7. Client Side Authentication Service - API Design

Here we will implement signup functionality. We will send data to backend, create new user.

For this we defined auth service, in service folder. In this service will allow us to perform operations like- signup, signin, signout and it will also have data of user. depending upon presence of data of the user, we will display logout or signin,signup buttons. Then we defined user model.

In this auth service we defined a variable which is of type Observable which returns the data which is of type user.

Any part of application that needs user data , need to have AuthService injected and subscribe to this $user observable

Auth.service-

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

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

import { IfObservable } from 'rxjs/observable/IfObservable';

import { User } from '../model/user';

import { Observable } from 'rxjs/Observable';

@Injectable()

export class AuthService {

user$: Observable<User>;

constructor(private http: HttpClient) { }

}

Before implementing this observable lets see how this observable will be used by other parts of our application . we are going to see whether it is good idea to consume this observable directly or we should derive certain observables from it, and consume those instead.

8)Client Authentication Service- Design and Implementation

Lets continue designing api of auth service and see how we can use it in other parts of app to see if user is logged in or not without code repetition.

We injected our service in app.component. the we get the $user variable fro service. Now we will use this variable to show buttons like login,logout,signup etc.

Right now we are shwing all the buttons.

<li>

<a routerLink="/lessons">Lessons</a>

</li>

<li>

<a routerLink="/signup">Sign Up</a>

</li>

<li>

<a routerLink="/login">Login</a>

</li>

<li>

<a>Logout</a>

</li>

This could cause some flickering effects because depending upon timming of initialization of auth service we dnt know which buttons to show. We might fall in situation where we show all buttons then one dsappaears. To avoid this we will render these buttons only when we will have user information. So we do this-

<ng-container \*ngIf="users$ | async as user">

</ng-container>

We use ng-container so that we can wrap all other tags and then apply \*ngIf on it. We use ngIf to test for presence of data. We use async pipe to subscribe to user$ observable. Now value emitted by user$ observable will be simply called user. That will be our user data.now we want to show or hide some buttons depending upon presence of user data.

<ng-container \*ngIf="users$ | async as user">

<li>

<a routerLink="/lessons">Lessons</a>

</li>

<li>

<a routerLink="/signup">Sign Up</a>

</li>

<li>

<a routerLink="/login">Login</a>

</li>

<li>

<a>Logout</a>

</li>

</ng-container>

App.component.ts-

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

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

import { Observable } from 'rxjs/Observable';

import { User } from './model/user';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

user$: Observable<User>;

constructor(private authService: AuthService) {

this.user$ = this.authService.user$;

}

ngOnInit() {

}

}

Lets go tour service now-

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

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

import { IfObservable } from 'rxjs/observable/IfObservable';

import { User } from '../model/user';

import { Observable } from 'rxjs/Observable';

@Injectable()

export class AuthService {

user$: Observable<User>;

constructor(private http: HttpClient) { }

signUp(email: string, password: string) {

}

}

This service has not yet emitted any value. Actually variable user$ is undefined. So we define one special type of user and call it anonymous user, which is value emitted by this observable if we are not logged in.

export const ANONYMOUS\_USER : User = {

id: undefined,

email: ''

};

we are going to define here this observable now. The way are going to define it is, we want to have a way of emitting new value for this observable. So we define a subject, that we keep private to this class. We name this variable subject, then we say that it is special type of subject , whenever we subscribe to it , will emit last value. So it is of type behaviorSubject. Its initial value(which is mandatory value) will be ANONYMOUS\_USER. As this type of subject always emits the last value, we need to provide atleast one initial value.

private subject = new BehaviorSubject<User>(ANONYMOUS\_USER);

now our user$ observable here will be derived from this subject.

user$: Observable<User> = this.subject.asObservable();

we use asObservable method to derive observable from it.

So whenever app.component subscribes to this user observable(in html file), it is going to get emitted first ANNONYMOUS\_USER as value of user variable (variable in app.component.html) and then on subsequent value of user Observable, we will receive the actual user data.

App.compoent.ts-

ngOnInit() {

this.user$ = this.authService.user$;

}

In html(we are subscribing to observable here)-

<ng-container \*ngIf="users$ | async as user">

so based on user data we could start writing condition suchs as show logout button only if user.id is undefined.

The problem with this approach would be, there might be other places in application where we also want to know whether user is logged in or not. So we do not want to sprinkle condition everywhere- **user.id == undefined or** test for tuethfullness of user.id. so instead,what we are going to do is in auth.service , we define other Observable called isLoggedIn. It is derived from user$observable but that is transparent for consumer of information.. We use map observable , we take userObservable and we map its value. Right now we do not have map operator , so we add this in app.module-

import 'rxjs/add/operator/map';

now we use map operator to test for truthfulness of user variable. So we know ,(because we are inside internal implementation of this service) that this will also emit a value-

isLoggedIn$: Observable<boolean> = this.user$.map(user => user.id);

in worst case it will emit the value of ANONYMOUS\_USER.

So we dnt need to test for truthfulness of user. The problem with this would be that we will be returning a numerical value. If we want to convert a number into result of its truthfulness-

isLoggedIn$: Observable<boolean> = this.user$.map(user => !!user.id);

when we apply !! to a number it will always return avalue that corresponds to truthfulness of expression to which it is applied. then we define other observable-

isLoggedOut$: Observable<boolean> = this.isLoggedIn$.map(isLoggedIn => !isLoggedIn);

auth.ervice.ts-

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

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

import { IfObservable } from 'rxjs/observable/IfObservable';

import { User } from '../model/user';

import { Observable } from 'rxjs/Observable';

import { BehaviorSubject} from 'rxjs/BehaviorSubject';

export const ANONYMOUS\_USER : User = {

id: undefined,

email: ''

};

@Injectable()

export class AuthService {

private subject = new BehaviorSubject<User>(ANONYMOUS\_USER);

user$: Observable<User> = this.subject.asObservable();

isLoggedIn$: Observable<boolean> = this.user$.map(user => !!user.id);

isLoggedOut$: Observable<boolean> = this.isLoggedIn$.map(isLoggedIn => !isLoggedIn);

constructor(private http: HttpClient) { }

signUp(email: string, password: string) {

}

}

Now we go to app.compoennt, there we define IsLoggedIn and IsLoggedout observables. Then we assigned same observable from auth services to these.

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

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

import { Observable } from 'rxjs/Observable';

import { User } from './model/user';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

isLoggedIn$: Observable<boolean>;

isLoggedOut$: Observable<boolean>;

user$: Observable<User>;

constructor(private authService: AuthService) {

}

ngOnInit() {

this.isLoggedIn$ = this.authService.isLoggedIn$;

this.isLoggedIn$ = this.authService.isLoggedOut$;

}

}

Then we go to app.component.html and use these observables. We want logout button to be showed only if user is logged in. and sign up and sign in should be shown if user is logged out. So we use these like this-

<li>

<a routerLink="/lessons">Lessons</a>

</li>

<li \*ngIf="isLoggedOut$ | async">

<a routerLink="/signup">Sign Up</a>

</li>

<li \*ngIf="isLoggedOut$ | async">

<a routerLink="/login">Login</a>

</li>

<li \*ngIf ="isLoggedIn$ | async">

<a>Logout</a>

</li>

Now if we reload our application we will not see logout button, because use r is not logged in. but signin and sigout buttons are shown/

.

9) The New Angular HTTP Client - Doing a POST Call To The Server

Here we send a request to backend using signup method of auth service. This mef]thod will be called when we click on signup button. Since name of parameters that we pass to function and name of proeprties of object that we want to send to backend is same, we use this short notation-

signUp(email: string, password: string) {

return this.http.post<User>('',{email, password});

}

When this http request is sent we will get a response. In response body we accept a object of type user. So we specify here, in post request.

Now because we are returning the http post observabe and we would like to make sure that we never run into situations where this was called multiple times , so to avoid to create multiple users. This is typical example of http call that we want to make sure that we dnt multiple times accidentally, we will use another operator that was specifically introduced to cover caching of http request. It is shareReplay operator. To use it, add this import to app.module-

import 'rxjs/add/operaotr/shareReplay';

the advantage of this new operator is that resulting observable is still retryable but result of http post is still being cached. Then we subscribe to this observable in signup component.

Now in auth Service, when we get the data back from backend, we also want to emit it here t any parts of application that would be notified that new user data is avalaible. For this we use do operator. So using subject we broadcast the data that we get back from backend. We used next function. now any part of application that is subscribed to user$ or any of its derived observables, will receive new value for that observable.

Authservice-

export class AuthService {

private subject = new BehaviorSubject<User>(ANONYMOUS\_USER);

user$: Observable<User> = this.subject.asObservable();

isLoggedIn$: Observable<boolean> = this.user$.map(user => !!user.id);

isLoggedOut$: Observable<boolean> = this.isLoggedIn$.map(isLoggedIn => !isLoggedIn);

constructor(private http: HttpClient) { }

signUp(email: string, password: string) {

return this.http.post<User>('/api/signup',{email, password})

.shareReplay()

.do( user => this.subject.next(user));

}

}

Signup.component.ts-

signUp() {

const val = this.form.value;

if(val.email && val.password && val.password == val.confirm) {

this.authService.signUp(val.email, val.passowrd)

.subscribe(

() => console.log('User Created Sucessfully'),

() => console.log('Error occured')

);

}

}

With this we have client side authentication service in place, if we try it now we will get error because our backemd is not implemented yet.

10) User Sign Up Server-Side Implementation in Express

We go to server.ts file. there we define new route for ‘/api/signup’

app.route('/api/signup')

.post(createUser);

Now we need to create a new route handler- createUser. Sow e define anew file called create-user.ts.

In tis file we define and export function called createUser.

Then in database.ts we define a new const –

export const USERS = {};

to increase type safety we define type for user . now USERS const in database.ts will be map. This is how we define the map.

We define type of key to be number, then we define the type of value which isDbUser.

export const USERS: {[key: number] : DbUser} = {};

with this we will have better auto completion for our application.

Now lets move to database.ts file. here we define craeteUser function, we create a user and return it.

Database.ts-

class InMemoryDatabase {

userCounter = 0;

readAllLessons() {

return \_.values(LESSONS);

}

createUser(email: string, password: string) {

const id = this.userCounter++;

const user:DbUser = {

id,

email,

password

};

USERS[id] = user;

return user;

}

}

Now lets go to ur route file, create-user.ts-

import \* as \_ from 'lodash';

import {LESSONS, USERS} from "./database-data";

import { DbUser } from './db-user';

class InMemoryDatabase {

userCounter = 0;

readAllLessons() {

return \_.values(LESSONS);

}

createUser(email: string, password: string) {

this.userCounter++;

const id = this.userCounter;

const user:DbUser = {

id,

email,

password

};

USERS[id] = user;

return user;

}

}

export const db = new InMemoryDatabase();

There we create user and we returned back id and email of it. Now run your code. Memoent you press sign in button, signup and signin buttons on menu should disappear.