Instrucciones

- La corrección del examen se realizará en términos de (funciona/ no funciona) para cada uno de los apartados y después se entrará a valorar aspectos del código.
- Optimice las reglas utilizando los selectores adecuados, se penalizará redundancias innecesarias.
- En los apartados se le puede indicar que utilice o no determinados elementos, si no lo realiza como se solicita se penalizará, la penalización dependerá de la importancia o gravedad en cuestión.
- NO modifique el fichero HTML
- Como entrega se subirá un archivo .zip con el nombre PO-SegundoTrimestre-<apellidosynombre>.zip
- Es responsabilidad del alumno/a tener todo el software y material necesario para la realización de esta prueba.
- Se permite utilizar apuntes y cualquier otro material en formato digital que el alumno considere de interés para la realización de esta prueba. **NO se permite el uso de Internet**, así como el intercambio de información entre el alumnado. Cualquier acto "sospechoso" invalidará la prueba para las personas implicadas. Si durante la corrección de un ejercicio queda probado que se ha producido una copia a todas las personas implicadas se les invalidará la prueba.
- En cada uno de los apartados de esta prueba**, se penalizará la puntuación si se detectan elementos innecesarios** o que no están relacionados con lo que se pide en el apartado.

Actividad

La actividad consiste en completar el código de la página entregada para que consiga tener la funcionalidad completa que se detalla en este documento.

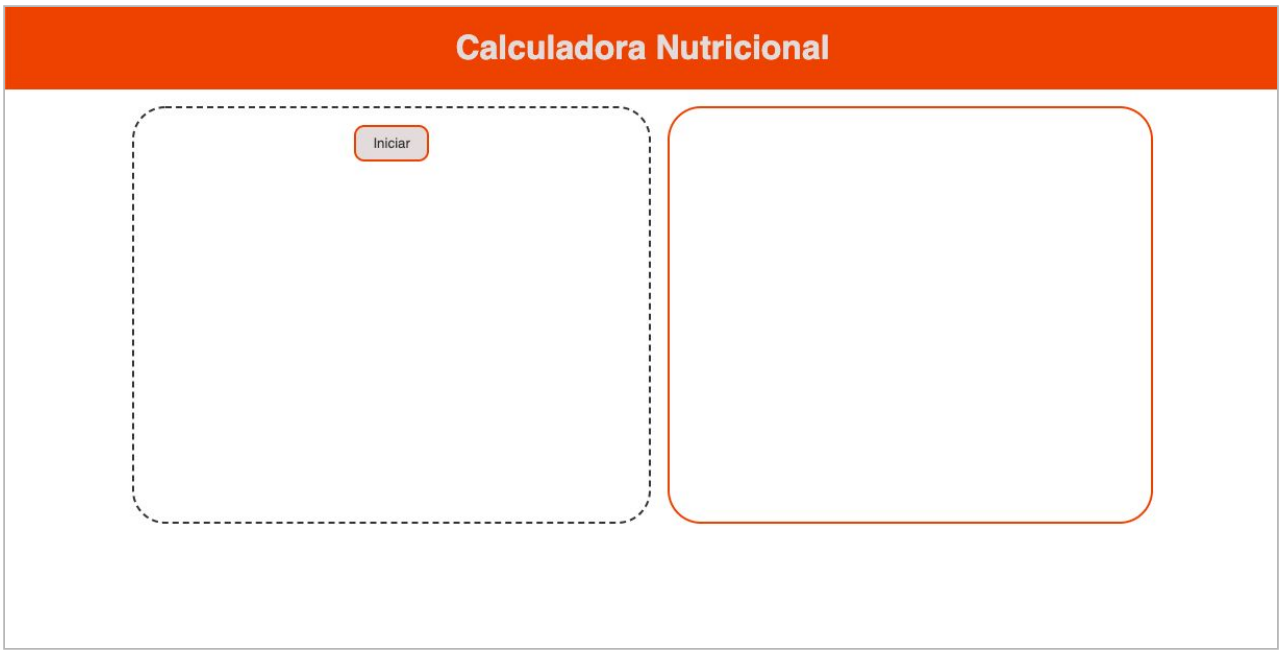
El paquete .zip contiene los siguientes ficheros:

- index.html: En este archivo los cambios a realizar estarán relacionados con la parte del formulario, donde habrá que añadir los atributos y elementos necesarios para que cumpla con los requisitos que se solicitan. Tambien habrá que añadir el/los scripts necesarios para enlazar la funcionalidad de la página.
- styles.scss: Este archivo contiene todos los estilos de la página y no habrá que añadir ni quitar nada. Será necesario usar el plugin “Live SASS compiler” de VSCode para generar el CSS que se enlaza en index.html
- main.js: Este es el archivo principal de la funcionalidad de la página. Contendrá toda la funcionalidad excepto la que haya que incluir en *ajax.js* o en otros archivos que se quieran crear para modularizar más el código.
- ajax.js: Este archivo contiene la función *getProducts*, que acepta una URL del *mock* con los productos por parámetro, se encargará de hacer la petición ajax (a elegir por el alumno) y tendrá que devolver el resultado a quien la invoque. Esta función no interactúa con la parte visual, ya que su función es retornar los datos recuperados en la llamada GET.
- mocks/products.json: contiene un JSON con un listado de productos de alimentación y sus valores nutricionales.
- lib/ : contiene las librerías de *jquery* y *jqueryUI*
- img/loader.gif: es el gif que habrá que mostrar mientras se realice la carga de productos.

FUNCIONAMIENTO DE LA PÁGINA

ESTADO INICIAL

Al cargar la página encontraremos lo siguiente:

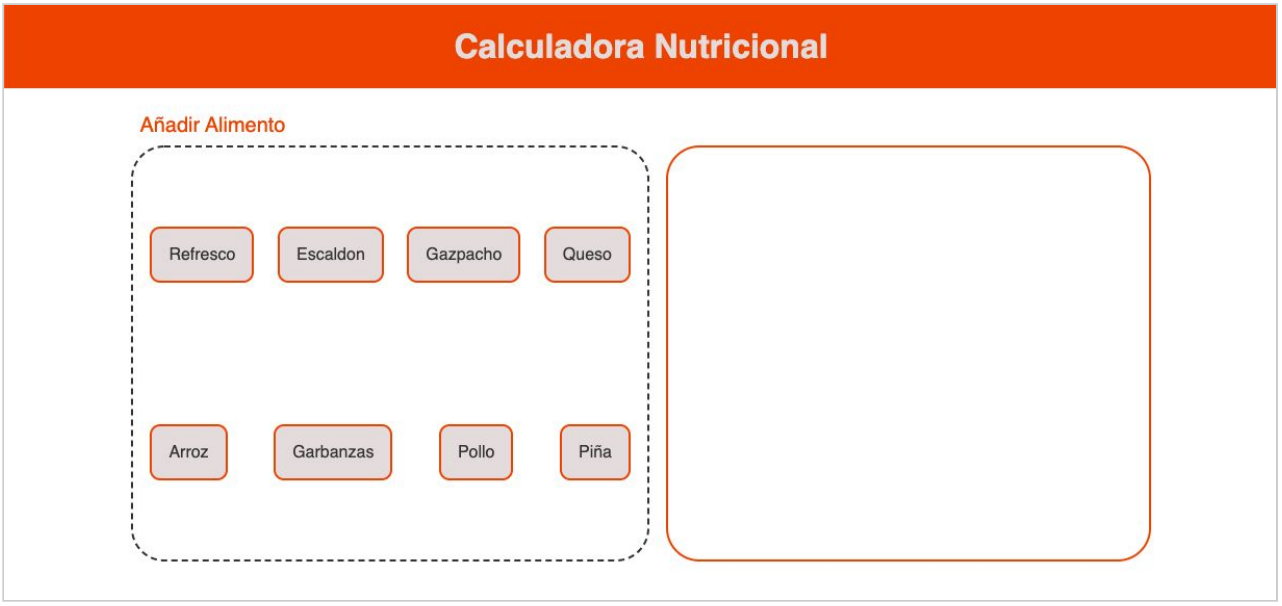


FLUJO PRINCIPAL

Para empezar habrá que pulsar el botón “Iniciar” para intentar recuperar los productos mediante la llamada al método *getProducts* de *ajax.js.* Esta llamada puede dar lugar a las siguientes dos situaciones:

- Se obtiene *null* por respuesta, en cuyo caso se mostrará un mensaje de error como se aprecia en la siguiente captura.

- Se obtiene una lista de productos. En este caso habrá que añadir estos productos a la región izquierda y hacer visible el *nav*, dando lugar a algo como lo siguiente:



En este caso habrá que tener en cuenta los siguientes aspectos:

- Los elementos que se añaden al DOM han de ser creados mediante clonación del template “productTemp” que se encuentra al final del *index.html*.
- Contendrán un atributo data con el código identificador del producto.
- Estos elementos tendrán que ser *draggables*
- En este punto se podrá hacer visible el *nav* que contiene el html, para poder crear nuevos productos más adelante.

Tanto si hay éxito como si no, habrá que tener en cuenta lo siguiente:

- Se tendrá que mostrar el loader proporcionado para indicar al usuario que se está realizando algo y la página no está congelada.
- Mientras se espera por el resultado, el botón de “Iniciar” tendrá que estar **deshabilitado.**

Para que el loader pueda ser visible, habrá que simular una ralentización de la petición ajax incluyendo la siguiente sentencia entre la muestra del loader y la petición a la función ajax

```
await new Promise(r => setTimeout(r, 3000));
```

Los productos cargados se podrán arrastrar a la región de la derecha. Cada vez que esto se haga, se mostrará por consola el resumen del total de nutrientes aportados por el conjunto de productos que se encuentren en la región de la derecha.

```
Información nutricional:
Grasas: 12
Hidratos: 60
Proteinas: 37
```

Todo este comportamiento se puede ver con más detalle en el video *flujo_normal.mov* que se incluye en el .zip de la actividad.

CREACIÓN DE PRODUCTOS

Además de los productos que se recuperan por AJAX, el usuario podrá añadir sus propios productos mediante un formulario.

Pulsando en “Añadir formulario” se mostrará el formulario, cuyo funcionamiento tendrá que ser similar al de la actividad de formularios realizada en clase.

Para poder crear un producto, todos los campos del formulario tendrán que pasar la validación correspondiente.

Pulsar el botón “Añadir producto” pasará la validación de todos los campos.

Los requisitos de validación de cada campo son los siguientes:

- Campo *code*:
    - Es obligatorio
    - Es de tipo texto
    - Tiene que tener un formato específico, teniendo que estar formado por 3 cifras y siendo la primera un 1.
    - Se ha de comprobar que no exista ningún producto con el código introducido
- Campo *name*:
    - Es obligatorio
    - Es de tipo texto
    - Sólo puede contener letras, mayúsculas o minúsculas.
    - Ha de tener un mínimo de 3 caracteres
- Campos *fat*, *hydrates* y protein
    - Son obligatorios
    - Son de tipo numérico
    - Su valor mínimo es 1

Otros aspectos que hay que tener muy en cuenta de forma general a la hora de implementar este formulario son las siguientes:

- Se han de informar al usuario si un campo no es válido, ofreciendo un mensaje **específico** para cada uno de los requisitos de validación expuestos arriba.
- Todos los requisitos de validación se han de reflejar como atributos en el input correspondiente.
- Todas las validaciones se han de realizar con la propiedad *validity*, checkeando los requisitos incluidos como atributos en cada uno de los campos.
- Todos los campos se validan para cada carácter introducido/quitado del campo.

- Al pulsar en el *label* de un campo, se pondrá el foco en el input correspondiente.
- Cada campo ha de mostrar un texto por defecto como se muestra en el video.
- Se ha de evitar que se produzca una validación por defecto de los campos del formulario.
- Los campos que pasen la validación pasarán a tener la clase *success-field,* mientras que los que no la pasen tendrán la clase *error-field*, ambas proporcionadas en la hoja de estilos.
- El envío del formulario no puede producir un refresco de la página.
- Se valora positivamente que el envío del formulario lo despierte el evento *submit* del formulario.
- En un formulario válido, la funcionalidad llevada a cabo será la de crear un nuevo producto y añadirlo al DOM en la región correspondiente.
- Tras la creación del producto, el formulario tendrá que volver a su estado inicial.
- Para simular que la creación del producto es una llamada POST asíncrona, se incluirá la siguiente sentencia desde que se considera válido hasta que se añade el elemento al DOM:

```
await new Promise(r => setTimeout(r, 3000));
```

- Durante el tiempo de espera, el botón aparecerá deshabilitado y cambiará su texto a “Cargando…”

El comportamiento del flujo completo de crear producto se puede observar en el vídeo incluido en el .zip de la actividad.