<https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html>

1. System requirements for angular6?

For this we need to install

Node.js,angular cli,visualstudiocode

To install this first we check the version of those by using command proment

Type node –v to check the version of node.js. if we don’t have anything thing related to node js in our system download latestversion of nodejs form <https://nodejs.org/en/>

To install angularcli

We can install that by using below command

Npm install –g @angular/cli@latest

By using above command we can install angularcli latest version at system globally

By using this link we can install visualstudio code

[https://code.visualstudio.com/](https://code.visualstudio.com/%20%20)

1. How to create an angular6 application?

Open command prompt as an administrator

Navigate to the folder path where we want to create a application folder in our system.by giving like this

Cd C:\WORKBASE\PRACTISE\SanjanaPractise\AngularJs

Ng new applicationname

Here in my case its “Angular6Practise”

To run this application we use ng serve

For suppose if we are in the other directory we can change it to application path and give this “ng serve”

We can give this ng serve in visual studio code command prompt to open this

View🡪 debug console🡪terminal

1. Open visualstudiocode “goto view🡪integraatedterminal
2. In that “cd C:\WORKBASE\PRACTISE\SanjanaPractise\Angular-6”
3. Ng new Angularsanjana
4. Cd C:\WORKBASE\PRACTISE\SanjanaPractise\Angular-6\Angularsanjana
5. Ng serve
6. For suppose we assign localhost://4200 to any project then we have to give the command like this ng serve –port 4201
7. In the explorer window click openfolder 🡪 goto where u saved the project in the folder and open that so that we can see the code on left side
8. To add bootstrap to the project
9. npm install --save bootstrap@3  => The @3  is important!
10. Additionally, when using a project created with Angular CLI 6+ (check via ng -v ), you'll have an angular.json  file instead of an .angular-cli.json file. In that file, you still need to add Bootstrap to the styles[]  array as shown in the next video, but the path should be node\_modules/bootstrap/dist/css/bootstrap.min.css , **NOT**../node\_modules/bootstrap/dist/css/bootstrap.min.css . The leading ../  must not be included.
11. If want to use angularmaterial instead of bootstrap we have to follow below stepas instead of bootstrap step

So, let’s get started by configuring our app for Material design. Execute the following commands to install Angular Material.

npm install --save @angular/material @angular/cdk

npm install --save @angular/animations

As you can see, I have installed 3 main dependencies - material, its cdk, and its animations .

**Note**We have installed animations because some of the Angular Material components are based on animation, like**-** swipe , zoom-in , zoom out etc .

#### NPM

npm install --save hammerjs

So now, we have completely configured our app.

Over night Oatmeal

1. How to create folders in Angular6?

We can create the components manually and

How to create manually

Under app folder we create one folder. Here I create folder with name header. Under this header folder I added one component file with name “header.component.ts”. As we know component is a typescript file.

In this file first we do

1.

export class HeaderComponent {

}

2.

@Component({

selector: 'app-header',

templateUrl: './header.component.html'

})

3.

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

after this we have to create file of “header.component.html” under header folder.

Now we have to add component to app.module.ts like below

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

After that we have to import the component in app.module.ts like below

import { HeaderComponent } from './Header/header.component';

if we add any module under header folder we have to add those modules in app.module.ts

imports: [

BrowserModule,

FormsModule,

HttpModule

],

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

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

In visual studio code command prompt

ng generate component componentname –spec false

-----or-----

Ng g c componentname

If we do this we get files

  create src/app/login/ componentname.component.html (24 bytes)

  create src/app/login/ componentname.component.spec.ts (621 bytes)

  create src/app/login/ componentname.component.ts (265 bytes)

  create src/app/login/ componentname.component.css (0 bytes)

  update src/app/app.module.ts (394 bytes)

to create subfolders under folder

ng g c recipes/recipies-list –spec false

**The Concepts of Angular Components  -**

Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to bring functionality to your applications.

Technically components are basically [TypeScript](https://www.code-sample.com/2017/06/typescript-interview-questions-and.html) classes that interact with the HTML files of the components, which get displayed on the browsers.

The component is the core functionality of Angular applications but you need to know to pass the data into the components to configure them.

Angular applications must have a root component that contains all other components.

[](https://3.bp.blogspot.com/-SG-9KGnLYE4/WvUtnVQkn-I/AAAAAAAASiE/cwlLIXKFM1siU-RZvdPxnWBPgrponMnHgCLcBGAs/s1600/Core-Concepts-Of-Angular-6-5.png)

Components are created using **@Component** decorator that is part of **@angular/core module**.

You can create your own project using Angular CLI, this command allows you to quickly create an Angular application like - generate components, services, pipes, directive, classes, and modules, and so on as per your requirements.

**Understanding directives:**

Attribute and structural directives

Diff between ngrepeat and ngfor

ngfor and \*ngfor

Attribute directives:----

ngclass and ngstyle

Both the NgStyle and NgClass directives can be used to conditionally set the look and feel of our application.

NgStyle gives you fine grained control on individual properties. But if you want to make changes to multiple properties at once, creating a class which bundles those properties and adding the class with NgClass makes more sense.

To create basic Attribute directive:

This is just like normal css class . w euse this to apply styles and some properties to particular element.

For this we have to create some folder to add directive

Here I gave the name to directive as basic-highlight.directive.ts

In this typescript file we need to do these steps to create our own directives

export class BasicHighlightDorective {

}

Just like as component here we should give

@directive

import { directive } from "@angular/core";

@directive

We need to configure a directive for that we use selector

import { Directive, OnInit, ElementRef } from '@angular/core';

@Directive({

selector: '[appBasicHighlight]'/\*\*to give the instruction to the angular we use selector.

it should be unique it can be recognised by the element without [] in html page\*/

})

export class BasicHighlightDirective implements OnInit {

constructor(private elementRef1: ElementRef) {/\*\*we are getting access to the element and we are overriding the existing style \*/

}

ngOnInit() {

this.elementRef1.nativeElement.style.backgroundcolor = 'orange';

}

}

Unlike a component, directive doesn’t have a view, it doesn’t have a template.

To use these directive in another component

we need to do 2 things . first we need to inform in app.module.ts , just like a component

import { BasicHighlightDirective } from './basic-highlight/basic-highlight.directive';

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

RecipesComponent,

RecipesListComponent,

RecipesDetailComponent,

RecipeItemComponent,

ShoppingListComponent,

ShoppingEditComponent,

RecipesListDirective,

DirectivesComponent,

BasicHighlightDirective

],

The better approach to create attribute directives is by using ‘renderer’.

In the below code we use elementref to acces these attribute to perticualr element in the component html page.

import { Directive, OnInit, ElementRef, Renderer2 } from '@angular/core';

@Directive({

selector: '[appBetterHighliterDirectiveTs]'

})

export class BetterHighliterDirective implements OnInit {

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

In html page:

<p appBetterHighliterDirectiveTs>Style from Better-highlighter </p>

Hostlistener

import { Directive, OnInit, ElementRef, Renderer2, HostListener } from '@angular/core';

@Directive({

selector: '[appBetterHighliterDirectiveTs]'

})

export class BetterHighliterDirective implements OnInit {

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

@HostListener('mouseenter') mouseover(eventData: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'green');

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');

}

}

<p appBetterHighliterDirectiveTs>Style from Better-highlighter </p>

Using HostBinding to Bind to Host Properties

import { Directive, OnInit, ElementRef, Renderer2, HostListener, HostBinding } from '@angular/core';

@Directive({

selector: '[appBetterHighliterDirectiveTs]'

})

export class BetterHighliterDirective implements OnInit {

@HostBinding('style.backgroundColor') backgroundColor: String = 'transperent';

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

@HostListener('mouseenter') mouseover(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'green');

this.backgroundColor = 'green';

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');

this.backgroundColor = 'transparent';

}

}

<p appBetterHighliterDirectiveTs>Style from Better-highlighter </p>

Custome Attribute directives

import { Directive, OnInit, ElementRef, Renderer2, HostListener, HostBinding, Input } from '@angular/core';

@Directive({

selector: '[appBetterHighliterDirectiveTs]'

})

export class BetterHighliterDirective implements OnInit {

@Input() defaultColor: String = 'transparent';

@Input() highlightColor: String = 'blue';

@HostBinding('style.backgroundColor') backgroundColor: String = 'transperent';

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

this.backgroundColor = this.defaultColor; /\*\* this is the first applicable \*/

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

@HostListener('mouseenter') mouseover(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'green');

// this.backgroundColor = 'green';

this.backgroundColor = this.highlightColor;

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');

// this.backgroundColor = 'transparent';

this.backgroundColor = this.defaultColor;

}

}

To overwrite the colors we can do like this in html page of component

<p appBetterHighliterDirectiveTs [defaultColor]="'green'" [highlightColor]="'red'">Style from Better-highlighter </p>

Structural directives

<div \*ngIf="!onlyOdd">

<li class="list-group-item"

[ngClass]="{odd: even % 2 !== 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'lightblue' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</div>

</ng-template>

<ng-template [ngIf]="!onlyOdd">

<div >

<li class="list-group-item"

[ngClass]="{odd: even % 2 !== 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'lightblue' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</div>

These two syntaxes are same . if we use the ng-template then [ngIf] internally convert by angular into \*ngIf

Building Structural directive

To create directive

In commandprompt ng g d foldername/directivename.directive.ts

Unless.directive.ts

import { Directive , Input, TemplateRef, ViewContainerRef} from '@angular/core';

@Directive({

selector: '[appUnless]'

})

export class UnlessDirective {

@Input() set appUnless(condition: boolean) {/\*\*here the method name same as selector name to identify by angular \*/

if ( !condition) {

this.vcRef.createEmbeddedView(this.templateRef1);

} else {

this.vcRef.clear();

}

}

constructor(private templateRef1: TemplateRef<any>, private vcRef: ViewContainerRef) {

/\*\*what temple should render and second parameter is in which viewcontainer should be render \*/

}

}

<div class="container">

<div class="row">

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

<button class="btn btn-primary"

(click)="onlyOdd = !onlyOdd">

Only show Odd Numbers

</button>

<br><br>

<ul class="list-group">

<div \*ngIf="onlyOdd">

<li class="list-group-item"

[ngClass]="{odd: odd % 2 !== 0}"

[ngStyle]="{backgroundColor: odd % 2 !== 0 ? 'lightblue' : 'transparent'}"

\*ngFor="let odd of oddNumbers">

{{ odd }}

</li>

</div>

<div \*appUnless="onlyOdd"><!—here we used \*appUnless to use the structural directive -->

<li class="list-group-item"

[ngClass]="{odd: even % 2 !== 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'lightblue' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</div>

</ul>

<p appBasicHighlight>Style from Basic-highlighter </p>

<p appBetterHighliterDirectiveTs [defaultColor]="'green'" [highlightColor]="'red'">Style from Better-highlighter </p>

</div>

</div>

</div>

What are services & Dependency injection?

Service is a broad category encompassing any value, function, or feature that an app needs. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well. 

Angular distinguishes components from services in order to increase modularity and reusability.

* By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient. Ideally, a component's job is to enable the user experience and nothing more. It should present properties and methods for data binding, in order to mediate between the view (rendered by the template) and the application logic (which often includes some notion of a model).
* A component should not need to define things like how to fetch data from the server, validate user input, or log directly to the console. Instead, it can delegate such tasks to services. By defining that kind of processing task in an injectable service class, you make it available to any component. You can also make your app more adaptable by injecting different providers of the same kind of service, as appropriate in different circumstances.

Create a log service:

Export class LoggingService {

logStatusChange(status: string) {

console.log(‘A server status changed, new status: ‘ + status);

}

}

## Dependency injection



Components consume services; that is, you can inject a service into a component, giving the component access to that service class.

To define a class as a service in Angular, use the @[Injectable](https://angular.io/api/core/Injectable) decorator to provide the metadata that allows Angular to inject it into a component as a dependency.

Similarly, use the @[Injectable](https://angular.io/api/core/Injectable) decorator to indicate that a component or other class (such as another service, a pipe, or an NgModule) has a dependency. A dependency doesn't have to be a service—it could be a function, for example, or a value.

How to inject the above log service into component :

# Introduction to modules

Angular apps are modular and Angular has its own modularity system called NgModules. It can contain components, service providers, and other code files whose scope is defined by the containing NgModule. It can import functionality that is exported from other NgModules, and export selected functionality for use by other NgModules.

Every Angular app has at least one NgModule class, [the root module](https://angular.io/guide/bootstrapping), which is conventionally named AppModule and resides in a file named app.module.ts. You launch your app by bootstrapping the root NgModule.

While a small application might have only one NgModule, most apps have many more feature modules. The rootNgModule for an app is so named because it can include child NgModules in a hierarchy of any depth.

## NgModule metadata

An NgModule is defined as a class decorated with @[NgModule](https://angular.io/api/core/NgModule). The @[NgModule](https://angular.io/api/core/NgModule) decorator is a function that takes a single metadata object, whose properties describe the module. The most important properties are as follows.

* [declarations](https://angular.io/api/core/NgModule#declarations)—The [components](https://angular.io/guide/architecture-components), directives, and pipes that belong to this NgModule.
* [exports](https://angular.io/api/core/NgModule#exports)—The subset of declarations that should be visible and usable in the component templates of other NgModules.
* [imports](https://angular.io/api/core/NgModule#imports)—Other modules whose exported classes are needed by component templates declared in thisNgModule.
* providers—Creators of [services](https://angular.io/guide/architecture-services) that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
* [bootstrap](https://angular.io/api/core/NgModule#bootstrap)—The main application view, called the root component, which hosts all other app views. Only the root NgModule should set this [bootstrap](https://angular.io/api/core/NgModule#bootstrap) property.

Here's a simple root NgModule definition:

src/app/app.module.ts

content\_copyimport { [NgModule](https://angular.io/api/core/NgModule) } from '@angular/core';

import { [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) } from '@angular/platform-browser';

@[NgModule](https://angular.io/api/core/NgModule)({

[imports](https://angular.io/api/core/NgModule#imports): [ [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) ],

providers: [ Logger ],

[declarations](https://angular.io/api/core/NgModule#declarations): [ AppComponent ],

[exports](https://angular.io/api/core/NgModule#exports): [ AppComponent ],

[bootstrap](https://angular.io/api/core/NgModule#bootstrap): [ AppComponent ]

})

export class AppModule { }

The export of AppComponent is just to show how to export; it isn't actually necessary in this example. A root NgModule has no reason to export anything because other modules don't need to import the root NgModule.

**Services:**

**Recipes.service.ts**

import { EventEmitter, Injectable } from '@angular/core';

import { Recipe } from '../recipes/recipe.model';

import { Ingredient } from '../shared/ingredient.model';

import { ShoppingListService } from '../shopping-list/shopping-list.service';

@Injectable()

export class RecipeService {

// recipeselected = new EventEmitter<Recipe>();

/\*\* below we metioned recipes as private , so we cannot direcly access this arrary from out side of this service \*/

private recipes: Recipe[] = [

new Recipe('Over Night Oatmeal',

'Oatmeal recipe description',

'https://c1.staticflickr.com/4/3468/3294687099\_f49fba48e7\_b.jpg',

[

new Ingredient('Oats', 1 ),

new Ingredient('banana', 1),

new Ingredient('walnuts', 5)

]),

new Recipe('Fruit salad',

'Fruit Salad recipe description',

'https://c1.staticflickr.com/6/5737/30622968353\_35e06fcb52\_b.jpg',

[

new Ingredient('pomogranate', 1),

new Ingredient('banana', 1),

new Ingredient('apple', 1),

new Ingredient('kiwi', 1)

]

),

new Recipe('Alu gravy Curry',

'Alu recipe description',

'https://c1.staticflickr.com/9/8585/28906445485\_ce32150295\_b.jpg',

[

new Ingredient('potato', 4),

new Ingredient('tomato', 2),

new Ingredient('onion', 2)

])

];

/\*\* as we mentioned above arrary as private we need some method to call that arrarylist \*/

getRecipes () {

return this.recipes.slice(); /\*\* by using slice() we can send the copy of the arrary to out side of the service \*/

}

getRecipe(index: number) {

return this.recipes[index];

}

constructor(private slService: ShoppingListService) {}

/\*\* above one is for adding ingredients to shopping list \*/

addIngredientsToShoppingList(ingredients: Ingredient[]) {

this.slService.addIngredients(ingredients);

}

}

**Shoppinlist.service.ts**

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

import { Ingredient } from '../shared/ingredient.model';

import { Subject } from 'rxjs/Subject';

export class ShoppingListService {

// ingredientsChanged = new EventEmitter<Ingredient[]>();

ingredientsChanged = new Subject<Ingredient[]>();

private ingredients: Ingredient[] = [

new Ingredient('apples', 10) ,

new Ingredient('banana', 5)

];

getIngredients() {

return this.ingredients.slice();

/\*\* by using slice() method we can access only copy of the ingredients not original ingrddient array data\*/

// return this.ingredients;

}

addIngredient(ingredient: Ingredient) {

this.ingredients.push(ingredient);

// this.ingredientsChanged.emit(this.ingredients.slice());

this.ingredientsChanged.next(this.ingredients.slice());

}

addIngredients(ingredients: Ingredient[]) {

// for (let ingredi of ingredients) {

// this.addIngredient(ingredi);

this.ingredients.push(...ingredients);

/\*\* this will convert an array of elements to list of elements. for that we used '...'spread operator infront of ingredients \*/

// this.ingredientsChanged.emit(this.ingredients.slice()); /\*\* after spread we have to emit \*/

this.ingredientsChanged.next(this.ingredients.slice());

}

}

to use the recipe service in recipes.component.ts

under component decorator we need to add providers

@Component({

selector: 'app-recipes',

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

styleUrls: ['./recipes.component.css'],

providers: [RecipeService]

/\*\* since it is the parent class of recipes folder we informed to angular which service we are using for recipes \*/

})

After that we have to inject the service like below

export class RecipesComponent implements OnInit {

// selectedrecipe: Recipe;

constructor(private recipeservice: RecipeService) { }

export class RecipesDetailComponent implements OnInit {

recipe: Recipe;

id: number;

constructor(private recipeService: RecipeService,

private route: ActivatedRoute,

private router: Router ) { }

/\*\* 3 parameter is navigate method. we injected through Router \*/

ngOnInit() {

// here we are retriving recipedetails

this.route.params

.subscribe((params: Params) => {

this.id = +params['id']; /\*\* here + operator for converting params['id'] string data to int

here we get the dynamic 'id' of recipes now we have to get the recipe details through recipe service\*/

this.recipe = this.recipeService.getRecipe(this.id);

}

);

}

onAddToShopingList() {

this.recipeService.addIngredientsToShoppingList(this.recipe.ingredients);

}

To use one service in another service we use @Injectable()

And import this

import { EventEmitter, Injectable } from '@angular/core';

constructor(private slService: ShoppingListService) {}

/\*\* above one is for adding ingredients to shopping list \*/

addIngredientsToShoppingList(ingredients: Ingredient[]) {

this.slService.addIngredients(ingredients);

}

In the above example we used shoppinglistservice in recipesservice

Shoppinglistservice

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

import { Ingredient } from '../shared/ingredient.model';

export class ShoppingListService {

ingredientsChanged = new EventEmitter<Ingredient[]>();

private ingredients: Ingredient[] = [

new Ingredient('apples', 10) ,

new Ingredient('banana', 5)

];

getIngredients() {

return this.ingredients.slice();

/\*\* by using slice() method we can access only copy of the ingredients not original ingrddient array data\*/

// return this.ingredients;

}

addIngredient(ingredient: Ingredient) {

this.ingredients.push(ingredient);

this.ingredientsChanged.emit(this.ingredients.slice());

}

addIngredients(ingredients: Ingredient[]) {

// for (let ingredi of ingredients) {

// this.addIngredient(ingredi);

this.ingredients.push(...ingredients);

/\*\* this will convert an array of elements to list of elements. for that we used '...'spread operator infront of ingredients \*/

this.ingredientsChanged.emit(this.ingredients.slice()); /\*\* after spread we have to emit \*/

}

}

**Routing :**

By using routing we can bind the pages with out page refreshing

By using routing navigation will occur from one page to other page ,With out page refreshing

to do routing we create a routing module

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

import { Routes, RouterModule } from '@angular/router';

import { RecipesComponent } from './recipes/recipes.component';

import { ShoppingListComponent } from './shopping-list/shopping-list.component';

const approuts: Routes = [

/\*\* what ever the paths we want to mention in the page we specify here \*/

{path: '', redirectTo: '/recipes', pathMatch: 'full'},

{path: 'recipes' , component: RecipesComponent},

{path: 'shopping-list' , component: ShoppingListComponent }

];

@NgModule({

imports: [RouterModule.forRoot(approuts)], /\*\*here we configure the routes with routermodule \*/

exports: [RouterModule] /\*\* we have to export this Router module to main module 'app.module' \*/

})

export class AppRoutingModule {

}

Here

AppRoutingModule {

Is the class

To this class we need to add @NgModule to transform the normal type script class to an angularmodule.

After that we have to add the arrary of routes

const approuts: Routes = [

/\*\* what ever the paths we want to mention in the page we specify here \*/

{path: '', redirectTo: '/recipes', pathMatch: 'full'}, // this is the default one

{path: 'recipes' , component: RecipesComponent},

{path: 'shopping-list' , component: ShoppingListComponent }

];

After that we have to

@NgModule({

imports: [RouterModule.forRoot(approuts)], /\*\*here we configure the routes with routermodule \*/

exports: [RouterModule] /\*\* we have to export this Router module to main module 'app.module' \*/

})

In app.module.ts we need to import the app.routing.module

imports: [

BrowserModule,

FormsModule,

HttpModule,

AppRoutingModule

],

In app.component.html page we use

<router-outlet>

</router-outlet>

To render the data

Navigation of routes:

For this what ever the link we selected to active that

We have class=”active” this is bootstrapclass

But it doesn’t switch so we use

<li routerLinkActive="active"><a routerLink="/recipes" >Recipies</a></li>

<li routerLinkActive="active"><a routerLink="/shopping-list" >Shopping List</a></li>

RouterlinkActive=”active”

**Child routing:**

{path: 'recipes' , component: RecipesComponent , children: [{

path: '', component: RecipeStartComponent

} ,

{path: ':id', component: RecipesDetailComponent}

]

},

In parent component we should use <routeroutlet></ routeroutlet> directive

Recipes.component.html

# RouterOutlet

DIRECTIVE

Acts as a placeholder that Angular dynamically fills based on the current router state.(means if we select recipes tab it binds recipeslist, if we select editrecipe then only it shows edit page)

<div class="row">

<div class="col-md-5">

<app-recipes-list></app-recipes-list>

<!-- (recipewasselected)="selectedrecipe = $event"></app-recipes-list> -->

</div>

<div class="col-md-7">

<router-outlet></router-outlet>

<!-- <app-recipes-detail

\*ngIf="selectedrecipe; else infoText" [recipe] = "selectedrecipe"></app-recipes-detail>

<ng-template #infoText>

<p>Please Select a Recipe!</p>

</ng-template> -->

</div>

</div>

How to bind data based on passing dynamic id(in the url)

export class RecipesDetailComponent implements OnInit {

recipe: Recipe;

constructor(private recipeService: RecipeService,

private route: ActivatedRoute ) { }

export class RecipesDetailComponent implements OnInit {

recipe: Recipe;

id: number;

constructor(private recipeService: RecipeService,

private route: ActivatedRoute ) { }

ngOnInit() {

// here we are retriving recipedetails

this.route.params

.subscribe((params: Params) => {

this.id = +params['id'];

}

);

}

Now in the recipeservice we add one method to get recipes based on id

getRecipe(index: number) {

return this.recipes[index];

}

After that in recipedetailcomponent we call that method.

this.recipe = this.recipeService.getRecipe(this.id);

export class RecipesDetailComponent implements OnInit {

recipe: Recipe;

id: number;

constructor(private recipeService: RecipeService,

private route: ActivatedRoute ) { }

ngOnInit() {

// here we are retriving recipedetails

this.route.params

.subscribe((params: Params) => {

this.id = +params['id']; /\*\* here + operator for converting params['id'] string data to int

here we get the dynamic 'id' of recipes now we have to get the recipe details through recipe service\*/

this.recipe = this.recipeService.getRecipe(this.id);

}

);

}

Passing dynamic parameters to links(if we click particular link, based on that

In recipe-itemcomponent

[routerLink]="['index']"

This routerlink binds the selected recipe index value to the link( page url)

<a

style="cursor: pointer;"

[routerLink]="['index']"

class="list-group-item clearfix" >

<!-- class="list-group-item clearfix" (click)="onselected()"> -->

<div class="pull-left">

<h4 class="list-group-item-heading"> {{recipe.name}} </h4>

<p class="list-group-item-text"> {{recipe.description}} </p>

</div>

<span class="pull-right">

<img src="{{recipe.imagePath}}"

alt="{{recipe.name}}" class="img-responsive" style="max-height:50px" >

</span>

</a>

To get that above index value

We need to get that index value from recipe-item component

@Input() index: number; /\*\* now we can pass this index from the outside of the component

now in this case from recipes-list component\*/

next in recipes-list.component.html

<app-recipe-item

\*ngFor="let recipeEl of recipes; let i = index"

[recipe]="recipeEl"

[index]="i">

</app-recipe-item>

To activate(which link we clicked, that link will highlight) the selected link dynamically we use

routerLinkActive = 'active'

in html page.

Here in (recipe-item.component.html) page

**Observables**

<http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html>

<https://www.youtube.com/watch?v=xvLsvyrAMNk>

# Observables in Angular

Angular makes use of observables as an interface to handle a variety of common asynchronous operations. For example:

* The [EventEmitter](https://angular.io/api/core/EventEmitter) class extends Observable.
* The HTTP module uses observables to handle AJAX requests and responses.
* The Router and Forms modules use observables to listen for and respond to user-input events.

## Event emitter

Angular provides an [EventEmitter](https://angular.io/api/core/EventEmitter) class that is used when publishing values from a component through the @[Output](https://angular.io/api/core/Output)() decorator. [EventEmitter](https://angular.io/api/core/EventEmitter) extends Observable, adding an [emit()](https://angular.io/api/core/EventEmitter#emit) method so it can send arbitrary values. When you call [emit()](https://angular.io/api/core/EventEmitter#emit), it passes the emitted value to the next() method of any subscribed observer.

A good example of usage can be found on the [EventEmitter](https://angular.io/api/core/EventEmitter) documentation. Here is the example component that listens for open and close events:

<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>

Here is the component definition:

EventEmitter

content\_copy

1. @[Component](https://angular.io/api/core/Component)({
2. selector: 'zippy',
3. [template](https://angular.io/api/core/Component#template): `
4. <div class="zippy">
5. <div (click)="toggle()">Toggle</div>
6. <div [hidden]="!visible">
7. <ng-content></ng-content>
8. </div>
9. </div>`})
11. export class ZippyComponent {
12. visible = true;
13. @[Output](https://angular.io/api/core/Output)() open = new [EventEmitter](https://angular.io/api/core/EventEmitter)<any>();
14. @[Output](https://angular.io/api/core/Output)() close = new [EventEmitter](https://angular.io/api/core/EventEmitter)<any>();
16. toggle() {
17. this.visible = !this.visible;
18. if (this.visible) {
19. this.open.emit(null);
20. } else {
21. this.close.emit(null);
22. }
23. }
24. }

## HTTP

Angular’s [HttpClient](https://angular.io/api/common/http/HttpClient) returns observables from HTTP method calls. For instance, http.get(‘/api’) returns an observable. This provides several advantages over promise-based HTTP APIs:

* Observables do not mutate the server response (as can occur through chained .then() calls on promises). Instead, you can use a series of operators to transform values as needed.
* HTTP requests are cancellable through the [unsubscribe()](https://angular.io/api/service-worker/SwPush#unsubscribe) method.
* Requests can be configured to get progress event updates.
* Failed requests can be retried easily.

## Async pipe

The [AsyncPipe](https://angular.io/api/common/AsyncPipe) subscribes to an observable or promise and returns the latest value it has emitted. When a new value is emitted, the pipe marks the component to be checked for changes.

The following example binds the time observable to the component's view. The observable continuously updates the view with the current time.

Using async pipe

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'async-observable-pipe',

[template](https://angular.io/api/core/Component#template): `<div><code>observable|async</code>:

[Time](https://angular.io/api/common/Time): {{ time | async }}</div>`

})

export class AsyncObservablePipeComponent {

time = new Observable(observer =>

setInterval(() => observer.next(new Date().toString()), 1000)

);

}

## Router

[Router.events](https://angular.io/api/router/Router#events) provides events as observables. You can use the [filter()](https://angular.io/api/core/QueryList#filter) operator from RxJS to look for events of interest, and subscribe to them in order to make decisions based on the sequence of events in the navigation process. Here's an example:

Router events

content\_copy

1. import { [Router](https://angular.io/api/router/Router), [NavigationStart](https://angular.io/api/router/NavigationStart) } from '@angular/router';
2. import { filter } from 'rxjs/operators';
4. @[Component](https://angular.io/api/core/Component)({
5. selector: 'app-routable',
6. templateUrl: './routable.component.html',
7. [styleUrls](https://angular.io/api/core/Component#styleUrls): ['./routable.component.css']
8. })
9. export class Routable1Component implements [OnInit](https://angular.io/api/core/OnInit) {
11. navStart: Observable<[NavigationStart](https://angular.io/api/router/NavigationStart)>;
13. constructor(private router: [Router](https://angular.io/api/router/Router)) {
14. // Create [a](https://angular.io/api/router/RouterLinkWithHref) new Observable the publishes only the [NavigationStart](https://angular.io/api/router/NavigationStart) event
15. this.navStart = router.events.pipe(
16. filter(evt => evt instanceof [NavigationStart](https://angular.io/api/router/NavigationStart))
17. ) as Observable<[NavigationStart](https://angular.io/api/router/NavigationStart)>;
18. }
20. ngOnInit() {
21. this.navStart.subscribe(evt => console.log('Navigation Started!'));
22. }
23. }

The [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) is an injected router service that makes use of observables to get information about a route path and parameters. For example, ActivateRoute.url contains an observable that reports the route path or paths. Here's an example:

ActivatedRoute

content\_copy

1. import { [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) } from '@angular/router';
3. @[Component](https://angular.io/api/core/Component)({
4. selector: 'app-routable',
5. templateUrl: './routable.component.html',
6. [styleUrls](https://angular.io/api/core/Component#styleUrls): ['./routable.component.css']
7. })
8. export class Routable2Component implements [OnInit](https://angular.io/api/core/OnInit) {
9. constructor(private [activatedRoute](https://angular.io/api/router/RouterOutlet#activatedRoute): [ActivatedRoute](https://angular.io/api/router/ActivatedRoute)) {}
11. ngOnInit() {
12. this.activatedRoute.url
13. .subscribe(url => console.log('The [URL](https://angular.io/api/core/SecurityContext#URL) changed to: ' + url));
14. }
15. }

## Reactive forms

Reactive forms have properties that use observables to monitor form control values. The [FormControl](https://angular.io/api/forms/FormControl) properties valueChanges and statusChanges contain observables that raise change events. Subscribing to an observable form-control property is a way of triggering application logic within the component class. For example:

Reactive forms

content\_copy

1. import { [FormGroup](https://angular.io/api/forms/FormGroup) } from '@angular/forms';
3. @[Component](https://angular.io/api/core/Component)({
4. selector: 'my-component',
5. [template](https://angular.io/api/core/Component#template): 'MyComponent Template'
6. })
7. export class MyComponent implements [OnInit](https://angular.io/api/core/OnInit) {
8. nameChangeLog: string[] = [];
9. heroForm: [FormGroup](https://angular.io/api/forms/FormGroup);
11. ngOnInit() {
12. this.logNameChange();
13. }
14. logNameChange() {
15. const nameControl = this.heroForm.get('name');
16. nameControl.valueChanges.forEach(
17. (value: string) => this.nameChangeLog.push(value)
18. );
19. }
20. }

Forms:

Angular have 2 types of approach to deal with forms

1. Template-driven : Angular inform the form object from the DOM
2. Reactive : Form is created programmatically and synchronized with DOM (complex way)

**Template driven :**

To add this forms-demo link in heard component :

1. created a new component with the name ng g c forms/forms-demo –spec false in angularcli command proment
2. in herader.component.html

<li routerLinkActive="active"><a routerLink="/forms-demo" >Forms</a></li>

To get the Link in header section

1. <div class="collapse navbar-collapse">
2. <ul class="nav navbar-nav">
3. <li routerLinkActive="active"><a routerLink="/recipes" >Recipies</a></li>
4. <li routerLinkActive="active"><a routerLink="/shopping-list" >Shopping List</a></li>
5. <li routerLinkActive="active"><a routerLink="/forms-demo" >Forms</a></li>
6. <!-- <li><a href="#" (click)="onSelect('recipe')">Recipies</a></li>
7. <li><a href="#" (click)="onSelect('shopping-list')" >Shopping List</a></li>
8. <li><a href='#' (click)="onSelect('directives')">Directives</a></li> -->
9. </ul>

3.next in the app-routing-module.ts

{path: 'forms-demo' , component: FormsDemoComponent }

const approuts: Routes = [

/\*\* what ever the paths we want to mention in the page we specify here \*/

{path: '', redirectTo: '/recipes', pathMatch: 'full'},

{path: 'recipes' , component: RecipesComponent , children: [

{path: '', component: RecipeStartComponent},

{path: 'new', component: RecipeEditComponent},

{path: ':id', component: RecipesDetailComponent},

{path: ':id/edit', component: RecipeEditComponent}

]

},

{path: 'shopping-list' , component: ShoppingListComponent },

{path: 'forms-demo' , component: FormsDemoComponent }

];

**For template-driven approach we need**

FormsModule,

In app.module.ts file , angular by default provide this to us.

imports: [

BrowserModule,

FormsModule,

HttpModule,

AppRoutingModule

],

To register the controls

ngModel name="username"

<label for="username">Username</label>

<input type="text" id="username" class="form-control" ngModel name="username">

If we write the method like this in forms.component.ts file

onSubmit() {

}

We need to call this function in html page. Usually we call this(click listener) in button control.

In angular we can do like below

<form (ngSubmit)="onSubmit()" #f>

Here #f is the local reference to the form . and we write the onsubmit() function like below then we get all the form content into log. But actually that Is not our requirement .

onSubmit(form: ElementRef) {

console.log(form);

}

So, we have to expose the object in the native reference.

<form #userform="ngForm" (ngSubmit)="onSubmit(userform)">

This is how we get the access to the object created by the javascript through angular.

onSubmit(userForm: NgForm) {

console.log(userForm);

}

Or we can do with @viewchild

<form #userform="ngForm" (ngSubmit)="onSubmit()">

@viewchild(‘userform’) signupForm: NgForm;

onSubmit() {

console.log(this.signupForm);

}

**Form Validations:**

Which Validators do ship with Angular?

Check out the Validators class: <https://angular.io/docs/ts/latest/api/forms/index/Validators-class.html> - these are all built-in validators, though that are the methods which actually get executed (and which you later can add when using the reactive approach).

For the template-driven approach, you need the directives. You can find out their names, by searching for "validator" in the official docs: <https://angular.io/api?type=directive> - everything marked with "D" is a directive and can be added to your template.

Additionally, you might also want to enable HTML5 validation (by default, Angular disables it). You can do so by adding the ngNativeValidate  to a control in your template.

To set the default value to the dropdown:

This is normal dropdown with these 2 questions. We can set one of those 2 questions as default value of dropdown.

<label for="secret">Secreat Questions</label>

<select id="secret"

class="form-control"

ngModel

name="secret">

<option value="pet">Your first pet?</option>

<option value="teacher">Your first teacher?</option>

</select>

Ngmodel with oneway binding

[ngModel]=”’pet’”

If we replace this with ngModel .here ‘pet’ is the value of options.

In some other way also we can set this value .

In component.ts file

export class FormsDemoComponent implements OnInit {

defaultdropdownvalue: 'pet';

and in html page [ngModel]=”defaultdropdownvalue”

we saw all 3 forms

with out binding : just tell to the angular the input is a controller

one-way binding : to give the input as a default value

2-way binding : instantly output the value whatever we want to do with that

If we have a big form some times we require grouping of some controls that time we use

Grouping form controls.

For this we need to add

ngModelGroup="userdata"

in the below form

<form #userform="ngForm" (ngSubmit)="onSubmit(userform)">

<!-- #userform is the local reference -->

<div id="user-data" ngModelGroup="userdata">

Setvalue,(overrides all the controles values in the form )

Patchvalue (overrides only particular value from the from)

suggestUsername() {

const suggestedName = 'Superuser';

/// below one byusing setvalue we are overriding form values

// this.signupform.setValue({

// userdata1: {

// username: suggestedName,

// email: ''

// },

// secret: 'pet',

// questionAnswer: '',

// gen: 'Male'

// });

// }

// whatever the value we want to override by using patchvalue()

this.signupform.form.patchValue({

userdata: {

username: suggestedName

}

});

}

Reactive Approach:

In this approach the form is created programmatically in typescript file.

We create the form by using the below code

Import { FormGroup } from ‘@angular/forms’

Export class reactorforms {

Genders = [‘Male’, ‘Female’];

signupForm: FormGroup;

}

After this in app.module.ts we have to remove ‘FormsModule’ from imports in that place we need to place ‘ReactiveFormsModule

The difference of these 2 approaches we don’t get the form through local reference.

In template driven approach we use local reference.

For this reactive approach in ngOnInt(){

This.signupform = new FormGroup({

‘username’: new formcontrol(null),

‘email’:new formcontrol(null),

‘gender’: new formcontrol(‘female’)

});

}

In the html page we write like this

<form [formGroup]=”signupform” (ngSubmit)=”onsubmit()”>

<div class=”form-group”>

<lable for=”username”> Username</lable>

<input type=”text” id=”username” [formcontrolName]= “’username’” class=”form-control”> or we can write like formcontrolname=”username”

</div>

</form>

formControlName, [formgroup] are important in this approach

Onsubmit()

{

Console.log(this.signupform);

What ever we set up as an argument(in model(component.ts file)) is passing as value to the form

How to add the validators :

ngOnInt(){

This.signupform = new FormGroup({

‘username’: new formcontrol(null, Validators.required),

‘email’:new formcontrol(null, [Validators.required, Validators.email]),

‘gender’: new formcontrol(‘female’)

});

How to show the invalid message

<form [formGroup]=”signupform” (ngSubmit)=”onsubmit()”>

<div class=”form-group”>

<lable for=”username”> Username</lable>

<input type=”text” id=”username” [formcontrolName]= “’username’” class=”form-control”> or we can write like formcontrolname=”username”

<span \*ngIf=”!signupform.get(‘username’).valid && signupform.get(‘username).touched” class=”help-block”> Please enter a valid username!

</span>

</div>

</form>

Grouping controls in reactive approach:

ngOnInt(){

This.signupform = new FormGroup({

‘userData’: new FormGroup({

‘username’: new formcontrol(null, Validators.required),

‘email’:new formcontrol(null, [Validators.required, Validators.email])

}),

‘gender’: new formcontrol(‘female’)

});

In html page we have to place div for to place the username and email as one group.

<form [formGroup]=”signupform” (ngSubmit)=”onsubmit()”>

<div formGroupName=” userData”>

<div class=”form-group”>

<lable for=”username”> Username</lable>

<input type=”text” id=”username” [formcontrolName]= “’username’” class=”form-control”> or we can write like formcontrolname=”username”

<span \*ngIf=”!signupform.get(‘userData.username’).valid && signupform.get(‘userData.username).touched” class=”help-block”> Please enter a valid username!

</span>

<lable for=”email”> email</lable>

<input type=”email” id=”email” [formcontrolName]= “email” class=”form-control”> or we can write like formcontrolname=”email”

<span \*ngIf=”!signupform.get(‘userData.email’).valid && signupform.get(‘userData.email’).touched” class=”help-block”> Please enter a validemail!

</span>

</div>

<div class=”radio” \*ngFor=”let gender for genders”>

<lable>

<input type=”radio” id=”gender” [formcontrolName]= “gender” [value]=”gender”>{{ gender}} or we can write like formcontrolname=”gender”

</lable>

</div>

</div>

</form>

FormArray:

**In Reactive approach:**

First we create an html page with normal input fields. With out any [formgroup ] [formcontrolname] [formareay]…. Normal Html page.

After that in component we initialize all the controls in onnginit()

ngOnInit() {

this.route.params

.subscribe(

(params: Params) => {

this.id = +params['id'];

this.editmode = params['id'] != null;

this.initform();

// console.log(this.editmode);

}

);

}

private initform() {

let recipename = '';

let recipeImagePath = '';

let recipedescription = '';

// for ingredients

let recipeIngredients = new FormArray([]);

if (this.editmode) {

const recipe = this.recipeService.getRecipe(this.id);

recipename = recipe.name;

recipeImagePath = recipe.imagePath;

recipedescription = recipe.description;

// to get the ingredients in recipe we are checking ingredients are there for that recipe or not

if (recipe['ingredients']) {

// if that recipe have more than one ingredient we are looping through

for (let ingredient of recipe.ingredients) {

// here we are pushing the ingredients to alredy defined array , recipeIngredients

recipeIngredients.push(

// here we specify formGroup because we have name and amount in the ingredient array.

new FormGroup({

'name': new FormControl(ingredient.name),

'amount': new FormControl(ingredient.amount)

})

);

}

}

}

this.recipeform = new FormGroup({

// we assign the values here. whatever

// the name given here we need to give the save name

// to formcontrolName in the html page. after this initialization we need to sync this with html page

'name': new FormControl(recipename),

'imagepath': new FormControl(recipeImagePath),

'description': new FormControl(recipedescription),

'ingredients': recipeIngredients

});

After this we need to add the reactivemodule in app.module.ts

imports: [

BrowserModule,

FormsModule,

ReactiveFormsModule,

HttpModule,

AppRoutingModule

],

Now we can map the things to html page by using formGroup, formcontrolname …

<form [formGroup]="recipeform" (ngSubmit)="onSubmit()">

<div class="row">

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

<button type="submit" class="btn btn-success">Save</button>

<button type="button" class="btn btn-danger">Cancel</button>

</div>

</div>

<div class="row">

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

<div class="form-group">

<label for="name">Name</label>

<input type="text" id="name"

[formControlName]= "'name'"

class="form-control">

</div>

</div>

</div>

<div class="row">

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

<div class="form-group">

<label for="imagepath">Image URL</label>

<input type="text" id="imagepath"

formControlName= "imagepath"

class="form-control">

</div>

</div>

</div>

<div class="row">

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

<img src="" class="img-responsive">

</div>

</div>

<div class="row">

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

<div class="form-group">

<label for="description">Description</label>

<textarea type="text" id="description"

formControlName= "description"

class="form-control" rows="6"> </textarea>

</div>

</div>

</div>

<div class="row">

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

<div class="row">

<div class="col-xs-8">

<input type="text" class="form-control">

</div>

<div class="col-xs-2">

<input type="number" class="form-control">

</div>

<div class="col-xs-2">

<button type="number" class="btn btn-danger">X</button>

</div>

</div>

</div>

</div>

</form>

Validations in Reactive approach:

Here in this approach we do validations in component not in html page . at the time of initialize value to the controllers in the component.

this.recipeform = new FormGroup({

// we assign the values here. whatever

// the name given here we need to give the save name

// to formcontrolName in the html page. after this initialization we need to sync this with html page

'name': new FormControl(recipename, Validators.required),

'imagepath': new FormControl(recipeImagePath, Validators.required),

'description': new FormControl(recipedescription, Validators.required),

'ingredients': recipeIngredients

});

}

For ingredients adding

private initform() {

let recipename = '';

let recipeImagePath = '';

let recipedescription = '';

// for ingredients

let recipeIngredients = new FormArray([]);

if (this.editmode) {

const recipe = this.recipeService.getRecipe(this.id);

recipename = recipe.name;

recipeImagePath = recipe.imagePath;

recipedescription = recipe.description;

// to get the ingredients in recipe

if (recipe['ingredients']) {

for (let ingredient of recipe.ingredients) {

recipeIngredients.push(

new FormGroup({

'name': new FormControl(ingredient.name, Validators.required),

'amount': new FormControl(ingredient.amount,

[Validators.required,

Validators.pattern(/^[1-9]+[0-9]\*$/)

])

})

);

onAddIngredient() {

(<FormArray>this.recipeform.get('ingredients')).push(

new FormGroup({

'name': new FormControl(null, Validators.required),

'amount': new FormControl(null, [Validators.required,

Validators.pattern(/^[1-9]+[0-9]\*$/)

])

})

);

}

In the html page

<form [formGroup]="recipeform" (ngSubmit)="onSubmit()">

<div class="row">

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

<button type="submit" class="btn btn-success" [disabled]="!recipeform.valid">Save</button>

<button type="button" class="btn btn-danger">Cancel</button>

</div>

</div>

**PIPES:**

Pipes are features built in to angular which basically transforms output in the template(in html).

A pipe takes in data as input and transforms it to a desired output.

Suppose we have the property like username=’sanjana’

We bind this by using property binding <p>{{ username }}</p>

We want to display the name in uppercase , we don’t want to change the property it’s self in that case we can use PIPEs.

<p>{{ username | uppercase}}</p>

We can add the parameters to the pipes by using ‘ : ‘

<p>{{DateofJoin | date}}</p>

<p>{{DateofJoin | date:’fullDate’}}</p> we can add any number of parameters by using :

We can see all built in pipes in below link

<https://angular.io/api?query=pipe>

example of custom pipe:

we can create pipe by using angularcli command prompt

ng g p shorten –spec false

shorten.pipe.ts

import { PipeTransform, Pipe } from '@angular/core';

@Pipe(

{

name : 'shorten' // this is the name to use it in Html page to apply this pipe

})

export class ShortenPipe implements PipeTransform {

// here we take pipetransform interface to transform the value into shorten value.

// after this we need to add this 'ShortenPipe' in declaration part of app.module.ts file to use in any html

transform(value: any, limit: number) {

// here first parameter is actual string, second one is limit of data type number

// we can give this as like this in html ' controlname| shorten:15 if we want to pass more parameters we have to add parameters here

if (value.length > limit) {

return value.substr(0, limit) + '...';

}

return value;

}

}

In app.module.ts

@NgModule({

declarations: [

AppComponent,

ShortenPipe

],

In recipes-detail.component.html

<h1>{{recipe.name | shorten:15 | uppercase }}</h1>

**HOW to add new link in Header component:**

First we have to create a new component . here it is ng g c filters/pipes –spec false

Next In app.module.ts file

const approuts: Routes = [

/\*\* what ever the paths we want to mention in the page we specify here \*/

{path: '', redirectTo: '/recipes', pathMatch: 'full'},

{path: 'recipes' , component: RecipesComponent , children: [

{path: '', component: RecipeStartComponent},

{path: 'new', component: RecipeEditComponent},

{path: ':id', component: RecipesDetailComponent},

{path: ':id/edit', component: RecipeEditComponent}

]

},

{path: 'shopping-list' , component: ShoppingListComponent },

{path: 'forms-demo' , component: FormsDemoComponent },

{path: 'pipes' , component: FiltersComponent }

];

@NgModule({

imports: [RouterModule.forRoot(approuts)], /\*\*here we configure the routes with routermodule \*/

exports: [RouterModule] /\*\* we have to export this Router module to main module 'app.module' \*/

})

In header.component.ts

<div class="collapse navbar-collapse">

<ul class="nav navbar-nav">

<li routerLinkActive="active"><a routerLink="/recipes" >Recipies</a></li>

<li routerLinkActive="active"><a routerLink="/shopping-list" >Shopping List</a></li>

<li routerLinkActive="active"><a routerLink="/forms-demo" >Forms</a></li>

<li routerLinkActive="active"><a routerLink="/pipes" >Pipes</a></li>

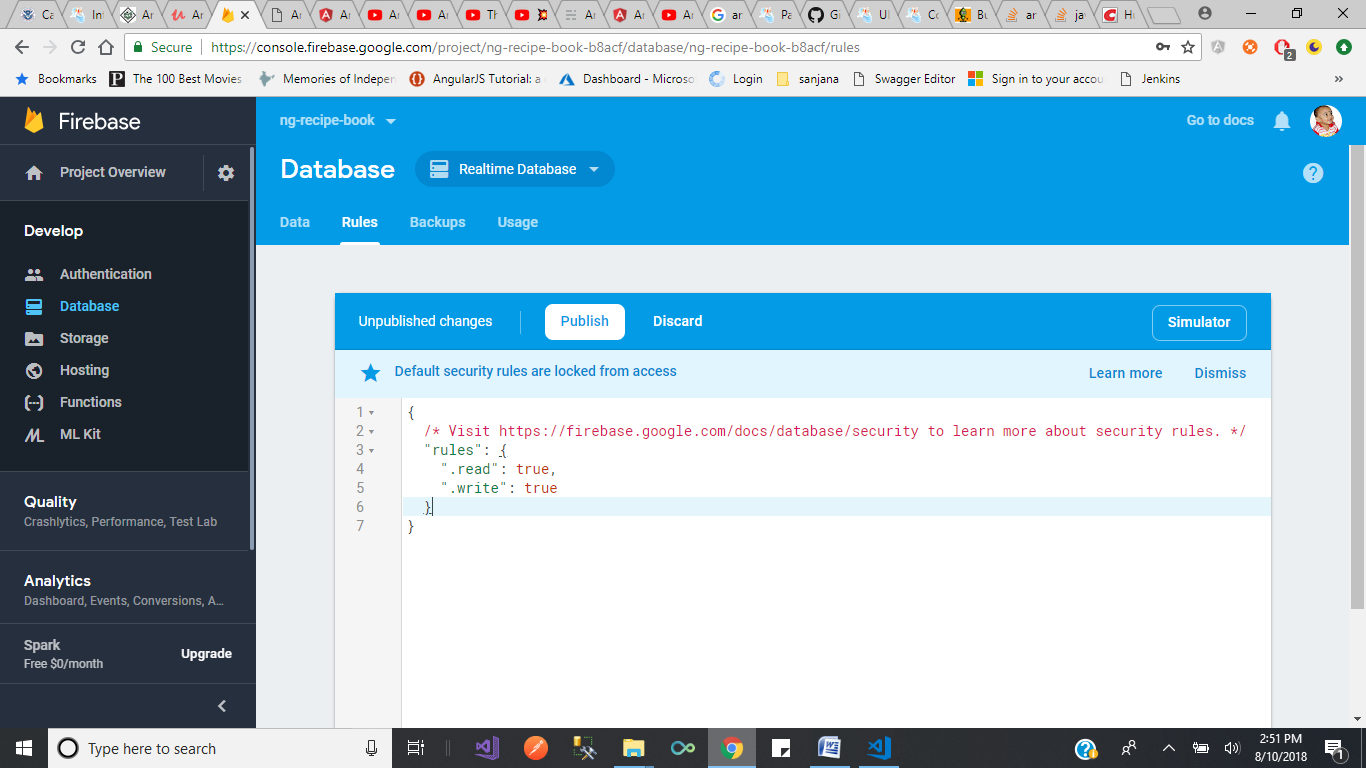
**We use google firebase as database**

For this

<https://firebase.google.com/>

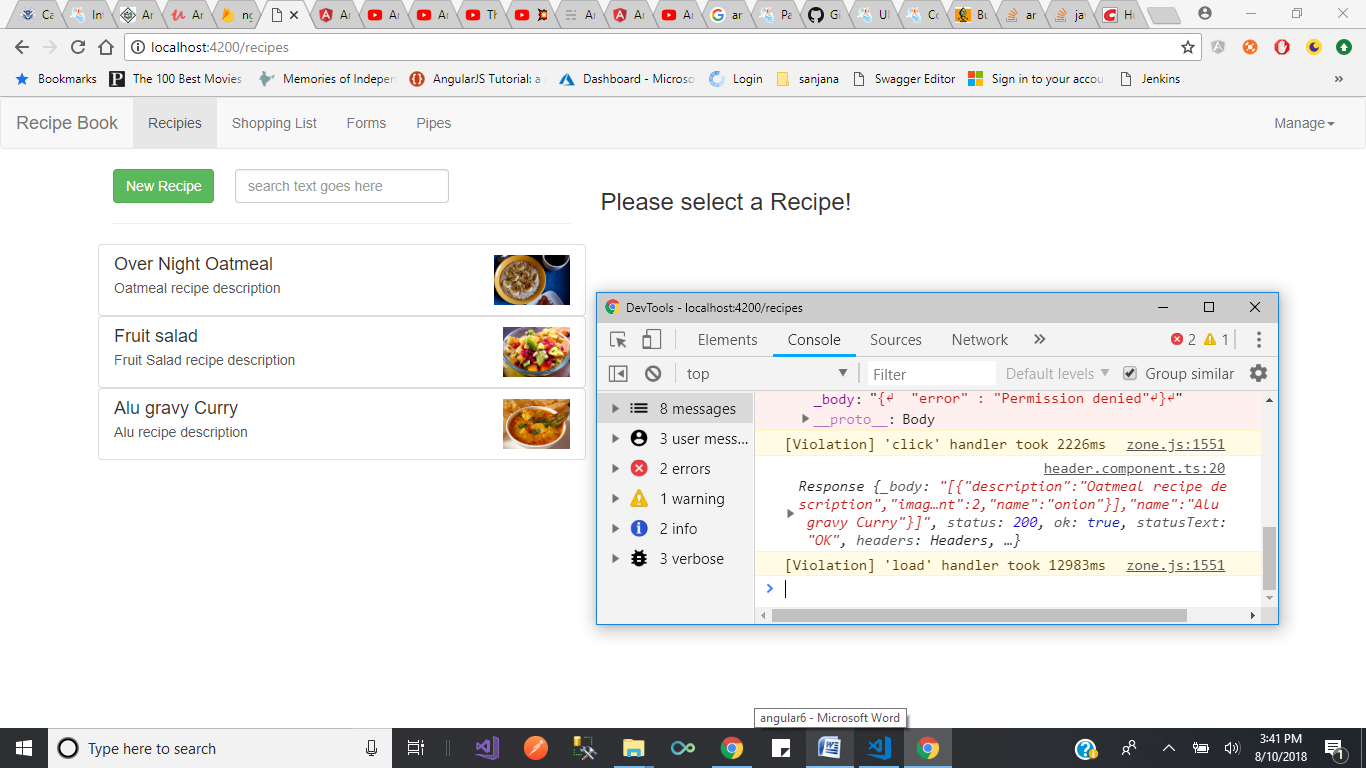
click go to console. It ask for gmail credentials if we give those we can create database

AddProject 🡪 ng-recipe-book 🡪database 🡪



We have to publish this to save the details . by default read and write is false.

Here we are trying to save the recipes into firebase by using put method



For this we are using recipeservice , headercomponent ( we have manage dropdown in headercomponent). When we click savedata under ‘mange’ dropdown . thse recipe details should save in the firedatabase .

For this in recipe.service.ts

constructor(private slService: ShoppingListService, private http: Http) {}

/\*\* above one is for adding ingredients to shopping list and second parameter to save the details into server to pass the request\*/

storeRecipes() {

// put method over rides all data in the firebase.we should mention 'recipes.json' at the end of url

// (we get it from firebase) if we dont mention recipes.json at the end we will get cross site error

return this.http.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json', this.getRecipes());

// the above statment returns only observable we have to subscribe this observable in header.component.ts

}

In app.module.ts

providers: [ShoppingListService, RecipeService],

in header.component.html

<ul class="nav navbar-nav navbar-right">

<li class="dropdown" appDropdown>

<a style="cursor: pointer" class="dropdown-toggle" role="button">Manage<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a style="cursor: pointer" (click) ="onSaveData()">Save Data</a></li>

<li><a style="cursor: pointer" >Fetch Data</a></li>

</ul>

</li>

</ul>

In heade.component.ts

export class HeaderComponent {

// @Output() featureSelected = new EventEmitter<string>();

// onSelect(feature: string) {

// this.featureSelected.emit(feature);

// }

constructor(private recipeService: RecipeService) {} // to use the service here we have ot write this

onSaveData() {

this.recipeService.storeRecipes()

.subscribe(

(response: Response) => {

console.log(response);

}

);

}

If we run the application and click on manage dropdown 🡪 savedata then details saved in the firebase database like below.



For get method.

In header.component.html

<ul class="nav navbar-nav navbar-right">

<li class="dropdown" appDropdown>

<a style="cursor: pointer" class="dropdown-toggle" role="button">Manage<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a style="cursor: pointer" (click) ="onSaveData()">Save Data</a></li>

<li><a style="cursor: pointer" (click)="onFetchData()" >Fetch Data</a></li>

</ul>

</li>

</ul>

In header.component.ts

onFetchData() {

this.recipeService.fetchRecipes();

}

In recipe.service.ts

fetchRecipes() {

this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json')

.subscribe(

(response: Response) => {

const recipes: Recipe[] = response.json();

this.setRecipes(recipes);

}

);

}

setRecipes(recipes: Recipe[]) {

this.recipes = recipes;

this.recipesChanged.next(this.recipes.slice());

}

Or we can write just like below.

In headercomponent.ts we can write the subscribe method instead of recipe.service.ts

onFetchData() {

this.recipeService.fetchRecipes()

.subscribe(

(response: Response) => {

const recipes: Recipe[] = response.json();

this.recipeService.setRecipes(recipes);

}

);

}

But we have to return observable data from recipe.service.ts

fetchRecipes() {

return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json');

// .subscribe(

// (response: Response) => {

// const recipes: Recipe[] = response.json();

// this.setRecipes(recipes);

// }

// );

}

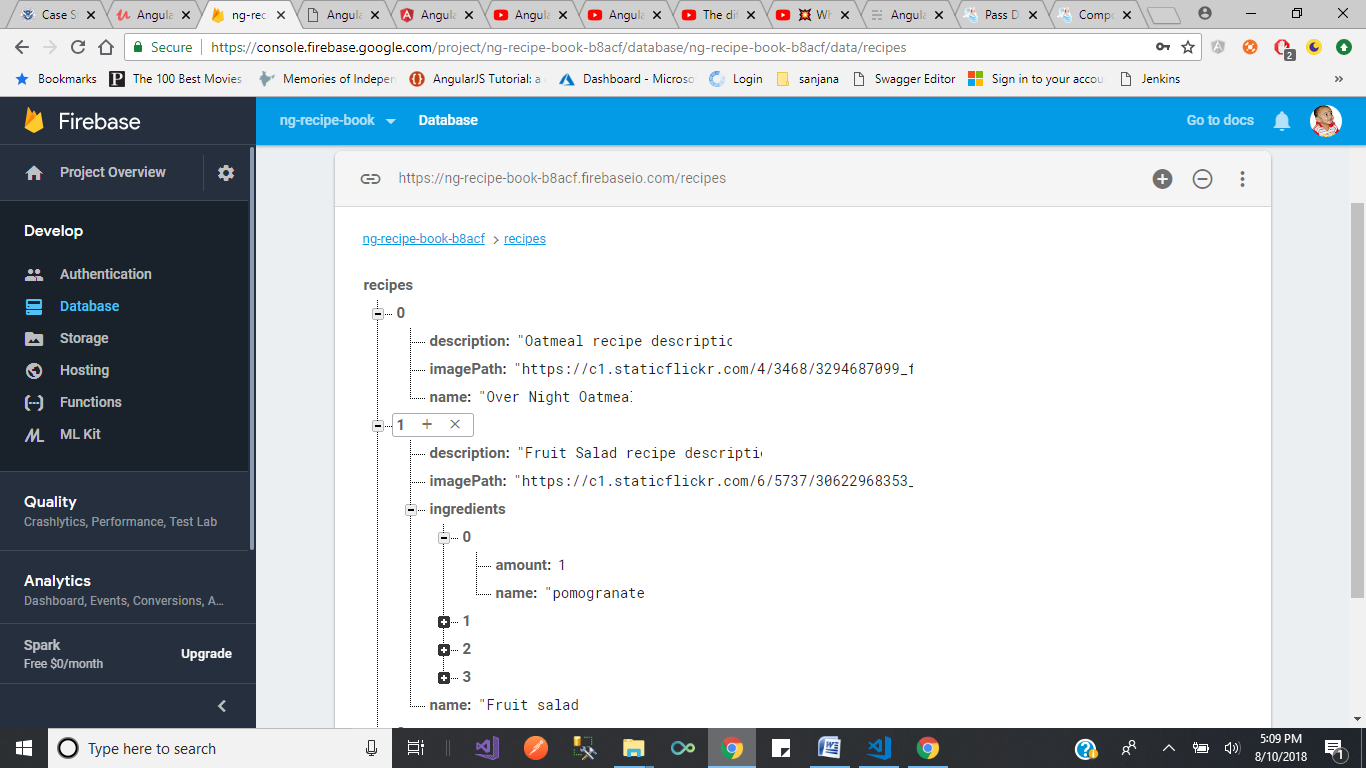
setRecipes(recipes: Recipe[]) {

this.recipes = recipes;

this.recipesChanged.next(this.recipes.slice());

}

Here we can delete all the ingredients from application . if we click savedata under mange dropdown in this scenario in firedatabase it stores the data with out any ingredient parameter in the data .

 here in 0 index there is noi ingredient parameter.

Tofix this we use .map before .subscribe method.

To work with map observable

Don't forget - if you're using Angular (and therefore also RxJS 6+) and you're NOT using rxjs-compat  (npm install --save rxjs-compat  - you may ignore this lecture then, use the code as shown in the videos!), you have to use operators like map()  differently:

Instead of

....map(...)

use

....pipe(map(...))

map also needs to be imported:

Instead of

import 'rxjs/Rx';

use

import { map } from 'rxjs/operators';

after installing that in header.component.ts we need to import map

import { map } from 'rxjs/operators';

import 'rxjs/RX';

then .map() will work , transform responses easily with observable operators(map())

fetchRecipes() {

this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json').map

(

(response: Response) => {

const recipes: Recipe[] = response.json();

for (let recipe of recipes) {

if (!recipe['ingredients']) {

recipe['ingredients'] = [];

}

}

return recipes;

}

)

.subscribe(

(response: Response) => {

const recipes: Recipe[] = response.json();

this.setRecipes(recipes);

}

);

}

Or

onFetchData() {

this.recipeService.fetchRecipes().

map(

(response: Response) => {

const recipes: Recipe[] = response.json();

for (const recipe of recipes) {

if (!recipe['ingredients']) {

recipe['ingredients'] = [];

}

}

return recipes;

}

)

.subscribe(

(recipes: Recipe[]) => {

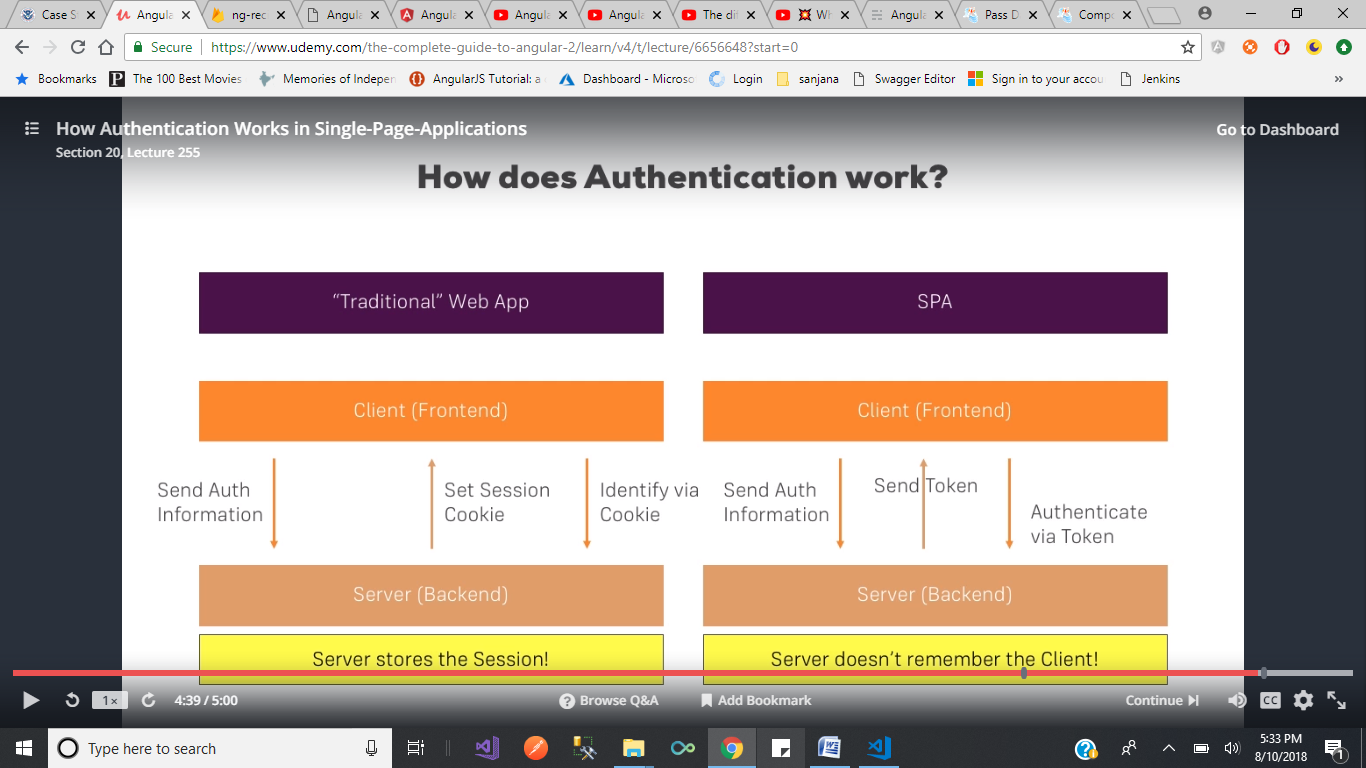
this.recipeService.setRecipes(recipes);

}

);

}

**Authentication:**



**Want to learn about the Token which is exchanged?**

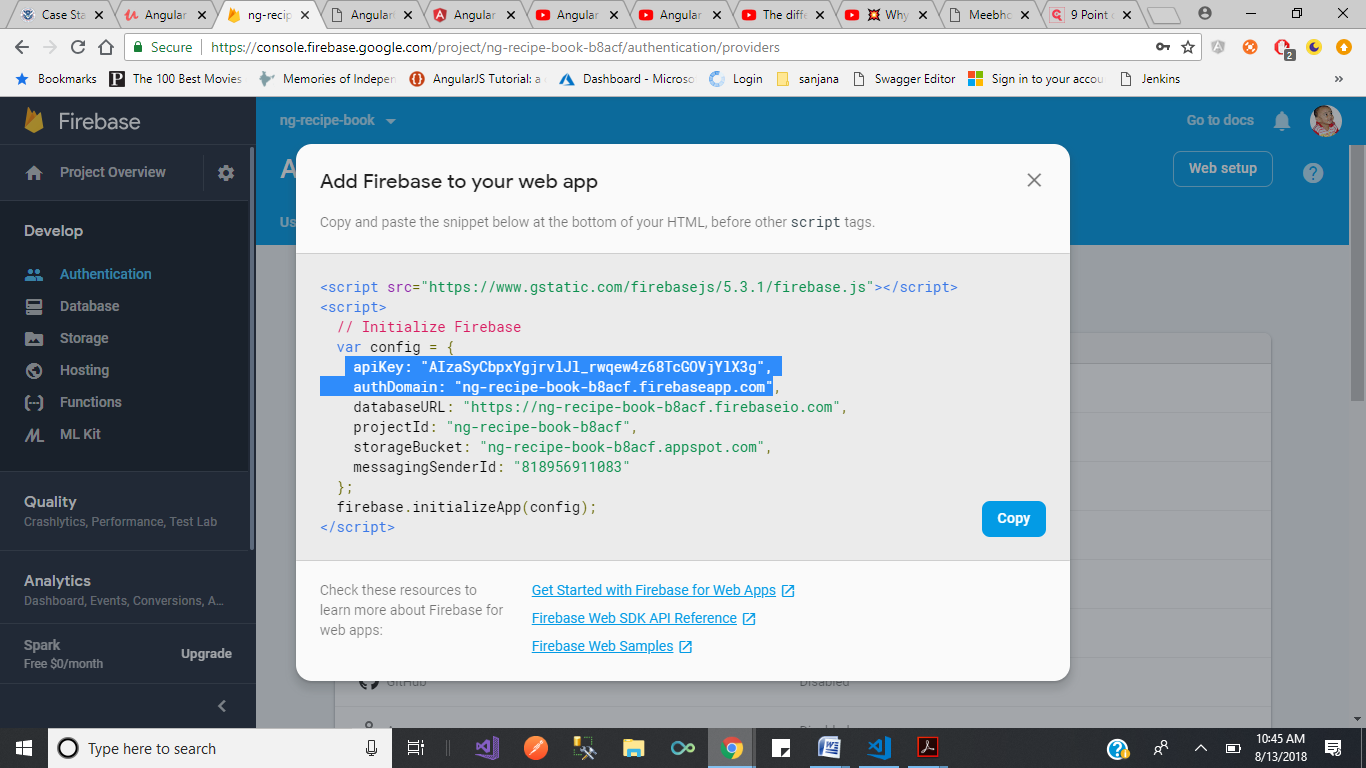
The following page should be helpful: <https://jwt.io/> - specifically, the introduction: <https://jwt.io/introduction/>

To provide authentication

Here we are using Google fire base. In this fire base we should open authentication tab on left side under this we need to enable the “email/password” provider.



Next we if we click on websetup we get setup



We need to install firebase into our system project. ng install –save firebase . create a service

export class AuthService {

signupUser(email: string, password: string) {}

}

In the above picture apikey, authDomain these two we need to mention in app.componet.ts file like below.

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

import \* as firebase from 'firebase' ;

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

ngOnInit() {

firebase.initializeApp({

apiKey: 'AIzaSyCbpxYgjrvlJl\_rwqew4z68TcGOVjYlX3g',

authDomain: 'ng-recipe-book-b8acf.firebaseapp.com'

});

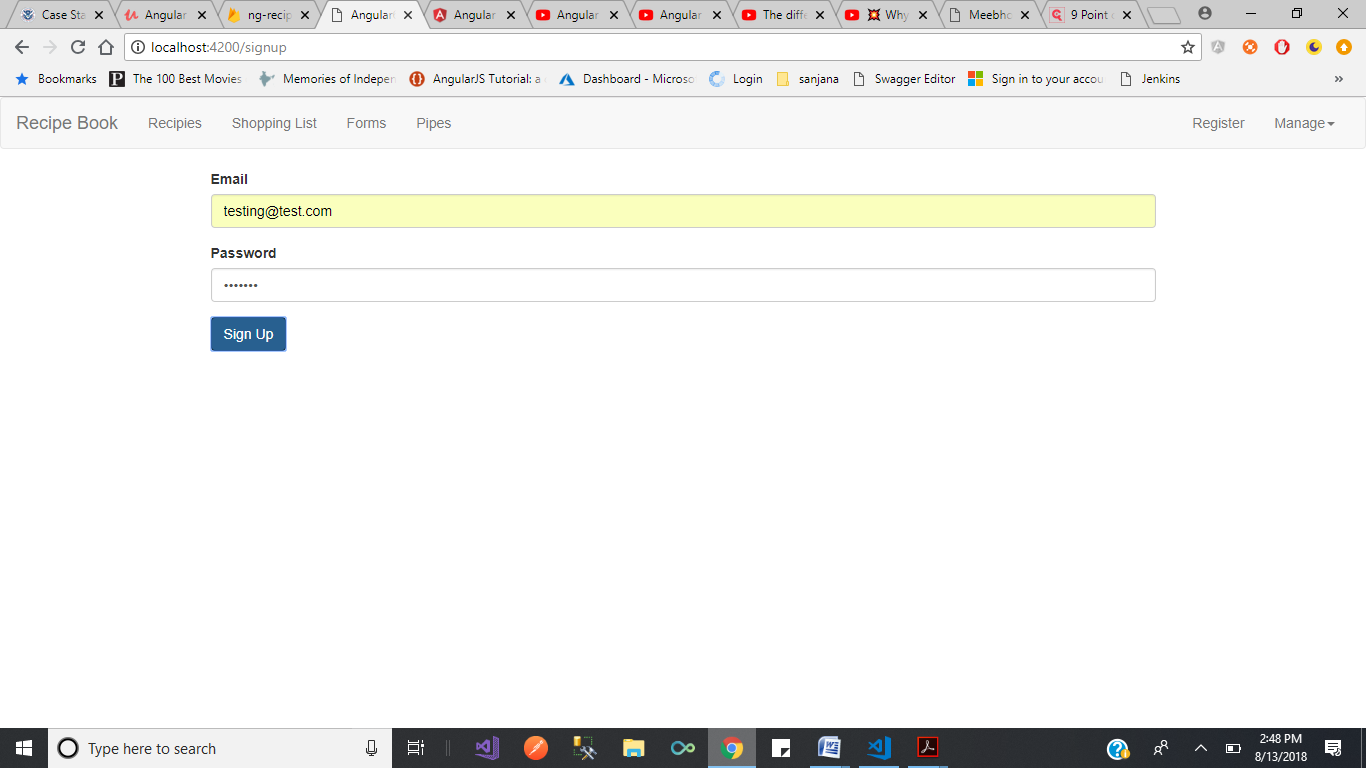
}

}

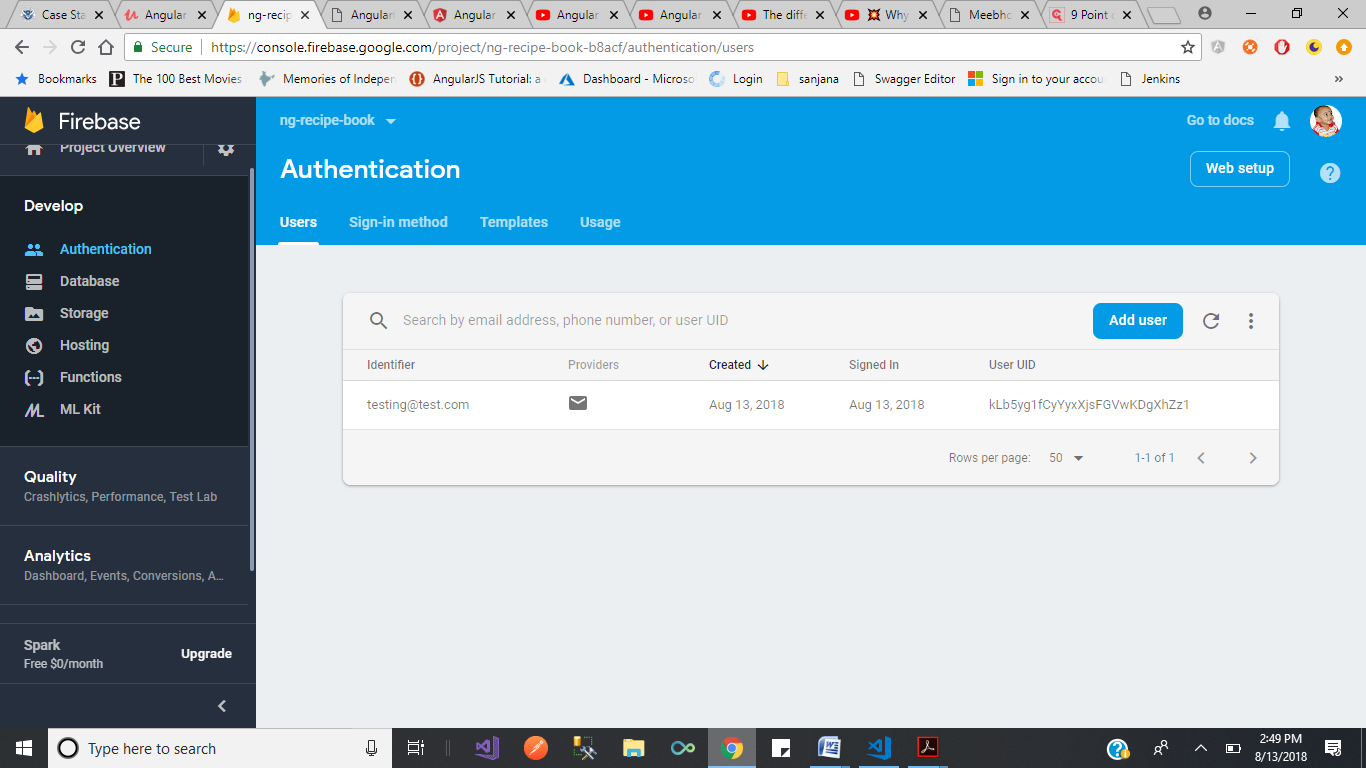
Use getIdToken() instead of getToken() Section 20, Lecture 259

Important note: If you're using**Firebase 5.x** or higher (you can check the package.json  file to find out), you should use getIdToken()  for obtaining the token, NOT getToken()  as shown in the next lectures.

To register the user and save the details into firebase server .



When we click on signup these user details saved into firebase database.



For this we write the code in auth.service.ts

import \* as firebase1 from 'firebase';

export class AuthService {

signupUser(email: string, password: string) {

firebase1.auth().createUserWithEmailAndPassword (email, password)

.catch(

error => console.log(error)

);

}

}

And we added the service reference in app.module.ts file

providers: [ShoppingListService, RecipeService, AuthService],

we call the service in signup form

export class SignupComponent implements OnInit {

constructor(private authService: AuthService) { }

ngOnInit() {

}

onSignUp(form: NgForm) {

const email = form.value.email;

const password = form.value.password;

this.authService.signupUser(email, password);

}

}

After singup we create Signin form

Html

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-offset-1 col-md-offset-2">

<form #f="ngForm" (ngSubmit)="onSignIn(f)">

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" name= "email" ngModel class="form-control">

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" name= "password" ngModel class="form-control">

</div>

<button type="submit" class="btn btn-primary" [disabled]="!f.valid"> Sign In</button>

</form>

</div>

</div>

Signin.Component.ts

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

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

import { AuthService } from '../auth.Service';

@Component({

selector: 'app-signin',

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

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

})

export class SigninComponent implements OnInit {

constructor(private authService: AuthService) { }

ngOnInit() {

}

onSignIn(form: NgForm) {

const email = form.value.email;

const password = form.value.password;

this.authService.signinuser(email, password);

}

}

Auth.service.ts

signinuser(email: string, password: string) {

firebase1.auth().signInWithEmailAndPassword (email, password)

.then(

response => console.log(response)

)

.catch(

error => console.log(error)

);

}

App.-routing.module.ts

{path: 'signin', component: SigninComponent}

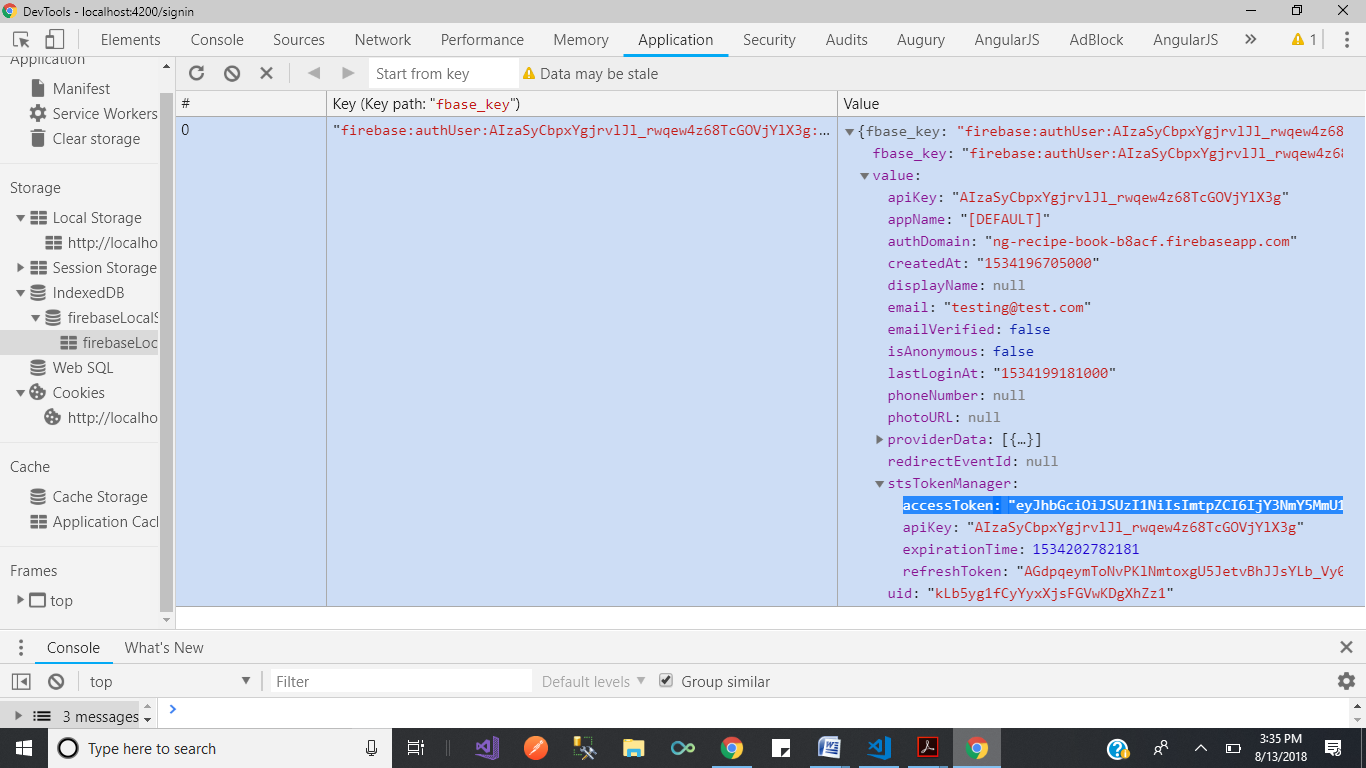
In header.component.html

<ul class="nav navbar-nav navbar-right">

<li ><a routerLink="/signup">Register</a></li>

<li><a routerLink="/signin">LogIn</a></li>

After this when we enter email and password and click sigin then firebase db creates a token to us we can see that token like below.



To authenticate ourself’s , we need to change the rules under database section in firebase database

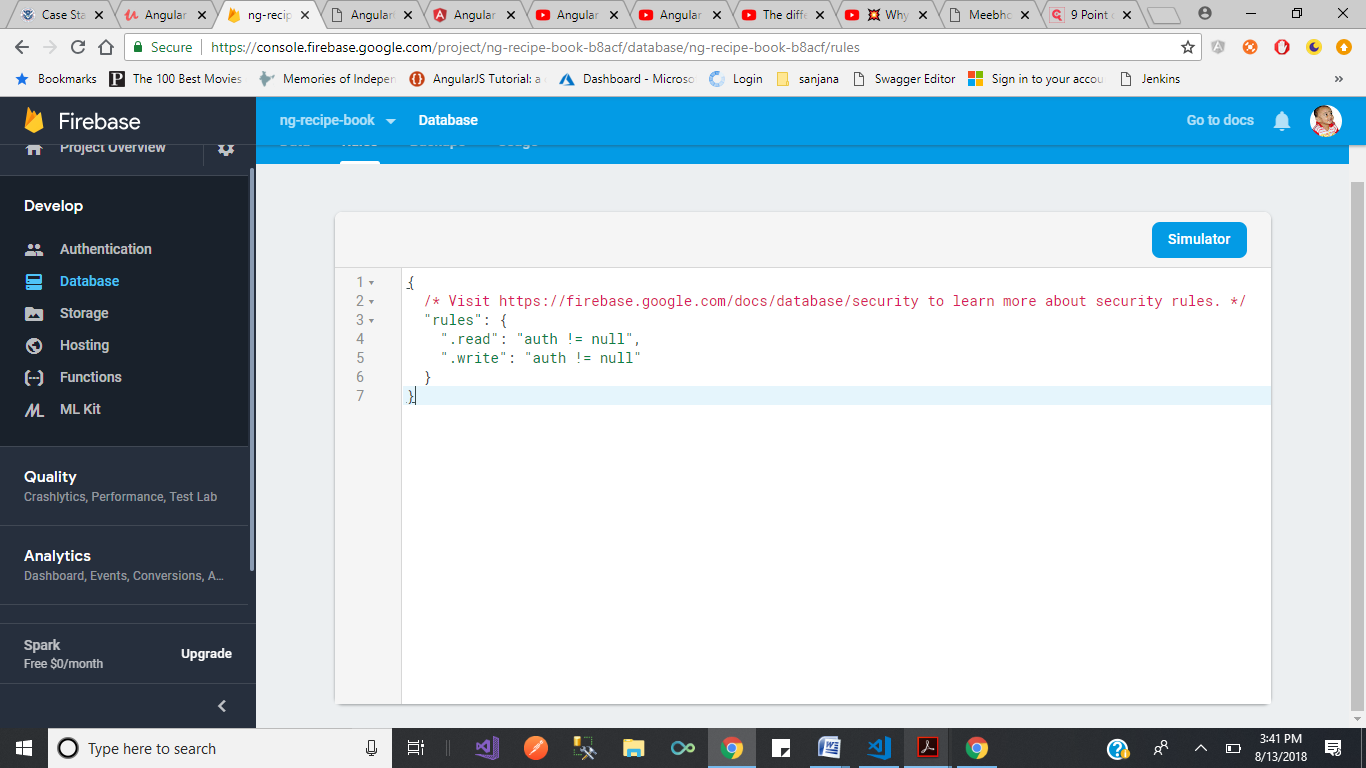
Just like below.

Earlier we mentioned

read= true,

Write=true

Now we changed it to authenticated mode like below



With out login if we try to do any savedata or fechdata under manage dropdown we get “unauthorized access error”

Sending the token (with login request)

For this first we create a method getToken() in authservice.ts file

We access this in recipe.service.ts file by adding service injection

In auth.service.ts

import \* as firebase1 from 'firebase';

export class AuthService {

token: string;

signupUser(email: string, password: string) {

firebase1.auth().createUserWithEmailAndPassword (email, password)

.catch(

error => console.log(error)

);

}

signinuser(email: string, password: string) {

firebase1.auth().signInWithEmailAndPassword (email, password)

.then(

response => {

firebase1.auth().currentUser.getIdToken()

.then(

(token: string) => this.token = token

);

}

)

.catch(

error => console.log(error)

);

}

getToken() {

firebase1.auth().currentUser.getIdToken()

.then(

(token: string) => this.token = token

);

return this.token;

}

}

In recipe.service.ts file

constructor(private slService: ShoppingListService,

private http: Http,

private authService: AuthService) {}

/\*\* above one is for adding ingredients to shopping list and second parameter to save the details into server to pass the request\*/

storeRecipes() {

const token = this.authService.getToken();

// put method over rides all data in the firebase.we should mention 'recipes.json' at the end of url

// (we get it from firebase) if we dont mention recipes.json at the end we will get cross site error.

// we can mention any name after 'https://ng-recipe-book-b8acf.firebaseio.com/' that is over choice.

// but in final we have to specify .json

return this.http.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token, this.getRecipes());

// the above statment returns only observable we have to subscribe this observable in header.component.ts

}

fetchRecipes() {

const token = this.authService.getToken();

return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token);

}

setRecipes(recipes: Recipe[]) {

this.recipes = recipes;

this.recipesChanged.next(this.recipes.slice());

}

Based on the authentication we show the links in the header section:

In authservice.ts file

isAuthenticated() {

return this.token != null;

}

In header.component.ts

We have to inject the auth.service

constructor(private recipeService: RecipeService,

public authService: AuthService) {}

In header.component.html

<ul class="nav navbar-nav navbar-right">

<ng-template [ngIf]="!authService.isAuthenticated()" >

<li ><a routerLink="/signup">Register</a></li>

<li><a routerLink="/signin">LogIn</a></li>

</ng-template>

<li class="dropdown" appDropdown \*ngIf ="authService.isAuthenticated()">

<a style="cursor: pointer" class="dropdown-toggle" role="button">Manage<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a style="cursor: pointer" (click) ="onSaveData()">Save Data</a></li>

<li><a style="cursor: pointer" (click)="onFetchData()" >Fetch Data</a></li>

</ul>

</li>

</ul>

By default we see the all the links except manage dropdown .

Once we login with correct credentials (authenticated) then it shows manage link in header instead of login and signup . on right side of the page.

**Route production and redirection:**

In auth.service.ts file

@Injectable()

export class AuthService {

token: string;

constructor(private router: Router) {}

to use one service in other service we use

@Injectable() decarator

In signinuser method , in the response , we redirect the page after successful login to main page . here in this case it is recipe pagesigninuser(email: string, password: string) {

firebase1.auth().signInWithEmailAndPassword (email, password)

.then(

response => {

this.router.navigate(['/']);

firebase1.auth().currentUser.getIdToken()

.then(

(token: string) => this.token = token

);

}

)

.catch(

error => console.log(error)

);

}

signinuser(email: string, password: string) {

firebase1.auth().signInWithEmailAndPassword (email, password)

.then(

response => {

this.router.navigate(['/']);

firebase1.auth().currentUser.getIdToken()

.then(

(token: string) => this.token = token

);

}

)

.catch(

error => console.log(error)

);

}

Create one service under auth with name auth-guard.service.ts

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

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

import { AuthService } from './auth.Service';

@Injectable()

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService) {}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {

return this.authService.isAuthenticated();

}

}

In app.routing.module.ts

const approuts: Routes = [

/\*\* what ever the paths we want to mention in the page we specify here \*/

{path: '', redirectTo: '/recipes', pathMatch: 'full'},

{path: 'recipes' , component: RecipesComponent , children: [

{path: '', component: RecipeStartComponent},

{path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard]},

{path: ':id', component: RecipesDetailComponent},

{path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard]}

]

},

If the user is authenticated then only we allow the user to create new recipe or edit the recipe details . for that we used this “canActivate” interface

We have to add the service reference in providers of app.module.ts file

providers: [ShoppingListService, RecipeService, AuthService, AuthGuard],

**MODULES:**

In general our application is a couple of components, services, pipes, directives.. we have to clearly tell to angular which component or which directive we are using in the application for that we use module. In general angular contains app.module. we can import all these application elements in this or we can maintain multiple modules in the application.

Here in app.module.ts file

All the imports are related to type script (language related)code not angular related . what ever the import:[] under @ngmodule those are related to angular(means what are the angular module we used in the application we can import here)

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

import { HeaderComponent } from './Header/header.component';

import { RecipesComponent } from './recipes/recipes.component';

import { RecipesListComponent } from './recipes/recipes-list/recipes-list.component';

import { RecipesDetailComponent } from './recipes/recipes-detail/recipes-detail.component';

import { RecipeItemComponent } from './recipes/recipes-list/recipe-item/recipe-item.component';

below are the related to angular code

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

RecipesComponent,

RecipesListComponent,

RecipesDetailComponent,

RecipeItemComponent,

ShoppingListComponent,

ShoppingEditComponent,

RecipesListDirective,

DirectivesComponent,

BasicHighlightDirective,

BetterHighliterDirective,

UnlessDirective,

DropdownDirective,

RecipeStartComponent,

RecipeEditComponent,

FormsDemoComponent,

// ReactiveApproachComponent

ShortenPipe,

FiltersComponent,

RecipesearchPipe,

SignupComponent,

SigninComponent

],

imports: [

BrowserModule,

FormsModule,

ReactiveFormsModule,

HttpModule,

AppRoutingModule

],

providers: [ShoppingListService, RecipeService, AuthService, AuthGuard],

bootstrap: [AppComponent]

})

Declarations part of the module contains components, pipes, directives

imports: part contains what are the other modules we used in the application.

Providers : contains all the services we used in the application.

Bootstrap : contains what is the main(root) component of the application

We can place all the application components , directives ,modules, services in app.module.ts. this is not a bad practice , but we can separate the modules ,based on the requirement. Here in this scenario we can separate recipe related ones in module. After creating that module we can import that module to app.module.ts file.

To create the feature module:

Here I am creating recipes feature module

This is the actual app.module.ts file

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

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

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

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

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

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

import { HeaderComponent } from './Header/header.component';

import { RecipesComponent } from './recipes/recipes.component';

import { RecipesListComponent } from './recipes/recipes-list/recipes-list.component';

import { RecipesDetailComponent } from './recipes/recipes-detail/recipes-detail.component';

import { RecipeItemComponent } from './recipes/recipes-list/recipe-item/recipe-item.component';

import { ShoppingListComponent } from './shopping-list/shopping-list.component';

import { ShoppingEditComponent } from './shopping-list/shopping-edit/shopping-edit.component';

import { RecipesListDirective } from './recipes/recipes-list.directive';

import { DirectivesComponent } from './directives/directives.component';

import { BasicHighlightDirective } from './basic-highlight/basic-highlight.directive';

import { BetterHighliterDirective } from './better-highliter/better-highliter.directive.ts.directive';

import { UnlessDirective } from './structuraldirective/unless.directive.ts.directive';

import { DropdownDirective } from './shared/dropdown.directive.ts.directive';

import { ShoppingListService } from './shopping-list/shopping-list.service';

import { AppRoutingModule } from './app-routing.module';

import { RecipeStartComponent } from './recipes/recipe-start/recipe-start.component';

import { RecipeEditComponent } from './recipes/recipe-edit/recipe-edit.component';

import { FormsDemoComponent } from './forms/forms-demo/forms-demo.component';

import { RecipeService } from './recipes/recipes.service';

import { ShortenPipe } from './shorten.pipe';

import { FiltersComponent } from './pipes/filters/filters.component';

import { RecipesearchPipe } from './pipes/recipesearch.pipe';

import { SignupComponent } from './auth/signup/signup.component';

import { SigninComponent } from './auth/signin/signin.component';

import { AuthService } from './auth/auth.Service';

import { AuthGuard } from './auth/auth-guard.service';

// import { ReactiveApproachComponent } from './forms/reactive-approach/reactive-approach.component';

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

RecipesComponent,

RecipesListComponent,

RecipesDetailComponent,

RecipeItemComponent,

ShoppingListComponent,

ShoppingEditComponent,

RecipesListDirective,

DirectivesComponent,

BasicHighlightDirective,

BetterHighliterDirective,

UnlessDirective,

DropdownDirective,

RecipeStartComponent,

RecipeEditComponent,

FormsDemoComponent,

// ReactiveApproachComponent

ShortenPipe,

FiltersComponent,

RecipesearchPipe,

SignupComponent,

SigninComponent

],

imports: [

BrowserModule,

FormsModule,

ReactiveFormsModule,

HttpModule,

AppRoutingModule

],

providers: [ShoppingListService, RecipeService, AuthService, AuthGuard],

bootstrap: [AppComponent]

})

export class AppModule { }

now I am going to simplify this by creating recipes feature module.

Here in this case I created one file “recipes.module.ts”

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

import { CommonModule } from '@angular/common';

import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';

import { RecipesComponent } from './recipes.component';

import { RecipeItemComponent } from './recipes-list/recipe-item/recipe-item.component';

import { RecipesListComponent } from './recipes-list/recipes-list.component';

import { RecipeStartComponent } from './recipe-start/recipe-start.component';

import { RecipesListDirective } from './recipes-list.directive';

import { RecipesDetailComponent } from './recipes-detail/recipes-detail.component';

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

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

@NgModule({

declarations: [

RecipeEditComponent,

RecipesComponent,

RecipeItemComponent,

RecipesListComponent,

RecipeStartComponent,

RecipesDetailComponent,

RecipesListDirective

],

imports: [

CommonModule, // it gives access to the common directives(ngclass,ngif , ngfor...) in the application

ReactiveFormsModule // we used this at only in recipes forms

]

})

export class RecipesModule { }

// note: we cannot duplicate the same module in formsmodule and appmodule. it should be only one. we must not dulicate our declaration.

// here in this case we use recipecomponet here we cannot declare the same component in app.module.ts file.

We added ever thing relarted to recipes into this file and deleted these imports from app.module.ts file

And added this recipe.module.ts file in the imports of app.module.ts

But we get the error. In the above code we didn’t add any routing related thins in the imports so we get that error . and dropdown also don’t work in recipe edit file , since we don’t add any directive related to that . so for this we have to create one routing module just like app-routing.module.ts . and specify the routes related to that recipes in recipes routing file.

Recipes.routing.ts

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

import { RouterModule, Routes } from '@angular/router';

import { AuthGuard } from '../auth/auth-guard.service';

import { RecipesComponent } from './recipes.component';

import { RecipeStartComponent } from '../recipes/recipe-start/recipe-start.component';

import { RecipesDetailComponent } from '../recipes/recipes-detail/recipes-detail.component';

import { RecipeEditComponent } from '../recipes/recipe-edit/recipe-edit.component';

const recipesRoutes: Routes = [

{path: 'recipes' , component: RecipesComponent , children: [

{path: '', component: RecipeStartComponent},

{path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard]},

{path: ':id', component: RecipesDetailComponent},

{path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard]}

]

},

];

@NgModule({

imports: [

RouterModule.forChild(recipesRoutes)

],

exports: [RouterModule]

})

After this we have to import this recipes.module.ts into app.module.ts

imports: [

BrowserModule, // here we dont have a commonmodule. since this browsermodule contains all the featers of

// commonmodule and some extra featers to need to start the application.

// so we use browser module here and commonmodule in othermoduls.

RecipesModule,

FormsModule,

HttpModule,

AppRoutingModule

],

That to it should place on top of AppRoutingModule

Feature Module:

A [feature module](https://angular.io/guide/ngmodule#feature-modules) is an ordinary Angular module for all intents and purposes, except the fact that isn’t the root module. Basically - it’s just an additional class with the @NgModule decorator and registered metadata. The feature module partitions areas of the application and collaborates with the root module (and with further feature modules).

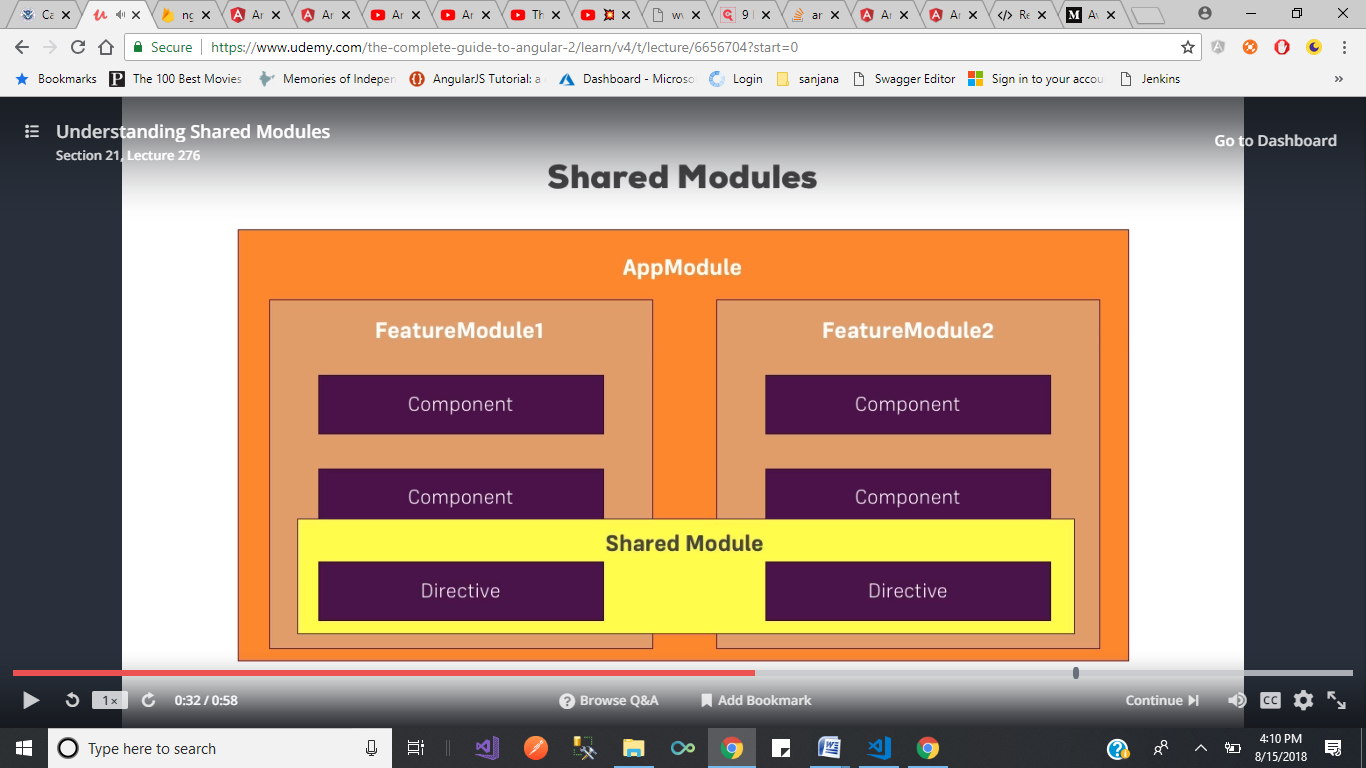
The main aim for feature modules is delimiting the functionality that focuses on particular internal business inside a dedicated module, in order to achieve modularity. In addition, it restricts the responsibilities of the root module and assists to keep it thin. Another advantage - it enables to define multiple directives with an identical selector, which means avoiding from directive conflicts.

As your app grows, you can organize code relevant for a specific feature. This helps apply clear boundaries for features. With feature modules, you can keep code related to a specific functionality or feature separate from other code. Delineating areas of your app helps with collaboration between developers and teams, separating directives, and managing the size of the root module.

There are five general categories of feature modules which tend to fall into the following groups:

* Domain feature modules.
* Routed feature modules.
* Routing modules.
* Service feature modules.
* Widget feature modules.

**Shared Module:**

****

In the shared folder create one module “shared.module.ts”

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

import { DropdownDirective } from './dropdown.directive.ts.directive';

import { CommonModule } from '@angular/common';

@NgModule({

declarations: [

DropdownDirective

],

exports: [

CommonModule,

DropdownDirective

]

})

export class Sharedmodule {

}

In app.module we have to import

imports: [

BrowserModule, // here we dont have a commonmodule. since this browsermodule contains all the featers of

// commonmodule and some extra featers to need to start the application.

// so we use browser module here and commonmodule in othermoduls.

RecipesModule,

FormsModule,

HttpModule,

AppRoutingModule,

Sharedmodule

],

In recipes module also we have to import ( since we are using dropdown in recipes , and header (till now we placed all the header related information in app.module.ts file, so we added shared module in app.module.ts)

@NgModule({

declarations: [

RecipeEditComponent,

RecipesComponent,

RecipeItemComponent,

RecipesListComponent,

RecipeStartComponent,

RecipesDetailComponent,

RecipesListDirective,

ShortenPipe,

RecipesearchPipe

],

imports: [

CommonModule, // it gives access to the common directives(ngclass,ngif , ngfor...) in the application

ReactiveFormsModule, // we used this at only in recipes forms

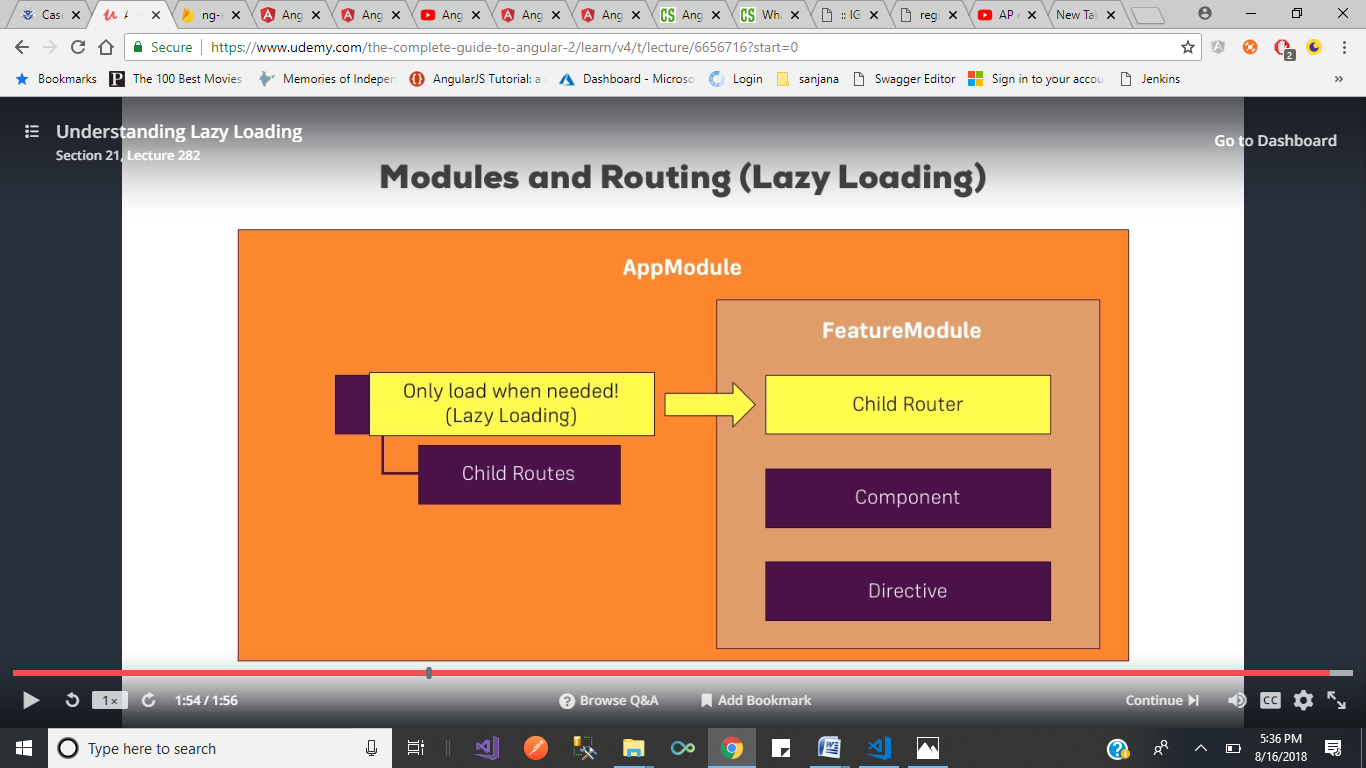
RecipesRoutingModule,

Sharedmodule

]

})

Modules and routing(Lazyloading)



Till now we did eager loading.

By providing the routes like {path: ‘recipes’, component: recipescomponent}

Now in lazy loding we have to specify the path like this

{path: ‘recipes’, loadchildern:’./recipes/recipes.module#RecipesModule’}

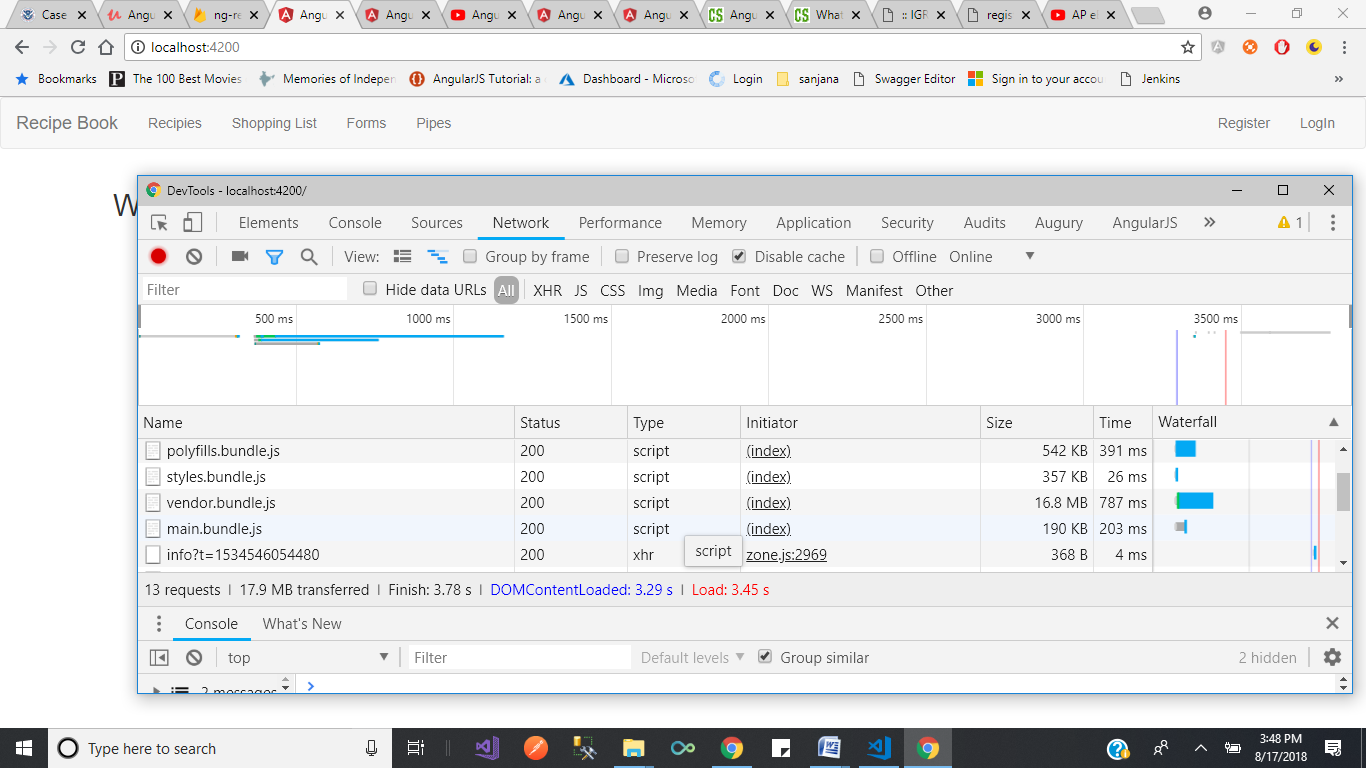
If we import the modules in the appcomponent then those components load eagerly

But the above path doesnot work when we use the httpclient interceptor so we can change it to

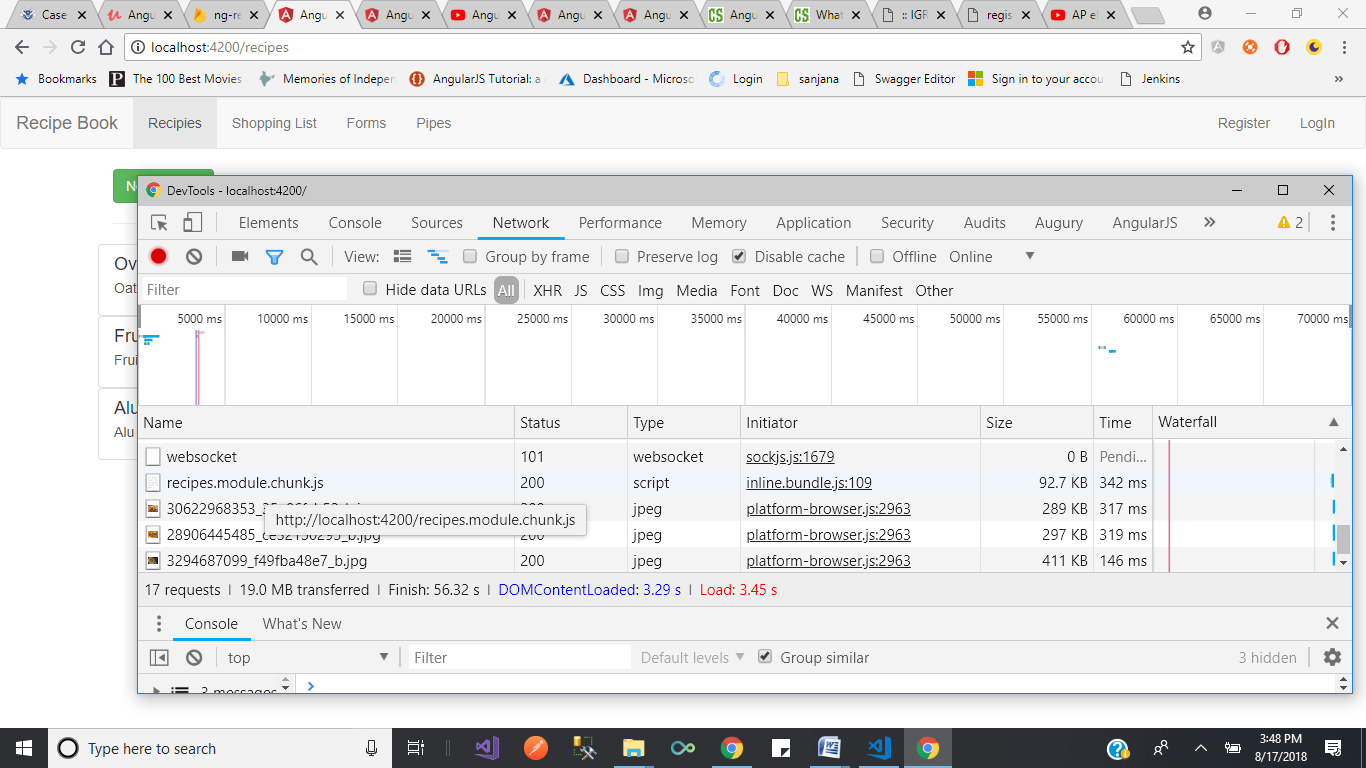
{path: 'recipes', loadChildren: () => RecipesModule},

// {path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule'},

// for lazy loding of recipes we commented recipes module in app.module and added the path above

****

**When we click on recipesbook we don’t get the recipes here under network**

****

**When we click on recipes link we got these chunk.js, recipes details.**

Protecting Lazy Loaded Routes with canLoad

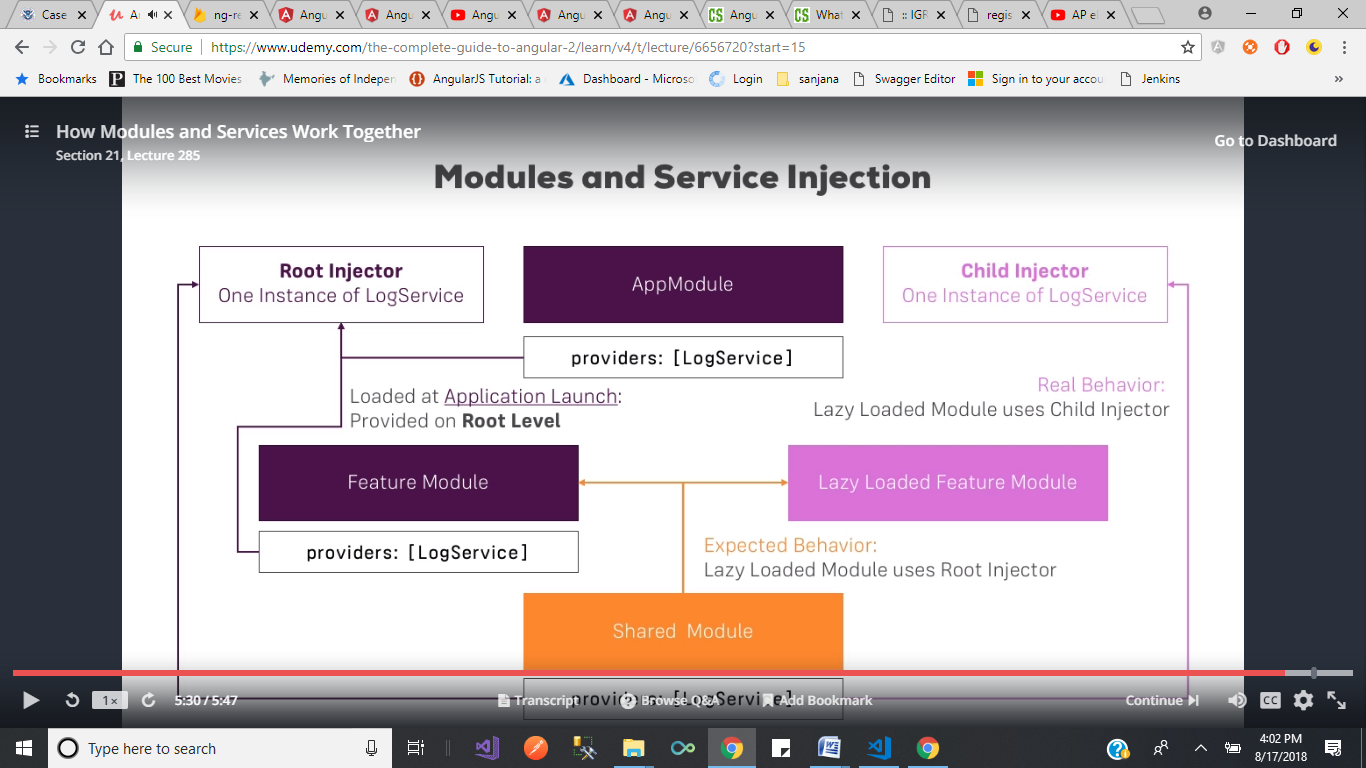
What if you want to use route protection (canActivate  to be precise) on lazily loaded routes?

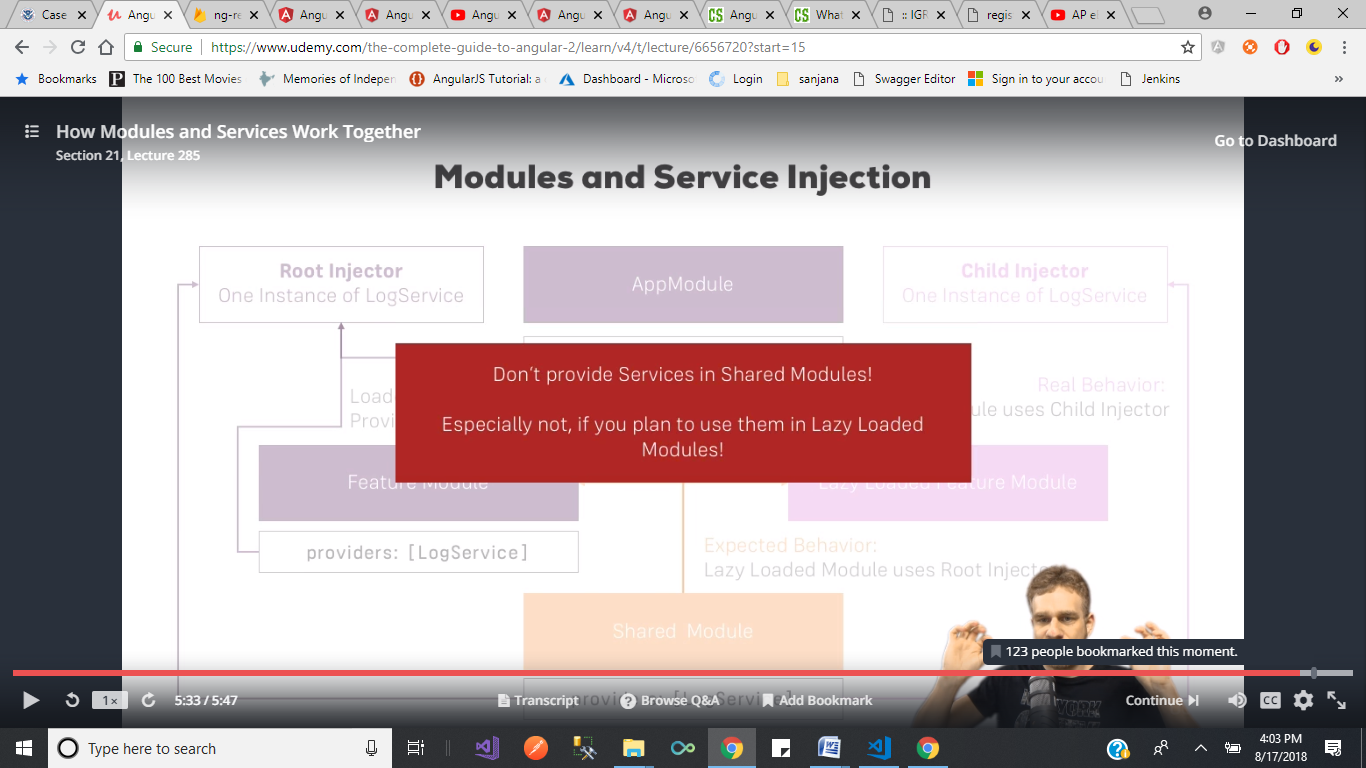
You can add canActivate to the lazy loaded routes but that of course means, that you might load code which in the end can't get accessed anyways. It would be better to check that BEFORE loading the code.

You can enforce this behavior by adding the canLoad  guard to the route which points to the lazily loaded module:

{ path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule', canLoad: [AuthGuard] }

In this example, the AuthGuard  should implement the [CanLoad interface](https://angular.io/docs/ts/latest/api/router/index/CanLoad-interface.html).

****

****

**Deployment:**

To deploy the application in the server we need to build the application by using below command

Ng build –prod –aot

**Httpclientmodule:**

From angular 4 ‘httpclient’ came into picture in the place of ‘http’ to get and put the details from the backend server. Instead of this httpclient we can use ‘http’ also . some more extra features are added to the ‘httpclient’. So I replaced http with httpclient.

To get .map method we need to import below one in recipes.service.cs

import 'rxjs/add/operator/map';

import { HttpClient } from '@angular/common/http';

constructor(private slService: ShoppingListService,

// private http: Http,

private httpclient: HttpClient,

private authService: AuthService) {}

storeRecipes() {

const token = this.authService.getToken();

// put method over rides all data in the firebase.we should mention 'recipes.json' at the end of url

// (we get it from firebase) if we dont mention recipes.json at the end we will get cross site error.

// we can mention any name after 'https://ng-recipe-book-b8acf.firebaseio.com/' that is over choice.

// but in final we have to specify .json

return this.httpclient.put

('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

this.getRecipes());

// return this.http.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token, this.getRecipes());

// the above statment returns only observable we have to subscribe this observable in header.component.ts

}

fetchRecipes() {

const token = this.authService.getToken();

// return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token);

// this.httpclient.get<Recipe[]>('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token).

this.httpclient.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

{ observe: 'body', // if we mention 'response' instead of 'body' we get all the resonse

responseType: 'json'}) // here we can mention 'text','blob'(to download the file),'arraybuffer'(to buffer the data)

.map(

(recipes: Recipe[]) => {

console.log(recipes);

for (const recipe of recipes) {

if (!recipe['ingredients']) {

recipe['ingredients'] = [];

}

}

return recipes;

}).

subscribe (

(recipes: Recipe[]) => {

// const recipes: Recipe[] = response.json();

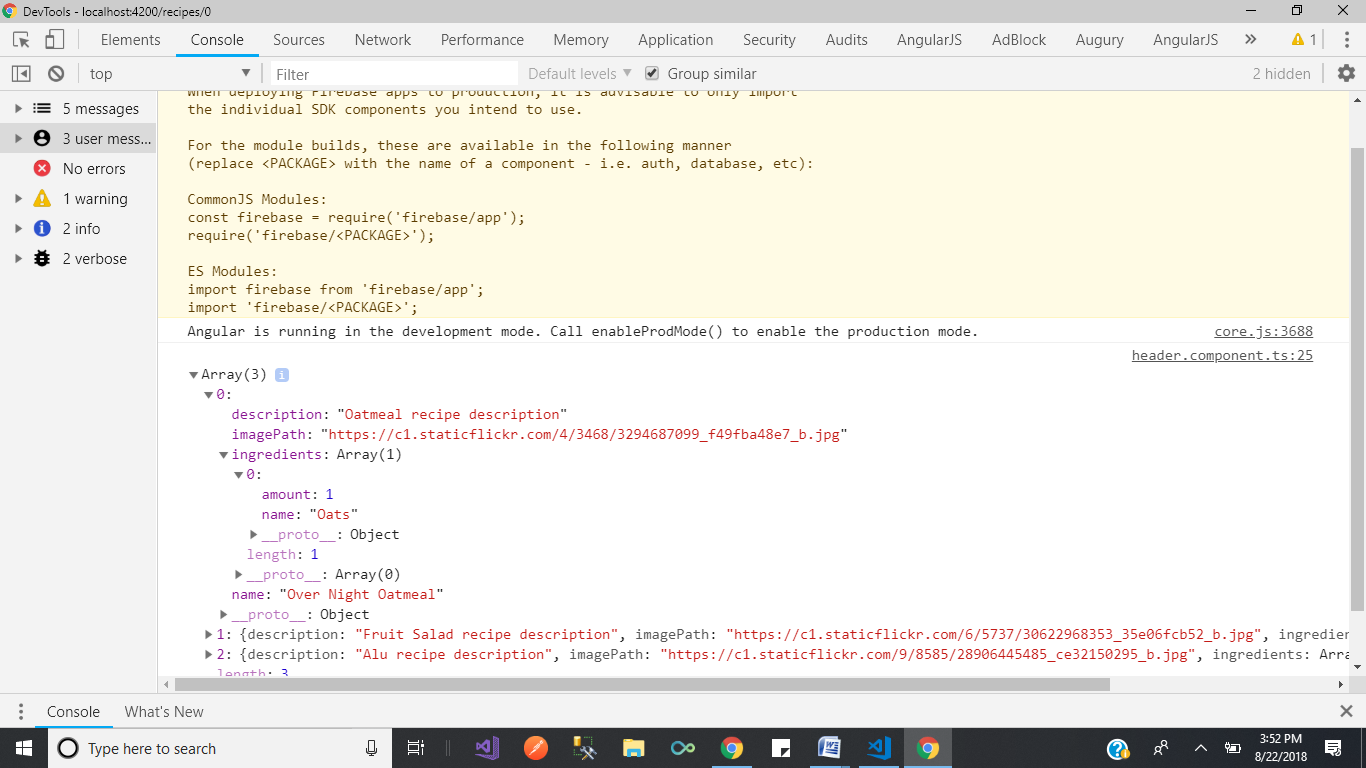
this.setRecipes(recipes);

},

(error) => console.log(error)

);

Since I mentioned console.log(recipes) I got the recipes log in console section as shown below while fetching the details from the firebase server.



Requesting Events:-

While getting the details from backendserver

In recipes.service.ts

storeRecipes() {

const token = this.authService.getToken();

// put method over rides all data in the firebase.we should mention 'recipes.json' at the end of url

// (we get it from firebase) if we dont mention recipes.json at the end we will get cross site error.

// we can mention any name after 'https://ng-recipe-book-b8acf.firebaseio.com/' that is over choice.

// but in final we have to specify .json

return this.httpclient.put

('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

this.getRecipes(), {observe: 'events'});

// return this.http.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token, this.getRecipes());

// the above statment returns only observable we have to subscribe this observable in header.component.ts

}

In header.componet.ts file

import { HttpEvent } from '@angular/common/http';

onSaveData() {

this.recipeService.storeRecipes()

.subscribe(

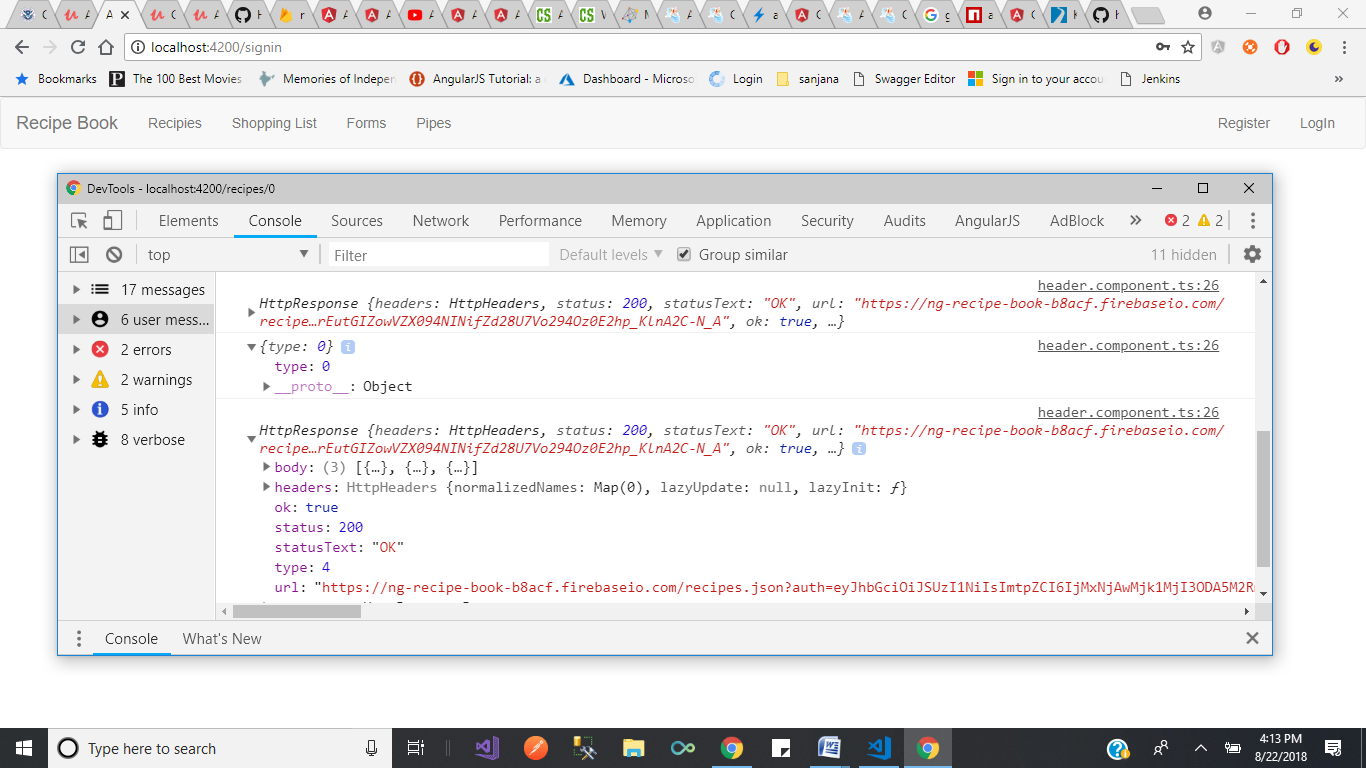
(response: HttpEvent<object>) => {

console.log(response);

}, (error) => console.log(error)

);

Because of this events we get the response like this in console



Type:0 and type:4 responses are there type4 is detailed one

* For suppose if we delete all the ingredients in’ edit recipes ‘ and saved the data to backend server’firebase’ in that scenario , firebase completely deleting the ingredients from the backend . but technically that is wrong to achive the ingredient column in this we use .map method

fetchRecipes() {

const token = this.authService.getToken();

// return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token);

// this.httpclient.get<Recipe[]>('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token).

this.httpclient.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

{ observe: 'body', // if we mention 'response' instead of 'body' we get all the resonse

responseType: 'json'}) // here we can mention 'text','blob'(to download the file),'arraybuffer'(to buffer the data)

.map(

(recipes: Recipe[]) => {

console.log(recipes);

for (const recipe of recipes) {

if (!recipe['ingredients']) {

console.log(recipe);

recipe['ingredients'] = [];

// for suppose if we delete all the ingredients from the recipes. that time in the firebase we dont get any ingredients.

// because of that reason,when the ingrdiens are empty.we added recipe['ingredients'] = [];for ingredient column in firebase

}

}

return recipes;

})

.subscribe (

(recipes: Recipe[]) => {

// const recipes: Recipe[] = response.json(); // for http we need this

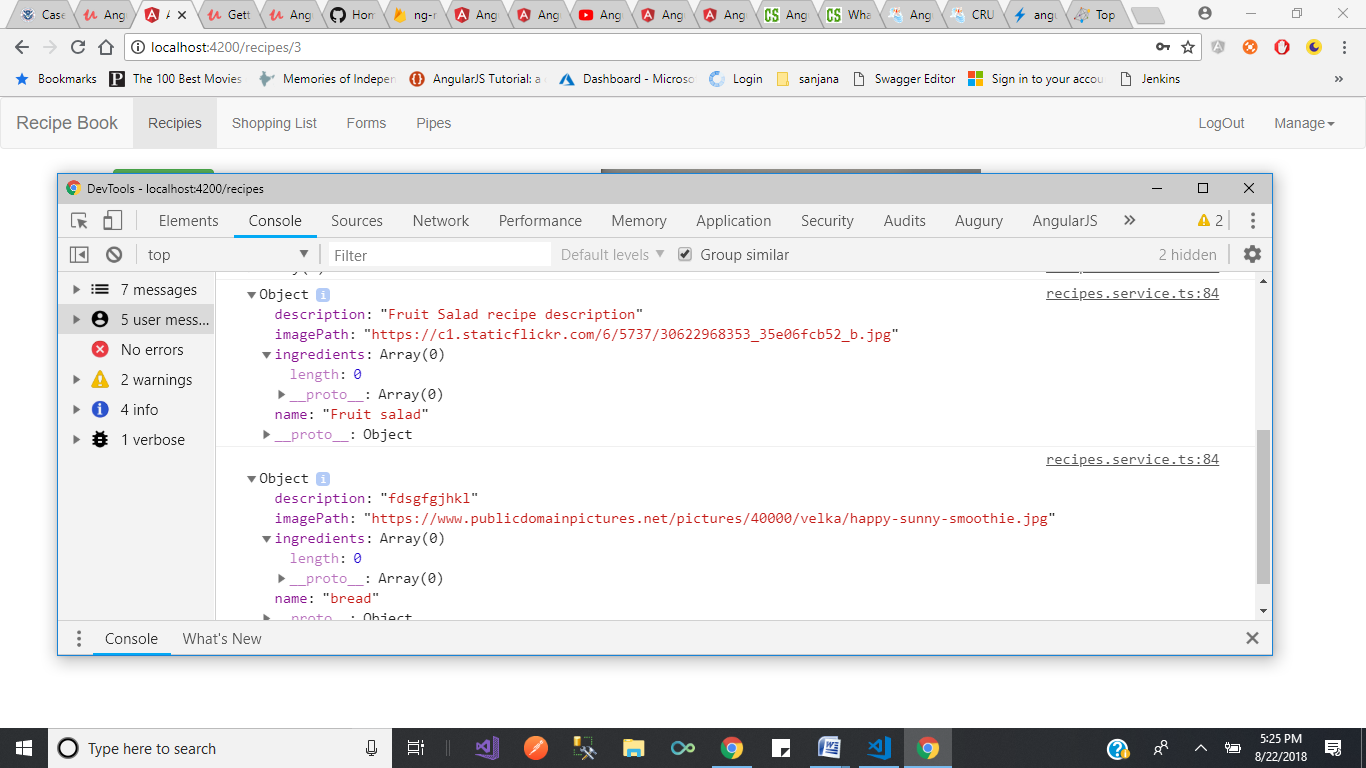
this.setRecipes(recipes);

},

(error) => console.log(error)

);

Since we write the log of recipes we can seet he ingredient column with empty array like in below screen



We can add headers and params in httpclient get and put methods like below. We use params to pass the token in the case of authentication checking while getting and put the details to backend server.

Earlier we passed the parameter in the url like this in recipes.service.ts file

this.httpclient.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

now we can pass this through params

storeRecipes() {

// const header = new HttpHeaders().set('Authorization', 'Bearer afdklasflaldf');

const token = this.authService.getToken();

return this.httpclient.put

('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

this.getRecipes(),

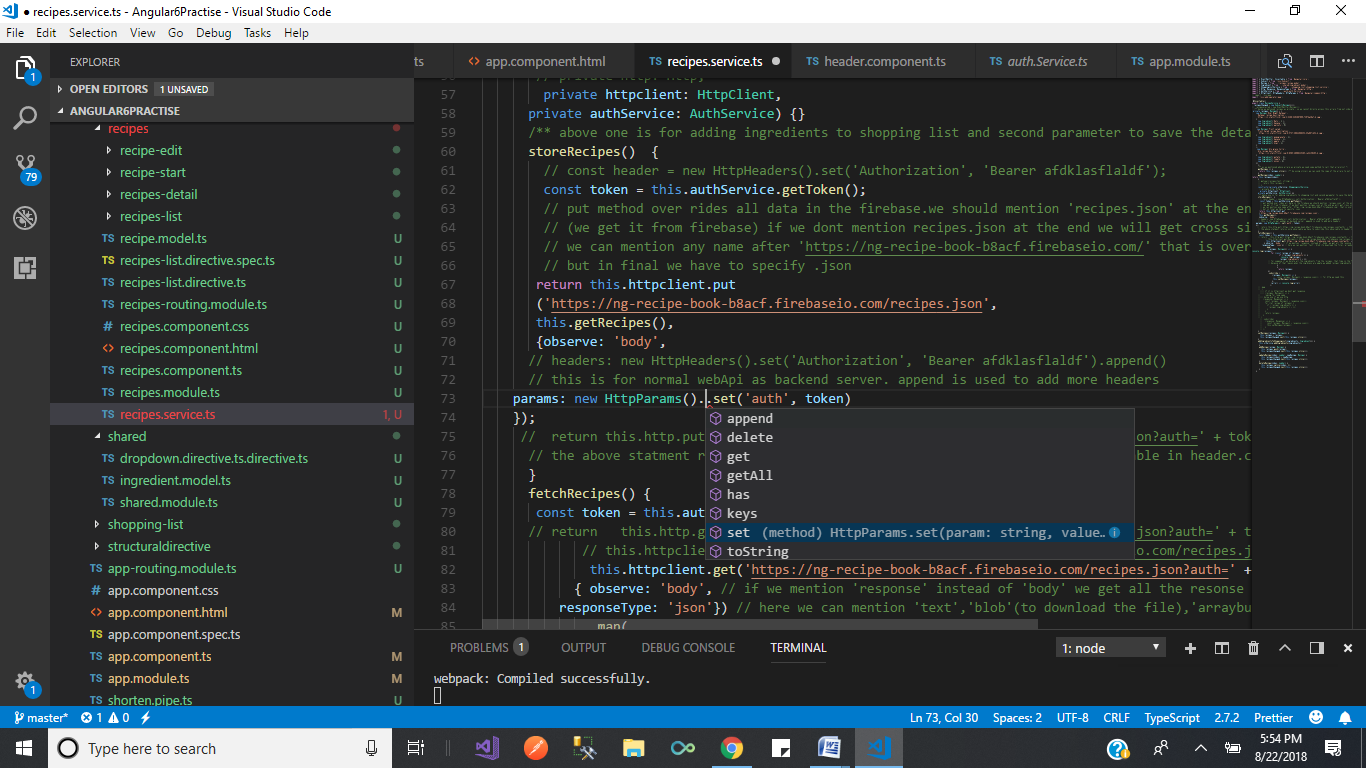
{observe: 'body',

params: new HttpParams().set('auth', token)

});

Here we have set, append,delete,get,getall .. we have . set will change all the values.

Append adds the values to the existing one



**Progress:**

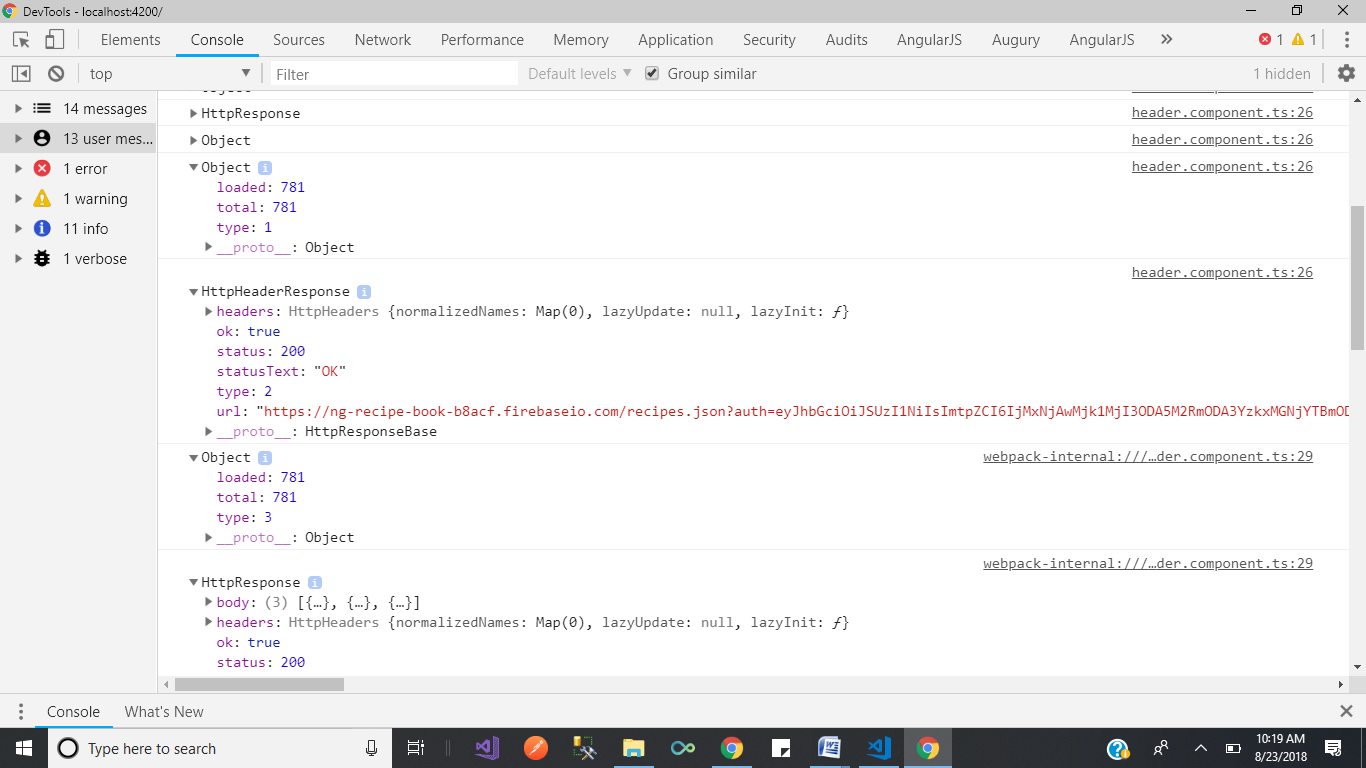
Uploading and downloading files using httpclient

In recipe.service.ts

const req = new HttpRequest('PUT', 'https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

this.getRecipes(), {reportProgress: true, params: new HttpParams().set('auth', token)});

return this.httpclient.request(req);



We can see the uploaded and downloaded response here

**Interceptors:**

Instead of passing query params related to authentication with each request to backend server(current we are doing manually with the request)

params: new HttpParams().set('auth', token)

for put request

const req = new HttpRequest('PUT', 'https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

this.getRecipes(), {reportProgress: true, params: new HttpParams().set('auth', token)});

return this.httpclient.request(req);

for get request

fetchRecipes() {

const token = this.authService.getToken();

// return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token);

// this.httpclient.get<Recipe[]>('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token).

this.httpclient.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token,

{ observe: 'body', // if we mention 'response' instead of 'body' we get all the resonse

responseType: 'json'})

we can do that in one interceptor and use that one while sending the request(identify every outgoing request and manipulate that with authentication token )

Many of us know the concept of middleware. It acts as the layer before sending a request and after receiving a response. What if we want to do the same in Angular? Now, the question arises what will be the cases where we will be needing this middleware.

Let’s take an example of adding the headers before sending a request or logging all the responses to the activities we do. Imagine, we are adding the headers before each call to the server or we are logging the data after every Response or more specifically, handling all the HTTP errors at one place. How we can achieve the same using Angular? Interceptors is the answer for that.

**Angular Interceptors**

Interceptors are the mechanism where we can operate on the outgoing request and the incoming response.

It is the same concept of middleware that we use in the ASP.NET Core where we have these features. Interceptors are the layer between our Angular application and the back-end services. Whenever the request is made from the application, the interceptor catches the request, transforms it, and passes to the back end. The same happens with the response; whenever it receives the response, we can make changes to the response and use them in the application.

**Setting up Interceptors**

To set up interceptor, we need a class which will be injectable class and implementing the HttpInterceptor. When we implement this Interface, then we have a method called intercept which has the body like below.

*intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {}*

It has two parameters "req" and "next" which are HttpRequest and HttpHandler respectively.

For more info

<https://www.c-sharpcorner.com/article/angular-5-http-client-interceptors/>

here in this project we created on interceptor with name auth.interceptor.ts in shared folder.

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';

import { Observable } from 'rxjs/observable';

export class AuthInterceptor implements HttpInterceptor {

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

console.log('intercepted!', req);

return next.handle(req);

}

}

We need to tell the angular to do this through interceptor.

In core.module.ts file

providers: [

ShoppingListService,

RecipeService,

AuthService,

{provide: HTTP\_INTERCEPTORS, useClass: AuthInterceptor, multi: true}

]

We removed token parameters in recipes.service.ts file

storeRecipes() {

// const header = new HttpHeaders().set('Authorization', 'Bearer afdklasflaldf');

// const token = this.authService.getToken();

// put method over rides all data in the firebase.we should mention 'recipes.json' at the end of url

// (we get it from firebase) if we dont mention recipes.json at the end we will get cross site error.

// we can mention any name after 'https://ng-recipe-book-b8acf.firebaseio.com/' that is over choice.

// but in final we have to specify .json

/\* return this.httpclient.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

this.getRecipes(),

{observe: 'body',

// headers: new HttpHeaders().set('Authorization', 'Bearer afdklasflaldf').append()

// this is for normal webApi as backend server. append is used to add more headers

params: new HttpParams().set('auth', token)

}); \*/

const req = new HttpRequest('PUT', 'https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

this.getRecipes(), {reportProgress: true});

// this.getRecipes(), {reportProgress: true, params: new HttpParams().set('auth', token)});

return this.httpclient.request(req);

/\* return this.http.put('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token, this.getRecipes());

// the above statment returns only observable we have to subscribe this observable in header.component.ts \*/

}

fetchRecipes() {

// const token = this.authService.getToken();

// return this.http.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token);

// this.httpclient.get<Recipe[]>('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json?auth=' + token).

this.httpclient.get('https://ng-recipe-book-b8acf.firebaseio.com/recipes.json',

{ observe: 'body', // if we mention 'response' instead of 'body' we get all the resonse

responseType: 'json'}) // here we can mention 'text','blob'(to download the file),'arraybuffer'(to buffer the data)

.map(

(recipes: Recipe[]) => {

console.log(recipes);

for (const recipe of recipes) {

if (!recipe['ingredients']) {

console.log(recipe);

recipe['ingredients'] = [];

// for suppose if we delete all the ingredients from the recipes. that time in the firebase we dont get any ingredients.

// because of that reason,when the ingrdiens are empty.we added recipe['ingredients'] = [];for ingredient column in firebase

}

}

return recipes;

})

.subscribe (

(recipes: Recipe[]) => {

// const recipes: Recipe[] = response.json(); // for http we need this

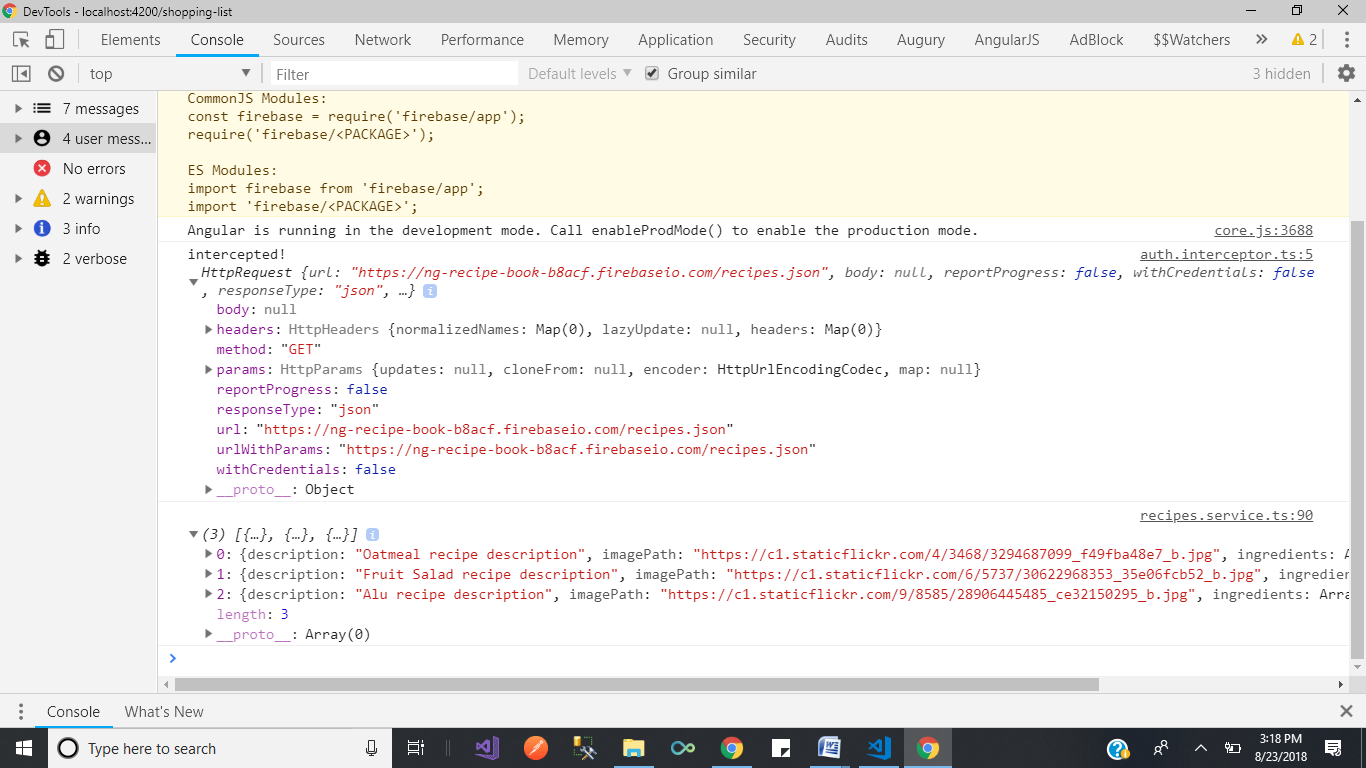
this.setRecipes(recipes);

},

(error) => console.log(error)

);

If we run the application we get the log like below



By default requests are immutable but we can do this with clone

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

import { AuthService } from '../auth/auth.Service';

@Injectable()

export class AuthInterceptor implements HttpInterceptor {

constructor(private authService: AuthService) {}

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

console.log('intercepted!', req);

const copiedreq = req.clone({params: req.params.set('auth', this.authService.getToken())});

// return next.handle(req); // we r continuing the request to the backend server through 'next'

return next.handle(copiedreq);

}

}

We can modify the response what we get from the backend server by using interceptor

Logging.interceptor.ts

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';

import {Observable } from 'rxjs/Observable';

export class LoggingInterceptor implements HttpInterceptor {

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

return next.handle(req).do(

event => {

console.log('logging Interceptor', event);

}

);

}

}

In core.module.ts file

providers: [

ShoppingListService,

RecipeService,

AuthService,

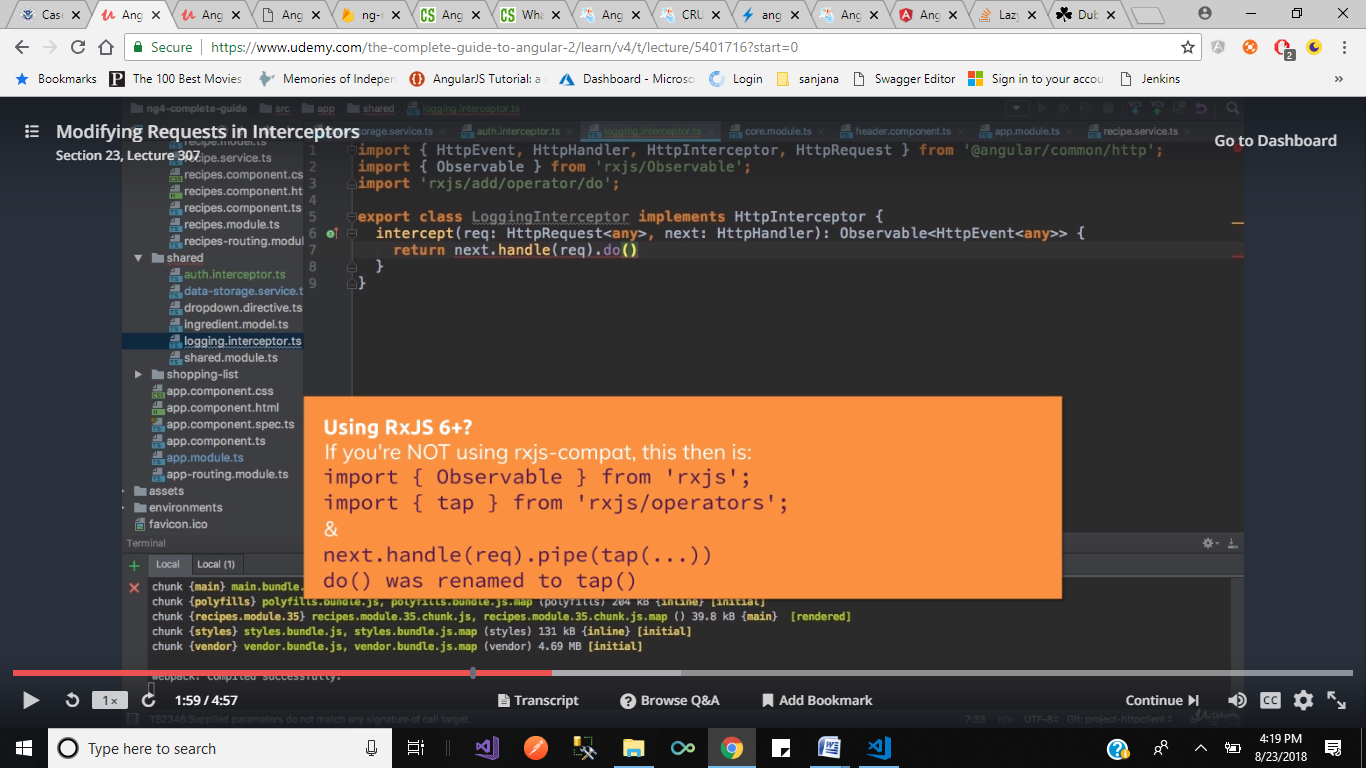
AuthGuard,

{provide: HTTP\_INTERCEPTORS, useClass: AuthInterceptor, multi: true},

{provide: HTTP\_INTERCEPTORS, useClass: LoggingInterceptor, multi: true}

// based on the order we set here interceptor will execute

]



<https://angular.io/guide/http>

**State Management:**

Here till now we used services .

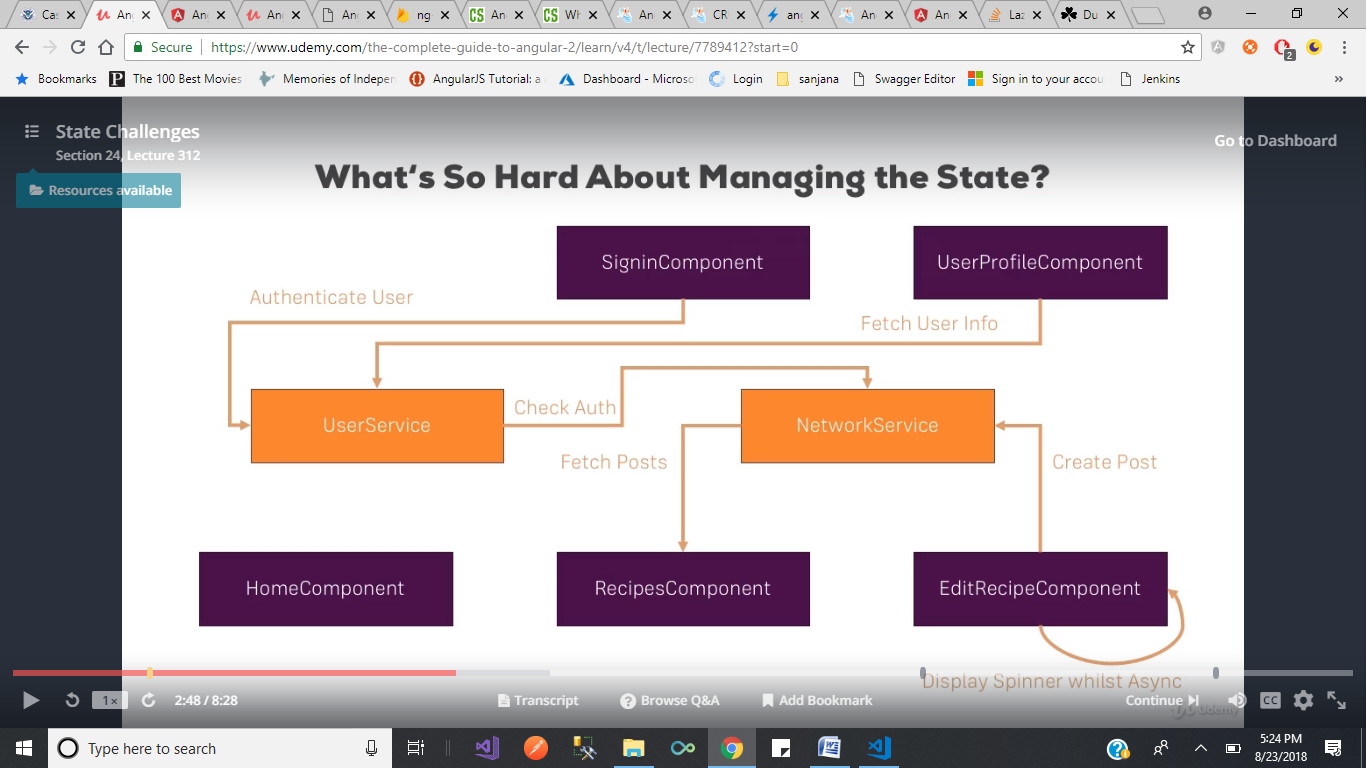
To inform to the application about state change we used ‘subject’ under service.

If our application is less we can go this process of using services in application.

@Injectable()

export class RecipeService {

recipesChanged = new Subject<Recipe[]>();



But in this approach it’s difficult to identify where state got changed in the application. If the application size grow. For maintaining the service, subject driven approach is difficult.

So we use below process

