Examen JavaScript - Código en la Cocina -

Se desea crear una Web de Recetas de cocina. Para ello se plantean los siguientes ejercicios

Ejercicio 0: Configuración del Proyecto (0.5 puntos)

Requisitos de partida:

- Crea un proyecto utilizando **Vite** y el scaffolding de carpetas habitual.
- Levanta una API en el puerto **3500** a través de un script llamado `examen-javascript` utilizando el fichero `db.json`.
- Crea un fichero **.env** para las variables globales con las variables **PORT y URL**.

1. Obtener, Almacenar y Devolver Recetas con Sistema de Caché (2 puntos)

Crea la función `getRecipesCache()` que:

1. Verifique si en el `localStorage` hay una clave llamada `recetas-cache`.
    - Si existe la clave `recetas-cache` y el valor de `timeStamp` es superior a 5 minutos (convirtiendo el tiempo a milisegundos), realiza una solicitud `fetch` a `http://localhost:3500/recetas`, almacena las recetas en `localStorage` junto con un nuevo

`timeStamp` actualizado porque se considera que la información es antigua pasados 5 minutos.
    - Si han pasado menos de 5 minutos desde la última solicitud almacenada, carga las recetas directamente desde el `localStorage` y devuélvelas sin realizar un nuevo `fetch`.

2. Si no existe la clave `recetas-cache` en el `localStorage`, realiza una solicitud `fetch` a la API, almacena las recetas en el `localStorage`, junto con un `timeStamp` que registre la hora actual, y devuelve las recetas.

3. En resumen, la función `getRecipesCache()` debe devolver siempre las recetas, ya sea desde el `localStorage` o desde la API, dependiendo de la validez del `timeStamp`.

Ejemplo de almacenamiento en `localStorage` de `recetas-cache`

```
{
  "recetas": [ArrayConTodasLasRecetasDeLaAPI],
  "timeStamp": 1730999964486
}
```

> **Nota**: Usa `Date.now()` para obtener el `timeStamp` actual.

2. Filtrar recetas por tiempo de preparación (1 puntos)

Crea la función `filterByTime(recetas, minTime, maxTime)` que:

- Devuelva todas las recetas cuyo tiempo de preparación esté entre los números enteros `minTime` y `maxTime`.
- Lance un error si `minTime` es mayor que `maxTime`.

> **Nota**: Asegúrate de manejar el error de forma clara y de documentar las recetas filtradas correctamente.

3. Obtener detalles de recetas en paralelo, con `Promise.all` (1.5 puntos)

Crea una función `getRecipeDetails(idsRecetas)` que, dado un array de `idsRecetas`, realice una solicitud `fetch` para obtener los detalles de cada receta individualmente usando `http://localhost:3500/recetas/{id}`. Es necesario utilizar `Promise.all` para obtener los detalles de todas las recetas de forma simultánea.

- Devuelve un array de los detalles de Todas las recetas solicitadas.
- En caso de que alguna de las solicitudes falle, maneja el error y devuelve un array vacío.

4. Organizar recetas por dificultad (1 punto)

Implementa `orderRecipesByDifficulty(recetas)`, que:

- Devuelva un `Map` en el que las claves sean las dificultades (`facil`, `intermedio`, `dificil`) y los valores sean arrays con las recetas organizadas y filtradas por cada dificultad.

5. Valorar una receta (1 punto)

Define `scoreRecipe(recetas, recetaId, nuevaValoracion)` que:

- Busque la receta por `recetaId` y **actualice** el campo `valoracion` con la `nuevaValoracion`.
- La nuevaValoración no puede ser inferior a 0 ni superior a 5.
- Devuelve el array de recetas actualizado con la valoración modificada.

> **Nota**: Asegúrate de que `nuevaValoracion` esté dentro de un rango lógico (por ejemplo, de 1 a 5) antes de asignarla.

6. Buscar recetas por ingredientes y añadir ingredientes únicos (2 puntos)

Genera la función `addIngredients(recetas, nombreReceta, ArrayingredientesParaAñadir)` que:

1. Le pasamos las recetas o la función que se trae las recetas.
2. Busca dentro de las recetas aquella que coincida con `nombreReceta`.

3. Añada los ingredientes de `ArrayingredientesParaAñadir` al array de `ingredientes` de la receta encontrada, usando un `SET` para asegurar que no haya ingredientes duplicados.
4. Realiza una solicitud *PUT (o PATCH)* a la API para actualizar la receta modificada y reflejar los nuevos ingredientes.

7. Verificación de las Funciones (1 punto)

Crea una función llamada `init()` en `main.js` que realice un test o comprobación de cada ejercicio realizado con las posibles variaciones que se planteen en cada apartado.

La función init() debe invocar y verificar el funcionamiento de todas las funciones implementadas, mostrando los resultados de las pruebas en la consola.

Es **IMPORTANTE** que cada vez que necesitemos las `recetas` deberemos o Pasar la función `getRecipesCache()` directamente o guardar en una `variable` el resultado de esa función y pasar la data devuelta por la función. (Elección tuya).

Nota:

> - Asegúrate de que todas las funciones se comporten como se espera y documenta todas las funciones utilizando **JSDoc**.
> - Añade test para comprobar que tus funciones controlan todo lo que se pide en el examen.
> - Recuerda añadir tu `nombre` a cada fichero .js que entregues con la sintaxis de **JSDoc**.
> - Recuerda gestionar toda la **Asincronía** controlando los `posibles errores` que puedan ocurrir
> - Este examen cubre todos los resultados de aprendizaje de Javascript sin incluir DOM ni POO.