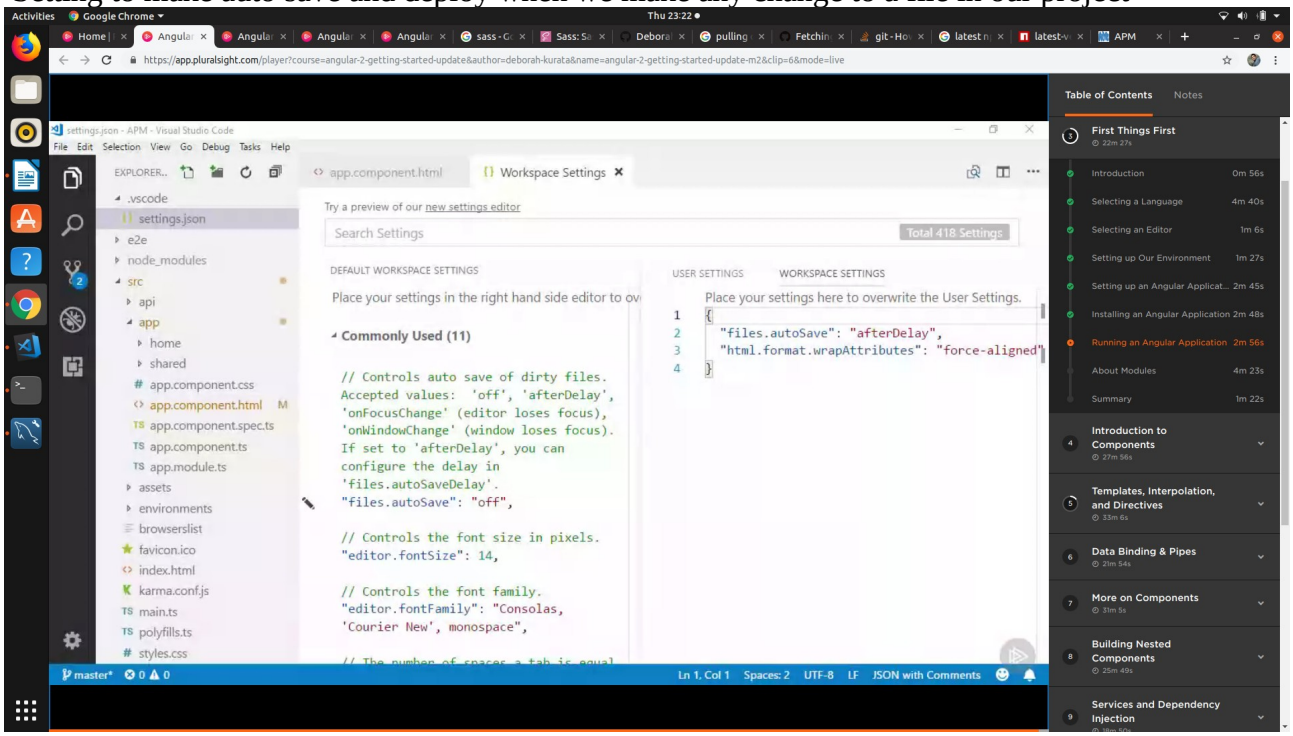
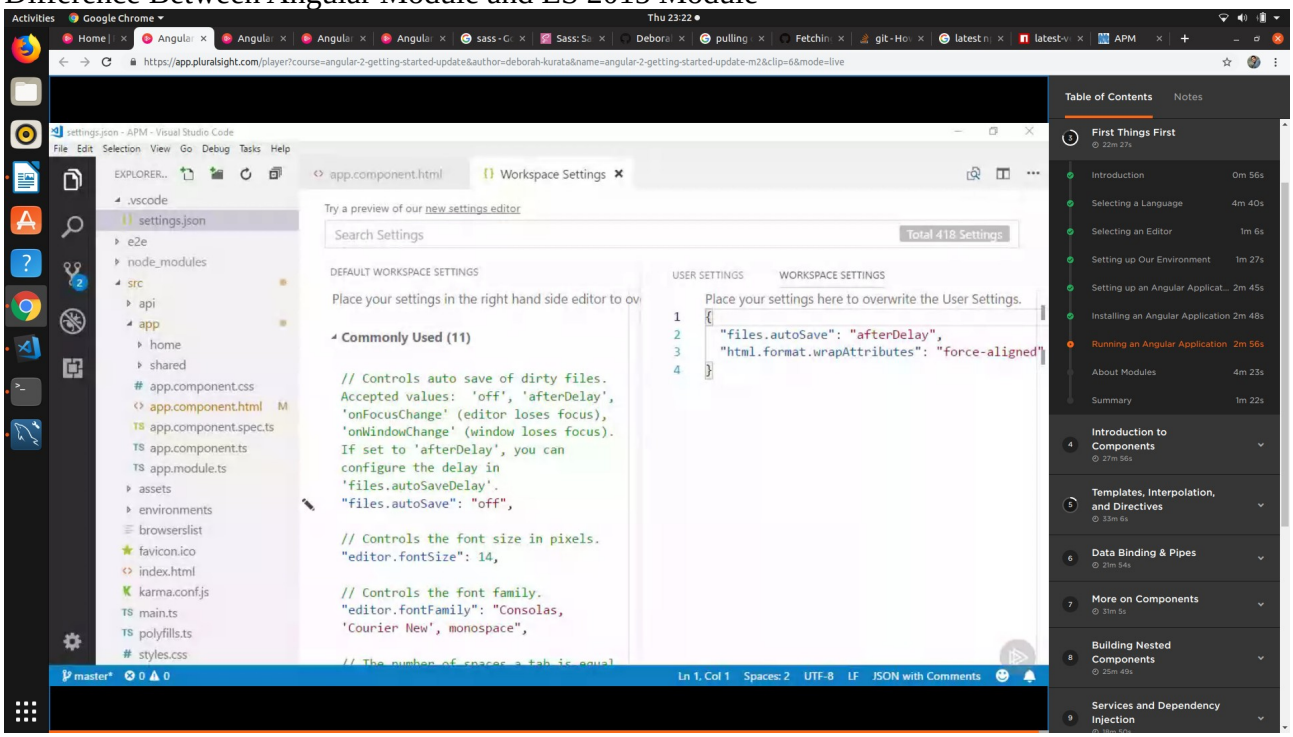
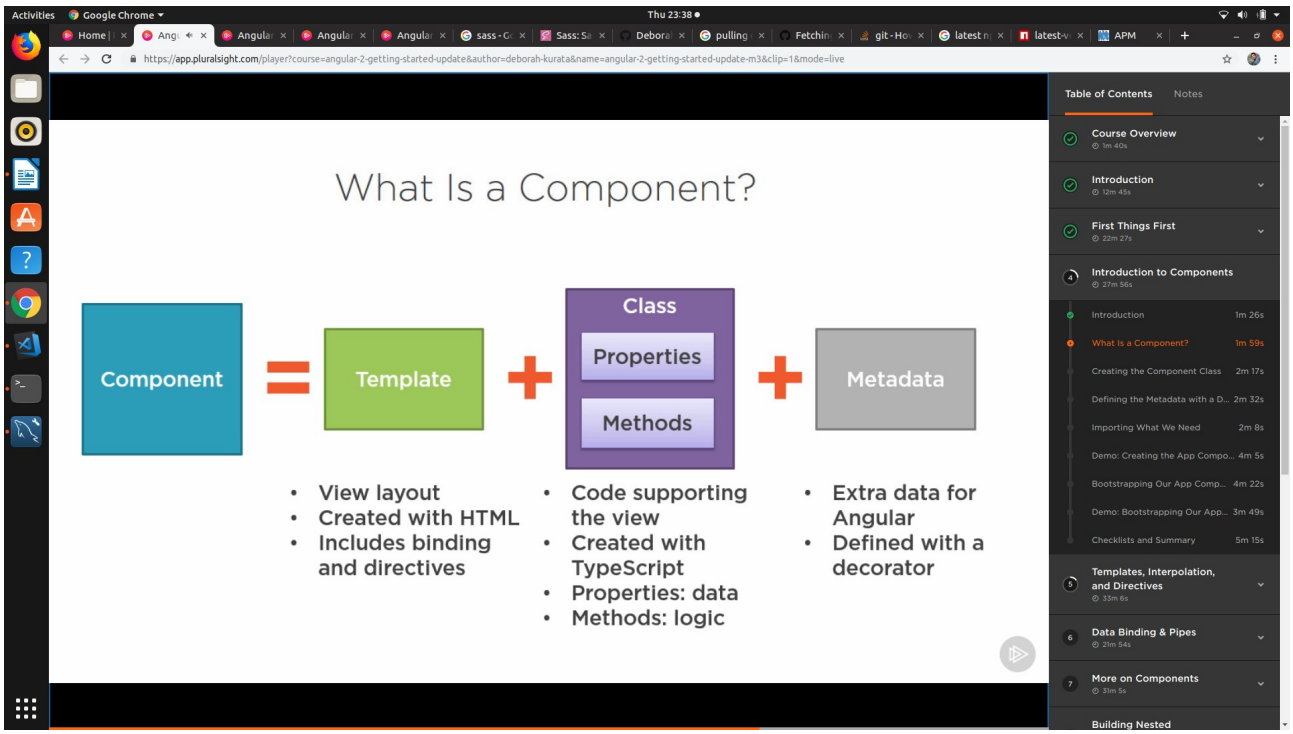
Setting to make auto save and deploy when we make any change to a file in our project



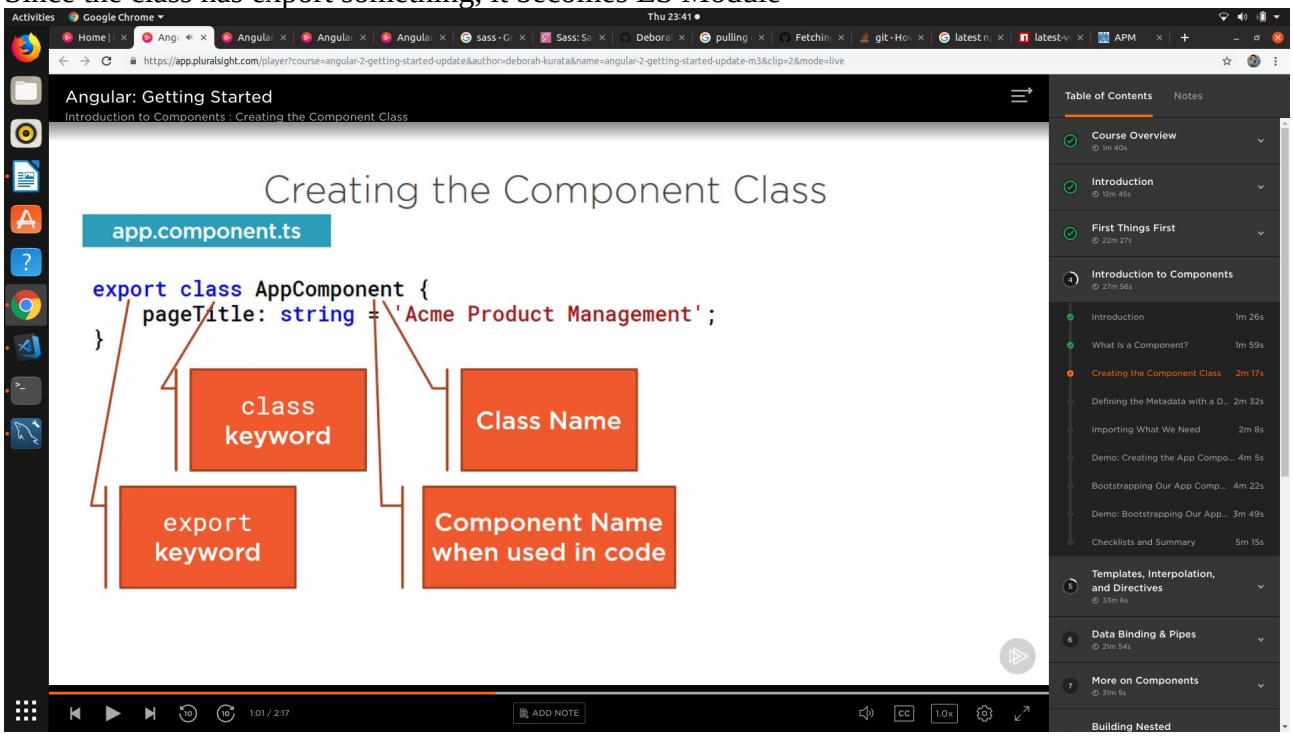
Difference Between Angular Module and ES 2015 Module



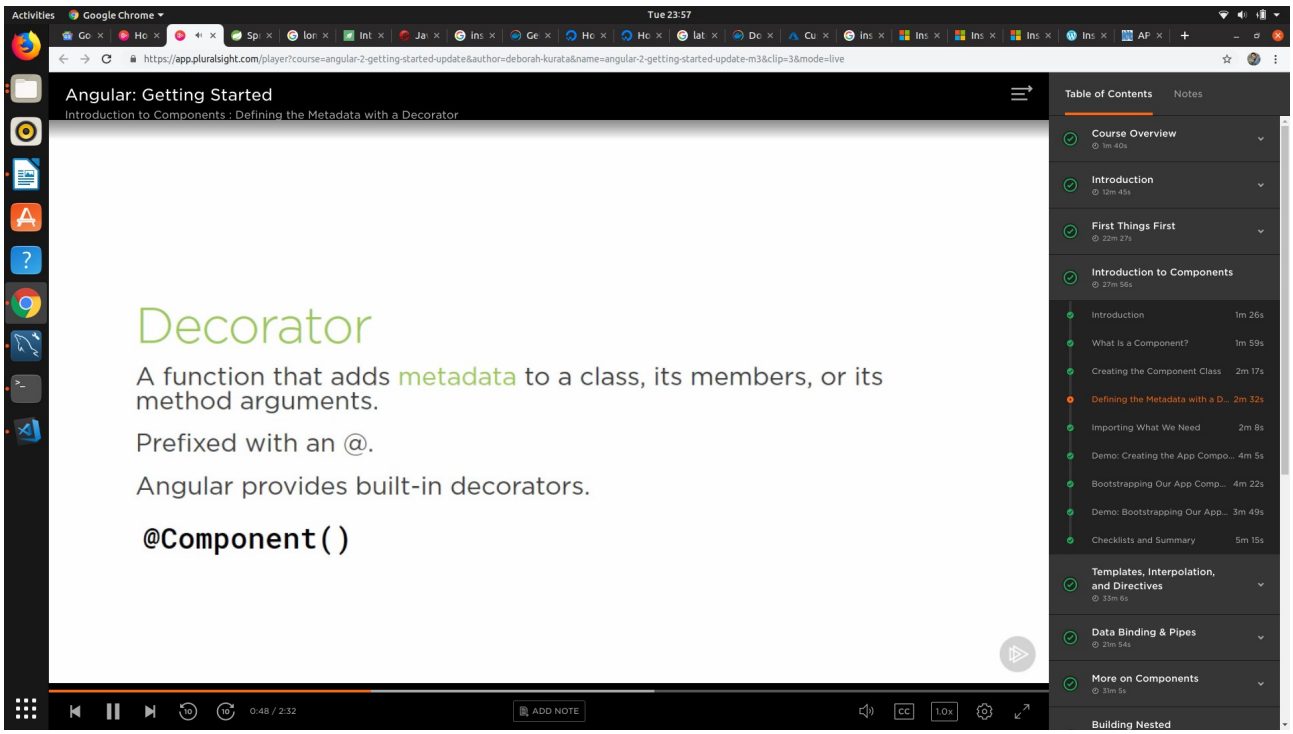
What is Component?

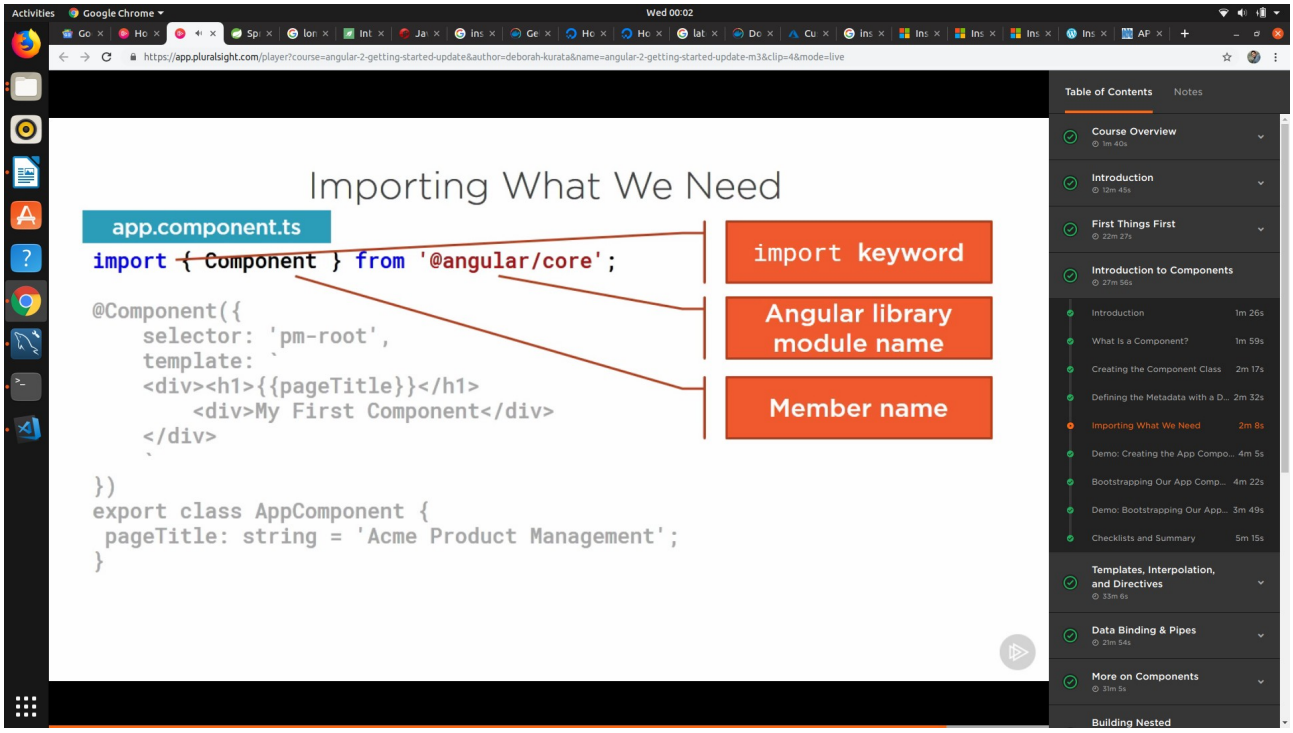


Since the class has export something, it becomes ES Module

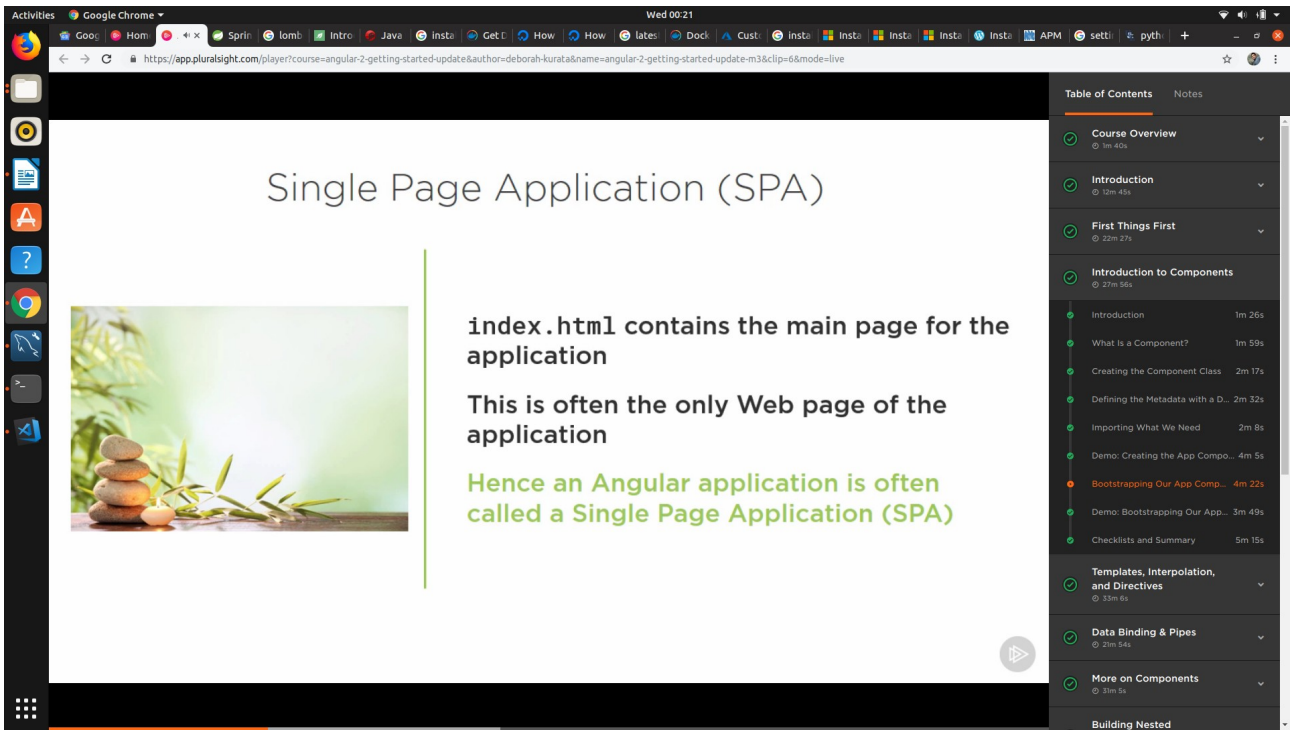


What is Decorator?

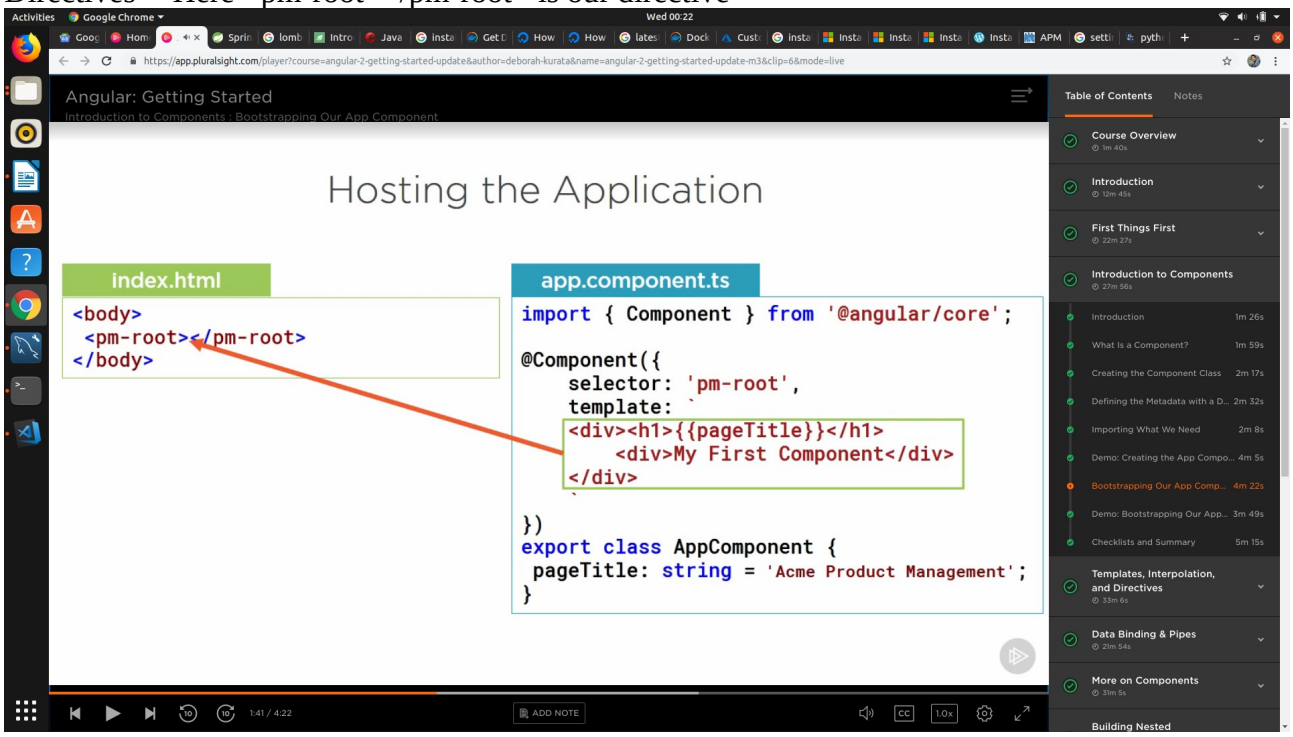




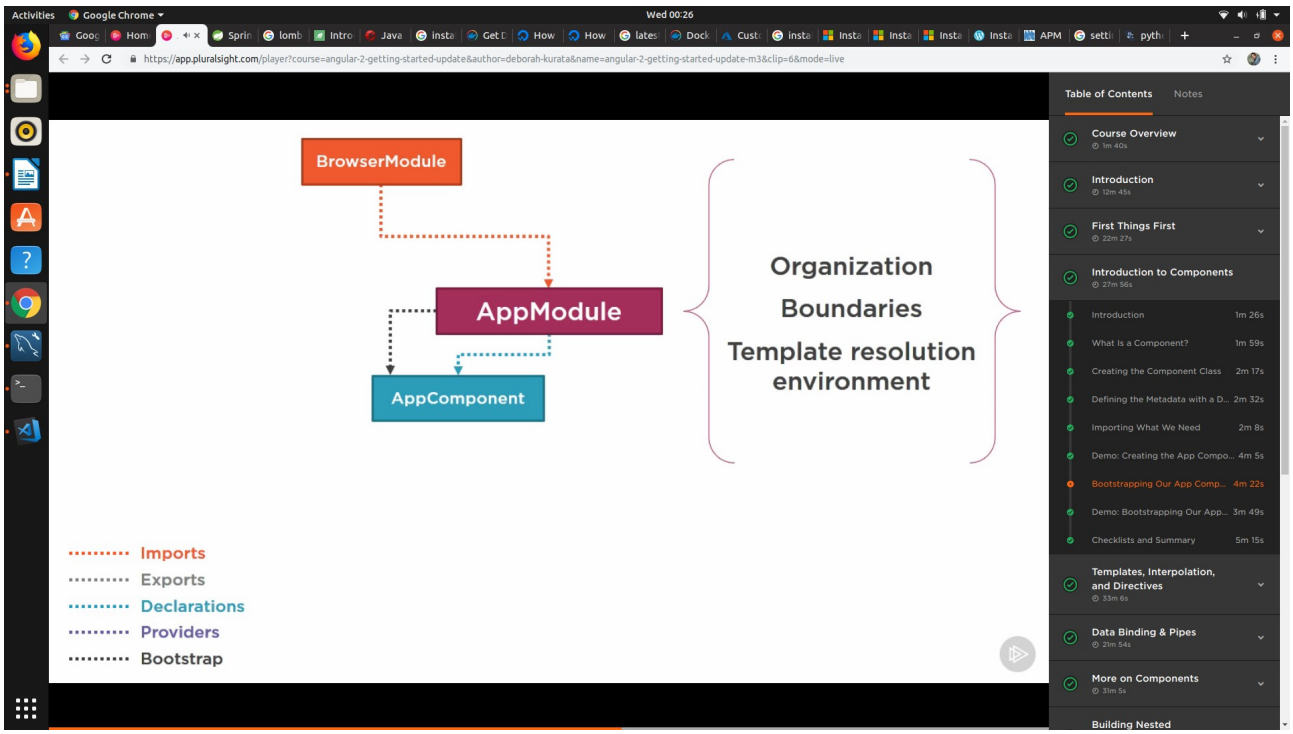
Single Page Application



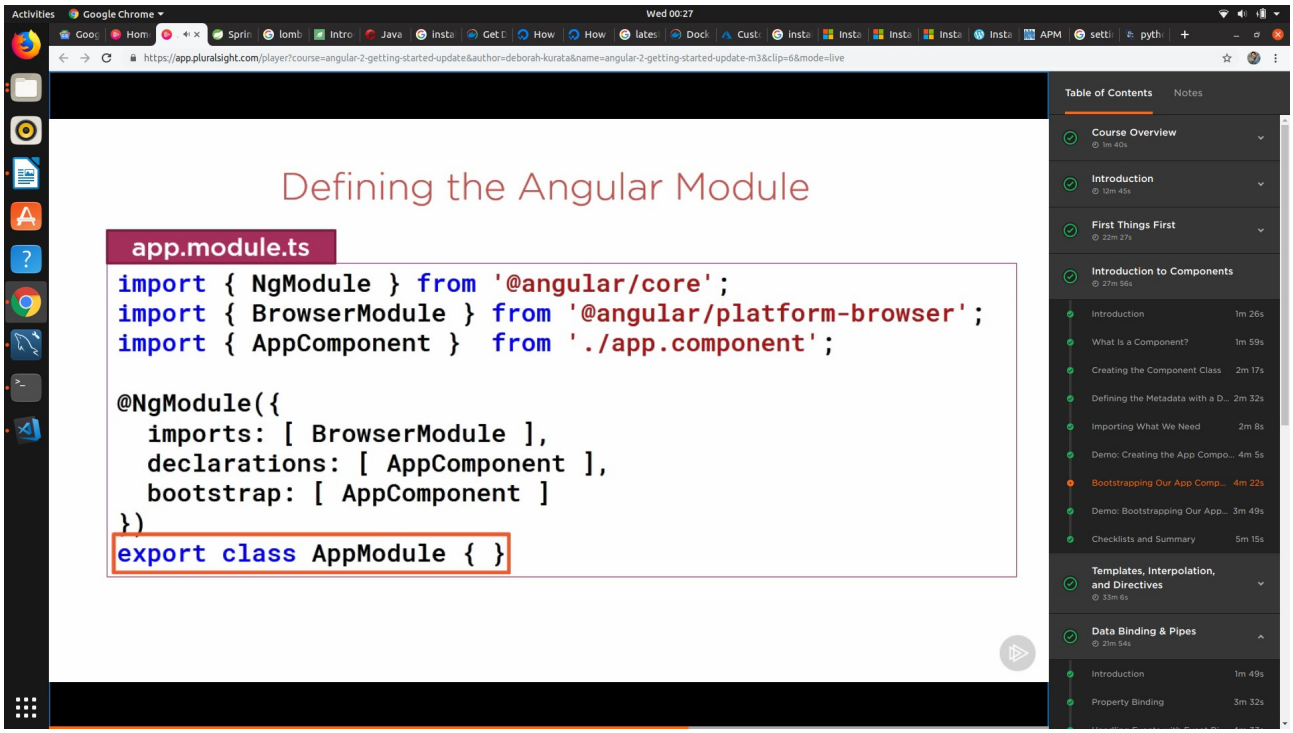
Directives-> Here <pm-root></pm-root> is our directive



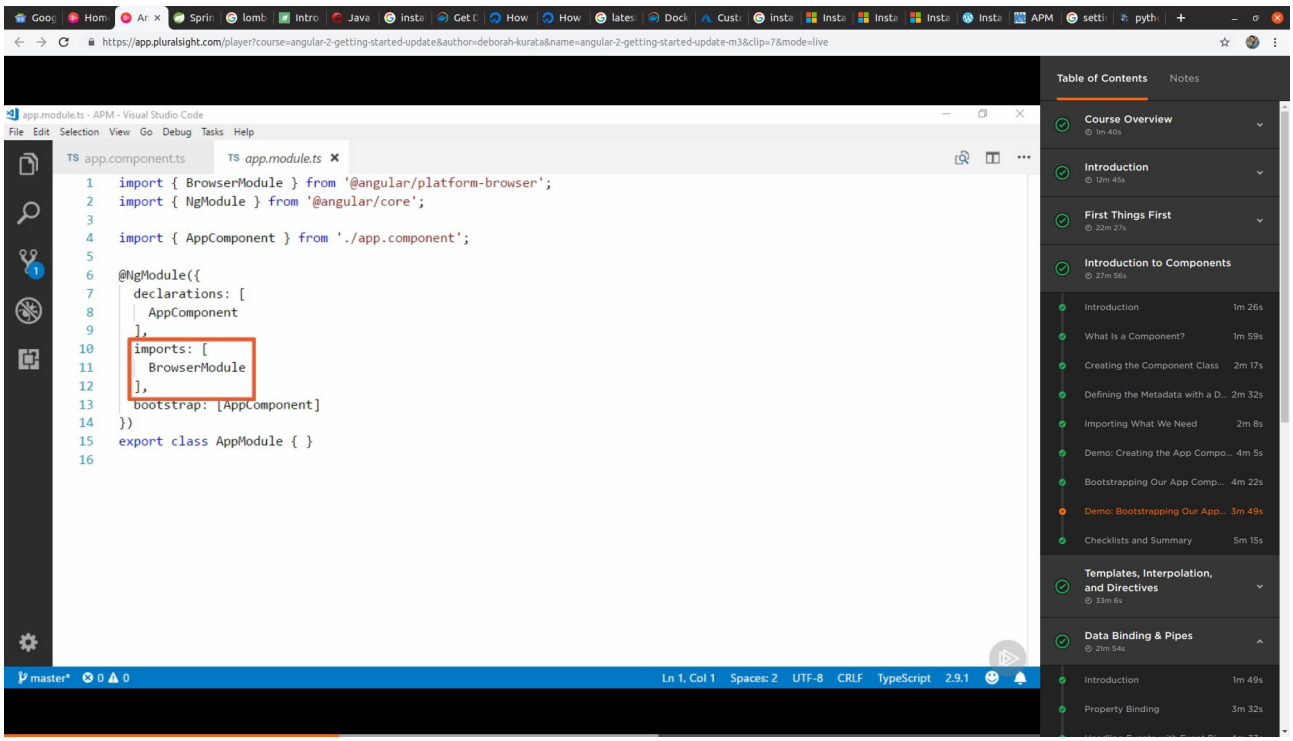
AppModule: Boot straps the application



Directive-> Metadata like imports usues browser module whcih is required for running application in browser

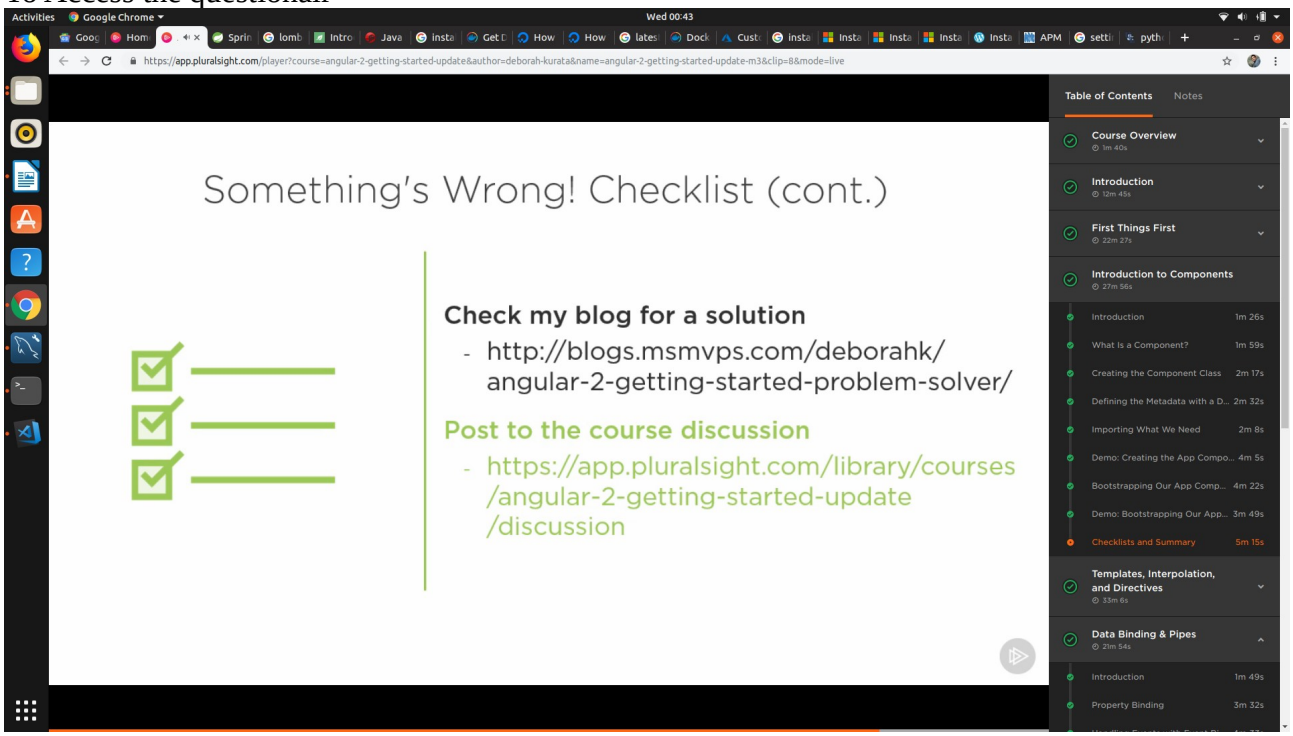


*Declaration array declares the AppComponent so that angular can locate its selector
Import array declares the browser Module so that application can run smmothy in browser
Bootstrap array list our AppComponent as the starting componnet for the application



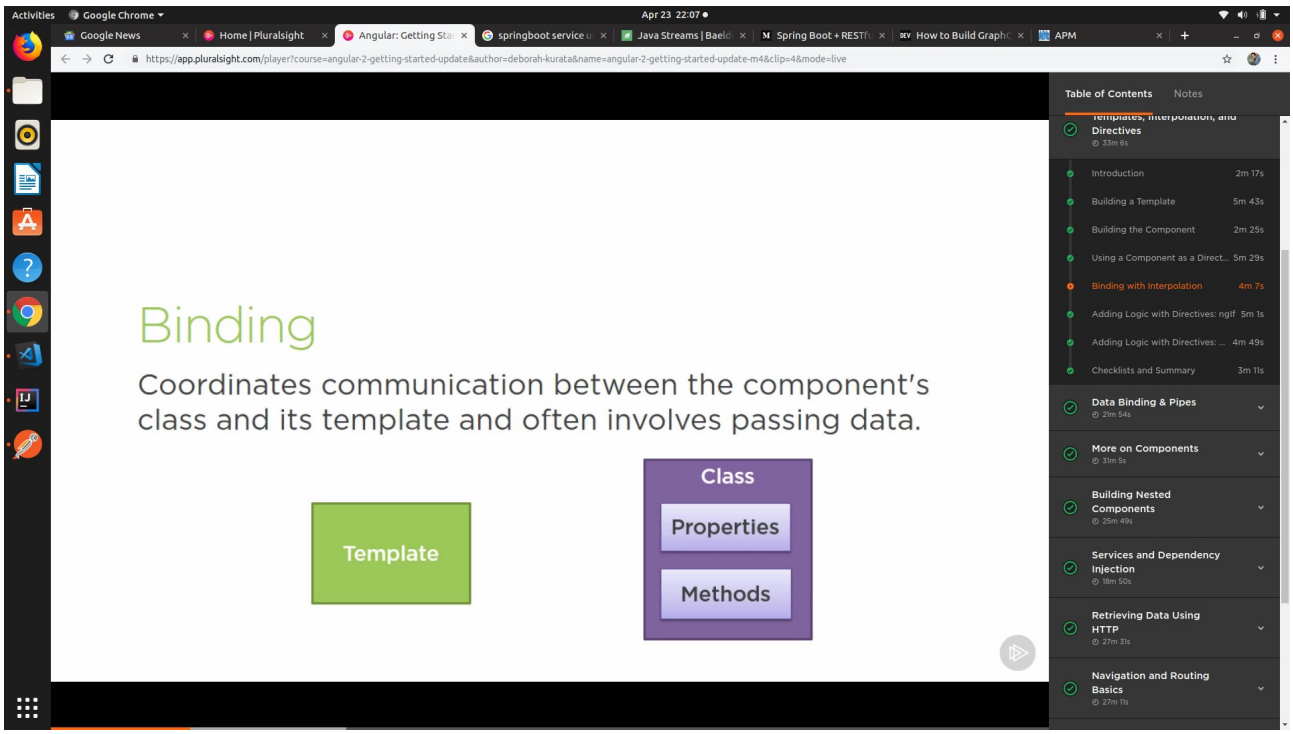
Note: export keyword makes the class accessible to be imported to the other class of the application

To Access the questionair

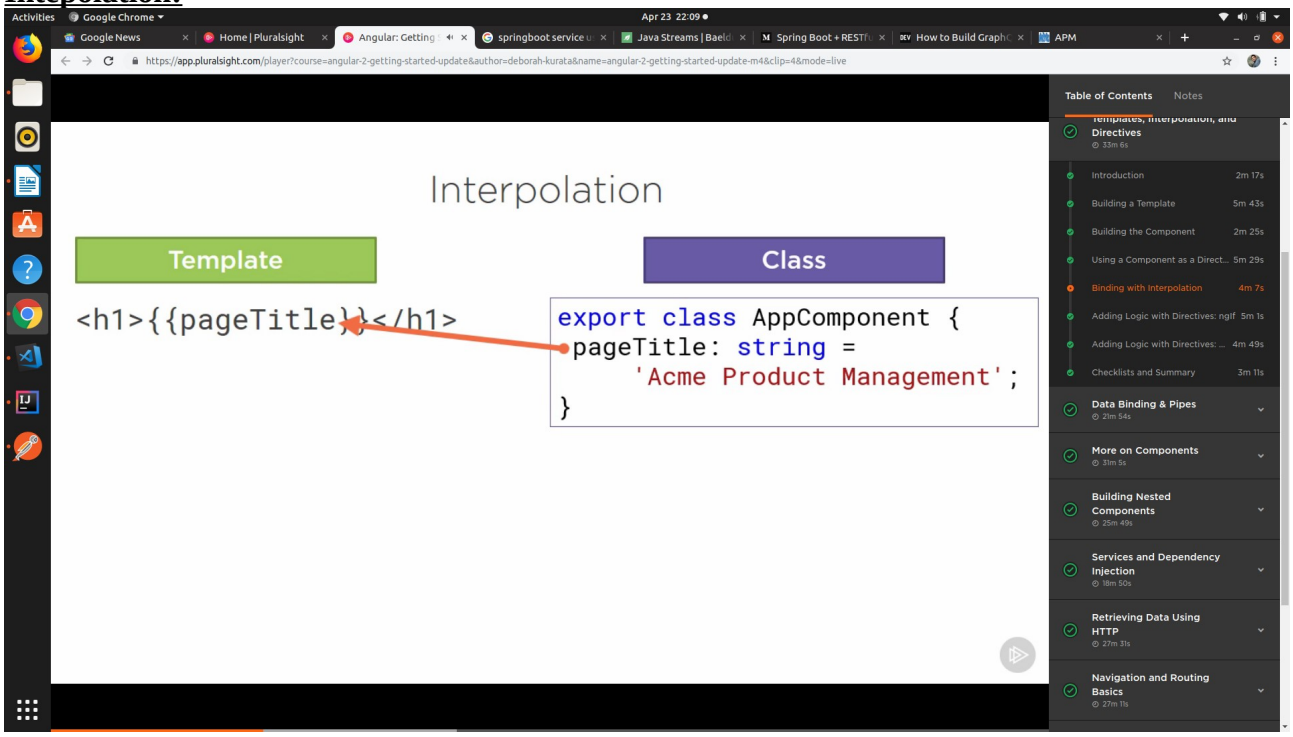


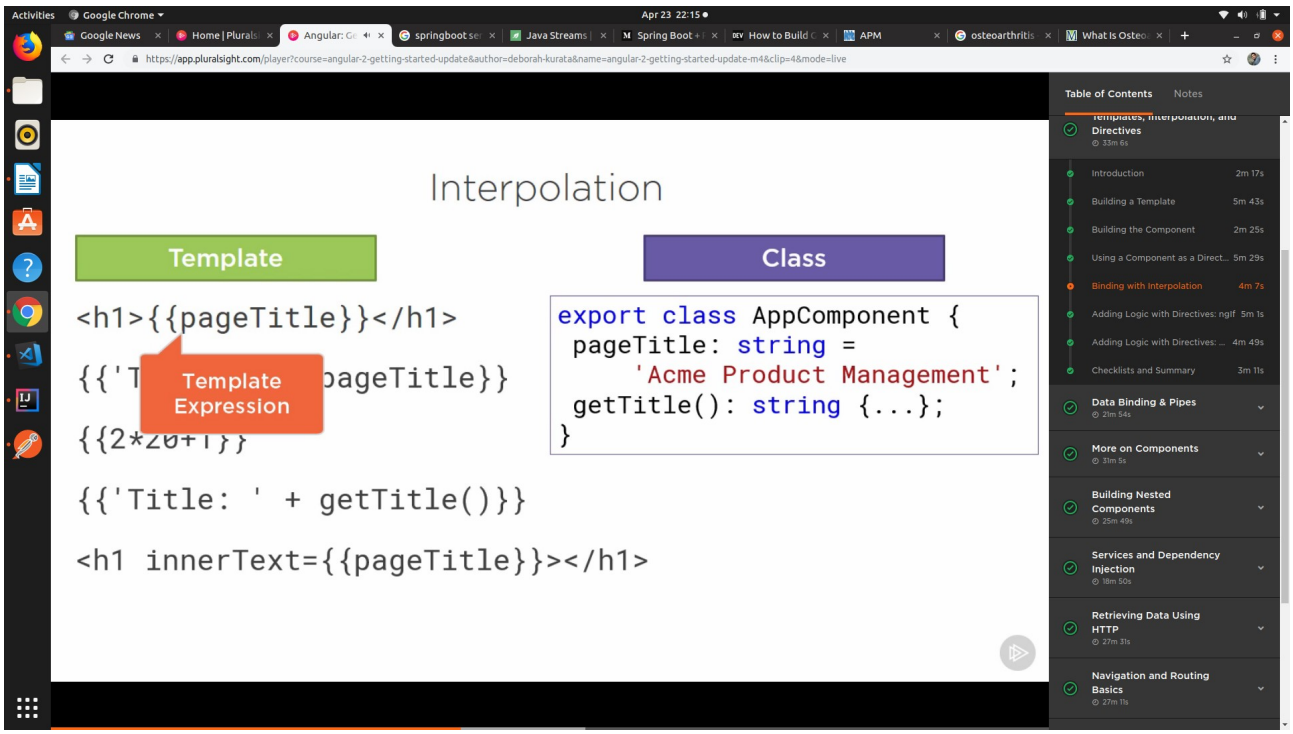
Template Interpolation and Directives

Binding:

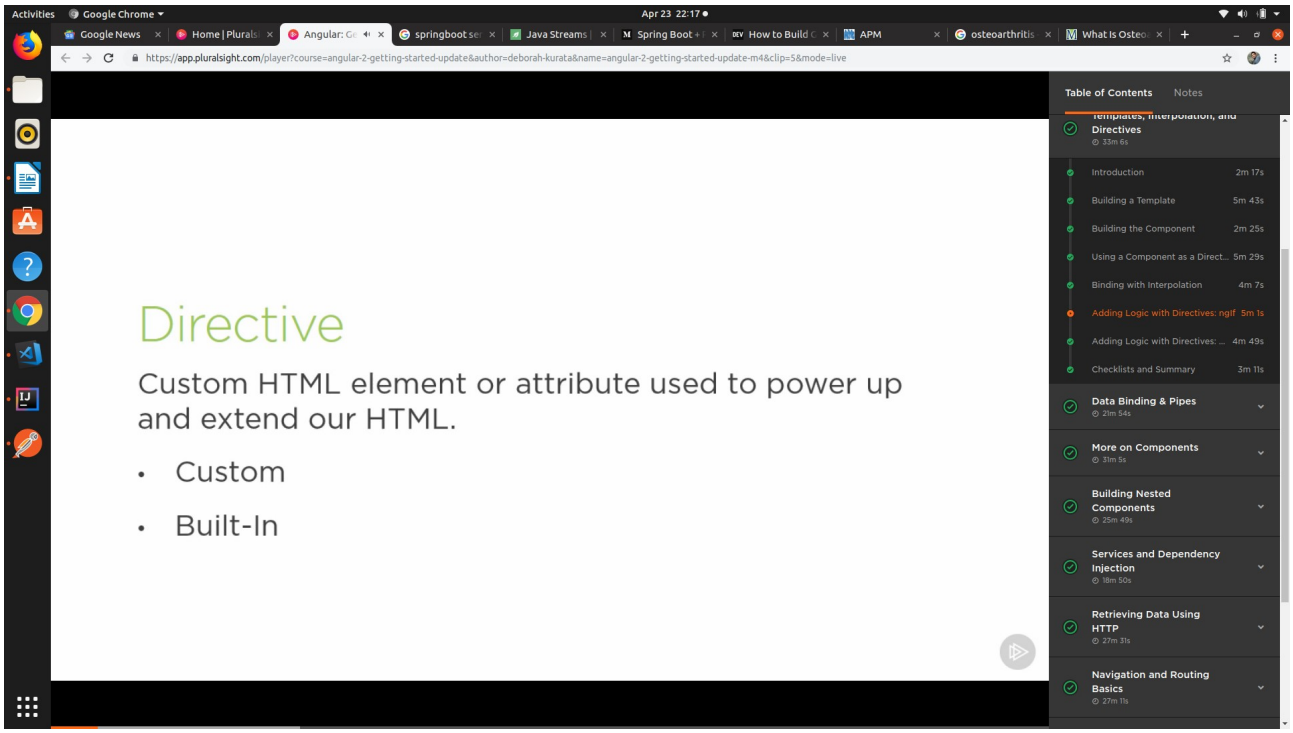


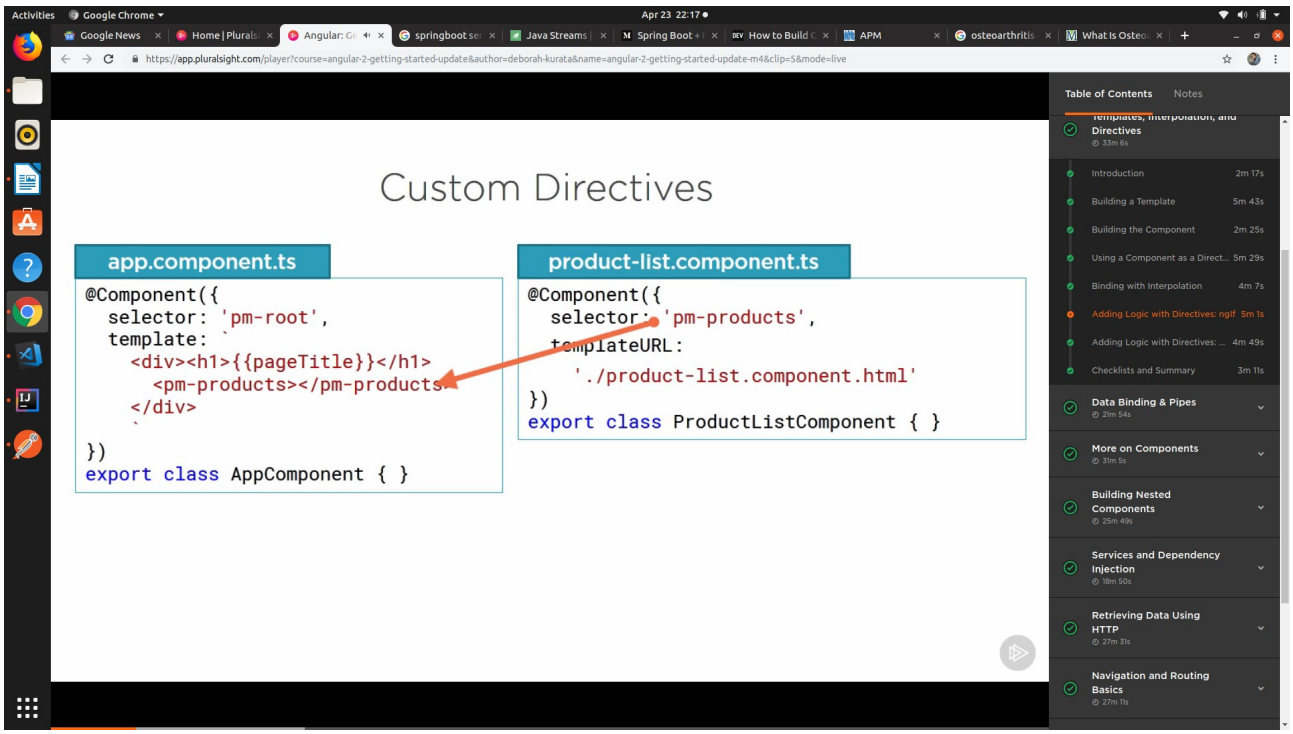
Intepolation:



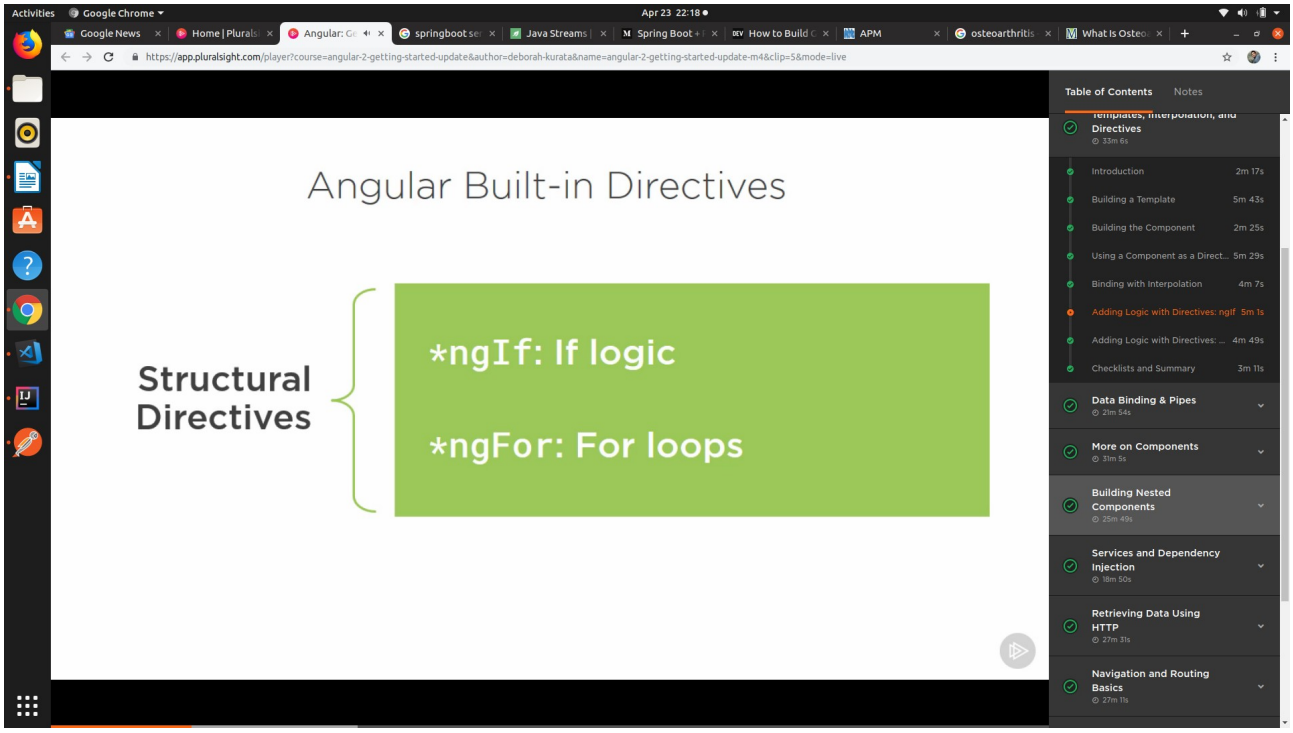

Interpolation
Template
Class
<h1>{{pageTitle}}</h1>
Template Expression
{{'Title: ' + getTitle()}}
<h1 innerText={{pageTitle}}></h1>
export class AppComponent {
pageTitle: string =
'Acme Product Management';
getTitle(): string {...};
}
Table of Contents
Notes
Introduction 2m 17s
Building a Template 5m 43s
Building the Component 2m 25s
Binding with Interpolation 4m 7s
Adding Logic with Directives: ngIf 5m 1s
Checklists and Summary 3m 11s
Data Binding & Pipes
More on Components
Building Nested Components
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics

Directives:

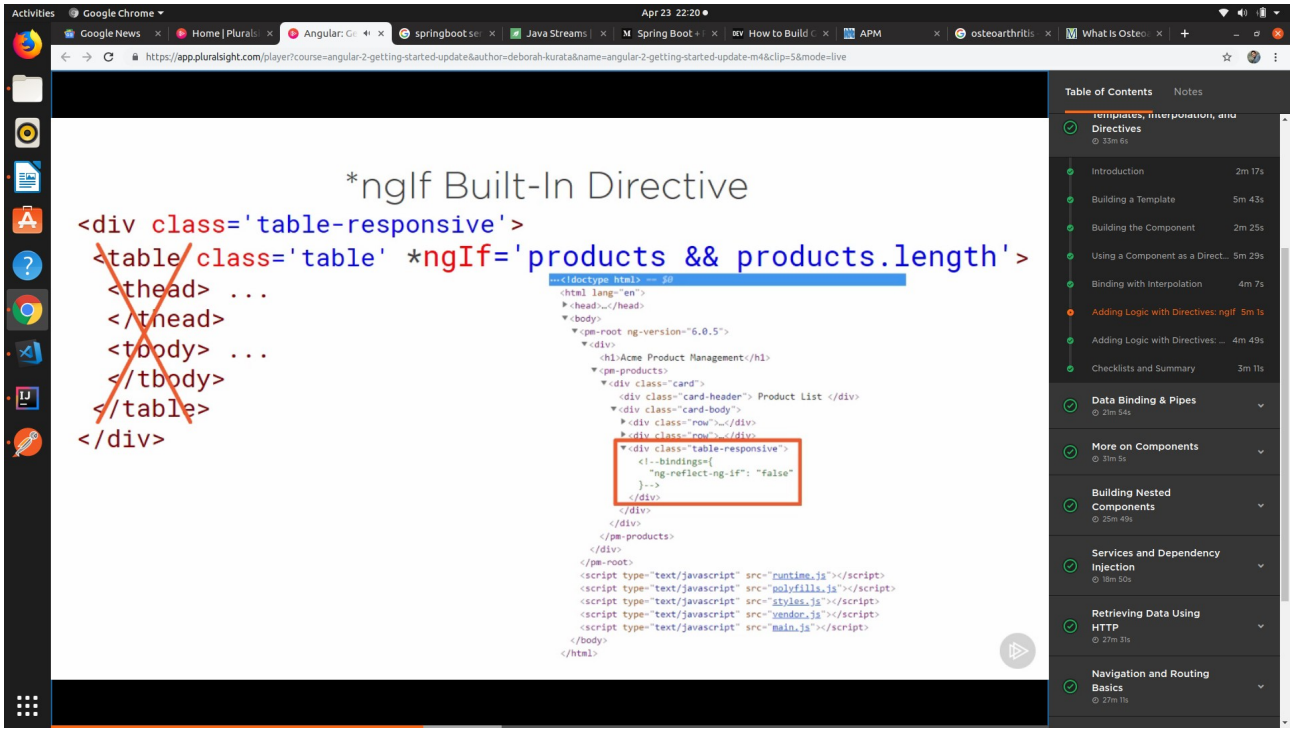

Directive
Custom HTML element or attribute used to power up and extend our HTML.
Custom
Built-In
Table of Contents
Notes
Introduction 2m 17s
Building a Template 5m 43s
Building the Component 2m 25s
Binding with Interpolation 4m 7s
Adding Logic with Directives: ngIf 5m 1s
Checklists and Summary 3m 11s
Data Binding & Pipes
More on Components
Building Nested Components
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics



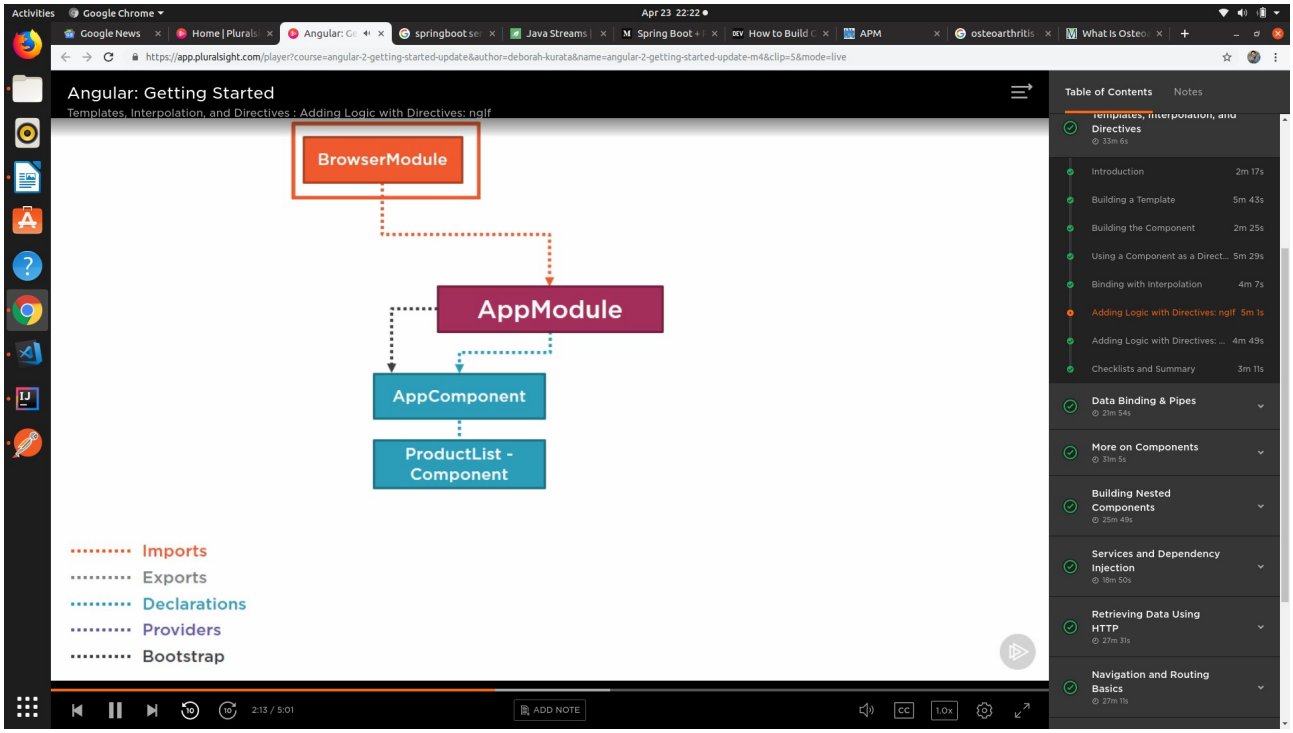
Structural Directives:: * infront denotes structural directives



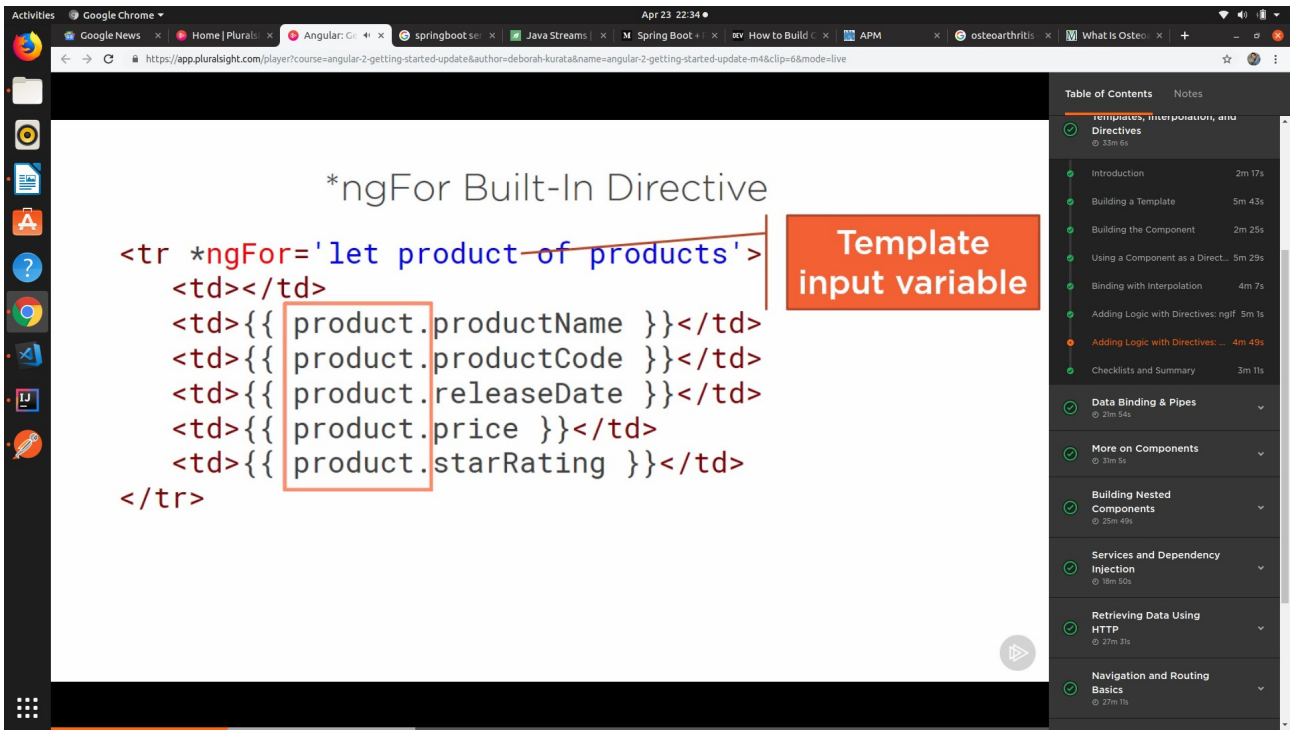
If the below condition fails it will not render anything under the <table> as shown in the right side of the page

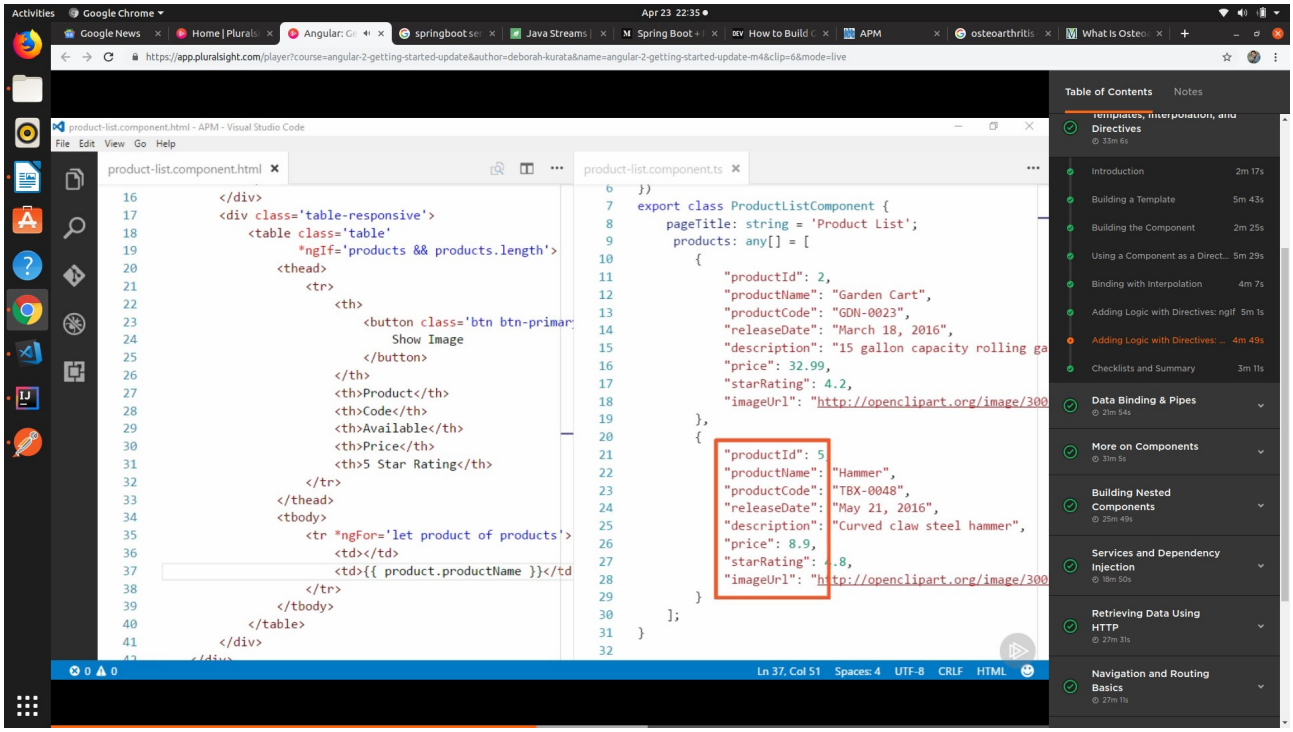


Browser Module is required for using the *ngIf and * ngFor

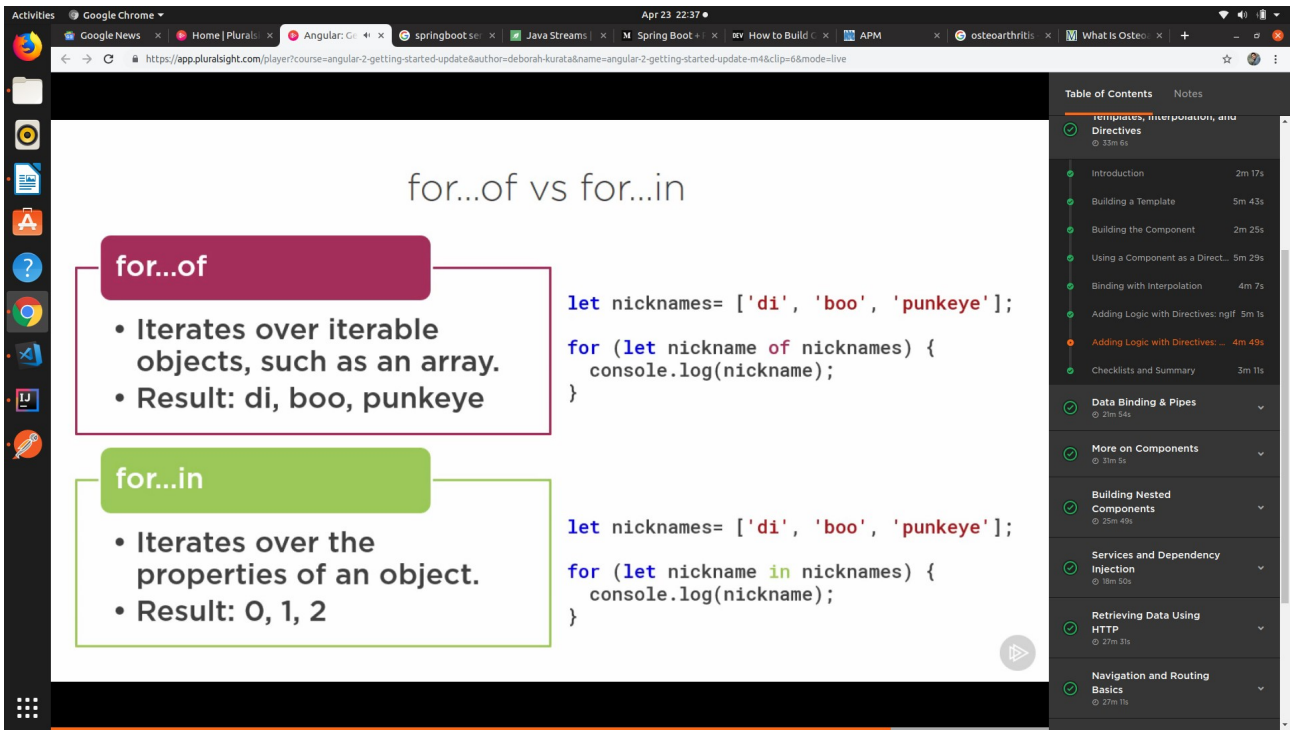


*nfFor: let makes product as template input variable



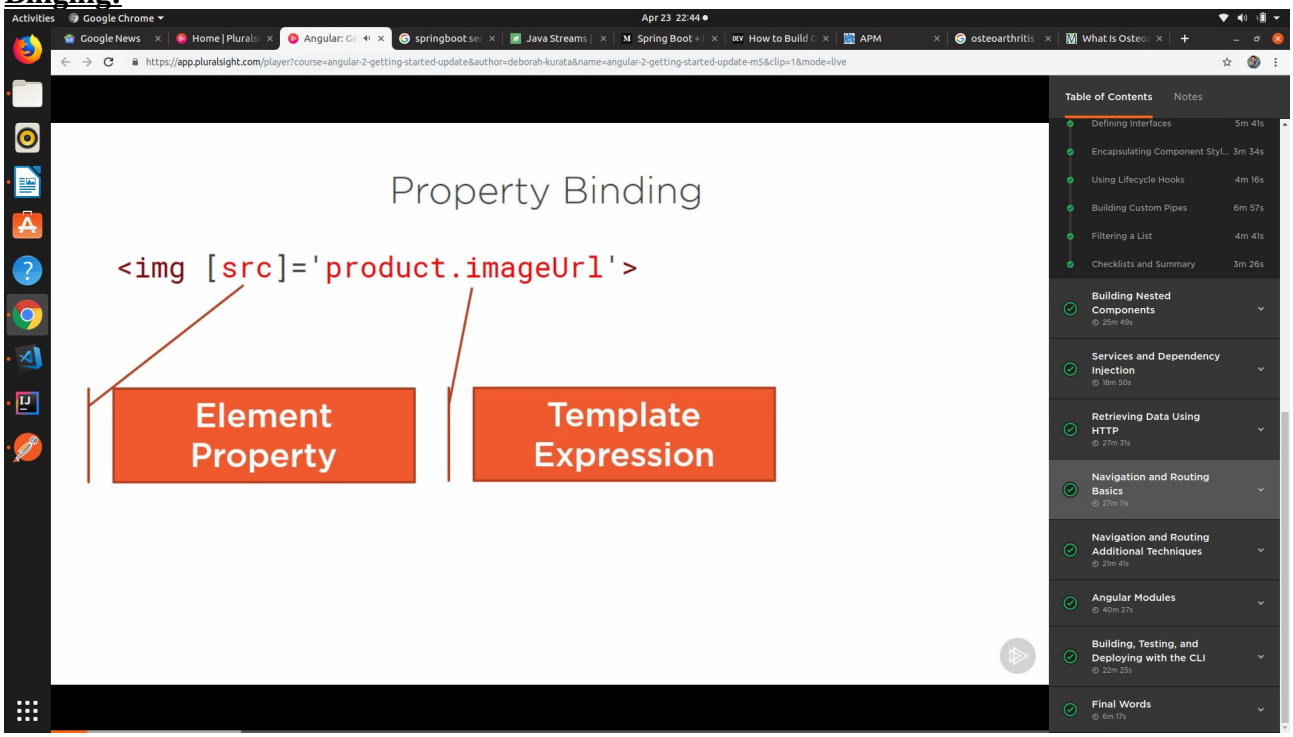


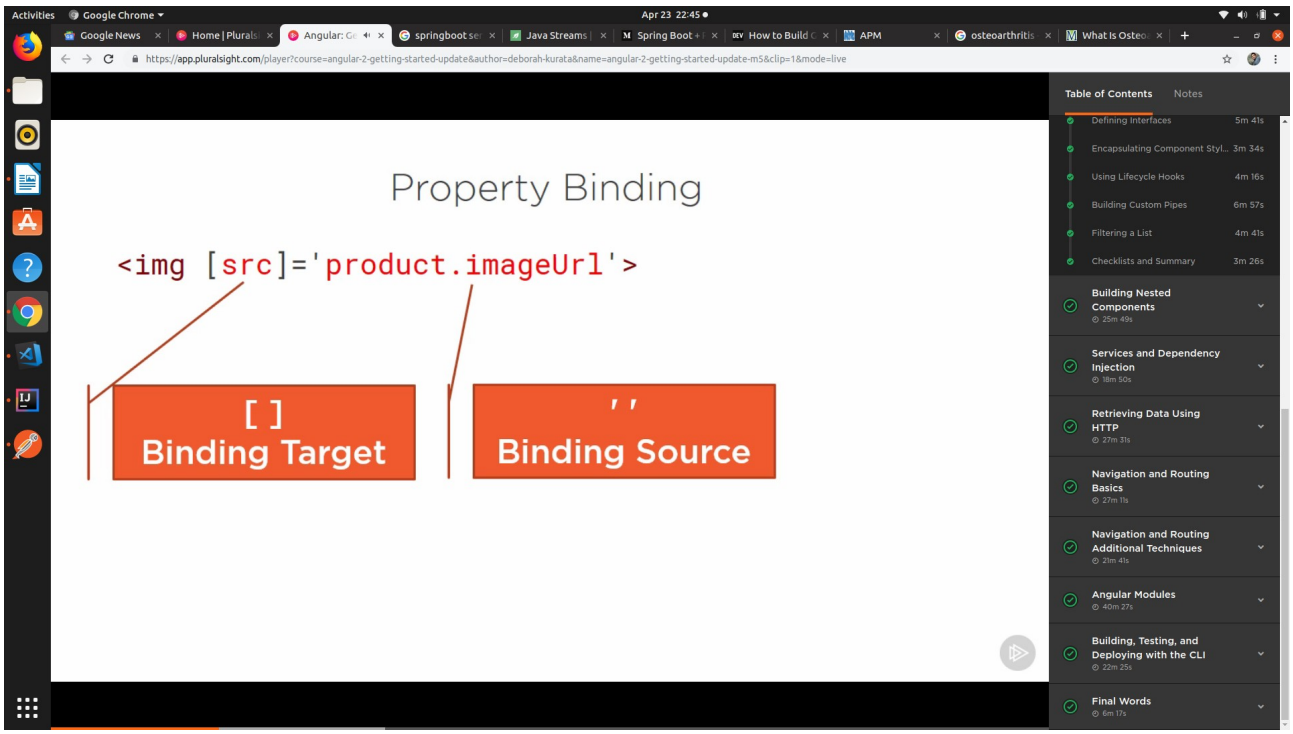
for Of VS for in ::



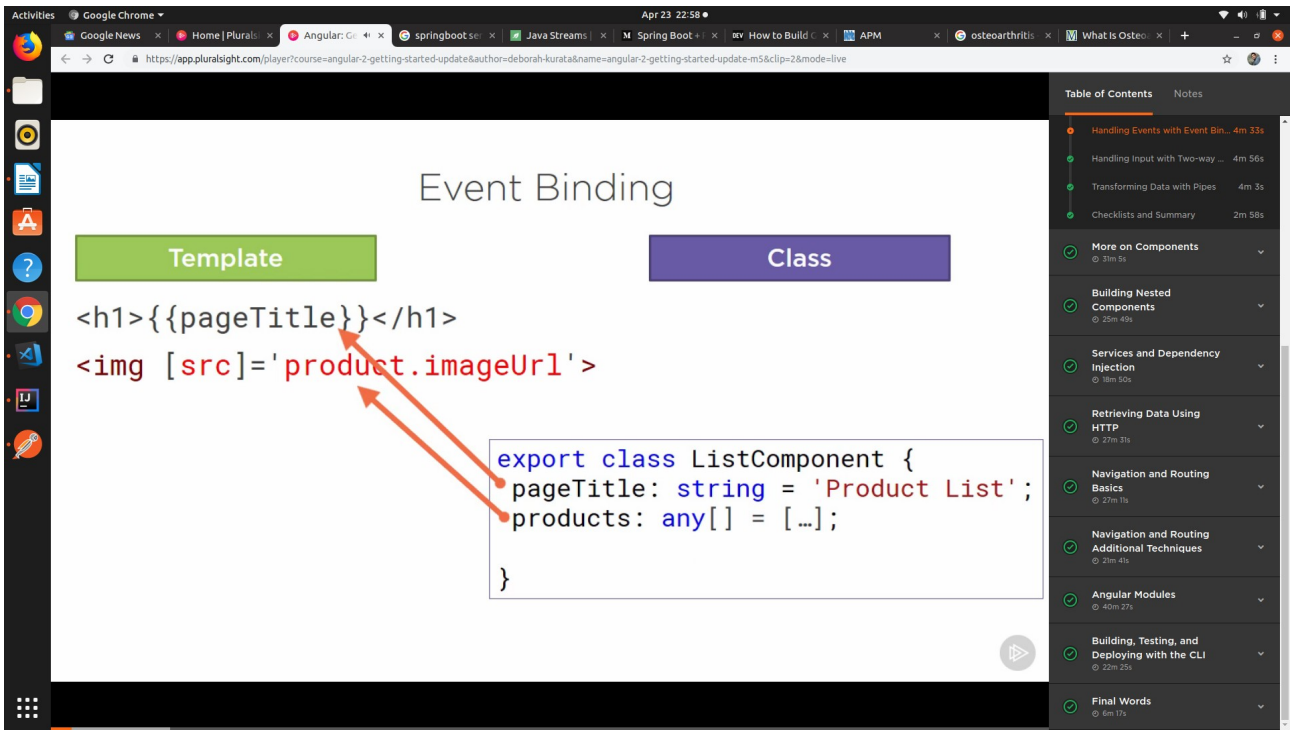
Data Binding and Pipes

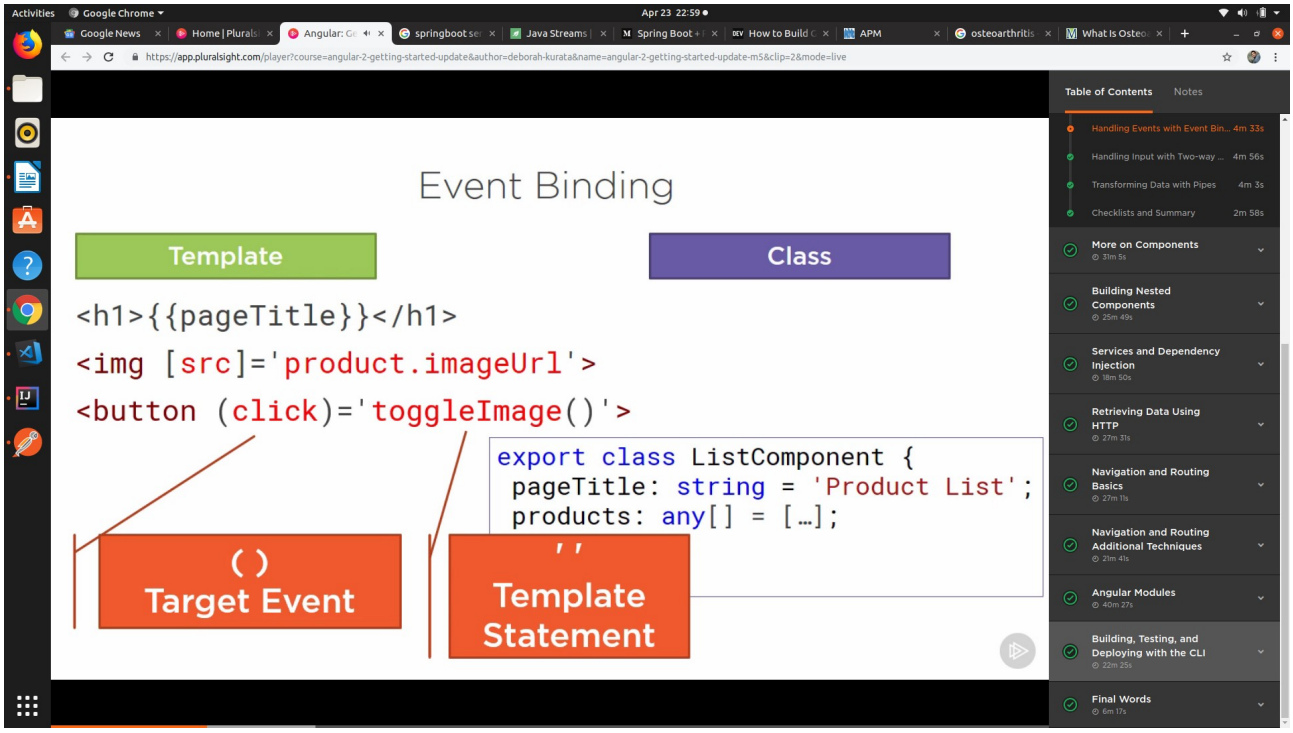
Binging:

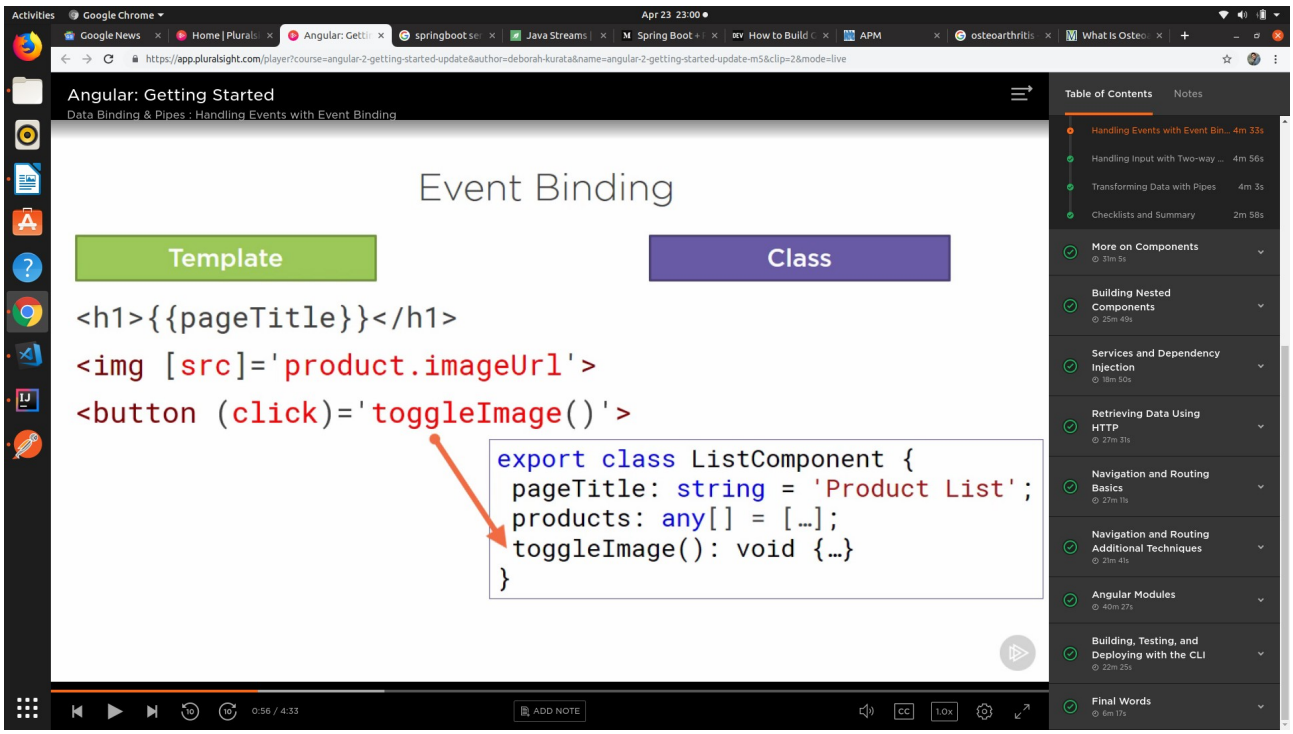




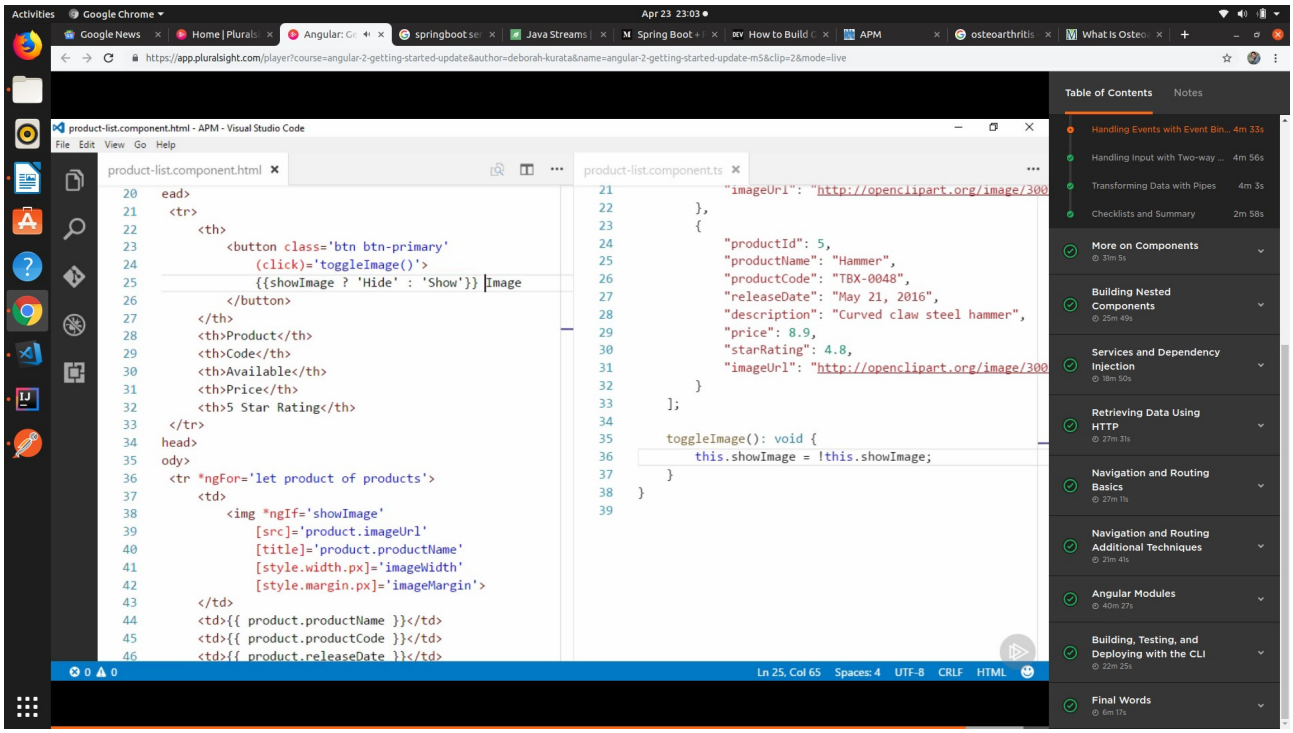
Event Binding:


Activities
Google Chrome
Apr 23 22:58
https://app.pluralsight.com/player?course=angular-2-getting-started-update&author=deborah-kurata&name=angular-2-getting-started-update-m5&clip=2&mode=live
Table of Contents
Notes
Handling Events with Event Bin... 4m 33s
Handling Input with Two-way ... 4m 56s
Transforming Data with Pipes 4m 3s
Checklists and Summary 2m 58s
More on Components
Building Nested Components
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Final Words
Event Binding
Template
Class
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
export class ListComponent {
pageTitle: string = 'Product List';
products: any[] = […];
}

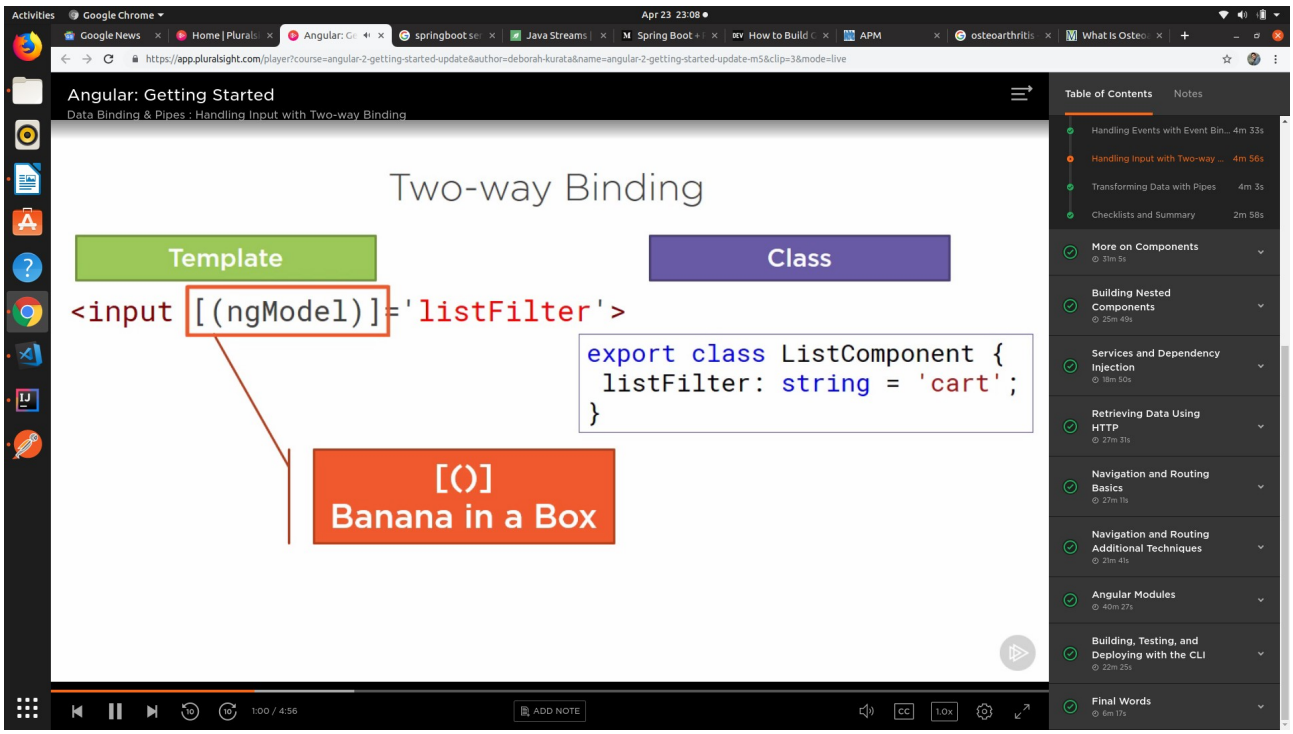

Activities
Google Chrome
Apr 23 22:59
https://app.pluralsight.com/player?course=angular-2-getting-started-update&author=deborah-kurata&name=angular-2-getting-started-update-m5&clip=2&mode=live
Table of Contents
Notes
Handling Events with Event Bin... 4m 33s
Handling Input with Two-way ... 4m 56s
Transforming Data with Pipes 4m 3s
Checklists and Summary 2m 58s
More on Components
Building Nested Components
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Final Words
Event Binding
Template
Class
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
<button (click)='toggleImage()'>
export class ListComponent {
pageTitle: string = 'Product List';
products: any[] = […];
( )
Target Event
' '
Template Statement



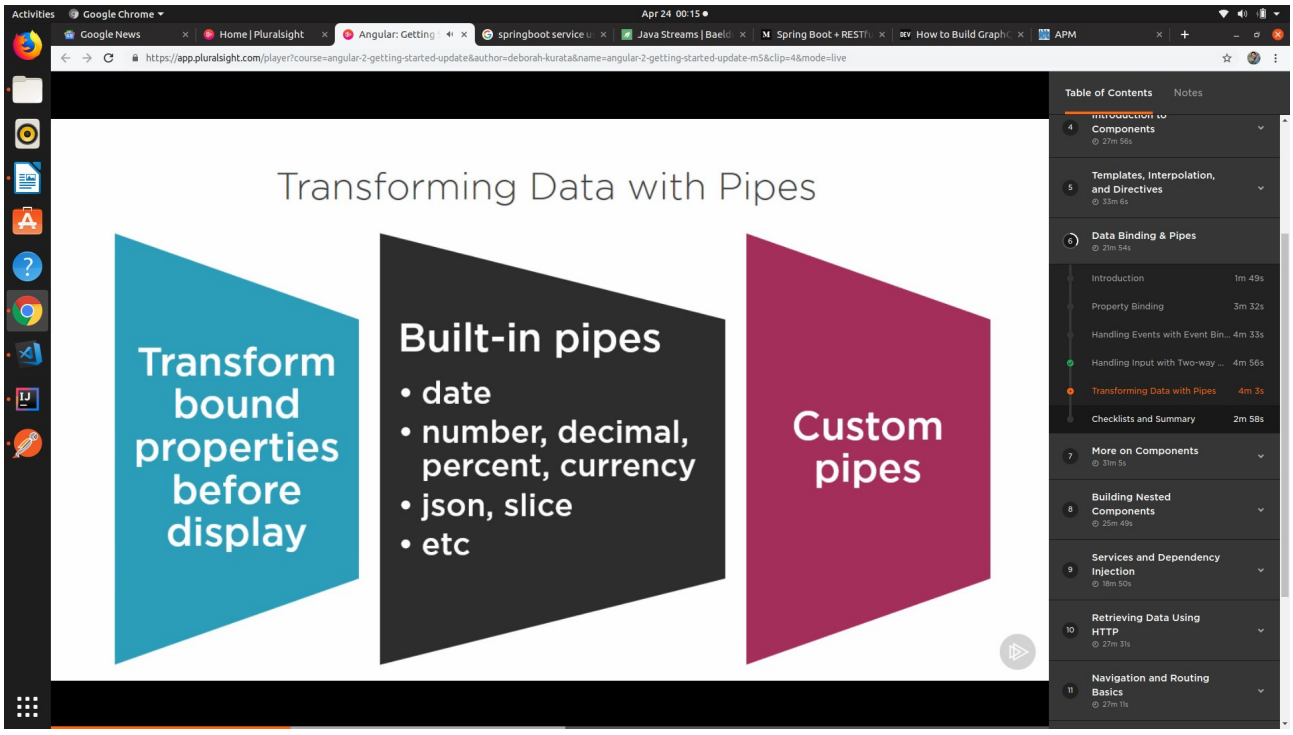
example: {{showImage?'Hide' : 'Show'}} Image

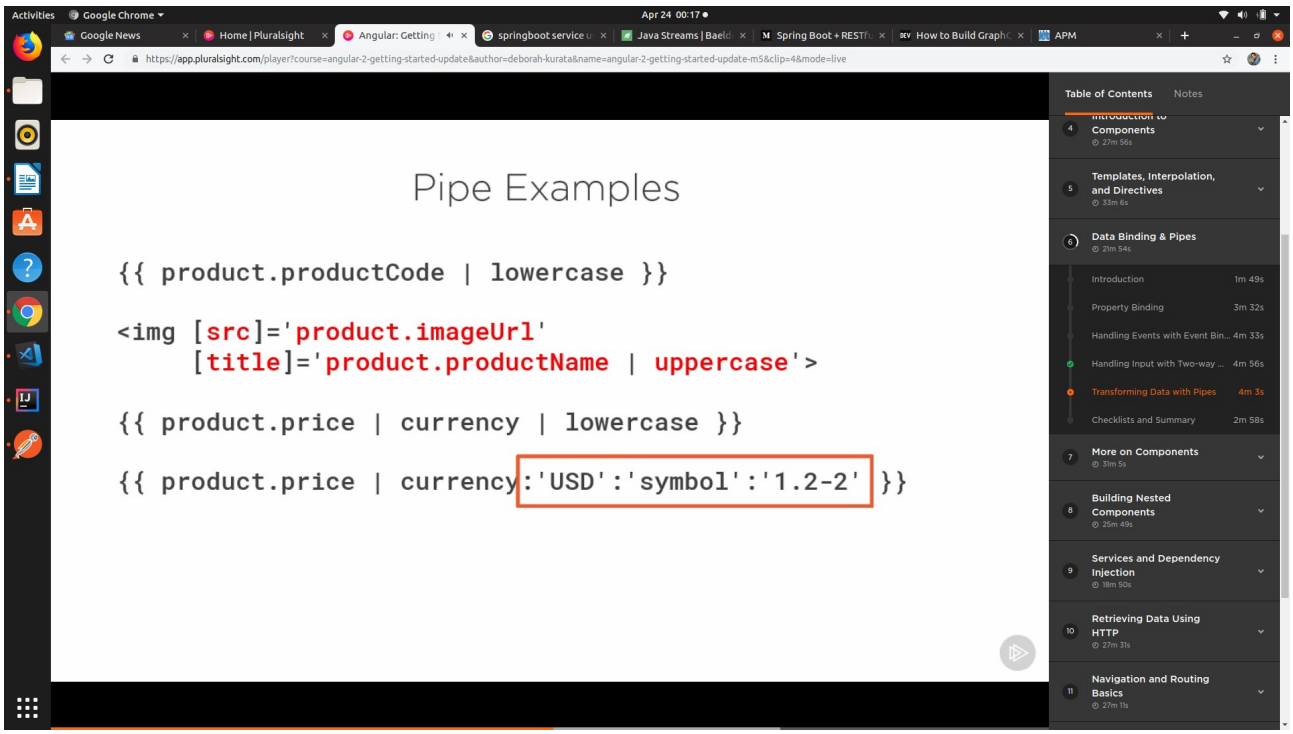


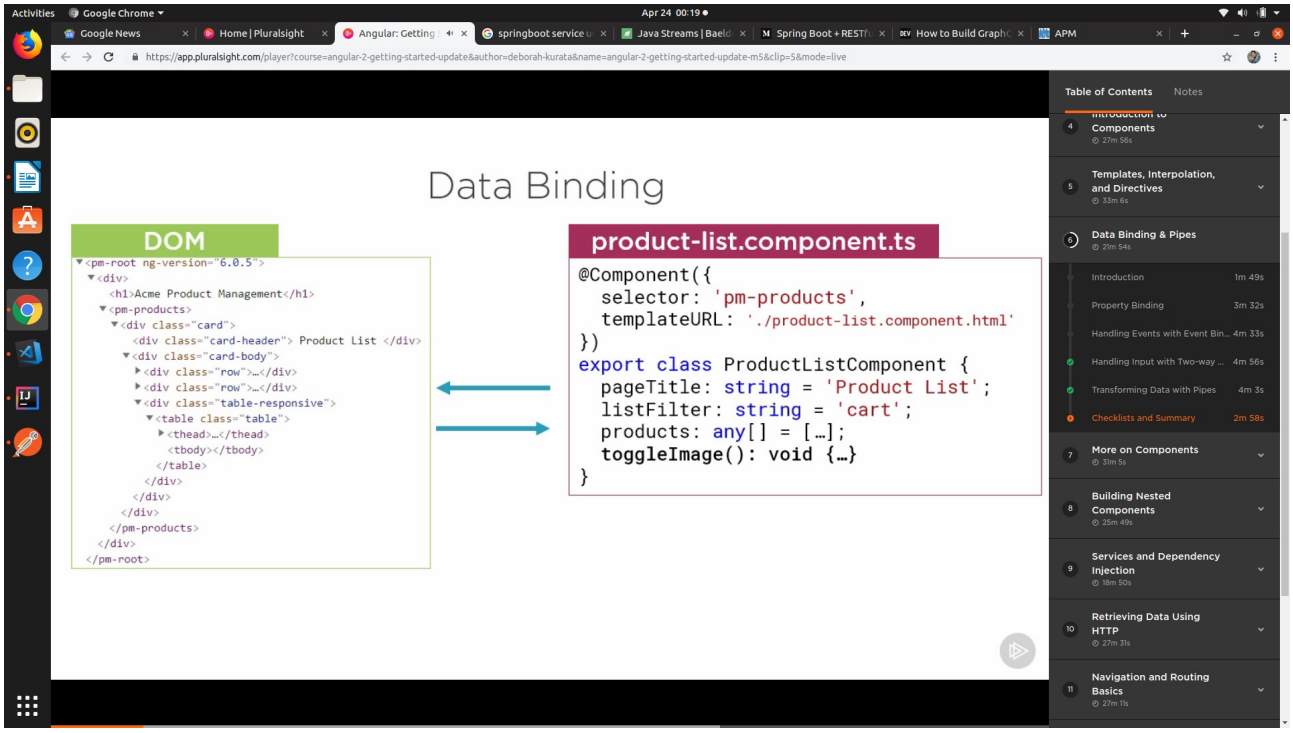
Two Way Binding: It uses FormModule


Angular: Getting Started
Data Binding & Pipes : Handling Input with Two-way Binding
Two-way Binding
Template
Class
<input [(ngModel)]='listFilter'>
export class ListComponent {
listFilter: string = 'cart';
}
[()]
Banana in a Box
Table of Contents
Notes

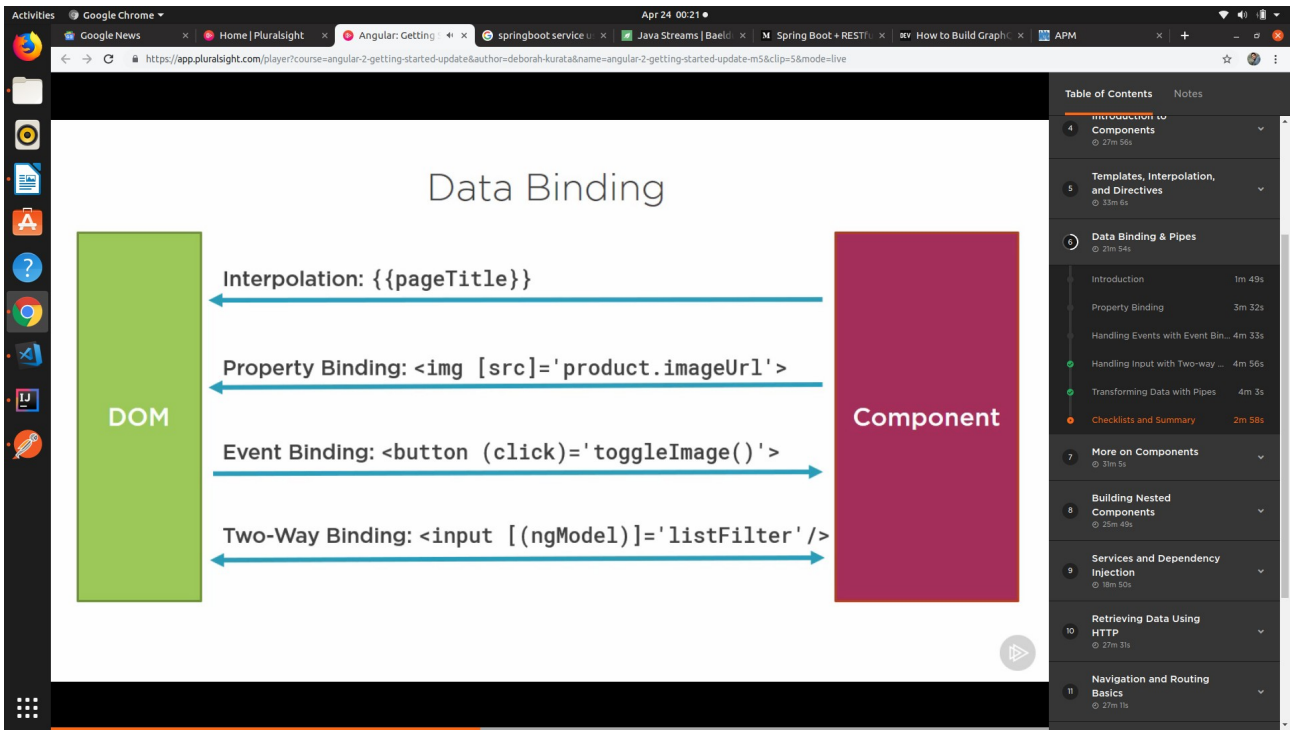
Pipes::


Transforming Data with Pipes
Transform bound properties before display
Built-in pipes
• date
• number, decimal, percent, currency
• json, slice
• etc
Custom pipes
Table of Contents
Notes

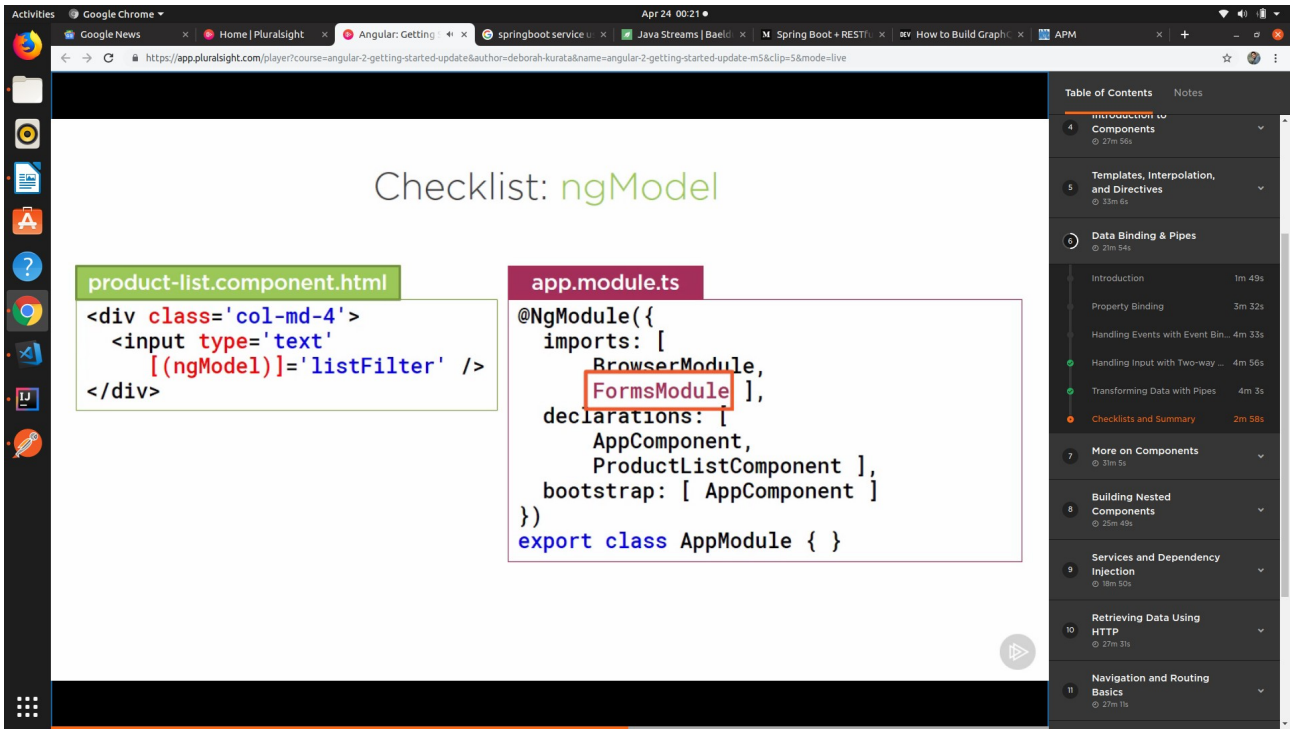


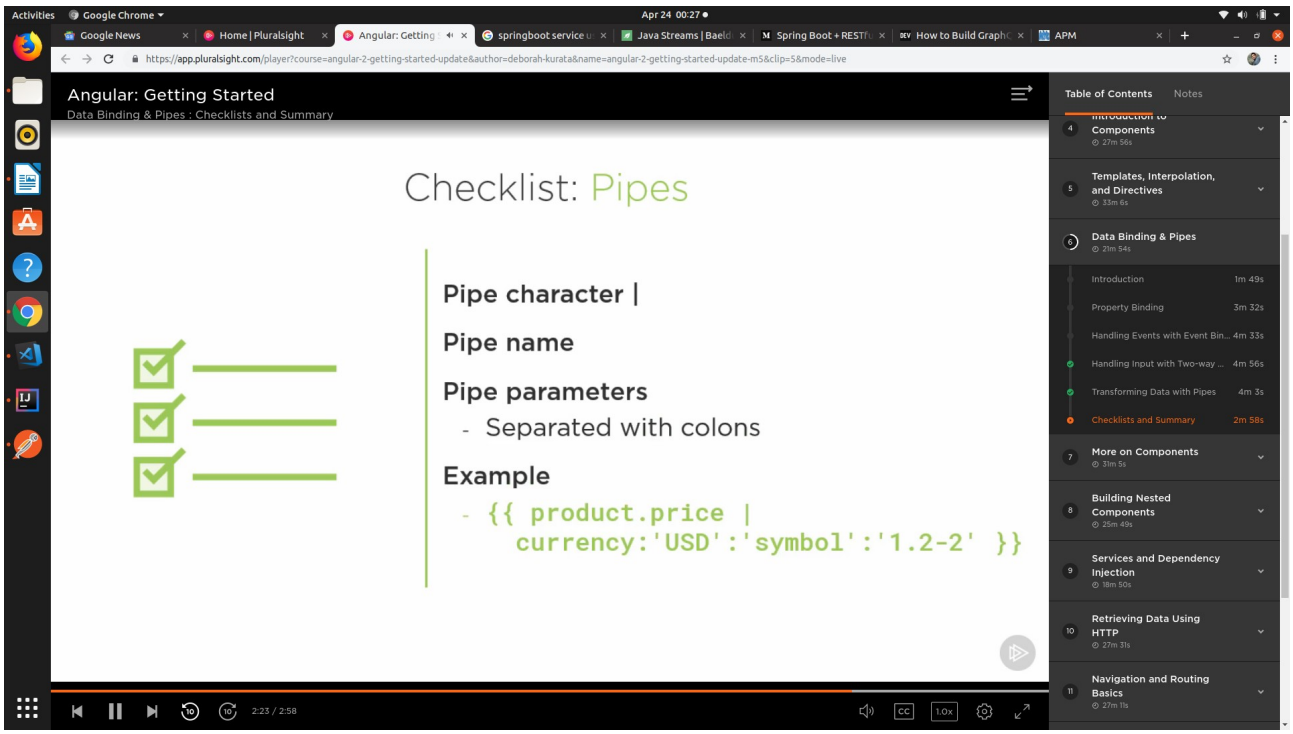


CheckList for Binding and Pipes::

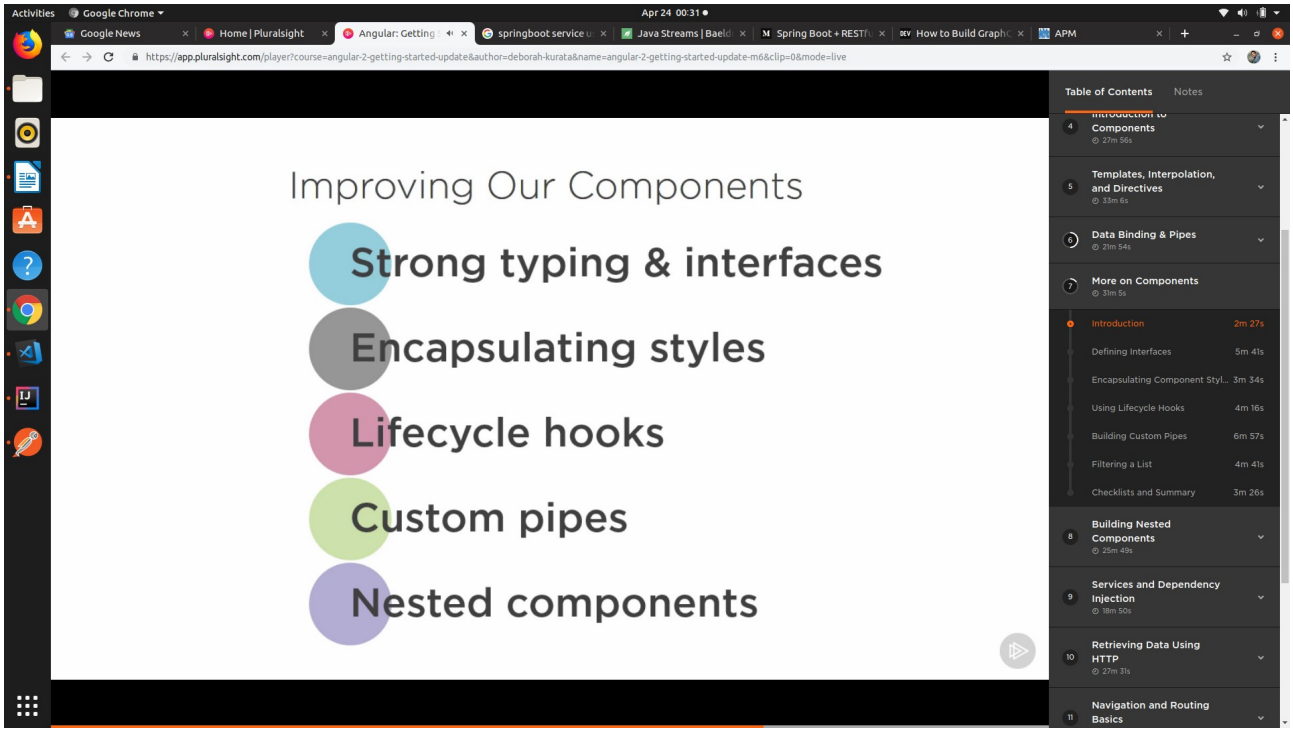


ngModule:Banana in box in html and FormsModule in AppModule are prerequiste

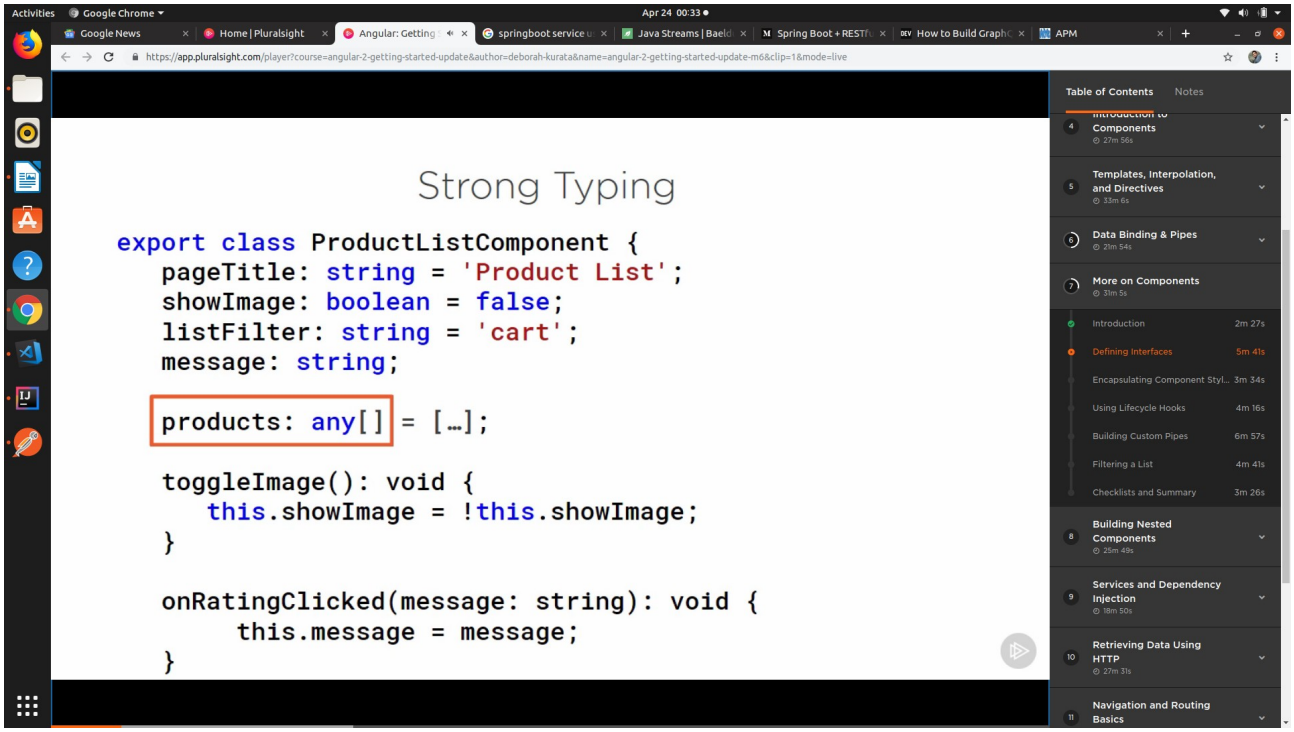




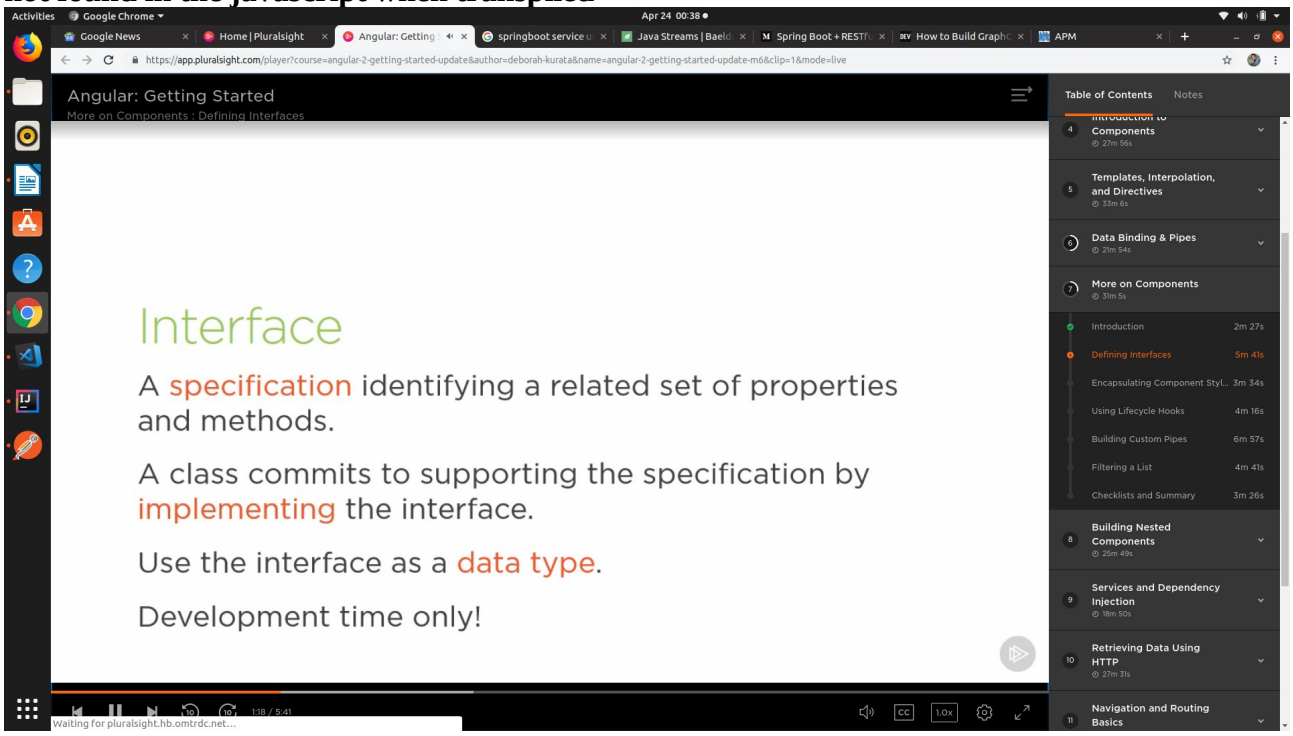
More on Components



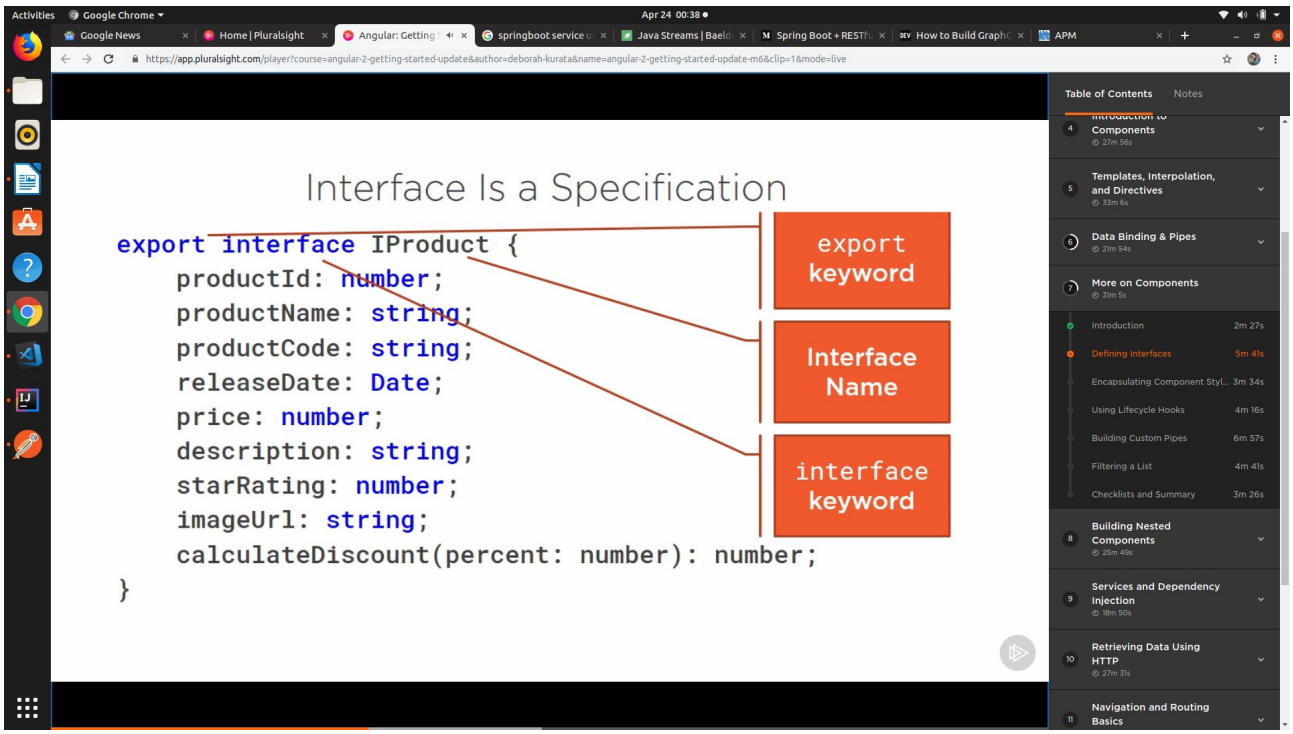
Strong Type: Like page title field has a type as string, method toggleImage() has a void type. Products has unknown type any.



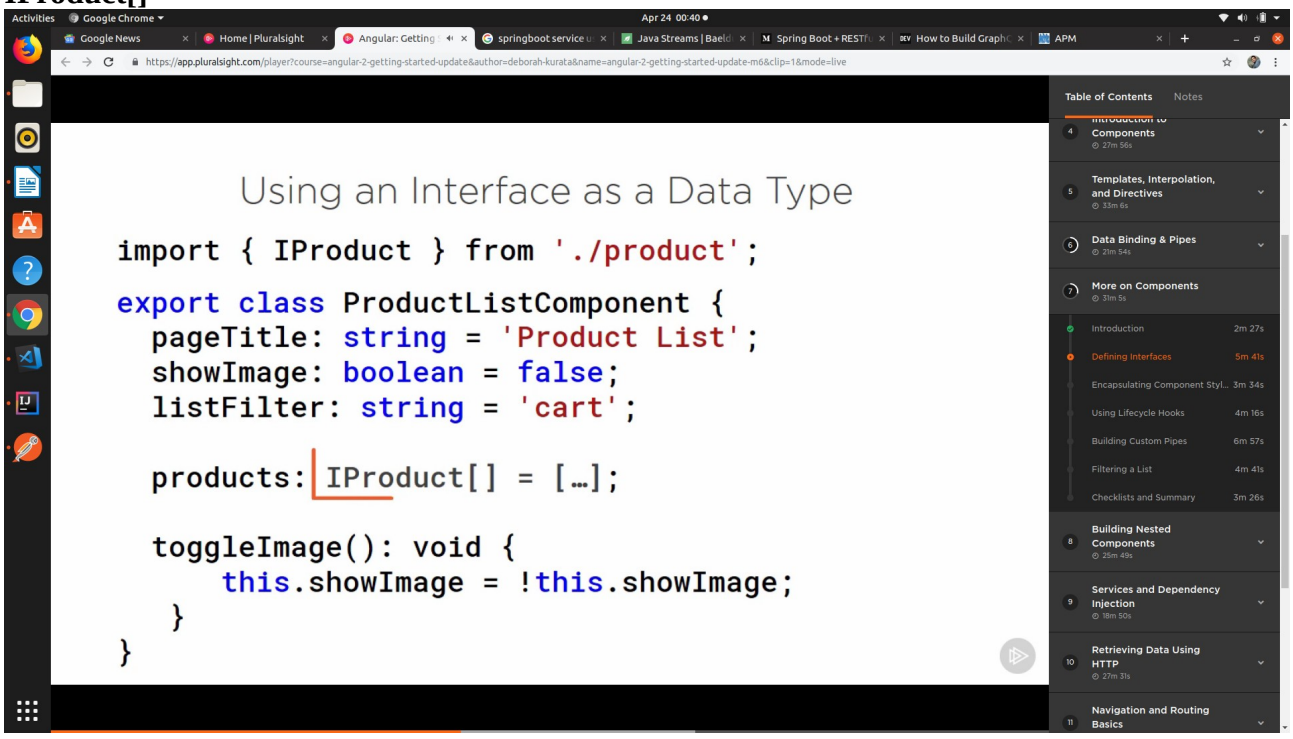
<u>Interface:</u>These are not supported by ES5 or ES2015 but supported by Typescript so they are not found in the javascript when transpiled



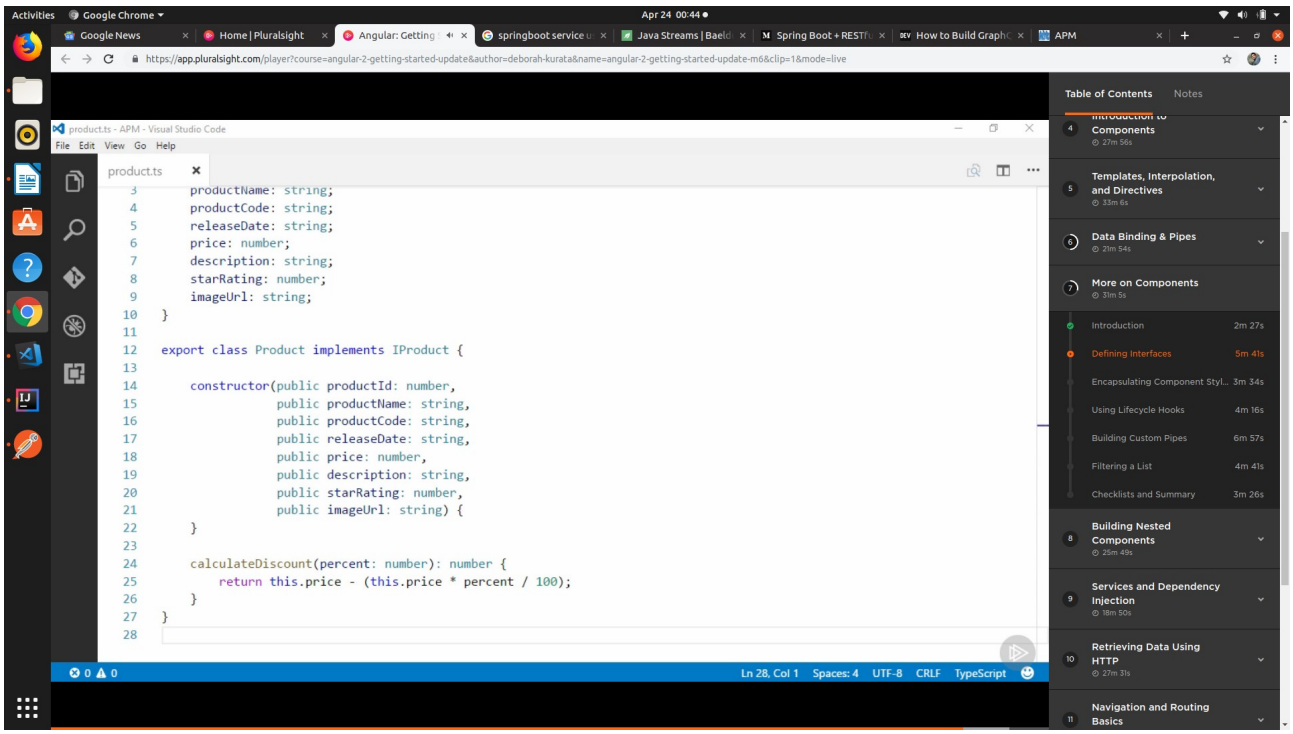
Example of interface: export keyword so that it can be used in other places as well



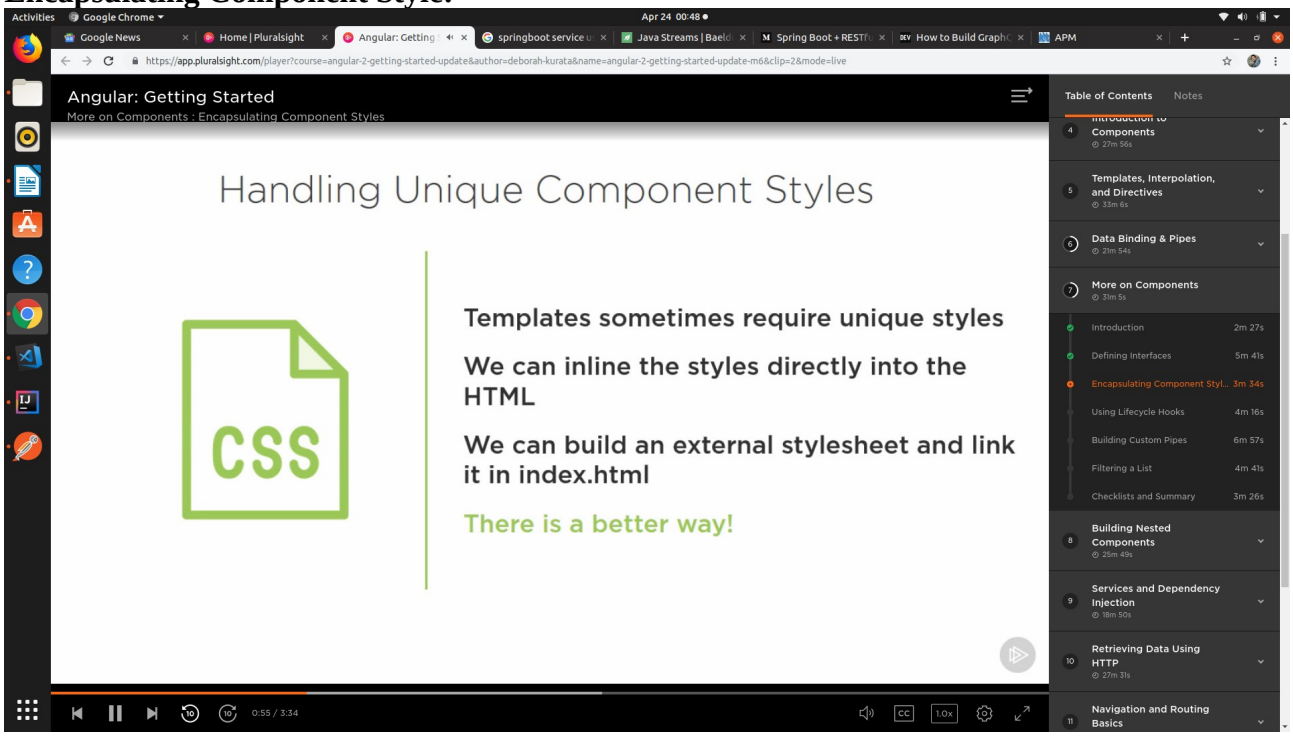
In the below example interface Iproduct is imported and used as a datatype by products: IProduct[]



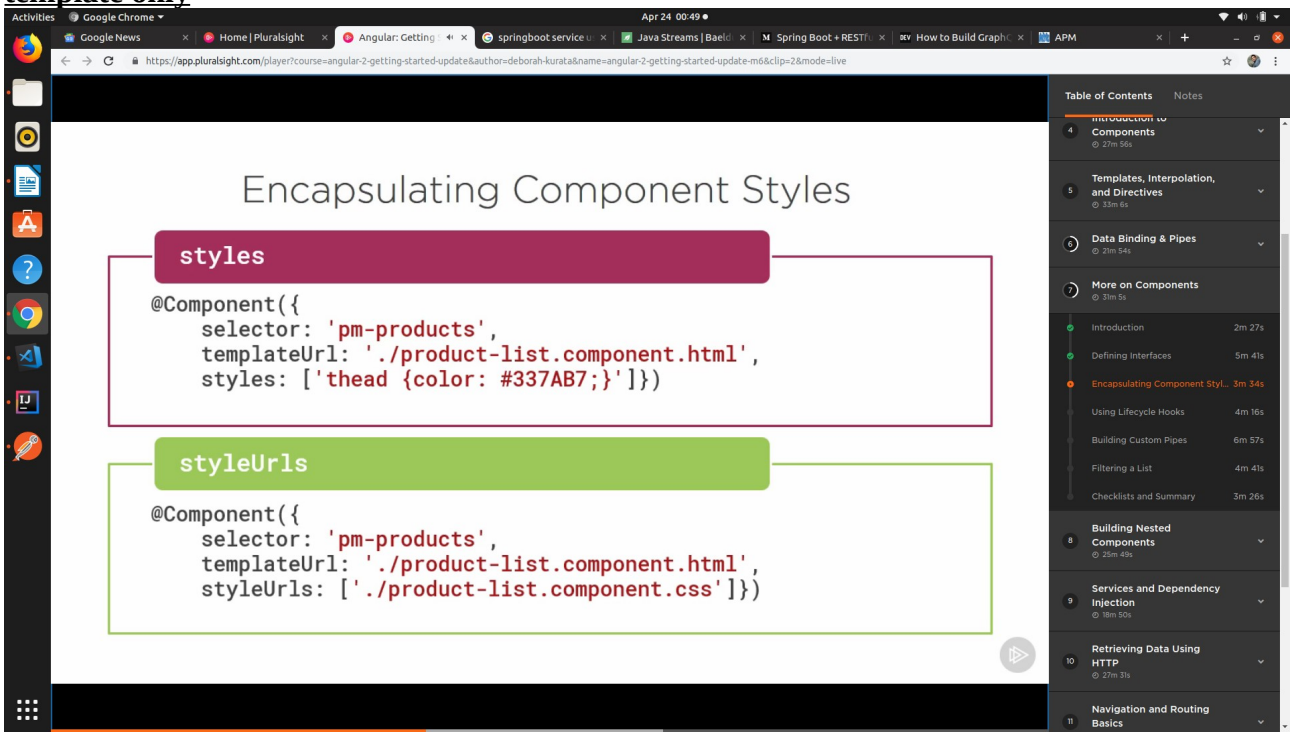
we can ad a class to our interface as shown below. We create a class when we require a method that will be used throught the application we use class. Ex in the below example calculateDiscount() method



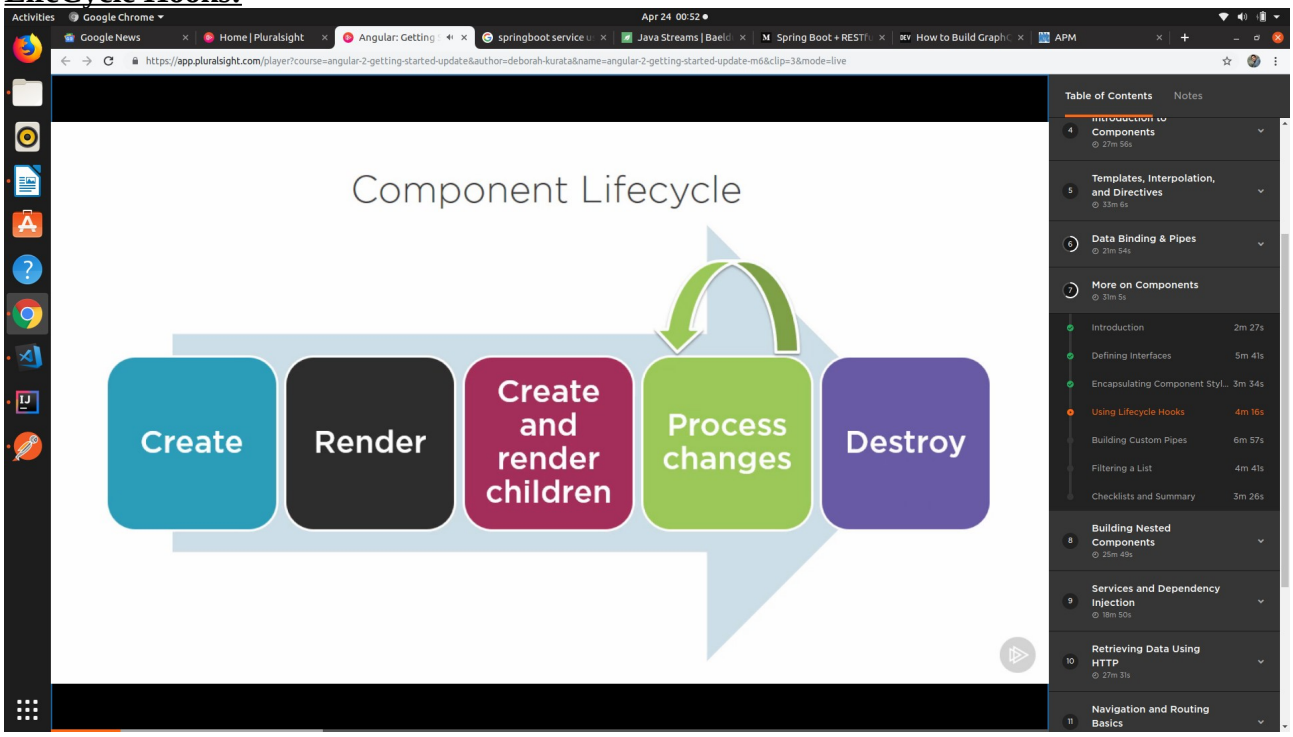
Encapsulating Component Style:



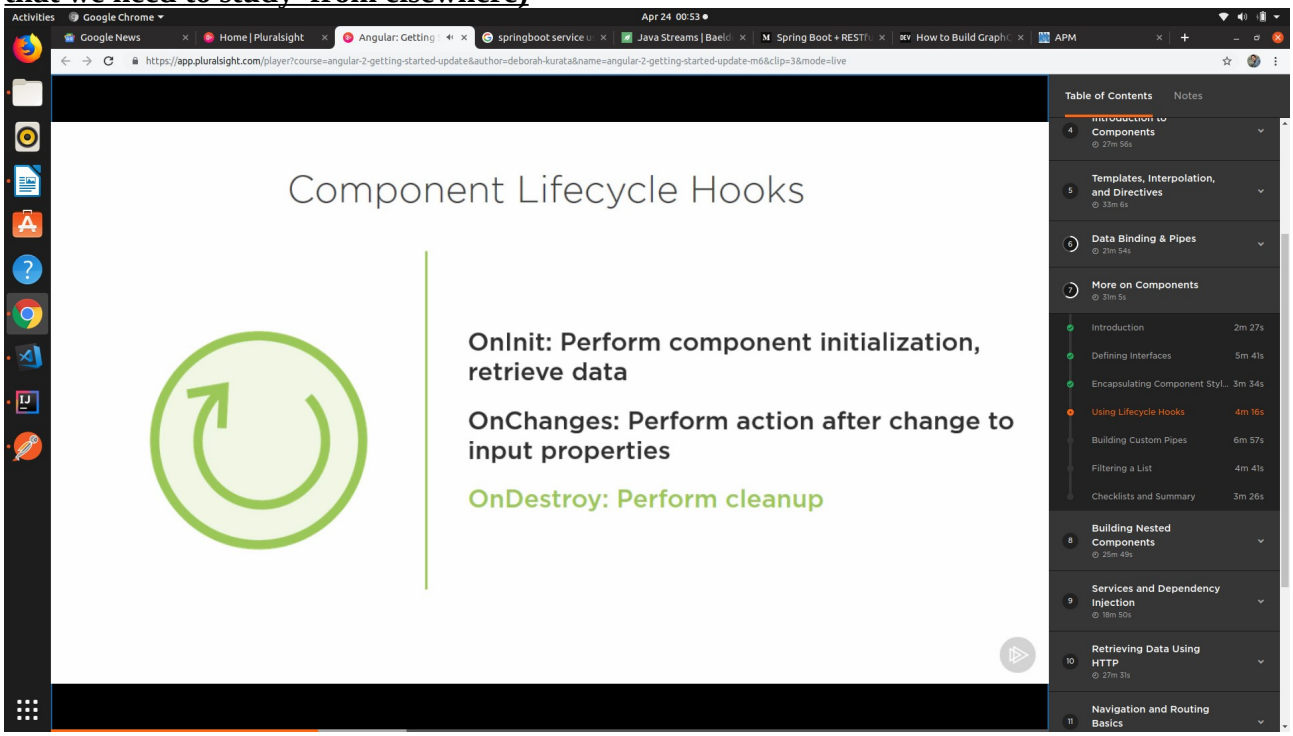
Green is the better option from the below image but it is restricted to the applied component template only



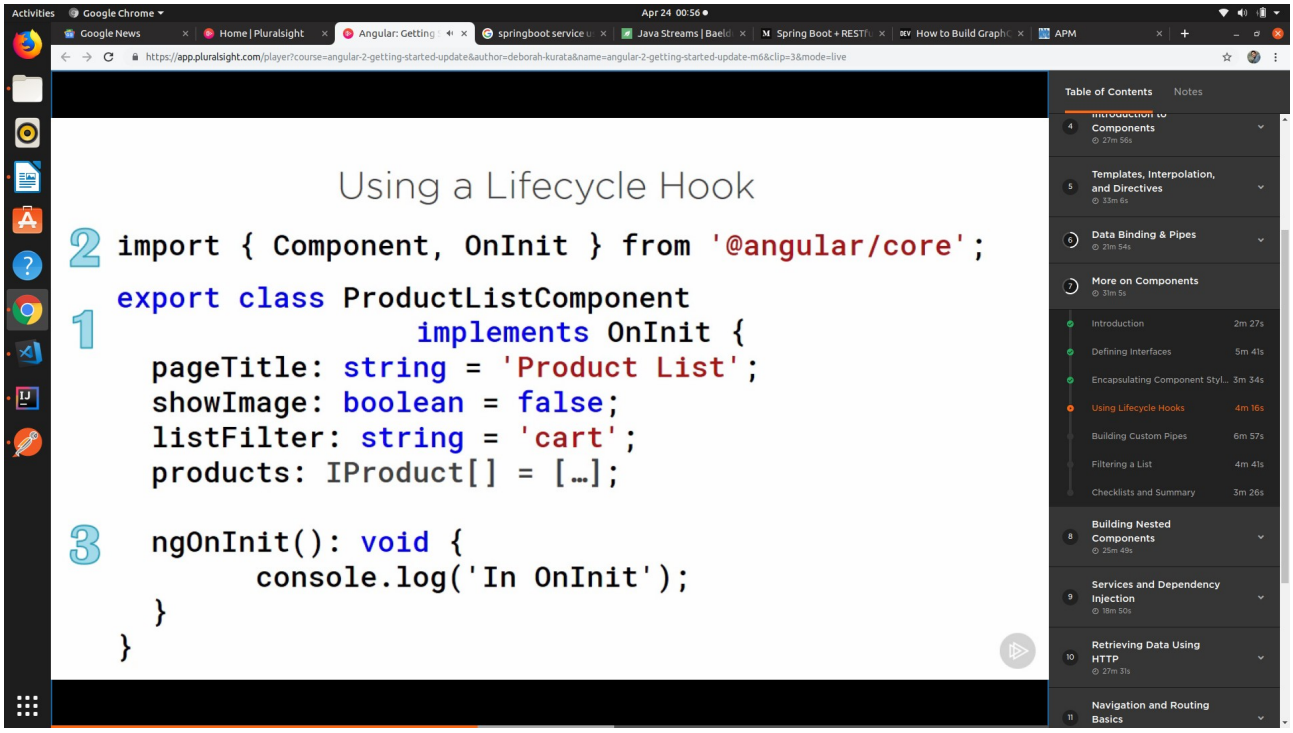
LifeCycle Hooks:



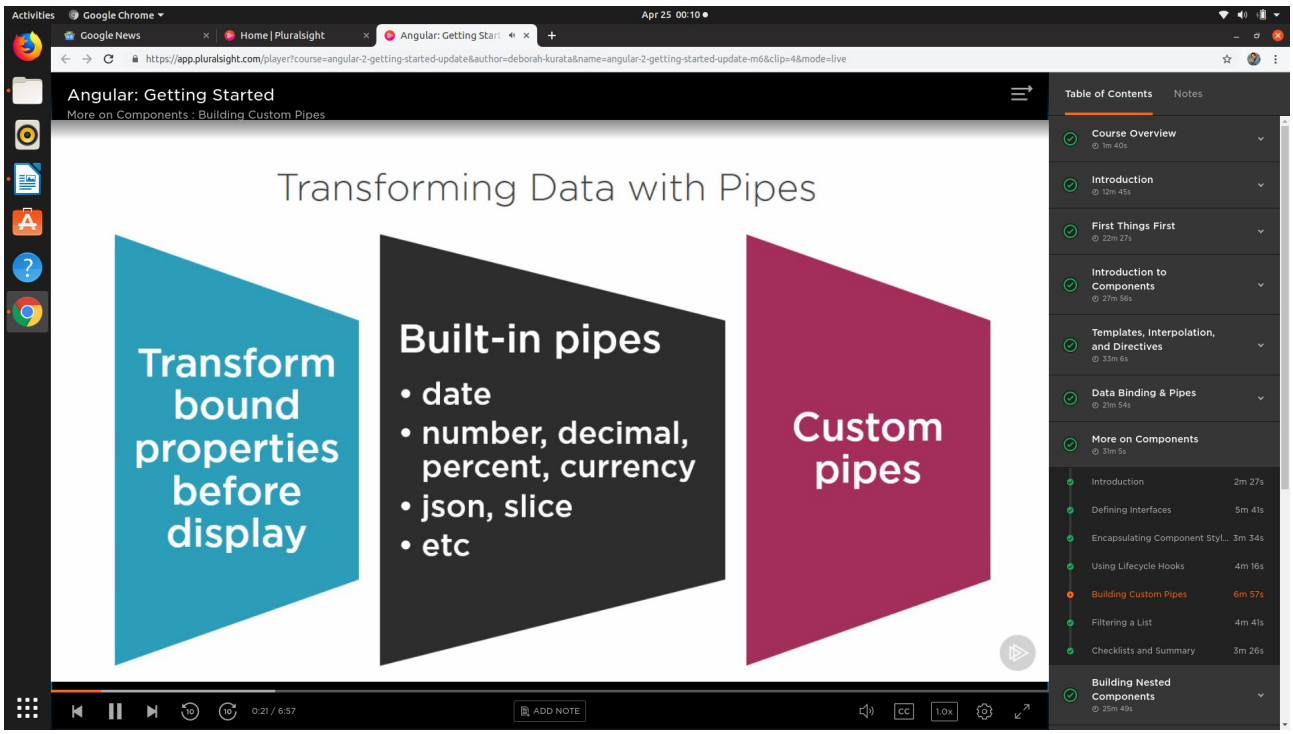
Some of the most widely used life cycle hooks shown below(they are not all some are not listed that we need to study from elsewhere)



As we know these lifecycle hooks are interfaces so they will not supported by ES2015 or ES5 so its not require to implement it on the component though we can implement it for clarity if we want



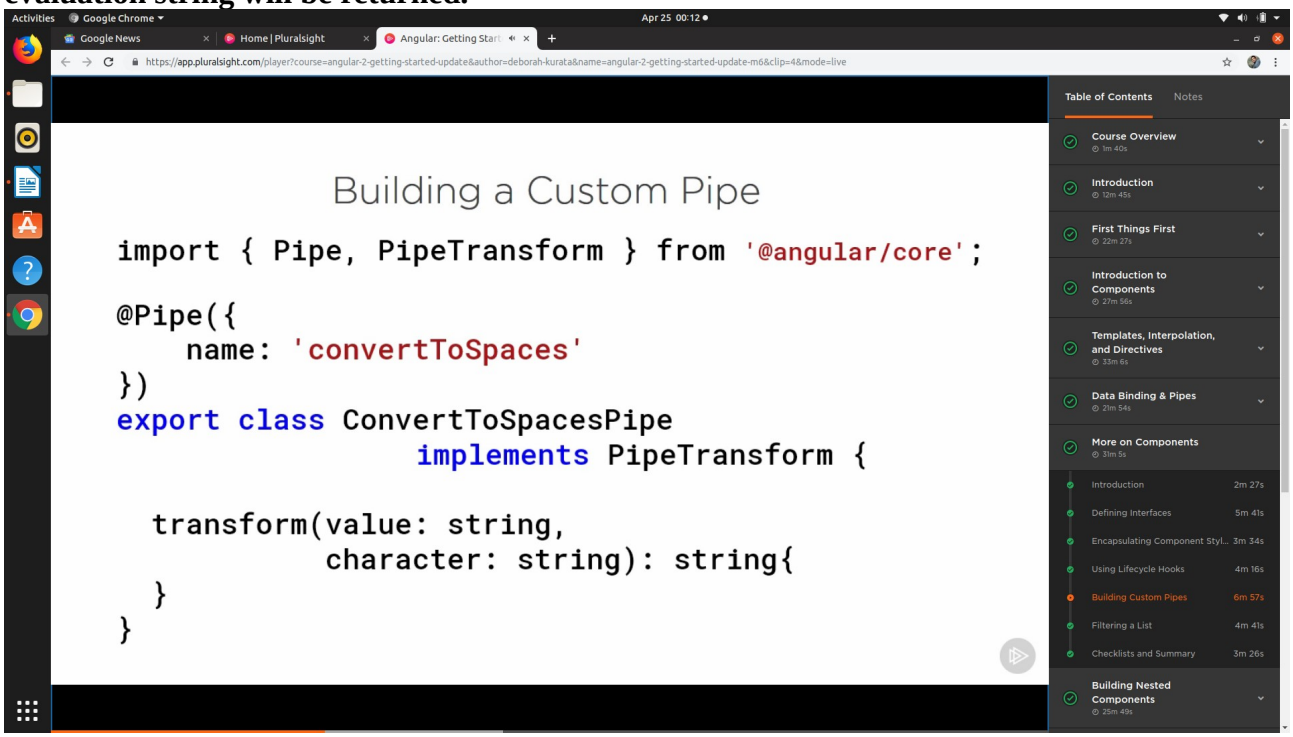
Custom Pipes :



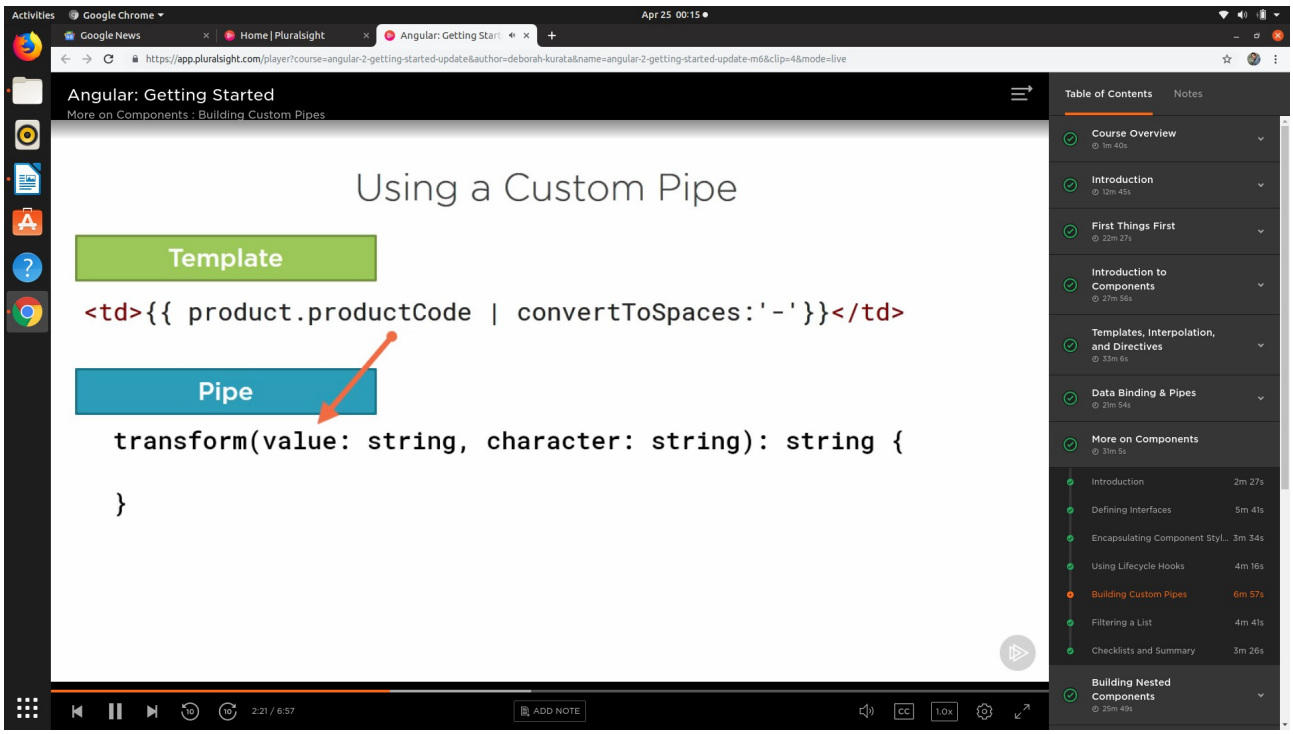
Creating a custom pipe we need to do the below steps

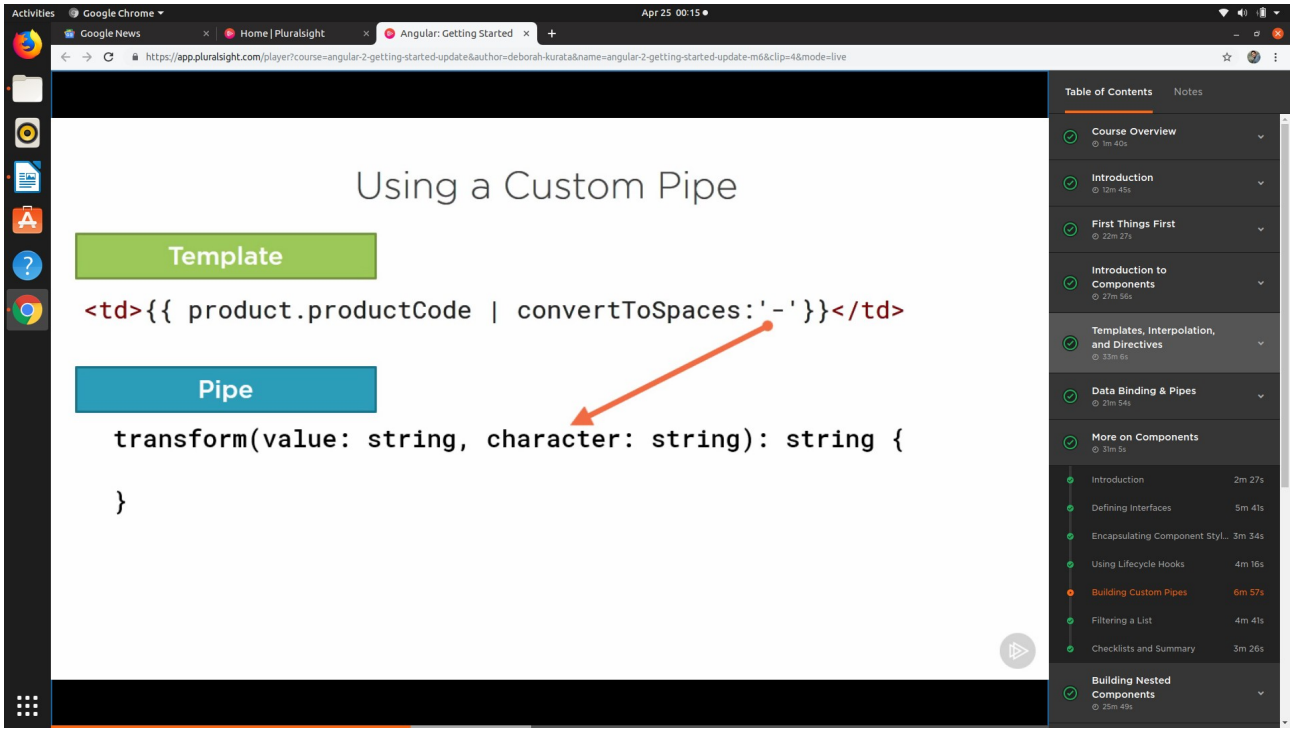
@Pipe(It is a decorator) is a method inside we are giveing the name of the pipe that will be referred in html

transform() method has a value that needs to be transformed, charcter is the field that is transformed like in value there is a – in the field and we will replace it with space(it is thc haracter feild in the method parameter) return type is string , which means after all evaluation string will be returned.

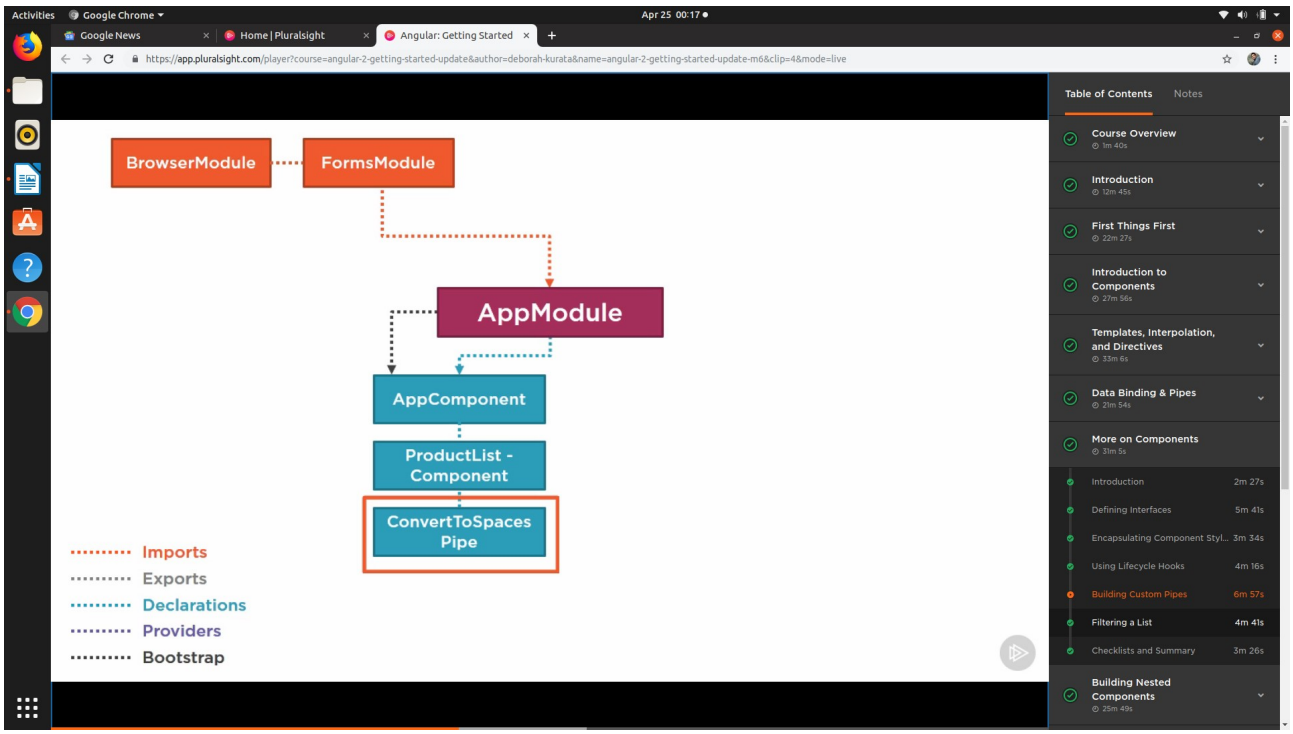


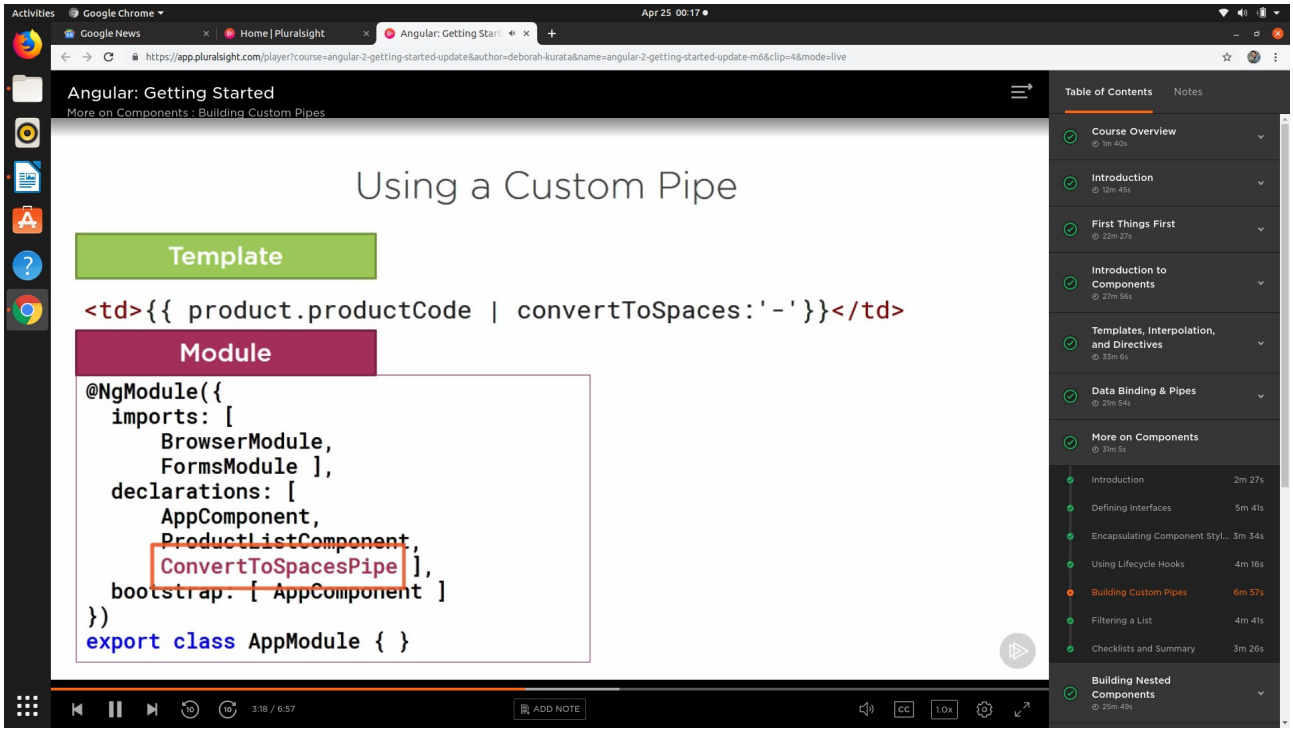
More explanation



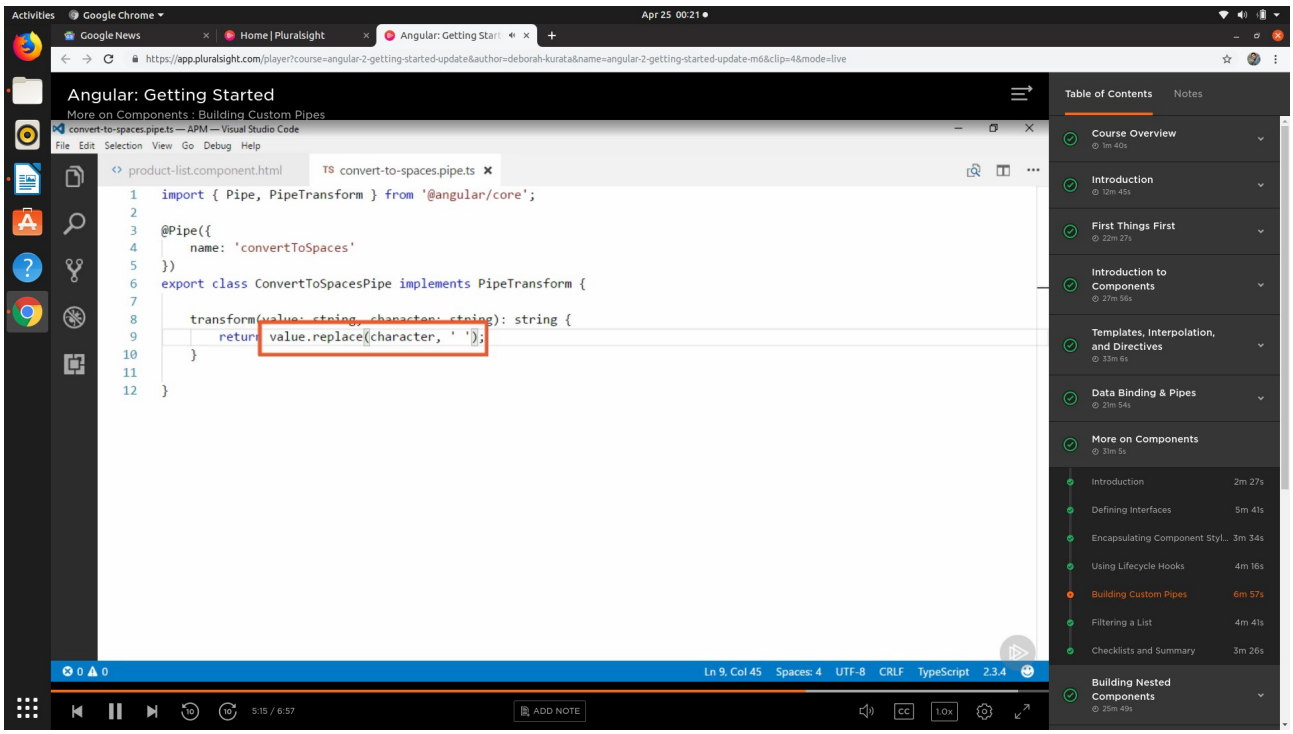


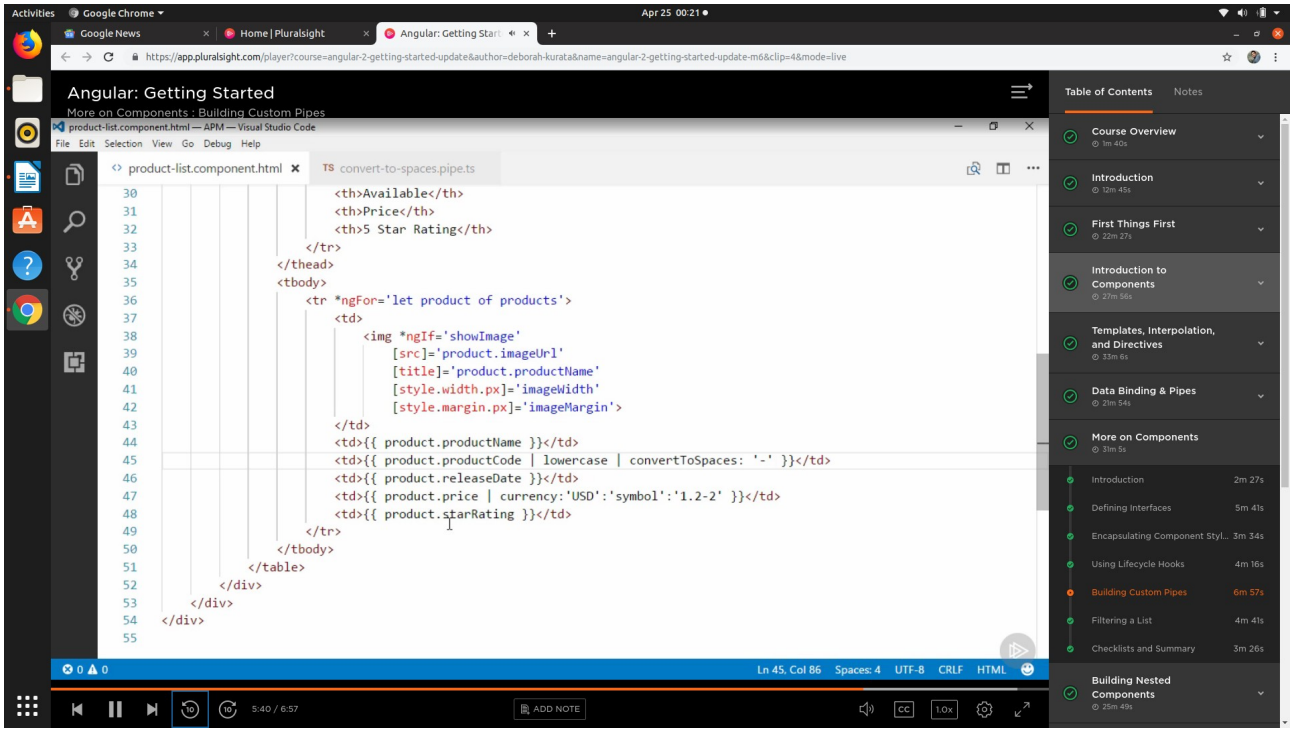
This pipe is added in the angular module otherwise it will not be visible





Code example

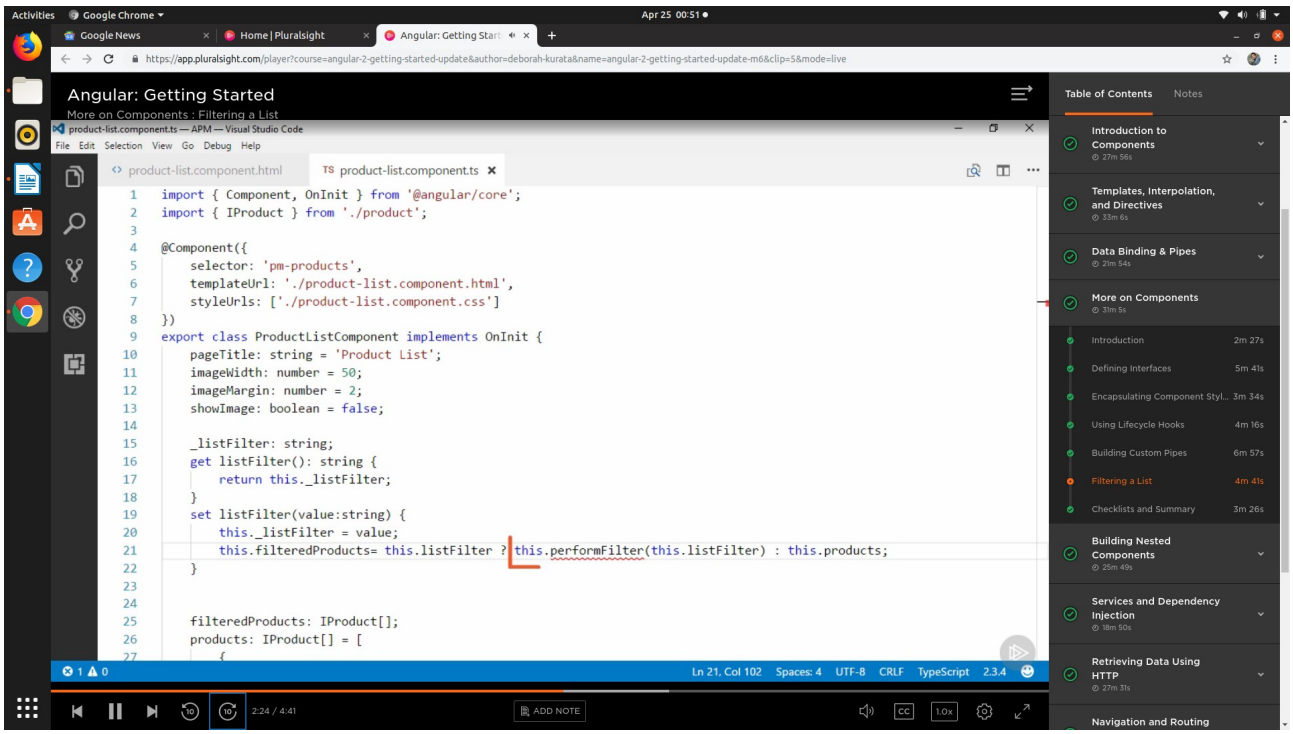




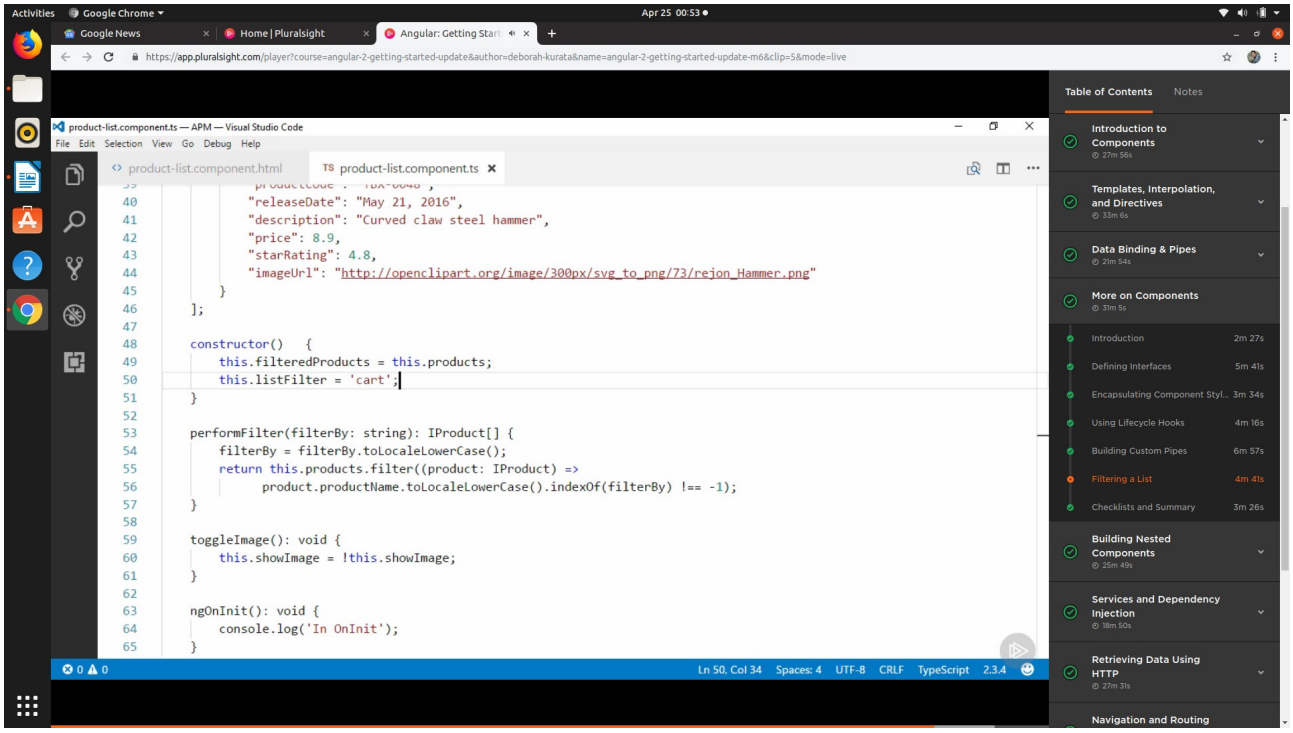
Custom Filter:

1)Created a filteredProducts to keep the result

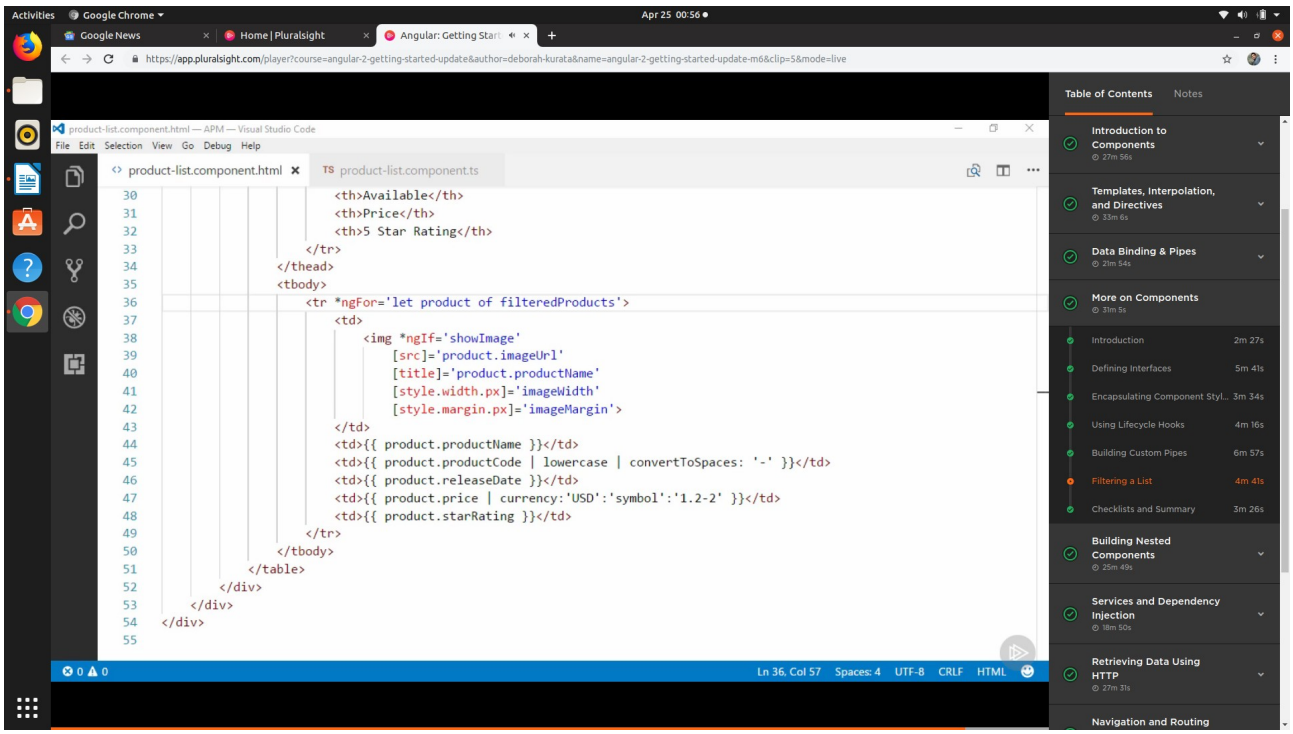
2)changed the filtered property with setter and getter methods



Now a new method performFilter() is added and in constructor we assing the filteredProducts with default value i.e which has the full list of content as filteration will change the data

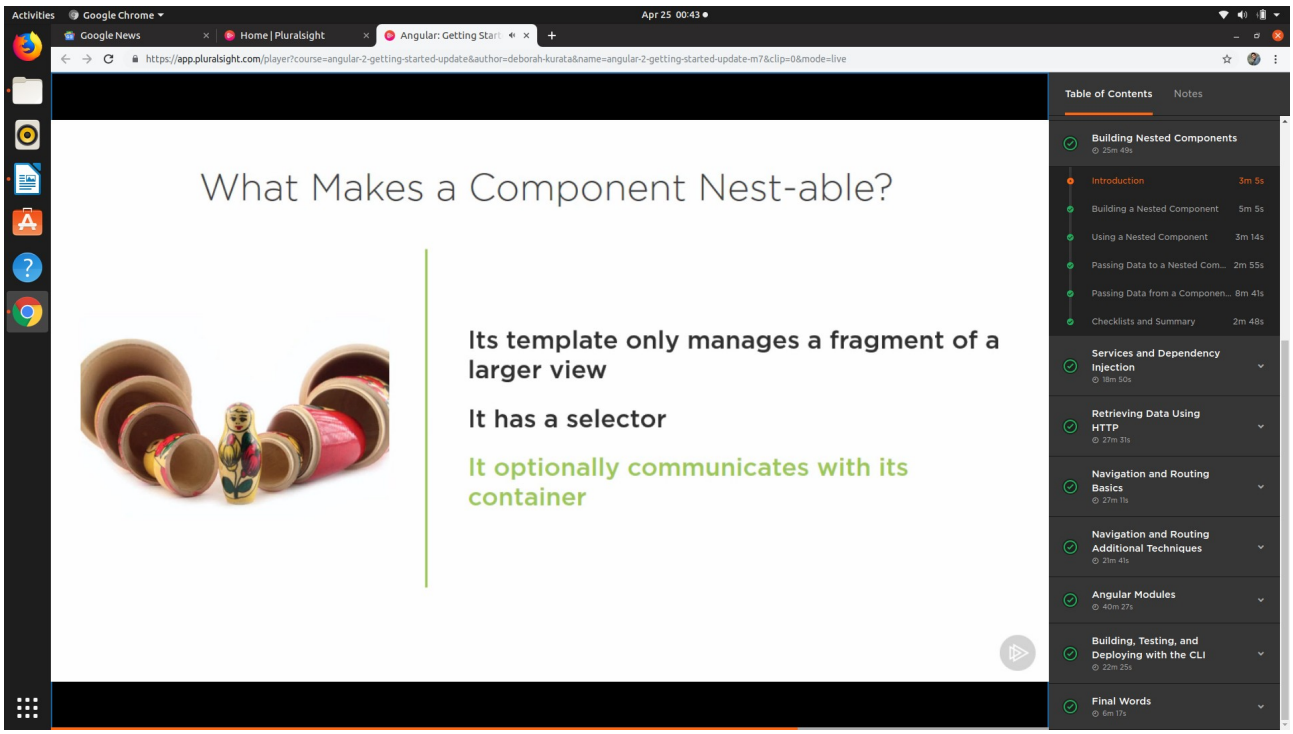


In html change let products of products to filteredProducts

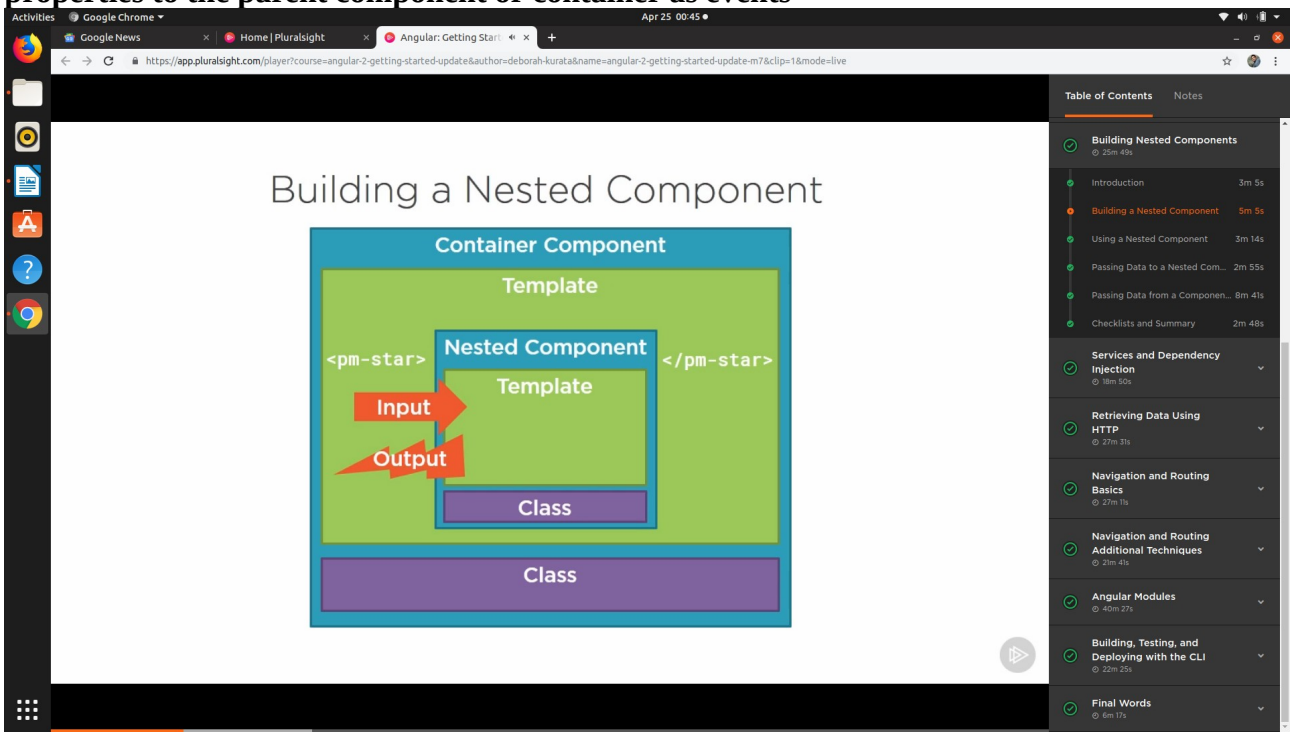


Building Nested Components

Conditions:

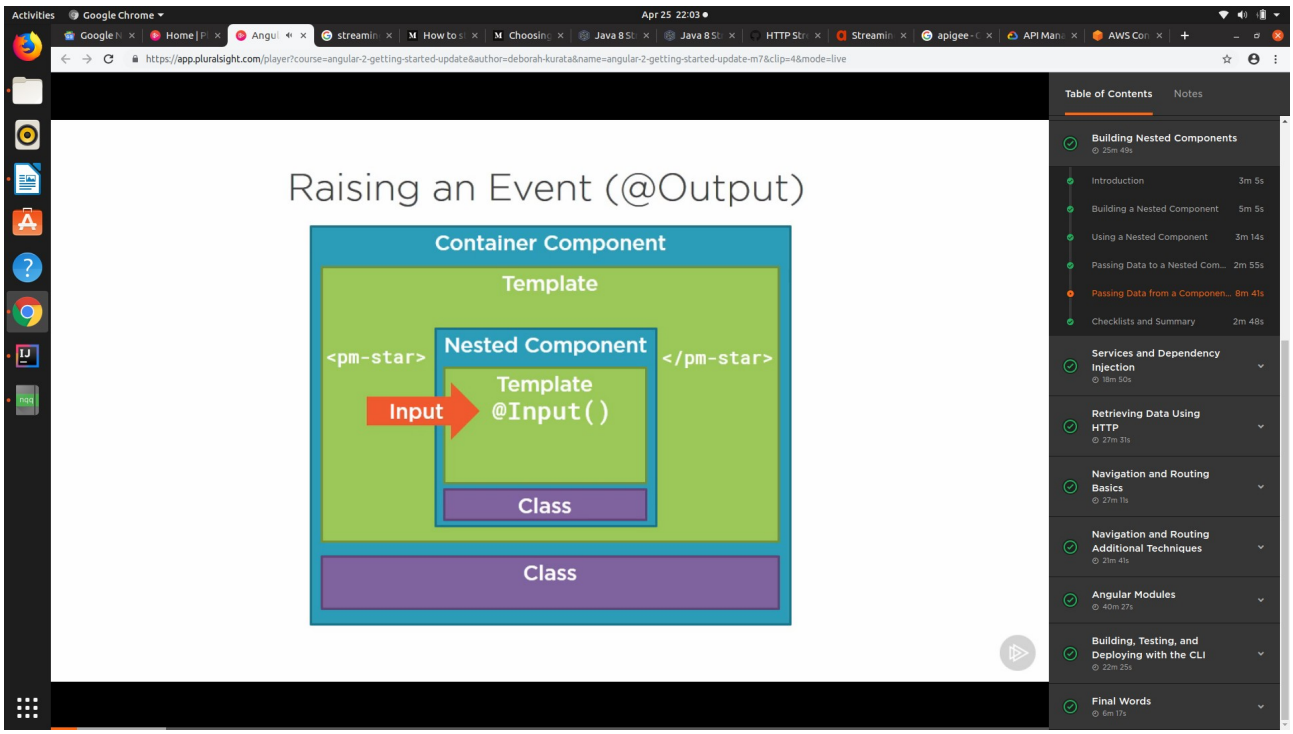


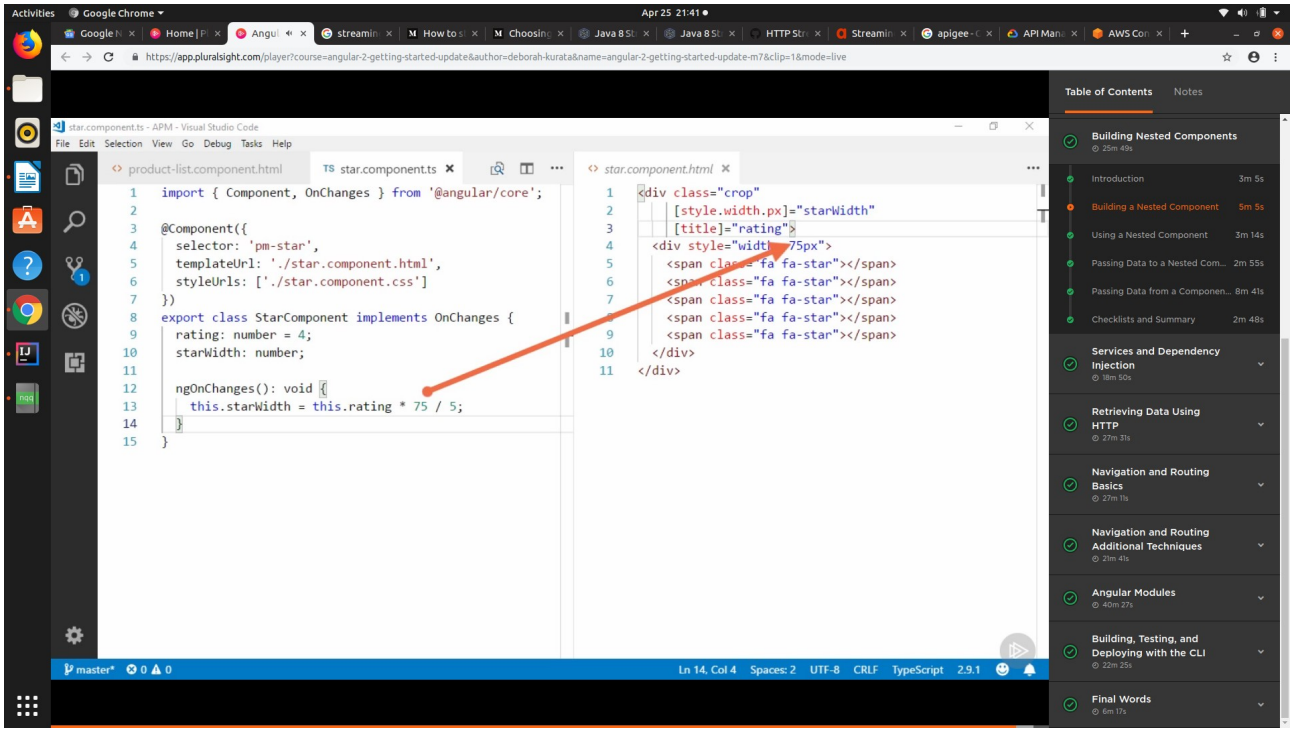
With the use of directive nested component is added to the template. Nested component receives the information from its parent component using input properties and it outputs its properties to the parent component or container as events



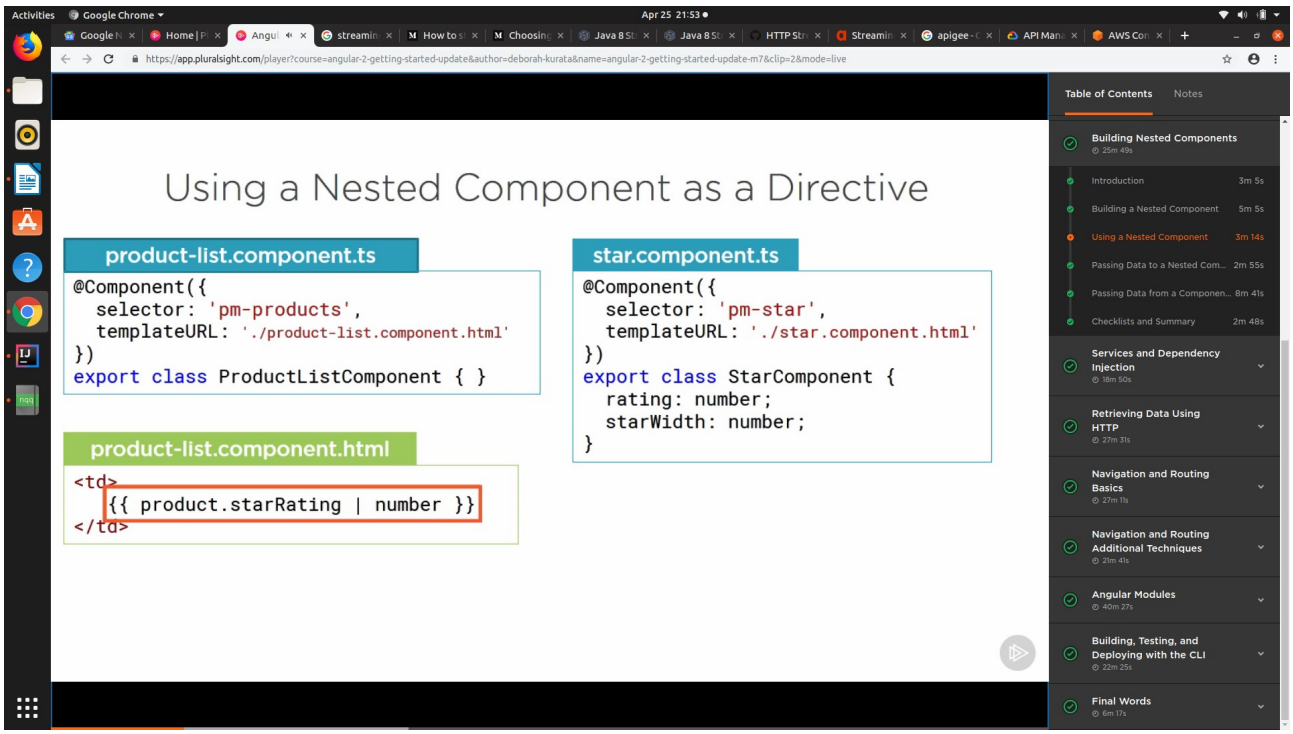
Sample Application for Nested Component:

Create a new star component as per our requirement(it will show rating as stars, earlier rating was coming in form of decimal numbers)
Below Picture is for sending data from one component to another using Input not output , picture is for reference
Below example is for passing value to the nested component which is inside a container

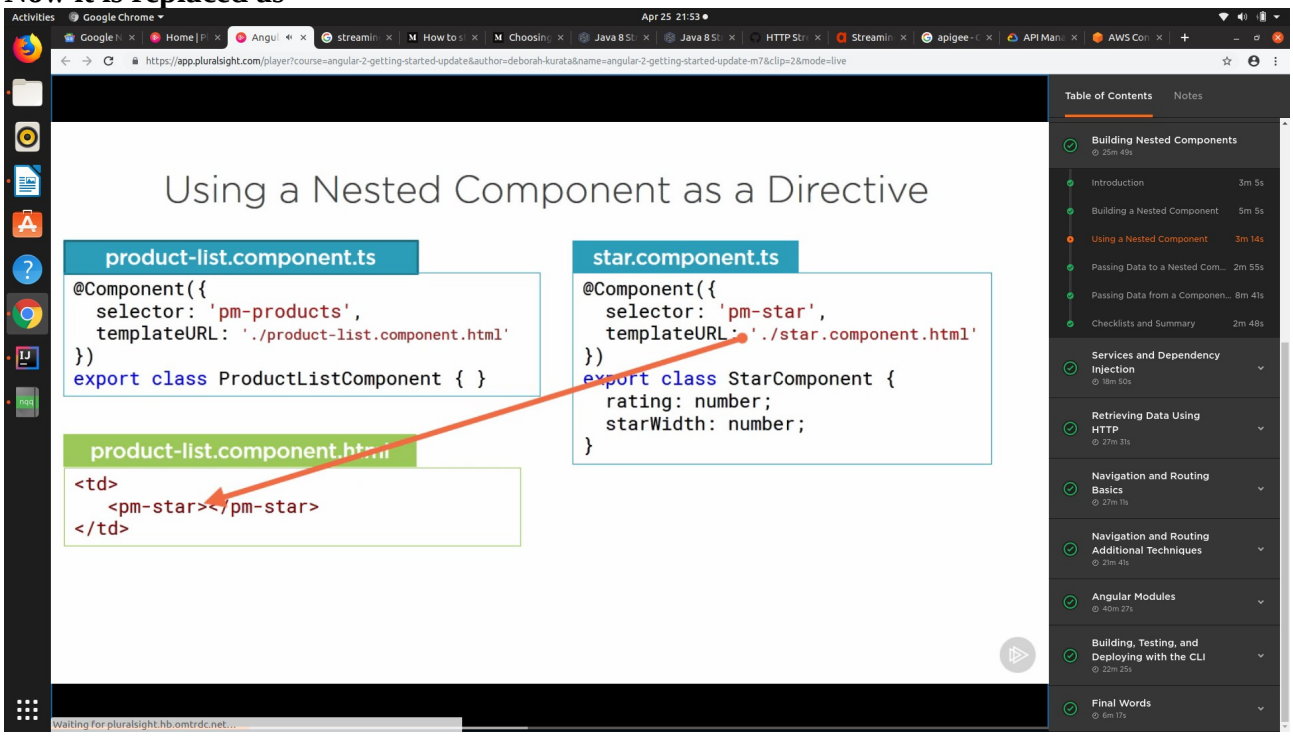


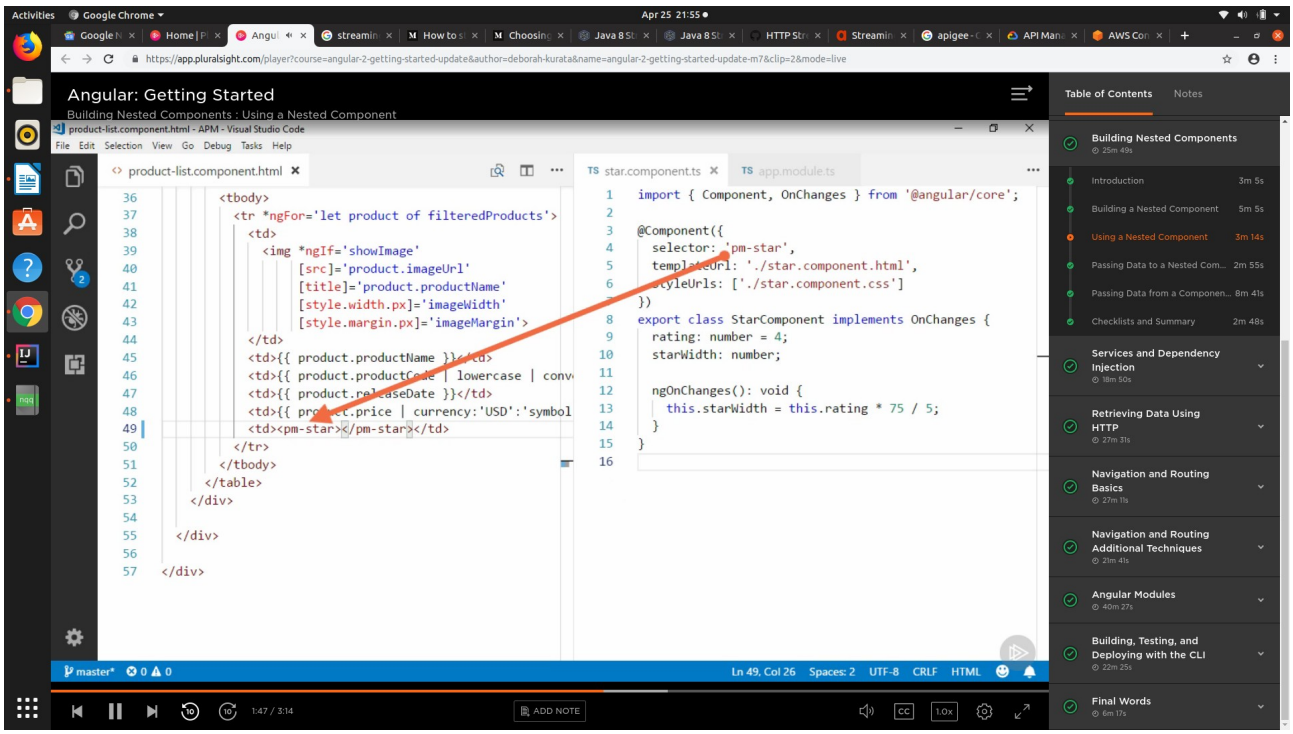


So initially it was a number as shown below

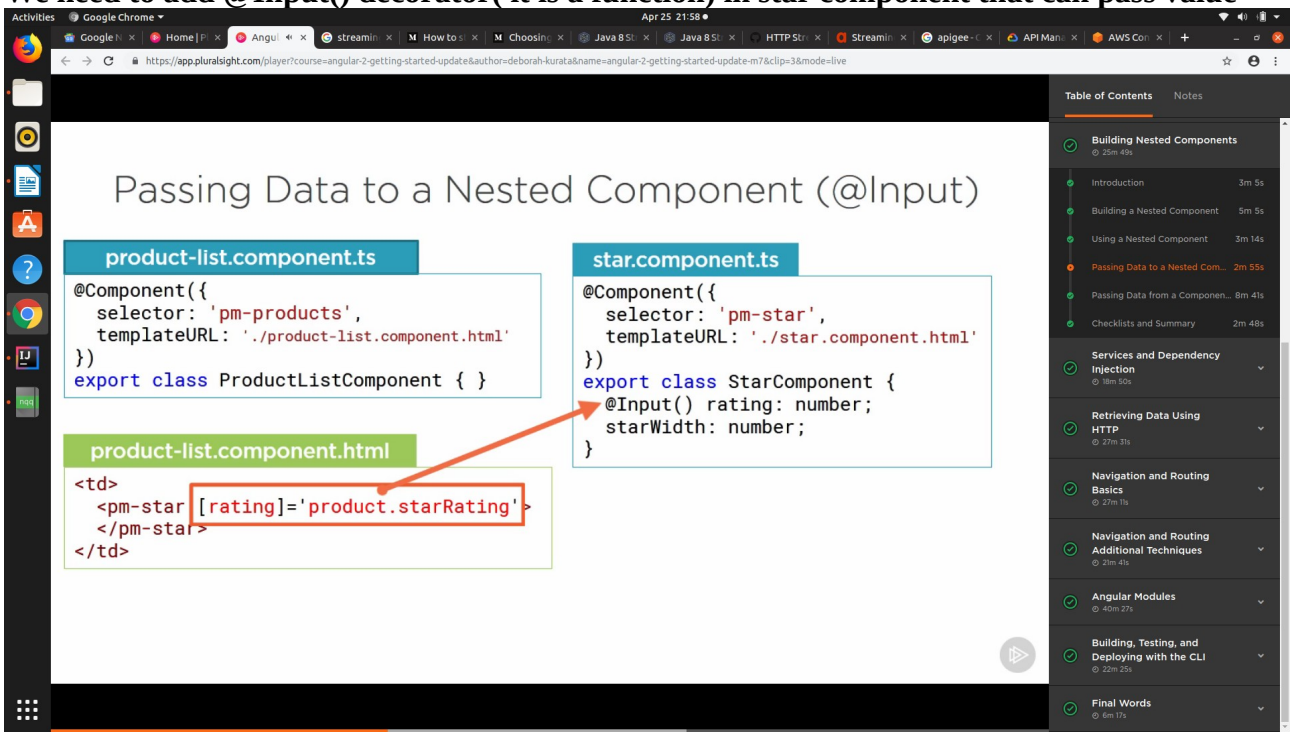


Now it is replaced as

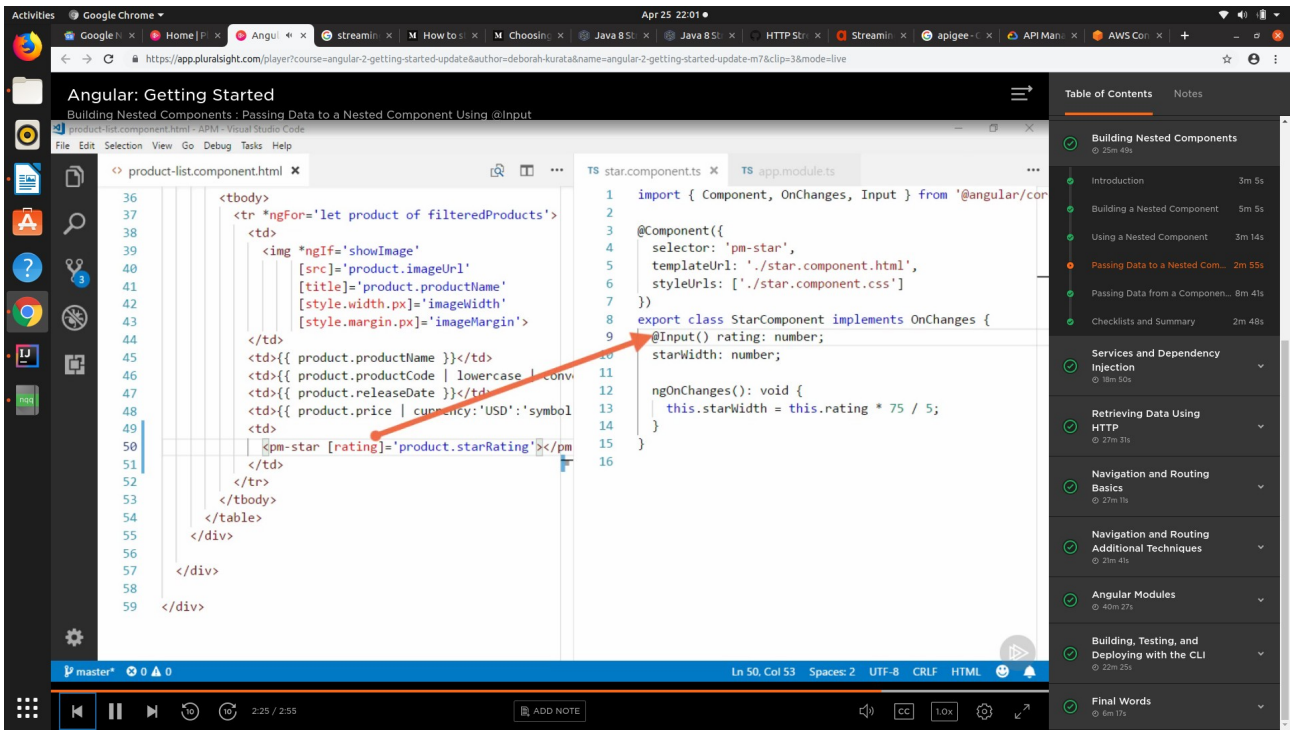




Currently it will show 5 stars as there is no interlinking between the Star component and product list component.
We need to add @Input() decorator(it is a function) in star component that can pass value

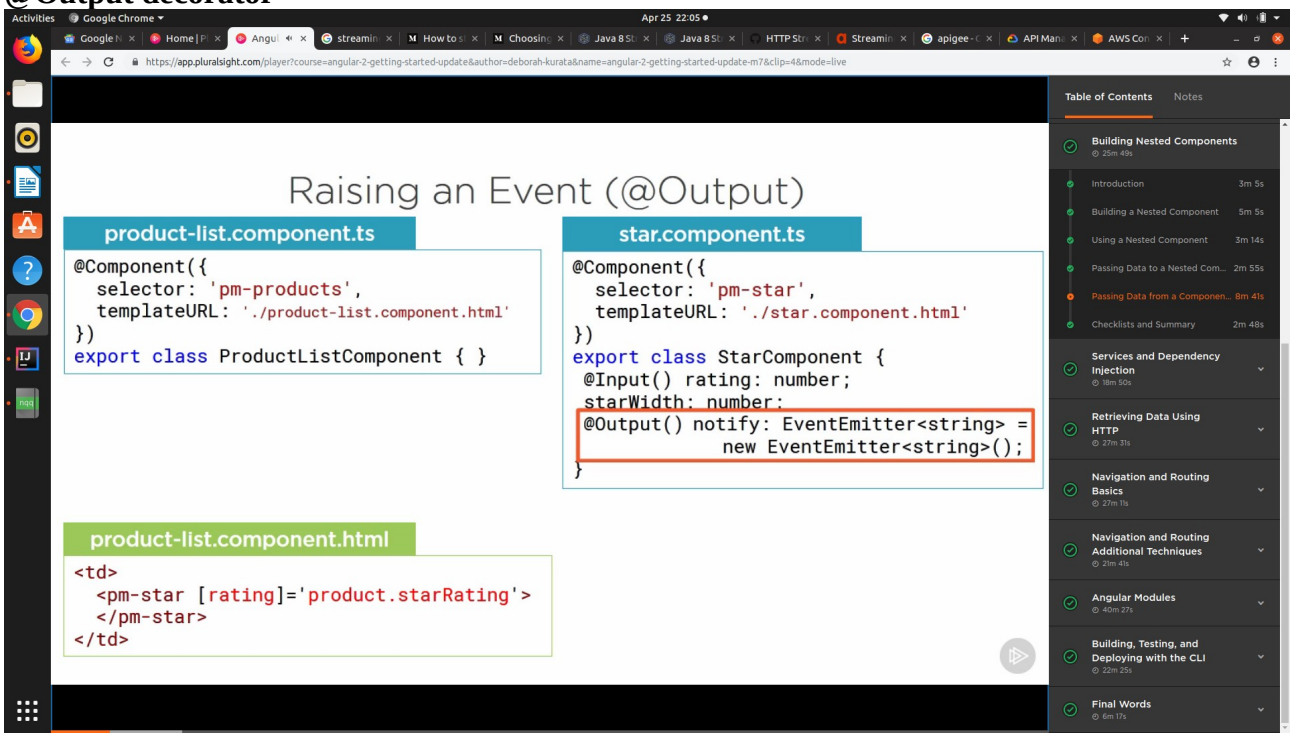


Now the product.starRating is mapped with the rating and it will reflect at any change as there is ngOnChanges life cycle hook attached to it

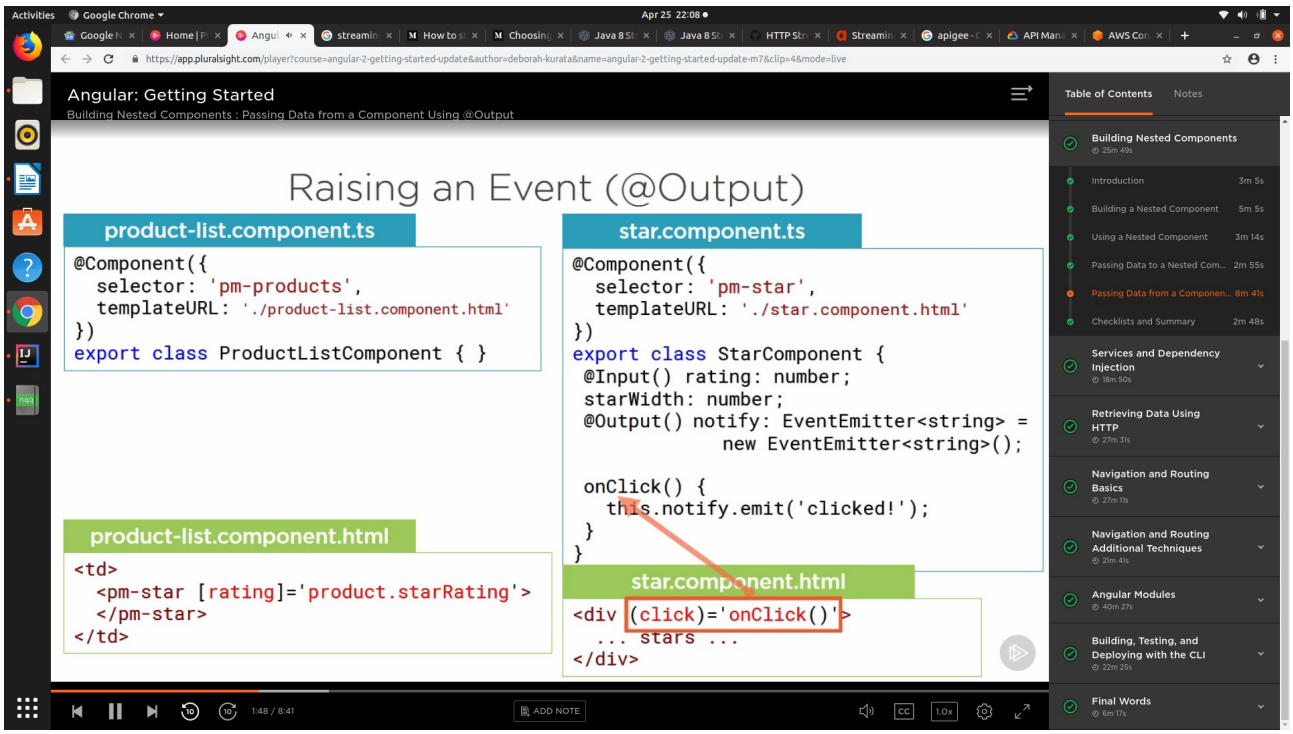


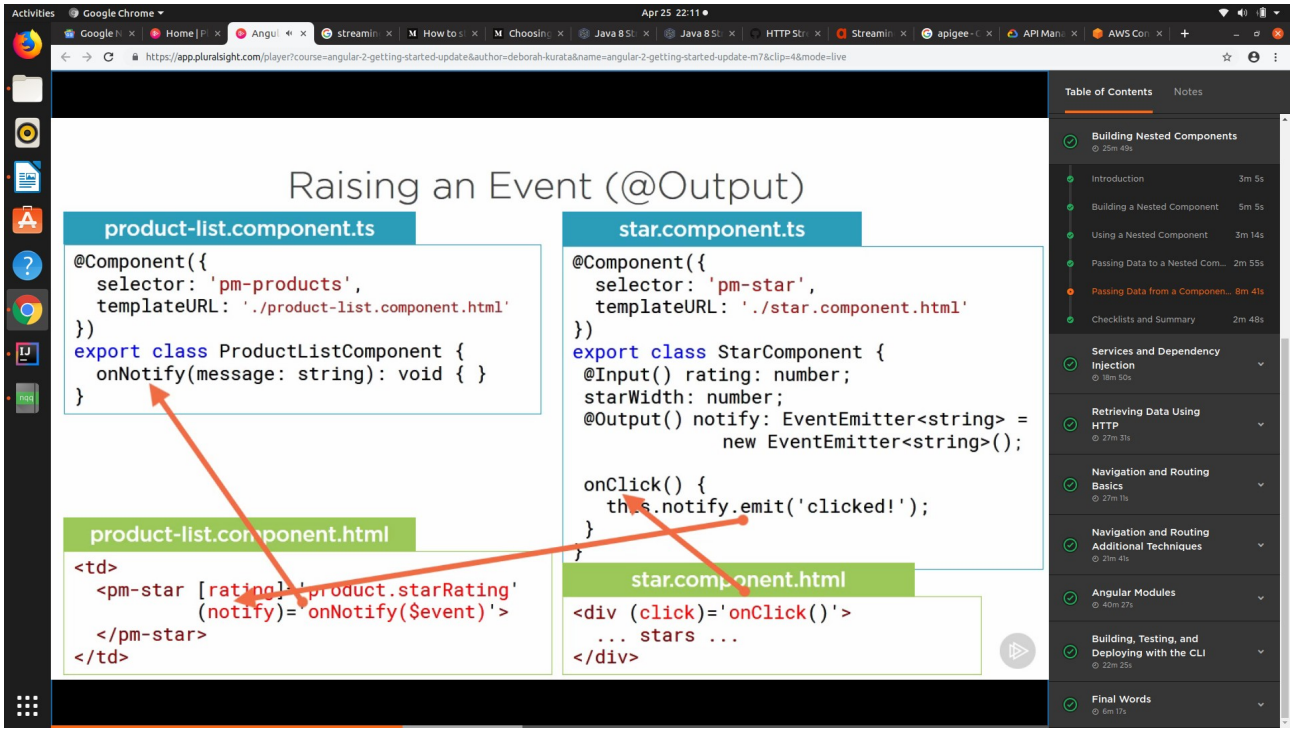
Passing Value from nested component to Container

Data can be passed to its Container only through an event using EventEmitter and with @Output decorator

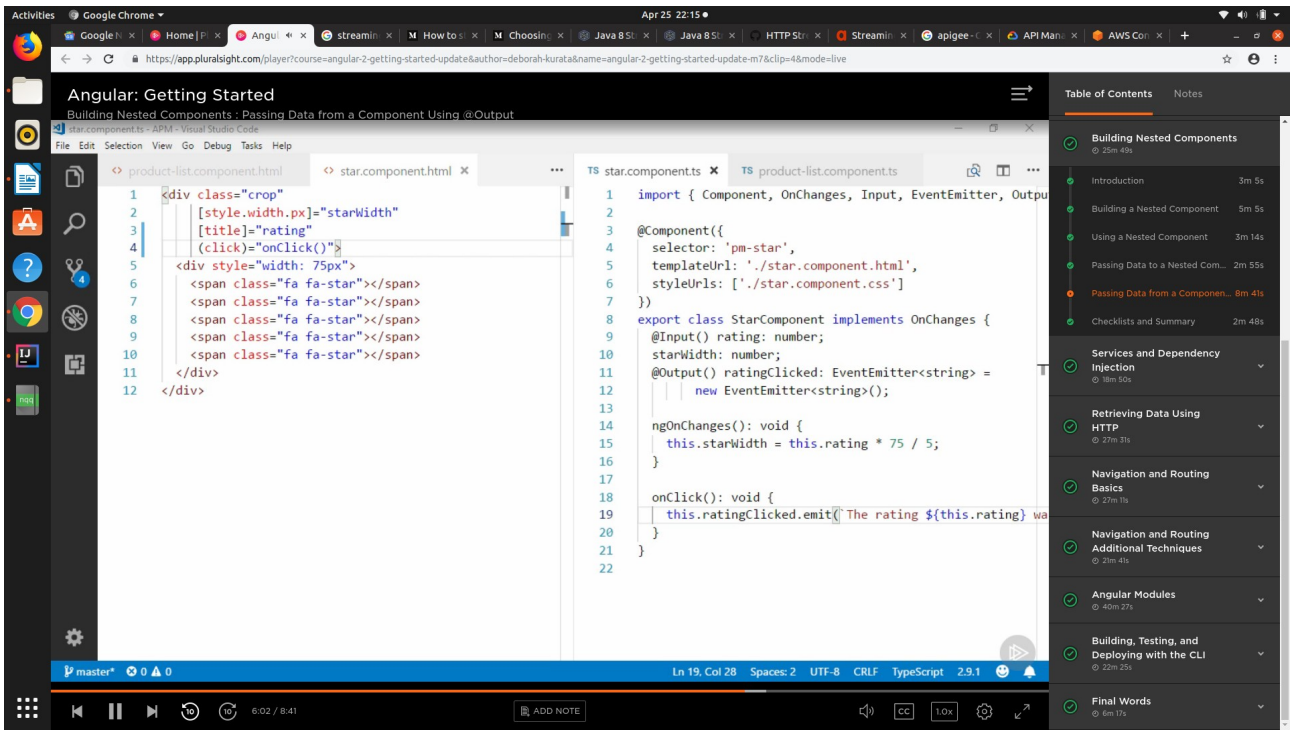


We will add an event to the star component and on click it will call the event onClick which in turn will call its notify event in @Outpur

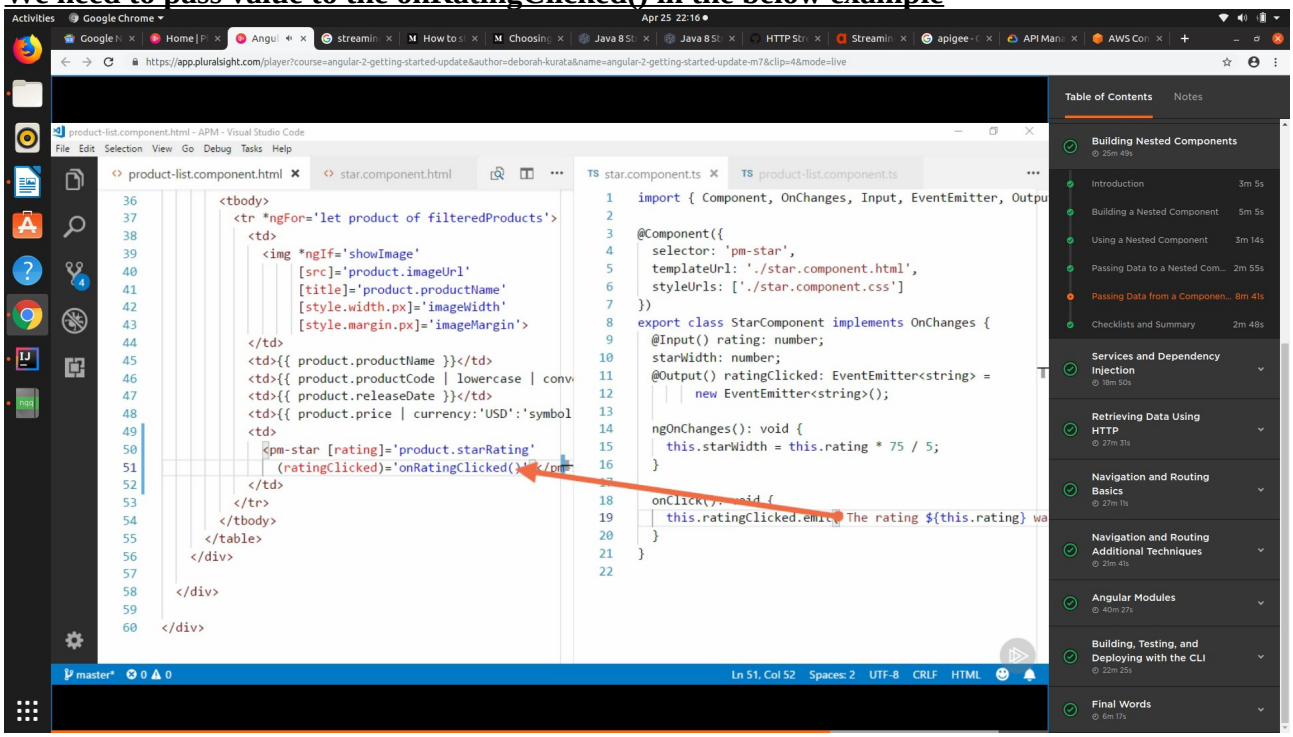




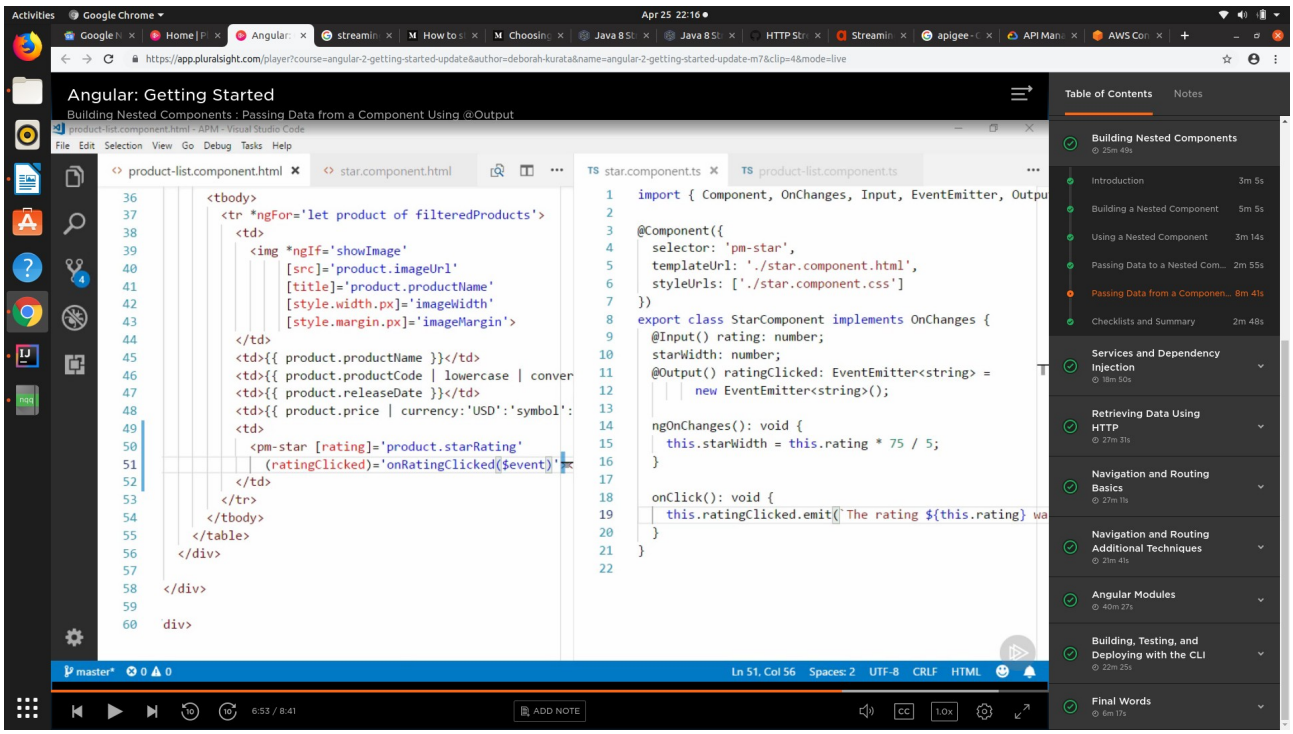
Implementaion



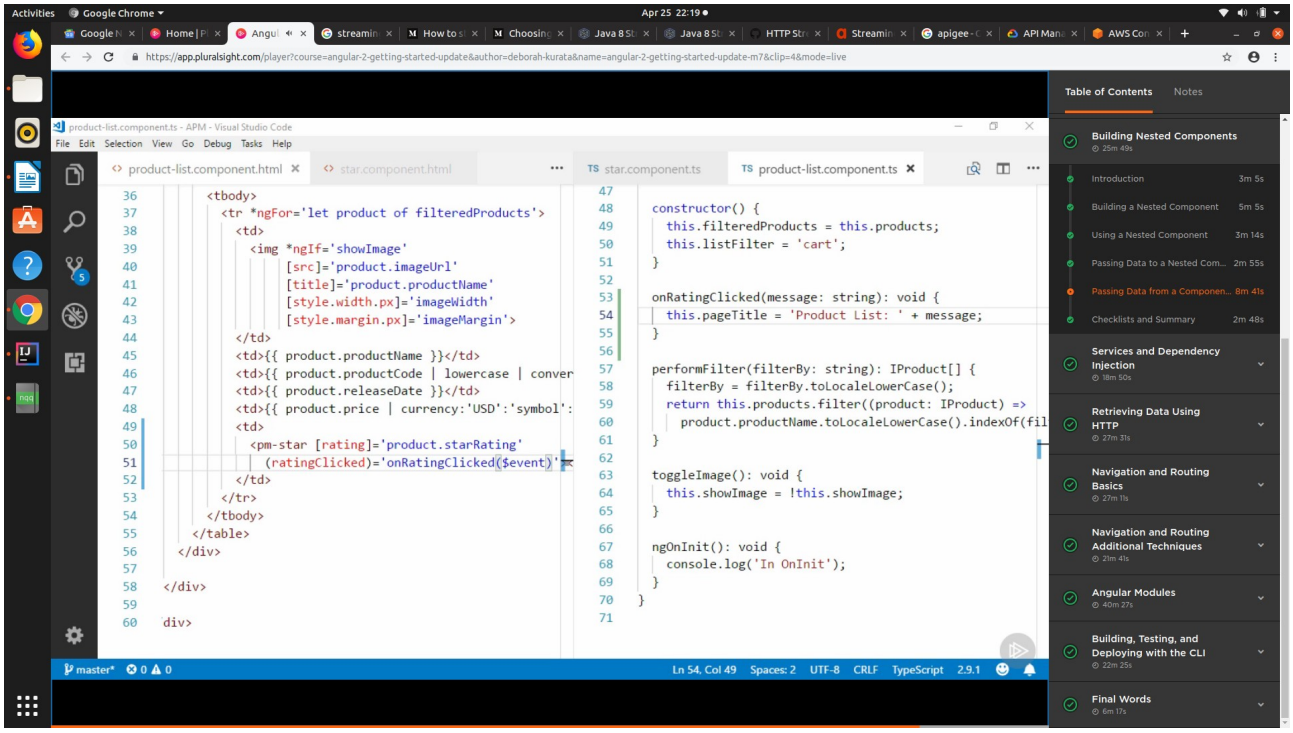
We need to pass value to the onRatingClicked() in the below example



It is passed with the $event

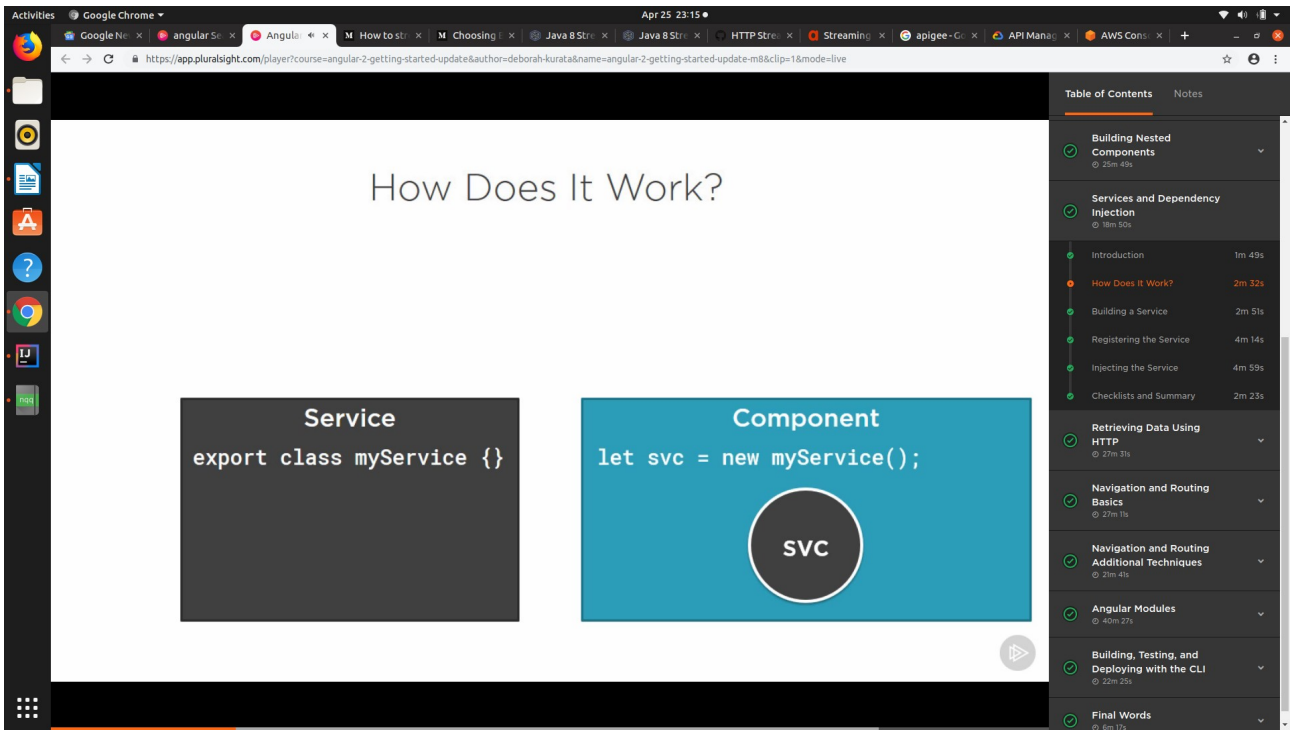


Now changes needs to be done in the productList Component

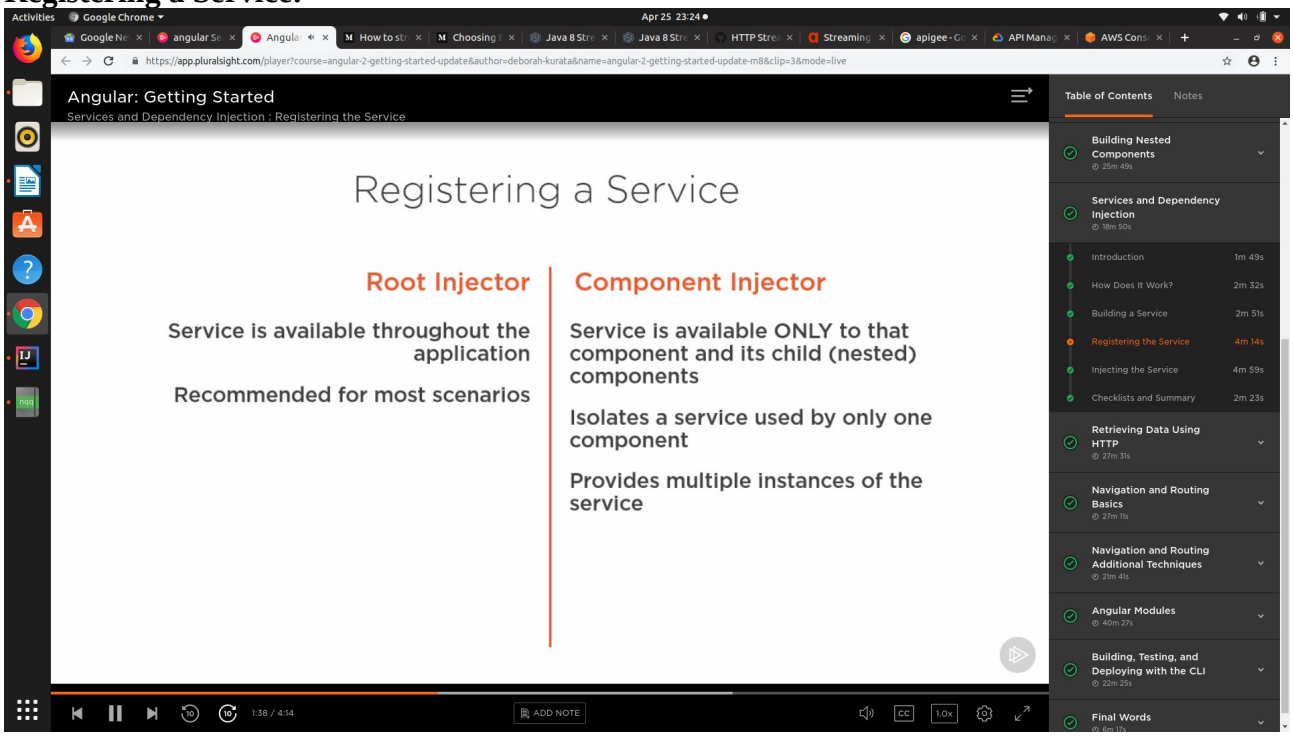


Service and Dependency Injection

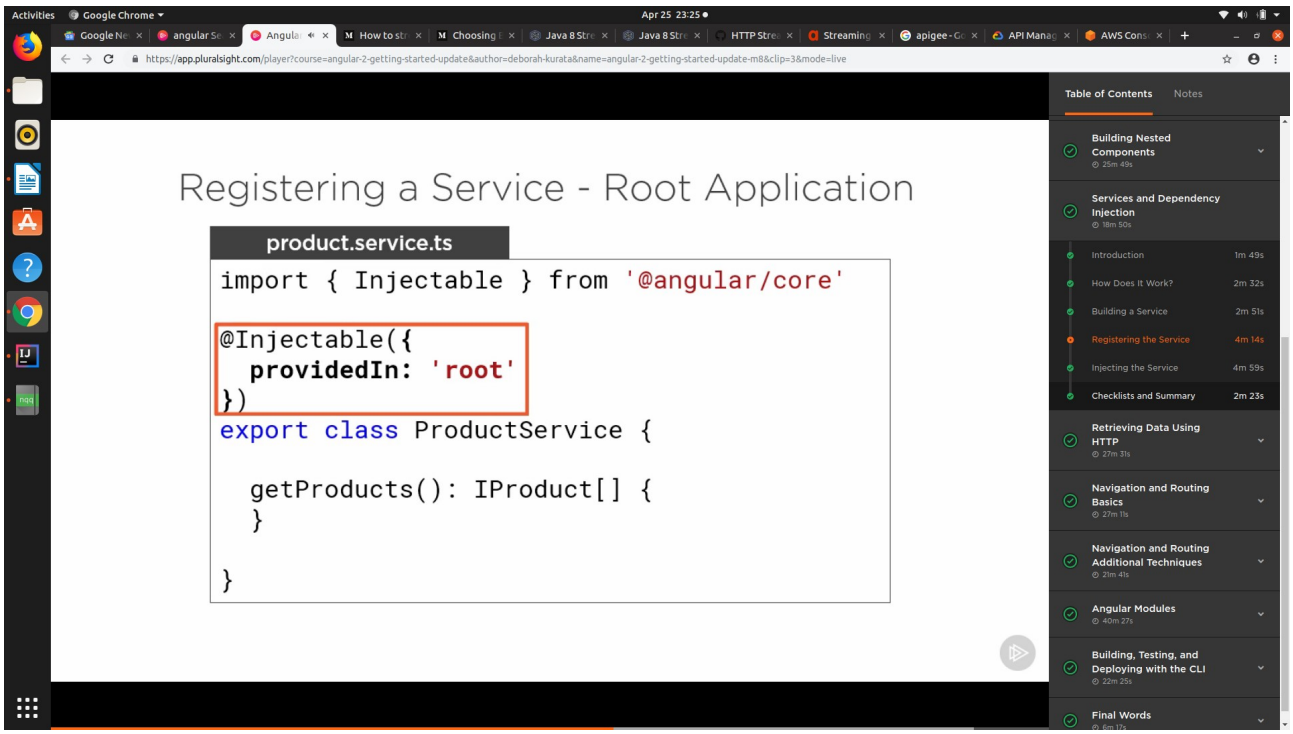
Service can be called as an object but in that case it cannot be shared accross



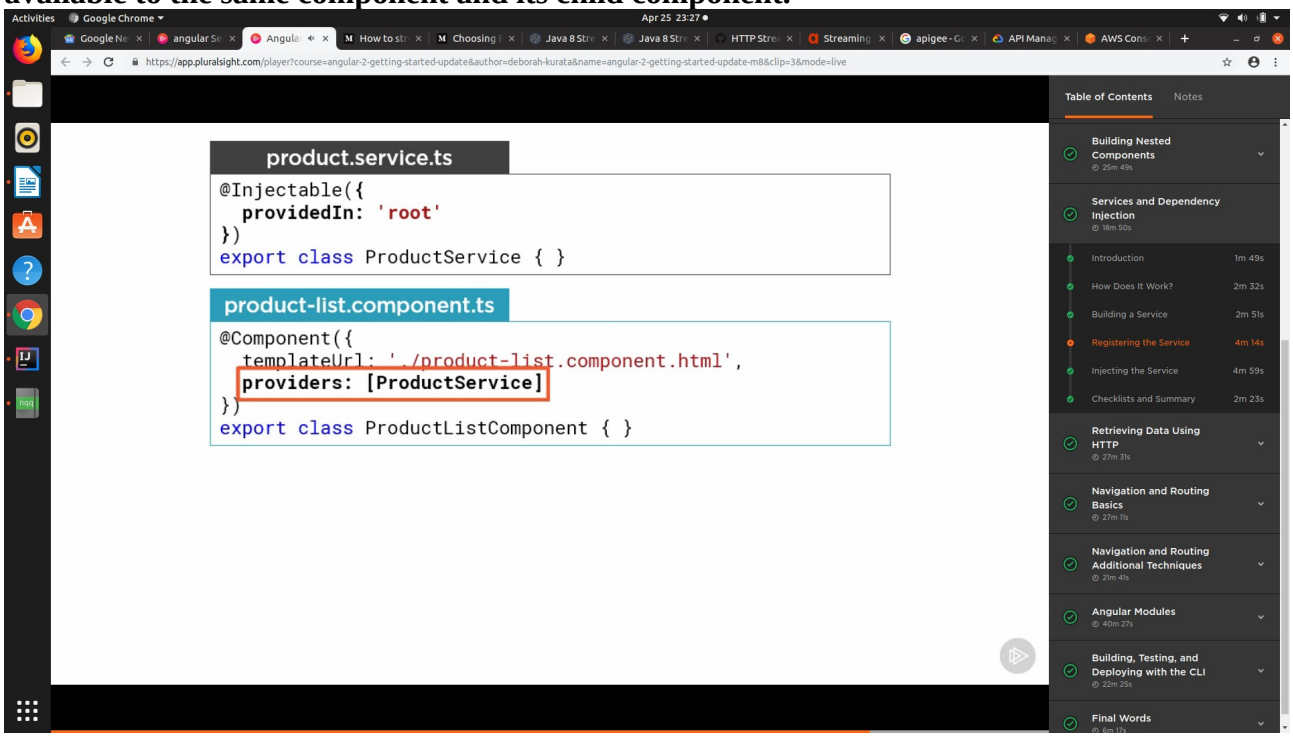
Registering a Service:



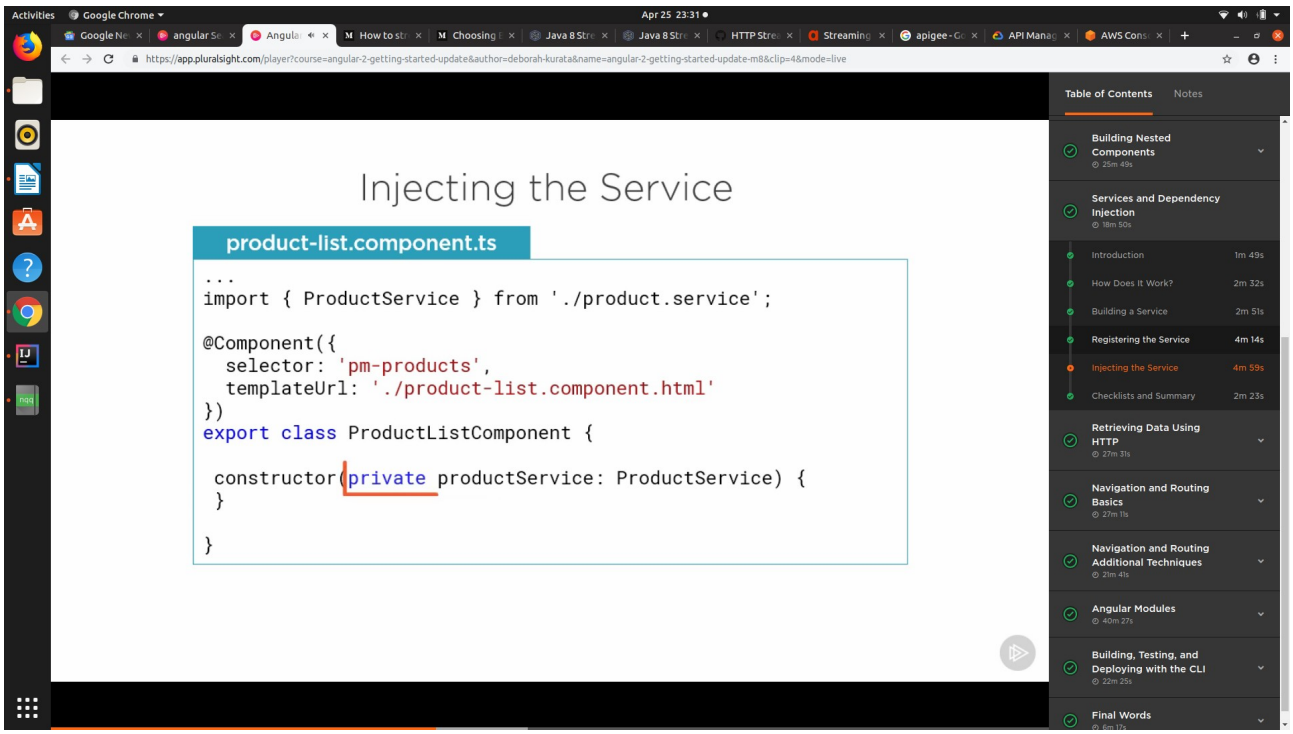
Root Injector: Here we have added providedIn: ‘root’ so it will be shared accross the application



Component Injector: We add it in specific component as providers: [ProductService] it will be available to the same component and its child component.



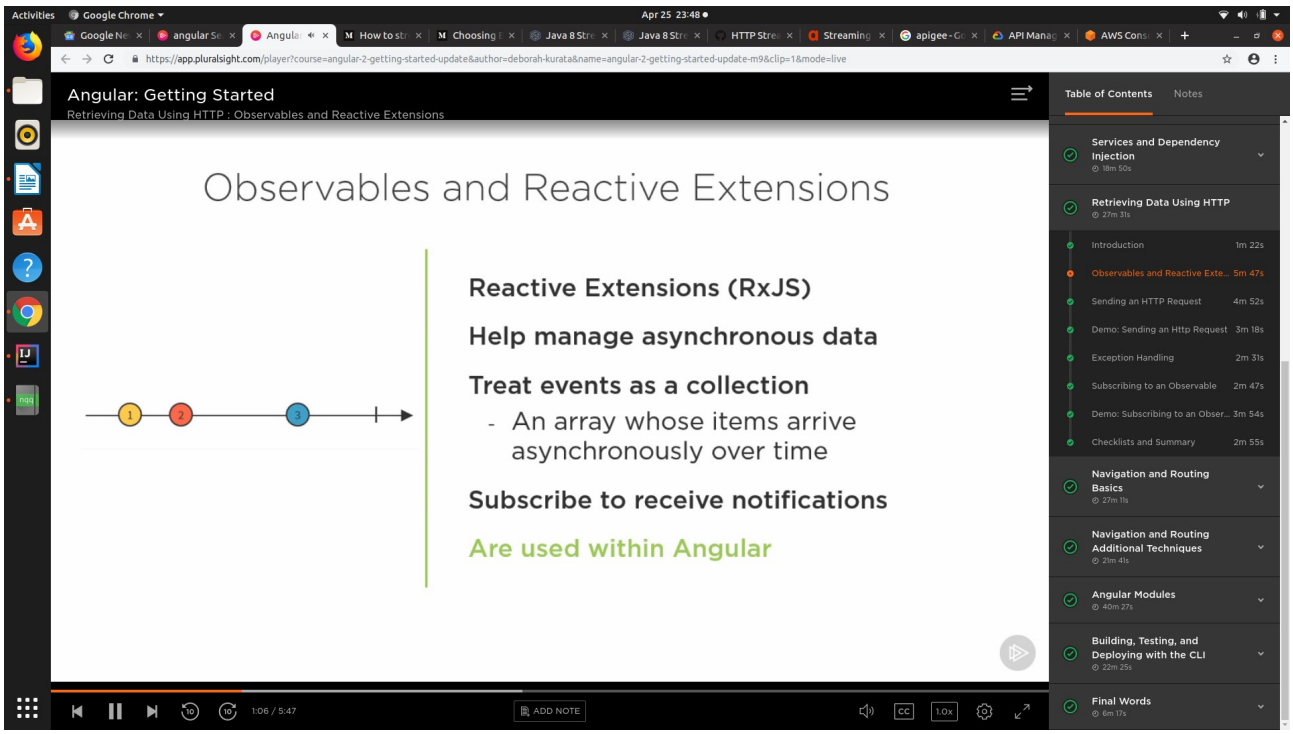
Injecting Service in a Component

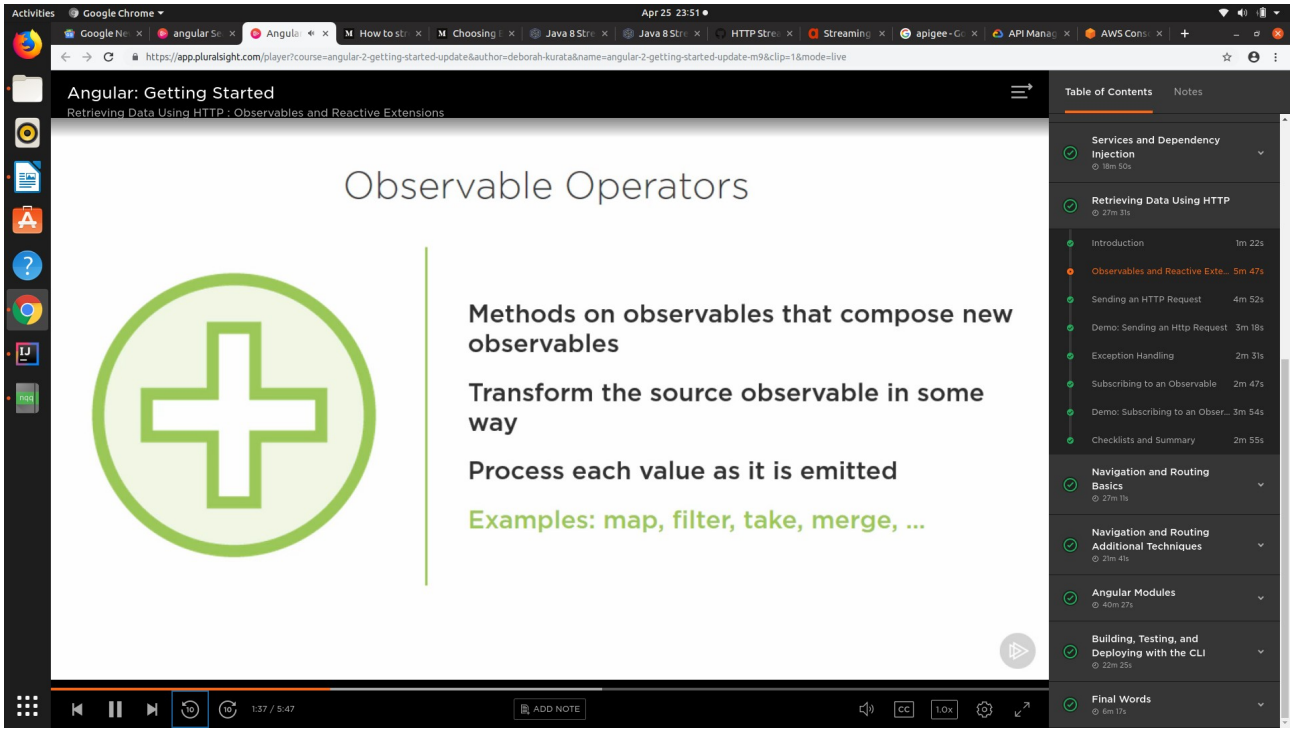


ngOnInit will be used to call the service.
Constructor is always called before the lifecycle hook method ngOnInit

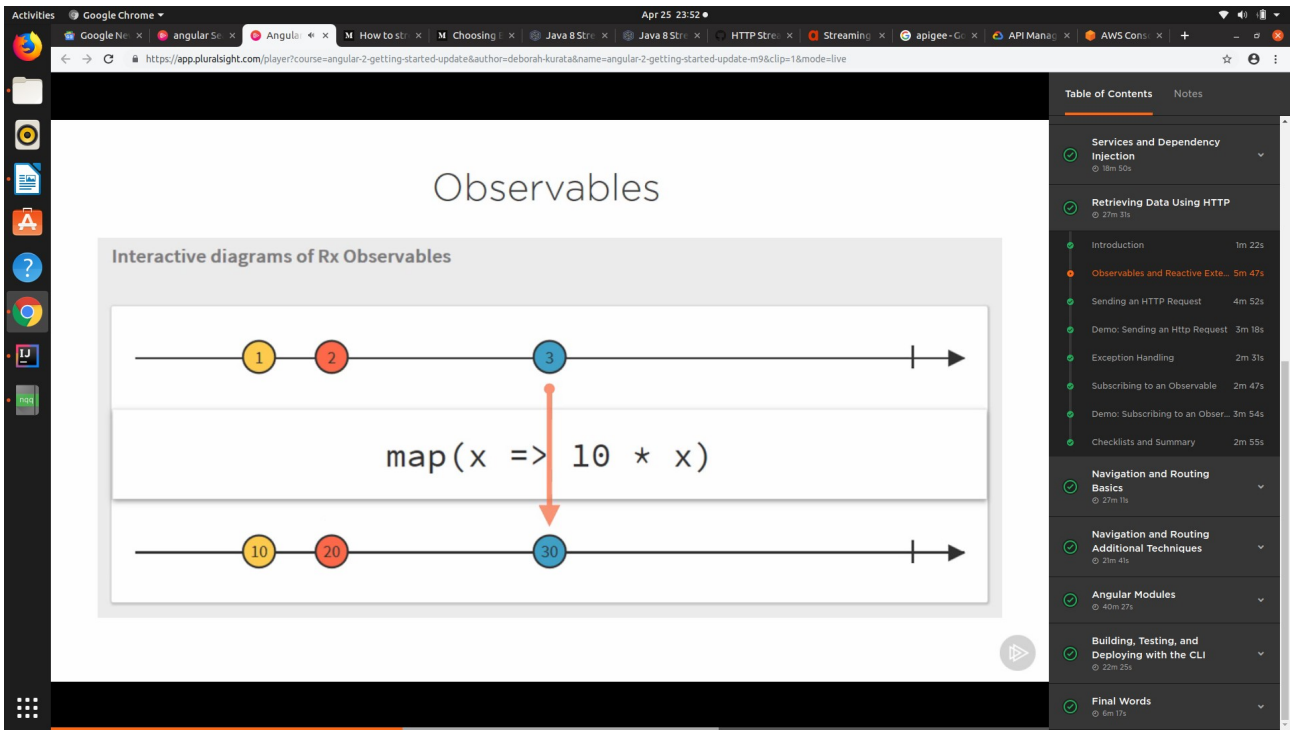
Retriveing Data using HTTP

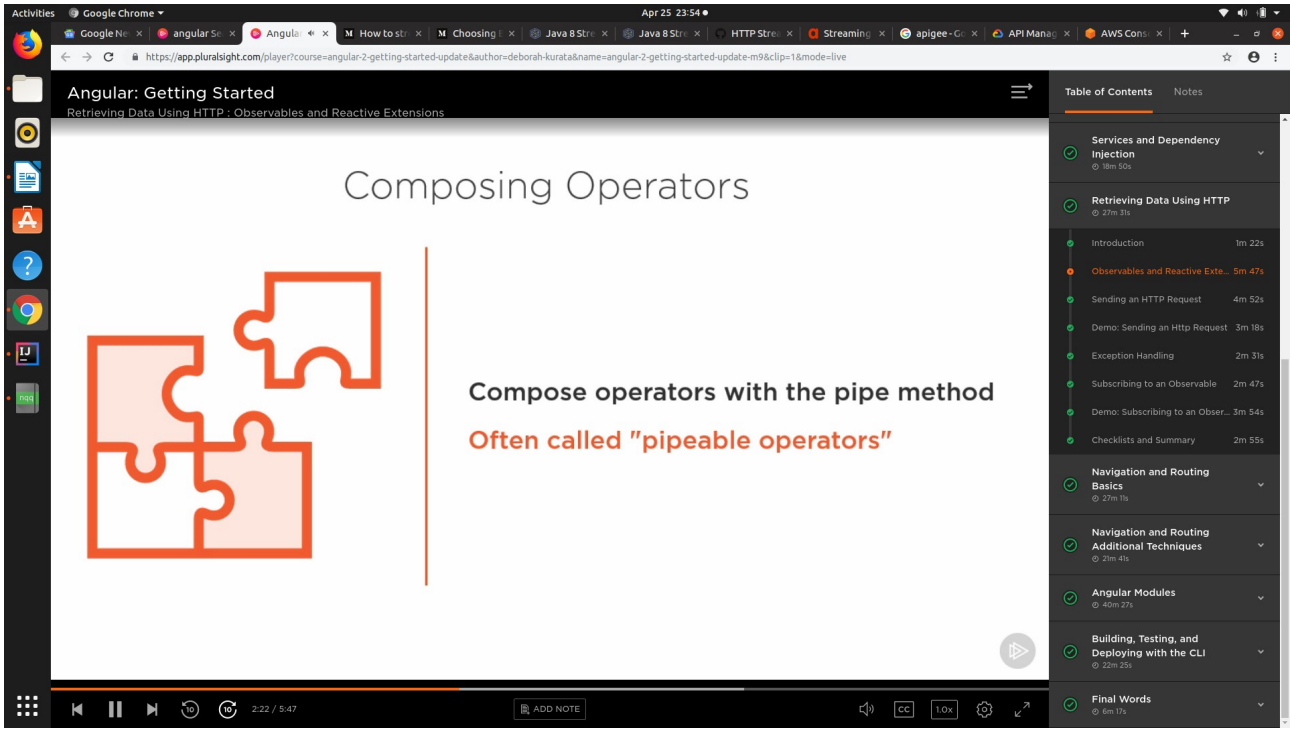
Observable is used to retrive data in async way



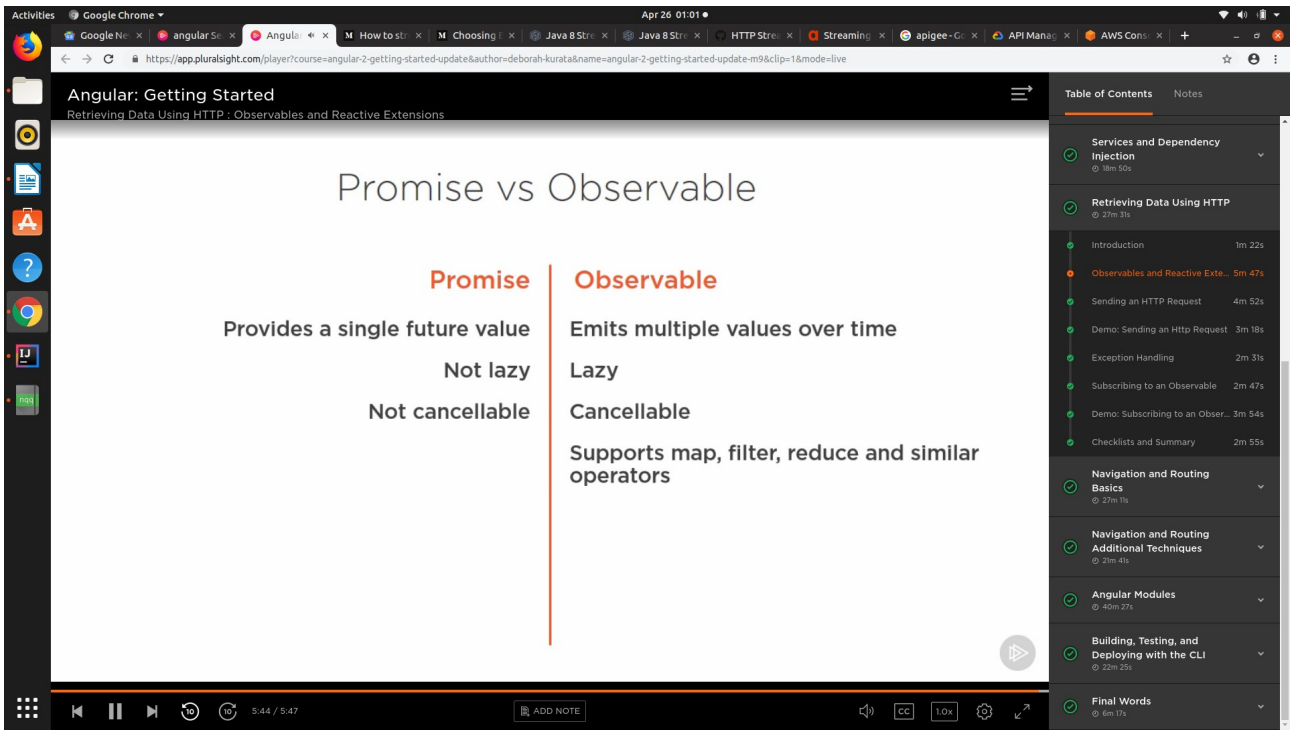


Map operator allows to transform the incoming data as shown below, 1-> is transformed to 10 and so on. The argument to the map function is an => function

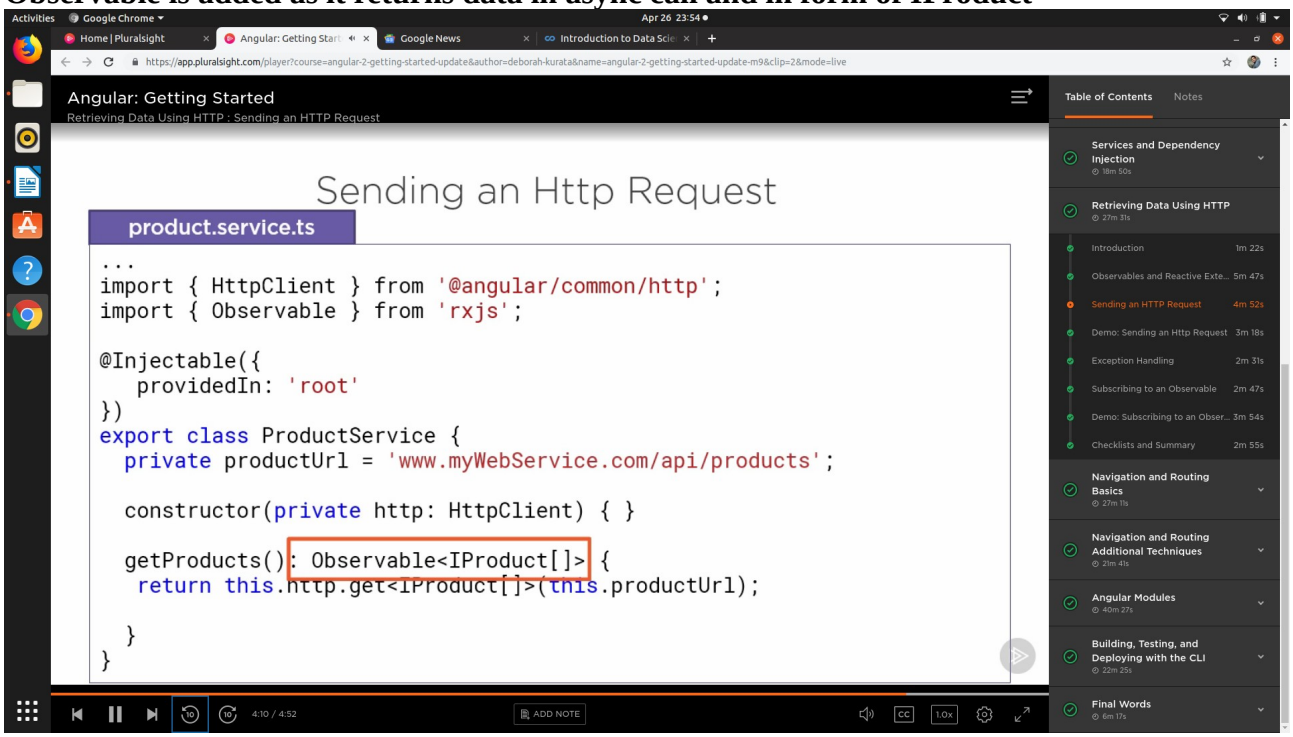




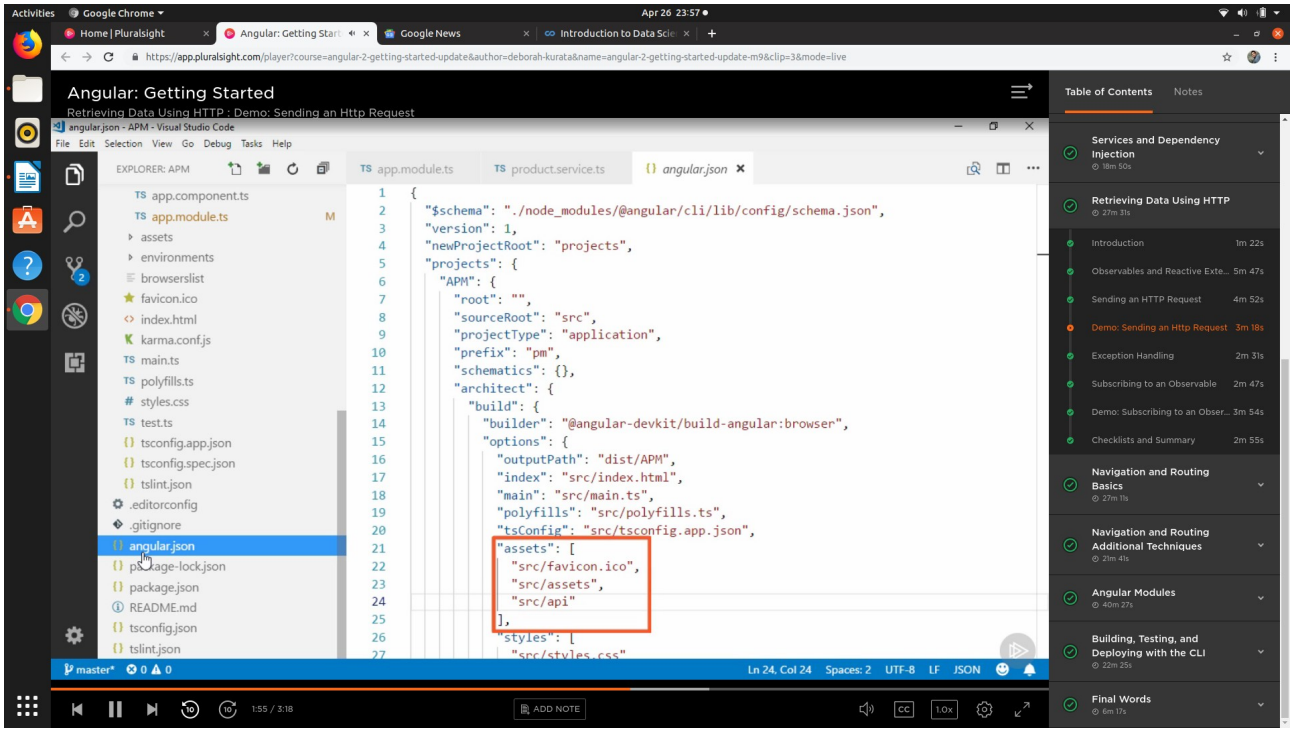
Difference between Promise and Observable



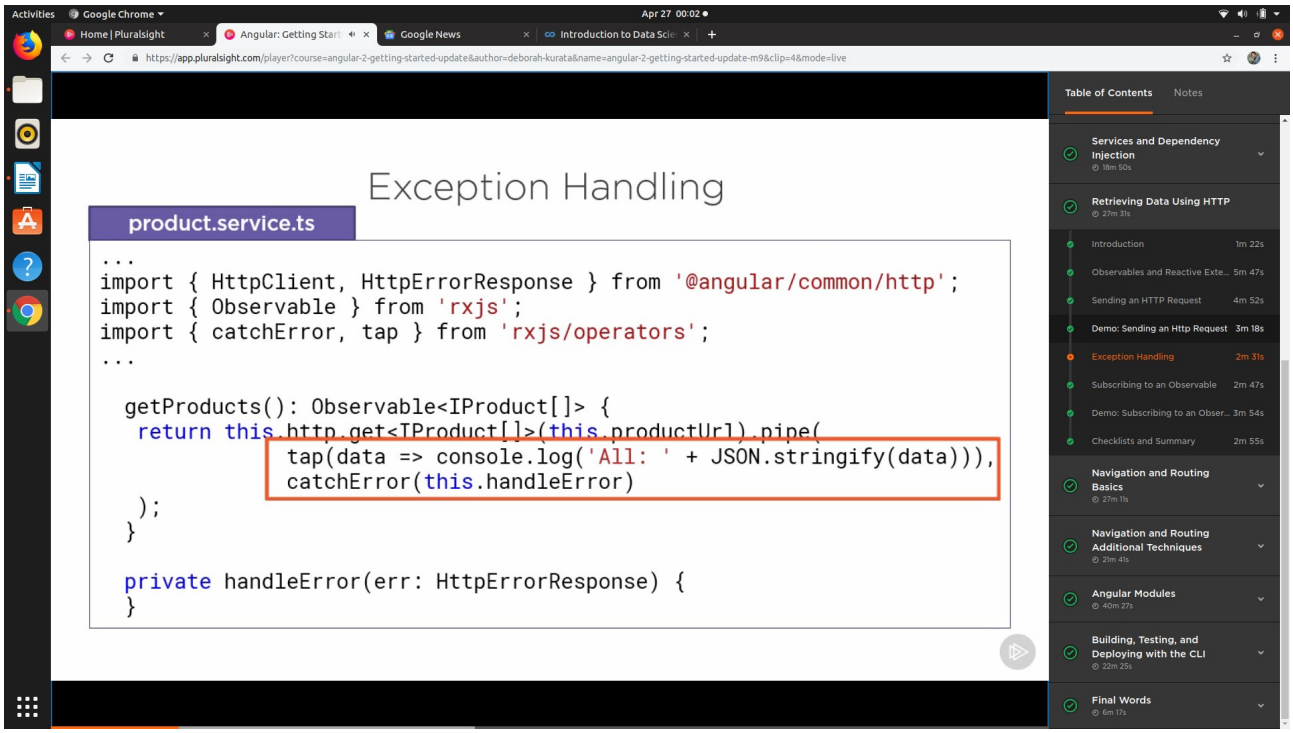
Example to create Service: We need to import HttpClientModule in AppModule
Observable is added as it returns data in async call and in form of IProduct



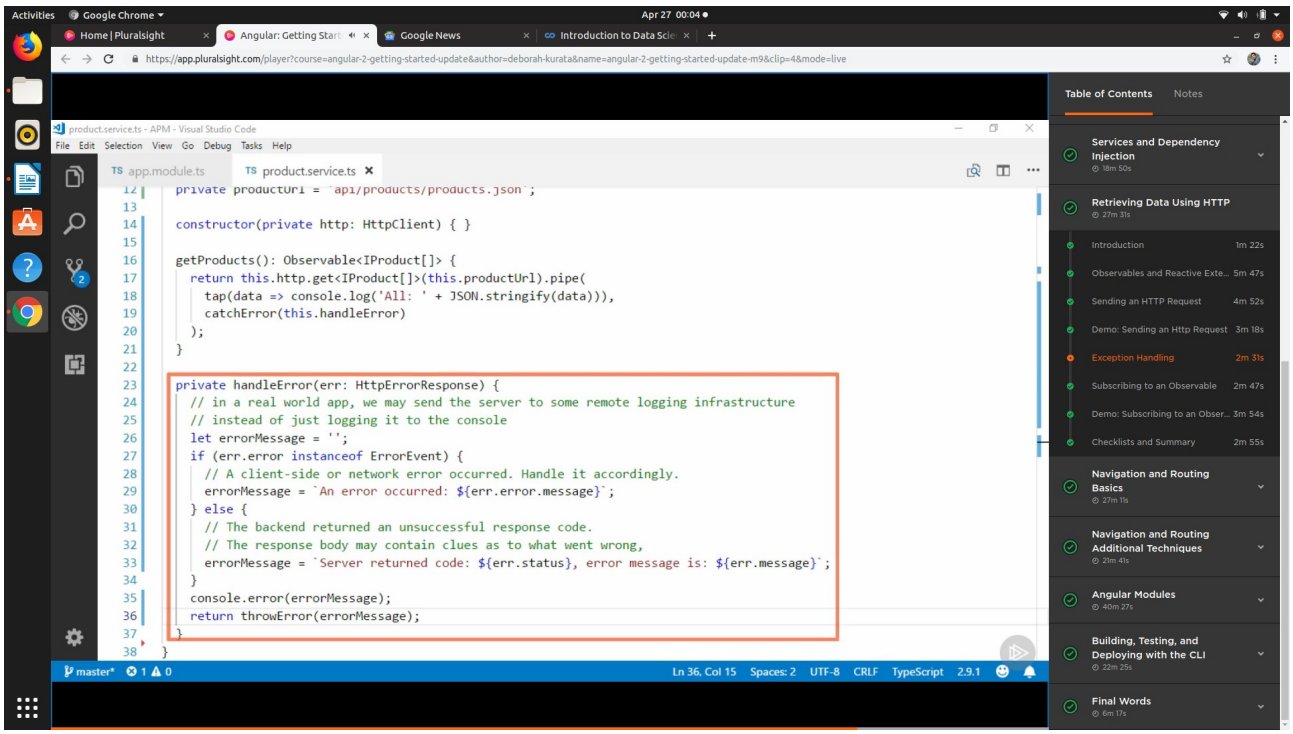
To make the json file available they we have mapped it in angular.json file in the below picture



Exception Handling in Service::



Example: working



Using observable we retrive data but we cannot use it until we subscribe it. Observable are lazy and they dont emitt values until subscribe is called.

Subscriber method takes upto three arguments each providing a handler function.

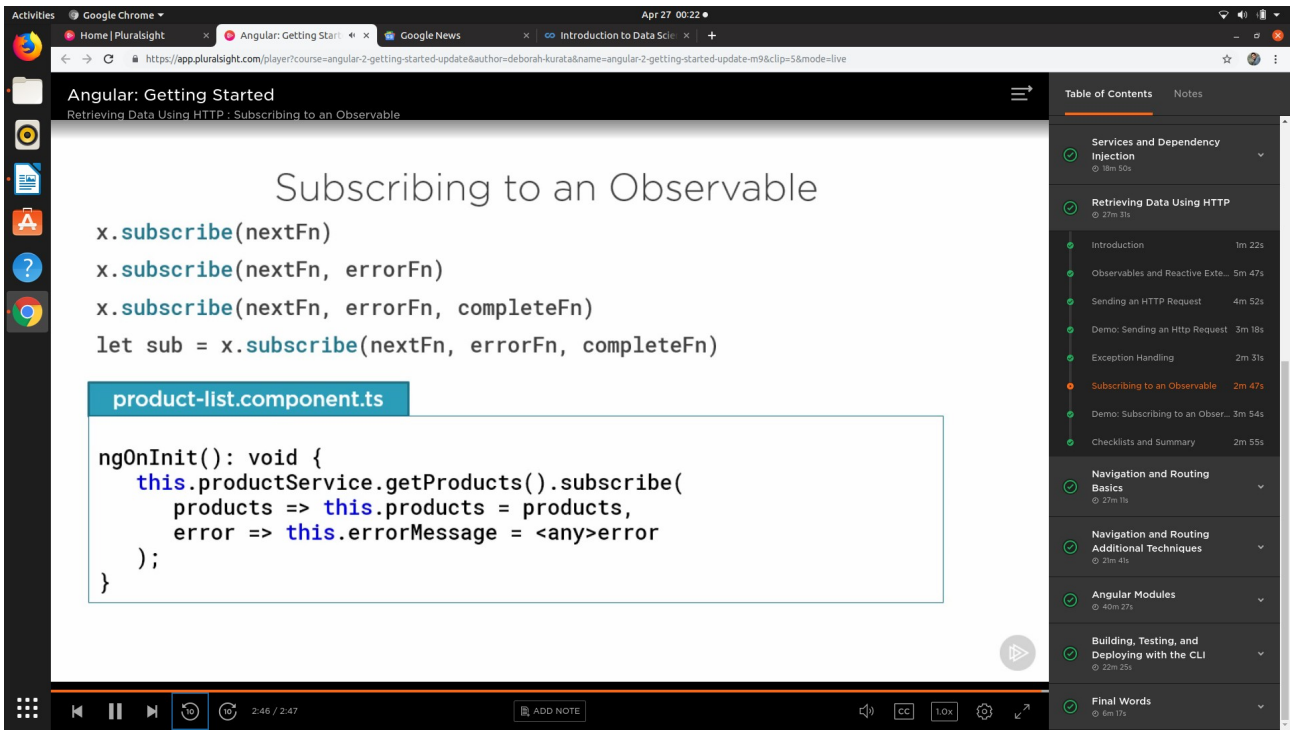
1) x.subscribe(nextFn): First argument is often called next function because it processes the next emitted value

2)x.subscribe(nextFn, errorFn): Next argumment is error handling function

3)x.subscribe(nextFn, errorFn, completeFn) : Third handler which is executed on completion

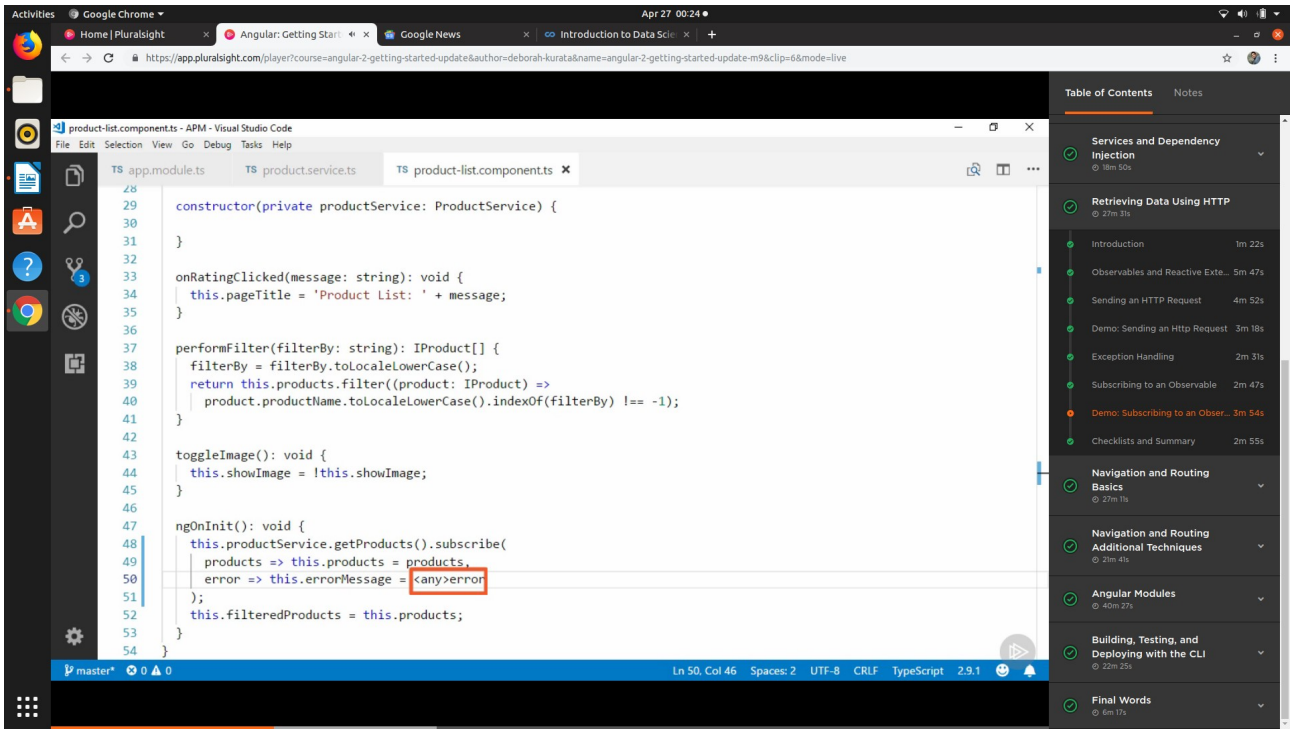
The subscribe method returns a subscription

let sub= x.subscribe(nextFn, errorFn, completeFn)

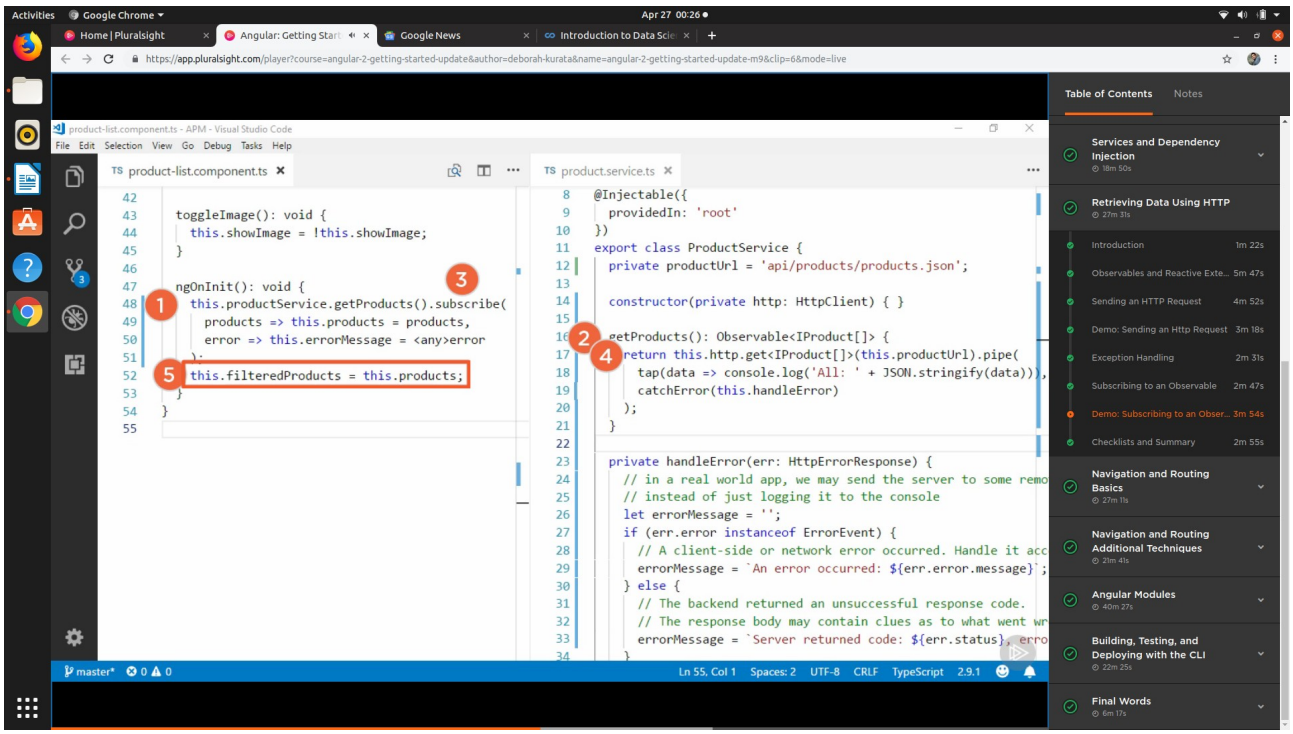


•

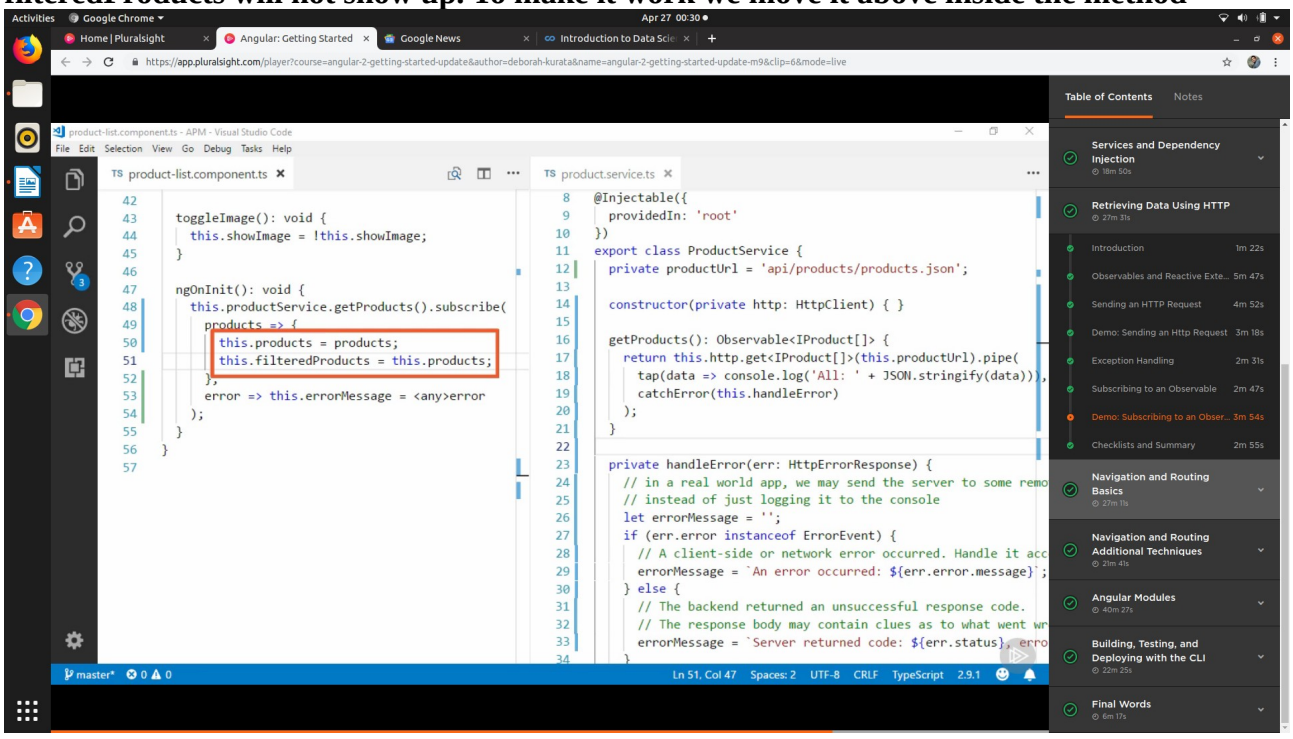
Example: we create a errorMessage:string property and use it in the method inside ngOnInit



Order of execution



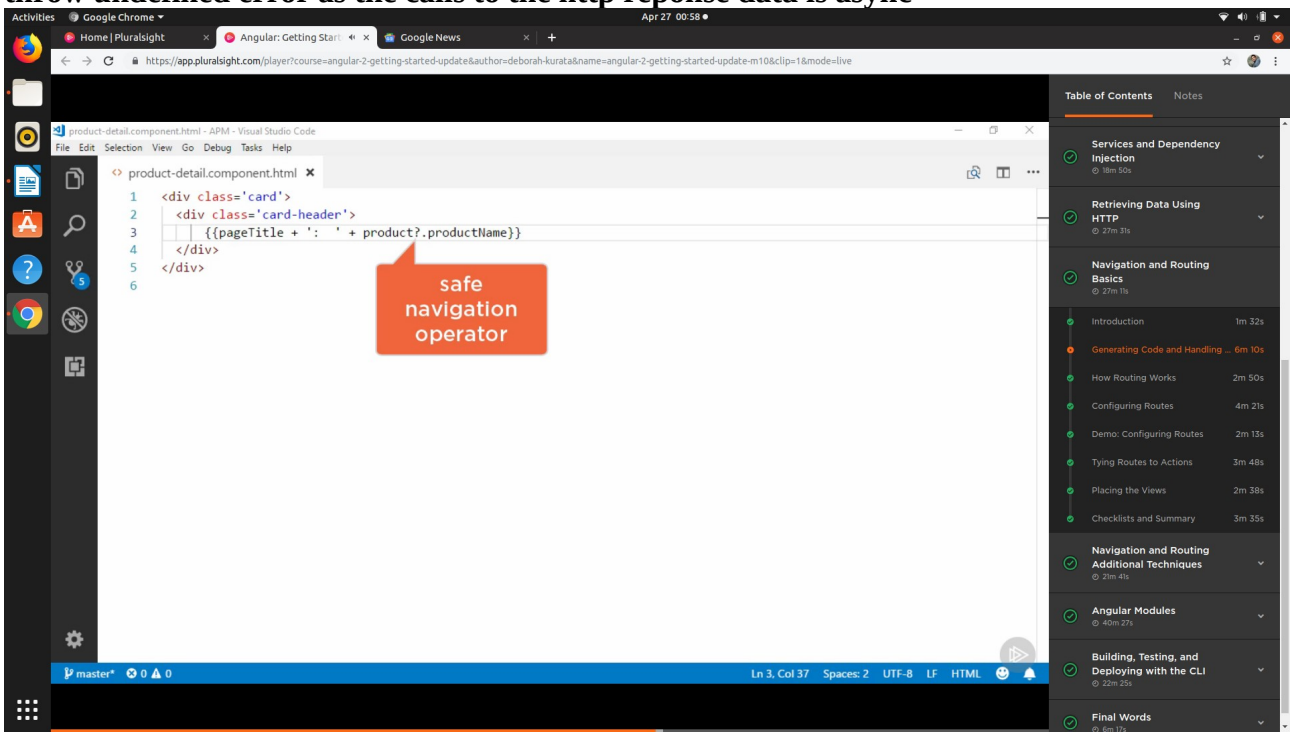
Since Observable does a sync call, in that case if data does not come until it is emitted so filteredProducts will not show up. To make it work we move it above inside the method



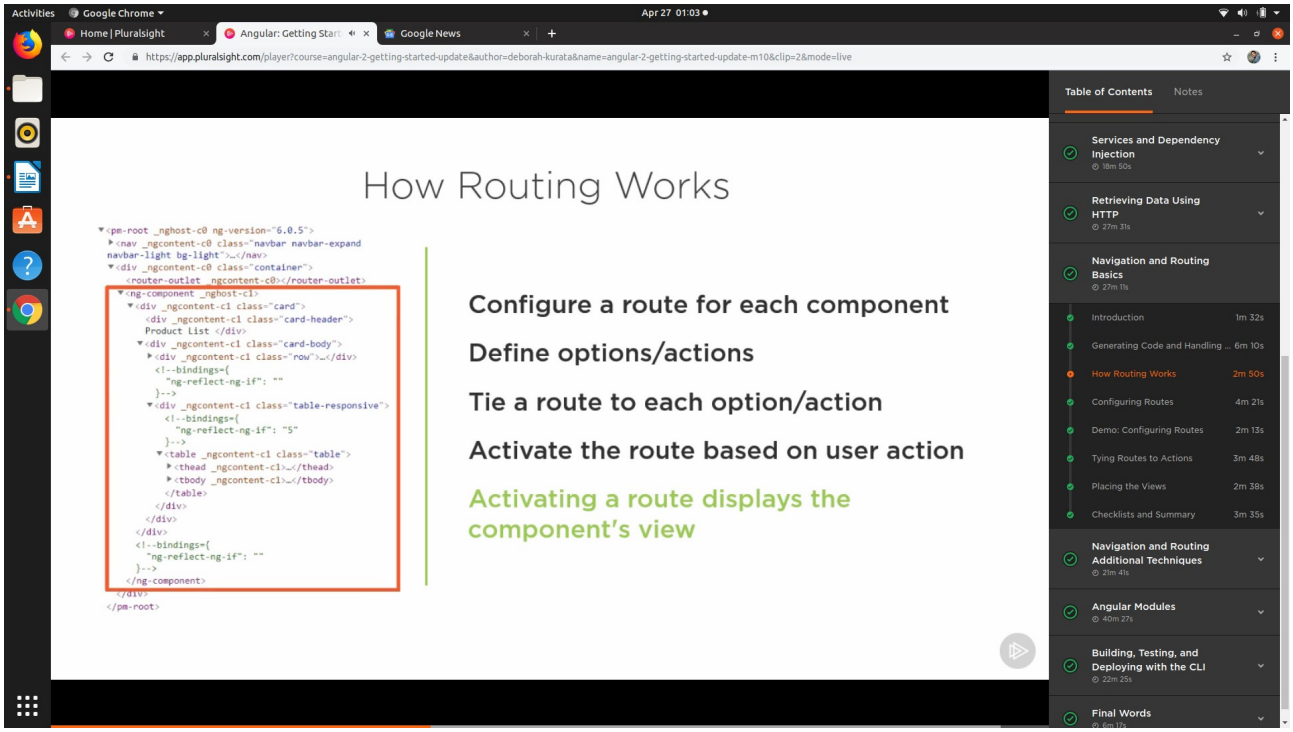
Navigation and Routing Basics

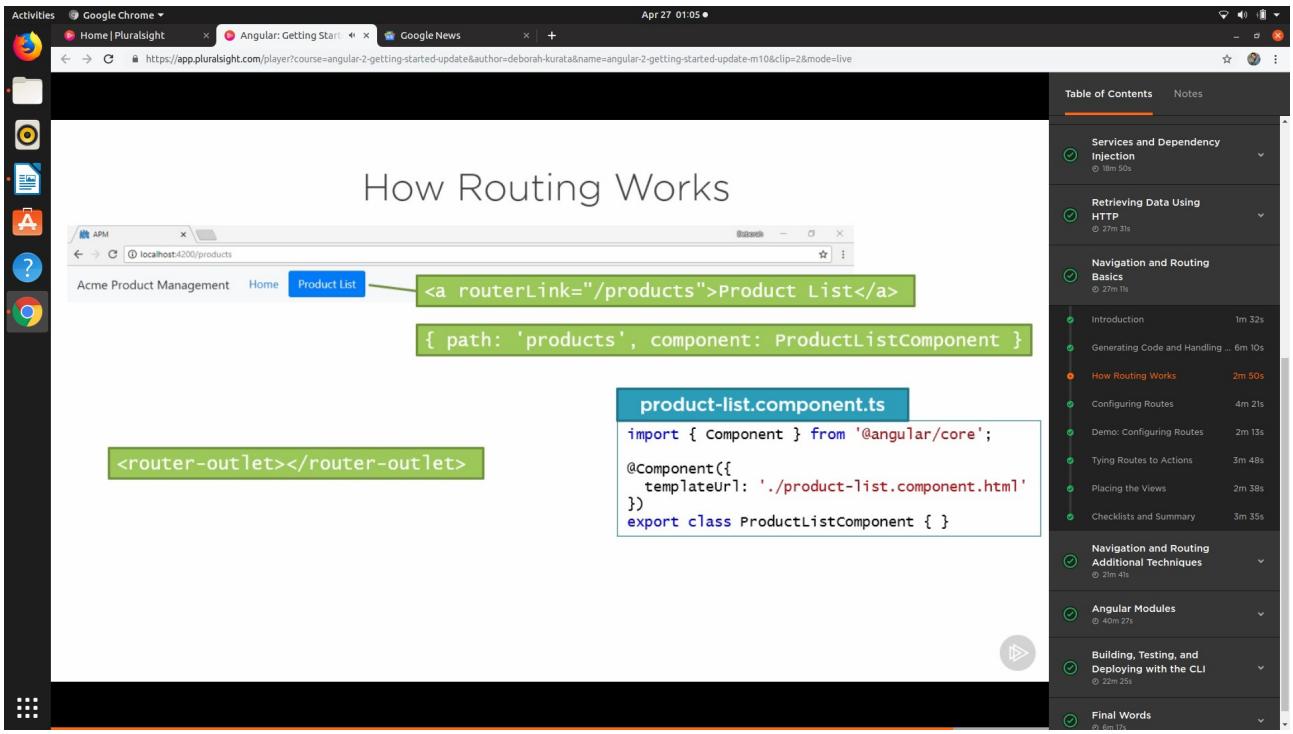
Command to create a component: ng g c products/product-detail –flat
g-> It means generate
c->component
path
--flat-> No folder to be created

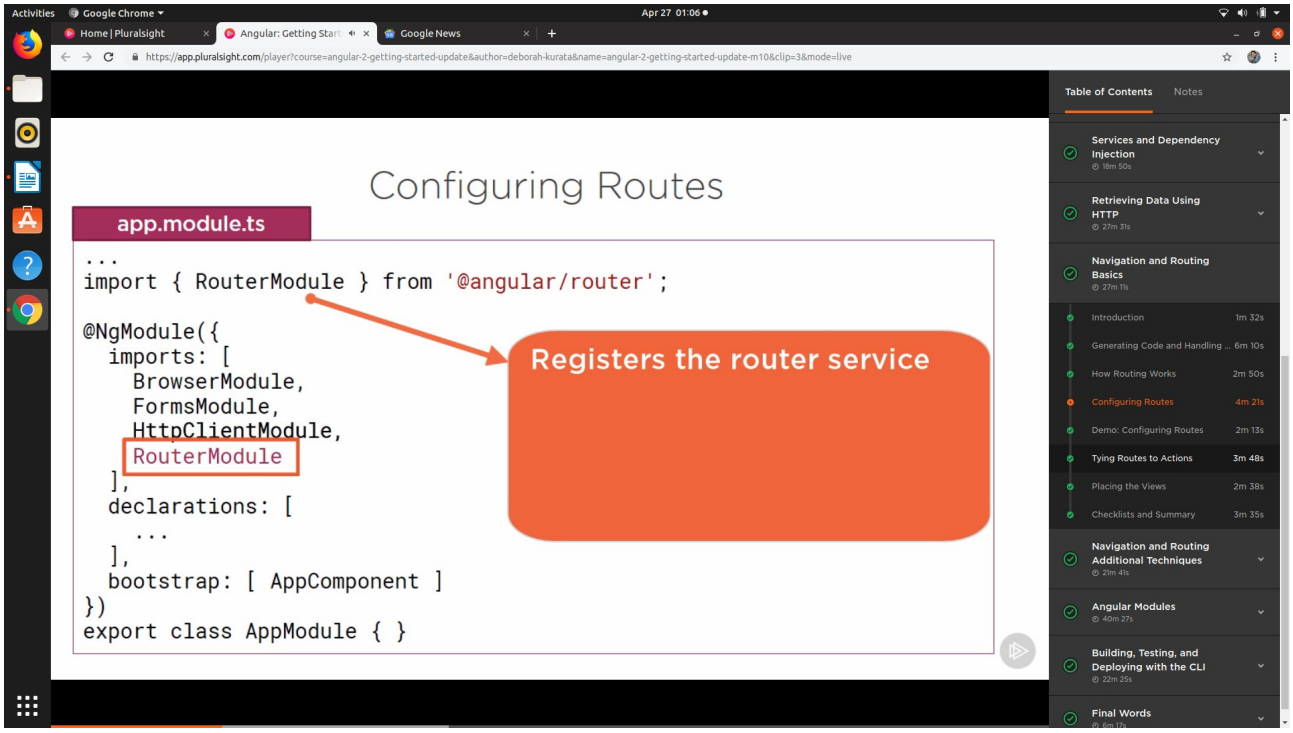
Safe navigation opertor(?)hecks if the product object has data otherwise the property will throw undefined error as the calls to the http reponse data is async

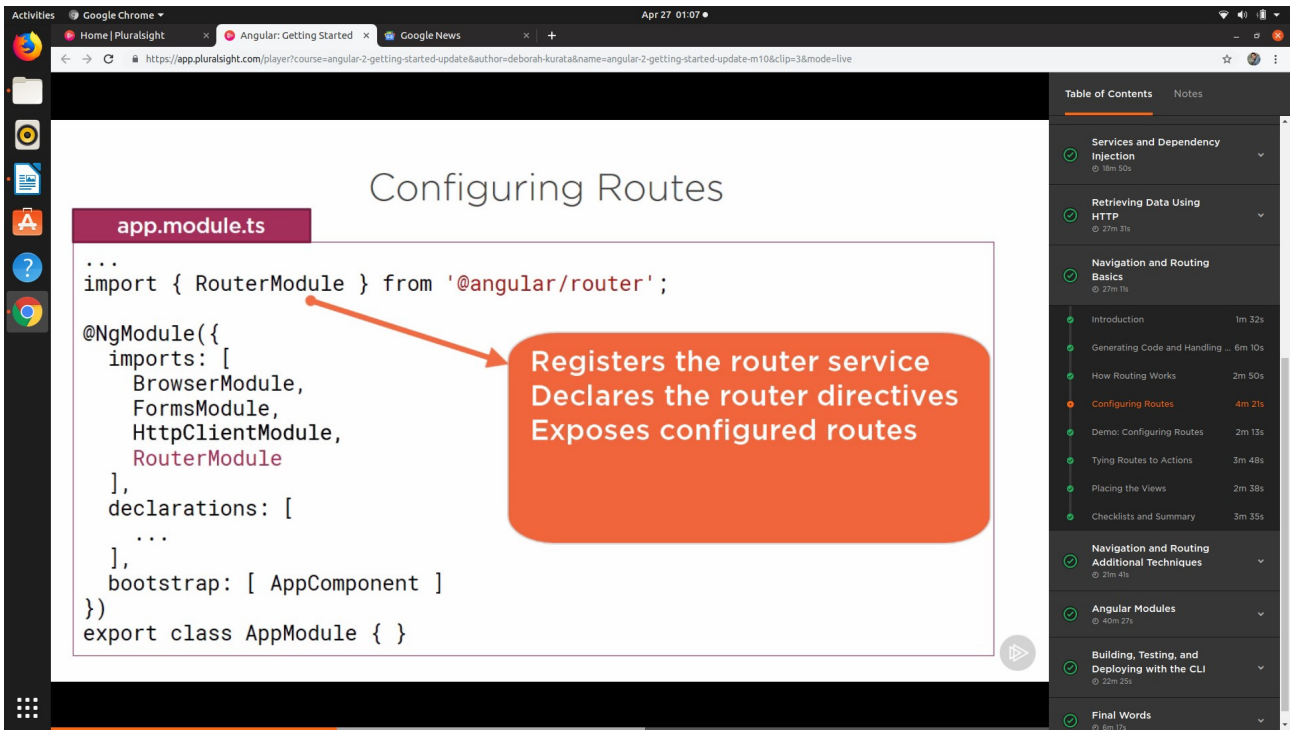


How Routing Works:



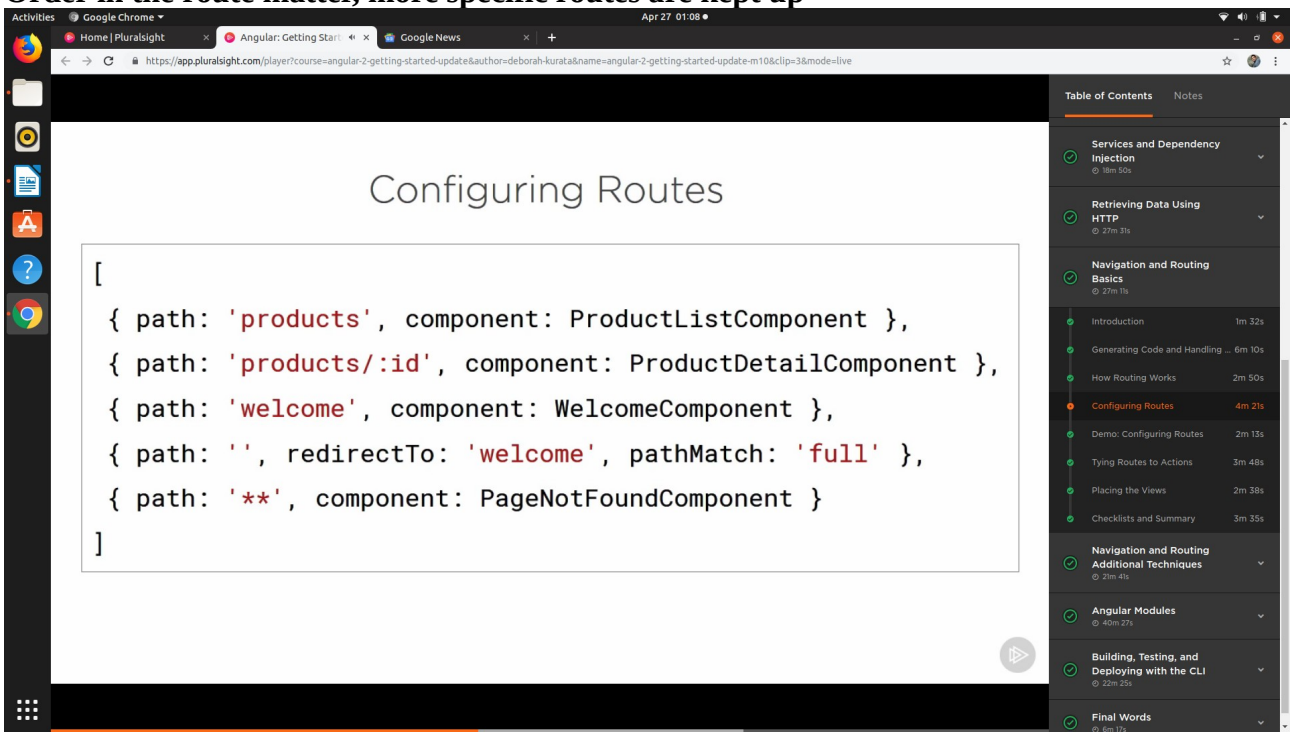

How Routing Works
Acme Product Management
Home
Product List
<a routerLink="/products">Product List
{ path: 'products', component: ProductListComponent }
<router-outlet></router-outlet>
product-list.component.ts
import { Component } from '@angular/core';
@Component({
templateUrl: './product-list.component.html'
})
export class ProductListComponent { }
Table of Contents
Notes
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Introduction 1m 32s
Generating Code and Handling ... 6m 10s
How Routing Works 2m 50s
Configuring Routes 4m 21s
Demo: Configuring Routes 2m 13s
Tying Routes to Actions 3m 48s
Placing the Views 2m 38s
Checklists and Summary 3m 35s
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Final Words


Configuring Routes
app.module.ts
...
import { RouterModule } from '@angular/router';
@NgModule({
imports: [
BrowserModule,
FormsModule,
HttpClientModule,
RouterModule
],
declarations: [
...
],
bootstrap: [AppComponent]
})
export class AppModule { }
Registers the router service
Table of Contents
Notes
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Introduction 1m 32s
Generating Code and Handling ... 6m 10s
How Routing Works 2m 50s
Configuring Routes 4m 21s
Demo: Configuring Routes 2m 13s
Tying Routes to Actions 3m 48s
Placing the Views 2m 38s
Checklists and Summary 3m 35s
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Final Words

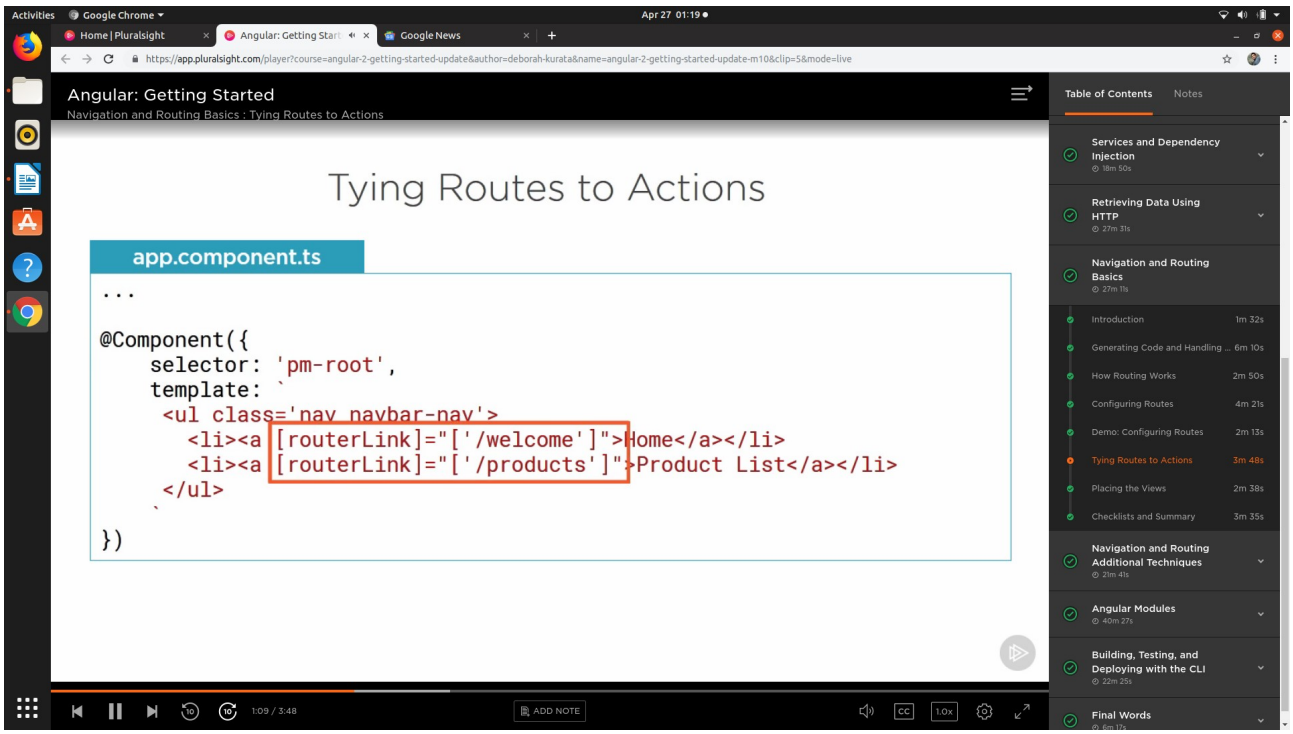


RouterModule will take and array of below data as RouterModule.forRoot([])

Order in the route matter, more specific routes are kept up



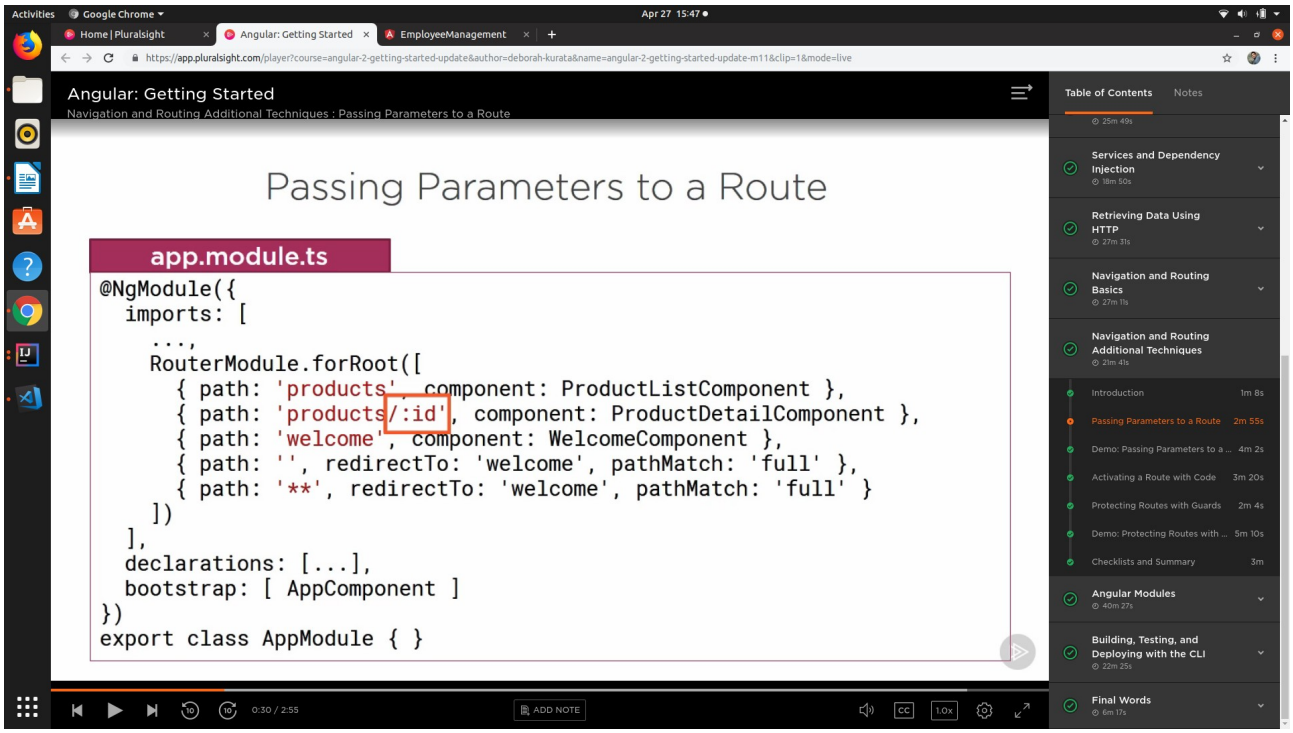
Setting up routes using routerlink directives



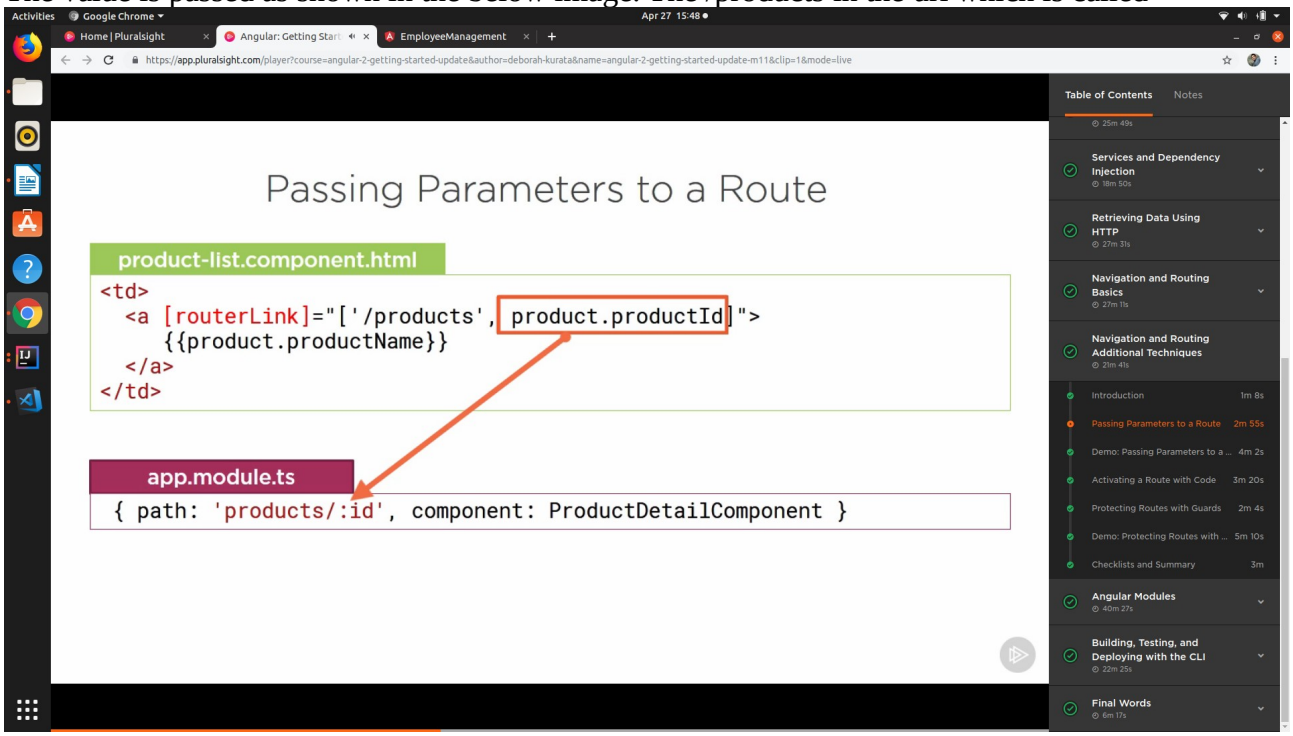
To displaye the router outlet on a page we use the <router-outlet></router-outlet> directive.

Navigation and Routing Additional Technique

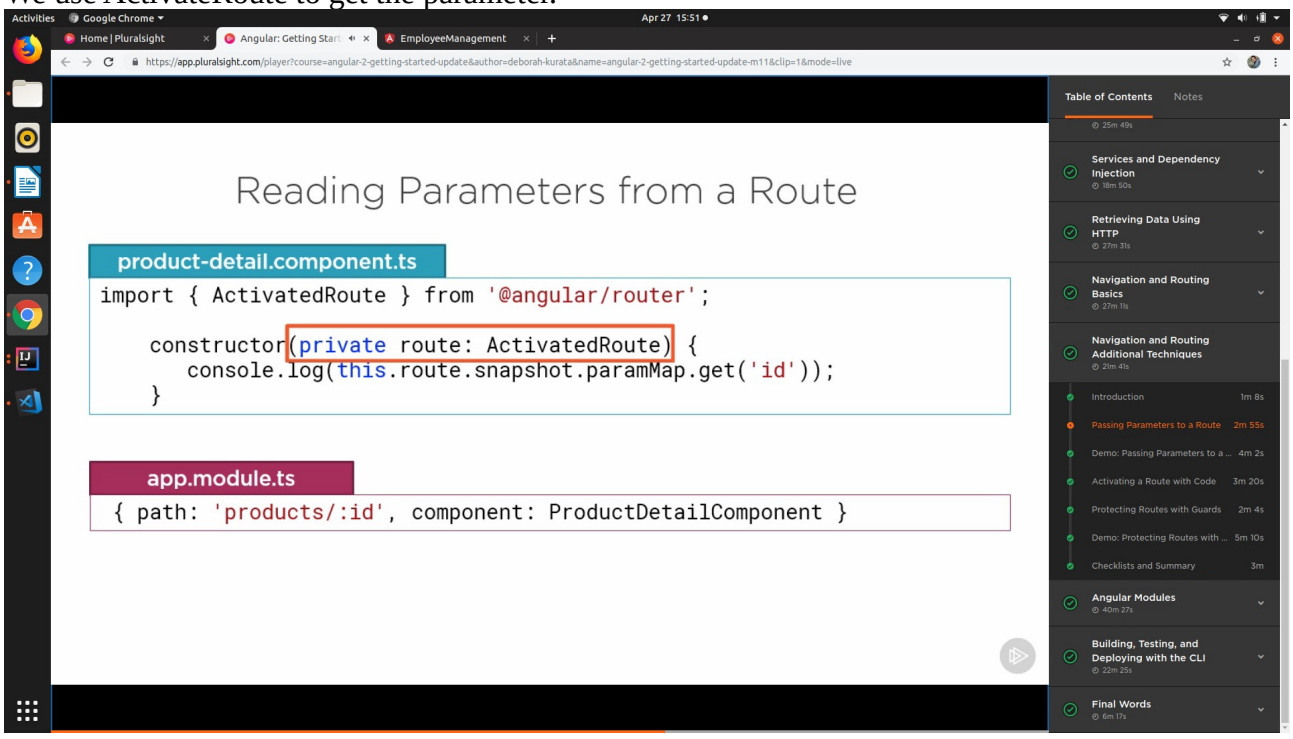
Passing Id or any value in the route



The value is passed as shown in the below image. The /products in the url which is called

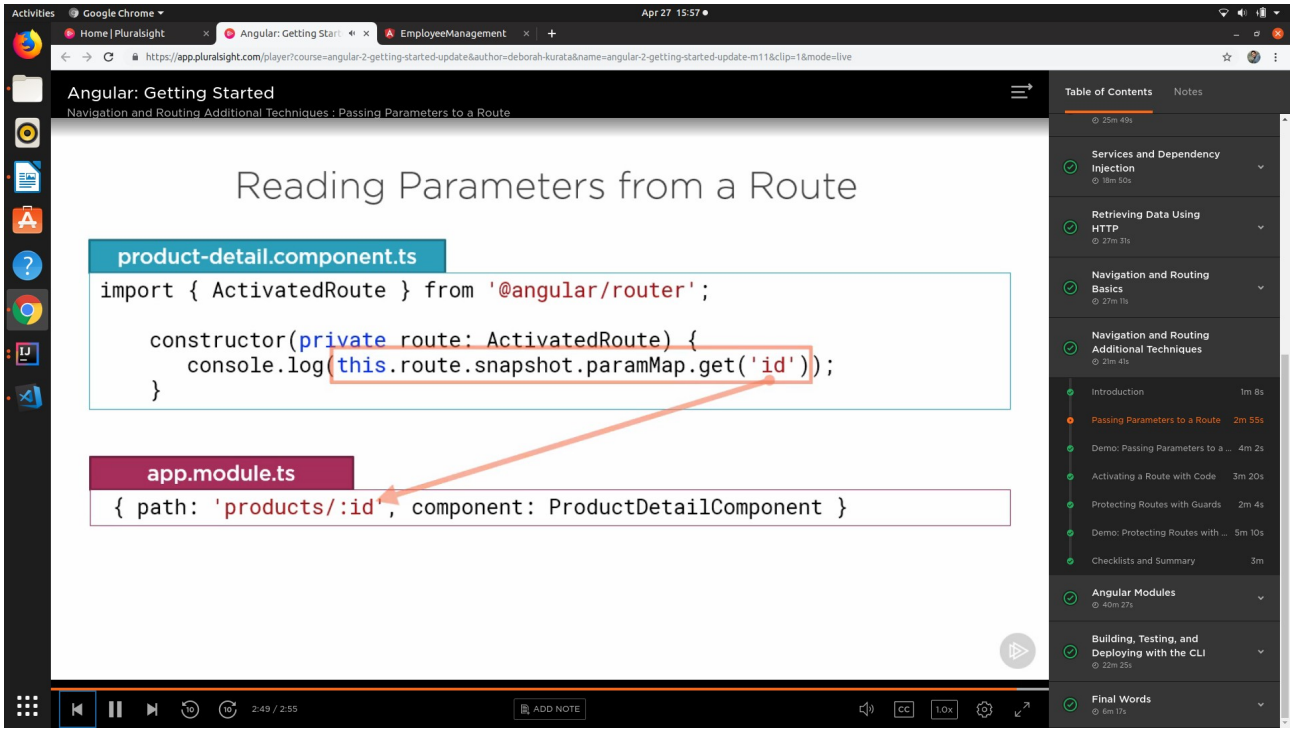


We use ActivateRoute to get the parameter.

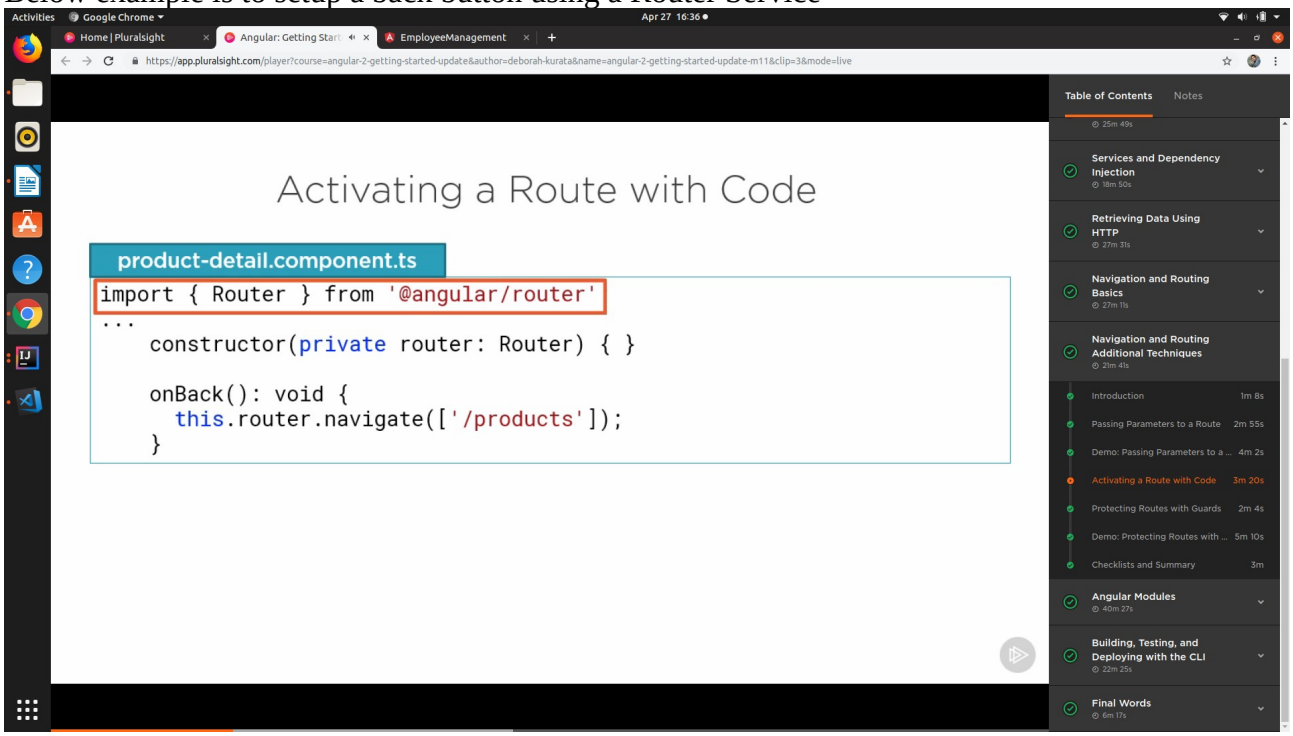


There are two ways of doing this ither use a snapshot as shown above or we can use Observable.
*If you need the initial value than use snapshot
* If there is a chance of parameter getting change without leaving the page use Observable.;
Example: If there is next button and after clicking it the id changes in the url then we should use Observable.

In the below image the string name given in the parametereMap should have the same value as in the route.

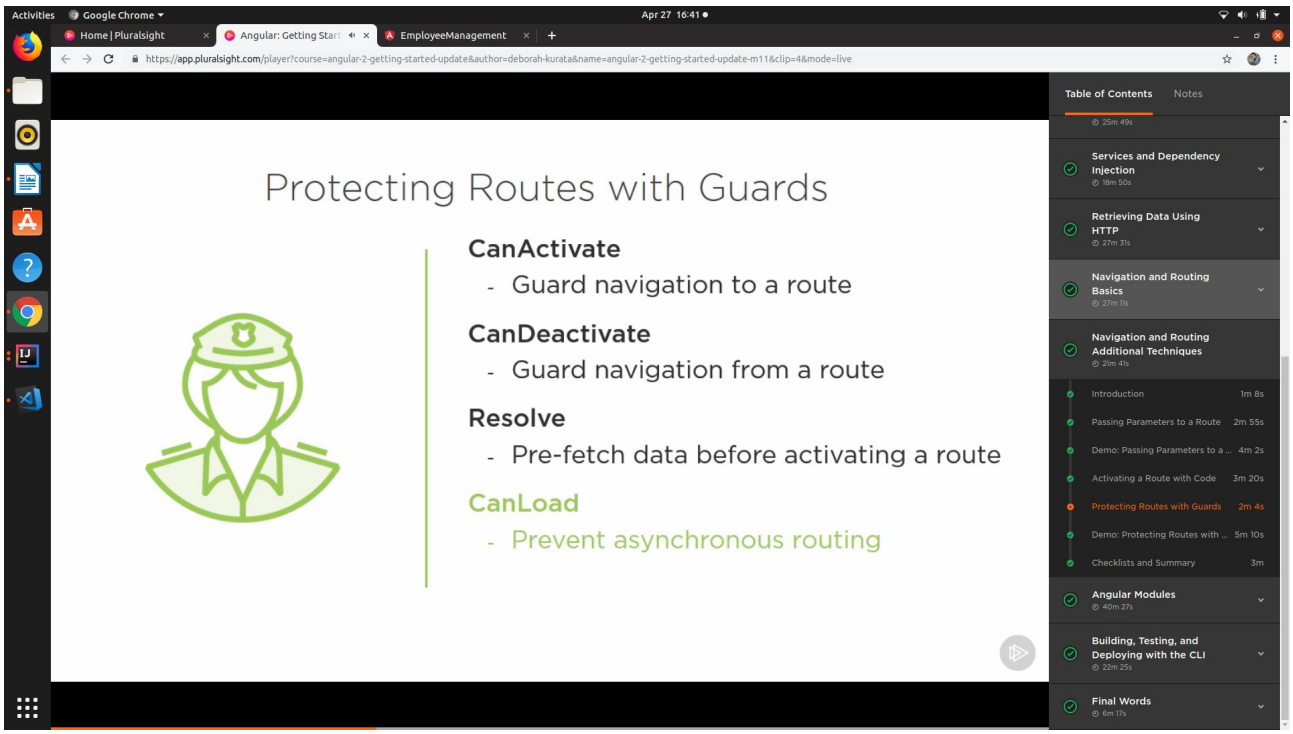


Below example is to setup a back button using a Router Service

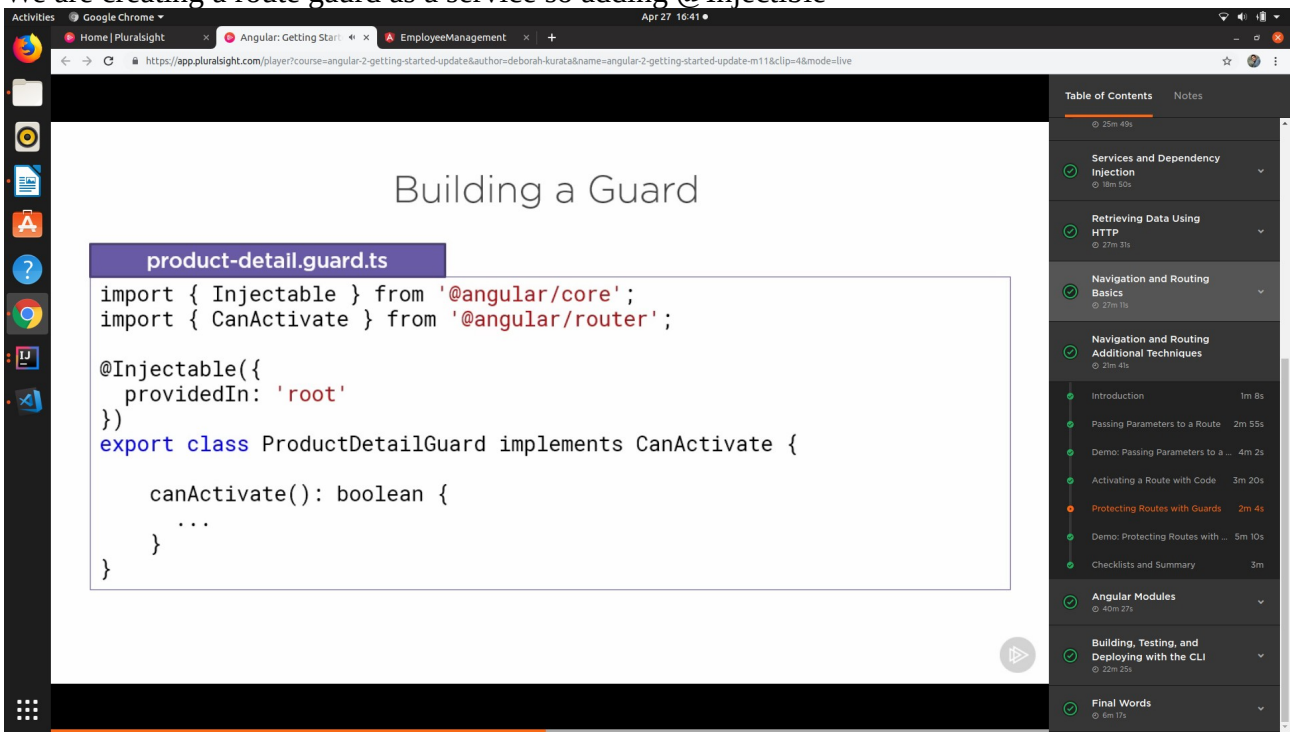


Protecting Routes with Gaurds(Route Gaurd)

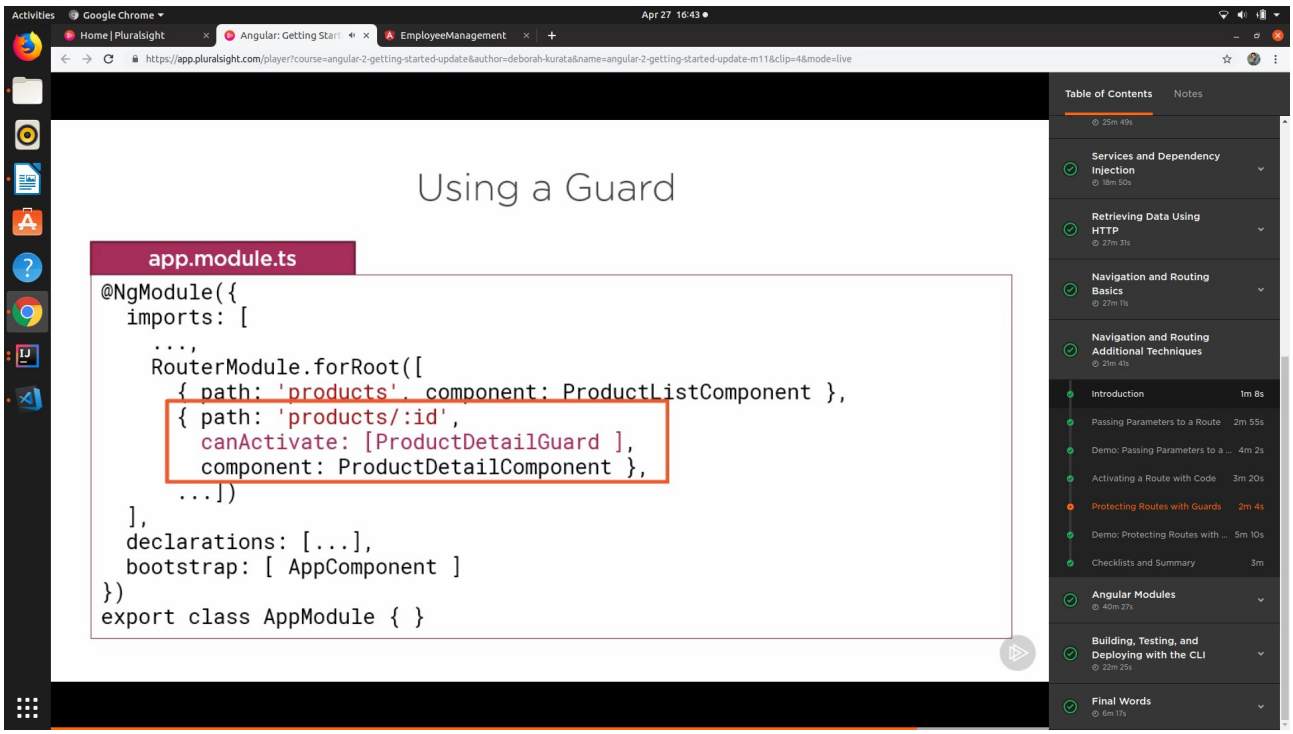
Types of Route Gaurds:



We are creating a route gaurd as a service so adding @Injectible

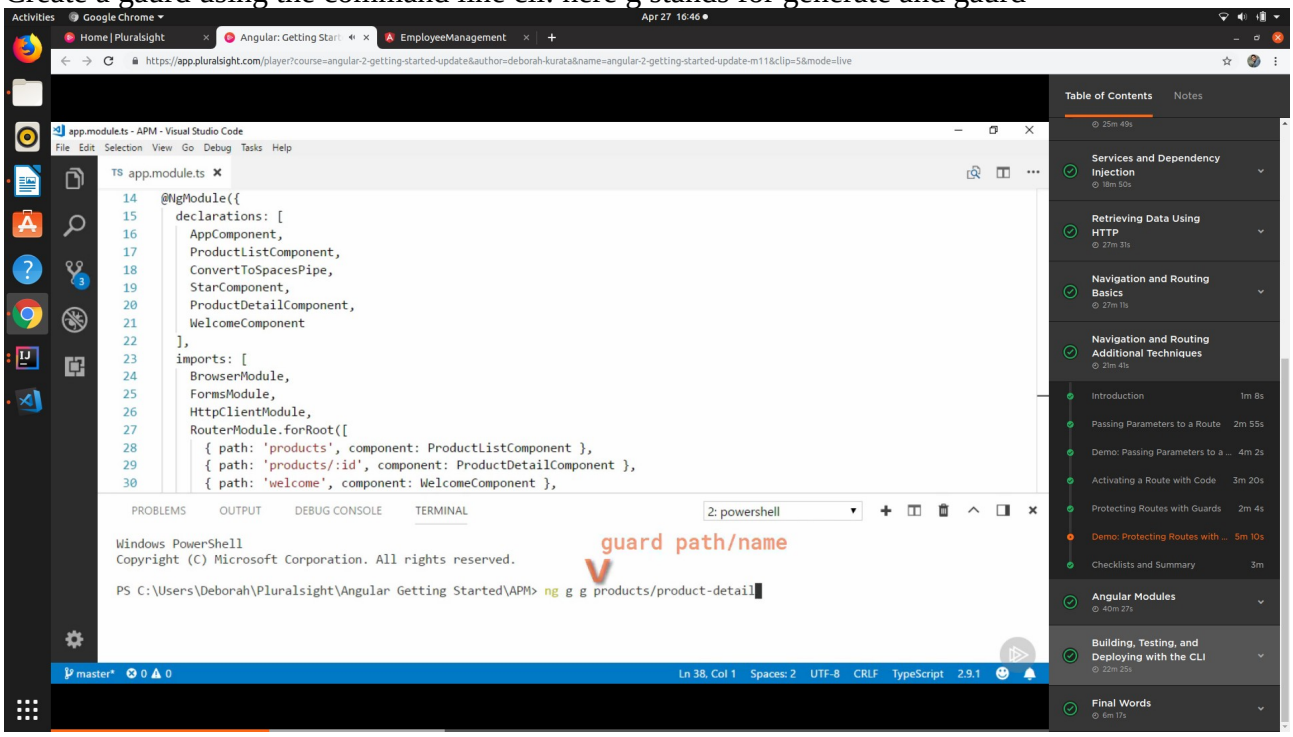


This route should be added to the RouterModule where we want to route based on some condition.

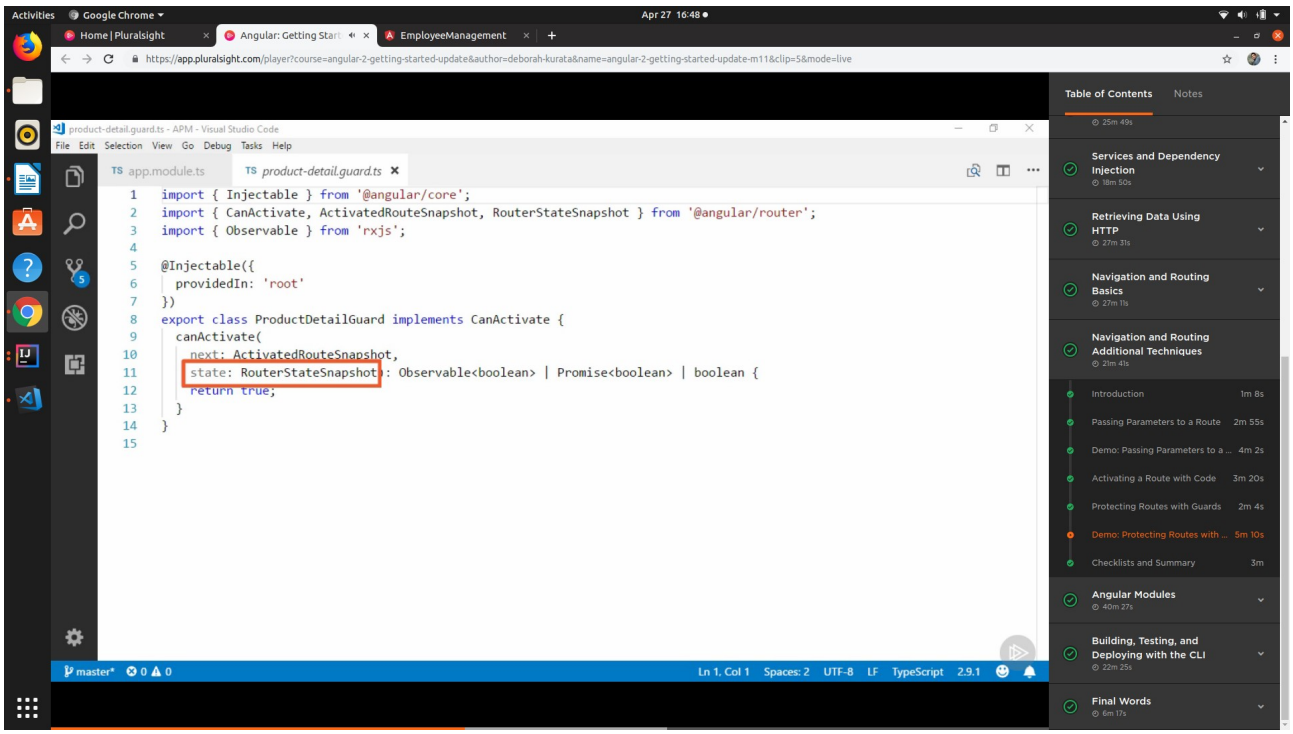


Below example will help us understand the implementation

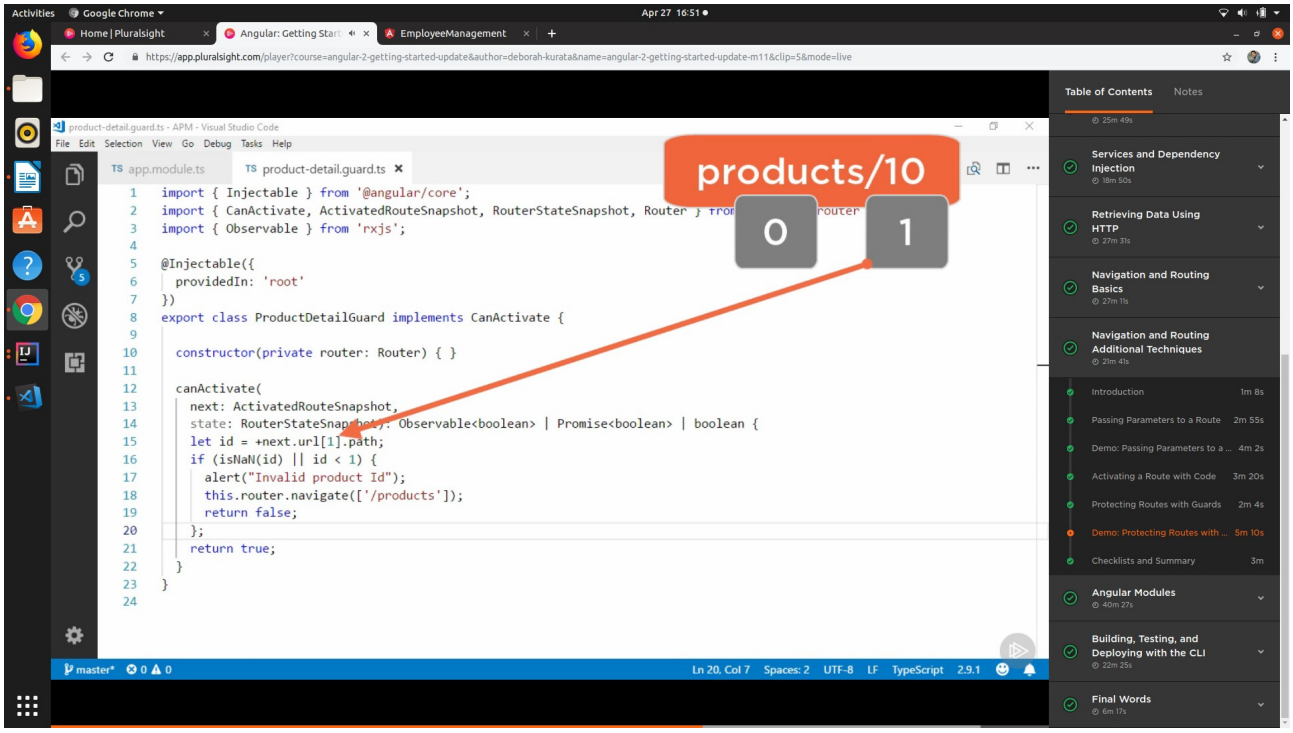
Create a gaurd using the command line cli: here g stands for generate and gaurd



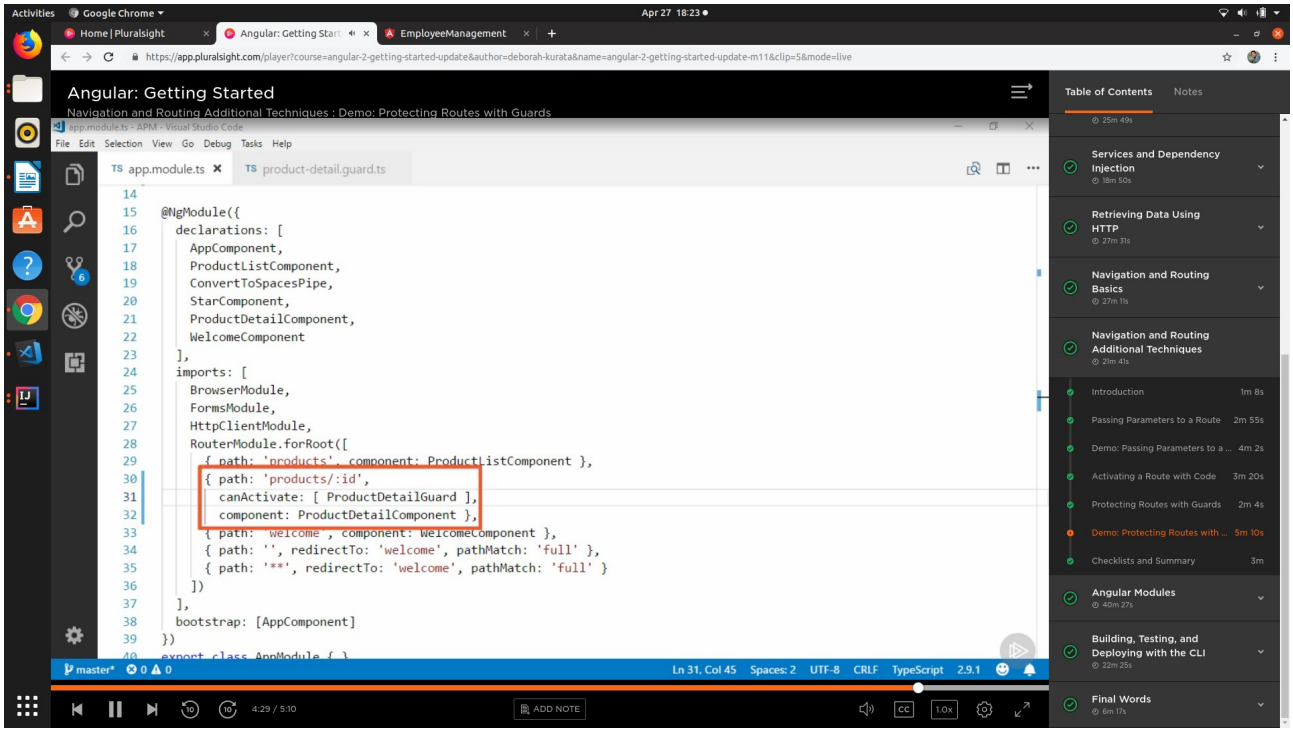
It automatically creates a route gaurd with canActivate as method. We can change it as per our requirement.
CanActivate has two paramaters :ActivatedRouteSnapshot = to provide current route information
RouterStateSnapshot= to provide state information of the route



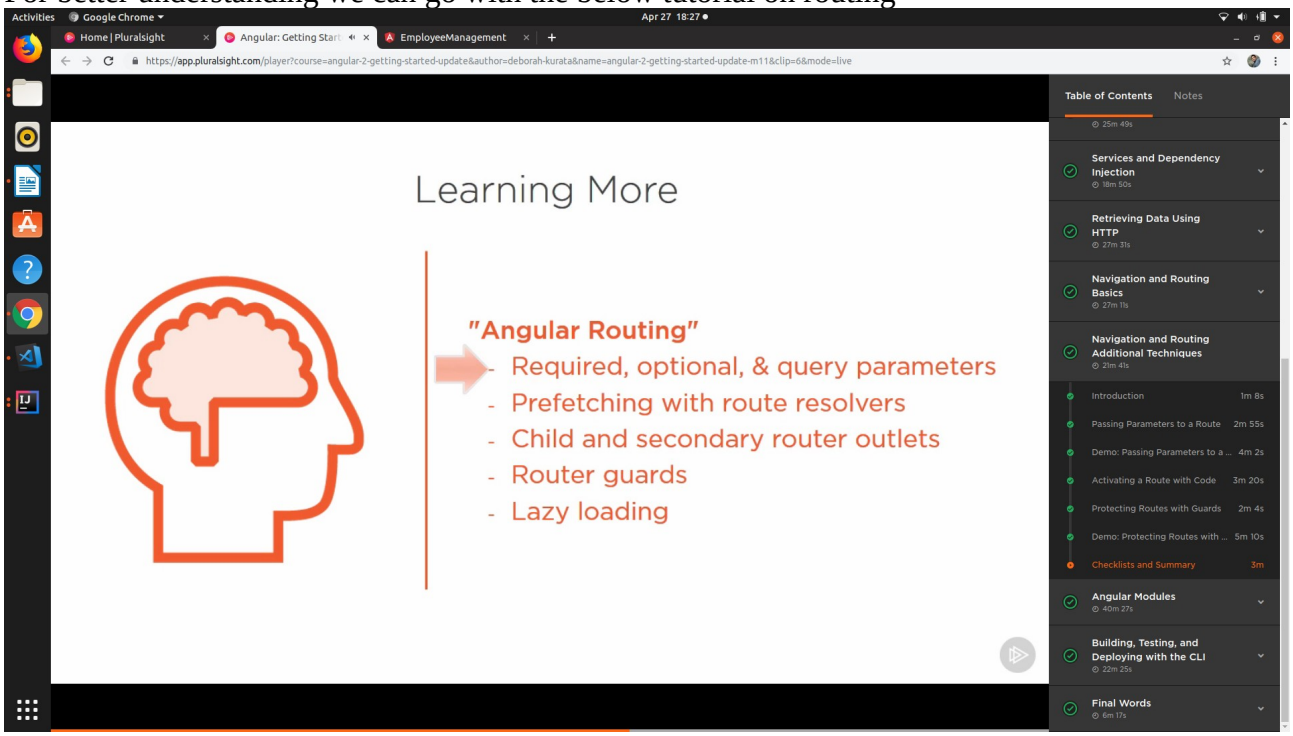
let id =+next.url[1].path;
here producects is the [0] th value and [1] is the value we need.
The below route we are checking the value, it should be in number type
+ infront of the above code converts the url passed string into a number



Now we can add it in the below location
canActivate has array type data and we can add multiple route gaurds to it , in the current scenario we have only one route gaurd

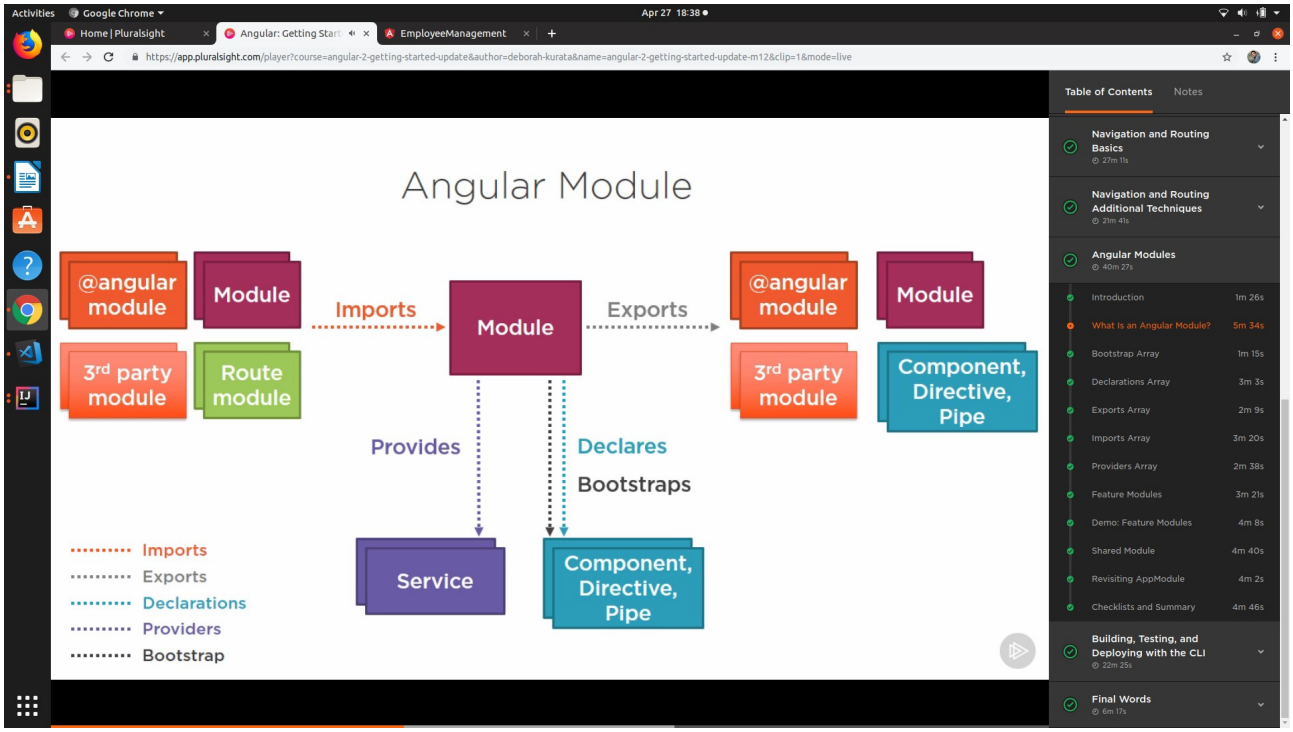


For better understanding we can go with the below tutorial on routing



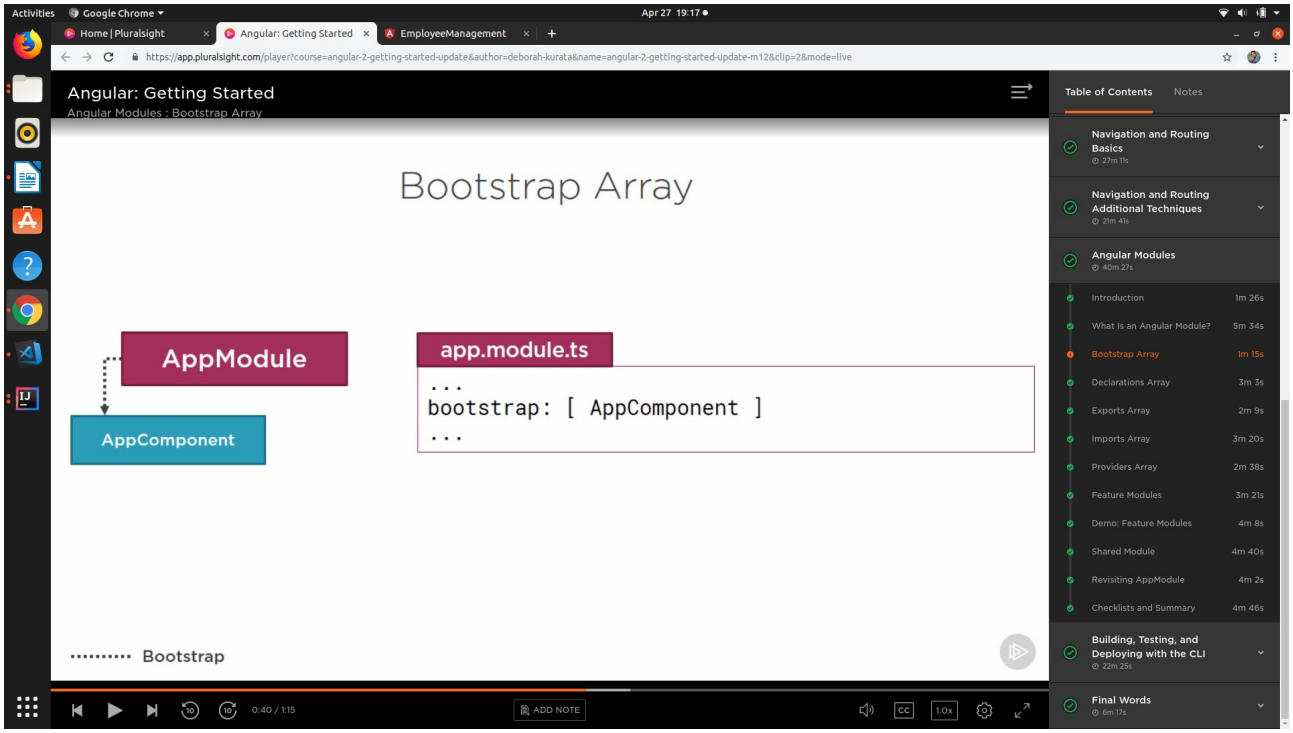
Angular Module

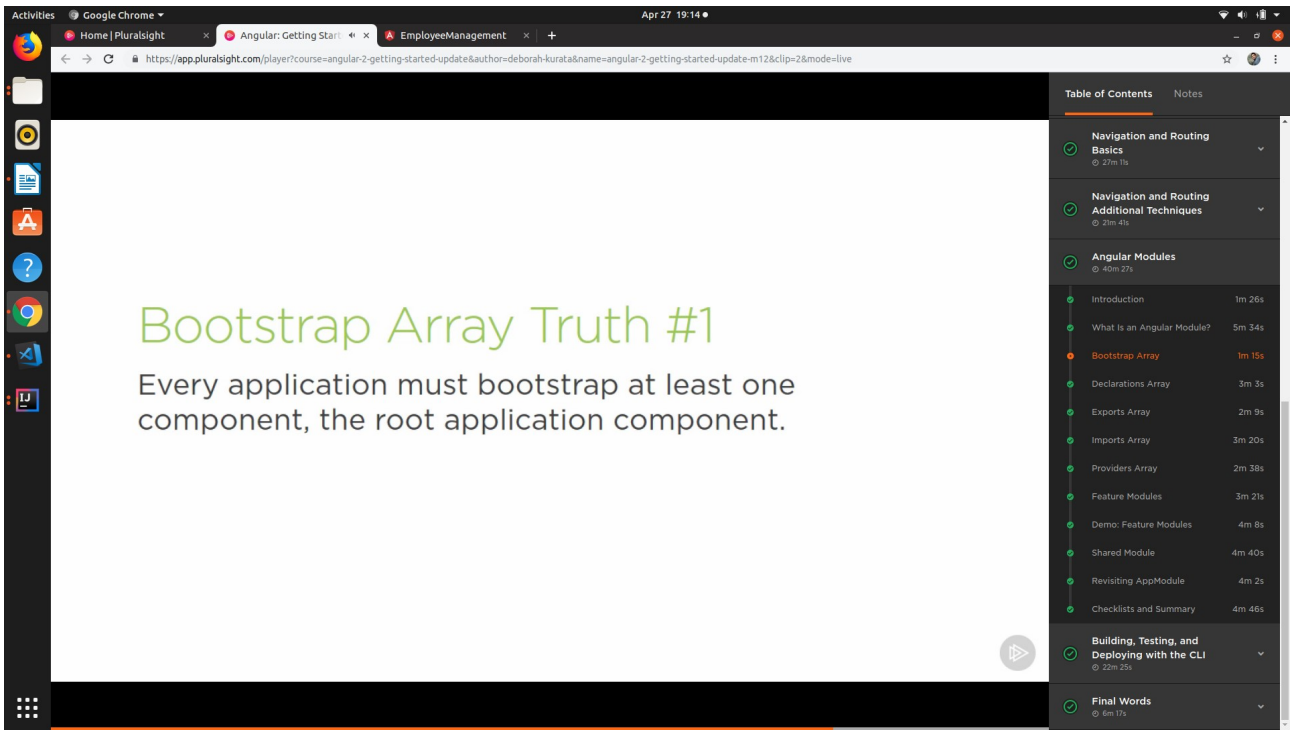
Angular module can loded eagerly and lazy to optimize the performance. In the current tutorial we have implemented the eagerly loded one.
Lazy loading one is in the routing tutorial

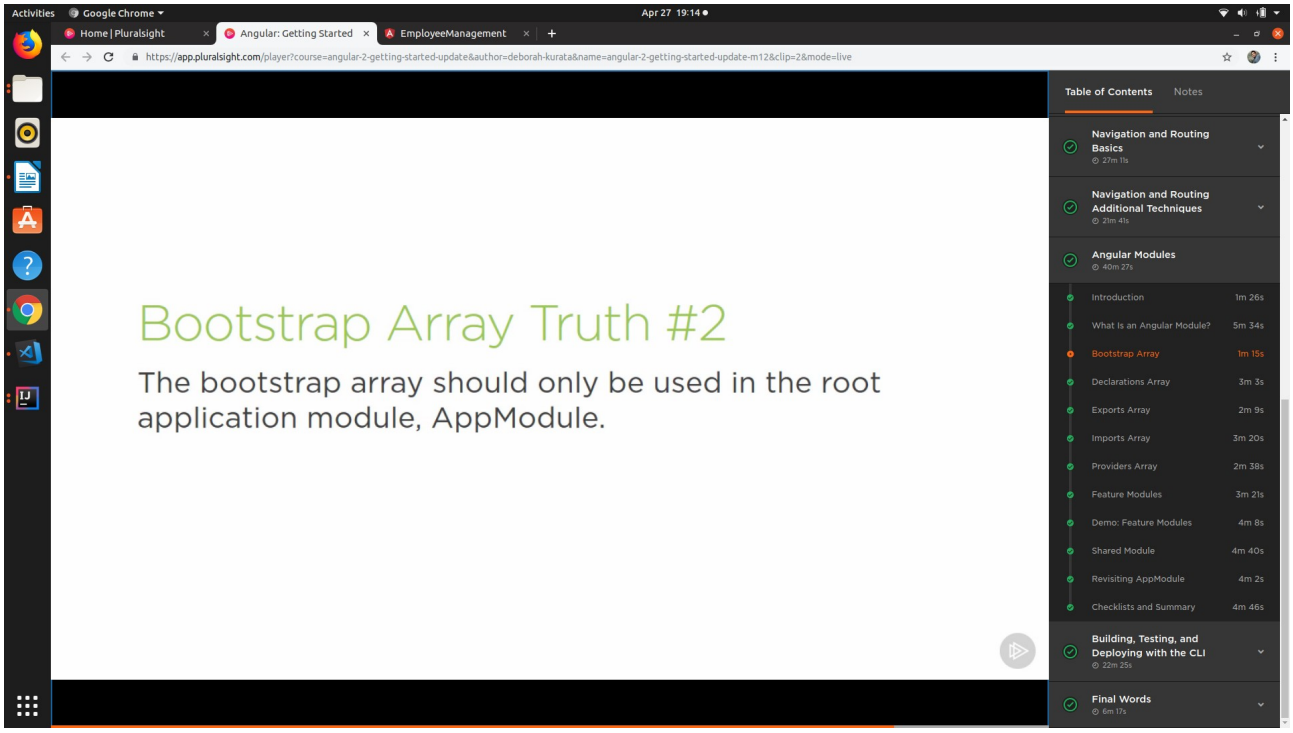


Bootstrap Array:

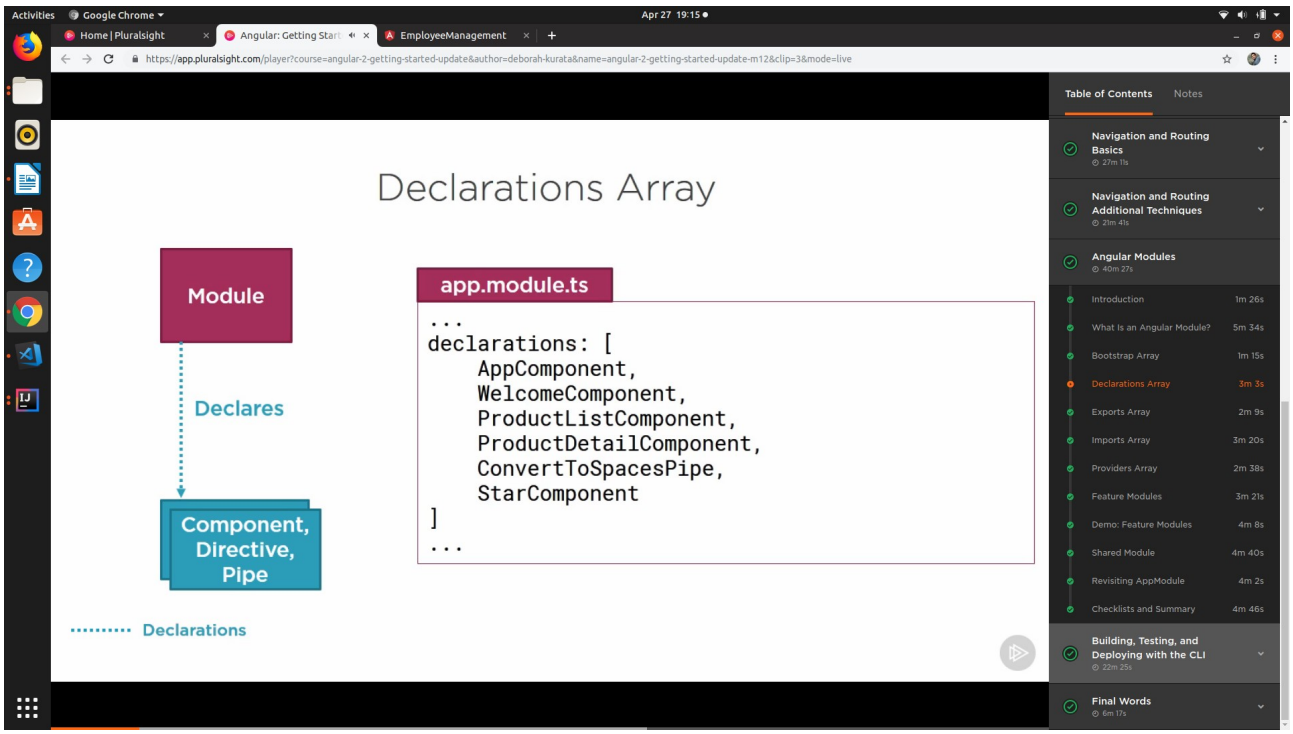
Bootstrapping must be done of at least one module

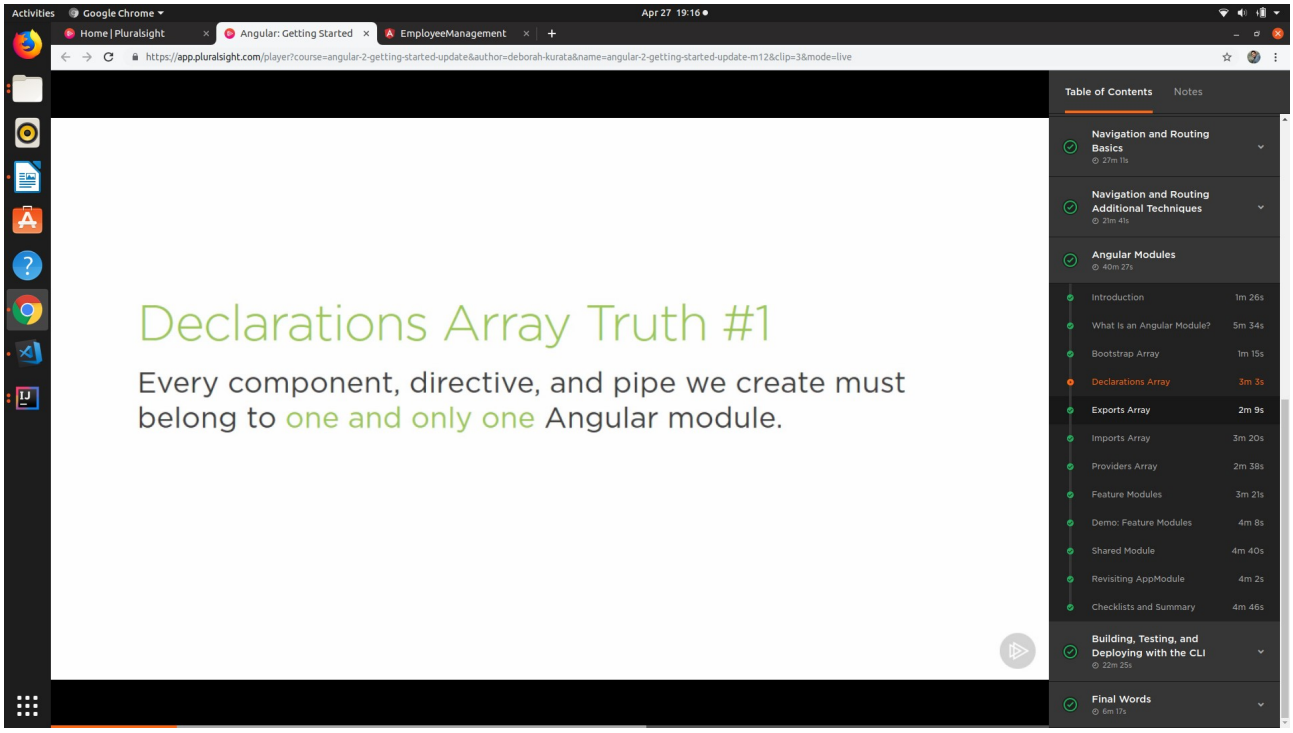




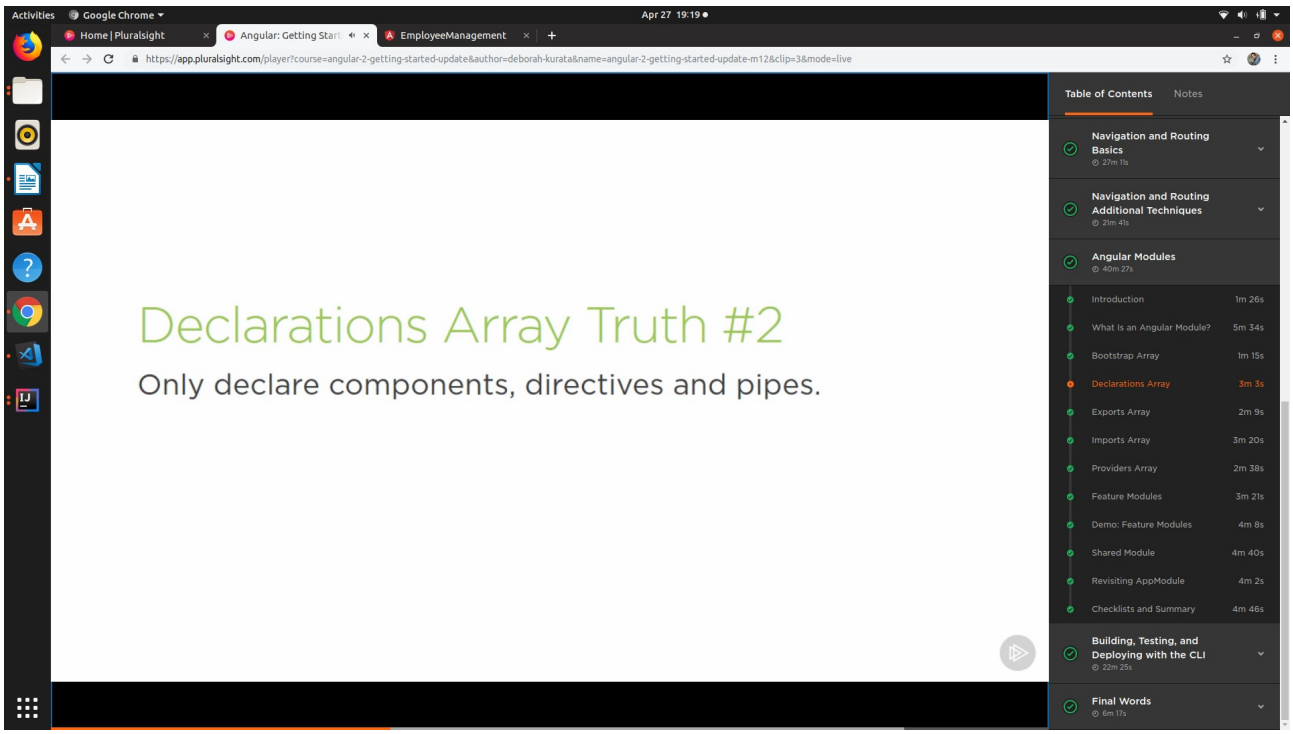


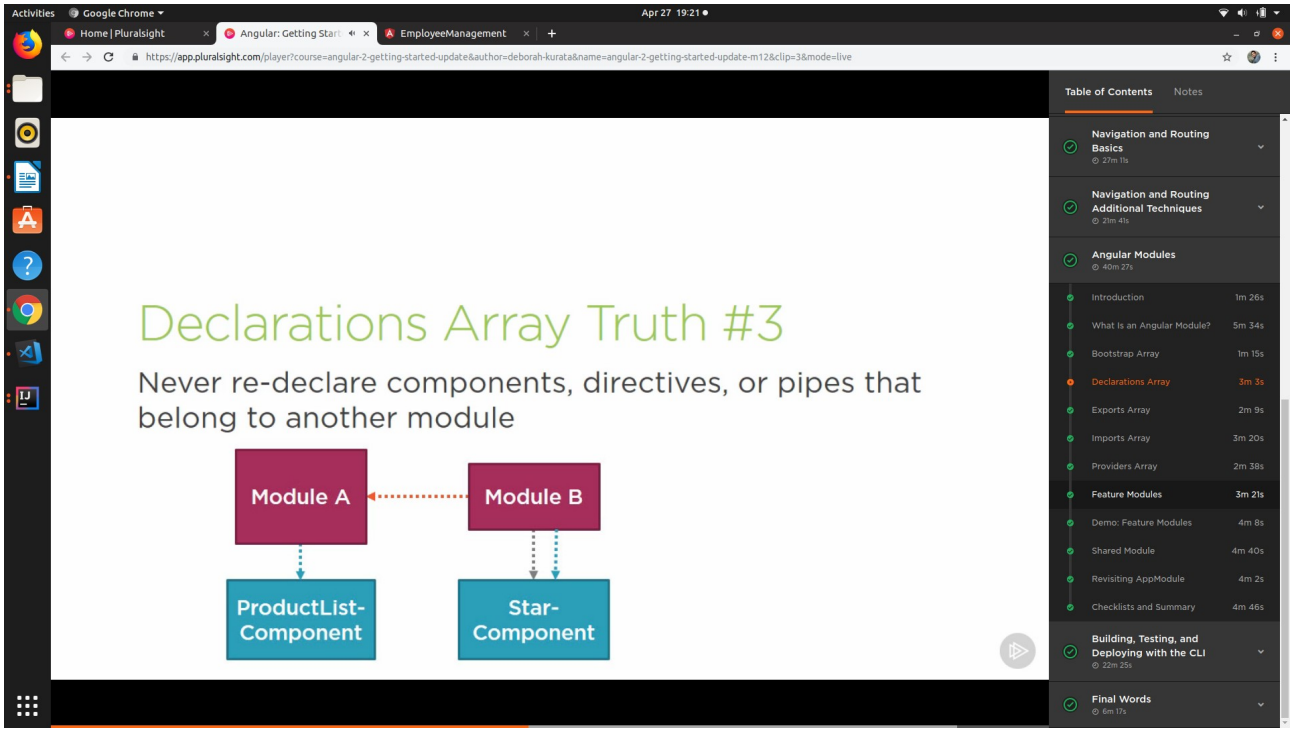
Declaration Array:



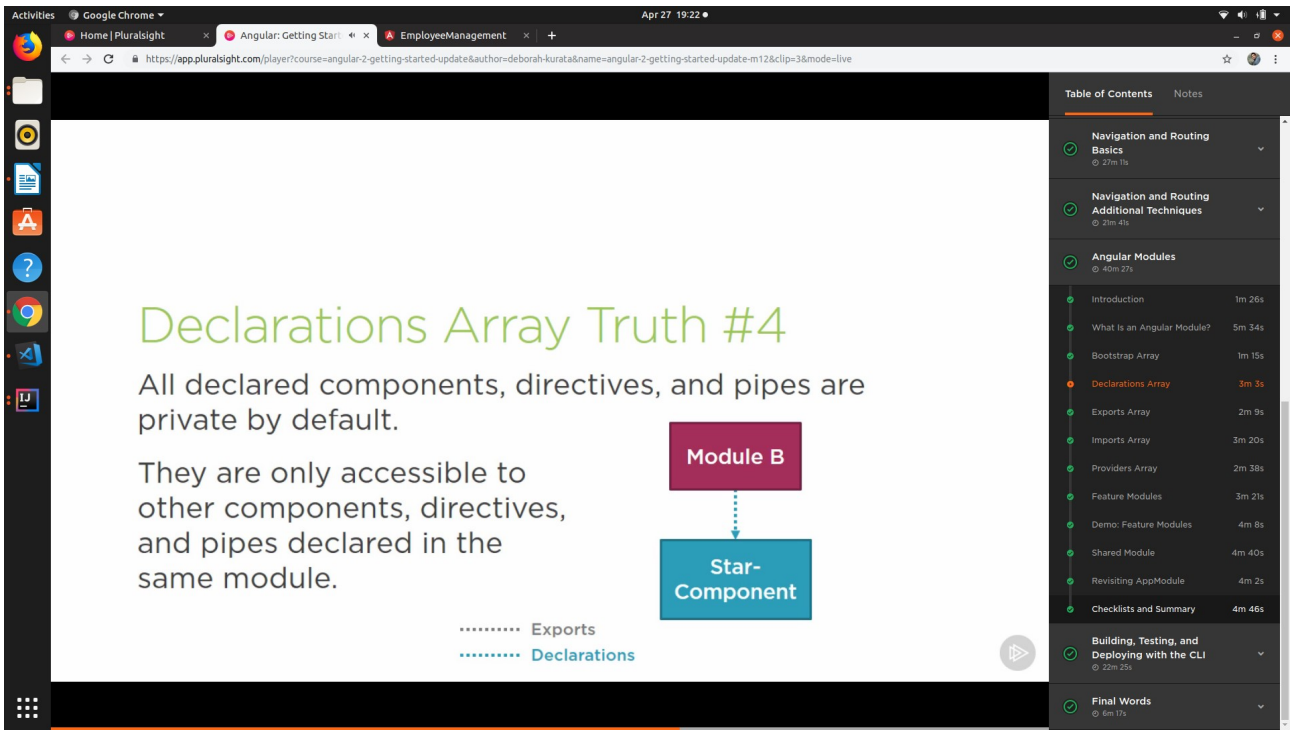


Dont add other classes, service or module to the declaration array.

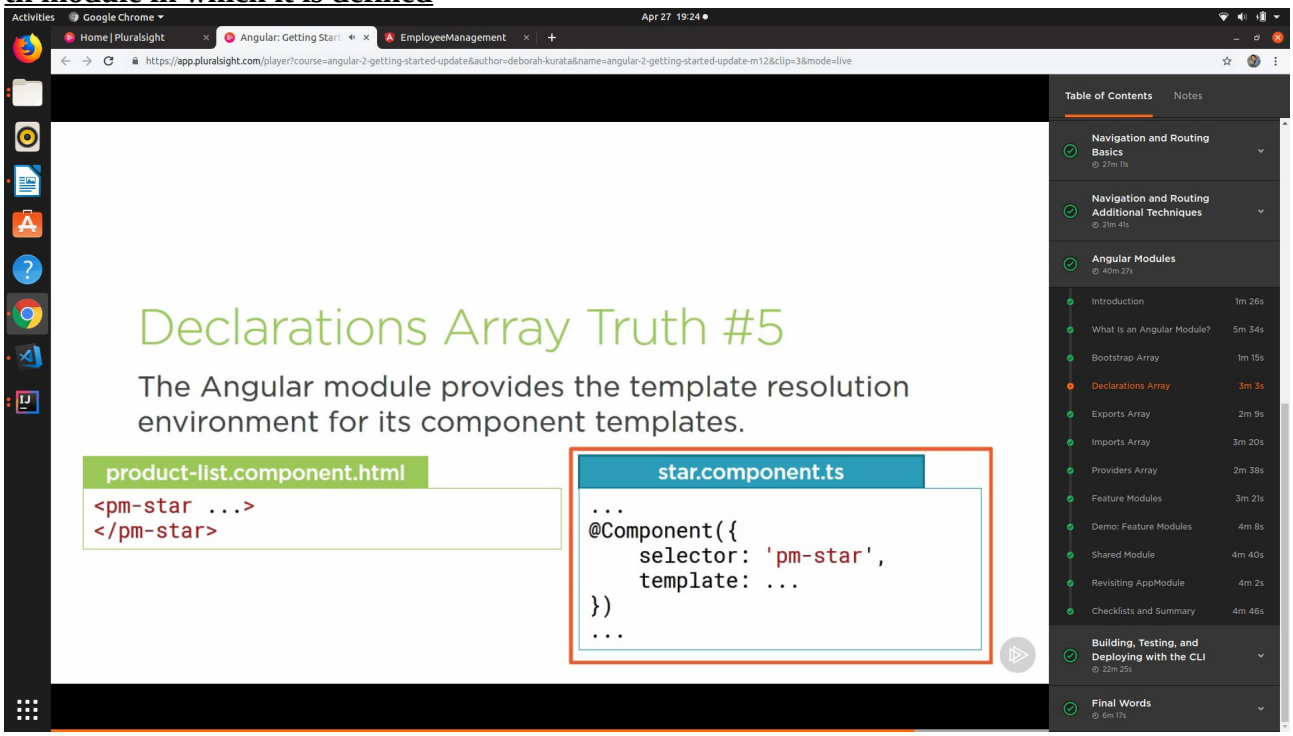




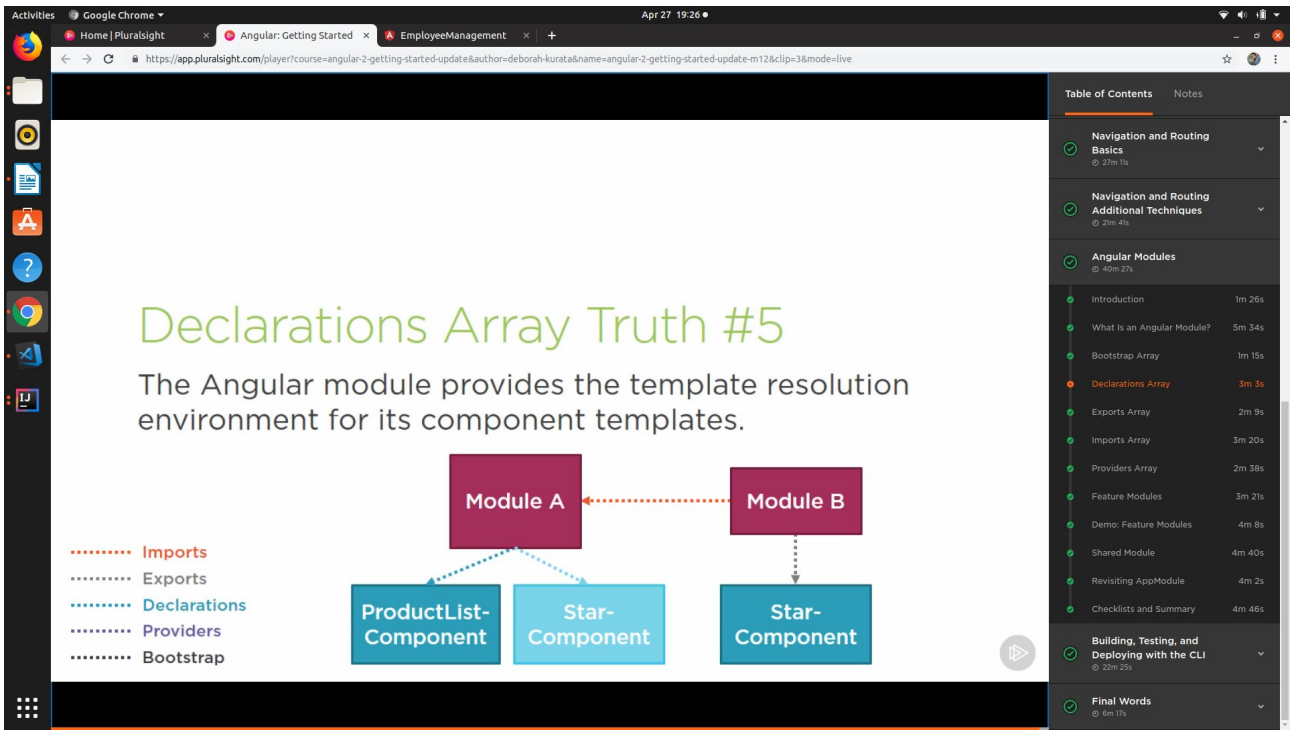
They can only be shared when exported



It means directive in the left html will look for selector of star component will bw pulled from th module in which it is defined

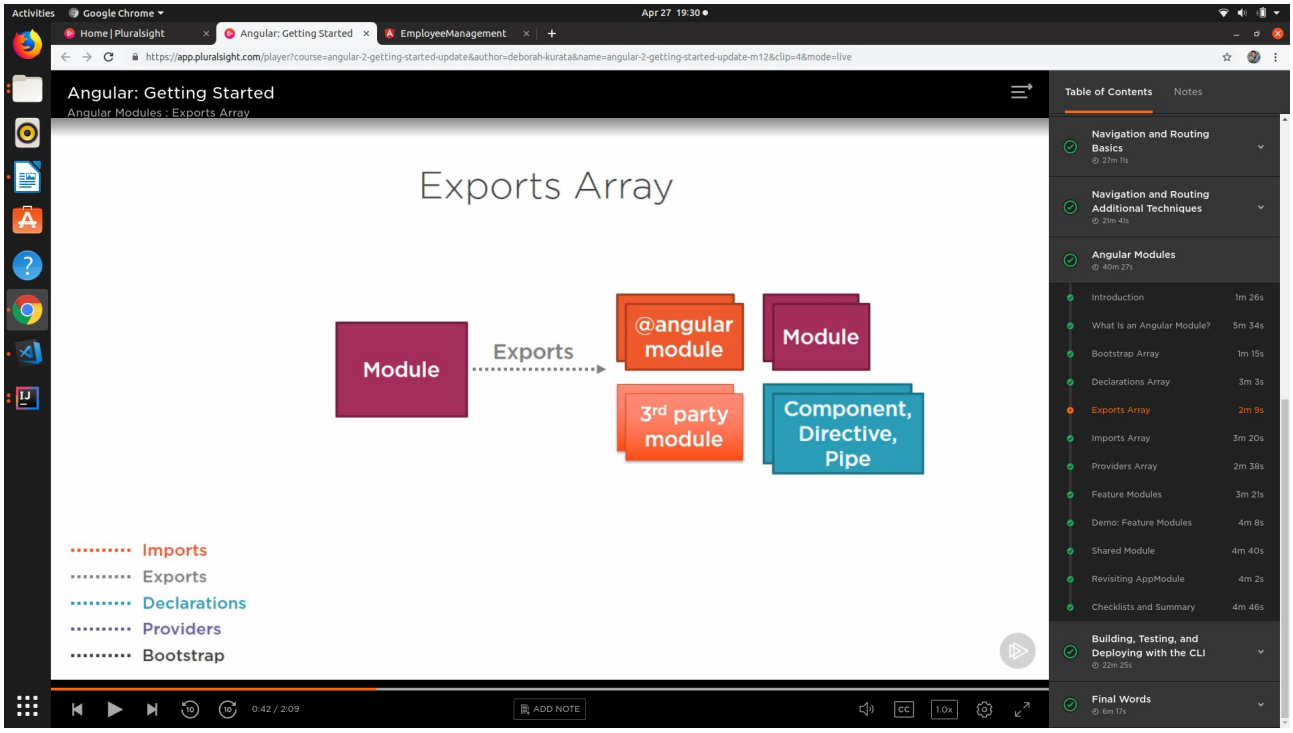


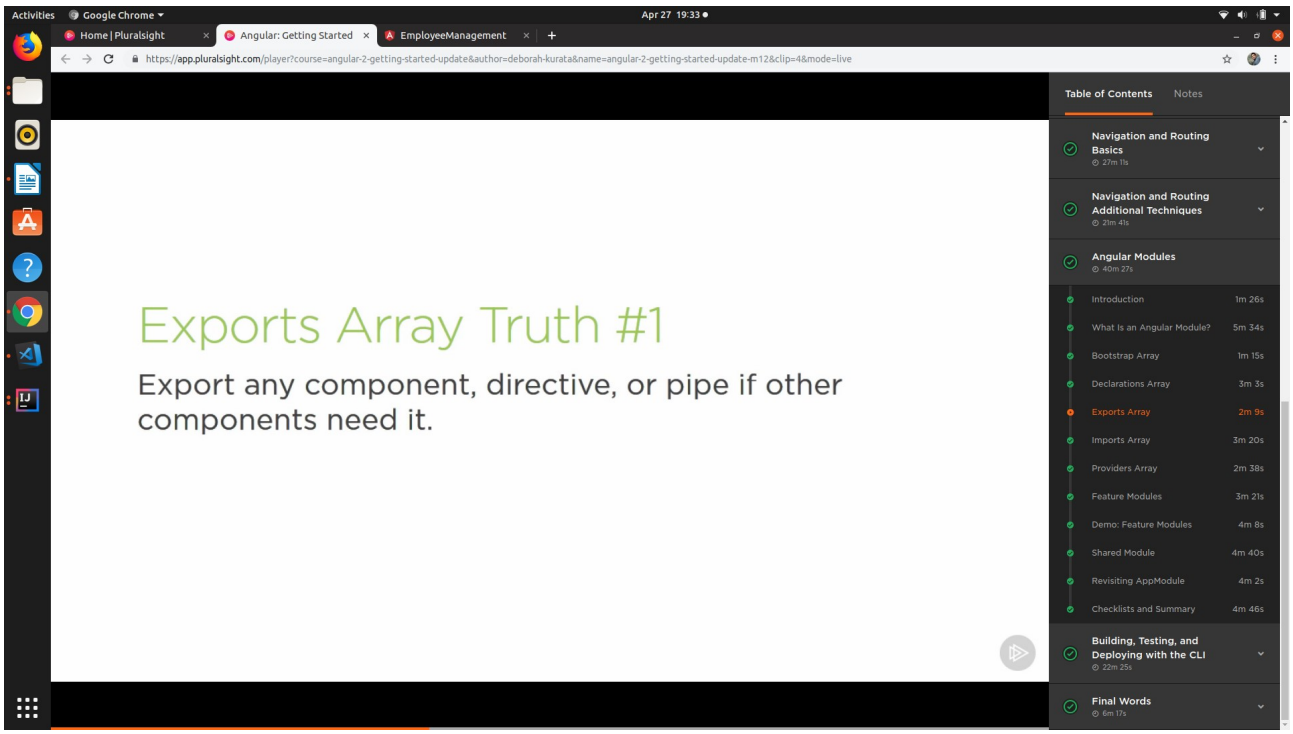
In our below example start component must be declared in the module A as it is shared by both proudct list and star component.
Another way star component can be imported from the exported module like we have shown B module below but **never both**.

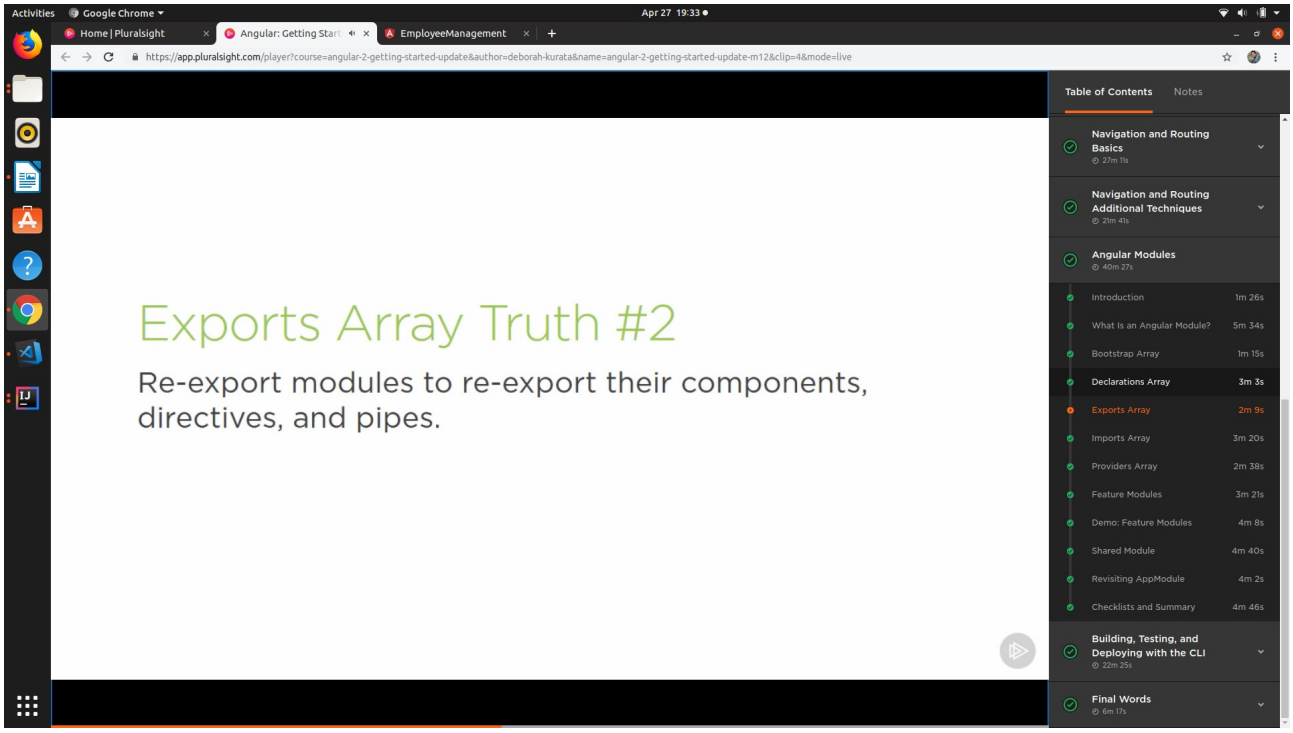


Exports Array:

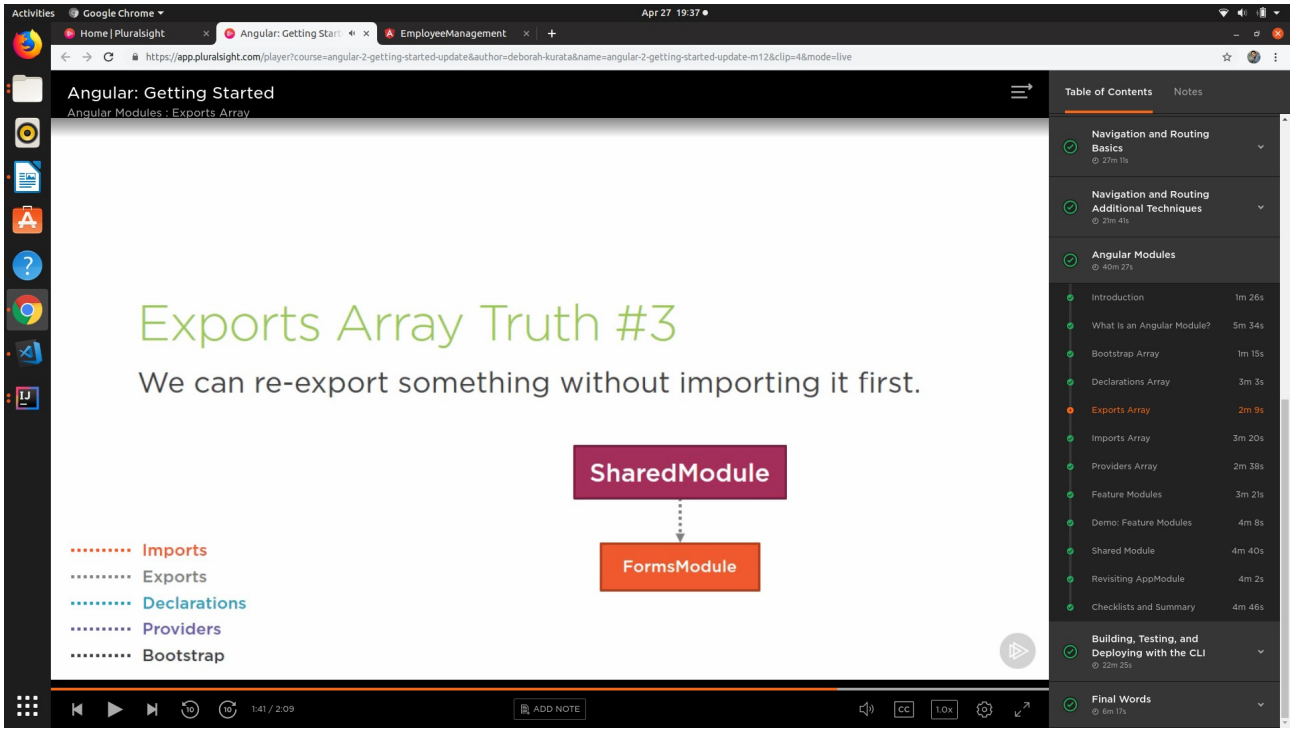
This helps to export all the shown features of a module to a different module. Like if we import the below module to another module all the feature in the right side will be available to it.



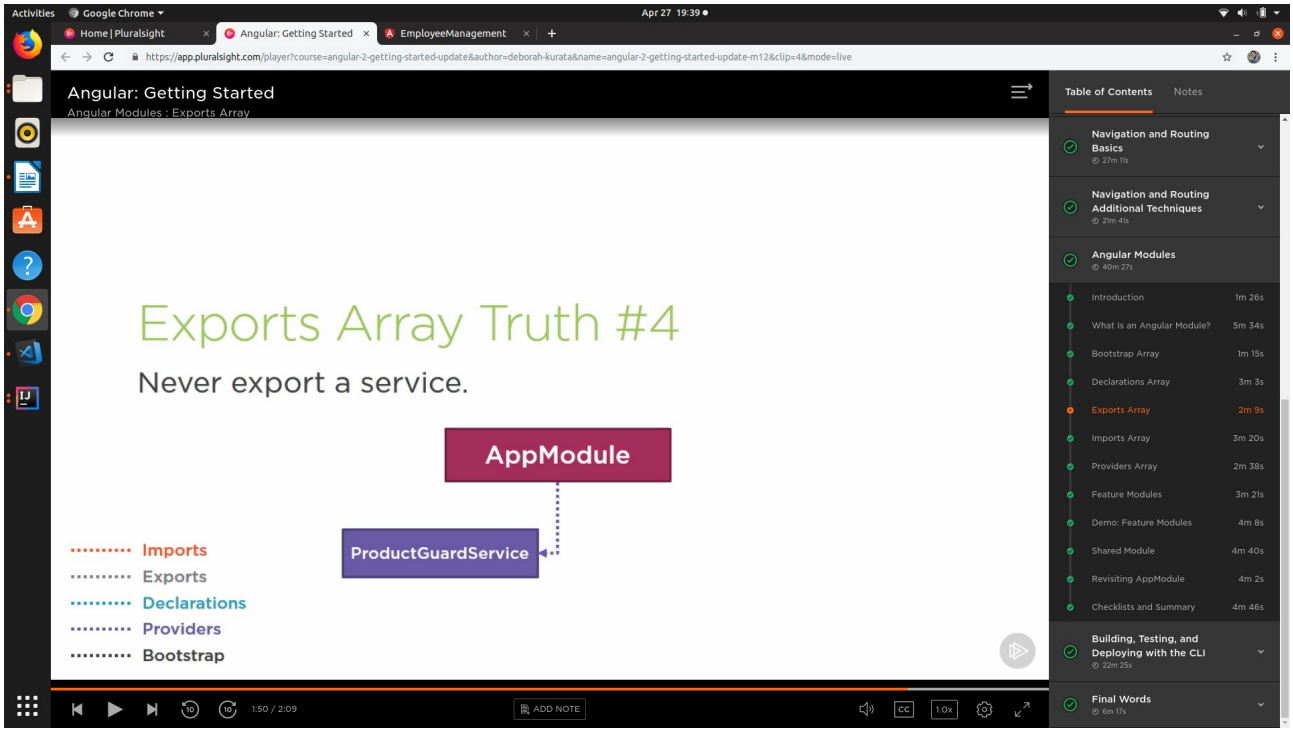




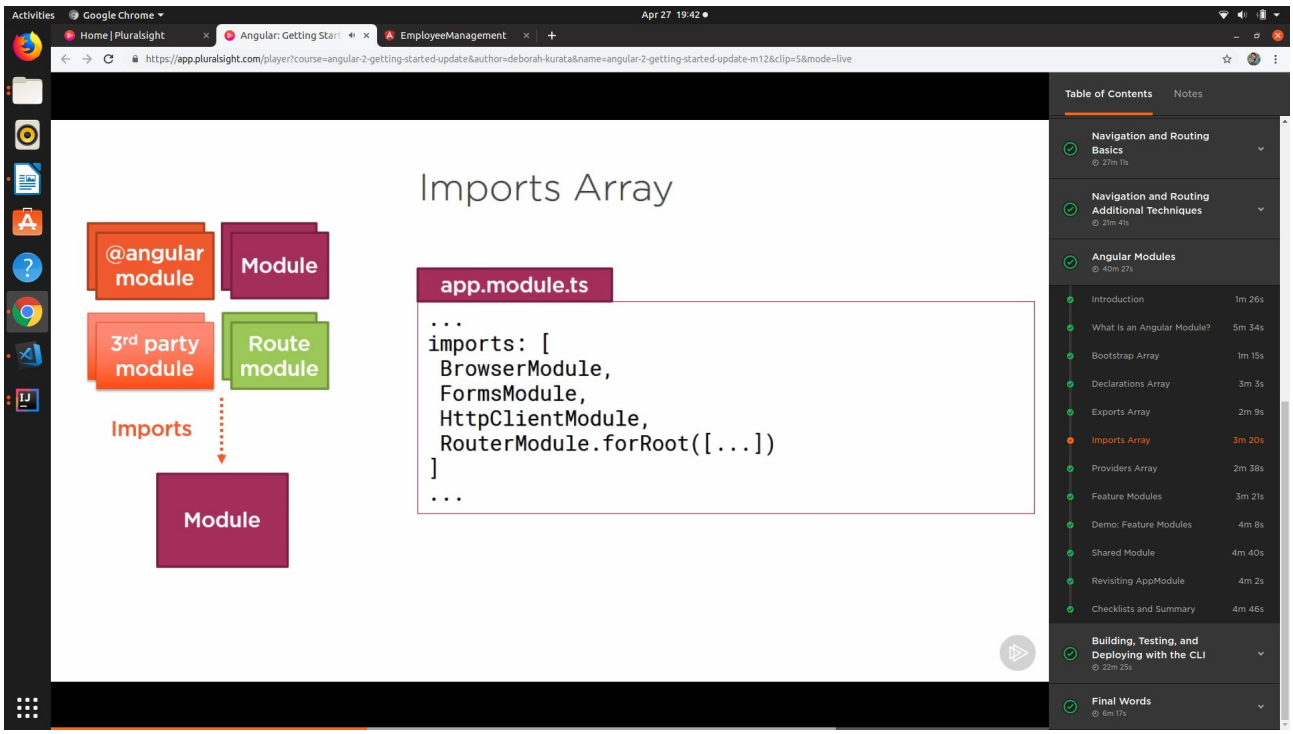
If we import a module we import all is features. In the below screenshot if we import the shared module to any module than that module will have Form Modue available to it.



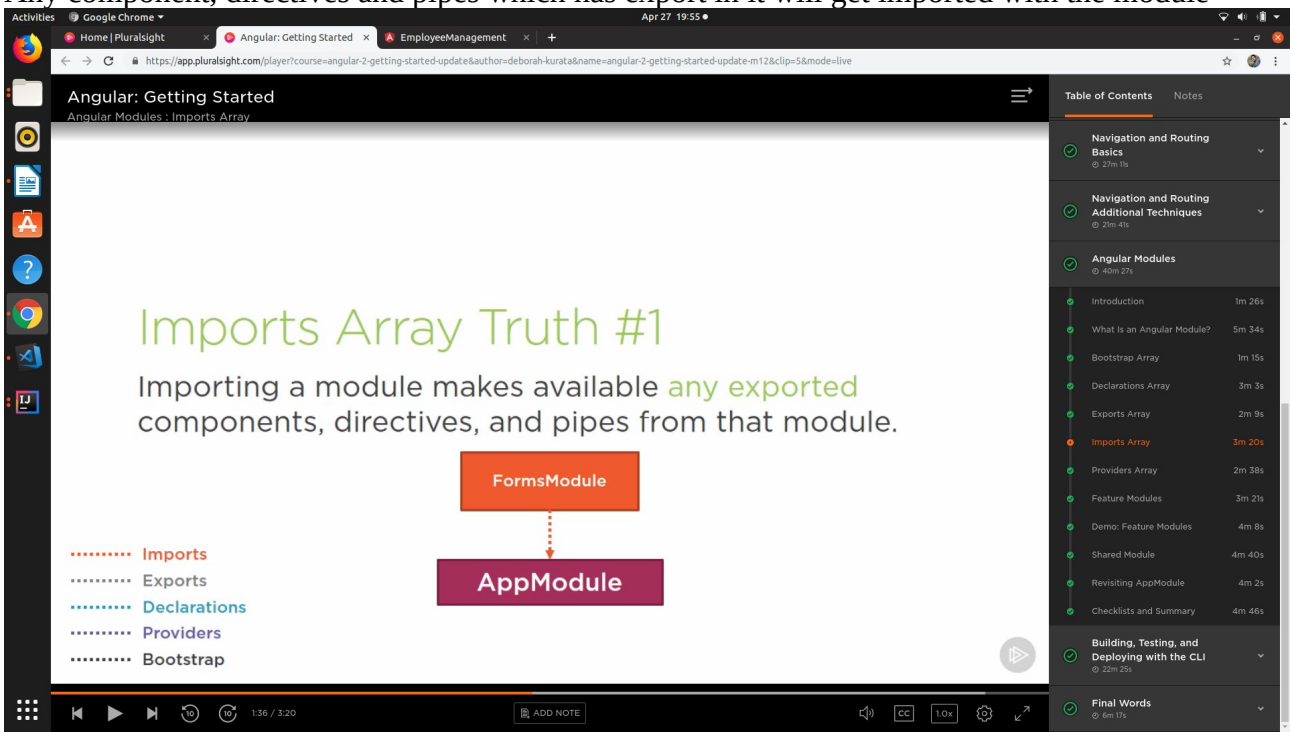
Services added to the providers array of an angular modul are registered with the root application injector making them available for injection in any class of the application

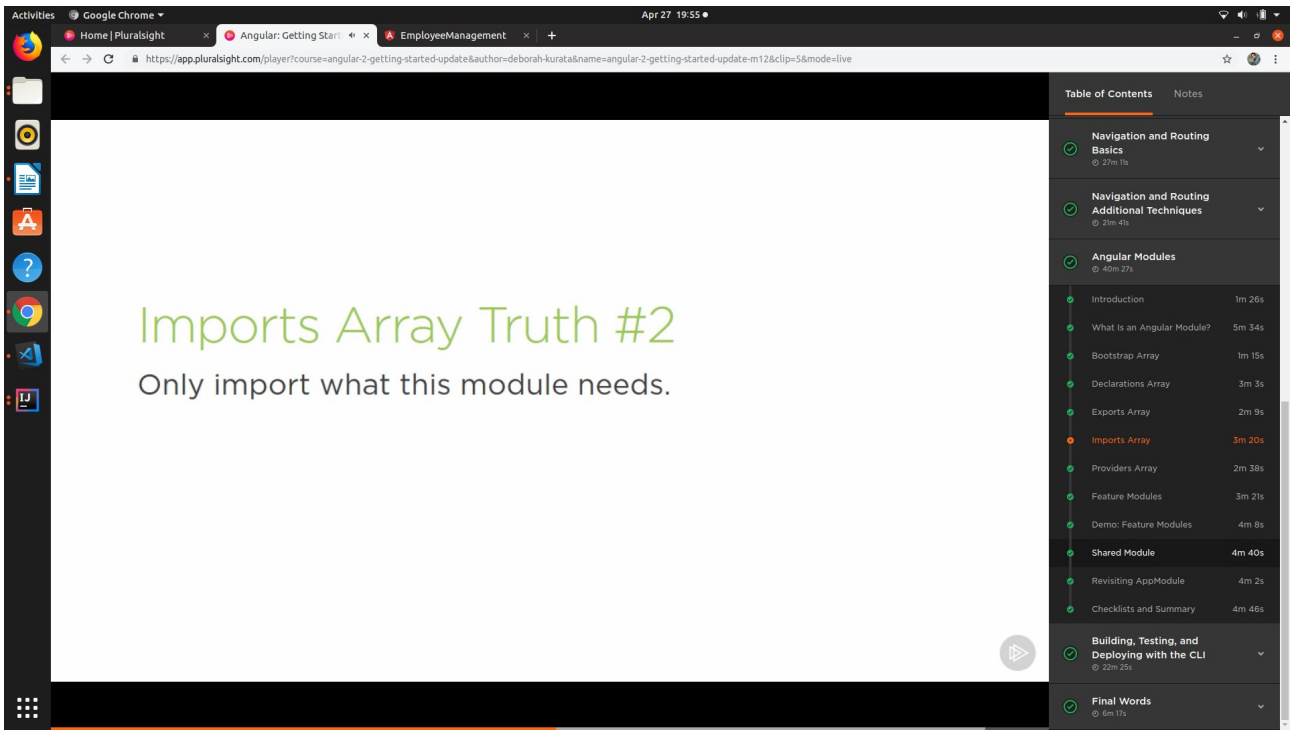


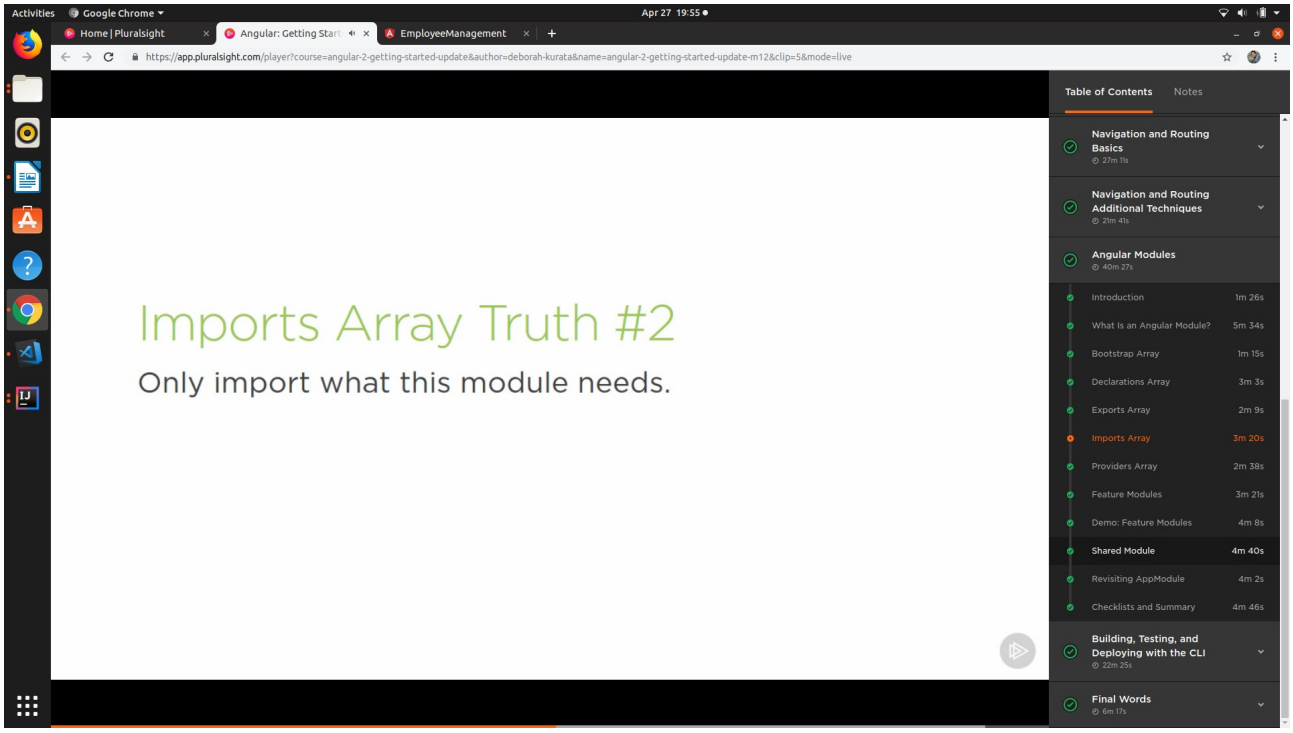
Imports Array:



Any component, directives and pipes which has export in it will get imported with the module



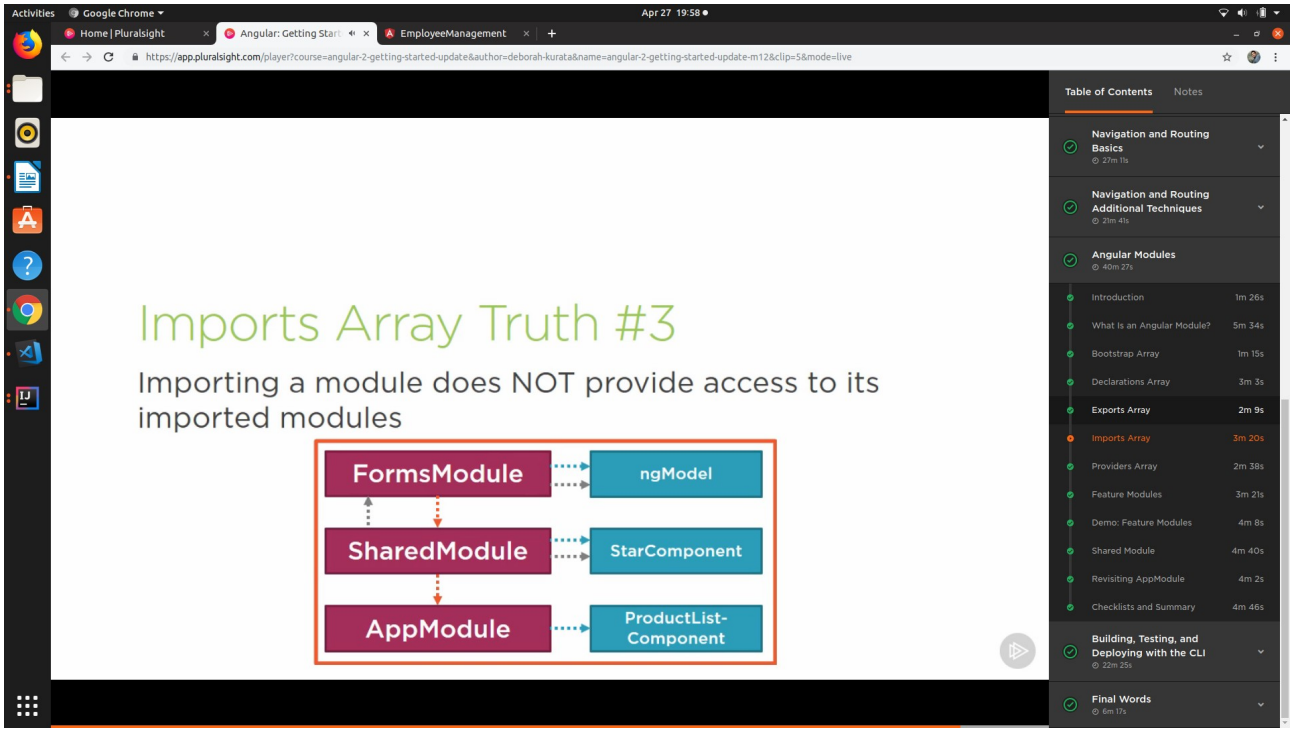




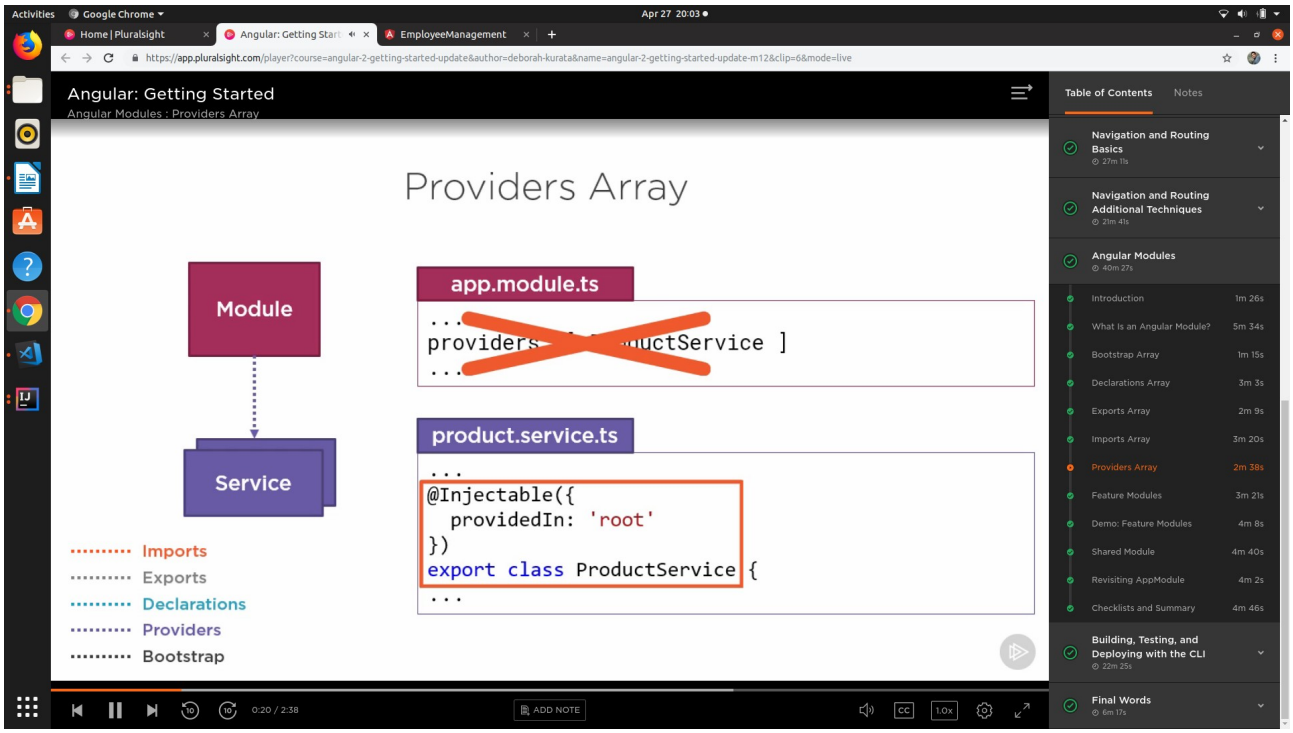
In the below rule FormModule is available to SharedModule but when SharedModule is imported to AppModule , FormModule will not be available to AppModule.

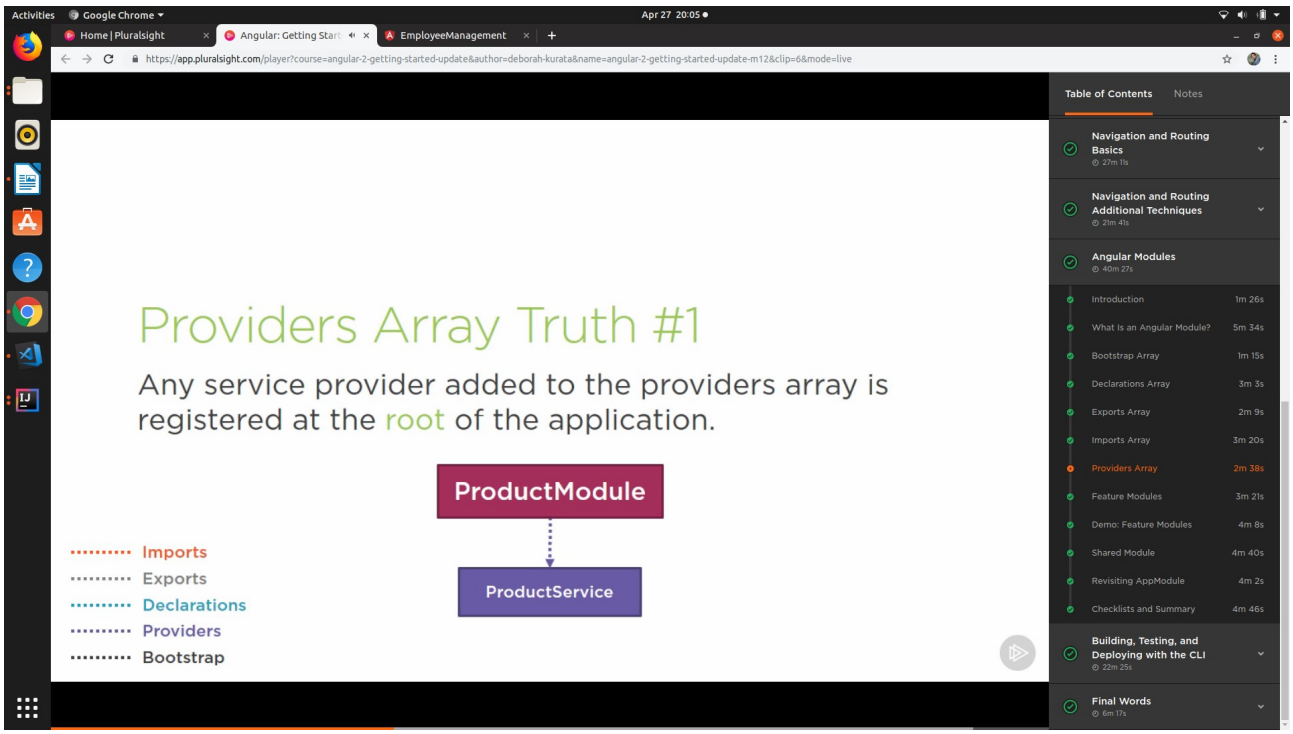
It means import are not inherited

If SharedModule reexports the FormModule then its features are available to AppModule

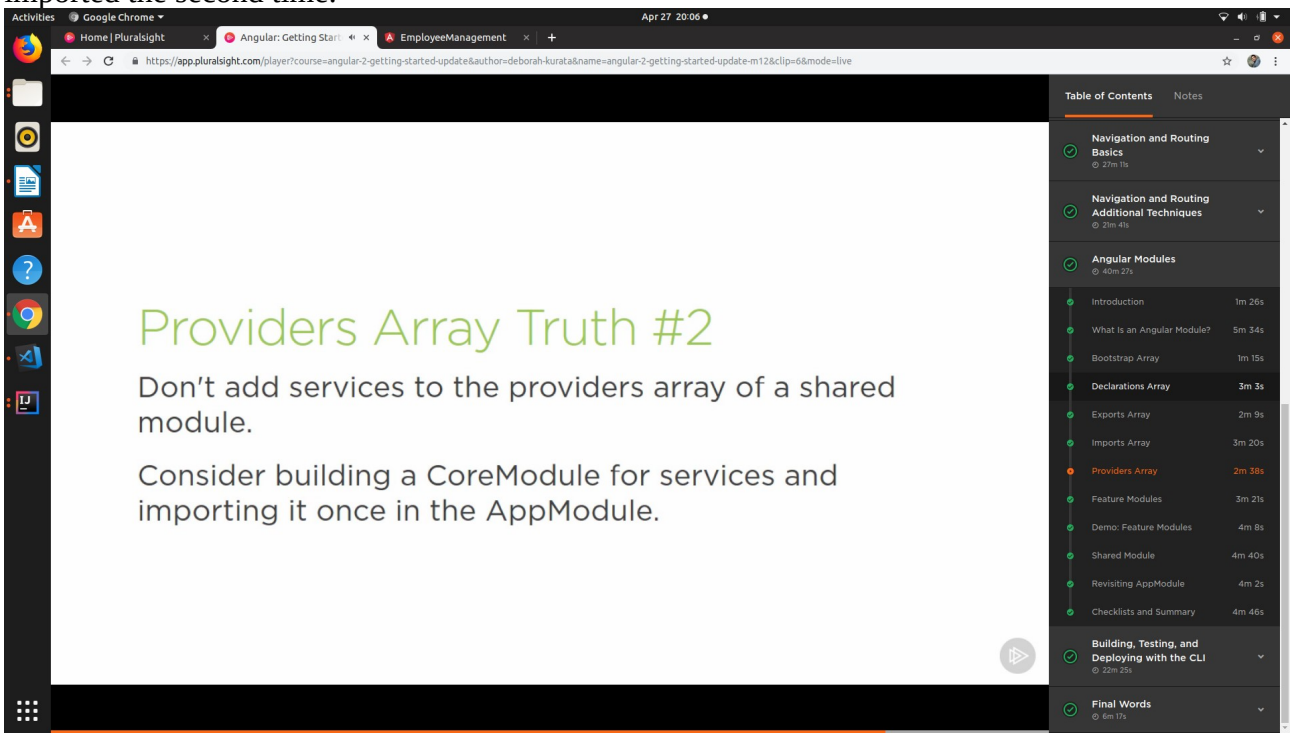


Providers Array: Starting Angular 6 the discarded approach in the below is not recommended





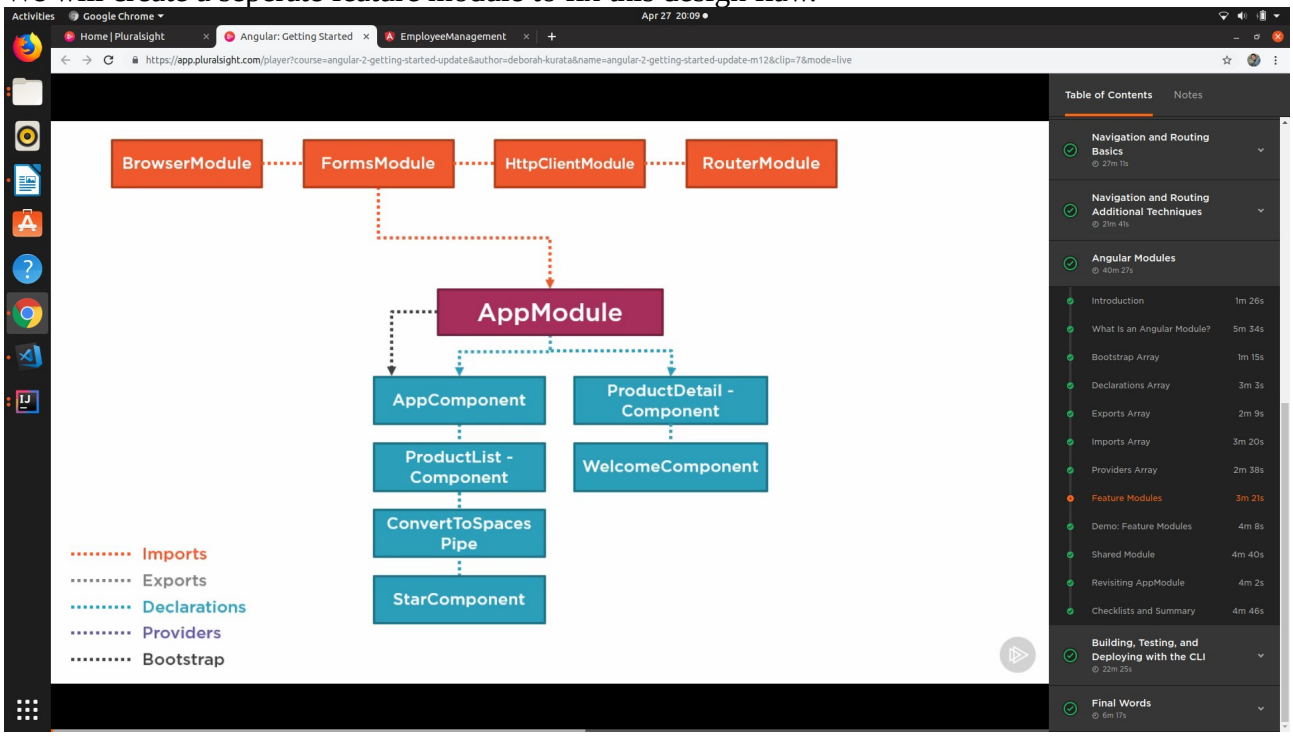
There should be only one instace of the service and it should be application wise singleton.
Build a CodeModule and we can even code in the code module constructor to ensure that it is never imported the second time.



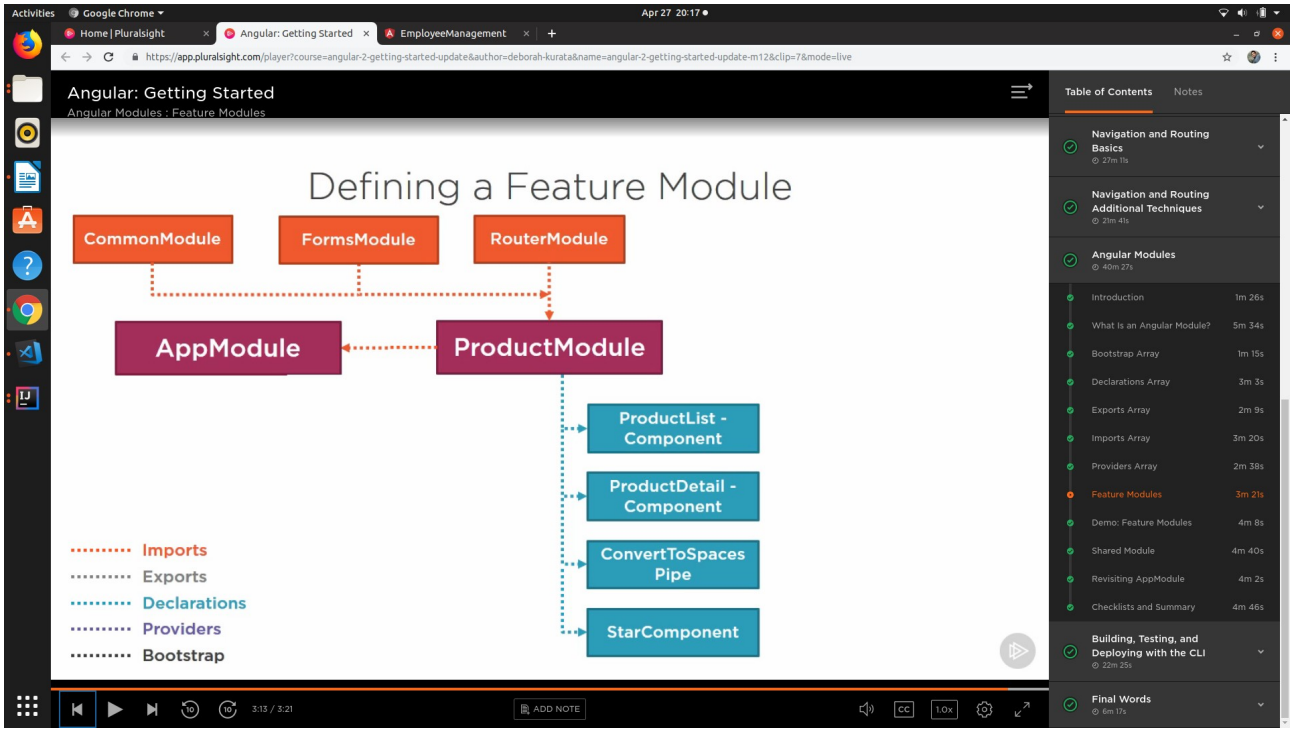
Feature Module:

The below image shows the design of our current application when the size grows it will be more difficult to manage the whole structure. There are some modules which are shared among different componenets.

We will create a seperate feature module to fix this design flaw.

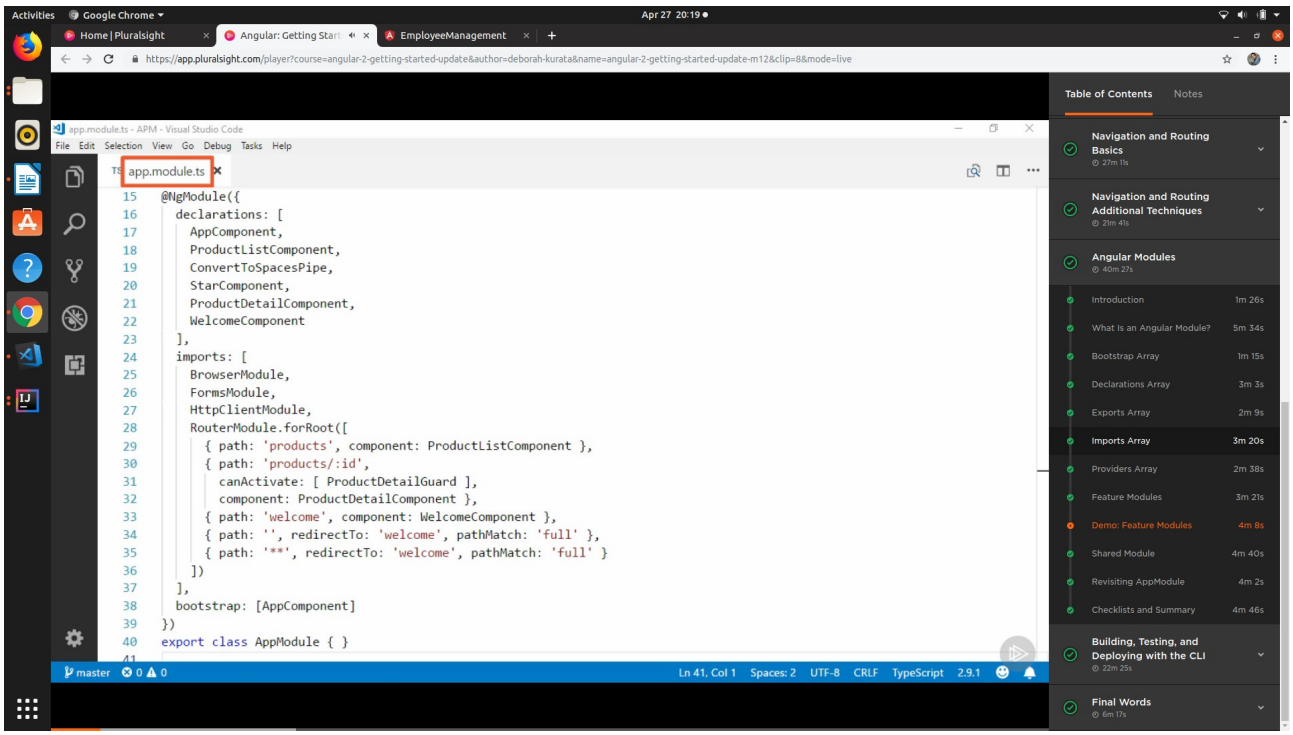


Creating a feature module
Note: We did not include the BrowserModule here as it should be included in the root application module.So we import the CommonModule which exposes the ngIf and ngFor directives.
BrowserModule exports and imports the CommonModule.
Since we have removed all the below details from the AppModule so we need to import the Product Module into the AppModule



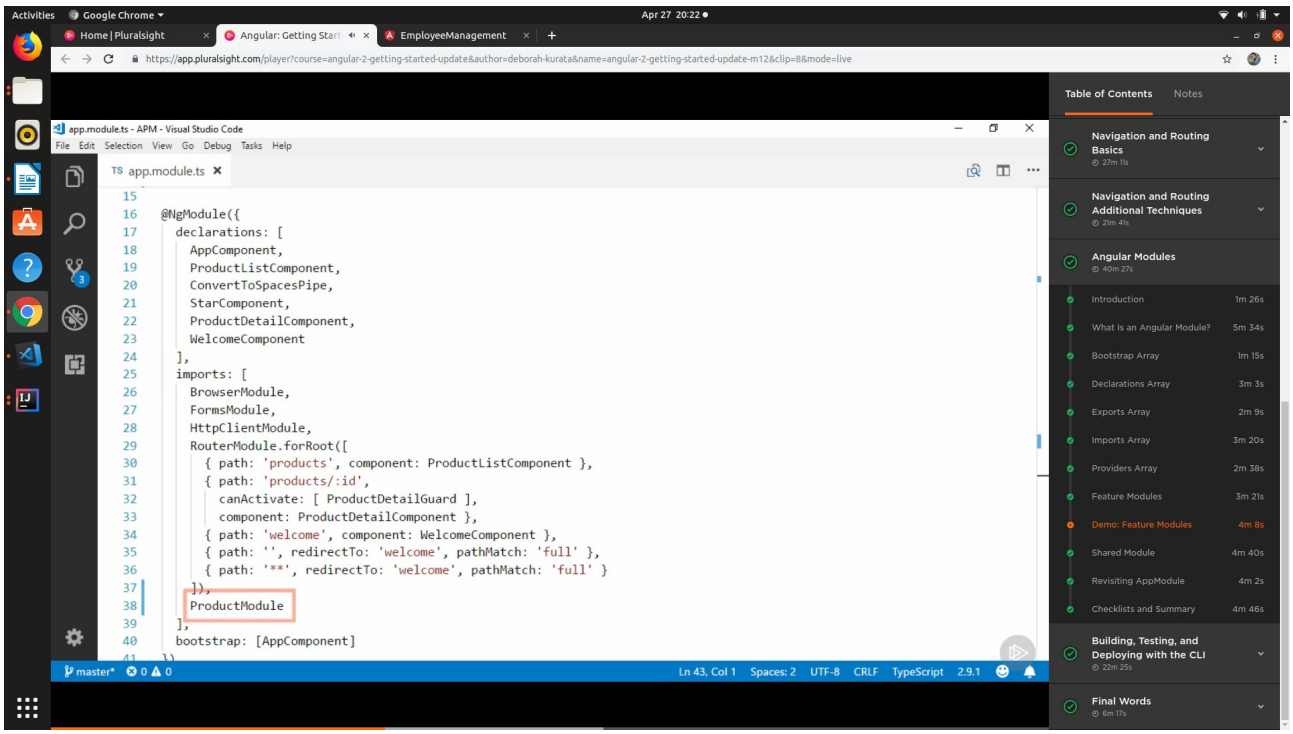
Example of the Above explanation:

Below image is of the exiting AppModule

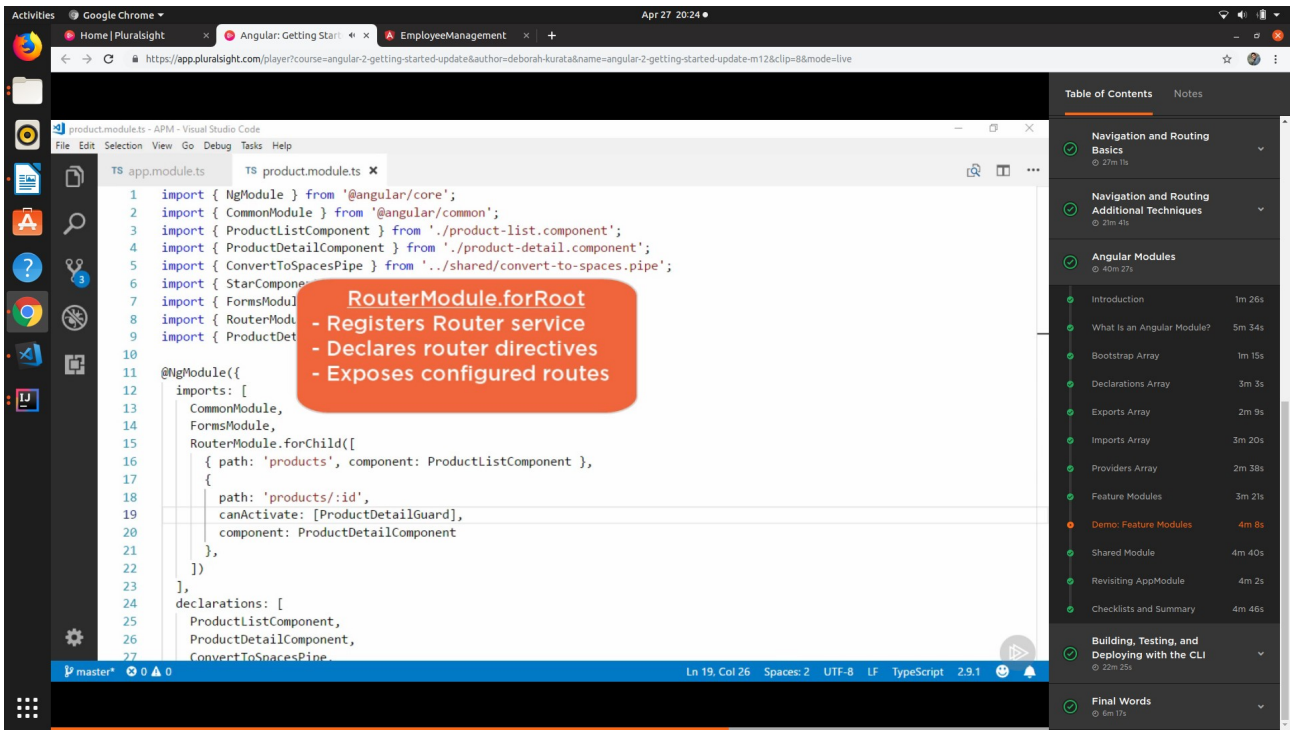


Now we will create a new product module using a CLI
ng g m products/product –flat -m app
m=for module
flat= so that it will not create a seperate folder for product module
-m=to import module
app = is the AppModule

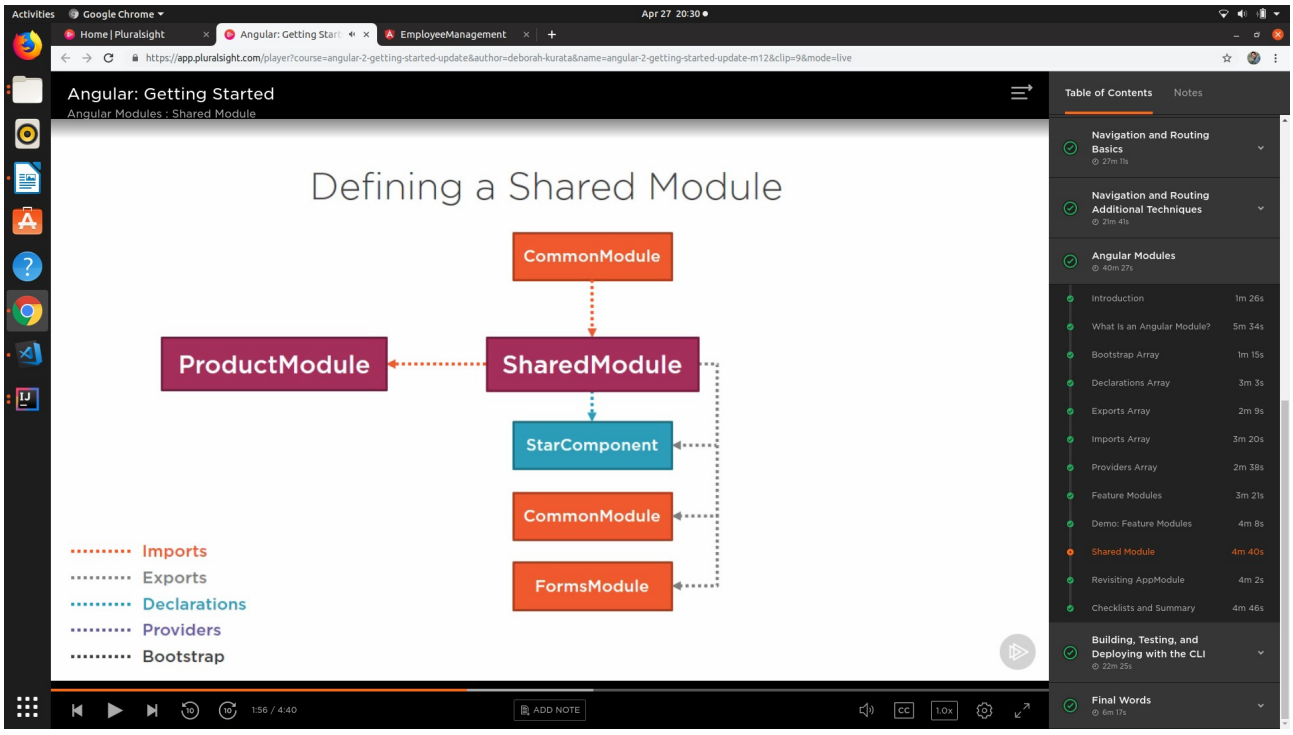
Once the command is executed it will add the product module into the app module as shown below



Now we remove the things from app module and add it to the product module



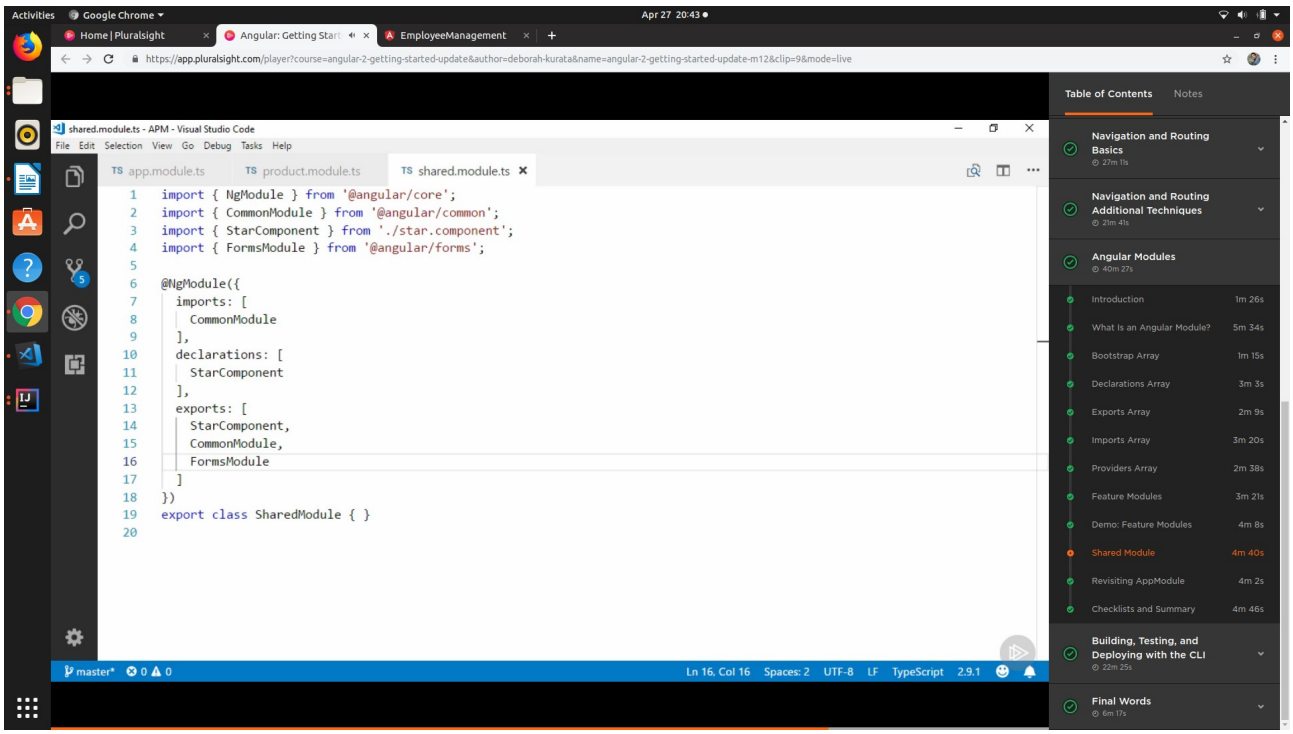
Shared Module:



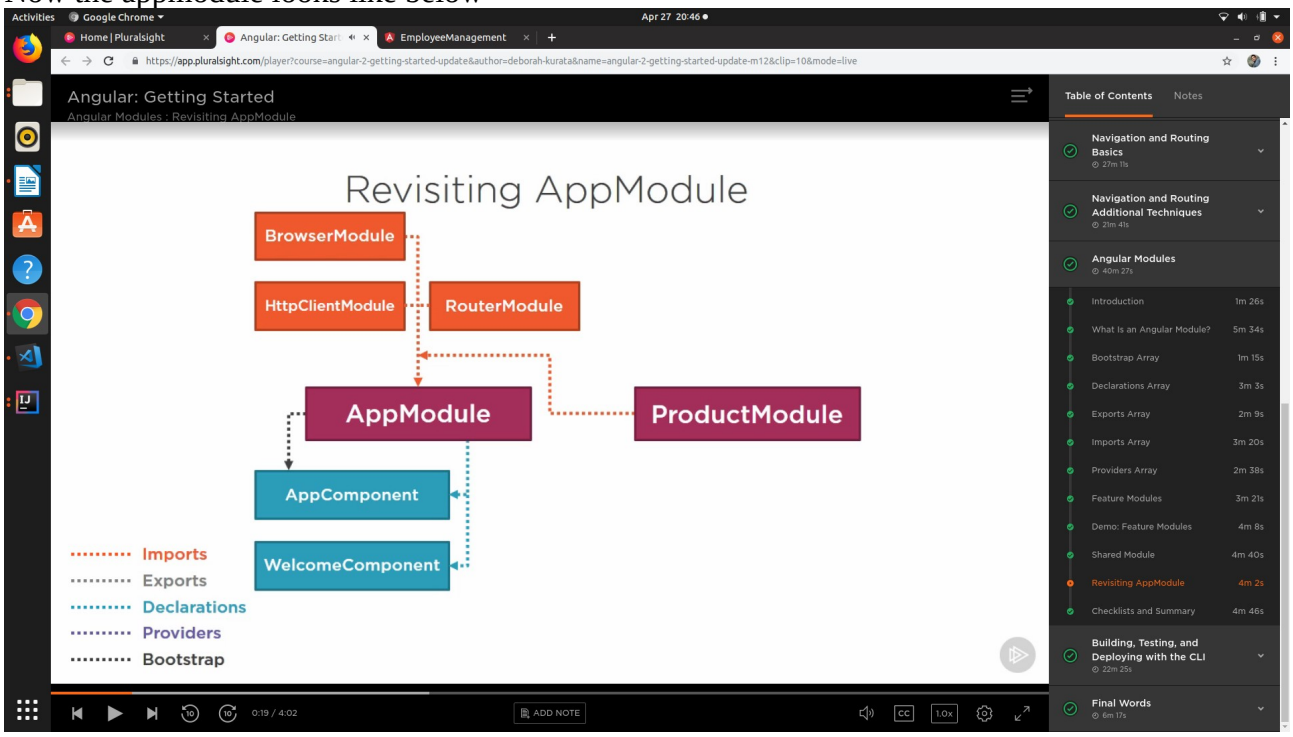
We will create a shared module using the cli
ng g m shared/shared - - flat -m products/product.module

The above command will add the enrty of shared module in the ProductModule

We can now put the modules that we need to share in the shared module

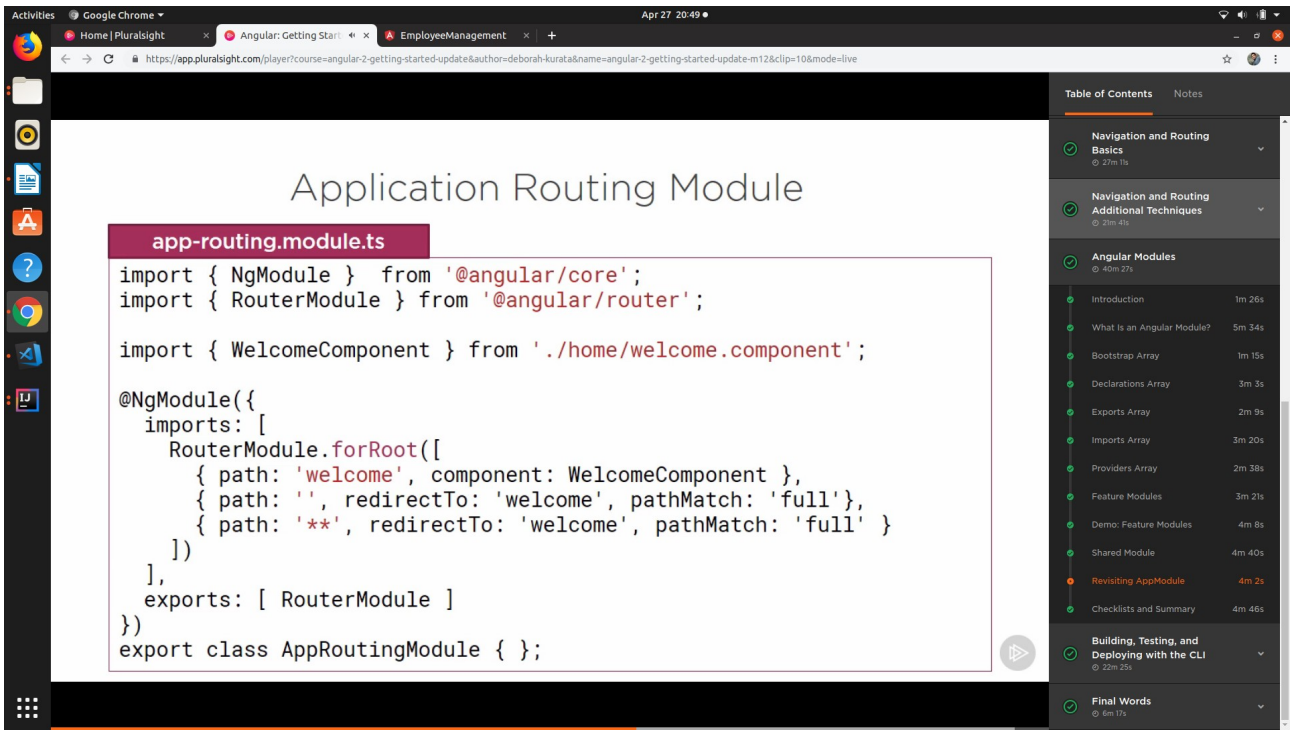


Now the appmodule looks like below

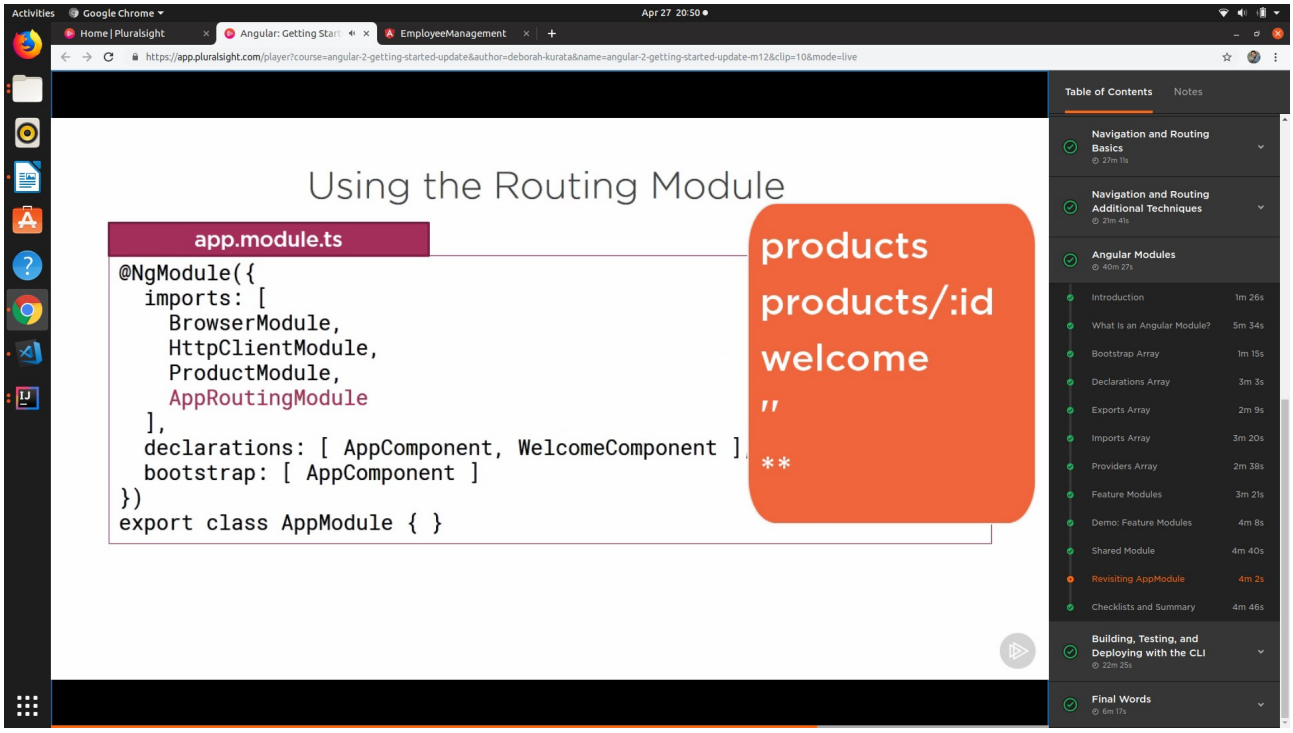


Creating a seperate RoutingModule:

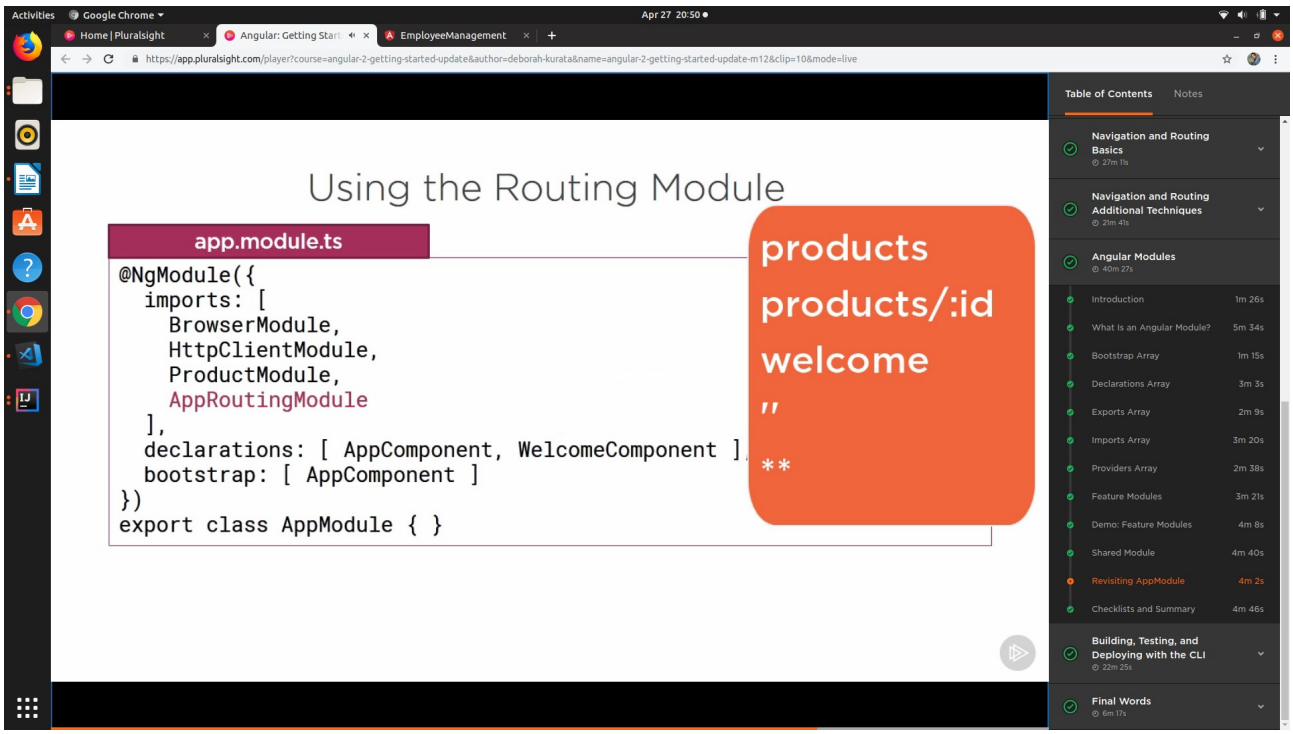
In the below image we have exported the RoutingModule so that it can be used in other application



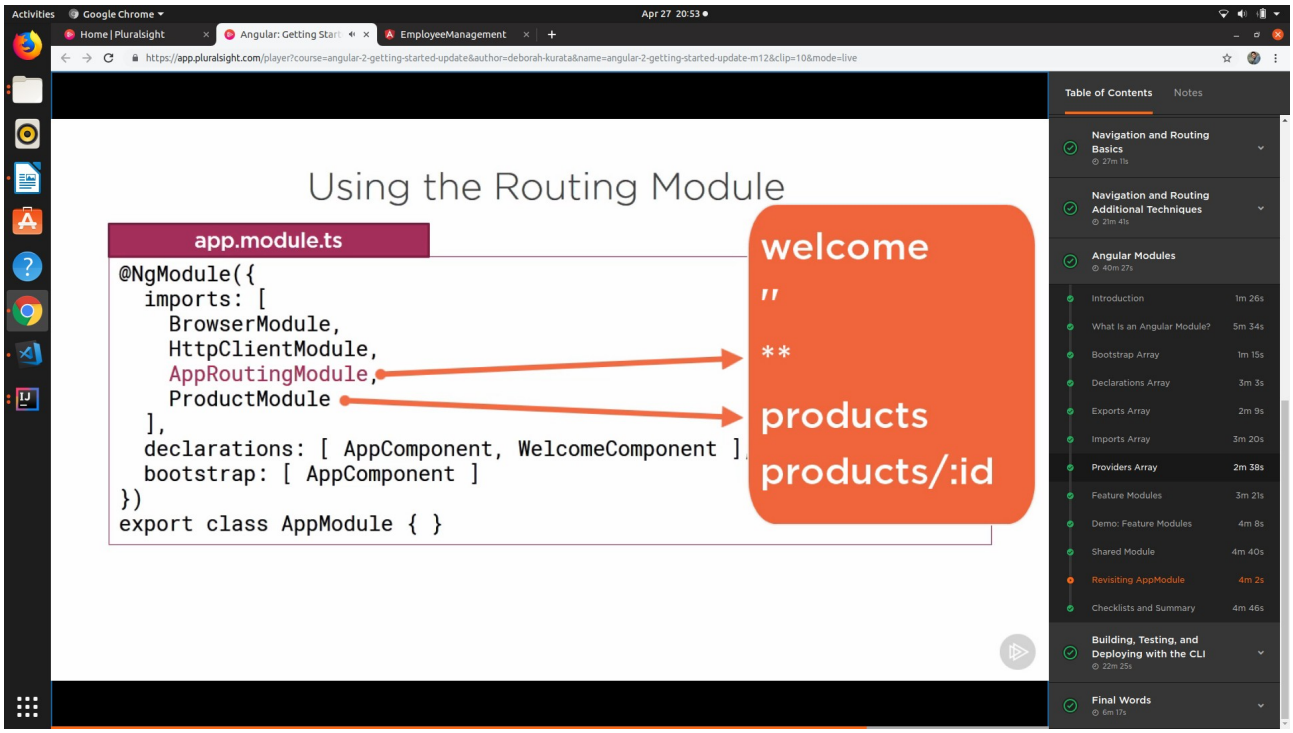
AppRoutingModule must be added below the ProductModule so that the hiererchy is maintained as shown in the right side.



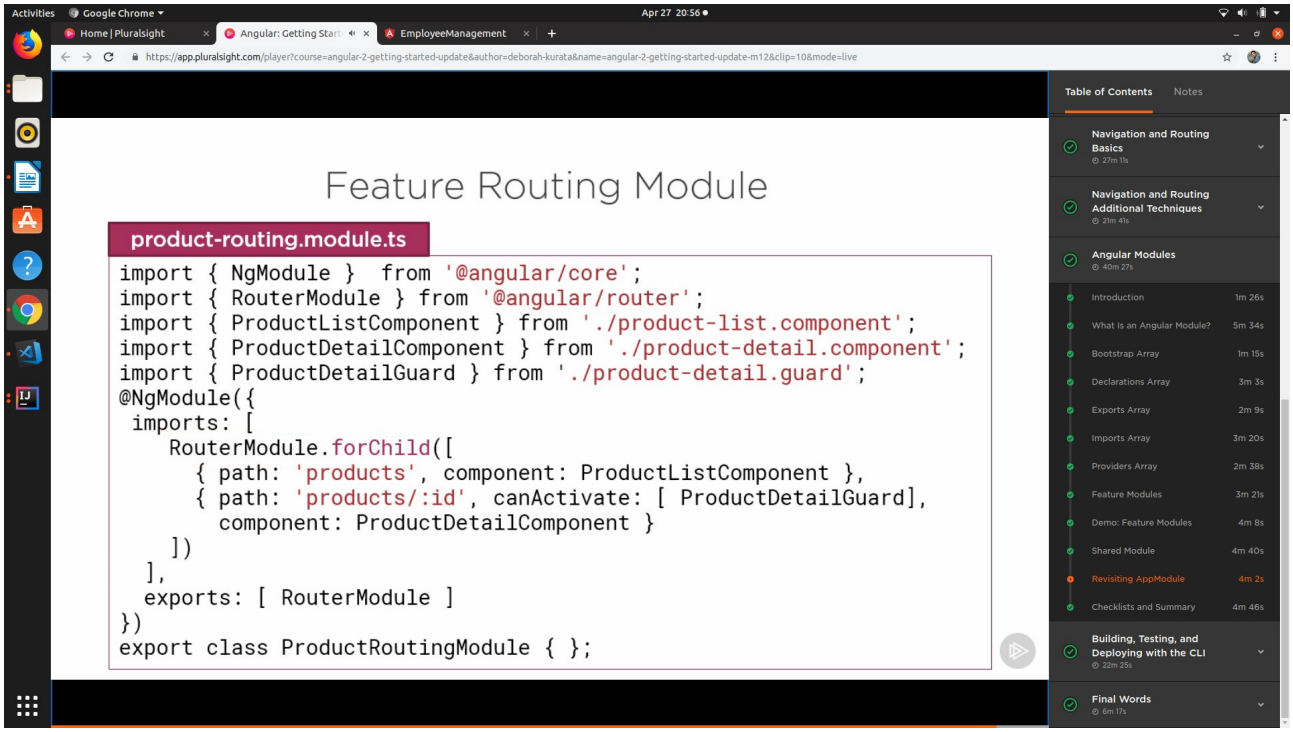
If we move RouterModule above the ProductModule the heirerchy will change as



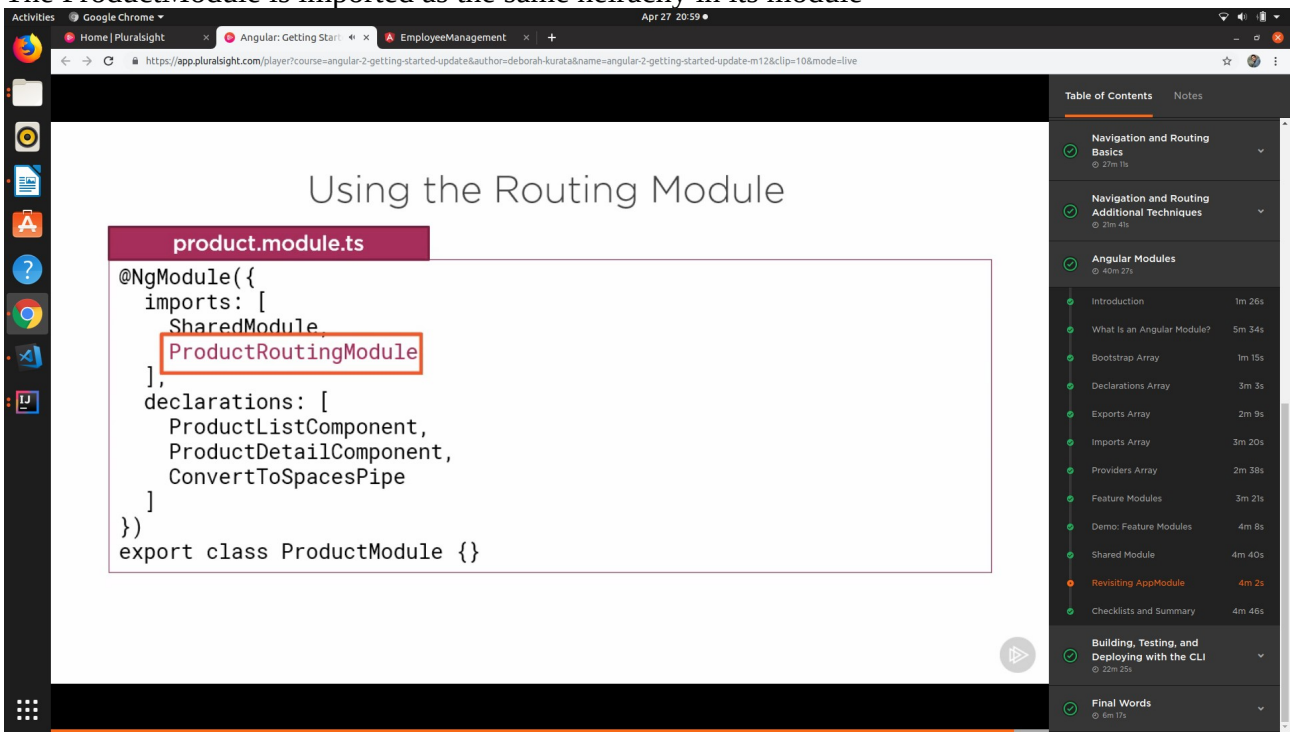
If we move RoutingModule above the ProductModule the Product will not be available when the routes are created as they will be set up before the ProductModule is even initialized.
Note: RoutingModule must always be the last to added in the imports of the AppModule.



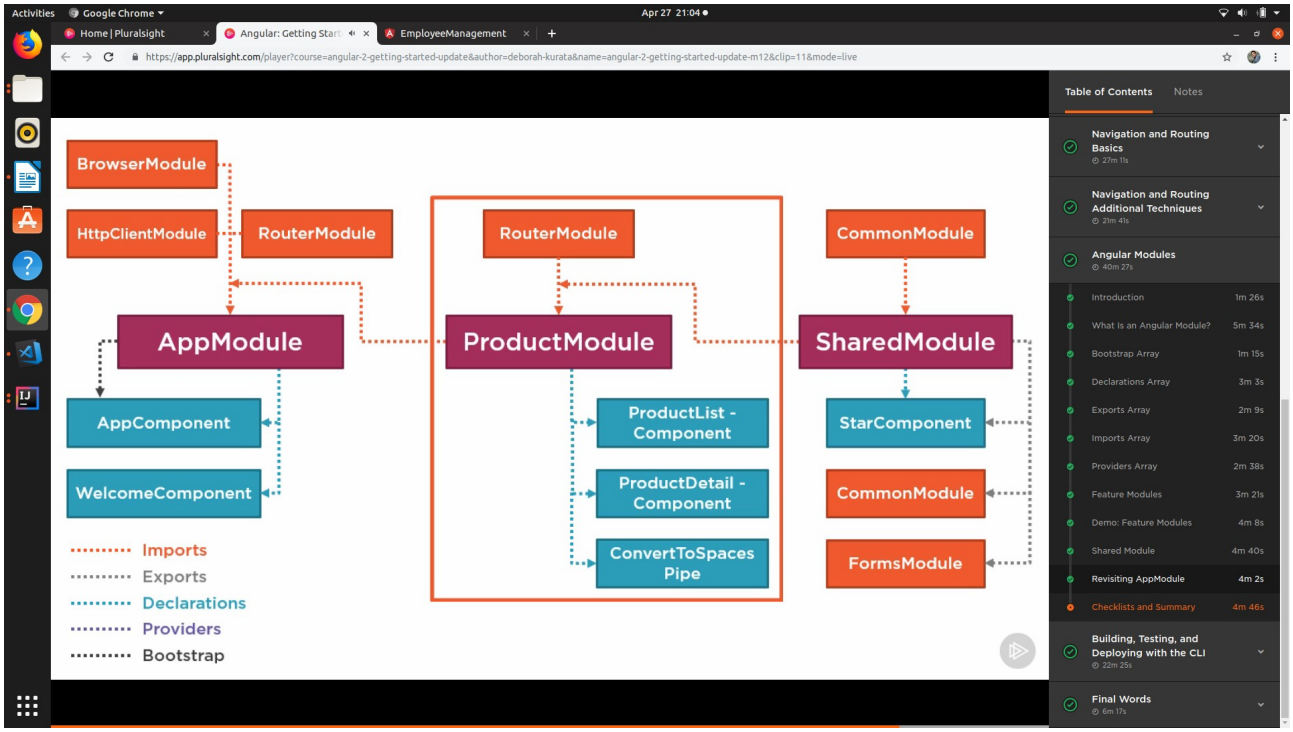
The route for the product modules must follow the same hierarchy and also it should use forChild not forRoot to ensure that we do not register the router service provider for the second time.



The ProductModule is imported as the same heirachy in its module

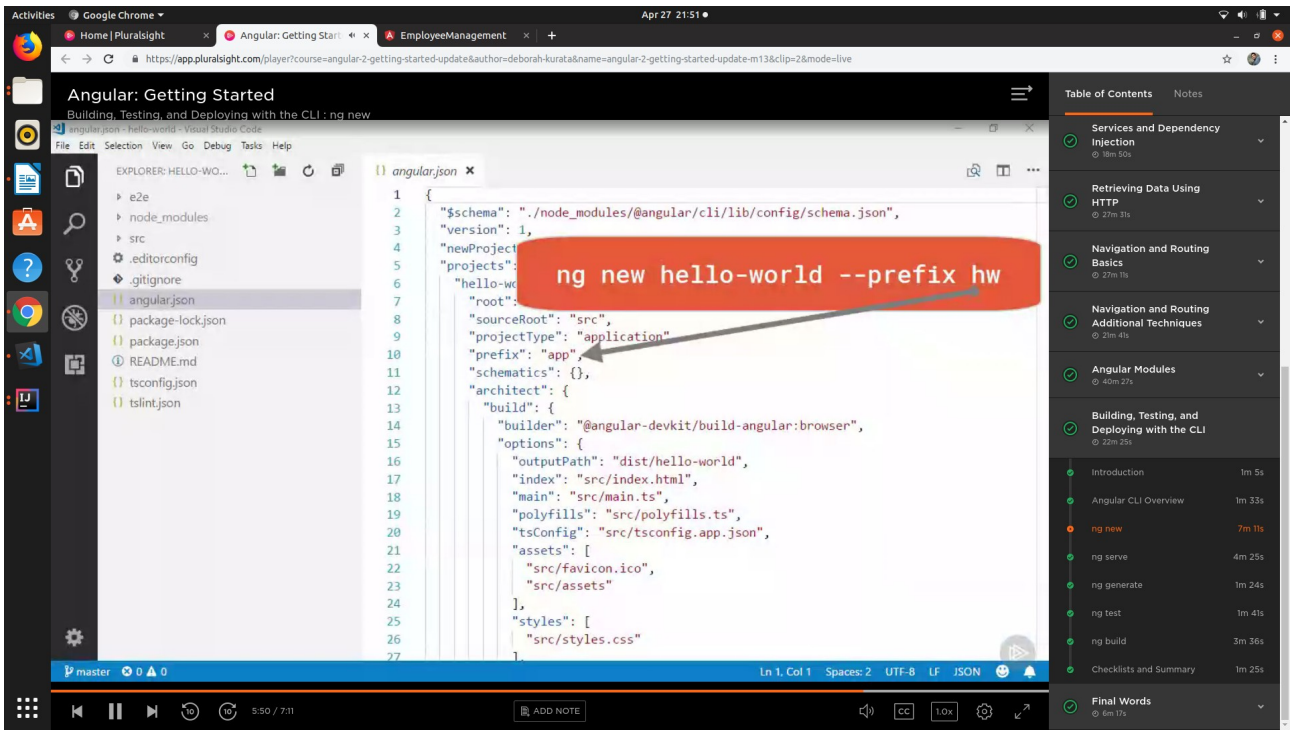


Final Arctitecture of the Angular Appilcation with new Feature and Shared Module:

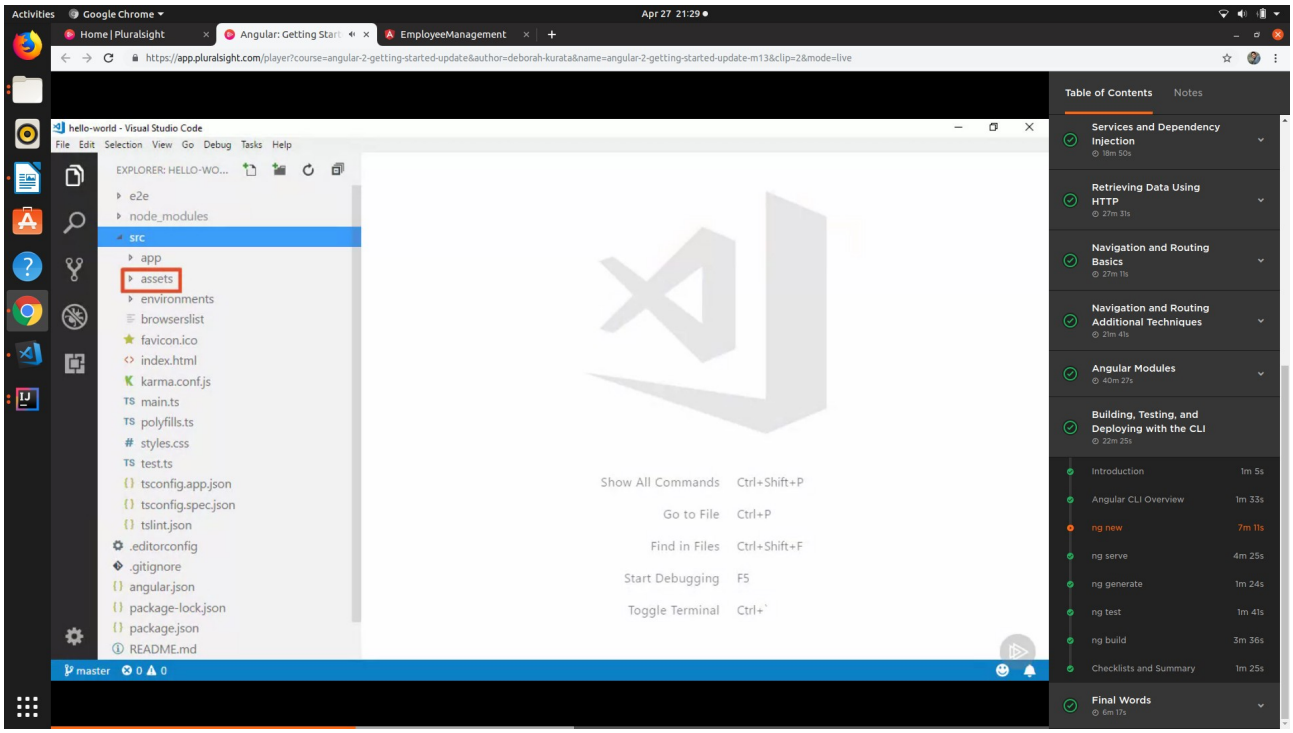


Building, Testing and Deploying using a CLI

In the below screenshot , the project was created using ng new projectname
1) e2e= This folder contains start of the end to end test of the application
2) node_modules=Contains all the libraries that are installed by CLI as defined in package.json file
3) src= It contains the source code of our application
    a) app = Here we add all our component, service and other files.
    b)assets= Its for any images or assets we want to include
    c)environments= It sets up our build environment, by default it generates 2 files one for development and one for production. Cli picks the appropriate file based on the flag we pick.
    d)favicon.ico= downloaded with angular cli can be replaced
    e)index.html= It is downloaded when application is created
    f)karma.config.js = Its a javascript test runner and will be used for unit testing
    g)main.ts= It is the file that bootstrap the application(this should not be modified)
    h)polyfills.ts= It aids in supporting both evergreen and classic browsers. It help in supporting the functionality writted for modern browser to work in older browser. Uncomment the code section in the file to get the support needed.
    i)style.css= Application wise style sheet
    j)test.ts= For Testing
    k)tsconfig.app.json=These files are for typescript configuration. This if for compiling our code files. It extends our tsconfig.json file.
    l)tsconfig.spec.ts=These files are for typescript configuration and this one for test specification
4)editorconfig=Editor configuration
5).gitignore=To ignore from checking into git
6)angular.json=This is a cli configuration file. This file is configure the way cli generates the codes and work with our files. This file hold the app profix name. We can change it as per our requirement.
It can also be changes using the cli :ng new hello-world --prefix hw



7)package-lock.json
8)package.json=It has all the files required by the project , dependecies etc.
9)README.md= It has no importance other than some info
10)tsconfig.json=
11)tslint.json=Typescript linter which checks the code on the set of style rules and notify if any rules are not followed. It can be changed based on the team's styling preferences.



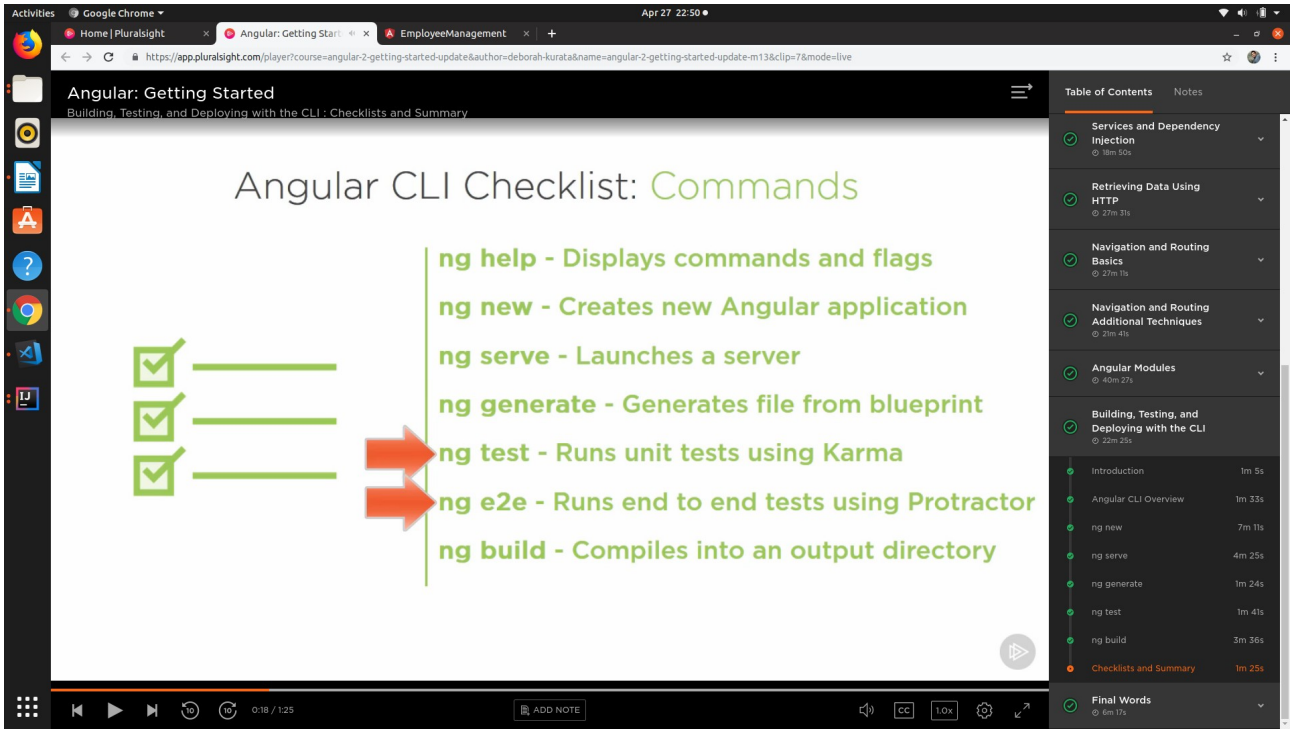
Some useful Commands:
1) ng new --help
a) –skip-install=To generate code without installing node packages which can be installed later using npm install.

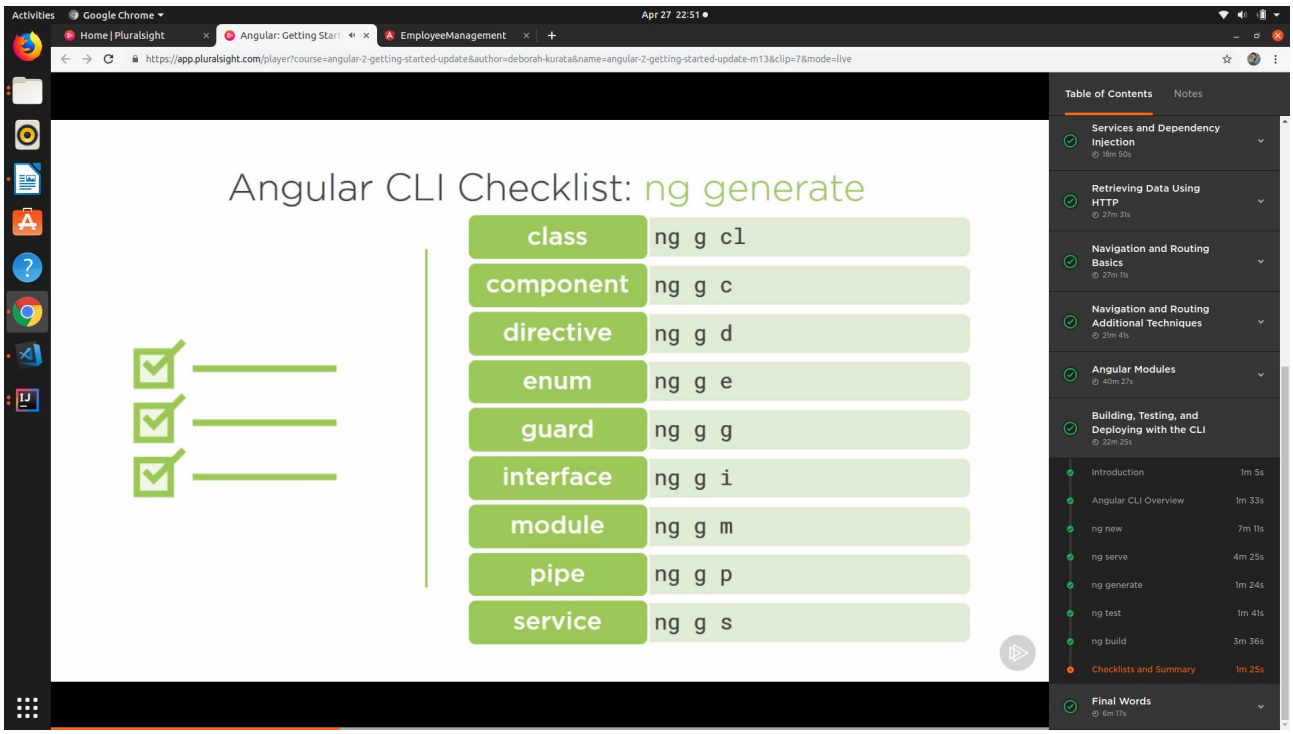
2) ng serve command: If we use ng serve –help(This shares all the commands used by serve)
--port=to provide alternate port
-o = open in browser

3) ng g/generate command: use ng g –help(It will show what can be created using this command)

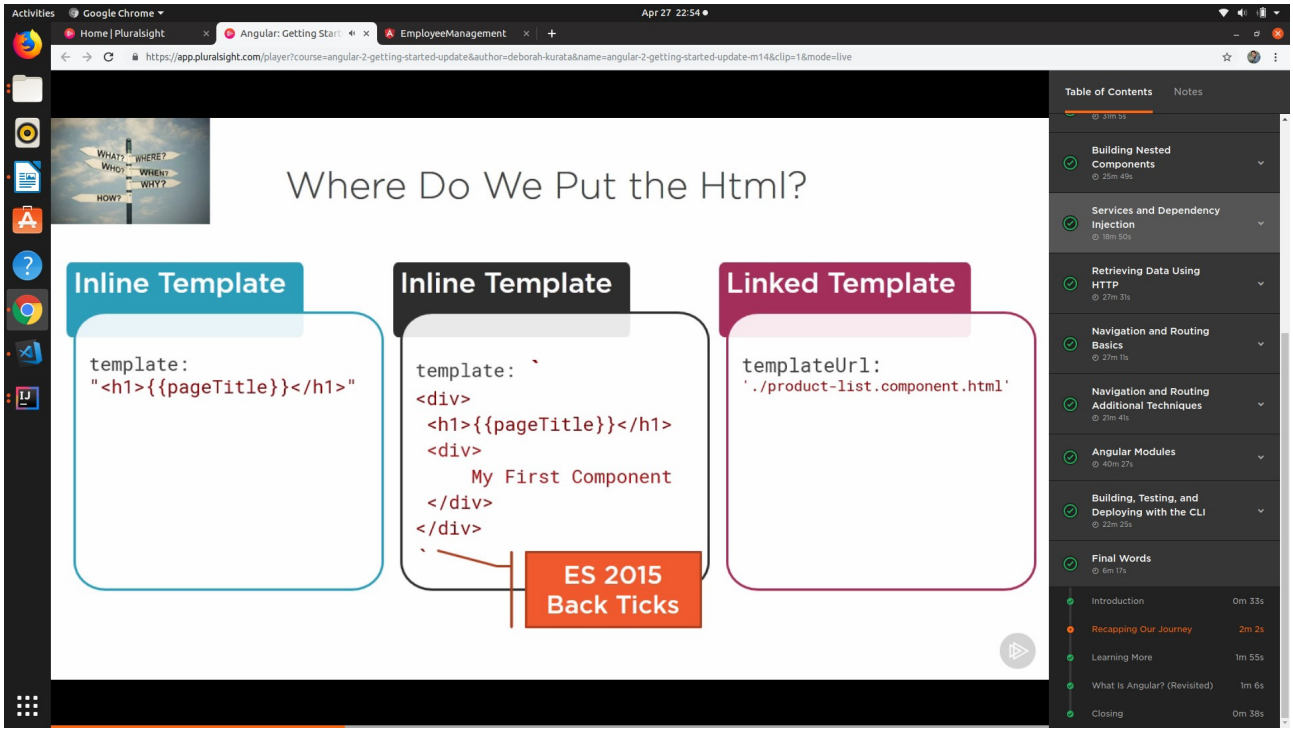
4) ng test command: Test can be run using the two commands . The test runner run in watch mode so that if we change the code in between it will re run
1) ng test
2)ng e2e = end to end testing using protractor

5) ng build: It generated the dist folder in our application
ng build - - prod: It generates the minified version of the files that needs to be deployed on server.




Angular CLI Checklist: ng generate
class ng g cl
component ng g c
directive ng g d
enum ng g e
guard ng g g
interface ng g i
module ng g m
pipe ng g p
service ng g s
Table of Contents
Notes
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Introduction 1m 5s
Angular CLI Overview 1m 33s
ng new 7m 11s
ng serve 4m 25s
ng generate 1m 24s
ng test 1m 41s
ng build 3m 36s
Checklists and Summary 1m 25s
Final Words

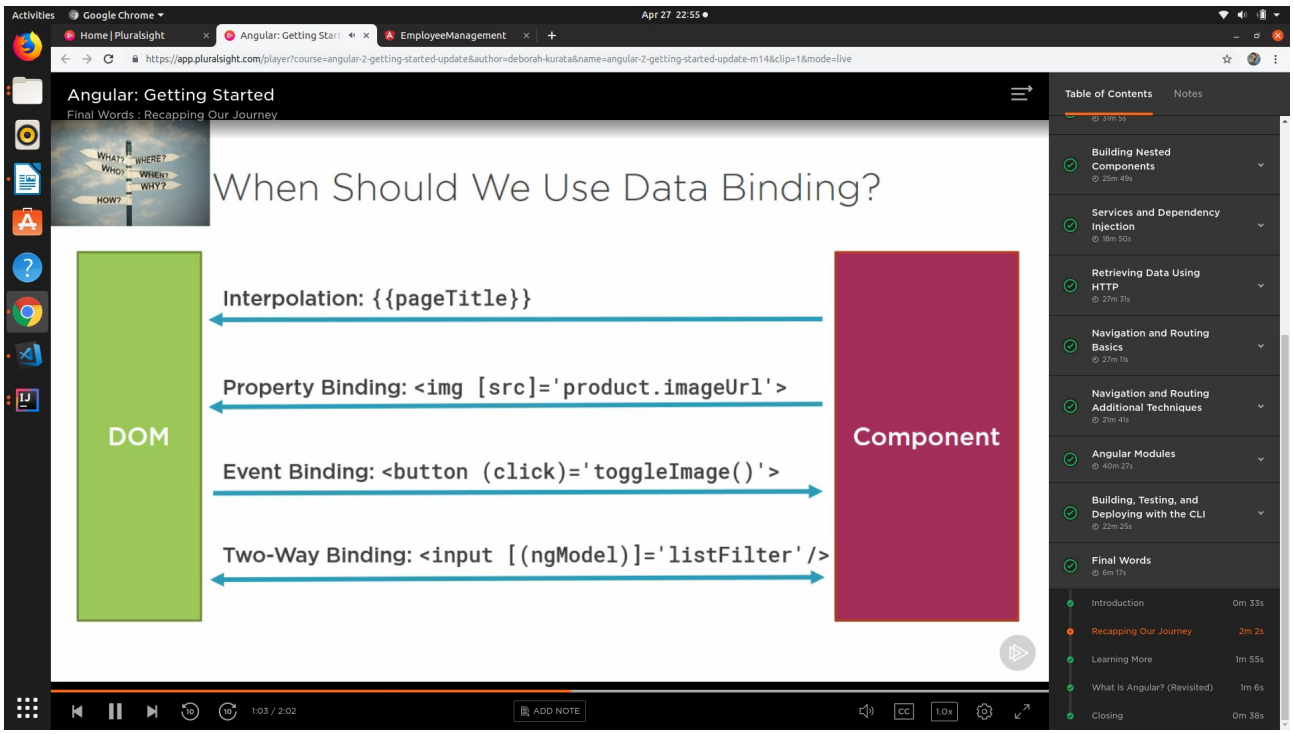
RECAP:

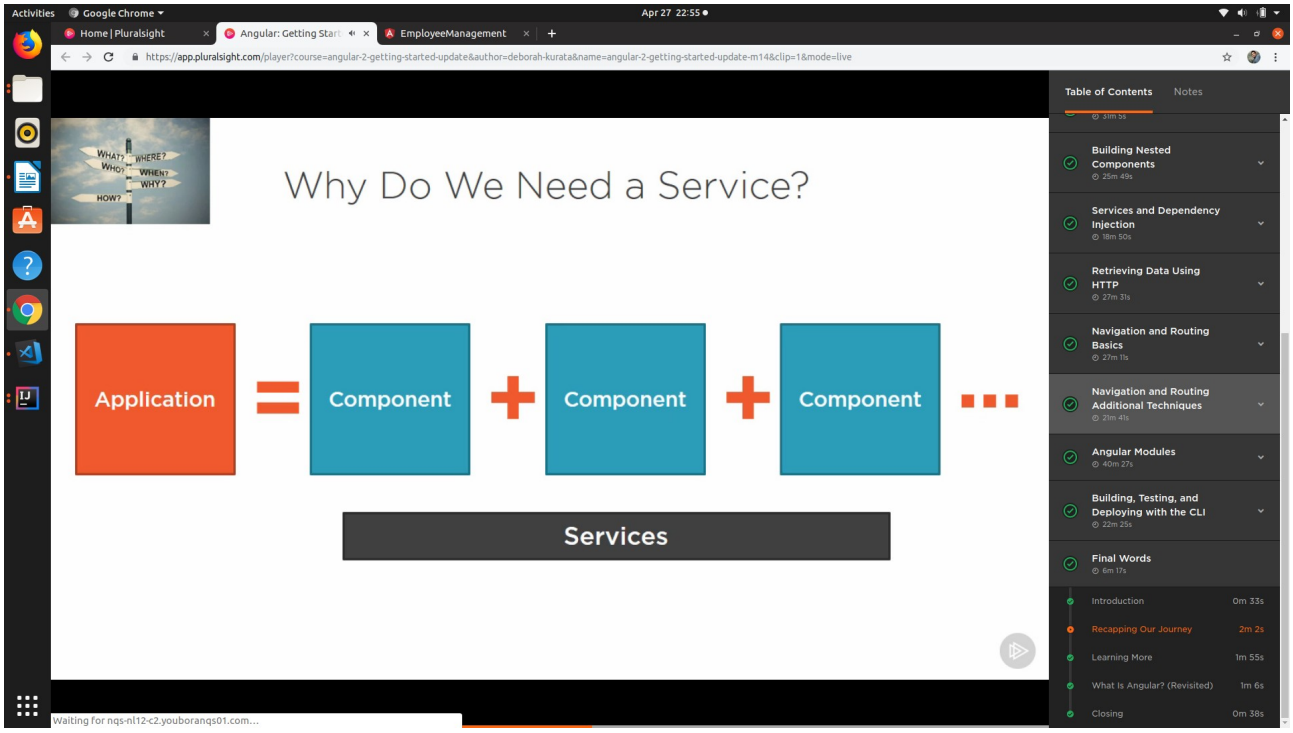

Where Do We Put the Html?
Inline Template
template:
"<h1>{{pageTitle}}</h1>"
Inline Template
template: `

<h1>{{pageTitle}}</h1>

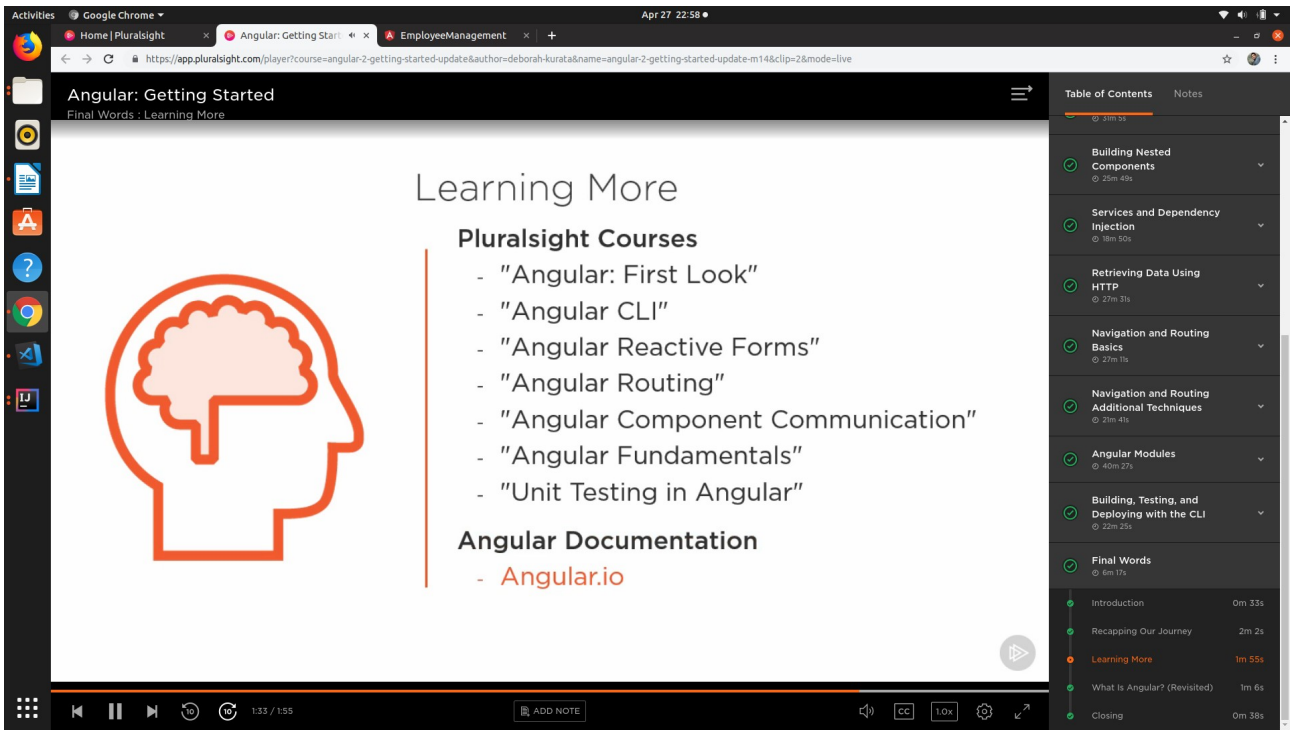
My First Component

`
ES 2015
Back Ticks
Linked Template
templateUrl:
'./product-list.component.html'
Table of Contents
Notes
Building Nested Components
Services and Dependency Injection
Retrieving Data Using HTTP
Navigation and Routing Basics
Navigation and Routing Additional Techniques
Angular Modules
Building, Testing, and Deploying with the CLI
Final Words
Introduction 0m 33s
Recapping Our Journey 2m 2s
Learning More 1m 55s
What Is Angular? (Revisited) 1m 6s
Closing 0m 38s

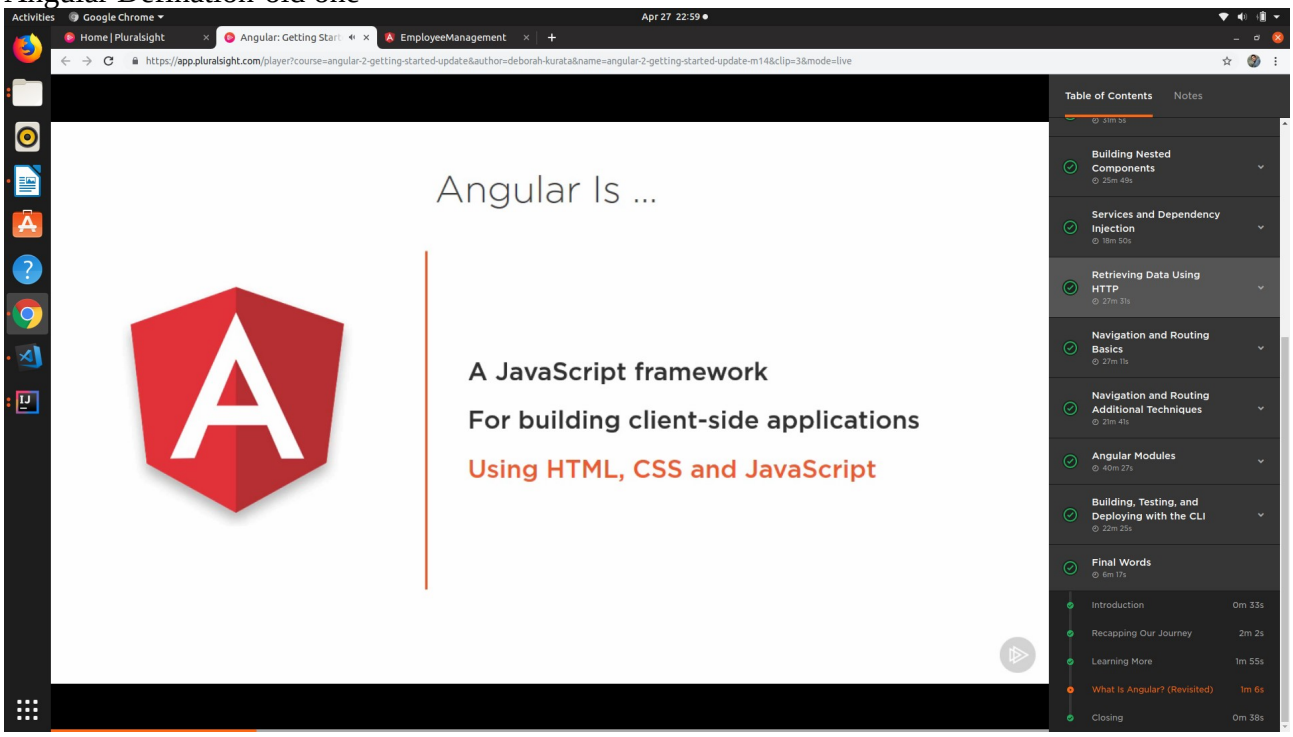


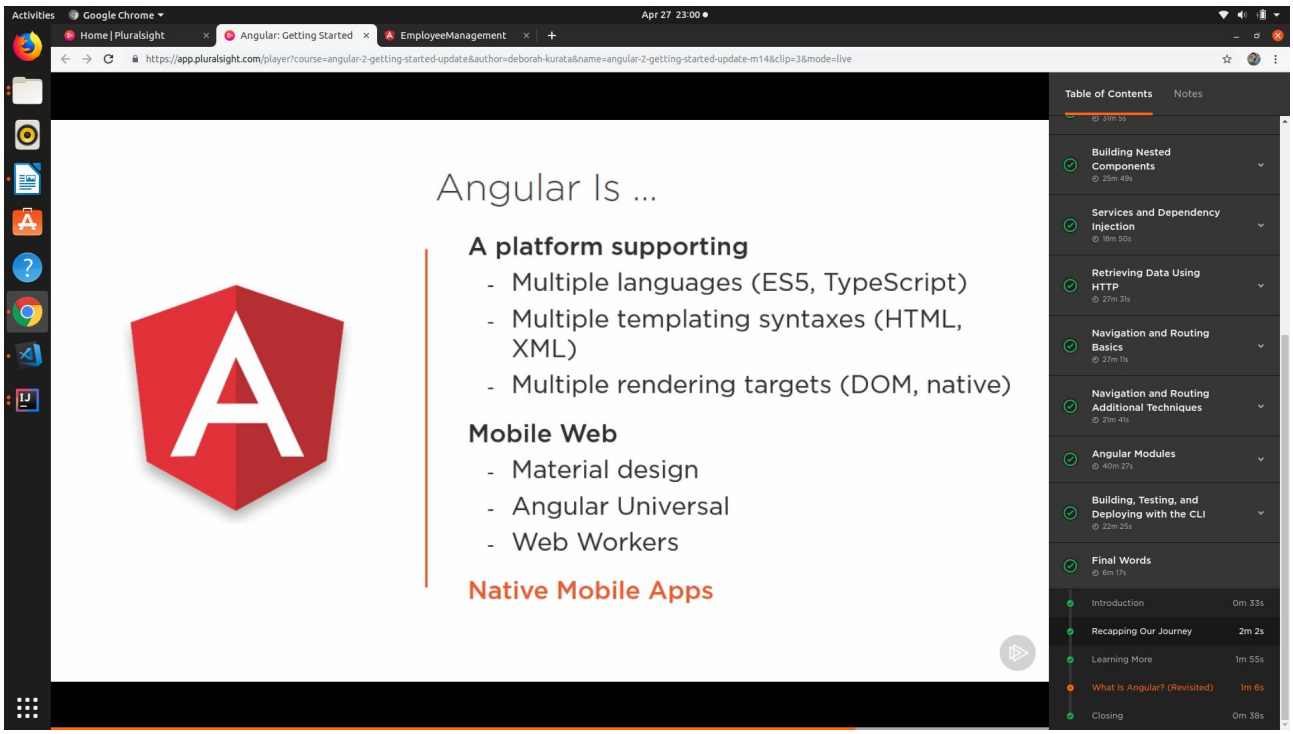


More Courses:



Angular Defination-old one




Activities
Google Chrome
Apr 27 23:00
Home | Pluralsight
Angular: Getting Started
EmployeeManagement
https://app.pluralsight.com/player?course=angular-2-getting-started-update&author=deborah-kurata&name=angular-2-getting-started-update-m14&clip=3&mode=live
Angular Is ...
A platform supporting
- Multiple languages (ES5, TypeScript)
- Multiple templating syntaxes (HTML, XML)
- Multiple rendering targets (DOM, native)
Mobile Web
- Material design
- Angular Universal
- Web Workers
Native Mobile Apps
Table of Contents
Notes
Building Nested Components
25m 49s
Services and Dependency Injection
18m 50s
Retrieving Data Using HTTP
27m 31s
Navigation and Routing Basics
27m 11s
Navigation and Routing Additional Techniques
21m 41s
Angular Modules
40m 27s
Building, Testing, and Deploying with the CLI
22m 25s
Final Words
6m 17s
Introduction 0m 33s
Recapping Our Journey 2m 2s
Learning More 1m 55s
What Is Angular? (Revisited) 1m 6s
Closing 0m 38s