allUser.js

All users must be stored on local storage ( it can be any database but I use js’ localStorage function )

First, we need to create a class that defines all Users. This class have a constructor which contains an empty array (arguments will be empty because we just add a user to the users class ). And we have getAllUsers method which returns all users we have. For adding user we create addUser method which takes user argument ( “user” will be another object ) and push it to our users array after this we must set these items to localStorage so we first change the array to string and set it localstorage ( we use stringify for changing to string ). And restore all users every time when we reload the page so we can take new users. addUser method we change the array to string so now we need a reverse function that’s why we parse localStorage users items and push them to our users array which we define in the constructor. And we need a reset method to remove all users with one button. We use the removeItem function for this method

userModel.js

I create userModel class to define each user. The user class has a constructor which is empty.

User class include password, username, pattern and grid order which some of which must be encrypted.

That’s why I created encrypt method which takes a parameter and hashes it then change to a hex base. For this, I used the crypto module which Node JS have.

In the user classModel class we have assUsernameAndPassword which takes username, password and allUsers then check username exists or not. So it made a map and if the user exists change value to false then we check “exist” includes any false or not if it includes then return false if not code continues.as a result add a username and encrypted password to class and return true.

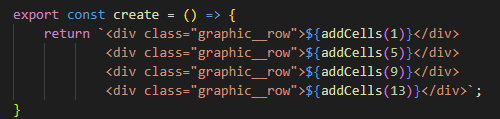
For level two and three we have addPattern and addGrid method they are simply adding encrypted code to the user model.

And we need to check this pattern, password or grid is valid or not so I create three methods (comparePatter, comparePassword and compareGrid ). They simply check inputs equal or not.

GridView.js

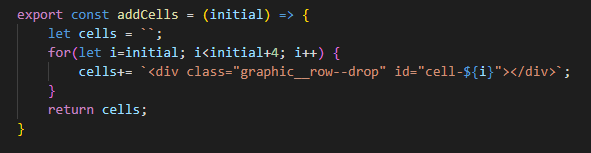
Due to webpack we must add images using import tag because when we start (in production or development mode) webpack detect and adding these pictures to the folder which we define in the config file.

Then we crate four row and four cell for each row.



For cell, I create addCells function which gets inputs and adds cell to each row

For this, I take input as the initial point and add 4 to this input and it will be my endpoint for the current row



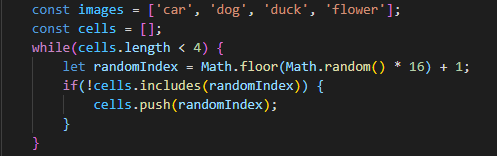
For adding images firstly I create an array that contains all four images ( car, dog, duck and flower )

Then I create an empty array and named it cells that will contain an index of these images with respect to order.

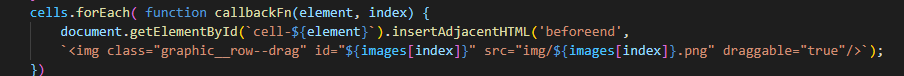
We have 16 cells that’s why I make a function that sets images randomly when we enter the third level.

I was using the math library for this. Math.random() return a random number between 0 and 1 but we need bigger numbers and they must be an integer so I multiplied it 16. Math.random()\*16 now it returns number between 0 and 15 but we have 16 cells so I need to add one, as a consequence, we got Math.floor(math.random()\*16)+1. Math.floor returns an integer value of numbers. However, we have

Four images but the above code work for one cell so we need a loop. I used a while loop, first creates a random index then check if this index not taken before then push this index to current images.



Then we need to add images to current cells that’s why I used forEach loop it takes the element and index of the element we use this argument on the <img> tag via insertAdhajacentHTML function



Base.js file was created to render all level one by one.

Elements object return button by using querySelector. loginTop and registerTop are the first two buttons on the header.

elementString returns element tag which we can use in js after.

“markups” has three components one, two and three these contain HTML template for each level.

Replacements object has six input for description ( one, log in, description – oneLD ) after we change them with placeholders which we wrote them in the markups templates.

Placeholders object contains elements that we use in the template and during the render, they change respectively to the replacements object.

Clear functions replace all HTML tag with an empty string. clearFields just remove inputs fields’ variable.

updatePattern takes current colour and updates them via concat function.

There are three functions for render each level renderOne, renderTwo and renderThree. All of them work in the same method. They replace button level and description check is login or register and adding new markup to the HTML page.

After all of these, we gather all of them together in index.js file which is our main file.