Programación Web Full Stack

JS101

Práctica clase en vivo

Objetivo

Finalizar la app de tareas utilizando las herramientas que aprendimos en todo el módulo de **JS101** partiendo de la ejercitación de la clase 2 finalizada**.**

** En caso de no haber finalizado la actividad anterior, se dará resuelta para que todos partamos del mismo punto.*

Micro desafíos

El tech leader de nuestro equipo de desarrollo propone refactorizar la app de tareas. Para esto nos encarga:

1. Modificar la funcionalidad de *listar* tareas. Deberemos utilizar el método **forEach**. Recordemos que **forEach** puede recibir dos parámetros, siendo el segundo nuestro index (es posible que tengamos que usarlo).
   Más información en: link.

2. Brindar la posibilidad al usuario de guardar nuevas tareas. Para ello tendremos que armar varias cosas:
   Por un lado, tendremos que **crear funciones** que nos permitan **escribir** información dentro del archivo .json que tiene nuestras tareas y **guardar** ese archivo actualizado.
   En segundo lugar, tendremos que crear una alternativa dentro de nuestro "switch" (**un nuevo case**) para identificar que el usuario quiere crear una tarea y ejecutar las funciones de "escribir json" y "guardar el archivo actualizado".
   Vamos por el primer paso: creemos las funciones.
   a. Dentro de nuestro archivo **tareas.js**, vamos a crear una función llamada **escribirJSON** que recibirá un array de tareas como parámetro. La función se encargará de:
      i. Convertir el array recibido como parámetro a un string en formato JSON.
      ii. Guardar la información en el archivo .json que contiene la lista de nuestras tareas. Para esto necesitaremos el método *writeFileSync* del módulo **FS**.
   b. Nuevamente, dentro del archivo **tareas.js**, vamos a crear otra función llamada **guardarTarea**. La función recibirá una *objeto* tarea y la guardará en el archivo **.json** junto con todas las tareas que ya estén allí. La función deberá:

i. Obtener toda la información del archivo .json en donde tenemos nuestras tareas. Recordemos que esta acción la hace la función **leerJSON**.
ii. Debemos agregar la tarea nueva al array que obtuvimos en el punto anterior.
iii. Guardar el array actualizado en el archivo .json. Recordemos que ya tenemos la función **escribirJSON** que ya sabe hacer este punto.
Vamos por el segundo paso: crear un nuevo *case* que identifique "crear" una nueva tarea y ejecute las funciones.

c. Vamos a trabajar en cómo ingresar la nueva tarea, cómo la obtendremos y cómo la guardaremos en nuestro archivo .json con todas las tareas.
i. El título de la nueva tarea lo ingresamos desde la terminal usando argumentos en la línea de comando. Vamos a necesitar una nueva acción "crear" y luego el título de la tarea. De esta forma: *node app.js **crear "Una nueva tarea"***.
ii. En el archivo **app.js** necesitaremos un nuevo "case" que pueda procesar la opción **crear** ingresada como argumento.
iii. Dentro del *case* "**crear**", crearemos una variable de tipo objeto literal con dos atributos: título y estado. El título vendrá del argumento ingresado por consola y el estado de la tarea será siempre "**pendiente**".
Ahora debemos guardar esta tarea junto con las otras que ya están en el archivo **.json**. Ya tenemos una función que sabe hacerlo. ¿Cuál es?

3. Vamos ahora a **filtrar** las tareas por estado. Crearemos una función llamada **filtrarPorEstado** en nuestro módulo de tareas (archivo **tareas.js**) que nos permita filtrar tareas por estado y luego mostrarlas en consola. Para esto:
a. Necesitaremos crear en **tareas.js** un nuevo método "**leerPorEstado**" que reciba un estado como parámetro.
b. La función deberá, en primer lugar, obtener todas las tareas de nuestro archivo **.json**. Tenemos una función que ya sabe hacerlo. ¿Cuál es?
c. Ahora vamos a necesitar obtener únicamente las tareas cuyo estado coincida con el parámetro ingresado. Para lograrlo sabemos que los arrays tienen

métodos propios que nos podrían ayudar. ¿Cuál de ellos puede ayudarnos a filtrar un array?

d. Por último, la función deberá retornar el nuevo array con las tareas ya filtradas.
e. Vamos ahora a nuestro archivo **app.js**. Allí tendremos que crear un nuevo "**case**" que se encargue de identificar el estado ingresado por el usuario, ejecute el método "**filtrarPorEstado**" y muestre las tareas al usuario.
    i. El estado de las tareas que queremos obtener lo ingresamos desde la terminal usando argumentos en la línea de comando. Vamos a necesitar una nueva acción "filtrar" y luego el estado de la tarea. De esta forma: *node app.js **filtrar pendiente.***
    ii. Dentro del case "**filtrar**" guardaremos en una variable el estado ingresado desde la terminal.
    iii. El estado que acabamos de guardar será el parámetro con el que podemos ejecutar la función **leerPorEstado**. Recordemos que esta función retorna un array. Vamos a necesitar guardar la ejecución en una variable.
    iv. Por último, nos queda mostrar al usuario en la consola la lista de tareas que coinciden con el estado ingresado. Es decir, debemos recorrer el array obtenido y mostrar cada una de las tareas al usuario. Tenemos un método de arrays que nos puede ayudar. Uno que nos permite ejecutar una acción "para cada elemento de un array". ¿Cuál es?

Si llegamos hasta aquí, el tech leader del equipo debe estar extremadamente alegre con nuestro trabajo y desempeño. ¡Buen trabajo!

Si nos trabamos en alguna parte, a no preocuparnos, es totalmente natural. Son muchos conceptos nuevos y necesitan de un tiempo prudente para asentarse. No olvidemos compartir las partes que no salieron con nuestros profesores y compañeros para despejar dudas.

¡Hasta la próxima!