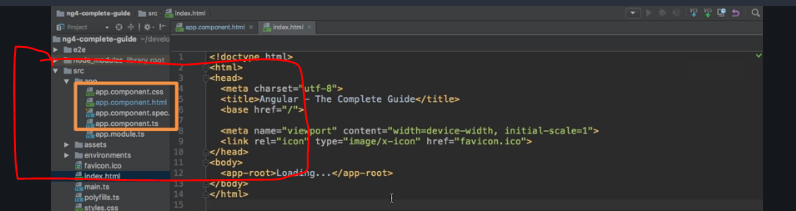
- the index.html file here is served by the server and remember that I told you that Angular

is a framework which allows you to create single page application,this is the single page which is served, the index.html file.



the CLI created one for us,

the root component of our application, the component which will tie together our whole application in



look at the app.component.ts file, the TypeScript file here.

Here you can see that we have this @component decorator, this seems to be important but more importantly

right now, you'll see that there, we have this selector property which assigns a string as a value and this string holds app-root. Now this clearly is the same text as in our index.html file

and this actually is the information Angular needed to be able to replace this part here in this index.html

**how is Angular triggered?**

how is it kicked off to actually run over our body here of this index.html file?

And the answer is in the final index.html file, getting served in the browser and we can verify

this by inspecting the source code here,



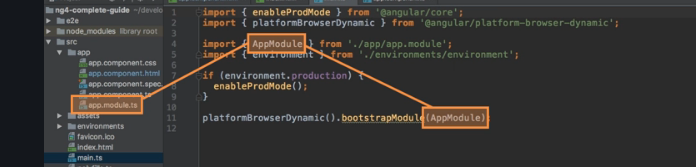
we got a couple of script imports at the end.These are injected by the CLI automatically,so that is why we don't see it here in the raw index.html file, here we don't have any script imports but whenever this **ng serve process rebuilds**

**our project, it will create bundles,Javascript script bundles and automatically add the right imports in the index.html file, a little convenience functionality for us.**

So in the final file, these script imports here are present and these script imports will contain our own code too.

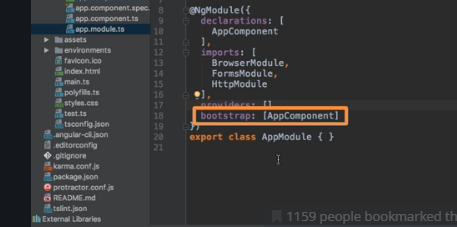
So these script files are **therefore executed and they're actually the first code to be executed,and that is just something you have to keep in mind, is the code we write in our main.ts file,that is why it's called main,**

this is the first code which gets executed.



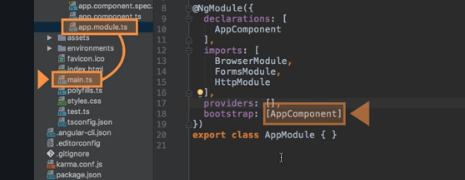
this line, this now bootstraps starts our Angular application by passing an app

module to this method and app module refers to this file here.



**here we get this bootstrap array which basically lists all the components**

**which should be known to Angular at the point of time it analyzes our index.html file and here**

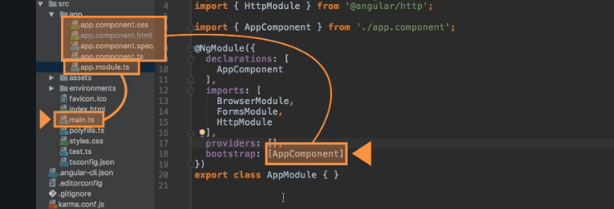


we bootstrap an Angular application and we pass this module as an argument. In this module, we tell Angular

hey there is this app component which you know when you try to start yourself and Angular

now analyzes the app component, reads the set up we pass here and therefore knows this selector, app-root

and now Angular is able to handle app-root in the index.html file



**So Angular gets started, this main.ts file gets started, there**

**we bootstrap an Angular application and we pass this module as an argument. In this module, we tell Angular**

**hey there is this app component which you know when you try to start yourself and Angular**

**now analyzes the app component, reads the set up we pass here and therefore knows this selector, app-root**

**and now Angular is able to handle app-root in the index.html file and it knows**

**all right this is the selector I know, you told me that I should know it because it was listed in this**

**bootstrap array in the app module, this component.**

**Last section overview**

In the last lecture, we had a close look at what happens at the point of time we visit localhost:4200

here in the browser.

We understood that the index.html file is served, that it will contain a bunch of scripts here which get executed which then basically start the Angular app, the Angular app gets the important information,that it should know, the app component that it should analyze it with that information the Angular code is able to parse this here, this app-root component here

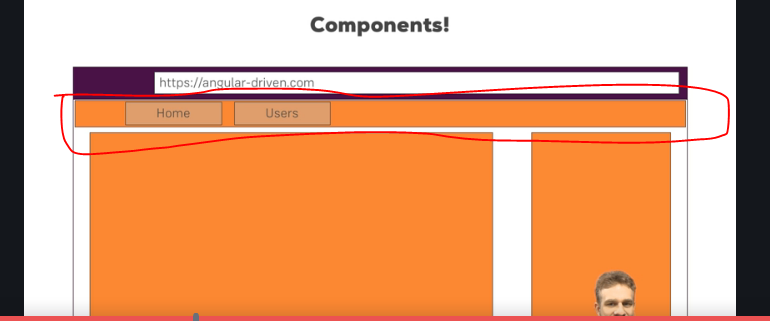
understand it and insert our Angular application at this point and that is why we don't see loading...

which would still be in the source code of this page as you can see

but instead why we see this because Angular overwrites this at runtime because that is what it does,

Components are important

So this root component, this app component will be the component where we later nest or add our other components too,



-Header may be a component

-home and users may be other components

each component has its own template, its own HTML code, maybe its own styling

and more importantly also, its own business logic and this is the great benefit.

It allows you to split up your complex application, your complex webpage into reusable parts,

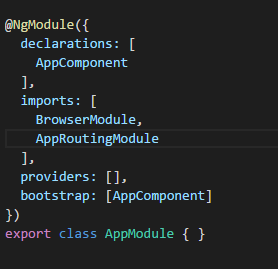
you may use a component more than once and that allows you to easily replicate that business logic,

replicate that styling or in general, make a finely controlled piece in your application without having

to crunch everything into one single script file, one single HTML file, instead

it's very easy to update, very easy to exchange and again re-usable.

Creating a new component

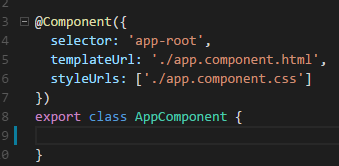
-in appmodule.ts we will mention in bootstrap array that our root component is this.Ask angular to bootstrap the application with this.

-if we want to add new component,it will be added in appcomponent.html and not appmodule.ts.

-A good practice is having a foldername = componentname.

Decorators

-ts feature that helps in enhancing elements that you use in code,ex -@component



**-selector –by which name you will be using in code**

**-templateurl-url which html template need to use**

**Stylesurl- url which style file template need to use**

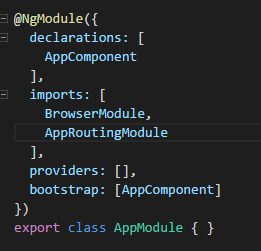
Understanding the role of appmodule and component

-appcomponents are used in angular to build web pages.

**-appmodules helps to bind components to your application package.**

-we will say this component is part of our modules.

@ngmodule



Imports adds other components into our module.

Using custom components

**emmet**-one of the plugin which is used to write the html code much faster.

Working with components template

-template we can write code.

-We can inline your impliments:

Working with componts styles

-external & internal for template.

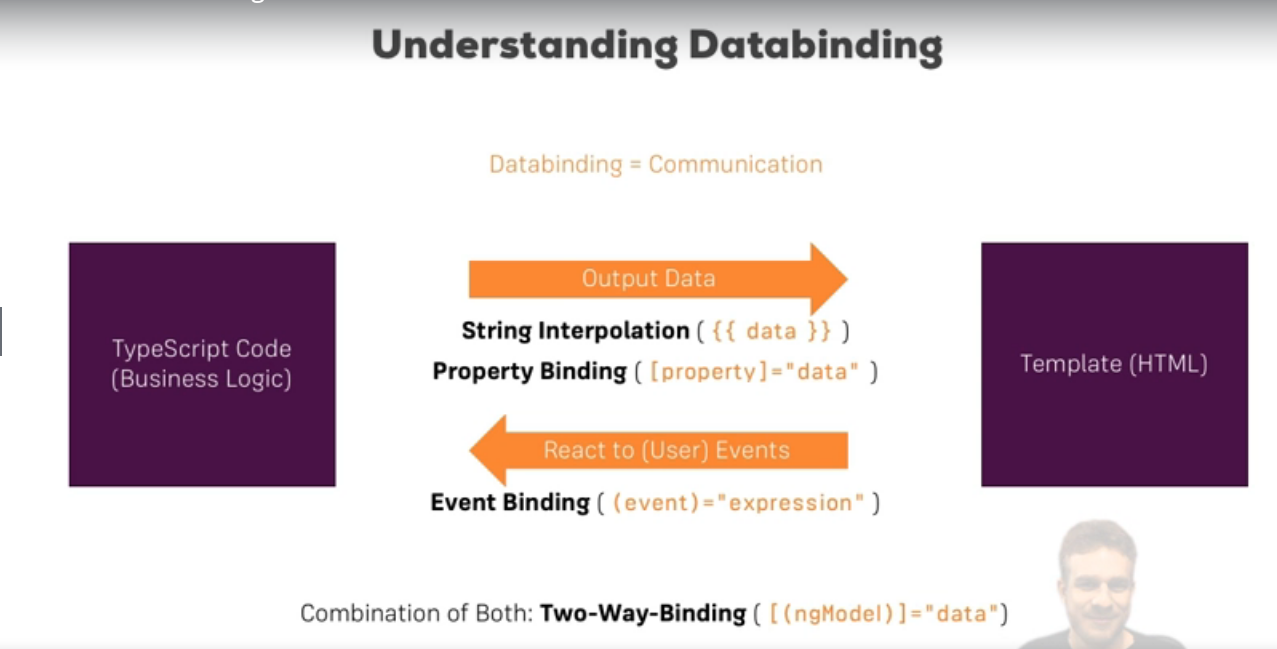
-same way we can do it for styling.

Fully understanding the component selector

-Angular selector by attribute also like example selector:’[app server]’

Now by <div app-server>

Data Binding



What is data binding?

- You could basically translate data binding with communication. Communication between your TypeScript code of your component, your business logic and the template.

We get different ways of communication now

-  for example we want to output data from our TypeScript code

in the HTML code in the template. We can use string interpolation for this,

- some property name or some expression in between or property binding.

syntax basically uses these strange square brackets around HTML attributes

Sometimes though, the other direction is interesting too,

- So if we click a button, we maybe want to trigger something in our TypeScript code,

so now we need the other direction and we can get this other direction,

we can react to user events with event binding. So we can bind to for example a click event to execute

- and we also have one additional form of data binding where we combine both directions, two-way data binding,

where we are able to react events and output something at the same time

- STRING Interprolation

-server.compoent.html



Scenario where he want to display server with id (dynamic content) and status is (active)?

Server component.ts

-we will write the type script code.



-**Use case 2-**

you could also for example here for server simply hardcode a string in there.

any expression which canbe resolved to a string in the end,that's the only condition for a string interpolation syntax here.

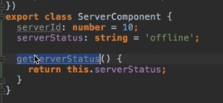
So whatever you have between the curly braces, in the end it somehow has to return a string,

so you could call a method here which returns a string in the end. The only other restriction is you

can't write multi-line expressions here,



So yes, you have to get a string in the end or something which can be converted to a string to reallybe correct here.



Calling a method

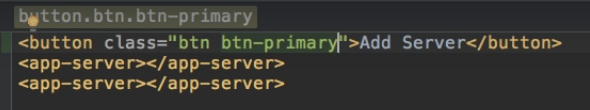


25.Property Binding

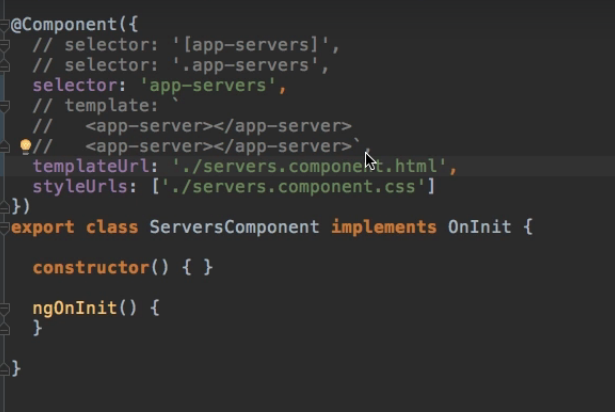
Senario- adding a server when button is clicked?

Server-component.html file

-I have to add a button



In server-cpmponent.ts file



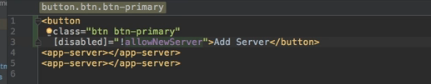
So I will add a new property in my TypeScript code here which I'll name allowNewServer and set it

to false.



Here we're binding to this disabled property, this native element property,

we're binding this to our own TypeScript property here and the convenient thing is and this is what Angular isall about, that this will update dynamically.



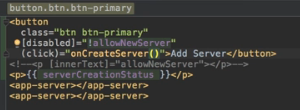
Property Binding vs String Binding

When to use Propety Binding and String Binding

Well basically if you want to output something in your template, print some text to it, use string interpolation.

if you want to change some property, be that of a HTML element or as you will later learn, of a directive or a component,typically use property binding.

Event Binding





Two way Data Binding

we use it therefore by combining the syntaxes, square brackets and within these, parentheses.

Now here, we have to use a special directive and we will learn more about directives in a second, @NgModel.

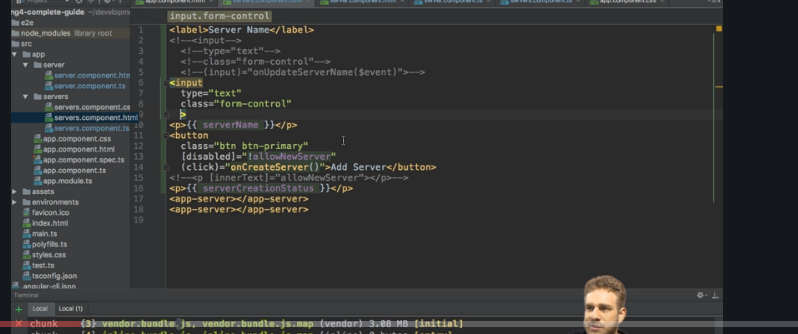
Now we can set this equal to some property defined in our TypeScript code,

so of to the server name for example. This set up will do the following,

it will trigger on the input event and update the value of server name in our component automatically.

On the other hand, since it is two-way binding, it will also update the value of the input element

if we change server name somewhere else and I can demonstrate this by going back to the TypeScript code



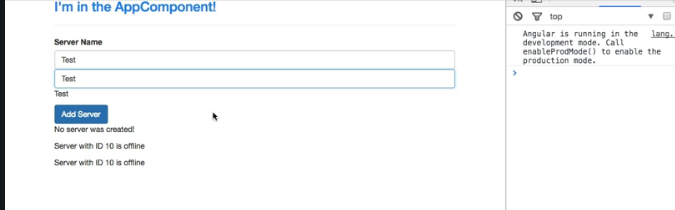


This set up will do the following,

it will trigger on the input event and update the value of server name in our component automatically.

On the other hand, since it is two-way binding, it will also update the value of the input element

if we change server name somewhere else and I can demonstrate this by going back to the TypeScript code



Directives

What are directives?

Directives are instructions in the DOM

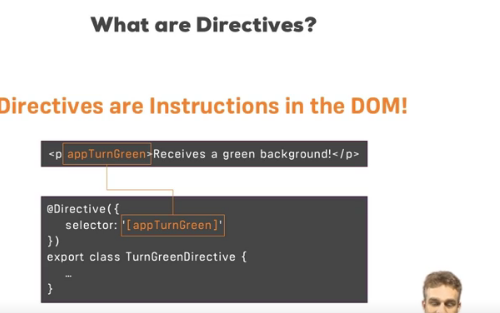
and we actually already use directives without knowing it,components are kind of such instructions in the DOM.

Once we place the selector of our component somewhere in our templates,at this point of time we're instructing Angular to add the content of our component template and the

business logic in our TypeScript code in this place where we use the selector.

This was our instruction, Angular

please add our component in this place and indeed components are directives but directives with a template,



this directive might simply color the text green you could say. So Angular would

find this instruction, here we would have defined our directive with the directive decorator to inform

Angular that this class holds a directive

and there, we might have the logic to turn this green.

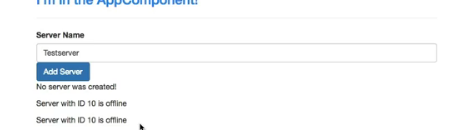
Using ngif to output data conditionally

Scenario-Now let's say one thing we want to do is we only want to show this server was created text here,

there is no need to have this

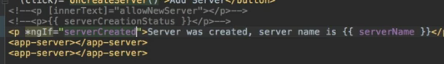
no server was created text to be displayed.

So it would be nice if we could conditionally show this message



-ngIf. The star is required because ngIf is a structural directive which means it changes the structure of our DOM, it either adds this element or it doesn't add it.

- For ngIf,this has to beany expression returning true or false,deciding whether this should be added or not.



Now with this if we save this, what you see is that there is no text

but if I name this Testserver 2 and click the button, now the text is added here

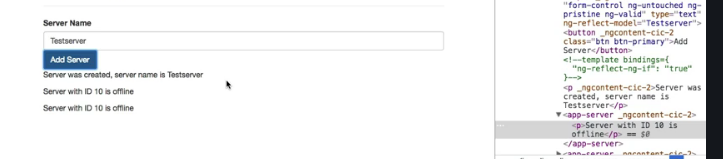
and the interesting thing is if I reload the app and we inspect our DOM here, you'll see here that's our app server

component, above that is the button

but if I click this button, a new element was entered here,



the paragraph and here you see kind of the hook which Angular created to know where this should be entered.



But the important thing is it's really added or removed to or from the DOM,

it's not there all the time, it's not hidden, it's just not there

which is super important to understand.

Enhancing ngif with a else condition-

-Sometimes you have not only the if condition but also an else condition,

so here we might want to say server was created, server name is something with ngIf but we could also add an else block where we then want to say no server was created.

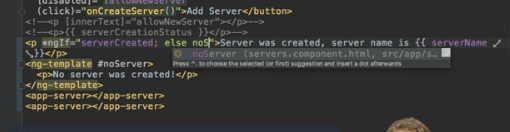


Scenario:-

-but imagine that we do want to show that alternative,

so no server was created could be the text here and now we want to show this paragraph only if ngIf is

not true or this condition, server created, is not true.



We can achieve this by placing a local reference on this element here and local reference is something

I will dive deeper into in the understanding components and data binding section of this course.

For now let's think of it as a marker,

so I will name this noserver,

any name you like,

noServer

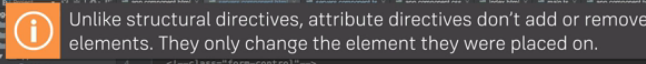
with a capital s

and now we need to change the paragraph here to ng-template. That is a component, that directive shipping

with Angular which you can use to mark places in the DOM



Styling Elements Dynamically with ngstyle

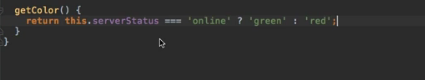


This ngStyle property expects to get a Javascript object

and here, we define key-value pairs of the style name as the key and the value of the style as the

value. So for example, we could bind background-color



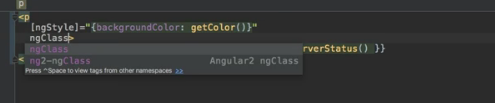


Applying css classes Dynamically

 -ngStyle allows us to dynamically assign a style,

now I will also add ngClass here and where ngStyle allowed us to change the CSS style itself,

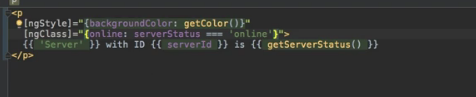
ngClass allows us to dynamically add or remove CSS classes.





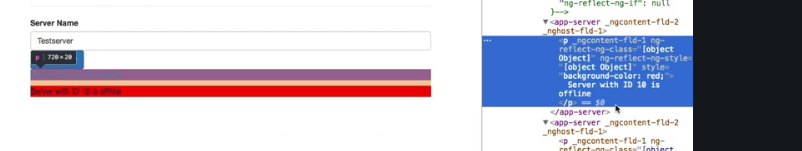
ngClass object works like this.We also have key-value pairs,

the keys are the CSS class names and the values are the conditions determining whether this class should be attached or not.



both are offline so none of the two should have the online CSS class added to it

and indeed, I can't find the class on any of the two.

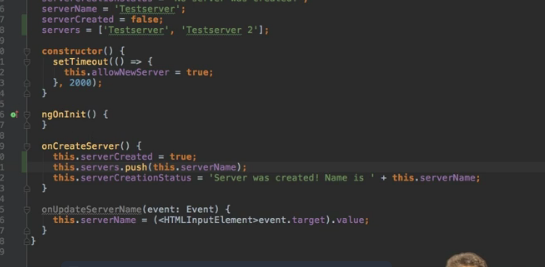


Outputting List with ngFor

Started with a scenario

Scenario-you are adding servers ,but you are not storing it in list?

ngFor



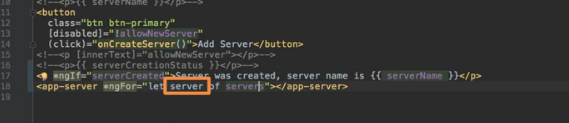
The ngFor base syntax looks like this,

we define a temporary variable for inside the loop with let, give it any name you like, like server for

example and then of servers. Servers here is the property we defined in the TypeScript file and this will

now loop through all elements in this array and assign the individual element to this dynamic server

variable.

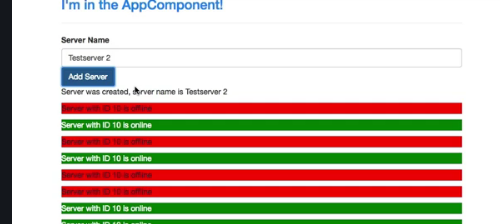


Of course the content of the individual server is still static because we can't pass the data like the

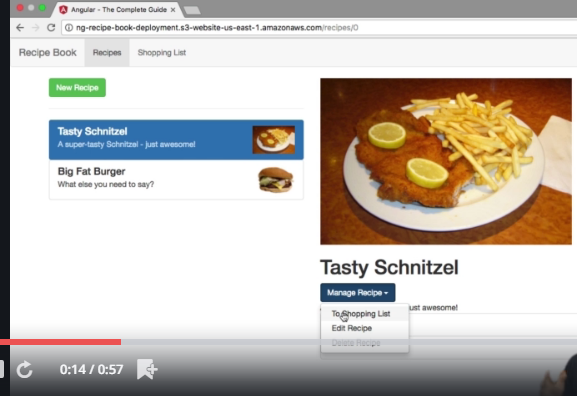
server name to that component but that is something we will learn in the next course section when we

dive deeper into components and we will learn how we can create our own properties on components

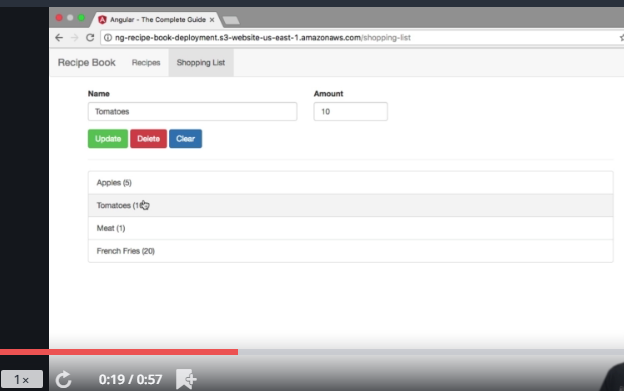
we can then set from outside the feature which would be great to have here.



Project overview



We are going to build a application with a list of Recipes, ShoppingList



My goal in this project which we will build throughout the course, to which we will come back after the individual

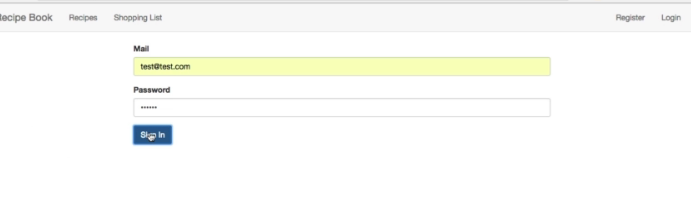
sections of this course, is to build a recipe book and shopping list app,

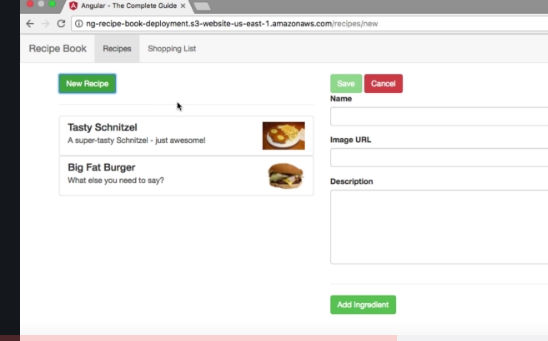
you saw it in the intro video of the course.

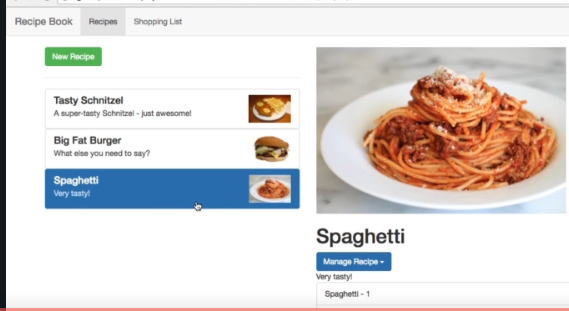
Basically there, we're going to have two sections, shopping list and recipe book and we will be able to

manage our recipes, view them in detail and also to manage our shopping list and even push ingredients

from a recipe directly to the shopping list.



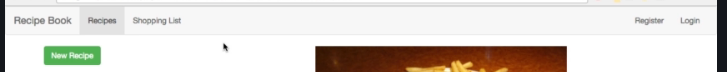


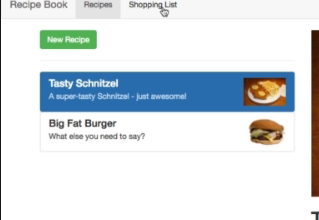


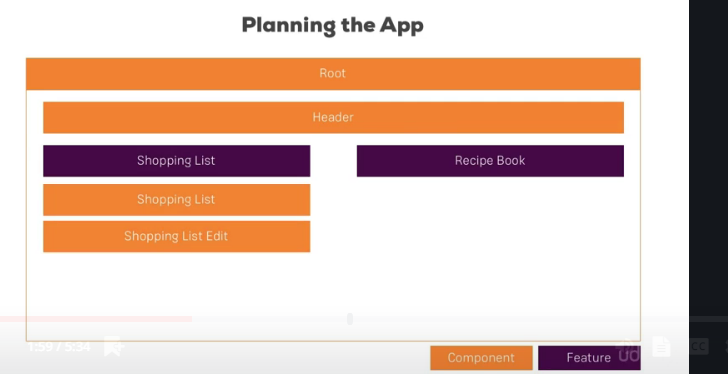


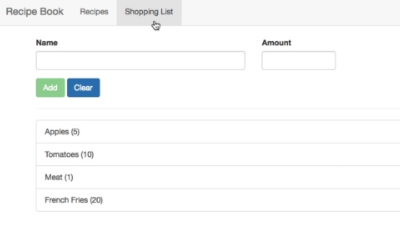
Planning the APP

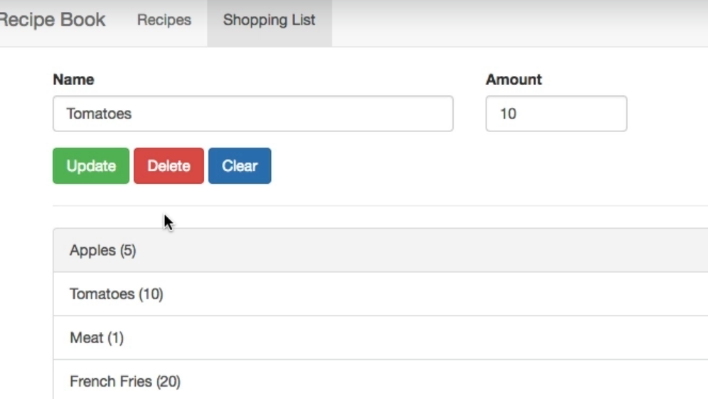
Steps in planning a app

* a root component holding our overall application, the app component, every Angular app has this.
* hen since we have two sections, it probably makes sense
* to have some kind of header component where we can navigate between these two sections.
* 
* Of course you could also hardcode the headers simply into the app component but since it will contain its
* own business logic, in the end it will trigger a routing action later and we will also add a dropdown



* there in the end which allows us to store our recipes on the server and fetch them from there.
* So since we will have that logic attached to the header, outsourcing it into its own component makes,sense so that we don't have to put all the logic into our root component which really mainly should only,be responsible for holding our overall app.
* 





let's start with the shopping list. Here I guess it makes sense to have the overall shopping-list component,

remember the purple box is only the feature so we need a component holding our shopping list and maybe in this shopping-list component

- we also want to have a shopping list edit part which allows us to add new items, so an input field and a button because that again has its own logic,

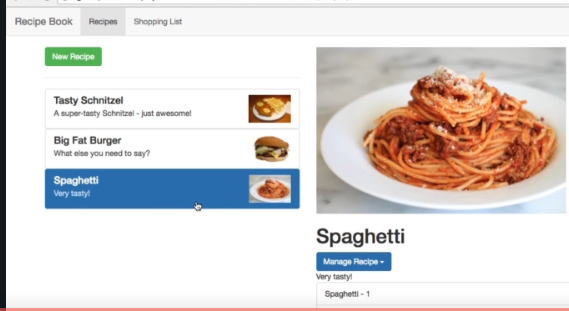
- So if we have an overall list component, mainly holding the data of the list and the edit component

being responsible for managing this data kind of, though the edit component will be nested inside the list component.



 a recipe-list component which

shows us a list of all our recipes. Here we might later also put each individual recipe into its own item



Here we might later also put each individual recipe into its own

item

-though since it holds a little bit more information than just one line of HTML code and a recipe-detail

area would be great too. So an area where once we select a recipe, we can see informations about that.

-So a list with items and then the details section which displays the details for the currently

selected item.

-sketch out here, we're really having each component focus on one main topic, displaying a list, displaying

information about a single item, displaying detail information about a selected item and so on.

which models will be use in this app?

mean which data. be clear about the data you're going to use and put it into its own class, so that you have your own

type you can use later on and that you have a clear interface or definition of what your data looks

like so that you can easily have your components talk with each other,

- So besides the ingredients which probably is a very simple model, only controlling the name and amount

maybe, we also will need a model for the recipe which contains things like title, the description, the ingredients

and so on.

Setting up the application

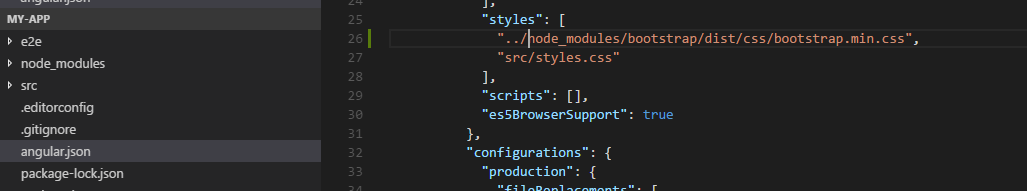
- now we also need to inform the CLI that Bootstrap should be included in our final bundle it creates

for us because of course, the CLI bundles all our script files and style files and by default, it would

not include Bootstrap. So to inform the CLI, we should go to the .angular-cli.json file and here

-  So to inform the CLI, we should go to the .angular-cli.json file and here

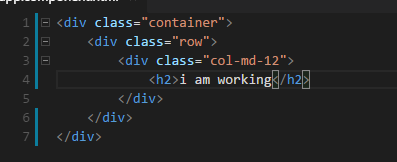
in styles, in this styles array, we can add any global stylesheets we want to add to our whole project.



-To be precise, it would even be better to target the minified version which is a file also lying in that

folder,

so let's add .min between bootstrap and .css, so this will already give us the minified code

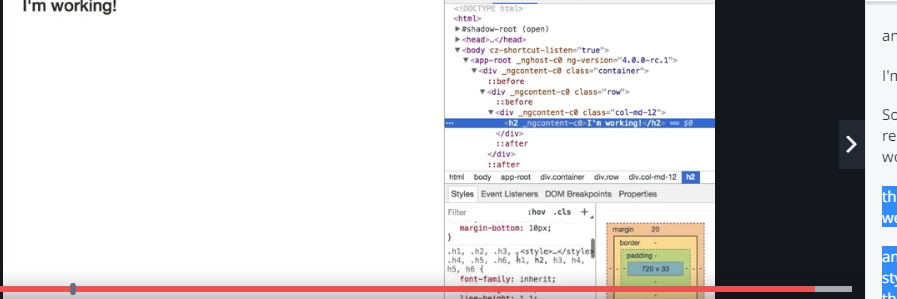


- plugin called emmet which allows me to just type dot claas name and hit tab to autocomplete it.

this looks like Bootstrap is working too, we can see those on the font size

and also if we inspect this here in the styles, all these styles here and down there

are coming from Bootstrap.



Creating the components

Now as you learned, a component is simply just a TypeScript class, so let's name it HeaderComponent here and right now, that would not be recognized as a component by Angular.

instead here we have to add the @component decorator.

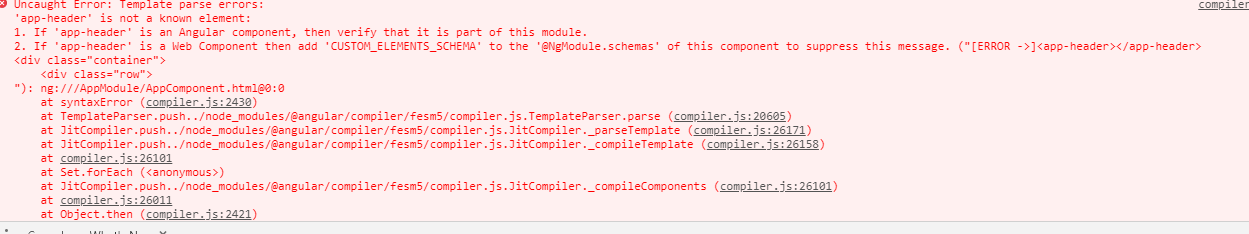
-import from angular core from @angular/core,



Now with that, we need to pass a Javascript object to this decorator to configure this component,

specifically we need to add a template and I'll put this in external file, so templateUrl in this





app-header is not a known element.

This is a common gotcha which is why I wanted to show it.

Why is it not known?

We added it here and we set up selector here, right?

Well remember, you have to register all the features you're going to use in the app module

and since we didn't use the CLI to create this component, it wasn't added there automatically.

So we have to add it manually to our declarations array

and that means we also have to add the import.

So we should import header component from and now pointing to the header folder and then in there,

the header.component file without the file extension.

So that now unlocks the header component in our app and if we save this, now we see the header.

-So we should import header component from and now pointing to the header folder and then in there,

the header.component file without the file extension.



So we had a couple of components we wanted to add and I will use the ng generate command or just ng g as

a shortcut and then we want to generate a component

or just c as a shortcut and the first component I want to add is the recipes component for this recipes

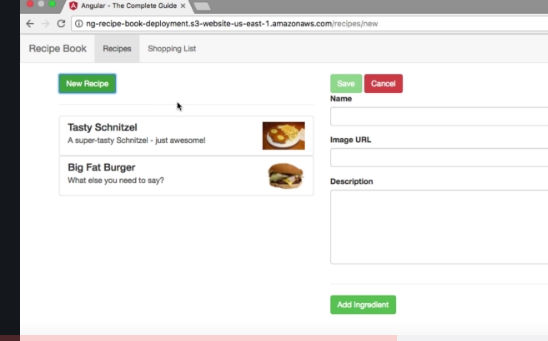
feature area.

I'll add spec false to prevent the creation of a testing file which I don't need.]



Using the components

as I said in my set up, I want to have a recipes list and detail next to each other.



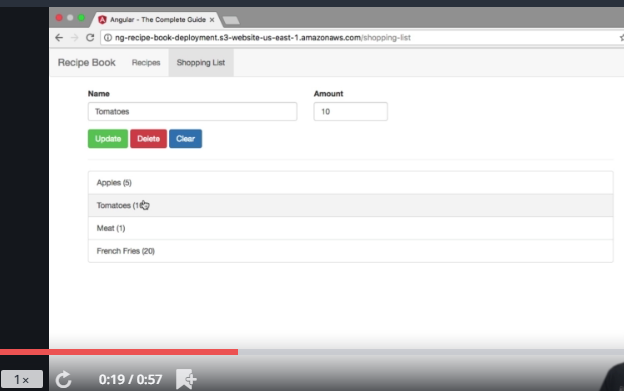
col-xs-10 to create a very wide column here, I'll first

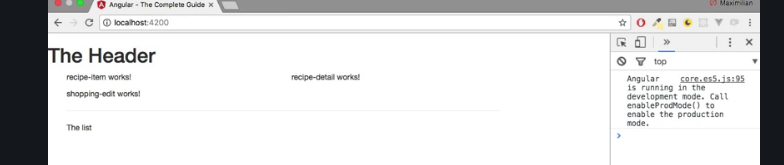
add my shopping edit component to have this editing area at the top where we can add new ingredients

or edit existing ones

and then I want to have a horizontal line here and somehow output my list below this.

So that's the rough setup I want to have in this component.





-here header is not pretty,lets bring it to the final stage in next video.

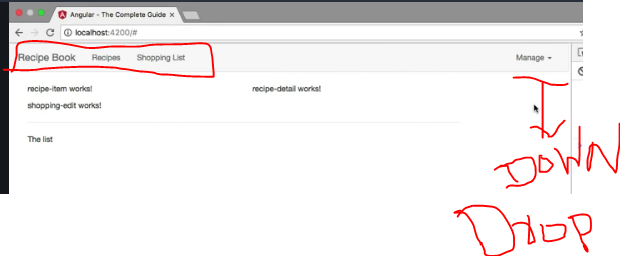
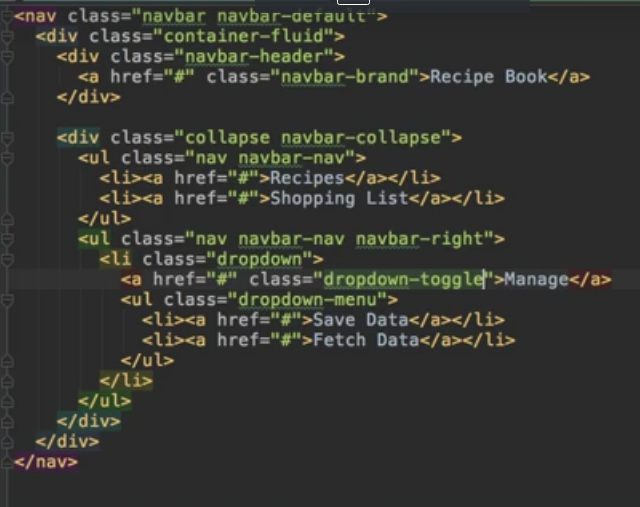
Adding a navigation bar

-instead I will use a couple of Bootstrap classes to create a nice looking header. So I'll create a wrapping

nav element and on this element, I'll add the navbar class and the navbar-default.

Again, these are just Bootstrap classes which will give me this default Bootstrap header.- I'll place a container-fluid div to have my own view container basically in there and the first thing

I want to have here is a header.



Again, these are just Bootstrap classes which will give me this default Bootstrap header. In there,

I'll place a container-fluid div to have my own view container basically in there and the first thing

I want to have here is a header.

For that, we get the navbar header class

we can add to a div and in this div, I want to have a clickable header,

so I'll add an anchor tag pointing to hash

for now, we'll later of course add a route there once we learned how to do this and this should have

the class navbar-brand, to give this this brand-like text style,

I'll say recipe book here since this is the name of our application. So that's the header part of our

header, so this main brand part,

now let's work on the actual links. For this,

I'll add a div which will receive the collaps class and then the navbar-collaps class.

Now, I won't really use the collapse feature,

I won't add a hamburger

icon here since we will not have many links on our header anyways but this gives us some nice styles

and of course, collaps here should have an e at the end, like that.

So in here, I now want to have a group of list items, so a group of links in the end, this will be an unordered

list with the CSS classes, nav and then navbar-nav.

So make sure to create this unordered list with these classes and in there, if we add some list items,

they will actually sit next to each other, so they will automatically be positioned correctly by Bootstrap.

Each list item should hold a link,

so here I'll add an anchor tag, also pointing to just hash for now, one for the recipes section and

the second one for the shopping list section.

Now with that added, I'll add another area in my header here,

this will be another unordered list which will receive the classes, nav and now also navbar-nav

like before but then also navbar-right to position these links at the right and actually, this will

only be one link here which will receive a class of dropdown, this list item

will, because later, we will add our own directive which will turn this into a working dropdown where we

will then choose some options to store our data on a server, something we'll also implement later

in this course or fetch it from there.

So in the dropdown, I simply want to have my dropdown menu which when using Bootstrap is added by adding

an unordered list with the class dropdown-menu,

so this will hold the items this dropdown and here, I'll add a couple of list items which all hold some

links pointing to just hash for now, one for saving data

and one for fetching data. Again this will be filled with working code later.

So that's my header,

let's save it and see how it looks.

Looks ok,

I don't see my dropdown though,

the brand looks off,

so I'm not entirely happy with that.

Let's see what is wrong.

The brand is not working because that should be navbar, not have a h in between, so if you fix this, the

brand looks all right.

Now what's wrong with the dropdown?

Well the dropdown simply misses the link we would see to open the dropdown because this unordered

class here simply is the content of the dropdown,

we of course also need a link to trigger it.

So let's add an anchor tag pointing to hashtag here

where I will say manage, just manage maybe, you could of course change that text and this should have

a class of dropdown-toggle because this will be the button toggling the dropdown in the end.

Therefore it will have a row of button to add some accessibility to this and also required for this to

work correctly though

and with this now if we save this, we see the dropdown. It would be nice to have a little arrow next

to it indicating that it is a dropdown.

So next to manage, I'll add a span with a class of caret which will give us this arrow pointing to the

bottom,

this arrow here and now we get a dropdown.

It won't work if we click it because we haven't added any code which would open the dropdown,

we will do this after the directives section though.

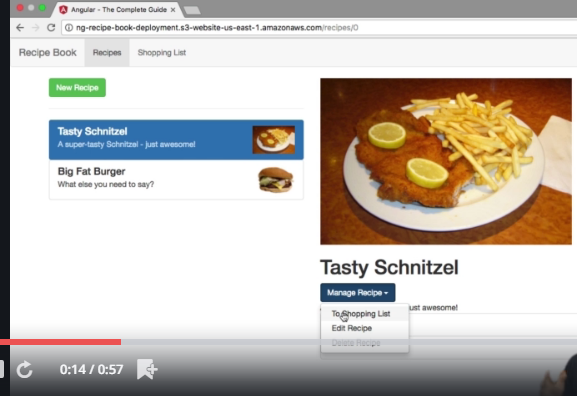
So the header is now working,

with that let's move on

and let's work on our recipes section.

Creating Recepies model

Final output of recepies model

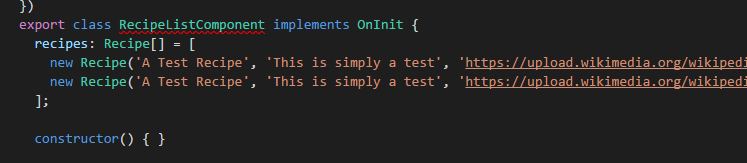


-we get the recipes component which holds the recipe-list,

-for now what I want to do here is I want to add my recipes array, my array of recipes which for now is

an empty array

recepie-list-component.ts



-want to define how a recipe should look like.

We're going to use a recipe a lot throughout this app and therefore we should definitely clearly define how

a recipe looks like, so that whenever we use it in any component, we're always talking about the same

structure, about the same kind of object.

So for this, I will create a model. What is a model then? A model simply is a TypeScript file

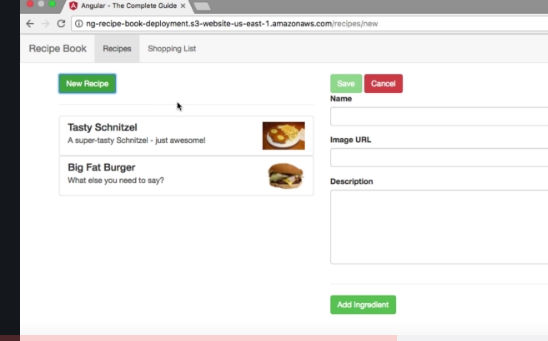
**File:-Recepie.model.ts in recepies folder**

First of all, I'm going to export a ts class, a TypeScript class which is simply named recipe

because we define how a single recipe looks like here.

**-**@model here, now we're not going to do this,

there is no decorator like this,we can use vanilla TypeScript for this. A model

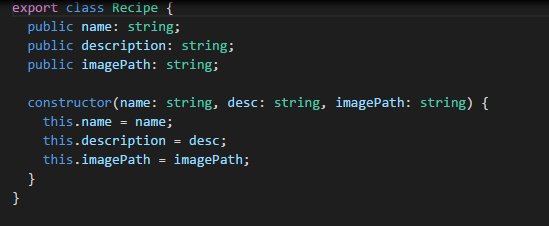


**Concept similar to class in java**

A class can be instantiated, so we can create new objects based on the setup we provide here in this

class. So we can define how a recipe should look like in this class,so let's do this.

-A recipe should have a name and I'll add an accessor to be really clear about that this is publicly available, so that this can be accessed from outside



-So I'll add public in front of it and then name as the property name.

I'll also assign the type of this by adding a colon and the type will be string and that is just how you

assign types in TypeScript, you add a colon after the property name and then the type you want to assign,

so string

in this case

-I'll also add a constructor to it so that we can instantiate this with the new keyword and pass the arguments

right to the constructor,

-constructor is simply a built-in function

every class has and which will be executed once you create a new instance of this class,

-we get a recipe model we can use,

**Adding content to Recepies Component**

Back to recepie-list component



-I added my recipes array and now that we created the model, we can also define a type for this. So we

can add colon to inform TypeScript that now we will define the type for this property and the title

-will be recipe, so that will use our own model and not just a recipe but an array of recipes which is

indicated by adding square brackets after it.

-So now TypeScript knows

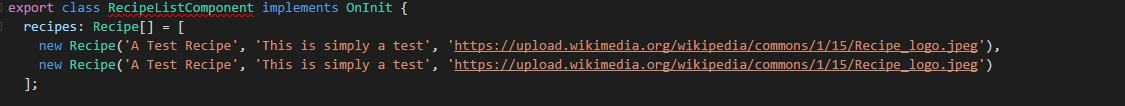
the only thing which will ever get stored in this recipes property is an array which holds a couple

of recipe object

-Now to inform TypeScript where this type is coming from, we need to add an import.

So I'll import recipe, that is what we named it in here, right,

that's our class name so that's what we have to specify between the curly braces here, from

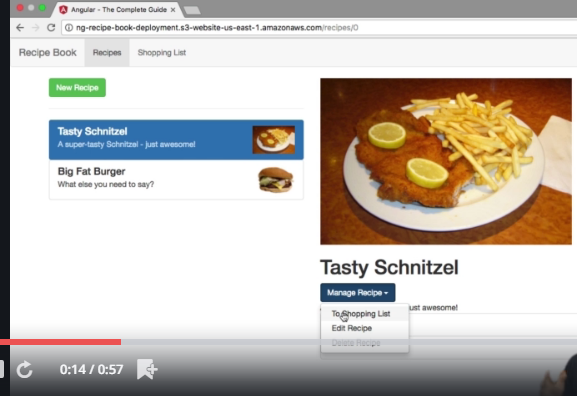


So in this array, I will create a new recipe and I can use the new keyword to create a new object based

on this recipe blueprint, based on our class

and now when executing this like a method, we're actually calling the constructor,

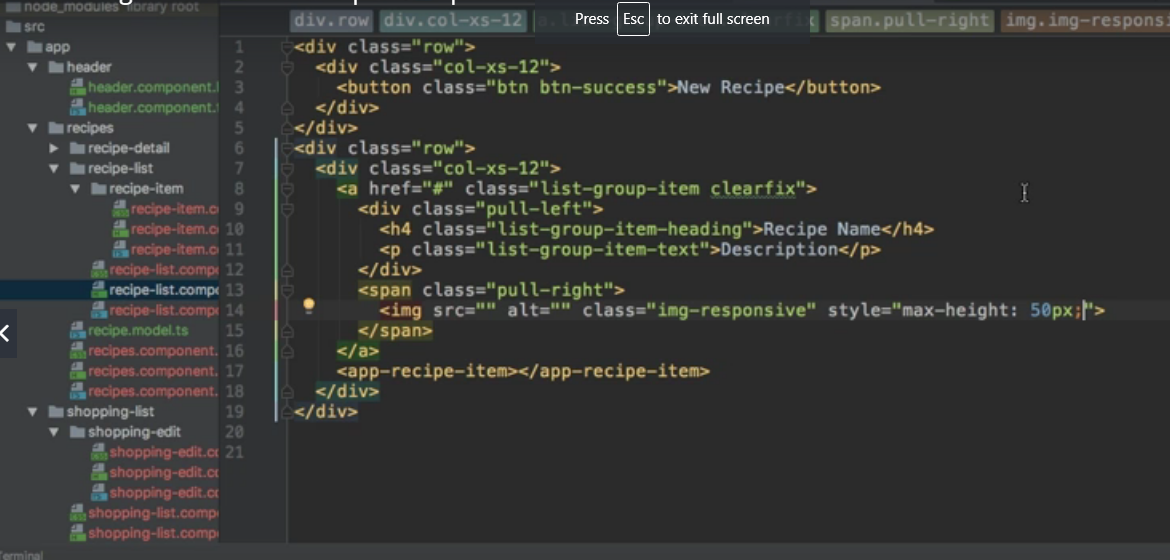
so now here we need to pass the arguments we're expecting in this constructor, so name, description and image,path.



so to see that, we should actually do something in the template of the recipe-list component. Here I'm

right now only outputting my recipe-item and I want to loop through all these items and use them to

display my recipes. For now, what I will do is I will add a Bootstrap row



-So here, I'll simply set up a button with the classes btn and then btn-success where I say new

recipe, like this.

Now again, this will be hooked up later.

So I'll add another row below this which should now actually hold my list of recipes, here I'll

again use col-xs-12

to have this span over the full width and keep in mind, this will already be nested inside of this

md-5 columns,

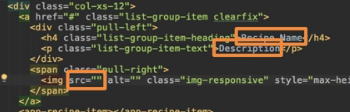
so it's only that wide to begin with

but inside this column, I wanted to span over the full width

and now here, we will later output my recipe items. Now with that, the goal is to replicate this item, again we will later put it into its own component

and then fill these spaces here with dynamic data.

This is your challenge,

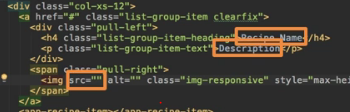


**and again all that content will go into the app-recipe-item component after the next course section.**

Outputting a list of Recepies with ngFOR

-So were you successful using our recipes array here and replicating this part in the code as often as

needed and fill it with the data for the recipe?



-So let's add ngFor, this is the directive which allows us to repeat code in our template

and here, I want to loop through all the recipes.

So I will actually create a local variable named recipe,

this name is totally up to you,

that will simply store the recipe of each iteration in our loop and I will loop through all the recipes,

so through this recipes property we created here in the TypeScript file.

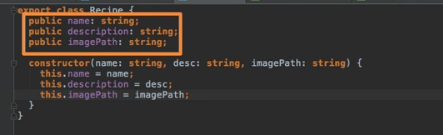
-So let's use it to output the name,

we can use string interpolation for that.

String interpolation as we learned allows us to output some text anywhere in our template.

Now we got the single recipe but that's going to be an object looking like our model because remember,

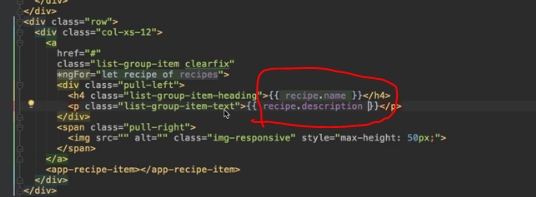
the recipes array simply holds an array of our recipes,

-

-

and for the description, we can replace this with recipe description.

So we're going to use the property names



we set up here in the model and important, you have to use the ones here at the top of course because

down there, these are only the constructor argument names,

they are not available outside of this model.



So here is a great example of where you may use both, string interpolation with the normal HTML attribute

or not using the attribute but directly binding to the source property of the image and then just putting

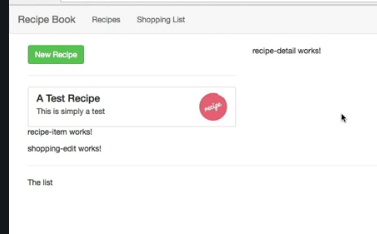
in a valid expression, returning what this property expects, in this case a string between the quotation marks.

-You can either use string interpolation here to output recipe imagePath here

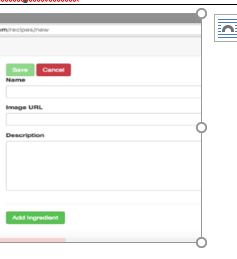
and that should work fine

or as an alternative, you cannot use the attribute plus string interpolation but you could bind to the

source property itself



Displaying Recepie details

So a button which allows us to add it to the shopping-list and then one which allows us to edit it orto delete it and actually, we could use a dropdown here. Such dropdown may be added here by adding a div,with the Bootstrap class btn-group

<**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="#">Delete Recipe</a></li>**

**</ul>**

**</div>**

**</div>**

**</div>**

**<div class="row">**

**<div class="col-xs-12">**

**Description**

**</div>**

**</div>**

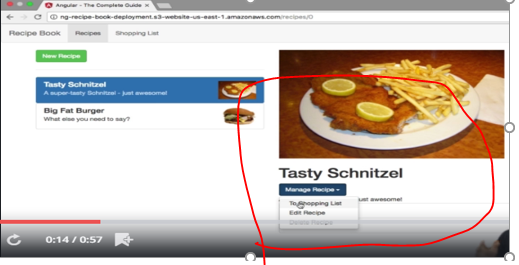
**<div class="row">**

**<div class="col-xs-12">**

**Ingredients**

**</div>**

**</div>**





So that cross-component communication is a huge issue right now but no worries, that is something we're

going to dive deeper into in the next core section and thereafter, we'll come back to this and enable

all that communication which allows us to outsource this code into its own component and make these

links clickable so that we actually load the right recipe on the right. Before doing this,

Working on shopping list component