serLearning cources: Academind.com/learn

Angular Form validation

<https://angular.io/api/forms/Validators>

1. Installations

To install Bootstrap: npm install –save bootstrap@3

Run Project in different port: ng serve --port 5050

1. Create Components

Components can be created manually / wth CLI

1. Data Binding

Data Binding is communication of typescript Code and HTML

1. One way Data Binding
   * 1. Output Data
        1. String Interpolation **{{ number }}, {{ getServerStatus() }}**
        2. Property binding
           1. **[disabled]="!allowNewServer",**
           2. **[ngClass]=”{odd: number % 2 !==0 }”**
           3. **[ngStyle]=”{backgroundColor: number %2 !== 0 ? ‘yellow’ : ‘transparent’}”**
     2. React to (user) Events
        1. (click)="myFunction()"
        2. (dblclick)="myFunction()"
        3. (submit)="myFunction()"
        4. (blur)="myFunction()"
        5. (focus)="myFunction()"
        6. (scroll)="myFunction()"
        7. (cut)="myFunction()"
        8. (copy)="myFunction()"
        9. (paste)="myFunction()"
        10. (keyup)="myFunction()"
        11. (keypress)="myFunction()"
        12. (keydown)="myFunction()"
        13. (mouseup)="myFunction()"
        14. (mousedown)="myFunction()"
        15. (mouseenter)="myFunction()"
        16. (drag)="myFunction()"
        17. (drop)="myFunction()"
        18. (dragover)="myFunction()"
2. Two way Data Binding
   1. **[(**ngModel**)]=”data”**import { FormsModule } from '@angular/forms'
3. Directives

Directives are instruction in the DOM

1. In-Built Derectives
   1. Attribute Directives  
      Never destroy element from the DOM.   
      Only property of element is changed.
      * 1. [ngClass]=”{odd: number % 2 !==0 }”
        2. [ngClass]="{online: serverStatus === 'Online'}"
        3. [ngStyle]=”{backgroundColor: number %2 !== 0 ? ‘yellow’ : ‘transparent’}”
        4. [ngStyle]="{background: getColor()}"
   2. Structural Directives

* The DOM is getting changed so overall view container is affected

<p \*ngIf="serverCreated; else noServer">Server is created and name is {{serverName}}</p>

<ng-template #noServer>

<p>No Server Created</p>

</ng-template>

<p \*ngFor="let server of Serves">Server is created and name is {{server.serverName}}</p>

<div [ngSwitch]="value">

<p \*ngSwitchCase="5">Value is 5</p>

<p \*ngSwitchCase="10">Value is 10</p>

<p \*ngSwitchCase="100">Value is 100</p>

<p \*ngSwitchDefult">Value is Default</p>

</div>

1. Create basic Attribute Directives
   1. Create a folder
   2. Create a file name with basic-hilight.directive.ts
   3. In highlight.directive.ts

Import { Drective, ElementRef, OnInit } from ‘@angular/core’

@Drective({

Selector: ‘[appBasicHighlight]’

})

Export class BasicHilightDirective implements OnInit{

Constructor(private elementRef: ElementRef){}

ngOnInit(){

this.elementRef.nativeElement.style.backgroundColor = ‘green’;

}

}

* 1. In appModule.ts

Import { BasicHilightDirective } from ……

@NgModule({

Declarations: [

……

BasicHilightDirective

]

})

* 1. **HTML:** <p appBasicHighlight >Style me with Basic Directive</p>

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

@Directive({

  selector: '[NumberSpinnerDirective]'

})

export class NumberSpinnerDirective {

  numberValue:number;

  NumberElement;

  constructor(private elRef: ElementRef) {}

  @HostListener('click', ['$event'])

  clickEvent(event) {

    this.NumberElement = this.elRef.nativeElement.getElementsByClassName('numberValue').item(0);

    this.numberValue = this.NumberElement.value;

    if(event.target.classList.contains('count-down')){

      if(this.numberValue >= 0){

         this.NumberElement.value = +this.NumberElement.value + 1;

         this.elRef.nativeElement.getElementsByClassName('numberValue').item(0).dispatchEvent(new Event('change'));

      }

      else{

      }

    }

    if(event.target.classList.contains('count-up')){

      if(this.numberValue > 0){

        this.NumberElement.value = +this.NumberElement.value - 1;

        this.elRef.nativeElement.getElementsByClassName('numberValue').item(0).dispatchEvent(new Event('change'));

      }

      else{}

    }

  }

}

1. Create basic Attribute Directive in better way and using Renderer
2. ng n d directives/better-highlight
3. In better-highlight.directive.ts

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

@Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective implements OnInit {

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

ngOnInit(){

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

}

}

<p appBetterHighlight>Createing attribute director</p>

Page.html

1. Using HostListner to listen Host Events

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

@Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective {

@HostListener('mouseenter') onMouseEnter() {

alert("Don't touch my bacon!");

}

}

<p appBetterHighlight>Createing attribute director</p>

Page.html

1. Passing value from Child to parent component using EventEmitter (@Output())
2. Passing Variable

<div class="btn btn-success" (click) = 'onCourierSend()'>Send Courier</div>

Child.html

courier = "Gift from Surat";

@Output() gift = new EventEmitter<string>();

onCourierSend()

{

console.log("Courier Sent to Valsad");

this.gift.emit(this.courier);

}

Child.ts

<app-surat (gift) = "giftReceived ($event)"></app-surat>

Parent.html

1. Passing Value from Parent to Child (@Input())
2. Passing Variable

<app-malad [giftToMalad] = "giftToMumbai"></app-malad>

parent.html

@Input() giftToMalad;

child.ts

1. View Encapsulation

selector: 'app-server-element',

templateUrl: './server-element.component.html',

styleUrls: ['./server-element.component.css'],

encapsulation: ViewEncapsulation.Emulated

<strong \_ngcontent-wqv-c2="" style="color: red">

None / Native / ShadowDom

1. Local Reference

Can be used on any HTML element. Not only on input element.

declare a reference variable by using the hash symbol (#)

you can access the variable anywhere inside the template

* 1. To access in same template

<input type="text" #lastNameInput>

<button (click)="show(lastNameInput)">Show</button>

Page.html

show(lastName: HTMLInputElement){

console.log(lastName.value);

}

Page.ts

* 1. To Access from other parts of components

<input type="text" #lastNameInput>

<button (click)="show(lastNameInput)">Show</button>

<p>Access value is{{ lstnmInput.nativeElement.value }}</p>

Page.html

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

export class AppComponent {

@ViewChild('lastNameInput') lstnmInput: ElementRef;

show(lastName: HTMLInputElement){

//console.log(this.lstnmInput.nativeElement.value);

}

}

Page.ts

1. Projecting content into component using ng-cotent

<app-server-element>

<p>

<strong >…………..</strong> </p>

</app-server-element>

Parent.html

<div class="panel panel-default">

<div class="panel-heading">……………….div>

<div class="panel-body"> <ng-content></ng-content></div>

</div>

</app-server-element>

child.html

1. Local Reference for ng-cotent

@ContentChild('contentParagraph') paragraph : ElementRef;

ngOnInit() {

console.log('text content of paragraph' + this.paragraph.nativeElement.textContent);

}

ngAfterContentInit()

{

console.log('text content of paragraph' + this.paragraph.nativeElement.textContent);

}

Page.ts

<app-server-element>

<p #contentParagraph>…………… </p>

</app-server-element>

Parent.html

1. Life Cycle Hooks
   1. ngOnChanges:
      1. Can be executed multiple times
      2. Executed when new component is created
      3. Whenever bound input property changes (Property decorator @ input – when it receive new value)

import { Component, Input, SimpleChanges, OnChanges } from '@angular/core';

export class MyTodoComponent implements OnChanges {

ngOnChanges(changes: SimpleChanges) {

this.doStuff();

}

}

* 1. ngOnInit
     1. Called once component is initialize ( object is created you can say )
     2. It runs after the constructor

export class AppComponent implements OnInit {

constructor() {}

ngOnInit() {

this.doStuff();

}

}

* 1. ngDoCheck
     1. Runs multiple times and runs a lot
     2. Whenever the is change in the template
     3. Even if there is no change and click on button it run as it need to check whether there are changes
  2. ngAfterContentInit
     1. Executes when the content via ng-content has been initialized
     2. Executes in the parent component and not in the existing component
  3. ngAfterContentChecked
     1. When ng-content has been checked
  4. ngAfterViewInit
     1. when our view of the component has been initialized
  5. ngAfterViewChecked
     1. when view of the component has been checked
  6. ngOnDestroy
     1. Component is about to be destroyed

1. Bug Fixing
   1. Through Console
   2. Using Source Map (Debugging tool/resources main.bundle.js) / Webpack
   3. Using Augury
   4. Splitting the project in to components
      1. Shift HTML to desire components
      2. Shift methods to desire components
      3. Shift defined variables
2. Using HostBinder to add / remove class to element on (Mouse over / Mouse leave )

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

@Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective {

@HostBinding('class.someClass') someField: boolean = false;

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

this.someField = true;

}

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

this.someField = false;

}

}

1. Using Listner / HostBinder to change background color dynamically

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

@Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective implements OnInit {

highlightColor = "transparent";

defaultColor = "blue";

@HostBinding('style.backgroundColor') backgroundColor: string = this.highlightColor;;

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

this.backgroundColor=this.highlightColor;

}

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

this.backgroundColor= this.defaultColor;

}

}

1. Building own / Custom structural directive
2. Creating a service

* AngularJS services are substitutable objects that are wired together using dependency injection
* Angular services are singleton objects which get instantiated only once during the lifetime of an application
* They contain methods that maintain data throughout the life of an application, i.e. data does not get refreshed and is available all the time.

export class LoggingService{

logStatusChange(stauts: string)

{

console.log('A Serve status changed, new status is ' + stauts);

}

}

Logging.service.ts

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

import { LoggingService } from '../shared/logging.service';

@Component({

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

providers: [LoggingService]

})

export class AccountComponent {

constructor(private logingService: LoggingService){}

onSetTo(status: string) {

this.statusChanged.emit({id: this.id, newStatus: status});

this.logingService.logStatusChange(status);

}

}

component.ts

1. Create Data Service

export class AccountsService{

accounts = [

{

name: 'Master Account',

status: 'active'

},

{

name: 'Testaccount',

status: 'inactive'

},

{

name: 'Hidden Account',

status: 'unknown'

}

];

addAccount(name: string, status: string)

{

this.accounts.push({name: name, status: status});

}

updateAccount(id: number, status: string)

{

this.accounts[id].status = status;

}

}

accountService.ts

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

import { AccountsService } from '../app/shared/accounts.service'

@Component({

selector: 'app-root',

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

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

providers: [AccountsService]

})

export class AppComponent implements OnInit {

accounts: {name: string, status: string}[] = [];

constructor(private accountsService: AccountsService ){}

ngOnInit(){

this.accounts = this.accountsService.accounts;

}

}

appcomponent.ts

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

import { AccountsService } from '../shared/accounts.service';

@Component({

selector: 'app-new-account',

templateUrl: './new-account.component.html',

styleUrls: ['./new-account.component.css'],

providers: [LoggingService]

})

export class NewAccountComponent {

constructor(private loggingService: LoggingService, private accountsService: AccountsService){}

onCreateAccount(accountName: string, accountStatus: string) {

this.accountsService.addAccount(accountName, accountStatus)

this.loggingService.logStatusChange(accountStatus);

}

}

New-account.component.ts

<div class="row">

<div class="col-xs-12 col-md-8 col-md-offset-2">

<div class="form-group">

<label>Account Name</label>

<input

type="text"

class="form-control"

#accountName>

</div>

<div class="form-group">

<select class="form-control" #status>

<option value="active">Active</option>

<option value="inactive">Inactive</option>

<option value="hidden">Hidden</option>

</select>

</div>

<button

class="btn btn-primary"

(click)="onCreateAccount(accountName.value, status.value)">

Add Account

</button>

</div>

</div>

New-account.component.html

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

import { LoggingService } from '../shared/logging.service';

import { AccountsService } from '../shared/accounts.service';

@Component({

selector: 'app-account',

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

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

providers: [LoggingService]

})

export class AccountComponent {

constructor(private logingService: LoggingService, private accountsService: AccountsService){}

@Input() account: {name: string, status: string};

@Input() id: number;

onSetTo(status: string) {

this.accountsService.updateAccount(this.id, status);

this.logingService.logStatusChange(status);

}

}

Account-component.ts

<div class="row">

<div class="col-xs-12 col-md-8 col-md-offset-2">

<h5>{{ account.name }}</h5>

<hr>

<p>This account is {{ account.status }}</p>

<button class="btn btn-default" (click)="onSetTo('active')">Set to 'active'</button>

<button class="btn btn-default" (click)="onSetTo('inactive')">Set to 'inactive'</button>

<button class="btn btn-default" (click)="onSetTo('unknown')">Set to 'unknown'</button>

</div>

</div>

1. Injecting service into service

If you inject services into services:

* 1. Make sure to provide the service on the app module level
  2. Add @injectable to a service where you want to inject it in.

import { AccountsService } from './shared/accounts.service';

providers: [AccountsService, LoggingService],

bootstrap: [AppComponent]

})

export class AppModule { }

app.module.ts

export class LoggingService{

logStatusChange(stauts: string)

{

console.log('A Serve status changed, new status is ' + stauts);

}

}

child.service.ts

import { LoggingService } from "./logging.service";

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

@Injectable()

export class AccountsService{

constructor(private loggingService: LoggingService){}

addAccount(name: string, status: string)

{

this.accounts.push({name: name, status: status});

this.loggingService.logStatusChange(status);

}

updateAccount(id: number, status: string)

{

this.accounts[id].status = status;

this.loggingService.logStatusChange(status);

}

}

parent.service.ts

1. Cross component communication with EvenEmmiter and subscribe

import { LoggingService } from "./logging.service";

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

@Injectable()

export class AccountsService{

statusUpdated = new EventEmitter<string>();

constructor(private loggingService: LoggingService){}

addAccount(name: string, status: string)

{

this.accounts.push({name: name, status: status});

this.loggingService.logStatusChange(status);

}

}

}

accountService.ts

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

import { AccountsService } from '../shared/accounts.service';

export class AccountComponent {

constructor(private accountsService: AccountsService){}

onSetTo(status: string) {

this.accountsService.updateAccount(this.id, status);

this.accountsService.statusUpdated.emit(status);

}

}

Account.component.ts

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

import { AccountsService } from '../shared/accounts.service';

export class NewAccountComponent {

constructor(private accountsService: AccountsService){

this.accountsService.statusUpdated.subscribe(

(status: string) => alert ('New Status: ' + status)

);

}

}

new-account.component.ts

1. Routing

Static Routing

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

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

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'users', component: UsersComponent},

{ path: 'servers', component: ServersComponent},

{ path: 'not-found', component: PageNotFoundComponent},

{ path: '\*\*', redirectTo: ‘/not-found’},

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

]

@NgModule({

declarations: [

AppComponent,

],

imports: [

BrowserModule,

FormsModule,

RouterModule.forRoot(appRoutes)

],

App-module.ts

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<router-outlet></router-outlet>

</div>

</div>

App.component.html

<ul class="nav nav-tabs">

<li role="presentation"

routerLinkActive="active"

[routerLinkActiveOptions]="{exact: true}"

><a routerLink="/">Home</a></li>

<li role="presentation"

routerLinkActive="active"

><a routerLink="servers">Servers</a></li>

<li role="presentation"

routerLinkActive="active"

><a [routerLink]="['users']">Users</a></li>

</ul>

header.html

Dynamic Routing

<h4>Welcome to Server Manager 4.0</h4>

<p>Manage your Servers and Users.</p>

<button class="btn btn-primary" (click)="onLoadServers()">Load Server</button>

home.html

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

import { Router } from '@angular/router';

export class HomeComponent implements OnInit {

constructor(private router: Router) { }

onLoadServers(){

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

}

}

Using relative path for Dynamic Routing

<div class="col-xs-12 col-sm-4">

<button class="btn btn-primary" (click) = "onReload()">Reload Page</button>

<app-edit-server></app-edit-server>

<hr>

<app-server></app-server>

</div>

server.html

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

import { ServersService } from './servers.service';

import { Router, ActivatedRoute } from '@angular/router';

@Component({

selector: 'app-servers',

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

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

})

export class ServersComponent implements OnInit {

constructor(

private router: Router,

private route: ActivatedRoute

) { }

onReload()

{

this.router.navigate(['servers'], {relativeTo: this.route });

}

}

Server.component.ts

Passing parameters to route

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'users', component: UsersComponent},

{ path: 'users/:id', component: UserComponent},

{ path: 'servers', component: ServersComponent},

]

@NgModule({

declarations: [

app-module.ts

Fetching parameters from route

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'users', component: UsersComponent},

{ path: 'users/:id/:name', component: UserComponent},

{ path: 'servers', component: ServersComponent},

]

@NgModule({

declarations: [

app-module.ts

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

import { ActivatedRoute } from '@angular/router';

export class UserComponent implements OnInit {

user: {id: number, name: string};

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user ={

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name'],

}

}

}

User.component.ts

<p>User with ID {{user.id}} loaded.</p>

<p>User name is {{user.name}}</p>

User.component.html

Fetching parameters from route Reacively

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'users', component: UsersComponent},

{ path: 'users/:id/:name', component: UserComponent},

{ path: 'servers', component: ServersComponent},

]

@NgModule({

declarations: [

app-module.ts

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

import { ActivatedRoute, Params } from '@angular/router';

export class UserComponent implements OnInit {

user: {id: number, name: string};

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user ={

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name'],

}

this.route.params

.subscribe(

(params: Params) => {

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

this.user.name = params['name'];

}

)

}

}

}

User.component.ts

<p>User with ID {{user.id}} loaded.</p>

<p>User name is {{user.name}}</p>

<hr>

<a [routerLink]="['/users',10,'Anna']">Load Anna</a>

User.component.html

Passing Query Parameters and Fragments

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'servers', component: ServersComponent},

{ path: 'servers/:id/edit', component: EditServerComponent}

App.module.ts

<div class="list-group">

<a

[routerLink]="['/servers', 5, 'edit']"

[queryParams]="{allowEdit: '1'}"

fragment="loading"

href="#"

class="list-group-item"

\*ngFor="let server of servers">

{{ server.name }}

</a>

</div>

Page.html

Passing Query Parameters and Fragments Dynamicaly

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent},

{ path: 'servers', component: ServersComponent},

{ path: 'servers/:id/edit', component: EditServerComponent}

App.module.ts

<button class="btn btn-primary" (click)="onLoadServers(1)">Load Server</button>

home.html

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

export class HomeComponent implements OnInit {

onLoadServers(id){

this.router.navigate(['/servers', 'id', 'edit'], {queryParams: {allowEdit: '1'}, fragment: 'loading'})

}

}

Home.component.ts

import { ActivatedRoute } from '@angular/router';

export class EditServerComponent implements OnInit {

constructor(private route: ActivatedRoute) { }

ngOnInit() {

// console.log(this.route.snapshot.queryParamMap);

//console.log(this.route.snapshot.fragment);

this.route.queryParamMap.subscribe();

this.route.fragment.subscribe();

}

}

Edit-server.component.ts

Configuring handling of Query Paramaters

onEdit()

{

this.router.navigate(['edit'], {relativeTo: this.route, queryParamsHandling: 'preserve'});

}

Setting up child (Nested) Routes

<div class="row">

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

<ul class="list-group">

<li

routerLinkActive="active"

[routerLinkActiveOptions]="{exact: true}"

><a class="list-group-item list-group-item-action"

routerLink="/about">About Us</a></li>

<li

routerLinkActive="active"

><a class="list-group-item list-group-item-action"

routerLink="reach/by-car">How to Reach</a></li>

<li routerLinkActive="active"><a class="list-group-item list-group-item-action"

routerLink="tradition">Our Tradition</a></li>

</ul>

</div>

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

<router-outlet></router-outlet>

</div>

</div>

Child.html

const appRoutes: Routes =[

{path: '', component: HomeComponent},

{path: 'about', component: AboutComponent, children:[

{path: '', component: AboutUsComponent},

{path: 'reach', component: ReachComponent

{path: 'tradition', component: TraditionComponent}

]},

{path: 'services', component: ServicesComponent}

]

App-module.ts

1. Sdfg
2. Bootstrap dropdown functionality using hostlistner and hostbinder

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

@Directive({

selector: '[appDropdown]'

})

export class DropdownDirective {

constructor(private elRef: ElementRef) {}

@HostBinding ('class.open') isOpen: boolean = false;

// @HostListener('click') toggleOp(){

// this.isOpen = !this.isOpen;

// }

@HostListener('document:click', ['$event']) toggleOpn(event: Event) {

this.isOpen = this.elRef.nativeElement.contains(event.target) ? !this.isOpen : false;

}

}

Dropdown.directive.ts

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

<li class="dropdown" appDropdown>

<a href="#" class="dropdown-toggle" role="button">Manage <span class="caret"></span></a>

<ul class=dropdown-menu>

<li><a href="#">Saving Data</a></li>

<li><a href="#">Fetching Data</a></li>

</ul>

</li>

</ul>

Page.html

...

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

@NgModule({

declarations: [

DropdownDirective

],

App.module.ts

1. Observables

Assume, you’re working with an application containing a source code in which a few lines are written for the server interaction (request, response). Now, when you run the application, your script executes line by line and when it comes to the server interaction of line of code, the execution is paused because now, the request is interacting with the server and waiting for the response. Until we get the response back, next lines of code will never execute. And the response might depend on the internet connection and many other factors. Let’s suppose the response comes back after 3 seconds. Only after these three seconds, the further code is executed. Well, this is an issue. We have to wait until the response comes back.

So in Angular, when we’re using HTTP request, the HTTP service sets the Observable object during the interaction of the request with the server. So, Observable is responsible for server interaction. It sends the request to the server and receives the response as well. In simple words, Observable observes the response on the server. So when we’re using Observable, during execution, we make Observable object responsible for handling the response and we subscribe that Observable object in some point in our code. In this way, our script remains in execution state and doesn’t wait for a response. And then, when the response comes back, the Observable object handles that response and implements that response in all the places where we subscribed it. So this is

why we subscribe to the Observable.

* Observables are various data sources
* Object we import third party package rxjs
* We can handle
  + Data
  + Error
  + Completion
* We very rarely built our own observables

Inbuilt observable

Angular unsubscribe these observables automatically

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

import { ActivatedRoute, Params } from '@angular/router';

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit {

id: number;

constructor(private route: ActivatedRoute) {

}

ngOnInit() {

this.route.params.subscribe((params: Params) => {

this.id = +params.id;

});

}

}

Custom observable

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

import { Subscription, Observable } from 'rxjs';

export class HomeComponent implements OnInit, OnDestroy {

private firstObsSubscripton: Subscription;

constructor() { }

ngOnInit() {

const customIntervalObservable = Observable.create( observer => {

let count = 0;

setInterval(() => {

observer.next(count);

count++;

},1000);

});

this.firstObsSubscripton = customIntervalObservable.subscribe(data =>{

console.log(data);

})

ngOnDestroy()

{

this.firstObsSubscripton.unsubscribe();

}

}

Observable – Error Handling

ngOnInit() {

const customIntervalObservable = Observable.create( observer => {

let count = 0;

setInterval(() => {

observer.next(count);

if(count > 3){

observer.error(new Error('Count is greater that 3'));

}

count++;

},1000);

});

this.firstObsSubscripton = customIntervalObservable.subscribe(data =>{

console.log(data);

}, error =>{

console.log(error);

alert(error.message);

})

}

Completing Observable

ngOnInit() {

const customIntervalObservable = Observable.create( observer => {

let count = 0;

setInterval(() => {

observer.next(count);

if(count === 2){

observer.complete();

}

if(count > 3){

observer.error(new Error('Count is greater that 3'));

}

count++;

},1000);

});

this.firstObsSubscripton = customIntervalObservable.subscribe(data =>{

console.log(data);

}, error =>{

console.log(error);

alert(error.message);

}, () => {

console.log('completed');

})

1. Operators

<https://www.learnrxjs.io/operators/>

* 1. Features of Rxjs library
  2. Data reach to operators

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

import { Subscription, Observable } from 'rxjs';

import { map, filter } from 'rxjs/operators';

this.firstObsSubscripton = customIntervalObservable.pipe(filter(data => {

return data > 0;

}), map( data => {

return 'Round:' + (data+1);

})).subscribe(data =>{

console.log(data);

}, error =>{

console.log(error);

alert(error.message);

}, () => {

console.log('completed');

})

1. Subjects
   1. Subjects are imported from rxs
   2. Subjects are substitute to EventEmmiter
   3. Subjects need to be unsubscribed
   4. Use it for cross component communication through services
   5. Not to use subjects for output
   6. Instead of EventEmmite -> subject, instead of emit -> next

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

import { Subject } from 'rxjs'

@Injectable()

export class UserService{

userActivatedEmmiter = new Subject<boolean>();

}

User.service.ts

this.userService.userActivatedEmmiter.subscribe(data => {

this.userActivated = data;

})

App-component.ts

Un Suscribe subject

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

import { UserService } from './user.service';

import { Subscription } from 'rxjs'

export class AppComponent implements OnInit, OnDestroy {

userActivated = false;

private activatedSub: Subscription;

constructor(private userService: UserService) {}

ngOnInit() {

this.activatedSub = this.userService.userActivatedEmmiter.subscribe(data => {

this.userActivated = data;

})

}

ngOnDestroy()

{

this.activatedSub.unsubscribe();

}

}

1. Forms

Form States

* Dirty – input has been changed
* Touched – clicked on the input
* Invalid – not valid

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

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

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

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

@NgModule({

imports: [

BrowserModule,

FormsModule,

],

providers: [],

app.module.ts

Template – Driven approach

* 1. Angular infers the form object from the DOM

Submit and Using Form using ngForm

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

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

<div id="user-data">

<div class="form-group">

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

<input

type="text"

id="username"

class="form-control"

ngModel

name = "username" >

</div>

<button class="btn btn-default" type="button">Suggest an Username</button>

<div class="form-group">

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

<input

type="email"

id="email"

class="form-control"

ngModel

name = "email">

</div>

</div>

<div class="form-group">

<label for="secret">Secret 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>

</div>

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

>Submit</button>

</form>

</div>

</div>

</div>

Page.component.html

onCurrSubmit(form: NgForm)

{

this.currUser.email = form.value.email;

this.currUser.stage = form.value.subscripton;

this.currUser.password = form.value.passowrd;

}

Page.component.ts

Accessing form using @ViewChild

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

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

@ViewChild('f') signupForm: NgForm;

suggestUserName() {

const suggestedName = 'Superuser';

}

// onSubmit(form: NgForm)

// {

// console.log(form);

// }

onSubmit()

{

console.log(this.signupForm);

}

}

Adding validation to form

<div class="form-group">

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

<input

type="email"

id="email"

class="form-control"

ngModel

name = "email"

required

email

#email="ngModel">

<span class="help-block" \*ngIf="!email.valid && email.touched">Enter Valid Email ID</span>

</div>

Page.html

input.ng-invalid.ng-touched{border: 2px solid red}

Page.css

Set default values with ngModel property (One way binding)

<select

id="secret"

class="form-control"

[ngModel]="defaultQuestion"

name = "secret">

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

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

</select>

Page.html

export class AppComponent {

@ViewChild('f') signupForm: NgForm;

defaultQuestion = "pet";

}

using ngModel with two way binding

<div class="form-group">

<label for="my-input">Text</label>

<textarea

name="questionAnswer"

rows="10"

class="form-control"

[(ngModel)]="answer"></textarea>

</div>

<span class="help-block">You have entered: {{answer}}</span>

Page.html

Grouping form controls

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

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

<div id="user-data"

ngModelGroup

#userData = "ngModelGroup">

<div class="form-group">

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

<input

type="text"

id="username"

class="form-control"

ngModel

name = "username"

required >

</div>

</div>

<span class="help-block" \*ngIf="!userData.valid && userData.touched">User data is invalid</span>

Page.html

Handlng Radio buttons

<div class="radio" \*ngFor="let gender of genders">

<label>

<input

type="radio"

name="gender"

[ngModel]="genders[1]"

[value]="gender"

> {{gender}}

</label>

</div>

Page.html

genders=['Male', 'Female'];

Page.component.ts

Setting and patching form values

* Set value is not recommendable as it replace data of all fields in the form
* Patch values is highly recommendable

Set Values method

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

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

export class AppComponent {

@ViewChild('f') signupForm: NgForm;

defaultQuestion = "pet";

genders=['Male', 'Female'];

suggestUserName() {

const suggestedName = 'Superuser';

this.signupForm.setValue({

userData:{

username: suggestedName,

email: ''

},

secret: 'pet',

questionAnswer: '',

gender: 'Female'

})

}

}

Page.component.ts

<form #f="ngForm">

<div id="user-data"

ngModelGroup = "userData"

#userData = "ngModelGroup">

<div class="form-group">

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

<input

type="text"

id="username"

class="form-control"

ngModel

name = "username"

required >

</div>

<button class="btn btn-default" type="button" (click)="suggestUserName()">Suggest an Username</button>

<div class="form-group">

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

<input

type="email"

id="email"

class="form-control"

ngModel

name = "email"

required

email

#email="ngModel">

<span class="help-block" \*ngIf="!email.valid && email.touched">Enter Valid Email ID</span>

</div>

</div>

<span class="help-block" \*ngIf="!userData.valid && userData.touched">User data is invalid</span>

<div class="form-group">

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

<select

id="secret"

class="form-control"

[ngModel]="defaultQuestion"

name = "secret">

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

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

</select>

</div>

<div class="form-group">

<label for="my-input">Text</label>

<textarea

name="questionAnswer"

rows="10"

class="form-control"

[(ngModel)]="answer"></textarea>

</div>

<span class="help-block">You have entered: {{answer}}</span>

<div class="radio" \*ngFor="let gender of genders">

<label>

<input

type="radio"

name="gender"

[ngModel]="genders[1]"

[value]="gender"

> {{gender}}

</label>

</div>

<button class="btn btn-primary" type="submit">Submit</button>

</form>

Patch Values method

this.signupForm.form.patchValue({

userData:{

username: suggestedName

},

gender: 'Female'

})

Using Form data

<div \*ngIf="fromSubmitted">

<h3>Your Data</h3>

<p>User name: {{user.username}}</p>

<p>Email: {{user.email}}</p>

<p>Secret Question: {{user.secretQuestion}} </p>

<p>Comment: {{user.comment}} </p>

<p>Gender: {{user.gender}} </p>

</div>

Page.component.html

@ViewChild('f') signupForm: NgForm;

fromSubmitted = false;

user={

username: "",

email: "",

secretQuestion: "",

comment: "",

gender: ""

}

onSubmit()

{

this.fromSubmitted=true;

this.user.username=this.signupForm.value.userData.username;

this.user.email=this.signupForm.value.userData.email;

this.user.secretQuestion=this.signupForm.value.secret;

this.user.comment=this.signupForm.value.questionAnswer;

this.user.gender=this.signupForm.value.gender;

}

}

Reset the form

onReset()

{

this.signupForm.reset();

}

Reactive Forms

1. Form is created programmatically and synchronically with DOM

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

import { FormGroup, FormControl, Validators } from '@angular/forms'

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

ngOnInit()

{

this.signupForm = new FormGroup({

'userData': new FormGroup({

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

'email': new FormControl(null, [Validators.required, Validators.email])

}),

'gender': new FormControl('male')

})

}

onSubmit()

{

console.log(this.signupForm.value);

}

}

Page.component.ts

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

<div formGroupName="userData">

<div class="form-group">

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

<input

type="text"

id="username"

formControlName="username"

class="form-control">

<span class="help-block" \*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched">Please enter valid user name</span>

</div>

<div class="form-group">

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

<input

type="text"

id="email"

formControlName="email"

class="form-control">

<span class="help-block" \*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched">Please enter valid Email ID</span>

</div>

</div>

<div class="radio" \*ngFor="let gender of genders">

<label>

<input

type="radio"

formControlName="gender"

[value]="gender">{{ gender }}

</label>

</div>

<button class="btn btn-primary" type="submit">Submit</button>

</form>

Page.component.html

Reactive Forms – Array of form control

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

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

<div formArrayName="hobbies">

<h4>Your Hobbies</h4>

<button class="btn btn-default" type="button" (click)="onAddHobbies()">Add your bobbies</button>

<div class="form-group"

\*ngFor = "let hobbyControl of signupForm.get('hobbies').controls, let i = index"

>

<input class="form-control" type="text" [formControlName]="i" >

</div>

</div>

<button class="btn btn-primary" type="submit">Submit</button>

</form>

</div>

</div>

Page.component.html

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

import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms'

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

ngOnInit()

{

this.signupForm = new FormGroup({

'userData': new FormGroup({

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

'email': new FormControl(null, [Validators.required, Validators.email])

}),

'gender': new FormControl('male'),

'hobbies': new FormArray([])

})

}

onSubmit()

{

console.log(this.signupForm.value);

}

onAddHobbies()

{

const control = new FormControl(null, Validators.required);

(<FormArray>this.signupForm.get('hobbies')).push(control);

}

}

Reactive Forms – Create your own validations

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

import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms'

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

forbiddentUsernames = ['Chriss', 'Anna'];

ngOnInit()

{

this.signupForm = new FormGroup({

'userData': new FormGroup({

'username': new FormControl(null, [Validators.required, this.forbiddentNames.bind(this)]),

'email': new FormControl(null, [Validators.required, Validators.email])

}),

'gender': new FormControl('male'),

'hobbies': new FormArray([])

})

}

forbiddentNames(control: FormControl): {[s: string]: boolean}

{

if(this.forbiddentUsernames.indexOf(control.value) !== -1){

return{'nameIsForbidden': true};

}

return null;

}

}

Reactive Forms – Create your own validations through TS file

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

import { FormGroup, FormControl, Validators } from '@angular/forms';

import { CustomValidators } from './custom.validators'

export class AppComponent implements OnInit {

statusForm: FormGroup

ngOnInit()

{

this.statusForm = new FormGroup({

'projectname': new FormControl(null, [Validators.required, CustomValidators.invalidProjectName], CustomValidators.asyncInvalidProjectName),

'email': new FormControl(null, [Validators.required, Validators.email]),

'formStatus': new FormControl('critical')

})

}

page.component.ts

import { FormControl } from "@angular/forms";

import { Observable } from 'rxjs/observable';

export class CustomValidators{

static invalidProjectName(control: FormControl): {[s: string]: boolean}

{

if(control.value === 'Test'){

return{'invalidProjectName': true};

}

return null;

}

static asyncInvalidProjectName(control: FormControl): Promise<any> | Observable<any>{

const promise = new Promise<any>((resolve, reject) => {

setTimeout(() => {

if(control.value == 'test@test.com'){

resolve({'invalidProjectName': true});

}

else{

resolve(null);

}

}, 2000);

});

return promise;

}

}

custom.validatiors.ts

Using Error codes

<div class="form-group">

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

<input

type="text"

id="username"

formControlName="username"

class="form-control">

<span class="help-block" \*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched">

<span \*ngIf="signupForm.get('userData.username').errors['nameIsForbidden']">This name is Invalid</span>

<span \*ngIf="signupForm.get('userData.username').errors['required']">User Name required</span>

</span>

</div>

Page.component.html

Creating custom Async validators

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

import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms'

import { Observable } from 'rxjs/observable';

export class AppComponent implements OnInit {

ngOnInit()

{

this.signupForm = new FormGroup({

'userData': new FormGroup({

'username': new FormControl(null, [Validators.required, this.forbiddentNames.bind(this)]),

'email': new FormControl(null, [Validators.required, Validators.email], this.fobiddentEmails)

}),

'gender': new FormControl('male'),

'hobbies': new FormArray([])

})

}

fobiddentEmails(control: FormControl): Promise<any> | Observable<any>{

const promise = new Promise<any>((resolve, reject) => {

setTimeout(() => {

if(control.value == 'test@test.com'){

resolve({'emailIsForbiddent': true});

}

else{

resolve(null);

}

}, 1500);

});

return promise;

}

}

Page.component.ts

Reacting to status or value changes

ngOnInit()

{

})

// this.signupForm.valueChanges.subscribe(

// (value) => console.log(value)

// );

this.signupForm.statusChanges.subscribe(

(status) => console.log(status)

);

}

Page.component.ts

SetValues() / PatchValues(), Reset()

this.signupForm.setValue({

'userData': {

'username': 'Max',

'email': 'max@max.com'

},

'gender': 'Male',

'hobbies': []

})

this.signupForm.patchValue({

'userData': {

'username': 'Anna'

}

})

}

onReset()

{

this.signupForm.reset();

}

Page.component.ts

1. Pipe
   1. Pipes are a feature built into Angular 2 which basically allows you to transform output in your template
   2. There are pipes for different types of output and also for synchronous and asynchronous data.

<https://angular.io/api/core/Pipe>

Chaining multiple pipe

{{server.started | date:’fullDate’ | uppercase}}

Creating custom pipe

{{server.started | date:’fullDate’ | uppercase}}

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

{{title | shortern : 7}}

</div>

Page.component.html

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

@Pipe({

name: 'shortern'

})

export class ShorternPipe implements PipeTransform{

transform(value: any, limit: number) {

if(value.length > limit){

return value.substr(0, limit);

}

return value;

}

}

shortern.pipe.ts

Creating filter pipe

{{server.started | date:’fullDate’ | uppercase}}

<input type="text" [(ngModel)]="filteredSatus">

Page.component.html

filteredSatus=" "

Page.component.ts

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

@Pipe(

{

name: 'filter'

pure: false // not recommended

})

export class FilterPipe implements PipeTransform{

transform(value: any, filterString: string, propName: string):any {

if(value.length === 0 || filterString === ''){

return value;

}

for(const item of value){

const resultArray = [];

if(item[propName]===filterString){

resultArray.push(item);

return resultArray;

}

}

}

}

Filter.pipe.ts

Async pipe

appServerStatus = new Promise((resolve, reject) => {

setTimeout(() =>{

resolve ('stable');

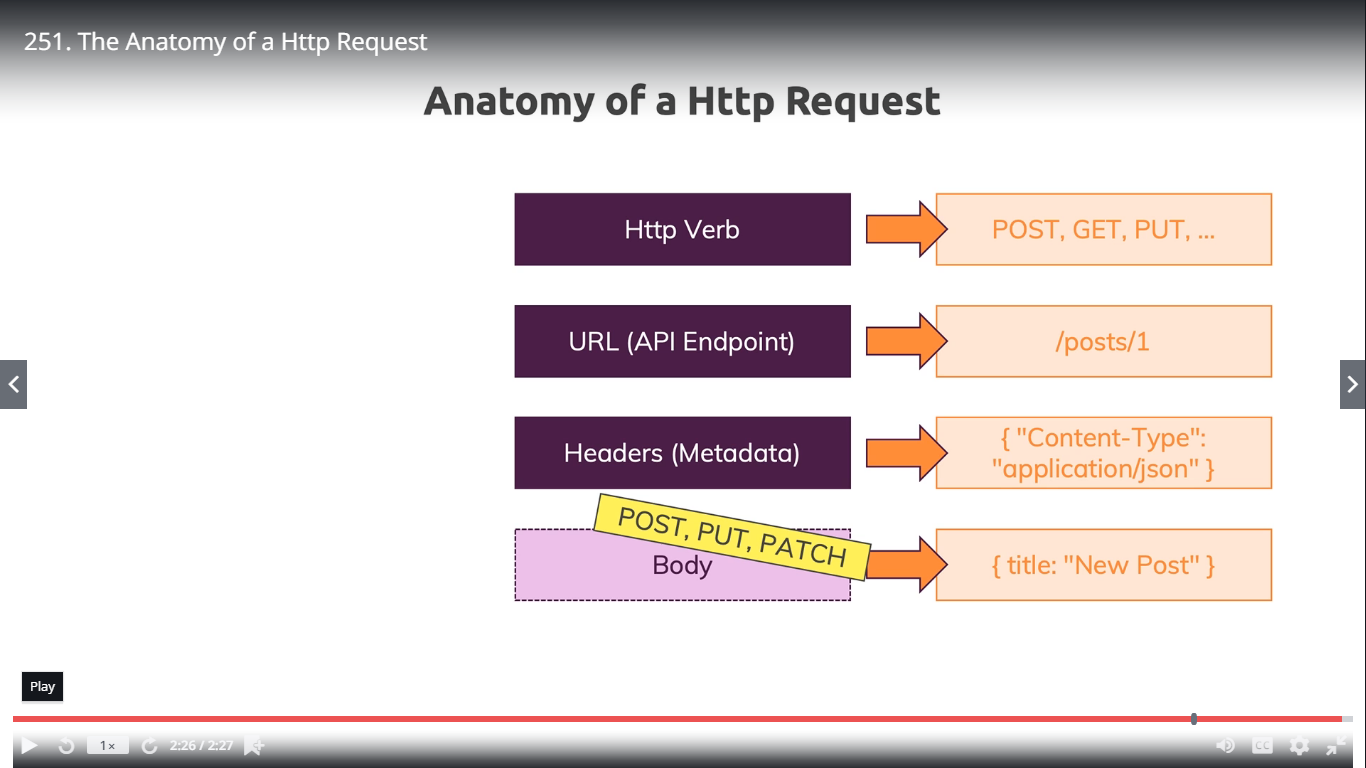
},2000)

});

page.component.ts

{{appServerStatus | async}}

1. Anatomy of HTTP request



1. Sending POST request

 onCreatePost(postData: Post) { //model

    // Send Http request

    this.http

      .post<{ name: string }>(

        'https://search-itinerary.firebaseio.com/posts.json',

        postData

      )

      .subscribe(responseData => {

        console.log(responseData);

      });

  }

page.component.ts

1. GETting Data

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json') .subscribe(responseData => {

        console.log(responseData);

      });

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

<div class="row">

    <div class="col-xs-12 col-md-6 col-md-offset-3">

      <p \*ngIf="loadedPosts.length < 1 && !isFetching">No posts available!</p>

      <p \*ngIf="isFetching">...Loading</p>

      <ul class="list-group" \*ngIf="loadedPosts.length >= 1 && !isFetching">

        <li class="list-group-item" \*ngFor="let post of loadedPosts |  filter:selectedPrice:'price'">

          <h3>{{post.city}}</h3>

          <p>{{post.price}}</p>

          <p>{{post.duration}}</p>

        </li>

      </ul>

    </div>

  </div>

page.component.html

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

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

import { Observable } from 'rxjs';

import { map, catchError } from 'rxjs/operators';

@Injectable({

  providedIn: 'root'

})

export class RestApiService {

  items = [];

  constructor(private http: HttpClient) { }

  getAllScheduleService(): Observable<any> {

    return this.http.get('http://api.tvmaze.com/schedule/full').pipe(

      map(this.extractData),

      catchError(this.handleErrorObservable)

    );

  }

  getAllScheduleDetails(id): Observable<any> {

    return this.http.get(`http://api.tvmaze.com/shows/${id}/episodes`).pipe(

      map(this.extractData),

      catchError(this.handleErrorObservable)

    );

  }

  private extractData(res: Response) {

    let body = res;

    return body;

}

private handleErrorObservable (error: Response | any) {

  console.error(error.message || error);

  return Observable.throw(error.message || error);

}

}

1. Using RxJS operator to transfer Response data

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

1. Using types with HttpClient

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

1. Delete Posts

onClearPosts() {

    this.http.delete('https://search-itinerary.firebaseio.com/posts.json')

    .subscribe(() =>{

      this.loadedPosts = [];

    });

  }

page.component.ts

1. Handling Errors

private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    }, error =>{

        this.error = error.message;

        console.log(error);

    });

  }

page.component.ts

1. Using subject to Errors Handling

import { Subject } from 'rxjs';

export class AppComponent implements OnInit {

  error = new Subject<string>();

  onCreatePost(postData: Post) { //model

    // Send Http request

    this.http

      .post<{ name: string }>(

        'https://search-itinerary.firebaseio.com/posts.json',

        postData

      )

      .subscribe(responseData => {

        console.log(responseData);

      }, error =>{

        this.error.next(error.message);

      }

      );

  }

}

post.service.ts

import { Subject, Subscription } from 'rxjs';

export class AppComponent implements OnInit {

  private errorSub: Subscription;

  constructor(private http: HttpClient) {}

  ngOnInit() {

    this.errorSub = this.postsService.error.subscribe(errorMessage =>{

      this.error = errorMessage;

    })

  }

page.component.ts

1. Catch Error Operator

import { map, catchError } from 'rxjs/operators';

import { Subject,throwError, Subscription } from 'rxjs';

export class AppComponent implements OnInit {

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      }),

      catchError( errorRes =>{

       return throwError(errorRes);

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    }, error =>{

        this.error = error.message;

        console.log(error);

    });

  }

page.component.ts

1. Setting Headers

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

export class AppComponent implements OnInit {

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'})

    }

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    });

  }

page.component.ts

1. Adding Query Parameter

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

export class AppComponent implements OnInit {

  onFetchPosts() {

    this.fetchPost();

  }

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

      params: new HttpParams().set('print', 'pretty')

    }

    .pipe(map(responseData =>{

page.component.ts

Adding Mulitple Query Params

private fetchPost(){

    this.isFetching = true;

    let searchParams = new HttpParams();

    searchParams = searchParams.append('print', 'pretty');

    searchParams = searchParams.append('custom', 'key');

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

     // params: new HttpParams().set('print', 'pretty')

      params: searchParams

    }

1. Observing different types of response

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

export class AppComponent implements OnInit {

  onFetchPosts() {

    this.fetchPost();

  }

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

      params: new HttpParams().set('print', 'pretty')

    }

    .pipe(map(responseData =>{

page.component.ts

1. Interceptor

Interceptors in Angular, is a simple way provided by the framework to intercept and modify the application's http requests globally before they are sent to the server

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

export class AuthInterceptorService implements HttpInterceptor {

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

    console.log('Request is on its way');

    return next.handle(req);

  }

}

Auth-interceptor.service.ts

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

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

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

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

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

import { AuthInterceptorService } from './auth-interceptor.service';

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, FormsModule, HttpClientModule],

  providers: [

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptorService,

      multi: true

    }

  ],

  bootstrap: [AppComponent]

})

export class AppModule {}

app-module.ts

1. Manipulating Request object

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

export class AuthInterceptorService implements HttpInterceptor {

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

    console.log('Request is on its way');

    console.log(req.url);

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest);

  }

}

Auth-interceptor.service.ts

1. Response Interceptors

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

import { tap }  from 'rxjs/operators'

export class AuthInterceptorService implements HttpInterceptor {

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

    console.log('Request is on its way');

    console.log(req.url);

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest).pipe(

      tap(event =>{

        console.log(event);

        if(event.type === HttpEventType.Response){

          console.log('Response arrived, Body data: ');

          console.log(event.body);

        }

      })

    );

  }

}

Auth-interceptor.service.ts

1. Mulitple Interceptors

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

import { tap } from 'rxjs/operators';

export class LoggingInterceptorSerice implements HttpInterceptor{

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

        console.log('Outgoing Request');

        console.log(req.url);

        console.log(req.headers);

        return next.handle(req).pipe(

            tap(event =>{

            if(event.type === HttpEventType.Response){

                console.log('Incoming Response');

                console.log(event.body);

            }

        }));

    };

}

logging-interceptor.service.ts

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

export class AuthInterceptorService implements HttpInterceptor {

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

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest);

  }

}

Auth-interceptor.service.ts

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, FormsModule, HttpClientModule],

  providers: [

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptorService,

      multi: true

    },

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: LoggingInterceptorSerice,

      multi: true

    }

  ],

  bootstrap: [AppComponent]

})

App-module.ts

1. Resolver

* The Angular Router provides a resolve property that takes a route resolver and allows your application to fetch data before navigating to the route (i.e resolving route data). You can create a route resolver by implementing the Resolve interface
* Resolver is some code that run before route is loaded.

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

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

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

import { DataStoreageService } from '../shared/data-storeage.service';

@Injectable({providedIn: 'root'})

export class RecipesResolverService implements Resolve<Recipe[]>{

constructor(private dataStorageService: DataStoreageService){}

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot ){

    return this.dataStorageService.fetchData();

}

}

Recipe-resolver.service.ts

const routes: Routes = [

  {path: '', component:RecipesComponent, pathMatch: 'full'},

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

    {path: '', component:RecipeStartComponent},

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

    {

      path: ':id',

      component:RecipeDetailComponent,

      resolve: [RecipesResolverService]

    },

    {

      path: ':id/edit',

      component:RecipeEditComponent,

      resolve: [RecipesResolverService]

    }

  ]

  },

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

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

];

App-routing-module.ts

 onFetchData()

    {

        this.dataStoreageService.fetchData().subscribe();

    }

header.component.ts

1. Fixing bug with Resolver

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot ){

    const recipes = this.recipeService.getRecipes();

    if(recipes.length === 0){

        return this.dataStorageService.fetchData();

    }else{

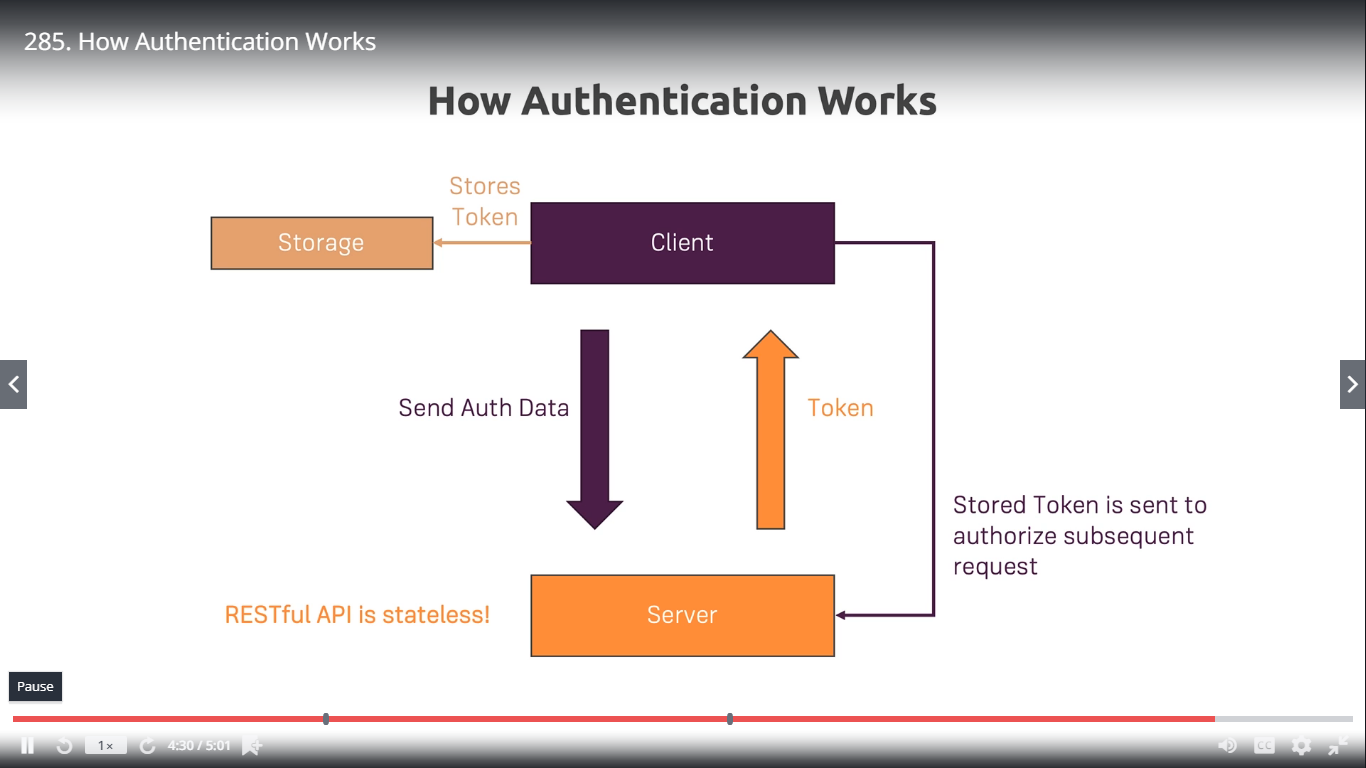
        return recipes;

    }

}

Recipe-resolver.service.ts

1. Authentication



1. Sending the Signup request

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

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

interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

}

@Injectable({providedIn: 'root'})

export class AuthService{

    constructor(private http: HttpClient){}

    signUp(email: string, password: string){

       return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }

        );

    }

}

auth.service.ts

 onSubmit(form: NgForm){

    if(!form.valid){

      return;

    }

    const email = form.value.email;

    const password = form.value.password;

    this.isLoading=true;

    if(this.isLoginMode){

    }else{

      this.authService.signUp(email, password).subscribe(

      responseData =>{

        console.log(responseData);

        this.isLoading=false;

      }, error =>{

        console.log(error);

        this.error = 'Error Occored';

        this.isLoading=false;

      }

    );

    }

auth.component.ts

1. Login Error Handling

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

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

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

}

@Injectable({providedIn: 'root'})

export class AuthService{

    constructor(private http: HttpClient){}

    signUp(email: string, password: string){

       return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }

        ).pipe(

            catchError(errorRes =>{

                let errormessage = 'An unknown error occorred !'

                if(!errorRes.error || !errorRes.error.error){

                    return throwError(errormessage);

                }

                switch(errorRes.error.error.message){

                    case 'EMAIL\_EXISTS':

                        errormessage = 'The email address is already in use by another account.'

                }

                return throwError(errormessage);

            })

        );

    }

}

auth.service.ts

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

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

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

@Component({

  selector: 'app-auth',

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

  styleUrls: ['./auth.component.css']

})

export class AuthComponent implements OnInit {

  isLoginMode = true;

  isLoading = false;

  error: string = null;

  constructor(private authService: AuthService) { }

  ngOnInit() {

  }

  onSwithMode(){

    this.isLoginMode = !this.isLoginMode;

  }

  onSubmit(form: NgForm){

    if(!form.valid){

      return;

    }

    const email = form.value.email;

    const password = form.value.password;

    this.isLoading=true;

    if(this.isLoginMode){

    }else{

      this.authService.signUp(email, password).subscribe(

      responseData =>{

        console.log(responseData);

        this.isLoading=false;

      }, errorMessage =>{

        console.log(errorMessage);

        this.error = errorMessage;

        this.isLoading=false;

      }

    );

    }

    form.reset();

  }

}

auth.component.ts

1. Sending Login Request

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

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

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    constructor(private http: HttpClient){}

      login(email: string, password: string){

        return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        });

    }

}

auth.service.ts

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

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

import { AuthService, AuthResponseData } from './auth.service';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-auth',

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

  styleUrls: ['./auth.component.css']

})

export class AuthComponent implements OnInit {

  isLoginMode = true;

  isLoading = false;

  error: string = null;

  constructor(private authService: AuthService) { }

  ngOnInit() {

  }

  onSwithMode(){

    this.isLoginMode = !this.isLoginMode;

  }

  onSubmit(form: NgForm){

    if(!form.valid){

      return;

    }

    this.error = null;

    const email = form.value.email;

    const password = form.value.password;

    this.isLoading=true;

    let authObs: Observable<AuthResponseData>;

    if(this.isLoginMode){

      authObs = this.authService.login(email, password);

    }else{

      authObs = this.authService.signUp(email, password);

    }

   authObs.subscribe(

    responseData =>{

      console.log(responseData);

      this.isLoading=false;

    }, errorMessage =>{

      console.log(errorMessage);

      this.error = errorMessage;

      this.isLoading=false;

    }

  );

    form.reset();

  }

}

Auth.component.ts

1. Common error handling for login and signup

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

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

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    constructor(private http: HttpClient){}

    signUp(email: string, password: string){

       return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }

        ).pipe(

            catchError(this.handleError)

        );

    }

    login(email: string, password: string){

        return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }).pipe(

            catchError(this.handleError)

        );

    }

    private handleError(errorRes: HttpErrorResponse){

        let errormessage = 'An unknown error occorred !'

                if(!errorRes.error || !errorRes.error.error){

                    return throwError(errormessage);

                }

                switch(errorRes.error.error.message){

                    case 'EMAIL\_EXISTS':

                        errormessage = 'The email address is already in use by another account.';

                        break;

                    case 'EMAIL\_NOT\_FOUND':

                         errormessage = 'Invalid Email ID';

                         break;

                    case 'INVALID\_PASSWORD':

                         errormessage = 'Invalid Password';

                         break;

                }

                return throwError(errormessage);

    }

}

auth.service.ts

1. Creating and Sorting User Data

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

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

import { catchError, tap } from 'rxjs/operators';

import { throwError, Subject } from 'rxjs';

import { User } from './user.model';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    user = new Subject<User>();

    constructor(private http: HttpClient){}

    signUp(email: string, password: string){

       return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }

        ).pipe(

            catchError(this.handleError), tap(resData =>{

            this.handleAuthentication(resData.email, resData.localId, resData.idToken, +resData.expiresIn);

            })

        );

    }

    login(email: string, password: string){

        return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }).pipe(

            catchError(this.handleError), tap(resData =>{

                this.handleAuthentication(resData.email, resData.localId, resData.idToken, +resData.expiresIn);

                })

        );

    }

    private handleAuthentication(email: string, userId: string, token: string, expiresIn: number){

        const expirationDate = new Date(new Date().getTime() + expiresIn \* 1000);

        const user = new User(email, userId, token, expirationDate);

        this.user.next(user);

    }

    private handleError(errorRes: HttpErrorResponse){

        let errormessage = 'An unknown error occorred !'

                if(!errorRes.error || !errorRes.error.error){

                    return throwError(errormessage);

                }

                switch(errorRes.error.error.message){

                    case 'EMAIL\_EXISTS':

                        errormessage = 'The email address is already in use by another account.';

                        break;

                    case 'EMAIL\_NOT\_FOUND':

                         errormessage = 'Invalid Email ID';

                         break;

                    case 'INVALID\_PASSWORD':

                         errormessage = 'Invalid Password';

                         break;

                }

                return throwError(errormessage);

    }

}

auth.service.ts

1. Reflecting the auth state in UI

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

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

import { catchError, tap } from 'rxjs/operators';

import { throwError, Subject } from 'rxjs';

import { User } from './user.model';

@Injectable({providedIn: 'root'})

export class AuthService{

    user = new Subject<User>();

    constructor(private http: HttpClient){}

    login(email: string, password: string){

        return this.http.post<AuthResponseData>(

        'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyCvRzcCojbrjXNnovnjm9ksnNn2aLlNiAs',

        {

            email: email,

            password: password,

            returnSecureToken: true

        }).pipe(

            catchError(this.handleError), tap(resData =>{

                this.handleAuthentication(resData.email, resData.localId, resData.idToken, +resData.expiresIn);

                })

        );

    }

    private handleAuthentication(email: string, userId: string, token: string, expiresIn: number){

        const expirationDate = new Date(new Date().getTime() + expiresIn \* 1000);

        const user = new User(email, userId, token, expirationDate);

        this.user.next(user);

    }

auth.service.ts

export class HeaderComponent implements OnInit, OnDestroy

{

    isAuthenticated = false;

    private userSub: Subscription;

    constructor(private dataStoreageService: DataStoreageService, private authService: AuthService){}

    onSaveData(){

      this.dataStoreageService.storeRecipe();

    }

    ngOnInit(){

        this.userSub = this.authService.user.subscribe( user =>{

            if(user){

                //this.isAuthenticated = !user ? false : true;

                this.isAuthenticated = !!user;

                console.log(!user);

                console.log(!!user);

            }

        })

    }

    onFetchData()

    {

        this.dataStoreageService.fetchData().subscribe();

    }

    ngOnDestroy()

    {

        this.userSub.unsubscribe();

    }

}

Header.component.ts

1. Adding token to outgoing request

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

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

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

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

import { map, tap, take, exhaustMap } from 'rxjs/operators';

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

@Injectable({providedIn: 'root'})

export class DataStoreageService{

constructor(private http: HttpClient, private receipeService: RecipeService, private authService: AuthService ){}

storeRecipe(){

    const recipes = this.receipeService.getRecipes();

    this.http

    .put('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json', recipes)

    .subscribe(response => {

        console.log(response);

    });

}

fetchData(){

   return this.authService.user.pipe(take(1), exhaustMap(user =>{

    return this.http

    .get<Recipe[]>('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json',

    {

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

    });

    }),

    map(recipes => {

        return recipes.map(recipe => {

            return {...recipe, ingredients: recipe.ingredients ? recipe.ingredients : []

            };

        });

    }),

    tap(recipes =>{

        this.receipeService.setRecipes(recipes);

    })

    );

}

}

Data-storage.service.ts

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

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

import { catchError, tap } from 'rxjs/operators';

import { throwError, BehaviorSubject } from 'rxjs';

import { User } from './user.model';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    user = new BehaviorSubject<User>(null);

}

auth.service.ts

1. Attaching the token with interceptor

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

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

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

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

import { map, tap, take, exhaustMap } from 'rxjs/operators';

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

@Injectable({providedIn: 'root'})

export class DataStoreageService{

constructor(private http: HttpClient, private receipeService: RecipeService, private authService: AuthService ){}

storeRecipe(){

    const recipes = this.receipeService.getRecipes();

    this.http

    .put('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json', recipes)

    .subscribe(response => {

        console.log(response);

    });

}

fetchData(){

   return this.authService.user.pipe(take(1), exhaustMap(user =>{

    return this.http

    .get<Recipe[]>('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json',

    {

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

    });

    }),

    map(recipes => {

        return recipes.map(recipe => {

            return {...recipe, ingredients: recipe.ingredients ? recipe.ingredients : []

            };

        });

    }),

    tap(recipes =>{

        this.receipeService.setRecipes(recipes);

    })

    );

}

}

Data-storage.service.ts

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

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

import { catchError, tap } from 'rxjs/operators';

import { throwError, BehaviorSubject } from 'rxjs';

import { User } from './user.model';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    user = new BehaviorSubject<User>(null);

}

auth.service.ts

1. Adding Logout

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

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

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

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

import { map, tap, take, exhaustMap } from 'rxjs/operators';

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

@Injectable({providedIn: 'root'})

export class DataStoreageService{

constructor(private http: HttpClient, private receipeService: RecipeService, private authService: AuthService ){}

storeRecipe(){

    const recipes = this.receipeService.getRecipes();

    this.http

    .put('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json', recipes)

    .subscribe(response => {

        console.log(response);

    });

}

fetchData(){

   return this.authService.user.pipe(take(1), exhaustMap(user =>{

    return this.http

    .get<Recipe[]>('https://ng-project-recipe-book-2ba4a.firebaseio.com/recipes.json',

    {

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

    });

    }),

    map(recipes => {

        return recipes.map(recipe => {

            return {...recipe, ingredients: recipe.ingredients ? recipe.ingredients : []

            };

        });

    }),

    tap(recipes =>{

        this.receipeService.setRecipes(recipes);

    })

    );

}

}

Data-storage.service.ts

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

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

import { catchError, tap } from 'rxjs/operators';

import { throwError, BehaviorSubject } from 'rxjs';

import { User } from './user.model';

export interface AuthResponseData{

    idToken: string,

    email: string,

    refreshToken: string,

    expiresIn: string,

    localId: string,

    registered?:boolean

}

@Injectable({providedIn: 'root'})

export class AuthService{

    user = new BehaviorSubject<User>(null);

}

auth.service.ts

1. Store data in local storage

 localStorage.setItem('userData', JSON.stringify(user)); //check in console -> application -> local storage

1. Auto login

logout() {

this.user.next(null);

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

localStorage.removeItem('userData');

if (this.tokenExpirationTimer) {

clearTimeout(this.tokenExpirationTimer);

}

this.tokenExpirationTimer = null;

  }

1. Auto logout

  autoLogout(expirationDuration: number) {

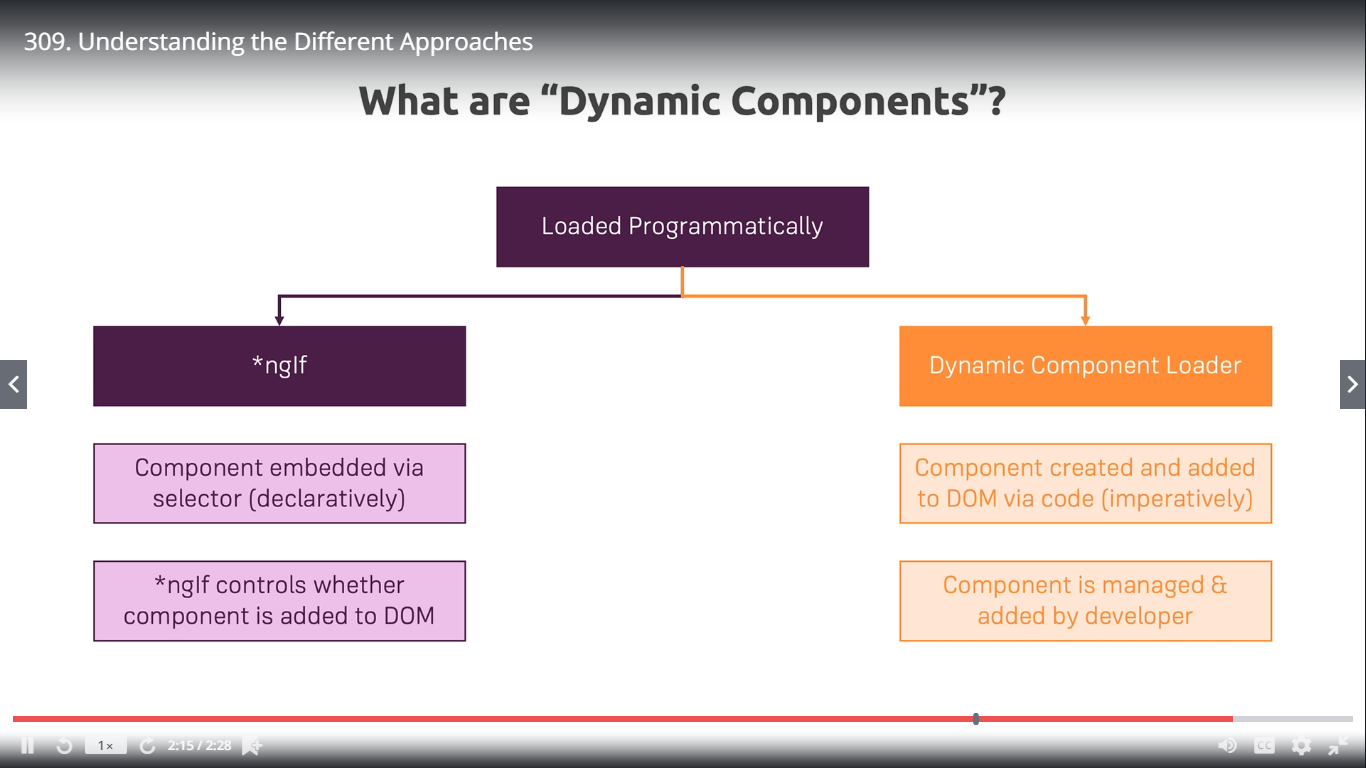
    this.tokenExpirationTimer = setTimeout(() => {

      this.logout();

    }, expirationDuration);

  }

1. Dynamic component



1. Alert box using ngIf

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

@Component({

    selector: 'app-alert',

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

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

})

export class AlertComponent{

    @Input() message: string;

    @Output() close = new EventEmitter<void>();

    onClose(){

        this.close.emit();

    }

}

alert.component.ts

<div class="backdrop" (click)="onClose()"></div>

<div class="alert-box">

   <p>{{ message }}</p>

   <div class="alert-box-actions">

        <button class="btn btn-primary" (click)="onClose()">Close</button>

   </div>

</div>

alert.component.html

  <app-alert [message]="error"  \*ngIf="error" (close)="onErrorHandle()"></app-alert>

page.component.html

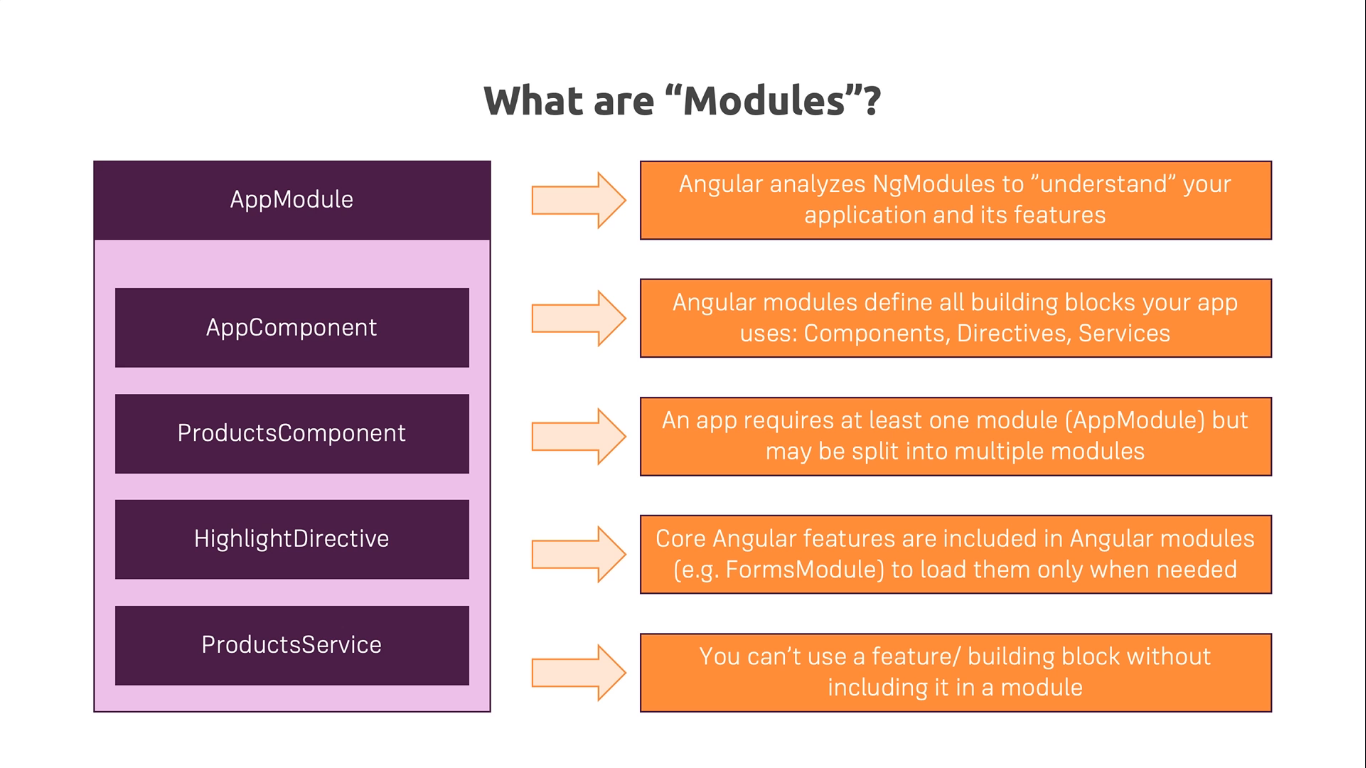
  onErrorHandle(){

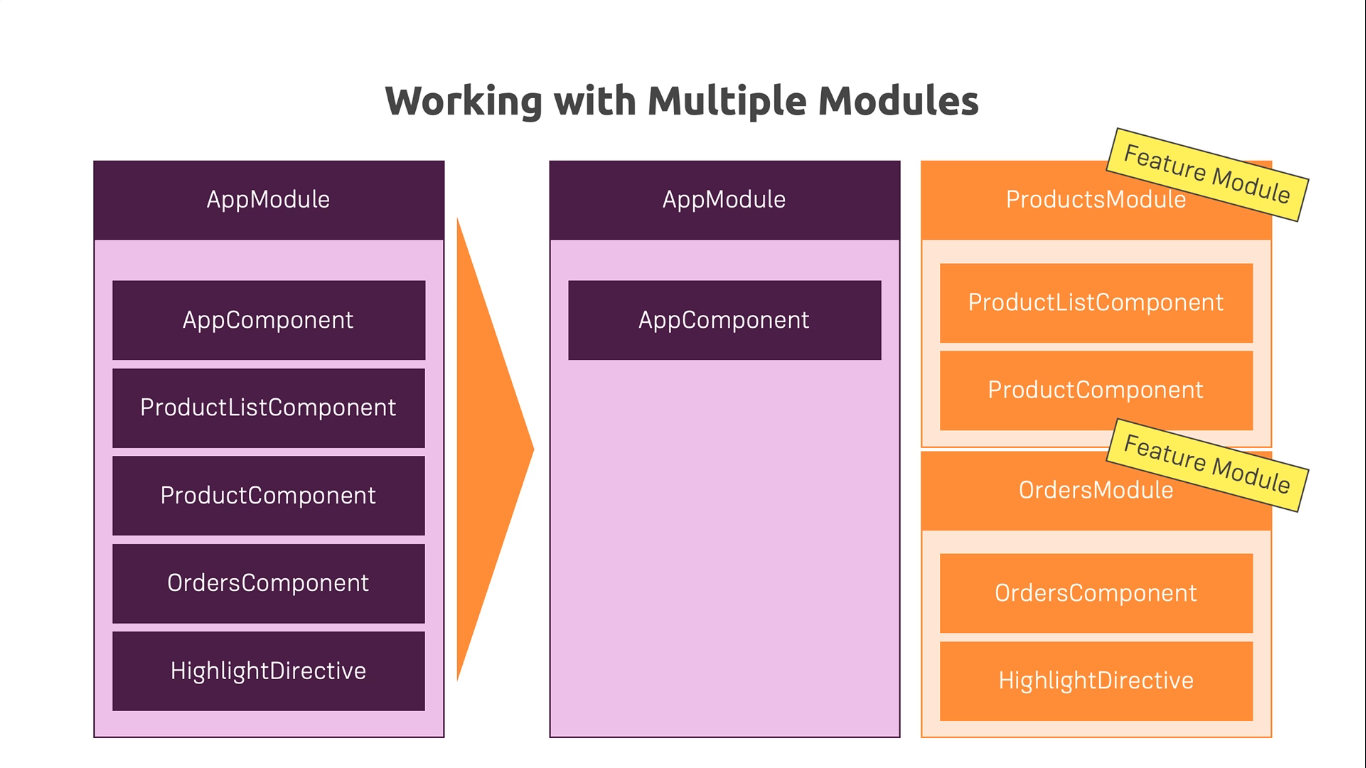
    this.error = null;

  }

page.component.ts

1. Dynamically creating component inside the code
2. NgModuls and optimization





1. Split app into multiple Modules

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

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

import { RecipeListComponent } from './recipe-list/recipe-list.component';

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

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

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

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

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

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

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

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

@NgModule({

    declarations:[

        RecipesComponent,

        RecipeListComponent,

        RecipeDetailComponent,

        RecipeItemComponent,

        RecipeStartComponent,

        RecipeEditComponent

    ],

    imports:[

        CommonModule,  // instead of BrowserModule

        FormsModule,

        ReactiveFormsModule,

        HttpClientModule,

        AppRoutingModule

    ],

    exports:[ // no need if we spit routing module also

        RecipesComponent,

        RecipeListComponent,

        RecipeDetailComponent,

        RecipeItemComponent,

        RecipeStartComponent,

        RecipeEditComponent

    ]

})

export class RecipesModule{}

recipes.module.ts

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

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

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

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

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

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

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

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

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

import { RecipesModule } from './recipes/recipes.module';

@NgModule({

  imports: [

    RecipesModule

  ],})

app.module.ts

1. Split routing module into multiple routing Modules

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

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

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

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

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

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

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

import { RecipesResolverService } from './recipes-resolver.service';

const routes: Routes = [

    {

        path: 'recipes',

        component: RecipesComponent,

        canActivate: [AuthGuard],

        children: [

          { path: '', component: RecipeStartComponent },

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

          {

            path: ':id',

            component: RecipeDetailComponent,

            resolve: [RecipesResolverService]

          },

          {

            path: ':id/edit',

            component: RecipeEditComponent,

            resolve: [RecipesResolverService]

          }

        ]

      }

];

@NgModule({

    imports: [RouterModule.forChild(routes)],

    exports: [RouterModule]

})

export class RecipesRoutingModule{}

recipe-routing.module.ts

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

import { RecipesRoutingModule } from './recipes-routing.module';

  imports:[

        RecipesRoutingModule

    ],

})

recipes.module.ts

1. Feature module and routing module merged together

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

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

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

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

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

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

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

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

@NgModule({

    declarations:[

        ShoppingListComponent,

        ShoppingEditComponent

    ],

    imports:[

        RouterModule.forChild([

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

        ]),

        CommonModule,

        FormsModule,

        ReactiveFormsModule,

        HttpClientModule,

        AppRoutingModule

    ],

    exports:[

        ShoppingListComponent,

        ShoppingEditComponent

    ]

})

export class ShoppingListModule{}

1. Shared module

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

import { AlertComponent } from './alert/alert.component';

import { LoadingSpinnerComponent } from './loading-spinner/loading-spinner.component';

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

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

@NgModule({

    declarations: [

        AlertComponent, // remove this from app.module.ts

        LoadingSpinnerComponent, // remove this from app.module.ts

        DropdownDirective // remove this from app.module.ts

    ],

    imports:[

        CommonModule

    ],

    exports:[

        AlertComponent,

        LoadingSpinnerComponent,

        DropdownDirective,

        CommonModule

    ]

})

export class SharedModule {}

shared.module.ts

import { SharedModule } from '../shared/shared.module';

@NgModule({

    imports:[

        SharedModule

    ]

})

page.module.ts

1. The Core module / Optional

It is used to make the app module liner :

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

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

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

import { HTTP\_INTERCEPTORS } from '@angular/common/http';

import { AuthInterceptorService } from './auth/auth-interceptor.service';

@NgModule({

    providers:[

    ShoppingListService,

    RecipeService,

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptorService,

      multi: true

    }

    ]

})

export class CoreModule{}

core.module.ts

import { CoreModule } from './core.module';

@NgModule({

  imports: [

    CoreModule

  ], // Removed Providers

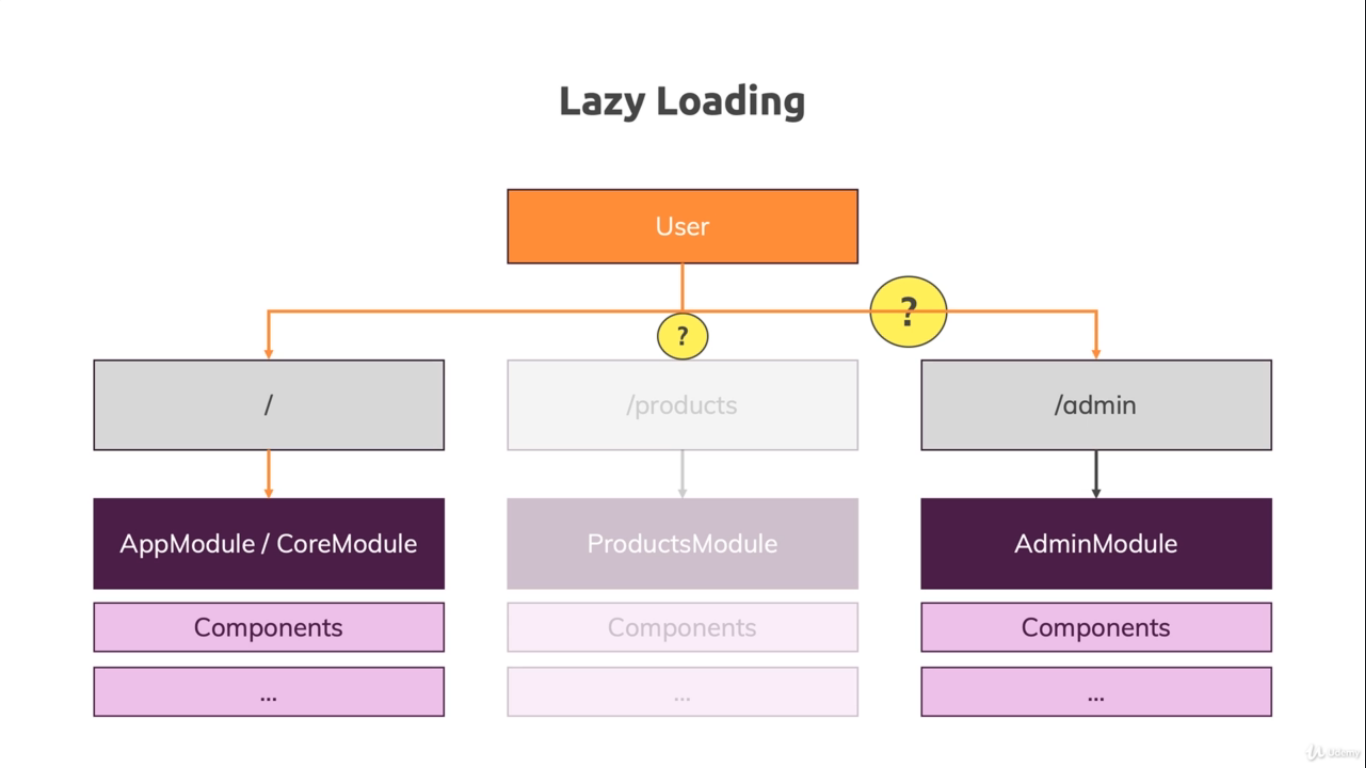
  bootstrap: [AppComponent]

})

export class AppModule {}

app.module.ts

1. Lazy Loading



Check loaded stuff in console - network :

It is necessary to restart server after adding lazy loading

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

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

const appRoutes: Routes = [

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

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

    path: 'recipes', loadChildren: () => import('./recipes/recipes.module').then(m => m.RecipesModule)

},

  {

    path: 'shopping-list',

    loadChildren: './shopping-list/shopping-list.module#ShoppingListModule'

  },

  {

    path: 'auth',

    loadChildren: './auth/auth.module#AuthModule'

  }

];

@NgModule({

  imports: [

    RouterModule.forRoot(appRoutes, { preloadingStrategy: PreloadAllModules })

  ],

  exports: [RouterModule]

})

export class AppRoutingModule {}

app-routing.module.ts

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

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

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

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

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

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

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

import { RecipesResolverService } from './recipes-resolver.service';

const routes: Routes = [

  {

    path: '',

    component: RecipesComponent,

    canActivate: [AuthGuard],

    children: [

      { path: '', component: RecipeStartComponent

 },

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

      {

        path: ':id',

        component: RecipeDetailComponent,

        resolve: [RecipesResolverService]

      },

      {

        path: ':id/edit',

        component: RecipeEditComponent,

        resolve: [RecipesResolverService]

      }

    ]

  }

];

@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class RecipesRoutingModule {}

recipe-routing.module.ts

@NgModule({

  declarations: [AppComponent, HeaderComponent],

  imports: [

    BrowserModule,

    HttpClientModule,

    AppRoutingModule,

    SharedModule,

// RecipeModule, //It should not be included

    CoreModule

  ],

  bootstrap: [AppComponent],

  // providers: [LoggingService]

})

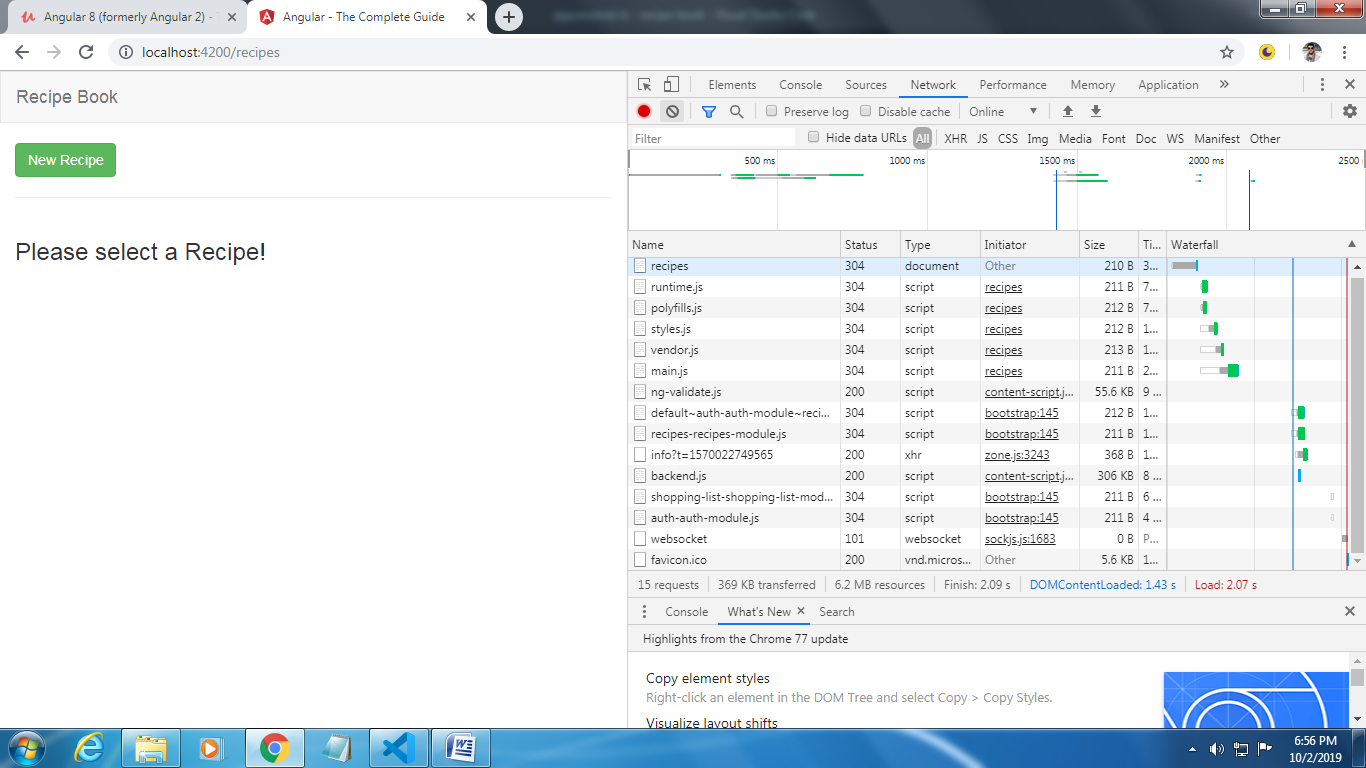
export class AppModule {}

app.module.ts

   // "module": "es2015",

    "module": "esnext",

tsconfig.json



1. Pre Loading Stategy

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

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

const appRoutes: Routes = [

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

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

    path: 'recipes',

    loadChildren: () => import('./recipes/recipes.module').then(m => m.RecipesModule)

},

  {

    path: 'shopping-list',

    loadChildren: () => import('./shopping-list/shopping-list.module').then(m => m.ShoppingListModule)

  },

  {

    path: 'auth',

    loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule)

  }

];

@NgModule({

  imports: [

    RouterModule.forRoot(appRoutes, { preloadingStrategy: PreloadAllModules })

  ],

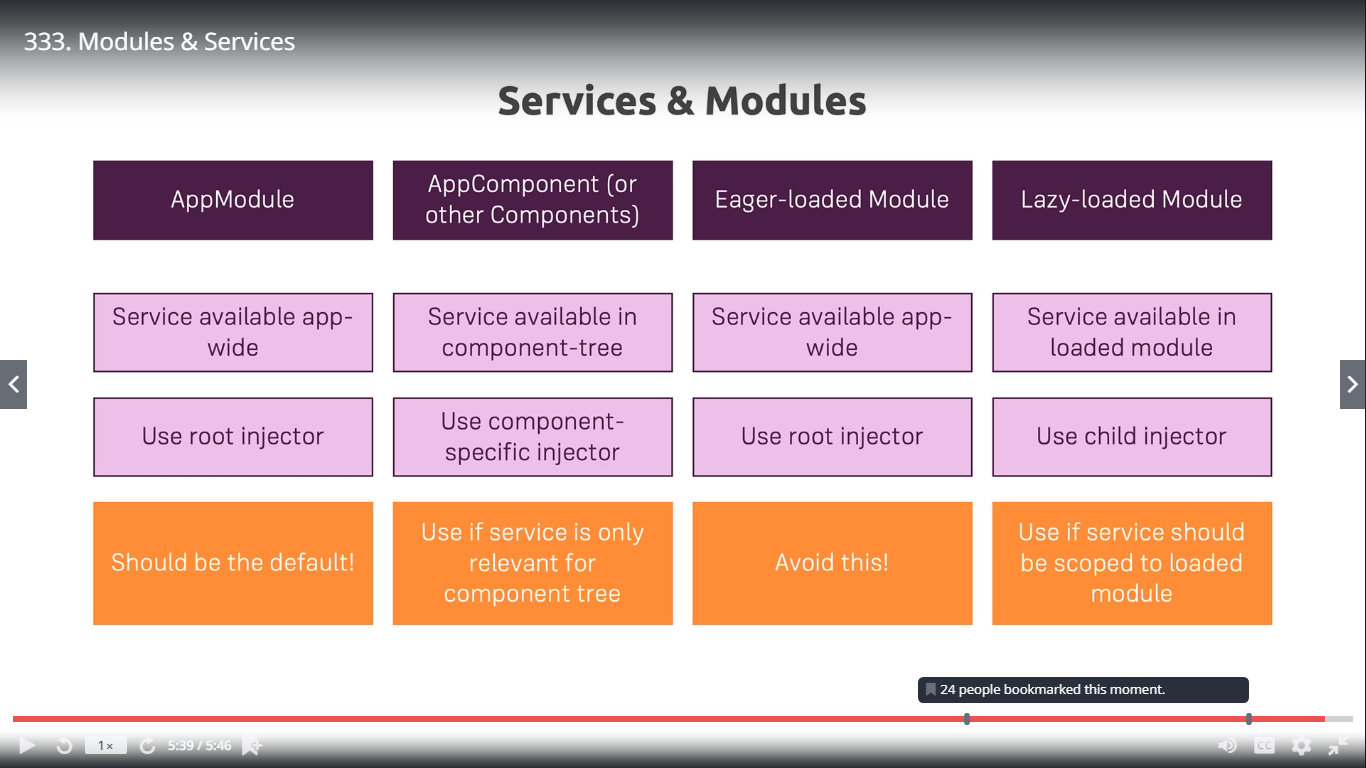
  exports: [RouterModule]

})

export class AppRoutingModule {}

app-routing.module.ts

1. Modules and services



1. Loading Services differently

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

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

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

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

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

import { SharedModule } from '../shared/shared.module';

import { LoggingService } from '../logging.service';

@NgModule({

  declarations: [ShoppingListComponent, ShoppingEditComponent],

  imports: [

    FormsModule,

    RouterModule.forChild([{ path: '', component: ShoppingListComponent }]),

    SharedModule

  ],

  providers: [LoggingService] // This will create different instance

})

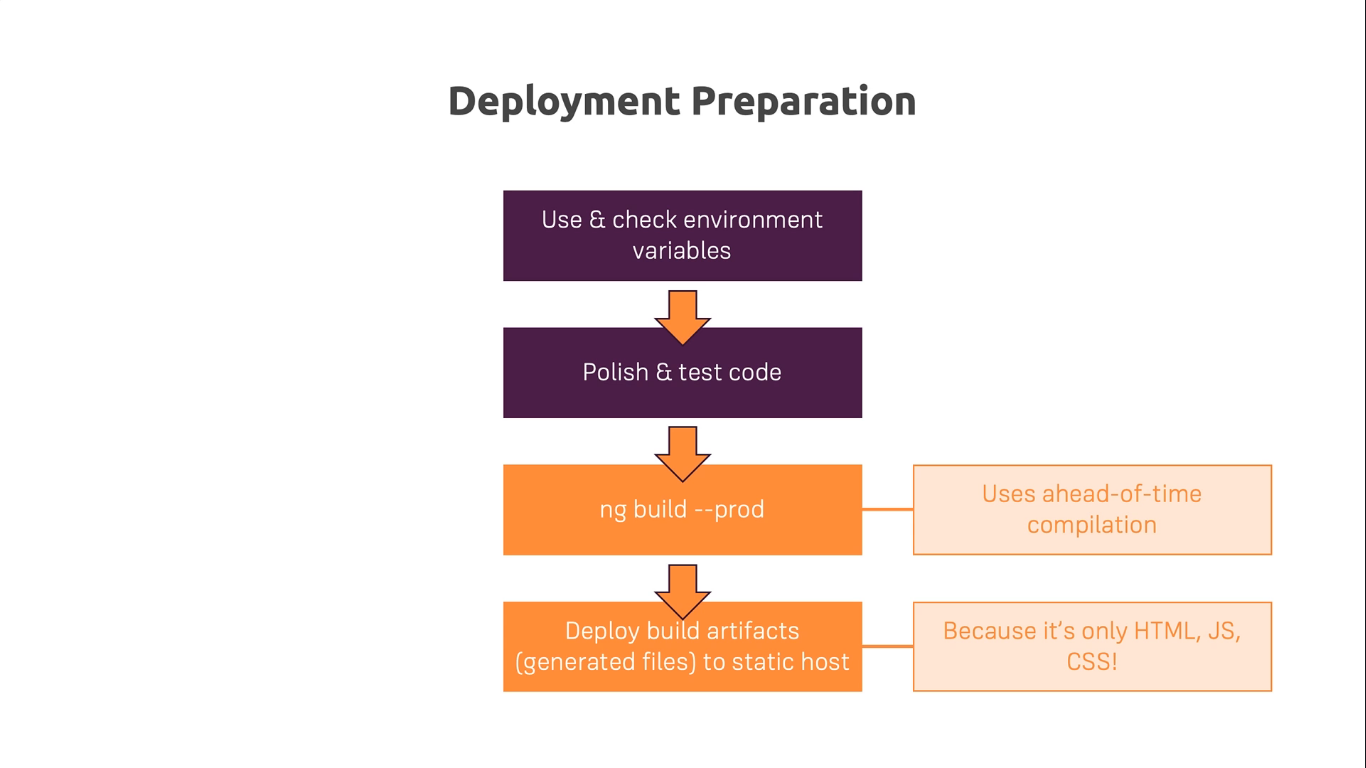
export class ShoppingListModule {}

Shopping-list.module.ts

1. Ahead –of-Time (AoT) vs Just in Time (JiT) compilation (Optimization)

Ng build –prod // for Ahead of Time

1. Deploying Angular App



1. Using Environment Variables

Define variables in environment.ts

1. Deploying project on firebase

npm install -g firebase-tools

firebase login

firebase init //should be in angular project

Select hosting form given options

Select project

Select public directirt -> dist/foldername

Single page app? yes

Existing HTML file? No

Firebase deploy

Get the url and enter it in the browser

1. Server routing vs browser routing

Reference

https://academind.com/learn/angular/angular-q-a/#how-to-fix-broken-routes-after-deployment

1. Unit Testing

Why Unit Test?

Guard against breaking changes

Analyze code Behavior ( Expected and unexpected )

Reveal design mistakes

1. Promise

That is a promise. A promise has 3 states. They are:

1. **Pending:** You don't know if you will get that phone
2. **Fulfilled:** Mom is happy, she buys you a brand new phone
3. **Rejected:** Your mom is happy, she withholds the phone
4. Dependency Injection

* **Pending:** You don't know if you will get that phone
* **Fulfilled:** Mom is happy, she buys you a brand new phone
* **Rejected:** Your mom is happy, she withholds the phone

1. How to install npm for existing project at another sysem
   1. Go inside the project
   2. Npm install
   3. ng update
   4. npm update
   5. Npm start
   6. npm install --save-dev @angular-devkit/build-angular (if required)
   7. npm uninstall -g angular-cli @angular/cli
   8. npm cache clean –force
   9. npm install -g @angular/cli
2. Host listning
3. Host Binding
4. Renderer
5. The Spread Syntax
6. CanActivate / Can Deactivate / Resolver
7. HttpClient & Http Services to Consume RESTful API

<https://www.positronx.io/angular-7-httpclient-http-service/>

1. Topics need to be revised