Entrega 3 - Quiz Command

Versión: 12 de Febrero de 2019

Objetivo

Practicar con un comando típico en node.js de cierta complejidad que incluya: acceso por consola, módulos, promesas, ficheros en JSON, paquetes npm y todo gestionado con Git.

Resumen de la práctica

Desarrollar un programa **quiz_cmd** que permita:

- Gestionar una lista de preguntas y respuestas: debe permitir que el usuario añada, borre, edite o consulte preguntas con sus respuestas.
- Probar a responder correctamente una pregunta: el usuario podrá elegir una determinada pregunta, e intentar contestarla correctamente.
- Jugar a contestar correctamente todas las preguntas: el usuario le pedirá al programa que presente todas las preguntas existentes de una en una, en orden aleatorio, y sin repeticiones, para intentar acertar las respuestas. Cuando se produzca un fallo se detendrá el juego.

El programa **quiz_cmd** se ejecuta en un terminal. Implementa un interprete de comandos que lee del teclado y atiende los comandos introducidos por el usuario.

El proyecto https://github.com/practicas-ging/CORE19-03_quiz_cmd contiene un esqueleto del programa a realizar, además de las baterías de test para probarlo. El esqueleto descargado tiene la estructura de paquete npm, incluido el fichero package.json. package.json incluye tanto las dependencias del esqueleto del programa a completar, como las del programa de test

El siguiente screencast de video https://www.youtube.com/watch?v=6KjAcYI_Pcg describe los detalles de la creación del esqueleto y puede ser de gran ayuda para entender el programa.

¡Cuidado! El código dado funciona en UNIX/Linux. Para ejecutar en Windows habrá que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

El esqueleto incluye los ficheros **cmds.js**, **main.js**, **model.js** y **out.js** y para realizar la práctica se deben completar los métodos **test**, **play** y **credits** en el fichero **cmds.js,** para que el programa **quiz_cmd** funcione como se indica a continuación.

Comandos del programa quiz_cmd

El programa **quiz_cmd** debe implementar los comandos que se enumeran a continuación:

help o **h**	El programa debe responder a este comando mostrando un mensaje de ayuda que liste todos los comandos existentes y una breve descripción de cada uno de ellos.
list	El programa debe responder a este comando listando todas las preguntas (no deben mostrarse las respuestas) almacenadas en el programa. Cada pregunta se precederá de un identificador numérico que podrá utilizarse en otros comandos para referirse a esa pregunta.

show <id>	En respuesta a este comando, el programa mostrará la pregunta y la respuesta asociadas al identificador id. Debe comprobarse que se ha introducido un valor para id y que éste es correcto.
add	Este comando se utilizará para añadir interactivamente un nuevo quiz al programa. En primer lugar se le pedirá al usuario que introduzca el texto de la pregunta, y a continuación se le pedirá que introduzca el texto de la respuesta correcta.
delete <id>	Este comando se usará para eliminar del programa el quiz asociado al identificador id. Debe comprobarse que se ha introducido un valor para id y que éste es correcto.
edit <id>	Este comando se usará para editar la pregunta y las respuesta del quiz indicado. Primero se le mostrará al usuario el texto de la pregunta para que lo modifique, y a continuación se hará lo mismo con el texto de la respuesta. Debe comprobarse que se ha introducido un valor para id y que éste es correcto.
test <id>	Este comando sirve para que el usuario pruebe el quiz asociado con el identificador id. Se le presentará al usuario la pregunta del quiz para que introduzca una respuesta. Una vez contestada la pregunta, se le indicara si ha contestado correctamente o no. Debe comprobarse que se ha introducido un valor para id y que éste es correcto.
play o **p**	Este es el comando principal del programa. Se usa para jugar a contestar todas las preguntas correctamente. Este comando presentará todas las preguntas existentes de una en una al usuario para que las responda. Si una pregunta se responde correctamente, entonces se debe continuar presentando otra pregunta al azar. Se sigue jugando hasta que se falle una respuesta. El objetivo del juego es alcanzar el mayor número de respuestas acertadas consecutivamente. No deben mostrarse preguntas repetidas. Cuando no existan más preguntas que mostrar, se termina el juego, indicando al usuario el número de quizzes que ha contestado correctamente.
credits	El programa responde a este comando mostrando el nombre de los alumnos autores de la práctica.
quit o **q**	Se usa para terminar la ejecución del programa.

Algunos de los comando anteriores pueden invocarse tecleando únicamente la primera letra del nombre del comando. Así, "h" sería equivalente a ejecutar “help”, "p" a “play” y "q" a “quit”.

Persistencia

El programa debe guardar las preguntas que maneja en un fichero de texto llamado **quizzes.json** para que las modificaciones que se realicen persistan entre diferentes ejecuciones del programa.

El programa debe leer este fichero cuando arranca para cargar las preguntas existentes. Cada vez que se añada, borre o modifique una pregunta, debe actualizarse el contenido de este fichero.

El contenido del fichero **quizzes.json** debe ser un texto JSON con todas las preguntas manejadas. Será un array de objetos quiz, donde cada objeto quiz es un diccionario con las claves question y answer, y los valores asociados serán el texto de la pregunta y el de la respuesta. Un ejemplo del contenido de este fichero puede ser el siguiente:

```
[ { "question": "Capital de Italia",
    "answer": "Roma"
  },
  { "question": "Capital de Francia",
```

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

Si este fichero no existe al arrancar el programa, debe crearse con las preguntas anteriores. Si al leer o escribir en este fichero se produce algún error, debe abortarse la ejecución del programa.

Ejemplo de uso del programa

A continuación se muestra un ejemplo de como debe funcionar este programa. Reproduzca este comportamiento con la mayor exactitud posible, presentando los mismos mensajes ilustrados el ejemplo.

```
$ npm start
   ____    ___    ____    _____      ___            _
  / ___|  / _ \  |  _ \  | ____|    / _ \   _   _  (_)  ____
 | |     | | | | | |_) | |  _|     | | | | | | | | | | |_  /
 | |___  | |_| | |  _ <  | |___    | |_| | | |_| | | |  / /
  \____|  \___/  |_| \_\ |_____|    \__\_\  \__,_| |_| /___|

quiz >
quiz >
quiz >
quiz > help
Commandos:
  h|help - Muestra esta ayuda.
  list - Listar los quizzes existentes.
  show <id> - Muestra la pregunta y la respuesta el quiz indicado.
  add - Añadir un nuevo quiz interactivamente.
  delete <id> - Borrar el quiz indicado.
  edit <id> - Editar el quiz indicado.
  test <id> - Probar el quiz indicado.
  p|play - Jugar a preguntar aleatoriamente todos los quizzes.
  credits - Créditos.
  q|quit - Salir del programa.
quiz > h
Commandos:
  h|help - Muestra esta ayuda.
  list - Listar los quizzes existentes.
  show <id> - Muestra la pregunta y la respuesta el quiz indicado.
  add - Añadir un nuevo quiz interactivamente.
  delete <id> - Borrar el quiz indicado.
  edit <id> - Editar el quiz indicado.
  test <id> - Probar el quiz indicado.
  p|play - Jugar a preguntar aleatoriamente todos los quizzes.
  credits - Créditos.
  q|quit - Salir del programa.
quiz > list
 [0]:  Capital de Italia
 [1]:  Capital de Francia
 [2]:  Capital de España
 [3]:  Capital de Portugal
quiz > show
Error: El valor del parámetro id no es válido.
quiz > show 2
 [2]:  Capital de España => Madrid
quiz > add
 Introduzca una pregunta: Cuantas son 2 + 2
 Introduzca la respuesta 4
 Se ha añadido: Cuantas son 2 + 2 => 4
quiz > list
 [0]:  Capital de Italia
 [1]:  Capital de Francia
 [2]:  Capital de España
 [3]:  Capital de Portugal
 [4]:  Cuantas son 2 + 2
quiz > show 4
 [4]:  Cuantas son 2 + 2 => 4
```

```
quiz > edit 4
 Introduzca una pregunta: Cuantas son 2+2
 Introduzca la respuesta 4
 Se ha cambiado el quiz 4 por: Cuantas son 2+2 => 4

quiz > list
 [0]:  Capital de Italia
 [1]:  Capital de Francia
 [2]:  Capital de España
 [3]:  Capital de Portugal
 [4]:  Cuantas son 2+2
quiz > show 4
 [4]:  Cuantas son 2+2 => 4
quiz > delete 4
quiz > test 2
Capital de España? Madrid
Su respuesta es correcta.
```

```
quiz > test 2
Capital de España?     MADRID
Su respuesta es correcta.
```

```
quiz > test 2
Capital de España? hola
Su respuesta es incorrecta.
```

```
quiz > p
Capital de España? madrid
CORRECTO - Lleva 1 aciertos.
Capital de Italia? roma
CORRECTO - Lleva 2 aciertos.
Capital de Portugal? lisboa
CORRECTO - Lleva 3 aciertos.
Capital de Francia? París
CORRECTO - Lleva 4 aciertos.
No hay nada más que preguntar.
Fin del juego. Aciertos: 4
```

```
quiz > play
Capital de Francia? París
CORRECTO - Lleva 1 aciertos.
Capital de España? hola
INCORRECTO.
Fin del juego. Aciertos: 1
```

```
quiz > credits
Autores de la práctica:
Nombre 1
Nombre 2
quiz > test 55
Error: El valor del parámetro id no es válido.
quiz > test kk
Error: El valor del parámetro id no es válido.
quiz > q
Adios!
```

Realización y 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/CORE19-03_quiz_cmd

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/CORE19-03_quiz_cmd
$
$ cd CORE19-03_quiz_cmd          ## 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 a los ficheros del proyecto
..................................... ## solicitado en el directorio de trabajo
.....................................
... (resultado de los tests)
$
```

El proyecto local descargado estará en el directorio **quiz_cmd**. Los ficheros **main.js**, **model.js** y **out.js** de este directorio están completos y se deben dejar como vienen. El fichero **cmds.js** esta incompleto. Falta el cuerpo de los métodos de los comandos **test**, **play** y **credits**, que deben completarse con el editor. Los tests determinan si los nuevos desarrollos son correctos o no, además de la puntuación alcanzada. Es conveniente guardar versiones en la rama principal, cada vez que se consolide una nueva funcionalidad.

¡Cuidado! El código dado funciona en UNIX/Linux. Para ejecutar en Windows habrá que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

Los tests pueden pasarse las veces que sea necesario y en cualquiera de las etapas de desarrollo. Si se pasan nada más descargar el proyecto, indicarán que no se ha realizado todavía nada. El programa de test incluye además un comando para generar el fichero ZIP una vez finalizada la práctica

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

Este genera el fichero **CORE19-03_quiz_cmd_entregable.zip** con el directorio de la practica comprimido. Este fichero ZIP debe subirse a la plataforma para su evaluación.

Algunas pistas sobre el proyecto y el screencast

El screencast de video https://www.youtube.com/watch?v=6KjAcYI_Pcg describe los detalles de la creación del esqueleto y es de gran ayuda para entender el programa.

Los que tengan tiempo, además de ver el screencast, pueden realizar el proyecto desde cero siguiendo los pasos que describe. Los test también se pueden pasar al desarrollo desde cero. Para ello hay que sustituir el contenido de los ficheros (**cmds.js**, **main.js**, **model.js** y **out.js**) del

proyecto copiado de GitHub, por los desarrollos propios. El fichero package.json no puede ser sustituido, porque se perdería la configuración del entorno de pruebas.

Al arrancar un proyecto se suele crear un directorio. Los ficheros LICENSE y README.md pueden añadirse al principio. El paquete npm se configura invocando en el directorio:

$ npm init

Este comando crea el fichero **package.json**, que inicializa con información que va solicitando a través de la consola.

Los textos grandes de los mensajes de consola se ham implementado usando el **módulo figlet** (https://www.npmjs.com/package/figlet).

Los textos de colores de los mensajes de consola se han implementado usando el **módulo chalk** (https://www.npmjs.com/package/chalk).

Ambos módulos se distribuyen a través del registro central de módulos npm y pueden instalarse e incluirse como dependencias en el fichero **package.json** con el comando npm:

$ npm install --save <modulo>

Es conveniente inicializar el proyecto con un repositorio Git para guardar versiones a medida que se van consolidando pasos del desarrollo.

El intérprete de comandos de consola utiliza el **módulo readline** de nodejs, cuya documentación (https://nodejs.org/dist/latest-v8.x/docs/api/) incluye un ejemplo de intérprete de comandos (CLI).

El acceso al fichero **quizzes.json** se realiza con el módulo **fs** de nodejs.

El error que se produce al intentar leer un **fichero que no existe** es **ENOENT**. En este caso debe crearse el fichero con el contenido por defecto.

Instrucciones para la Entrega.

La entrega debe realizarse subiendo a la plataforma un fichero ZIP con el contenido comprimido del directorio del proyecto **CORE19-03_quiz_cmd,** generado con el comando “npm run zip”. Este contiene los ficheros ***cmds.js, main.js, model.js y out.js, package.json***.