Extractor de prototipos de comportamientos o perfiles

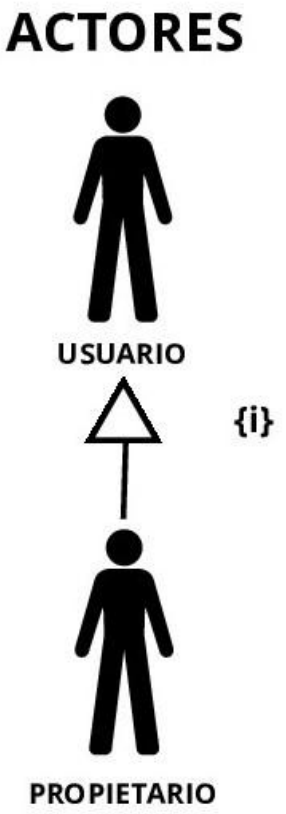
Grup 14.2
versió 1.0

Bel Fonollosa, Marc [marc.bel]
Casas Riera, Berta [berta.casas]
Coré Milla, Daniel [daniel.core]
Pedrejón Sobrino, Pau [pau.pedrejon]
Sohanda Rochani, Dhriti [dhriti.sohanda]

ÍNDICE

1. CASOS DE USO

1.1. Diagrama casos de uso

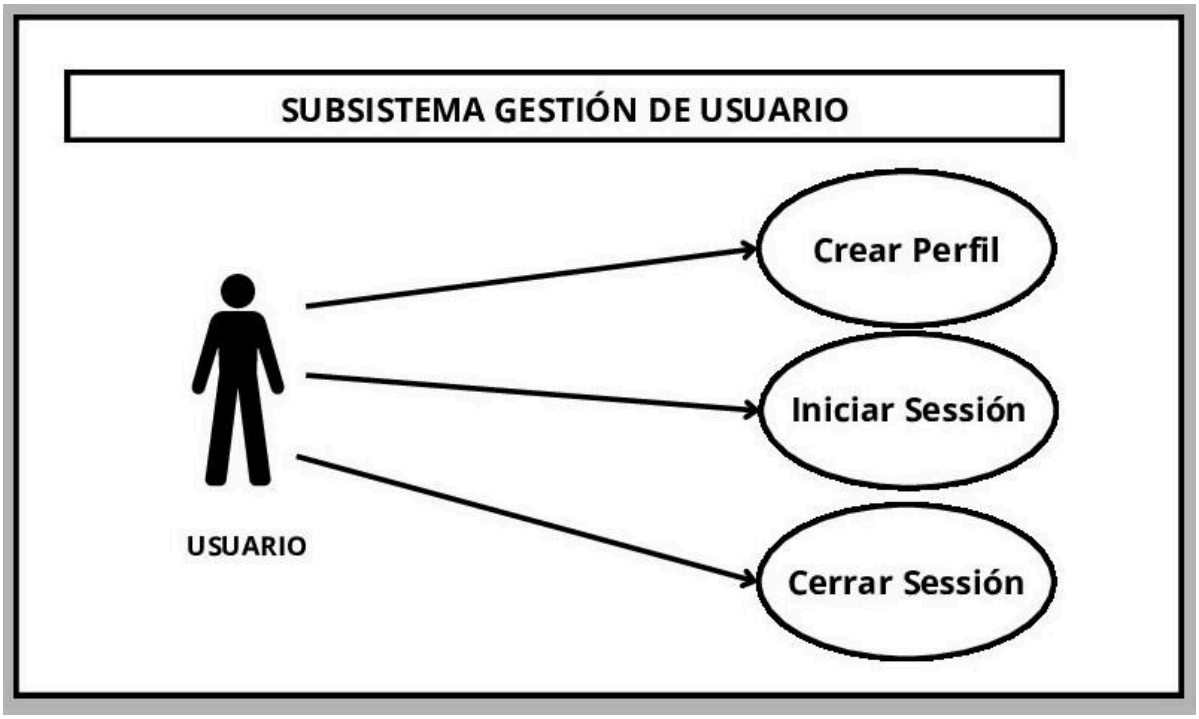


Hemos identificado que el sistema requiere de estos dos actores:

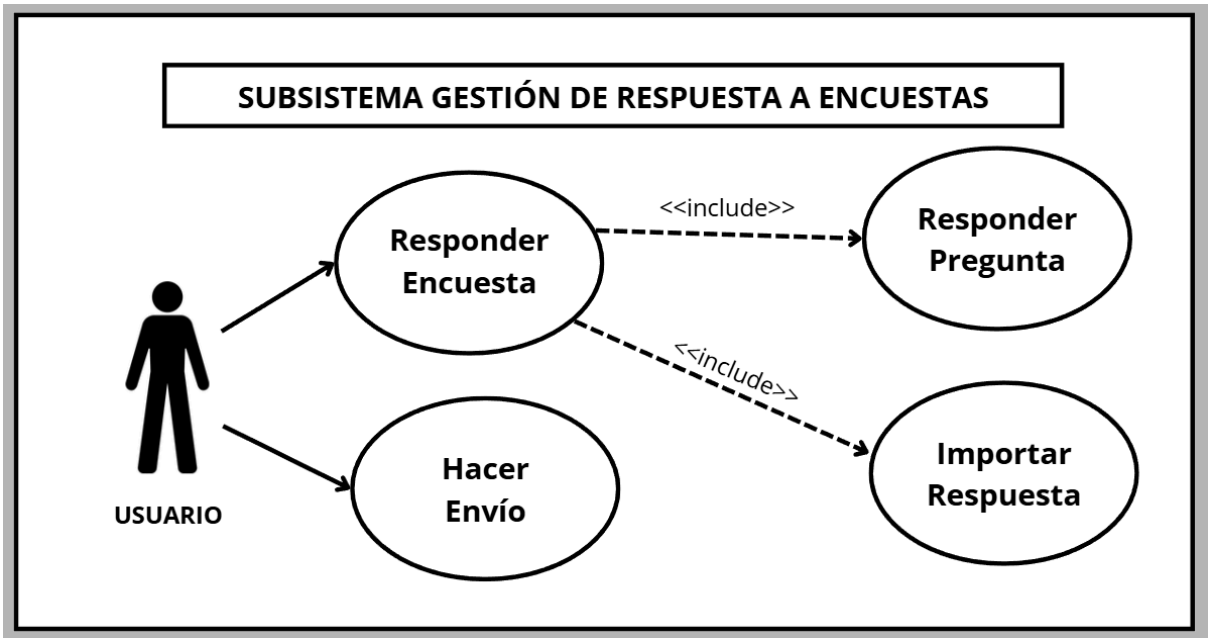
Usuario : Cualquier persona que accede al sistema, debe autenticarse para poder usar el sistema.

Propietario : El propietario es un usuario que ha creado una encuesta, que tiene permisos adicionales sobre esta encuesta para modificarla, publicarla, gestionarla y analizar sus respuestas.

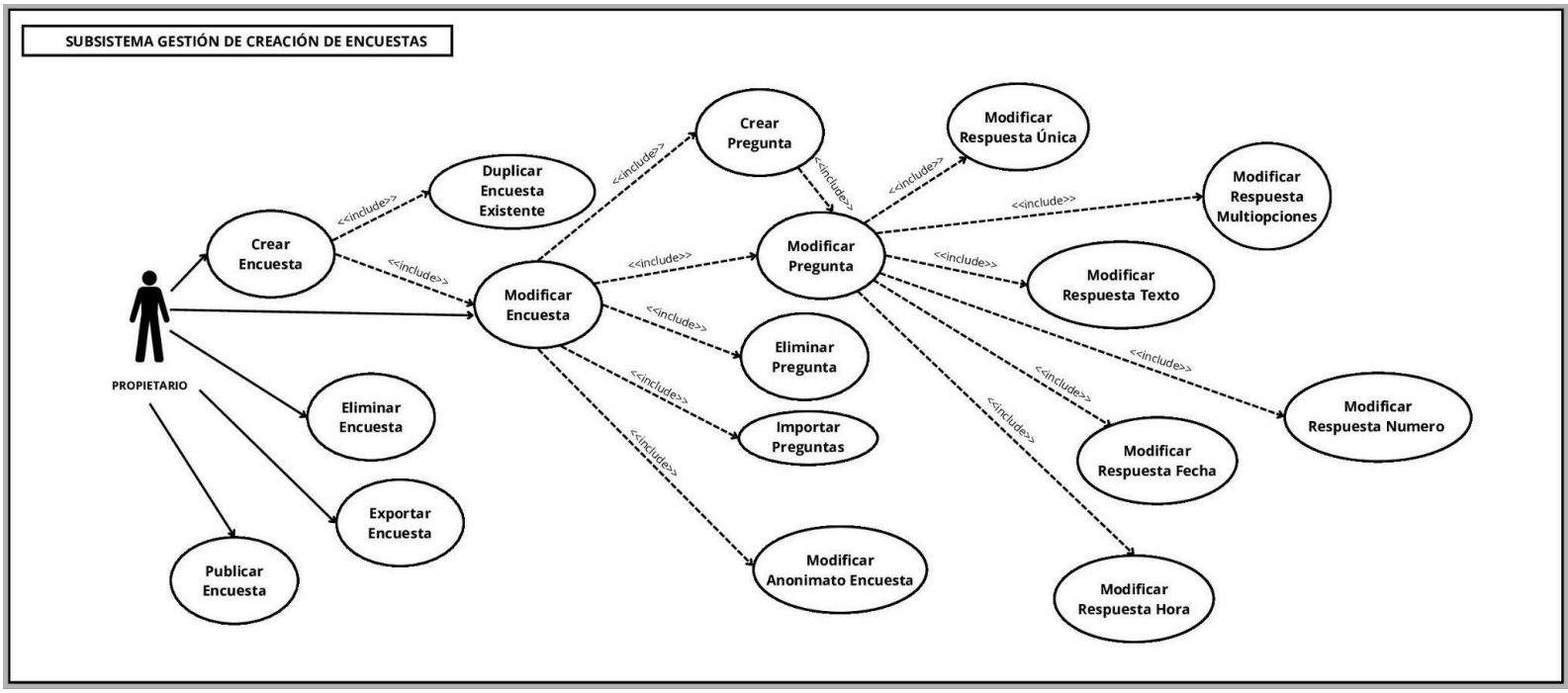
Justificación : Diferenciar estos roles nos permite distinguir qué casos de uso requieren permisos especiales y cuáles no, ya que aunque en nuestro diagrama de clases posteriormente veremos que no existe la clase propietario, su rol si es esencial.



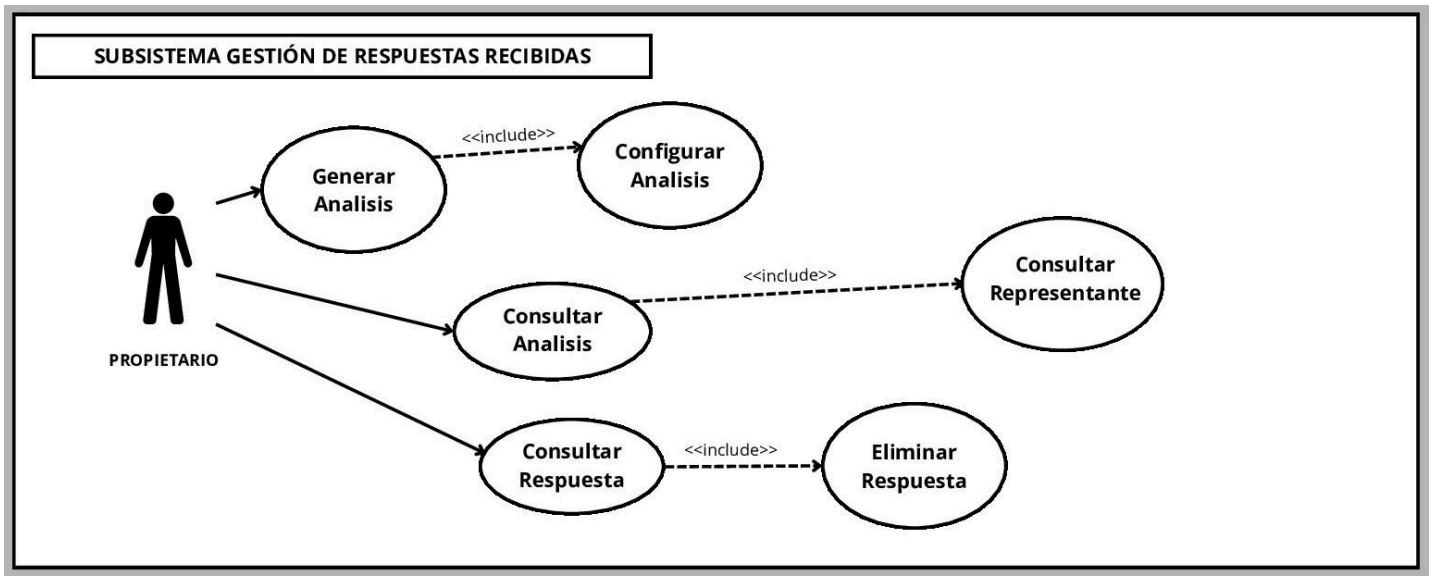
Este subsistema agrupa todas las funcionalidades que tiene un usuario para identificarse en el sistema.



Este subsistema se encarga de las funcionalidades disponibles para que cualquier usuario pueda contestar una encuesta, ya sea propietario de esta encuesta o no.



Este subsistema recoge todas las acciones posibles que puede realizar un propietario sobre sus encuestas no publicadas.



Este subsistema agrupa las acciones que un propietario puede hacer sobre sus encuestas publicadas.

1.2. Descripción casos de uso

1.2.1. Subsistema gestión de usuario

1. **Nombre:** Crear Perfil
   Actor: Usuario
   Comportamiento:
   - El usuario define el nombre de perfil, el correo electrónico, la contraseña y la contraseña confirmada.
   - El sistema verifica que no haya otro perfil con el mismo correo electrónico, valida los datos introducidos y crea el nuevo perfil.
   - El sistema inicia sesión automáticamente si se crea el perfil correctamente.

   Errores posibles y cursos alternativos:
   - Si el correo electrónico ya está asociado a otro perfil, el sistema informa de la coincidencia al usuario y no crea el perfil.
   - Si la contraseña y su confirmación son diferentes, el sistema informa al usuario del error y no crea el perfil.
   - Si el correo electrónico o la contraseña o el nombre son una cadena de carácteres vacía, el sistema informa al usuario y no crea el perfil.

2. **Nombre:** Iniciar sesión
   Actor: Usuario
   Comportamiento:
   - El usuario introduce su correo y su contraseña para acceder al sistema
   - El sistema verifica que existe un perfil con ese correo y que la contraseña coincida. Si la autenticación es correcta, el usuario accede al sistema.

   Errores posibles y cursos alternativos:
   - Si no existe un perfil con ese correo asignado o la contraseña no coincide con la del correo, el sistema informa del error.
   - Si el correo electrónico es una cadena vacía de carácteres, el sistema informa al usuario que tiene que completar este campo.

3. **Nombre:** Cerrar sesión
   Actor: Usuario
   Comportamiento:
   - El usuario con la sesión iniciada cierra sesión y el sistema finaliza la sesión actual.

   Errores posibles y cursos alternativos: -

1.2.2. Subsistema gestión de encuestas

4. **Nombre:** Responder Encuesta
   Actor: Usuario
   Comportamiento:
   - El usuario selecciona una encuesta disponible y comienza el proceso de responderla.
   - El sistema carga todas las preguntas de la encuesta y las muestra, deja al usuario escoger la pregunta a responder.

   Errores posibles y cursos alternativos:
   - Si no existe, el sistema informa del error y no permite responder.

   4.1. Nombre: Responder Pregunta
   Actor: Usuario
   Comportamiento:
   - El Usuario introduce su respuesta a una pregunta concreta (única opción, múltiples opciones, texto, numérica, fecha, etc.).
   - El sistema valida la respuesta según las restricciones del tipo de pregunta (por ejemplo en un número, su rango).

   Errores posibles y cursos alternativos:
   - Si la respuesta no cumple los requisitos de validación (tipo incorrecto, formato no válido, número fuera de rango, etc.), el sistema muestra un mensaje y se cancela el envío.

4.2. Nombre: Importar Respuesta
Actor: Usuario
Comportamiento:

- El usuario selecciona un fichero con respuestas previamente guardadas (por ejemplo, una sesión anterior) y lo carga en el sistema.
- El sistema verifica que el formato y la encuesta correspondan, y completa las respuestas automáticamente.

Errores posibles y cursos alternativos:

- Si el fichero no corresponde a un formato válido de fichero, el sistema muestra un error y no importa ninguna respuesta.
- Si el fichero está incompleto, el sistema informa al usuario.
- Si alguna respuesta individual no es válida, el sistema notifica los errores parciales.

5. **Nombre:** Hacer envío
Actor: Usuario
Comportamiento:

- El sistema verifica que todas las respuestas obligatorias estén contestadas y registra el envío sobre la encuesta.

Errores posibles y cursos alternativos:

- Si faltan respuestas obligatorias, el sistema informa al usuario e impide el envío hasta que se completen.
- Si la encuesta no existe, se lanza una excepción.

1.2.3. Subsistema gestión de creación de encuestas

6. **Nombre:** Crear encuesta
Actor: Propietario
Comportamiento:

- El propietario accede al sistema y selecciona la opción de crear una nueva encuesta.
- Define el título, descripción, y anonimato de la encuesta.
- El sistema verifica que no exista otra encuesta con el mismo ID.
- El sistema registra la nueva encuesta vacía en la base de datos y la asocia al propietario.

Errors possibles i cursos alternatius:

- Si falta algún dato obligatorio (por ejemplo, el título), el sistema muestra un mensaje de error y no permite continuar.
- Si existe una encuesta con el mismo ID, el sistema informa del error y no permite la creación de esta.

6.1. Nombre: Duplicar encuesta existente
Actor: Propietario
Comportamiento:

- El propietario accede al sistema y selecciona la opción de duplicar una encuesta ya existente.
- El sistema registra la encuesta y la asocia al propietario.
- Se duplican todas las preguntas y el título (añadiendo “(copia)”).

Errores posibles y cursos alternativos:

- Si no es propietario de la encuesta, no se permite duplicar y se informa al usuario.
- Si no hay encuestas disponibles, muestra un mensaje de error.

7. Nombre: Modificar encuesta
Actor: Propietario
Comportamiento:

- El propietario accede a una encuesta ya creada y puede realizar distintas acciones de edición sobre ella: añadir preguntas, modificarlas, eliminarlas o importar preguntas desde otra encuesta.
- El sistema actualiza la encuesta con las modificaciones realizadas.

Errores posibles y cursos alternativos:

- Si ocurre un fallo al guardar las modificaciones, el sistema informa al usuario y no deja aplicar los cambios.
- Si no es propietario de la encuesta, no se permite duplicar y se informa al usuario.
- Si la encuesta no existe o ya está publicada, muestra un mensaje de error.

7.1. Nombre: Crear Pregunta
Actor: Propietario
Comportamiento:

- El propietario añade una nueva pregunta a la encuesta, definiendo su enunciado, tipo (respuesta única, múltiple, string, numérica, etc.) y obligatoriedad de responder.
- El sistema añade la pregunta a la encuesta.

Errores posibles y cursos alternativos:

- Si el enunciado está vacío o el tipo no es válido, el sistema informa al usuario.

7.2. Nombre: Modificar Pregunta
Actor: Propietario
Comportamiento:

- El propietario selecciona un número de pregunta.
- En caso de que exista la pregunta, cambia su enunciado y su obligatoriedad a los nuevos establecidos por el usuario.
- Si no existe la pregunta, la crea.

Errores posibles y cursos alternativos:

- Si los datos introducidos son inválidos (por ejemplo, sin opciones en una pregunta de tipo múltiple), el sistema informa al usuario y cancela la operación.
- Si la pregunta no existe, se lanza una excepción.

7.2.1. Nombre: Modificar respuesta única
Actor: Propietario
Comportamiento:

- El propietario introduce el número de opciones que quiere y las nuevas opciones.
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- No hay suficientes opciones, es decir, el número de opciones es menor o igual a 0. Entonces se informa al usuario y se cancela la operación.
- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.2.2. Nombre: Modificar respuesta multiopciones
Actor: Propietario
Comportamiento:

- El propietario introduce el número de opciones que quiere y las nuevas opciones.
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- No hay suficientes opciones, es decir, el número de opciones es menor o igual a 0. Entonces se informa al usuario y se cancela la operación.
- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.2.3. Nombre: Modificar respuesta texto
Actor: Propietario

- El propietario define la pregunta nueva que sustituirá a la antigua.
- El sistema comprueba que el formato sea texto.
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.2.4. Nombre: Modificar respuesta número
Actor: Propietario
Comportamiento:

- El propietario define la pregunta nueva que sustituirá a la antigua.
- El sistema comprueba que el formato sea un número entero.
- El propietario puede introducir el rango del número.
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.2.5. Nombre: Modificar respuesta fecha
Actor: Propietario
Comportamiento:

- El propietario define la pregunta nueva que sustituirá a la antigua.
- El sistema comprueba que el formato sea un texto con ISO (YYYY-MM-DD).
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.2.6. Nombre: Modificar respuesta hora
Actor: Propietario
Comportamiento:

- El propietario define la pregunta nueva que sustituirá a la antigua.
- El sistema comprueba que el formato sea un texto con ISO (HH-mm-ss).
- El sistema actualiza la pregunta en la encuesta.

Errores posibles y cursos alternativos:

- Si introduce un formato inválido, entonces se informa al usuario y se cancela la operación.

7.3. Nombre: Eliminar Pregunta
Actor: Propietario
Comportamiento:
- El propietario selecciona una pregunta y solicita eliminarla.
- El sistema la borra de la encuesta y actualiza la estructura.

Errores posibles y cursos alternativos:
- Si ocurre un fallo al eliminarla, el sistema informa al usuario.

7.4. Nombre: Importar Pregunta
Actor: Propietario
Comportamiento:
- El propietario selecciona la ruta a un archivo como fuente válida.
- El sistema las añade a la encuesta actual conservando sus propiedades.

Errores posibles y cursos alternativos:
- Si alguna pregunta no se puede importar por incompatibilidades (como propiedades diferentes), el sistema lo indica.

7.5. Nombre: Modificar Anonimato Respuestas
Actor: Propietario
Comportamiento:
- El propietario selecciona una encuesta existente y cambia la configuración de anonimato de respuestas (por ejemplo, de anónimo a no anónimo, o viceversa).
- El sistema actualiza esta propiedad en la configuración de la encuesta.

Errores posibles y cursos alternativos:
- Si ocurre un fallo en la actualización, el sistema notifica al usuario.

8. Nombre: Eliminar encuesta
Actor: Propietario
Comportamiento:
- El propietario selecciona una encuesta y solicita su eliminación.
- El sistema pide una confirmación de la acción al propietario y el propietario confirma o cancela la acción.
- El sistema elimina la encuesta y todas sus preguntas asociadas de la base de datos.

Errores posibles y cursos alternativos:
- Si se quiere borrar una encuesta que se está modificando o está abierta, el sistema notifica el error.

9. **Nombre:** Exportar encuesta
Actor: Propietario
Comportamiento:
- El propietario selecciona la encuesta que quiere exportar.
- Se exporta a un fichero de texto o formato compatible para compartirla o guardarla externamente.
- El sistema genera el archivo y lo pone a disposición para descarga.

Errores posibles y cursos alternativos:
- Si ocurre un error al generar el archivo, se muestra un mensaje al usuario.

10. **Nombre:** Publicar encuesta
Actor: Propietario
Comportamiento:
- El propietario selecciona la encuesta que quiere publicar.
- La encuesta aparece visible para todos los usuarios y a partir de este momento se permite hacer envíos.

Errores posibles y cursos alternativos:
- Si la encuesta no existe o el usuario no es propietario, salta una excepción.
- Si no tiene ninguna pregunta no se puede publicar la encuesta, mostrando un mensaje al usuario.

1.2.4. Subsistema gestión de respuestas recibidas

11. **Nombre:** Generar análisis
Actor: Propietario
Comportamiento:
- El Propietario selecciona una encuesta publicada para generar su análisis.
- El sistema le redirige a la pestaña del configurador de análisis.
- Cuando se tiene la configuración, se normalizan los datos y se ejecuta el algoritmo seleccionado.
- Se devuelven los perfiles generados por el análisis y el coeficiente de silueta.

Errores posibles y cursos alternativos:
- Si no es el propietario o no hay ningún envío a la encuesta, se muestra un mensaje de error y no se genera el análisis.

12.1. Nombre: Configurar análisis
Actor: Propietario
Comportamiento:
- El propietario selecciona el algoritmo que quiere usar, introduce el número de representantes que quiere y las iteraciones máximas a hacer.
- El sistema actualiza las configuraciones

Errores posibles y cursos alternativos:
- Datos incorrectos
    - Número de representantes mayor que número de respuestas

12. **Nombre:** Consultar análisis
Actor: Propietario
Comportamiento:
- El propietario selecciona qué análisis quiere consultar.
- El sistema devuelve el número de clusters y el coeficiente de silueta.

Errores posibles y cursos alternativos:
- Si el análisis no existe, se muestra un mensaje de error.
- Si el algoritmo de entrada no existe, no se cumple la restricción textual de 0 <= k <= número de envíos, se muestra un mensaje de error.

12.1. Nombre: Consultar representante
Actor: Propietario
Comportamiento:
- El propietario selecciona qué análisis quiere consultar.
- El sistema devuelve los diferentes perfiles generados por el análisis.
- El usuario selecciona qué perfil consultar.
- El sistema devuelve las respuestas del representante, es decir, las respuestas medias o relevantes entre los envíos del cluster.

Errores posibles y cursos alternativos:
- Si el análisis no existe, se muestra un mensaje de error.

13. **Nombre:** Consultar respuesta
Actor: Propietario
Comportamiento:
- El propietario escoge la encuesta sobre la que quiere consultar
- El sistema le muestra todos los envíos de esa encuesta.
- El propietario selecciona el envío al cual ver sus respuestas.
- El sistema muestra todas las respuestas del envío.

Errores posibles y cursos alternativos:

- La encuesta no existe
- Si el usuario no es propietario de esa encuesta, se cancela la operación y se muestra un mensaje de error.

14. **Nombre:** Eliminar respuesta
Actor: Propietario
Comportamiento:

- El propietario selecciona un envío que quiere eliminar.
- El sistema elimina los datos de ese envío.

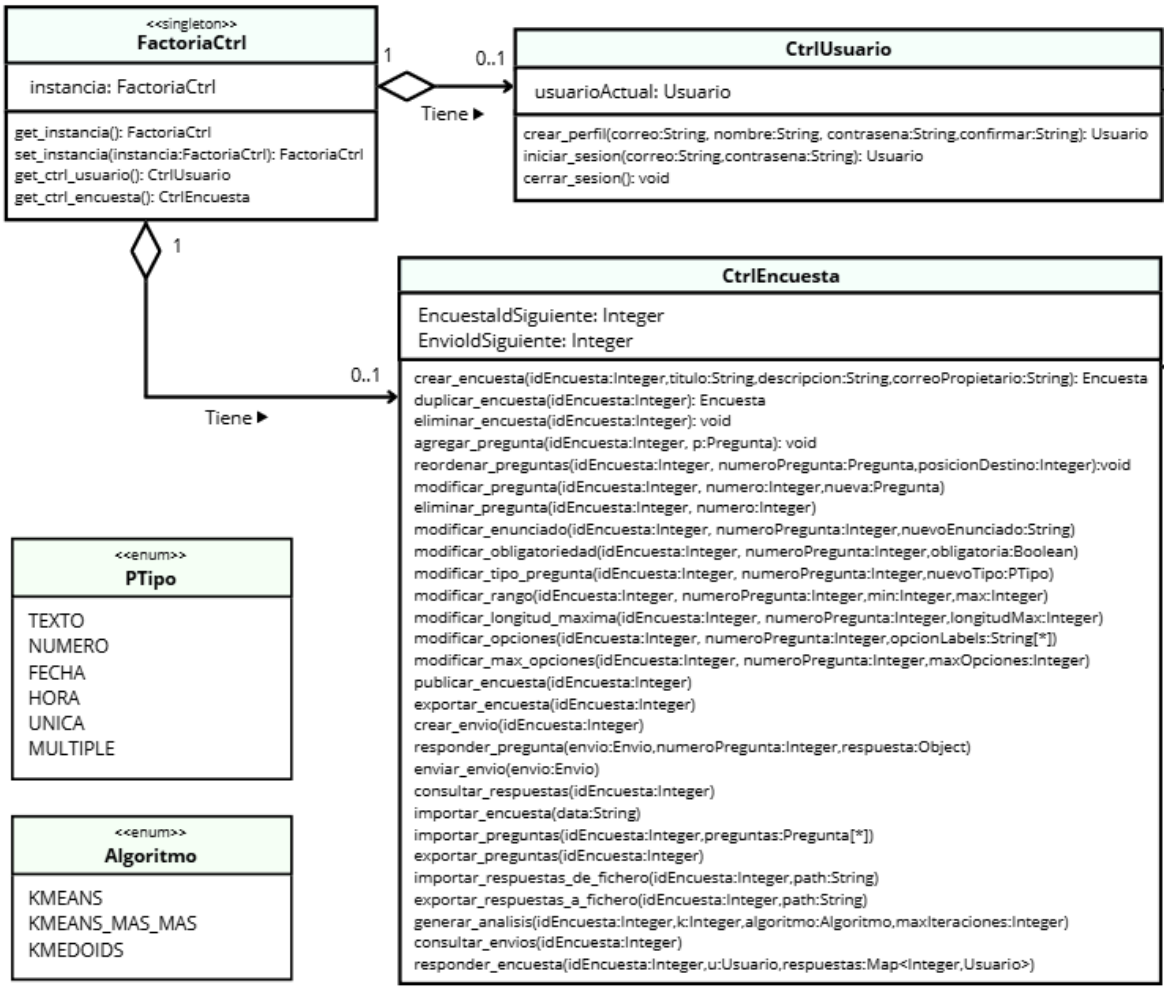
Errores posibles y cursos alternativos:

- Si no hay envío, se cancela la operación y se muestra un mensaje de error.
- Si el usuario que quiere eliminar el envío no es propietario, se cancela la operación y se muestra un mensaje de error.
- Si se borra un envío con análisis ya generado, este se recalcula.

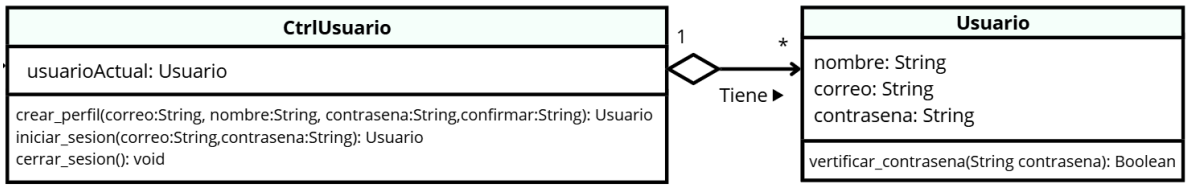
2. MODELO

2.1. Diagrama del modelo

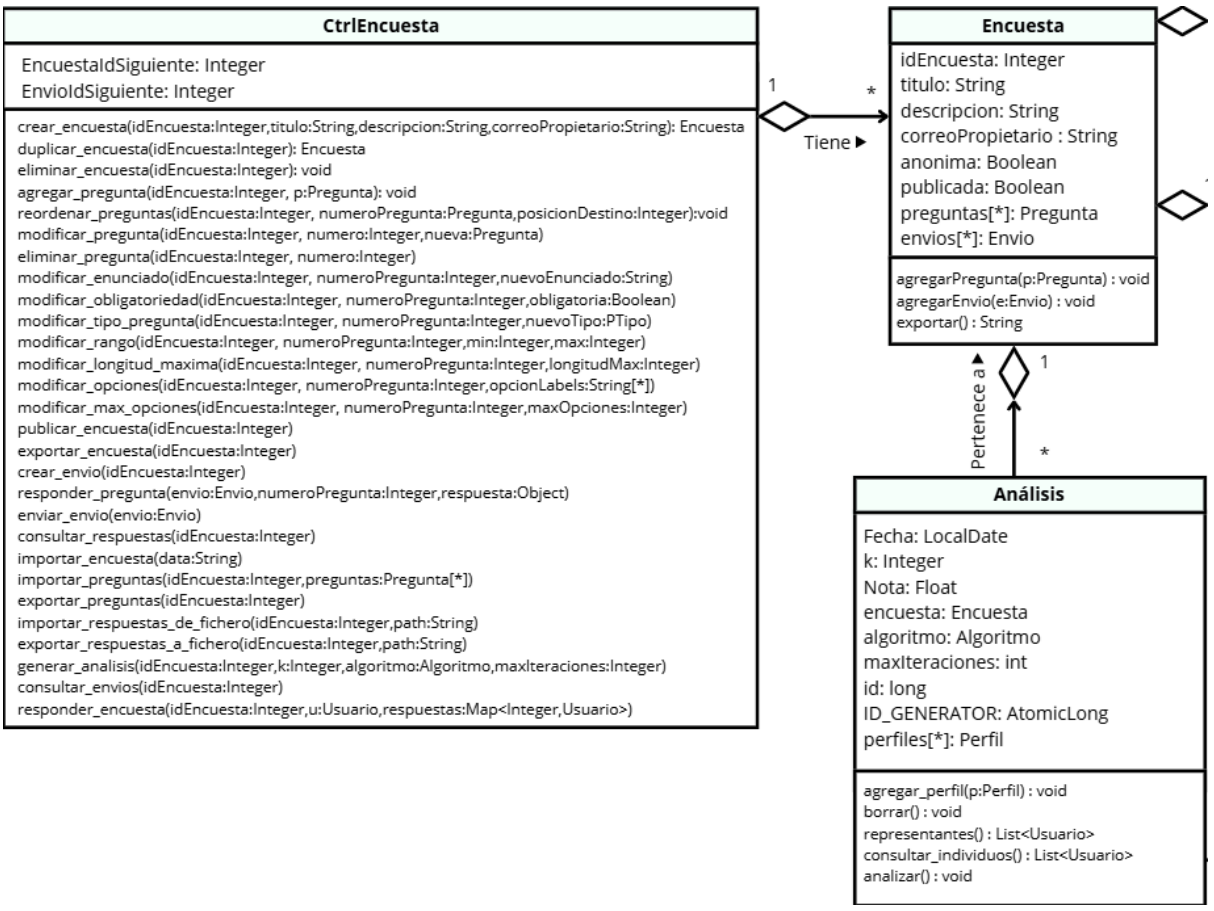
El diagrama de clases completo se encuentra en /DOCS/Diagrama_de_clases



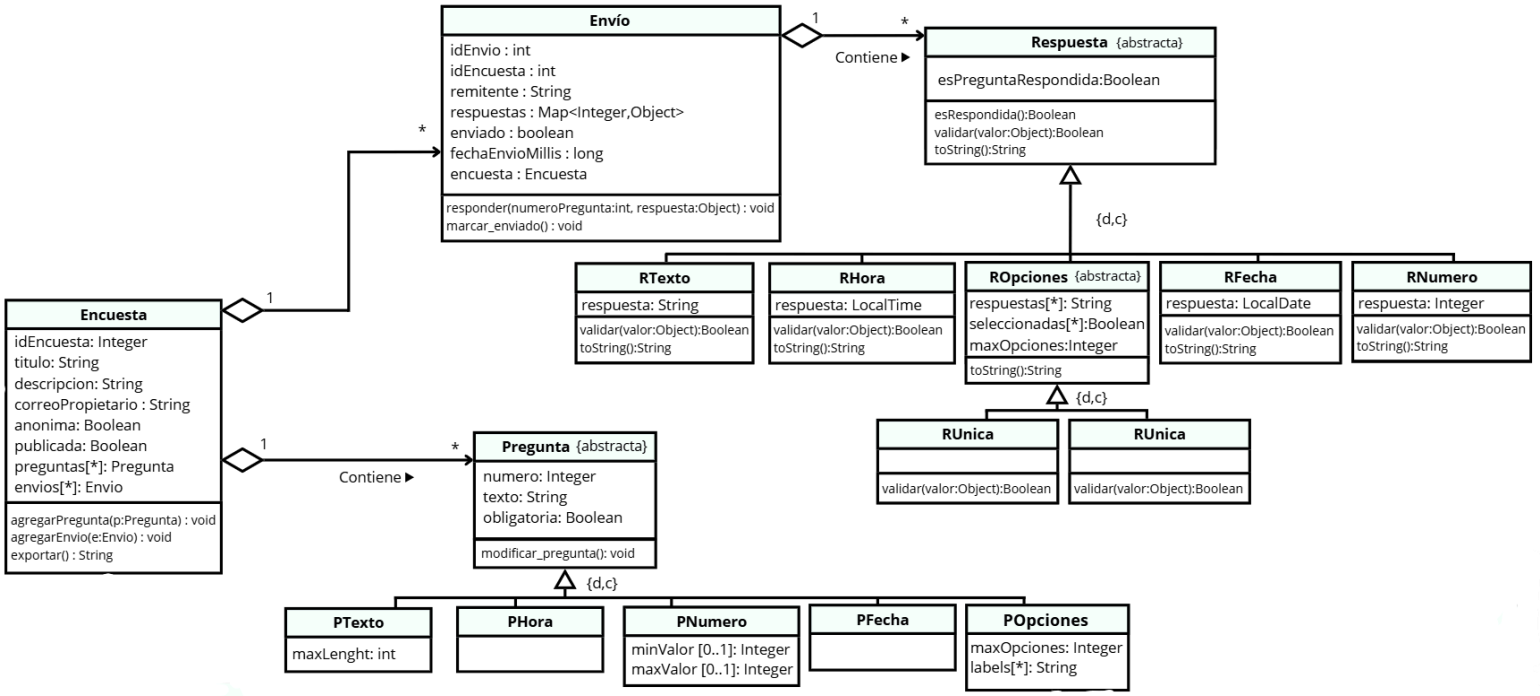
...



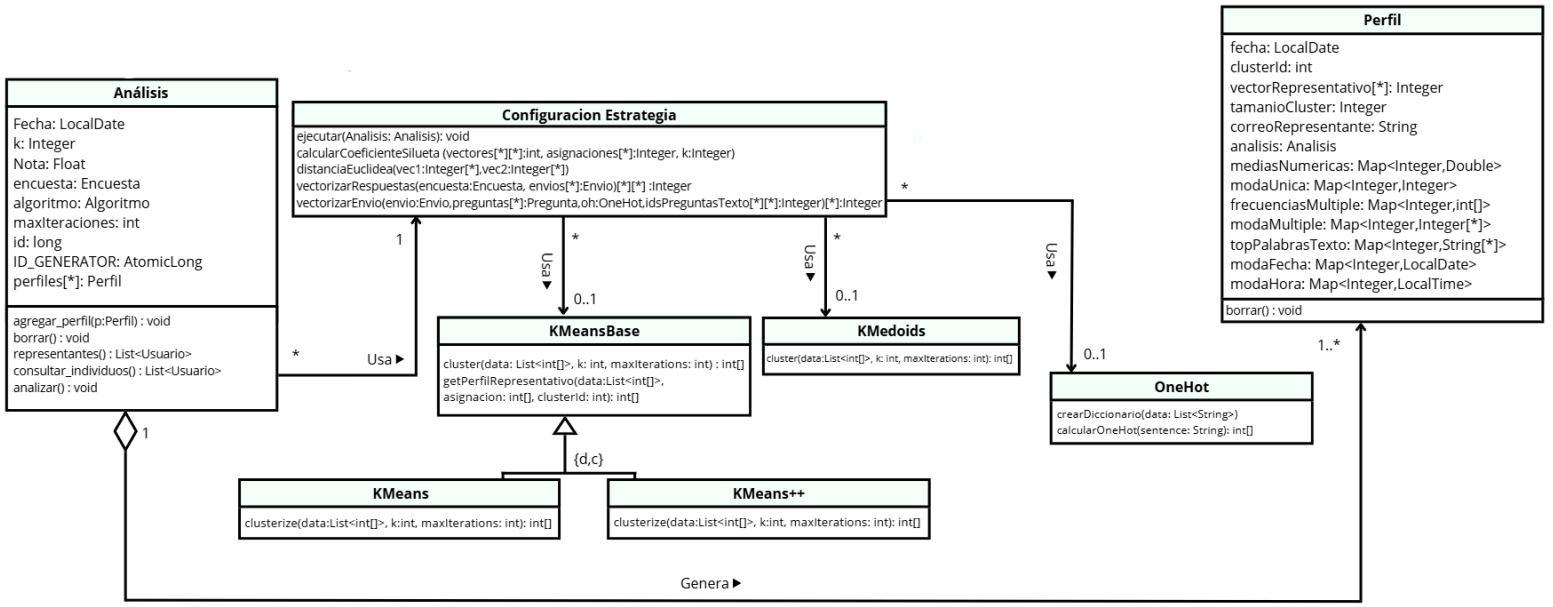
...


CtrlEncuesta
EncuestaIdSiguiente: Integer
EnvioIdSiguiente: Integer
crear_encuesta(idEncuesta:Integer,titulo:String,descripcion:String,correoPropietario:String): Encuesta
duplicar_encuesta(idEncuesta:Integer): Encuesta
eliminar_encuesta(idEncuesta:Integer): void
agregar_pregunta(idEncuesta:Integer, p:Pregunta): void
reordenar_preguntas(idEncuesta:Integer, numeroPregunta:Pregunta,posicionDestino:Integer):void
modificar_pregunta(idEncuesta:Integer, numero:Integer,nueva:Pregunta)
eliminar_pregunta(idEncuesta:Integer, numero:Integer)
modificar_enunciado(idEncuesta:Integer, numeroPregunta:Integer,nuevoEnunciado:String)
modificar_obligatoriedad(idEncuesta:Integer, numeroPregunta:Integer,obligatoria:Boolean)
modificar_tipo_pregunta(idEncuesta:Integer, numeroPregunta:Integer,nuevoTipo:PTipo)
modificar_rango(idEncuesta:Integer, numeroPregunta:Integer,min:Integer,max:Integer)
modificar_longitud_maxima(idEncuesta:Integer, numeroPregunta:Integer,longitudMax:Integer)
modificar_opciones(idEncuesta:Integer, numeroPregunta:Integer,opcionLabels:String[*])
modificar_max_opciones(idEncuesta:Integer, numeroPregunta:Integer,maxOpciones:Integer)
publicar_encuesta(idEncuesta:Integer)
exportar_encuesta(idEncuesta:Integer)
crear_envio(idEncuesta:Integer)
responder_pregunta(envio:Envio,numeroPregunta:Integer,respuesta:Object)
enviar_envio(envio:Envio)
consultar_respuestas(idEncuesta:Integer)
importar_encuesta(data:String)
importar_preguntas(idEncuesta:Integer,preguntas:Pregunta[*])
exportar_preguntas(idEncuesta:Integer)
importar_respuestas_de_fichero(idEncuesta:Integer,path:String)
exportar_respuestas_a_fichero(idEncuesta:Integer,path:String)
generar_analisis(idEncuesta:Integer,k:Integer,algoritmo:Algoritmo,maxIteraciones:Integer)
consultar_envios(idEncuesta:Integer)
responder_encuesta(idEncuesta:Integer,u:Usuario,respuestas:Map<Integer,Usuario>)
1
*
Tiene ▶
Encuesta
idEncuesta: Integer
titulo: String
descripcion: String
correoPropietario : String
anonima: Boolean
publicada: Boolean
preguntas[*]: Pregunta
envios[*]: Envio
agregarPregunta(p:Pregunta) : void
agregarEnvio(e:Envio) : void
exportar() : String
Pertenece a ▶
1
*
Análisis
Fecha: LocalDate
k: Integer
Nota: Float
encuesta: Encuesta
algoritmo: Algoritmo
maxIteraciones: int
id: long
ID_GENERATOR: AtomicLong
perfiles[*]: Perfil
agregar_perfil(p:Perfil) : void
borrar() : void
representantes() : List<Usuario>
consultar_individuos() : List<Usuario>
analizar() : void

...


Envío
idEnvio : int
idEncuesta : int
remitente : String
respuestas : Map<Integer,Object>
enviado : boolean
fechaEnvioMillis : long
encuesta : Encuesta
responder(numeroPregunta:int, respuesta:Object) : void
marcar_enviado() : void
1
*
Contiene ▶
Respuesta {abstracta}
esPreguntaRespondida:Boolean
esRespondida():Boolean
validar(valor:Object):Boolean
toString():String
{d,c}
RTexto
respuesta: String
validar(valor:Object):Boolean
toString():String
RHora
respuesta: LocalTime
validar(valor:Object):Boolean
toString():String
ROpciones {abstracta}
respuestas[*]: String
seleccionadas[*]:Boolean
maxOpciones:Integer
toString():String
RFecha
respuesta: LocalDate
validar(valor:Object):Boolean
toString():String
RNumero
respuesta: Integer
validar(valor:Object):Boolean
toString():String
{d,c}
RUnica
validar(valor:Object):Boolean
RUnica
validar(valor:Object):Boolean
Encuesta
idEncuesta: Integer
titulo: String
descripcion: String
correoPropietario : String
anonima: Boolean
publicada: Boolean
preguntas[*]: Pregunta
envios[*]: Envio
agregarPregunta(p:Pregunta) : void
agregarEnvio(e:Envio) : void
exportar() : String
1
*
Contiene ▶
Pregunta {abstracta}
numero: Integer
texto: String
obligatoria: Boolean
modificar_pregunta(): void
{d,c}
PTexto
maxLenght: int
PHora
PNumero
minValor [0..1]: Integer
maxValor [0..1]: Integer
PFecha
POpciones
maxOpciones: Integer
labels[*]: String

...



2.1.1. Restricciones Textuales

RT1: Claves Primarias (Usuario,correo), (Encuesta, idEncuesta), (Pregunta, numero + Encuesta::idEncuesta), (Respuesta,numero + Envio::idEnvio) (Envio, idEnvio), (Analisis, fecha + Encuesta::idEncuesta), (Perfil,correoRepresentante + Analisis)

RT2: Solo el propietario de una encuesta puede modificarla.

RT3: No se puede hacer un envío si no tiene todas las preguntas obligatorias contestadas.

RT4: No se puede hacer un envío si la encuesta no está publicada.

RT5: El texto de una respuesta del tipo texto no puede superar el máximo de carácteres establecido.

RT6: Un análisis solo puede generarse de una encuesta con mínimo un envío.

RT7: El número de clusters k del análisis tiene que ser mayor a uno y menor al número de envíos.

RT8: La nota del clustering tiene que ser un valor entre 0 y 10 /10.

RT9: Las opciones marcadas en la respuesta tienen que ser pertenecientes a la pregunta con POpciones.

RT10: Si la respuesta es anónima todos los correos de los envíos de la encuesta deben de ser nulos.

RT11: Una respuesta tiene que ser del mismo tipo que la pregunta a la que responde.

RT12: El número máximo de iteraciones del análisis tiene que ser un entero positivo.

---

3. ALGORITMOS

En este apartado se explica el proceso de análisis, como funcionan los algoritmos y cómo se relacionan entre ellos.

1.1. Configuración Estrategia

Esta clase es la encargada de orquestar los diferentes algoritmos para la generación del análisis y sus perfiles a partir de la encuesta y los siguientes parámetros:

- Número de Clusters (k): tiene que haber mínimo uno y máximo el número de envíos.

- Uno de los algoritmos siguientes: KMeans, KMeans++ (algoritmo predeterminado) o KMedoids.

- Iteraciones máximas: tiene que ser un entero positivo mayor a 0.

Ruta de ejecución:

- Validación
    - Se verifica que existe la encuesta que se quiere analizar
    - Comprueba que haya envíos realizados en la encuesta
    - Valida: k>0

- Vectorización
    - Recorre las repuestas de todos los envíos de la encuesta que se analiza
    - Por cada envío, se crea un vector numérico por envío, uniendo las codificaciones de cada pregunta según su tipo

- Clustering
    - Ejecuta el algoritmo deseado (KMeans, KMeans++ o KMedoids)
    - Obtiene el resultado del cluster para cada envío

- Construcciones de perfiles
    - Se calcula un vector representativo por cada cluster
    - Genera agregados estadísticos por cada tipo de pregunta: medias, modas, frecuencias, top 3 de palabras, fecha más frecuente y hora más frecuente.

- Evaluación de calidad
    - Calcula el coeficiente de silueta
    - Se escala la nota a un valor siendo el 0 el valor mínimo y 10 el máximo

1.1.1 Medida de distancia

La distancia entre dos envíos se calcula mediante distancia euclídea normalizada por dimensión:
fórmula

Normalización por dimensión:

- Componentes binarios (TEXTO, UNICA, MULTIPLE): d=1
- Componentes numéricas (NUMERO, FECHA, HORA): d=100

1.1.2 Vectorización por tipo de respuesta

- Respuestas **NUMERO**
  - Codificación: valor entero
  - Distancia:

$$\triangle i(x,y) = \left((x_i - y_i) \div 100 \right)^2$$
$$\text{on } x_i, y_i \in \mathbb{Z}$$

- Respuestas **FECHA**
  - Codificación: hashCode() % 1000
  - Distancia: misma que NUMERO

- Repuestas **HORA**
  - Codificación: igual a FECHA
  - Distancia: misma que NUMERO

- Respuestas **TEXTO**
  - Codifiación:
    - Se llama a OneHot para crear un diccionario de palabras global
    - Se transforma cada respuesta con OneHot.calcularOneHot
  - Distancia: binaria

$$\triangle i(x,y) = \left((x_i - y_i) \right)^2$$

- Respuestas **UNICA**
  - Codificación: vector OneHot de longitud igual al número de opciones
  - Distancia: misma que TEXTO

- Respuestas **MULTIPLE**
  - Codificación: vector OneHot con 1 en cada opción seleccionada
  - Distancia: misma que TEXTO

1.2. KMeans Base

La clase Kmeans Base ejecuta el algoritmo Kmeans. Inicialmente, inicializa los centroides (funcion de Kmeans o Kmeans++). Después itera sobre todos los puntos (maxIterations veces o hasta converger) para asignarles el centroide más cercano utilizando la distancia euclídea. Y recalcula los centroides para buscar una mejor configuración. Cuando termina devuelve los centroides con mejor configuración.

Coste:
public int[ ] cluster(List<int[ ]> data, int k, int maxIterations)
Coste = cost(inicializarCentroides) + O(n) + maxIterations*(O(n*dim*k)+O(dim))
Coste = cost(inicializarCentroides) + O(maxIt*n*dim*k)

protected double[ ][ ] recalcularCentroides(List<int[ ]> data, int[ ] asignacion, int k, int dim)
Coste = O(k) + O(k * dim) + O(n * dim) + O(k * dim) + O(k * dim)
Coste = O(k*dim + n*dim)

protected static double distanciaEuclidea(int[] vec, double[ ] centroide)
Coste = O(dim) + O(1) = O(dim)

public static int[ ] getPerfilRepresentativo(List<int[ ]> data, int[ ] asignacion, int clusterId)
Coste = O(dim) + O(2dim) + O(dim) = O(dim)

public static int[ ] contarPorCluster(int[ ] asignacion, int k)
Coste = O(k) + O(n) = O(k + n)

1.3. KMeans

Esta clase hereda de KmeansBase y modifica la función inicializarCentroides para que cuando se ejecute el algoritmo se inicialicen los centroides de forma Kmeans.

Coste:
protected double[ ][ ] inicializarCentroides(List<int[ ]> data, int k, int dim)
Coste = O(k * dim) + O(n) + O(k * dim) = O(k*dim)

1.4. KMeans++

Esta clase hereda de KmeansBase y modifica la función inicializarCentroides para que cuando se ejecute el algoritmo se inicialicen los centroides de forma Kmeans++.

Coste:
protected double[ ][ ] inicializarCentroides(List<int[ ]> data, int k, int dim)
Coste = O(k*dim) + O(n) + O(dim) + O(n*dim) + O (k*n*dim)
Coste = O(k*n*dim)

1.5. KMedoids

La clase KMedoids implementa una versión simplificada del algoritmo k-medoids aplicada a vectores one-hot. El algoritmo selecciona k medoids reales del conjunto de datos, asigna cada vector al medoid más cercano usando distancia Manhattan normalizada, recalcula los medoids de cada cluster minimizando la suma de distancias internas y repite el proceso hasta converger. Finalmente devuelve el cluster asignado a cada vector. Esta implementación es adecuada para agrupar respuestas de encuestas codificadas con vectores one-hot.

Coste:
public static int[] cluster(List<int[]> data, int k, int maxIterations)
Coste = O(k) + O(k*k*n*data.get(i).length) + O(maxIterations * n * data.get(i).length * k * data.get(i).length) + O(maxIterations * k)
Coste = O(maxIterations * n * k * data.get(i).length²)

private static double distancia(int[] a, int[] b)
Coste = O(a.length) + O(b.length) = O(a.length + b.length)

private static boolean sonIguales(int[] a, int[] b)
Coste = O(a.length)

private static boolean contieneIndice(int[] arr, int value)
Coste = O(arr.length)

1.6. OneHot

La clase OneHot crea un diccionario (crearDiccionario) dada una lista de strings y después calcula el onehot de cada string con calcularOneHot.

Coste:
public void crearDiccionario(List<String> data)
Coste = O(words.length) + O(words.length) = O(words.length)

public int[ ] calcularOneHot(String sentence)
Coste = O(TamDiccionario) + O(words.length) = O(TamDiccionario + words.length)

1.7. Construcción de perfiles

Tras asignar cada envío a un cluster:

Elementos de perfil

- Vector representativo: calculado mediante KMeansBase.getPerfilRepresentativo(...)

- Agregados según tipo de pregunta:
    - **NUMERO:** media por pregunta
    - **UNICA:** moda
    - **MULTIPLE:** frecuencias por opción y modas
    - **TEXTO:** top 3 palabras más frecuentes
    - **FECHA:** moda
    - **HORA:** moda

Cada cluster genera una instancia Perfil con: tamaño, nombre, vector representativo y datos agregados por pregunta.

1.8. Medida de calidad: Coeficiente de silueta

Para cada envío i:

- $s(i)$ es el coeficiente de silueta del envío.
- $a(i)$ es la distancia media a los puntos de su propio cluster.
- $b(i)$ es la mínima distancia media a puntos de otros clusters.

$$s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$$

La nota final se calcula promediando las siluetas y escalando a 0-10:

$$nota = clamp_{[0,10]}\left((\bar{s} + 1) \times 5\right)$$

donde:

- $(\bar{s})$ es la silueta promedio
- $clamp_{[0,10]}$ es una función matemática que limita el valor mínimo a 0 y máximo a 10

4. ESTRUCTURAS DE DATOS

1. Maps

En todos los casos utilizamos los mapas porque nos interesa más el acceso rápido a través de la clave (se implementan bastantes funciones de buscar) . Además no nos importa el orden en que se guarda la información y, por lo tanto, es mejor para estos casos implementar los mapas más que, por ejemplo, una lista o una cola.

Por un lado, en el caso de las listas, si hubiéramos utilizado una lista, deberíamos iterar toda la lista para encontrar el valor que nos interesa y esto es más ineficiente.

Por otro lado, en el caso de las colas, como no nos interesa el orden no es necesario utilizar colas ya que al añadir o quitar no buscamos que sea ni el primero en haberse añadido o el último.

2. Lists (List, ArrayList)

Para estos casos se necesitaba que los datos estuviesen ordenados, a parte de que se pudiera acceder a través de un identificador. Por lo tanto, se utilizan las listas para poder ahorrar la complicación de tener que atribuir a cada valor una clave sino que el identificador es la posición dentro del orden.
La mayoría de operaciones que utilizan estas estructuras iteran todos los valores o piden coger la siguiente y no un valor aleatorio de la lista, por lo tanto, una lista era lo más eficiente posible.

3. Arrays ([])

En estos casos se utilizan arrays ya que nos interesa sobre todo ordenar los valores por posición. Además, la posición se utiliza para atribuir a ciertos parámetros como,por ejemplo, el número de pregunta, así que como la mayoría de operaciones que se hacen son para acceder a una posición en específico, nos interesa algo más parecido a un vector o una lista que a un mapa.