Higher Order Array Functions JavaScript JS

A higher order function performs one or both of the following functionalities

- Takes one or more functions as parameter
- Returns a function as parameter

These are the higher order functions which we are interested in Discussing:

1. forEach()
2. map()
3. filter()
4. reduce()

1. forEach():

forEach() function is a higher order function which returns a call back function that runs for Each Element

```
array.forEach(
  (iterableElement, index, array) => {
    //Callback Function
  }
)
```

Mutates the original array (if the call back function does so) unlike remaining higher order functions

```
//forEach Method
const arr = [1,2,3];
//          0 1 2

let sum =0;
let maxVal = 0;


function calculateMaxValue (e,index,array){
       maxVal = Math.max((e*index),maxVal);
   }

// arr.forEach(
//     (e) => {
//         sum += Math.pow(e,2);
//     }

//     )

arr.forEach((e) => sum += Math.pow(e,2))
console.log(sum);


arr.forEach(calculateMaxValue);
```

Angular Authentication With JSON Web Tokens (JWT):

Complete Tutorial : [Angular Authentication With JSON Web Tokens (JWT): The Complete Guide]

JWT : JSON WEB TOKEN

Bearer Token :

```
{
        “sub” : “3454325678”
        “exp”: “1504699256”
}
```

The “sub” property contains the user identifier, and the “exp” property contains the expiration timestamp. This type of token is known as a Bearer Token, meaning that it identifies the user that owns it, and defines a user session.

The very first step for implementing JWT-based Authentication is to issue a bearer token and give it to the user, and that is the main purpose of a Login / Sign up page.

1. Capture Login Credentials in the component and pass them to the authService
2. Pass these credentials along with the url to strike the respective login API
3. Add .shareReplay() method to the end of http method
    a. We are calling “shareReplay” to prevent the receiver of this Observable from accidentally triggering multiple POST requests due to multiple subscriptions.
4. The goal is to validate the password and establish a session. If the password is correct, then the server will issue a bearer token.

Cookies are a browser data storage mechanism, a place where we can safely store a small amount of data.advantages and disadvantages of using cookies to store JWTs, when compared to other methods.

```
login(email:string, password:string ) {
    return this.http.post<User>('/api/login', {email, password})
        .do(res => this.setSession)
        .shareReplay();
}

private setSession(authResult) {
    const expiresAt = moment().add(authResult.expiresIn,'second');

    localStorage.setItem('id_token', authResult.idToken);
    localStorage.setItem("expires_at", JSON.stringify(expiresAt.valueOf()) );
}

logout() {
    localStorage.removeItem("id_token");
    localStorage.removeItem("expires_at");
}

public isLoggedIn() {
    return moment().isBefore(this.getExpiration());
}


isLoggedOut() {
    return !this.isLoggedIn();
}

getExpiration() {
    const expiration = localStorage.getItem("expires_at");
    const expiresAt = JSON.parse(expiration);
    return moment(expiresAt);
}
```

Modify loginService as

1) validateCredentials() :
    a) Takes the email and password and hits the signIn API
    b) Generates Token and userId on success
    c) Navigates to automations page
2) setToken():
    a) Takes the token obtained from the response
    b) Sets the token into localStorage
3) isLoggedIn():
    a) Obtains the token from the localStorage
    b) Checks whether the token is undefined/empty/null (if yes returns false else returns true)
4) logout():
    a) Removes the token from the local storage and navigates back to the login page

Guard is being validated from isLoggedIn() method

So we need to make sure isLoggedIn() method returns false as soon as the token expires(); this can be obtained from 3b