Aplicaciones Telemáticas, examen práctico, 16/06/2026
Grado en Ingeniería Telemática
Universidad Rey Juan Carlos

Instrucciones

- Emplea lo visto en clase, y solo lo visto en clase (que puedes consultar en las transparencias). No será válido ni JavaScript más antiguo ni más avanzado. Usa el modo estricto. Si algún ejercicio tiene errores de sintaxis y/o no hace nada, su nota será nula.

Ejercicio 1. (4.0 puntos)

Partiendo de la estructura de datos utilizada en la práctica 4, escribe un programa llamado *productos.js* que contenga una función llamada *crear_productos()* que reciba como parámetro una lista con ese formato y devuelva una nueva lista de objetos.

La lista recibida tendrá la misma estructura que *lista_menu*. Esto es: cada sublista representa una categoría. El primer elemento de cada sublista será el nombre de la categoría y los restantes elementos serán identificadores de productos.

Ejemplo:

```
let lista_menu =
[
    [ "Primeros platos", "p001", "p002"],
    [ "Segundos platos", "p101", "p102"],
    [ "Postres", "p201", "p202", "p203"]
];
```

Además, el programa dispondrá del objeto *productos_es*, que asocia cada identificador de producto con su nombre. Exactamente como en la práctica 4.

1. La función *crear_productos()* deberá construir y devolver una nueva lista. Cada elemento de esta lista será un objeto con las siguientes propiedades:

   - id: identificador del producto.
   - nombre: nombre del producto.
   - categoria: nombre de la categoría a la que pertenece.

   Para este ejemplo, el resultado sería:

   ```
   [
       {
           id: "p001",
           nombre: "Ensalada de la casa",
           categoria: "Primeros platos"
       },
       {
           id: "p002",
           nombre: "Macarrones al horno",
           categoria: "Primeros platos"
       },
       {
           id: "p101",
           nombre: "Chuletas de cerdo",
           categoria: "Segundos platos"
       },

       ...
   ]
   ```

2. El orden de los elementos en la lista devuelta deberá coincidir con el orden en el que aparecen los productos en *lista_menu*.

3. Las comprobaciones a realizar sobre la lista de entrada serán las mismas que indicaba el enunciado de la práctica 4. Puedes reaprovechar tu código.

4. La función podrá llamar a otras funciones. Y podrá tener el número de parámetros que consideres oportuno.

solución

```
'use strict';

function crear_productos(lista_menu, productos_es) {
    validarListaMenu(lista_menu);

    if (arguments.length !== 2) {
        throw new Error("crear_productos necesita 2 argumentos");
    }

    let productos = [];

    for (let categoria of lista_menu) {
        let nombre_categoria = categoria[0]; // Primer elemento, categoría.

        // Ahora un bucle desde el segundo hasta el final.
        // No usamos for-of porque necesitamos el índice.
        for (let j = 1; j < categoria.length; j++) {
            let id = categoria[j];

            let producto = {};

            producto.id = id;
            producto.nombre = productos_es[id];
            producto.categoria = nombre_categoria;

            productos.push(producto);
        }
    }

    return productos;
}

function validarListaMenu(lista) {
    if (!Array.isArray(lista)) {
        throw new Error("lista_menu no es una lista");
    }

    if (lista.length === 0) {
        throw new Error("lista_menu está vacía");
    }

    for (let categoria of lista) {
        if (!Array.isArray(categoria)) {
            throw new Error("Una categoría no es una lista");
        }

        if (categoria.length < 2) {
            throw new Error("Cada categoría debe contener al menos un producto");
        }

        for (let elemento of categoria) {
            if (typeof(elemento) !== "string") {
                throw new Error("Todos los elementos deben ser cadenas");
            }
        }
    }
}

```

```
function main() {
    let lista_menu =
    [
        [ "Primeros platos", "p001", "p002"],
        [ "Segundos platos", "p101", "p102"],
        [ "Postres", "p201", "p202", "p203"]
    ];

    let productos_es =
    {
        p001: "Ensalada de la casa",
        p002: "Macarrones al horno",
        p101: "Chuletas de cerdo",
        p102: "Merluza a la romana",
        p201: "Flan",
        p202: "Arroz con leche",
        p203: "Tarta de la casa"
    };

    console.log(crear_productos(lista_menu, productos_es));
}

main();
```

Ejercicio 2. (6.0 puntos)

Modifica tus ficheros *carta.html* y *carta.js* para que:

1. El sistema recuerde la fecha y hora en la que el cliente pide cada producto. En la tabla de pedidos habrá una columna adicional, donde se verá la fecha y hora de la última vez que se pidió ese producto: año, mes, día, hora, minutos, segundos. En el formato que quieras, pero legible.

2. Solo se verá fecha y hora de la última unidad pedida de cada producto.

   - Si un cliente pide p.e. cerveza, se verá junto a ese pedido la fecha y hora.
   - Si luego pide p.e. un flan, se verá fecha y hora junto al flan. Naturalmente, la hora de la cerveza seguirá en su sitio.
   - Si luego pide otra cerveza, la hora de la primera cerveza desaparecerá, y será reemplazada por la hora actual. Recuerda: se verá la hora del último pedido de cada producto.
   - Si luego cancela una cerveza, pasará a verse la hora de esta cancelación. Por tanto podemos entender que quitar una unidad de un pedido es, a estos efectos, un pedido.

3. Habrá un botón llamado *borrar pedidos antiguos*. Al pulsarse, se cancelarán todos los pedidos que tengan cierta antigüedad. Donde *cierta antigüedad* será lo indicado por la constante `const ANTIGUEDAD_MAXIMA = 30;`, definida al comienzo del código fuente. Esta constante será un número, que intepretaremos como segundos. Lo razonable sería que esa constante fuera por ejemplo 300 segundos (5 minutos). Pero eso sería incómodo para probarlo. De ahí este valor de 30 segundos.

4. Cuando se pulse el botón *borrar pedidos antiguos*, será equivalente a lo que pasaría si el usuario borra, uno a uno, todos los pedidos de ese producto. Ejemplo: si el cliente pidió 3 cervezas, pero este pedido es *antiguo*, y se pulsa este botón, entonces el sistema cancelará las 3 cervezas.

5. Normalmente los clientes no deberían ver esta información sobre las fechas de los pedidos, ni el borrado de pedidos antiguos. Para ello, el programa tendrá dos modos de funcionamiento.

- *Modo cliente*. La carta no muestra fechas ni botón de borrado.
- *Modo administrador*. La carta sí muestra fechas y botón de borrado.

Un botón en la carta permitirá pasar de un modo a otro. La aplicación responderá inmediatamente a este botón. Cuando el programa esté en *modo cliente*, el texto del botón será *modo administrador*. Y viceversa. Esto es, el botón indica qué hará el botón si se pulsa, no el modo actual.

6. Muy importante: siguiendo el principio básico de diseño en el que tanto hemos insistido durante el curso, la información de fecha y hora deberá almacenarse en estructuras de datos de JavaScript, no extraerse del contenido HTML.