2015

DOCUMENTACIÓN

Grupo Trivial 3a

Álvaro Panizo Romano (UO227748)

Carmen Peñalver San Cristóbal (UO229705)

Iñigo Llaneza Aller (UO206367)

Javier Iglesias García (UO232562)

Liliana Villar Iglesias (UO232510)

Marcos Abdel-Fattah Martínez (UO230956)

Pablo Bravo Mediavilla (UO223531)

Raúl Herrador Colino (UO234551)

Versión 1.1

DOCUMENTACIÓN

Ámbito y alcance del problema

En esta primera toma de contacto con el problema nos encuadramos ante un entorno vacío, con limitaciones básicas como son las de crear.

Como limitaciones del parser están los formatos de salida propuestos, básicamente json y los formatos de entrada GIFT, y los propuestos (y aun así desarrollados) de XML.

El alcance del problema será obtener el equivalente en el formato de salida, dado el fichero gift con los datos.

Opcionalmente se plantea un entorno de persistencia como MongoDB.

Planteamiento del problema

Una empresa dedicada a la creación de videojuegos necesita crear un nuevo juego de preguntas/respuestas de naturaleza similar al trivial clásico.

- Es primer lugar es necesario procesar los datos relativos a las preguntas y respuestas de las que constará el juego. Durante este primer procesamiento la aplicación debe informarnos de los posibles errores en los ficheros de entrada. Estos ficheros de entrada deberán de usar el formato GIFT, pero posteriormente la aplicación podría requerir del uso de ficheros de entrada con otros formatos como QTI o XML. Una vez procesados los datos han de tener una representación interna JSON en nuestra aplicación.
- Posteriormente estos datos se serializaran en una base de datos no relacional como MongoDB aunque esto es una decisión aun por tomar y podría haber variaciones.

Los pasos anteriores deberán estar claramente diferenciados para facilitar que una automatización o un operario tengan la posibilidad de ejecutar estas etapas en momentos diferentes.

Por el momento no se requiere que ninguna de estas dos etapas ofrezcan un gran rendimiento, ya que solamente van a ser utilizadas en las fase de construcción de una base de preguntas/respuestas.

Metodología usada

El estudio y diseño de esta arquitectura se llevara a cabo mediante la utilización del método Atribute-Driven-Design (ADD) y el estándar del SEI (ANSI/IEEE 1471, 2000).

El método ADD esta basado en la identificación de los atributos de calidad y la creación de una serie de escenarios que esos atributos deben cumplir.

Identificación de los interesados (Stakeholders)

Este proyecto esta nombrado como “Trivial3a” por lo que los interesados están contextualizados dentro del dicho proyecto.

En este caso los interesados son:

- **Responsables de la empresa:** Los dirigentes que toman las decisiones sobre el proyecto, presupuesto y evolución.
- **Usuario administrador:** estará en contacto con el módulo de parseo para utilizarlo a la hora de cargar datos o introducir nuevos datos.
- **Equipo de desarrollo:** el diseño modular permitirá edificar la aplicación de modo que este módulo no se vea influido por el desarrollo de otros módulos.

#Responsables de la empresa

Se trata de los equipos directivos de la corporación, son responsables de los presupuestos y toman las decisiones que comprometen fondos de dicho presupuesto. En nuestro caso los profesores. Entre sus objetivos están:

- Bajo coste de desarrollo, esto es, el desarrollo del proyecto debe ser corto y con un coste reducido.
- Guiar el proceso de evolución de la aplicación.

#Usuario administrador

El/los usuarios administrador/es son un grupo de usuarios encargados del mantenimiento de la aplicación, en este módulo se encargaran de la carga del fichero mediante la interfaz del parser de modo que su trabajo será simple e intuitivo, y realizado de forma secuencia por la estructura BATCH. Entre sus objetivos:

- Trabajo lo más simple posible, con el menor número de fallos posibles.
- Aprendizaje simplificado e facilidad de la interfaz.

#Equipo de desarrollo

Todos los miembros del equipo, afectados de manera directa pues este módulo será una de las partes básicas de la aplicación a construir. Lo realizaremos de forma independiente para que los módulos puedan cambiarse y modificarse sin necesidad de alterar el resto de módulos. Por tanto sus objetivos simplificados son:

- Simplificar al máximo y modularizar el parser para mejorar la integración en el sistema y el desarrollo de otros módulos.
- Proyecto rentable, esto es, que permita ser desarrollado por el precio establecido con un grado de rentabilidad que haga atractivo el desarrollo. En nuestro caso obtener la mayor nota con el mínimo esfuerzo posible.

Lista de interesados (Stakeholders)

Código	StakeHolder	Intereses
ST-01	Responsables de la empresa	*Bajo coste de desarrollo, esto es, el desarrollo del proyecto debe ser corto y con un coste reducido. Eficiencia e independencia del parser.*
ST-02	Usuario administrador	*Simplificación de aprendizaje y utilización. Fácil interacción con la interfaz.*
ST-03	Equipo de desarrollo	*Proyecto escalable y modularizado. Reparto de tareas y desarrollo eficiente.*

Lista de atributos de calidad

- ✓ AT001 Facilidad de cambio de los mecanismos de parseo y/o almacenamiento de datos por si es necesario modificarlos en el futuro. Modificabilidad.
- ✓ AT002 Escalabilidad del sistema para poder añadirlo a otro tipo de aplicaciones. Modificabilidad.
- ✓ AT003 Facilidad para probar el correcto funcionamiento del sistema, garantía de que los datos se han procesado correctamente. Testabilidad.
- ✓ AT004 Tiempo de desarrollo corto. Time to markey.
- ✓ AT005 Facilidad de uso de la aplicación. Usabilidad.
- ✓ AT006 Coste de desarrollo bajo. Coste-Beneficio.
- ✓ AT007 Protección del equipo que ejecute la aplicación. Confiabilidad.
- ✓ AT008 Garantizar que la aplicación procese los datos correctamente y en un tiempo óptimo. Rendimiento

Atributos de calidad e interesados

Atributos vs Interesados	ST-01	ST-02	ST-03
AT001			X
AT002	X		X
AT003			X
AT004	X		X
AT005		X	
AT006	X		
AT007	X	X	
AT008	X		

Descripción de negocio de la solución

La solución que planteamos esta compuesta de varios módulos integrados todos ellos en una aplicación desarrollada en el lenguaje Java.

- Parser: el modulo encargado de la lectura y transformación a un formato de representación interna en nuestra aplicación de los datos obtenidos a partir de los ficheros de entrada en formato GIFT, QTI o XML.
- Serializador: este módulo es el encargado de pasar a formato JSON los datos que obtuvimos en la fase anterior.
- Modulo de persistencia: el modulo encargado de hacer los datos obtenidos en la fase anterior persistentes. Dicha persistencia se lleva a cabo en la base de datos no relacional MongoDB.

Cada uno de los módulos anteriormente descritos representa una etapa de las que se compone nuestra aplicación **BATCH** de procesamiento por lotes.

- Funcionamiento del módulo parser.

Recibe los ficheros que contienen las preguntas en uno de los formatos aceptados para procesarlos y convertirlos a datos que podamos usar en nuestra aplicación. También se encarga de avisarnos si se encuentra algún error en los datos de entrada durante su procesado.

Cuando los datos de los ficheros de entrada estén completamente procesados estos datos ya estarán listos para ser enviados al módulo de serialización.

- Funcionamiento del módulo de serialización.

Este modulo depende de los datos procesados por el parser para su funcionamiento. Una vez recibe esos datos se encarga de serializar estos datos al formato JSON. Una vez transformados a JSON los datos ya estarán preparados para ser procesados en la siguiente etapa, la persistencia.

- Funcionamiento del modulo de persistencia.

Recibe una serie de datos JSON que mediante una implementación de los métodos de la interfaz JsonSerial serán guardados en la base de datos elegida, MongoDB.

Como se puede observar esta solución elegida se pretende asemejar al estilo arquitectónico de procesamiento por lotes tradicional, el cual consiste en una serie de etapas bien diferenciadas que se ejecutan de manera secuencial, al depender las etapas del resultado obtenido de etapas anteriores.

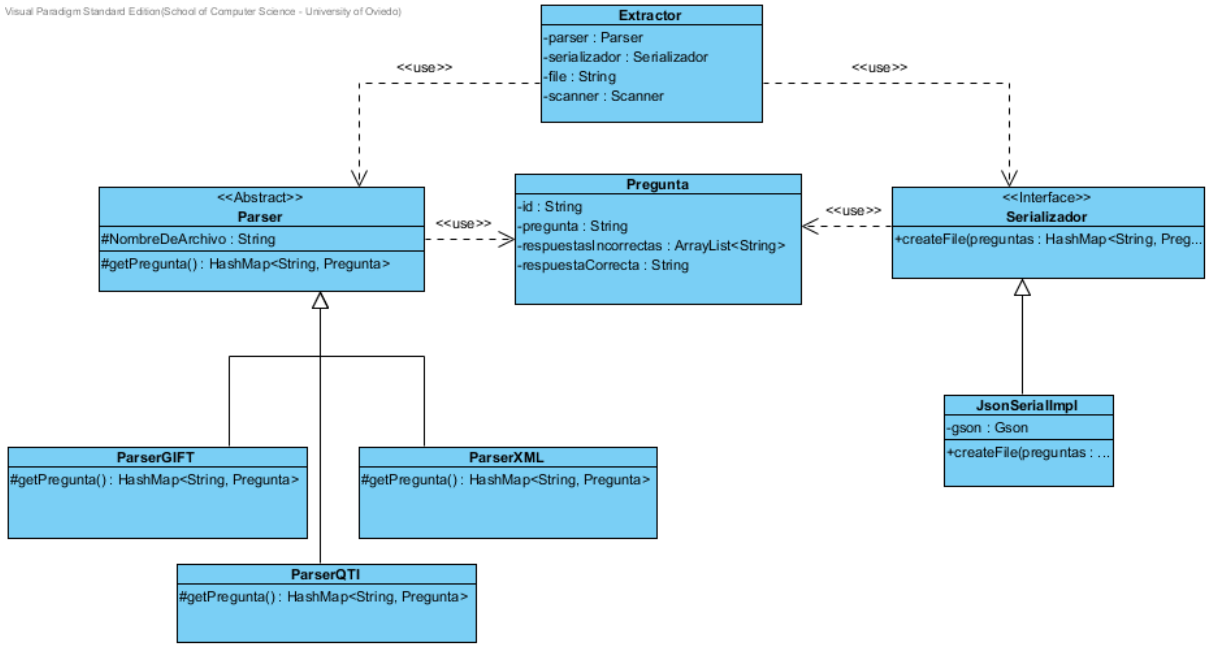
Escenarios de calidad

Escenario Nº	**Fuente de estímulo**	**Estímulo**	**Entorno**	**Artefacto**	**Respuesta**	**Medición de la respuesta**	**Atributo de calidad afectado**
1	Nuevo mecanismo de parseo	Parseo de datos	Explotación	Parser de datos	Actualización de parser	100% de preguntas procesadas correctamente	AT001
2	Nuevo sistema a integrar	Cambio de sistema que utiliza el parser	Explotación	Parser de datos y nuevo módulo donde integrarlo	Sincronización de ambos módulos	0% de fallos en la sincronización	AT002
3	Central control	Recepción de datos procesados	Explotación	Parser de datos y funcionalidad de testeo	Fichero procesado a Json	El 100% de las preguntas del fichero están correctamente procesadas	AT003
4	Sistema en desarrollo	Fin del desarrollo	Desarrollo	Parser de datos	Tiempo de desarrollo	Antes del día 4 de Marzo	AT004
5	Uso del sistema	Uso por usuario final	Explotación	Parser de datos	Opinión de usuarios beta-tester	90% de votos a favor de la aplicación	AT005
6	Sistema en desarrollo	Fin del desarrollo	Desarrollo	Parser de datos	Coste de desarrollo	Sin coste monetario	AT006
7	Uso del sistema	Cualquier momento de uso del sistema	Explotación	Parser de datos e interfaz del mismo	Medidas estándar de protección	No hay amenaza potencial	AT007
8	Uso del sistema	Uso ordinario	Explotacion	Parser de datos	Relación rendimiento óptima	Medidas de Benchmark de menos de 20s	AT008

DOCUMENTACIÓN CASE

Diagrama de Clases

Diagrama de clases - Parser



Resumen

Nombre	Documentación
Extractor	Clase de integración (controladora) de todos los submodulos del modulo principal de parseo, que crea a partir de fichero de entrada el formato de salida desado.
Pregunta	Es la clase principal de parseo puesto que contiene el formato intermedio entre el formato de entrada de la implementación concreta del parser y el formato de salida del serializador. Contiene métodos getter y setter para todos los atributos privados, así como los métodos de hashCode, equals y toString que aportan funcionalidad para comparaciones y otras utilidades.
Parser	Esta clase abstracta actua como superclase para las clases hijas que espeficicaran el tipo de parser según el formato de entrada del archivo que es el atributo de esta clase (nombreArchivo)
Serializador	Esta interfaz forma la superclase de las implementaciones concretas de serializado a formatos de salida concretos. Utiliza el formato intermedio

	generado por el parser y crea el fichero de salida según la implementación concreta.
JsonSerialImpl	Implementación concreta de la superclase para transformar a formato JSON.
ParserGIFT	Esta es la implementación de la superclase para un formato de entrada GIFT, toda la funcionalidad está en el método protegido getPregunta()
ParserXML	Esta es la implementación de la superclase para un formato de entrada XML plano, toda la funcionalidad está en el método protegido getPregunta()
ParserQTI	Esta es la implementación de la superclase para un formato de entrada XML QTI, toda la funcionalidad está en el método protegido getPregunta()

Detalles

Extractor

Nombre	Valor
Documentación	Clase de integración (controladora) de todos los submodulos del modulo principal de parseo, que crea a partir de fichero de entrada el formato de salida desado.

Attributes

private parser : Parser	
Documentación	Instancia de la clase Parser. tendrá una implementación concreta dependiendo del tipo de formato de entrada.
Tipo	Parser

private serializador : Serializador	
Documentación	Instancia de la clase Serializador que tiene una implementacion concreta dependiendo del fichero de salida deseado.
Tipo	Serializador

private file : String	
Documentación	Ruta del archivo de entrada que se pasa al serializador.
Tipo	String

private scanner : Scanner	
Documentación	Scanner para pasar por consola el fichero de entrada.
Tipo	Scanner

Relationships

Unnamed Uso	
To	Parser
Visibilidad	Unspecified
Estereotipos	use
Quality Score	Good

Unnamed Uso	
To	Serializador
Visibilidad	Unspecified
Estereotipos	use

Pregunta

Nombre	Valor
Documentación	Es la clase principal de parseo puesto que contiene el formato intermedio entre el formato de entrada de la implementación concreta del parser y el formato de salida del serializador. Contiene métodos getter y setter para todos los atributos privados, así como los métodos de hashCode, equals y toString que aportan funcionalidad para comparaciones y otras utilidades.
Visibilidad	public

Attributes

private id : String	
Documentación	El identificador de la pregunta, es una marca disponible en las preguntas GIFT y en los otros formatos para diferenciarla de el resto de su tipo.

Tipo	String

private pregunta : String	
Documentación	Este atributo es el enunciado de la pregunta, puede ir entre interrogaciones o ser un enunciado afirmativo.
Tipo	String

private respuestasIncorrectas : ArrayList	
Documentación	Este atributo es una lista donde se incluirán todas las respuestas incorrectas.
Tipo	ArrayList

private respuestaCorrecta : String	
Documentación	En este atributo se guarda la UNICA respuesta correcta de cada pregunta.
Tipo	String

Relationships

Unnamed Uso	
From	Parser
Estereotipos	use
Quality Score	Good

Unnamed Uso	
From	Serializador
Estereotipos	use
Quality Score	Good

Parser

Nombre	**Valor**
Documentación	Esta clase abstracta actua como superclase para las clases hijas que espeficicaran el tipo de parser según el formato de entrada del archivo que es el atributo de esta clase (nombreArchivo)

Visibilidad	public
Estereotipos	Abstract

Attributes

protected NombreDeArchivo : String	
Documentación	`Este atributo será el nombre del archivo de entrada que recibe el parser en uno de los subformatos hijos para realizar la conversion`
Tipo	String

Operations

protected getPregunta () : HashMap	
Documentación	`Este método es el genérico que indicará a los métodos de la subclase para que lo reimplementen especializandolo según el formato de entrada`

Relationships

Unnamed Generalization	
To	ParserGIFT
Quality Score	Good

Unnamed Generalization	
To	ParserXML
Quality Score	Good

Unnamed Generalization	
To	ParserQTI
Quality Score	Good

Unnamed Uso	
To	Pregunta
Visibilidad	Unspecified
Estereotipos	use
Unnamed Uso	

From	Extractor
Visibilidad	Unspecified
Estereotipos	use
Quality Score	Good

Serializador

Nombre	Valor
Documentación	Esta interfaz forma la superclase de las implementaciones concretas de serializado a formatos de salida concretos. Utiliza el formato intermedio generado por el parser y crea el fichero de salida según la implementación concreta.
Activo	false
Visibilidad	public
Estereotipos	Interface

Operations

public createFile (preguntas : HashMap) : void		
Parameters	**preguntas**	
	Multiplicidad	Unspecified
	Template Type Bind Info	N/A
	Tipo	HashMap
	Direction	inout
Documentación	Este método será implementado de manera concreta con las subclases para generar un fichero en el formato de salida deseado.	

Relationships

Unnamed Uso	
To	Pregunta
Visibilidad	Unspecified
Estereotipos	use

Unnamed Generalization	
To	JsonSerialImpl
Quality Score	Good

Unnamed Uso	
From	Extractor
Visibilidad	Unspecified
Estereotipos	use

JsonSerialImpl

Nombre	Valor
Documentación	Implementación concreta de la superclase para transformar a formato JSON.
Visibilidad	public

Attributes

private gson : Gson	
Documentación	Utilizamos la librería Gson de google para parsear el formato intermedio de Hash de preguntas a Json, esta librería consigue con pocas líneas de código una transformación eficiente para introducir en nuestra bdd MongoDB
Tipo	Gson

Operations

public createFile (preguntas : HashMap) : void		
Parameters	**preguntas**	
	Multiplicidad	Unspecified
	Template Type Bind Info	N/A
	Tipo	HashMap
	Direction	inout
Documentación	Implementación concreta del método de la superclase para transformar el formato intermedio a un fichero json que meter en la bdd.	

Relationships

Unnamed Generalization	
From	Serializador

N/A

Nombre	Valor
Visibilidad	Unspecified
Quality Score	Good

ParserGIFT

Nombre	Valor
Documentación	Esta es la implementación de la superclase para un formato de entrada GIFT, toda la funcionalidad está en el método protegido getPregunta()
Activo	false
Visibilidad	public

Operations

protected getPregunta () : HashMap	
Documentación	Este método es la implementación concreta del método de la superclase. En este caso toma el fichero de entrada en formato GIFT y lo parsea para generar un formato intermedio en un HashMap

Relationships

Unnamed Generalization	
From	Parser
Quality Score	Good

ParserXML

Nombre	Valor
Documentación	Esta es la implementación de la superclase para un formato de entrada XML plano, toda la funcionalidad está en el método protegido getPregunta()
Visibilidad	public

Operations

protected getPregunta () : HashMap	
Documentación	Este método es la implementación concreta del método de la superclase. En este caso toma el fichero de entrada en formato XML plano y lo parsea para generar un formato intermedio en un HashMap

Relationships

Unnamed Generalization	
From	Parser
Quality Score	Good

ParserQTI

Nombre	Valor
Documentación	Esta es la implementación de la superclase para un formato de entrada XML QTI, toda la funcionalidad está en el método protegido getPregunta()

Operations

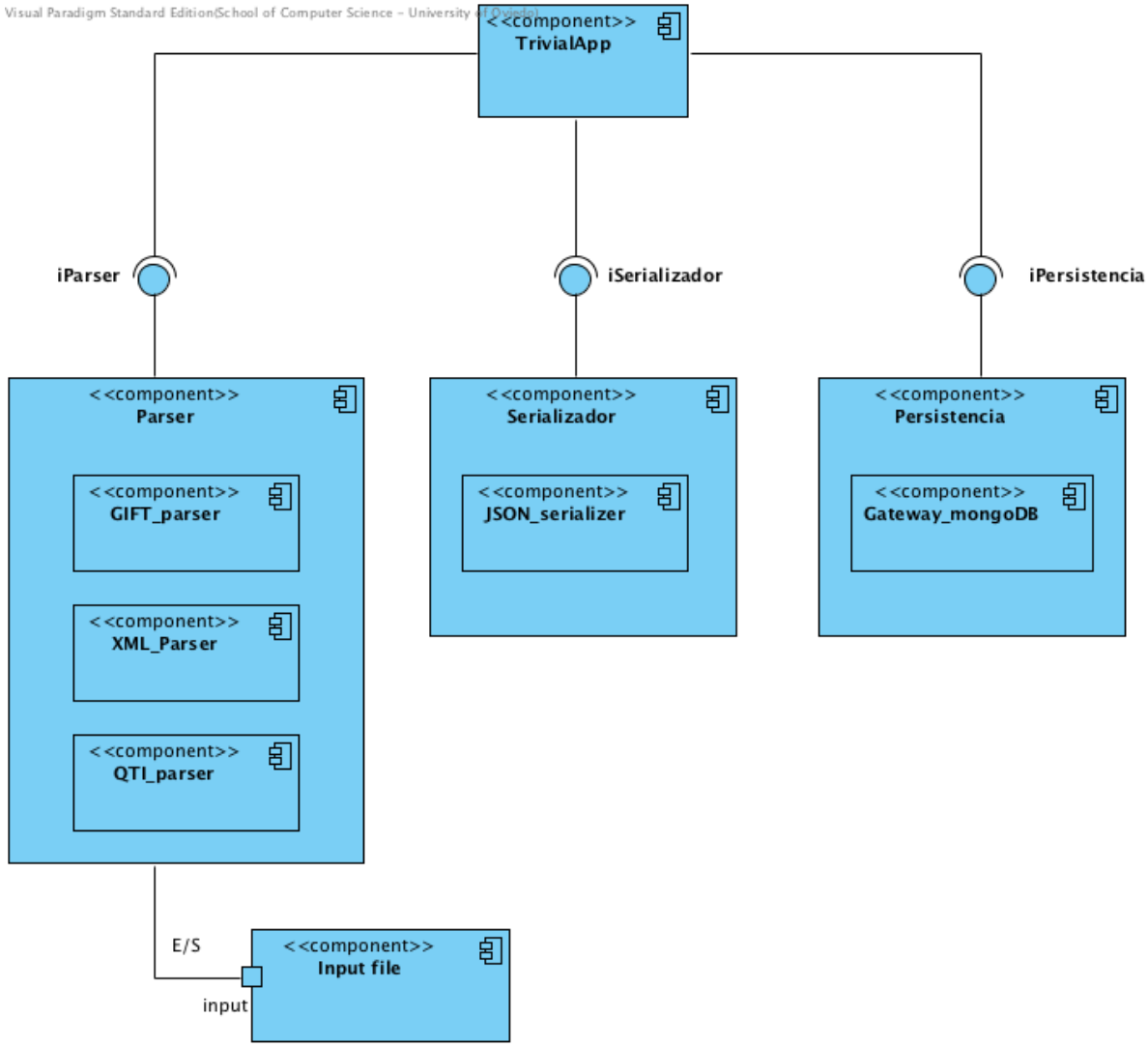
protected getPregunta () : HashMap	
Documentación	Este método es la implementación concreta del método de la superclase. En este caso toma el fichero de entrada en formato XML QTI y lo parsea para generar un formato intermedio en un HashMap

Relationships

Unnamed Generalization	
From	Parser

Diagrama de Componentes

Componentes



Resumen

Nombre	Documentación
TrivialApp	Componente desde el que se controla la aplicación. Es el encargado de utilizar el resto de componentes de manera secuencial, suministrándoles todo lo que necesitan para su funcionamiento. Se comunica con el resto de componentes haciendo uso de las interfaces que estos proven
iParser	Interfaz que provee el componente parser para facilitar su uso desde un componente exterior.

iSerializador	Interfaz que provee el componente Serializador para facilitar su uso desde un componente exterior.
iPersistencia	Interfaz que provee el componente de persistencia para facilitar su uso desde un componente exterior.
Parser	Componente de parseamiento de la aplicación. Encargado de proveer una interfaz común de todos los tipos de parsers disponibles.
Serializador	Componente serializador de la aplicación. Encargado de proveer una interfaz común de todos los tipos de serializadores de la aplicación.
Persistencia	Componente encargado de la persistencia de los datos. Provee una interfaz única para la comunicación con otros componentes de la aplicación, con el objetivo de que el cambio de SGBD no afecte al resto de componentes.
GIFT_parser	Parser para ficheros con formato GIFT
JSON_serializer	Componente de serialización a JSON. Genera un fichero de salida en el que los datos de nuestro programa están en el formato JSON.
Gateway_mongoDB	Componente encargado de proveernos de las acciones necesarias para almacenar los datos de la aplicación en la base de datos MongoDB.
XML_Parser	Parser para ficheros con formato XML
QTI_parser	Parser para ficheros con formato QTI
Input file	Entrada de ficheros GIFT, QTI o XML
input	

Detalles

TrivialApp

Nombre	Valor
Documentación	Componente desde el que se controla la aplicación. Es el encargado de utilizar el resto de componentes de manera secuencial, suministrándoles todo lo que necesitan para su funcionamiento. Se comunica con el resto de componentes haciendo uso de las interfaces que estos proven
Visibilidad	public

Relationships

Unnamed Uso	
To	iParser

Unnamed Uso	
To	iSerializador

Unnamed Uso	
To	iPersistencia

iParser

Nombre	Valor
Documentación	Interfaz que provee el componente parser para facilitar su uso desde un componente exterior.
Visibilidad	public
Estereotipos	Interface

Relationships

Unnamed Realization	
To	Parser
Visibilidad	Unspecified

iSerializador

Nombre	Valor
Documentación	Interfaz que provee el componente Serializador para facilitar su uso desde un componente exterior.
Visibilidad	public
Estereotipos	Interface

Relationships

Unnamed Realization	
To	Serializador

Unnamed Uso	
From	TrivialApp

iPersistencia

Nombre	Valor
Documentación	Interfaz que provee el componente de persistencia para facilitar su uso desde un componente exterior.
Visibilidad	public
Estereotipos	Interface

Relationships

Unnamed Realization	
To	Persistencia
Unnamed Uso	
From	TrivialApp

Parser

Nombre	Valor
Documentación	Componente de parseamiento de la aplicación. Encargado de proveer una interfaz común de todos los tipos de parsers disponibles.
Visibilidad	public

Hijos

Nombre	Documentación
iParser	Interfaz que provee el componente parser para facilitar su uso desde un componente exterior.
GIFT_parser	Parser para ficheros con formato GIFT
XML_Parser	Parser para ficheros con formato XML
QTI_parser	Parser para ficheros con formato QTI

Relationships

Unnamed Realization	
From	iParser

Unnamed Asociación		
From	**Nombre**	**Valor**
	End Model Element	input
	Tipo	input

Resident Components

Nombre	Documentación
GIFT_parser	Parser para ficheros con formato GIFT
XML_Parser	Parser para ficheros con formato XML
QTI_parser	Parser para ficheros con formato QTI

Serializador

Nombre	Valor
Documentación	Componente serializador de la aplicación. Encargado de proveer una interfaz común de todos los tipos de serializadores de la aplicación.
Visibilidad	public

Hijos

Nombre	Documentación
iSerializador	Interfaz que provee el componente Serializador para facilitar su uso desde un componente exterior.
JSON_serializer	Componente de serialización a JSON. Genera un fichero de salida en el que los datos de nuestro programa están en el formato JSON.

Relationships

Unnamed Realization	
From	iSerializador

Resident Components

Nombre	Documentación
JSON_serializer	Componente de serialización a JSON. Genera un fichero de salida en el que los datos de nuestro programa están en el formato JSON.

Persistencia

Nombre	Valor
Documentación	Componente encargado de la persistencia de los datos. Provee una interfaz única para la comunicación con otros componentes de la aplicación, con el objetivo de que el cambio de SGBD no afecte al resto de componentes.
Visibilidad	public

Hijos

Nombre	Documentación
iPersistencia	Interfaz que provee el componente de persistencia para facilitar su uso desde un componente exterior.
Gateway_mongoDB	Componente encargado de proveernos de las acciones necesarias para almacenar los datos de la aplicación en la base de datos MongoDB.

Relationships

Unnamed Realization	
From	iPersistencia

Resident Components

Nombre	Documentación
Gateway_mongoDB	Componente encargado de proveernos de las acciones necesarias para almacenar los datos de la aplicación en la base de datos MongoDB.

GIFT_parser

Nombre	Valor
Documentación	Parser para ficheros con formato GIFT
Visibilidad	public

JSON_serializer

Nombre	Valor
Documentación	Componente de serialización a JSON. Genera un fichero de salida en el que los datos de nuestro programa están en el formato JSON.
Visibilidad	public

Gateway_mongoDB

Nombre	Valor
Documentación	Componente encargado de proveernos de las acciones necesarias para almacenar los datos de la aplicación en la base de datos MongoDB.
Visibilidad	public

XML_Parser

Nombre	Valor
Documentación	Parser para ficheros con formato XML
Visibilidad	public

QTI_parser

Nombre	Valor
Documentación	`Parser para ficheros con formato QTI`
Visibilidad	public

Input file

Nombre	Valor
Documentación	`Entrada de ficheros GIFT, QTI o XML`
Visibilidad	public

Hijos

Nombre	Documentación
input	

Ports

Nombre	Documentación
input	

input

Nombre	Valor
Servicio	true
Comportamiento	false
Conjugated	false
Derivado	false
Is ID	false
Derived Union	false
Aggregation	Unspecified
Multiplicidad	Unspecified
Read Only	false
Static	false

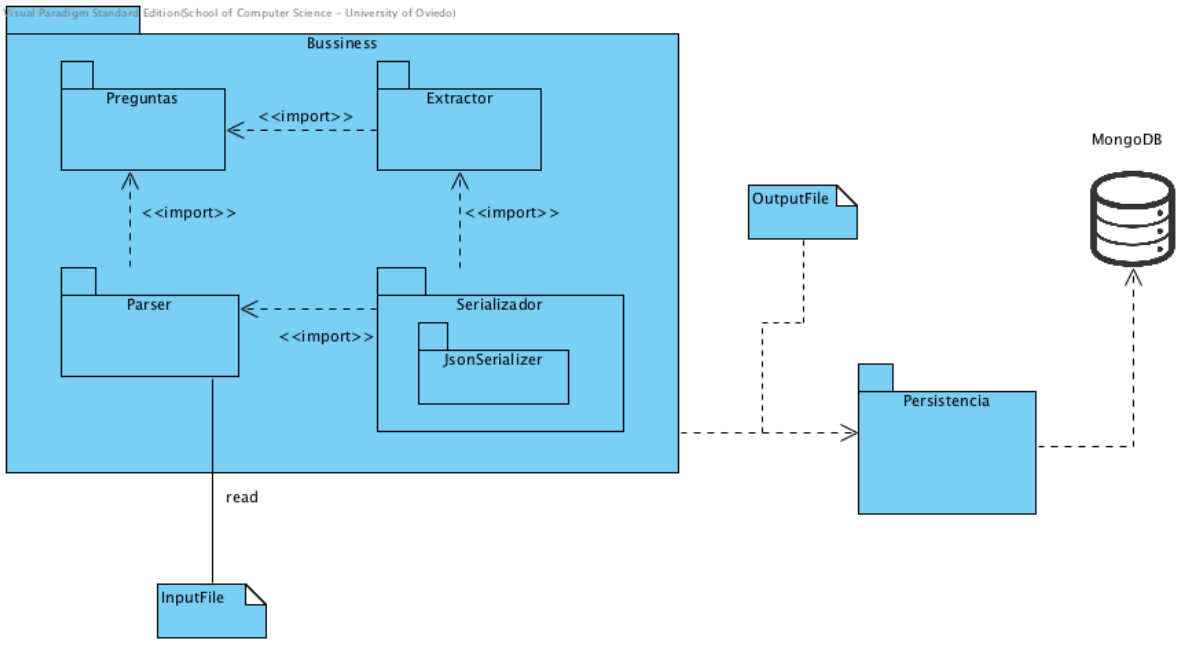
Visibilidad	public

Relationships

Unnamed Asociación		
To (E/S)	**Nombre**	**Valor**
	End Model Element	Parser
	Tipo	Parser

Diagrama de paquetes

Paquetes



Resumen

Nombre	Documentación
Bussiness	Paquete que contiene toda la lógica de nuestra aplicación.
Preguntas	Representación interna que usa la aplicación. Es el tipo de objeto creado a partir de los datos leídos por uno de los parsers disponibles al leer un fichero de entrada.
Extractor	Paquete que actúa a modo de clase controladora de la aplicación. Es en este paquete en el que se

	presentan los modos de interacción disponibles al usuario del sistema.
MongoDB	Base de datos de nuestra aplicación, MongoDB
Parser	Paquete parser: Engloba a todos los tipos de parsers que soporta la aplicación, habiendo un tipo de parser por cada formato de fichero de entrada aceptado.
Serializador	
JsonSerializer	Paquete en el que se lleva a cabo la serialización al formato JSON a partir de el formato de representación interna que utiliza la aplicación.
Persistencia	Modulo encargado de la persistencia de los datos a la base de datos.
N/A	OutputFile
N/A	InputFile

Detalles

Bussiness

Nombre	Valor
Documentación	Paquete que contiene toda la lógica de nuestra aplicación.
Visibilidad	public

Hijos

Nombre	Documentación
Preguntas	Representación interna que usa la aplicación. Es el tipo de objeto creado a partir de los datos leídos por uno de los parsers disponibles al leer un fichero de entrada.
Extractor	Paquete que actúa a modo de clase controladora de la aplicación. Es en este paquete en el que se presentan los modos de interacción disponibles al usuario del sistema.
Parser	Paquete parser: Engloba a todos los tipos de parsers que soporta la aplicación, habiendo un tipo de parser por cada formato de fichero de entrada aceptado.
Serializador	

Relationships

Unnamed Dependency	
To	Persistencia

Preguntas

Nombre	Valor
Documentación	Representación interna que usa la aplicación. Es el tipo de objeto creado a partir de los datos leídos por uno de los parsers disponibles al leer un fichero de entrada.

Relationships

Unnamed Importar	
From	Parser
Estereotipos	import

Unnamed Importar	
From	Extractor
Estereotipos	import

Extractor

Nombre	Valor
Documentación	Paquete que actúa a modo de clase controladora de la aplicación. Es en este paquete en el que se presentan los modos de interacción disponibles al usuario del sistema.
Visibilidad	public

Relationships

Unnamed Importar	
To	Preguntas
Visibilidad	Unspecified
Estereotipos	import

Unnamed Importar	
From	Serializador
Visibilidad	Unspecified
Estereotipos	import

MongoDB

Nombre	Valor
Documentación	Base de datos de nuestra aplicación, MongoDB

Relationships

Unnamed Dependency	
From	Persistencia

Parser

Nombre	Valor
Documentación	Paquete parser: Engloba a todos los tipos de parsers que soporta la aplicación, habiendo un tipo de parser por cada formato de fichero de entrada aceptado.
Visibilidad	public

Relationships

Unnamed Importar	
To	Preguntas
Estereotipos	import

read : Conector Genérico	
To	N/A
Unnamed Importar	
From	Serializador
Visibilidad	Unspecified
Estereotipos	import

Serializador

Nombre	Valor
Abstracto	false
Visibilidad	public

Hijos

Nombre	Documentación
JsonSerializer	Paquete en el que se lleva a cabo la serialización al formato JSON a partir de el formato de representación interna que utiliza la aplicación.

Relationships

Unnamed Importar	
To	Parser
Estereotipos	import

Unnamed Importar	
To	Extractor
Estereotipos	import

JsonSerializer

Nombre	Valor
Documentación	Paquete en el que se lleva a cabo la serialización al formato JSON a partir de el formato de representación interna que utiliza la aplicación.
Visibilidad	public

Persistencia

Nombre	Valor
Documentación	Modulo encargado de la persistencia de los datos a la base de datos.
Visibilidad	public

Relationships

Unnamed Dependency	
To	MongoDB
Unnamed Dependency	
From	Bussiness

N/A

Nombre	Valor
Documentación	OutputFile

Comments

Documentación	Fichero de salida que se genera tras haber procesado los datos en la lógica. Este fichero tiene formato JSON y será enviado al modulo de persistencia para que se encargue de guardar dichos datos en la base de datos

N/A

Nombre	Valor
Documentación	InputFile

Relationships

read : Conector Genérico	
From	Parser

Comments

Documentación	Puerta de entrada de los datos a la aplicación. InputFile puede ser un fichero de tipo Gift, Qti o XML. Estos ficheros contendrán los datos de preguntas y respuestas que mas tarde serán usados durante el juego

MANUAL

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>
```