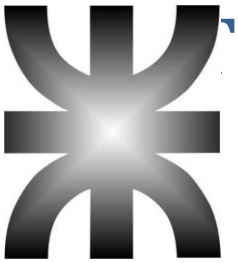

TRABAJO PRÁCTICO INTEGRADOR

ETAPA #3 - JS

TUP - Laboratorio de Computación II

Etapa #3 - JavaScript

Continuando con el desarrollo de nuestra aplicación del Clima. En esta etapa del TP Integrador, tenemos como objetivo realizar las implementaciones de lógica JavaScript para lograr que la aplicación sea funcional. En resumen, vamos a hacer que la App almacene las ciudades cargadas por el usuario y consultar el clima de las mismas, exponiendo dicha información al usuario final.

Para esta entrega, se presentan requerimientos EXIGIDOS los cuales se deberán implementar sí o sí en su programa para poder aprobar el TP. Al mismo tiempo, tendrán un listado de requerimientos EXTRAS los cuales son opcionales.

Consideraciones importantes:

1. Antes de avanzar con esta etapa, tener la entrega de la Etapa #2 corregida por su profesor y haber realizado los ajustes que se le han indicado vía email.
2. Realizar los cambios en el código HTML que requiera o considere necesarios.
3. Enlazar los archivos **.js** correspondientes a cada página. Por ejemplo, en el documento add-city.html agregar el common.js y el add-city.js. No te olvides de colocarle el “defer”-
4. Implementar lógica JS a cada página para lograr los requerimientos exigidos en las siguientes secciones. En caso de detectar que una funcionalidad se repite en diferentes páginas colocarla dentro del common.js, sino trabajar en el js correspondiente a cada página.
5. Finalizar todos los requerimientos EXIGIDOS y una vez cerrados avanzar, en caso de que cuente con el tiempo, con los requerimientos EXTRAS.

Recomendación: Subir los cambios al repo a medida que se va avanzando con el desarrollo.

Conceptos JavaScript a poner en práctica:

- Manipulación del DOM
- Manejo de objetos y arreglos
- Local Storage
- Fetch API

Requerimientos Exigidos

Se propone avanzar el desarrollo de JS primero con la pantalla de Agregar Ciudad y luego con Consultar Clima. De esta manera se presentan los apartados de esta sección, de todas maneras está en el desarrollador comenzar la implementación por donde crea más conveniente.

Agregar ciudad (add-city.html)

Esta página servirá para agregar una ciudad a la lista de ciudades. Debe consistir en un formulario con un único **campo de texto** que permita ingresar el **nombre de una ciudad** y un **botón** que al ser clickeado **guarde la nueva ciudad** en el listado de ciudades del localStorage. Si se intenta agregar una ciudad que ya está guardada, se debe mostrar un cartel de error. En el caso de que la ciudad no se encuentre guardada, agregarla al listado e informar el guardado con un mensaje de éxito.

Código HTML inicial

Al iniciar, la pantalla no debe mostrar ningún mensaje ya que los mismos deberán mostrarse de acuerdo al escenario generado por el usuario cuando el mismo ingrese una nueva ciudad.

Guardar Ciudades

Utilizar la funcionalidad **localStorage** la cual nos permite almacenar información de forma permanente, en este caso particular, las ciudades. Se debe tener en cuenta que en localStorage sólo podemos almacenar strings, por lo que se debe realizar un parseo para poder almacenar el arreglo de ciudades en formato JSON.

Tener en consideración las siguiente funciones JS para obtener las ciudades del localStorage y para agregar una nueva ciudad:

```
function getCitiesFromLocalStorage() {
   let cities = localStorage.getItem("CITIES");
   if(cities) {
       cities = JSON.parse(cities);
   } else {
       cities = [];
   }
   return cities;
}

function addNewCityToLocalStorage(newCity) {
   let cities = getCitiesFromLocalStorage();
   cities.push(newCity);
   localStorage.setItem("CITIES", JSON.stringify(cities));
}
```

Validaciones

Antes de guardar la ciudad debe validarse que la misma no esté incluida en el listado de ciudades.

Consultar clima (index.html)

Esta página le va a brindar al usuario la opción de seleccionar una ciudad dentro del listado de ciudades y ver el clima actual de la misma. La lista de ciudades debe obtenerse del localStorage (ciudades anteriormente cargadas en la página de Agregar Ciudad).

En este punto, tenemos desarrollado sólo la maqueta del escenario principal y ahora le daremos vida o comportamiento a través de JS. Por este motivo, se deberán hacer ajustes HTML para que la pantalla, cuando ingresa el usuario se encuentre vacía y la CARD con información del clima se deberá mostrar recién una vez el usuario seleccione una ciudad del SELECT y presione CONSULTAR

Consideraciones técnicas:

Select de Ciudades

En el código HTML, el select de ciudades no deberá contener ninguna ciudad precargada ya que las mismas se cargarán a través de JS con la lógica que usted implemente para tal fin.

Entonces, cuando inicie su aplicación, deberá obtener la información del localStorage y cargar las ciudades que allí se encuentren almacenadas. En caso de que no haya ciudades guardadas sería ideal que se le muestre al usuario un mensaje informando tal situación.

Información del Clima (Card)

Esta card, al inicio deberá estar oculta y lógicamente, mostrarse cuando se obtiene la información de la API con la ciudad seleccionada por el usuario.

localStorage (JS)

Cuando se muestre la página index.html y se cargue el script index.js, deberá obtener las ciudades del localStorage y general el código HTML de las options de cada ciudad para ponerlo en el select de Ciudades.

Consultar el CLIMA a través de una API

El clima será consultado a través de solicitudes a una API externa utilizando la herramienta **fetch**.

La API donde deben consultar esta información es **OpenWeatherMap** y se solicita mediante la siguiente URL:

`api.openweathermap.org/data/2.5/weather?q=`**`{nombre ciudad}`**`&appid=`**`{clave API}`**`&units=metric&lang=es`

Importante: obtener la clave de la API logueandose en la plataforma (la misma es gratuita).

Entonces si quisiéramos consultar el clima actual en la ciudad de Rosario deberíamos usar fetch y pasarle como parámetro la siguiente url:

`api.openweathermap.org/data/2.5/weather?q=`**`Rosario`**`&appid=`**`3936d0749fdc3124c6566ed26XXXXX`**`&units=metric&lang=es`

La respuesta que obtendremos vendrá en formato de objeto JSON. Este es un ejemplo de la misma:


```
{
    "coord": {
        "lon": -60.6393,
        "lat": -32.9468
    },
    "weather": [
        {
            "id": 800,
            "main": "Clear",
            "description": "cielo claro",
            "icon": "01d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 28.23,
        "feels_like": 27.02,
        "temp_min": 27.12,
        "temp_max": 29.97,
        "pressure": 1011,
        "humidity": 24
    },
    "visibility": 10000,
    "wind": {
        "speed": 0.89,
        "deg": 21,
        "gust": 3.13
    },
```

```
    "clouds": {
        "all": 0
    },
    "dt": 1637269040,
    "sys": {
        "type": 2,
        "id": 2000719,
        "country": "AR",
        "sunrise": 1637225467,
        "sunset": 1637275485
    },
    "timezone": -10800,
    "id": 3838583,
    "name": "Rosario",
    "cod": 200
}
```


El contenido de esta respuesta puede ser manipulado como un **objeto** en JavaScript. Para poder obtener las condiciones climáticas actuales debemos inspeccionar las propiedades de ese objeto. Las únicas propiedades sobre las cuales nos vamos a centrar van a ser sobre *weather* y *main*.

El objetivo será elaborar una página que tenga un **desplegable de selección**, usando como **opciones la lista de ciudades** y un **bloque informativo** sobre el clima en la **ciudad seleccionada**.

Considerar que la aplicación debe solicitarle a la interfaz externa de **OpenWeatherMap** las condiciones climáticas actuales de la ciudad seleccionada **cada vez que se elija una ciudad**. Es decir, si entro en la página y seleccione la ciudad de Rosario, la aplicación debería consultar el clima actual en la ciudad y mostrarlo en la misma página. Del mismo modo, si luego de haber consultado el clima en esa ciudad quisiera ver los datos climáticos de otra, podría cambiar la opción seleccionada en el desplegable y obtener la información del tiempo de esa nueva ubicación.

Claves para este enunciado

- Utilizar async/await o promesas para las llamadas realizadas con fetch.
- Para obtener una descripción del clima actual, ver los datos dentro de la propiedad *weather* en el objeto de la respuesta de la solicitud a la API.
- Para obtener los datos de temperatura, ver las propiedades dentro de *main* en el objeto de la respuesta de la solicitud a la API.

- Armar una función que realice el llamado a la API e invocar cada vez que cambie el valor del desplegable.

Subir aplicación a Github Pages

Seguir instrucciones dadas en la página oficial de **Github Pages** (https://pages.github.com/) para la correcta subida y despliegue de la aplicación en la plataforma.

Requerimientos Extras

Estos requerimientos a continuación son opcionales y servirán para sumar nota a la calificación final y por ende facilitará alcanzar la condición de aprobación directa.

Contemplar escenario de carga

Esta funcionalidad permitirá darle al usuario un mensaje para informarle cuando una operación está siendo realizada. Puede hacer uso de iconos o spinners en el HTML.

El usuario debería ver este aviso de cargando en la pantalla solo de modo informativo cuando realiza las siguientes operaciones:

- Cargar una ciudad a la lista de ciudades
- Consultar clima

Si considera que hay operaciones que se realizan de forma muy rápida, que no se alcanza a visualizar el icono de carga, puede utilizar setTimeout para dar una mejor experiencia al usuario.

Validar la existencia de una ciudad previo a incluirla en la lista

Previo al agregado de la ciudad al listado de ciudades sería elemental validar que la misma sea una ciudad existente y no se cargue una que no se pueda consultar sobre su clima.

Esto puede lograrse haciendo una llamada extra a la API del clima e inspeccionar en su respuesta si retorna datos de una locación o si la respuesta es un error. Puede probar ambos casos con datos de prueba y evaluar qué respuesta recibe del servicio externo.

En caso de obtener un error, notificar con un mensaje de error (los mismos que se usaron hasta ahora en las entregas anteriores) para avisar al usuario del correcto ingreso de la ciudad.

En caso de que la ciudad sea una existente, agregarla al listado de ciudades para poder consultar el clima en la otra página.

Contemplar escenarios de error retornados por las API

En las solicitudes a las API puede que haya un error. Esta funcionalidad servirá para manejar esos errores de forma apropiada e informar al usuario de los mismos para que realice algún cambio sobre la información enviada.

En caso de obtener un error, notificar con un mensaje de error (los mismos que se usaron hasta ahora en las entregas anteriores) para avisar al usuario del correcto ingreso de la ciudad.

Se recomienda el uso de try catch para proteger diferentes secciones de la aplicación que sean susceptibles a errores e informar los mismos en caso de que ocurran.

Integración con API de envío de emails

Integrar la aplicación actual al servicio de mensajería por email usando la librería EmailJS. Esta librería JS permite el envío de emails únicamente utilizando su API JavaScript que realiza el la operación internamente.

Esta implementación deberá realizarse sobre la página **help.html**. Pueden consultar aquí la documentación de la librería para obtener instrucciones precisas de cómo integrarla correctamente: https://www.emailjs.com/docs/tutorial/overview/.

Condiciones de entrega

La fecha límite para la entrega es el **xx/xx/xxxx**, la cual se deberá realizar a través de email al profesor de la materia.

En el asunto del correo se deberá colocar: **TP INTEGRADOR CLIMA - [NOMBRE Y APELLIDO]**

En el cuerpo del mensaje, se debe indicar:

- URL del repositorio.
- URL de la aplicación corriendo en Github Pages.
- Consideraciones: cualquier consideración relacionada al desarrollo, resolución y/o entrega del problema.

El trabajo podrá ser defendido en una fecha acordada con los profesores luego de haberlo entregado y haber obtenido una corrección.