In the name of God

Parham Rezaei

Watchable main purpose is to store data’s’ about movies. Actually, it is movie library, that each user can store the data like movie title, movie description and the URL of the image of that movie. Users can log-in or sign-up on the app and after that watchable sent a get request API and load the user’s movies.

For log-in part it checks if the username or password is correct or not, if the user inputs were correct it goes to the home page, and if the user inputs were wrong a large green text will appears on the top of the page and said, “your username or password is wrong”. And, for the sign-up part, it sends a get request to check if the username is already existing or not and after that it will go to the user’s home page.

When the home page is loading, app sends a get request to our mock-api and reads the user’s movies and for each movie it will create a new card and insert the movie data in it.

There is a button called Add movies with pressing this button, a module will open on your screen, and you can add your movies title, short summary and the image of cover of movie’s URL (it is good to know that if you missed an input and didn’t type anything inside it app will show an error). And by clicking the “store file” button app will send a post request and it will post your data to the mock-api and after that it will refresh the page to read the movies again from mock-api.

Finally, there is a log-out button which by clicking on it you can log-out of your account, and it will take you back on log-in page.

And the most important thing that I have the honor to explain that is the watchable is sensitive on which person is logged-in that means the app database is different for each person and each person can have themselves movies.

Technologies:

I used mostly pure front-end technologies for this project. Like HTML, CSS, JavaScript. But I used a little from some other libraries like Axios ( used it once Just for fetching Data from Mock-Api), jQuery (For the animation of header of the home page when scrolling it down), Bootstrap ( I used it first for the add movies module but then I handle this part with <dialog>…</dialog> tag for it), Google Font API ( For adding font to the app), SASS ( For creating CSS file), and the very most important one Mock-api ( It is totally used for imitating a real API server by providing realistic responses to requests, where I send my api requests (You can see users mocks from https://6489722d5fa58521caaf9b75.mockapi.io/users )

Requirements:

Regularly you can start pure front-end project with opening index.html file. But now you should clone the files from “https://github.com/prhmrz/watchable.git” and open the project on VS-code and please remember that you should start the project with live server “ <https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer> ” extension of vs-code and it will handle page URLs. And it is nice to keep in mind that there are lots of fetch requests inside of the program so you should keep the internet on when you want to work with the app.

Explaining the functions inside the indexScript.js:

Line 4: formLoginSubmission(){

This function at firsts checks if inputs on log-in page are valid or not. If they are not valid it will show a warning message and if they are valid, it will send a get request for reading users.

}

Line 16: Arrow Function()=>{

This function will take response from request and return them in json format.

}

Line 19: Arrow Function()=>{

It will read all users from previous function, and it will check if the username and password are correct or not. If they were false, it will show a warning message. And if they were correct it will go to home page.

}

Line 34: Arrow Function()=>{

This function will get the errors and will log them inside of the console.

}

Line 42: formSignupSubmission(){

This function at firsts checks if inputs on sign up page are valid or not. If they are not valid it will show a warning message and if they are valid, it will send a get request for reading users.

}

Line 52: Arrow Function()=>{

This function will take response from request and return them in json format

}

Line 55: Arrow Function()=>{

It will read all users from previous function, and it will look for the username that user entered is available or not. If it was false, it will show a warning message. And if it was correct function will set a value of a variable to true.

}

Line 65: Arrow Function()=>{

It will send a post request to our mock-api with values of name, userName and userPassword

}

Line 80: Arrow Function()=>{

It will get responses from post request and make them json, and after that the function will set userName of the user on the localStorage of the user’s browser, and just after that app will change the url and it goes to home page.

}

Line 85: Arrow Function()=>{

This function will take response from request and return them in json format

}

Line 88: Arrow Function()=>{

This function will get the errors and will log them inside of the console.

}

Line 94: Arrow Function()=>{

This function will get the errors and will log them inside of the console.

}

Line 105: signedInHandler(){

It will check if the remember me check box is clicked or not. If it was checked it will set a variable to the local storage and set another variable to false. And if it was not, it will set a variable to the local storage and set another variable to false.

}

Line 114: loadingLoginPage(){

It will check if check box wasn’t checked it will clear local storage and otherwise it will change the url of the page to the home page. It means that if user didn’t use remember me check box whenever he navigates to the log-in page it will change the url of the page to the home page.

}

Explaining the functions inside the homeScript.js:

Line 2: signedInChecker(){

It will check the local storage if userName doesn’t exist it will change the url of the page to the log-in page.

}

Line 8: function (){

This part will handle the scrolling in the home page if user scroll more than 40 % it will add some styles to the class named top and library.

}

Line 22: Arrow Function(){

It will create a loading animation with three div and the main css and animation code is on the css folder. It will just create three div and add classes inside of that.

And after that it will send a get request to mock api. To read data about Movies.

}

Line 34: Arrow Function()=>{

This function will take response from request and return them in json format

}

Line 37: Arrow Function ()=>{

It will filter the movies that have username attribute and which the user name should be equaled to username of the user. And if the user doesn’t have any movies in his list a paragraph tag should be appeared on the screen that said, “your library is empty”.

}

Line 49: Arrow Function(){

It will create a card for each user’s movies.

}

Line 75: Arrow Function()=>{

This function will get the errors and will log them inside of the console.

}

Line 81: handleSubmit(e){

It will check if user fill all the inputs and the users did it properly it will send an api post request and it will send the image, title, details and the username of the user to the mock-api. And after that it will reload the page in order to read new movies and add them to the home page.

}

Line 100: logout(){

It will clear the local storage and it will navigate to the log-in page.  
}

Line 112: editPanel(e){

Select the parent element that is clicked, and it will delete it by sending a remove fetch api request. And after It if that the api sent succrsfull recived it will reload the page.

}