Refresher Assignment: Report

By Siddharth Sircar - 015384343

# Problem Statement

*SortYourLife* is a task management app. Users can log in and view their unique tasks, add more tasks, and mark them as complete. It has a profile page that shows user details and an option to connect with fellow users to collaborate with their tasks.

# Deployment URL

<http://ec2-3-144-121-83.us-east-2.compute.amazonaws.com:3000/>

# ****GitHub Link****

<https://github.com/siddharthsircar/CMPE-273-Refresher/>

# ****App Video****

<https://youtu.be/9U55ZuRa21c>

# JavaScript: Introduction to Topic

1. **LET, VAR and CONST:**

***LET:*** variable declared with LET have a block scope. They cannot be referenced outside that block.

***VAR:*** variable declared with VAR is defined throughout the program.

***CONST:*** is used to declare constant variables. These variables cannot be updated once declared.

1. **SPLIT:** is used to split a string based on some pattern.
2. **ARROW FUNCTION:** is a new way of declaring functions in ES6.
3. **INCLUDES:** is used to verify if a certain string is present in an array or another string. Returns True if found.
4. **REGEX:** regular expression is a sequence of characters that can be used as a validation/search pattern by following certain sequence conventions.
5. **OBJECT.ASSIGN:** can be used to modify an object by updating its values or concatenating a new object to it.
6. **CALLBACKS:** allows user to pass a function as an argument to another functions which is helpful when we want a function to call another function.
7. **PROMISE:** utilizes Callbacks in a different way. We do not pass a callback function as argument we attach it in the form of *.then()*. It has 2 outcomes- Success/Resolve or Failure/ Reject. It is used where there is a blocking code like making an API call.
8. **ASYNC / AWAIT:** is a better way of writing promises. Does the same thing.
9. **JSON.STRINGIFY:** is used to convert a JSON object into string.

## Code Snippet for above topics:

### login\_register.js

'use strict'

// Using CONST to declare constant variable which can not be updated later

const emailTxtbx = document.getElementById('email');

const nameTxtbx = document.getElementById('name');

const passwordTxtbx = document.getElementById('password');

const loginBtn = document.getElementById('login');

const registerBtn = document.getElementById('register');

// Using VAR

var firstName;

var lastName;

// Using SPLIT func to split full name.

// Using ARROW FUNCTION

const splitName = (name) => {

    // Use of LET to create block scope variable (Can not be accessed outside this block)

    let fullName = name.split(' ');

    fullName = [fullName[0], fullName[fullName.length - 1]]

    return fullName;

}

// console.log(fullName); will throw an ERROR

// Using AXIOS API call to validate user

function authenticateUser() {

    if (document.title === 'REGISTER') location.href = '../modules/login.html';

    else {

        axios.get('https://jsonplaceholder.typicode.com/users').then(response => {

            let userEmails = ['sid@gmail.com'];

            for (let i = 0; i < response.data.length; i++) {

                userEmails.push(response.data[i].email);

            }

            // Using LET to declare function variables

            let email = emailTxtbx.value;

            let password = passwordTxtbx.value;

            if (email.length <= 0) alert('Please enter email');

            else if (password.length <= 0) alert('Please enter password');

            else {

                if (validateEmail(email)) {

                    // Using INCLUDES to verify if the entered email exists in the registered email list.

                    if (userEmails.includes(email)) {

                        if (password === '1234') {

                            storeUserDetails(email);

                            location.href = '../modules/todo.html';

                        }

                        else alert('Incorrect Password!');

                    }

                    else alert('Email not registered')

                }

                else alert('Enter Valid Email.')

            }

        }).catch(error => {

            console.log(error);

            alert('Could not connect to database. ', error);

        });

    }

}

// Used Regular Expression to validate email format.

function validateEmail(email) {

    if (/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(email)) {

        return true;

    }

    return false;

}

// OBJECT

let userDeets = {

    'id': 1,

    'name': 'Siddharth Sircar',

    'email': 'sid@gmail.com',

    'address': {

        'city': 'San Jose',

        'zipcode': '95112',

        'geo': {

            'lat': '',

            'lng': ''

        }

    },

    'phone': '1-408-207-7389'

}

// Using ASYNC / AWAIT

async function getUserData(email) {

    let response = await fetch(`https://jsonplaceholder.typicode.com/users?email=${email}`);

    let data = await response.json();

    return data;

}

function storeUserDetails(email) {

    if (email === 'sid@gmail.com') {

        // Use JSON.stringify to convert JSON object into string

        let userData = JSON.stringify(userDeets);

        sessionStorage.setItem('user-data', userData);

    } else {

        // Using ASYNC/AWAIT

        getUserData(email).then((data) => {

            let apiResponse = data[0];

            // Using OBJECT.ASSIGN

            Object.assign(userDeets, {

                'id': apiResponse.id,

                'name': apiResponse.name,

                'email': `${email}`,

                'address': {

                    'city': `${apiResponse.address.city}`,

                    'zipcode': `${apiResponse.address.zipcode}`,

                    'geo': {

                        'lat': `${apiResponse.address.geo.lat}`,

                        'lng': `${apiResponse.address.geo.lng}`

                    }

                },

                'phone': `${apiResponse.phone}`

            })

            let userData = JSON.stringify(userDeets);

// Using SessionStorage

            sessionStorage.setItem('user-data', userData);

        })

    }

}

function registerUser() {

    if (document.title === 'LOGIN') {

        location.href = '../modules/register.html';

        return false;

    }

    else {

        // Using Promise

        registerPromise().then((message) => {

            let userData = JSON.stringify(userDeets);

            sessionStorage.setItem('user-data', userData);

            location.href = '../modules/todo.html';

            console.log(message);

        }).catch((error) => {

            alert(error);

        })

        return false;

    }

}

// Using PROMISE

function registerPromise() {

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

        let emailTxtbx = document.getElementById('email');

        let email = emailTxtbx.value;

        if (validateEmail(email)) {

            resolve('Registration Successful!');

        } else reject('Invalid Email!');

    })

}

loginBtn.addEventListener('click', authenticateUser);

1. **SLICE:** return a sub portion / substring of an array/string based on start and end index.
2. **JSON.PARSE:** converts a JSON string into a JSON object.
3. **TYPEOF:** is used to validate the datatype of an expression or variable.
4. **DESTRUCTURING:** is used to unpack values from an ARRAY or an OBJECT
5. **DEFAULT ARGUMENTS:** are used when we want to specify default values to its arguments in a scenario when the function call does not have the same parameter.
6. **EXPORT:** is used when we want to use an object in one module from another. We export the desired object.
7. **REQUIRE / IMPORT:** is used to utilize the exported object in the desired module.
8. **STATIC METHOD:** are referenced by the class itself. We do not need to create an object of the class to call this function.
9. **METHOD OVERRIDING:** when a subclass has defined a function which already exists in the parent class with same name, param and return types, the subclass method overrides the parent class method.
10. **INHERITENCE:** is a way of acquiring properties of a class by extending it.
11. **CLOSURE:** allows a subfunction to have access to parent function scope even after parent function has been executed.
12. **REST:** allows functions to dynamically accept arguments. In this case, unique parameters are not specified.
13. **SPREAD:** allows user to expand arrays. It can be used to concatenate values of an array into another by unpacking them.
14. **CALL, BIND and APPLY:**

***Call:*** is used to invoke a function call. Arguments are passed separately.

***Apply:*** is also used to invoke a function call but unlike Call, the arguments are passed as array.

***Bind:*** take an object as an argument and creates a new function.

## Code Snippet for above topics:

### to\_do.js

'use strict'

const input = document.getElementById('todo-title');

const addButton = document.getElementById('new-todo');

const taskList = document.querySelector('#tasks');

function inputLength() {

    return input.value.length

}

var userId;

var sessionData;

// Use of SLICE to get first letter of name

const getFirstLetter = (n) => {

    return n.slice(0, 1);

};

function profileName() {

    sessionData = sessionStorage.getItem('user-data');

    // using JSON.Parse to convert JSON string into JSON object

    sessionData = JSON.parse(sessionData);

    let [firstName, lastName] = sessionData.name.split(' ');

    let firstNameInit = getFirstLetter(firstName);

    let secondNameInit = getFirstLetter(lastName);

    document.getElementById('initialsText').innerHTML = firstNameInit + secondNameInit;

}

// Object

let fullName = {

    firstName: 'Siddharth',

    lastName: 'Sircar'

}

const greetUser = () => {

    // Using TYPEOF to check datatype of sessionData.name

    if (typeof sessionData.name !== 'undefined' && sessionData.name !== null) {

        // Destructuring ARRAY

        let [fName, lName] = sessionData.name.split(' ');

        // Using OBJECT.ASSIGN

        Object.assign(fullName, { firstName: fName, lastName: lName });

    }

    // Destructuring OBJECT

    const { firstName, lastName } = fullName;

    alert(`Hi ${firstName} ${lastName}`);

}

let tasksTitle = []

function displayUserTasks() {

    userId = sessionData.id;

    axios.get(`https://jsonplaceholder.typicode.com/users/${userId}/todos`).then(response => {

        for (const task of response.data) {

            if (task.completed === false) {

                tasksTitle.push(task.title);

                addListItem(task.title);

            }

        }

        // Using CALLBACK: passing a function as an argument to another function

        taskCount(tasksTitle, displayCount);

    }).catch(error => {

        console.log(error);

        alert('Could not connect to database. ', error);

    });

}

// Callback

const taskCount = (tasks, myCallback) => {

    let count = tasks.length;

    myCallback(count);

};

function displayCount(count) {

    let countEl = document.getElementById('task-count');

    countEl.innerHTML = `No. of tasks: ${count}`;

}

// Using DEFAULT ARGUMENTS in function in case function call doesnot send any params

function addListItem(title = null) {

    let li = document.createElement('li');

    let delButton = document.createElement('button')

    delButton.appendChild(document.createTextNode('Delete'));

    delButton.setAttribute('class', 'delete');

    if (title === null) {

        title = input.value;

        // Using LOCALSTORAGE (storage does not expire)

        localStorage.setItem('taks', title);

        tasksTitle.push(title);

        taskCount(tasksTitle, displayCount);

    }

    li.appendChild(document.createTextNode(title));

    li.appendChild(delButton)

    taskList.appendChild(li)

    input.value = '';

    delButton.addEventListener('click', function () {

        delButton.parentNode.parentNode.removeChild(li);

    });

    li.addEventListener('click', function () {

        li.classList.toggle('done');

    });

}

function addTodoOnclick() {

    if (inputLength() > 0) {

        addListItem();

    }

}

function addTodoOnEnter(event) {

    if (inputLength() > 0 && event.keyCode == 13) {

        addListItem();

    }

}

profileName();

displayUserTasks();

addButton.addEventListener('click', addTodoOnclick);

input.addEventListener('keypress', addTodoOnEnter);

### helper.js

// Using EXPORT

export default function getFirstLetter(n) {

    return n.slice(0, 1);

};

export function validateEmail(email) {

    if (/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(email)) {

        return true;

    }

    return false;

}

### profile.js

'use strict'

// Using IMPORT as REQUIRE needed NodeJS integration

import getFirstLetter from './helper.js';

import validateEmail from './helper.js';

const nameTxtbx = document.getElementById('name');

const phoneTxtbx = document.getElementById('phone');

const emailTxtbx = document.getElementById('email');

const cityTxtbx = document.getElementById('city');

const dobTxtbx = document.getElementById('bdate');

const followBtn = document.getElementById('follow');

const locateButton = document.getElementById('locate');

// Use of Class

class user {

    constructor(userData) {

        this.userData = userData;

    }

    getUserDetails() {

        return this.userData;

    }

    // Using Static Method

    static getBirthDate() {

        return '07/29/1998';

    }

}

// Inheritence

class profile extends user {

    constructor(userData) {

        super(userData);

    }

    displayProfile() {

        let userData = this.getUserDetails();

        nameTxtbx.value = userData.name;

        emailTxtbx.value = userData.email;

        phoneTxtbx.value = userData.phone;

        dobTxtbx.value = user.getBirthDate();

        cityTxtbx.value = userData.address.city;

    }

}

let sessionData;

function displayProfileName() {

    sessionData = sessionStorage.getItem('user-data');

    // using JSON.Parse to convert JSON string into JSON object

    sessionData = JSON.parse(sessionData);

    let [firstName, lastName] = sessionData.name.split(' ');

    let firstNameInit = getFirstLetter(firstName);

    let secondNameInit = getFirstLetter(lastName);

    document.getElementById('initialsText').innerHTML = firstNameInit + secondNameInit;

}

// Using Geolocation

const getMyLocation = () => {

    if (navigator.geolocation) {

        navigator.geolocation.getCurrentPosition((position) => {

            let latitude = position.coords.latitude;

            latitude = latitude.toFixed(5);

            let longitude = position.coords.longitude;

            longitude = longitude.toFixed(5);

            document.getElementById('location').value = latitude + ',' + longitude;

        });

    }

    else {

        document.getElementById('location').value = 'No permission to fetch location';

    }

}

// Using CLOSURES

const reqCounter = (function () {

    let requests = 0;

    return function () {

        requests += 1;

        localStorage.setItem('follow-requests', requests);

        return requests;

    }

})()

function displayRequestCount() {

    document.getElementById('reqcount').innerHTML = `Follow Requests Sent: ${reqCounter()}`;

}

// Using REST operator

const requestInfo = (...rest) => {

    document.getElementById('followText').innerHTML = `You sent request to: ${rest}`;

}

function followUser() {

    const folName = document.getElementById('fName');

    const folEmail = document.getElementById('fEmail');

    console.log(folEmail.value);

    // Using CALL and APPLY

    if (folName.value === "" && folEmail.value !== "") {

        // Using CALL

        if (validateEmail(folEmail.value)) {

            requestInfo.call("", folEmail.value);

            folEmail.value = "";

            displayRequestCount();

        } else alert('Invalid Email!');

    } else if (folName.value !== "" && folEmail.value === "") {

        requestInfo.call("", folName.value);

        folName.value = "";

        displayRequestCount();

    } else if (folName.value !== "" && folEmail.value !== "") {

        // Using APPLY

        if (validateEmail(folEmail.value)) {

            requestInfo.apply("", [folName.value, folEmail.value]);

            folName.value = "";

            folEmail.value = "";

            displayRequestCount();

        } else alert('Invalid Email!');

    } else alert('Enter either follower Name or Email!');

}

displayProfileName();

let userProfile = new profile(sessionData);

userProfile.displayProfile();

locateButton.addEventListener('click', getMyLocation);

followBtn.addEventListener('click', followUser);

# HTML5: Introduction to Topic

1. **LOCALSTORAGE:** stores data with no expiration date. Data is not deleted once the session ends or browser is closed. [used in [**to\_do.js**](#_to_do.js)]
2. **SESSIONSTORAGE:** the data is stored only for the duration of the active session, once the session ends the data is deleted. **[**used in[**login\_register.js**](#_Login_register.js)**]**
3. **GEOLOCATION:** api is used to fetch the longitude and latitude. [not working after being deployed to AWS hence screenshot attached below][used in [**profile.js**](#_profile.js)]
4. **EVENTS:** we can execute functions when certain events are triggered like onClick, onSubmit, onload etc.
5. **VALIDATIONS:** used on inputs to validate entered data.
   1. **Pattern:** data validated using regex patterns. Eg. For validating emails.
   2. **Autofocus:** puts cursor focus on the specified element when HTML loads.
   3. **Required:** marks an input field as required. Throws error if field left empty.
   4. **Email:** allows user to only enter email.

## Screenshots:

A picture containing text, monitor, indoor, computer

Description automatically generated

A picture containing text, monitor, indoor, computer

Description automatically generated

A screenshot of a computer

Description automatically generated

*Click ‘LOCATE ME!’ button to fetch lat and long using geolocation*

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated



## Code Snippet for above topics:

### login.html

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>LOGIN</title>

    <link rel="stylesheet" type="text/css" href="../styles/style.css" />

    <link rel="stylesheet" type="text/css" href="../styles/login.css" />

  </head>

  <body>

    <nav class="navigation-bar">

      <a href="../index.html"><h1 class="company-logo">Sort Your Life</h1></a>

    </nav>

    <div class="form-container">

      <header>

        <h2><strong>LOGIN</strong></h2>

      </header>

      <div class="form" >

        <!-- Used AUTOFOCUS -->

        <input

          type="email"

          name="email"

          id="email"

          placeholder="Email"

          required

          autofocus

        />

        <input

          type="password"

          name="password"

          id="password"

          placeholder="Password"

          required

        />

        <div class="buttons">

          <input type="submit" value="LOGIN" id="login" />

          <input

            type="button"

            value="SIGNUP"

            id="register"

            onclick="location.href='./register.html'"

          />

        </div>

        <p>If you don't have an account please click SIGNUP</p>

      </div>

    </div>

    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

    <script type="text/javascript" src="../scripts/login\_register.js"></script>

  </body>

</html>

### register.html

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>REGISTER</title>

    <link rel="stylesheet" type="text/css" href="../styles/style.css" />

    <link rel="stylesheet" type="text/css" href="../styles/login.css" />

  </head>

  <body>

    <nav class="navigation-bar">

      <a href="../index.html"><h1 class="company-logo">Sort Your Life</h1></a>

    </nav>

    <div class="form-container">

      <header>

        <h2><strong>SIGNUP</strong></h2>

      </header>

      <!-- Using REQUIRED, AUTOFOCUS, EMAIL and PASSWORD type fields for validation -->

      <form class="form" action="../modules/todo.html" onsubmit="registerUser();">

        <input

          type="text"

          value=""

          name="name"

          id="name"

          placeholder="Full Name"

          required

          autofocus

        />

        <input

          type="email"

          name="email"

          id="email"

          placeholder="Email"

          required

        />

        <input

          type="password"

          name="password"

          id="password"

          placeholder="Password"

          required

        />

        <div class="buttons">

          <input type="submit" value="SIGNUP" id="register" />

          <input

            type="button"

            value="LOGIN"

            id="login"

          />

        </div>

        <p>If you already have an account please click LOGIN</p>

      </form>

    </div>

    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

    <script type="text/javascript" src="../scripts/login\_register.js"></script>

  </body>

</html>

### to\_do.html

<!DOCTYPE html>

<html>

  <head>

    <title>HOME</title>

    <link rel="stylesheet" href="../styles/todo.css" />

  </head>

  <!-- Added HTML Events -->

  <body onload="greetUser();">

    <nav class="navigation-bar">

      <a class = "logo" href="./todo.html"><h1 class="company-logo">Sort Your Life</h1></a>

      <ul class="nav-container">

        <li class="push-right">

          <div class="initials">

            <span id="initialsText"></span>

          </div>

          <a class="profile" href="./profile.html">My Profile</a>

        </li>

        <a id="logout" href="./login.html">Logout</a>

      </ul>

    </nav>

    <div class="container">

      <h1>My Tasks</h1>

      <p id="task-count"></p>

      <input

        type="text"

        name="to-do"

        id="todo-title"

        placeholder="To-Do Title"

      />

      <button id="new-todo">Add To-Do</button>

      <ul id = "tasks"></ul>

    </div>

    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

    <script type="text/javascript" src="../scripts/to\_do.js"></script>

  </body>

</html>

# DEPLOYMENT

#### Problem Statement: Deploy the developed application, demonstrated in JavaScript and HTML topics, to AWS ECS.

## Introduction To Topic:

Docker enables the separation of our application from the infra using containers. It uses OS level virtualization where the docker works like a virtual OS thus making deployments easier.

## Docker Screenshot:

Graphical user interface

Description automatically generated

## Code Snippet:

### NodeJs File

// Using Require

var express = require('express');

var app = express();

app.set('views', './views');

app.set('view engine', 'ejs');

app.engine('html', require('ejs').renderFile);

app.use(express.static(\_\_dirname + '/public'));

app.get('/', (req, res) => {

    res.render('index.html');

});

app.get('/login', (req, res) => {

    res.render('login.html');

});

app.get('/register', (req, res) => {

    res.render('register.html');

});

app.get('/profile', (req, res) => {

    res.render('profile.html');

});

app.get('/todo', (req, res) => {

    res.render('todo.html');

});

var server = app.listen(3000, function () {

    console.log("Server listening on port 3000");

});

### Docker File

FROM node:14.15.5

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY . .

RUN npm install

EXPOSE 3000

CMD [ "node", "index" ]