

SEMINARIO DE PRÁCTICA INFORMÁTICA

Trabajo Práctico N° 3

**UNIVERSIDAD
SIGLO**



Alumno: Marcos Romano - VINF013857

CATEDRA - E - INF275 - EDH – PR

Profesores: SERGIO DANIEL CONDE

HUGO FERNANDO FRIAS

Fecha de entrega: 28/10/2024

Contenido

OBJETIVO	3
CONSIGNA ABIERTA.....	3
RESOLUCIÓN PROBLEMÁTICA	5
1) Alcance y Justificación del Proyecto Informático.....	5
2) Justificación del Proyecto y Definición de Objetivos	5
3) Aplicación del Proceso Unificado de Desarrollo (PUD) para el Sistema de Registro y Gestión de Clientes	6
4) Diagrama de Dominio.....	8
Requerimientos del sistema	9
Requerimientos funcionales del sistema (RFS).....	9
Requerimientos no funcionales del sistema (RNF)	10
Diagrama de caso de uso	11
5) Etapa de análisis	17
Diagrama de secuencia.....	17
Diagrama de clases de diseño	19
Etapa de implementación	19
Diagrama de Despliegue	20
Plan de Pruebas del Sistema de Registro y Cruce de Datos con RePET	20
Base de datos para el prototipo.....	25
Creación de tablas	26
Trabajo Práctico 3.....	29
Enlace a GitHub	37
BIBLIOGRAFIA.....	38

OBJETIVO

Esta actividad busca que seas capaz de plantear una solución problemática que pueda resolverse mediante la realización de un proyecto informático.

Para llevar adelante este desafío, podrás aplicar muchos de los conceptos más importantes abordados durante el desarrollo del módulo 1, que recupera tu trabajo en materias troncales de la carrera.

Durante su desarrollo, lograrás aplicar tus conocimientos para alcanzar los siguientes objetivos:

- Definir el alcance y justificación de un proyecto informático para dar solución a una situación problemática.
- Realizar la justificación de un proyecto y la definición de objetivos.
- Aplicar el proceso unificado de desarrollo (PUD).
- Realizar el análisis del modelo de negocio.
- Plantear requerimientos funcionales y no funcionales.

En esta actividad, deberás seleccionar el problema a resolver, realizar el análisis del área problemática, definir el título y objetivos del proyecto, realizar la elicitación, plantear la propuesta de solución y comenzar con la etapa de análisis.

SITUACIÓN PROBLEMÁTICA

Lee con atención el siguiente caso teniendo presente cada uno de los contenidos que hemos desarrollado en el módulo. Una vez leído, tendrás que resolver preguntas cerradas en base al mismo. Haz clic sobre el siguiente enlace para descargar el enunciado:

Los sistemas informáticos desempeñan un papel muy importante para la optimización de procesos en diversas áreas de cualquier organización. La tecnología brinda la posibilidad de automatizar tareas, recopilar y analizar datos a gran escala, agilizar procesos y proporcionar soluciones a desafíos complejos.

Te propongo que determines con claridad un problema que pueda resolverse con la implementación de un proyecto informático y realices una entrega de acuerdo con lo solicitado en la consigna. Puedes definir cualquier organización real y explorar oportunidades en relación con su seguridad, logística, gestión de inventarios, optimización de procesos industriales, análisis de datos, cuidado de la salud, la educación, o cualquier otra área.

Para la realización de un proyecto de desarrollo informático, se requiere realizar un profundo análisis del negocio y comprender la dinámica de la organización. De esta manera, será posible comprender correctamente las necesidades, identificar los procesos, flujos de trabajo y desafíos que la organización enfrenta a diario y que pueden optimizarse con la aplicación de un desarrollo. Servirá también como base para justificar el proyecto, definir el alcance y objetivos.

Es clave llevar adelante un correcto proceso de elicitación, que involucra la definición de requerimientos funcionales y no funcionales.

Una vez completada esta fase, se avanza en el análisis y diseño detallado del sistema.

Esto implica la creación de modelos, la definición de la arquitectura, la planificación de la estructura de datos y la lógica de funcionamiento. Solo con un correcto análisis y diseño se puede garantizar que el sistema se construirá de manera eficiente y que cumplirá con los objetivos establecidos.

CONSIGNA ABIERTA

Considera que te encuentras trabajando en la organización cuyo problema definiste resolver con un proyecto informático, y te solicitan liderar el equipo de desarrollo que va a llevar adelante el mismo.

El proyecto informático que definas debe ser distinto al considerado en la lectura y el entregable final debe cumplir con los siguientes objetivos:

- Realizar el análisis del modelo de negocios para definir y justificar el proyecto.
- Aplicar el proceso unificado de desarrollo (PUD) para garantizar la calidad, escalabilidad y eficiencia en el ciclo de desarrollo.
- Utilizar una base de datos MySQL para la persistencia de los datos.
- Emplear Java como lenguaje de programación para el desarrollo del sistema.

A los fines del trabajo, para la implementación de la base de datos y el desarrollo con Java, puedes presentar un prototipo, que es “es la creación de un modelo operacional que incluya solo algunas características del sistema final” (Kendall & Kendall, 2011).

El concepto de operacional es clave, ya que no se trata de un simple modelo, sino que permite desarrollar módulos que se van integrando en la versión final del sistema.

Trabajo Práctico 2:

- Etapa de análisis.
 - Etapa de diseño.
 - Etapa de implementación.
 - Etapa de pruebas.
 - Definición de base de datos para el sistema.
 - Diagrama entidad-relación de la base de datos.
 - Creación de las tablas MySQL.
 - Inserción, consulta y borrado de registros.
 - Presentación de las consultas SQL.
 - Definiciones de comunicación.
-

RESOLUCIÓN PROBLEMÁTICA

1) Alcance y Justificación del Proyecto Informático

Justificación:

El objetivo principal de este proyecto es desarrollar un sistema informático capaz de registrar y gestionar datos de clientes, con la funcionalidad adicional de cruzar esta información con una base de datos de personas vinculadas a actividades terroristas. Esta solución es esencial para cumplir con los lineamientos de la Unidad de Información Financiera (UIF) de Argentina, que supervisa la prevención del lavado de dinero y la financiación del terrorismo.

La UIF exige que las instituciones financieras identifiquen y reporten operaciones sospechosas que puedan estar relacionadas con actividades ilícitas, incluyendo el terrorismo. El sistema propuesto facilitará la detección temprana de posibles riesgos, permitiendo un análisis exhaustivo y automatizado de los datos de los clientes en conformidad con las normativas vigentes.

Dado mi rol en Nación Bursátil S.A., he detectado la necesidad de implementar una solución que no solo cumpla con estos requisitos regulatorios, sino que también esté alineada con los objetivos estratégicos de la empresa. Este proyecto, aunque iniciado como una tarea académica, está diseñado para ser potencialmente implementado en mi organización, con el fin de mejorar los procesos internos y optimizar la gestión de riesgos.

Alcance: El proyecto se enfocará en el desarrollo de un sistema de registro de clientes que incluirá funcionalidades básicas para el almacenamiento y la consulta de datos personales. Además, el sistema incorporará una herramienta para cargar manualmente archivos JSON desde fuentes externas, permitiendo el cruce de esta información con listas oficiales de personas vinculadas al terrorismo. Esta funcionalidad de carga manual es una implementación inicial, que busca simplificar el desarrollo en esta primera fase del proyecto.

El diseño del sistema es adaptable a las necesidades específicas de Nación Bursátil S.A., garantizando que la solución no solo cumpla con las normativas regulatorias, sino que también aporte un valor tangible a la organización, mejorando la eficiencia en la gestión de datos y la identificación de riesgos.

2) Justificación del Proyecto y Definición de Objetivos

Justificación del Proyecto: La implementación de un sistema de registro de clientes con la capacidad de cruzar datos con listas de personas vinculadas al terrorismo es una necesidad crítica para asegurar el cumplimiento de las normativas establecidas por la Unidad de Información Financiera (UIF) de Argentina. La UIF exige que las entidades financieras identifiquen y reporten operaciones sospechosas relacionadas con el lavado de dinero y la financiación del terrorismo, y este proyecto tiene como finalidad garantizar que Nación Bursátil S.A. cumpla rigurosamente con estas obligaciones legales, protegiendo tanto a la empresa como a sus clientes. Además de responder a una necesidad legal, el desarrollo de este sistema ofrece una oportunidad estratégica para optimizar los procesos internos de gestión de datos dentro de la organización. Al automatizar el cruce de registros de clientes con listas oficiales de terroristas, se mejorará significativamente la eficiencia operativa y se mitigará el riesgo de incumplimiento de las regulaciones, reduciendo la exposición a sanciones y fortaleciendo la reputación de la empresa en el mercado.

Definición de Objetivos:

1. Cumplimiento Normativo:

- Desarrollar un sistema que permita a Nación Bursátil S.A. adherirse estrictamente a las normativas de la UIF, asegurando la identificación y el reporte oportuno de operaciones sospechosas relacionadas con el terrorismo y el lavado de dinero.

2. Eficiencia Operativa:

- Implementar una solución que automatice el cruce de datos entre la base de clientes de la empresa y las listas de personas identificadas como terroristas, mejorando la eficiencia y la precisión en la detección de riesgos potenciales.
3. **Seguridad y Escalabilidad:**
 - Asegurar que el sistema sea robusto, escalable y capaz de gestionar grandes volúmenes de datos, tanto en la administración de registros de clientes como en la consulta de archivos JSON cargados manualmente.
 4. **Adaptabilidad a las Necesidades de la Empresa:**
 - Diseñar un sistema que sea flexible y adaptable a las necesidades cambiantes de Nación Bursátil S.A., permitiendo futuras expansiones o modificaciones en función de nuevas normativas o políticas internas.
 5. **Facilidad de Uso y Mantenimiento:**
 - Desarrollar una interfaz de usuario intuitiva que facilite la carga manual de archivos JSON y la gestión de la base de datos, asegurando además que el sistema sea fácil de mantener y actualizar a lo largo del tiempo.
-

3) Aplicación del Proceso Unificado de Desarrollo (PUD) para el Sistema de Registro y Gestión de Clientes

1. Inicio

- **Objetivo:**
 - Definir el proyecto, establecer su alcance y evaluar su viabilidad.
- **Actividades:**
 - **Definición del Proyecto:**
 - **Visión del Proyecto:** Desarrollar un sistema informático para registrar y gestionar datos de clientes, con capacidad para cruzar esta información con listas de personas vinculadas al terrorismo.
 - **Alcance Inicial:** Incluir funcionalidades para el registro de clientes, carga manual de archivos JSON, y cruce de datos con listas de terroristas.
 - **Documentación:** Crear el documento de visión y alcance del proyecto.
 - **Estudio de Viabilidad:**
 - **Evaluación Técnica:** Revisar la viabilidad técnica del sistema utilizando Java y MySQL, y la integración con bases de datos JSON externas (**la idea inicial era consumir el JSON via API, pero RePET no dispone de dicho servicio**).
 - **Evaluación Económica:** Analizar el presupuesto necesario y los recursos disponibles para desarrollar el sistema.
 - **Evaluación Operativa:** Asegurar que el sistema se alinee con los procesos internos de Nación Bursátil S.A. y cumpla con los requisitos de la UIF.
 - **Planificación Inicial:**
 - **Plan de Proyecto:** Elaborar un cronograma, asignar recursos y elaborar un plan de comunicación.

2. Elaboración

- **Objetivo:**
 - Detallar los requisitos del sistema y diseñar su arquitectura.
- **Actividades:**

- **Recolección de Requisitos:**
 - **Requisitos Funcionales:** Documentar funcionalidades como el registro de clientes, cruce de datos con listas de terroristas, y carga de archivos JSON.
 - **Requisitos No Funcionales:** Incluir aspectos como seguridad, escalabilidad y usabilidad del sistema.
 - **Requisitos de Cumplimiento:** Asegurar que los requisitos cumplan con las normativas de la UIF.
- **Análisis del Modelo de Negocio:**
 - **Modelo de Nación Bursátil S.A.:** Definir cómo el sistema se integrará con el modelo de negocio y los procesos internos de la empresa.
 - **Documentación:** Crear un documento que describa cómo el sistema apoyará los objetivos y procesos de negocio de la empresa.
- **Diseño Arquitectónico:**
 - **Arquitectura del Sistema:** Diseñar la estructura del sistema, incluyendo la base de datos MySQL, la interfaz de usuario en Java, y la integración con las bases de datos JSON.
 - **Diagramas UML:** Crear diagramas de casos de uso, de clases y de secuencia para representar la arquitectura y el flujo del sistema.

3. Construcción

- **Objetivo:**
 - Desarrollar el sistema en iteraciones, asegurando calidad y funcionalidad.
- **Actividades:**
 - **Desarrollo Iterativo:**
 - **Codificación:** Desarrollar el código según el diseño arquitectónico y los requisitos documentados.
 - **Pruebas Unitarias:** Realizar pruebas unitarias para cada componente desarrollado.
 - **Pruebas:**
 - **Pruebas de Integración:** Verificar que los componentes del sistema funcionen juntos como un todo.
 - **Pruebas de Aceptación:** Validar que el sistema cumpla con los requisitos funcionales y no funcionales establecidos.
 - **Documentación de Pruebas:** Documentar los resultados de las pruebas y cualquier defecto encontrado.
 - **Documentación:**
 - **Documentación Técnica:** Actualizar la documentación técnica con detalles del desarrollo y la implementación.
 - **Manual de Usuario:** Crear y mantener manuales y guías para los usuarios finales.

4. Transición

- **Objetivo:**
 - Implementar el sistema en producción y asegurar una transición exitosa.
- **Actividades:**
 - **Despliegue:**
 - **Preparación del Entorno:** Configurar el entorno de producción, realizar la migración de datos y garantizar la infraestructura necesaria.

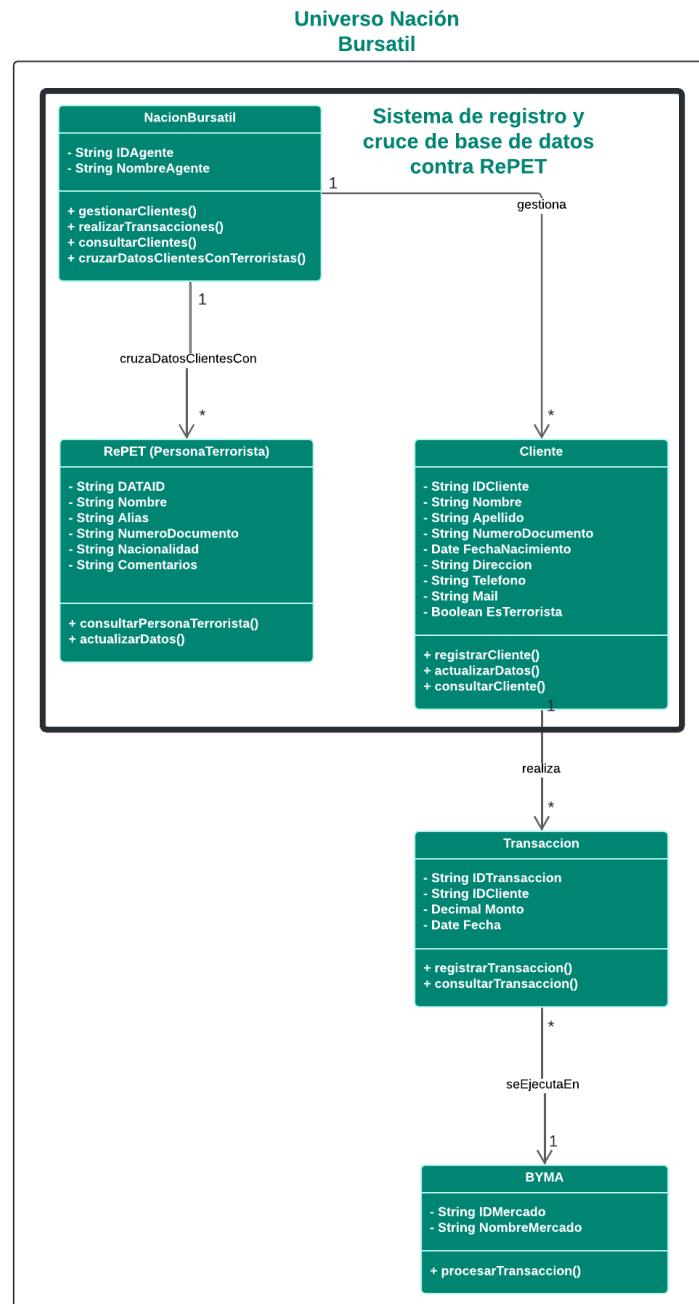
- **Implementación:** Desplegar el sistema y realizar pruebas finales en el entorno de producción para asegurar que todo funcione correctamente.
- **Capacitación:**
 - **Formación de Usuarios:** Capacitar a los usuarios finales y al personal de soporte con la información necesaria para usar el sistema.
 - **Materiales de Capacitación:** Desarrollar y distribuir materiales de capacitación y guías de usuario.
- **Soporte Post-Implementación:**
 - **Plan de Soporte:** Establecer un plan de soporte técnico para resolver problemas y atender consultas post-implementación.
 - **Monitoreo:** Monitorear el rendimiento del sistema y gestionar cualquier incidencia que pueda surgir.

5. Mantenimiento

- **Objetivo:**
 - Asegurar el funcionamiento continuo y la mejora del sistema a lo largo del tiempo.
- **Actividades:**
 - **Actualizaciones y Mejoras:**
 - **Implementación de Cambios:** Desarrollar e implementar actualizaciones y mejoras basadas en la retroalimentación de los usuarios y en cambios en las normativas.
 - **Corrección de Errores:** Identificar y corregir errores que se descubran después del despliegue.
 - **Monitoreo Continuo:**
 - **Evaluación del Rendimiento:** Monitorear el rendimiento y la seguridad del sistema para asegurar que sigue cumpliendo con los requisitos.
 - **Retroalimentación:** Recoger y analizar la retroalimentación de los usuarios para realizar ajustes y mejoras continuas.

4) Diagrama de Dominio

El diagrama de dominio ilustra la estructura y los componentes principales del sistema de registro y cruce de base de datos, proporcionando una visión general de los elementos involucrados y sus relaciones. Aunque el sistema se centra en el registro y cruce de datos con RePET, el diagrama incluye un universo un poco más amplio a modo de referencia, que muestra interacciones con BYMA a través de Nación Bursátil.



Requerimientos del sistema

Requerimientos funcionales del sistema (RFS)

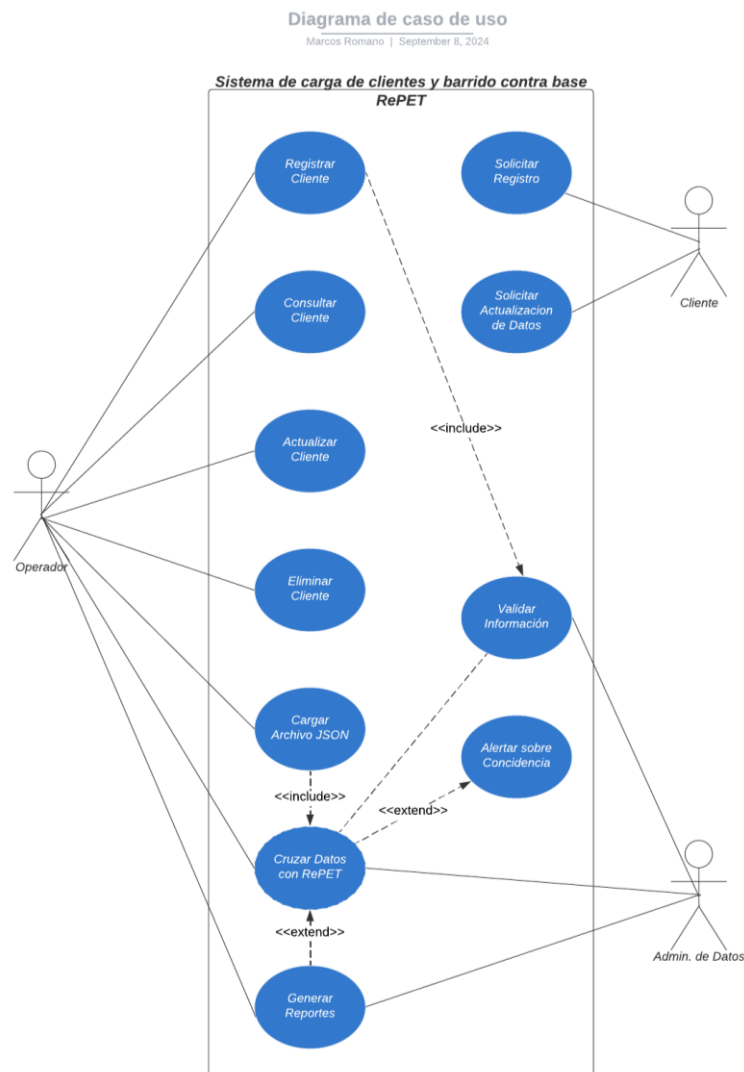
ID	Nombre	Descripción
RFS01	Registro de Clientes	Permitir el registro de nuevos clientes con sus datos personales, cumpliendo con las normativas vigentes.
RFS02	Edición de Datos de Clientes	Facilitar la modificación de los datos de clientes previamente registrados en el sistema.
RFS03	Eliminación de Clientes	Permitir la eliminación de clientes del sistema en caso de inactividad o necesidad operativa.
RFS04	Carga Manual de Archivos JSON	Permitir al operador cargar archivos JSON con información externa, como listas de personas vinculadas al terrorismo.

RFS05	Cruce de Datos con RePET	Realizar automáticamente el cruce de datos de los clientes con la base de datos RePET para verificar posibles coincidencias.
RFS06	Notificación de Alertas	Enviar notificaciones automáticas cuando se detecten coincidencias entre los datos del cliente y las listas de RePET.
RFS07	Consulta de Clientes	Permitir la búsqueda y consulta de clientes registrados para acceder a sus datos y al estado de la verificación con RePET.
RFS08	Seguridad y Roles de Usuario	Definir distintos niveles de acceso al sistema según los roles (administrador, operador, auditor) para proteger los datos sensibles.
RFS09	Informe de Resultados	Generar informes detallados de los resultados del cruce de datos con RePET, incluyendo alertas y acciones tomadas.
RFS10	Integración con la Base de Datos	Asegurar la correcta conexión e integración con la base de datos MySQL para almacenar y gestionar la información de clientes.
RFS11	Interfaz Intuitiva	Desarrollar una interfaz de usuario sencilla e intuitiva para facilitar la navegación y la carga de información por parte de los operadores.

Requerimientos no funcionales del sistema (RNF)

ID	Nombre	Descripción
RNF01	Compatibilidad con Windows	El sistema debe ser compatible con Windows Server 2019 y versiones posteriores, así como con Windows 10 y 11.
RNF02	Rendimiento	El sistema debe ser capaz de procesar el cruce de datos con la base de datos RePET en menos de 2 segundos por registro.
RNF03	Escalabilidad	El sistema debe ser capaz de manejar un incremento en el número de clientes y tamaño de archivos JSON sin necesidad de modificaciones significativas en su arquitectura.
RNF04	Seguridad de Datos	Los datos personales de los clientes deben almacenarse y transmitirse de manera segura, utilizando encriptación para garantizar la confidencialidad y evitar accesos no autorizados.
RNF05	Tolerancia a Fallos	El sistema debe garantizar la continuidad de las operaciones en caso de fallos menores (como pérdida de conexión a la base de datos), y permitir una recuperación rápida.
RNF06	Facilidad de Instalación	La instalación del sistema debe ser sencilla y contar con un instalador amigable que guíe al usuario durante todo el proceso en Windows.
RNF07	Documentación	El sistema debe contar con documentación clara y detallada para los usuarios finales y administradores del sistema, explicando el uso y la administración del software.
RNF08	Usabilidad	La interfaz del sistema debe ser intuitiva y fácil de usar, permitiendo a los operadores realizar sus tareas sin necesidad de entrenamiento intensivo.
RNF09	Tiempo de Respuesta	El tiempo de respuesta de las consultas y acciones del sistema no debe exceder los 3 segundos en condiciones normales de uso.
RNF10	Mantenimiento y Actualizaciones	El sistema debe ser fácil de mantener y permitir actualizaciones sin interrumpir las operaciones críticas de Nación Bursátil S.A.

Diagrama de caso de uso



Casos de Uso:

- **Registrar Cliente (CU001):**
 - Descripción: Permite al operador ingresar nuevos clientes al sistema. Este caso de uso se refiere al almacenamiento de la información personal y financiera de un nuevo cliente.
 - Actor Relacionado: Operador.
- **Consultar Cliente (CU002):**
 - Descripción: Permite al operador buscar y visualizar la información de un cliente previamente registrado en la base de datos.
 - Actor Relacionado: Operador.
- **Actualizar Cliente (CU003):**
 - Descripción: Permite al operador modificar la información de un cliente existente en la base de datos. Incluye tanto información personal como datos financieros.
 - Actor Relacionado: Operador.
- **Eliminar Cliente (CU004):**
 - Descripción: Permite al operador borrar la información de un cliente que ya no está activo o que fue registrado incorrectamente.
 - Actor Relacionado: Operador.

- **Cargar Archivo JSON (CU005):**
 - Descripción: Permite al operador subir archivos JSON con listas de personas vinculadas al terrorismo para su cruce con la base de datos de clientes.
 - Actor Relacionado: Operador.
- **Cruzar Datos con RePET (CU006):**
 - Descripción: Este caso de uso cruza los datos de los clientes registrados con las listas en el archivo JSON para identificar posibles vínculos con personas incluidas en RePET.
 - Actor Relacionado: Operador, Administrador de Datos.
- **Generar Reportes (CU007):**
 - Descripción: El sistema genera reportes con los resultados del cruce de datos, mostrando los clientes que podrían estar vinculados a actividades terroristas. Estos reportes pueden ser consultados tanto por el operador como por el administrador.
 - Actor Relacionado: Operador, Administrador de Datos.
- **Validar Información (CU008):**
 - Descripción: El administrador revisa y valida la información cruzada en el sistema, asegurándose de que los datos sean correctos y cumplan con los requisitos normativos.
 - Actor Relacionado: Administrador de Datos.
- **Solicitar Registro (CU009):**
 - Descripción: El Cliente solicita al Operador el registro de nuevos datos en el sistema. El Operador recibe la solicitud y procede a ingresar la información requerida en el sistema.
 - Actor Relacionado: Cliente.
- **Solicitar Actualización de Datos (CU010):**
 - Descripción: El Cliente solicita al Operador la actualización de datos existentes en el sistema. El Operador recibe la solicitud y realiza las modificaciones necesarias en la base de datos.
 - Actor Relacionado: Cliente.

Relación entre Actores y Casos de Uso:

- **Operador:** Realiza todas las operaciones básicas del sistema como registrar, consultar, actualizar y eliminar clientes, así como cargar archivos JSON, cruzar datos y generar reportes.
- **Administrador de Datos:** Supervisa las operaciones de cruce de datos y generación de reportes, además de validar la información para garantizar la exactitud.
- **Cliente:** Puede solicitar el registro de nuevos datos o la actualización de datos existentes. Estas solicitudes son canalizadas a través del Operador, quien realiza las acciones correspondientes en el sistema. El Cliente no interactúa directamente con el sistema, sino que utiliza al Operador para llevar a cabo estas operaciones.

Descripción de casos de uso

Campo	Descripción
Caso de uso	CU001 Registrar Cliente
Actores	Operador
Referencias	RFS01, RFS07
Descripción	Permite al Operador ingresar nuevos clientes al sistema. Este caso de uso se refiere al almacenamiento de la información personal y financiera de un nuevo cliente.
Precondición	El Operador ha iniciado sesión en el sistema. El Operador tiene los permisos necesarios para registrar nuevos clientes.

Flujo principal	1. El Operador selecciona la opción «Registrar Cliente» en la interfaz de usuario.
	2. El sistema muestra un formulario vacío para ingresar los detalles del cliente.
	3. El Operador completa el formulario con la información personal y financiera del nuevo cliente.
	4. El Operador confirma la entrada de datos.
	5. El sistema valida la información ingresada.
	6. El sistema agrega la nueva información del cliente a la base de datos.
	7. El sistema muestra un mensaje de confirmación al Operador.
Postcondición	Se ha registrado un nuevo cliente en el sistema con la información proporcionada por el Operador.
Flujo alternativo	Validación de datos con falla. El sistema muestra un mensaje de error. El sistema regresa al paso 2 para que el Operador corrija los errores.

Campo	Descripción
Caso de uso	CU002 Consultar Cliente
Actores	Operador
Referencias	RFS07
Descripción	Permite al Operador buscar y visualizar la información de un cliente previamente registrado en la base de datos.
Precondición	El Operador ha iniciado sesión en el sistema.
Flujo principal	1. El Operador selecciona la opción «Consultar Cliente» en la interfaz de usuario.
	2. El sistema muestra un formulario para ingresar los datos del cliente a consultar.
	3. El Operador ingresa los datos necesarios (ID de cliente, nombre, etc.).
	4. El sistema busca el cliente en la base de datos.
	5. El sistema muestra la información del cliente solicitado.
Postcondición	El sistema muestra la información detallada del cliente consultado.
Flujo alternativo	Datos no encontrados. El sistema muestra un mensaje indicando que el cliente no se encuentra en la base de datos.
Excepciones	No contempla.

Campo	Descripción
Caso de uso	CU003 Actualizar Cliente
Actores	Operador
Referencias	RFS02, RFS07, RFS10
Descripción	Permite al Operador modificar la información de un cliente existente en la base de datos. Incluye tanto información personal como datos financieros.
Precondición	El Operador ha iniciado sesión en el sistema.
	El Operador tiene los permisos necesarios para actualizar la información del cliente.
Flujo principal	1. El Operador selecciona la opción «Actualizar Cliente» en la interfaz de usuario.
	2. El sistema muestra un formulario con la información actual del cliente.
	3. El Operador modifica los datos según sea necesario.

	4. El Operador confirma la actualización.
	5. El sistema valida la información actualizada.
	6. El sistema guarda los cambios en la base de datos.
	7. El sistema muestra un mensaje de confirmación al Operador.

Campo	Descripción
Caso de uso	CU004 Eliminar Cliente
Actores	Operador
Referencias	RFS03, RFS07, RFS10
Descripción	Permite al Operador borrar la información de un cliente que ya no está activo o que fue registrado incorrectamente.
Precondición	El Operador ha iniciado sesión en el sistema.
	El Operador tiene los permisos necesarios para eliminar clientes.
Flujo principal	1. El Operador selecciona la opción «Eliminar Cliente» en la interfaz de usuario.
	2. El sistema muestra un formulario para ingresar el ID del cliente a eliminar.
	3. El Operador ingresa el ID del cliente.
	4. El sistema busca el cliente en la base de datos.
	5. El sistema elimina la información del cliente.
	6. El sistema muestra un mensaje de confirmación al Operador.
Postcondición	La información del cliente ha sido eliminada del sistema.

Campo	Descripción
Caso de uso	CU005 Cargar Archivo JSON
Actores	Operador
Referencias	RFS04, RFS10
Descripción	Permite al Operador subir archivos JSON con listas de personas vinculadas al terrorismo para su cruce con la base de datos de clientes.
Precondición	El Operador ha iniciado sesión en el sistema.
	El Operador tiene los permisos necesarios para cargar archivos.
Flujo principal	1. El Operador selecciona la opción «Cargar Archivo JSON» en la interfaz de usuario.
	2. El sistema muestra un formulario para seleccionar el archivo JSON.
	3. El Operador elige el archivo JSON y lo carga.
	4. El sistema valida el archivo cargado.
	5. El sistema guarda los datos del archivo en el sistema.
	6. El sistema muestra un mensaje de confirmación al Operador.
Postcondición	El archivo JSON ha sido cargado y los datos han sido almacenados en el sistema.

Campo	Descripción
Caso de uso	CU006 Cruzar Datos con RePET
Actores	Operador, Administrador de Datos
Referencias	RFS04, RFS05, RFS06, RFS07, RFS09, RFS10

Descripción	Este caso de uso cruza los datos de los clientes registrados con las listas en el archivo JSON para identificar posibles vínculos con personas incluidas en RePET.
Precondición	El archivo JSON ha sido cargado en el sistema.
	El Operador ha iniciado sesión en el sistema.
	El Operador tiene los permisos necesarios para cruzar datos.
Flujo principal	1. El Operador selecciona la opción «Cruzar Datos con RePET» en la interfaz de usuario.
	2. El sistema solicita los datos para el cruce.
	3. El Operador inicia el proceso de cruce de datos.
	4. El sistema realiza el cruce de datos.
	5. El sistema muestra los resultados del cruce.
	6. El sistema guarda los resultados para su revisión.

Campo	Descripción
Caso de uso	CU007 Generar Reportes
Actores	Operador, Administrador de Datos
Referencias	RFS09, RFS10
Descripción	El sistema genera reportes con los resultados del cruce de datos, mostrando los clientes que podrían estar vinculados a actividades terroristas. Estos reportes pueden ser consultados tanto por el Operador como por el Administrador.
Precondición	El cruce de datos con RePET ha sido completado.
	El Usuario ha iniciado sesión en el sistema.
Flujo principal	1. El Usuario selecciona la opción «Generar Reportes» en la interfaz de usuario.
	2. El sistema muestra las opciones de reporte disponibles.
	3. El Usuario selecciona el tipo de reporte deseado.
	4. El sistema genera el reporte.
	5. El sistema muestra el reporte al Usuario.
	6. El Usuario puede descargar o imprimir el reporte.
Postcondición	Se ha generado un reporte con los resultados del cruce de datos y está disponible para su consulta.

Campo	Descripción
Caso de uso	CU008 Validar Información
Actores	Administrador de Datos
Referencias	RFS09, RFS10
Descripción	El Administrador revisa y valida la información cruzada en el sistema, asegurándose de que los datos sean correctos y cumplan con los requisitos normativos.
Precondición	El cruce de datos con RePET ha sido completado.
	El Administrador de Datos ha iniciado sesión en el sistema.
Flujo principal	1. El Administrador de Datos selecciona la opción «Validar Información» en la interfaz de usuario.
	2. El sistema muestra los datos cruzados para su revisión.

	3. El Administrador revisa los datos y realiza las validaciones necesarias.
	4. El Administrador confirma la validez de los datos.
	5. El sistema actualiza el estado de los datos como validados.
	6. El sistema muestra un mensaje de confirmación al Administrador.
Postcondición	La información cruzada ha sido validada y el estado de los datos ha sido actualizado en el sistema.

Caso de uso	CU009 Solicitar Registro
Actores	Cliente, Operador
Referencias	RFS01, RFS10
Descripción	Permite al Cliente solicitar el registro de nuevos datos a través del Operador.
Precondición	El Cliente tiene acceso al sistema a través del Operador.
Flujo principal	1. El Cliente solicita al Operador el registro de nuevos datos. 2. El Operador recibe la solicitud y verifica la información proporcionada por el Cliente. 3. El Operador ingresa la información en el sistema. 4. El sistema valida la información ingresada por el Operador. 5. El sistema agrega la nueva información a la base de datos. 6. El sistema muestra un mensaje de confirmación al Operador. 7. El Operador informa al Cliente que el registro se ha completado con éxito.
Postcondición	Se ha registrado la nueva información en el sistema, y el Cliente ha sido informado de la operación.
Flujo alternativo	S4: Validación de datos con falla: El sistema muestra un mensaje de error. El Operador informa al Cliente sobre los errores y solicita la corrección de la información. El proceso se regresa al paso 2.
Excepciones	No contempla.

Campo	Descripción
Caso de uso	CU010 Solicitar Actualización
Actores	Cliente, Operador
Referencias	RFS02, RFS07, RFS10
Descripción	Permite al Cliente solicitar la actualización de datos existentes a través del Operador.
Precondición	El Cliente tiene acceso al sistema a través del Operador.
Flujo principal	1. El Cliente solicita al Operador la actualización de datos existentes. 2. El Operador recibe la solicitud y verifica la información proporcionada por el Cliente. 3. El Operador localiza los datos existentes en el sistema. 4. El Operador actualiza la información según las instrucciones del Cliente. 5. El sistema valida la información actualizada por el Operador. 6. El sistema guarda la información actualizada en la base de datos. 7. El sistema muestra un mensaje de confirmación al Operador. 8. El Operador informa al Cliente que la actualización se ha completado con éxito.

Postcondición	Los datos existentes han sido actualizados en el sistema, y el Cliente ha sido informado de la operación.
Flujo alternativo	Validación de datos con falla: El sistema muestra un mensaje de error. El Operador informa al Cliente sobre los errores y solicita la corrección de la información. El proceso se regresa al paso 3.

5) Etapa de análisis

Diagrama de secuencia

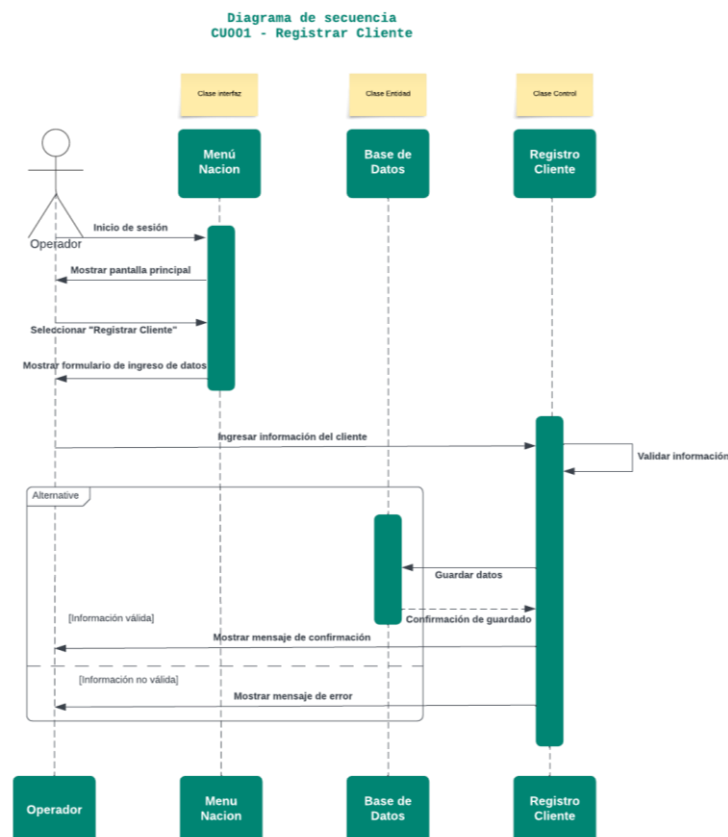
Caso de Uso: CU001 - Registrar Cliente

Descripción: En este diagrama de secuencia, se describe el proceso de registro de nuevos clientes en el sistema. El objetivo principal es ilustrar cómo un operador interactúa con el sistema para ingresar información personal de un cliente. El diagrama detalla cada paso del proceso, desde la selección de la opción de registro hasta la confirmación del almacenamiento exitoso de los datos.

Actores: Operador, Sistema

Secuencia de pasos:

1. El Operador inicia sesión en el sistema.
2. El Operador selecciona "Registrar Cliente".
3. El Sistema muestra un formulario para ingresar datos.
4. El Operador ingresa la información del cliente.
5. El Sistema valida la información.
6. Si es válida, el Sistema guarda los datos en la base de datos.
7. El Sistema muestra un mensaje de confirmación al Operador.



Caso de uso: CU006 - Cruzar Datos con RePET

Actores: Operador, Administrador de Datos

Descripción: Este diagrama de secuencia detalla el proceso mediante el cual el sistema cruza los datos de los clientes registrados en la base de datos interna con la información contenida en un archivo JSON proporcionado por RePET, que se carga de manera manual en el sistema. El flujo de trabajo describe cómo el operador carga el archivo, cómo el sistema procesa los datos del archivo y los cruza con la base de datos de clientes, generando alertas si se encuentran coincidencias con personas vinculadas a actividades terroristas. Esto asegura el cumplimiento normativo y facilita la gestión de riesgos.

Secuencia de pasos:

1. El Operador inicia sesión en el sistema.
2. El Operador selecciona la opción "Cargar Archivo JSON de RePET" en el Menú Nación.
3. El Sistema (Menú Nación) muestra un formulario para seleccionar el archivo JSON.
4. El Operador elige el archivo JSON desde su computadora y lo carga al sistema.
5. El Sistema (Controlador de Cruzamiento) valida el formato del archivo JSON (estructura correcta, datos esperados).
6. Si el archivo es válido, el Sistema guarda el archivo JSON y pasa a la etapa de cruce de datos.
7. El Operador selecciona la opción "Cruzar Datos con RePET" en el Menú Nación.
8. El Sistema (Controlador de Cruzamiento) toma los datos del archivo JSON cargado y los cruza con la base de datos interna de clientes.
9. El Sistema genera un reporte de resultados y muestra si hubo coincidencias entre los clientes y las personas en el archivo JSON de RePET.
10. Si se encuentran coincidencias, el Sistema genera alertas automáticas. El Operador revisa los resultados y las alertas generadas, tomando las acciones necesarias.

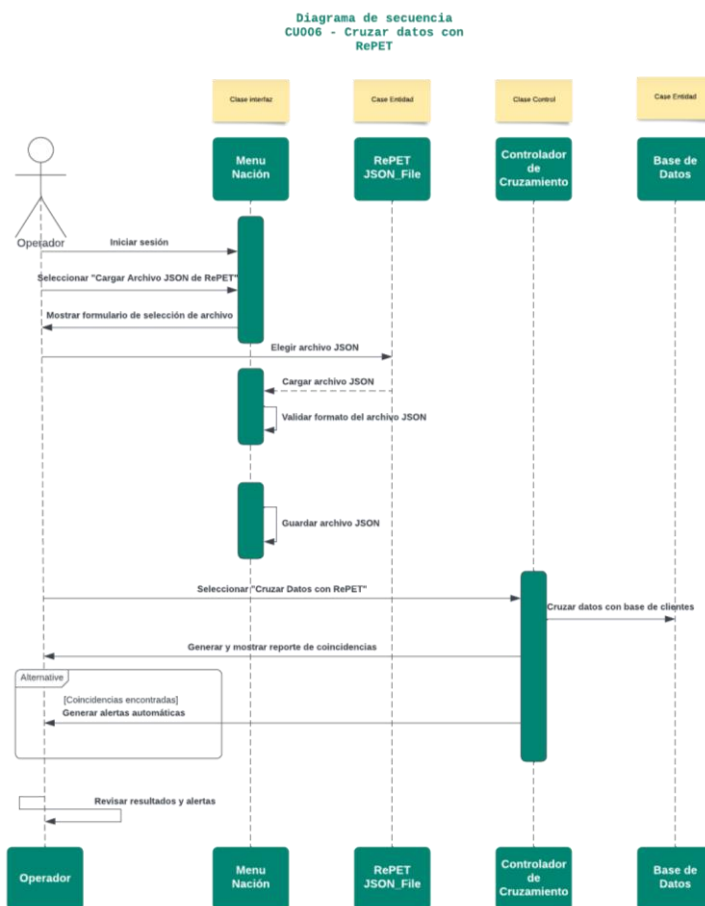
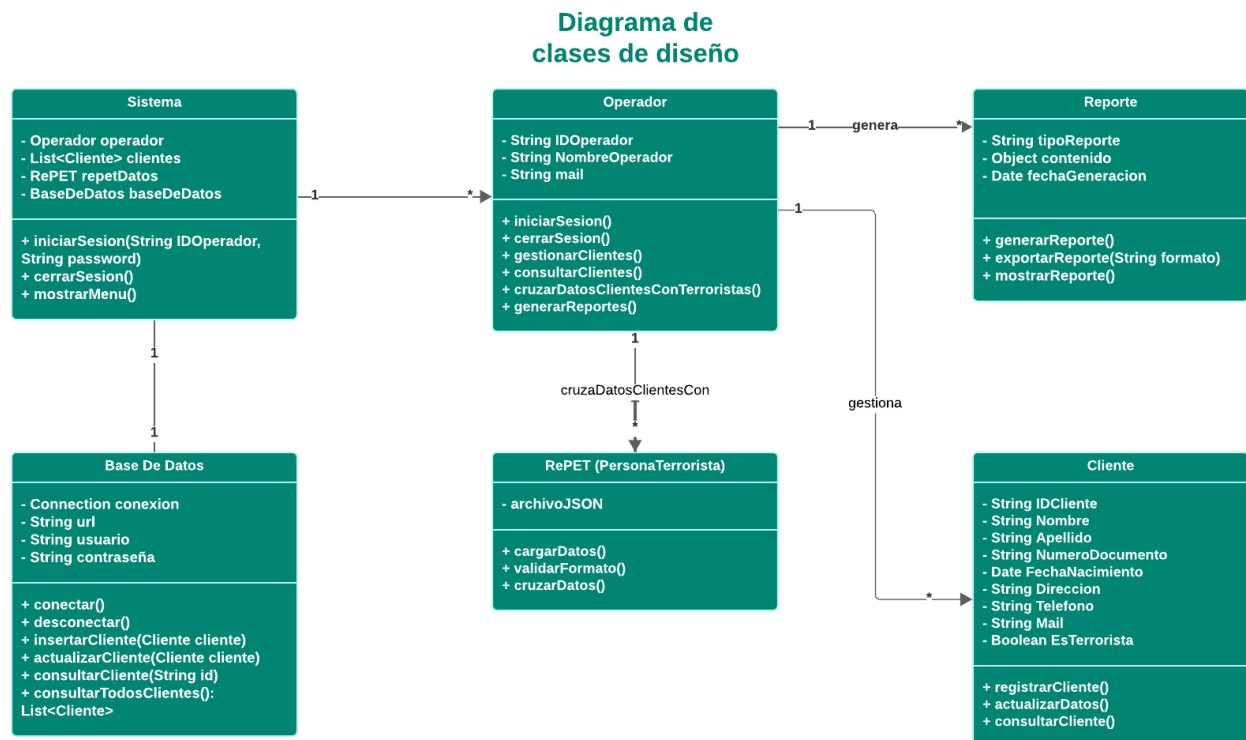


Diagrama de clases de diseño

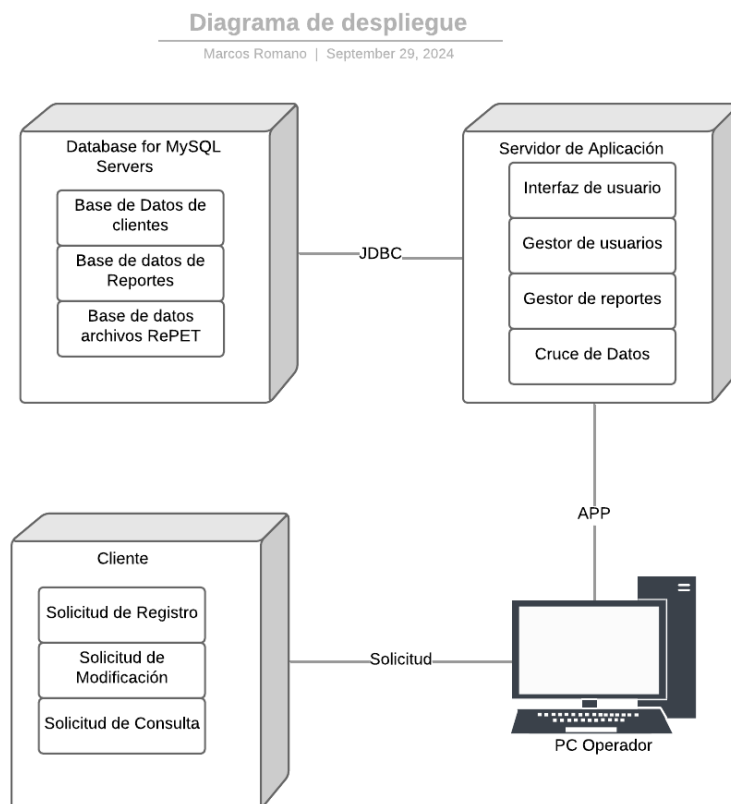


Etapa de implementación

Elemento	Descripción	Referencia RNF
Cliente (PC del Operador)	Dispositivo donde se ejecuta la aplicación de escritorio.	RNF01 - Compatibilidad con Windows
	- Debe ser compatible con Windows Server 2019, Windows 10 y 11.	RNF01 - Compatibilidad con Windows
	- La instalación debe ser sencilla y amigable.	RNF06 - Facilidad de Instalación
	- La interfaz debe ser intuitiva y fácil de usar.	RNF08 - Usabilidad
	- El tiempo de respuesta no debe exceder 3 segundos en uso normal.	RNF09 - Tiempo de Respuesta
Servidor de Base de Datos	Almacena la información del sistema.	RNF04 - Seguridad de Datos
	- Debe garantizar continuidad en caso de fallos menores.	RNF05 - Tolerancia a Fallos
	- Debe procesar el cruce de datos en menos de 2 segundos por registro.	RNF02 - Rendimiento
	- El sistema debe contar con una base de datos MySQL.	RNF04 - Seguridad de Datos

Servidor de Aplicaciones	- Debe manejar incrementos en el número de clientes y tamaño de archivos JSON.	RNF03 - Escalabilidad
Requerimiento General	- El sistema debe estar desarrollado en Java.	RNF11 - Desarrollo

Diagrama de Despliegue



Plan de Pruebas del Sistema de Registro y Cruce de Datos con RePET

Introducción

Este plan de pruebas describe el enfoque, el alcance y las pruebas a realizar para garantizar la calidad y correcto funcionamiento del sistema de registro y cruce de datos contra la base de datos RePET. El objetivo es verificar que el sistema cumpla con los requerimientos funcionales y no funcionales, y que su comportamiento sea estable y seguro bajo diferentes escenarios.

Objetivos de las Pruebas

Los objetivos principales de este plan de pruebas son:

- Validar que el sistema cumple con los requerimientos funcionales y no funcionales definidos.
- Verificar que las operaciones críticas, como el registro de clientes y el cruce de datos con RePET, se realizan correctamente.
- Garantizar que el sistema es estable, seguro y eficiente.
- Identificar defectos y asegurar su resolución antes del despliegue.

Alcance de las Pruebas

El plan de pruebas cubre las siguientes áreas:

- Funcionalidades principales del sistema:

- Registro de clientes.
- Validación de datos.
- Cruce de datos con la base RePET.
- Generación de reportes.
- Gestión de alertas.
- **Requerimientos no funcionales:**
 - **RNF02 - Rendimiento:** Tiempo de procesamiento y respuesta.
 - **RNF01 - Compatibilidad con Windows:** Compatibilidad con las plataformas indicadas (Windows y MySQL).

Tipos de Pruebas

Pruebas Funcionales

Validan que las funcionalidades del sistema se comporten según lo esperado.

- **Pruebas de Registro de Clientes:**
 - **RF001 - Registrar Cliente:** Verificar que se pueden registrar nuevos clientes con todos los datos requeridos.
 - **RF002 - Actualizar Cliente:** Validar la actualización de clientes ya existentes.
 - **RF003 - Consultar Cliente:** El sistema debe permitir la consulta de los datos de clientes registrados.
- **Pruebas de Cruce de Datos con RePET:**
 - Verificar que el sistema cruza correctamente los datos del cliente con el archivo JSON de RePET.
 - Comprobar la generación de alertas cuando se encuentran coincidencias.
 - Validar el formato de los reportes generados.

Pruebas de Rendimiento

Evalúan la eficiencia del sistema en términos de tiempos de procesamiento y respuesta.

- **Tiempo de Cruce de Datos con RePET:**
 - **RNF09 - Tiempo de Respuesta:** Validar que el cruce de datos no excede los 2 segundos por registro.
- **Tiempo de Respuesta del Sistema:**
 - **RNF09 - Tiempo de Respuesta:** Comprobar que las acciones del operador no superen los 3 segundos de respuesta en condiciones normales.

Pruebas de Seguridad

Evalúan que el sistema proteja adecuadamente la confidencialidad y la integridad de los datos.

- **Pruebas de Acceso No Autorizado:**
 - Intentar realizar acciones no permitidas para validar que el sistema protege los datos frente a usuarios no autorizados.

Pruebas de Escalabilidad

Verifican que el sistema pueda manejar un aumento en el número de clientes o en el tamaño de los archivos JSON sin afectar el rendimiento.

- **Carga Masiva de Clientes:**
 - Validar que el sistema puede manejar el registro masivo de clientes sin errores ni ralentizaciones.

Criterios de Aceptación

El sistema se considerará exitosamente probado si cumple con los siguientes criterios:

- Todas las pruebas funcionales han sido superadas sin defectos críticos.

- Los tiempos de procesamiento y respuesta cumplen con los límites establecidos (RNF02 y RNF09).
- El sistema ha demostrado ser seguro frente a accesos no autorizados y protege adecuadamente los datos personales.
- La arquitectura del sistema puede manejar un aumento significativo en la carga sin problemas de rendimiento.

Entorno de Pruebas

El entorno de pruebas será el siguiente:

- Plataforma: Windows Server 2019, Windows 10 y 11.
- Base de Datos: MySQL 8.x.

Roles y Responsabilidades

- **Equipo de Desarrollo:** Corregir los defectos reportados.
- **Equipo de Pruebas:** Ejecutar las pruebas, reportar los resultados y documentar los defectos.
- **Gerente de Proyecto:** Revisar y aprobar los resultados de las pruebas.

Comportamiento del sistema

Datos de Entrada	Comportamiento Esperado	Mensaje Emitido por el Sistema
Datos válidos de un nuevo cliente.	El sistema registra al cliente correctamente.	"Cliente registrado con éxito."
Datos inválidos (campo vacío).	El sistema rechaza el registro y no guarda datos.	"Error: Todos los campos son obligatorios."
Datos de un cliente existente.	El sistema actualiza los datos correctamente.	"Datos del cliente actualizados."
Datos inválidos durante la actualización.	El sistema rechaza la modificación y mantiene datos previos.	"Error: Datos incorrectos. Por favor, revise."
Archivo JSON de RePET cargado correctamente.	El sistema procede al cruce de datos con la base de datos.	"Archivo JSON cargado. Iniciando cruce de datos."
Coincidencias encontradas durante el cruce.	El sistema genera alertas sobre coincidencias.	"Se encontraron coincidencias. Revise las alertas."
Sin coincidencias encontradas durante el cruce.	El sistema notifica que no hay coincidencias.	"No se encontraron coincidencias."

Escenario de Pruebas

Campo	Descripción
Nombre del Módulo	Gestión de Clientes - Registro
ID de la Prueba	CP001
Objetivo	Verificar que un operador puede registrar un nuevo cliente con todos los datos requeridos.
Precondiciones	El operador debe estar autenticado en el sistema. El sistema debe estar en funcionamiento y accesible.
Datos de Entrada	<ul style="list-style-type: none"> - Nombre: Juan - Apellido: Pérez - Documento: 12345678 - Fecha de Nacimiento: 1980-01-01 - Dirección: Calle Falsa 123

			- Teléfono: 1112345678 - Email: juan.perez@mail.com			
Pasos			1. El operador inicia sesión en el sistema. 2. El operador selecciona la opción "Registrar Cliente". 3. El sistema muestra un formulario para ingresar datos. 4. El operador ingresa la información del cliente. 5. El operador hace clic en "Guardar".			
Resultado Esperado			- El sistema valida la información ingresada. - Los datos se almacenan correctamente en la base de datos. - El sistema muestra un mensaje de confirmación: "Cliente registrado exitosamente." - El nuevo cliente aparece en la lista de clientes.			
Prueba	Fecha	Acción	Datos Ingresados	Datos Obtenidos	Resultado	Estado
1	12/09	Registro de Cliente.	El operador intenta registrar un nuevo cliente.	El sistema guarda el cliente exitosamente.	El sistema registra el cliente.	Aceptado.
2	14/09	Registro de Cliente.	El operador intenta registrar un cliente con datos incompletos	El sistema muestra un error de validación.	El sistema rechaza el registro.	Rechazado.
3	15/09	Registro de Cliente.	El operador intenta registrar un nuevo cliente.	Los datos se actualizan correctamente.	El sistema registra el cliente.	Aceptado.
Ejecutante de pruebas		OP001: Marcos Romano				
Observaciones generales		El sistema de registro de clientes ha demostrado ser eficiente y fácil de usar para los operadores. La interfaz es intuitiva, lo que permite a los usuarios realizar tareas sin necesidad de un entrenamiento extenso. La validación de datos se ejecuta correctamente, impidiendo la entrada de información incompleta o incorrecta. En general, el sistema cumple con los requisitos establecidos y proporciona un buen rendimiento en las operaciones de registro.				

Campo	Descripción
Nombre del Módulo	Gestión de Clientes - Registro y RePET
ID de la Prueba	CP002

Objetivo		Verificar que el sistema impide registrar un cliente que ya esté presente en la base de datos RePET.				
Precondiciones		<ul style="list-style-type: none"> - El operador debe estar autenticado en el sistema. - El sistema debe estar en funcionamiento y accesible. - El cliente debe existir previamente en la base de datos RePET. 				
Datos de Entrada		<ul style="list-style-type: none"> - Formulario de alta de cliente. <p>Archivo JSON con información de personas en la lista RePET, que incluye:</p> <ul style="list-style-type: none"> - Nombres: Aamir Ali Chaudhry - Documento: BN 4196361 - Nacionalidad: Pakistan 				
Pasos		<ol style="list-style-type: none"> 1. El operador inicia sesión en el sistema. 2. El operador selecciona la opción "Registrar Cliente". 3. El sistema muestra un formulario para ingresar datos. 4. El operador ingresa la información del cliente. 5. El sistema verifica automáticamente si el cliente ya está en la base RePET. 6. El operador hace clic en "Guardar". 				
Resultado Esperado		<ul style="list-style-type: none"> - El sistema detecta que el cliente está registrado en la base RePET y no permite el registro. - Se muestra un mensaje de advertencia: "El cliente está listado en la base RePET. No puede ser registrado." 				
Prueba	Fecha	Acción	Datos Ingresados	Datos Obtenidos	Resultado	Estado
1	19/09	Registro de Cliente.	El operador intenta registrar un cliente.	El sistema muestra un error de validación.	El sistema rechaza el registro.	Rechazado.
2	19/09	Registro de Cliente.	El operador intenta registrar un cliente.	El sistema muestra un error de validación.	El sistema rechaza el registro.	Rechazado.
Ejecutante de pruebas		OP001: Marcos Romano				
Observaciones generales		El sistema ha demostrado ser efectivo al cruzar datos con RePET, evitando el registro de individuos listados. Las validaciones de datos y el cruce con RePET se realizaron de manera eficiente, cumpliendo con los requisitos de seguridad y prevención de riesgos. El operador recibe la notificación adecuada al intentar registrar un cliente presente en la base RePET, lo que garantiza la protección contra actividades potencialmente ilícitas.				

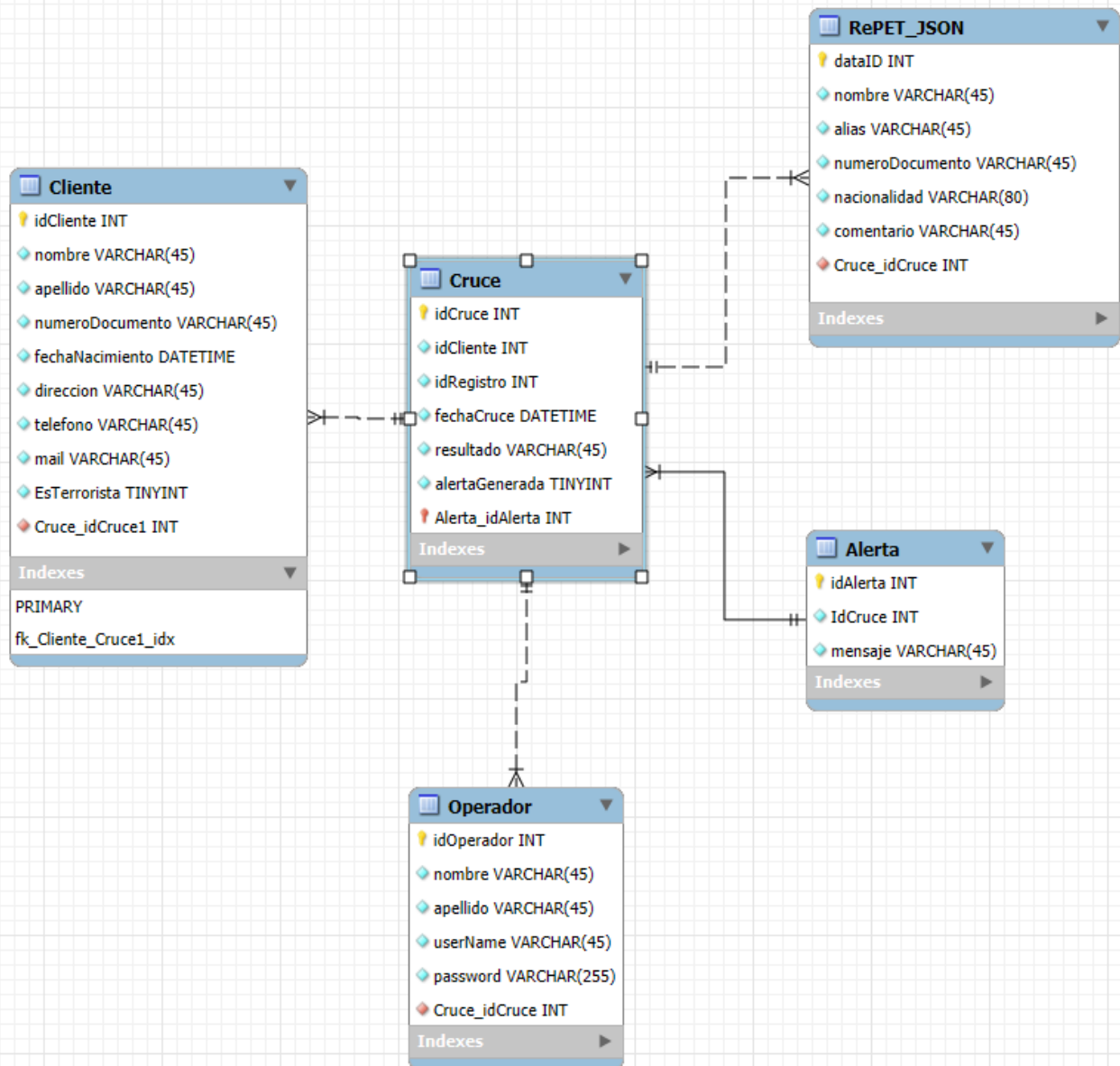
Campo	Descripción
Nombre del Módulo	Gestión de clientes – Cruce con RePET
ID de la Prueba	CP003

Objetivo			Verificar que el operador puede realizar un cruce de datos cargando un archivo JSON contra la base de datos de clientes y que el sistema detecta posibles coincidencias con la base RePET.			
Precondiciones			<ul style="list-style-type: none"> - El operador debe estar autenticado en el sistema. - El sistema debe estar en funcionamiento y accesible. - El archivo JSON debe estar disponible para ser cargado. 			
Datos de Entrada			Archivo JSON con información de personas en la lista RePET.			
Pasos			<ol style="list-style-type: none"> 1. El operador inicia sesión en el sistema. 2. El operador selecciona la opción "Cargar JSON RePET". 3. El sistema muestra un formulario para subir el archivo JSON. 4. El operador carga el archivo JSON. 5. El sistema analiza los datos del JSON y los cruza contra la base de clientes. 6. El sistema genera un reporte con los resultados del cruce. 			
Resultado Esperado			<ul style="list-style-type: none"> - El sistema identifica coincidencias entre los clientes y los datos de RePET. - Si hay coincidencias, se generan alertas automáticas y se marca el cliente como "Posible Terrorista" en la base de datos. - El sistema muestra un mensaje de confirmación: "Cruce realizado exitosamente. Coincidencias detectadas." 			
Prueba	Fecha	Acción	Datos Ingresados	Datos Obtenidos	Resultado	Estado
1	30/09	Cruce de archivo RePET contra base de datos.	El operador realiza la carga de un archivo RePET.	El sistema muestra un error de validación.	Archivo con formato invalido.	Rechazado.
2	30/09	Cruce de archivo RePET contra base de datos.	El operador realiza la carga de un archivo RePET.	El sistema muestra un mensaje de confirmación de carga.	El sistema encuentra coincidencia y realiza la alerta correspondiente.	Aceptado.
Ejecutante de pruebas		OP001: Marcos Romano				
Observaciones generales		El sistema ha mostrado un comportamiento adecuado al realizar el cruce de datos contra la base RePET. La detección de coincidencias y el marcado de alertas se llevaron a cabo con éxito, y los mensajes de confirmación proporcionaron una clara retroalimentación al operador. En el caso de errores en el archivo JSON, el sistema fue capaz de gestionar correctamente la situación mostrando los errores correspondientes.				

Base de datos para el prototipo

El sistema de gestión de clientes utilizará una base de datos relacional en MySQL, lo que resulta adecuado para el manejo de datos interrelacionados y transacciones complejas. Las bases de datos relacionales permiten

mantener la integridad referencial y asegurar la consistencia de los datos, lo cual es crucial para la gestión de clientes y cruces de información. MySQL, reconocido por su rendimiento y escalabilidad, satisface los requisitos no funcionales del sistema, garantizando una solución eficaz que puede crecer con la demanda y manejar grandes volúmenes de datos de manera confiable.



Creación de tablas

Base de datos “seminario”

```
-- Schema seminario
```

```
CREATE SCHEMA IF NOT EXISTS `seminario` DEFAULT CHARACTER SET utf8mb3 ;  
USE `seminario` ;
```

Tabla “cliente”

```
50 -- Table `seminario`.`Cliente`
51 -----
52 CREATE TABLE IF NOT EXISTS `seminario`.`Cliente` (
53   `idCliente` INT NOT NULL AUTO_INCREMENT,
54   `nombre` VARCHAR(45) NOT NULL,
55   `apellido` VARCHAR(45) NOT NULL,
56   `numeroDocumento` VARCHAR(45) NOT NULL,
57   `fechaNacimiento` DATETIME NOT NULL,
58   `direccion` VARCHAR(45) NOT NULL,
59   `telefono` VARCHAR(45) NOT NULL,
60   `mail` VARCHAR(45) NOT NULL,
61   `EsTerrorista` TINYINT NOT NULL,
62   `Cruce_idCruce1` INT NOT NULL,
63   PRIMARY KEY (`idCliente`),
64   INDEX `fk_Cliente_Cruce1_idx` (`Cruce_idCruce1` ASC) VISIBLE,
65   CONSTRAINT `fk_Cliente_Cruce1`
66     FOREIGN KEY (`Cruce_idCruce1`)
67       REFERENCES `seminario`.`Cruce` (`idCruce`)
68     ON DELETE NO ACTION
69     ON UPDATE NO ACTION)
70 ENGINE = InnoDB;
```

Tabla “alerta”

```
20 CREATE TABLE IF NOT EXISTS `seminario`.`Alerta` (
21   `idAlerta` INT NOT NULL AUTO_INCREMENT,
22   `IdCruce` INT NOT NULL,
23   `mensaje` VARCHAR(45) NOT NULL,
24   PRIMARY KEY (`idAlerta`))
25 ENGINE = InnoDB;
```

Tabla “cruce”

```
29 -- Table `seminario`.`Cruce`
30 -----
31 CREATE TABLE IF NOT EXISTS `seminario`.`Cruce` (
32   `idCruce` INT NOT NULL AUTO_INCREMENT,
33   `idCliente` INT NOT NULL,
34   `idRegistro` INT NOT NULL,
35   `fechaCruce` DATETIME NOT NULL,
36   `resultado` VARCHAR(45) NOT NULL,
37   `alertaGenerada` TINYINT NOT NULL,
38   `Alerta_idAlerta` INT NOT NULL,
39   PRIMARY KEY (`idCruce`, `Alerta_idAlerta`),
40   INDEX `fk_Cruce_Alerta1_idx` (`Alerta_idAlerta` ASC) VISIBLE,
41   CONSTRAINT `fk_Cruce_Alerta1`
42     FOREIGN KEY (`Alerta_idAlerta`)
43       REFERENCES `seminario`.`Alerta` (`idAlerta`)
44     ON DELETE NO ACTION
45     ON UPDATE NO ACTION)
46 ENGINE = InnoDB;
```

Tabla “operador”

```
95 -- Table `seminario`.`Operador`
96 -----
97 CREATE TABLE IF NOT EXISTS `seminario`.`Operador` (
98   `idOperador` INT NOT NULL AUTO_INCREMENT,
99   `nombre` VARCHAR(45) NOT NULL,
100   `apellido` VARCHAR(45) NOT NULL,
101   `userName` VARCHAR(45) NOT NULL,
102   `password` VARCHAR(255) NOT NULL,
103   `Cruce_idCruce` INT NOT NULL,
104   PRIMARY KEY (`idOperador`),
105   INDEX `fk_Operador_Cruce1_idx` (`Cruce_idCruce` ASC) VISIBLE,
106   CONSTRAINT `fk_Operador_Cruce1`
107     FOREIGN KEY (`Cruce_idCruce`)
108       REFERENCES `seminario`.`Cruce` (`idCruce`)
109     ON DELETE NO ACTION
110     ON UPDATE NO ACTION)
111 ENGINE = InnoDB;
```

Tabla “repet_json”

```
74 -- Table `seminario`.`RePET_JSON`
75 -----
76 CREATE TABLE IF NOT EXISTS `seminario`.`RePET_JSON` (
77   `dataID` INT NOT NULL AUTO_INCREMENT,
78   `nombre` VARCHAR(45) NOT NULL,
79   `alias` VARCHAR(45) NOT NULL,
80   `numeroDocumento` VARCHAR(45) NOT NULL,
81   `nacionalidad` VARCHAR(80) NOT NULL,
82   `comentario` VARCHAR(45) NOT NULL,
83   `Cruce_idCruce` INT NOT NULL,
84   PRIMARY KEY (`dataID`),
85   INDEX `fk_RePET_JSON_Cruce1_idx` (`Cruce_idCruce` ASC) VISIBLE,
86   CONSTRAINT `fk_RePET_JSON_Cruce1`
87     FOREIGN KEY (`Cruce_idCruce`)
88       REFERENCES `seminario`.`Cruce` (`idCruce`)
89     ON DELETE NO ACTION
90     ON UPDATE NO ACTION)
91 ENGINE = InnoDB;
```

Operaciones de carga de datos

Carga de un operador

```
1 USE `seminario`;
2
3 • INSERT INTO `Operador` (`nombre`, `apellido`, `userName`, `password`, `Cruce_idCruce`)
4   VALUES
5   ('Marcos', 'Romano', 'marcos.romano', '1234', 1);
6
```

Carga de clientes

```
1 • USE `seminario`;
2 • INSERT INTO `Cliente` (`nombre`, `apellido`, `numeroDocumento`, `fechaNacimiento`, `direccion`, `telefono`, `mail`, `EsTerrorista`, `Cruce_idCruce1`) VALUES
3   ('Juan', 'Pérez', '12345678', '1985-06-15', 'Av. Siempre Viva 742', '011-5555-5555', 'juan.perez@gmail.com', 0, 1),
4   ('María', 'Gómez', '23456789', '1990-01-22', 'Calle Falsa 123', '011-6666-6666', 'maria.gomez@hotmail.com', 1, 2),
5   ('Carlos', 'López', '34567890', '1988-03-11', 'Av. Libertador 456', '011-7777-7777', 'carlos.lopez@yahoo.com.ar', 0, 3),
6   ('Ana', 'Martínez', '45678901', '1995-05-03', 'Calle 7 890', '011-8888-8888', 'ana.martinez@gmail.com', 0, 4),
7   ('Pedro', 'Rodríguez', '56789012', '1980-09-25', 'Av. 9 de Julio 123', '011-9999-9999', 'pedro.rodriguez@hotmail.com', 1, 5),
8   ('Lucía', 'Fernández', '67890123', '1992-02-14', 'Calle Nueva 234', '011-4444-4444', 'lucia.fernandez@yahoo.com.ar', 0, 1),
9   ('Fernando', 'Hernández', '78901234', '1987-08-19', 'Av. Rivadavia 321', '011-3333-3333', 'fernando.hernandez@gmail.com', 1, 2),
10  ('Sofía', 'González', '89012345', '1991-07-30', 'Calle San Martín 654', '011-2222-2222', 'sofia.gonzalez@hotmail.com', 0, 3),
11  ('Javier', 'Ramírez', '90123456', '1993-10-12', 'Calle 11 987', '011-1111-1111', 'javier.ramirez@yahoo.com.ar', 0, 4),
12  ('Isabel', 'Castro', '10234567', '1986-04-04', 'Av. Corrientes 852', '011-5555-5556', 'isabel.castro@gmail.com', 1, 5),
13  ('Mateo', 'Torres', '21345678', '1990-12-29', 'Calle Lima 369', '011-6666-6667', 'mateo.torres@hotmail.com', 0, 1),
14  ('Valentina', 'Salazar', '32456789', '1994-11-18', 'Av. San Juan 789', '011-7777-7778', 'valentina.salazar@yahoo.com.ar', 0, 2),
```

Insertar un cliente terrorista a la base de datos

```
1 USE `seminario`;
2
3 • INSERT INTO `Cliente` (`nombre`, `apellido`, `numeroDocumento`, `fechaNacimiento`, `direccion`, `telefono`, `mail`, `EsTerrorista`, `Cruce_idCruce1`)
4   VALUES
5   ('AAMIR', 'ALI CHAUDHRY', 'BN 4196361', '1985-10-18', 'Calle Falsa 123', '123456789', 'aamir.ali@hotmail.com', 1, 1);
6
```

Eliminar un cliente

```
1 DELETE FROM Cliente
2 WHERE nombre = 'AAMIR'
3 AND apellido = 'ALI CHAUDHRY'
4 AND numeroDocumento = 'BN 4196361';
```

Action Output			
#	Time	Action	
✓ 8	19:44:29	DELETE FROM Cliente WHERE nombre = 'AAMIR' AND apellido = 'ALI CHAUDHRY' AND numeroDocumento = 'BN 4196361'	
✓ 9	19:44:32	SELECT * FROM `seminario`.`cliente` LIMIT 0, 200	

Se agrega la siguiente restricción para impedir que se agreguen usuarios duplicados en la base de datos.

```
1 • ALTER TABLE Cliente
2   ADD CONSTRAINT unique_numeroDocumento UNIQUE (numeroDocumento);
```

Action Output			
#	Time	Action	Message
✓ 16	19:49:16	USE `seminario`	0 row(s) affected
✗ 17	19:49:16	INSERT INTO `Cliente` (`nombre`, `apellido`, `numeroDocumento`, `FechaNacimiento`, `direccion`, `telefono`, `mail`, `EsTerrorista`, `Cruce_idCruce1`)...	Error Code: 1062. Duplicate entry 'BN 4196361' for key 'cliente.unique_numeroDocumento'

Consulta de clientes

MySQL Workbench

Local instance MySQL80 (se...)

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

mydb

seminario

Tables

cliente

cruce

operador

repet_json

Views

Stored Procedures

Functions

sys

Administration Schemas

Information

Table: cliente

Columns:

idCliente int AI PK

nombre varchar(45)

apellido varchar(45)

numeroDocumento int

fechaNacimiento datetime

direccion varchar(45)

telefono varchar(45)

mail varchar(45)

EsTerrorista tinyint

Cruce_idCruce1 int

Result Grid

Filter Rows

Export/Import

Wrap Cell Contents

	idCliente	nombre	apellido	numeroDocumento	fechaNacimiento	direccion	telefono	mail	EsTerrorista	Cruce_idCruce1
1	Juan	Pérez	12345678	1980-05-12 00:00:00	Av. Siempre Viva 742	01123456789	juan.perez@gmail.com	0	1	
2	Ana	García	23456789	1990-07-15 00:00:00	Calle Falsa 123	01123456790	ana.garcia@hotmail.com	0	1	
3	Carlos	López	34567890	1985-01-30 00:00:00	Av. Libertador 456	01123456791	carlos.lopez@yahoo.com.ar	0	1	
4	Laura	Martínez	45678901	1992-03-20 00:00:00	Av. Corrientes 789	01123456792	laura.martinez@gmail.com	0	1	
5	Jorge	Fernández	56789012	1988-11-11 00:00:00	Calle Rivadavia 321	01123456793	jorge.fernandez@hotmail.com	0	1	
6	María	González	67890123	1995-02-14 00:00:00	Av. San Martín 654	01123456794	maria.gonzalez@yahoo.com.ar	0	1	
7	Pedro	Ramírez	78901234	1983-06-28 00:00:00	Calle Mendoza 987	01123456795	pedro.ramirez@gmail.com	0	1	
8	Sofía	Díaz	89012345	1994-08-22 00:00:00	Av. Belgrano 852	01123456796	sofia.diaz@hotmail.com	0	1	
9	Ricardo	Morales	90123456	1987-12-05 00:00:00	Calle Tucumán 159	01123456797	ricardo.morales@yahoo.com.ar	0	1	
10	Valeria	Cruz	12345679	1991-10-10 00:00:00	Av. de los Incas 111	01123456798	valeria.cruz@gmail.com	0	1	
11	Martín	Reyes	23456780	1986-09-09 00:00:00	Calle Mapu 222	01123456799	martin.reyes@hotmail.com	0	1	
12	Patricia	Cordero	34567891	1989-04-18 00:00:00	Av. La Plata 333	01123456800	patricia.cordero@yahoo.com.ar	0	1	
13	Hugo	Salazar	45678902	1982-05-25 00:00:00	Calle Godoy Cruz 444	01123456801	hugo.salazar@gmail.com	0	1	
14	Estela	Hernández	56789013	1993-07-30 00:00:00	Av. Pueyrredón 555	01123456802	estela.hernandez@hotmail.com	0	1	
15	Fernando	Soto	67890124	1980-08-17 00:00:00	Calle Santa Fe 666	01123456803	fernando.soto@yahoo.com.ar	0	1	
16	Lucía	Jiménez	78901235	1992-12-12 00:00:00	Av. Cincuentenario ...	01123456804	lucia.jimenez@gmail.com	0	1	
17	Diego	Rojas	89012346	1984-01-22 00:00:00	Calle Libertad 888	01123456805	diego.rojas@hotmail.com	0	1	
18	Sandra	Vargas	90123457	1990-03-15 00:00:00	Av. Sarmiento 999	01123456806	sandra.vargas@yahoo.com.ar	0	1	
19	Gustavo	Ponce	12345680	1985-11-30 00:00:00	Calle Xáxuv 1000	01123456807	gustavo.ponce@gmail.com	0	1	
20	Nadia	Silva	23456781	1984-04-11 00:00:00	Av. 9 de Julio 1001	01123456808	nadia.silva@hotmail.com	0	1	
21	Gabriel	Ortega	34567892	1987-05-14 00:00:00	Calle 25 de Mayo 1002	01123456809	gabriel.ortega@yahoo.com.ar	0	1	
22	Inés	Cano	45678903	1989-06-19 00:00:00	Av. Santa Fe 1003	01123456810	ines.cano@gmail.com	0	1	
23	Felipe	Alonso	56789014	1983-09-05 00:00:00	Calle Defensa 1004	01123456811	felipe.alonso@hotmail.com	0	1	
24	Cecilia	Bermúdez	67890125	1990-07-30 00:00:00	Av. Rivadavia 1005	01123456812	cecilia.bermudez@yahoo.com.ar	0	1	
25	Salvador	Mora	78901236	1986-12-15 00:00:00	Calle Salta 1006	01123456813	salvador.mora@gmail.com	0	1	
26	Lorena	Gutiérrez	89012347	1992-01-20 00:00:00	Av. Brasil 1007	01123456814	lorena.gutierrez@hotmail.com	0	1	
27	Diego	Córdoba	90123458	1984-04-11 00:00:00	Calle San Juan 1008	01123456815	diego.cordoba@yahoo.com.ar	0	1	
28	Mónica	Cisneros	12345681	1985-03-22 00:00:00	Av. Colón 1009	01123456816	monica.cisneros@gmail.com	0	1	
29	Roberto	Salinas	23456782	1993-02-09 00:00:00	Calle Corrientes 1010	01123456817	roberto.salinas@hotmail.com	0	1	
30	AAMIR	ALI CHAU...	BN 4196361	1985-10-18 00:00:00	Calle Falsa 123	123456789	aamir.ali@hotmail.com	1	1	

cliente 2 x

Output

Action Output

Time Action

1 19:32:00 SELECT * FROM seminario.cliente LIMIT 0, 200

Message

30 row(s) returned

Trabajo Práctico 3

Tras varios días de intenso trabajo y dedicación en el desarrollo del código para asegurar que se cumplieran todas las funcionalidades planteadas al inicio del proyecto, me complace informar que he alcanzado un producto casi completo. Solo queda realizar la conexión con las bases de datos, y el sistema estará listo para operar de acuerdo con los requerimientos establecidos.

Realizaré algunos ajustes en el código y en las bases de datos, ya que el archivo JSON de RePET no incluye, en todos los casos, el dato de DNI o pasaporte de los terroristas listados. En esta primera fase, realizaré el cruce de datos utilizando únicamente el nombre del cliente-terrorista. Sin embargo, se podría mejorar el producto para permitir el cruce de datos utilizando cualquiera de los tres criterios válidos: pasaporte, DNI o nombre completo.

Clase Main

Project

RePET_v2

src

Cliente

DatabaseManager

JsonLoader

Main

MainFrame

Terrorista

.gitignore

RePET_v2.iml

External Libraries

Scratches and Consoles

Main.java

```

import javax.swing.*; // Importa la biblioteca Swing para la interfaz gráfica

public class Main {
    // Metodo principal que inicia la aplicación
    public static void main(String[] args) {
        // Ejecuta la creación del marco en el hilo de despacho de eventos
        SwingUtilities.invokeLater(() -> {
            DatabaseManager dbManager = new DatabaseManager(); // Crea una instancia de DatabaseManager
            MainFrame frame = new MainFrame(dbManager); // Pasa dbManager a MainFrame
            frame.setVisible(true); // Hace visible el marco
        });
    }
}

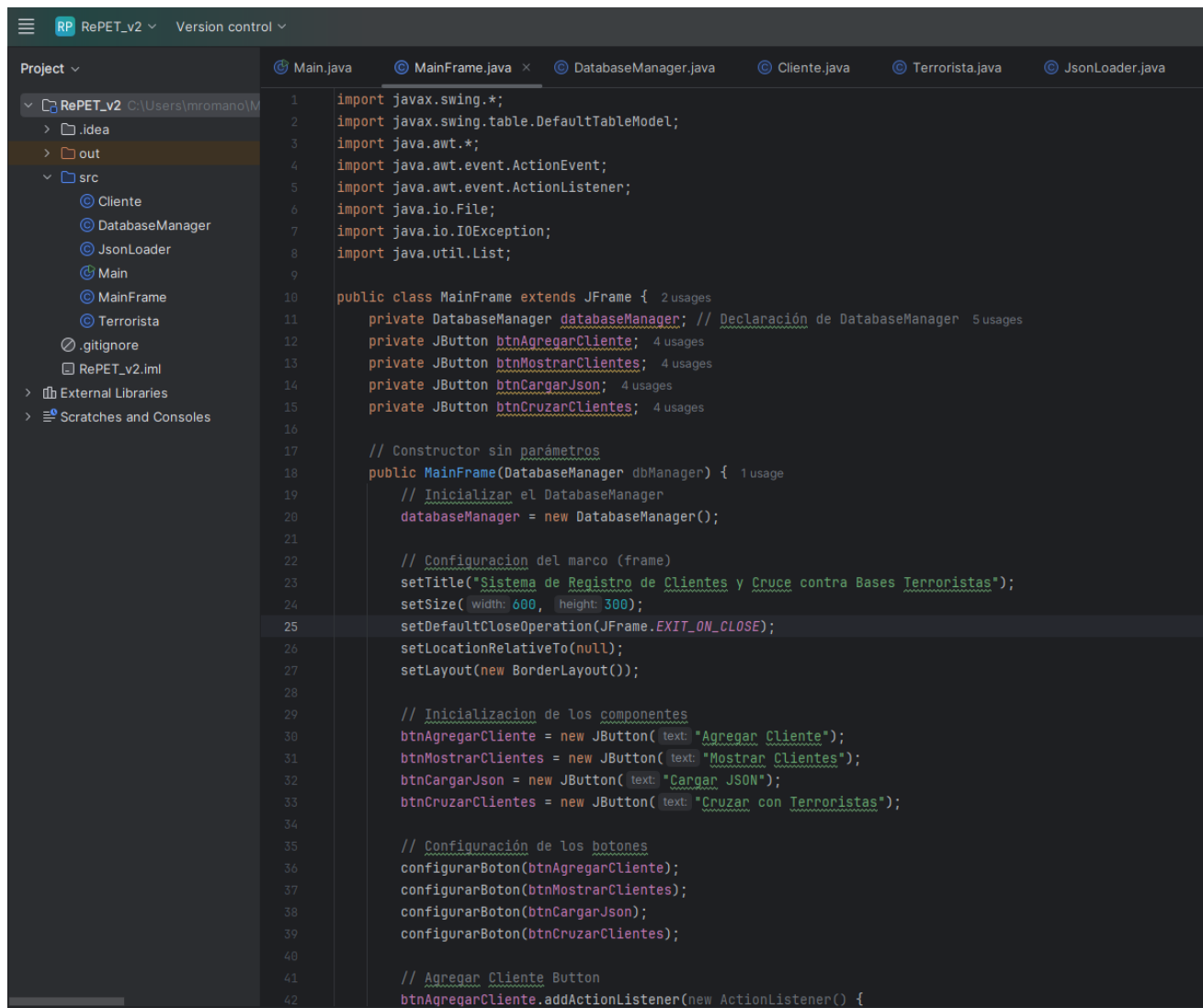
```

Esta es la clase principal del programa que inicia la aplicación.

Funcionalidades:

- Contiene el método main, que es el punto de entrada del programa.
- Utiliza `SwingUtilities.invokeLater` para garantizar que la creación y visualización de la interfaz gráfica se realice en el hilo de despacho de eventos, lo cual es una buena práctica en programación con Swing.
- Crea una instancia de `DatabaseManager`, que se encarga de gestionar la base de datos de clientes.
- Crea una instancia de `MainFrame`, que representa la ventana principal de la aplicación, pasándole el `dbManager` para que pueda interactuar con la base de datos.
- Finalmente, establece la visibilidad del marco (frame) para mostrar la interfaz al usuario.

Clase MainFrame



```
1 import javax.swing.*;
2 import javax.swing.table.DefaultTableModel;
3 import java.awt.*;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.io.File;
7 import java.io.IOException;
8 import java.util.List;
9
10 public class MainFrame extends JFrame {
11     private DatabaseManager databaseManager; // Declaración de DatabaseManager
12     private JButton btnAgregarCliente; // 4 usages
13     private JButton btnMostrarClientes; // 4 usages
14     private JButton btnCargarJson; // 4 usages
15     private JButton btnCruzarClientes; // 4 usages
16
17     // Constructor sin parámetros
18     public MainFrame(DatabaseManager dbManager) {
19         // Inicializar el DatabaseManager
20         databaseManager = new DatabaseManager();
21
22         // Configuración del marco (frame)
23         setTitle("Sistema de Registro de Clientes y Cruce contra Bases Terroristas");
24         setSize(600, 300);
25         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26         setLocationRelativeTo(null);
27         setLayout(new BorderLayout());
28
29         // Inicialización de los componentes
30         btnAgregarCliente = new JButton("Agregar Cliente");
31         btnMostrarClientes = new JButton("Mostrar Clientes");
32         btnCargarJson = new JButton("Cargar JSON");
33         btnCruzarClientes = new JButton("Cruzar con Terroristas");
34
35         // Configuración de los botones
36         configurarBoton(btnAgregarCliente);
37         configurarBoton(btnMostrarClientes);
38         configurarBoton(btnCargarJson);
39         configurarBoton(btnCruzarClientes);
40
41         // Agregar Cliente Button
42         btnAgregarCliente.addActionListener(new ActionListener() {
```

La clase `MainFrame` representa la ventana principal de la aplicación y es responsable de gestionar la interfaz gráfica y las interacciones del usuario.

Funcionalidades

1. Constructor:

- Recibe un objeto de `DatabaseManager` para interactuar con la base de datos de clientes.
- Configura el marco (título, tamaño, comportamiento al cerrar) y el diseño de la interfaz.
- Inicializa los botones para cada funcionalidad y los agrega a un panel con un diseño en cuadrícula.

2. Configuración de Botones:

- **Método configurarBoton:** Establece propiedades visuales (fuente y color) para los botones.

3. Agregar Cliente:

- **Método agregarCliente:**
 - Muestra un diálogo para que el usuario ingrese los datos del cliente.
 - Crea un objeto Cliente y lo agrega a la base de datos a través de databaseManager.
 - Notifica al usuario si la operación fue exitosa o si ya existe un cliente con el mismo DNI o pasaporte.

4. Mostrar Clientes:

- **Método mostrarClientes:**
 - Recupera la lista de clientes de databaseManager.
 - Si hay clientes registrados, crea una tabla con los datos y la muestra en un diálogo.
 - Notifica al usuario si no hay clientes registrados.

5. Cargar JSON:

- **Método en el ActionListener de btnCargarJson:**
 - Permite seleccionar un archivo JSON desde el sistema de archivos.
 - Utiliza JsonLoader para cargar los datos de terroristas y actualiza la lista en databaseManager.
 - Muestra los terroristas cargados en un diálogo y notifica el éxito o error en la carga.

6. Cruzar Clientes:

- **Método cruzarClientes:**
 - Llama a databaseManager para cruzar la lista de clientes con la de terroristas.
 - Notifica al usuario si se encontraron coincidencias o no.

7. Mostrar Datos Cargados:

- **Método mostrarDatosCargados:**
 - Recibe una lista de objetos Terrorista y crea una tabla similar a la de clientes para mostrar la información.
 - Utiliza un JScrollPane para mostrar la tabla en un diálogo.

Clase DatabaseManager

```

1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class DatabaseManager {
5     private List<Cliente> listaClientes; // Lista de clientes
6     private List<Terrorista> listaTerroristas; // Lista de terroristas
7
8     public DatabaseManager() {
9         listaClientes = new ArrayList<>(); // Inicializa la lista de clientes
10        listaTerroristas = new ArrayList<>(); // Inicializa la lista de terroristas
11        cargarClientes(this); // Carga los clientes al inicializar
12    }
13
14    // Método para cargar los clientes
15    private static void cargarClientes(DatabaseManager dbManager) {
16        dbManager.agregarCliente(new Cliente("Juan", "Perez", "12345678", "1980-01-01", "Calle Pasa 123", "01123456789"));
17        dbManager.agregarCliente(new Cliente("Ana", "Gomez", "987654321", "1992-02-01", "Av. Siempre Viva 456", "01198765432"));
18        dbManager.agregarCliente(new Cliente("Carlos", "Martinez", "234567890", "1985-03-15", "Calle 8 de Julio 789", "01123456789"));
19        dbManager.agregarCliente(new Cliente("Lucia", "Rodriguez", "456789012", "1995-04-04", "Calle 9 de Julio 789", "01187654321"));
20        dbManager.agregarCliente(new Cliente("MAMAD IRANI", "SMILEN", "90876543", "1980-05-05", "Calle del Terrorista 101", "01154321098"));
21        dbManager.agregarCliente(new Cliente("Martín", "Fernández", "567890123", "1982-06-04", "Calle Libertador 201", "01154321098"));
22        dbManager.agregarCliente(new Cliente("Dora", "Lopez", "098765432", "1990-07-07", "Calle San Martín 301", "01123456789"));
23        dbManager.agregarCliente(new Cliente("Pablo", "Bartia", "78901234", "1988-08-08", "Calle Barrientos 404", "01123456789"));
24        dbManager.agregarCliente(new Cliente("María", "Hernández", "89012345", "1993-09-09", "Calle Belgrano 505", "01154321098"));
25        dbManager.agregarCliente(new Cliente("Andrés", "Martinez", "90123456", "1985-10-10", "Calle 25 de Mayo 606", "01154321098"));
26        dbManager.agregarCliente(new Cliente("Elena", "Cruz", "11234567", "1984-11-11", "Calle El Dorado 707", "01154321098"));
27        dbManager.agregarCliente(new Cliente("Javier", "Vazquez", "12345678", "1991-12-12", "Calle Moreno 808", "01178901234"));
28        dbManager.agregarCliente(new Cliente("Gabriela", "Alvarez", "23456789", "1989-01-13", "Calle Alina 909", "01123456789"));
29        dbManager.agregarCliente(new Cliente("Raúl", "Diaz", "34567890", "1987-02-14", "Calle 101", "01109876543"));
30        dbManager.agregarCliente(new Cliente("Carla", "Ponce", "45678901", "1990-03-15", "Calle Córdoba 202", "01154321098"));
31        dbManager.agregarCliente(new Cliente("Nicolás", "Gonzalez", "56789012", "1983-04-16", "Calle Mendoza 303", "01154321098"));
32        dbManager.agregarCliente(new Cliente("Paola", "Ramirez", "67890123", "1994-05-17", "Calle Tucuman 404", "01176543210"));
33        dbManager.agregarCliente(new Cliente("Leonardo", "Gota", "78901234", "1988-06-18", "Calle La Rioja 505", "01154321098"));
34        dbManager.agregarCliente(new Cliente("Fernando", "Garcia", "89012345", "1991-07-19", "Calle Santa Fe 606", "01123456789"));
35        dbManager.agregarCliente(new Cliente("Ignacio", "Balboa", "90123456", "1980-08-20", "Calle Chaco 707", "01178901234"));
36        dbManager.agregarCliente(new Cliente("Victoria", "Ibarra", "01234567", "1985-09-21", "Calle Catamarca 808", "011123456789"));
37    }
38
39    // Método para obtener la lista de clientes
40    public List<Cliente> getListaClientes() { return listaClientes; } // Retorna la lista de clientes
41
42    // Método para cargar los clientes

```

1) Atributos Principales:

- **listaClientes:** Lista que almacena objetos Cliente.
- **listaTerroristas:** Lista que almacena objetos Terrorista.

2) Constructor:

- Inicializa ambas listas (listaClientes y listaTerroristas) y llama al método cargarClientes() para agregar ejemplos de clientes a la lista inicial.

3) Métodos:

- **cargarClientes(DatabaseManager dbManager):** Agrega ejemplos de clientes a la lista de clientes.
- **getListaClientes():** Retorna la lista de clientes.
- **agregarCliente(Cliente nuevoCliente):** Agrega un cliente a la lista, validando que no exista otro con el mismo DNI o pasaporte.
- **getListaTerroristas():** Retorna la lista de terroristas.
- **setListaTerroristas(List<Terrorista> terroristas):** Permite establecer la lista de terroristas.
- **cruzarClientes():** Compara los nombres de los clientes con los de terroristas y retorna una lista con aquellos clientes que tienen coincidencias de nombres con terroristas.

Clase Cliente

```

1 public class Cliente { 34 usages
2     private String nombre; // Nombre del cliente 4 usages
3     private String apellido; // Apellido del cliente 4 usages
4     private String dni; // Documento Nacional de Identidad 3 usages
5     private String pasaporte; // Numero de pasaporte 3 usages
6     private String fechaNacimiento; // Fecha de nacimiento 3 usages
7     private String direccion; // Direccion de residencia 3 usages
8     private String telefono; // Numero de telefono 3 usages
9     private String mail; // Correo electronico 3 usages
10
11     // Constructor de la clase Cliente
12     public Cliente(String nombre, String apellido, String dni, String pasaporte, String fechaNacimiento, 22 usages
13                     String direccion, String telefono, String mail) {
14         this.nombre = nombre; // Inicializa el nombre
15         this.apellido = apellido; // Inicializa el apellido
16         this.dni = dni; // Inicializa el DNI
17         this.pasaporte = pasaporte; // Inicializa el pasaporte
18         this.fechaNacimiento = fechaNacimiento; // Inicializa la fecha de nacimiento
19         this.direccion = direccion; // Inicializa la direccion
20         this.telefono = telefono; // Inicializa el telefono
21         this.mail = mail; // Inicializa el correo electronico
22     }
23
24     // Getters para obtener los atributos del cliente
25     public String getNombre() { 1 usage
26         return nombre; // Retorna el nombre
27     }
28
29     public String getApellido() { 1 usage
30         return apellido; // Retorna el apellido
31     }
32
33     public String getDni() { 3 usages
34         return dni; // Retorna el DNI
35     }
36
37     public String getPasaporte() { 4 usages
38         return pasaporte; // Retorna el pasaporte
39     }
40
41     public String getFechaNacimiento() { 1 usage
42         return fechaNacimiento; // Retorna la fecha de nacimiento
  
```

1) Atributos:

- Almacena información clave del cliente: nombre, apellido, dni, pasaporte, fechaNacimiento, direccion, telefono, y mail.

2) Constructor:

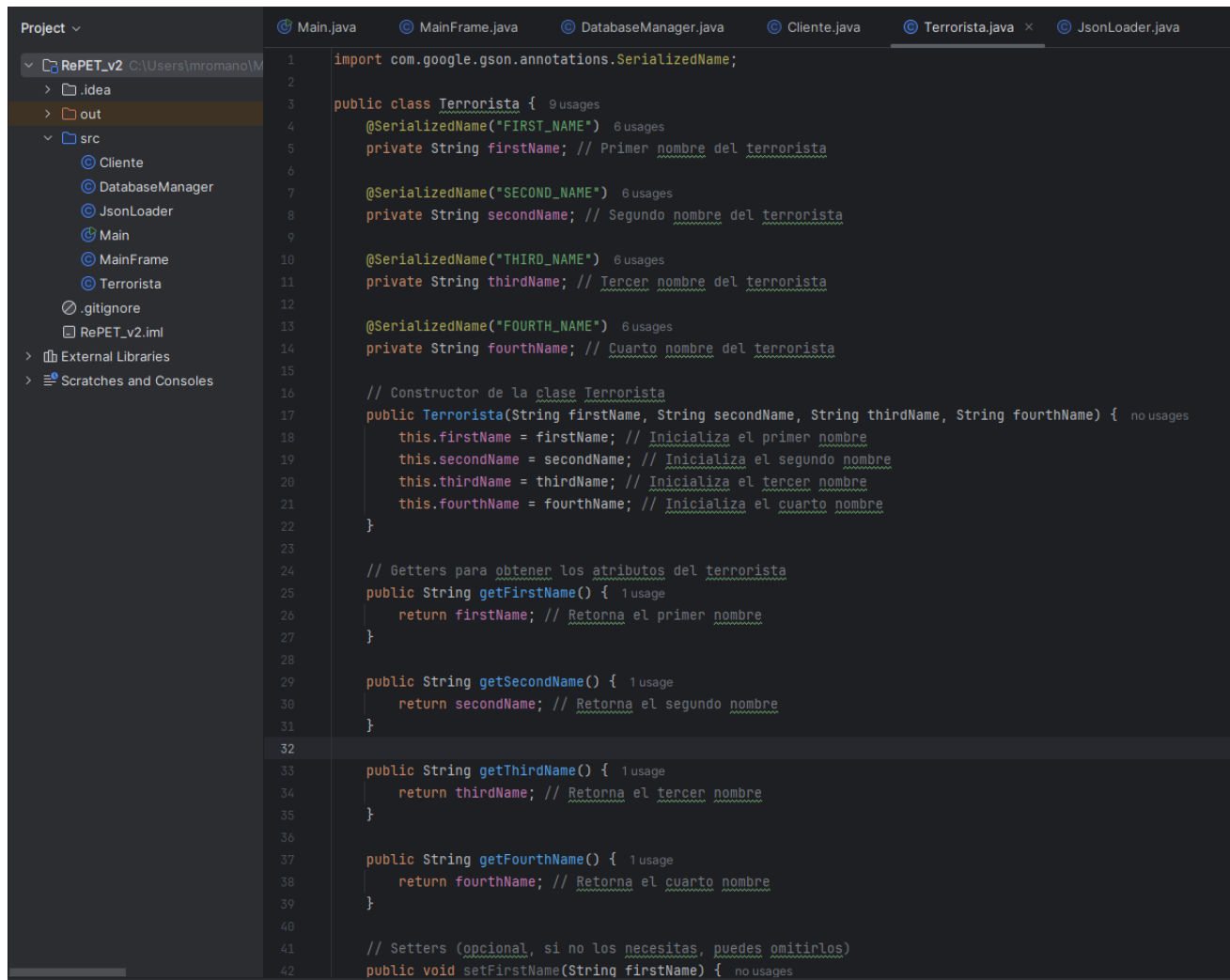
- Recibe los valores de cada atributo y los asigna para inicializar un objeto Cliente.

3) Getters:

- Métodos que permiten acceder a cada atributo de forma individual (getNombre(), getApellido(), getDni(), etc.).

4) Métodos Adicionales:

- `getNombreCompleto()`: Retorna el nombre completo del cliente concatenando nombre y apellido.
- `toString()`: Proporciona una representación en texto de todos los atributos del cliente para una fácil visualización.

Clase Terrorista

```
1 import com.google.gson.annotations.SerializedName;
2
3 public class Terrorista {
4     @SerializedName("FIRST_NAME")
5     private String firstName; // Primer nombre del terrorista
6
7     @SerializedName("SECOND_NAME")
8     private String secondName; // Segundo nombre del terrorista
9
10    @SerializedName("THIRD_NAME")
11    private String thirdName; // Tercer nombre del terrorista
12
13    @SerializedName("FOURTH_NAME")
14    private String fourthName; // Cuarto nombre del terrorista
15
16    // Constructor de la clase Terrorista
17    public Terrorista(String firstName, String secondName, String thirdName, String fourthName) {
18        this.firstName = firstName; // Inicializa el primer nombre
19        this.secondName = secondName; // Inicializa el segundo nombre
20        this.thirdName = thirdName; // Inicializa el tercer nombre
21        this.fourthName = fourthName; // Inicializa el cuarto nombre
22    }
23
24    // Getters para obtener los atributos del terrorista
25    public String getFirstName() {
26        return firstName; // Retorna el primer nombre
27    }
28
29    public String getSecondName() {
30        return secondName; // Retorna el segundo nombre
31    }
32
33    public String getThirdName() {
34        return thirdName; // Retorna el tercer nombre
35    }
36
37    public String getFourthName() {
38        return fourthName; // Retorna el cuarto nombre
39    }
40
41    // Setters (opcional, si no los necesitas, puedes omitirlos)
42    public void setFirstName(String firstName) {
```

1) Atributos:

- Contiene información de identificación del terrorista: `firstName`, `secondName`, `thirdName` y `fourthName`.
- Los nombres están mapeados a etiquetas JSON mediante `@SerializedName`, lo cual ayuda a que la clase se serialice/deserialice correctamente usando nombres específicos en formato JSON.

2) Constructor:

- Inicializa un objeto `Terrorista` con valores para cada nombre.

3) Getters y Setters:

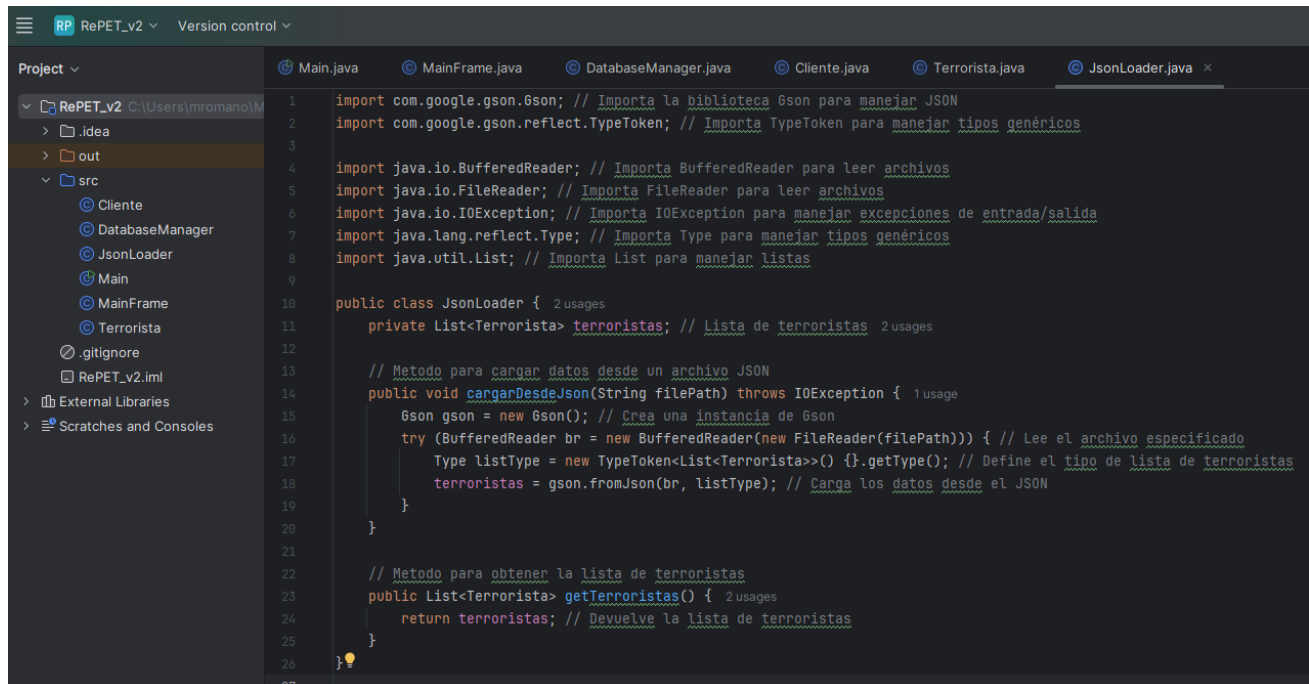
- Métodos `get` para acceder a cada atributo y `set` (opcional) para modificar cada nombre (`getFirstName()`, `setFirstName()`, etc.).

4) Método Adicional:

- `getNombreCompleto()`: Concatena los nombres que no sean nulos y devuelve el nombre completo en un solo String, eliminando espacios adicionales.

5) Método toString():

- Devuelve una representación en texto del objeto Terrorista mostrando los nombres almacenados.

Clase JsonLoader

```
1 import com.google.gson.Gson; // Importa la biblioteca Gson para manejar JSON
2 import com.google.gson.reflect.TypeToken; // Importa TypeToken para manejar tipos genéricos
3
4 import java.io.BufferedReader; // Importa BufferedReader para leer archivos
5 import java.io.FileReader; // Importa FileReader para leer archivos
6 import java.io.IOException; // Importa IOException para manejar excepciones de entrada/salida
7 import java.lang.reflect.Type; // Importa Type para manejar tipos genéricos
8 import java.util.List; // Importa List para manejar listas
9
10 public class JsonLoader { 2 usages
11     private List<Terrorista> terroristas; // Lista de terroristas 2 usages
12
13     // Metodo para cargar datos desde un archivo JSON
14     public void cargarDesdeJson(String filePath) throws IOException { 1 usage
15         Gson gson = new Gson(); // Crea una instancia de Gson
16         try (BufferedReader br = new BufferedReader(new FileReader(filePath))) { // Lee el archivo especificado
17             Type listType = new TypeToken<List<Terrorista>>().getType(); // Define el tipo de lista de terroristas
18             terroristas = gson.fromJson(br, listType); // Carga los datos desde el JSON
19         }
20     }
21
22     // Metodo para obtener la lista de terroristas
23     public List<Terrorista> getTerroristas() { 2 usages
24         return terroristas; // Devuelve la lista de terroristas
25     }
26 }
27
```

1) Atributo:

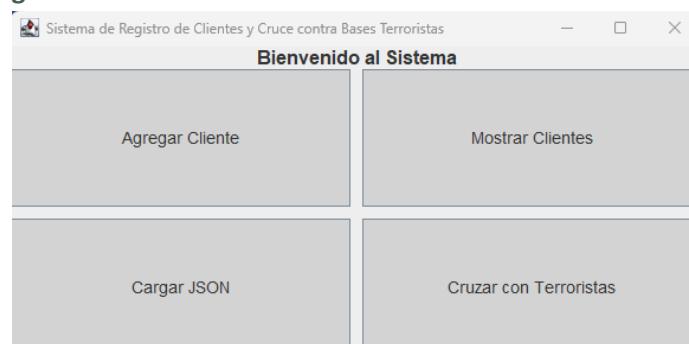
- terroristas: Una lista de objetos Terrorista que almacenará los datos cargados desde un archivo JSON.

2) Método cargarDesdeJson:

- Utiliza la biblioteca Gson para cargar un archivo JSON.
- Recibe la ruta del archivo (filePath) como parámetro y convierte su contenido en una lista de objetos Terrorista.
- Usa BufferedReader para leer el archivo y TypeToken para definir el tipo de datos (lista de Terrorista).
- Los datos JSON se deserializan y se asignan a la lista terroristas.

3) Método getTerroristas:

- Devuelve la lista de objetos Terrorista cargada desde el archivo JSON para que pueda ser utilizada en otros procesos.

Pantalla principal del programa:

Como primera instancia, procederé a presentar el funcionamiento de la primera utilidad del sistema: la carga de clientes.

En el código, se incluye una carga inicial que contiene un terrorista listado en RePET, lo que permite verificar el correcto funcionamiento del proceso de cruce de datos. Es importante destacar que este terrorista puede aparecer en cualquier orden respecto a los nombres y apellidos, lo que garantiza la flexibilidad y robustez del

sistema en la identificación de coincidencias. Además, el sistema no distingue entre mayúsculas y minúsculas al realizar el cruce, lo que facilita aún más la comparación de nombres.

```

public class DatabaseManager {
    public DatabaseManager() {
    }

    private static void cargarClientes(DatabaseManager dbManager) {
        // Ejemplo de carga de 20 clientes
        dbManager.agregarCliente(new Cliente(nombre: "Juan", apellido: "Pérez", dni: "12345678", pasaporte: "", fechaNacimiento: "1990-01-01"));
        dbManager.agregarCliente(new Cliente(nombre: "Ana", apellido: "Gómez", dni: "87654321", pasaporte: "", fechaNacimiento: "1992-05-15"));
        dbManager.agregarCliente(new Cliente(nombre: "Carlos", apellido: "Martínez", dni: "34567890", pasaporte: "", fechaNacimiento: "1988-12-03"));
        dbManager.agregarCliente(new Cliente(nombre: "Lucia", apellido: "Rodríguez", dni: "45678901", pasaporte: "", fechaNacimiento: "1995-07-22"));
        dbManager.agregarCliente(new Cliente(nombre: "MUHAMMAD IBRAHIM", apellido: "SHOLEH", dni: "6908545", pasaporte: "", fechaNacimiento: "1990-03-10"));
        dbManager.agregarCliente(new Cliente(nombre: "Martín", apellido: "Fernández", dni: "56789012", pasaporte: "", fechaNacimiento: "1991-08-05"));
        dbManager.agregarCliente(new Cliente(nombre: "Sofía", apellido: "López", dni: "67890123", pasaporte: "", fechaNacimiento: "1993-02-18"));
        dbManager.agregarCliente(new Cliente(nombre: "Pablo", apellido: "García", dni: "78901234", pasaporte: "", fechaNacimiento: "1989-11-07"));
        dbManager.agregarCliente(new Cliente(nombre: "Maria", apellido: "Hernández", dni: "89012345", pasaporte: "", fechaNacimiento: "1994-06-25"));
        dbManager.agregarCliente(new Cliente(nombre: "Andrés", apellido: "Martínez", dni: "90123456", pasaporte: "", fechaNacimiento: "1996-04-12"));
        dbManager.agregarCliente(new Cliente(nombre: "Elena", apellido: "Cruz", dni: "01234567", pasaporte: "", fechaNacimiento: "1997-09-01"));
        dbManager.agregarCliente(new Cliente(nombre: "Javier", apellido: "Vázquez", dni: "12345679", pasaporte: "", fechaNacimiento: "1998-01-20"));
        dbManager.agregarCliente(new Cliente(nombre: "Gabriela", apellido: "Alvarez", dni: "23456780", pasaporte: "", fechaNacimiento: "1999-05-08"));
        dbManager.agregarCliente(new Cliente(nombre: "Facundo", apellido: "Díaz", dni: "34567890", pasaporte: "", fechaNacimiento: "2000-03-15"));
        dbManager.agregarCliente(new Cliente(nombre: "Carla", apellido: "Ponce", dni: "45678901", pasaporte: "", fechaNacimiento: "2001-07-03"));
        dbManager.agregarCliente(new Cliente(nombre: "Nicolás", apellido: "González", dni: "56789012", pasaporte: "", fechaNacimiento: "2002-11-10"));
        dbManager.agregarCliente(new Cliente(nombre: "Paola", apellido: "Ramírez", dni: "67890123", pasaporte: "", fechaNacimiento: "2003-04-22"));
        dbManager.agregarCliente(new Cliente(nombre: "Leonardo", apellido: "Soto", dni: "78901234", pasaporte: "", fechaNacimiento: "2004-08-05"));
        dbManager.agregarCliente(new Cliente(nombre: "Verónica", apellido: "Castro", dni: "89012345", pasaporte: "", fechaNacimiento: "2005-12-18"));
        dbManager.agregarCliente(new Cliente(nombre: "Ignacio", apellido: "Maldonado", dni: "90123456", pasaporte: "", fechaNacimiento: "2006-06-01"));
        dbManager.agregarCliente(new Cliente(nombre: "Victoria", apellido: "Ibarra", dni: "01234567", pasaporte: "", fechaNacimiento: "2007-10-15"));
    }
}

```

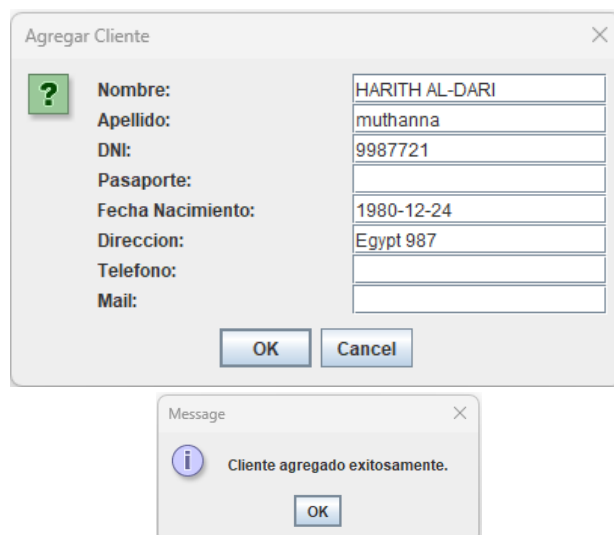
En primera instancia procedemos a probar el botón de “Agregar Cliente”:

Para verificar que el proceso de cruce funciona adecuadamente, vamos a añadir un terrorista con el nombre en un orden desordenado. Debemos tener presente este nombre para el cruce, ya que debería aparecer en la lista cuando lo comparemos con el archivo JSON de RePET.

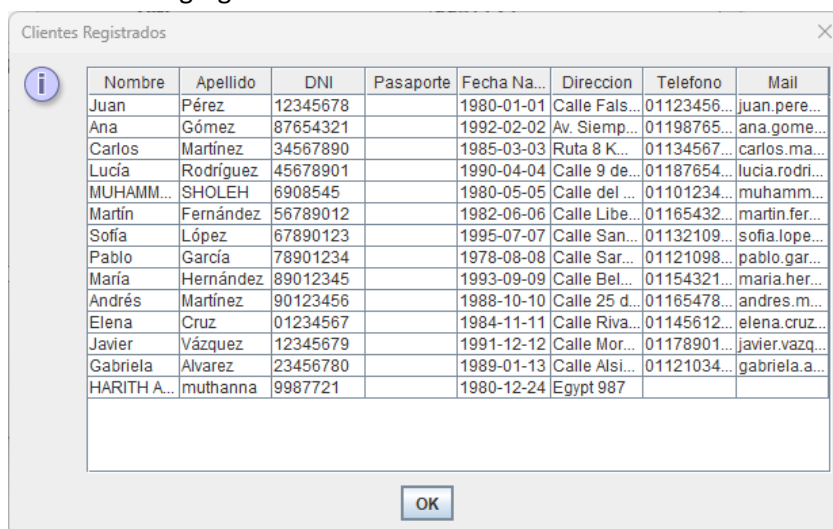
```

29029 {
29030     "DATAID": "2894441",
29031     "VERSIONNUM": "1",
29032     "FIRST_NAME": "MUTHANNA",
29033     "SECOND_NAME": "HARITH",
29034     "THIRD_NAME": "AL-DARI",
29035     "FOURTH_NAME": "",
29036     "UN_LIST_TYPE": "Al-Qaida",
29037     "REFERENCE_NUMBER": "QDi.278",
29038     "LISTED ON": "25/03/2010",
29039     "COMMENTS1": "Mother's name: Heba Khamis Dari. Pro
smuggling. Wanted by the Iraqi security forces. Photo avail
Council resolution 2610 (2021) was concluded on 30 October
29040     "GENDER": "",
29041     "SUBMITTED BY": "",
29042     "NAME_ORIGINAL_SCRIPT": "مثنى حارث الضاري",
29043     "NATIONALITY2": "",
29044     "SORT_KEY": "",
29045     "SORT_KEY_LAST_MOD": "",
29046     "DELISTED ON": "",
29047     "LIST TYPE": "UN List",
29048     "DESIGNATION": [],
29049     "INDIVIDUAL_ADDRESS": [
29050     {

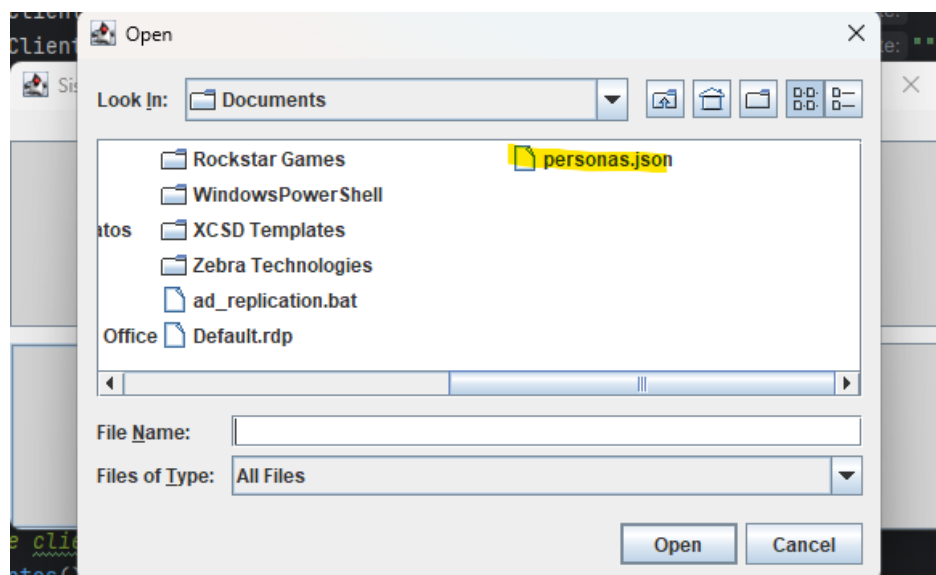
```



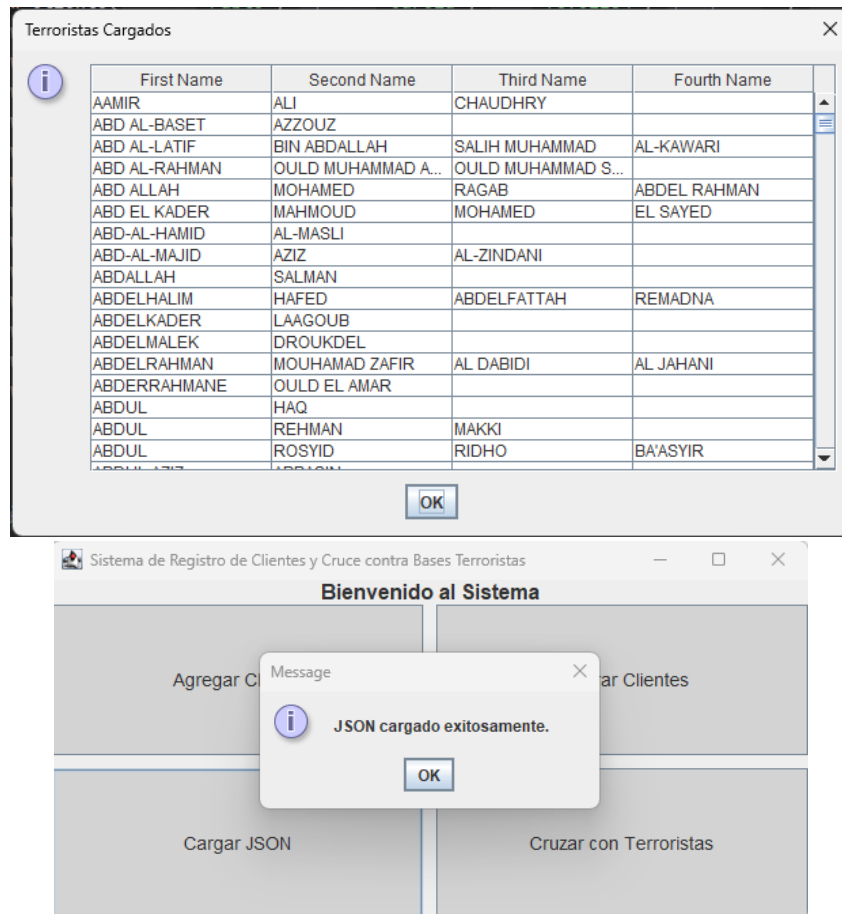
Ahora procederemos a mostrar la función "Mostrar Clientes", donde podremos ver tanto los clientes pre-cargados como el último cliente agregado.



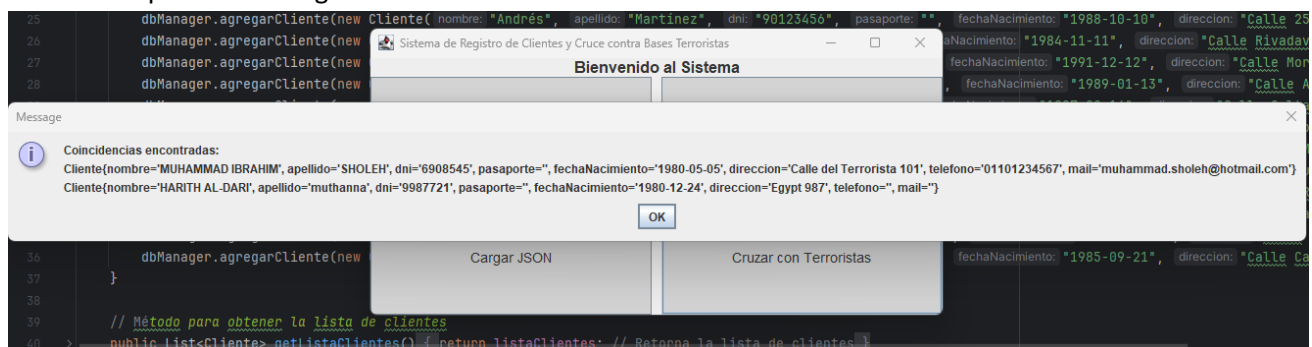
Ahora procederemos a realizar la carga del archivo JSON que hemos descargado desde la página del Ministerio de Justicia.



El sistema proyectara el listado de terroristas extraídos desde el archivo JSON y cargados al sistema:



Finalmente, llevaremos a cabo el cruce de datos entre la base de datos de clientes y el archivo JSON de terroristas que hemos cargado:



En esta sección, podemos observar que están listados los dos terroristas infiltrados: uno que agregamos durante la precarga y otro que se añadió de forma desordenada al registrar un nuevo cliente. Esto demuestra que el sistema está funcionando correctamente, tal como se había planteado en los objetivos iniciales del proyecto.

Enlace a GitHub

<https://github.com/millonariosoy14/seminario/blob/main/seminario.sql>

BIBLIOGRAFIA

- https://siglo21.instructure.com/courses/35767/pages/seminario-de-practica#lectura_01
- Encuentro sincrónico: <https://www.youtube.com/watch?v=K6COsdQiDMc>
- <https://lucid.app/> - Armado de diagrama de dominio, casos de uso.
- Encuentro sincrónico: <https://www.youtube.com/watch?v=nAUEyIOpvco>
- https://siglo21.instructure.com/courses/35767/pages/seminario-de-practica#lectura_02