Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI0126 – Ingeniería de Software
Profesora Rebeca Obando

Laboratorio 07

Estudiante:
Emmanuel D. Solís - [redacted]

6 de junio de 2022

1. Entendiendo JavaScript

Captura de pantalla de curso completado, Figura 1.

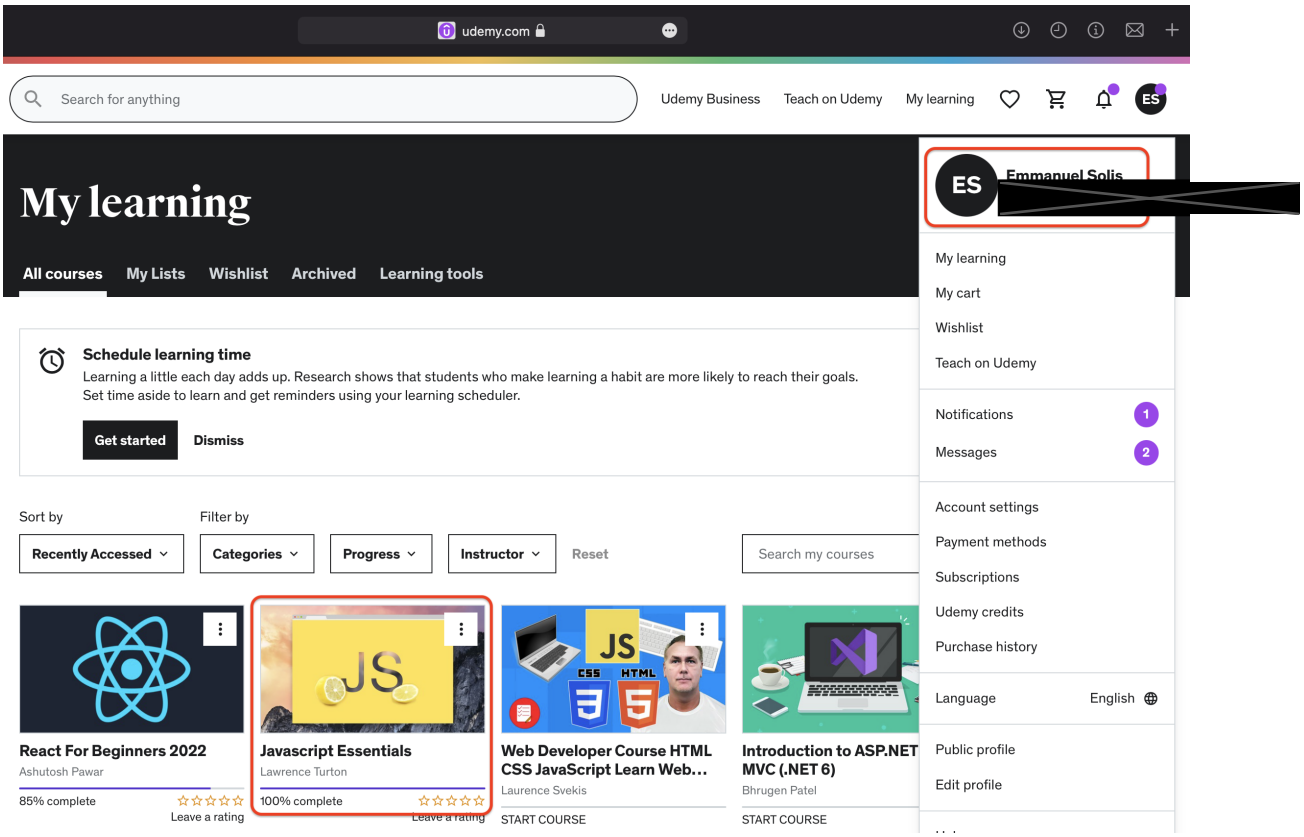

Figura 1: Curso de javaScript en Udemy completado.

2. Respuestas a las preguntas

1. **¿Cual es la diferencia entre *undefined* y *null*?**
   Dado que en JavaScript todo es un objeto y la idea primordial del lenguaje es silenciar muchos errores para que las cosas funcionen la mayor parte del tiempo lo que hace undefined es definir un tipo de dato especial que se hace cuando JavaScript no sabe o no se le indica como clasificar cierto elemento; mientras que null es meramente lo que ya conocemos de otros lenguajes de programación como algo nulo que no existe o algo que apunta a nada.

2. **¿Qué es el *DOM*?**
   Su traducción literal es *Modelo del Objeto Documento*, para tener una idea más clara podemos considerar que el DOM es una estructura de datos que contiene todos los elementos de nuestro HTML, como un arbol de nodos que se va formando segun el navegador va procesando nuestro codigo HTML y CSS si fuera el caso. Este modelo del objeto documento (arbol de nodos) es lo que nos permite luego con nuestro JavaScript acceder a ciertas partes de este este y poder modificarlo; ya sea agregando o eliminando elementos, o modificar sus atributos.

3. **Usando Javascript se puede acceder a diferentes elementos del DOM. ¿Qué hacen, que retorna y para qué funcionan las funciones *getElement* y *querySelector*? Cree un ejemplo.**

   - **getElement:** me retorna una referencia al objeto dentro del DOM del elemento que yo le indique, puedo acceder a él por medio ya sea del *tag, id, class, etc*; podemos ver un ejemplo usando el HTML de este mismo laboratorio donde necesitaba acceder al nodo identificado por el id *lista*:

     ```
     var element = document.getElementById('lista');
     ```

   - **querySelector:** por otro lado el usar el *querySelector* me permite acceder a los nodos identificados por los *selectors* de CSS; por ejemplo:

     ```
     var element = document.querySelector('#pClass');
     ```

4. **Investigue cómo se pueden crear nuevos elementos en el *DOM* usando Javascript. De un ejemplo.**
   Para esto ya el DOM incorpora ciertas funciones, que apoyandose en el uso de las anteriores de acceso una vez que tenemos la referencia a donde queremos colocar ciertas cosas podemos entonces usarlas para añadirlas, por ejemplo si queremos agregar un nuevo elemento `<li></li>` se puede hacer y de igual forma varios; entre las funciones más importantes son `[elementRef].appendChild([parameters])` y `[element].insertBefore([parameters])`; un ejemplo de esto sería el siguiente:

   ```
   function add_element() {
     var element = document.getElementById("lista");
     const new_element = document.createElement("li");
     new_element.innerHTML = "Elemento " + (current_size + 1);
     element.appendChild(new_element);
   }
   ```

5. **¿Qué es un *Promise* en *JavaScript*? De un ejemplo.**
   Dado que JavaScript es un lenguaje unihilo asincrónico hay ciertas operaciones por las que él espera y otras por las que no, por ejemplo cuando hacemos un llamado a un servidor con `fetch(...)` el interpretador en ciertas ocasiones no espera a la respuesta sino que sigue ejecutando la siguiente linea mientras delega la responsabilidad de ese llamado a un servidor al sistema operativo; el *Promise* lo que nos permite decirle al interprete es que siga ejecutando y que esa otra tarea le *"prometió"* que le daría el resultado y entonces con ese el interprete puede manejar dos posibles casos; que hacer si la cumple y que hacer si no la cumple y de esa forma poder manejar dicho resultado asincrónico. Podemos ver un diagrama de cómo esto funciona en la Figura 2, y podemos ver el siguiente ejemplo de una función *Promise*:

   ```
   let myPromise = new Promise(function(myResolve, myReject) {
     let req = new XMLHttpRequest();
     req.open('GET', "mycar.htm");
     req.onload = function() {
       if (req.status == 200) {
         myResolve(req.response);
       } else {
         myReject("File not Found");
       }
     };
     req.send();
   });

   myPromise.then(
     function(value) {myDisplayer(value);},
   ```

```
  function(error) {myDisplayer(error);}
);
```

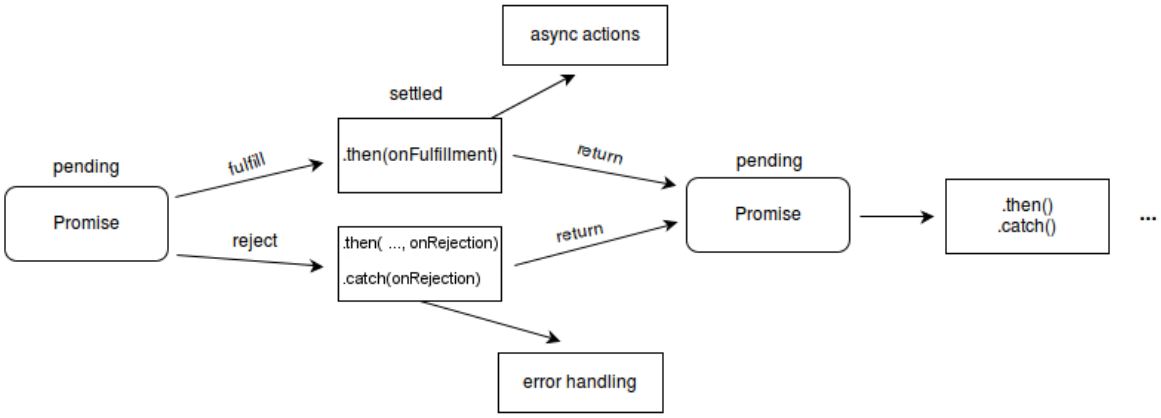

Figura 2: Funcionamiento de los *Promises* de JavaScript.

6. **¿Qué es *Fetch* en *JavaScript*? De un ejemplo.**
   Esta es una función que me permite hacer llamados y solicitudes de red, por ejemplo el hacer una solicitud HTTP a un API server; por ejemplo:

```
function fetch_api() {
  fetch('https://jsonplaceholder.typicode.com/users')
    .then(response => response.json())
    .then(data => console.log(data));
}
```

7. **¿Qué es *Async / Await* en *JavaScript*? De un ejemplo.**
   Como se mencionó anteriormente como JavaScript es un lenguaje asincrónico el uso de estos *Async / Await* nos permite decirle al compilador que cierta función se debe ejecutar de cierta forma y luego esperarla; entonces el *async* nos permite indicarle que una función será ejecutada de forma asincrónica y más adelante en el punto donde necesitemos usar los datos de esa operación podemos decirle al interprete que debe esperar por dichos resultados con *await*; un ejemplo de esto es el siguiente:

```
getUsers = async () => {
  let res = await axios.get("https://reqres.in/api/users?page=1");
  let { data } = res.data;
  this.setState({ users: data });
};
```

8. **¿Qué es un *Callback*? De un ejemplo.**
   Los *callback* son funciones asociadas con un evento específico, al menos en lo que respecta a JavaScript en lado del cliente, estas son invocadas cuando cierto evento se da, lo cuál se pudo saber gracias a su listener; un ejemplo de un *callback* es el siguiente:

```
var select = document.getElementsByName('cars')[0];

select.onclick = function(event) {
  console.log(event);
}
```

```
function click_callback(event) {
  console.log('clicked by add event listener');
}

select.addEventListener('click', click_callback);
select.removeEventListener('click', click_callback);
```

3. Ejercicio practico

Solución se encuentra en los archivos *index.html y script.js*.

4. Referencias

- JavaScript Promises. (2022). W3 Schools. https://www.w3schools.com/Js/js_promise.asp.
- Promise - JavaScript | MDN. (2022, May 27). MDN Mozilla Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.
- Hidalgo, J. (2020). Desarrollo de aplicaciones web (1st ed.). http://jeisson.ecci.ucr.ac.cr/appweb/material/.