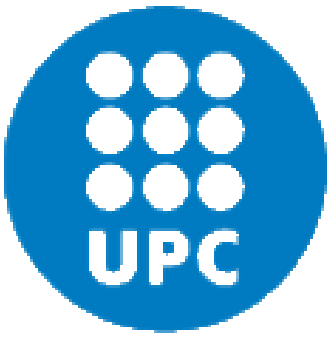


UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

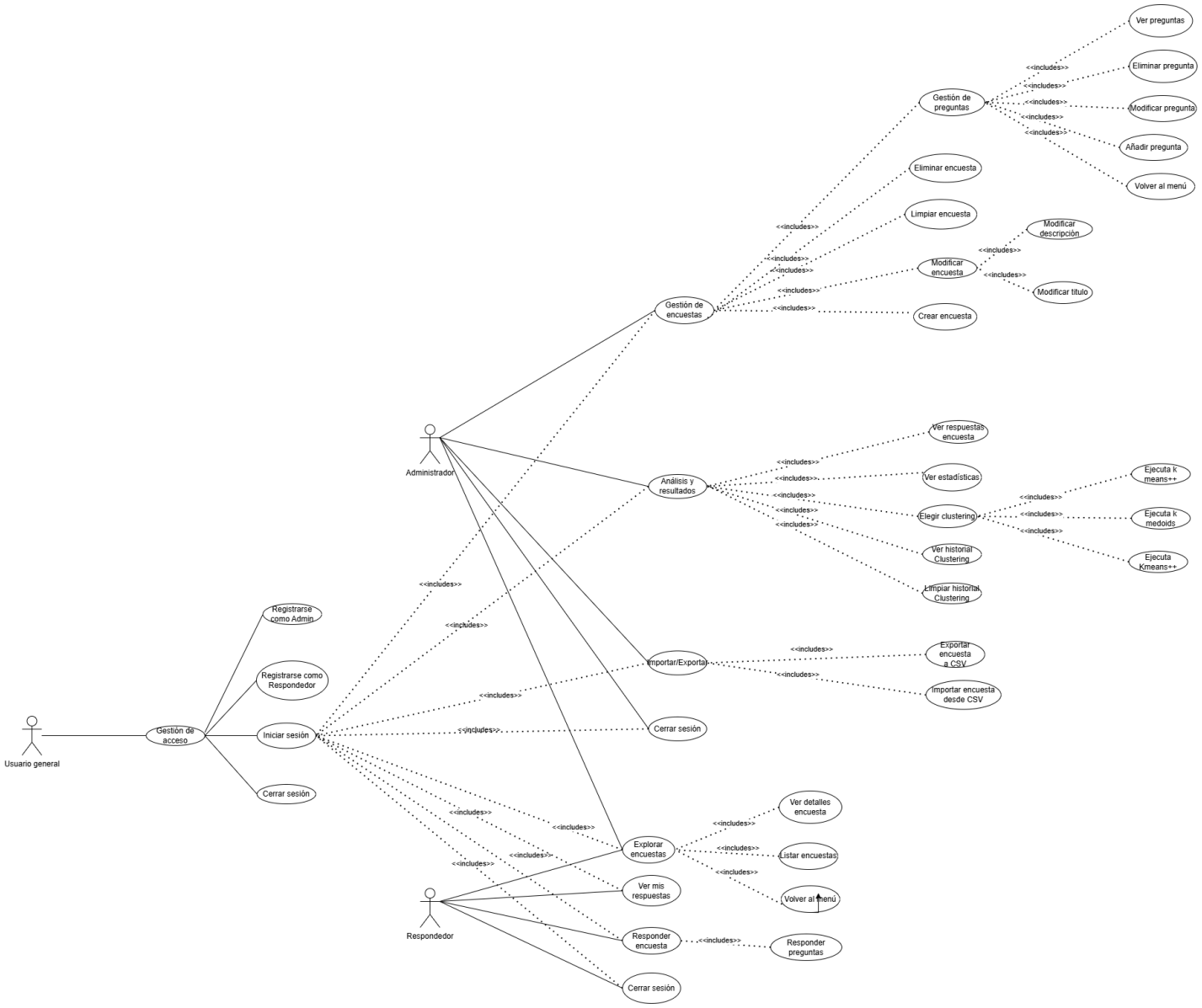
1ª ENTREGA PROP

Identificador de l'equip: 41.1

Mohamed Amara El Houti : mohamed.amara.el.houiti@estudiantat.upc.edu
Andreea Cerchia : andreea.cerchia@estudiantat.upc.edu
Ossama Chaer Dalerou : ossama.chaer@estudiantat.upc.edu
Arnau Serra Florenciano : arnau.serra.florenciano@estudiantat.upc.edu
Adam Ziani Hassun : adam.ziani@estudiantat.upc.edu

Índice

1. Casos de Uso



1.1 Nombre: Registrarse (crear cuenta)

- **Actor:** Respondedor o Administrador.
- **Comportamiento:**
    - El usuario accede a la opción de **registrarse** e introduce un **identificador**, un **nombre visible** y una **contraseña**.
    - El sistema comprueba que los datos sean válidos y que el identificador no esté ya registrado.
    - Si todo es correcto, el sistema crea la **nueva cuenta** y confirma al usuario que el registro se ha completado.
- **Errores posibles:**
    - Si ya existe un usuario con ese identificador, el sistema informa del error y pide elegir otro.
    - Si falta algún dato obligatorio, el sistema informa de que deben completarse todos los campos.

1.2 Nombre: Iniciar sesión

- **Actor:** Respondedor o Administrador.
- **Comportamiento:**
    - El usuario introduce su identificador y contraseña.
    - El sistema verifica que el usuario existe y que la contraseña es correcta.
    - Si las credenciales son válidas, el sistema inicia la sesión y muestra el menú correspondiente al rol del usuario.
- **Errores posibles:**
    - Si el identificador no existe o la contraseña es incorrecta, el sistema muestra un mensaje de credenciales no válidas.

1.3 Nombre: Cerrar sesión

- **Actor:** Respondedor o Administrador.
- **Comportamiento:**
    - El usuario selecciona la opción de cerrar sesión.
    - El sistema finaliza la sesión actual y vuelve a la pantalla de inicio.
- **Errores posibles:**
    - No se esperan errores. Si no había sesión activa, el sistema simplemente vuelve al inicio.

1.4 Nombre: Explorar encuestas

- **Actor:** Respondedor (y también Administrador).
- **Comportamiento:**
    - El usuario selecciona la opción "*Explorar encuestas*".
    - El sistema muestra una lista con todas las encuestas disponibles.
    - El usuario puede **seleccionar** una encuesta para ver más detalles.
- **Errores posibles:**
    - Si no hay encuestas disponibles, el sistema informa de ello.
    - Si el usuario selecciona una encuesta inexistente, el sistema indica el error.

1.5 Nombre: Ver detalle de una encuesta

- **Actor:** Respondedor o Administrador.
- **Comportamiento:**
    - El usuario elige una encuesta y solicita ver sus detalles.
    - El sistema muestra el **título**, la **descripción** y la **lista de preguntas**.
- **Errores posibles:**
    - Si la encuesta ya no existe, el sistema informa del error.

1.6 Nombre: Responder preguntas

- **Actor:** Respondedor.
- **Comportamiento:**
    - El usuario inicia el **proceso de respuesta** de una encuesta.
    - Para cada pregunta, el sistema muestra su enunciado y el tipo de respuesta esperado.
    - El usuario introduce su respuesta y el sistema la valida.
    - Tras completar todas las preguntas, el sistema confirma que la encuesta ha sido respondida correctamente.
- **Errores posibles:**
    - Si una pregunta obligatoria queda sin respuesta, el sistema pide completarla.
    - Si la respuesta no tiene el formato esperado, el sistema pide corregirla.
    - Si se excede el número permitido de opciones en preguntas de selección múltiple, el sistema lo indica.
    - Si el usuario abandona la encuesta, el sistema informa que la encuesta queda incompleta.

1.7 Nombre: Crear encuesta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador elige la opción "*Crear nueva encuesta*".
    - El sistema solicita el título y la descripción.
    - Al confirmar, el sistema crea una encuesta **vacía**.
- **Errores posibles:**
    - Si el usuario no es administrador, el sistema deniega la acción.
    - Si el título está vacío, el sistema indica que debe incluirse un título válido.

1.8 Nombre: Importar encuesta (desde CSV)

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona la opción de **importar** y proporciona la ruta del archivo **CSV**.
    - El sistema valida que el archivo existe y tiene el **formato adecuado**.
    - Muestra una vista previa con número de preguntas, respuestas y tipos detectados.
    - Si se confirma, el sistema crea la encuesta, genera las preguntas y carga los usuarios y sus respuestas.
- **Errores posibles:**
    - Archivo inexistente o ilegible.
    - Falta de cabeceras o filas de datos.
    - Tipos de preguntas no válidos.
    - Columnas numéricas con valores no numéricos.
    - Preguntas categóricas con menos de dos opciones.
    - El administrador puede cancelar antes de confirmar.

1.9 Nombre: Listar encuestas

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador solicita ver todas las encuestas.
    - El sistema muestra la lista completa.
- **Errores posibles:**
    - Si no hay encuestas, el sistema indica que la lista está vacía.

1.10 Nombre: Añadir pregunta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona la encuesta y elige **añadir** pregunta.
    - El sistema solicita el enunciado y el tipo de pregunta.
    - Según el tipo, pide configuraciones adicionales (opciones, rangos, etc.).
    - Al confirmar, el sistema añade la pregunta a la encuesta.
- **Errores posibles:**
    - Si la encuesta no existe, el sistema informa del error.
    - Si se introducen opciones insuficientes para preguntas categóricas, el sistema pide corregirlo.
    - Si se define un rango numérico inválido, el sistema avisa.

1.11 Nombre: Eliminar pregunta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona una pregunta y solicita **eliminarla**.
    - El sistema pide confirmación y, si se aprueba, la borra.
- **Errores posibles:**
    - Si el índice de la pregunta no existe, el sistema avisa.
    - Si la encuesta no existe, se informa del error.

1.12 Nombre: Limpiar encuesta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona "*Eliminar todas las preguntas*".
    - El sistema pide confirmación y, si se acepta, **elimina todas las preguntas**.
- **Errores posibles:**
    - Si la encuesta no existe, el sistema no realiza la acción.
    - Si se cancela la confirmación, no se hace ningún cambio.

1.13 Nombre: Modificar pregunta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona la pregunta a **modificar**.
    - El sistema muestra la configuración **actual**.
    - El administrador introduce los cambios y el sistema actualiza la pregunta.
- **Errores posibles:**
    - Si la pregunta no existe, el sistema indica el error.
    - Si los datos nuevos son inválidos, el sistema solicita corregirlos.

1.14 Nombre: Eliminar encuesta

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona una encuesta y pide **eliminarla**.
    - El sistema pide confirmación.
    - Si se confirma, elimina la encuesta y sus datos.
- **Errores posibles:**
    - Si la encuesta no existe, el sistema informa del error.

1.15 Nombre: Exportar encuesta a CSV

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona una encuesta y elige **exportarla**.
    - El sistema comprueba que tiene preguntas y respuestas.
    - El sistema pide un nombre de archivo y genera el **CSV**.
- **Errores posibles:**
    - Encuesta sin preguntas o sin respuestas.
    - Problemas al escribir el archivo (permisos, ruta inválida).

1.16 Nombre: Ver todas las respuestas

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador **selecciona** una encuesta.
    - El sistema reúne las respuestas de todos los usuarios y las muestra en una tabla.

- **Errores posibles:**
    - Encuesta inexistente.
    - No hay respuestas disponibles.

1.17 Nombre: Ver clusterings

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador selecciona una encuesta y elige la opción "*Ver clusterings*".
    - El sistema localiza los análisis de clustering **previamente ejecutados** sobre esa encuesta.
    - El sistema muestra una lista con todos los clusterings disponibles, indicando para cada uno:
        - el algoritmo utilizado (*K-Means, K-Means++, K-Medoids*).
        - el número de grupos k.
        - y el valor del índice de calidad (por ejemplo, silhouette).
    - El administrador puede seleccionar uno de los clusterings para visualizar sus detalles (si el sistema lo permite en otros casos de uso).
- **Errores posibles:**
    - Si la encuesta no existe, el sistema informa de que no se pueden recuperar los clusterings.
    - Si aún no se ha ejecutado ningún clustering para esa encuesta, el sistema informa de que no hay análisis disponibles.
    - Si se selecciona un clustering inexistente, el sistema indica el error y solicita elegir uno válido.

1.18 Nombre: Ejecutar clustering

- **Actor:** Administrador.
- **Comportamiento:**
    - El administrador elige una encuesta y selecciona la opción de *análisis por clustering*.
    - El sistema solicita el algoritmo (*K-Means, K-Means++, K-Medoids*) y el número de grupos.
    - El sistema ejecuta el análisis y muestra los grupos encontrados y el valor del indicador de calidad.
- **Errores posibles:**
    - Valor de k inválido.
    - Encuesta sin respuestas completas.
    - Algoritmo no reconocido.

1.19 Nombre: Ejecutar K-Means / Ejecutar K-Means++ / Ejecutar K-Medoids

- **Actor:** Administrador.

- **Comportamiento:**

    - El administrador elige la variante deseada del clustering.
    - El sistema ejecuta el análisis y muestra los grupos obtenidos.

- **Errores posibles:**

    - Valor de k inválido.
    - Encuesta sin respuestas completas.
    - Algoritmo no reconocido.

1.20 Nombre: Ver mis respuestas

- **Actor:** Respondedor
- **Comportamiento:**
    - El usuario respondedor selecciona la opción "*Ver mis respuestas*"
    - El sistema solicita el **ID** de la encuesta
    - El sistema muestra todas las respuestas que el usuario ha dado a esa encuesta específica
    - Para cada pregunta se muestra el enunciado y la respuesta proporcionada
- **Errores posibles:**
    - Si la encuesta no existe, el sistema informa del error
    - Si el usuario no ha respondido esa encuesta, el sistema informa que no hay respuestas

1.21 Nombre: Ver estadísticas de encuesta

- **Actor:** Administrador
- **Comportamiento:**
    - El administrador selecciona una encuesta
    - El sistema muestra estadísticas generales: título, número de preguntas, número de respuestas totales
    - El sistema lista los usuarios que han respondido la encuesta
- **Errores posibles:**
    - Si la encuesta no existe, el sistema informa del error

1.22 Nombre: Ver historial de clustering

- **Actor:** Administrador
- **Comportamiento:**
    - El administrador selecciona una encuesta y elige "*Ver historial de clustering*"
    - El sistema muestra todas las ejecuciones previas de clustering realizadas sobre esa encuesta
    - Para cada ejecución se muestra: algoritmo, K, Silhouette, Inercia
    - El administrador puede seleccionar una ejecución **específica** para ver sus detalles completos

- **Errores posibles:**
  - Si la encuesta no existe, el sistema informa del error
  - Si no hay ejecuciones previas, el sistema informa que el historial está vacío
  - Si se selecciona un índice inválido, el sistema indica el error

1.23 Nombre: Limpiar historial de clustering

- **Actor:** Administrador
- **Comportamiento:**
  - El administrador selecciona una encuesta y elige "*Limpiar historial de clustering*"
  - El sistema muestra cuántos resultados se van a **eliminar**
  - El sistema pide **confirmación explícita**
  - Si se confirma, el sistema **elimina todos los resultados de clustering** de esa encuesta
- **Errores posibles:**
  - Si la encuesta no existe, el sistema informa del error
  - Si no hay historial, el sistema informa que no hay nada que limpiar
  - Si se cancela la confirmación, no se realiza ningún cambio

1.24 Nombre: Ver centros de clusters

- **Actor:** Administrador
- **Comportamiento:**
  - Tras visualizar un resultado de clustering, el administrador puede solicitar ver los centros
  - El sistema muestra los valores centrales de cada cluster para cada pregunta
  - Se presenta de forma estructurada: Centro 1, Centro 2, etc., con los valores por pregunta
- **Errores posibles:**
  - Si el resultado de clustering no tiene centros calculados, el sistema informa del error

1.25 Nombre: Ver tabla comparativa de K's

- **Actor:** Administrador
- **Comportamiento:**
  - Al ejecutar clustering con múltiples valores de K, el sistema automáticamente genera una **tabla comparativa**
  - La tabla muestra para cada K: Silhouette Score, Inercia, y número de iteraciones
  - El sistema marca automáticamente el **K óptimo** (mayor Silhouette)
- **Errores posibles:**
  - Si no se pudieron generar resultados para ningún K, el sistema informa del error

1.26 Nombre: Modificar encuesta (título y descripción conjuntamente)

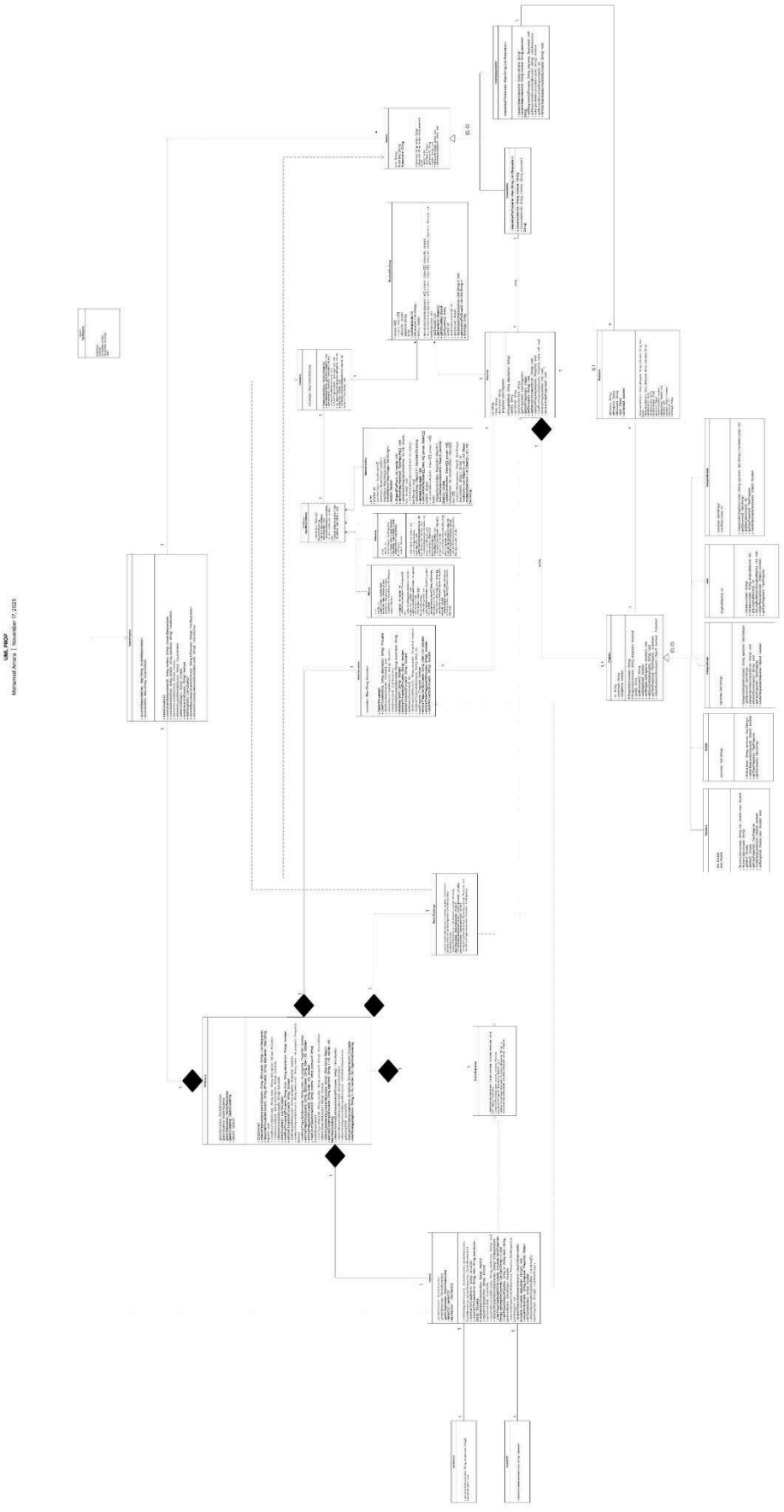
- **Actor:** Administrador
- **Comportamiento:**
  - El administrador selecciona "*modificar encuesta*"
  - El sistema permite cambiar **título** y **descripción** en una sola operación
  - Se puede dejar algún campo vacío para **mantener el valor actual**
  - El sistema actualiza solo los campos que tienen nuevo valor
- **Errores posibles:**
  - Si la encuesta no existe, el sistema informa del error
  - Si ambos campos están vacíos, no se realiza ningún cambio

1.27 Nombre: Ver preguntas detallado

- **Actor:** Administrador
- **Comportamiento:**
  - El administrador selecciona una encuesta
  - El sistema muestra todas las preguntas con información completa:
    - Número de orden
    - Enunciado
    - Tipo de pregunta
    - Si es obligatoria
    - Opciones/rangos/configuraciones específicas según el tipo

- **Errores posibles:**
  - Si la encuesta no existe, el sistema informa del error
  - Si no hay preguntas, el sistema informa que la encuesta está vacía

2. Diagrama de clases del modelo conceptual



2.1 Visión general del diagrama de clases

El sistema modela una aplicación de gestión de encuestas y clustering de respuestas. El modelo se organiza en tres bloques principales:

1. **Bloque de dominio de encuestas**
    - Clases: *Encuesta*, *Pregunta* y sus subclases (*Numerica*, *Ordinal*, *CategoriaSimple*, *CategoriaMultiple*, *Libre*), *Respuesta*, *Usuario* y sus subclases (*UsuarioRespondedor*, *UsuarioAdmin*), el *enum TipoPregunta* y la clase de resultado *ResultadoClustering*.
    - Aquí está toda la lógica de cómo funciona nuestra gestión: qué es una encuesta, qué tipos de preguntas hay, cómo se valida una respuesta y cómo se representan los resultados del *clustering*.

2. **Bloque de servicios de entrada/salida**
    - Clases: *LectorCSV, EscritorCSV* y el controlador *CtrlCSV*.
    - Se encargan de importar encuestas y respuestas desde **CSV** y de exportar los resultados a **CSV**, aislando los detalles de ficheros del resto del dominio.
3. **Bloque de control y clustering**
    - Clases: *GestorEncuestas, GestorUsuarios, GestorRespuestas, GestorClustering*, el contexto *Clustering*, las estrategias *KMeans*, *KMedoids*, *KMeansPlusPlus* y el CtrlDominio.
    - Este bloque relaciona los objetos de dominio, aplica los algoritmos de clustering y proporciona una interfaz básica para la futura **capa de presentación**.

Identificadores y asociaciones clave:

- Cada *Encuesta* tiene un *id : String* único.
- Cada *Pregunta* tiene un *id : String* único y pertenece exactamente a **una** *Encuesta*.
- Cada *Usuario* (tanto admin como respondedor) tiene un *id : String* único.
- *Respuesta* se identifica por el triplete (*idUsuario, idPregunta, idEncuesta*), almacenado como tres atributos.
- La relación encuesta–pregunta es de composición: una *Encuesta* contiene 1..* *Pregunta*.
- Un *Usuario* puede responder 0..* encuestas, y una *Encuesta* puede tener respuestas de 0..* usuarios. Las respuestas de un usuario se agrupan por encuesta.

2.2 Descripción de las clases de dominio

2.2.1 Clase Encuesta

- **Nombre**: Encuesta (*main.domain.classes.Encuesta*)
- **Descripción breve**: representa una encuesta con un identificador único, un título, una descripción y una lista ordenada de preguntas.
- **Atributos principales**:
  - *id : String* – identificador único (*UUID*).
  - *titulo : String* – título de la encuesta.
  - *descripcion : String* – descripción corta de la encuesta.
  - *preguntas : List<Pregunta>* – lista ordenada de preguntas que componen la encuesta.
- **Relaciones**:
  - Una *Encuesta* contiene 1..* *Pregunta*.
  - Es usada por *GestorEncuestas, CtrlDominio, GestorRespuestas* y *GestorClustering*.
- **Métodos principales**:
  - Constructora: *Encuesta(String titulo, String descripcion)*
  - Consultoras: *getId(), getTitulo(), getDescripcion(), getPreguntas(), getNumPreguntas(), getPregunta(int)*
  - Modificadoras: *setTitulo(String)*, *setDescripcion(String)*, *agregarPregunta(Pregunta)*, *modificarPregunta(Pregunta, int)*, *eliminarPregunta(int)*, *eliminarTodasPreguntas()*, *getIndicePregunta(Pregunta)*

2.2.2 Clase abstracta Pregunta y *enum TipoPregunta*

- **Nombre**: Pregunta *(main.domain.classes.Pregunta)*
- **Descripción breve**: clase abstracta que agrupa lo común a todas las preguntas: identificador, enunciado, obligatoriedad y la capacidad de validar una respuesta.
- **Atributos**:
  - *id : String* – identificador único (*UUID*).
  - *enunciado : String* – texto de la pregunta.
  - *obligatoria : boolean* – indica si la pregunta debe contestarse obligatoriamente.
- **Métodos principales**:
  - Constructoras protegidas:
    - *Pregunta(String enunciado)*
    - *Pregunta(String enunciado, boolean obligatoria)*
  - Consultoras: *getId(), getEnunciado(), esObligatoria()*
  - Modificadoras: *setEnunciado(String), setObligatoria(boolean)*
  - Abstractas:
    - *getTipoPregunta() : TipoPregunta*
    - *validarRespuesta(Object respuesta) : boolean* – comprueba que el valor sea válido para este tipo de pregunta.

- *Enum TipoPregunta (main.domain.classes.TipoPregunta):*
    - Valores: NUMERICA, ORDINAL, CATEGORIA_SIMPLE, CATEGORIA_MULTIPLE, LIBRE.
    - Se usa tanto en las subclases de Pregunta como en los algoritmos de *clustering* para decidir el tratamiento de cada columna.

2.2.3 Subclase Numerica

- **Nombre:** Numerica (*main.domain.classes.Numerica*)
- **Descripción breve**: representa una pregunta **numérica** (por ejemplo, edad, nota, etc.) con un rango opcional [min, max].
- **Atributos adicionales**:
    - *min : Double* – valor mínimo permitido (puede ser *null* si no hay límite).
    - *max : Double* – valor máximo permitido (puede ser *null*).
- **Relaciones**:
    - Hereda de *Pregunta*.
    - El rango se usa en el *clustering* para normalizar las distancias numéricas.
- **Métodos principales**:
    - Constructoras:
        - *Numerica(String enunciado, Double min, Double max)*
        - *Numerica(String enunciado)* (sin restricciones explícitas)
    - Consultoras: *getMin(), getMax()*
    - Modificadora: *setRango(Double min, Double max)* (valida que min <= max cuando ambos no son nulos).
    - Overrides:
        - *getTipoPregunta() → TipoPregunta.NUMERICA*
        - *validarRespuesta(Object valor)* – convierte a *Double* (desde Number o String), rechaza *NaN/infinito* y comprueba que el valor respete el rango si hay min/max.

2.2.4 Subclase Ordinal

- **Nombre:** Ordinal (*main.domain.classes.Ordinal*)
- **Descripción breve**: pregunta donde la respuesta es una opción textual de un conjunto ordenado (por ejemplo, “Bajo”, “Medio”, “Alto”).
- **Atributos adicionales**:
    - *opciones : Set<String>* – conjunto de opciones ordinales válidas.
- **Relaciones**:
    - Hereda de *Pregunta*.
    - Sus opciones se usan en el *clustering* para definir distancias **ordinales**.
- **Métodos principales**:
    - Constructora: *Ordinal(String texto, Set<String> opciones)*
    - Consultora: *getOpciones() : Set<String>* (devuelve una copia defensiva).
    - Overrides:
        - *getTipoPregunta() → TipoPregunta.ORDINAL*
        - *validarRespuesta(Object respuesta)* – acepta String no vacía incluida en opciones (o *null* si la pregunta no es obligatoria).

2.2.5 Subclases CategoriaSimple y CategoriaMultiple

CategoriaSimple:

- **Nombre:** CategoriaSimple (*main.domain.classes.CategoriaSimple*)
- **Descripción breve**: pregunta de **selección única** dentro de un conjunto **finito** de opciones (por ejemplo, "Color preferido").
- **Atributos**:
  - *opciones : Set<String>* – conjunto de opciones disponibles.
- **Métodos principales**:
  - Constructora: *CategoriaSimple(String enunciado, Set<String> opciones)*
  - Consultoras: *getOpciones()*
  - Modificadoras de conveniencia: *agregarOpciones(String), eliminarOpcion(String)*
  - Overrides:
    - *getTipoPregunta() → TipoPregunta.CATEGORIA_SIMPLE*
    - *validarRespuesta(Object respuesta)* – pide un String que sea una de las opciones (o *vacío/null* si no es obligatoria).

CategoriaMultiple

- **Nombre:** CategoriaMultiple (*main.domain.classes.CategoriaMultiple*)
- **Descripción breve**: pregunta que permite seleccionar varias opciones de un conjunto, con límite máximo de selecciones.
- **Atributos**:
  - *opciones : Set<String>* – conjunto de opciones posibles.
  - *maxSelecciones : int* – número máximo de opciones seleccionables.
- **Métodos principales**:
  - Constructora: *CategoriaMultiple(String enunciado, Set<String> opciones, int maxSelecciones)*
  - Consultoras: *getOpciones(), getMaxSelecciones()*
  - Overrides:
    - *getTipoPregunta() → TipoPregunta.CATEGORIA_MULTIPLE*
    - *validarRespuesta(Object respuesta)* – espera un *Set<?>* de cadenas, subconjunto de opciones y con tamaño ≤ maxSelecciones.

2.2.6 Subclase Libre

- **Nombre:** Libre (*main.domain.classes.Libre*)
- **Descripción breve**: modela una pregunta de respuesta abierta (**texto libre**), con una **longitud máxima** configurable.
- **Atributos**:
  - *longitudMaxima : int* – máximo de caracteres permitidos (por defecto 1000).
- **Métodos principales**:
  - Constructoras: *Libre(String enunciado), Libre(String enunciado, int longitudMaxima)*
  - Consultora: *getLongitudMaxima()*

- Modificadora: *setLongitudMaxima(int)*
- Overrides:
  - *getTipoPregunta() → TipoPregunta.LIBRE*
  - *validarRespuesta(Object valor)* – permite *null* si no es obligatoria; si hay texto, comprueba que la longitud no supere *longitudMaxima*.

2.2.7 Clase Respuesta

- **Nombre:** Respuesta (*main.domain.classes.Respuesta*)
- **Descripción breve**: representa la respuesta de un usuario a una pregunta concreta de una encuesta.
- **Atributos:**
  - *idUsuario : String*
  - *idPregunta : String*
  - *idEncuesta : String*
  - *valor : Object* – valor de la respuesta (puede ser numérico, texto, conjunto de opciones, etc.).
  - *contestada : boolean* – indica si la pregunta está contestada o no.
- **Métodos principales:**
  - Constructoras:
    - *Respuesta(String idUsuario, String idPregunta, String idEncuesta, Object valor)*
    - *Respuesta(String idUsuario, String idPregunta, String idEncuesta)* (sin valor inicial)
  - Consultoras: *getIdUsuario(), getIdPregunta(), getIdEncuesta(), getValor(), estaContestada()*
  - Modificadoras: *setValor(Object), limpiar()* (pone el valor a null y marca contestada = *false*)
  - Redefinición de *equals()* y *hashCode()* basada en (*idUsuario, idPregunta, idEncuesta*).

2.2.8 Clase abstracta Usuario y subclases

Usuario

- **Nombre:** Usuario (*main.domain.classes.Usuario*)
- **Descripción breve:** superclase común para cualquier usuario del sistema (administrador o respondedor).
- **Atributos:**
  - *id : String*
  - *nombre : String*
  - *password : String*
- **Métodos:**
  - Consultoras: *getId(), getNombre(), getPassword()*
  - Modificadoras: *setId(String), setNombre(String), setPassword(String)*

- Abstractas:
  - *addRespuesta(String idEncuesta, Respuesta respuesta)*
  - *getRespuestasEncuesta(String idEncuesta) : List<Respuesta>*

UsuarioRespondedor

- **Nombre:** UsuarioRespondedor (*main.domain.classes.UsuarioRespondedor*)
- **Descripción breve:** usuario que responde encuestas. Almacena sus respuestas agrupadas por encuesta.
- **Atributos adicionales:**
  - *respuestasPorEncuesta : Map<String, List<Respuesta>>*
- **Métodos relevantes:**
  - Constructoras con y sin password.
  - *addRespuesta(String idEncuesta, Respuesta respuesta)*
  - *getRespuestasEncuesta(String idEncuesta)*
  - *haRespondidoEncuesta(String idEncuesta) : boolean*
  - *getNumeroEncuestasRespondidas() : int*
  - *eliminarRespuestasEncuesta(String idEncuesta)*

UsuarioAdmin

- **Nombre:** UsuarioAdmin (*main.domain.classes.UsuarioAdmin*)
- **Descripción breve:** usuario con permisos de administración (crear/modificar encuestas) que también puede responder encuestas.
- **Atributos adicionales:**
  - *respuestasPorEncuesta : Map<String, List<Respuesta>>* (mismo patrón que el respondedor).
- **Métodos relevantes:**
  - Constructoras con y sin password.
  - *addRespuesta(String idEncuesta, Respuesta respuesta)*
  - *getRespuestasEncuesta(String idEncuesta)*
  - *haRespondidoEncuesta(String idEncuesta) : boolean*

2.2.9 Clase ResultadoClustering

- **Nombre:** ResultadoClustering (*main.domain.classes.ResultadoClustering*)
- **Descripción breve:** encapsula el resultado de una ejecución de clustering.
- **Atributos:**
  - *groups : int[]* – para cada usuario, el índice de cluster asignado.
  - *centers : Object[][]* – centros (o medoides) de cada cluster.
  - *silhouette : double* – índice de silueta global.
  - *algoritmo : String* – nombre del algoritmo empleado.
  - *k : int* – número de clusters.
  - *numIteraciones : int* – número de iteraciones realizadas.
  - *idsUsuarios : List<String>* – lista de IDs de usuario en el mismo orden que groups.

- **Métodos principales:**
  - Constructor básico y constructor enriquecido (con metadatos).
  - Consultoras: *getGroups(), getCenters(), getSilhouette(), getAlgoritmo(), getK(), getNumIteraciones()*
  - *setIdsUsuarios(List<String>)*
  - *getUsuariosPorGrupo() : List<List<String>>* – devuelve la lista de usuarios agrupados por cluster.
  - *toString()* – resumen legible del resultado.

2.2.10 Clases LectorCSV y EscritorCSV

LectorCSV

- **Nombre:** LectorCSV (*main.domain.classes.LectorCSV*)
- **Descripción breve:** clase de bajo nivel para leer ficheros CSV, con separador configurable y opción de ignorar líneas vacías. Tiene una clase interna DatosCSV.
- **Atributos:**
  - *separador : String*
  - *saltarLineasVacias : boolean*
- **Métodos principales:**
  - *leerArchivo(String ruta) : List<String[]>*
  - *leerPrimeraLinea(String ruta) : String[]*
  - *leerSinPrimeraLinea(String ruta) : List<String[]>*
  - *contarLineas(String ruta) : int*
  - *contarColumnas(String ruta) : int*
  - *leerConEncabezados(String ruta) : DatosCSV*
  - Getters y setters del separador y de *saltarLineasVacias*.
- **Clase interna DatosCSV:**
  - Atributos: *encabezados : String[], filas : List<String[]>*
  - Métodos: *getEncabezados(), getFilas(), getNumeroColumnas(), getNumeroFilas(), getFila(int), getValor(int fila, int columna)*

EscritorCSV

- **Nombre:** EscritorCSV (*main.domain.classes.EscritorCSV*)
- **Descripción breve:** encapsula la escritura de datos a CSV, con control de separador y encabezados.
- **Atributos:**
  - *separador : String*
  - *incluirEncabezados : boolean*
- **Método principal:**
  - *escribirArchivo(String rutaArchivo, String[] encabezados, List<String[]> filas) : void*

2.3 Descripción de las clases de servicio y control

2.3.1 Clase GestorEncuestas

- **Nombre:** GestorEncuestas (*main.domain.controller.GestorEncuestas*)
- **Descripción breve:** gestor *CRUD* de encuestas. Mantiene todas las encuestas en memoria y centraliza su creación, modificación y borrado.
- **Atributos:**
  - *encuestas : Map<String, Encuesta>* – diccionario idEncuesta → Encuesta.
- **Métodos principales:**
  - Crear/añadir: *crearEncuesta(String titulo, String descripcion), addEncuesta(Encuesta)*
  - Consultar: *obtenerEncuesta(String id), listarEncuestas(), existeEncuesta(String id), getNumeroEncuestas()*
  - Modificar: *modificarEncuesta(String id, String nuevoTitulo, String nuevaDescripcion)*
  - Eliminar: *eliminarEncuesta(String id)*
  - Gestión de preguntas: *addPregunta(String idEncuesta, Pregunta), modificarPregunta(String idEncuesta, int index, Pregunta), eliminarPregunta(String idEncuesta, int index), eliminarTodasPreguntas(String idEncuesta)*

2.3.2 Clase GestorUsuarios

- **Nombre:** GestorUsuarios (*main.domain.controller.GestorUsuarios*)
- **Descripción breve:** gestiona la creación y consulta de usuarios, distinguiendo entre respondedores y administradores.
- **Atributos:**
  - *usuariosRespondedores : Map<String, UsuarioRespondedor>*
  - *usuariosAdmin : Map<String, UsuarioAdmin>*
- **Métodos principales:**
  - Creación: *crearUsuarioRespondedor(String id, String nombre, String password),* versión sin password, y *crearUsuarioAdmin(String id, String nombre, String password)*
  - Consulta: *obtenerUsuario(String id), esAdmin(String id), existeUsuario(String id), listarUsuarios()*
  - Respuestas: *obtenerRespuestasUsuario(String idUsuario, String idEncuesta) : List<Respuesta>*
  - Análisis: *obtenerUsuariosQueRespondieron(String idEncuesta) : List<Usuario>*

2.3.3 Clase GestorRespuestas

- **Nombre:** GestorRespuestas (*main.domain.controller.GestorRespuestas*)
- **Descripción breve:** servicio encargado de validar y registrar respuestas de usuarios a encuestas. No mantiene estado propio.

- **Métodos principales:**
  - *responderPregunta(Usuario usuario, Encuesta encuesta, String idPregunta, Object valor) : Respuesta*
    - Localiza la Pregunta, llama a *validarRespuesta*, y si es válida crea una Respuesta y la añade al Usuario.
  - *responderEncuesta(Usuario usuario, Encuesta encuesta, Map<String,Object> respuestasPorPregunta) : void*
    - Verifica que se hayan contestado todas las preguntas obligatorias y llama internamente a *responderPregunta* para cada entrada.
  - Métodos de ayuda:
    - *obtenerIndicePregunta(Encuesta encuesta, String idPregunta) : int*
    - *encontrarPregunta(Encuesta encuesta, String idPregunta)* (privado)

2.3.4 Clase GestorClustering y contexto Clustering

GestorClustering

- **Nombre:** GestorClustering (*main.domain.controller.GestorClustering*)
- **Descripción breve:** prepara los datos para el clustering y configura el algoritmo. Convierte las respuestas en una matriz *Object[][]* y llama al contexto *Clustering*.
- **Atributos:**
  - *idsUsuarios : List<String>* – IDs de los usuarios en el mismo orden que las filas de datos.
- **Métodos principales:**
  - *ejecutarClustering(Clustering clustering, List<Usuario> usuarios, Encuesta encuesta, GestorRespuestas gestorRespuestas) : ResultadoClustering*
  - Métodos privados:
    - *prepararDatos(...)* – construye la matriz de respuestas, sustituyendo valores nulos haciendo uso del algoritmo KNN.
    - *convertirRespuestasAArray(...)*
    - *configurarAlgoritmo(Clustering clustering, Encuesta encuesta)* – pasa rangos numéricos y opciones ordinales.
    - *extraerTiposPreguntas(Encuesta encuesta) : TipoPregunta[]*
    - *imputarValoresNull(List<Object[]> datos, Encuesta encuesta)*
    - *imputarConKNN(...) - imputa la nueva información a la lista de respuestas otorgada (no modifica el valor real de la respuesta)*
    - *calcularDistanciaParcial(...) : Double*
    - *calcularDistanciaUnaPregunta(...) : Double*
    - *calcularValorImputado(...)*
    - *obtenerValorPorDefecto(...)*
    - *levenshteinDistance(...) : Int*
    - *convertToSet(Object obj) : Set<String>*

Clustering

- **Nombre:** Clustering (*main.domain.classes.Clustering*)
- **Descripción breve:** contexto del patrón *Strategy*. Contiene una referencia a una implementación de AlgoritmoClustering y ofrece una interfaz básica para ejecutar el algoritmo y configurar parámetros.
- **Atributos**:
  - *estrategia : AlgoritmoClustering*
- **Métodos principales:**
  - *Constructor: Clustering(AlgoritmoClustering estrategia)*
  - *setEstrategia(AlgoritmoClustering estrategia)*
  - *ejecutar(Object[][] datos) : ResultadoClustering*
  - *setTiposPreguntas(TipoPregunta[] tipos)*
  - *configurarRangoNumerico(int indicePregunta, double min, double max)*
  - *configurarOpcionesOrdinales(int indicePregunta, Set<String> opciones)*

2.3.5 Estrategias KMeans, KMedoids, KMeansPlusPlus (AlgoritmoClustering)

- **Nombre genérico:** KMeans, KMedoids, KMeansPlusPlus
- **Interfaz:** AlgoritmoClustering (*main.domain.classes.AlgoritmoClustering*)
- **Descripción breve:** tres implementaciones distintas de clustering que comparten una interfaz común:
  - Reciben una matriz *Object[][]* de datos ya procesados.
  - Usan *TipoPregunta[],* rangos numéricos y opciones ordinales para calcular distancias adecuadas a cada tipo de pregunta.
  - Devuelven un ResultadoClustering con grupos, centros y silueta.

- **Atributos comunes:**
  - *k : int* – número de clusters.
  - *maxIter : int* – número máximo de iteraciones.
  - *questionTypes : TipoPregunta[]*
  - *minValues, maxValues : Map<Integer, Double>* – mínimos y máximos por pregunta numérica.
  - *ordinalOptions : Map<Integer, Set<String>>* – opciones por pregunta ordinal.
  - *random : Random*
- **Métodos de la interfaz AlgoritmoClustering:**
  - *ResultadoClustering execute(Object[][] data)*
  - *void setTipoPreguntas(TipoPregunta[] tipos)*
  - *void setNumericRange(int questionIndex, double min, double max)*
  - *void setOrdinalOptions(int questionIndex, Set<String> options)*

Cada clase concreta implementa:

- Su estrategia de inicialización de centros/medoides.
- Cálculo de distancias según tipo de pregunta (numérica, ordinal, categórica simple, múltiple, libre).
- Recalculo de centros/medoides.
- Cálculo del índice de silueta.

2.3.6 Controlador CtrlCSV

- **Nombre:** CtrlCSV (*main.domain.controller.CtrlCSV*)
- **Descripción breve:** controlador de dominio que encapsula toda la lógica de **importación/exportación** de encuestas y respuestas en formato **CSV**, usando *LectorCSV, EscritorCSV, GestorEncuestas, GestorUsuarios* y *GestorRespuestas*.
- **Atributos:**
  - *gestorUsuarios : GestorUsuarios*
  - *gestorEncuestas : GestorEncuestas*
  - *gestorRespuestas : GestorRespuestas*
  - *lectorCSV : LectorCSV*
  - *escritorCSV : EscritorCSV*
- **Métodos principales:**
  - Importación:
    - *Encuesta importarCSV(String rutaArchivo)*
    - *Encuesta importarCSV(String rutaArchivo, String titulo, String descripcion)*
  - Exportación:
    - *void exportarEncuesta(String idEncuesta, String rutaArchivo)*
    - *InfoExportacion obtenerInfoExportacion(String idEncuesta)*
  - Información/validación:
    - *InfoCSV obtenerInfoCSV(String rutaArchivo)*
    - *boolean validarCSV(String rutaArchivo)*
  - Métodos privados auxiliares para:
    - Extraer opciones por columna.
    - Crear preguntas a partir de los tipos del CSV.
    - Convertir valores de texto a los tipos esperados.

2.3.7 Controlador de alto nivel CtrlDominio

- **Nombre:** CtrlDominio (*main.domain.controller.CtrlDominio*)
- **Descripción breve:** representación de la capa de dominio. Agrupa los gestores y controladores internos y expone métodos que usará la futura capa de presentación (*GUI/CLI*).
- **Atributos:**
  - *gestorEncuestas : GestorEncuestas*
  - *gestorUsuarios : GestorUsuarios*
  - *gestorRespuestas : GestorRespuestas*
  - *gestorClustering : GestorClustering*
  - *ctrlCSV : CtrlCSV*

- **Responsabilidades principales:**
  - Encuestas: *crearEncuesta(...), obtenerEncuesta(...), listarEncuestas(), modificarEncuesta(...), eliminarEncuesta(...), addPregunta(...), modificarPregunta(...), eliminarPregunta(...).*
  - Usuarios: *crearUsuarioRespondedor(...), crearUsuarioAdmin(...), obtenerUsuario(...).*
  - Respuestas: *responderPregunta(...), responderEncuesta(...), obtenerRespuestasUsuario(...).*
  - Estadísticas y clustering: *obtenerEstadisticasEncuesta(...), ejecutarClustering(...)* y variantes *(ejecutarClusteringKMeans, KMedoids, KMeansPlusPlus).*
  - CSV: *importarEncuestaDesdeCSV(...), obtenerInfoCSV(...), validarCSV(...), exportarEncuestaCSV(...), obtenerInfoExportacion(...).*

3. Estructuras de datos y algoritmos utilizados

3.1 Estructuras de datos

3.1.1 List y ArrayList

En nuestro proyecto se utiliza la interfaz **List** y su implementación más común, **ArrayList**. Este tipo de estructura aparece, por ejemplo, en:

- La lista de preguntas de una encuesta → *Encuesta.getPreguntas()*
- Las listas de respuestas de un usuario a una encuesta → *Map<String, List<Respuesta>>* en *UsuarioAdmin* y *UsuarioRespondedor*
- Las filas leídas de un CSV en *LectorCSV* → *List<String[]>*
- La colección de usuarios o encuestas devuelta por los gestores: *listarUsuarios*, *listarEncuestas*.
- El almacenamiento temporal de filas válidas en *GestorClustering.prepararDatos*: lista de *Object[]* posteriormente convertida a *Object[][]*

Se ha optado por **ArrayList** porque:

- Permite acceso por índice en **O(1)**.
- Ofrece buen rendimiento en recorridos secuenciales y accesos por posición.
- La inserción al final también es **O(1)**.
- No se necesita insertar en el medio ni mantener el orden explícitamente.

En este sistema, las listas se usan como:

- Contenedores de objetos de dominio (**preguntas**, **encuestas**, **usuarios**).
- Estructuras temporales para construir matrices de datos utilizadas en *clustering*.
- Representaciones directas de filas provenientes de un CSV.

3.1.2 Map y HashMap

La estructura clave del dominio es **Map**, implementado mediante **HashMap**. Se utiliza para:

- Guardar encuestas por identificador en ***GestorEncuestas***
  Map<String, Encuesta> encuestas
- Guardar usuarios por identificador en ***GestorUsuarios***
  Map<String, UsuarioRespondedor> y *Map<String, UsuarioAdmin>*
- Asociar a cada encuesta la lista de respuestas de un usuario
  Map<String, List<Respuesta>> respuestasPorEncuesta
- Mapear preguntas con respuestas al exportar a **CSV**
  Map<String, Respuesta> interno en *CtrlCSV.exportarEncuesta*
- Representar conjuntos gestionados por **ID** con acceso rápido

Ventajas de usar **HashMap**:

- Búsqueda y actualización en **O(1)** promedio
- Simplifica la lógica del código evitando recorridos secuenciales
- Reduce acoplamiento y aumenta legibilidad, usando simplemente *map.get(id)*

3.1.3 Set, HashSet y LinkedHashSet

Se utiliza la interfaz **Set** mediante dos implementaciones:

- **HashSet**, por ejemplo:
  - En preguntas de tipo *Ordinal* para definir opciones válidas → *Set<String> opciones*
  - En respuestas de categoría múltiple → *Set<String> opcionesSeleccionadas*

- **LinkedHashSet**, usado en:
  - CtrlCSV.extraerOpcionesPorColumna para recolectar opciones del CSV manteniendo orden

Finalidades principales de los *sets*:

1. **Definir dominios de valores válidos**
   - Utilizado en preguntas *ORDINAL*, *CATEGORIA_SIMPLE* y *CATEGORIA_MULTIPLE*

2. **Representar respuestas con múltiples opciones**
   - Evita duplicados de forma automática

La elección de **LinkedHashSet** asegura que el orden original del CSV se mantenga en la interfaz final.

3.1.4 Arrays y matrices (String[], Object[], Object[][], int[])

Aunque el dominio se basa en colecciones dinámicas, se utilizan **arrays** en partes numéricas:

- En *LectorCSV*
  - Cada fila → *String[]*
  - Cabeceras → *String[]*

- En *ResultadoClustering*
  - Grupos por usuario → *int[] groups*
  - Centros de clúster → *Object[][] centers*

- En *GestorClustering*
  - *prepararDatos()* genera una matriz *Object[][]* donde cada fila es un usuario y cada columna una respuesta

Motivos de uso:

- Acceso por índice **rápido** y **directo**
- Tamaño fijo → apropiado para vectores de características
- Facilita cálculos algorítmicos y numéricos

3.1.5 Otras estructuras

- ***List<List<String>>*** en *ResultadoClustering.getUsuariosPorGrupo*
  Para agrupar IDs por clúster.
- ***enum TipoPregunta***
  Para modelar tipos válidos de pregunta evitando errores por *strings*.
- ***UUID*** en Pregunta
  Para generar IDs únicos y controlados incluso tras importación de datos.

3.2 Algoritmos

3.2.1 Gestión de usuarios, encuestas y preguntas

En los gestores (*GestorUsuarios, GestorEncuestas*) se implementan algoritmos básicos de alta, baja y modificación:

- **Creación de encuestas**
  GestorEncuestas.crearEncuesta valida que el título no esté vacío, instancia la encuesta y la guarda en el *Map<String, Encuesta>*.
  Complejidad: **O(1)** para la inserción en el mapa.
- **Modificación de encuestas**
  modificarEncuesta comprueba si la encuesta existe y, si es así, actualiza título y/o descripción solo si no son *null*.

- **Gestión de preguntas**
    - *addPregunta* añade la pregunta a la lista interna de la encuesta.
    - *modificarPregunta* comprueba que el índice sea válido antes de sustituir una pregunta por otra.
    - *eliminarPregunta* encapsula la lógica de borrado de una pregunta concreta, capturando posibles *IndexOutOfBoundsException*.

En GestorUsuarios se sigue un patrón similar:

- Antes de crear un usuario se comprueba con *existeUsuario(id)* que no haya otro con el mismo identificador.
- La obtención de usuarios que han respondido a una encuesta (*obtenerUsuariosQueRespondieron*) recorre todos los usuarios y consulta en cada uno si tiene respuestas para ese id de encuesta. Es un algoritmo lineal respecto al número de usuarios.

3.2.2 Validación de respuestas por tipo de pregunta

La clase abstracta *Pregunta* define el contrato *boolean validarRespuesta(Object respuesta)*, que cada subtipo implementa según su semántica.

- Preguntas numéricas (*Numerica*)

La validación sigue los pasos:

1. Si valor es *null*, solo se acepta si la pregunta no es obligatoria (*!esObligatoria()*).
2. Si el valor es un *Number*, se convierte a *double*.
3. Si es un *String*, se hace un *trim()* y se intenta *parsear* con *Double.valueOf*. Si lanza *NumberFormatException*, la respuesta no es válida.
4. Se descartan explícitamente valores *NaN* o *infinitos*.
5. Si hay **mínimos** y **máximos** definidos (min y max no nulos), se comprueba que el valor esté dentro del rango. En caso contrario se lanza una excepción (*IllegalArgumentException*) que permite detectar datos inconsistentes.

Este algoritmo garantiza que todas las respuestas numéricas que llegan al sistema sean realmente números y estén dentro de los límites definidos al crear la pregunta.

- Preguntas de texto libre (*Libre*)

En *Libre.validarRespuesta*:

1. Si el valor es *null*, se acepta solo si la pregunta no es obligatoria.
2. En caso contrario, se convierte a *String* y se comprueba que su longitud no supere *longitudMaxima*.

Es un algoritmo sencillo de coste **O(n)** en la longitud del texto, suficiente para controlar que el usuario no escriba respuestas excesivamente largas.

- Preguntas ordinales (*Ordinal*)

La validación de una respuesta ordinal consiste en:

1. Tratar *null* y cadenas vacías como "no responde" y aceptar solo si la pregunta no es obligatoria.
2. Comprobar que el valor sea un *String*.
3. Verificar que *opciones.contains(valor)*.

El uso de un *Set<String>* hace que esta comprobación sea **O(1)** promedio. La lógica de orden (la semántica de "más alto" o "más bajo") se utiliza posteriormente en el *clustering*, pero la validación se limita a garantizar que la opción existe.

Otras preguntas categóricas (simple y múltiple), aunque no se muestran aquí, siguen una idea similar: comprobar pertenencia a un conjunto de opciones y, en el caso de selección múltiple, comprobar que el número de opciones marcadas no supere el máximo permitido.

3.2.3 Algoritmos de respuesta a encuestas

La clase *GestorRespuestas* centraliza la lógica de registro de respuestas:

- *responderPregunta(Usuario usuario, Encuesta encuesta, String idPregunta, Object valor)*:
    1. Localiza la pregunta dentro de la encuesta usando *encontrarPregunta*.
    2. Llama a *pregunta.validarRespuesta(valor)*. Si devuelve *false*, se lanza una **excepción**.
    3. Crea un objeto *Respuesta* con ids de usuario, pregunta y encuesta.
    4. Registra esta respuesta en el propio usuario mediante *usuario.addRespuesta*.

El coste de este algoritmo es **lineal** en el número de preguntas de la encuesta, ya que la búsqueda de la pregunta se hace recorriendo la lista (*for (Pregunta p : encuesta.getPreguntas())*).

- *responderEncuesta(Usuario usuario, Encuesta encuesta, Map<String, Object> respuestasPorPregunta*):
    1. Primero comprueba que todas las preguntas obligatorias tengan una entrada en el mapa de respuestas.
    2. Después recorre las entradas del mapa y va llamando a *responderPregunta* para cada pregunta, capturando posibles errores para no abortar todo el proceso.

Este algoritmo es **lineal** en el número de respuestas que llegan en el mapa, y se apoya en la validación específica de cada tipo de pregunta.

3.2.4 Importación y exportación de encuestas en CSV

El controlador *CtrlCSV* encapsula dos algoritmos importantes: **exportación** e **importación**.

Exportación *(exportarEncuesta)*

A grandes rasgos, el proceso es:

1. Recuperar la encuesta y sus preguntas.
2. Buscar todos los usuarios que han respondido a esa encuesta.
3. Construir la primera fila del CSV con los tipos de pregunta (NUMERICA, ORDINAL, etc.).
4. Para cada usuario:
    - Se construye un mapa *idPregunta → Respuesta* para acceder rápido.
    - Se recorre la lista de preguntas en orden, rellenando un *String[]* con el valor formateado de cada respuesta mediante *formatearRespuestaParaCSV*.
5. Finalmente, se llama a EscritorCSV para volcar los encabezados y todas las filas en el fichero.

El algoritmo recorre usuarios × preguntas, es decir, su coste es **O(U·P)** donde U es el número de usuarios que han respondido y P el número de preguntas.

Importación *(importarCSV)*

La importación es algo más elaborada:

1. Se lee el archivo CSV con *LectorCSV.leerConEncabezados*, obteniendo:
    - Una primera fila con los tipos de pregunta.
    - Una lista de filas con las respuestas de cada usuario.
2. Se crea una nueva Encuesta con el título y descripción proporcionados.
3. Se llama a *extraerOpcionesPorColumna*:
    - Para cada columna se recorre todas las filas y se guardan los valores distintos en un *LinkedHashSet*.
    - En las columnas de tipo CATEGORIA_MULTIPLE se separan los valores por comas para extraer todas las opciones posibles.
    - El resultado es una lista de conjuntos de opciones reales por columna.
4. Se construyen las preguntas reales a partir de los tipos y de las opciones calculadas en el paso anterior:
    - Para NUMERICA se calcula el mínimo y máximo de los valores presentes y se da un pequeño margen.
    - Para tipos ordinales y categóricos se crean los conjuntos de opciones directamente a partir de los valores diferentes que aparecen en el CSV.
    - Para LIBRE se toma como referencia la longitud máxima observada y se le añade un margen.

5. Se procesan las filas de respuestas (*procesarRespuestasCSV*):
    - Para cada fila se crea un usuario respondedor "ficticio" (*user_csv_i*).
    - Se convierten los valores de la fila al tipo adecuado según la pregunta (*convertirValor*).
    - Se registran las respuestas usando *gestorRespuestas.responderPregunta*.

Este algoritmo también tiene coste **O(U·P),** pero con cierta sobrecarga inicial para calcular opciones y rangos a partir de los datos.

Además, el método *validarCSV* permite comprobar antes de importar que:

- Hay al menos una columna y una fila de datos.
- Todos los tipos de pregunta en la primera línea pertenecen al conjunto permitido.

3.2.5 Preparación de datos para el clustering

El componente *GestorClustering* se encarga de transformar las respuestas en una representación numérica adecuada para los algoritmos de agrupamiento:

1. Construcción de la matriz de datos (*prepararDatos*)
    - Recorre la lista de usuarios que han respondido.
    - Para cada usuario:
        - Llama a *convertirRespuestasAArray*, que:
            - Obtiene las respuestas del usuario en esa encuesta.
            - Para cada respuesta, calcula el índice de la pregunta con *obtenerIndicePregunta*.
            - Inserta el valor en la posición adecuada del array.
        - Si alguna posición queda a *null* (es decir, el usuario no respondió todas las preguntas), se utiliza un algoritmo de *Machine Learning* no supervisado llamado *KNN* que es capaz de rellenar los *nulls* con valores que se asemejan a los k puntos (o en este caso respuestas) más cercanos.
    - El resultado final es un *Object[][]* donde:
        - Cada fila representa un usuario.
        - Cada columna representa una pregunta.
    - Paralelamente se guarda la lista de ids de usuario en el mismo orden para poder reconstruir los grupos después.

2. Configuración del algoritmo (*configurarAlgoritmo*)
    - Se recorre la lista de preguntas de la encuesta:
        - Si la pregunta es Numerica, se llama a *clustering.configurarRangoNumerico* con los valores **min** y **max**.
        - Si la pregunta es Ordinal, se llama a *clustering.configurarOpcionesOrdinales* con el conjunto de opciones.
    - De esta forma, la lógica de distancia/similitud interna del algoritmo de clustering conoce el dominio de cada dimensión.

3. Extracción de tipos de pregunta (*extraerTiposPreguntas*)
    - Se construye un array *TipoPregunta[]* que se pasa al contexto de clustering para que el algoritmo sepa en cada dimensión si está tratando un número, una categoría, un ordinal o una respuesta libre.

3.2.6 Algoritmos de clustering y selección del mejor k

La clase *CtrlDominio* expone el método:

public *ResultadoClustering ejecutarClustering(String idEncuesta, String algoritmo, int k, int maxIter)*

Este método no solo lanza el *clustering*, sino que busca **el mejor número de grupos** entre 2 y k utilizando la medida de calidad *silhouette*:

1. Se obtiene la encuesta y la lista de usuarios que han respondido.
2. Para cada valor de i entre 2 y k:
    - Se crea una estrategia concreta (*KMeans, KMedoids o KMeans++*) a partir del string *algoritmo*.
    - Se instancia el contexto *Clustering* con esa estrategia.
    - Se llama a *gestorClustering.ejecutarClustering*, que devuelve un *ResultadoClustering*.
3. Se va guardando el resultado con mejor silhouette y al final se devuelve solo ese.

De este modo, el sistema no fija a priori un único valor de k, sino que explora varios y se queda con el que mejor agrupa a los usuarios según sus respuestas.

3.2.7 Generación de grupos de usuarios

Finalmente, la clase *ResultadoClustering* incluye un método auxiliar:

public List<List<String>> getUsuariosPorGrupo()

Este método reconstruye la estructura grupo → lista de usuarios de forma eficiente:

1. Crea una lista de listas, una por cada grupo (de 0 a k−1).
2. Recorre el array *groups*, que indica el número de grupo asignado a cada usuario.
3. Usa idsUsuarios.get(i) para recuperar el id del usuario correspondiente y lo añade a la lista de su grupo.

Esto facilita la presentación de los clústeres en la interfaz o su exportación a formatos como CSV.

4. Algoritmos de Clustering

4.1 ResultadoClustering

En esta clase encapsulamos y agrupamos todo el resultado del proceso ***clustering*** y sirve como transferencia de datos hacia la **capa de dominio** o de **presentación**.

Contenido:

- *int[] groups*
  Vector que indica, para cada usuario (fila del dataset), el clúster asignado.
- *Object[][] centers*
  - En *KMeans/KMeans++* → centroides.
  - En *KMedoids* → los usuarios que son medoides.
- *double silhouette*
  Silhouette medio del clustering final.
- *List<String> idsUsuarios*
  Mantiene el "mapeo" fila → id original del usuario, necesario para que la matriz *Object[][]* no pierda la información de **identidad**.
- Metadades:
  - Nombre del algorismo utilizado.
  - *k*
  - Número de iteraciones totales.

Métodos auxiliares

- *obtenerUsuariosPorGrupo()*

Devuelve una estructura *List<List<String>>* donde cada lista contiene los **IDs** de los usuarios que pertenecen a cada cluster, esto es útil para mostrar los resultados en una interfaz o incluso si queremos exportarlos a un **CSV**.

4.2 AlgoritmoClustering

La capa del clustering sigue el patrón **estrategia** que nos permite encapsular diferentes algorismos en una misma interfaz (*AlgoritmoClustering*). Esto facilita cambiar de algorismo sin modificar código externo y, además, añade extensibilidad para futuros métodos de clustering que queramos añadir.

Hay **tres** implementaciones: *KMeans*, *KMeansPlusPlus* y *KMedoids* junto a la clase auxiliar *ResultadoClustering*

4.3 KMeans

Estructuras de datos principales

1. *int k* → número de **clusters** deseados.
2. *int maxIter* → número máximo de **iteraciones** del proceso.
3. *Random random* → utilizado para la inicialización **aleatoria**.
4. *Object[][] centers* → matriz con los centroides (un vector para cada cluster).
5. *int[] groups* → para cada usuario, el índice del clúster al cual ha sido asignado.
6. Información adicional heredada de *AlgoritmoClustering*:
    - *TipoPregunta[] questionTypes*
    - Diccionarios con rangos números, opciones ordenadas, etc.

Algorismo detallado:

1. **Inicialización de los centroides**
    - Se escogen *k* usuarios aleatoriamente del dataset y sus respuestas se copian como centroides iniciales.
    - Esto como ventaja tiene que el **coste** es bajo
    - Como desventaja tiene que puede dar soluciones de **baja calidad** si la cantidad de puntos iniciales están muy juntos
2. **Bucle iterativo**

    Se repite hasta que
    - No hay cambios en las asignaciones o se ha llegado a *maxIter*

    La iteración tiene dos pasos:

    a. **Asignación a los clusters**
        - Para cada usuario se calcula la distancia a cada centro usando *calculateDistance*.
        - La distancia es mixta, depende el tipo de la pregunta:
            - Numérica: diferencia normalizada
            - Ordinal: diferencia de posiciones
            - Categoria simple: 0/1
            - Categoria múltiple: distancia tipo Jaccard
            - Texto libre: similitud basada en Levenshtein
        - El usuario asigna el centro más cercano → **O(k·m)** por usuario

    b. **Recalcular los centroides:**
        - Para cada cluster y por cada pregunta:
            - Pregunta numérica: se calcula la media
            - Pregunta categórica simple: se coge la moda (la opción más frecuente)
            - Ordinal: se coge el valor con posición en la mitad
            - Categoría múltiple: se coge la unión/intersección ponderada
            - Texto libre: se usa un "centroide simbólico", escogiendo el texto más representativo
        - Los nuevos centros se guardan a centers

3. **Criterio de parada**
    - Si *groups* no cambia en toda una iteración, se asume convergencia

4. **Evaluación con Silhouette**
    - Para cada usuario se calcula:
        - distancia media a su cluster (a)
        - distancia mínima a sus otros clusters (b)
        - silhouette = (b - a) / max(a, b)
    - La media de todos los silhouettes da la calidad global

Observaciones

- Complejidad general:
  O(maxIter · n · k · m)
- Muy eficiente en datasets grandes pero sensible a los puntos iniciales.

4.4 KMeansPlusPlus

Es una versión mejorada de *KMeans* que mantiene exactamente el mismo núcleo de asignación y recomputación, pero substituye la inicialización

Diferencias principales

- En lugar de escoger *k* muestras aleatorias, se usa una inicialización por probabilidad
- Para cada centro nuevo
    1. El primero se escoge **aleatoriamente**.
    2. Se calcula, para cada usuario, la **distancia al centro más cercano** ya seleccionado.
    3. Se escoge el siguiente centro con una probabilidad **proporcional** a la **distància²**.

Esto reduce drásticamente la probabilidad de obtener centroides muy juntos entre sí ya que tiende a colocar los primeros centros en zonas muy separadas del dataset. Por ende acostumbra a producir unas mejores soluciones y llegamos a necesitar menos iteraciones para dejar de tener cambios. El resto del bucle (asignación, recomputación, criterio de parada y silhouette) son idénticos al *KMeans* tradicional que hemos mencionado anteriormente.

4.5 KMedoids

A diferencia de *KMeans*, que calcula centroides "virtuales", *KMedoids* selecciona medoides reales del dataset (son usuarios reales que representan el cluster). Con esto conseguimos dos cosas:

- Más robusto a los valores extremos.
- Más interpretables los centros (son usuarios existentes)

Estructuras de datos

- *int[] medoidIndices* → índices de los usuarios que son los medoides.
- *int[] groups* → asignación usuario → clúster.
- Comparteix amb *KMeans*:
    - *TipoPregunta[] questionTypes*
    - Diccionario de rangos y opciones
    - Función *calculateDistance*

Algorismo detallado

1. Inicialización de **medoides** con una variante de ***k-medoids***++.
    - Se escoge un medoide inicial aleatorio.
    - El resto se escogen parecido a *KMeans*++ pero asegurando que no existan duplicados, esto se hace gracias a un *Set<Integer>*.
    - Se reduce el riesgo de escoger dos medoides muy cercanos.

2. **Bucle principal**:

Como hemos comentado ya, hasta convergencia o *maxIter*.

   a. **Asignación**
      - Cada usuario se asigna al medoide más cercano.
      - Equivalente a *KMeans* pero con medoides en vez de centroides.
   b. **Actualización de los medoides**
      - Per a cada clúster:
        1. Se consideran todos sus miembros como posibles candidatos a nuevos medoides.
        2. Para cada candidato se calcula **suma**(distancias del candidato a todos los miembros del cluster).
        3. Se escoge el candidato que **minimiza** esta suma (**medoide óptimo local**).
      - Esto es más costoso que un *KMeans* pero es mucho más **robusto**.
   c. **Gestión de clusters vacíos**
      - Si un cluster queda sin usuarios: se selecciona el nuevo medoide entre los puntos que no han sido utilizados hasta ahora, para evitar duplicados utilizamos un *Set<Integer>*.

3. **Cálculo de silhouette**
    - Lo hacemos igual que en *KMeans*:
        - Para cada usuario se calcula:
            1. distancia media a su cluster (a)
            2. distancia mínima a sus otros clusters (b)
            3. silhouette = (b - a) / max(a, b)

        - La media de todos los silhouettes da la calidad global
    - Se utiliza para dar una medida de la **calidad** del clustering final.

Observaciones

- Complejidad aproximada:

  O(maxIter · k · (n/k)² · m)

- Muy útil cuando
    - El dataset tiene valores extremos.
    - Las variables son categóricas o mixtas.
    - Se quieren centros interpretables

5. Jocs de proves

5.1 TestGestorEncuestas

5.1.1 Objetivo de la prueba

Probar todas las operaciones *CRUD (Create, Read, Update, Delete)* del *GestorEncuestas:*

- Creación de encuestas con título y descripción
- Listado y consulta de encuestas existentes
- Modificación de datos de encuestas
- Eliminación de encuestas
- Gestión completa de preguntas (añadir, modificar, eliminar)
- Verificación de existencia de encuestas
- Estadísticas del sistema

Clases

- *GestorEncuestas* (clase principal del gestor)
- *Encuesta* (manipulamos indirectamente)
- Todas las subclases de *Pregunta: Numerica, Libre, CategoriaSimple, Ordinal, CategoriaMultiple*

5.1.2 Otros elementos integrados

- ***Scanner****:* para interacción con el usuario
- ***List, Set, Map***: estructuras de datos de Java utilizadas para gestionar colecciones.
- **Jerarquía de Pregunta**: prueba la creación y manipulación de diferentes tipos de preguntas

5.1.3 Archivos de datos necesarios

No se requiere ningún archivo externo, toda la información se genera de forma interactiva durante la ejecución.

5.1.4 Valores estudiados

Enfocamiento:
Caja Negra – Particiones equivalentes:

1. **Títulos de encuesta**
    - Títulos válidos no vacíos
    - Títulos con espacios al inicio o al final (verificar trim)
    - Títulos muy largos

2. **Preguntas numéricas**
    - Sin límites (min = null, max = null)
    - Solo mínimo definido Solo máximo definido
    - Ambos límites definidos
    - Límites inválidos (min > max)

3. **Preguntas de texto**
    - Longitud por defecto (1000 caracteres)
    - Longitud personalizada
    - Longitud cero o negativa (debe provocar error)

4. **Preguntas categóricas**
    - Con 2 opciones (caso mínimo)
    - Con varias opciones (3–10)
    - Opciones duplicadas
    - Opciones vacías
5. **Categoría múltiple**
    - *maxSelecciones* ≤ número de opciones
    - *maxSelecciones* > número de opciones (inválido)
    - *maxSelecciones* = 1 (equivale a categoría simple)

Caja Blanca – Cobertura de caminos

- Modificación con valores *null*: comprobar que los valores anteriores se mantienen
- Eliminación con confirmación: probar confirmación y cancelación
- Listas vacías: comprobar comportamiento sin encuestas/preguntas
- Índices de pregunta: fuera de rango, válidos y en los límites (0, size-1)

5.1.5 Efectos estudiados

- **Interfaz visual**
    - Uso de caracteres *Unicode* para cabeceras
    - Líneas separadoras
    - Iconos
    - Identación y formato tabular

- **Flujos de navegación**
    - Menú principal con 9 opciones
    - Submenú de gestión de preguntas
    - Opción de retorno "← Volver"
    - Confirmaciones en operaciones destructivas (eliminación)

- **Validación de entrada**
    - Manejo de enteros inválidos (método *leerEntero* con bucle)
    - Gestión de líneas vacías (*trim, isEmpty*)
    - Confirmaciones tipo **s/n**

- **Conservación de información**
    - IDs generados automáticamente
    - Estado consistente durante toda la sesión
    - Contadores actualizados (por ejemplo, número de preguntas)

- **Mensajes informativos**
    - Éxito
    - Error
    - Advertencias
    - Información general

5.1.6 Operativa

1. Crear una encuesta (Opción 1)

- Título: "Satisfacción del servicio"
- Descripción: "Evaluación de la calidad"
- Añadir preguntas → Sí
- Pregunta numérica: "Puntuación del 0 al 10" (min=0, max=10)
- Pregunta de texto: "Comentarios" (longitud=500)
- Finalizar: Opción 6

2. Listar encuestas (Opción 2)

- Verificar que aparece la encuesta recién creada
- Comprobar el recuento total

3. Ver detalle (Opción 3)

- Introducir ID
- Revisar que se muestran todas las preguntas

4. Gestionar preguntas (Opción 6)

- Añadir categoría simple
- Modificar una pregunta existente
- Eliminar una pregunta
- Confirmar los cambios con “Ver preguntas”

5. Modificar encuesta (Opción 4)

- Cambiar solo el título
- Dejar descripción vacía
- Verificar el resultado

6. Estadísticas (Opción 8)

- Verificar conteos internos

7. Eliminar encuesta (Opción 5)

- Probar cancelación (no escribir “CONFIRMAR”)
- Probar eliminación correcta (escribir “CONFIRMAR”)

Casos de error

- ID inexistente
- Números inválidos en cualquier menú
- Confirmaciones incorrectas
- Rangos numéricos inválidos (min > max)

5.2. TestGestorUsuarios

5.2.1 Objetivo de la prueba

Validar todas las operaciones del gestor de usuarios:

- Creación de usuarios respondedores (con y sin contraseña)
- Creación de administradores
- Consulta y listado
- Verificación de tipo y existencia
- Consulta de respuestas
- Estadísticas de actividad

Clases cubiertas

- *GestorUsuarios*
- *Usuario* (abstracta)
- *UsuarioRespondedor*
- *UsuarioAdmin*
- *Respuesta* (consulta indirecta)

5.2.2 Elementos adicionales

- Herencia: navegación por la jerarquía
- Polimorfismo con *instanceof*
- Uso de *List* y *Set* para almacenar usuarios y respuestas

5.2.3 Archivos de datos necesarios

Ninguno, el sistema funciona completamente en memoria.

5.2.4 Valores estudiados

Enfocamiento:
Caja Negra – Particiones equivalentes

- **IDs de usuario**
  - IDs válidos y únicos (alfanuméricos)
  - IDs duplicados (deben rechazarse)
  - IDs vacíos o solo espacios
  - IDs con caracteres especiales

- **Nombres**
  - Nombres completos válidos
  - Nombres con múltiples espacios
  - Nombres vacíos

- **Passwords**
  - Contraseñas normales
  - Contraseñas vacías (caso import CSV)
  - Contraseñas extremadamente cortas o largas

- **Tipos de usuario**
  - Respondedor con contraseña
  - Respondedor sin contraseña
  - Administrador

- **Valores límite**
  - Listas vacías
  - Usuarios sin encuestas respondidas
  - Usuarios con múltiples respuestas

5.2.5 Efectos estudiados

- **Separación por tipo**
    - Listado independiente de Administradores y Respondedores
    - Iconos distintos
    - Información específica por tipo

- **Actividad**
    - Número de encuestas respondidas
    - Listado de respuestas
    - Ranking de usuarios más activos (Top 3)

- **Validaciones**
    - Detección de IDs duplicados
    - Verificación del tipo (isAdmin)
    - Existencia del usuario

- **Estadísticas**
    - Promedios (encuestas/usuario, respuestas/encuesta)
    - Totales agregados
    - Ordenación por actividad

5.2.6 Operativa

- **Crear usuarios**
    - Admin: id=admin1
    - Respondedor: user1
    - Respondedor: user2
    - Respondedor sin pass (csv001)

- **Listar usuarios**
    - Verificar separación por tipo

- **Buscar usuario**
    - Examinar user1
    - Examinar admin1

- **Verificar si es admin**
    - admin1 → Sí
    - user1 → No

- **Verificar existencia**
    - user1 → existe
    - user999 → no existe

- **Ver respuestas**
    - Usuario sin respuestas
    - (Requiere haber ejecutado TestGestorRespuestas antes)

- **Estadísticas**
    - Totales
    - Promedios
    - Top 3

- **Casos de error**
    - Crear usuario con ID duplicado
    - Buscar ID vacío
    - Pedir respuestas de un admin

5.3. TestGestorRespuestas

5.3.1 Objetivo de la prueba

Validar la integración completa del sistema:

- Respuesta a preguntas individuales
- Respuesta a encuestas enteras
- Consulta por usuario y por encuesta
- Validación según tipo de pregunta
- Gestión de índices

Clases probadas

- *GestorRespuestas*
- *Respuesta*
- Integración con *GestorUsuarios* y *GestorEncuestas*

5.3.2 Integración

Este driver actúa como un test de integración completo.

Usa:

- *GestorEncuestas*
- *GestorUsuarios*
- Todas las clases de tipo *Pregunta*
- *Usuario* y derivados
- *Encuesta*

5.3.3 Archivos de datos necesarios

Ninguno, el método *crearDatosPrueba()* genera automáticamente:

- user1, user2, admin1
- Una encuesta “Encuesta de Satisfacción” con 4 preguntas:
    - Numérica
    - Categoría simple
    - Texto libre
    - Categoría múltiple

5.3.4 Valores estudiados

Validación por tipo de pregunta:

- **Numérica**
    - Valores dentro del rango
    - Valores fuera de rango
    - Enteros y decimales
    - Entrada no numérica

- **Texto libre**
    - Texto corto
    - Texto de límite (500 chars)
    - Texto excedido
    - Texto vacío

- **Categoría simple**
    - Opción válida
    - Opción fuera de rango
    - Respuesta nula

- **Categoría múltiple**
    - Selecciones válidas
    - Exceso de selecciones
    - Lista vacía
    - Duplicados
    - Índices inválidos

- **Obligatoriedad:**
    - Preguntas obligatorias sin responder
    - Opcionales sin responder
    - Combinación de ambas

Escenarios de integración:

- **Respuesta individual**
    - Usuario válido + encuesta válida + pregunta válida
    - Usuario inexistente
    - Encuesta inexistente
    - Índice fuera de rango
    - Pregunta que no pertenece a la encuesta

- **Respuesta completa**
    - Todas contestadas
    - Solo obligatorias
    - Falta una obligatoria
    - Un usuario respondiendo varias veces

- **Consultas**
    - Usuario con respuestas
    - Usuario sin respuestas
    - Encuesta con respuestas de varios usuarios
    - Encuesta sin respuestas

5.3.5 Efectos estudiados

- **Datos de prueba automáticos**
    - Mensajes informativos
    - IDs visibles

- **Interacción**
    - Prompts dependientes del tipo de pregunta
    - Indicadores de rango, límites y opciones

- **Visualización**
    - Formatos distintos por tipo
    - Separación por usuario
    - Agrupación por encuesta

- **Navegación compleja**
    - Usuario → encuesta → pregunta
    - Índices dinámicos

- **Validación en tiempo real**
    - Mensajes de error específicos
    - Correcciones en caso de error
    - Mensaje final de confirmación

5.3.6 Operativa

- **Flujo que vamos a seguir:**
    - Verificar datos generados
    - Responder pregunta individual
    - Responder encuesta completa
    - Ver respuestas de un usuario
    - Ver todas las respuestas
    - Obtener índice de pregunta
    - Ver estadísticas

- **Casos de error**
    - Responder como admin
    - Valor numérico fuera de rango
    - Exceso de opciones en selección múltiple
    - Omitir obligatorias
    - IDs inexistentes

5.4. TestGestorClustering

5.4.1 Objetivo de la prueba

Validar el funcionamiento completo del sistema de clustering y análisis de datos:

- Ejecución de algoritmos de clustering: *K-Means*, *K-Means++* y *K-Medoids*
- Comparación entre algoritmos con diferentes valores de K
- Cálculo de métricas de calidad: Silhouette e Inercia
- Conversión de respuestas a vectores numéricos
- Asignación de usuarios a grupos
- Medición de rendimiento temporal

Clases probadas

- *GestorClustering* (clase principal)
- *Clustering* (patrón Strategy)
- *AlgoritmoClustering* (interfaz)
- *KMeans* (implementación concreta)
- *KMeansPlusPlus* (implementación concreta)
- *KMedoids* (implementación concreta)
- *ResultadoClustering* (contenedor de resultados)

5.4.2 Integración con otros módulos

Este driver actúa como un **test de integración completo del sistema**, ya que requiere:

- *GestorUsuarios*: creación de usuarios respondedores
- *GestorEncuestas*: creación de encuestas con diferentes tipos de preguntas
- *GestorRespuestas*: almacenamiento y recuperación de respuestas
- Jerarquía de *Pregunta*: Numerica, Ordinal, CategoriaSimple
- *UsuarioRespondedor*: usuarios que proporcionan los datos a agrupar
- *Encuesta:* estructura que contiene las preguntas a analizar

5.4.3 Archivos de datos necesarios

Ninguno, el método *crearDatosPrueba()* genera automáticamente:

- **1 encuesta** "Encuesta de Preferencias" con 4 preguntas:
    - Pregunta numérica: "Edad" (rango 18-80)
    - Pregunta numérica: "Ingresos" (rango 0-10000)
    - Pregunta ordinal: "Estudios" (Primaria, Secundaria, Universidad, Postgrado)
    - Pregunta categórica simple: "Vehículo" (Sí/No)

- **10 usuarios** con perfiles diversos:
    - user1 a user10
    - Edades entre 20 y 50 años
    - Ingresos entre 800 y 5000
    - Diferentes niveles educativos
    - Con y sin vehículo

5.4.4 Valores estudiados

Caja Negra – Particiones equivalentes

- **Número de clusters (K)**
    - K = 2: clustering binario (mínimo práctico)
    - K = 3: clustering típico
    - K = N/2: clusters medianos
    - K ≥ N: clusters vacíos o degenerados (caso límite)
    - K = 1: todos en un grupo (caso trivial)

- **Número de iteraciones**
    - MaxIter = 10: convergencia rápida
    - MaxIter = 100: convergencia estándar
    - MaxIter = 1000: convergencia garantizada
    - MaxIter = 1: sin iteraciones (solo inicialización)

- **Tipos de datos en las preguntas**
    - Numéricas continuas: edad, ingresos (rangos amplios)
    - Ordinales: estudios (orden jerárquico)
    - Categóricas binarias: vehículo (Sí/No)
    - Combinaciones: todas juntas en la misma encuesta

- **Distribución de usuarios**
    - Grupos claramente separados: perfiles diferenciados
    - Grupos solapados: perfiles similares
    - *Outliers*: usuarios atípicos (ej: edad 50, ingresos 5000)
    - Distribución uniforme: usuarios homogéneos

Caja Blanca – Cobertura de caminos

- **Inicialización de centroides**
    - *K-Means:* selección aleatoria
    - *K-Means++:* selección probabilística (mejor distribución inicial)
    - *K-Medoids:* selección de puntos reales como centros

- **Proceso iterativo**
    - Asignación → Actualización: ciclo completo
    - Convergencia: centroides estables antes de maxIter
    - Sin convergencia: maxIter alcanzado
    - Clusters vacíos: reasignación necesaria

- **Cálculo de métricas**
    - Silhouette > 0.7: clustering excelente
    - Silhouette 0.5-0.7: clustering aceptable
    - Silhouette < 0.5: clustering pobre
    - Inercia: suma de distancias cuadradas (debe disminuir con K)

- **Conversión de respuestas**
    - Valores numéricos: uso directo
    - Valores ordinales: mapeo a índices (0, 1, 2, 3)
    - Valores categóricos: codificación binaria (0/1)
    - Valores *null*: manejo de datos faltantes

5.4.5 Efectos estudiados

- **Datos de prueba automáticos**
    - Generación de 10 usuarios con perfiles realistas
    - Diversidad en todas las dimensiones
    - Patrones detectables para validar clustering
    - Mensajes informativos de creación

- **Patrón Strategy**
    - Intercambiabilidad de algoritmos
    - Misma interfaz para diferentes implementaciones
    - Fácil extensión con nuevos algoritmos

- **Medición de rendimiento**
    - Tiempo de ejecución en milisegundos
    - Comparación de eficiencia entre algoritmos
    - Número de iteraciones hasta convergencia

- **Visualización de resultados**
    - Métricas de calidad claramente presentadas
    - Usuarios agrupados por cluster
    - Comparativa tabulada entre algoritmos
    - Formato estructurado y legible

- **Métricas de calidad**
    - Silhouette Score: cohesión intra-cluster vs separación inter-cluster
    - Inercia (WCSS): suma de distancias al cuadrado dentro de clusters
    - Número de iteraciones: indicador de convergencia

- **Comparación sistemática**
    - Evaluación con múltiples valores de K (2 a K_max)
    - Comparación de los 3 algoritmos para cada K
    - Identificación del K óptimo (método del codo)

5.4.6 Operativa

Flujo que vamos a seguir:

1. Verificar datos generados

   Acción: El sistema crea automáticamente los datos al iniciar

   Verificar:

   - Mensaje "10 usuarios creados"
   - Mensaje "Encuesta lista para clustering"

2. Ejecutar K-Means (Opción 1)

   Parámetros:

   - K: 3
   - MaxIter: 100

Verificar:

- Algoritmo: "KMeans"
- K: 3
- Iteraciones: ≤ 100 (debe converger antes)
- Silhouette: entre 0.3 y 0.8
- Inercia: valor positivo
- Tiempo: < 100ms para 10 usuarios
- 3 grupos con usuarios distribuidos

3. Ejecutar K-Means++ (Opción 2)

   Parámetros:

   - K: 3
   - MaxIter: 100

   Verificar:

   - Silhouette generalmente > K-Means (mejor inicialización)
   - Menos iteraciones hasta convergencia
   - Distribución más equilibrada de usuarios

4. Ejecutar K-Medoids (Opción 3)

   Parámetros:

   - K: 3
   - MaxIter: 100

   Verificar:

   - Centros son usuarios reales (no promedios)
   - Mayor tiempo de ejecución que K-Means
   - Robustez ante outliers

5. Comparar algoritmos (Opción 4)

   Parámetros:

   - K máximo: 5

   Verificar:

   - Ejecución para K = 2, 3, 4, 5
   - Comparativa de 3 algoritmos por cada K
   - Silhouette disminuye con K alto (sobreajuste)
   - Inercia siempre disminuye con K mayor
   - Formato tabular de comparación

Casos de error a probar

- Valores de K inválidos
    - K = 0: debe producir error o comportamiento indefinido
    - K negativo: debe producir error
    - K > número de usuarios: clusters vacíos o errores

- *MaxIter* insuficiente
    - MaxIter = 1: sin convergencia, resultados pobres
    - MaxIter = 0: debe producir error

- Entrada de usuario
    - Valor no numérico en K o MaxIter: bucle de reintento
    - Valores muy grandes (K=1000, MaxIter=10000): tiempo excesivo

- Encuestas sin respuestas
    - Lista de usuarios vacía: debe manejar el caso gracefully

- Preguntas sin variabilidad
    - Todos los usuarios con mismas respuestas: clustering degenerado

Escenarios avanzados

- Convergencia
    - Verificar que el algoritmo se detiene cuando los centroides no cambian
    - Comprobar que iteraciones < maxIter indica convergencia exitosa

- Estabilidad
    - Ejecutar el mismo algoritmo varias veces
    - K-Means puede dar resultados diferentes (aleatorización)
    - K-Means++ debería ser más consistente

- Escalabilidad
    - Con 10 usuarios: respuesta inmediata
    - Tiempo de ejecución debe ser < 1 segundo

- Calidad del clustering
    - Para los datos generados, **K = 3** debería dar mejor Silhouette
    - Grupos esperados: bajo/medio/alto ingresos + educación correlacionada
    - Vehículo y edad como factores secundarios

Verificaciones de integración

- Con *GestorRespuestas*
    - Recuperación correcta de todas las respuestas
    - Manejo de preguntas obligatorias vs opcionales
    - Conversión correcta a vectores numéricos

- Con *GestorEncuestas*
    - Obtención correcta del orden de preguntas
    - Acceso a metadatos (rangos, opciones)

- Con *GestorUsuarios*
    - Filtrado de usuarios que respondieron
    - Exclusión de administradores
    - Manejo de listas vacías

Resultados esperados

Para K=3 con los datos de prueba:

- Silhouette Score: 0.4 - 0.7 (clustering razonable)
- Grupos típicos:
    - Grupo 0: jóvenes universitarios, ingresos bajos (user3, user7, user9)
    - Grupo 1: adultos con postgrado, ingresos medios (user1, user5)
    - Grupo 2: profesionales senior, ingresos altos (user2, user4, user6, user8, user10)

Métricas de rendimiento:

- *K-Means++* > *K-Means* en Silhouette
- *K-Medoids* similar en calidad pero más lento
- Todos convergen en < 50 iteraciones con datos limpios

5.4. Resumen de cobertura

- Cobertura de clases
    - *GestorEncuestas*
    - *GestorUsuarios*
    - *GestorRespuestas*
    - *GestorClustering*
    - *Encuesta*
    - *Usuario / UsuarioAdmin / UsuarioRespondedor*
    - Todas las clases de *Pregunta*
    - *Respuesta*
- Cobertura de casos de uso
    - Creación y gestión de encuestas
    - Creación y edición de preguntas
    - Gestión de usuarios
    - Respuestas individuales y completas
    - Consultas y estadísticas
    - Validaciones de entrada
    - Navegación y confirmaciones

- Método de prueba
    - Caja Negra: particiones equivalentes y valores límite
    - Caja Blanca: caminos críticos
    - Integración: enfoque *top-down* con datos generados

Calidad de los drivers:

- Puntos fuertes
    - Interfaz clara y cuidada
    - Manejo robusto de errores
    - Cobertura amplia
    - Drivers fáciles de usar
    - Mensajes informativos y consistentes
- Mejoras posibles
    - Añadir *logs* automáticos
    - Ejecutar baterías de prueba automáticas
    - Comparar resultados esperados *vs* obtenidos

Reparto del trabajo

Mohamed Amara	Arnau Serra	Adam Ziani	Ossama Chaer	Andreea Cerchia
CtrlDominio	Gestor Clustering	Pregunta	Kmeans	CategoriaSimple
GestorUsuarios	Kmedoids	Numerica	Kmeans++	Encuesta
GestorRespuestas	EscritorCSV	Ordinal	Resultado Clustering	CategoriaMultipleTest
UsuarioAdmin	CtrlCSV	Libre	TestGestorClustering	CategoriaSimpleTest
UsuarioRespondedor	Clustering	Categoria Multiple	TestKMeans	ClusteringTest
Usuario	Test Gestor Respuestas	Gestor Encuesta	TestKMeans++	EncuestaTest
Respuesta	Test Gestor Usuarios	TestGestorEncuesta	TestKMedoids	LibreTest
AlgortimoClustering (Strategy)		TipoPregunta		NumericaTest
TestControladores				OrdinalTest
LectorCSV				RespuestaTest
				ResultadoClusteringTest
				UsuarioAdminTest
				UsuarioRespondedorTest