Entrega 9: RandomPlay

Versión: 9 de Abril de 2019

Objetivo

Entender el uso de la sesión para guardar valores entre transacciones HTTP y de las búsquedas específicas en la BBDD. Para ello se debe añadir al servidor Quiz una nueva función que ofrezca jugar a todas las preguntas en un orden aleatorio, sin repetir ninguna.

Prueba de la práctica

Utilizar este validador para comprobar que la práctica realizada es correcta

https://github.com/practicas-ging/CORE19-09_quiz_random

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

```
$                                ## El proyecto debe clonarse en el ordenador local
$ git clone https://github.com/practicas-ging/CORE19-09_quiz_random
$
$ cd CORE19-09_quiz_random                  ## Entrar en el directorio de trabajo
$
$ npm install                               ## Instala el programa de test
$
```

El proyecto **CORE19-09_quiz_random** descargado solo contiene el entorno de test. Clonar a continuación el proyecto **quiz_2019** en el directorio **CORE19-09_quiz_random** del proyecto de prueba y crear la rama **entrega9** en el commit **Step 10 - Heroku Postgres** con:

```
$
$ git clone https://github.com/CORE-UPM/quiz_2019     ## clona el proyecto
$ cd quiz_2019                                        ## entrar en el proyecto
$
$ git checkout -b entrega9  30ed272                   ## crear rama de la entrega
$   …..
$
```

A partir de este momento ya se puede añadir código al esqueleto, pasar los tests o arrancar el servidor para acceder con un navegador. Para arrancar el servidor y acceder con un navegador en http://localhost:5000, se debe estar en el directorio **quiz_2019**:

```
$
$ npm install              ##  instala el proyecto generado
$
$ npm start                ##  arranca con el script de arranque de package.json, o
$
$ node bin/www             ##  arranca el fichero directamente, o
$
$ npx supervisor bin/www   ##  arranca con el supervisor* si está instalado
$
```

***Nota**: **supervisor** se puede instalar también con la opción -g y el comando sudo. En ese caso, se podrá invocar por su nombre (**supervisor**) como los demás comandos UNIX (sin **npx**).

Los tests se pueden ejecutar así (se debe estar en el directorio `CORE19-09_quiz_random`):

```
$
$     # este comando debe invocarse estando en CORE19-09_quiz_random
$
$ npm run checks                            ## Pasa los tests al proyecto en quiz_2019/
.....................................
.....................................
... (resultado de los tests)
$
```

Los tests pueden pasarse las veces que sea necesario. Si se pasan nada más descargar el proyecto, los test fallarán. También pueden utilizarse para probar otro desarrollo completo cambiando el proyecto completo en el directorio **quiz_2019** por otro.

El programa de test incluye además el siguiente comando para generar el fichero ZIP a entregar. Antes de generar el fichero ZIP debe haberse cerrado un commit en la **rama entrega9** que contenga todos los desarrollos pedidos.

```
$
$     # este comando debe invocarse estando en CORE19-09_quiz_random
$
$ npm run zip         ##  Comprime los ficheros del directorio en un fichero .zip
$
```

Este genera el fichero **CORE19-09_quiz_random_entregable.zip** con el directorio de la practica comprimido. Este fichero ZIP debe subirse a la plataforma moodle para su evaluación.

Instrucciones para la Entrega y Evaluación.

Se debe entregar el fichero **CORE19-08_quiz_random_entregable.zip** a través de Moodle. El repositorio comprimido en dicho fichero debe contener la entrega en el último commit de la **rama entrega9**.

Descripción

El servidor Quiz debe enriquecerse con una nueva funcionalidad para jugar a todos las preguntas existentes, presentándolas al azar y sin repetir ninguna.

El juego comenzará con una pregunta elegida al azar. Si esta se acierta se presentará la siguiente, elegida al azar también y así sucesivamente. El juego continuará hasta finalizar todos los quizzes de la BBDD o hasta fallar una respuesta. El jugador deberá tratar de acertar el máximo de respuestas.

Para ello deben añadirse las siguientes primitivas a la API:

```
GET /quizzes/randomplay  ## muestra random_play.ejs con una pregunta a contestar
                         ## o muestra random_none.ejs si no quedan preguntas por contestar

GET /quizzes/randomcheck/:quizId?answer=respuesta
                         ## muestra el resultado con random_result.ejs
```

Este desarrollo debe funcionar tanto en local como desplegado en heroku.

Detalles de la realización de esta práctica

Para realizar esta práctica debe clonar el repositorio https://github.com/CORE-UPM/quiz_2019.

Después se debe crear una rama, denominada **entrega9,** en el commit identificado como **"Step 10 - Heroku Postgres".** El último commit de esta rama será la entrega.

Marco de navegación

Para añadir la funcionalidad, lo primero es añadir un enlace a la barra de navegación (en **layout.ejs**) de nombre "**Play**" que envíe una solicitud: GET /quizzes/randomplay.

Router y controladores: GET /quizzes/randomplay

A continuación habrá que definir la ruta y la acción del controlador que atiende a la primitiva:

GET /quizzes/randomplay.

Esta acción del controlador debe buscar una pregunta al azar y devolver, en la respuesta HTTP, la página Web renderizada con la vista **random_play.ejs** y la pregunta que el cliente debe responder.

Esta misma primitiva se utiliza para enviar al cliente las siguientes preguntas del juego, que se renderizarán con la misma vista.

Cuando el cliente ha respondido bien a todas las preguntas de la BBDD, la acción del controlador debe responder renderizando la vista **random_none.ejs** y enviando la respuesta para indicar que el juego ha finalizado con éxito.

Para no duplicar preguntas es necesario guardar los identificadores de las preguntas ya contestadas en algún lugar, para así poder elegir una que tenga un identificador diferente.

HTTP es un protocolo transaccional que no guarda información de los clientes entre transacciones. Se debe utilizar la sesión para guardar un array con los identificadores de las preguntas ya respondidas. Como cada cliente tiene una sesión diferente, varios clientes podrán jugar simultáneamente sin interferir entre si.

Se recomienda crear la propiedad **randomPlay** en el almacén de sesión **req.session** y guardar en ella un array con los ids de las preguntas contestadas anteriormente.

Y cada vez que se busca una nueva pregunta en la BBDD hay que buscarla indicando que su identificador no debe estar incluido en el array **randomPlay** de preguntas contestadas. Esto se puede hacer buscando con **findAll(…)** con las opciones **limit=1** y **[Op.notin]: randomPlay**. Ver:

http://docs.sequelizejs.com/manual/querying.html

Router y controladores: GET /quizzes/randomcheck/:quizId

El formulario de la vista **random_play.ejs** envía la respuesta al pulsar el botón de "check", generando la siguientes solicitud HTTP:

GET /quizzes/randomcheck/:quizId?answer=<respuesta>

donde **quizId** es el id de la pregunta contestada, y **<respuesta>** es el texto de la respuesta introducida.

También habrá que definir la ruta y la acción del controlador que atiende a esta primitiva.

La acción del controlador debe buscar primero la respuesta correcta de la pregunta y comprobar si coincide con la enviada por el cliente. Si coincide, se añadirá su identificador al array **req.session.randomPlay**.

Una vez comprobado si la respuesta es correcta debe generar la respuesta renderizando la vista **random_result.ejs** correctamente configurada.

Solo se debe permitir que el juego continue si se ha acertado.

Vistas a utilizar

Para realizar esta práctica deben usarse los 3 ficheros de vistas siguientes sin modificarlos. Estos ficheros se proporcionan en un ZIP después de este enunciado:

La vista **views/quizzes/random_play.ejs** muestra un quiz y su cajetín de respuesta.

```
=================== random_play.ejs ==================
<h1>
    Random Play:
</h1>

<div>
    Successful answers = <span id="score"><%= score %></span>
</div>

<form method="get" action="/quizzes/randomcheck/<%= quiz.id %>">

    <p>
        Question: <b><%= quiz.question %></b>
    </p>

    <div class="wideRow">
        <input type="text" class="itemWide" id="answer" name="answer"
                           value="" placeholder="Answer" autocomplete="off"/>
        <input type="submit" class="itemNarrow" id="send" value="Check">
    </div>
</form>
=========================================================
```

La vista **views/quizzes/random_none.ejs** muestra el resultado obtenido al finalizar el juego.

=================== random_nomore.ejs ==================

```
<h1>
    End of Random Play:
</h1>

<div>
    Successful answers = <span id="score"><%= score %></span>
</div>
```

```
<p>
    You answered all questions. You are a champion.
</p>

<p>
    <a href="/quizzes"><button>Go Back</button></a>
</p>
============================================================
```

La vista **views/quizzes/random_result.ejs** informa si se contesto correctamente al quiz de la vista anterior.

```
==================== random_result.ejs =================
<h1>
    Random Play:
</h1>

<div>
    Successful answers = <span id="score"><%= score %></span>
</div>

<p>
    The answer <strong> <%= answer %> </strong> is <%= result ? 'right' : 'wrong' %>.
</p>

<p>
    <% if (!result) { %>
    You have failed.
    <a href="/quizzes"> End of Play </a>
    <% } else { %>
    You have succeeded.
    <a href="/quizzes/randomplay"><button>Continue playing</button></a>
    <% } %>
</p>
============================================================
```