2015

Versión 1.0

Manual

Grupo Trivial 3a

Contents

Visión general

En este manual, se describe el funcionamiento de un módulo, que permite, obtener un fichero de salida compatible con una base de datos a partir de otro de entrada, en un formato predeterminado. Nos centraremos en dos tipos diferentes de visiones:

1. **A nivel de usuario**: esta parte de la aplicación está pensada para que un usuario administrador haga uso de ella. Se explicará cómo se realiza la interacción con la interfaz.
2. **A nivel de sistema**: instrucciones para la utilización de este código por distintos desarrolladores. Gracias a éste se permitirá una lectura rápida que nos ayudará a entender el código y funcionamiento del proyecto para su utilización y modificación posterior. Se explicará detalladamente la funcionalidad de las diversas clases que conforman el proyecto. El sistema está formado por una serie de jerarquías de clases que cooperan entre sí para lograr el correcto funcionamiento de la aplicación.

Manual de usuario

Descripción del funcionamiento de la interfaz que será utilizada por un usuario administrador, a fin de introducir en la base de datos del sistema una lista de preguntas con respuestas verdaderas y falsas. Esto, nos servirá más adelante, para implementar diferentes tipos de juegos que puedan hacer uso de estas preguntas.

Administrador

El administrador, al iniciar la aplicación, solo tendrá acceso a una pantalla, en ella, tendrá una lista de opciones a realizar y se le pedirá que introduzca el número de la opción que desee:

- Introduciendo **0**: salir de la aplicación.
- Introduciendo **1**: nos pedirá que introduzcamos la ruta del fichero con las preguntas que queramos añadir a la base de datos. Éste fichero debe tener uno de los siguientes formatos:
    a. **GITF**
    b. **XML simple (QTI)**

  Si la ruta del fichero es correcta, y el fichero no contiene ningún error, las preguntas se almacenarán en la base de datos.

- Introduciendo **2**: esta opción nos permite leer un fichero de manera automática. Nos pedirá que introduzcamos cada cuanto tiempo queremos que se actualicen los datos del fichero en la base de datos. Por último, nos pedirá que introduzcamos la ruta del fichero como en el caso anterior.
- Introduciendo **otra cosa**: nos mostrará un mensaje indicando que la opción elegida no es correcta.

Manual de sistema

Clases implementadas

Main

Esta clase únicamente se encarga de lanzar la aplicación, en ella se crea una instancia de la clase Extractor y se ejecuta su método run.

Bussiness. Extractor

MenuExtractor

Esta clase se encarga de la interacción con el administrador de la aplicación, en este caso, el usuario. En ella mediante el constructor y otros métodos se imprimen las distintas opciones con las que cuenta el programa, dependiendo de la respuesta del usuario se realizará una tarea u otra.

handleOptions: método que se encarga de realizar una determinada acción dependiendo de la opción introducida por el usuario.

- Si se introduce 0, la aplicación finalizará.
- Si se introduce 1, se comenzará con la lectura y subida de datos de forma manual.
- Si se introduce 2, se realizará la lectura y subida de datos de forma automática cada cierto tiempo.
- En otro caso, no hará nada.

handleTime: este método se encarga de guardar el tiempo que el usuario introduce para realizará la lectura y subida de datos de forma automática.

handleConnection: este método según la opción elegida por el administrador, se conectará con la base para almacenar los datos o no.

Extractor

Realiza la tarea de intermediario ya que se encarga de trasladarle al *parser* la información proporcionada por el usuario que este necesita para su correcto funcionamiento.

En su método principal, run, toma los datos que ha introducido el usuario de la aplicación a través de una instancia de la clase anteriormente descrita, entre ellos el nombre del fichero que va a ser *parseado*.

Para ello realiza una serie de invocaciones a instancias de clases que posteriormente serán descritas así como a métodos propios de la clase:

initialize, lleva a cabo una serie de tareas:

- Selecciona el tipo de *parser* que va a ejecutarse en función del formato de fichero de entrada.

- Le solicita al *parser* seleccionado el mapa con los objetos Pregunta que ha creado a partir del fichero.
- Crea el fichero de salida en el formato JSON correspondiente.

- En función de la elección del usuario lleva a cabo la carga del fichero de salido en la base de datos.

setParser :se encarga de crear la instancia correcta del *parseador* en función del formato del fichero de entrada.

Business. Parser

Parser

Se trata de una clase abstracta que sirve como generalización para los diferentes *parser* que aparecen en la aplicación. Cuenta con un atributo, "**nombreArchivo**" que será heredado por todas sus clases hijas. Se declara la signatura del método abstracto "**getPregunta**" que sus clases hijas implementaran posteriormente.

ParserGIFT - ParserQTI- ParserXML

Cuentan con una función principal, denominada **getPregunta** cuya función es generar un mapa de objetos Pregunta a partir del fichero especificado. Para ello realiza las siguientes acciones:

- Selecciona y carga el fichero adecuada para poder recorrer su contenido.

- Crea las diferentes partes de las cuales consta un objeto de tipo Pregunta, es decir, su identificador, la pregunta, el "*array*" con las posibles respuestas y la respuesta correctas.

- A continuación comienza a recorrer el archivo de entrada para ir identificando los atributos antes mencionados a partir de las especificaciones del formato seleccionado (Anexo1).

- Cada vez que se han identificado en el archivo los cuatro atributos necesarios se crea un objeto Pregunta y este es almacenado en un mapa de preguntas a través de su identificador.

- Finalmente el método retorna el mapa con todas las preguntas que aparecen en el fichero y cierra este.

ParserQTI

Además de la función mencionada anteriormente, esta clase cuenta con un privado **getIDRespuestaCorrecta,** que pasándole por parámetro la lista de respuestas y el id de la pregunta y el "namespace" del "*xml*" , retorna el id de la respuestas correcta.

Business. Preguntas

Pregunta

Se trata de una clase, que representa una pregunta en un formato de datos intermedio. Cuenta con todos los atributos necesarios que forman una pregunta:

- *pregunta*
- *respuestaCorrecta*
- *respuestasIncorrectas*

Este último es de tipo "*array*", ya que cada pregunta cuenta con varias respuestas incorrectas. Además cada uno de estos cuenta con sus métodos de acceso y modificación ("*getters*" y "*setters*") correspondientes.

Serializador

Serializador

Se trata de una interfaz que se usa como puerta de entrada para la transformación del tipo de datos almacenado internamente al formato de salida especificado. Únicamente cuenta con la descripción del método **createFile** que se encargan de implementar sus clases hijas descritas posteriormente, se creará una clase hija por cada formato de salida que el sistema quiera soportar.

Serializador. JSON

JsonSerialImpl

Se trata de la clase concreta que implementa la interfaz anteriormente descrita, en este caso se encarga de transformar los datos internos al formato JSON.

Como ya se sabía esta clase cuenta con la implementación del método **createFile** el cual hace uso de la librería "*Gson*" para crear el fichero de salida. La librería recibe como parámetro los objetos Pregunta que se han extraído del fichero de entrada y los transforma correctamente al formato JSON por lo tanto únicamente queda ir añadiendo los resultados que nos proporciona la librería al fichero deseado que es lo que se realiza al final de dicho método.

Persistencia

Connection

Clase que se encarga de conectar la aplicación con la base de datos en la que se desea trabajar, en este caso utilizaremos el sistema de *base de datos "NoSQL"*, "**mongoDB"**.

Insert

Esta clase se encarga de la inserción en la base de datos del fichero JSON creado por el *Serializador*. Cuenta con una función principal que se encarga de invocar a los métodos necesarios para la incorporación de las preguntas del fichero a la base de datos. Estos métodos son:

- **importJSON**: que se encarga de cargar el fichero generado en la salida para poder leer su contenido.

- **Insert:** se encarga de ir leyendo las diferentes líneas del fichero JSON y mediante la transformación de objetos que permite el uso de la clase JSON.java se introduce cada pregunta en la base de datos. Como las operaciones de lectura de fichero pueden causar problemas se tienen en cuenta varios errores que pueden ocurrir durante el procesamiento del archivo.

Anexos

Anexo 1. Distintos formatos de entrada

Formato GIFT

```
GIT::Q1::

¿Cuantas copas de europa tiene el Real Madrid? {
~ 8
= 10
~ 2
~ 9
}
```

Formato QIT

```
<responseDeclaration identifier="R1"
        cardinality="single" baseType="identifier">
        <correctResponse>
                <value>ChoiceC</value>
        </correctResponse>
</responseDeclaration>

<itemBody>
        <choiceInteraction responseIdentifier="R1"
                shuffle="false" maxChoices="1">
                <prompt>¿Cuantas copas de europa tiene el Real Madrid?</prompt>
                <simpleChoice identifier="ChoiceA">8.</simpleChoice>
                <simpleChoice identifier="ChoiceB">3.</simpleChoice>
                <simpleChoice identifier="ChoiceC">10</simpleChoice>
                <simpleChoice identifier="ChoiceD">10.</simpleChoice>
        </choiceInteraction>
```

```
</itemBody>
```

Formato XML

```
<cuestionario>
   <pregunta id="R1">
      <textoPregunta>¿Cuantas copas de europa tiene el Real Madrid?</textoPregunta>
      <respuestaIncorrecta>8</respuestaIncorrecta>
      <respuestaIncorrecta>2</respuestaIncorrecta>
      <respuestaCorrecta>10</respuestaCorrecta>
      <respuestaIncorrecta>9</respuestaIncorrecta>
   </pregunta>
</cuestionario>
```