Trabajo de Fin de Grado

Herramienta para el análisis de resultados de las encuestas de opinión de los estudiantes de la EPS

ERNESTO YANAC HUERTAS

Tutor
Carlos Guerrero Tomé

Escuela Politécnica Superior
Universidad de las Islas Baleares
Palma, 11 de septiembre del 2018

Contenido

Acrónimos

- **CSV**
  Comma-**S**eparated **V**alues
- **JVM**
  Java **V**irtual **M**achine
- **JRE**
  Java **R**untime **E**nvironment
- **SDK**
  Software **D**evelopment **K**it
- **API**
  Application **P**rogramming **I**nterface
- **JXL**
  Java E**x**cel **L**ibrary
- **MVC**
  Model-**V**iew-**C**ontroller
- **EPS**
  Escola **P**olitècnica **S**uperior
- **XML**
  E**x**tensible **M**arkup **L**anguage

RESUMEN

El propósito de este trabajo de fin de grado es el de realizar una aplicación de ordenador, basada en Java, que permita generar reportes por asignatura y profesor en base a encuestas realizadas a los alumnos de la Escuela Politécnica Superior.

Los usuarios de la aplicación ingresarán los documentos y datos necesarios para generar los reportes: las encuestas, datos de las asignaturas y preguntas específicas sobre el formato de las encuestas. Según esta información, y una vez comprobada que ésta sea correcta, la aplicación generará los reportes.

Los reportes son documentos Word que reúnen e interpretan las respuestas dadas por los alumnos en las encuestas.
Las respuestas textuales a preguntas de texto libre se muestran de forma agrupada, mientras que las respuestas numéricas, calificaciones dentro de un intervalo de valores enteros, son procesadas para obtener valores estadísticos que serán mostrados en los reportes.

Para poder realizar este proyecto sea ha usado Java 8.0 y bibliotecas de terceros con el fin de poder generar documentos Word, interactuar con documentos Excel y leer CSV's.

“A mis padres y hermanos por hacer de mi quien soy ahora”

CAPÍTULO 1

INTRODUCCIÓN

Este capítulo tiene como propósito explicar la razón del trabajo de fin de grado y sus objetivos.

1.1 Motivación:

La gran cantidad de tiempo empleada por los jefes de estudio para crear reportes en base a las encuestas realizadas por los alumnos de la EPS sobre las asignaturas que cursan y los profesores de las mismas, sumado a que el proceso de generar dichos reportes es bastante mecánico llevó a la conclusión que la mejor solución era automatizar todo el proceso y así aligerar significativamente el trabajo de los jefes de estudio.

1.2 Estado actual:

El proceso consiste en digitalizar las respuestas que se recoge de las encuestas y crear los reportes manualmente en base a esas respuestas. El trabajo se puede dividir en:

- Pasar la información de las encuestas a documentos Excel
- Calcular los datos estadísticos con ayuda de fórmulas, en los documentos Excel
- Por cada asignatura y profesor crear un documento Word
    - Copiar las preguntas de texto libre y sus respuestas en el documento Word
    - Copiar las preguntas numéricas y los valores estadísticos, calculados en el documento Excel, al documento Word

1.3 Objetivo:

Minimizar considerablemente el trabajo de los jefes de estudio. Dejar de crear los reportes manualmente y en su lugar automatizar todo el proceso. Limitarlo a sólo un par de minutos para introducir datos en la aplicación y que este proceso sea intuitivo y rápido.

Por otra parte, como objetivo personal, aplicar los algoritmos y tecnologías aprendidas en la carrera, buenas prácticas y experiencia laboral adquirida en el ámbito profesional.

CAPÍTULO 2

MARCO TECNOLÓGICO

En este capítulo se explica el lenguaje usado para el desarrollo de la aplicación, las librerías y tecnologías de las que se han hecho uso.

2.1 Java:

Lenguaje basado en clases, orientado a objetos, y específicamente diseñado para tener la menor cantidad posible de dependencias de implementación. El principal objetivo de Java es que el código, una vez compilado, se puede ejecutar en todas las plataformas que admitan Java sin la necesidad de recompilarlo. Las aplicaciones Java generalmente se compilan en bytecode que se puede ejecutar en cualquier máquina virtual Java (**JVM**) independientemente de la arquitectura de la computadora. El lenguaje deriva gran parte de su sintaxis de **C** y **C ++**, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Se puede encontrar más información sobre Java y sus beneficios en [1]

El proyecto está basado en Java ya que este lenguaje tiene como principal característica la portabilidad siendo solo necesario tener *Java Runtime Environment* (**JRE**) para poder ser ejecutada

2.2 Librerías:

2.2.1 Docx4j:

Es una biblioteca Java de código abierto para crear y manipular archivos Microsoft Open **XML** (Word docx, Powerpoint pptx y Excel xlsx).
Es similar al SDK OpenXML de Microsoft, pero para Java. Docx4j usa **JAXB** para crear la representación de objetos en memoria.

Una dependencia de Docx4j es **xmlgraphics,** una librería que permite manipular imágenes dentro de documentos. [2]

Dentro del proyecto **Docx4j** es usada para generar los documentos Word **docx** y **Xmlgraphics** para insertar el logo de la UIB como inicio de página.

2.2.2 JXL:

Es la **API** más utilizada para leer, escribir, crear y modificar hojas en un libro de Excel (.xls) en tiempo de ejecución. [3]

Figura 2.1: JXL

En el proyecto la API **JXL 2.6.9** es usada para leer las encuestas que luego serán procesadas para obtener los reportes. Permite gestionar cada hoja de un documento Excel como si fueran tablas con columnas y filas.

2.2.3 OpenCSV:

Es una biblioteca para gestionar CSV's con Java. Opencsv admite las interacciones más comunes con documentos CSV: [4]

- Números arbitrarios de valores por línea.
- Ignorando comas en elementos con comillas.
- Manejo de líneas con más de un fin de línea
- Separador configurable y caracteres de comillas.

En el proyecto se usa para leer los datos de cada asignatura. Estos datos vienen dados en un CSV donde se especifica el nombre de cada asignatura, el estudio al que pertenece, los alumnos matriculados, etc.

CAPÍTULO 3

REQUISITOS DE LA APLICACIÓN

3.1 Análisis de requisitos:

En este capítulo se analizará la aplicación y se extraerá los requisitos, para así poder observar y documentar las necesidades funcionales que satisfaga el proyecto.

3.1.1 Requisitos del usuario:

Los requisitos de usuario especifican una necesidad documentada sobre el contenido, la forma o la funcionalidad de un producto o servicio. Estos se encuentran escritos desde el punto de vista de los usuarios finales, es decir, no contienen términos técnicos, y se encuentran generalmente escritos de forma narrada.

Requisitos:

- **RU01**
  El usuario ha de poder ingresar las encuestas
- **RU02**
  El usuario ha de poder ingresar el CSV con los datos de las asignaturas
- **RU03**
  El usuario ha de poder ingresar la ruta destino donde se guardarán los reportes
- **RU04**
  El usuario ha de poder recibir un reporte por cada profesor dentro de una asignatura
- **RU05**
  El usuario debe poder visualizar en los reportes, un extracto de todas las preguntas de texto libre
- **RU06**
  El usuario debe poder visualizar en los reportes, valores estadísticos de las preguntas numéricas

3.1.2 Requisitos de sistema

Los requisitos de sistema son especificaciones detalladas que describen las funciones que ha de realizar el sistema. Estos se encuentran escritos desde el punto de vista técnico, es decir, contienen palabras y descripciones técnicas del ámbito.

3.1.2.1 Requisitos funcionales:

- **RS01**
  El sistema ha de permitir ingresar la ruta a los reportes
- **RS02**
  El sistema ha de permitir ingresa la ruta a los CSV's con los datos de las asignaturas
- **RS03**
  El sistema ha de permitir ingresar la ruta de destino donde se guardarán los reportes generados
- **RS04**
  El sistema ha de generar un reporte por cada asignatura y profesor
- **RS05**
  El sistema ha de poder obtener los datos de cada asignatura a partir del CSV
- **RS06**
  El sistema ha de poder calcular valores estadísticos a partir de las respuestas numéricas de las encuestas
- **RS07**
  El sistema ha de mostrar en los reportes, un extracto de todas las preguntas de texto libre
- **RS08**
  El sistema ha de mostrar en los reportes, valores estadísticos de las preguntas numéricas

3.1.2.2 Requisitos de interfaz

- **RS09**
  El sistema se podrá ejecutar sobre cualquier sistema operativo que tenga JAVA JRE instalado

3.1.2.3 Requisitos de recursos

- **RS10**
  El sistema requiere que el documento con los datos de las asignaturas tenga el formato CSV.
- **RS11**
  El sistema requiere que la encuesta sea un fichero con formato **.xls**

CAPÍTULO 4

Desarrollo de la aplicación:

Este capítulo tiene como objetivo explicar el desarrollo del proyecto: Qué problemas se han encontrado y qué soluciones se han dado a lo largo del desarrollo del proyecto.

4.1 Aplicación Gestor de Encuestas EPS:

La aplicación tiene como objetivo dar una solución rápida y automática a la gestión y creación de reportes basados en las encuestas realizadas en la EPS.

4.1.1 Mockup de la aplicación

El mockup refleja elecciones de diseño, íconos, navegación e interfaz de la aplicación. Además de responder las preguntas visuales importantes, el mockup ayuda a encontrar errores u omisiones en la entrada/salida de datos antes de encontrarse con ellos en el código de la aplicación.

En cuanto al diseño de la interfaz de la aplicación, se pueden distinguir tres maquetas:

4.1.1.1 Landing

En el diseño se puede ver como el landing se divide en 3 secciones: El título, los inputs para las rutas de los documentos y los botones para aceptar o cancelar [*Anexo E*].

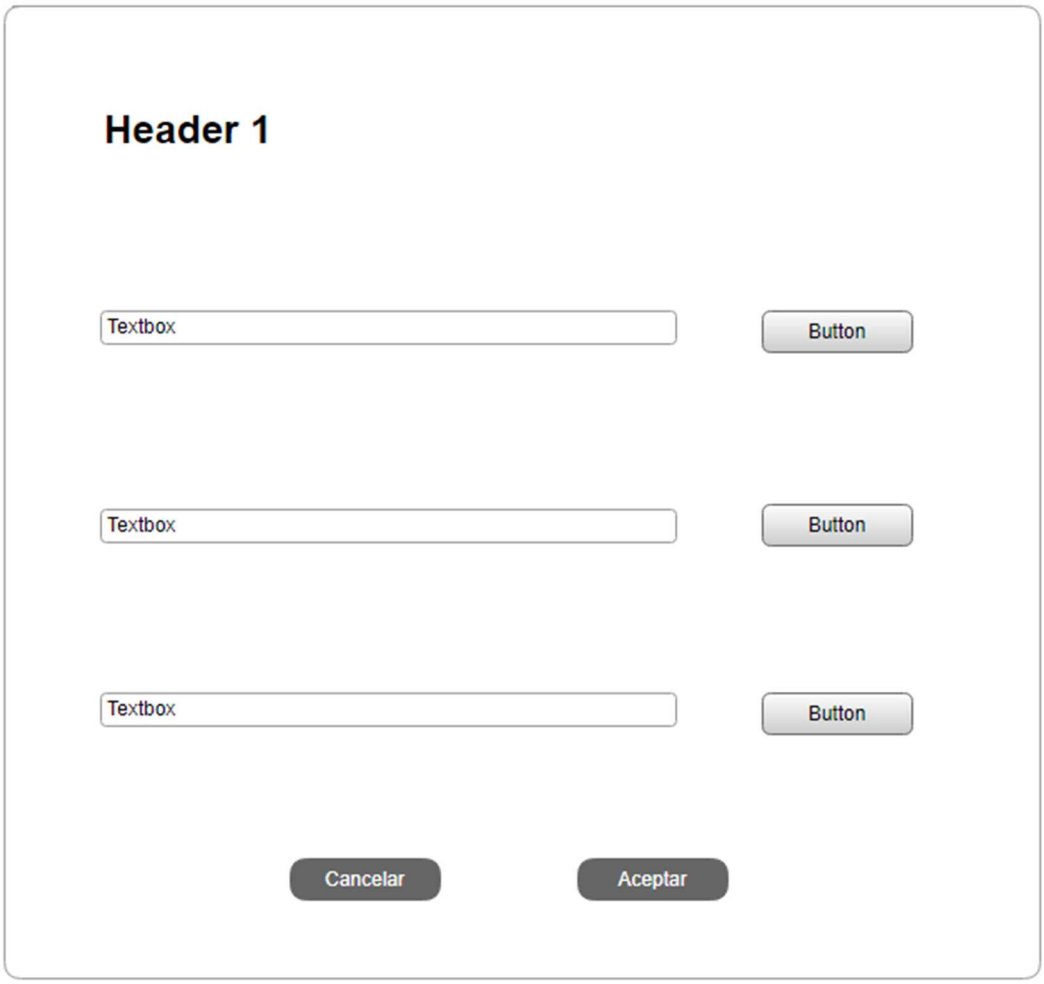

Figura 4.1: Landing

Las rutas pueden ser escritas manualmente o se puede usar el botón ubicado al lado derecho de cada input para buscar el documento

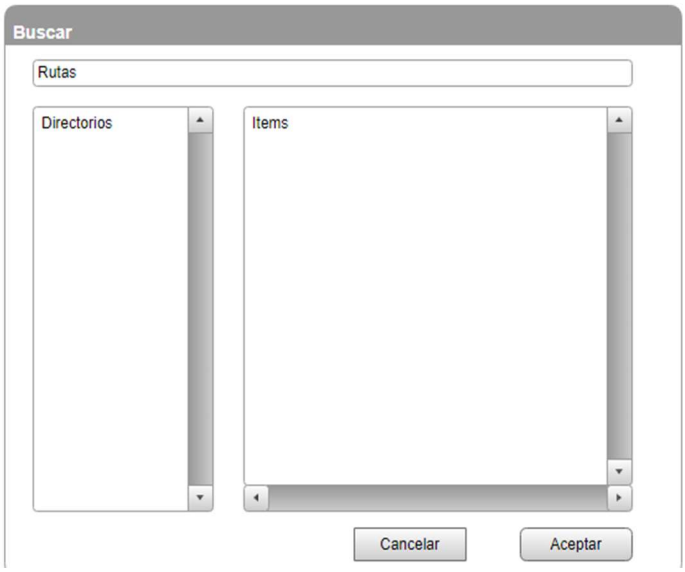

Figura 4.2: Buscar archivos

4.1.1.2 Wizard:

En el wizard se diferencian dos diseños:
El primero de ellos tiene como objetivo permitir que el usuario pueda seleccionar las preguntas y respuestas desde una vista muy parecida al Excel original [*Anexo F*].
El diseño se repite para los siguientes 5 pasos del wizard.

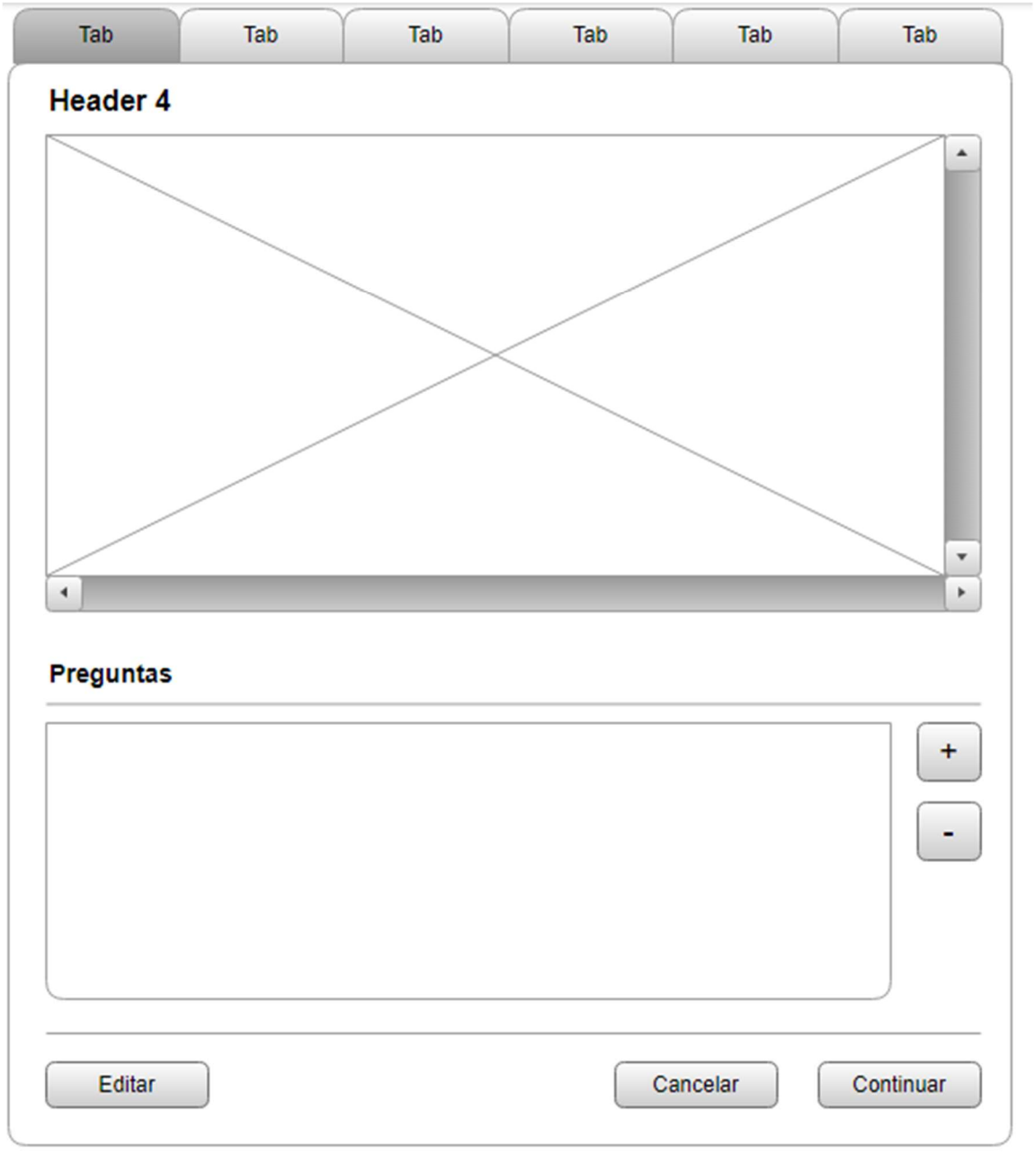

Figura 4.3: Diseño de wizard - 1

En el bloque superior se muestran los datos del documento Excel con las preguntas y respuestas de las encuestas. Los datos se muestran en formato de tablas, indicando las columnas y filas. El usuario se puede desplazar por la tabla usando el cursor o las teclas direccionales.

En el bloque inferior se irán mostrando los datos que serán ingresados a la aplicación. Los datos pueden ser ingresados manualmente o a través de la interfaz gráfica que representa los datos del Excel, ubicada en el bloque superior.
Los datos a ingresar pueden ser el contenido de una celda o el nombre de alguna columna, esto dependerá de la pregunta.

El segundo diseño del wizard es usado en el apartado Resumen donde se muestran los datos que serán procesados por la aplicación: Las preguntas y las respuestas ingresadas, separadas por el tipo de pregunta, numéricas o textuales.
Una vez se valida la información que muestra el resumen se comienza a generar los reportes

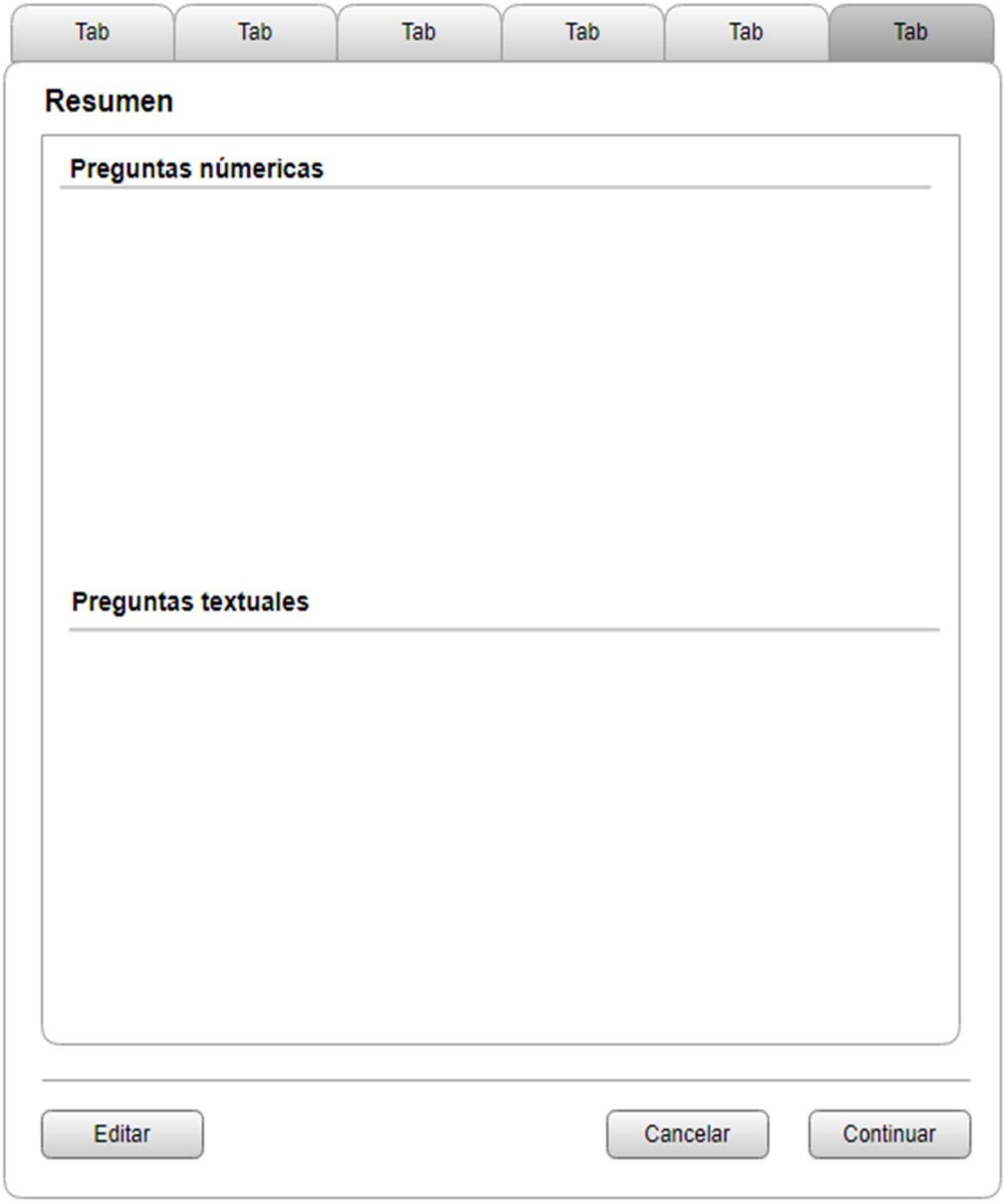

Figura 4.4: Diseño de wizard - 2

4.1.2 La estructura de la aplicación:

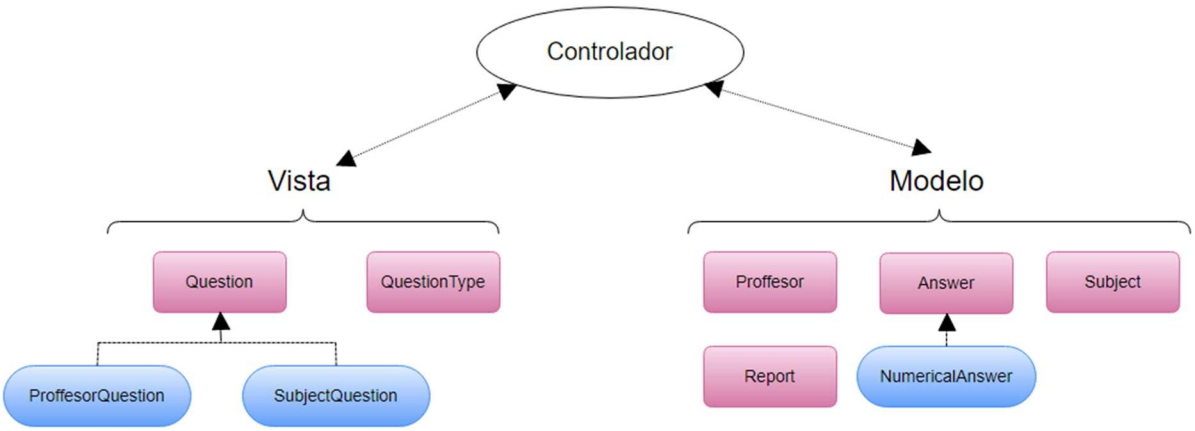

Figura 4.5: Estructura de la aplicación

La lógica de la aplicación sigue el diseño **Modelo-Vista-Controlador (MVC)**. Es un patrón de arquitectura de software, que separa los datos de una aplicación (Modelo), de su representación (Vista) y el módulo encargado de gestionar los eventos y las comunicaciones (Controlador). Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, por un lado, se definen componentes para la representación de la información, y por otro lado para la interacción del usuario. Las principales características de este patrón de arquitectura del software es que facilita la **reutilización de código** y la **separación de conceptos**, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. [5,6,7]

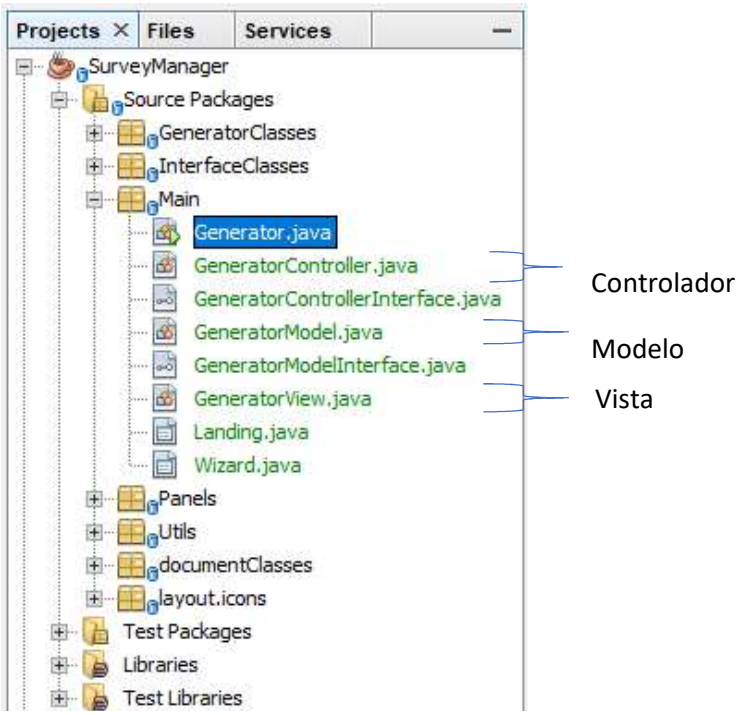

Figura 4.6: Clases del MVC

4.1.2.1 El Modelo:

Es la representación de la información con la que trabajará la aplicación, contiene únicamente los datos puros de aplicación; no contiene lógica que describe cómo pueden presentarse los datos a un usuario. Envía a la **Vista** la información que se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al **Modelo** a través del **Controlador** [*Anexo C*].

```
import GeneratorClasses.Proffesor;
import GeneratorClasses.Report;
import InterfaceClasses.Question;
import InterfaceClasses.QuestionType;
import InterfaceClasses.Questions;
import java.util.ArrayList;
import java.util.List;

/**
 * @author ernesto
 */
public class GeneratorModel implements GeneratorModelInterface{
    private String  excelPath;
    // path to excel file wich contains the surveys answers
    private String  csvPath;
    // path to CSV file wich contains the subjects information
    private String  reportPath;
    // path to the file where generatedwich contains the surveys answers
    private Questions professorsName;
    // array of professor's names
    private Questions subjectNumericalData      = new Questions(QuestionType.category.SUBJECT, QuestionType.type.NUMERICAL);
    // numerical type question about subjects, and their answers
    private Questions subjectTextualData        = new Questions(QuestionType.category.SUBJECT, QuestionType.type.TEXTUAL);
    // textual type question about subjects, and their answers
    private Questions professorNumericalData    = new Questions(QuestionType.category.PROFFESOR, QuestionType.type.NUMERICAL);
    // numerical type question about professors, and their answers
    private Questions professorTextualData      = new Questions(QuestionType.category.PROFFESOR, QuestionType.type.TEXTUAL);
    // textual type question about professors, and their answers
    private List<Proffesor> professors          = new ArrayList<Proffesor>();
    // list of professor's objects
    private ArrayList<Report> statisticReports      = new ArrayList<Report>();
    // list of reports
```

Figura 4.7: El Modelo

Clases

- **Answers:**
  Conjunto de datos que representa una pregunta, su tipo y una lista con las respuestas a esa pregunta [*Anexo K*].
  - **NumericalAnswers:** Contiene las mismas propiedades que **Answers** pero permite tratar las respuestas para poder obtener datos estadísticos [*Anexo L*].

- **Professor**:
  Representa a un profesor de alguna asignatura por sus nombres y apellidos, y una lista de **Answers.** Un profesor podría estar relacionado a más de una asignatura [*Anexo G*].

- **Subject:**
  Una de las asignaturas, se identifica por un código, un grupo, estudio al que pertenece, los alumnos matriculados y un listado de **Answers.** Una asignatura puede tener varios profesores que la impartan [*Anexo H*].

- **Report:**
  El reporte final generado por la aplicación. Se crea un reporte por cada combinación Profesor-Asignatura. Se crearán tantos reportes como profesores haya en una asignatura [*Anexo J*].

Modelo de datos:
Si bien la aplicación no necesita de la implementación de una base de datos, la estructura del MODELO sigue una lógica que puede ser fácilmente representada con un diagrama UML según las siguientes condiciones:

- Un Profesor puede dar clases en más de una asignatura y a su vez una asignatura puede ser impartida por más de un profesor.
- Las respuestas pueden pertenecer a una asignatura o un profesor que imparte cierta asignatura.
- Las respuestas pueden ser textuales o numéricas.

El reporte no forma parte de esta representación de la teórica base de datos ya que un reporte no es más que un compendio de datos de las clases Proffesor, Subject y Answer.

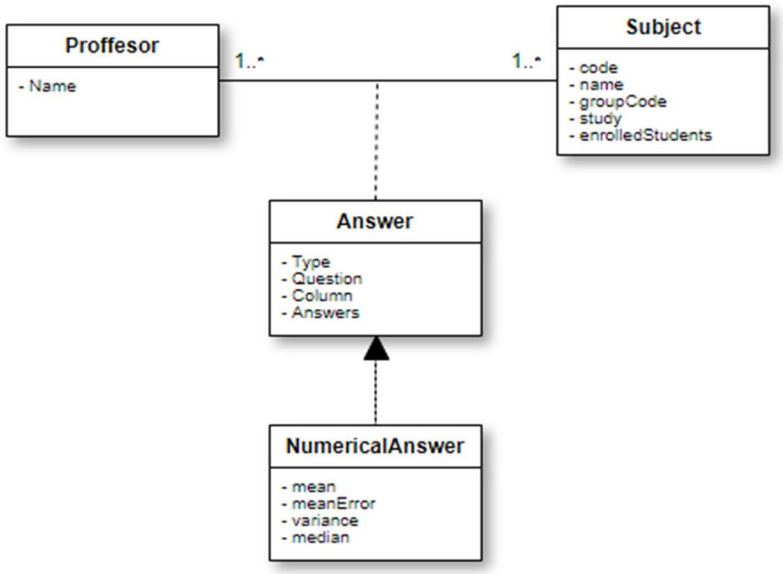

Figura 4.8: Modelo de datos

4.1.2.2 La vista:

Presenta al usuario los datos del modelo. La vista sabe cómo acceder a los datos del modelo, pero no sabe el significado de estos datos ni lo que el usuario puede hacer para manipularlos.

Con el fin de poder recibir la información que necesita la aplicación para generar los reportes, la vista está compuesta por las siguientes clases [*Anexo B*]:

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GeneratorView implements ItemListener {
    JPanel cards;
    // each wizard's tab
    Landing landing;
    // first aplication window, the file inputs are here
    Wizard wizard;
    // set of tabs which help the inputs management
    JFrame frame;
    final static String LANDING = "landing";
    final static String WIZARD = "wizard";
    static GeneratorControllerInterface gci;
    GeneratorModelInterface gmi;


    public GeneratorView(GeneratorControllerInterface gci, GeneratorModelInterface gmi){
        this.gci    = gci;
        this.gmi    = gmi;
    }

    public void addComponentToPane(Container pane) {
        landing = new Landing(gci);
        wizard = new Wizard(gci);

        //Create the panel that contains the "cards".
        cards = new JPanel(new CardLayout());
        cards.add(landing, LANDING);
        cards.add(wizard, WIZARD);

        pane.add(cards, BorderLayout.CENTER);
    }
```

Figura 4.9: La Vista

- **Question:**
  El dato que se solicita ingresar al usuario. El dato ingresado es una cadena de caracteres; si la pregunta es de tipo numérica, el dato será convertido a un valor numérico; de ser textual, el dato seguirá siendo una cadena de texto. Por cada **step** del wizard puede haber más de un Question.
- **ProffesorQuestion:**
  Es una clase que hereda los atributos de Question, además tiene una lista de cadena de caracteres que referencian a uno o más profesores ya que una pregunta puede estar dirigida a calificar a más de un profesor.
- **SubjectQuestion:**
  Es una clase que hereda los atributos de Question, además tiene una cadena de caracteres en la que se guarda el identificador de la columna en la que se encuentran las respuestas de los encuestados.
- **QuestionType:**
  Un dato ingresado, para la Vista, siempre será una cadena de caracteres sin embargo en el Controlador se tratará de manera diferente si el dato es de tipo numérico o de texto libre, es por esto por lo que es necesario saber de qué tipo es el dato.

4.1.2.3 El Controlador

Está entre la vista y el modelo. Escucha los eventos desencadenados por la vista y ejecuta la reacción apropiada a estos eventos. En la mayoría de los casos, la reacción es llamar a un método del modelo. Toda la lógica para representar o guardar los datos del modelo en o hacia la vista, se encuentran en el controlador [*Anexo C*].

El controlador se encargará de gestionar todos los datos ingresados en la vista, tratarlos, modificarlos de ser necesario y guardarlos en el Modelo, para ellos se necesitan clases que hagan más fácil las acciones mencionadas.

```
public class GeneratorController implements GeneratorControllerInterface{

    GeneratorView gview;
    // MVC view
    GeneratorModelInterface gmi;
    // MVC controller

    public GeneratorController(GeneratorModelInterface gmi){
        this.gmi = gmi;
        gview = new GeneratorView(this, gmi);
        gview.createAndShowGUI();
    }
```

Figura 4.10: El Controlador

- **FileUtils**:
  En esta clase se encuentran los métodos que permiten interpretar las rutas, ingresadas por los usuarios, a los documentos que necesita la aplicación.
- **MathUtils:**
  Tiene métodos que permiten obtener los valores estadísticos que se calculan a partir de las respuestas numéricas. A partir de una lista de valores se puede obtener la media, la mediana, la varianza y el error medio.
- **WorldDocument**:
  Usando la librería **docx4j,** esta clase permite gestionar los reportes que son documentos en formato Word. Tiene los métodos que permiten crear un documento Word y construir un reporte insertando párrafos, tablas e imágenes [*Anexo P*].

CAPÍTULO 5

Problemas encontrados durante el desarrollo de la aplicación

5.1 Wizard

Para generar los reportes, la aplicación necesita varios datos sobre las encuestas, para ello es necesario usar varios inputs de información para que el usuario pueda ingresar todos los valores necesarios. Todos estos valores pueden ser agrupados por:

- Preguntas dirigidas a calificar los profesores
- Preguntas dirigidas a calificar las asignaturas
- Preguntas dirigidas a identificar a los profesores, en el documento de las encuestas.

Además de este agrupamiento, los valores ingresados pueden ser tratados luego como de tipo numérico o textual.

Teniendo en cuenta estas dos consideraciones además bajo el objetivo de guiar al usuario durante el proceso de ingresar todos los datos necesarios de manera intuitiva, la mejor opción fue usar un Wizard. [8,9,10]

Figura 5.1: Estructura del Wizard

Un wizard permitirá agrupar las preguntas por a quién o qué está dirigida a calificar y por si es numérica o de texto libre.

Otro beneficio de usar un wizard es que en el último paso se puede mostrar un resumen de todos los datos ya ingresados y permitir al usuario poder confirmar si todos los datos son correctos antes de que la aplicación comience a generar los reportes.

5.2 Cantidad indefinida de inputs:

Un problema hallado en la vista fue que por cada pestaña del wizard, la cantidad de inputs ingresado no estaba definida, en algunos casos podría ser necesario ingresar como mínimo un valor y en otros no hacía falta un mínimo, y la cantidad máxima de valores a ingresar no estaba definido.

Una posible solución era preguntar al usuario cada vez antes de pasar a la siguiente pestaña del wizard, la cantidad de datos que iba a ingresar y a partir de esa cantidad mostrar los inputs necesarios. Pero esta solución obligaba a duplicar las pestañas en el wizard o usar pop ups entre pestaña y pestaña. En ambos casos la labor del usuario se duplica y resta rapidez al proceso.

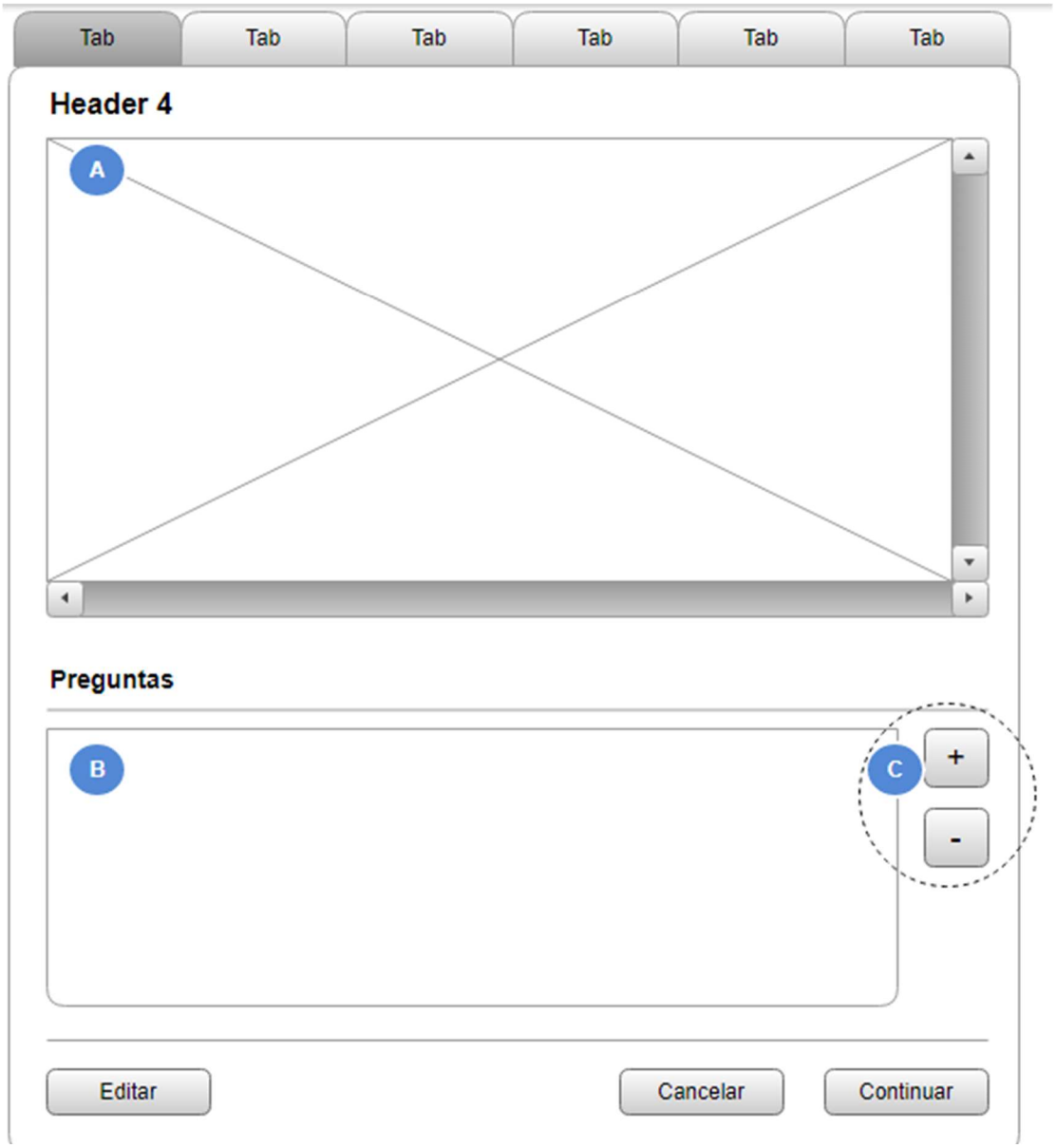

Figura 5.2: Inputs del wizard

Finalmente, la mejor solución fue hacer que la cantidad de inputs a ingresar fuera dinámica permitiendo al usuario poder agregar o quitar inputs, pero respetando la cantidad mínima necesaria de inputs, según cada pestaña del wizard. para ello el usuario cuenta con dos botones **(Zona C)** en cada una de las pestañas, estos botones son los que le permiten agregar o quitar inputs

Los inputs van apareciendo o desapareciendo en la zona **B** que se encuentra al lado izquierdo de los botones de control **+** y **-**

5.3 Gestionar los inputs:

Debido a que para una misma pregunta se pueden necesitar más de un input y sus títulos, además en la zona **B** se puede representar más de una pregunta. Lo mejor es pensar en la zona B como si fuera una tabla, en el proyecto se interpreta como una clase llamada **ModelTable** [*Anexo N*], compuesta de muchas filas donde cada fila a su vez está compuesta de varios objetos, estos objetos pueden ser títulos de inputs o los propios inputs.

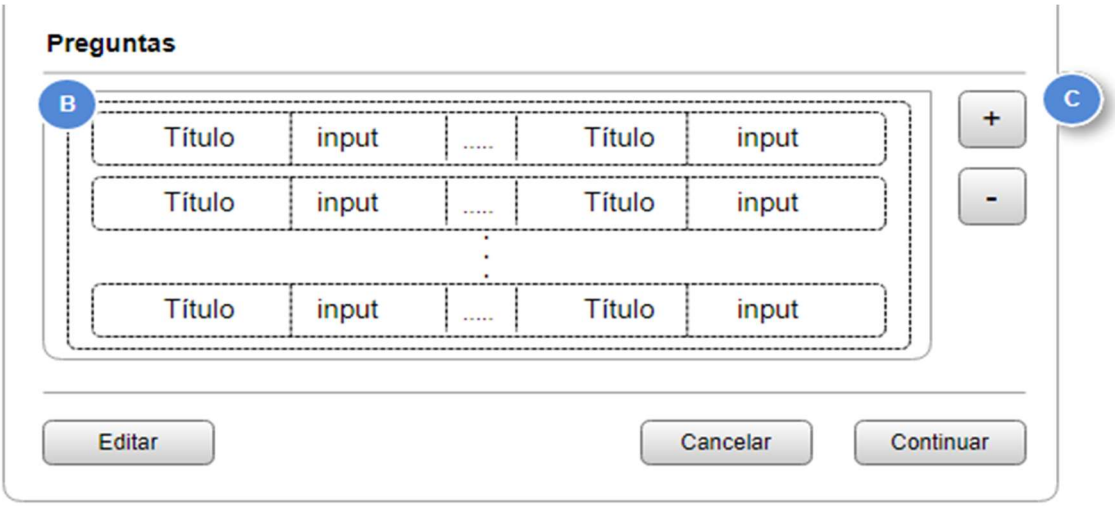

Figura 5.3: Gestión de inputs

Cada una de las filas se gestiona como una clase llamada **RowTable** [*Anexo O*], esta clase es una lista de objetos y cada objeto puede ser un título o un input. Por tanto, **ModelTable** está compuesta por una lista de **RowTable'**s.

Ambas clases **ModelTable** y **RowTable** son clases que ayudan a entender y manipular mejor los valores que se van solicitando al usuario, pero ninguna de estas clases se encarga de mostrar gráficamente los valores (títulos e inputs), de esa labor se encarga la clase **Table** [Anexo N]. **Table** está compuesto por un **ModelTable** y su labor es mostrar en el wizard el contenido actualizado de **ModelTable.**
La lógica es la siguiente:

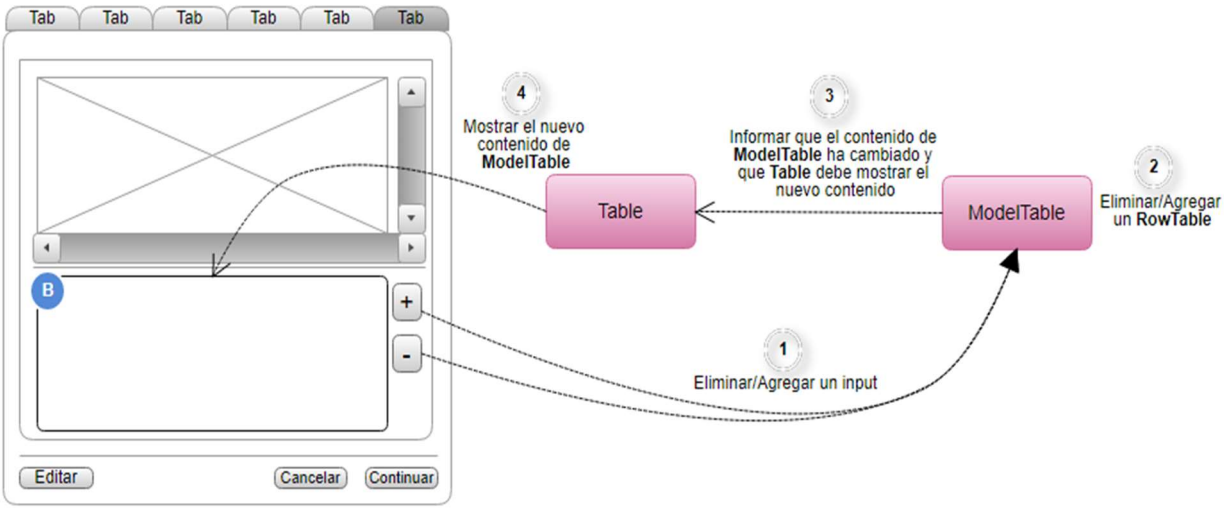

Figura 5.4: Diseño Observer aplicado sobre la lógica de gestión de inputs

La comunicación entre **Table** y **ModelTable** sigue el patrón **Observer** (también conocido como **Listener**). Este patrón es particularmente útil para aplicaciones gráficas. [11,12]

La idea general es que uno o más objetos (Observadores, en nuestro caso **Table**) registran su interés en recibir notificaciones de cambios en otro objeto (Sujeto, para la aplicación sería **ModelTable**).

El Sujeto mantiene así una lista de las referencias a sus observadores. Los observadores a su vez están obligados a implementar unos métodos determinados mediante los cuales el Sujeto es capaz de notificar a sus observadores suscritos los cambios que sufre para que todos ellos tengan la oportunidad de refrescar el contenido representado. De manera que cuando se produce un cambio en el Sujeto, ejecutado, por ejemplo, por alguno de los observadores, el objeto de datos puede recorrer la lista de observadores avisando a cada uno. Este patrón suele utilizarse en los entornos de trabajo de interfaces gráficas orientados a objetos, en los que la forma de capturar los eventos es suscribir *listeners* a los objetos que pueden disparar eventos.

El patrón observador es la clave del patrón de arquitectura Modelo Vista Controlador (MVC)

5.4 Completar los inputs

El usuario tiene que ingresar preguntas, que son textos largos; nombre de profesores, textos menos largos que las preguntas; o bien nombres de columnas que son textos que como máximo tendrán 3 caracteres.
Además, las preguntas y los nombres de los profesores son textos que se encuentran dentro del Excel de las encuestas.

Sabiendo esto se puede facilitar la labor del usuario haciendo que él no tenga que ingresar ningún dato manualmente sino simplemente seleccionando el contenido desde el documento Excel.

Los inputs tienen una longitud inicial, a partir de esa longitud se puede saber si el input está diseñado para recibir un texto largo o un texto corto, si la longitud inicial del input es larga (30 caracteres) entonces se colocará en el input el texto contenido en la celda marcada, si la longitud inicial es corta (3 caracteres) entonces se colocará en el input el nombre de la columna marcada.

CAPÍTULO 6

Diseño de la aplicación

En esta sección se mostrará el contenido de cada pestaña de la aplicación y todas las posibilidades y facilidades que ofrece cada una.

6.1 Pantalla inicial:

Figura 6.1: Pantalla inicial

En esta pantalla inicial se solicita al usuario ingresar las rutas a los documentos que necesita la aplicación para generar los reportes. Una vez ingresadas las rutas de los tres documentos se puede hacer clic sobre el botón **COMENZAR**. Se iniciará un wizard que guiará al usuario en el proceso de ingresar datos de las encuestas según vayan siendo solicitados.
Las rutas pueden ser ingresadas manualmente o se puede hacer uso de los botones ubicados al lado derecho de cada input. Cada botón abre un selector de documentos a excepción del botón destinado a la ruta de los reportes, el cual muestra un selector de carpetas.
Cada selector solo muestra los documentos con la extensión adecuada para cada input. El botón ubicado al lado derecho del input "**Documento Excel con las encuestas**" solo muestra documentos de extensión **xls**, lo mismo sucede con el botón ubicado al lado derecho del input "**CSV de los estudios"**, solo muestra documentos con la extensión **csv.**

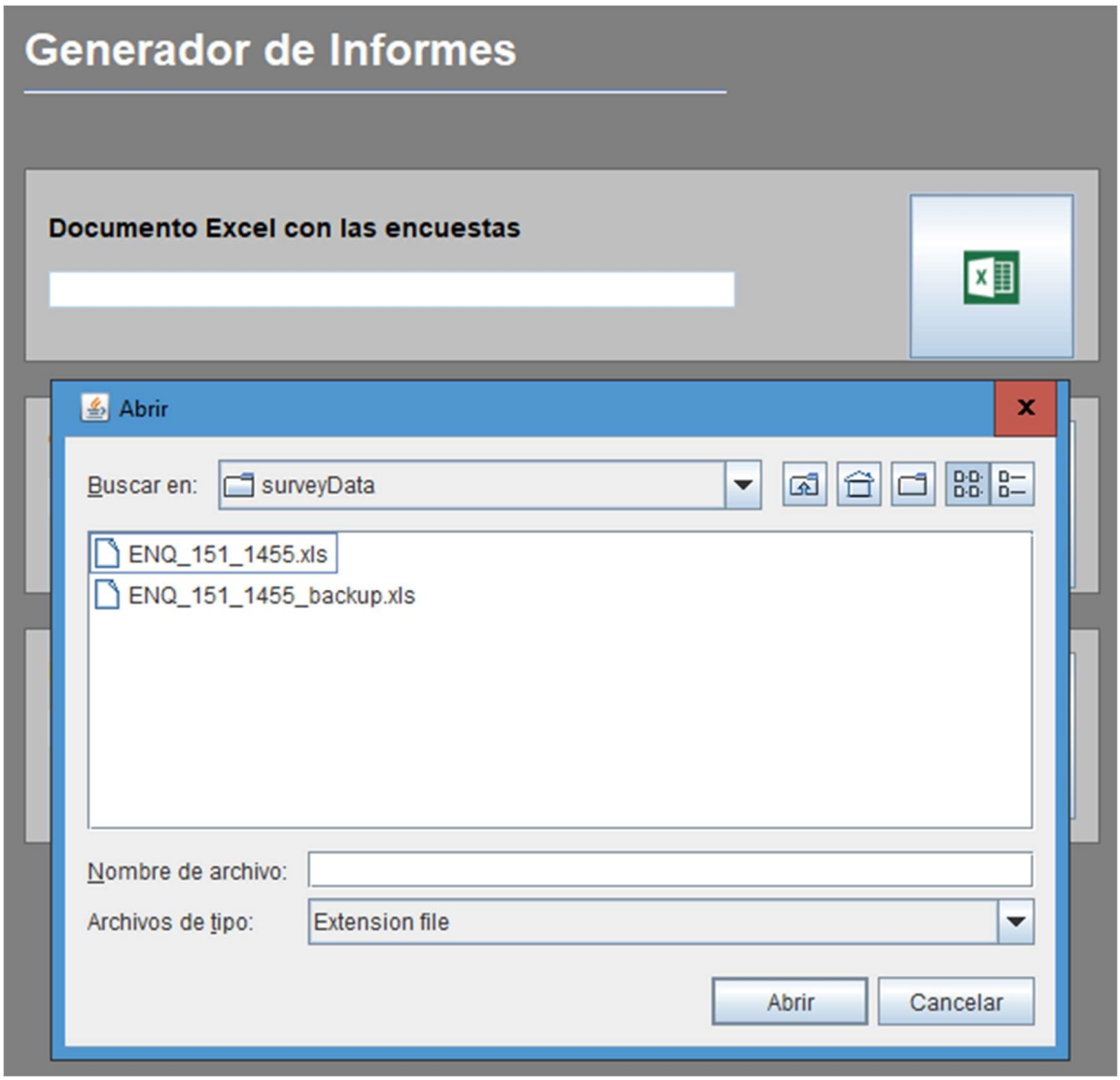

Figura 6.2: Buscar archivos

6.2 Wizard

6.2.1 Preguntas relacionadas a las asignaturas

6.2.1.1 Preguntas Numéricas y de Texto:

El usuario debe ingresar la pregunta en el input de la columna **"pregunta"** y el nombre de la columna en el input "**columna".**

Si hubiera que ingresar más de una pregunta, bastará con agregar más inputs usando el botón **"+"** y el botón **"-"** para eliminar inputs.

Los datos pueden ser ingresados manualmente o pueden ser seleccionados desde el visor Excel ubicado en la zona superior. Los inputs de la columna **"pregunta"** cogerán el contenido de la celda seleccionada mientras que los inputs de la columna **"columna"** el nombre de la columna seleccionada.

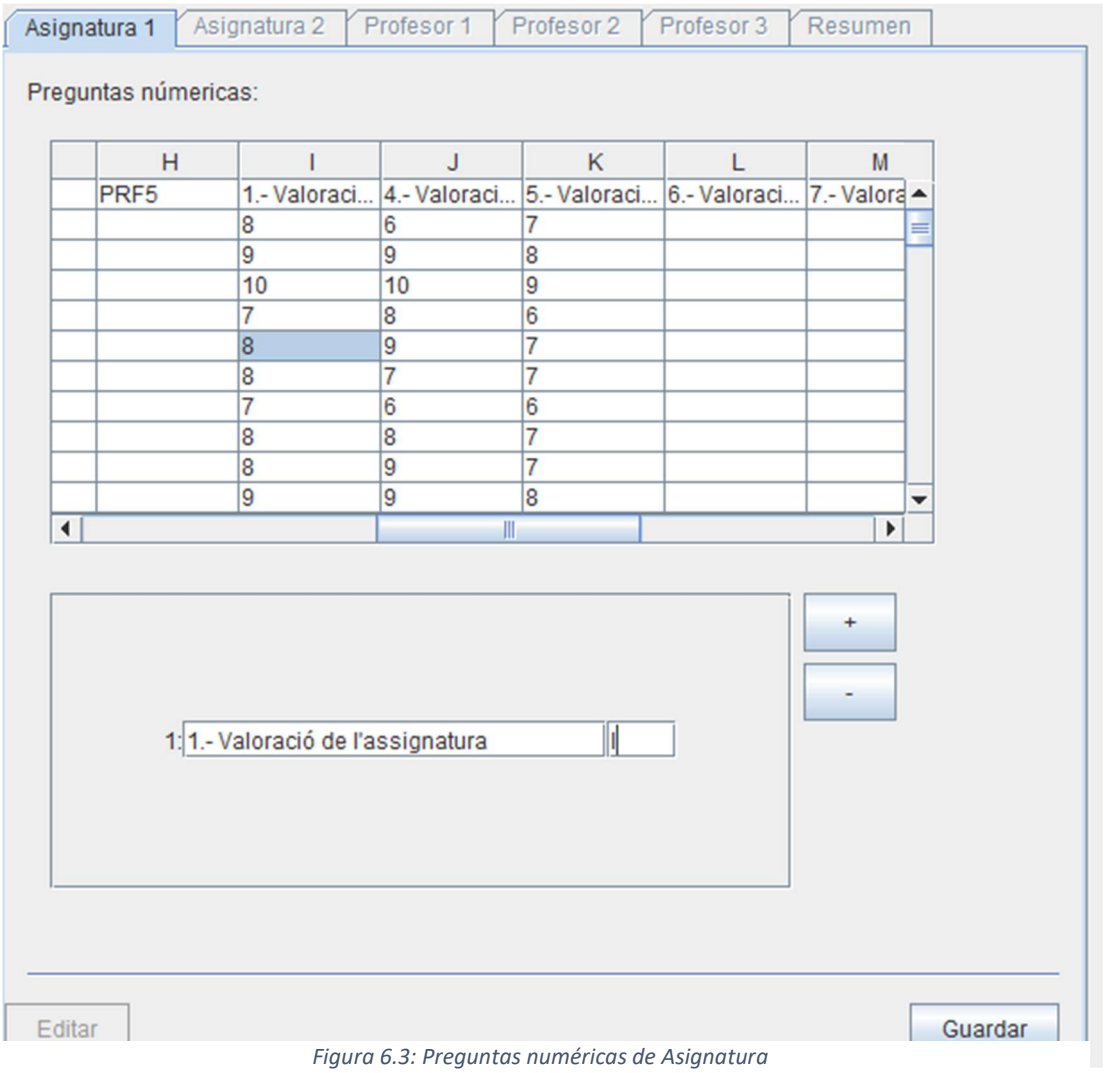

Figura 6.3: Preguntas numéricas de Asignatura

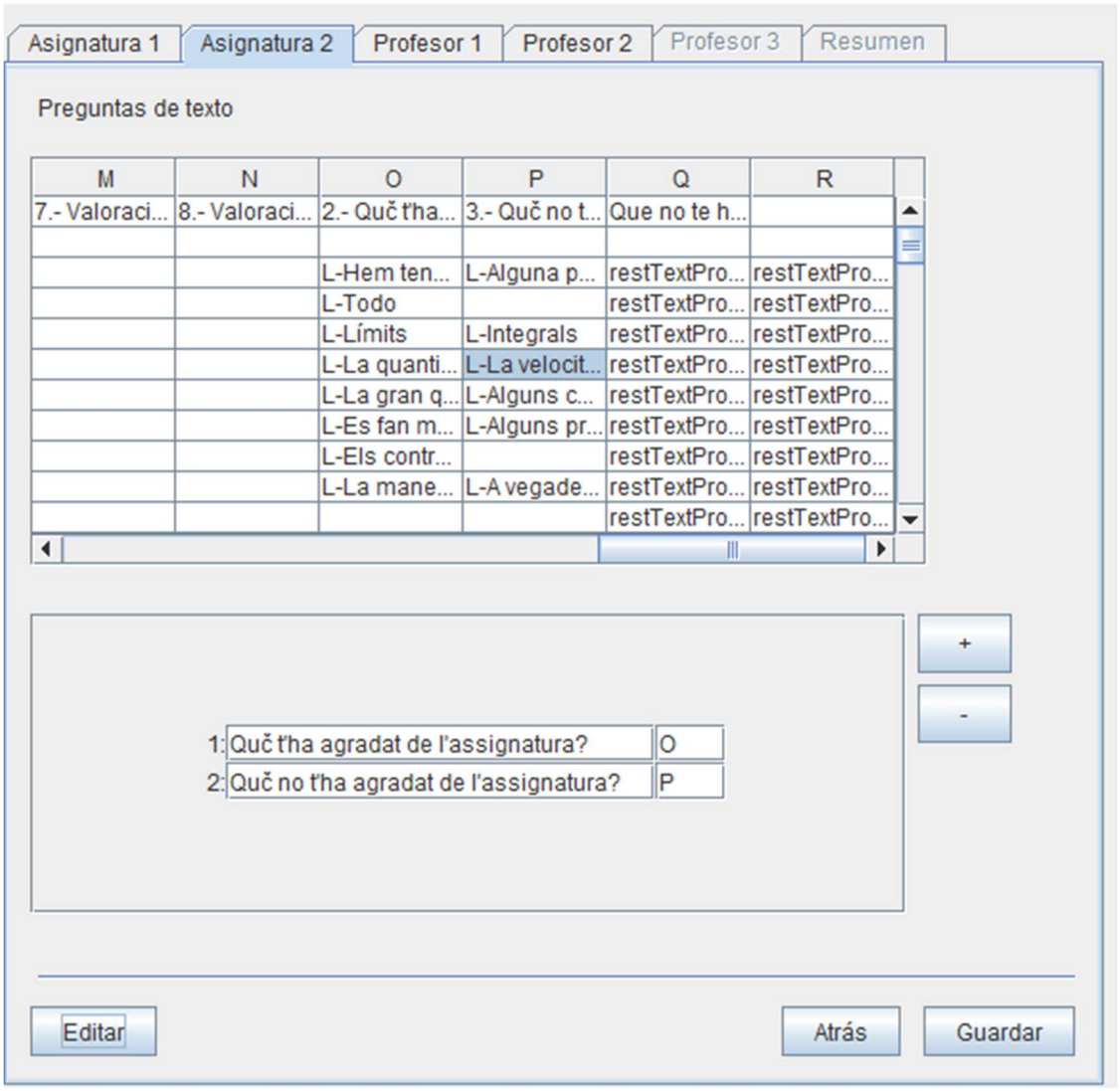

Figura 6.4: Preguntas de texto libre sobre asignaturas

6.2.2 Profesores

El usuario debe ingresar el nombre de la columna donde se encuentran el nombre de un profesor. Como máximo se pueden evaluar 5 profesores por asignatura razón por la cual solo se pueden ingresar como máximo 5 nombres de columnas.

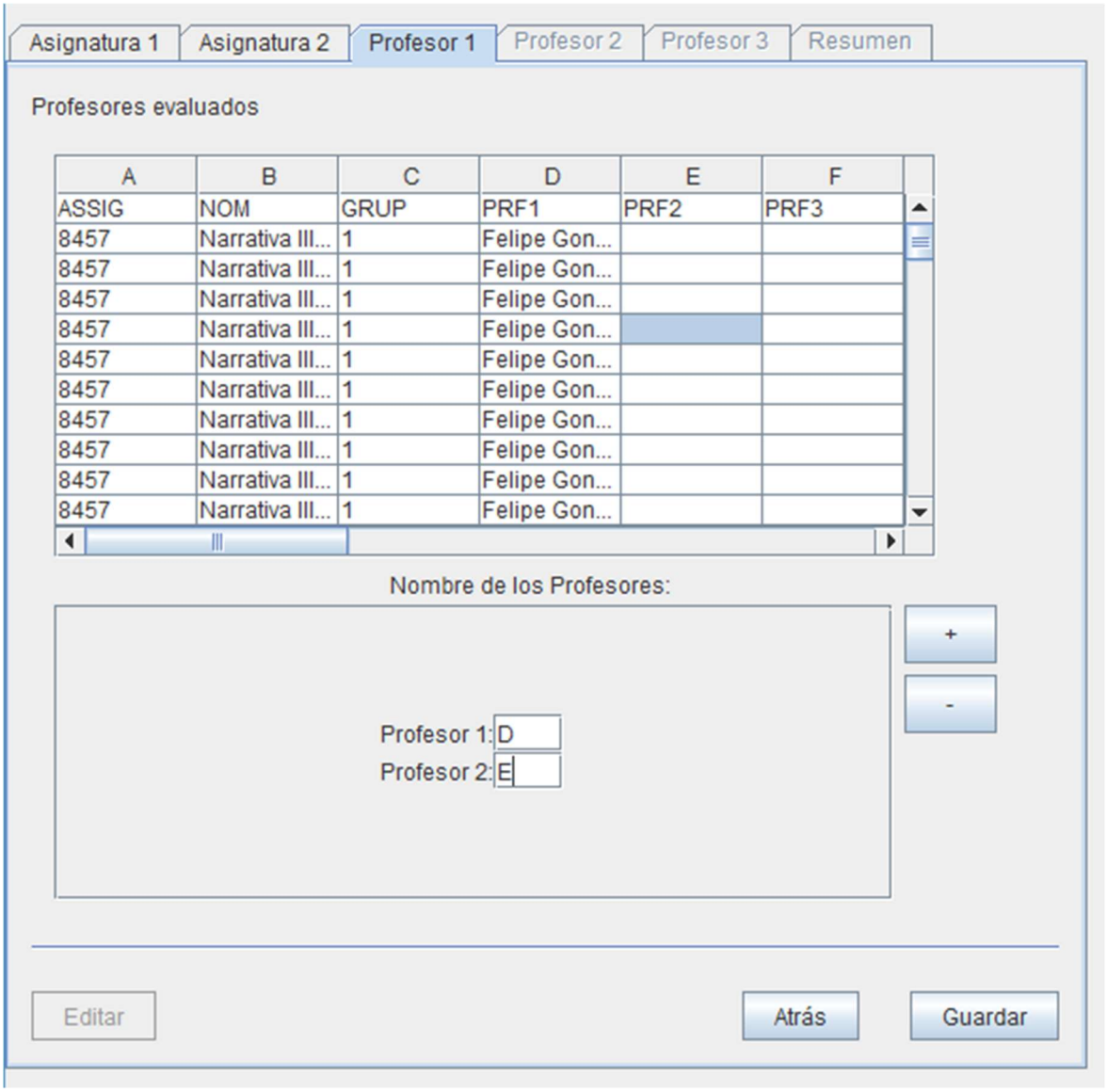

Figura 6.5: Profesores que serán evaluados

6.2.3 Preguntas relacionadas a los profesores

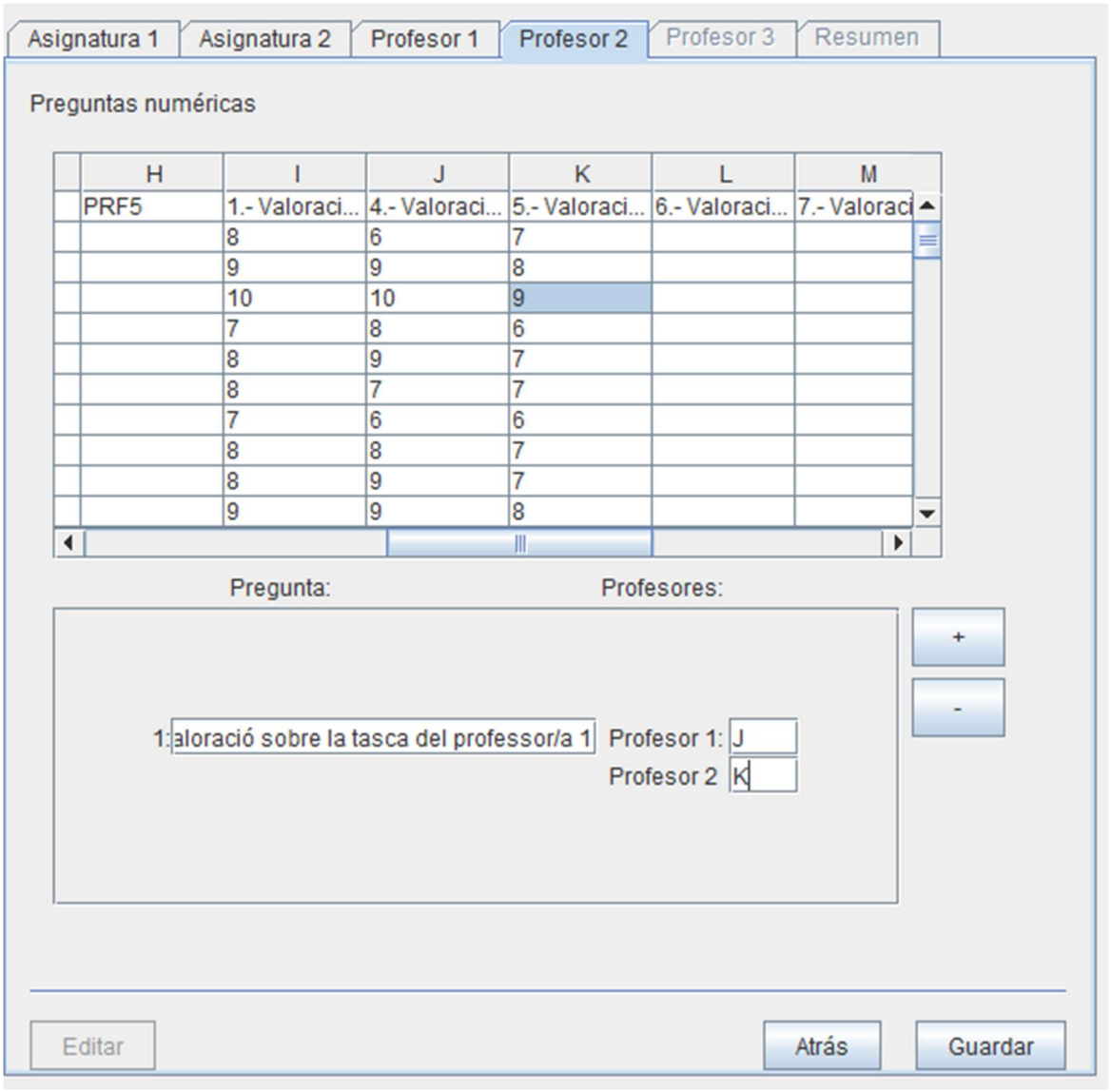

Figura 6.6: Preguntas numéricas de profesores

El usuario deberá ingresar la pregunta en el input de la columna **"pregunta"** y por cada profesor, previamente ingresado en la pestaña anterior, se tendrá que ingresar el nombre de la columna en la que se hallarán sus respuestas. Por ejemplo, en el ejemplo de la pestaña anterior, se ingresaron 2 profesores, por lo tanto, en esta pestaña se tendrá que agregar dos nombres de columna, una por cada profesor.
Al igual que en las pestañas relacionadas a las asignaturas, todos los inputs pueden ser ingresados manualmente o seleccionando las celdas del visor ubicado en la zona superior.

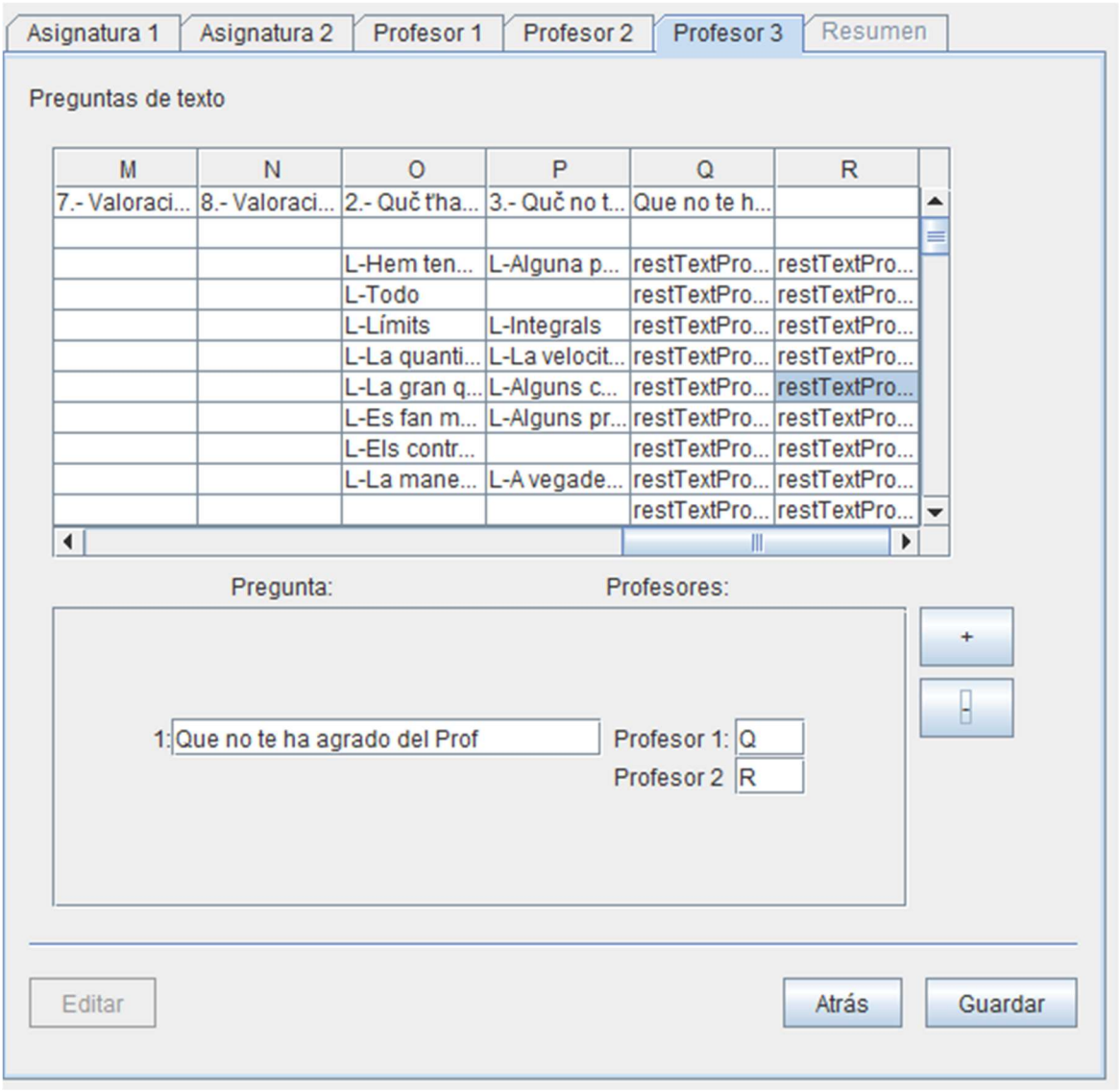

Figura 6.7: Preguntas de texto de profesores

6.2.4 Resumen

Como paso final el usuario deberá confirmar que todos los datos ingresados son correctos. Para ello se le muestra un extracto de todos los datos, agrupados por **Asignatura** y **Profesor**. Una vez que el usuario compruebe que los datos son correctos, bastará con clicar en el botón **Crear reportes** para comenzar a generar los reportes. En el caso de que el usuario desee editar algún dato podrá volver a la pestaña en la que se encuentra el dato a modificar y editarlo.

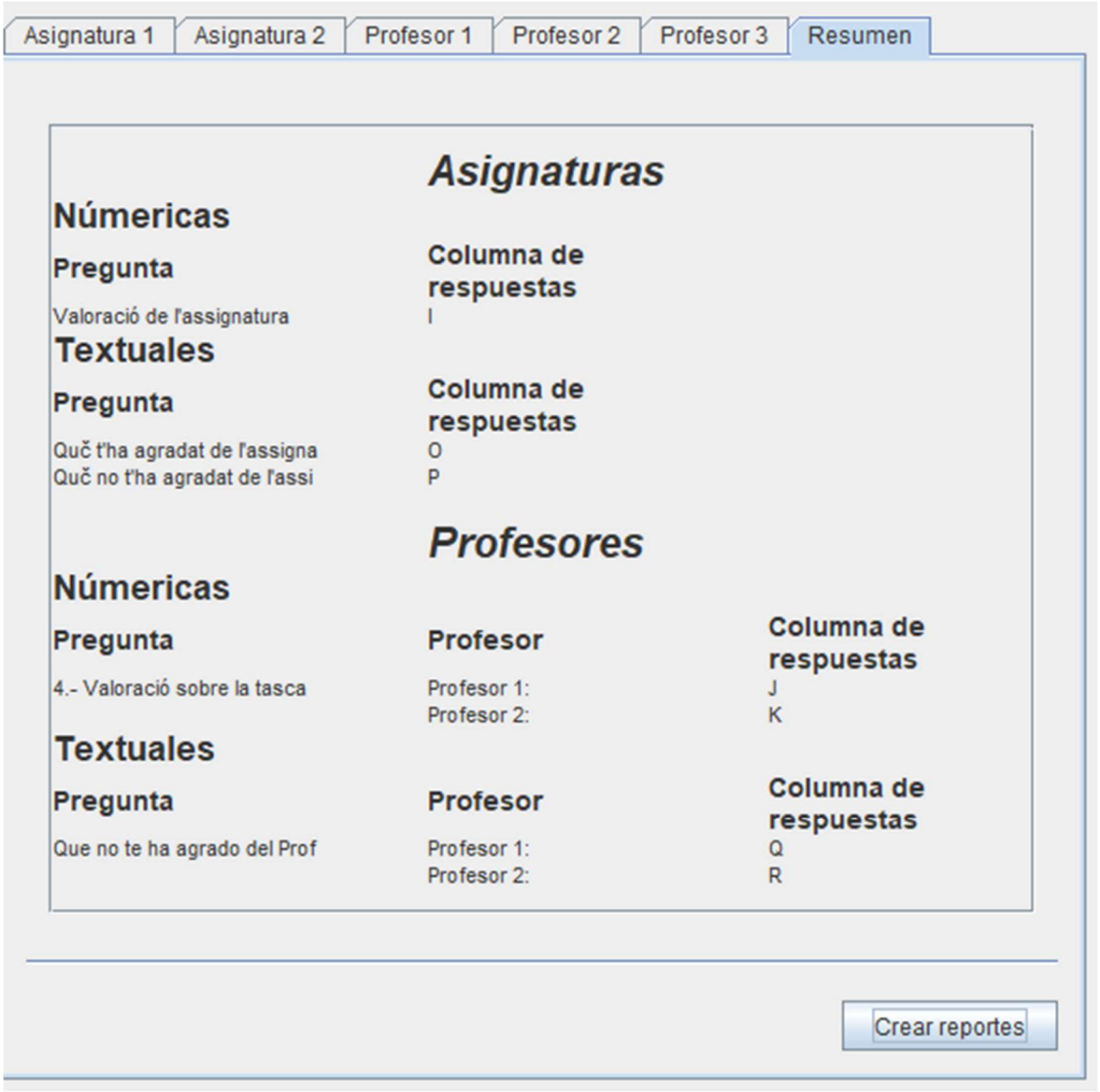

Figura 6.8: Resumen

6.2.5 Gráficos

Luego de haber sido generados los reportes, en la última pestaña de la aplicación se muestra un pequeño dashboard donde se representan datos obtenidos en las encuestas.
El dashboard está formado por tres selectores y dos gráficos.

6.2.5.1 Los selectores:

el bloque de los selectores está conformado por tres combos:

- **El combo de los Estudios**:
  Se muestran todos los estudios encontrados en el CSV de estudios que se pasó como input al inicio del programa.
- **El combo de las Asignaturas:**
  Muestra únicamente las asignaturas que correspondan al estudio que se ha seleccionado en el *combo de Estudios*
- **El combo de Profesores:**
  Muestra únicamente a los profesores que impartan la asignatura que se ha seleccionado en el *combo de Asignaturas*

  Para poder visualizar los gráficos se necesita que como mínimos se haya seleccionado algún estudio del combo de Estudios

6.2.5.2 Los Gráficos:

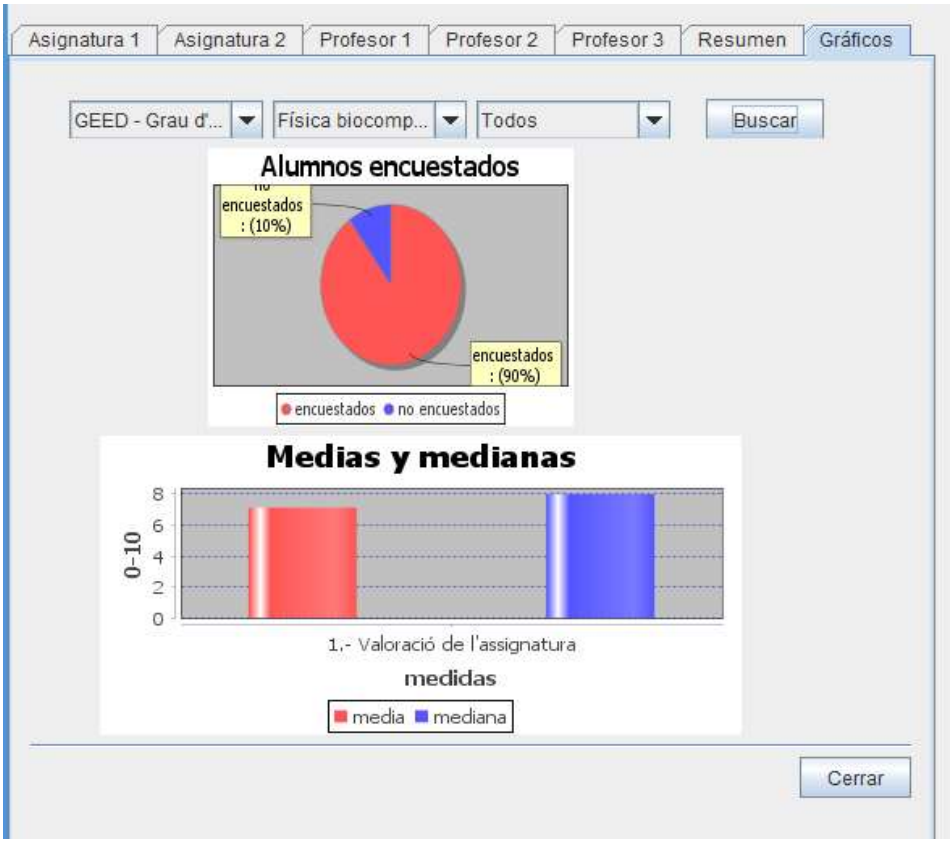

Gráfico de alumnos encuestados:

Este gráfico muestra la cantidad de alumnos que respondieron a la encuesta en contraste con la cantidad de alumnos inscritos con el profesor seleccionado. Si no se hubiera seleccionado un profesor, pero si una asignatura entonces los datos serán los de la asignatura seleccionada. En caso de que tampoco no se haya seleccionado una asignatura y solo esté seleccionado un Estudio entonces los datos serán los del Estudio seleccionado.

Gráfico de medias y medianas

Este gráfico muestra la media y la mediana obtenida por el profesor o asignatura seleccionada.

En el caso de que solo se haya seleccionado una asignatura y no un profesor, se mostrará una barra por cada pregunta numérica de la asignatura.

Si se ha seleccionado un profesor entonces se mostrará una barra por cada pregunta numérica de profesor y las barras serán agrupadas por profesor.

6.3 Los reportes:

Los reportes generados por la aplicación tienen la siguiente estructura:

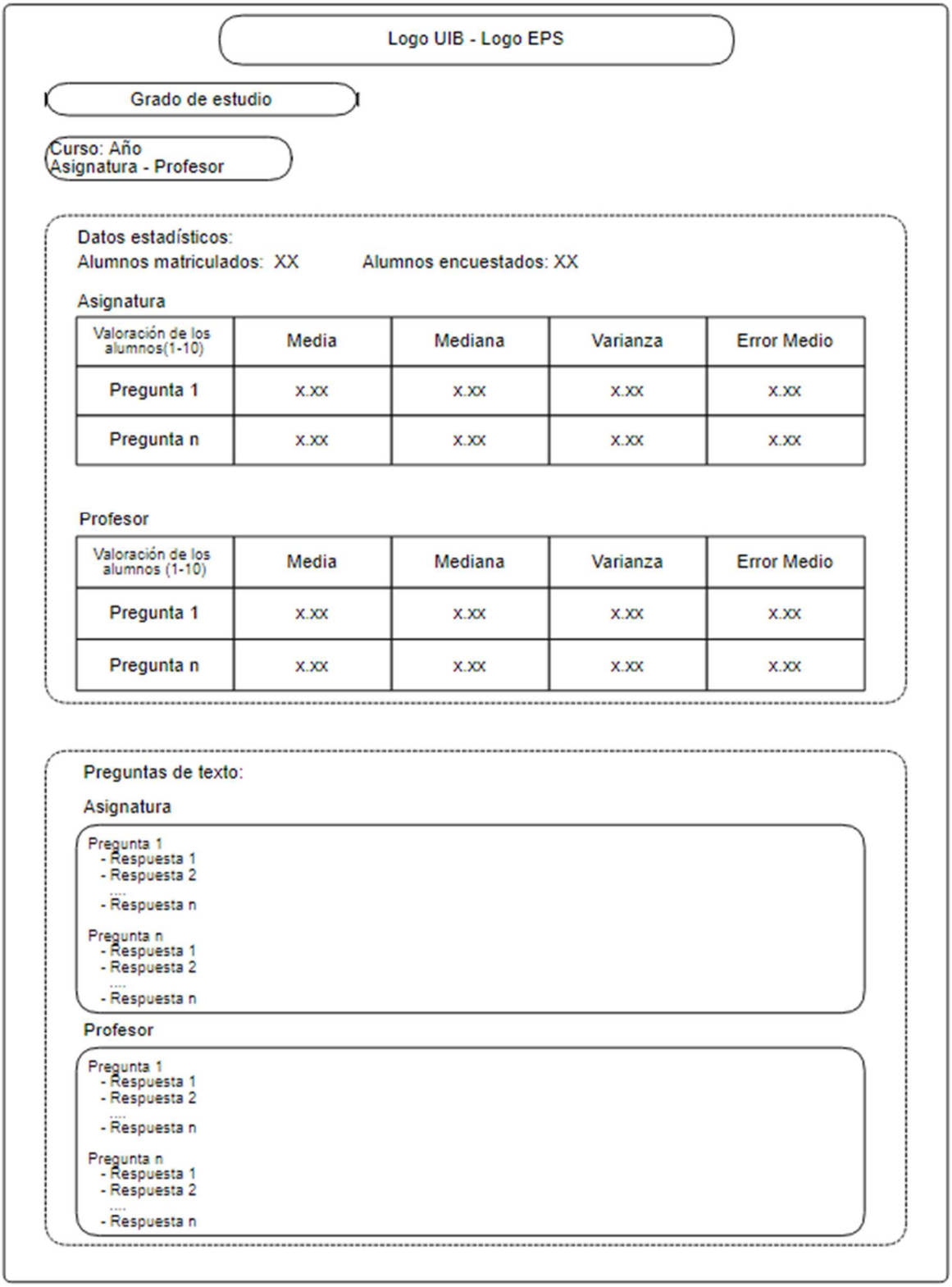

Figura 6.9: Los reportes

El reporte divide los datos generados y recopilados, en dos grupos, el primero contiene los datos estadísticos obtenidos a partir de las respuestas a preguntas numéricas de las encuestas, y el segundo recopila las preguntas de texto.
El siguiente es un ejemplo de un reporte generado por la aplicación:

Universitat de les Illes Balears

escola politècnica superior

ENQUESTES GEED - Grau d'Edificación

Curs 18-19

7148 Física biocomputacional avanzada (Pau Gasol)

Dades Estadístiques

77 Alumnes matriculats	134 Enquestes

ASSIGNATURA				
Valoració alumnes (1 a 10)	Mitjana	Mediana	Variaça	Error Mitjana (±)
Valoració de l'assignatura	7.125926	7.125926	5.63	---

PROFESSOR				
Valoració alumnes (1 a 10)	Mitjana	Mediana	Variaça	Error Mitjana (±)
4.- Valoració sobre la tasca del professor/a 1	7.029851	7.029851	6.43	---

Respostes de text lliure sobre l'assignatura

Quč t'ha agradat de l'assignatura?

- Les explicacions teoriques del professor

- La forma de explicar es bona.

- Tot

- L'estil de les pŕictiques de programació, és a dir, el temps que es dona per fer-les i les eines que se'ns han donat.

- Aprender a buscar la forma más eficiente

- La progressió de l'assignatura de manera contínua perň no excessivament rŕpida i l'atenció a l'alumne.

- La implementación de programas efectivos y interesantes

Figura 7.10: Reporte – documento Word

molt importants el que ha suposat una recerca personal.

- Poca ajuda al l'hora de resoldre els exercicis, hi podria haver les solucions dels exercicis.

- A mesura que avança el temari la dificultat s'incrementa molt notablament.

- Explicacions molt teñriques quan realment tot s'ha d'aplicar als ordinadors. Falta més ajuda en quant a exercicis i solucionaris. Es fa difícil estudiar per un tot sol.

- que es donen per entes moltes coses daquesta asignatura que no s'han vist mai i van massa rapid!!

- A veces el profesor desanima a los alumnos

- La metodología de evaluación.

- en acabar aquesta asignatura no tenim ni idea de programar

- Demasiada presión sobre el alumno,el profesor constantemente repitiendo que o haces esto, o a septiembre.Así no se hace las cosas.Mal explicado algunos conceptos que el profesor da como fáciles(evidentemente para él), pero que el alumnado le cuesta entender y tiene que recurrir a internet, otros profesionales, familiares...

- massa cotiguts.

- Les explicacions, a l'hora d'estudiar no sabem amb que basar-nos, en quina matčria fixa. Va d'un lloc a un altre i a pics no sabem quins apunts treure o el que es important...

- Faltaria una mica més d'organització.

Respostes de text lliure sobre l'professor

Que no te ha agrado del Prof

- Trob que son necessaries més hores de pŕctiques.

- Mai havia vist res d'aquesta assignatura, i el professor explica d'una manera rŕpida i al meu parer no massa comprensible, o si en alguns aspectes, perň que no ens queden grabats. Feim exercicis més complicats que els coneixements que tenim, i ens resulta molt difícil treure la solució.

- El poc temps que hi ha per aprendre bé la assigantura, sobretot a classe, i daquí els resultats d'aquesta assignatura, cada any.

- Algunos de los ejercicios de las practicas. Pero en general, me ha gustado bastante la asignatura.

- la velocidad con que se daba el temario

- La practica es demasiado compleja, y lleva excesivo tiempo realizarla a tiempo para la fecha de entrega

- La exigencia que demanen per aprovar, quan no havies fet programació en la vida.

Figura 6.11: Reporte: documento - Word

CAPÍTULO 7

Conclusión

El objetivo de este proyecto era el de desarrollar una aplicación que permitiera generar reportes a partir de encuestas realizadas a los alumnos sobre su satisfacción con la asignatura y el profesor que la imparte.
En un inicio la automatizar la generación de los reportes parecía una labor difícil debido a la poco que sabía sobre generar documentos Word usando Java además de que la mayoría de bibliotecas que permiten realizar tal labor son de pago. Otro gran reto para mí fue hacer que la aplicación sea lo más intuitiva posible, que el usuario no tenga que tomarse mucho tiempo para decidir entre muchas opciones, que cometa errores al ingresar los datos o que la aplicación le parezca confusa.

El descubrimiento y manejo de bibliotecas que me facilitaron el proceso de desarrollo no fue muy complicado. Tras horas de investigación y probar con varias bibliotecas pude hallar las que mejores soluciones daban a mis necesidades. Mantener el código de la aplicación tampoco será difícil ya que casi todas las bibliotecas usadas por la aplicación siguen siendo actualizadas por sus desarrolladores, esto permitirá ir agregando mejoras a la aplicación, de ser necesarias.

Para que la aplicación fuese lo más intuitiva y menos confusa posible la mejor solución fue usar un Wizard que no sólo guía al usuario durante todo el proceso, sino que brinda una interfaz limpia que en todo momento permite al usuario ingresar datos tan solo usando el ratón, lo que hace del uso de la aplicación una labor fácil y rápida.

Un reto personal fue agregar al código patrones de diseño de software como el MVC y el Observer. Si bien durante el desarrollo, el uso de dichos patrones no era algo necesario debido a lo poca complejidad de la aplicación, decidí implementarlos para así afianzar mis conocimientos sobre diseño de software y además tener un código fácil de mantener en el futuro.

Bibliografía

[1] Simon Ritter, “Reasons Why Java is Still #1” 2016. [Online]. Available: https://www.azul.com/4-reasons-java-still-1/

[2] “Apache™ XML Graphics Commons” 2016. [Online] Available:

https://xmlgraphics.apache.org/commons/

[3] “Java Excel API - A Java API to read, write, and modify Excel spreadsheets” [Online] Available:

http://jexcelapi.sourceforge.net/

[4] “OpenCSV” 2018. [Online] Available:

https://opencv.org/

[5] Miguel Angel Alvarez “¿Qué es MVC?” 2014. [Online]. Available:

https://desarrolloweb.com/articulos/que-es-mvc.html

[6] Universidad de Alicante “Modelo vista controlador” 2018. [Online]. Available:

https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[7] Juan Pavón Mestras “El patrón Modelo-Vista-Controlador (MVC)” 2009. [Online]. Available:

https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf

[8] Nick Babich “Wizard Design Pattern” 2017. [Online]. Available:

https://uxplanet.org/wizard-design-pattern-8c86e14f2a38

[9] “Wizard Design Patter” 2018. [Online]. Available

http://ui-patterns.com/patterns/Wizard

[10] Jodi Bollaert “Creating a Wizard” 2001. [Online]. Available:

https://www.ibm.com/developerworks/library/us-wizard/

[11] “Observer Design Pattern” 2018. [Online]. Available:

https://sourcemaking.com/design_patterns/observer

[12] “Observer Pattern” 2018. [Online]. Available:

https://www.oodesign.com/observer-pattern.html

Anexos

Anexo A. *Generator:*

Clase inicial desde la que se ejecuta la aplicación. Esta clase crea las instancias del Controlador y del Modelo, luego será el Controlador quien cree la instancia de la Vista

```
public class Generator {


    public static void main(String[] args) {
        /* Use an appropriate Look and Feel */
        try {
        //UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        } catch (UnsupportedLookAndFeelException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        /* Turn off metal's use of bold fonts */
        UIManager.put("swing.boldMetal", Boolean.FALSE);

        //Schedule a job for the event dispatch thread:
        //creating and showing this application's GUI.

            GeneratorModelInterface gmi = new GeneratorModel();
            GeneratorControllerInterface gci = new GeneratorController(gmi);
    }
}
```

Clases relacionadas al patrón de diseño MVC

Anexo B. *GeneratorView:*

La Vista del patrón MVC presenta al usuario los datos del modelo.

Es esta clase la que se encarga de crear y mostrar la interfaz de la aplicación

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GeneratorView implements ItemListener {
    JPanel cards;
    // each wizard's tab
    Landing landing;
    // first aplication window, the file inputs are here
    Wizard wizard;
    // set of tabs which help the inputs management
    JFrame frame;
    final static String LANDING = "landing";
    final static String WIZARD = "wizard";
    static GeneratorControllerInterface gci;
    GeneratorModelInterface gmi;


    public GeneratorView(GeneratorControllerInterface gci,
            GeneratorModelInterface gmi){
        this.gci    = gci;
        this.gmi    = gmi;
    }

    public void addComponentToPane(Container pane) {
        landing = new Landing(gci);
        wizard = new Wizard(gci);

        //Create the panel that contains the "cards".
        cards = new JPanel(new CardLayout());
        cards.add(landing, LANDING);
        cards.add(wizard, WIZARD);

        pane.add(cards, BorderLayout.CENTER);
    }

    public void itemStateChanged(ItemEvent evt) {
        System.out.println("el boton de afuera");
    }


    public void createAndShowGUI() {
        //Create and set up the window.
        frame = new JFrame("CardLayoutDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Create and set up the content pane.
        addComponentToPane(frame.getContentPane());

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public void closeGUIAndProgram(){
        frame.setVisible(false);
        frame.dispose();
        // if you have other similar frames around, you should dispose them, too.

        // finally, call this to really exit.
        System.exit(0);
    }

}
```

Anexo C: *GeneratorModel*:

Es el Modelo del patrón MVC, contiene únicamente los datos puros de aplicación; no contiene lógica que describe cómo pueden presentarse los datos al usuario. Envía a la **Vista** la información que se le solicita, a través del controlador, para ser mostrada

```java
package Main;

import GeneratorClasses.Proffesor;
import GeneratorClasses.Report;
import InterfaceClasses.Question;
import InterfaceClasses.QuestionType;
import InterfaceClasses.Questions;
import java.util.ArrayList;
import java.util.List;

/**
 * @author ernesto
 */
public class GeneratorModel implements GeneratorModelInterface{
    private String  excelPath;
    // path to excel file wich contains the surveys answers
    private String  csvPath;
    // path to CSV file wich contains the subjects information
    private String  reportPath;
    // path to the file where generatedwich contains the surveys answers
    private Questions professorsName;
    // array of professor's names
    private Questions subjectNumericalData
            = new Questions(QuestionType.category.SUBJECT, QuestionType.type.NUMERICAL);
    // numerical type question about subjects, and their answers
    private Questions subjectTextualData
            = new Questions(QuestionType.category.SUBJECT, QuestionType.type.TEXTUAL);
    // textual type question about subjects, and their answers
    private Questions professorNumericalData
            = new Questions(QuestionType.category.PROFFESOR, QuestionType.type.NUMERICAL);
    // numerical type question about professors, and their answers
    private Questions professorTextualData
            = new Questions(QuestionType.category.PROFFESOR, QuestionType.type.TEXTUAL);
    // textual type question about professors, and their answers
    private List<Proffesor> profesores          = new ArrayList<Proffesor>();
    // list of professor's objects
    private ArrayList<Report> reports  = new ArrayList<Report>();
    // list of reports

    public Questions getProffesorsColName(){
        if(professorsName == null) {
            professorsName = new Questions(QuestionType.category.DEFAULT, QuestionType.type.SIMPLE);
            Question evaluatedProffesor = new Question();
            professorsName.add(evaluatedProffesor);
        }
        return professorsName;
    }

    public Question getEvaluatedProffesor(int ord){
        if(professorsName == null) {
            professorsName = new Questions(QuestionType.category.DEFAULT, QuestionType.type.SIMPLE);
            Question evaluatedProffesor = new Question();
            professorsName.add(evaluatedProffesor);
        }
        return professorsName.getQuestions().get(ord);
    }
```

Anexo D. *GeneratorController:*

Es el Controlador del patrón MVC, escucha los eventos desencadenados por la vista y ejecuta la reacción apropiada a estos eventos. Toda la lógica para representar o guardar los datos del modelo en la vista, se encuentran en el

```java
import InterfaceClasses.Question;
import InterfaceClasses.QuestionType;
import InterfaceClasses.Questions;
import InterfaceClasses.SubjectQuestion;
import InterfaceClasses.Table;
import Panels.QuestionsPanel;
import Utils.MathUtils;
import Utils.PanelDataException;
import documentClasses.WordDocument;
import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import jxl.*;

/**
 *
 * @author ernesto
 */
public class GeneratorController implements GeneratorControllerInterface{

    GeneratorView gview;
    // MVC view
    GeneratorModelInterface gmi;
    // MVC controller

    public GeneratorController(GeneratorModelInterface gmi){
        this.gmi = gmi;
        gview = new GeneratorView(this, gmi);
        gview.createAndShowGUI();
    }

    public void nextButton(JButton jb , JTabbedPane jtp ){
        System.out.println(jtp.getSelectedIndex());
        if(jtp.getSelectedIndex() < jtp.getTabCount() - 1){
            jtp.setSelectedIndex(jtp.getSelectedIndex() + 1);
            jtp.setEnabledAt(jtp.getSelectedIndex(), true);
        }else{
            System.out.println("no existe siguiente");
        }
    }

    public void previousButton(JButton jb, JTabbedPane jtp){
        System.out.println(jtp.getSelectedIndex());
        if(jtp.getSelectedIndex() > 0){
            jtp.setSelectedIndex(jtp.getSelectedIndex() - 1);
        }else{
            System.out.println("no existe anterior");
        }
    }

    public void setTextTextfield(JTextField textField, String text){
        textField.setText(text);
    }

    public void setExcelPath(String excelPath){
        gmi.setExcelPath(excelPath);
    };

    public void setCsvPath(String csvPath){
        gmi.setCsvPath(csvPath);
    };

    public void setReportPath(String reportPath){
        gmi.setReportPath(reportPath);
```

controlador.

```
public String getExcelPath(){
    return gmi.getExcelPath();
}

public void loadExcelTable(String excelPath){
    gview.wizard.subjNumerical.loadExcel(excelPath);
    gview.wizard.subjTextual.loadExcel(excelPath);
    gview.wizard.proffEvaluated.loadExcel(excelPath);
    gview.wizard.proffTextual.loadExcel(excelPath);
    gview.wizard.proffNumerical.loadExcel(excelPath);

}

public void questions_buttonAdd(Table questionTable,ModelTable questionTableModel){
}

public void question_buttonRemove(Table questionTable, ModelTable questionTableModel){
    questionTableModel.removeLastRowTable();
};

public Questions getProffesors(){
    return gmi.getProffesorsColName();
}

public void updateTableData(QuestionsPanel questionPanel) throws PanelDataException{
    Questions questions = new Questions(questionPanel.getCategory(), questionPanel.getType());
    ArrayList<String>   questionsRawData = questionPanel.getQuestions();
    ArrayList<String>   answersRawData = questionPanel.getAnswers();
    Integer[]       error = checkData(questionsRawData);
    if(error[0] != 0)
        throw new PanelDataException("Falta ingresar la pregunta en la fila: " + error[1].toString());
    error = checkData(answersRawData);
    if(error[0] == 1)
        throw new PanelDataException("Falta ingresar la columna de respuesta en la fila: " + error[1].toString())
    if(error[0] == 2)
        throw new PanelDataException("La columna de respuesta no es alfanumérica: " + error[1].toString());

    if(questionPanel.getCategory() == QuestionType.category.SUBJECT){
        SubjectQuestion     subjectQuestion;
        for(int i = 0; i < questionPanel.getQuestions().size(); i++){
            subjectQuestion = new SubjectQuestion();
            subjectQuestion.setQuestion(questionsRawData.get(i));
            subjectQuestion.setAnswer(answersRawData.get(i));
            questions.add(subjectQuestion);
        }
        if(questionPanel.getType() == QuestionType.type.NUMERICAL){
            gmi.setSubjectNumericalData(questions);
        }else if(questionPanel.getType() == QuestionType.type.TEXTUAL){
            gmi.setSubjectTextualData(questions);
        }
    }else if(questionPanel.getCategory() == QuestionType.category.PROFFESOR){
        ProffesorQuestion   proffesorQuestion;
        int index;
        int proffesorsSize = gmi.getProffesorsColName().getQuestions().size();
        for(int i = 0; i < questionPanel.getQuestions().size(); i++){
            proffesorQuestion = new ProffesorQuestion();
            proffesorQuestion.setQuestion(questionsRawData.get(i));
            for(int j = 0; j < proffesorsSize ; j++){
                index = (proffesorsSize * i) + j;
                proffesorQuestion.addProffesorAnswers(answersRawData.get(index));
            }
            questions.add(proffesorQuestion);
        }
        if(questionPanel.getType() == QuestionType.type.NUMERICAL){
            gmi.setProfessorNumericalData(questions);
        }else if(questionPanel.getType() == QuestionType.type.TEXTUAL){
            gmi.setProfessorTextualData(questions);
        }
    }else{
        Question    simpleQuestion;
        for(int i = 0; i < questionPanel.getAnswers().size(); i++){
            simpleQuestion = new Question();
            simpleQuestion.setQuestion(answersRawData.get(i));
            questions.add(simpleQuestion);
        }
        if(questionPanel.getType() == QuestionType.type.SIMPLE){
            clearProffesorsData();
            gmi.setProffesorsColName(questions);
            updateProffesorsData();
        }
```

-

```java
private Integer[] checkData(ArrayList<String> questions){
    // error = 1 -> blank TextField or full of whitespaces only
    // error = 2 -> TextField value is not alphanumeric
    int cont = 0;
    QuestionsPanel auxPanel = new QuestionsPanel();
    Integer[] errorData = {0,0};
    Iterator iterator;
    String  data;

    iterator = questions.iterator();
    while(iterator.hasNext() && errorData[0] == 0){
        data = (String) iterator.next();
        if(data.equals("") || data.trim().isEmpty()) {
            errorData[0] = 1;
        }else if(data.length() == auxPanel.ANSWERS_TEXTFIELD_WIDTH){
            if(data.matches("^.*[^a-zA-Z0-9 ].*$")){
                errorData[0] = 2;
            }
        }
        if(errorData[0] != 0) errorData[1] = cont;
        cont++;
    }
    return errorData;
}



public void clearProffesorsData(){
    gview.wizard.proffNumerical.clearData();
    gview.wizard.proffTextual.clearData();
}

public void updateProffesorsData(){
    gview.wizard.proffNumerical.setEnabled(true);
    gview.wizard.tab4_save.setEnabled(true);
    gview.wizard.proffTextual.setEnabled(true);
    gview.wizard.tab5_save.setEnabled(true);
    gview.wizard.proffNumerical.myInitComponents();
    gview.wizard.proffTextual.myInitComponents();
}

public void loadResumen(){
    gview.wizard.resumen.myInitComponents();
}

public Questions getEvaluatedProffesors(){
    return gmi.getProffesorsColName();
};

public Questions getSubjectNumericalData(){
    return gmi.getSubjectNumericalData();
};

public Questions getSubjectTextualData(){
    return gmi.getSubjectTextualData();
};
public Questions getProffesorNumericalData(){
    return gmi.getProfessorNumericalData();
};

public Questions getProffesorTextualData(){
    return gmi.getProfessorTextualData();
};

public ArrayList<Report> getStatisticsResume(){
    return gmi.getStatisticsResume();
}

public void generateReports(){

    /*  abrir pdf de datos por asignatura
        Leer csv de respuestas
            crear subject
            llenar subject con datos para preguntas numéricas
            llenar subject con datos para preguntas textuales
            llenar profesor con datos para preguntas numéricas
            llenar profesor con datos para preguntas textuales
            llenar subject con información del pdf de asignaturas
            crear reporte de asignatura
        terminar de leer */
```

```
SubjectsInformation subjectsInformation = new SubjectsInformation(";", gmi.getCsvPath());
subjectsInformation.loadSubejctsInformation();
ArrayList<Report> reports = new ArrayList();

String cod1="", cod2="", group1="", group2="";

try {
    WorkbookSettings workbookSettings = new WorkbookSettings();
    workbookSettings.setEncoding("cp1250");

    Workbook archivoExcel = Workbook.getWorkbook(new File(gmi.getExcelPath()), workbookSettings);
    for (int sheetNo = 0; sheetNo < archivoExcel.getNumberOfSheets(); sheetNo++){
        // each excel page
        Sheet page = archivoExcel.getSheet(sheetNo);
        for(int indexRow = 1; indexRow < page.getRows(); indexRow++){
        // Skipping the header line
            cod1 = page.getCell(0,indexRow).getContents();
            group1 = page.getCell(2,indexRow).getContents();
            if(!cod1.equals(cod2) || !group1.equals(group2)){
                Report report = new Report();
                Subject subject = new Subject(cod1, page.getCell(1,indexRow).getContents(), group1);
                subject.setEstudio(subjectsInformation.getDegree(cod1, group1));
                subject.setEnrolledStudents(subjectsInformation.getEnrolledStudents(cod1, group1));
                if(subject.getEnrolledStudents() != null && subject.getEstudio() != null){
                    initializeSubject(subject);
                    // Initialize questions and answers array from subjects
                    for(Question question : gmi.getProffesorsColName().getQuestions()){
                        String colProffesorName = question.getQuestion();
                        String proffName = page.getCell(
                                Integer.valueOf(MathUtils.columnNameToInteger(colProffesorName)),
                                indexRow).getContents();
                        if(proffName != null && !proffName.trim().isEmpty() ){
                            Proffesor proffesor = new Proffesor(proffName);
                            // Initialize each professor's questions
                            initialiseProffesors(proffesor,report.getProfessors().size());
                            report.addProffesor(proffesor);
                        }
                    }
                    report.setSubject(subject);
                    reports.add(report);
                }
            }
            Report lastReport = reports.get(reports.size()-1);

            for(Answers numericalAnswers : lastReport.getSubject().getAnswers(QuestionType.type.NUMERICAL)){
                numericalAnswers.add(page.getCell(numericalAnswers.getColumn(), indexRow).getContents());
            }
            for(Answers textualAnswers : lastReport.getSubject().getAnswers(QuestionType.type.TEXTUAL)){
                textualAnswers.add(page.getCell(textualAnswers.getColumn(), indexRow).getContents());
            }

            //Proffesor's answers
            for(Proffesor proffesor : lastReport.getProfessors()){
                for(Answers numericalAnswers : proffesor.getAnswers(QuestionType.type.NUMERICAL)){
                    numericalAnswers.add(page.getCell(numericalAnswers.getColumn(), indexRow).getContents());
                }
                for(Answers textualAnswers : proffesor.getAnswers(QuestionType.type.TEXTUAL)){
                    textualAnswers.add(page.getCell(textualAnswers.getColumn(), indexRow).getContents());
                }
            }
            cod2 = cod1;
            group2 = group1;
        }
    }
} catch (Exception ioe) {
    ioe.printStackTrace();
}


for(Report report : reports){
    try {
        for(Answers numericalAnswers : report.getSubject().getAnswers(QuestionType.type.NUMERICAL)){
            NumericalAnswers numAnswers = (NumericalAnswers) numericalAnswers;
            setStatisticalValues(numAnswers, report.getSubject().getEnrolledStudents());
        };
        for(Proffesor proffesor : report.getProfessors()){
            for(Answers numericalAnswers : proffesor.getAnswers(QuestionType.type.NUMERICAL)){
                NumericalAnswers numAnswers = (NumericalAnswers) numericalAnswers;
                setStatisticalValues(numAnswers, report.getSubject().getEnrolledStudents());
            }
```

```
            createReport(report);
            System.out.println(report.toString());
            saveReportToResume(report);
        } catch (Exception ex) {
            Logger.getLogger(GeneratorController.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

private void saveReportToResume(Report report){
    gmi.addReportToStatisticResume(report);
}

private void setStatisticalValues(NumericalAnswers numericalAnswers, Integer enrolledStudents){
    NumericalAnswers numAnswers = (NumericalAnswers) numericalAnswers;
    numAnswers.setMean(MathUtils.getMean(numAnswers.getAnswers()));
    numAnswers.setMedian(MathUtils.getMedian(numAnswers.getAnswers()));
    numAnswers.setVariance(MathUtils.getVariance(numAnswers.getAnswers(), numAnswers.getMedian()));
    numAnswers.setMeanError(MathUtils.getMeanError(
            numAnswers.getAnswers(), numAnswers.getVariance(), enrolledStudents));
}

private void initializeProffesors(Proffesor proffesor, int proffesorIndex){

        for(Question question : gmi.getProfessorNumericalData().getQuestions()){
            ProffesorQuestion proffNumericalQuestion = (ProffesorQuestion) question;
            NumericalAnswers numericalAnswers = new NumericalAnswers();
            numericalAnswers.setType(QuestionType.type.NUMERICAL);
            numericalAnswers.setQuestion(proffNumericalQuestion.getQuestion());
            numericalAnswers.setColumn(Integer.valueOf(MathUtils.columnNameToInteger(
                    proffNumericalQuestion.getProffesorsAnswers().get(proffesorIndex))));
            numericalAnswers.setAnswers(new ArrayList());
            proffesor.addAnswers(numericalAnswers);

        }
        for(Question question : gmi.getProfessorTextualData().getQuestions()){
            ProffesorQuestion proffTextualQuestion = (ProffesorQuestion) question;
            Answers textualAnswers = new Answers();
            textualAnswers.setType(QuestionType.type.TEXTUAL);
            textualAnswers.setQuestion(proffTextualQuestion.getQuestion());
            textualAnswers.setColumn(Integer.valueOf(MathUtils.columnNameToInteger(
                    proffTextualQuestion.getProffesorsAnswers().get(proffesorIndex))));
            textualAnswers.setAnswers(new ArrayList());
            proffesor.addAnswers(textualAnswers);
        }
}

private void initializeSubject(Subject subject){
    for (Question subjNumQuestion : gmi.getSubjectNumericalData().getQuestions()){
        SubjectQuestion subjQuestion = (SubjectQuestion) subjNumQuestion;
        NumericalAnswers numericalAnswers = new NumericalAnswers();
        numericalAnswers.setType(QuestionType.type.NUMERICAL);
        numericalAnswers.setQuestion(subjQuestion.getQuestion());
        numericalAnswers.setColumn(Integer.valueOf(MathUtils.columnNameToInteger(
                subjQuestion.getAnswer())));
        numericalAnswers.setAnswers(new ArrayList());
        subject.addAnswers(numericalAnswers);
    }
    for (Question subjTextQuestion : gmi.getSubjectTextualData().getQuestions()){
        SubjectQuestion subjQuestion = (SubjectQuestion) subjTextQuestion;
        Answers textualAnswers = new Answers();
        textualAnswers.setType(QuestionType.type.TEXTUAL);
        textualAnswers.setQuestion(subjQuestion.getQuestion());
        textualAnswers.setColumn(Integer.valueOf(MathUtils.columnNameToInteger(
                subjQuestion.getAnswer())));
        textualAnswers.setAnswers(new ArrayList());
        subject.addAnswers(textualAnswers);
    }

}

private void createReport(Report report) throws Exception {
    for (Proffesor proffesor : report.getProfessors()){
        WordDocument wordDocument = new WordDocument(
                report.getSubject(), proffesor, gmi.getReportsPath());
        wordDocument.generateDocument();
    }
}
```

Anexo E. *Landing:*

Esta clase crea la Landing de la aplicación, gestiona los inputs relacionados a los archivos y controla que los mismos tengan el formato correcto

```
import Utils.FileUtils;
import Utils.PanelDataException;

import java.awt.CardLayout;
import javax.swing.*;

/**
 * @author ernesto
 */
public class Landing extends javax.swing.JPanel {

    private GeneratorControllerInterface gci;
    private FileUtils fileUtils;
    public Landing(GeneratorControllerInterface gti) {
        this.gci = gti;
        fileUtils = new FileUtils();
        initComponents();
    }

    @SuppressWarnings("unchecked")
    Generated Code


    private void LandingPanelOkActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            if (!excelPanelTextfield.getText().isEmpty()
                    && !csvPanelTextfield.getText().isEmpty() && !reportPanelTextfield.getText().isEmpty()) {
                JPanel cards = (JPanel) getParent();
                CardLayout cl = (CardLayout) (cards.getLayout());
                cl.next(cards);
                gci.loadExcelTable(gci.getExcelPath());
            } else {
                throw new PanelDataException("Falta ingresar alguna de las rutas necesarias");
            }
        } catch (PanelDataException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
                    JOptionPane.WARNING_MESSAGE);
        }
    }

    private void csvPanelButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser();
        String[] filePathAndName = fileUtils.getFilePathAndName(fileChooser, new String[] {"csv"});
        gci.setCsvPath(filePathAndName[0]);
        gci.setTextTextfield(csvPanelTextfield, filePathAndName[1]);
    }
    private void reportPanelButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser();
        String[] filePathAndName = fileUtils.getFilePathAndName(fileChooser, null);
        gci.setReportPath(filePathAndName[0]);
        gci.setTextTextfield(reportPanelTextfield, filePathAndName[1]);
    }

    private void excelPanelButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser();
        String[] filePathAndName = fileUtils.getFilePathAndName(fileChooser, new String[] {"xls", "xlsx"});
        gci.setExcelPath(filePathAndName[0]);
        gci.setTextTextfield(excelPanelTextfield, filePathAndName[1]);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton LandingPanelCancel;
    private javax.swing.JButton LandingPanelOk;
    private javax.swing.JButton csvPanelButton;
    private javax.swing.JLabel csvPanelLabel;
    private javax.swing.JTextField csvPanelTextfield;
    private javax.swing.JButton excelPanelButton;
    private javax.swing.JLabel excelPanelLabel;
    private javax.swing.JTextField excelPanelTextfield;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JButton reportPanelButton;
    private javax.swing.JLabel reportPanelLabel;
    private javax.swing.JTextField reportPanelTextfield;
    // End of variables declaration
```

Anexo F. *Wizard*

Crea y gestiona el conjunto de tabs además de controlar los botones e inputs de cada uno de los tabs.

```
import Utils.PanelDataException;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JOptionPane;

public class Wizard extends javax.swing.JPanel {

    private GeneratorControllerInterface gci;
    public SubjectQuestionsPanel    subjNumerical;
    public SubjectQuestionsPanel    subjTextual;
    public SimpleQuestionsPanel     evaluatedProf;
    public ProffesorQuestionsPanel  proffNumerical;
    public ProffesorQuestionsPanel  proffTextual;
    public Resumen                  resumen;
    public GraphsPanel2             graphsPanel2;

    public Wizard(){};

    public Wizard(GeneratorControllerInterface gci) {
        this.gci = gci;
        initComponents();
        myInit();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void tab6_generateReports(java.awt.event.ActionEvent evt) {
        gci.generateReports();
        graphsPanel2.myInitComponents();
        gci.nextButton((JButton) evt.getSource(), tabbedPane);
    }

    private void tab5_editActionPerformed(java.awt.event.ActionEvent evt) {
        proffTextual.setEnabled(true);
        tab5_save.setEnabled(true);
    }

    private void tab5_saveActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            gci.updateTableData(proffTextual);
            proffTextual.setEnabled(false);
            gci.loadResumen();
            gci.nextButton((JButton) evt.getSource(), tabbedPane);
            tab5_edit.setEnabled(true);
            tab5_save.setEnabled(false);
        } catch (PanelDataException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
                JOptionPane.ERROR_MESSAGE);
        }
    }

    private void tab5_backActionPerformed(java.awt.event.ActionEvent evt) {
        gci.previousButton((JButton) evt.getSource(), tabbedPane);
    }

    private void tab4_editActionPerformed(java.awt.event.ActionEvent evt) {
        proffNumerical.setEnabled(true);
        tab4_save.setEnabled(true);
    }

    private void tab4_saveActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            gci.updateTableData(proffNumerical);
            proffNumerical.setEnabled(false);
            gci.nextButton((JButton) evt.getSource(), tabbedPane);
            tab4_edit.setEnabled(true);
            tab4_save.setEnabled(false);
        } catch (PanelDataException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
                JOptionPane.ERROR_MESSAGE);
        }
    }

    private void tab4_backActionPerformed(java.awt.event.ActionEvent evt) {
        gci.previousButton((JButton) evt.getSource(), tabbedPane);
    }

    private void tab3_editActionPerformed(java.awt.event.ActionEvent evt) {
        evaluatedProf.setEnabled(true);
```

```
private void tab3_backActionPerformed(java.awt.event.ActionEvent evt) {
    gci.previousButton((JButton) evt.getSource(), tabbedPane);
}

private void tab3_saveActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        gci.updateTableData(evaluatedProf);
        evaluatedProf.setEnabled(false);
        gci.nextButton((JButton) evt.getSource(), tabbedPane);
        tabbedPane.setEnabledAt(tabbedPane.getSelectedIndex() + 1, false);
        //we disable evaluatedProf
        tab3_edit.setEnabled(true);
        tab3_save.setEnabled(false);
    } catch (PanelDataException ex) {
        JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
            JOptionPane.ERROR_MESSAGE);
    }
}

private void tab2_editActionPerformed(java.awt.event.ActionEvent evt) {
    subjTextual.setEnabled(true);
    tab2_save.setEnabled(true);
}

private void tab2_backActionPerformed(java.awt.event.ActionEvent evt) {

    gci.previousButton((JButton) evt.getSource(), tabbedPane);
}

private void tab2_saveActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        gci.updateTableData(subjTextual);
        subjTextual.setEnabled(false);
        gci.nextButton((JButton) evt.getSource(), tabbedPane);
        tab2_edit.setEnabled(true);
        tab2_save.setEnabled(false);
    } catch (PanelDataException ex) {
        JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
            JOptionPane.ERROR_MESSAGE);
    }
}

private void tab1_editActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    subjNumerical.setEnabled(true);
    tab1_save.setEnabled(true);
}

private void tab1_saveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        gci.updateTableData(subjNumerical);
        subjNumerical.setEnabled(false);
        gci.nextButton((JButton) evt.getSource(), tabbedPane);
        tab1_edit.setEnabled(true);
        tab1_save.setEnabled(false);
    } catch (PanelDataException ex){
        JOptionPane.showMessageDialog(this, ex.getMessage(), "Dialog",
            JOptionPane.ERROR_MESSAGE);
    }
}

private void tab7_closeActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

public void myInit(){
    System.out.println("inicio del wizzard");

    for(int i = 1; i < tabbedPane.getTabCount(); i++) tabbedPane.setEnabledAt(i, false);

    subjNumerical   = new SubjectQuestionsPanel(gci, QuestionType.type.NUMERICAL);
    subjTextual     = new SubjectQuestionsPanel(gci, QuestionType.type.TEXTUAL);
    evaluatedProf  = new SimpleQuestionsPanel(gci, QuestionType.type.SIMPLE);
    proffNumerical  = new ProffesorQuestionsPanel(gci, QuestionType.type.NUMERICAL);
    proffTextual    = new ProffesorQuestionsPanel(gci, QuestionType.type.TEXTUAL);
    resumen         = new Resumen(gci);
    graphsPanel2     = new GraphsPanel2(gci);
    wizard_tab1Panel.setLayout(new BorderLayout());
```

```
        wizard_tab2Panel.add(subjTextual,BorderLayout.NORTH);
        wizard_tab3Panel.setLayout(new BorderLayout());
        wizard_tab3Panel.add(evaluatedProf,BorderLayout.NORTH);
        wizard_tab4Panel.setLayout(new BorderLayout());
        wizard_tab4Panel.add(proffNumerical,BorderLayout.NORTH);
        wizard_tab5Panel.setLayout(new BorderLayout());
        wizard_tab5Panel.add(proffTextual,BorderLayout.NORTH);
        wizard_tab6Panel.setLayout(new BorderLayout());
        wizard_tab6Panel.add(resumen);
        wizard_tab7Panel.setLayout(new BorderLayout());
        wizard_tab7Panel.add(graphsPanel2);

        tab1_edit.setEnabled(false);
        tab2_edit.setEnabled(false);
        tab3_edit.setEnabled(false);
        tab4_edit.setEnabled(false);
        tab5_edit.setEnabled(false);
    }



    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    private javax.swing.JSeparator jSeparator4;
    private javax.swing.JSeparator jSeparator5;
    private javax.swing.JSeparator jSeparator6;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JButton tab1_edit;
    private javax.swing.JButton tab1_save;
    public javax.swing.JButton tab2_back;
    private javax.swing.JButton tab2_edit;
    private javax.swing.JButton tab2_save;
    private javax.swing.JButton tab3_back;
    private javax.swing.JButton tab3_edit;
    private javax.swing.JButton tab3_save;
    public javax.swing.JButton tab4_back;
    public javax.swing.JButton tab4_edit;
    public javax.swing.JButton tab4_save;
    private javax.swing.JButton tab5_back;
    private javax.swing.JButton tab5_edit;
    public javax.swing.JButton tab5_save;
    private javax.swing.JButton tab6_run;
    private javax.swing.JButton tab7_close;
    private javax.swing.JSeparator tab7_separator;
    private javax.swing.JTabbedPane tabbedPane;
    private javax.swing.JPanel wizard_tab1;
    private javax.swing.JPanel wizard_tab1Panel;
    private javax.swing.JPanel wizard_tab2Panel;
    private javax.swing.JPanel wizard_tab3Panel;
    private javax.swing.JPanel wizard_tab4Panel;
    private javax.swing.JPanel wizard_tab5Panel;
    private javax.swing.JPanel wizard_tab6;
    private javax.swing.JPanel wizard_tab6Panel;
    private javax.swing.JPanel wizard_tab7;
    private javax.swing.JPanel wizard_tab7Panel;
    // End of variables declaration
}
```

Clases relacionadas al Modelo

Anexo G. Professor:

La clase Professor almacena a un profesor como un nombre y una lista de preguntas de la encuesta con sus respectivas respuestas

```java
package GeneratorClasses;

import InterfaceClasses.QuestionType;
import java.util.ArrayList;
import java.util.stream.Collectors;

public class Professor {
    private String name;
    private ArrayList<Answers> answers;

    public Professor() {
        this.name = "";
        this.answers = new ArrayList<Answers>();
    }

    public Professor(String name){
        this.name = name;
        this.answers = new ArrayList<Answers>();
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public ArrayList<Answers> getAnswers() {
        return answers;
    }

    public void setAnswers(ArrayList<Answers> answers) {
        this.answers = answers;
    }

    public void addAnswer(String answer, int col){
        if(!this.answers.isEmpty() && col <= this.answers.size()){
            Answers answ = this.answers.stream().filter(ans -> ans.getColumn() == col).
                    findFirst().orElse(null);
            if(answ != null) answ.add(answer);
        }
    }

    public void addAnswers(Answers answers){
        this.answers.add(answers);
    }

    public ArrayList<Answers> getAnswers(QuestionType.type type){
        return this.answers.stream().filter(ans -> ans.getType() == type).
                collect(Collectors.toCollection(ArrayList::new));
    }

    @Override
    public String toString() {
        return "Proffesor{" + "name=" + name + ", answersLength=" + answers.toString() + '}';
    }

    public Integer getSurveyedAmount(){
        Integer maxAmount = null, tempAmount = 0;
        for(Answers numericalAnswers : getAnswers(QuestionType.type.NUMERICAL)){
            maxAmount = numericalAnswers.getAnswers().size();
            if(maxAmount > tempAmount) tempAmount = maxAmount;
        };
        return maxAmount;
    }

}
```

Anexo H. *Subject:*

La clase Subject representa a una asignatura como un conjunto compuesto por un nombre de asignatura, código, código de grupo de estudio, estudio al que pertenece, cantidad de alumnos inscritos en la asignatura y una lista de preguntas de la encuesta con sus respectivas respuestas.

```java
package GeneratorClasses;

import InterfaceClasses.QuestionType;
import java.util.ArrayList;
import java.util.stream.Collectors;

public class Subject {
    private ArrayList<Answers> answers;
    private String code;
    private String name;
    private String groupCode;
    private String estudio;
    private Integer enrolledStudents;

    public Subject(){
        this.answers = new ArrayList<Answers>();
        this.code        = "";
        this.name        = "";
        this.groupCode   = "";
        this.estudio     = "";
        this.enrolledStudents = 0;
    }

    public Subject(String code, String name, String groupCode) {
        this.code        = code;
        this.name        = name;
        this.groupCode   = groupCode;
        this.answers     = new ArrayList<Answers>();
    }

    public ArrayList<Answers> getAnswers() { return answers; }

    public void setAnswers(ArrayList<Answers> answers) { this.answers = answers; }

    public String getode() { return code; }

    public void setCode(String code) { this.code = code; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getGroupCode() { return groupCode; }

    public void setGroupCode(String groupCode) { this.groupCode = groupCode; }

    public String getEstudio() { return estudio; }

    public void setEstudio(String estudio) { this.estudio = estudio; }

    public Integer getEnrolledStudents() { return enrolledStudents; }

    public void setEnrolledStudents(Integer enrolledStudents) { this.enrolledStudents = enrolledStudents; }

    public void addAnswer(String answer, int col){
        if(!this.answers.isEmpty() && col <= this.answers.size()){
            Answers answ = this.answers.stream().filter(ans -> ans.getColumn() == col).
                    findFirst().orElse(null);
            if(answ != null) answ.add(answer);
        }
    }

    public void addAnswers(Answers answers){
        this.answers.add(answers);
    }

    public ArrayList<Answers> getAnswers(QuestionType.type type){
        return this.answers.stream().filter(ans -> ans.getType() == type).
                collect(Collectors.toCollection(ArrayList::new));
    }

    @Override
    public String toString() {
        return "Subject{" +
                "answersLengt=" + answers.size() + ", code=" + code +
                ", name=" + name + ", groupCode=" + groupCode + ", estudio=" +
                estudio + ", enrolledStudents=" + enrolledStudents + '}';
    }
}
```

Anexo I. *SubjectInformation*

En esta clase tiene como principal propósito obtener, desde los csv's, información sobre las asignaturas, información como el código de grupo de estudio, el grupo de estudio y la cantidad de alumnos inscritos por asignatura

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.util.Pair;

/**
 * @author ernesto
 * Store Subjects information from the CSV
 */
public class SubjectsInformation {
    Map<String, Pair<String, Integer>> subjectsInformation;
    String csvSeparator;
    String csvFilePath;
    int SUBJECT_COD_COLUMN      = 12;
    int SUBJECT_COD_MARK_COLUMN = 13;
    int SUBJECT_ENROLLED_COLUMN = 15;
    int SUBJECT_DEGREE_COLUMN   = 1;

    public SubjectsInformation(String csvSeparator, String csvFilePath) {
        this.subjectsInformation = new HashMap<String, Pair<String, Integer>>();
        this.csvSeparator = csvSeparator;
        this.csvFilePath = csvFilePath;
    }

    public void loadSubejctsInformation(){
        String line1 = "", line2, subCod, subCodMark, degree;
        String[] formatedLine;
        Integer enrolledStudents;
        try {
            File file = new File(csvFilePath);
            BufferedReader br = new BufferedReader(new FileReader(file));
            while((line2 = br.readLine()) != null){
                formatedLine = line1.split(csvSeparator);
                if(formatedLine.length > 0 && formatedLine.length >= SUBJECT_COD_MARK_COLUMN){
                    subCod = formatedLine[SUBJECT_COD_COLUMN];
                    subCodMark = formatedLine[SUBJECT_COD_MARK_COLUMN];
                    degree = formatedLine[SUBJECT_DEGREE_COLUMN];
                    if(subCod.contains("-") && subCodMark.contains("[")){formatedLine = line2.split(csvSeparator);
                        if(formatedLine.length > 0 &&
                                formatedLine.length >= SUBJECT_ENROLLED_COLUMN &&
                                formatedLine[SUBJECT_ENROLLED_COLUMN].matches("^\\d+$")){
                            enrolledStudents = Integer.parseInt(formatedLine[SUBJECT_ENROLLED_COLUMN]);
                            subCod = subCod.split("-")[0] + "-" + String.valueOf(Integer.valueOf(subCod.split("-")[1]
                            //removing the 0 pad in code group
                            subjectsInformation.put(subCod.trim(), new Pair(degree, enrolledStudents));
                        }
                    }
                }
                line1 = line2;
            }
            br.close();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Csv.class.getName()).log(Level.SEVERE, null, ex);
        } catch (NumberFormatException | IOException ex){
            Logger.getLogger(Csv.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public String getDegree(String cod, String group){
        if(!this.subjectsInformation.isEmpty() && cod != null && group != null){String code = cod + "-" + group;
            if(this.subjectsInformation.containsKey(code))
                return this.subjectsInformation.get(code).getKey();
        }
        return null;
    }

    public Integer getEnrolledStudents(String cod, String group){
        if(!this.subjectsInformation.isEmpty() && cod != null && group != null){String code = cod + "-" + group;
            if(this.subjectsInformation.containsKey(code))
                return this.subjectsInformation.get(code).getValue();
        }
        return null;
    }
}
```

Anexo J. *Report*

Se crea un reporte por cada combinación Profesor-Asignatura. Se crearán tantos reportes como profesores haya en una asignatura.

```java
package GeneratorClasses;

import java.util.ArrayList;

/**
 * @author ernesto
 */
public class Report {
    Subject subject;
    ArrayList<Professor> professors;

    public Report(){
        this.subject = null;
        this.professors = new ArrayList<>();
    }

    public Subject getSubject() {
        return subject;
    }

    public void setSubject(Subject subject) {
        this.subject = subject;
    }

    public ArrayList<Professor> getProfessors() {
        return professors;
    }

    public void setProfessors(ArrayList<Professor> professors) {
        this.professors = professors;
    }

    public void addProffesor(Professor proffesor){
        this.professors.add(proffesor);
    }

    @Override
    public String toString() {
        return "Report{" + "subject=" + subject + ", professors=" + professors + '}';
    }

}
```

Anexo K. *Answers*

Para una mejor gestión, las respuestas obtenidos desde las encuestas se guardan listas de Answers. La clase Answers esta compuesta por una pregunta, que tipo de pregunta es (numérica o de texto libre), la columna donde se encuentra dentro del documento Excel y una lista de respuestas a tal pregunta

```java
import InterfaceClasses.QuestionType;
import java.util.ArrayList;

/**
 * @author ernesto
 */
public class Answers{
    private QuestionType.type type;
    private String question;
    private Integer column;
    private ArrayList answers;

    public Answers(){
        this.answers    = new ArrayList<>();
        column          = null;
        question        = null;
        this.type       = QuestionType.type.SIMPLE;
    }

    public Answers(String question){
        this.answers = new ArrayList<>();
        this.question = question;
        this.type    = QuestionType.type.SIMPLE;
    }

    public Answers(QuestionType.type type, String question, Integer column) {
        this.answers = new ArrayList<>();
        this.type = type;
        this.question = question;
        this.column = column;
    }

    public QuestionType.type getType() {
        return type;
    }

    public Integer getColumn() {
        return column;
    }

    public void setColumn(Integer column) {
        this.column = column;
    }

    public void setType(QuestionType.type type) {
        this.type = type;
    }

    public void add(String obj){
        if(obj != null && obj.trim().length() > 0){
            if(this.type == QuestionType.type.NUMERICAL){
                try{
                    this.answers.add(Integer.valueOf(obj));
                }catch(NumberFormatException e){
                }
            }else{
                this.answers.add(obj);
            }
        }
    }

    public String getQuestion() {
        return question;
    }

    public void setQuestion(String question) {
        this.question = question;
    }

    public ArrayList getAnswers() {
        return answers;
    }

    public int size(){
        return this.answers.size();
    }

    public void setAnswers(ArrayList answers) {
        this.answers = answers;
    }
```

Anexo L. *NumericalAnswers*

La clase NumericalAnswers es una hereda todas las propiedades de la clase Answer además de tener datos numéricos como media, mediana, varianza y moda.

```java
package GeneratorClasses;

/**
 * @author ernesto
 */
public class NumericalAnswers extends Answers{
    private Float mean;
    private Float meanError;
    private Float variance;
    private Float median;


    public NumericalAnswers(){
        super();
        this.mean       = null;
        this.meanError  = null;
        this.median     = null;
        this.variance   = null;
    }

    public Float getMean() {
        return mean;
    }

    public void setMean(Float mean) {
        this.mean = mean;
    }

    public Float getMeanError() {
        return meanError;
    }

    public void setMeanError(Float meanError) {
        this.meanError = meanError;
    }

    public Float getVariance() {
        return variance;
    }

    public void setVariance(Float variance) {
        this.variance = variance;
    }

    public Float getMedian() {
        return median;
    }

    public void setMedian(Float median) {
        this.median = median;
    }

    @Override
    public String toString() {
        return "NumericalAnswers{" +
                "size:" + super.getAnswers().size() + ", mean=" + mean +
                ", meanError=" + meanError + ", variance=" + variance +
                ", median=" + median + '}';
    }

}
```

Clases relacionadas a la vista

Anexo M. *Questions*

Los datos sobre las preguntas de la encuesta son guardados en una clase llamada Questions que esta compuesta por una lista de preguntas, que tipo de pregunta es (numérica o de texto libre) y la categoría a la que pertenece (de asignatura o de profesor)

```java
package InterfaceClasses;

import java.util.ArrayList;

/**
 * @author ernesto
 */
public class Questions {
    public QuestionType.category category;
    public QuestionType.type type;
    public ArrayList<Question> questions;

    public Questions(){};

    public Questions(QuestionType.category category, QuestionType.type type){
        this.type = type;
        this.category = category;
        questions = new ArrayList<>();
    }

    public QuestionType.type getType() {
        return type;
    }

    public void setType(QuestionType.type type) {
        this.type = type;
    }

    public ArrayList<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(ArrayList<Question> questions) {
        this.questions = questions;
    }

    public QuestionType.category getCategory() {
        return category;
    }

    public void setCategory(QuestionType.category category) {
        this.category = category;
    }

    public void add(Question data){
        questions.add(data);
    }

    public int size(){
        return this.questions.size();
    }
}
```

Anexo N. *Table y ModelTable:*

Para gestionar de forma dinámica los inputs que serán ingresados, los inputs se representan como una tabla física (*Table*) y una lógica (*ModelTable*). *Table* es un observer de *ModelTable* y representa toda la información contenida en *ModelTable.*

```java
package InterfaceClasses;

import java.util.ArrayList;
import java.util.List;

/**
 * @author ernesto
 */
public class ModelTable {
    List<RowTable> rows = new ArrayList<RowTable>();
    int maxRows;
    boolean mustNotBeEmpty = false;
    List<Table> listeners = new ArrayList<Table>();

    public ModelTable(){};

    public List<RowTable> getRows() { return rows; }

    public void setRows(List<RowTable> rows) { this.rows = rows; }

    public int getMaxRows() { return maxRows; }

    public void setMaxRows(int maxRows) { this.maxRows = maxRows; }

    public boolean mustNotBeEmpty() { return mustNotBeEmpty; }

    public void setMustNotBeEmpty(boolean mustNotBeEmpty) { this.mustNotBeEmpty = mustNotBeEmpty; }

    public void addRowTable(RowTable row){
        if(rows == null) rows = new ArrayList<RowTable>();
        if(this.rows.size() < maxRows){
            rows.add(row);
        }
        notifyListeners();
    }


    public void removeRowTable(int i){
        if(i < rows.size()){
            if(mustNotBeEmpty){
                if(i>0){
                    rows.remove(i);
                }
            }else{
                rows.remove(i);
            }
        }
        notifyListeners();
    }


    public void removeLastRowTable(){
        if(!rows.isEmpty()){
            if(mustNotBeEmpty){
                if(rows.size()>1){
                    rows.remove(rows.size()-1);
                }
            }else{
                rows.remove(rows.size()-1);
            }
        }
        notifyListeners();
    }

    public boolean getMustNotBeEmpty() { return mustNotBeEmpty; }

    public void mustNotBeEmpty(boolean mustNotBeEmpty) { this.mustNotBeEmpty = mustNotBeEmpty; }

    public int getTotalRows(){ return rows.size(); }

    public void setEmpty(){ rows.clear(); }

    public void addListeners(Table table){ this.listeners.add(table); }

    private void notifyListeners() {
        for(Table table : listeners){
            table.updateTable(this);
        }
    }
}
```

```java
package InterfaceClasses;

import java.awt.Component;
import java.awt.GridBagConstraints;
import javax.swing.JPanel;

public class Table extends JPanel{

    ModelTable logicTable;

    public Table(ModelTable logicTable){
        logicTable.addListeners(this);
        updateTable(logicTable);
    };

    private void addComponents(RowTable row){
        int paneTotalComponents = getComponentCount();
        int rowTotalComponents  = row.getSize();

        int maxX = rowTotalComponents;
        int actualY = paneTotalComponents/rowTotalComponents;

        GridBagConstraints gbc  = new GridBagConstraints();
        gbc.fill     = GridBagConstraints.HORIZONTAL;

        for(int x = 0; x < maxX; x++){
            gbc.gridx    = x;
            gbc.gridy    = actualY;
            add((Component) row.getComponent(x), gbc);
        }
    }

    public void updateTable(ModelTable logicTable){
        this.removeAll();

        if(logicTable != null){
            if(logicTable.rows != null || !logicTable.getRows().isEmpty()){
                for(RowTable row : logicTable.getRows()){
                    addComponents(row);
                }
            }
        }
        this.revalidate();
        this.repaint();
    }

}
```

Anexo O. *RowTable*

La clase *ModelTable* esta compuesta por una lista de *RowTable*'s. Una *RowTable* es una lista de componentes gráficos, esto permite que la rowtable pueda contener diferente tipos de inputs.

```
package InterfaceClasses;

import java.awt.Component;
import java.awt.GridBagConstraints;
import javax.swing.JPanel;

public class Table extends JPanel{

    ModelTable logicTable;

    public Table(ModelTable logicTable){
        logicTable.addListeners(this);
        updateTable(logicTable);
    };

    private void addComponents(RowTable row){
        int paneTotalComponents = getComponentCount();
        int rowTotalComponents  = row.getSize();

        int maxX = rowTotalComponents;
        int actualY = paneTotalComponents/rowTotalComponents;

        GridBagConstraints gbc  = new GridBagConstraints();
        gbc.fill    = GridBagConstraints.HORIZONTAL;

        for(int x = 0; x < maxX; x++){
            gbc.gridx   = x;
            gbc.gridy   = actualY;
            add((Component) row.getComponent(x), gbc);
        }
    }

    public void updateTable(ModelTable logicTable){
        this.removeAll();

        if(logicTable != null){
            if(logicTable.rows != null || !logicTable.getRows().isEmpty()){
                for(RowTable row : logicTable.getRows()){
                    addComponents(row);
                }
            }
        }
        this.revalidate();
        this.repaint();
    }

}
```

Clases relacionadas a los reportes

Anexo P. *WordDocument:*

En esta clase se construye el documento Word que representa un reporte

```
import ...26 lines

/**
 * @author ernesto
 */
public class WordDocument {

    static WordprocessingMLPackage plantilla;
    private static ObjectFactory factory;
    private Subject subject;
    private Professor proffesor;
    private String archDestino;

    public WordDocument(Subject subject, Professor proffesor, String archDest) {
        this.subject = subject;
        this.proffesor = proffesor;
        archDestino = archDest;
    }

    public void generateDocument() throws FileNotFoundException, IOException, Exception {
        // BasicConfigurator.configure();
        plantilla = WordprocessingMLPackage.createPackage();

        // The image to add
        File uibLogo = new File(System.getProperty("user.dir") + "/imagenes/logoUib.png");
        addImage(uibLogo);

        int currentYear = Year.now().getValue();
        String estudio1 = subject.getEstudio();
        String title = "ENQUESTES " + estudio1 + "\n";
        String curso = (currentYear - 2000) + "-" + (currentYear -1999);
        String mainData = subject.getCode() + " " + subject.getName() + " (" + proffesor.getName() + ')';
        //adding current data
        addParagraph(title);
        addParagraph("Curs " + curso);
        addParagraph(mainData);

        //------tabla AlumnosMatriculados-------
        // agregar la tabla
        factory = Context.getWmlObjectFactory();
        plantilla.getMainDocumentPart().addStyledParagraphOfText("Subtitle", "Dades Estadístiques");
        Tbl table = factory.createTbl();
        //Row with enrolled students and surveyed students
        Tr tableRow0 = factory.createTr();
        addTableCell(tableRow0, Integer.toString(subject.getEnrolledStudents()) + " Alumnes matriculats");
        addTableCell(tableRow0, Integer.toString(proffesor.getSurveyedAmount()) + " Enquestes");
        table.getContent().add(tableRow0);
        plantilla.getMainDocumentPart().addObject(table);
        //------tablaAlumnosMatriculados fi------------
        addParagraph("");
        //------tabla Asignatura -----------------
        Tbl tableA = factory.createTbl();
        //fila con encabezados
        Tr tableRowA0 = factory.createTr();
        addStyledTableCell(tableRowA0, "ASSIGNATURA", true, null);

        tableA.getContent().add(tableRowA0);
        Tr tableRowA1 = factory.createTr();
        addTableCell(tableRowA1, "Valoració alumnes (1 a 10)");
        addTableCell(tableRowA1, "Mitjana");
        addTableCell(tableRowA1, "Mediana");
        addTableCell(tableRowA1, "Variança");
        addTableCell(tableRowA1, "Error Mitjana (±)");
        tableA.getContent().add(tableRowA1);

        DecimalFormat df = new DecimalFormat("#.##");
        for(Answers answers : subject.getAnswers(QuestionType.type.NUMERICAL)){
            Tr tableRow = factory.createTr();
            NumericalAnswers numericalAnswer = (NumericalAnswers) answers;
            addTableCell(tableRow, numericalAnswer.getQuestion());
            addTableCell(tableRow, String.valueOf(numericalAnswer.getMean()));
            addTableCell(tableRow, String.valueOf(numericalAnswer.getMean()));
            addTableCell(tableRow, String.valueOf(numericalAnswer.getVariance()));
            addTableCell(tableRow, String.valueOf(numericalAnswer.getMeanError()));
            tableA.getContent().add(tableRow);
        }

        plantilla.getMainDocumentPart().addObject(tableA);
        //------tabla Asignatura fi --------------
        addParagraph("");
```

```
    Tbl tableP = factory.createTbl();
    Tr tableRowP0 = factory.createTr();
    addStyledTableCell(tableRowP0, "PROFESSOR", true, null);
    tableP.getContent().add(tableRowP0);
    tableP.getContent().add(tableRowA1);

    for(Answers answers : profesor.getAnswers(QuestionType.type.NUMERICAL)){
        Tr tableRow = factory.createTr();
        NumericalAnswers numericalAnswer = (NumericalAnswers) answers;
        addTableCell(tableRow, numericalAnswer.getQuestion());
        addTableCell(tableRow, String.valueOf(numericalAnswer.getMean()));
        addTableCell(tableRow, String.valueOf(numericalAnswer.getMean()));
        addTableCell(tableRow, String.valueOf(numericalAnswer.getVariance()));
        addTableCell(tableRow, String.valueOf(numericalAnswer.getMeanError()));
        tableP.getContent().add(tableRow);
    }
    plantilla.getMainDocumentPart().addObject(tableP);
    //------tabla Profesor fi ---------------

    if (!subject.getAnswers(QuestionType.type.TEXTUAL).isEmpty()) {
        addParagraph("");
        plantilla.getMainDocumentPart().
                addStyledParagraphOfText("Subtitle", "Respostes de text lliure sobre l'assignatura");
        addTextualAnswers(subject.getAnswers(QuestionType.type.TEXTUAL));
        //0 = De Asignatura; 1 = De Profesor
    }
    if (!profesor.getAnswers(QuestionType.type.TEXTUAL).isEmpty()) {
        addParagraph("");
        plantilla.getMainDocumentPart().
                addStyledParagraphOfText("Subtitle", "Respostes de text lliure sobre l'professor");
        addTextualAnswers(profesor.getAnswers(QuestionType.type.TEXTUAL));
        //0 = De Asignatura; 1 = De Profesor
    }
    plantilla.save(new java.io.File(
            archDestino + "/" + subject.getCode() + "_" + subject.getGroupCode() +
                    "_" + profesor.getName() + ".docx"));
}

private void addImage(File file) throws FileNotFoundException, IOException, Exception {
    java.io.InputStream is = new java.io.FileInputStream(file);
    long length = file.length();
    if (length > Integer.MAX_VALUE) {
        System.out.println("File too large!!");
    }
    byte[] bytes = new byte[(int) length];
    int offset = 0;
    int numRead = 0;
    while (offset < bytes.length
            && (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0) {
        offset += numRead;
    }
    // Ensure all the bytes have been read in
    if (offset < bytes.length) {
        System.out.println("Could not completely read file " + file.getName());
    }
    is.close();

    String filenameHint = null;
    String altText = null;
    int id1 = 0;
    int id2 = 1;

Image 2: width 9000
    org.docx4j.wml.P p2 = newImage(plantilla, bytes,
            filenameHint, altText,
            id1, id2, 9000);
    plantilla.getMainDocumentPart().addObject(p2);

}

public static org.docx4j.wml.P newImage(WordprocessingMLPackage wordMLPackage,
        byte[] bytes,
        String filenameHint, String altText,
        int id1, int id2, long cx) throws Exception {

    BinaryPartAbstractImage imagePart = BinaryPartAbstractImage.createImagePart(wordMLPackage, bytes);

    Inline inline = imagePart.createImageInline(filenameHint, altText,
            id1, id2, false);
```

```
        p.getContent().add(run);
        org.docx4j.wml.Drawing drawing = factory.createDrawing();
        run.getContent().add(drawing);
        drawing.getAnchorOrInline().add(inline);
        return p;
    }

    private static void addParagraph(String textToAdd) {
        ObjectFactory obj = new ObjectFactory();
        P paragraph = obj.createP();
        R run = obj.createR();
        Text text = obj.createText();
        text.setValue(textToAdd);
        run.getContent().add(text);
        paragraph.getContent().add(run);
        plantilla.getMainDocumentPart().addObject(paragraph);
    }

    private static void addTableCell(Tr tableRow, String content) {
        Tc tableCell = factory.createTc();
        tableCell.getContent().add(
                plantilla.getMainDocumentPart().createParagraphOfText(content));
        tableRow.getContent().add(tableCell);
    }

    private static void addStyledTableCell(Tr tableRow, String content,
            boolean bold, String fontSize) {
        Tc tableCell = factory.createTc();
        addStyling(tableCell, content, bold, fontSize);
        tableRow.getContent().add(tableCell);
    }

    private static void addStyling(Tc tableCell, String content,
            boolean bold, String fontSize) {
        P paragraph = factory.createP();
        Text text = factory.createText();
        text.setValue(content);
        R run = factory.createR();
        run.getContent().add(text);
        paragraph.getContent().add(run);

        RPr runProperties = factory.createRPr();
        if (bold) { addBoldStyle(runProperties); }
        if (fontSize != null && !fontSize.isEmpty()) { setFontSize(runProperties, fontSize);}
        run.setRPr(runProperties);
        tableCell.getContent().add(paragraph);
    }

    private static void setFontSize(RPr runProperties, String fontSize) {
        HpsMeasure size = new HpsMeasure();
        size.setVal(new BigInteger(fontSize));
        runProperties.setSz(size);
        runProperties.setSzCs(size);
    }

    private static void addBoldStyle(RPr runProperties) {
        BooleanDefaultTrue b = new BooleanDefaultTrue();
        b.setVal(true);
        runProperties.setB(b);
    }

    private void addTextualAnswers(ArrayList<Answers> answers) {
        for(Answers textualAnswer : answers){
            addParagraph(textualAnswer.getQuestion());
            for(Object answer : textualAnswer.getAnswers()){
                String strAnswer = (String) answer;
                if (strAnswer.startsWith("L-")) strAnswer = strAnswer.substring(2);
                addParagraph("\t" + "- " + strAnswer);
            }
        }
    }
```

Anexo Q. *CSV:*

Es una clase cuyo objetivo es el de poder recorrer el documento csv que contiene la información de las asignaturas y grados.

```java
package documentClasses;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * @author ernesto
 */
public class Csv {
    ArrayList<String[]> csv = null;
    String csvSeparator;
    int totalColumns;
    int totalRows;

    public Csv(){}

    public Csv(File file, String csvSeparator){
        this.csvSeparator   = csvSeparator;
        totalColumns        = 0;
        totalRows           = 0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(file));
            csv = new ArrayList<String[]>();
            csv = loadCsv(br);
            totalColumns    = csv.get(0).length;
            totalRows       = csv.size();

        } catch (FileNotFoundException ex) {
            Logger.getLogger(Csv.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private ArrayList<String[]> loadCsv(BufferedReader br){
        String line;
        String[] formatedLine;
        ArrayList<String[]> csv = new ArrayList<>();
        try {
            while((line = br.readLine()) != null){
                formatedLine = line.split(csvSeparator);
                csv.add(formatedLine);
            }
        } catch (IOException ex) {
            Logger.getLogger(Csv.class.getName()).log(Level.SEVERE, null, ex);
        }
        return csv;
    };

    public String getValue(int column, int row){
        String value = null;
        String[] line;
        if(!csv.isEmpty() || !(totalColumns < column) || !(totalRows < row)){
            line    = csv.get(row);
            value   = line[column];
        }
        return value;
    }

    public int getTotalColumns() {
        return totalColumns;
    }

    public int getTotalRows() {
        return totalRows;
    }

}
```