ELORRIETA ERREKA MARI CIFP LHII	Curso / *Kurtsoa*	Fecha / *Data*	Nivel / *Maila*	Eval. /*Ebal.*
	2024/25	2024-11-19	2DAW	1 EVA
Módulo / Modulua			***Mota /Tipo***	***Calificación / Kalifikazioa***
Desarrollo Web en Entorno Cliente			Procedimientos	
Contenidos (U. Didácticas) / Edukiak (U. Didáktikoak)	UD01 - UD06			
Nombre y apellidos / Izen abizenak				

Debes entregar el examen en formato PDF
en la plataforma Moodle/Classroom correspondiente

Organizador de planes con amigos/as

Se debe desarrollar una aplicación web dedicada a organizar los diferentes planes que pueda el/la usuario/a tener con sus amistades, llevando un control de éstos y de los/as asistentes a ellos.

Completa los ejercicios, organizando el código en módulos ES6 para la comunicación entre scripts y asegurando una estructura clara. La aplicación debe permitir la gestión de asistentes, incluyendo al usuario, y colorear los planes y asistentes en función de su asistencia. Sigue el orden de los ejercicios para construir progresivamente la aplicación.

Ejercicio 1: Almacenamiento con Cookies (0,5 ptos.)

Crea un módulo llamado *userModule.js* que maneje el nombre del usuario:

Este módulo debe verificar si hay un nombre guardado en las cookies del usuario. Si no hay, pide el nombre al usuario y guárdalo en una cookie.

Exporta una función que muestre un saludo en la parte superior de la aplicación con el nombre del usuario cada vez que carga la página (por ejemplo, "Hola, [nombre]! Aquí están tus planes:").

Ejercicio 2: Creación y visualización de planes (2 ptos.)

Crea un módulo llamado *Plan.js* que contenga una clase llamada *Plan* y que cuente como propiedades:

- *id* (que será el número de milisegundos transcurridos desde las 00:00:00 UTC del 1 de enero de 1970 hasta el momento de creación de la instancia)
- *nombre* (que llegará como parámetro)
- *asistencia* (que se preguntará al usuario)
- *asistentes* (que se inicializará como un array vacío).
- También deberá tener un método estático llamado *toggleAsistenciaPlan* que reciba un plan como parámetro y cambie el valor de su propiedad asistencia de true a false o viceversa.

Crea un módulo llamado *plan-manager.js* para gestionar los planes:

- Declara e inicializa un array vacío que albergue todos los futuros planes.
- Define y exporta una función llamada *addPlan*. Esta función se ejecutará al hacer clic en el botón "Añadir Plan" o al presionar Enter dentro del campo de texto.
    - Por defecto, al añadir un nuevo plan, se debe confirmar la asistencia del usuario, lo que se reflejará en el color del fondo del plan (verde para asistencia confirmada y rojo para no confirmada) que programarás más adelante.
    - Añade validaciones para evitar que se añadan planes vacíos o duplicados. Si el plan ya existe o el campo está vacío, muestra un mensaje de error.
- Define y programa una función llamada *toggleAsistenciaUsuario* que reciba como parámetro un *planID*, encuentre el plan dentro del array de planes y llame al método estático de la clase Plan.
- Define y programa una función llamada *removePlan* que reciba como parámetro un *planID* y borre este plan del array de planes.

Ejercicio 3: Creación de AsistenciaAmigo y añadir amigos (2 ptos)

En *AsistenciaAmigo.js*, define y exporta una clase *AsistenciaAmigo*, con las propiedades:

- *nombre* (nombre del amigo).
- *asistencia* (booleano que indica si asistirá al plan, configurado como true por defecto al agregar el amigo).

Dentro del módulo *plan-manager.js*, crea una función para que el/la usuario/a pueda añadir amigos/as a un plan específico, almacenándolos/as en el array *asistentes* dentro del objeto del *Plan*. Esta función recibirá un *planId* y un nombre de amigo/a y la función creará, si no existe un/a asistente con ese nombre en dicho plan, una nueva instancia de *AsistenciaAmigo,* y la añadirá al array de *asistentes* del plan.

Además de ello, también debes crear una función que reciba un *planId* y un *amigoName* y cambie la asistencia de dicho/a asistente.

Ejercicio 4: Eventos y actualización del DOM (2 ptos.)

En *plan-manager.js*, programa y exporta una función llamada *renderPlans* que recoja y resetee el contenido del elemento *planList* y, por cada plan del array de planes, cree en el DOM un nuevo div con un fondo del color dependiendo de la asistencia del usuario, un h3 con el nombre del plan, los botones para gestión del plan y un listado de los/as amigos/as que hayamos añadido. En resumen, cada plan debe:

- Tener un fondo que muestre la asistencia del usuario.
- Mostrar el nombre ingresado por el usuario.
- Tener un botón para cambiar el estado de asistencia del usuario.
- Tener un botón para eliminar el plan.
- Tener un botón para "Añadir asistentes".
- Mostrar una lista de asistentes.
    - Cada asistente debe tener:
        - Su color de fondo dependiendo de su asistencia.
        - Su nombre.
        - Un botón para cambiar su asistencia.

Para que los botones funcionen también deberás programar los siguientes eventos:

- Asistencia del usuario: Al hacer clic en el botón de asistencia del usuario, permite activar o desactivar la asistencia. Cuando está confirmado, el fondo del plan debe ser verde; si no, debe ser rojo.
- Eliminar Plan: Programa el botón de eliminar para eliminar el plan del DOM.
- Añadir asistentes: Cuando se hace clic en este botón, permite agregar amigos/as al plan, mostrando su estado de asistencia y aplicando el color correspondiente (verde o rojo) según su asistencia. El nombre del/ de la amigo/a se deberá recibir a través de un prompt.
- Asistencia amigo/a: Al hacer clic en el botón del/ de la asistente de la lista se deberá cambiar su estado y color de fondo (verde o rojo).

Ejercicio 5: Almacenamiento con localStorage (1 pto.)

Crea un módulo llamado *storageModule.js* para gestionar el almacenamiento de los planes y asistentes en localStorage:

Crea y exporta la función para guardar cada plan de salida junto con su lista de asistentes en localStorage, de modo que se mantengan al recargar la página.

Al cargar la aplicación, verifica si hay planes y asistentes guardados en localStorage y cárgalos en el array *planList*.

Además, cada vez que se añada un nuevo plan, se cambie la asistencia del usuario a algún plan, se añada la asistencia de algún/a amigo/a a algún plan o se cambia la asistencia de éste/a o se borre algún plan, se deberá actualizar el listado de planes en el localStorage.

Ejercicio 6: Funciones avanzadas y manejo de objetos (1 pto.)

Crea una función que permita añadir varios amigos a un plan mediante el parámetro rest, en caso de ser necesario. Modifica tu código para que en caso de que se escriba un solo asistente en el prompt de añadir amigos/as a un plan se ejecute la función creada en el ejercicio 3, pero si se reciben varios/as asistentes separadas por coma se extraiga cada uno de los nombres y se haga una llamada a la función definida en este ejercicio pasando todos/as los/as asistentes como parámetro.

Ejercicio 7: Estadísticas (1 pto.)

En *plan-manager.js*, crea una función que muestre en tres alerts consecutivos:

- El número total de planes.
- El número de planes con asistencia confirmada del usuario.
- El número total de amigos/as y el número de asistentes confirmados para cada plan.

Añade un botón a pié de página para ejecutar esta función.

Ejercicio 8: Archivo Principal (main.js) (0,5 ptos.)

En el archivo *main.js*, importa y usa las funciones de cada módulo para que la aplicación funcione de manera integral.

Organiza el flujo de ejecución para que las funciones de cada módulo trabajen juntas de forma efectiva.

Asegúrate de que el botón "Añadir Plan" y la tecla Enter en el campo de entrada ejecuten la misma función de añadir el plan, incluyendo la confirmación de asistencia del usuario.