Proyectos de Programación

Equipo 31.2
Versión 2.0

Extractor de prototipos de comportamiento o perfiles

DOCUMENTACIÓN

Este documento contiene el diseño arquitectónico en 3 capas (Dominio, Presentación y Persistencia), junto con la documentación de estructuras de datos y algoritmos mejorados respecto a la primera entrega.

INTEGRANTES

Yassin El Yemlahi
yassin.el.yemlahi@estudiantat.upc.edu

Anna Campderros
anna.campderras@estudiantat.upc.edu

Aleks Shahverdyan
aleks.shahverdyan@estudiantat.upc.edu

Pol Giralt Salarich
pol.giralt.salarich@estudiantat.upc.edu

Santiago Riera
santiago.riera@estudiantat.upc.edu

Índice

1. Descripción del diagrama de casos de uso

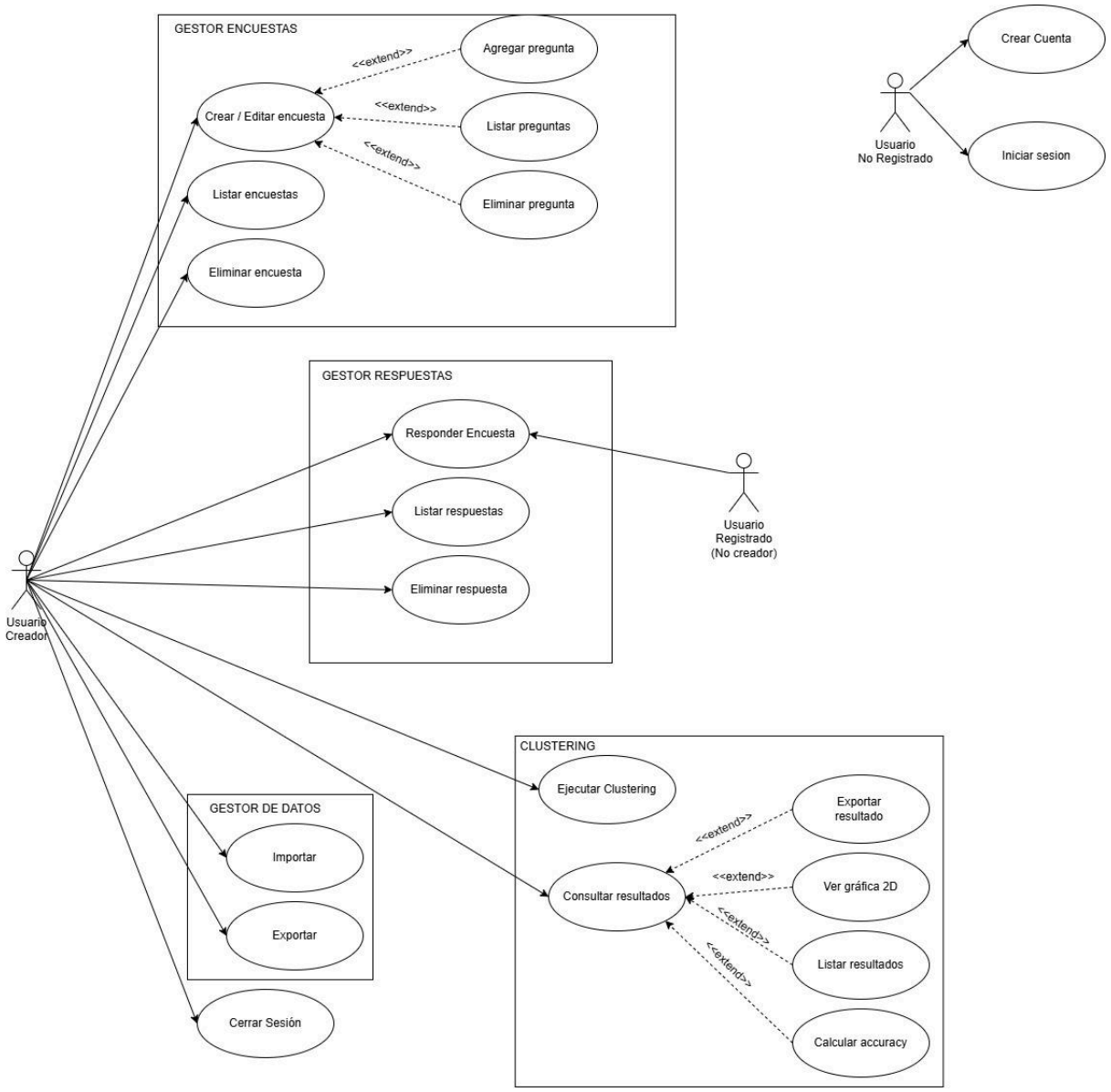

Diagrama de casos de uso. El diagrama se puede ver com más detalle en el archivo Diagrama_Casos_Uso de la carpeta Diagramas

1.1. Gestor de encuestas

Caso de uso #1 - Crear / Editar encuesta

Actor: Usuario Creador

Precondición: El usuario ha iniciado sesión correctamente.

Detonante: El usuario selecciona la opción "Crear nueva encuesta" o selecciona una existente para "Editar".

Escenario Principal:

1. El sistema solicita el título de la encuesta y una descripción opcional.
2. El usuario introduce el título y la descripción.
3. El sistema valida que el título no esté vacío.
4. El sistema guarda la cabecera de la encuesta.
5. El sistema muestra el menú de edición de la encuesta con las opciones disponibles (Agregar, Listar, Eliminar preguntas).
6. El usuario selecciona "Guardar y Salir" cuando ha terminado de modificar.
7. El sistema confirma el guardado final de la encuesta.

Extensiones:

3a) El usuario deja el título vacío: El sistema muestra un error indicando que el título es obligatorio y vuelve al paso 1.

Caso de uso #2 - Listar encuestas

Actor: Usuario Creador **Precondición:** El usuario ha iniciado sesión. **Detonante:** El usuario selecciona la opción "Ver mis encuestas".

Escenario Principal:

1. El sistema recupera todas las encuestas asociadas al ID del usuario creador.
2. El sistema muestra una lista con los títulos y fechas de creación de las encuestas.
3. El usuario visualiza la lista.

Extensiones:

1a) No existen encuestas creadas: El sistema muestra un mensaje indicando "No se encontraron encuestas" y ofrece la opción de crear una nueva.

Caso de uso #3 - Eliminar encuesta

Actor: Usuario Creador **Precondición:** El usuario ha listado las encuestas (Caso de uso #2). **Detonante:** El usuario selecciona una encuesta y pulsa el botón "Eliminar".

Escenario Principal:

1. El sistema solicita confirmación para borrar la encuesta y todos sus datos asociados.
2. El usuario confirma la operación.
3. El sistema elimina la encuesta de la base de datos.
4. El sistema muestra un mensaje de éxito y actualiza la lista de encuestas.

Extensiones:

2a) El usuario cancela la confirmación: El sistema aborta la operación y regresa al listado sin cambios.

Caso de uso #4 - Agregar pregunta

Actor: Usuario Creador

Precondición: El usuario se encuentra dentro del caso de uso #1 (Creando/Editando encuesta).

Detonante: El usuario selecciona la opción "Agregar pregunta".

Escenario Principal:

1. El sistema solicita el texto/identificador de la nueva pregunta.
2. El usuario introduce un enunciado no vacío.
3. El sistema comprueba que no exista ya una pregunta con ese identificador en la encuesta.
4. El sistema solicita el tipo (NUMERICA, TEXTO, OPCION_UNICA, OPCION_MULTIPLE).
5. El usuario introduce un tipo válido.
6. Si es de opción, el sistema pide las opciones separadas por “|”.
7. El usuario introduce los datos solicitados y si es de opción el número de opciones seguido de las opciones separadas por “|”.
8. El sistema crea la pregunta y la añade a la encuesta.

Extensiones:

2a) El usuario introduce un enunciado vacío: el sistema muestra un mensaje de error indicando que la pregunta no puede estar vacía y cancela la operación.

3a) El usuario introduce un identificador que ya existe: el sistema informa de que ya hay una pregunta con ese texto y cancela la operación.

5a) El usuario introduce un tipo inválido: el sistema muestra los tipos válidos disponibles y vuelve a solicitar el tipo (vuelve al paso 4).

7a) El usuario introduce un valor no numérico para el máximo de selección: el sistema informa de que el valor no es válido, establece el límite en 0 (sin límite).

7b) El usuario introduce un valor negativo para el máximo de selección: el sistema informa de que no puede ser negativo, establece el límite en 0 y continúa.

Caso de uso #5 - Listar preguntas

Precondición: El usuario se encuentra dentro del caso de uso #1 (Creando/Editando encuesta).

Detonante: El usuario selecciona la opción "Listar preguntas".

Escenario Principal:

1. El sistema recupera todas las preguntas asociadas a la encuesta que se está editando actualmente.
2. El sistema muestra una lista con el identificador y el tipo de cada pregunta.
3. El usuario visualiza la lista de preguntas existentes.

Extensiones:

1a) La encuesta no tiene preguntas todavía: El sistema muestra un mensaje indicando que la encuesta está vacía y vuelve al menú de edición.

Caso de uso #6 - Eliminar pregunta

Actor: Usuario Creador

Precondición: El usuario se encuentra dentro del caso de uso #1 (Creando/Editando encuesta).

Detonante: El usuario selecciona la opción "Eliminar pregunta".

Escenario Principal:

1. El sistema solicita el identificador de la pregunta a eliminar.
2. El usuario introduce el identificador de la pregunta.
3. El sistema valida que la pregunta existe en la encuesta actual.
4. El sistema elimina la pregunta de la encuesta.
5. El sistema confirma la eliminación y muestra la lista de preguntas actualizada.

Extensiones:

3a) El usuario introduce un identificador que no existe: El sistema muestra un mensaje de error indicando que no se encontró ninguna pregunta con ese identificador y vuelve al paso 1.

1.2. Gestor de respuestas

Caso de uso #7 - Responder encuesta

Actor: Usuario Registrado (No creador)

Precondición: El usuario ha iniciado sesión y tiene acceso a una encuesta activa.

Detonante: El usuario selecciona una encuesta para responder.

Escenario Principal:

1. El sistema carga las preguntas de la encuesta seleccionada.
2. El sistema presenta la primera pregunta al usuario.
3. El usuario introduce o selecciona su respuesta.
4. El sistema valida el formato de la respuesta.
5. El sistema avanza a la siguiente pregunta.
6. El usuario envía las respuestas finales.
7. El sistema guarda las respuestas asociadas al usuario y a la encuesta.

Extensiones:

4a) El usuario introduce un formato inválido: El sistema muestra error y solicita corregir la entrada.

Caso de uso #8 - Listar respuestas

Actor: Usuario Creador

Precondición: El usuario se encuentra gestionando una encuesta específica.

Detonante: El usuario selecciona la opción "Listar respuestas".

Escenario Principal:

1. El sistema consulta la base de datos buscando respuestas para la encuesta actual.
2. El sistema muestra un listado de las respuestas registradas.
3. El usuario selecciona una respuesta para ver el detalle.
4. El sistema muestra el detalle de las respuestas.

Extensiones:

1a) La encuesta no tiene respuestas aún: El sistema informa que aún no hay respuestas para esta encuesta.

Caso de uso #9 - Eliminar respuesta

Actor: Usuario Creador

Precondición: El usuario está visualizando el listado de respuestas (Caso de uso #8).

Detonante: El usuario selecciona una respuesta específica y pulsa "Eliminar".

Escenario Principal:

1. El sistema solicita confirmación para eliminar esa respuesta concreta.
2. El usuario confirma.
3. El sistema borra la respuesta de la base de datos.
4. El sistema actualiza el listado de respuestas.

Extensiones:

2a) El usuario cancela la eliminación: El sistema vuelve al listado sin realizar cambios.

1.3. Clustering

Caso de uso #10 - Ejecutar clustering

Actor: Usuario Creador

Precondición: Existe una encuesta con respuestas suficientes para analizar.

Detonante: El usuario selecciona "Ejecutar Clustering".

Escenario Principal:

1. El sistema solicita los parámetros para el algoritmo (ej. número de clusters k).
2. El usuario introduce los parámetros deseados.
3. El sistema valida los parámetros.
4. El sistema ejecuta el algoritmo de clustering sobre los datos de las respuestas.
5. El sistema notifica que el análisis ha finalizado correctamente.
6. El sistema muestra las opciones para consultar los resultados.

Extensiones:

3a) El usuario introduce un número de clusters inválido: El sistema muestra error y pide reingresar el valor.

Caso de uso #11 - Consultar resultados

Actor: Usuario Creador

Precondición: Se ha ejecutado un proceso de clustering (Caso de uso #10).

Detonante: El usuario accede a la vista de resultados.

Escenario Principal:

1. El sistema recupera los resultados del último análisis.
2. El sistema muestra el menú de resultados con las opciones disponibles (Exportar, Ver gráfica, Listar, Accuracy).
3. El usuario visualiza el resumen general.

Caso de uso #12 - Exportar resultado

Actor: Usuario Creador

Precondición: El usuario se encuentra en la vista de resultados (Caso de uso #11).

Detonante: El usuario selecciona la opción "Exportar resultado".

Escenario Principal:

1. El sistema solicita el formato de exportación deseado (CSV, PDF, JSON).
2. El usuario selecciona el formato.
3. El sistema genera el archivo con la información de los clusters.
4. El sistema inicia la descarga del archivo en el dispositivo del usuario.

Caso de uso #13 - Ver gráfica 2D

Actor: Usuario Creador

Precondición: El usuario se encuentra en la vista de resultados (Caso de uso #11).

Detonante: El usuario selecciona la opción "Ver gráfica 2D".

Escenario Principal:

1. El sistema procesa los datos para generar una representación visual en 2D (PCA/t-SNE).
2. El sistema renderiza el gráfico mostrando los puntos y los centroides de los clusters.
3. El usuario visualiza e interactúa con la gráfica.

Caso de uso #14 - Listar resultados

Actor: Usuario Creador

Precondición: El usuario se encuentra en la vista de resultados (Caso de uso #11).

Detonante: El usuario selecciona la opción "Listar resultados".

Escenario Principal:

1. El sistema recupera el detalle de asignación de cada respuesta a su cluster correspondiente.
2. El sistema muestra una tabla paginada con los datos detallados.
3. El usuario navega por la lista de resultados.

Caso de uso #15 - Calcular accuracy

Actor: Usuario Creador

Precondición: El usuario se encuentra en la vista de resultados (Caso de uso #11).

Detonante: El usuario selecciona la opción "Calcular accuracy".

Escenario Principal:

1. El sistema ejecuta el cálculo de métricas de calidad (Silhouette, Inercia, etc.).
2. El sistema muestra en pantalla los valores de precisión y métricas de evaluación del modelo.
3. El usuario visualiza la información de rendimiento.

1.4. Gestor de datos

Caso de uso #16 - Importar

Actor: Usuario Creador

Precondición: El usuario está en el panel de gestión de datos.

Detonante: El usuario selecciona "Importar".

Escenario Principal:

1. El sistema abre una ventana de selección de archivos.
2. El usuario selecciona un archivo válido del dispositivo.
3. El sistema valida el formato y la estructura del archivo.
4. El sistema procesa e inserta los datos en la base de datos.
5. El sistema confirma el número de registros importados.

Extensiones:

3a) El archivo tiene un formato corrupto: El sistema muestra error de formato y cancela la importación.

Caso de uso #17 - Exportar

Actor: Usuario Creador

Precondición: Existen datos en el sistema para exportar.

Detonante: El usuario selecciona "Exportar".

Escenario Principal:

1. El sistema solicita el formato de salida deseado.
2. El usuario selecciona el formato.
3. El sistema recopila la información seleccionada.
4. El sistema genera el archivo y lanza la descarga.

1.5. Registrarse

Caso de uso #18 - Crear cuenta

Actor: Usuario No Registrado

Precondición: El usuario no está logueado.

Detonante: El usuario selecciona "Crear Cuenta".

Escenario Principal:

1. El sistema solicita correo, nombre de usuario y contraseña.
2. El usuario introduce los datos.
3. El sistema valida que el correo no exista ya en la base de datos.
4. El sistema crea la cuenta y confirma el registro.

Extensiones:

3a) El correo ya está registrado: El sistema informa al usuario de que la cuenta ya existe.

Caso de uso #19 - Iniciar sesión

Actor: Usuario No Registrado

Precondición: El usuario tiene una cuenta creada.

Detonante: El usuario accede a la pantalla de login.

Escenario Principal:

1. El sistema solicita usuario y contraseña.
2. El usuario introduce sus credenciales.
3. El sistema valida las credenciales.
4. El sistema concede acceso y redirige al menú principal.

Extensiones:

3a) Credenciales incorrectas: El sistema muestra error de autenticación y permite reintentar.

1.6. Salir

Caso de uso #20 - Cerrar sesión

Actor: Usuario Creador / Usuario Registrado

Precondición: El usuario tiene una sesión activa.

Detonante: El usuario pulsa el botón "Cerrar Sesión".

Escenario Principal:

1. El usuario confirma que desea salir.
2. El sistema invalida la sesión actual.
3. El sistema redirige a la pantalla de inicio.

2. Descripción del diagrama de modelo

El diagrama completo se puede ver en el archivo Diagrama_Modelo de la carpeta Diagramas

Los cambios que se han hecho respecto a la primera entrega del proyecto de PROP en el diagrama de modelo son los siguientes:

- Implementación del diseño en 3 capas (Presentación, Dominio y Persistencia)
- Implementación del diseño de la capa de Presentación
- Implementación del diseño de la capa de Persistencia
- Implementación del diseño de mejoras en la capa de Dominio
- Implementación del diseño de extras (Sistema de Login, Tratamiento de valores faltantes (Missing Values), Detección y tratamiento de valores extremos (Outliers) y Sistema de Guardado automático)

A continuación, se explica más específicamente los cambios realizados en el diagrama del modelo conceptual para cada capa. Para la capa de presentación y la capa de persistencia (capas nuevas), se presentan las clases añadidas. Para la capa de dominio (capa de la primera entrega), también se muestran las clases modificadas y eliminadas, a más de las clases añadidas.

2.1. Capa de Presentación

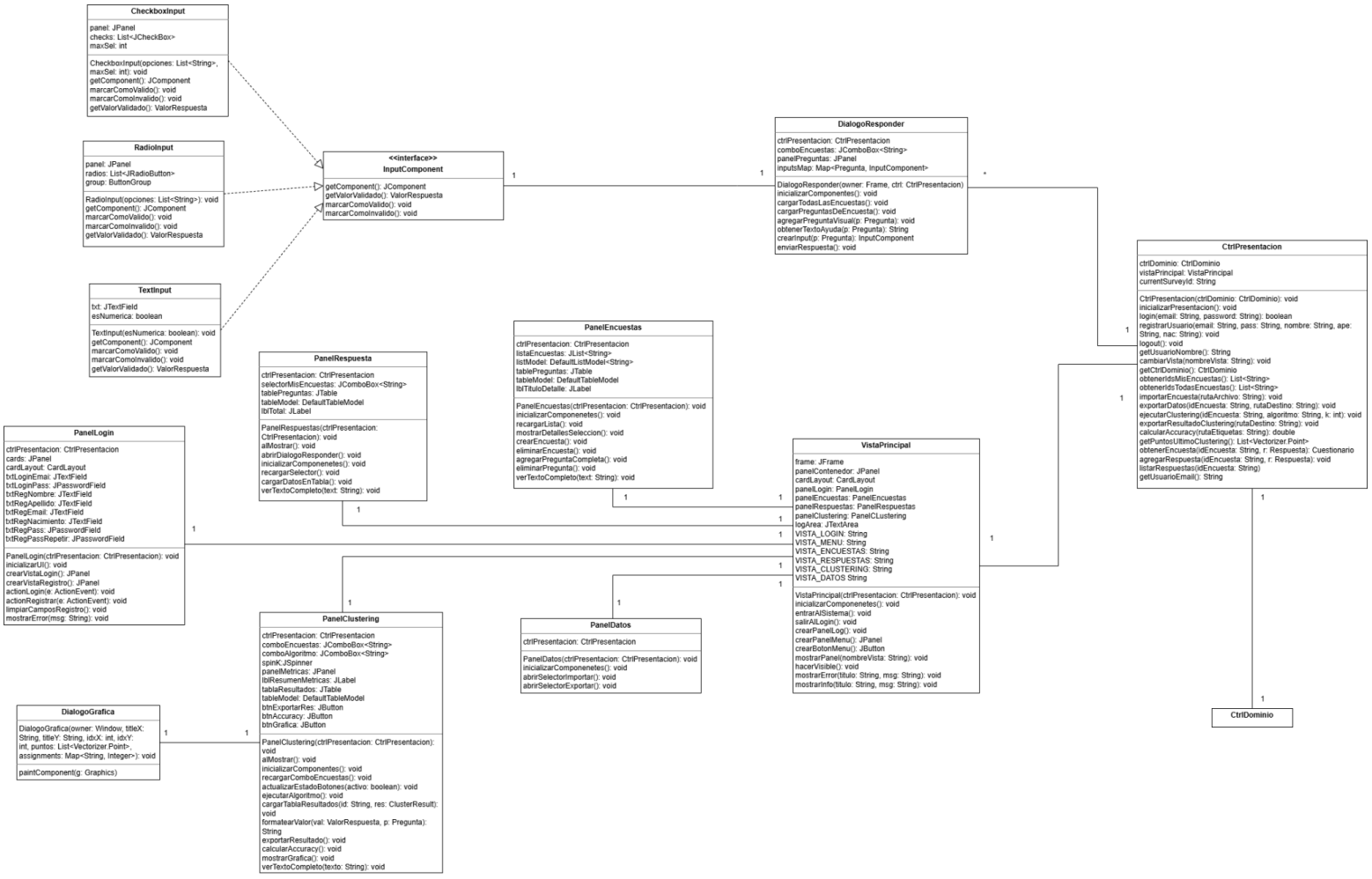

Diagrama de modelo de la capa de presentación. El diagrama se puede ver com más detalle en el archivo Diagrama_Modelo_Presentacion de la carpeta Diagramas

Clases añadidas

En la primera entrega únicamente teníamos la capa de Dominio, que se encargaba de toda la lógica de la aplicación. Para esta entrega, se ha añadido la capa de presentación al proyecto. Esta capa permite interactuar al usuario con la aplicación mediante una interfaz gráfica. Más específicamente, se ha añadido un controlador de la capa (CtrlPresentacion), los paneles y vistas de la interfaz (VistaPrincipal, PanelEncuesta, PanelRespuesta, PanelClustering, PanelDatos, PanelLogin y DialogoResponder). También hay clases añadidas que hacen referencia a los extras implementados.

2.1.1. CtrlPresentacion

Clase añadida - Breve descripción de la clase: Esta es la clase controladora de la capa de presentación. Actúa como el puente entre la interfaz de usuario y la lógica de negocio de la aplicación. Se encarga de recibir las acciones del usuario, coordinar las operaciones con el CtrlDominio y gestionar la navegación entre las diferentes vistas de la aplicación.

Cardinalidad

- 1:1 con CtrlDominio: Almacena una referencia a un único CtrlDominio para acceder a la lógica de la aplicación.
- 1:1 con VistaPrincipal: Tiene una única instancia de VistaPrincipal para gestionar la interfaz gráfica y la navegación.

Descripción de atributos

- ctrlDominio: CtrlDominio. Referencia al Controlador de Dominio.
- vistaPrincipal: VistaPrincipal. Referencia a la interfaz gráfica de la aplicación.
- currentSurveyId: String. Almacena el ID de la última encuesta importada o sobre la que se está trabajando activamente.

Descripción de las funciones

- CtrlPresentacion(CtrlDominio): Constructor. Recibe el controlador de dominio e inicializa la VistaPrincipal.
- inicializarPresentacion(): Hace visible la ventana principal (vistaPrincipal).
- login(String, String): Devuelve cierto si el login se ha hecho correctamente. Llama a CtrlDominio.
- registrarUsuario(String, String, String, String, String): Delega el registro al CtrlDominio.
- logout(): Se cierra la sesión desde el CtrlDominio.
- getUsuarioNombre(): Devuelve el nombre del usuario.
- cambiarVista(String): Solicita a vistaPrincipal que muestre un panel específico.
- getCtrlDominio(): Devuelve la referencia al controlador de dominio (getter).
- obtenerIdsMisEncuestas(): Llama a CtrlDominio para obtener una lista de todos los IDs de las encuestas del usuario.
- obtenerIdsEncuestas(): Llama a CtrlDominio para obtener una lista de todos los IDs de las encuestas.

- obtenerIdsEncuestas(): Devuelve los ids de las encuestas.
- importarEncuesta(String): Carga una encuesta y sus respuestas desde un archivo CSV (CsvSurveyImporter). Guarda los datos en CtrlDominio.
- exportarDatos(String, String): Exporta una encuesta y sus respuestas a un archivo CSV (CsvSurveyExporter).
- ejecutarClustering(String, String, int): Ejecuta un algoritmo de Clustering sobre una encuesta. Llama al método correspondiente en CtrlDominio.
- exportarResultadoClustering(String): Exporta el resultado del último clustering calculado a un archivo CSV (CsvClusterExporter).
- calcularAccuracy(String): Calcula la precisión (Accuracy) del último resultado de clustering comparándolo con unas etiquetas de verdad (ClusteringAccuracy).
- getPuntosUltimoClustering() Obtiene los datos vectorizados de la última encuesta para su visualización en una gráfica.
- obtenerEncuesta(String): Devuelve la encuesta correspondiente.
- agregarRespuesta(String, Respuesta): Añade una respuesta a una encuesta.
- listarRespuestas(String): Devuelve la lista de respuestas a una encuesta.
- getUsuarioEmail(): Deveulve el correo electrónico del usuario.

2.1.2. VistaPrincipal

Clase añadida - Breve descripción de la clase: Representa la vista principal de la interfaz gráfica.

Cardinalidad

- 1:1 con CtrlPresentacion: Almacena una referencia para poder comunicar las acciones de la interfaz al controlador.
- 1:1 con PanelEncuestas, PanelRespuestas, PanelClustering y PanelDatos: Almacena referencias y contiene los paneles específicos de la aplicación.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia al controlador de presentación.
- frame: JFrame. La ventana principal de la aplicación.
- panelContenedor: JPanel. El panel que contiene todas las vistas, gestionado por cardLayout.

- cardLayout: CardLayout. Gestor de diseño que permite alternar entre paneles como si fueran cartas.
- panelLogin, panelEncuestas, panelRespuestas, panelClustering: JPanel. Referencias a los paneles específicos para poder actualizarlos cuando se muestran.
- logArea: JTextArea. Área de texto para mostrar mensajes del sistema.
- VISTA_LOGIN, VISTA_MENU, VISTA_ENCUESTAS, VISTA_RESPUESTAS, VISTA_CLUSTERING, VISTA_DATOS: String. IDs usados para identificar y navegar entre los paneles con el CardLayout.

Descripción de las funciones

- VistaPrincipal(CtrlPresentacion): Constructor. Inicializa los componentes de la ventana.
- inicializarComponentes(): Configura el JFrame, el CardLayout y añade los paneles de la aplicación al contenedor.
- entrarAlSistema(): Da el mensaje inicial al usuario y muestra la vista del menú.
- salirAlLogin(): Da el mensaje final y muestra la vista de login.
- crearPanelLog(): Crea y añade la JTextArea de la consola en la parte inferior de la ventana.
- crearPanelMenu(): Construye el panel del menú principal con los botones de navegación.
- crearBotonMenu(String, String, Color): Función auxiliar para crear un botón de navegación con color.
- mostrarPanel(String): Muestra el panel asociado al nombreVista dado y llama a los métodos de actualización si es necesario.
- hacerVisible(): Hace visible la ventana principal.
- log(String): Escribe un mensaje en la consola y desplaza el scroll al final.
- mostrarError(String, String): Muestra una ventana emergente de error.
- mostrarInfo(String, String): Muestra una ventana emergente de información.

2.1.3. PanelEncuesta

Clase añadida - Breve descripción de la clase: Se encarga de la interfaz gráfica para la gestión de encuestas. Permite al usuario ver una lista de encuestas disponibles, mostrar los detalles de las preguntas de una encuesta seleccionada, y realizar operaciones como crear, eliminar encuestas y añadir/eliminar preguntas a una encuesta.

Cardinalidad

- 1:1 con VistaPrincipal: Se conecta al controlador de presentación para acceder a la lógica de gestión de encuestas y preguntas.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia al controlador de presentación que le pasa VistaPrincipal.
- listaEncuestas: JList<String>. Lista visual de los IDs de las encuestas disponibles.
- listModel: DefaultListModel<String>. Modelo de datos para gestionar los elementos de listaEncuestas.
- tablaPreguntas: JTable. Tabla que muestra los detalles de las preguntas de la encuesta seleccionada.
- tableModel: DefaultTableModel. Modelo de datos para gestionar las filas y columnas de tablaPreguntas.
- lblTituloDetalle: JLabel. Etiqueta de texto para mostrar el ID de la encuesta seleccionada en la zona de detalles.

Descripción de las funciones

- PanelEncuestas(CtrlPresentacion): Constructor. Inicializa los componentes de la interfaz de este panel.
- inicializarComponentes(): Configura el diseño, la lista y la tabla, y añade los botones de acciones.
- recargarLista(): Borra y vuelve a cargar el model de listaEncuestas.
- mostrarDetallesSeleccion(): Se llama al seleccionar un ID en la lista. Carga y muestra las preguntas de la encuesta seleccionada.
- crearEncuesta(): Solicita un ID al usuario y llama a CtrlDominio para crear una nueva encuesta vacía.
- eliminarEncuesta(): Elimina la encuesta seleccionada y todas sus respuestas de la capa de dominio.
- agregarPreguntaCompleta(): Muestra un formulario para definir el enunciado, tipo, opciones, y máx. selecciones de una nueva pregunta, y luego la añade a la encuesta seleccionada en CtrlDominio.
- eliminarPregunta(): Elimina la pregunta seleccionada de la tabla de la encuesta seleccionada en CtrlDominio.
- verTextoCompleto(String): Crea una nueva área de texto y muestra el panel.

2.1.4. PanelRespuesta

Clase añadida - Breve descripción de la clase: Este panel gestiona la visualización y gestión de las respuestas asociadas a las encuestas. Permite al usuario seleccionar una encuesta, ver todas sus respuestas y realizar operaciones.

Cardinalidad

- 1:1 con VistaPrincipal: Se conecta al controlador de presentación para acceder a la lógica de gestión de encuestas y preguntas.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia al controlador de presentación que le pasa VistaPrincipal.
- selectorMisEncuestas: JComboBox<String>. Desplegable para seleccionar la encuesta.
- tableRespuestas: JTable. Tabla que muestra las respuestas. Cada columna representa una pregunta.
- tableModel: DefaultTableModel. Modelo para gestionar los datos y las cabeceras de tableRespuestas.
- lblTotal: JLabel. Etiqueta que muestra el número total de respuestas cargadas para la encuesta actual.

Descripción de las funciones

- PanelRespuestas(CtrlPresentacion): Constructor. Inicializa los componentes.
- alMostrar(): Se llama cuando el panel se hace visible. Recarga el desplegable de encuestas.
- inicializarComponentes(): Configura el diseño, la tabla, el selector de encuestas y los botones.
- abrirDialogoResponder(): Se crea y se abre una pantalla de diálogo para responder a la encuesta.
- recargarSelector(): Carga los IDs de las encuestas disponibles.
- cargarDatosEnTabla(): Se obtiene la encuesta y las preguntas y se muestra.
- verTextoCompleto(String): Crea una nueva área de texto y muestra el panel.

2.1.5. PanelClustering

Clase añadida - Breve descripción de la clase: Este panel se encarga de la configuración, ejecución y visualización de los resultados de los algoritmos de Clustering.

Permite seleccionar la encuesta y el algoritmo. Muestra las métricas de resultado y ofrece opciones para exportar el resultado y visualizar los datos en una gráfica.

Cardinalidad

- 1:1 con VistaPrincipal: Se conecta al controlador de presentación para acceder a la lógica de gestión de encuestas y preguntas.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia al controlador de presentación que le pasa VistaPrincipal.
- comboEncuestas: JComboBox<String>. Desplegable para seleccionar la encuesta a analizar.
- comboAlgoritmo: JComboBox<String>. Desplegable para elegir el algoritmo de clustering.
- spinK: JSpinner. Selector numérico para definir el número de clusters.
- btnEjecutar: JButton. Botón para iniciar la ejecución del algoritmo en un hilo separado.
- panelMetricas: JPanel. Panel para mostrar las métricas de rendimiento.
- lblResumenMetricas: JLabel. Mostrador de métricas resultantes.
- tablaResultados: JTable. Tabla principal que muestra la asignación de cluster a cada participante, junto con sus respuestas.
- tableModel: DefaultTableModel. Modelo de datos de la tabla.
- btnExportarRes, btnAccuracy, btnGrafica: JButton. Botones de acciones para después de la ejecución (exportar, calcular accuracy, visualizar gráfica).

Descripción de clases internas

- DialogoGrafica: JDialog que contiene un JPanel personalizado. En su método paintComponent, dibuja los puntos vectorizados de la encuesta asignando un color diferente a cada cluster y ajustando las escalas.

Descripción de las funciones

- PanelClustering(CtrlPresentacion): Constructor. Inicializa los componentes.
- alMostrar(): Se llama al hacer visible el panel. Recarga el desplegable de encuestas y deshabilita los botones de post-ejecución.
- inicializarComponentes(): Configura el diseño, los selectores y el área de resultados.
- recargarComboEncuestas(): Carga los IDs de las encuestas disponibles.

- actualizarEstadoBotones(boolean): Habilita/deshabilita los botones de acciones post-ejecución (Exportar, Accuracy, Gráfica).
- ejecutarAlgoritmo(): Recoge la configuración, muestra un mensaje de espera, llama a ejecutar clustering y muestra las métricas.
- cargarTablaResultados(String, ClusterResult): Rellena la tabla de resultados con la información del participante.
- formatearValor(ValorRespuesta, Pregunta): Convierte ValorRespuesta a una representación para el usuario. En vez de tener índices y números, tenemos las opciones.
- exportarResultado(): Abre un selector de archivo y llama a ctrlPresentacion para exportar el resultado.
- calcularAccuracy(): Abre un selector de archivo para elegir el CSV de etiquetas de verdad y llama a ctrlPresentacion para comparar el resultado.
- mostrarGrafica(): Muestra un diálogo para que el usuario elija dos preguntas para los ejes X/Y. Crea y muestra un gráfico.
- verTextoCompleto(String): Muestra el texto completo de una celda de la tabla en un diálogo.

2.1.6. PanelDatos

Clase añadida - Breve descripción de la clase: Este panel se centra en la funcionalidad de Importación y Exportación de encuestas completas utilizando archivos CSV.

Cardinalidad

- 1:1 con VistaPrincipal: Se conecta al controlador de presentación para acceder a la lógica de gestión de encuestas y preguntas.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia al controlador de presentación que le pasa VistaPrincipal.

Descripción de las funciones

- PanelDatos(CtrlPresentacion): Constructor. Inicializa los componentes.
- inicializarComponentes(): Configura el diseño y añade los botones de "Importar Encuesta" y "Exportar Encuesta".
- abrirSelectorImportar(): Abre un selector de archivos para que el usuario elija un archivo CSV de entrada y luego llama a importarEncuesta en ctrlPresentacion.

- abrirSelectorExportar(): Pide al usuario que seleccione la encuesta a exportar y abre un selector de archivos para elegir la ruta de destino. Llama a exportarDatos ctrlPresentacion.

2.1.7. PanelLogin

Clase añadida - Breve descripción de la clase: Panel encargado de gestionar la interacción con el usuario de los procesos de autenticación.

Cardinalidad

- 1:1 con VistaPrincipal: Se conecta a su vista principal.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion.
- cards: JPanel.
- cardLayout: CardLayout.
- txtLoginEmail: JTextField. Campo de login para introducir el correo electrónico.
- txtLoginPass: JPasswordField. Campo de login para introducir la contraseña.
- txtRegNombre, txtRegApellido, txtRegEmail, txtRegNacimiento: JTextField. Campos de registro para recopilar datos del nuevo usuario.
- txtRegPass, txtRegPassRepetir: JPasswordField. Campos de registro para introducir y confirmar la contraseña.

Descripción de las funciones

- PanelLogin(CtrlPresentacion): Constructor. Inicializa la referencia al controlador y inicializa la UI.
- inicializarUI(): Configura el diseño principal del panel y establece el CardLayout para las dos vistas de autenticación.
- crearVistaLogin(): Crea el panel con los campos de entrada, el botón "INICIAR SESIÓN" y el enlace para ir a la vista de Registro.
- crearVistaRegistro(): Crea el panel con todos los campos necesarios para el registro, el botón "CREAR CUENTA" y el botón para volver al Login.
- actionLogin(ActionEvent): Recopila los datos del formulario de Login, realiza validaciones básicas y delega la autenticación al controlador de presentación.
- actionRegistrar(ActionEvent): Recopila los datos del formulario de Registro, comprueba que las contraseñas coincidan y delega la creación del usuario al controlador.

- limpiarCamposRegistro(): Vacía todos los campos de texto del formulario de Registro.
- mostrarError(String):Presenta un mensaje de error al usuario utilizando un diálogo emergente.

2.1.8. DialogoResponder

Clase añadida - Breve descripción de la clase: Ventana de diálogo que permite a un usuario responder una encuesta.

Cardinalidad

- 1:1 con CtrlPresentacion: Delega la obtención de encuestas y el guardado de la respuesta final.

Descripción de atributos

- ctrlPresentacion: CtrlPresentacion. Referencia para interactuar con la Capa de Dominio.
- comboEncuestas: JComboBox<String>. Desplegable para seleccionar qué encuesta responder.
- panelPreguntas: JPanel. Panel dinámico donde se añaden los componentes de las preguntas.
- inputsMap: Map<Pregunta, InputComponent>. Mapa que asocia cada Pregunta de la encuesta con el componente visual que la maneja.

Descripción de clases internas

- InputComponent: Define el contrato para cualquier componente que capture un valor de respuesta, incluyendo métodos para obtener el componente visual, validar y obtener el ValorRespuesta.
- TextInput: Implementa InputComponent para TEXTO y NUMERICA usando un JTextField, añadiendo lógica de validación para double en el caso numérico.
- RadioInput): Implementa InputComponent para OPCION_UNICA usando JRadioButton agrupados, y devuelve el índice de la opción seleccionada.
- CheckboxInput: Implementa InputComponent para OPCION_MULTIPLE usando JCheckBox, con lógica para validar si el número de selecciones excede el maxSeleccion definido en la pregunta.

Descripción de las funciones

- DialogoResponder(Frame owner, CtrlPresentacion ctrl): Constructor que crea la ventana de diálogo modal, establece el tamaño y la centra en la pantalla.

- inicializarComponentes(): Establece el layout principal y añade el selector, las preguntas en scroll y los botones de acción.
- cargarTodasLasEncuestas(): Rellena el JComboBox obteniendo los IDs de todas las encuestas disponibles a través del controlador de presentación.
- cargarPreguntasDeEncuesta(): Se dispara al seleccionar una encuesta del desplegable. Obtiene el Cuestionario y llama a agregarPreguntaVisual() para cada pregunta que contiene.
- agregarPreguntaVisual(Pregunta): Crea un JPanel por pregunta, añade el enunciado con texto de ayuda y genera el componente de entrada adecuado para el tipo de pregunta.
- obtenerTextoAyuda(Pregunta): Genera una cadena de texto auxiliar basada en el TipoPregunta.
- crearInputSegunTipo(Pregunta): Decide qué implementación de InputComponent crear basándose en el TipoPregunta.
- enviarRespuesta(): Se crea una respuesta, se miran todas las preguntas y se obtiene el ValorRespuesta.

2.2. Capa de Dominio

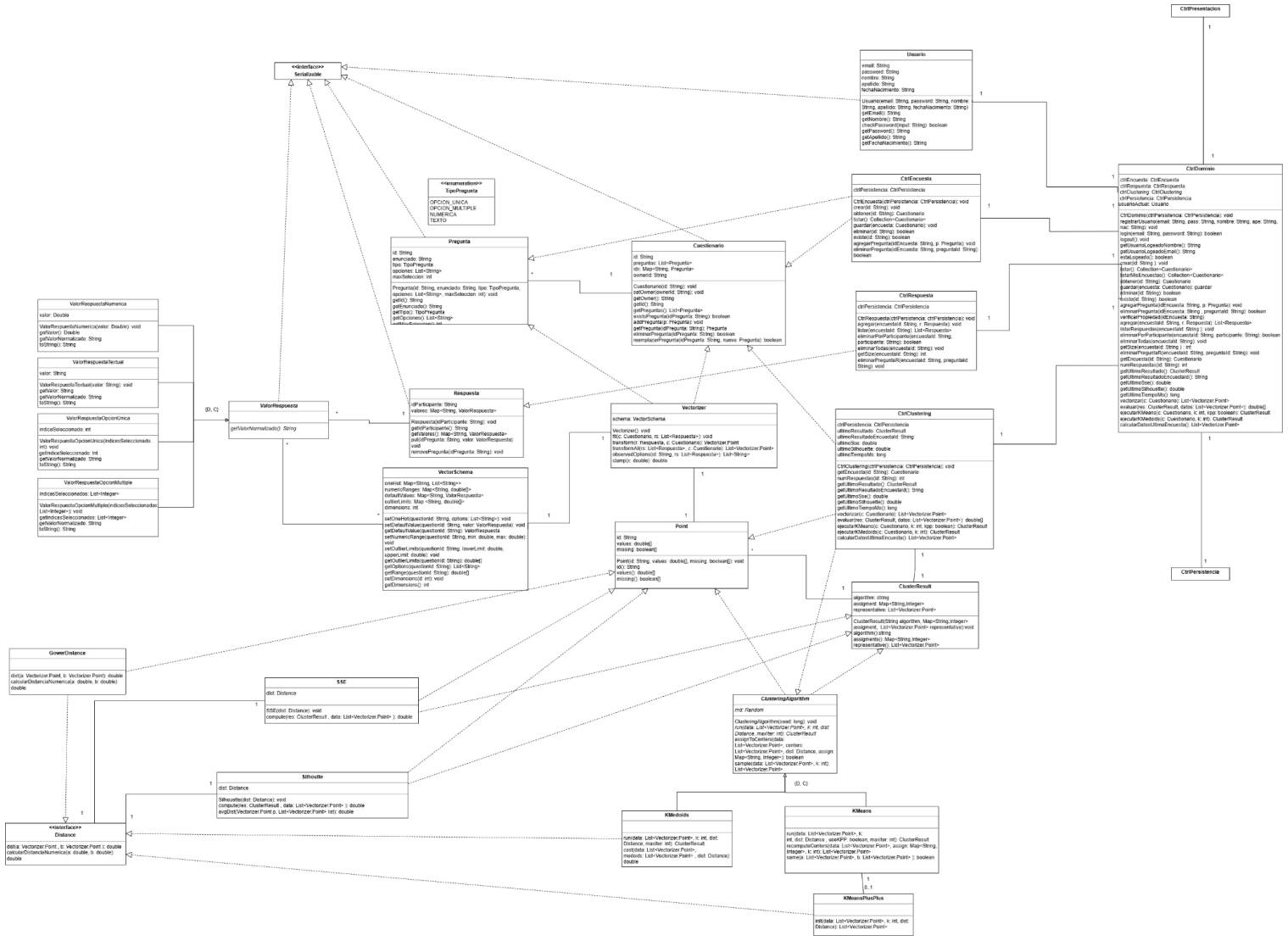

Diagrama de modelo de la capa de dominio. El diagrama se puede ver com más detalle en el archivo Diagrama_Modelo_Dominio de la carpeta Diagramas

Clases añadidas

La capa de dominio ya estaba diseñada e implementada para la primera entrega. Para esta entrega hemos añadido un controlador de dominio (CtrlDominio) que gestione los controladores específicos que teníamos antes (CtrlEncuesta, CtrlRespuesta, CtrlClustering) y se relacione con la capa de presentación y la capa de persistencia.

También se han mejorado algunos aspectos respecto a la anterior entrega: se ha añadido una clase que contenga el valor de las respuesta (ValorRespuesta) y una clasificación según el tipo de respuestas (ValorRespuestaNumerica, ValorRespuestaTextual, ValorRespuestaOpcionUnica, ValorRespuestaOpcionMultiple). Además se ha añadido una

clase abstracta de clustering (ClusteringAlgorithm) para evitar código repetido en los diferentes métodos.

En la implementación del extra “Sistema de Login” se ha añadido la clase Usuario.

Clases modificadas

Las clases que han sido modificadas son los controladores del dominio anteriores (CtrlEncuesta, CtrlRespuesta, CtrlClustering), ya que han cambiado su relación con las otras clases, la clase de Respuesta, que ahora contiene un ValorRespuesta, y las clases KMeans y KMedoids, que han cambiado su estructura y código debido a la nueva clase abstracta. Además, se ha modificado la interfaz Distance y la clase GowerDistance para implementar el cálculo de la distancia numérica como submétodo de distancia.

En la implementación del extra “Sistema de Login” se ha modificado la clase Cuestionario, ya que ahora este pertenece a un usuario.

En la implementación del extra “Tratamiento de valores faltantes (Missing Values)” y “Detección y tratamiento de valores extremos (Outliers)” se ha modificado la clase VectorSchema.

En la implementación del extra “Sistema de Guardado automático” las clases Cuestionario, Usuario, Pregunta, Respuesta y ValorRespuesta pasan a implementar la interfaz Serializable para poder guardar correctamente todos los datos.

Clases eliminadas

Por último, se han eliminado los repositorios de esta capa y su conexión con los controladores y se ha movido a la capa de persistencia.

2.2.1. CtrlDominio

Clase añadida - Breve descripción de la clase: El controlador de dominio actúa como una fachada para toda la lógica de la aplicación. Delega todas las responsabilidades específicas (gestión de encuestas, gestión de respuestas, y clustering) a sus respectivos controladores especializados (CtrlEncuesta, CtrlRespuesta, CtrlClustering). Su objetivo es ofrecer una interfaz limpia y única a la capa de presentación y capa de persistencia, desacoplando la complejidad interna de las otras capas.

Cardinalidad

- 1:1 con CtrlPresentacion: La referencia a la capa de presentación de la aplicación.
- 1:1 con CtrlPersistencia: La referencia a la capa de persistencia de la aplicación.
- 1:1 con CtrlEncuesta: Contiene y delega toda la lógica de gestión de encuestas.
- 1:1 con CtrlRespuesta: Contiene y delega toda la lógica de gestión de respuestas.
- 1:1 con CtrlClustering: Contiene y delega toda la lógica de algoritmos y evaluación.

- 1:1 con Usuario: Contiene el usuario de la sesión.

Descripción de atributos

- ctrlEncuesta: CtrlEncuesta. Controlador especializado en la lógica de Cuestionario.
- ctrlRespuesta: CtrlRespuesta. Controlador especializado en la lógica de Respuesta.
- ctrlClustering: CtrlClustering. Controlador especializado en algoritmos de clustering y vectorización.
- ctrlPersistencia: CtrlPersistencia. Referencia directa para la gestión del repositorio de usuarios.
- usuarioActual: Usuario. Objeto del usuario que ha iniciado sesión.

Descripción de las funciones

- CtrlDominio(CtrlPersistencia): Constructor que inicializa los controladores especializados y guarda la referencia a la persistencia.
- registrarUsuario(String, String, String, String), login(String, String), getUsuarioLogueadoNombre(), getUsuarioLogueadoEmail(), estaLogueado(): Gestión del usuario.
- crear(String), obtener(String), listar(), guardar(Cuestionario), eliminar(String), existe(String), agregarPregunta(String, Pregunta) y eliminarPregunta(String, String): Gestión de la existencia y estructura de los cuestionarios.
- agregar(String, Respuesta), listarRespuestas(String), eliminarPorParticipante(String, String), eliminarTodas(String) y getSize(String): Gestión de las colecciones de respuestas para cada encuesta.
- getEncuesta(String), numRespuestas(String), getUltimoResultado(), getUltimoResultadoEncuestaId(), getUltimoSse(), getUltimoSilhouette(), getUltimoTiempoMs(), vectorizar(Cuestionario), ejecutarKMeans(Cuestionario, int, boolean), ejecutarKMedoids(Cuestionario, int) y calcularDatosUltimaEncuesta(): Ejecución de los algoritmos, vectorización de datos, y recuperación de métricas de calidad.

2.2.2. CtrlEncuesta

Clase modificada - Breve descripción de la clase: Controlador de dominio especializado en la gestión de la estructura de Cuestionario. Gestiona la lógica específica para crear, obtener, listar y eliminar encuestas, y las operaciones internas de preguntas.

Cardinalidad

- 1:1 con CtrlDominio: Se conecta directamente al controlador general.

Descripción de atributos

- ctrlPersistencia: CtrlPersistencia. Referencia al controlador de la capa de persistencia para acceder al SurveyRepository. Se lo pasa CtrlDominio.

Descripción de las funciones

- CtrlEncuesta(CtrlPersistencia): Constructor.
- crear(), obtener(), listar(), guardar(), eliminar(), existe(): Llaman a las funciones de los repositorios.
- agregarPregunta(String, Pregunta): Obtiene el cuestionario, llama a añadirPregunta de Cuestionario y la guarda.
- eliminarPregunta(String, String): Obtiene el cuestionario, llama a eliminarPregunta de Cuestionario.

2.2.3. CtrlRespuesta

Clase modificada - Breve descripción de la clase: Controlador de dominio especializado en la gestión de las respuestas. Se ocupa de las operaciones de respuestas para una encuesta, como agregar, listar, eliminar por participante…

Cardinalidad

- 1:1 con CtrlDominio: Se conecta directamente al controlador general.

Descripción de atributos

- ctrlPersistencia: CtrlPersistencia. Referencia al controlador de la capa de persistencia para acceder al ResponseRepository. Se lo pasa CtrlDominio.

Descripción de las funciones

- CtrlRespuesta(CtrlPersistencia): Constructor.
- agregar(String, Respuesta), listarRespuestas(String), eliminarPorParticipante(String, String), eliminarTodas(String), getSize(String), eliminarPreguntaR(String, String): Llama a las funciones de los repositorios.

2.2.4. CtrlClustering

Clase modificada - Breve descripción de la clase: Controlador de dominio especializado en el proceso de Clustering, que incluye: Vectorización de respuestas, ejecución de algoritmos y evaluación de resultados. Mantiene el estado del último resultado de clustering ejecutado para que otros componentes puedan acceder a las métricas y los datos vectorizados.

Cardinalidad

- 1:1 con CtrlDominio: Se conecta directamente al controlador general.
- 1:1 con ClusterResult: Almacena el resultado del último algoritmo ejecutado.

Descripción de atributos

- ctrlPersistencia: CtrlPersistencia. Referencia al controlador de la capa de persistencia para acceder al SurveyRepository y ResponseRepository. Se lo pasa CtrlDominio.
- ultimoResultado: ClusterResult. Objeto que contiene las asignaciones de clusters y representantes del último cálculo.
- ultimoResultadoEncuestaId: String. ID de la encuesta utilizada en el último cálculo.
- ultimoSse, ultimoSilhouette: double. Métricas de calidad del último resultado de clustering.
- ultimoTiempoMs: long. Tiempo de ejecución del último algoritmo en milisegundos.

Descripción de las funciones

- CtrlClustering(CtrlPersistencia): Constructor.
- vectorizar(Cuestionario): Obtiene las respuestas, crea un Vectorizer, lo ajusta al cuestionario, y transforma todas las respuestas en una lista de Vectorizer.Point.
- evaluar(ClusterResult, List<Point>): Utiliza la GowerDistance y los algoritmos SSE y Silhouette para calcular las métricas del resultado. Almacena las métricas en los atributos.
- ejecutarKMeans(Cuestionario, k, kpp): Vectoriza los datos. Inicializa el algoritmo KMeans y lo ejecuta. Mide el tiempo, almacena el resultado y las métricas.
- ejecutarKMedoids(Cuestionario, k) Vectoriza los datos. Inicializa el algoritmo KMedoids y lo ejecuta. Mide el tiempo, almacena el resultado y las métricas.
- getUltimoResultado(), getUltimoResultadoEncuestaId(), getUltimoSse(), getUltimoSilhouette(), getUltimoTiempoMs(): Getters que permiten a la capa de presentación acceder a las métricas y al resultado del último cálculo.
- calcularDatosUltimaEncuesta(): Recupera la encuesta usada en el último cálculo y vuelve a vectorizar sus respuestas para obtener los datos para la gráfica.

2.2.5. ValorRespuesta

Clase añadida - Breve descripción de la clase: Es la clase base abstracta para todos los tipos de valores que puede tomar una respuesta. Implementa la interfaz Serializable para permitir la persistencia de las respuestas en disco.

Cardinalidad

- 1:1 con Respuesta. Cada Respuesta tiene asociado un ValorRespuesta.
- Generalización: Es la raíz de los valores de respuesta.

Descripción de las funciones

- getValorNormalizado(): Método abstracto que implementan las subclases. Devuelve el valor de la respuesta como una cadena de texto utilizable en la lógica de vectorización.

2.2.6. ValorRespuestaNumerica

Clase añadida - Breve descripción de la clase: Representa el valor de una respuesta cuando la pregunta asociada es numérica. Almacena el valor como un double.

Cardinalidad

- Subclase de ValorRespuesta.

Descripción de atributos

- valor: El valor numérico exacto que el participante introdujo como respuesta.

Descripción de las funciones

- getValor(): Devuelve el valor numérico en formato double.
- getValorNormalizado(): Implementación que devuelve el valor numérico como un String.
- toString(): Devuelve el valor de la respuesta en string.

2.2.7. ValorRespuestaTextual

Clase añadida - Breve descripción de la clase: Representa el valor de una respuesta cuando la pregunta asociada es textual. Almacena la cadena de texto literal introducida por el participante.

Cardinalidad

- Subclase de ValorRespuesta.

Descripción de atributos

- valor: La cadena de texto introducida por el participante.

Descripción de las funciones

- getValor(): Devuelve el valor textual en formato String.
- getValorNormalizado(): Devuelve la misma cadena de texto.

- toString(): Devuelve el valor de la respuesta en string.

2.2.8. ValorRespuestaOpcionUnica

Clase añadida - Breve descripción de la clase: Representa el valor de una respuesta a una pregunta de Opción Única. Almacena el índice (posición en la lista de opciones de la pregunta) de la opción que fue seleccionada por el participante.

Cardinalidad

- Subclase de ValorRespuesta.

Descripción de atributos

- indiceSeleccionado: Almacena el índice de la opción seleccionada.

Descripción de las funciones

- getIndiceSeleccionado(): Devuelve el índice de la opción elegida.
- getValorNormalizado(): Devuelve el índice como un String.
- toString(): Devuelve el valor de la respuesta en String.

2.2.9. ValorRespuestaOpcionMultiple

Clase añadida - Breve descripción de la clase: Representa el valor de una respuesta a una pregunta de Opción Múltiple. Almacena una lista de los índices correspondientes a todas las opciones que fueron seleccionadas por el participante.

Cardinalidad

- Subclase de ValorRespuesta.

Descripción de atributos

- indicesSeleccionados List<Integer>: Lista de los índices de las opciones elegidas.

Descripción de las funciones

- getIndicesSeleccionados(): Devuelve la lista de los índices seleccionados.
- getValorNormalizado(): Devuelve los índices seleccionados como una única String separada por comas.
- toString(): Devuelve el valor de la respuesta en string.

2.2.10. Respuesta

Clase modificada - Breve descripción de la clase: Representa el conjunto de valores respondidos por un único participante a un cuestionario. Implementa la interfaz Serializable

para permitir que el conjunto completo de respuestas de un participante se pueda guardar y recuperar desde disco.

Cardinalidad

- 1:N con ValorRespuesta: Todos los valores de las respuestas.

Descripción de atributos

- idParticipante: String. Identificador único del participante que ha respondido al cuestionario.
- valores: Map<String, ValorRespuesta>. El almacenamiento principal de la respuesta.

Descripción de las funciones

- Respuesta(String): Constructor. Inicializa la instancia con el ID del participante.
- getIdParticipante(): Devuelve el identificador del participante.
- getValores(): Devuelve un mapa de los pares IdPregunta, ValorRespuesta.
- put(String, ValorRespuesta): Añade o actualiza el valor para una pregunta específica.
- removePregunta(String): Elimina una pregunta.

2.2.11. ClusteringAlgorithm

Clase añadida - Breve descripción de la clase: Es la clase base abstracta para todos los algoritmos de clustering implementados en el dominio. Define la estructura y las funcionalidades comunes necesarias para realizar el clustering, como la inicialización aleatoria y las operaciones básicas de asignación de puntos a centros.

Cardinalidad

- Generalización: Es la raíz de los algoritmos de clustering.

Descripción de atributos

- rnd: Generador de números aleatorios, inicializado con una semilla.

Descripción de las funciones

- ClusteringAlgorithm(long): Constructor. Inicializa el generador rnd con la semilla.
- run(List<Point>, int, Distance, int): Se implementa por las subclases (K-Means, K-Medoids).
- assignToCenters(List<Point>, List<Point>, Distance, Map<String,Integer>): Asigna cada punto de data al centro (centers) más cercano, basándose en la métrica de distancia.

- sample(List<Point>, int): Realiza un muestreo aleatorio simple de puntos de la lista. Se usa para la inicialización básica de los centros/medoides.

2.2.12. KMeans

Clase modificada - Breve descripción de la clase: Implementa el algoritmo de clustering K-Means. Su objetivo es particionar puntos de datos en grupos. Soporta la inicialización simple y la inicialización K-Means++.

Cardinalidad

- Subclase de ClusteringAlgorithm

Descripción de atributos

- Hereda los atributos de ClusteringAlgorithm

Descripción de las funciones

- run(List<Point>, int, Distance, boolean, int: Inicializa los centros (usando K-Means++ o inicialización simple). Asigna puntos y recalcula centros
- run(List<Point>, int, Distance, int): Llama al método principal usando la inicialización simple.
- recomputeCenters(List<Point>, Map<String,Integer>, int): Recalcula la posición de cada centroide de todos los puntos asignados a ese cluster.
- same(List<Point>, List<Point>): Comprueba si la lista de nuevos centros es idéntica a la lista de centros anterior, lo que indica convergencia.

2.2.13. KMedoids

Clase modificada - Breve descripción de la clase: Implementa el algoritmo K-Medoids. A diferencia de K-Means, los centros de los clusters (medoides) deben ser puntos reales de los datos de entrada, no el promedio.

Cardinalidad

- Subclase de ClusteringAlgorithm

Descripción de atributos

- Hereda los atributos de ClusteringAlgorithm

Descripción de las funciones

- run(List<Point>, int, Distance, int): Inicializa los medoides con muestreo simple. Ejecuta un bucle iterativo que intenta mejorar el resultado.

- cost(List<Point>, List<Point>, Distance): Calcula la suma de las distancias mínimas de cada punto de datos al medoide más cercano.0 Este valor se usa para determinar si un intercambio es beneficioso.

2.2.14. Usuario

Clase añadida - Breve descripción de la clase: La clase Usuario representa una entidad de persona dentro del sistema. Implementa la interfaz Serializable para permitir que los datos de usuarios se persistan en disco durante el guardado automático de la aplicación.

Cardinalidad

- 1:1 con CtrlDominio: El usuario pertenece a una sesión, manejada por CtrlDominio.

Descripción de atributos

- email: String. El correo electrónico del usuario, utilizado como identificador único para la autenticación y la gestión de datos.
- password: String. La contraseña del usuario.
- nombre: String. El nombre de pila del usuario.
- apellido: String. El apellido o apellidos del usuario.
- fechaNacimiento: String. La fecha de nacimiento del usuario, almacenada como texto (formato DD/MM/AAAA).

Descripción de las funciones

- Usuario(String, String, String, String, String): Constructor que inicializa todos los datos de la instancia de usuario.
- getEmail(): Devuelve el identificador único del usuario (su correo electrónico).
- getNombre(): Devuelve el nombre de pila del usuario.
- checkPassword(String): Verifica si la contraseña de entrada coincide con la contraseña almacenada por el usuario.

2.2.15. Cuestionario

Clase modificada - Breve descripción de la clase: La clase Cuestionario representa una encuesta o formulario en el sistema. Implementa la interfaz Serializable para permitir que los cuestionarios y su estructura se persistan en disco.

Cardinalidad

- 1:N con Pregunta: Contiene múltiples instancias de Pregunta organizadas en una lista ordenada.

Descripción de atributos

- id: String. El identificador único del cuestionario.
- ownerId: String. El email del usuario propietario del cuestionario (utilizado para el extra de Login y control de permisos).
- preguntas: List<Pregunta>. Una lista que almacena las preguntas en el orden en que deben aparecer.
- idx: Map<String, Pregunta>. Un mapaque mapea el ID de la pregunta a su objeto Pregunta para permitir un acceso rápido a la pregunta sin tener que iterar sobre la lista.

Descripción de las funciones

- Cuestionario(String): Constructor que inicializa el cuestionario con un ID único.
- setOwner(String): Establece el ID del usuario propietario de este cuestionario.
- getOwner(): Devuelve el ID del usuario propietario.
- getId(): Devuelve el identificador único del cuestionario.
- getPreguntas(): Devuelve una lista inmutable de las preguntas, manteniendo el orden de inserción.
- existsPregunta(String idPregunta): Comprueba rápidamente si una pregunta con el ID dado ya existe en el cuestionario.
- addPregunta(Pregunta p): Añade una nueva pregunta a la lista y al mapa auxiliar, lanzando una excepción si el ID ya existe.
- getPregunta(String idPregunta): Recupera el objeto Pregunta utilizando el mapa auxiliar para un acceso eficiente.
- eliminarPregunta(String idPregunta): Elimina la pregunta de la lista y del mapa auxiliar, devolviendo true si la eliminación fue exitosa.
- reemplazarPregunta(String idPregunta, Pregunta nueva): Reemplaza una pregunta existente (localizada por idPregunta) con un objeto nueva Pregunta, manteniendo su posición en la lista.

2.2.16. VectorSchema

Clase modificada - Breve descripción de la clase: Esta clase actúa como una plantilla para la vectorización de datos. Guarda información como los rangos de valores para la normalización numérica.

Cardinalidad

- 1:1 con Vectorizer: El Vectorizer contiene y utiliza una instancia de VectorSchema para realizar todas las transformaciones.
- 1:N con Pregunta: Almacena información de cada pregunta.

Descripción de atributos

- oneHot: Map<String, List<String>>. Almacena las opciones para cada pregunta.
- numericRanges: Map<String, double[]>. Almacena el rango de valores ([min, max]) para cada pregunta numérica.
- defaultValues: Map<String, ValorRespuesta>. Almacena el valor medio para las preguntas numéricas o la moda para las de opción única.
- outlierLimits: Map<String, double[]>. Almacena los límites inferior y superior (Q1-1.5*IQR y Q3+1.5*IQR) calculados para cada pregunta numérica usando el método IQR. Se utiliza para detectar y filtrar valores extremos (outliers).
- dimensions: int. El número total de dimensiones que tendrá el vector final después de aplicar todas las codificaciones.

Descripción de las funciones

- setOneHot(String, List<String>): Registra las opciones para una pregunta.
- setNumericRange(String, double, double): Registra el rango [min, max] de valores para una pregunta numérica.
- setDefaultValue(String, ValorRespuesta): Registra el valor de la media o moda para la pregunta.
- getDefaultValue(String): Devuelve el valor de media o moda para una pregunta dada.
- getOptions(String): Devuelve la lista de opciones registradas para una pregunta.
- getRange(String): Devuelve el rango [min, max] de una pregunta numérica.
- setOutlierLimits(String, double, double): Registra el valor de los límites, de los valores extremos.
- getOutlierLimits(String): Devuelve los valores extremos registrados.
- setDimensions(int): Establece el tamaño final que debe tener el vector.
- getDimensions(): Devuelve el tamaño total del vector.

2.2.17. Vectorizer

Clase modificada - Breve descripción de la clase: Clase encargada de la transformación matemática de las respuestas a un espacio vectorial normalizado procesable por los algoritmos de clustering. Implementa las fases de ajuste (fit) y transformación (transform).

Cardinalidad

- 1:1 con VectorSchema: Contiene una instancia de VectorSchema que configura durante fit y utiliza durante transform.

Descripción de atributos

- schema: VectorSchema. La plantilla que define cómo convertir cada pregunta a dimensiones vectoriales.

Descripción de clases internas

- Point: Clase interna que representa un punto vectorizado.
    - id: String. ID del participante.
    - values: double[]. Array de valores normalizados entre 0 y 1.
    - missing: boolean[]. Máscara que indica qué dimensiones son nulas.

Descripción de las funciones

- fit(Cuestionario, List<Respuesta>): Analiza todas las respuestas para calcular estadísticas globales (outliers para las numéricas, moda para las demás y dimensiones)
- transform(Respuesta, Cuestionario): Point. Convierte una respuesta individual en un vector numérico.
- transformAll(List<Respuesta>, Cuestionario): List<Point>. Aplica transform a todas las respuestas.

2.2.18. Distance

Clase modificada - Breve descripción de la clase: Interfaz que define una métrica de distancia entre dos puntos vectorizados. Permite cambiar el tipo de distancia sin modificar los algoritmos de clustering.

Cardinalidad

- 1:1 con SSE: SSE tiene una distancia.
- 1:1 con Silhoutte: Silhoutte tiene una distancia.

Descripción de las funciones

- dist(Vectorizer.Point, Vectorizer.Point): Devuelve la distancia entre dos puntos vectoriales, según la métrica concreta de la implementación.
- calcularDistanciaNumerica(double, double): Devuelve la distancia numérica entre dos puntos normalizados.

2.2.19. GowerDistance

Clase modificada - Breve descripción de la clase: Implementación de Distance que calcula la distancia de Gower entre dos puntos normalizados en [0,1]: la media de las diferencias absolutas en las dimensiones presentes.

Cardinalidad

- Implementa la interfaz Distance

Descripción de las funciones

- double dist(Vectorizer.Point a, Vectorizer.Point b): Recorre todas las componentes, ignora las que están marcadas como missing en alguno de los puntos, suma $|x_i - y_i|$ y devuelve la media sobre las componentes válidas. Si todas están ausentes, devuelve 1.0.
- calcularDistanciaNumerica(double, double): Devuelve la distancia numérica entre dos puntos normalizados.

2.3. Capa de Persistencia

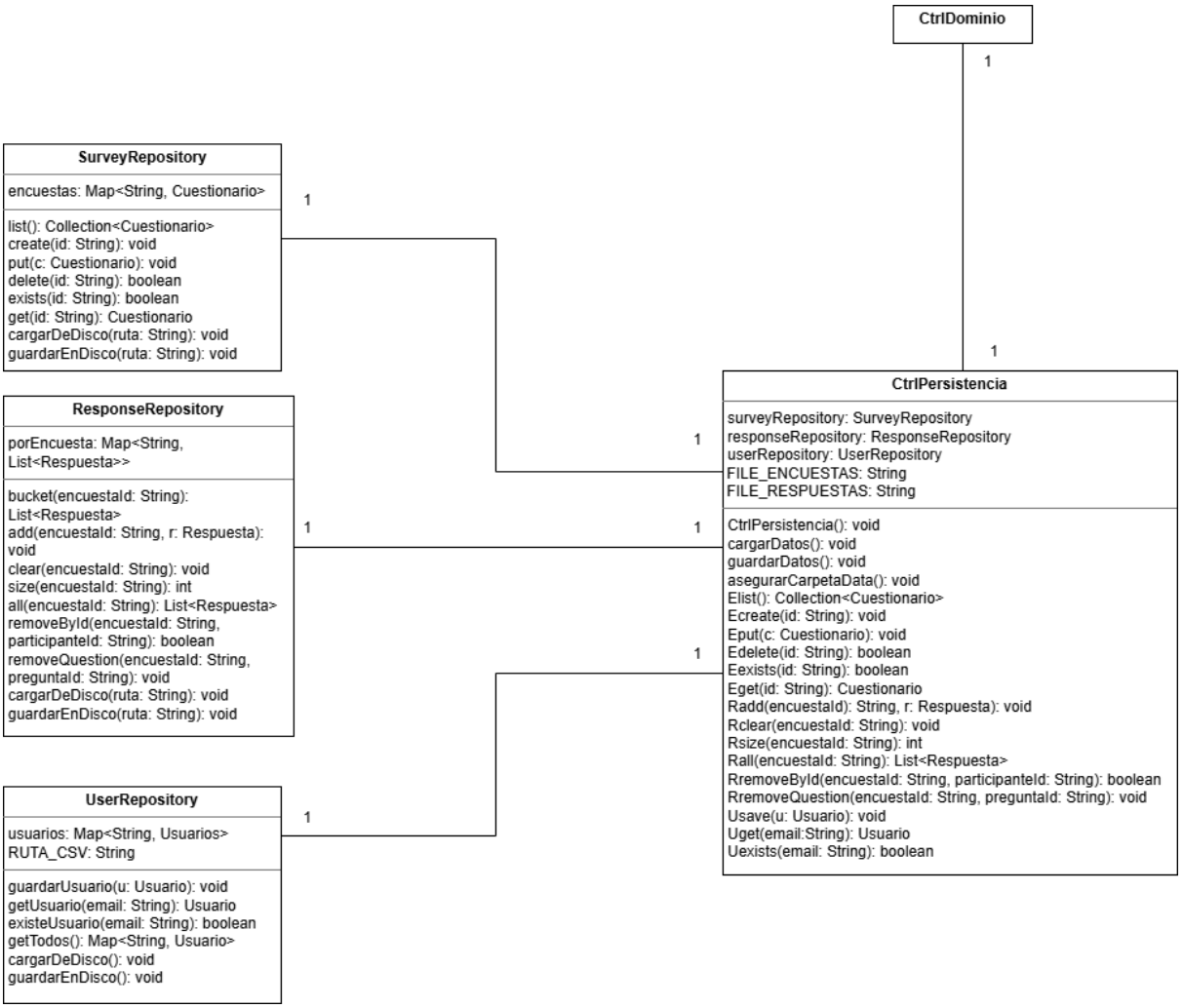

Diagrama de modelo de la capa de persistencia. El diagrama se puede ver com más detalle en el archivo Diagrama_Modelo_Persistencia de la carpeta Diagramas

Clases añadidas

Respecto a la primera entrega se ha añadido la capa de persistencia. Esta capa guarda los datos de la aplicación. Se ha creado un controlador para la capa (CtrlPersistencia) y se han añadido los repositorios (SurveyRepository, ResponseRepository) que estaban antes en dominio.

En la implementación del extra “Sistema de Login” se ha añadido la clase UserRepository, que guardará todos los usuarios de la aplicación.

En la implementación del extra “Sistema de guardado automático” también se han modificado las clases SurveyRepository y ResponseRepository para poder guardar el contenido de los repositorios al cerrar la aplicación.

2.3.1. CtrlPersistencia

Clase añadida - Breve descripción de la clase: El CtrlPersistencia actúa como el punto de entrada a la Capa de Persistencia. Su función principal es centralizar el acceso a los datos creando y manteniendo instancias de los Repositorios. De esta manera, aísla la Capa de Dominio de la implementación real de cómo se almacenan y recuperan los datos.

Cardinalidad

- 1:1 con CtrlDominio: Guarda la referencia a el controlador de dominio.
- 1:1 con SurveyRepository: Almacena una referencia a un único repositorio de encuestas.
- 1:1 con ResponseRepository: Almacena una referencia a un único repositorio de respuestas.
- 1:1 con UserRepository: Almacena una referencia a un único repositorio de usuarios.

Descripción de atributos

- surveyRepository: SurveyRepository. Referencia al repositorio encargado de la gestión de objetos Cuestionario.
- responseRepository: ResponseRepository. Referencia al repositorio encargado de la gestión de objetos Respuesta.
- userRepository: userRepository. Referencia al repositorio encargado de la gestión de objetos Usuario.
- FILE_ENCUESTAS, FILE_RESPUESTAS: String. Archivos de guardado.

Descripción de las funciones

- CtrlPersistencia(): Constructor. Inicializa las instancias de SurveyRepository y ResponseRepository.
- cargarDatos(): Carga los datos guardados al iniciar la aplicación.
- guardarDatos(): Guarda los datos de la aplicación al finalizar la ejecución.
- asegurarCarpetaData(): Crea una carpeta para guardar los datos si no existe.
- Elist(): Lista todas las Encuestas almacenadas.
- Ecreate(String): Crea una nueva Encuesta (objeto Cuestionario) por su ID.
- Eput(Cuestionario): Almacena o actualiza una Encuesta completa.
- Edelete(String): Elimina una Encuesta por su ID.
- Eexists(String): Comprueba si una Encuesta con un ID existe.

- Eget(String): Obtiene una Encuesta por su ID.
- Radd(String, Respuesta): Añade una Respuesta a una encuesta específica.
- Rclear(String): Elimina todas las respuestas asociadas a una encuesta.
- Rsize(String): Devuelve el número de Respuestas para una encuesta.
- Rall(String): Lista todas las Respuestas para una encuesta.
- RremoveById(String, String): Elimina una Respuesta específica por el ID del participante.
- RremoveQuestion(String, String): Elimina los valores de una pregunta específica en todas las respuestas de una encuesta.
- Usave(Usuario): Guarda el usuario en el repositorio.
- Uget(String): Devuelve el usuario.
- Uexists(String): Devuelve cierto si el usuario existe.

2.3.2. SurveyRepository

Clase añadida - Breve descripción de la clase: Implementa el repositorio y se encarga de la gestión de los objetos Cuestionario. Almacena estos objetos en memoria utilizando un Map. Es responsable de realizar las operaciones básicas de crear, modificar y eliminar para las encuestas.

Cardinalidad

- 1:1 con CtrlPersistencia: Guarda la referencia a su controlador de persistencia.

Descripción de atributos

- encuestas: Map<String, Cuestionario>. Estructura principal de almacenamiento, donde la clave es el ID de la encuesta y el valor es el objeto Cuestionario.

Descripción de las funciones

- list(): Devuelve todas las encuestas almacenadas.
- create(String): Crea una nueva instancia de Cuestionario con el ID dado y la almacena. Lanza una excepción si el ID ya existe o está vacío.
- put(Cuestionario): Almacena o sobrescribe una instancia de Cuestionario.
- delete(String): Elimina el Cuestionario asociado al ID.
- exists(String): Comprueba si un ID está presente en el mapa, es decir, si existe la encuesta.

- get(String): Recupera el objeto Cuestionario asociado al ID.
- cargarDeDisco(String): Carga las encuestas del archivo.
- guardarEnDisco(String): Guarda las encuestas en el archivo.

2.3.3. ResponseRepository

Clase añadida - Breve descripción de la clase: Implementa repositorio y se encarga de gestionar la colección de Respuesta por encuesta. Utiliza una estructura Map de listas para almacenar las respuestas, agrupándolas bajo el ID de la encuesta a la que pertenecen.

Cardinalidad

- 1:1 con CtrlPersistencia: Guarda la referencia a su controlador de persistencia.

Descripción de atributos

- porEncuesta: Map<String, List<Respuesta>>. Estructura principal para guardar las respuestas a las encuestas. La clave es el ID de la encuesta y el valor es una lista de todas las respuestas para esa encuesta.

Descripción de las funciones

- bucket(String): Función auxiliar privada que garantiza que existe una List<Respuesta> asociada al encuestaId. Si no existe, crea y retorna una nueva lista.
- add(String, Respuesta): Añade una respuesta a la lista de respuestas de la encuesta especificada.
- clear(String): Elimina toda la lista de respuestas asociada a una encuesta.
- size(String): Devuelve el número de respuestas para la encuesta.
- all(String): Devuelve una lista de todas las respuestas de una encuesta.
- removeById(String, String): Busca y elimina la respuesta con el participanteId dado dentro de la lista de respuestas de la encuestaId.
- removeQuestion(String, String): Itera sobre todas las respuestas de una encuesta y elimina el valor de una pregunta específica (preguntaId) de cada una.
- cargarDeDisco(String): Carga las respuestas del archivo.
- guardarEnDisco(String): Guarda las respuestas en el archivo.

2.3.4. UserRepository

Clase añadida - Breve descripción de la clase: Implementa el repositorio y se encarga de la gestión de los objetos Usuario. Almacena estos objetos en memoria utilizando un Map.

Cardinalidad

- 1:1 con CtrlPersistencia: Guarda la referencia a su controlador de persistencia.

Descripción de atributos

- usuarios: Map<String, Usuario>. Estructura principal para guardar laos usuarios. La clave es el email y el valor es el usuario.
- RUTA_CSV: String. Ruta del archivo donde se guardarán los datos.

Descripción de las funciones

- guardarUsuario(Usuario): Almacena un nuevo objeto Usuario, verificando que no sea nulo y que el email no esté ya en uso.
- getUsuario(String): Recupera el objeto Usuario asociado al correo electrónico proporcionado.
- existeUsuario(String): Comprueba si existe un usuario registrado con el email dado.
- getTodos(): Devuelve el mapa completo de usuarios almacenados en el repositorio.
- cargarDeDisco(): Carga los usuarios del archivo.
- guardarEnDisco(): Guarda los usuarios en el archivo.

3. Descripción ED&ALG. Funcionalidades Ppal.

Este apartado describe las estructuras de datos y algoritmos que constituyen el núcleo funcional del proyecto. En esta entrega, el énfasis ha estado en la refactorización arquitectónica para dotar al sistema de mayor extensibilidad, seguridad de tipos y robustez en el manejo de datos heterogéneos.

3.1. Estructuras de datos

El sistema se basa en entidades que representan encuestas, preguntas y respuestas. La principal evolución estructural ha sido la sustitución de implementaciones genéricas por jerarquías de clases tipadas.

3.1.1. Entidades del dominio

Una encuesta se modela mediante la clase Cuestionario, que mantiene lista e índice de preguntas. Las preguntas (Pregunta) incluyen enunciado, tipo y opciones.

Mejora importante: Las respuestas individuales (Respuesta) han evolucionado. En lugar de almacenar valores genéricos (Object) como en la primera versión, se ha implementado una jerarquía polimórfica ValorRespuesta con subclases específicas (ValorRespuestaNumerica, ValorRespuestaTextual, ValorRespuestaOpcionUnica, ValorRespuestaOpcionMultiple). Este cambio elimina la necesidad de castings inseguros y permite añadir lógica específica para cada tipo de dato.

Otra novedad de esta entrega es la introducción explícita de la entidad Usuario, que representa a cada persona registrada en el sistema. La clase Usuario almacena email (identificador), contraseña y datos básicos, y se usa para asociar cada Cuestionario con un propietario. De este modo, las operaciones de edición y eliminación se restringen al dueño de la encuesta.

Límite de selección en OPCION_MULTIPLE (maxSeleccion): La clase Pregunta incorpora el atributo maxSeleccion para restringir el número máximo de opciones seleccionables en preguntas de tipo OPCION_MULTIPLE. Este límite se usa tanto en la interfaz (validación al responder) como en importación/exportación, permitiendo encuestas con restricciones explícitas de selección.

3.1.2. Repositorios

Se mantienen SurveyRepository y ResponseRepository para la gestión en memoria, optimizados ahora para trabajar con las nuevas estructuras tipadas. Adicionalmente, se ha integrado UserRepository para soportar la autenticación y la propiedad de encuestas, garantizando el aislamiento de datos entre usuarios.

Para soportar esta funcionalidad, se ha incorporado el repositorio UserRepository, encargado de almacenar y recuperar usuarios por su email. Los controladores de dominio

consultan este repositorio para validar credenciales y comprobar permisos antes de permitir modificaciones sobre una encuesta.

3.2. Vectorización de datos

La transformación de respuestas a vectores numéricos se encapsula en el componente Vectorizer, que utiliza un VectorSchema para definir dinámicamente la representación vectorial a partir de los datos observados. Este proceso se divide claramente en dos fases: ajuste (fit) y transformación (transform).

Durante la fase fit, el vectorizador analiza todas las respuestas de una encuesta para construir el esquema de vectorización. Por convenio de implementación, la primera columna/pregunta del cuestionario se reserva como identificador del participante y no se incluye en la vectorización, por lo que el análisis comienza a partir de la segunda pregunta.

Para cada pregunta numérica, se recopilan los valores observados y se aplica un filtrado de valores extremos (outliers) basado en el Rango Intercuartílico (IQR). Se calculan los cuartiles Q1 y Q3 a partir de los valores ordenados y se definen los límites de Tukey:

- Límite inferior: Q1 − 1.5 × IQR
- Límite superior: Q3 + 1.5 × IQR donde IQR = Q3 − Q1.

Los valores que quedan fuera de este intervalo se consideran outliers y se excluyen de los cálculos estadísticos. Con los valores válidos restantes se calculan el mínimo, el máximo y la media, que se almacenan en el VectorSchema. La media se utiliza como valor por defecto para la imputación de valores faltantes. En el caso de preguntas categóricas (texto u opciones), se calcula y almacena la moda como valor representativo.

En la fase transform, cada respuesta individual se convierte en un objeto Point. Si una respuesta numérica está ausente o se detecta como outlier, se sustituye automáticamente por el valor por defecto (media). Si no existe un valor representativo válido, la dimensión correspondiente se marca como ausente. Las preguntas numéricas se normalizan al rango [0,1] mediante normalización min–max; si el rango es degenerado (max = min), se asigna el valor 0.5 como punto neutro.

Las preguntas categóricas se codifican mediante one-hot encoding. En el caso de preguntas de selección múltiple, el peso se reparte entre todas las opciones seleccionadas. El resultado final es un vector numérico acompañado de una máscara explícita de valores ausentes (boolean[] missing), que permite a las métricas de distancia ignorar dimensiones sin información en lugar de asumir valores artificiales.

Esta combinación de imputación de valores faltantes, filtrado de outliers y normalización controlada mejora la robustez del proceso de clustering, evitando que datos incompletos o anómalos distorsionen los resultados.

3.3. Cálculo de distancias

La lógica de cálculo de similitud se ha desacoplado de los algoritmos mediante el patrón Strategy. Se ha definido la interfaz Distance, siendo GowerDistance su implementación principal.

Ventaja arquitectónica: Esto permite que tanto K-Means como K-Medoids utilicen la misma lógica de distancia sin duplicar código. Si se quisiera añadir distancia Euclídea en el futuro, bastaría con crear una nueva implementación de Distance sin modificar los algoritmos existentes.

3.4. Algoritmos de clustering

Los algoritmos de clustering implementados en el sistema son K-Means, K-Means++ y K-Medoids (PAM). En esta entrega, todos ellos han sido refactorizados bajo la clase abstracta ClusteringAlgorithm, siguiendo el patrón Template Method. Esta abstracción define la estructura general de ejecución y centraliza la lógica común, como la inicialización básica de centros, la asignación de puntos al centro más cercano y el uso de la métrica de distancia.

Gracias a esta refactorización, los distintos algoritmos comparten una interfaz homogénea de ejecución (run(...)) y pueden intercambiarse fácilmente sin afectar al resto del sistema. La selección del algoritmo se realiza a nivel del controlador (CtrlClustering), que delega completamente la ejecución en la implementación concreta.

3.4.1. K-means y K-means++

El algoritmo K-Means implementa el proceso iterativo clásico de clustering basado en centroides. A partir de un conjunto de puntos vectorizados, el algoritmo inicializa un conjunto de centros y repite el siguiente ciclo hasta la convergencia o hasta alcanzar un número máximo de iteraciones:

1. Asignación de cada punto al centro más cercano utilizando la métrica de distancia configurada.
2. Recomputación de los centroides como el promedio componente a componente de los puntos asignados a cada cluster, respetando la máscara de valores ausentes.
3. Comprobación de convergencia mediante la comparación entre los centroides antiguos y los nuevos.

El sistema soporta dos estrategias de inicialización de centros:

- Inicialización simple, mediante muestreo aleatorio de puntos del conjunto de datos.
- Inicialización K-Means++, implementada de forma desacoplada en la clase KMeansPlusPlus. Esta estrategia selecciona los centros iniciales de manera probabilística, favoreciendo puntos alejados entre sí y reduciendo la probabilidad de converger a mínimos locales.

La clase KMeans decide qué estrategia de inicialización utilizar en función de los parámetros recibidos, pero el resto del ciclo del algoritmo permanece idéntico en ambos casos. El resultado final se encapsula en un objeto ClusterResult, que incluye las asignaciones de cada punto a su cluster y los centroides finales como representantes.

3.4.2. K-medoids (PAM)

El algoritmo K-Medoids, implementado siguiendo la variante PAM (Partitioning Around Medoids), utiliza como representantes de los clusters puntos reales del conjunto de datos, en lugar de centroides sintéticos. Esto lo hace especialmente robusto frente a valores atípicos y adecuado para datos mixtos o categóricos.

El algoritmo comienza seleccionando un conjunto inicial de medoides mediante muestreo aleatorio. A continuación, asigna cada punto al medoide más cercano y evalúa posibles intercambios entre medoides actuales y otros puntos del conjunto. Un intercambio se acepta únicamente si reduce el coste total del clustering, definido como la suma de las distancias de cada punto a su medoide más cercano.

Al igual que K-Means, K-Medoids hereda de ClusteringAlgorithm, lo que garantiza que su integración con el sistema sea idéntica en términos de invocación, uso de la métrica de distancia y encapsulación del resultado. El algoritmo devuelve un ClusterResult cuyos representantes son los propios medoides, es decir, respuestas reales de participantes.

3.5. Métricas de evaluación

Para evaluar la calidad de los resultados de clustering, el sistema implementa tres métricas complementarias: SSE, Silhouette y ClusteringAccuracy. Todas ellas se han integrado de forma coherente con la arquitectura refactorizada del proyecto.

Una característica clave de esta entrega es que todas las métricas utilizan la misma interfaz Distance que los algoritmos de clustering, garantizando que la evaluación sea consistente con la noción de similitud empleada durante la ejecución del algoritmo. Esto evita discrepancias entre el proceso de agrupamiento y su posterior análisis.

- **SSE (Sum of Squared Errors):** mide la cohesión interna de los clusters calculando la suma de las distancias al cuadrado entre cada punto y el representante de su cluster (centroide o medoide). Valores más bajos indican clusters más compactos.
- **Índice de Silhouette:** evalúa simultáneamente la cohesión interna y la separación entre clusters. Para cada punto, compara la distancia media a los puntos de su propio cluster con la distancia media al cluster más cercano distinto. El valor final es la media de todos los puntos y se encuentra en el intervalo [-1, 1], donde valores cercanos a 1 indican una buena separación.
- **ClusteringAccuracy:** cuando se dispone de etiquetas de verdad externas, esta métrica permite evaluar la correspondencia entre los clusters obtenidos y las clases reales. El sistema asigna a cada cluster la etiqueta predominante y calcula el porcentaje de coincidencias globales.

Gracias al uso de vectores con máscara explícita de valores ausentes (boolean[] missing), las métricas pueden ignorar dimensiones sin información en lugar de asumir valores artificiales. Esto mejora la robustez de los cálculos, especialmente en métricas costosas como Silhouette, y permite evaluar resultados obtenidos a partir de datos incompletos o heterogéneos de forma más fiable.

4. Relación de Clases por Miembro del Grupo

En esta entrega, el equipo ha trabajado en la refactorización de la arquitectura hacia un diseño en 3 capas, la implementación de una interfaz gráfica completa y la mejora de los algoritmos y modelo de datos. A continuación se detallan las responsabilidades y nuevas implementaciones de cada miembro.

4.1. Aleks: Interfaz Gráfica (Visualización y Testing)

Aleks ha complementado el desarrollo de la interfaz gráfica enfocándose en la interactiva de resultados de clustering, la evaluación de su calidad, y el mantenimiento de la coherencia del sistema de testing con los nuevos cambios arquitectónicos.

Nuevas implementaciones:

● Visualización de Clustering:

- PanelClustering (Nueva): Panel gráfico para la configuración de algoritmos (selección de K, algoritmo, encuesta), ejecución de clustering, y visualización textual de métricas (SSE, Silhouette, tiempo de ejecución).
- DialogoGrafica (Nueva): Ventana emergente para la visualización 2D de los clústers mediante Scatter Plot interactivo. Permite seleccionar dos dimensiones de los datos vectorizados y dibuja los puntos coloreados según su asignación a cluster.

● Exportación de Resultados:

- CsvClusterExporter (Mejorada): Exporta el clustering a CSV incluyendo metadatos (algorithm, SSE, Silhouette, time_ms), bloque de representantes por cluster y, para centroides sintéticos (K-Means/K-Means++), calcula y reporta el participante real más cercano a cada centroide (nearest + distancia).

● Testing y Coherencia:

- Drivers y Testing: Mantenimiento integral del sistema de testing con los nuevos cambios arquitectónicos, asegurando que todos los módulos funcionen correctamente de forma integrada y que se cumplan los criterios de calidad.

4.2. Yassin: Análisis, Vectorización y Algoritmos de Clustering

Yassin ha liderado todo el pipeline de clustering, desde la transformación de datos brutoshasta los algoritmos de agrupamiento y su evaluación. Su trabajo asegura que el flujo de análisis sea coherente, robusto y eficiente.

Nuevas implementaciones y refactorizaciones:

● Vectorización y Transformación de Datos:

- Vectorizer (Nueva): Clase encargada de transformar respuestas crudas a vectores numéricos normalizados, incluyendo:
  - Detección de outliers mediante Rango Intercuartílico (IQR)
  - Imputación automática de valores faltantes (media/moda)
  - Normalización al rango [0, 1]
  - Codificación one-hot para categóricas
- VectorSchema (Modificada): Estructura que almacena estadísticas globales (rangos, cuartiles, valores por defecto) necesarias para la transformación de datos.

● Algoritmos de Clustering:

- ClusteringAlgorithm (Mejorada): Clase abstracta que define el patrón Template Method para todos los algoritmos, centralizando lógica común.
- Adaptación de KMeans y KMedoids: Modificación para heredar de ClusteringAlgorithm y trabajar con la nueva estructura de datos vectorizados y Distance.
- KMeansPlusPlus (Refactorizada): Clase auxiliar responsable exclusivamente de la inicialización de centroides mediante la estrategia K-Means++. Aunque la inicialización ya existía en la primera entrega, en esta versión se ha desacoplado completamente del algoritmo principal KMeans, mejorando la separación de responsabilidades y la mantenibilidad del código.

● Métricas y Evaluación:

- Adaptación de SSE y Silhouette: Ajuste para trabajar con la interfaz Distance y gestionar valores nulos (missing values) mediante máscara boolean[].
- GowerDistance (Modificada): Actualización para compatibilidad con nuevos vectores.
- ClusteringAccuracy (Nueva): Utilidad para evaluar la precisión de clustering comparando resultados con etiquetas de verdad.

● Controlador de Clustering:

- CtrlClustering (Modificada): Controlador que coordina:
  - Vectorización de datos
  - Ejecución de algoritmos
  - Cálculo de métricas
  - Almacenamiento de resultados

4.3. Pol: Núcleo del Dominio y Modelo de Datos

Pol se ha encargado de una de las mejoras arquitectónicas más importantes: la eliminación de tipos genéricos en favor de una jerarquía de clases polimórfica, junto con la implementación de los controladores centrales que coordinan toda la lógica de negocio.

Nuevas implementaciones y refactorizaciones:

● Jerarquía de Respuestas (Modelo de Datos Polimórfico):

- ValorRespuesta (Nueva Abstracta): Clase base para todos los tipos de respuesta. Implementa Serializable.
- ValorRespuestaNumerica, ValorRespuestaTextual, ValorRespuestaOpcionUnica, ValorRespuestaOpcionMultiple (Nuevas): Implementaciones concretas que encapsulan la lógica específica de cada tipo de dato (normalización, conversión a string, etc.). Todas implementan Serializable.

● Modelo de Datos Principal:

- Respuesta (Modificada): Refactorización para usar LinkedHashMap<String, ValorRespuesta> en lugar de Map<String, Object>. Implementa Serializable.
- Pregunta (Modificada): Añadido parámetro maxSeleccion para limitar opciones en preguntas OPCION_MULTIPLE. Implementa Serializable.
- Cuestionario (Modificada): Actualizado ownerId para almacenar email del propietario del cuestionario. Implementa Serializable.
- Usuario (Nueva): Entidad que representa a los usuarios del sistema para control de acceso. Implementa Serializable.

● Controladores del Dominio (Centro de Coordinación):

- CtrlDominio (Nueva): Controlador central que actúa como fachada para toda la lógica de negocio, coordinando encuestas, respuestas y clustering. Delega operaciones a controladores especializados.
- CtrlEncuesta (Modificada): Controlador especializado en gestión de cuestionarios (crear, obtener, listar, eliminar, agregar/eliminar preguntas).
- CtrlRespuesta (Modificada): Controlador especializado en gestión de respuestas (agregar, listar, eliminar).

● Persistencia de Usuarios:

- UserRepository (Nueva): Repositorio para persistencia de usuarios y credenciales, almacenando datos en disco.

4.4. Santi: Interfaz Gráfica (Gestión de Datos y Login)

Santi ha coordinado todo el sistema de persistencia y autenticación, integrando la importación/exportación de datos con una interfaz gráfica moderna y asegurando que todos los datos se guarden automáticamente al cerrar la aplicación.

Nuevas implementaciones:

● Autenticación y Control de Usuarios:

- PanelLogin (Nueva): Interfaz visual para login y registro de usuarios, sustituyendo completamente la interacción por consola de la entrega anterior.

● Gestión de Datos (Import/Export):

- PanelDatos (Nueva): Panel gráfico para importar y exportar encuestas completas en formato CSV desde la interfaz gráfica.
- CsvSurveyImporter (Nueva): Adaptador para importar encuestas desde CSV invocado desde la GUI.
- Formato CSV ampliado: En la fila de opciones se usa | como separador, y para OPCION_MULTIPLE se añade <<MAX:n>> para persistir maxSeleccion.
- Importación de respuestas múltiple: Las respuestas de selección múltiple se leen separadas por | (no por comas).

● Capa de Persistencia:

- CtrlPersistencia (Nueva): Controlador central de persistencia que coordina todos los repositorios y maneja la serialización automática de objetos Java a disco.
- SurveyRepository (Nueva): Repositorio para persistencia de encuestas (Cuestionario y Pregunta) en disco.
- ResponseRepository (Nueva): Repositorio para persistencia de respuestas en disco.

● Automatización:

- App (Modificada): Implementación del ShutdownHook que guarda automáticamente todos los datos (encuestas, respuestas, usuarios) al cerrar la aplicación, gracias a que todas las clases del dominio implementan Serializable. El usuario no necesita guardar manualmente.

4.5. Anna: Interfaz Gráfica (Vistas Principales y Controladores)

Anna ha liderado el desarrollo de la capa de presentación, diseñando la página principal de la aplicación y los paneles fundamentales de interacción con el usuario, así como el controlador que comunica las vistas con la lógica de negocio.

Nuevas implementaciones:

● Estructura Principal:

- VistaPrincipal (Nueva): Ventana principal que organiza la navegación mediante CardLayout, permitiendo alternar entre vistas (Login, Menú, Encuestas, Respuestas, Clustering, Datos). Actúa como contenedor de todos los paneles de la aplicación.
- CtrlPresentacion (Nueva): Controlador de la capa de presentación que actúa como fachada, desacoplando las vistas de la lógica de negocio y delegando todas las operaciones a CtrlDominio.

● Paneles de Gestión:

- PanelEncuestas (Nueva): Panel para el CRUD de encuestas, visualización de preguntas en tabla y acciones de crear/eliminar/modificar preguntas.
- PanelRespuestas (Nueva): Panel para visualizar respuestas de una encuesta seleccionada en formato tabular.

● Diálogo de Respuestas:

- DialogoResponder (Nueva): Ventana modal emergente que permite a usuarios rellenar encuestas de forma interactiva, con componentes dinámicos según el tipo de pregunta.
    - InputComponent (Clase Interna): Interfaz que define el contrato para componentes que capturan respuestas.
    - TextInput (Clase Interna): Implementa InputComponent para preguntas NUMERICA y TEXTO usando JTextField.
    - RadioInput (Clase Interna): Implementa InputComponent para OPCION_UNICA usando JRadioButton.
    - CheckboxInput (Clase Interna): Implementa InputComponent para OPCION_MULTIPLE usando JCheckBox, con validación de maxSeleccion.

5. Código Modelo | Funcionalidades Ppal.

En esta entrega no se ha reescrito el modelo desde cero, sino que se han ido ampliando y refactorizando las clases existentes para mejorar la organización del código y soportar nuevas funcionalidades. A continuación se describen los cambios más relevantes respecto a la primera versión y cómo se han implementado los extras acordados.

5.1. Cambios en el modelo de datos

En la primera entrega, la clase Respuesta almacenaba los valores en un Map<String, Object>, lo que obligaba a hacer castings según el tipo de pregunta y complicaba la validación. En esta entrega se ha sustituido por una jerarquía de clases:

- ValorRespuesta (abstracta)
- ValorRespuestaNumerica
- ValorRespuestaTextual
- ValorRespuestaOpcionUnica
- ValorRespuestaOpcionMultiple

Respuesta pasa a ser un Map<String, ValorRespuesta>. Este cambio simplifica el código del vectorizador y de los controladores, porque cada tipo de valor tiene su propia lógica y se evitan errores por cast incorrectos. Además, facilita añadir nuevos tipos de respuesta si hiciera falta.

También se ha añadido explícitamente la clase Usuario, que representa a cada persona registrada en el sistema. Usuario almacena email, contraseña y algunos datos básicos. Cada Cuestionario tiene ahora un campo “owner” que guarda el email del propietario. Esta relación se utiliza en el dominio para comprobar que solo el dueño puede modificar o eliminar una encuesta.

5.2. Sistema de Login y control de permisos

Para soportar un entorno multiusuario y garantizar el aislamiento de datos, en esta entrega se ha implementado un sistema completo de autenticación y control de permisos, integrado tanto en la capa de presentación como en la capa de dominio y persistencia.

El sistema incorpora la entidad Usuario, que representa a cada persona registrada en la aplicación. Cada usuario se identifica de forma única mediante su correo electrónico (email) y almacena, además, una contraseña y datos personales básicos. Los usuarios se gestionan a través del repositorio UserRepository, que permite su almacenamiento y recuperación persistente.

Desde la capa de presentación, el usuario puede registrarse y iniciar sesión mediante el panel gráfico de autenticación (PanelLogin). El controlador de presentación delega estas operaciones en el controlador de dominio (CtrlDominio), que valida las credenciales y mantiene el estado del usuario actualmente autenticado durante la sesión.

Cada encuesta (Cuestionario) incorpora un atributo ownerId que almacena el correo electrónico del usuario propietario. Este valor se establece automáticamente al crear una nueva encuesta o al importar una encuesta desde un archivo CSV, asociándola al usuario que realiza la operación. A partir de esta información, el sistema aplica las siguientes reglas de control de permisos:

- Solo el usuario propietario puede modificar la estructura de una encuesta (añadir o eliminar preguntas).
- Solo el usuario propietario puede eliminar una encuesta y sus respuestas asociadas.
- Las operaciones de visualización y análisis (listado, clustering, exportación de resultados) están disponibles para los usuarios con sesión iniciada, respetando el aislamiento de encuestas por propietario cuando corresponde.

El controlador de dominio utiliza esta información para validar las operaciones antes de delegarlas a los controladores especializados (CtrlEncuesta, CtrlRespuesta y CtrlClustering). De este modo, se evita que un usuario pueda acceder o modificar encuestas que no le pertenecen, manteniendo la coherencia y seguridad del sistema sin introducir dependencias directas entre la capa de presentación y el modelo de datos.

5.3. Vectorización e imputación (media/moda)

El código del Vectorizer se ha ampliado significativamente respecto a la primera entrega para mejorar la calidad de los datos. Se han implementado dos estrategias conjuntas:

1. Imputación de valores faltantes (Missing Values): En la primera entrega, si una respuesta estaba vacía, simplemente se marcaba como missing y esa dimensión se ignoraba en el cálculo de distancias. Ahora, durante la fase de entrenamiento (fit), el sistema calcula y guarda un valor "representativo" para cada pregunta:
    - Para preguntas numéricas: la media de los valores válidos observados.
    - Para preguntas categóricas: la opción más frecuente (moda).
2. Detección y tratamiento de valores extremos (Outliers): Se ha añadido una fórmula estadística basada en el Rango Intercuartílico (IQR) para detectar datos anómalos (muy grandes o muy pequeños). El sistema calcula los cuartiles Q1 y Q3, y define unos límites de validez:
    - Límite Inferior: Q1 - 1.5 × IQR
    - Límite Superior: Q3 + 1.5 × IQR
    - Donde IQR = Q3 - Q1

Durante la fase de transformación (transform), cualquier respuesta cuyo valor numérico caiga fuera de estos límites se considera "ruido" y se marca automáticamente como nulo. El sistema lo reemplaza después por la media calculada de los valores válidos, asegurando que los datos extremos no afecten negativamente a la calidad de los clústeres generados.

Integración: La lógica combinada funciona de la siguiente manera: cuando el vectorizador encuentra un valor, primero verifica si es un outlier. Si lo es, o si el valor original ya estaba vacío, el sistema lo sustituye automáticamente por el valor representativo (media o moda) calculado previamente. Esto permite no perder la respuesta completa del participante por un solo dato erróneo o faltante, suavizando los datos para el clustering.

5.4. Refactorización de algoritmos de clustering

Los algoritmos KMeans y KMedoids ya existían en la primera entrega, pero su código compartía lógica duplicada (búsqueda de centro más cercano, asignación de puntos, etc.). En esta entrega se ha introducido la clase base ClusteringAlgorithm, que agrupa esa lógica común y define la interfaz run(...). Las clases concretas solo se encargan de los pasos específicos de cada algoritmo. Esto ha reducido la cantidad de código repetido y hace más fácil mantener o añadir variaciones en el futuro.

La inicialización K-Means++ se desacopla en una clase específica (KMeansPlusPlus), y KMeans la invoca cuando se activa el modo “++”, manteniendo el ciclo de iteración y convergencia del algoritmo sin duplicación.

5.5. Integración de métricas y distancia

Las métricas SSE, Silhouette y ClusteringAccuracy ya estaban implementadas, pero ahora se apoyan directamente en la interfaz Distance y en los nuevos vectores con máscara de ausencias. Esto significa que:

- Usan exactamente la misma métrica de distancia que los algoritmos (coherencia entre ejecución y evaluación).
- Pueden ignorar dimensiones marcadas como missing en lugar de asumir ceros u otros valores artificiales.

El código de evaluación en CtrlClustering se ha simplificado: ejecuta el algoritmo, guarda ClusterResult y luego llama a las clases de métricas para obtener SSE y Silhouette. Para Accuracy, el controlador lee el CSV de etiquetas, lo cruza con las asignaciones del resultado y delega el cálculo a la clase correspondiente.

5.6. Cambios y Funcionalidades Adicionales

En esta entrega, hemos diferenciado entre las mejoras estructurales necesarias para corregir deficiencias de la primera versión y las nuevas funcionalidades "extra" implementadas para aportar valor añadido al proyecto.

5.6.1. Mejoras de la Primera Entrega

Estos cambios estructurales eran necesarios para cumplir correctamente con los requisitos de diseño iniciales:

- **Refactorización de la clase Respuesta:** Se ha eliminado la clase única Respuesta que almacenaba un Map<String, Object> para implementar una jerarquía adecuada mediante composición. Ahora Respuesta contiene un mapa de ValorRespuesta, y se han creado subclases específicas (ValorRespuestaNumerica, ValorRespuestaTextual, ValorRespuestaOpcionUnica, ValorRespuestaOpcionMultiple) que heredan de la clase abstracta ValorRespuesta. Esto facilita la escalabilidad, evita castings inseguros, y permite añadir lógica específica para cada tipo de dato. Implementa Serializable para persistencia.

- **Abstracción del Algoritmo de Clustering:** Se ha introducido la clase abstracta ClusteringAlgorithm para desacoplar la lógica de agrupamiento. Esto permite intercambiar algoritmos (K-Means, K-Medoids) sin afectar al resto del sistema, cumpliendo con el principio de Inversión de Dependencias.

- **Integración de métricas y distancia:** Las métricas SSE, Silhouette y ClusteringAccuracy se apoyan ahora directamente en la interfaz Distance y en los nuevos vectores con máscara de ausencias (boolean[] missing). Esto permite que ignoren dimensiones vacías en lugar de asumir valores artificiales.

5.6.2. Extras Implementados

Se han añadido las siguientes funcionalidades que no estaban contempladas en el alcance básico obligatorio:

- **Sistema de Login y control de permisos:** Implementación de autenticación de usuarios mediante la clase Usuario y el repositorio UserRepository. Cada encuesta queda asociada a un propietario (ownerId), restringiendo las operaciones de edición y eliminación únicamente al usuario creador. Los usuarios se persisten en disco mediante serialización.
- **Tratamiento de valores faltantes (Missing Values):** Durante la fase de vectorización (fit), el sistema calcula valores representativos para cada pregunta (media para numéricas y moda para categóricas). En la fase de transformación (transform), las respuestas vacías se imputan automáticamente con estos valores para evitar la pérdida de información en el clustering.
- **Detección y tratamiento de valores extremos (Outliers):** Implementación de un método estadístico basado en el Rango Intercuartílico (IQR) para detectar respuestas numéricas anómalas. Los valores fuera de los límites definidos ($Q1 - 1.5 \cdot IQR$, $Q3 + 1.5 \cdot IQR$) se consideran ruido y se sustituyen por el valor representativo correspondiente, reduciendo su impacto en el análisis.
- **Sistema de guardado automático:** Incorporación de un mecanismo de persistencia automática mediante ShutdownHook, que guarda encuestas, respuestas y usuarios al cerrar la aplicación. Todas las clases del dominio implementan Serializable, permitiendo la recuperación completa del estado del sistema sin intervención del usuario.
- **Formato CSV ampliado para preguntas de opción múltiple:** La exportación de encuestas (CsvSurveyExporter) serializa las opciones usando el separador | e incluye el sufijo <<MAX:n>> para persistir el límite de selección (maxSeleccion). La importación (CsvSurveyImporter) detecta y reconstruye correctamente esta información.
- **Exportación enriquecida de resultados de clustering:** El exportador de clustering (CsvClusterExporter) incluye metadatos del análisis (algoritmo, SSE, Silhouette y tiempo de ejecución), la asignación de participantes a clusters y un bloque de representantes. En el caso de centroides sintéticos (K-Means / K-Means++), se calcula y reporta el participante real más cercano usando la distancia de Gower.

6. Testing

Para empezar, cabe destacar que vamos a hacer un mayor énfasis en la explicación y profundidad del testing de los juegos de prueba en esta documentación. Al igual que, siguiendo los comentarios del profesor de la asignatura, hemos implementado ejecutables automáticos para cada juego de prueba, que comparan el output obtenido de la ejecución, con un output prefijado correcto, para así ver la correcta ejecución de nuestro programa.

Introducción al Plan de Pruebas:

En este apartado se detallan las pruebas realizadas para verificar la corrección funcional, la integridad y la robustez del sistema desarrollado. El objetivo principal ha sido validar el cumplimiento de los requisitos funcionales definidos, asegurando que la aplicación responde correctamente tanto en flujos de éxito como en situaciones de error o datos atípicos.

Metodología:

La estrategia de pruebas adoptada se basa en pruebas de sistema (System Testing) con un enfoque principal de Caja Negra (Black-Box). Las pruebas se han diseñado centrándose en las entradas y salidas del sistema, verificando que el comportamiento observado coincide con el especificado en los casos de uso, independientemente de la implementación interna. Adicionalmente, se han incluido pruebas de Caja Blanca para validar caminos específicos, como la gestión de excepciones, límites de arrays y consistencia de algoritmos matemáticos (Clustering). Se efectúa el uso de clases drivers que permiten realizar las peticiones de menú correspondientes desde terminal.

Entorno de Automatización:

Para garantizar la reproducibilidad y facilitar la detección de regresiones, se ha implementado un conjunto de pruebas automatizado ubicado en el directorio /EXE. El mecanismo de prueba funciona de la siguiente manera:

Inyección de Datos: Cada prueba ejecuta el driver principal del sistema inyectando un fichero de texto predefinido (Input) que simula la interacción de un usuario real a través de la consola.

Captura de Resultados: La salida generada por el programa se captura y se almacena en un fichero temporal.

Verificación: Se compara automáticamente la salida obtenida con un fichero de "Salida Esperada" (validado previamente) mediante herramientas de diferenciación (diff). Esto permite verificar el éxito de la prueba, viendo si ha pasado o ha fallado.

Organización: Los tests constan de 24 juegos de prueba agrupados en 7 categorías temáticas (de la A a la G), que cubren desde la gestión básica de datos hasta flujos complejos de integración y análisis de algoritmos. A continuación, se describe exhaustivamente cada prueba, detallando su objetivo, los valores de entrada y los efectos esperados en el sistema. Además, disponemos de un ejecutable *ejecutar_bateria_tests.sh*

el cual ejecuta los 24 juegos de prueba seguidos y asimismo nos indica si han pasado todos, o por el contrario ha fallado alguno. En el caso hipotético de que alguno fallase, se crearía un fichero log con los fallos.

A continuación se detallan las pruebas de sistema realizadas. Para cada prueba se especifica el flujo de entrada predefinido (ficheros `.txt` de input) y la salida que el sistema debe generar (ficheros `output_esperado` de referencia).

A continuación se detallan las pruebas de sistema realizadas. Para cada prueba se especifica el flujo de entrada predefinido (ficheros `.txt` de input) y la salida que el sistema debe generar (ficheros `output_esperado` de referencia). cabe destacar que, en todos y cada uno de los juegos de prueba, al principio se realiza el proceso de registro e inicio de sesión del usuario, y en los casos en los que se quiera responder a la encuesta desde varios usuarios, se procede a realizar el cierre de sesión, el registro del usuario nuevo y el inicio de sesión por tal de realizar correctamente las respuestas a los cuestionarios.

6.1. Gestión de encuestas (Grupo A)

Objetivo: Validar el CRUD (Crear, Leer, Actualizar, Borrar) de la clase `Cuestionario`.

Juego A1: Crear y listar encuesta básica

- **Objeto de la prueba:** Verificar que el sistema instancia correctamente un `Cuestionario` y lo almacena en el `SurveyRepository`.
- **Entrada prefijada (`JuegoA1_CrearYListarEncuestaBasica.txt`):** Se navega al menú de encuestas, se selecciona "Crear", se introduce el ID encA1 y se vuelve al menú principal para seleccionar "Listar".
- **Salida esperada:** El sistema confirma "Encuesta creada con éxito" y, al listar, muestra la encuesta con el ID asignado.
- **Relación E/S:** La entrada provoca la creación del objeto en memoria. La salida confirma que el objeto persiste en el repositorio durante la ejecución.

Juego A2: Crear encuesta con todos los tipos de pregunta

- **Objeto de la prueba:** Validar el polimorfismo de la clase `Pregunta` y sus subclases.
- **Entrada prefijada (`JuegoA2_CrearConPreguntasDeTodosLosTipos.txt`):** Creación secuencial de preguntas introduciendo:
    1. Numero
    2. Texto libre.
    3. Opción Única (con 3 opciones definidas).
    4. Opción Múltiple (con 3 opciones definidas).

- **Salida esperada:** La visualización de la encuesta muestra cada pregunta con su formato específico:

  Listando preguntas de la encuesta 'encA2': 1) P1_edad NUMERICA 2) P2_nombre TEXTO 3) P3_opcion_unica OPCION_UNICA Opciones: [A, B, C] 4) P4_opcion_multiple OPCION_MULTIPLE Opciones: [Deporte, Musica, Cine] (max=0)

- **Relación E/S:** El input activa la *Factory* de preguntas correspondiente. La salida verifica que cada subclase se renderiza con su método `toString()` o visualizador específico.

Juego A3: Editar encuesta, agregar y eliminar pregunta

- **Objeto de la prueba:** Verificar la mutabilidad del cuestionario.
- **Entrada prefijada (`JuegoA3_EditarEncuesta_AgregarYEliminarPregunta.txt`):** se crea una encuesta, se selecciona la encuesta, se añaden 3 preguntas, luego se elimina la segunda pregunta por su índice.
- **Salida esperada:** El listado final muestra la encuesta sin la pregunta que se ha borrado, y reindexando las posiciones.
- **Relación E/S:** Confirma que las listas dinámicas (`ArrayList` de preguntas) dentro del dominio se actualizan correctamente tras las operaciones.

Juego A4: Eliminar encuesta y respuestas

- **Objeto de la prueba:** Validar la integridad referencial al borrar preguntas y encuestas, que un usuario que no haya creado una encuesta no se le permita borrarla.
- **Entrada prefijada (`JuegoA4_EliminarEncuestaYRespuestas.txt`):** Se crea un usuario, crea una encuesta, le añade preguntas, las responden, y luego se se selecciona "Eliminar" sobre la encuesta, que ya tiene respuestas registradas.
- **Salida esperada:** Mensaje de éxito. Al intentar listar encuestas o respuestas asociadas, se dice que no hay encuestas.
- **Relación E/S:** La acción de borrado en el controlador de dominio debe limpiar tanto el repositorio de encuestas como filtrar el repositorio de respuestas para borrar las huérfanas.

6.2. Gestión de respuestas y usuarios (Grupo B)

Objetivo: Validar la interacción entre `Usuario`, `Cuestionario` y `Respuesta`.

Juego B1: Múltiples respuestas de distintos usuarios

- **Objeto de la prueba:** Simular un entorno multi-usuario, agregando respuestas de distintos participantes.
- **Entrada prefijada (`JuegoB1_AgregarVariasRespuestasDistintosParticipantes.txt`):** Ciclo repetitivo: Registrar Usuario A -> Responder Encuesta -> Logout -> Registrar Usuario B -> Responder misma Encuesta.
- **Salida esperada:** El sistema muestra las respuestas al consultar los detalles de la encuesta, diciendo de que usuario es cada una.
- **Relación E/S:** Verifica que el sistema vincula cada respuesta a un ID de usuario único distinto, permitiendo múltiples instancias de `Respuesta` para un mismo `Cuestionario`.

Juego B2: Eliminar respuesta específica

- **Objeto de la prueba:** Precisión en la gestión de datos.
- **Entrada prefijada (`JuegoB2_EliminarRespuestaDeParticipante.txt`):** Se listan las respuestas de un usuario y se selecciona una por ID para borrarla.
- **Salida esperada:** La respuesta del usuario desaparece del listado de respuestas, pero el resto de respuestas de otros usuarios se mantienen intactas.
- **Relación E/S:** Prueba que el método de borrado utiliza el identificador único de respuesta y no afecta a la estructura global.

Juego B3: Eliminar todas las respuestas (Reset)

- **Objeto de la prueba:** Funcionalidad de limpieza para reiniciar un estudio.
- **Entrada prefijada (`JuegoB3_EliminarTodasLasRespuestas.txt`):** Opción "Eliminar todas las respuestas" sobre una encuesta con datos.
- **Salida esperada:** Contador de respuestas a 0. La encuesta sigue existiendo con sus preguntas, pero con ninugna respuesta:

  Encuestas disponibles:

  - encB3 (preguntas=2, respuestas=0)

  No hay respuestas para esta encuesta

- **Relación E/S:** Comprueba que se vacía la lista de respuestas vinculada a la encuesta sin destruir el objeto encuesta en sí.

Juego B4: Gestión de respuestas inválidas (opción inexistente)

- **Objeto de la prueba:** Validar la robustez del sistema y el manejo de errores cuando un usuario introduce una respuesta textual que no coincide con ninguna de las opciones predefinidas (`OPCION_UNICA`).
- **Entrada prefijada (`JuegoB4_ValidacionesDeOpcionesInvalidas.txt`):**
  - Se crea una encuesta con una pregunta de opción única: `Rojo|Verde|Azul`.
  - **Usuario 1** responde: "Rojo" (Opción válida).
  - **Usuario 2** responde: "Amarillo" (Opción inválida/inexistente).
  - **Usuario 3** responde: "Verde" (Opción válida).
- **Salida esperada:** El sistema no interrumpe la ejecución ni lanza excepciones. Al listar las respuestas finales, se observa:
  - Usuario 1: `0` (Índice de Rojo).
  - Usuario 2: `,` (Campo vacío).
  - Usuario 3: 1 (Índice de Verde).
- **Relación E/S:** La prueba demuestra que el mecanismo de asignación de respuestas es tolerante a fallos: al no encontrar la opción "Amarillo" en la lista definida, el sistema asigna un valor faltante en lugar de fallar. Esto prepara el terreno para las funcionalidades de imputación automática de datos probadas más adelante.

6.3. Persistencia (importar y exportar) (Grupo C)

Objetivo: Validar la conversión Objeto <-> CSV.

Juego C1: Importar y Exportar Mismo Encuesta

- **Objeto de la prueba:** Verificar la capacidad del sistema para importar datasets externos complejos (`loan-train.csv`), inferir automáticamente la estructura de las preguntas y volver a serializar los datos a disco sin pérdida de información.
- **Entrada prefijada (`JuegoC1_ImportarYExportarMismaEncuesta.txt`):**

1. Usuario inicia sesión, importa el fichero `../../csvFiles/loan-train.csv` asignándole el nombre "loan", y exporta inmediatamente dicha encuesta al fichero `../../csvFiles/expo-encuestas.csv`.

- **Salida esperada:** El sistema confirma la importación masiva mostrando las dimensiones exactas del dataset: "Importada encuesta loan con 13 preguntas y 614 respuestas".x Posteriormente confirma la exportación exitosa.
- **Relación E/S:**

  Inferencia de tipos: El `CsvSurveyImporter` analiza las columnas del CSV crudo. Detecta que columnas como "ApplicantIncome" son numéricas y "Education" son categóricas, creando las preguntas correspondientes en memoria.

Funcionalidad extra (formato ampliado): Al exportar, el `CsvSurveyExporter` recorre estas 614 respuestas. Si hubiera preguntas de tipo `OPCION_MULTIPLE` o configuraciones de límite, el exportador aplicaría el formato extendido (separador `|` y sufijo `<<MAX:n>>`), garantizando que el fichero de salida `expo-encuestas.csv` sea más rico semánticamente que un CSV estándar.

Juego C2: Exportar encuesta sin respuestas

- **Objeto de la prueba:** Verificar que el sistema permite persistir la estructura (metadatos) de un cuestionario aunque este no tenga datos de participación, actuando funcionalmente como un generador de plantillas vacías.
- **Entrada prefijada (`JuegoC2_ExportarEncuestaSinRespuestas.txt`):**
    1. Se crea manualmente una nueva encuesta (`encC4`).
    2. Se definen 3 preguntas de tipos variados para asegurar que la cabecera se forma bien: Numérica (`P1_id`), Texto (`P2_nombre`) y Opción Única (`P3_opcion`).
    3. Sin añadir ninguna respuesta, se accede inmediatamente al menú de Exportación y se guarda en `../../csvFiles/expo-encuestas.csv`.
- **Salida esperada:** El sistema informa explícitamente: "Exportada encuesta encC4 con 0 respuestas". No se producen errores en tiempo de ejecución.
- **Relación E/S:** El `CsvSurveyExporter` construye correctamente la cabecera del CSV (primera y segunda fila con enunciados y tipos) iterando sobre las preguntas definidas. Al detectar que la lista de respuestas tiene tamaño 0, finaliza el proceso de escritura limpiamente sin intentar acceder a datos inexistentes, generando un fichero CSV válido que contiene solo la definición del cuestionario.

6.4. Algoritmos de clustering básico (Grupo D)

Objetivo: Validar la lógica matemática (`KMeans`, `KMedoids`, `Gower`) y transformación de datos (`Vectorizer`).

Juego D1: KMeans (random) con dataset polarizado y tratamiento de nulos

- **Objeto de la prueba:** Verificar la capacidad del algoritmo KMeans para separar grupos claramente diferenciados y validar la funcionalidad extra de imputación automática de valores faltantes.
- **Entrada prefijada (`JuegoD1_ClusteringKMeansRandomConPocasRespuestas.txt`):**
    - Creación de encuesta con 3 variables numéricas (Edad, Ingresos, Experiencia).
    - 6 usuarios responden: 3 con perfil "Junior" (valores bajos: ~20 años, ~1000 ingresos) y 3 con perfil "Senior" (valores altos: ~50 años, ~5000 ingresos).
    - Ejecución de Clustering con k=2.

- **Salida esperada:** El sistema muestra métricas de calidad excelentes (Silhouette: 0.9658, indicando separación perfecta). Los representantes elegidos son p3 (grupo bajo) y p6 (grupo alto).
- **Relación E/S:**
    - **Imputación de datos:** Los mensajes DEBUG confirman que el Vectorizer ha detectado valores vacíos/nulos antes de procesar y los ha rellenado automáticamente utilizando la media de la columna (ej: 3100.0 para ingresos), permitiendo que el algoritmo matemático funcione sin interrupciones.
    - **Agrupación:** La gran distancia euclidiana entre los vectores "Junior" y "Senior" fuerza una convergencia rápida en dos clústeres puros.

Juego D2: Clustering con inicialización KMeans++

- **Objeto de la prueba:** Verificar la correcta integración de la variante KMeans++, diseñada para mejorar la selección inicial de centroides, y confirmar la consistencia con los resultados del estándar KMeans (D1) en datasets polarizados.
- **Entrada prefijada (JuegoD2_ClusteringKMeansPlusPlus.txt):**
    - Reutilización del escenario D1: Encuesta de perfilado laboral (Edad/Ingresos/Experiencia) con 6 respuestas divididas en dos grupos demográficos claros ("Junior" vs "Senior").
    - Se selecciona específicamente el algoritmo KMeans++ con k=2.
- **Salida esperada:** Métricas de calidad idénticas al caso anterior (Silhouette: 0.9658), confirmando que la inicialización optimizada converge correctamente a la solución global óptima.
- **Relación E/S:**
    - **Estabilidad del algoritmo:** KMeans++ selecciona probabilísticamente centroides iniciales distantes entre sí. En este dataset con grupos muy separados, esto garantiza evitar mínimos locales.
    - **Persistencia de limpieza de datos:** La salida vuelve a mostrar los mensajes de DEBUG, confirmando que el mecanismo de Imputación de Valores (Media) es transversal y se ejecuta automáticamente en la fase de vectorización, independientemente del algoritmo de clustering seleccionado posteriormente.

Juego D3: Clustering KMedoids (PAM) y representatividad

- **Objeto de la prueba:** Validar el algoritmo KMedoids (PAM), asegurando que los centros de los clústers sean elementos reales del dataset (y no promedios sintéticos), y confirmar la persistencia de la limpieza de datos.
- **Entrada prefijada (JuegoD3_ClusteringKMedoidsPAM.txt):**
    - Reutilización del dataset polarizado D1 ("Junior" vs "Senior").
    - Se selecciona el algoritmo K-Medoids (Opción 3) con k=2.
- **Salida esperada:** Métricas de calidad altas (Silhouette: 0.9658). El sistema identifica explícitamente a los representantes como medoid (dato real), seleccionando a p6@gmail.com y p3@gmail.com.
- **Relación E/S:**

- **Propiedad de medoides:** A diferencia de KMeans, la salida confirma que el algoritmo ha elegido respuestas existentes (p6 y p3) como centros, lo cual es vital para la interpretabilidad en datos cualitativos.
- **Robustez:** Los mensajes DEBUG reaparecen, demostrando que la capa de Imputación de Valores Faltantes (Media) es un módulo previo e independiente que sanea los datos antes de que cualquier algoritmo de clustering (sea KMeans o KMedoids) empiece a iterar.

Juego D4: Vectorización sin respuestas (control de error)

- **Objeto de la prueba:** Validar las precondiciones de los algoritmos de clustering, asegurando que el sistema detecta la ausencia de datos antes de intentar iniciar el proceso de vectorización.
- **Entrada prefijada (`JuegoD4_SinRespuestasNoSePuedeVectorizar.txt`):** Creación de una encuesta nueva con dos preguntas numéricas pero sin registrar ninguna respuesta de usuario. Acceso inmediato al menú de clustering e intento de ejecución de los tres algoritmos disponibles.
- **Salida esperada:** El sistema muestra un mensaje de advertencia explícito: "No hay respuestas. No se puede hacer clustering", impidiendo la ejecución del cálculo matemático y evitando excepciones en tiempo de ejecución.
- **Relación E/S:** El controlador de dominio consulta el repositorio antes de llamar al vectorizador. Al detectar que el tamaño de la lista de respuestas es 0, aborta la operación de forma controlada. Esto confirma que la lógica de negocio protege a la capa matemática de errores críticos como la división por cero o punteros nulos.

Juego D5: Vectorizar cuestionario con tipos mixtos

- **Objeto de la prueba:** Validar la capacidad del módulo `Vectorizer` y de la `GowerDistance` para procesar cuestionarios heterogéneos que combinan preguntas numéricas, textuales, de opción única y múltiple en la misma ejecución, sin lanzar errores de conversión de tipos.
- **Entrada prefijada (`JuegoD5_VectorizarCuestionarioConTiposMixtos.txt`):**
  1. Creación de una encuesta compleja con 4 tipos de preguntas: numérica (`P1_id`), texto libre (`P2_texto`), opción única (`P3_opcion`) y opción múltiple (`P4_multi`).
  2. Tres usuarios responden con combinaciones variadas de datos (ej: selecciones únicas distintas X, Y, Z y selecciones múltiples solapadas como A|B y B|C).
  3. Ejecución del algoritmo KMeans.
- **Salida esperada:** El sistema completa la ejecución mostrando los representantes y las métricas (SSE: 0.1543), confirmando que se ha podido construir la matriz de distancias correctamente.

- **Relación E/S:** La prueba confirma que la lógica del proyecto normaliza correctamente las variables numéricas y transforma las categóricas (opciones) en valores procesables para la distancia de Gower. Los mensajes de depuración (DEBUG) observados en la salida indican que la capa de preprocesamiento de datos y gestión de nulos está activa y revisa cada columna antes de pasar los datos al motor matemático.

6.5. Clustering avanzado (Grupo E)

Objetivo: Validar métricas de evaluación (`Silhouette`, `Accuracy`).

Juego E1: Persistencia y exportación de clustering enriquecida

- **Objeto de la prueba:** validar que el sistema mantiene en memoria el último análisis realizado y verificar la funcionalidad extra de exportación enriquecida, la cual genera un fichero CSV con metadatos del algoritmo, métricas de calidad y representantes reales calculados post-ejecución.
- **Entrada prefijada (JuegoE1_VerUltimoResultadoYExportarClustering.txt):**
  - Importación del dataset real loan-train.csv.
  - Ejecución del algoritmo KMeans con k=2.
  - Selección de la opción "Ver último resultado y exportar" hacia el fichero ../../csvFiles/expo-clustering.csv.
- **Salida esperada:** el sistema muestra una larga lista de mensajes DEBUG indicando la imputación de la media en columnas como 'Gender' o 'Married', debido a que no se han respondido. Finalmente, confirma: "Resultado exportado a ../../csvFiles/expo-clustering.csv".
- **Relación E/S:**
  - **Tratamiento de datos reales:** la salida confirma que el Vectorizer ha detectado múltiples celdas vacías en el dataset importado y las ha rellenado con la media/moda (representada como 0 o 1) para permitir el cálculo.
  - **Exportación avanzada:** el CsvClusterExporter no se limita a guardar pares (Usuario, Cluster). Al ser KMeans (que trabaja con centroides sintéticos/ficticios), el sistema calcula automáticamente qué usuario real (ej: imp_378) está más cerca de ese centroide usando la distancia de Gower y lo escribe en el CSV como representante, junto con el SSE y el Silhouette global.

Juego E2: Calcular accuracy con dataset externo (Loan Train)

- **Objeto de la prueba:** Validar la capacidad del sistema para realizar una evaluación externa de la calidad del agrupamiento, comparando los resultados generados por el algoritmo contra un conjunto de datos etiquetado (ground truth) para calcular la precisión (accuracy).
- **Entrada prefijada (JuegoE2_CalcularAccuracyLoanTrain.txt):**

---

1. Importación del dataset real loan-train.csv (que contiene la columna objetivo 'Loan_Status').
2. Ejecución del algoritmo kmeans con k=2.
3. Selección de la opción "calcular accuracy" proporcionando nuevamente la ruta del fichero loan-train.csv como referencia de la verdad.

- **Salida esperada:** Tras mostrar los mensajes de depuración por la imputación de nulos y las métricas internas, el sistema imprime el análisis de correspondencia: "Permutacion 1 -> accuracy = 1.0000".
- **Relación E/S:** El sistema cruza los identificadores de clúster generados con las etiquetas reales del fichero csv. Dado que el algoritmo no supervisado no conoce el significado semántico de "clúster 0" o "clúster 1", el módulo de evaluación genera todas las permutaciones posibles entre clústeres y etiquetas reales, reportando aquella asignación que maximiza el porcentaje de aciertos. El resultado de 1.0000 indica que, en esta ejecución, la agrupación coincidió perfectamente con las etiquetas proporcionadas.

Juego E3: Accuracy con K incorrecto que debe fallar

- **Objeto de la prueba:** Validar que el módulo de evaluación detecta inconsistencias dimensionales entre el resultado del clustering y el "Ground Truth" (etiquetas reales), impidiendo el cálculo de la precisión si el número de grupos no coincide.
- **Entrada prefijada (JuegoE3_AccuracyConKIncorrectoDebeFallar.txt):**
  - Importación del dataset loan-train.csv, el cual tiene una variable objetivo binaria (2 clases: Y/N).
  - Ejecución intencionada del algoritmo KMeans con k=3 (un valor incorrecto para este dataset).
  - Intento de calcular la Accuracy usando el mismo fichero CSV como referencia.
- **Salida esperada:** El clustering se ejecuta correctamente (calcula centroides y asigna usuarios), pero al intentar la evaluación, el sistema muestra: "Error calculando accuracy: Número de clusters distintos en el resultado (3) distinto al numero de clases... (2)".
- **Relación E/S:**
  - **Integridad matemática:** El cálculo de Accuracy en aprendizaje no supervisado se basa en encontrar la mejor permutación entre Clústers y Clases. Esto requiere que la cardinalidad sea idéntica ($k = N_{clases}$). El sistema detecta que $3 \neq 2$ y aborta la operación para evitar resultados erróneos o excepciones.
  - **Persistencia de limpieza:** Se observa nuevamente en los logs de DEBUG que el tratamiento de valores faltantes sigue funcionando correctamente antes de cualquier operación.

Juego E4: Comparativa de algoritmos (KMeans++ vs KMedoids) y exportación diferenciada

- **Objeto de la prueba:** Validar la capacidad del sistema para realizar un *benchmarking* o comparativa directa entre dos algoritmos de clustering (KMeans++ y

KMedoids) sobre el mismo conjunto de datos reales, observando las diferencias en métricas y en la selección de representantes, y verificando la exportación independiente de cada resultado.

- **Entrada prefijada (JuegoE4_ClusteringKMeansYKMedoidsComparacion.txt):**
  - Importación masiva de loan-train.csv.
  - Ejecución secuencial: primero KMeans++ (k=2) seguido de exportación inmediata a expo-clustering-kpp.csv.
  - Acto seguido, ejecución de KMedoids (k=2) sobre los mismos datos y exportación a expo-clustering-kmedoids.csv.
- **Salida esperada:** Se obtienen dos bloques de resultados distintos. KMeans++ presenta un SSE de 0.0798 y Silhouette de 0.1854, identificando representantes por proximidad al centroide. KMedoids presenta un SSE de 0.0505 y Silhouette de 0.1250, identificando explícitamente a los "medoids (dato real)".
- **Relación E/S:**
  - **Comparativa de rendimiento:** La prueba evidencia cómo cada algoritmo optimiza la agrupación de forma distinta. En este caso concreto con distancia de Gower, KMedoids logra un SSE menor (más compacto) que KMeans++, aunque este último obtiene mejor separación (Silhouette).
  - **Naturaleza del representante:** Se valida la distinción teórica en la salida: KMeans utiliza centroides sintéticos (promedios) y busca el vecino más cercano (imp_378), mientras que KMedoids selecciona obligatoriamente usuarios reales distintos (imp_96 vs imp_378), lo cual es crucial para la interpretabilidad en datos mixtos.

6.6. Comparación de ficheros (Grupo F)

Objetivo: Comprobar el comparador de ficheros.

Juego F1: Comparar Ficheros Iguales

- **Objeto de la prueba:** Verificar la corrección de la herramienta de utilidad interna para la comparación de ficheros, la cual es la base técnica del sistema de pruebas automatizadas para validar si el output coincide con el esperado.
- **Entrada prefijada (`JuegoF1_CompararFicherosIguales.txt`):**
  1. Acceso al menú de utilidades (Opción 5).
  2. Introducción de la misma ruta de fichero (`../../csvFiles/loan-train.csv`) tanto para el primer archivo como para el segundo.
- **Salida esperada:** El sistema procesa ambos flujos de datos y concluye con el mensaje explícito: "Los ficheros son IGUALES".
- **Relación E/S:** El algoritmo de comparación lee ambos ficheros simultáneamente. Al ser la misma ruta física, el contenido es bit a bit idéntico, por lo que el comparador

recorre todo el flujo sin encontrar discrepancias y valida la prueba positivamente. Esto confirma que el comparador de los tests no da falsos negativos.

Juego F2: Comparar ficheros con diferencias

- **Objeto de la prueba:** Verificar la sensibilidad y precisión de la herramienta de comparación de ficheros ante discrepancias en el contenido. Esto asegura que los tests automáticos sean capaces de reportar falsos positivos (detectar errores reales) y señalar la ubicación exacta del fallo.
- **Entrada prefijada (`JuegoF2_CompararFicherosConDiferencias.txt`):**
  - Acceso al menú de utilidades.
  - Introducción de dos rutas de ficheros que difieren ligeramente: `../../csvFiles/loan-test1.csv` y `../../csvFiles/loan-test2.csv`.
- **Salida esperada:** El sistema detecta la discrepancia y detalla el error: "Diferencia en línea 8", mostrando el contenido de ambos archivos en ese punto ("Aleks,24.0" vs "Aleks,26.0") y concluyendo con "Los ficheros son DIFERENTES".
- **Relación E/S:**
  - **Precisión del comparador:** El comparador lee los flujos de bytes línea a línea. Al llegar a la línea 8, la comparación de cadenas `String.equals()` devuelve falso debido al cambio en el valor numérico (24.0 vs 26.0).
  - **Utilidad de depuración:** La salida no solo indica el fallo binario, sino que imprime el contexto ("Archivo 1" vs "Archivo 2"), lo cual es la funcionalidad base que permite al desarrollador entender por qué ha fallado un test de prueba sin tener que abrir los ficheros manualmente.

6.7. Flujos integrados (Grupo G)

Objetivo: Pruebas de sistema completas con flujo de funcionalidades concatenadas

Juego G1: Ciclo de vida completo

- **Objeto de la prueba:** Validar el flujo principal de la aplicación integrando todos los subsistemas: gestión de usuarios, creación de encuestas, recogida de respuestas multi-usuario, ejecución de algoritmos de análisis y exportación final de datos y resultados.
- **Entrada prefijada (JuegoG1_FlujoCompleto_CrearEditarAnadir_ClusteringExportarComparar.txt):**

    - Un usuario administrador crea una encuesta manual (survey_manual) con preguntas heterogéneas (Numérica, Texto y Opción).
    - Simulación secuencial de 5 usuarios distintos (User1 a User5) que se registran, inician sesión y responden a la encuesta con perfiles variados.
    - El administrador vuelve a entrar, ejecuta KMeans++ con k=3 y exporta tanto el resultado del clustering como los datos brutos de la encuesta a ficheros CSV.
- **Salida esperada:** El sistema gestiona las sesiones correctamente acumulando 5 respuestas en la misma encuesta. El algoritmo se ejecuta con éxito mostrando métricas (SSE: 0.2361) y representantes, y finaliza confirmando la generación de los dos ficheros de exportación.
- **Relación E/S:**
    - **Integración de sistemas:** La prueba demuestra que los objetos creados en el módulo de Gestión persisten en memoria y son transformados correctamente por el módulo de Análisis (Vectorización) sin perder coherencia.
    - **Consistencia:** Verifica que las respuestas de diferentes instancias de Usuario se vinculan correctamente al mismo Cuestionario compartido.
    - **Persistencia:** Se valida que el sistema es capaz de "cerrar el ciclo", permitiendo guardar en disco tanto las encuestas y respuestas como el clustering y los representantes para su uso posterior.

Juego G2: Flujo de importar, añadir respuestas, detección de outliers (IQR) y accuracy

- **Objeto de la prueba:** Verificar la capacidad del sistema para trabajar con fuentes de datos híbridas (importación masiva + inserción manual) y validar dos funcionalidades críticas de calidad de datos: la imputación automática de valores faltantes y la detección y tratamiento de valores extremos (outliers) mediante rango intercuartílico (IQR).
- **Entrada prefijada (`JuegoG2_FlujoImportar_AnadirRespuestas_DosAlgoritmos_Accuracy.txt`):**
    - Importación del dataset `loan-train.csv` (614 registros).
    - Inserción manual de una respuesta adicional (`usertest@gmail.com`) introduciendo deliberadamente un valor numérico extremo en el campo de ingresos: `9988999889` (un valor anómalo muy superior al rango normal).
    - Ejecución secuencial de tres algoritmos (KMeans++, KMedoids, KMeans) solicitando métricas de accuracy.
- **Salida esperada:** El sistema integra la nueva respuesta (total 615). Se generan logs de `DEBUG` confirmando la limpieza de datos. Los algoritmos convergen con métricas de SSE controladas (ej: 0.0700) y se calcula la Accuracy final (59.45%).
- **Relación E/S:**

- **Robustez ante outliers:** El valor `9988999889` habría distorsionado totalmente los centroides y la distancia Euclidiana. El hecho de que el SSE sea bajo (0.07) confirma que el algoritmo de Detección por IQR interceptó este valor antes de la vectorización y lo sustituyó por un valor representativo (límite o media), protegiendo la integridad del análisis.
- **Gestión de nulos:** Los logs reafirman que, tanto para los datos importados como para los manuales, el sistema rellena huecos automáticamente (Media/Moda).
- **Evaluación supervisada:** Al final, el sistema logra comparar los clústers generados contra la columna etiqueta original, ofreciendo una métrica de precisión objetiva.

Juego G3: Aislamiento de datos entre múltiples encuestas

- **Objeto de la prueba:** Verificar la integridad y el aislamiento de datos en un entorno con múltiples encuestas activas simultáneamente. Se busca demostrar que las respuestas y los procesos de análisis (clustering) de una encuesta no interfieren ni se mezclan con los de otras, ya sean creadas manualmente o importadas.
- **Entrada prefijada (`JuegoG3_MultiplesEncuestasConDiferentesRespuestas.txt`):**
  - Creación de dos encuestas manuales vacías: `survey1` y `survey2`.
  - Tres usuarios distintos registran respuestas únicamente para `survey1`. `survey2` se deja intencionalmente vacía.
  - Importación de una tercera encuesta masiva: `loan-train`.
  - Intento secuencial de ejecutar KMeans sobre las tres encuestas.
- **Salida esperada:**
  - **survey1:** Ejecuta el clustering correctamente con 3 respuestas (SSE: 3.0000).
  - **loan-train:** Ejecuta el clustering correctamente con 614 respuestas (y sus correspondientes logs DEBUG de imputación de nulos).
  - **survey2:** Falla controladamente con el mensaje "No hay respuestas", a pesar de que en el sistema existen cientos de respuestas pertenecientes a las otras encuestas.
- **Relación E/S:**
  - **Encapsulamiento:** La prueba confirma que el `SurveyRepository` y los controladores mantienen listas de respuestas estrictamente separadas por ID de encuesta.
  - **Independencia de contexto:** El hecho de que `loan-train` funcione (con su limpieza de datos automática) mientras `survey2` da error en la misma sesión demuestra que el estado global del sistema no se corrompe al cambiar de un contexto de trabajo a otro.

7. Planificación

Durante todo el proceso hemos ido actualizando el diagrama de casos de uso, el diagrama del modelo conceptual y la documentación a medida que íbamos implementando los nuevos extras y las mejoras para la siguiente entrega.

Entrega 1 (17 de noviembre)

Semana 1 (17-23 de noviembre)

Durante esta semana revisamos la primera entrega mejorando los aspectos que nos dijo el profesor para antes de seguir con el proyecto tenerlo todo bien y seguir trabajando con una buena base. A parte empezamos a pensar en los extras que podríamos añadir en el trabajo para mejorarlo. Nos centramos especialmente en mejorar el diseño y organizar las tareas.

Semana 2 (24-30 de noviembre)

Seguimos trabajando en la corrección de errores de la Entrega 1 y adaptando los diagramas a los nuevos extras que queríamos añadir. Principalmente añadimos la implementación de las subclases de la clase Respuestas. También empezamos con la Capa de Presentación (la interfaz de Usuario). Para poder cubrir más escenarios que nos podíamos encontrar hicimos una ampliación de los juegos de prueba. Y teniendo en cuenta todas las modificaciones hicimos los cambios necesarios a los diagramas de Modelo y Casos de Uso.

Semana 3 (1-7 de diciembre)

Durante esta semana nos centramos en la integración de la arquitectura Modelo-Vista-Controlador. Implementamos los tres controladores principales y la lógica de comunicación entre capas, teniendo así una primera visión funcional de la interfaz gráfica y poder generar nuevos ejecutables. Paralelamente, se inició la implementación de los algoritmos correspondientes a los extras: el Sistema de Login y el Tratamiento de Valores Faltantes.

Semana 4 (8-14 de diciembre)

La última semana nos centramos en la finalización de los extras y también en la documentación final. Lo más destacado y lo que nos ocupó más tiempo fueron: la implementación completa de todos los extras, mejoras de usabilidad y correcciones visual en la interfaz del usuario, la creación y ejecución de tests unitarios para comprobar que todo sigue funcionando, la sincronización final de la documentación donde actualizamos definitivamente los diagramas de Casos de Uso y Modelo y redactar todo lo que nos faltaba.

Entrega 2 (15 de diciembre)

8. Conclusión

Llegar a este punto ha sido un viaje más largo y complicado de lo que esperábamos al principio. Cuando empezamos con la primera entrega, teníamos la estructura básica lista, pero cuando nos metimos en la segunda parte, nos dimos cuenta de que casi todo necesitaba ser repensado. Pasar de tener todo mezclado en la capa de dominio a organizar el código en tres capas fue brutalmente necesario, aunque nos costó más trabajo del que habríamos querido.

Lo que más nos ha sorprendido es cómo un cambio de arquitectura puede arreglarlo todo y romper todo a la vez. Decidimos usar el patrón MVC en serio, con controladores claros para presentación y persistencia, y aunque al principio fue frustrante reescribir funcionalidades, al final nos ahorró muchísimos dolores de cabeza cuando empezamos a meter extras. Las capas desacopladas significaban que cuando alguien necesitaba cambiar algo en persistencia, no afectaba a quién estaba metiendo mano en la interfaz gráfica.

La parte que más nos gustó fue implementar los extras. Empezar con el sistema de login parecía que iba a ser un fastidio, pero una vez lo metimos, los permisos de usuario y el aislamiento de datos casi salieron solos. El tratamiento de outliers usando el método IQR también fue satisfactorio por una razón tonta: veías a la aplicación detectar y arreglar datos raros automáticamente, y funcionaba bien. Los valores faltantes se imputaban con media o moda sin que el usuario tuviera que pensar en ello. Ese tipo de detalles hacen que se note que la aplicación está pensada de verdad, no que es un trabajo universitario chapucero.

El testing nos enseñó una lección importante: no es tan molesto hacer pruebas si lo haces mientras ibas desarrollando. Nuestros ejecutables que comparan el output esperado con el real nos salvaron en más de una ocasión cuando alguien tenía claro que su código estaba bien pero luego fallaba en el contexto general. Ver que hay tests que cubren desde crear una encuesta básica hasta comparar dos algoritmos con datos reales fue algo de lo que terminamos orgullosos.

Lo que no salió tan bien fue la comunicación interna al principio. Teníamos cinco personas tocando código y a veces nos pisábamos cambios. El diagrama de casos de uso y el diagrama de modelo terminaron siendo documentos vivos que estaban en constante cambio, así que por ahí fuimos aprendiendo a sincronizarnos mejor. Hacia el final, tener el repositorio bien organizado con directorios claros para tests, documentación y código fuente hizo toda la diferencia.

Si tuviéramos que hacerlo de nuevo (y ojalá no sea así), estaríamos más atentos con los tipos de datos desde el inicio. El cambiar ValorRespuesta de Object a una jerarquía polimórfica fue lo correcto, pero lo ideal hubiera sido pensarlo así desde el primer día en lugar de andar haciendo castings dudosos. Lo mismo con el VectorSchema; haberlo tenido mejor estructurado hubiera acelerado mucho la vectorización de datos.

En general, este proyecto nos ha enseñado que la ingeniería de software no es solo escribir código que funcione. Es pensar en cómo va a crecer, cómo van a trabajar otros en tu código, cómo vas a debuggear algo que no sabes dónde está roto. La cantidad de horas

que pasamos dibujando diagramas probablemente fue lo mejor invertido. Un diagrama bien hecho te ahorra horas de discusiones tontas sobre "¿dónde va esta clase exactamente?".

El resultado final es una aplicación que realmente funciona, que gestiona encuestas, que realiza clustering de manera robusta, y que aguanta datos reales sin desmoronarse. No es la aplicación más bonita del mundo y la interfaz gráfica es bastante funcional pero nada pretenciosa. Pero eso está bien. Hace su trabajo.

Nos vamos de aquí sabiendo un poco más sobre cómo las decisiones de arquitectura tempranas pueden mejorar o arruinar un proyecto. Sabiendo que los tests automatizados son nuestros amigos. Y sabiendo que la documentación, aunque sea aburrida de escribir, es lo que mantiene a una persona cuerda cuando vuelves al código tres semanas después y no recuerdas qué diablos estabas haciendo.

Ha sido un proyecto largo, a ratos tedioso, y a ratos bastante satisfactorio. Lo haríamos diferente la próxima vez. Pero no está nada mal para ser un trabajo de clase.