Entrega 8 - Quiz cliente

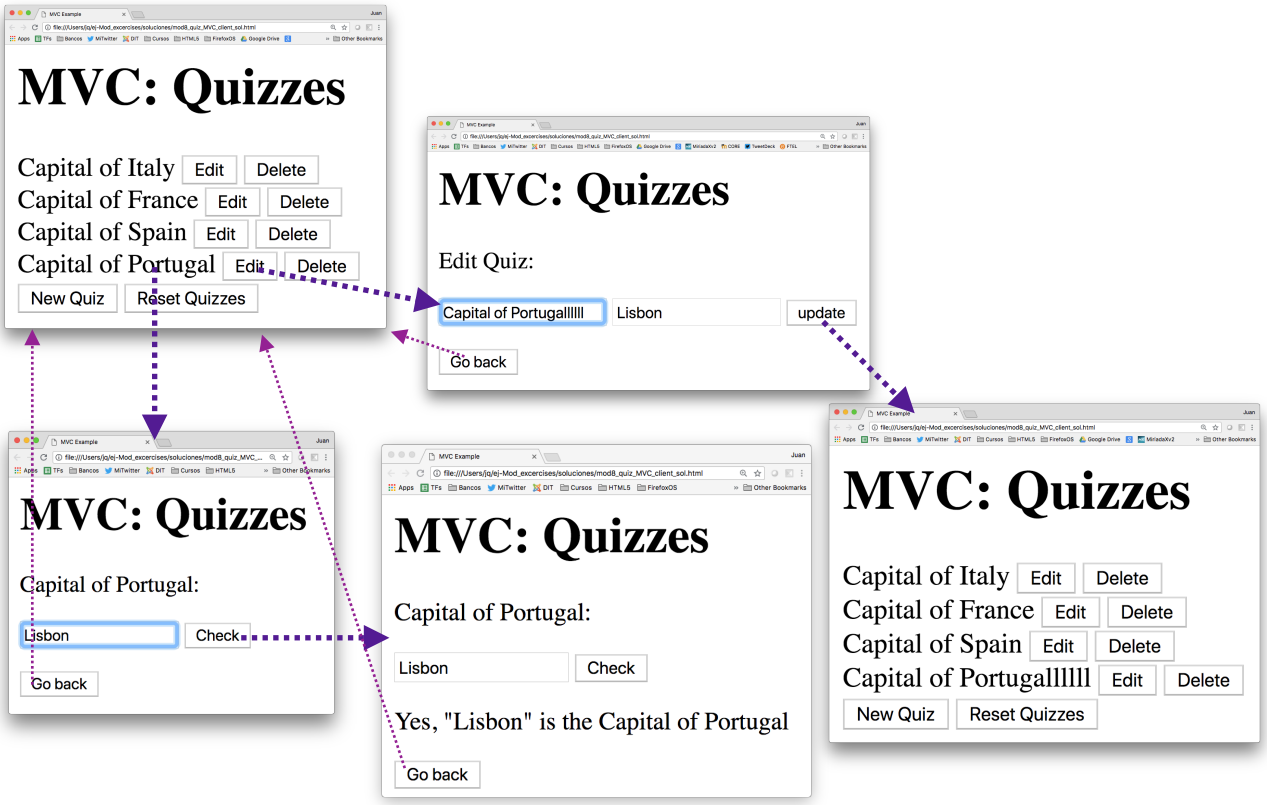
Versión: 12 de Febrero de 2019

Objetivo

Practicar con una aplicación de cliente realizada con HTML, CSS y JavaSCript y estructurada con MVC de cliente. Entender el uso y la problemática de localStorage como almacén de datos.

Descripción de la práctica

El fichero **mod8_quiz_MVC_client.html** contiene el esqueleto de una aplicación Web de cliente estructurada con el modelo MVC (Model-Vista-Controller). Este esqueleto se descarga del proyecto de prueba en https://github.com/practicas-ging/mooc_node-mod8_quiz_mvc_client). Al abrir **mod8_quiz_MVC_client.html** en un navegador se puede acceder a las siguientes vistas:



Al abrir este esqueleto de app en un navegador (hacer doble click en el fichero) nos muestra la lista de preguntas con botones asociados (Edit, Delete, New_Quiz y Reset_Quizzes). Pero este esqueleto de código funciona correctamente solo en lo relacionado con jugar a los quizzes (hacer click en la pregunta) y editarlos, tal y como muestran las capturas adjuntas.

Tal y como muestran las flechas, al hacer click en el texto de una pregunta (quiz), aparece el formulario para enviar la respuesta. Al enviar la respuesta con el formulario nos indica si es correcta o no. El esqueleto de app también permite editar cada uno de los quizzes mostrados, tal y como muestran las flechas entre capturas.

Pero este esqueleto no responde a los botones de crear, borrar o resetear los quizzes y en este ejercicio debe añadirse el código necesario al esqueleto del fichero **mod8-quiz_MVC_client.html**, en los lugares donde se indica con un comentario, para que funcione correctamente, en particular:

- El código con el que se instalan en el router los manejadores de los eventos asociados a clicar los botones: Delete, New_Quiz y Reset_Quizzes.
- El código de los controladores de los eventos *new*, *create*, *delete* o *reset*.

El esqueleto contenido en el fichero **mod8-quiz_MVC_client.html** se adjunta con este enunciado. Este esqueleto utiliza la librería jQuery (v3.3.1) que importa de la CDN de jQuery con un elemento <script type="text/javascript" src="https://code.jquery.com/jquery-3.3.1.min.js" >.

```
========== mod8-quiz_MVC_client.html: Comienzo del código de la ap. de cliente ============

<html>
<head><title>MVC Example</title><meta charset="utf-8"></head>
<script type="text/javascript" src="https://code.jquery.com/jquery-3.3.1.min.js" >

</script><script type="text/javascript">

    // MODEL

const initial_quizzes = JSON.stringify([
        { question: "Capital of Italy",    answer: "Rome" },
        { question: "Capital of France",   answer: "Paris" },
        { question: "Capital of Spain",    answer: "Madrid" },
        { question: "Capital of Portugal", answer: "Lisbon" }
    ]);

localStorage.quizzes = localStorage.quizzes || initial_quizzes;



    // VIEWs

const index = (quizzes) => quizzes.reduce((ac, quiz, id) =>
      ac +=`  <div>
            <span    class="play"   quizid="${id}" >${quiz.question}</span>
             <button class="edit"   quizid="${id}">Edit</button>
             <button class="delete" quizid="${id}">Delete</button>
        </div>\n`,
        ""
    )
  + `<button class="new"> New Quiz</button>
      <button class="reset">Reset Quizzes</button`;

const play = (id, question) =>
`   ${question}:
    <p>
        <input  type="text"  id="answer"  placeholder="Answer" >
        <button  class="check"  quizid="${id}">Check</button>
        <p>
            <div id='msg'></div>
        </p>
    </p>
    <button  class="index">Go back</button>
`;

const quizForm = (msg, event, id, question, answer) =>
`   ${msg}:
    <p>
        <input  type="text"  id="question" value="${question}" placeholder="Question" >
        <input  type="text"  id="answer"   value="${answer}"   placeholder="Answer" >
        <button class="${event}" quizid="${id}">${event}</button>
    </p>
    <button class="index">Go back</button>
`;
```

```
 // CONTROLLERs

const indexController = () => {
    let quizzes = JSON.parse(localStorage.quizzes);
    $('#main').html(index(quizzes));
};

const playController = (id) => {
    let { question } = JSON.parse(localStorage.quizzes)[id];
    $('#main').html(play(id, question));
};

const checkController = (id) => {
    let {question, answer } = JSON.parse(localStorage.quizzes)[id];
    let my_answer = $('#answer').val();
    let response = (my_answer===answer) ?
          `Yes, "${my_answer}" is the ${question}`
        : `No, "${my_answer}" is not the ${question}`;
    $('#msg').html(response);
};

const editController = (id) => {
    let {question, answer} = JSON.parse(localStorage.quizzes)[id];
    $('#main').html(quizForm('Edit Quiz', 'update', id, question, answer));
};

const updateController = (id) => {
    let quizzes = JSON.parse(localStorage.quizzes);
    quizzes[id].question = $('#question').val();
    quizzes[id].answer = $('#answer').val();
    localStorage.quizzes = JSON.stringify(quizzes);
    indexController();
};

const newController = () => {
    // ..... introducir código
};

const createController = () => {
    // ..... introducir código
};

const deleteController = (id) => {
    // ..... introducir código
};

const resetController = () => {
    // ..... introducir código
};



   // ROUTER

const eventsController = ()=> {
  $(document).on('click','.index',  ()  => indexController ());
  $(document).on('click','.play',   (e) => playController  (Number($(e.target).attr("quizid"))));
  $(document).on('click','.check',  (e) => checkController (Number($(e.target).attr("quizid"))));
  $(document).on('click','.edit',   (e) => editController  (Number($(e.target).attr("quizid"))));
  $(document).on('click','.update', (e) => updateController(Number($(e.target).attr("quizid"))));

    // ..... instalar controladores de eventos new, create, delete y reset
}
```

```
$(function(){
  indexController();
  eventsController();
});
</script>
</head>



<body>
    <h1>MVC: Quizzes</h1>
    <div id="main"><div>
</body>
</html>

========== Final del código ============
```

El esqueleto de esta app esta estructurada con el **patron MVC** (Model-Vista-Controller). **MVC** se utiliza mucho en aplicaciones Web de cliente y de servidor, porque estructura de forma clara y concisa una aplicación.

El Modelo.

El modelo (la lista de quizzes) define el almacén y el formato de datos. Es un array de objetos (de tipo quiz), guardado en formato JSON en la propiedad *quizzes* de *localStorage* del navegador donde se abre la app. El modelo se inicializa con:

```
[    { question: "Capital de Italia", answer: "Roma" },
     { question: "Capital de Francia", answer: "París"},
     { question: "Capital de España", answer: "Madrid"},
     { question: "Capital de Portugal", answer: "Lisboa"}
]
```

Hay que ser conscientes, que todos los cambios que se hagan quedarán guardados en la memoria del navegador, aunque cerremos dicha app. Siempre que volvamos a abrir dicha app en ese mismo navegador, veremos los cambios que hicimos en ese navegador y solo esos. Pero si abrimos la app en otro navegador los cambios del navegador anterior no se verán, solo los que se hicieron en el navegador en el que está abierta la app en ese momento. Conviene probar la app en varios navegadores para comprobarlo.

Además, esta app funciona en los navegadores Chrome o Firefox, pero no en Safari porque las mayores restricciones de seguridad por defecto de este último navegador no permiten guardar en *localStorage*. Solo ha sido probada en esos tres navegadores.

Las Vistas.

Las vistas generan el código HTML que se inserta con el método *html()* de jQuery en el bloque * ,* como respuesta a los eventos que la aplicación recibe. Las vistas son en esta app funciones JavaScript que insertan los parámetros recibidos en el código HTML que devuelve. El tipo de evento se define con la clase asociada a cada elemento HTML utilizado para interaccionar.

La vista ***index(quizzes)*** es la más compleja y la renderiza *indexController()*. Esta recibe el array con todos los quizzes como parámetro y genera la lista HTML de las preguntas de los quizzes con los botones asociados de Edit y Delete, además de los botones New_Quiz y Rest_Quizzes al final.

Los elementos HTML (texto, botón Edit y botón Delete) de cada quiz llevan el atributo *quizId* con el índice de su objeto en el array (*quizId="${id}"*) para indicar el quiz asociado. Así cuando hay un evento de tipo *play, edit* o *delete*, se puede saber a que quiz del array se refiere. Los botones New_Quiz y Reset_Quizes no están asociados a ningún quiz particular y por eso llevan solo el atributo class que define los eventos *new* y *reset*.

La vista ***play(id, question)*** es el formulario con un quiz que renderiza *playController(id)*. La respuesta se envía al clicar Check, que genera el evento *check*, el cual inserta el resultado en el bloque * * de la misma vista. La vista tiene además el botón Go_Back que genera el evento *index* que vuelve a mostrar la lista de quizzes.

La vista ***quizForm(msg, event, id, question, answer)*** se utiliza para renderizar el formulario de edición de quizzes en *editController()*. Este formulario debe re-utilizarse para renderizar el nuevo formulario de creación de quizzes en *newController()*.

Los Controladores.

indexController() renderiza la lista de los quizzes, junto con los botones para editar, borrar, crear y resetear. Regenera el array a partir del almacenado en formato JSON en *localStorage.quizzes* y renderizar la vista *index(quizzes)* con los quizzes guardados.

playController(id) renderiza el formulario para jugar con el quiz, identificado por *id*, al ocurrir el evento *play* (clicar en el texto del quiz)

checkController(id) comprueba si la respuesta enviada es correcta al ocurrir el evento *check* (clicar en el botón Check).

editController(id) renderiza el formulario de edición de un quiz al ocurrir el evento *edit* (clicar el botón Edit), inicializando los cajetines del formulario con la pregunta y la respuesta del quiz identificado por *id*.

updateController(id) actualiza el modelo con las nuevas pregunta y respuesta tecleadas en el quiz identificado por *id,* al ocurrir el evento *update* (clicar el botón update en vista quizForm). Finaliza renderizando la vista index.

Los nuevos controladores a diseñar deberán hacer lo siguiente:

newController() deberá renderizar el formulario de creación de un quiz (vista *quizForm(msg, event, id, question, answer)*). Se invoca al ocurrir el evento *new* (clicar el botón New_Quiz). El parámetro *id* se puede inicializar a 0 o a cualquier valor, porque no se utiliza en la creación de quizzes. Los parámetros *question y answer* deberán llevar el string vacío (“”) para que al renderizar los cajetines del formulario se muestren vacíos.

createController() debe actualizar el modelo introduciendo el nuevo quiz al final del array de quizzes. Se invoca al ocurrir el evento *create* (clicar el botón create), al ocurrir el evento *update* (clicar el botón create en vista *quizForm*). Debe finalizar renderizando la vista index.

deleteController(id) debe actualizar el modelo eliminando el quiz identificado for *id* del array de quizzes. Se invoca al ocurrir el evento *delete* (clicar el botón Delete). Antes de eliminarlo debe pedir confirmación con el método *confirm()* de JavaScript de cliente, que genera un pop-up. Debe finalizar renderizando la vista index.

resetController() debe actualizar el modelo con los quizzes iniciales, guardados en la constante *initial_quizzes*. Debe finalizar renderizando la vista index.

El Router de Eventos.

El router asocia manejadores a eventos de forma que la ocurrencia de un evento invoque el controlador asociado. Para ello se utiliza el método on(..) de jQuery de la siguiente forma:

$(document).on('click','.clase_x', (..) => clase_xController(..)). Aunque el manejador se asocia al evento 'click' sobre cualquier elemento HTML de *document* (página mostrada), el segundo parámetro filtra a los que pertenezcan a la clase *clase_x, de forma que solo se atienda al 'click' en los botones y elementos identificados por clase_x*. Por eso se identifica el tipo de evento con el nombre de la clase. Además se ha seguido el convenio de nombres habitual en MVC: el nombre del evento y el de la primera parte del controlador son el mismo.

El evento *index* se asocia al manejador que invoca el controlador indexController().

El evento *play* se asocia al manejador que invoca el controlador playController(id). El parámetro *id* se extrae del atributo *quizId* del objeto asociado al evento sobre el que se ha clicado, que esta disponible como parámetro del controlador.

El evento *check* se asocia al manejador que invoca el controlador checkController(id). El parámetro *id* se extrae del atributo *quizId* del objeto asociado al evento sobre el que se ha clicado.

El evento *edit* se asocia al manejador que invoca el controlador editController(id). El paramétro *id* se extrae del atributo *quizId* del objeto asociado al evento sobre el que se ha clicado.

El evento *update* se asocia al manejador que invoca el controlador updateController(id). El parámetro *id* se extrae del atributo *quizId* del objeto asociado al evento sobre el que se ha clicado.

A este router hay que añadir los maneadores de los eventos *new*, *create, delete y reset* de forma que invoquen los controladores asociados a estos eventos, pasando el parámetro *id* cuando lo necesite en el controlador.

Prueba de la práctica

Para comprobar que la práctica ha sido realizada correctamente hay que utilizar el validador de este repositorio

https://github.com/practicas-ging/mooc_node-mod8_quiz_mvc_client

Recuerde que para utilizar el validador se debe tener node.js (y npm) (https://nodejs.org/es/) y Git instalados. El proyecto se descarga, instala y ejecuta en el ordenador local con estos comandos:

```
$                                ## El proyecto debe clonarse en el ordenador local
$ git clone https://github.com/practicas-ging/mooc_node-mod8-quiz_MVC_client
$
$ cd mooc_node-mod8-quiz_MVC_client         ## Entrar en el directorio de trabajo
$
$ npm install                      ## Instala el programa de test
$
$                              ##  ->  Incluir la solución en el esqueleto clonado
$
$ npm run checks                              ## Pasa los tests al fichero solicitado
.....................................    ## en el directorio de trabajo
.....................................
... (resultado de los tests)
$
```

Una vez descargado el proyecto, se debe entrar en el directorio raiz **mod8-quiz_MVC_client**. El fichero **mod8-quiz_MVC_client.html**, esta incluido (incompleto) en el directorio raíz del proyecto descargado. Este debe completarse con el editor o sustituirse por otro del mismo nombre que contenga la solución.

Los tests pueden pasarse las veces que sea necesario. Si se pasan nada más descargar el proyecto, indicarán que no se ha realizado ninguno de los tres comandos. También pueden utilizarse para probar el programa de otro compañero sustituyendo los ficheros que se desee probar. El programa de test incluye además un comando para generar el fichero ZIP

```
$
$ npm run zip         ##  Comprime los ficheros del directorio en un fichero .zip
$
```

Este genera el fichero **mooc_node-mod8-quiz_MVC_client_entregable.zip** con el directorio de la practica comprimido. Este fichero ZIP debe subirse a la plataforma para su evaluación.

Instrucciones para la Entrega y Evaluación.

Se debe entregar el fichero **mooc_node-mod8-quiz_MVC_client_entregable.zip** con los ficheros comprimidos de la entrega.

El evaluador debe descargar el fichero entregado y comprobar que funciona correctamente. El fichero descargado es un paquete npm, que puede instalarse con todas sus dependencias con npm. Una vez instalado puede ejecutarse o pueden pasarse los test.

RUBRICA. Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:
- 20%: Si el controlador **newController** esta bien implementado
- 20%: Si el controlador **createController** esta bien implementado
- 20%: Si el controlador **deleteController** esta bien implementado
- 20%: Si el controlador **resetController** esta bien implementado
- 20%: Si la instalación de los 4 controladores en el router está bien realizada

Si pasa todos los tests se deberá dar la máxima puntuación.

El objetivo de este curso es sacar el máximo provecho al trabajo dedicado y para ello lo mejor es utilizar las evaluaciones para ayudar al evaluado, especialmente a los principiantes. Al evaluar se debe dar comentarios sobre la corrección del código, su claridad, legibilidad, estructuración y documentación, siempre que puedan ayudar al evaluado.

¡Cuidado! Una vez enviadas, tanto la entrega, como la evaluación, no se pueden cambiar. Esperar a tener completa y revisada, tanto la entrega, como la evaluación antes de enviarlas.