Extractor de prototipos de comportamiento o perfiles

Documentación

Versión 1.0

Proyectos de Programación QT 2025-26

Grupo 22-4

Hadeer Abbas Khalil Wysocka - hadeer.abbas.khalil

Yimin Jin - yimin.jin

Sergi Malaguilla Bombín - sergi.malaguilla

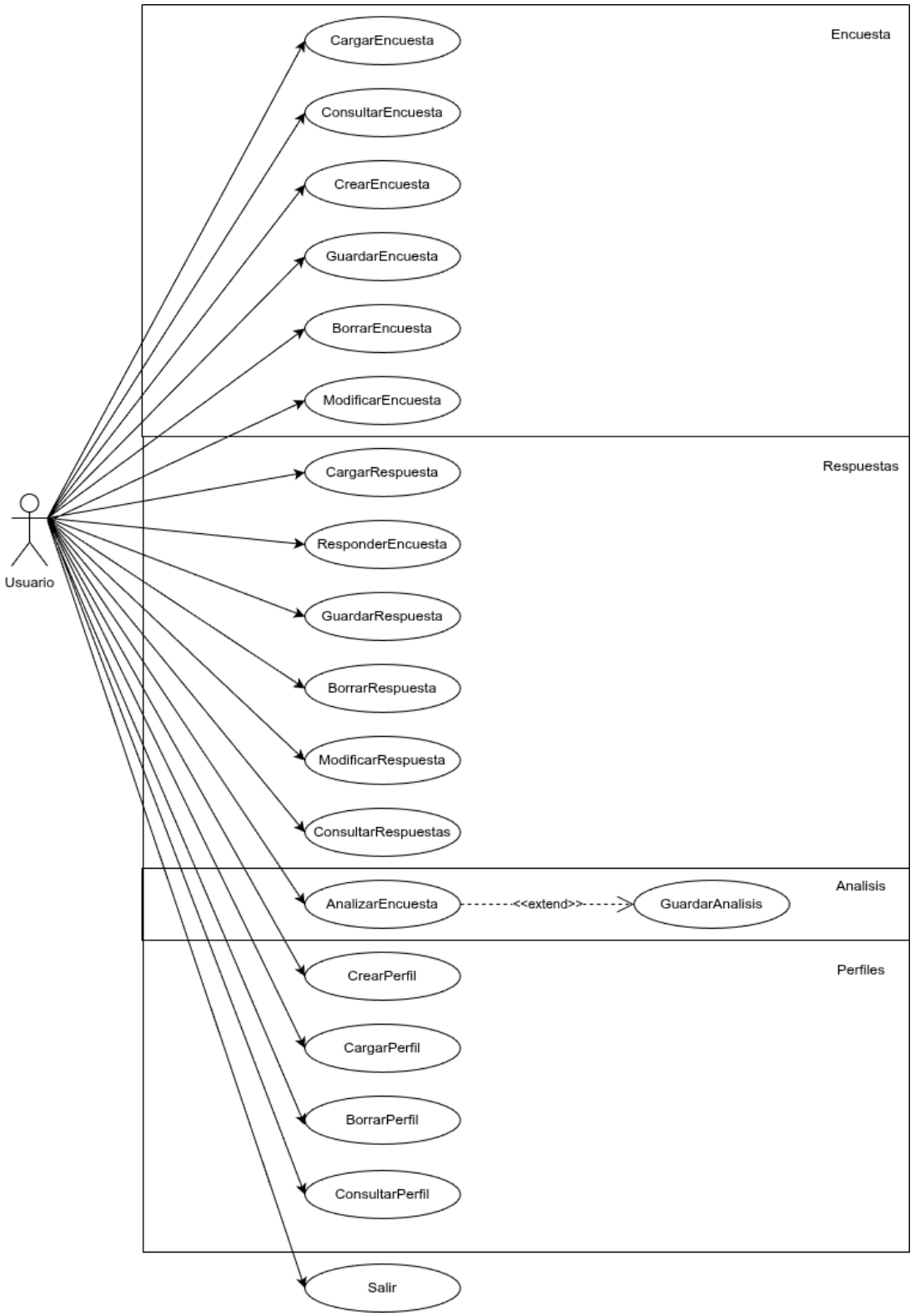
Javier Zhangpan - javier.zhangpan

Andrés Lafuente Patau - andres.lafuente

Índice

1. Casos de Uso

NOTA: Algunos aspectos de algunos casos de uso, no están completamente implementados para la primera entrega, se tendrán en cuenta y se completarán sus implementaciones en versiones posteriores.



1.1 Encuesta

- **Nombre:** Cargar encuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona la opción “cargar encuesta” en el menú.
    2. El sistema pide un fichero .txt de la encuesta.
    3. El usuario selecciona el fichero guardado en su ordenador.
    4. El sistema revisa el fichero proporcionado y lo valida .
    5. Si es correcto, carga la encuesta importada.
- **Errores posibles y cursos alternativos:**
    1. El fichero no existe o no se puede abrir.
    2. El formato no es válido.
    3. Ya está cargada la misma encuesta.

- **Nombre:** Consultar encuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona la opción “cargar consultar encuesta actual” en el menú.
    2. El sistema comprueba que haya una encuesta cargada.
    3. Si la hay, el sistema recorre las preguntas de la encuesta en orden según el ID y muestra la información de dichas preguntas, su ID, su enunciado, su tipo, su obligatoriedad, y su respectiva información adicional según el tipo.
- **Errores posibles y cursos alternativos:**
    1. No hay ninguna encuesta cargada

- **Nombre:** Crear encuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona “crear encuesta” en el menú de encuestas
    2. El sistema muestra un formulario para añadir título y pasar a modificar las preguntas

3. El usuario puede introducir o no las preguntas (puede crearla vacía)
4. El sistema valida y guarda la encuesta

- **Errores posibles y cursos alternativos:**
  1. Título repetido en sus encuestas
  2. Encuesta sin título

- **Nombre:** Guardar encuesta
- **Actor:** Usuario
- **Comportamiento:**
  1. El usuario quiere guardar la encuesta cargada que ha modificado.
  2. El sistema valida el formato de la encuesta y sus preguntas.
  3. El sistema guarda la encuesta en el directorio de sus encuestas.
- **Errores posibles y cursos alternativos:**
  1. Si una pregunta tiene un formato inválido, el sistema informa del error.
  2. Si ya existe una encuesta con el mismo nombre y creada por el mismo usuario, el sistema informa del error.

- **Nombre:** Borrar encuesta
- **Actor:** Usuario
- **Comportamiento:**
  1. El usuario selecciona la encuesta que quiere borrar indicando su título o borrando la encuesta actual cargada.
  2. El sistema manda un aviso al usuario de “Estás seguro de que quieres borrar esta encuesta de forma permanente?”, esperando una confirmación de su decisión de borrado.
  3. El sistema borra la instancia de la encuesta y el fichero asociado.
- **Errores posibles y vías alternativas:**
  1. Si el usuario decide no confirmar su decisión de borrado, el sistema no borra la instancia de esa encuesta.
  2. Si hay un error intentando borrar el fichero, el sistema informa al usuario del error.

- **Nombre:** Modificar encuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona qué encuesta quiere modificar cargándola en el sistema y selecciona la operación que quiera realizar (crear una nueva, modificar una existente o borrarla).
    2. Si el usuario quiere crear una pregunta, escoge el formato de la pregunta y la opción de respuesta; y se añade esta nueva pregunta a la encuesta, en la última posición.
    3. Si el usuario quiere modificar una pregunta, escoge qué pregunta de la encuesta quiere modificar indicando su ID y puede: cambiar el texto del enunciado de la pregunta y/o de las opciones de respuesta, cambiar el orden de la pregunta, su obligatoriedad, o cambiar el formato de la pregunta (en este caso, se mantiene el enunciado de la pregunta, pero las opciones de respuesta se restablece al valor inicial).
    4. Si el usuario quiere borrar una pregunta, escoge la pregunta de la encuesta que quiere borrar indicando su ID, se elimina esta instancia de la pregunta asociada a la encuesta cargada y las preguntas restantes enumeradas posteriormente a la pregunta borrada avanzan a la posición previa modificando su ID.
- **Errores posibles y vías alternativas:**
    1. Si el usuario escoge crear pregunta y la encuesta ya tiene el límite de preguntas, el sistema informa al usuario del error.
    2. Si el usuario al querer añadir una nueva opción de respuesta mientras crea o modifica una pregunta y sobrepasa el límite de opciones de respuesta en las preguntas con el formato de múltiples alternativas, no se hace el cambio y el sistema avisa del error.
    3. Si el usuario escoge borrar pregunta y no hay preguntas en la encuesta, el sistema informa al usuario del error.
    4. Si cualquiera de las opciones se ejecuta indicando un ID inexistente, el sistema informa del error.

1.2 Respuesta

- **Nombre:** Cargar respuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona la opción “cargar respuesta” en el menú de una encuesta.
    2. El sistema pide un fichero .txt de la respuesta.
    3. El usuario selecciona el fichero guardado en su ordenador.
    4. El sistema revisa el fichero proporcionado y lo valida.
    5. Si es correcto, añade la respuesta importada a las respuestas de la encuesta.
- **Errores posibles y cursos alternativos:**
    1. El fichero no existe o no se puede abrir
    2. El formato no es válido
    3. Ya existe una respuesta de un mismo perfil en la encuesta
    4. La respuesta no corresponde a la encuesta

- **Nombre:** Responder encuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario responde a las preguntas de la encuesta que está cargada, y envía una confirmación de que su respuesta es definitiva.
    2. Se crea una instancia de la respuesta ligada a esa encuesta en el sistema.
- **Errores posibles y vías alternativas:**
    1. Si el usuario contesta en un formato no válido en las preguntas con respuesta libre o numéricas, el sistema informa al usuario del error y no se crea la respuesta.
    2. Si hay una respuesta previa ligada a la misma encuesta que se ha respondido ahora, se sobreescriben los datos de esa respuesta en vez de crear una nueva instancia de respuesta.

- **Nombre:** Guardar respuesta

- **Actor:** Usuario
- **Comportamiento:**
    1. El sistema guarda la respuesta asociándola a la pregunta y al usuario.
- **Errores posibles y cursos alternativos:**
    1. Si la pregunta es obligatoria y está vacía, el sistema informa del error y solicita rellenar el campo.
    2. Si el usuario no tiene permiso para responder, el sistema informa del error.

- **Nombre:** Borrar respuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona la respuesta que desea borrar.
    2. El usuario confirma el borrado.
    3. El sistema elimina y marca como eliminada la respuesta y actualiza el orden de las respuestas.
- **Errores posibles y cursos alternativos:**
    1. Si el usuario cancela en la confirmación, el caso termina sin cambios.
    2. Si es una pregunta sin respuesta, el sistema notificará el error.

- **Nombre:** Modificar respuesta
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario selecciona una respuesta que ha sido previamente guardada.
    2. El usuario edita los datos y confirma los cambios.
- **Errores posibles y cursos alternativos:**
    1. Si la respuesta no existe, salta error.

- **Nombre**: Consultar respuesta
- **Actor**: Usuario
- **Comportamiento**:
    1. El usuario selecciona un fichero respuesta.

2. El sistema valida el fichero y muestra su contenido de ese fichero por pantalla.

- **Errores posibles:**
    1. El fichero seleccionado no es un fichero de respuesta. El sistema informa al usuario del error y le permite reintentar o abandonar la operación.
- **Cursos alternativos:**
    1. El usuario inicia el caso de uso “CargarRespuesta” y selecciona ese fichero.

1.3 Análisis

- **Nombre**: Analizar encuesta
- **Actor**: Usuario
- **Comportamiento**:
    1. El usuario informa al sistema del parámetro k, el algoritmo que quiere usar, la estrategia de inicialización que se seguirá y el criterio de calidad.
    2. El sistema valida que la información sea correcta.
    3. El sistema ejecuta el algoritmo de clustering previamente seleccionado sobre las respuestas cargadas de una encuesta.
    4. El sistema al acabar muestra los resultados por pantalla y deja la opción al usuario de guardarlos.
- **Errores posibles y cursos alternativos:**
    1. Si el usuario no introduce un parámetro k, se seleccionará automáticamente una k apropiada.
    2. El valor de k es incorrecto o el algoritmo seleccionado no existe, en ese caso se informa al usuario del error.
    3. El usuario interrumpe de alguna manera la ejecución del algoritmo de clustering, en ese caso el sistema informa del error y permite al usuario reintentar o abandonar la tarea.
    4. Si no hay respuestas cargadas de forma correcta en el sistema, el sistema informa del error al usuario.
    5. El usuario empieza el caso de uso “GuardarAnálisis”.

- **Nombre**: Guardar análisis
- **Actor**: Usuario
- **Comportamiento**:
    1. El usuario informa al sistema donde desea guardar los resultados del análisis.
    2. El sistema valida que el path sea correcto.
    3. El sistema guarda los resultados en el path correspondiente.
- **Errores posibles y cursos alternativos:**
    1. El path que el usuario ha designado no es válido, en ese caso el sistema informa del error y permite al usuario reintentar o abandonar la tarea.
    2. El usuario interrumpe de alguna manera el proceso de guardado, en ese caso el sistema informa del error y permite al usuario reintentar o abandonar la tarea.

1.4 Perfil

- **Nombre:** Crear perfil
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario informa de los datos del perfil, escribe la contraseña y confirma la contraseña.
    2. El sistema valida los valores y crea el perfil.
- **Errores posibles y cursos alternativos:**
- Si existe un perfil con el mismo identificador (email), el sistema informa del error.
- La contraseña y la confirmación no coinciden, el sistema informa del error.

- **Nombre:** Cargar perfil
- **Actor:** Usuario
- **Comportamiento:**

1. El usuario escribe todos los datos del perfil o selecciona el perfil que quiere cargar y escribe la contraseña.
2. El sistema valida los datos e inicia sesión con este perfil.

- **Errores posibles y cursos alternativos:**
    1. El usuario decide introducir todos los datos del perfil pero no existe ningún perfil con los datos introducidos, el sistema informa del error.
    2. Si la contraseña introducida no es correcta, el sistema informa del error.

- **Nombre:** Borrar perfil
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario que tiene su perfil asociado cargado en el sistema decide que borre ese perfil.
    2. El sistema borra la instancia del perfil cargado del sistema y también borra el fichero asociado al perfil que se ha borrado.
- **Errores posibles y cursos alternativos:**
    1. Si hay un error intentando borrar el fichero, el sistema deshace la transacción e informa al usuario del error.

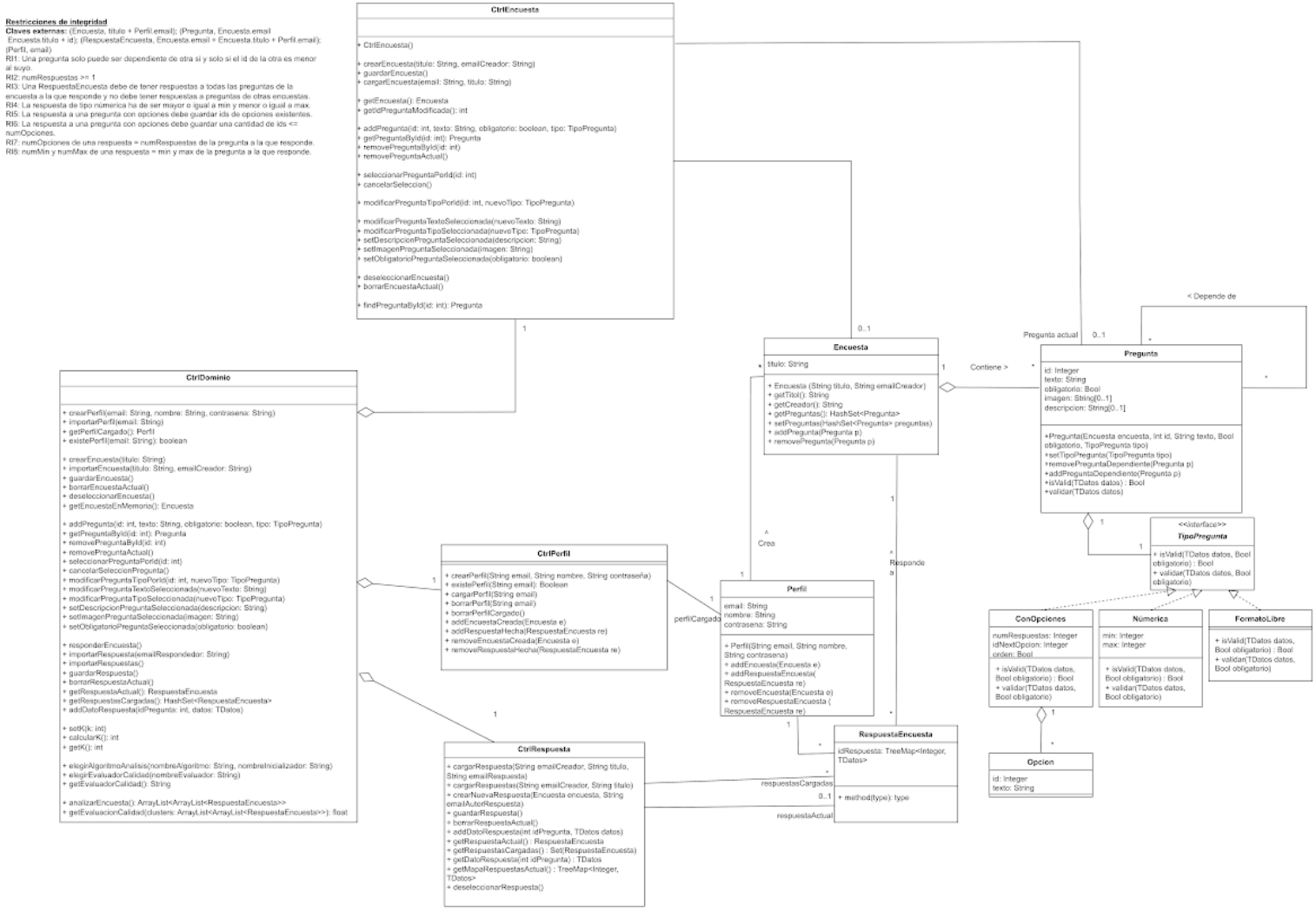
- **Nombre:** Consultar perfil
- **Actor:** Usuario
- **Comportamiento:**
    1. El usuario indica al sistema que quiere consultar los datos del perfil cargado.
    2. El sistema devuelve los datos, el email y el nombre del perfil cargado y se los muestra al usuario.
- **Errores posibles y cursos alternativos:**
    1. Si no hay un perfil cargado, el sistema informa al usuario del error.

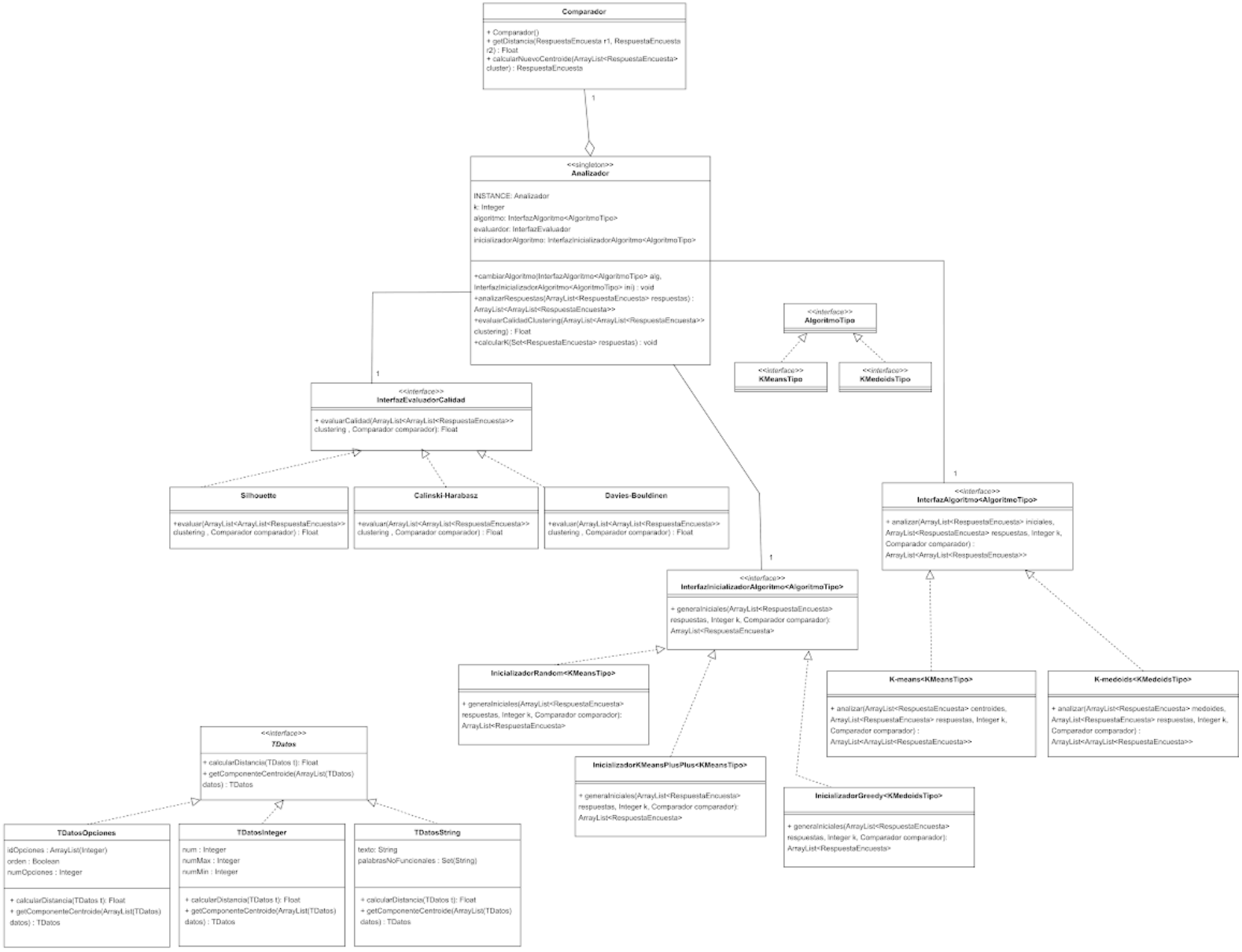
1.5 Salir

- **Nombre**: Salir
- **Actor**: Usuario
- **Comportamiento**:
    1. El usuario indica al sistema que quiere cerrar el programa.
    2. El sistema pregunta al usuario de nuevo si quiere cerrar el programa.
    3. El usuario confirma su decisión y cierra el programa.
- **Errores posibles y cursos alternativos:**
    1. El usuario niega su decisión y el programa permanece abierto en el mismo estado.

2. Diagrama del modelo conceptual de la capa de Dominio

Adjuntamos el diagrama en la carpeta DOCS. Hemos obviado los getters y setters.




Comparador
+ Comparador()
+ getDistancia(RespuestaEncuesta r1, RespuestaEncuesta r2) : Float
+ calcularNuevoCentroide(ArrayList<RespuestaEncuesta> cluster) : RespuestaEncuesta
1
<<singleton>>
Analizador
INSTANCE: Analizador
k: Integer
algoritmo: InterfazAlgoritmo<AlgoritmoTipo>
evaluador: InterfazEvaluador
inicializadorAlgoritmo: InterfazInicializadorAlgoritmo<AlgoritmoTipo>
+cambiarAlgoritmo(InterfazAlgoritmo<AlgoritmoTipo> alg, InterfazInicializadorAlgoritmo<AlgoritmoTipo> ini) : void
+analizarRespuestas(ArrayList<RespuestaEncuesta> respuestas) : ArrayList<ArrayList<RespuestaEncuesta>>
+evaluarCalidadClustering(ArrayList<ArrayList<RespuestaEncuesta>> clustering) : Float
+calcularK(Set<RespuestaEncuesta> respuestas) : void
1
<<interface>>
AlgoritmoTipo
<<interface>>
KMeansTipo
<<interface>>
KMedoidsTipo
1
<<interface>>
InterfazEvaluadorCalidad
+ evaluarCalidad(ArrayList<ArrayList<RespuestaEncuesta>> clustering , Comparador comparador): Float
Silhouette
+evaluar(ArrayList<ArrayList<RespuestaEncuesta>> clustering , Comparador comparador) : Float
Calinski-Harabasz
+evaluar(ArrayList<ArrayList<RespuestaEncuesta>> clustering , Comparador comparador) : Float
Davies-Bouldinen
+evaluar(ArrayList<ArrayList<RespuestaEncuesta>> clustering , Comparador comparador) : Float
1
<<interface>>
InterfazAlgoritmo<AlgoritmoTipo>
+ analizar(ArrayList<RespuestaEncuesta> iniciales, ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador) : ArrayList<ArrayList<RespuestaEncuesta>>
1
<<interface>>
InterfazInicializadorAlgoritmo<AlgoritmoTipo>
+ generaIniciales(ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador): ArrayList<RespuestaEncuesta>
InicializadorRandom<KMeansTipo>
+ generaIniciales(ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador): ArrayList<RespuestaEncuesta>
InicializadorKMeansPlusPlus<KMeansTipo>
+ generaIniciales(ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador): ArrayList<RespuestaEncuesta>
InicializadorGreedy<KMedoidsTipo>
+ generaIniciales(ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador): ArrayList<RespuestaEncuesta>
K-means<KMeansTipo>
+ analizar(ArrayList<RespuestaEncuesta> centroides, ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador) : ArrayList<ArrayList<RespuestaEncuesta>>
K-medoids<KMedoidsTipo>
+ analizar(ArrayList<RespuestaEncuesta> medoides, ArrayList<RespuestaEncuesta> respuestas, Integer k, Comparador comparador) : ArrayList<ArrayList<RespuestaEncuesta>>
<<interface>>
TDatos
+ calcularDistancia(TDatos t): Float
+ getComponenteCentroide(ArrayList(TDatos) datos) : TDatos
TDatosOpciones
idOpciones : ArrayList(Integer)
orden : Boolean
numOpciones : Integer
+ calcularDistancia(TDatos t): Float
+ getComponenteCentroide(ArrayList(TDatos) datos) : TDatos
TDatosInteger
num : Integer
numMax : Integer
numMin : Integer
+ calcularDistancia(TDatos t): Float
+ getComponenteCentroide(ArrayList(TDatos) datos) : TDatos
TDatosString
texto: String
palabrasNoFuncionales : Set(String)
+ calcularDistancia(TDatos t): Float
+ getComponenteCentroide(ArrayList(TDatos) datos) : TDatos

2.1 Descripción de las clases

Perfil

La clase Perfil representa a los usuarios de la aplicación de encuestas. Cada perfil tiene un nombre de usuario, contraseña y correo. El correo identifica cada perfil. El perfil también puede tener guardado sus encuestas creadas y sus respuestas a encuestas.

Encuesta

La clase Encuesta representa las encuestas. Cada encuesta es identificada por su título y su creador. Las encuestas guardan sus preguntas y las gestionan.

Pregunta

La clase Pregunta representa una pregunta de una encuesta. Cada pregunta almacena su identificador, texto y otros atributos como la obligatoriedad y si depende de otras preguntas. Cada pregunta tiene asociada un TipoPregunta, que contiene la información sobre el tipo de la pregunta.

RespuestaEncuesta

La clase RespuestaEncuesta representa una respuesta a una encuesta. Esta clase se identifica por el respondedor y la encuesta a la que responde. Cada RespuestaEncuesta guarda un TreeMap de TDatos, que son las respuestas a preguntas individuales de la encuesta.

Analizador

La clase Analizador es quien se encarga de toda la gestión de la algoritmia implicada en el clustering de las respuestas. Contiene: un comparador cuyo uso se emplea más adelante en los propios algoritmos, un evaluador de calidad (de diferentes tipos) que mediante un algoritmo determina un valor que indica su calidad, y un algoritmo de clustering (de dos tipos) que realiza la funcionalidad principal de analizar las respuestas y devolver los clusters resultantes, junto a su inicializador compatible correspondiente. Se le ha aplicado el patrón Singleton al

Analizador para poder instanciar el mismo objeto desde donde sea necesario y no tener que guardar diferentes instancias, ya que solo hace falta uno para realizar el análisis.

TipoPregunta

TipoPregunta es una interfaz que contiene los métodos isValid y validar. Esta interfaz la hemos creado aplicando el patrón estrategia para conseguir dos objetivos: poder modificar el tipo de la pregunta en tiempo de ejecución y tener diferentes estrategias de validación para cada tipo separado de la clase Pregunta.

ConOpciones

La clase ConOpciones es una implementación de TipoPregunta. Esta clase representa un tipo de múltiples opciones, guardando un ArrayList de Opcion. Esta clase se encarga de validar TDatosOpciones, implementando los métodos isValid y validar de TipoPregunta.

Numerica

La clase Numerica es una implementación de TipoPregunta. Esta clase representa un tipo de pregunta que acepta números enteros, guardando el rango de estos números. Esta clase se encarga de validar TDatosInteger, implementando los métodos isValid y validar de TipoPregunta.

FormatoLibre

La clase FormatoLibre es una implementación de TipoPregunta. Esta clase representa un tipo de pregunta con respuesta libre. Esta clase se encarga de validar TDatosString, implementando los métodos isValid y validar de TipoPregunta.

Opcion

La clase Opcion representa una opción en una pregunta de múltiples opciones. Esta clase almacena el identificador de la opción y su texto.

Comparador

La clase Comparador contiene el método para calcular la distancia entre dos RespuestaEncuesta y el método para calcular un nuevo centroide en un ArrayList de RespuestaEncuesta. Esta clase la hemos creado para no tener estas funcionalidades en la clase RespuestaEncuesta.

InterfazEvaluadorCalidad

InterfazEvaluadorCalidad es una interfaz que tiene el método para evaluar la calidad de un clustering. Esta interfaz la hemos creado aplicando el patrón estrategia, permitiendo modificar el algoritmo que el analizador debe utilizar en tiempo de ejecución y facilitar la creación de nuevos algoritmos.

Silhouette

La clase Silhouette implementa InterfazEvaluadorCalidad. Esta clase utiliza el Coeficiente de Silhouette para evaluar un clustering.

Calinski-Harabasz

La clase Calinski-Harabasz implementa InterfazEvaluadorCalidad. Esta clase utiliza el Índice de Calinski-Harabasz para evaluar un clustering.

Davies-Bouldinen

La clase Davies-Bouldinen implementa InterfazEvaluadorCalidad. Esta clase utiliza el Índice de Davies-Bouldin para evaluar un clustering.

TDatos

TDatos es una interfaz que hemos creado para abstraer el formato de las respuestas a preguntas, aplicando el patrón estrategia. Esta interfaz también tiene los métodos para calcular distancias entre dos TDatos y calcular el centroide de un ArrayList de TDatos. De esta manera, otras clases no necesitan conocer los TDatos concretos.

TDatosOpciones

La clase TDatosOpciones representa las opciones elegidas en una respuesta a una pregunta de tipo Opcion. La clase contiene atributos como “idOpciones” que contiene las opciones propiamente escogidas, “orden” para saber si las opciones tienen un orden asociado entre ellas y “numOpciones” que indica el número total de opciones que había en la pregunta. La clase también ofrece métodos para comparar, calcular distancias y calcular centroides de datos de este tipo.

TDatosInteger

La clase TDatosInteger representa el número entero escrito en una respuesta a una pregunta de tipo Numerica. La clase contiene atributos como “num”, que es la propia respuesta y “numMax” y “numMin” que son los valores máximos permitidos por la pregunta. La clase también ofrece métodos para comparar, calcular distancias y calcular centroides de datos de este tipo.

TDatosString

La clase TDatosString representa un texto escrito en una respuesta a una pregunta de tipo FormatoLibre. La clase contiene atributos como “texto”, que es la propia respuesta y “palabrasNoFuncionales” que es un listado de palabras sin significado semántico propio de la lengua española. La clase también ofrece métodos para comparar, calcular distancias y calcular centroides de datos de este tipo.

AlgoritmoTipo

AlgoritmoTipo es una interfaz de marcador cuyo propósito es diferenciar los inicializadores exclusivos a ciertos algoritmos. Esta interfaz permite imposibilitar la creación de combinaciones que consideramos no aptas para su uso y mejora la legibilidad del código fuente.

InterfazInicializador<AlgoritmoTipo>

InterfazInicializador es una interfaz que tiene el método para inicializar los centroides. Esta interfaz la hemos creado aplicando el patrón estrategia, permitiendo modificar el algoritmo que el analizador debe utilizar en tiempo de ejecución y facilitar la creación de nuevos algoritmos de inicialización.

InicializadorKMeansPlusPlus<KMeansTipo>

La clase InicializadorKMeansPlusPlus implementa un inicializador del algoritmo K-Means que sigue el algoritmo de K-Means++. La clase solamente contiene un atributo de la clase Random presente en las librerías de Java para generar números aleatorios. La clase ofrece un método para generar centroides iniciales dadas unas respuestas.

InicializadorRandom<KMeansTipo>

La clase InicializadorRandom implementa un inicializador del algoritmo K-Means que sigue un algoritmo trivial aleatorio. La clase solamente contiene un atributo de la clase Random presente en las librerías de Java para generar números aleatorios. La clase ofrece un método para generar centroides iniciales dadas unas respuestas.

InicializadorGreedy<KMedoidsTipo>

La clase InicializadorGreedy implementa un inicializador para el algoritmo k-medoids, basado en la estrategia Greedy. Su función es seleccionar los k medoids de forma determinista a través de la minimización de las distancias entre respuestas y los candidatos a medoid para elegir los medoids iniciales.

InterfazAlgoritmo<AlgoritmoTipo>

InterfazAlgoritmo es una interfaz que tiene el método para analizar un ArrayList de RespuestaEncuesta. Esta interfaz la hemos creado aplicando el patrón estrategia, permitiendo modificar el algoritmo que el analizador debe utilizar en tiempo de ejecución y facilitar la creación de nuevos algoritmos.

KMeans<KMeansTipo>

La clase KMeans implementa el algoritmo de clustering “naïve” de K-Means, también llamado algoritmo de Lloyd. La clase no contiene ningún atributo y simplemente ofrece un método que clasifica las respuestas dadas en K diferentes clusters. Se ha de tener en cuenta que es posible que se devuelvan clusters vacíos

en casos extremos, pero se asegura que todas respuestas formarán parte de algún cluster.

KMedoids<KMedoidsTipo>

La clase KMedoids implementa el algoritmo alternativo de clustering de las respuestas de una misma encuesta, recibe como parámetro unos medoids predeterminados/inicializados por el algoritmo greedy que son un subconjunto de las respuestas a las cuales queremos aplicar el clustering, que también pasamos como parámetro; junto a un entero k que indica el número de clusters que ha de devolver. El algoritmo devuelve un ArrayList de clusters que en si son ArrayLists de Respuestas.

CtrlEncuesta

La clase CtrlEncuesta implementa la gestión de las diferentes encuestas y mantiene una encuesta cargada posibilitando así la modificación de una encuesta cargada por el usuario. Se encarga de coordinar las operaciones sobre una encuesta, asegurando que todas las acciones sean válidas y mantengan la integridad de los datos. Las diferentes responsabilidades del controlador es la creación, carga, guardado, selección/deselección y modificación de una encuesta, esta última ofrece la posibilidad de añadir, eliminar y modificar preguntas individuales que el usuario puede ir seleccionando para acceder a cualquiera de ellas a través de su ID. También ofrece getters para que los diferentes drivers puedan consultar la diferente información de una encuesta y sus preguntas.

CtrlPerfil

La clase CtrlPerfil se encarga de hacer toda la gestión relacionada con los casos de uso donde se involucran los perfiles, contiene un único perfil que indica el perfil que está cargado en el sistema, permitiendo así que el usuario pueda responder y crear encuestas con su perfil asociado. Aporta funcionalidades para crear, cargar, guardar perfiles y gestionar las asociaciones que se generan después de crear y/o responder encuestas (precisamente la que está cargada en el CtrlEncuesta).

CtrlRespuesta

La clase CtrlRespuesta es la responsable de la gestión de todas las operaciones relacionadas con las respuestas de la clase RespuestaEncuesta. Contiene una respuesta que indica la respuesta actual que se está modificando y un conjunto de respuestas para que se pueda realizar el análisis. Aporta métodos para crear, cargar, guardar y borrar respuestas, y añadir los datos adecuados correspondientes a la pregunta de la encuesta cargada en el CtrlEncuesta en la respuesta actual cargada.

CtrlDominio

Se trata del controlador principal de la capa de dominio y su función es coordinar las operaciones relacionadas con la gestión de encuestas, perfiles, respuestas y análisis. Actúa como intermediario entre la “interfaz” y el dominio. Ofrece un punto de entrada unificado para implementar los casos de uso delegando las diferentes implementaciones a los controladores específicos CtrlEncuesta, CtrlPerfil, CtrlRespuesta y el singleton Analizador, sin implementar lógica compleja a sí mismo. Éste controlador desarrolla las funcionalidades principales que serían:

1. Gestión de encuestas: (delegado a CtrlEncuesta)
    1.1 Creación de encuestas
    1.2 Cargar/Guardar/Borrar encuesta
    1.3 Consultar encuesta
    1.4 Modificación de una encuesta
        1.4.1 Añadir preguntas
        1.4.2 Eliminar preguntas
        1.4.3 Modificar pregunta (tipo, obligatoriedad, texto,...)
    1.5 Seleccionar encuesta o pregunta para su respectiva modificación

2. Gestión de Perfiles (delegado a CtrlPerfil)
    2.1 Crear Perfil
    2.2 Cargar un Perfil existente
    2.3 Comprobar si existe un Perfil
    2.4 Asociar respuestas y encuestas creadas a un Perfil

3. Gestión de Respuestas (delegado a CtrlRespuesta)
    3.1 Crear/Guardar/Cargar/Borrar una Respuesta
    3.2 Añadir los datos de una Respuesta a Pregunta
    3.3 Consultar la Respuesta Actual

4. Análisis de Datos (delegado a Analizador)
    4.1 Elección de algoritmo de clustering
    4.2 Elección de Inicializador de algoritmo
    4.3 Elección de evaluador de calidad
    4.4 Elección del valor k
    4.5 Ejecución de análisis
    4.6 Obtención de la evaluación

En resumen, se trata de una clase que delega en los controladores especializados y realiza validaciones globales actuando como orquestador de la lógica del dominio.

3. Relación entre clases

En este apartado, indicamos en la siguiente tabla las clases implementadas por cada miembro de grupo. Los tests, de forma general (salvo TestCtrlRespuesta hecho por Hadeer), cada uno que ha implementado su clase también ha implementado su test correspondiente.

Hadeer	Yimin	Sergi	Javier	Andrés
Perfil	Pregunta	Encuesta	Analizador	RespuestaEncuesta
KMedoids	ConOpciones	InicializadorGreedy	TDatosOpciones	CtrlRespuesta
GestorPerfil	Opcion	GestorEncuesta	TDatosInteger	
GestorRespuestaEncuesta	Numerica	CtrlEncuesta	TDatosString	
CtrlPerfil	FormatoLibre	CtrlDominio	InicializadorKMeansPlusPlus	
CtrlPersistencia	Comparador	DriverGeneral	InicializadorRandom	
	Silhouette		KMeans	
	Calinski-Harabasz			
	Davies-Bouldinen			
	DriverEncuesta			

Tipo
Clase
Controlador
Driver

4. Estructuras de datos y algoritmos usados

4.1 Algoritmos

4.1.1 K-Means

El algoritmo de K-Means consiste en particionar el conjunto de respuestas de la encuesta en K clusters de manera que se minimice la varianza interna de los clusters. En este caso, la varianza se define como el sumatorio de las distancias al cuadrado entre los puntos de un cluster a su centroide. Algunas de las ventajas de usar esta métrica son: evitar calcular raíces cuadradas y el hecho de minimizar la suma de distancias cuadradas implica la minimización de la suma de distancias.

El problema a resolver se ha comprobado que está en la clase NP-Hard en general, por lo que se usan algoritmos heurísticos como el de Lloyd que usan búsqueda local en vez de fuerza bruta. No obstante, se ha podido comprobar experimentalmente que estas implementaciones siguen teniendo una complejidad superpolinómica, más concretamente, la complejidad del algoritmo de Lloyd es $O(nkdi)$. Donde n es el número de respuestas, k es el número de clusters, d es la dimensionalidad de cada respuesta, es decir, cuántas respuestas a preguntas individuales tiene esa respuesta, e i es el número de iteraciones hasta que el algoritmo converge, que se ha demostrado que es $2^{\Omega(\sqrt{n})}$.

El propio algoritmo de Lloyd consiste en inicializar K centroides, asignar cada respuesta al cluster cuyo centroide sea el más cercano a la respuesta, donde la noción de distancia es la distancia cuadrada, recalcular los centroides para cada cluster y repetir este proceso hasta que converja (en este caso cuando los centroides recalculados sean idénticos a los centroides originales).

Cabe destacar que en nuestra implementación admitimos la inclusión de clusters vacíos dentro de la solución final. Esto puede ser causado si se usa inicialización aleatoria de los centroides iniciales y 2 de ellos son respuestas idénticas. Consideramos que esta situación es suficientemente rara, sobre todo a medida que

crece n, que en esta implementación básica no hace falta tratarla de forma excepcional.

Las responsabilidades de generación de centroides iniciales, cálculo de distancias y recálculo de centroides se dejan a clases externas a la clase que implementa el propio algoritmo. Por lo que la implementación simplemente sigue el procedimiento explicado anteriormente.

Por último, destacamos la posibilidad de mejorar esta implementación a través de como precálculo de distancias, uso de cache o explotar la propiedad de desigualdad triangular de las distancias (no cuadradas) para ahorrar cómputo.

4.1.2 Inicialización Random

Los métodos de inicialización de centroides son muy importantes a la hora de determinar su velocidad de convergencia. Una inicialización buena permite reducir el factor exponencial del número de iteraciones del algoritmo de forma considerable.

Una implementación trivial es simplemente seleccionar de forma aleatoria K respuestas del conjunto para que actúen como centroides iniciales. Como se ha descrito anteriormente, esto conlleva ciertos inconvenientes en casos donde se seleccionan 2 respuestas idénticas como centroides. Aparte de eso, cabe destacar que es crucial usar un generador de números aleatorios que siga una distribución uniforme para acelerar la convergencia.

En nuestra implementación, simplemente se generan K números aleatorios usando una distribución uniforme entre 0 (inclusivo) y K (exclusivo) de manera que no se repita ninguno (se asegura usando un HashSet). A continuación, se devuelven los centroides que son las respuestas cuyo índice coincide con el del número generado. Por lo que la complejidad temporal es $O(k)$, siendo k el número de clusters deseados.

4.1.3 K-Means++

K-Means++ es otro método de inicialización que se especializa en encontrar centroides iniciales decentemente buenos para el algoritmo de K-Means de forma voraz. Más concretamente, el algoritmo asegura que la solución de clustering resultante después de usar K-Means++ es $O(log\ k)$ competitivo con la solución óptima global.

Este algoritmo soluciona un problema bastante importante de la inicialización aleatoria que es su pobre fiabilidad, ya que se pueden generar centroides arbitrariamente malos, que extiendan innecesariamente el tiempo de convergencia y/o produzcan clusterings lejanos al óptimo. Además, si se añade alguna componente estocástica a la implementación de K-Means++, entonces se puede argumentar que tampoco se pierde mucho acceso a las regiones del espacio de soluciones que nos interesan.

K-Means++ consiste en primeramente elegir un centroide de forma aleatoria siguiendo una distribución uniforme, en segundo lugar, calcular las distancias cuadradas de cada respuesta a ese centroide, en tercer lugar, elegir un nuevo centroide de forma aleatoria pero siguiendo una probabilidad con pesos proporcionales a la distancia cuadrada calculada anteriormente y, finalmente, repetir los pasos 2 y 3 hasta elegir K centroides.

Como se puede deducir de la explicación anterior, este algoritmo intenta asegurar que los centroides estén relativamente separados entre ellos, lo cual acelera el proceso de convergencia del algoritmo K-Means. Además es interesante destacar que este método asegura que se escoge la misma respuesta como centroide porque la distancia entre la respuesta y el centroide es 0, por lo que en consecuencia, la probabilidad de escoger esa respuesta también es 0.

En nuestra implementación, cabe destacar que la manera en la que se implementa la elección de centroide con probabilidad proporcional a la distancia cuadrada es mediante definir un umbral aleatorio entro 0 (inclusivo) y el sumatorio de distancias cuadradas total (exclusivo). Entonces, se escoge como centroide la primera

respuesta cuya contribución a la suma acumulada de distancias cuadradas supere ese umbral.

Es importante destacar que la implementación se podría optimizar de diversas maneras como por ejemplo manteniendo una caché o simplemente precalculando distancias.

4.1.4 K-Medoids

El problema de k-Medoids clustering, que tiene la misma finalidad que el k-Means de particionar respuestas en clusters, es clasificado como un problema NP-difícil, por tanto, la implementación que usamos para resolverlo aplica ciertas heurísticas para que se pueda ejecutar el algoritmo en tiempo polinómico de forma *greedy*, que nes más rápido que realizar una resolución exhaustiva, a pesar de que no resuelve el problema con los medoides óptimos para realizar el clustering.

El algoritmo empleado para resolver el clustering es el *Partitioning Around Medoids* (PAM), inventado por Leonard Kaufman y Peter J. Rosseeuw. Consiste en dos fases: una fase build que construye los medoides, del cual se encarga la inicialización Greedy (explicada más adelante), y la fase swap, cuyo funcionamiento és el siguiente, descrito en pseudocódigo:

$dado$: $X = \{x_1, \ldots, x_n\}\ puntos\ de\ datos,$

$k \in \mathbb{N},$

$M = \{Y \subseteq X \mid Y = greedy(X, k)\},$

$coste_{total}(M, X)$:

$c := 0$

$\forall x \in X$:

$dist_{min} := +\infty$

$\forall m \in M$:

$dist := \sum_{k=1}^{d} |x_k - m_k|$ (distancia entre x y m)

$si\ (dist < dist_{min})$:

$$dist_{min} := dist$$

$$c := c + dist_{min}$$

$devolver\ c$

$PAM_{SWAP}(M, X, k)$:

$converge := false$

$coste_{mejor} := coste_{total}(M, X)$

$while(\neg converge)$:

$converge := true$

$\forall x \in X$:

$\forall m \in M \mid m \neq x$:

$M := \{M \mid x \in M \wedge m \notin M\}$ (swap)

$coste := coste_{total}(M, X)$

$si\ (coste < coste_{mejor})$:

$coste_{mejor} := coste$

$converge := false$

$sino$:

$M := \{M \mid x \notin M \wedge m \in M\}$ (deshacer swap)

$devolver\ M$

El resumen básico del pseudocódigo es que calculamos el coste total, que es la distancia mínima total de cada respuesta con cada medoid del conjunto obtenido del inicializador Greedy; y luego, para cada respuesta que no sea un medoid, vamos intercambiándolo en el conjunto de medoids con cada medoid y recalculamos el coste para mejorar si es menor que el mejor coste total calculado hasta ahora; si el coste no mejora, quiere decir que el algoritmo ha convergido y el coste no se puede mejorar, devolviendo el conjunto de medoids en el cual luego con la función *construirClusters* vamos construyendo el conjunto de clusters, donde cada respuesta se la asigna al cluster que pertenece el medoide más cercano. El coste temporal de esta implementación básica del algoritmo, donde recalculamos todo el

rato el coste total, és de $O(n^2k^2)$, donde n es el número de elementos en el conjunto de respuesta.

4.1.5 Inicialización Greedy

El algoritmo Inicializador Greedy construye el conjunto inicial de medoids para k-medoids seleccionando el primer medoid de forma aleatoria e iterando hasta completar los k medoid, para cada respuesta candidata que no sea medoid se evalúa el resultado de incorporarla como medoid al conjunto actual. Esta evaluación consiste en calcular para cada respuesta del dataset la distancia mínima entre dicha respuesta y el conjunto de medoids con el medoid candidato. El medoid se selecciona si minimiza la suma total de las distancias, repitiendo este método hasta conseguir k medoids.

El método actúa como una heurística determinista (salvo la inicialización aleatoria) para aproximar una buena configuración inicial para el k-medoids.

4.1.6 Silhouette

El coeficiente de Silhouette es un método de interpretación y validación de la coherencia dentro del análisis de grupos.

El valor de Silhouette mide cuán similar es un punto a su propio cluster en comparación con otros clusters. Este valor va de -1 a +1, donde un valor más alto indica un mejor resultado.

Para cada punto i, Silhouette compara:

- a(i): la cohesión, es decir, la distancia promedio del punto a los demás puntos de su mismo clúster.
- b(i): la separación, o distancia promedio del punto al clúster vecino más cercano (aquel que minimiza esta distancia).
- Entonces, el coeficiente de Silhouette de un punto es $s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$

El coeficiente global será la media de todos los s(i).

Este algoritmo tiene un coste $O(n^2)$, y aunque exista el Silhouette simplificado con un coste menor, nosotros hemos decidido implementar el completo.

4.1.7 Calinski-Harabasz

El índice de Calinski-Harabasz compara la separación entre clusters (varianza entre grupos, BCSS) y la compacidad dentro de esos grupos (varianza interna, WCSS).

Esos valores se calculan así:

$$BCSS = \sum_{i=1}^{k} n_i \left|\left| c_i - c \right|\right|^2$$

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} \left|\left| x - c_i \right|\right|^2$$

La fórmula queda como $CH = \frac{BCSS/(k-1)}{WCSS/(n-k)}$, donde CH es mejor cuanto más alto sea su valor.

4.1.8 Davies-Bouldin

El índice de Davies-Bouldin mide cuán bien separados y cuán compactos son los clusters. Este índice indica un mejor clustering cuanto más cerca del 0 sea el resultado.

Para calcularlo, nosotros hemos hecho:

$$DB = \frac{1}{k} \sum_{i=1}^{k} R_i$$

donde:

$R_i = max_{j!=i} R_{ij}$ y $R_{ij} = \frac{S_i + S_j}{M_{ij}}$

4.2. Estructuras de Datos

En este apartado se describen y analizan las estructuras de datos más relevantes que se han usado en el proyecto.

4.2.1 HashSet

Es una estructura que guarda elementos únicos, donde no importa el orden el que son guardados. Evita duplicidad y se obtiene una búsqueda rápida con complejidad amortizada de $O(1)$ suponiendo que se usa una buena función de hash.

Sus usos más relevantes son en:

- Encuesta.java: Utiliza Set<Pregunta> preguntas = new HashSet<>();
- Perfil.java: Usa HashSet<String> para encuestasCreadas.
- CtrlDominio.java: Usa HashSet<RespuestaEncuesta> para cargar las respuestas antes de analizar.
- Pregunta.java: Usa HashSet<Pregunta> para guardar las preguntas dependientes.

Donde nos es particularmente útil el hecho que elimine duplicados. Por ejemplo: una Encuesta no puede tener la misma pregunta dos veces, ni un perfil puede tener dos encuestas creadas idénticas. Además, contains() o remove() son rápidos también, con una complejidad temporal de $O(1)$.

4.2.2 ArrayList

Es una estructura utilizada para guardar secuencias de elementos donde se hacen accesos por índice o secuenciales de forma frecuente gracias a su complejidad temporal de $O(1)$ por acceso por índice que se convierte en $O(n)$ por recorrido, donde n es el número de elementos en el contenedor.

Sus usos más relevantes son los siguientes:

- ConOpciones.java: Se usa ArrayList<Opcion> para guardar las opciones de una pregunta.
- KMedoids.java y KMeans.java: Se usa ArrayList<RespuestaEncuesta> para manejar los medoids(centroides) y para los clusters resultantes (ArrayList<ArrayList<...>>).

- RespuestaEncuesta.java: Se usa temporalmente en getDatos() para devolver los valores en forma de lista.

4.2.3 TreeMap

Es una estructura arbórea de tipo diccionario (*Key -> Value*) que mantiene las *Keys* ordenadas según el criterio de ordenación que establece la clase de *Key* a no ser que se especifique lo contrario en la constructora. Además, las operaciones de búsqueda, inserción y borrado tienen complejidad temporal asegurada de $O(log\ n)$, donde n es el número de elementos en el contenedor.

Sus usos más relevantes son en:

- RespuestaEncuesta.java: Usa TreeMap<Integer, TDatos> idRespuesta para mantener los datos correspondientes a las respuestas de cada pregunta individual del cuestionario siempre ordenados.
- TDatosOpciones.java : Se usa TreeMap<Integer, Integer> en el cálculo de la moda en los conjuntos de opciones. Más concretamente, a la hora de contar las frecuencias, es muy útil poder acceder al contador de una opción en tiempo $O(log\ n)$. Se ha optado a usar TreeMap en vez de HashMap a pesar de un mayor tiempo de búsqueda para poder asegurar una resolución clara en caso de empate entre diferentes opciones con la misma frecuencia.

4.2.4 SimpleEntry

Es una estructura que guarda un par de valores relacionados, actuando como una implementación muy simple de un *Pair*.

Su uso más relevante es en:

- Perfil.java: Usa Set<SimpleEntry<String,String>> respuestasHechas donde relacionamos el primer String de la pareja como el título de la encuesta y el segundo como el creador de la encuesta.