Ifrán, Julián Agustín

Data Quality Engineering

2022

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA

Informe final

Data Quality Enginnering

Julián Agustín Ifrán

Florencio Varela, agosto de 2022

Estudiante
Julián Agustín Ifrán
julian_ifran@hotmail.com

Organización donde se realiza la Práctica Profesional Supervisada
CONSULTORIA EN TECNOLOGIA DE INFORM. SA
Perón 1730, CABA.
+54 11 5263 3777
Sector: Desarrollo de Software

Tutor de la organizacional
Dr. Ing. Marcelo Cappelletti
mcappelletti@unaj.edu.ar

Docente supervisor
Ing. Julissa Atía
julissa.atia@gmail.com

Docente tutor del Taller de Apoyo para la Producción de Textos Académicos
Lic. Paula Bein
paula.bein@gmail.com

Coordinador de la carrera de Ingeniería en Informática
Dr. Ing. Morales, Martín
martin.morales@unaj.edu.ar

Resumen

Este proyecto se realizó en la empresa de consultoría de software Softtek para su cliente Techint Ingeniería & Construcción (TEIC) y consistió en el desarrollo y puesta en producción de un sistema que mida la calidad de sus datos almacenados Esta necesidad surge por parte del cliente, ya que necesitaba de una plataforma propia que le permitiera analizar sus bases de datos, encontrar inconsistencia y llevar a cabo acciones correctivas

Para ello, el objetivo del proyecto fue utilizar factores, métricas, métodos y reglas que permitieran medir (de forma cuantificable) la calidad de los datos. También se aprovechó el aprendizaje automático con la intención de detectar anomalías en los datos y así generar alertas tempranas.

Las tareas del proyecto abarcan: el desarrollo de una aplicación que permitiera la creación y ejecución de reglas de calidad, la búsqueda de anomalías y la generación de alertas tempranas. Para esta tarea se utilizó un ecosistema principal de aplicaciones compuesto por C#, SQL Server, React y DevOps.

Con este proyecto no solo se logró mejorar la calidad de los datos. Sino que también permitió acercar el cliente al desarrollo de métodos propios de medición, la utilización de inteligencia artificial, la detección temprana de anomalías. Para que en el futuro se pueda continuar en la implementación de un modelo de MLOPs y en nuevas acciones correctivas.

Gobernanza de Datos - Base de Datos - Calidad de Datos - Aprendizaje Automático - MLOPs

Abstract

This project was carried out in the software consulting company Softtek for its client Techint Ingeniería & Construcción (TEIC). It consisted of the development and production of a system that measures the quality of stored data. This need arises on the part of the client, since they needed their own platform that would allow them to analyze their databases, find inconsistencies and carry out corrective actions.

The objective of the project was to use factors, metrics, methods and rules that would allow to measure (in a quantifiable way) the quality of the data and thus meet the needs of the client. Machine learning was also leveraged with the intention of detecting anomalies in the data and being able to generate early warnings.

The tasks of the project were: the development of an application that would allow the creation and execution of quality rules, the search for anomalies and the generation of warnings. A core ecosystem of applications consisting of C#, SQL Server, React, and DevOps was used for this task.

With this project it was not only possible to improve the quality of the data, but it also allowed to bring to the client the development of their own measurement methods, the use of artificial intelligence and the early detection of anomalies. In this way, in the future it will be possible to continue in the implementation of a model of MLOPs and in new corrective actions.

Data Governance - Database - Data Quality- Machine Learning - MLOPs

Dedicatorias y agradecimientos

Este trabajo está dedicado a mi papá que me enseñó que un lápiz pesa menos que una pala y a mi mamá que, además de convencerme de estudiar ingeniería, fue el pilar más importante para llegar hasta aquí.

Agradezco a mis tutores Marcelo Cappelletti y Julissa Atia por acompañarme durante la realización de este trabajo, a la profesora Paula Bein por el apoyo, las enseñanzas y los consejos durante el desarrollo de este informe y a Christian Azcón por depositar su confianza y permitirme desarrollar esta aplicación a fines de egresarme.

También quiero agradecer a mi familia, amigos de la universidad y el trabajo. Gracias por estar presentes, por felicitarme cuando rendía bien y consolarme cuando rendía mal.

Por último, no quiero olvidarme de mi querida Universidad Nacional Arturo Jauretche por ser la sede de todo el conocimiento adquirido en estos años.

Julián

Índice

Índice de figuras

1 Introducción

La presente Práctica Profesional Supervisada (PPS) tiene como finalidad el desarrollo e implementación de un sistema de medición de calidad de datos de los aplicativos que tiene una empresa.

En una organización, la calidad de los datos es esencial para la consistencia de reportes, decisiones, eficacia de procesos operativos y la confianza de los clientes.

Por lo tanto, la falta de ésta es uno de los principales problemas a los que se enfrentan las empresas, ya que muchas veces permanece oculto o no es considerado como un aspecto prioritario.

Actualmente, el equipo desarrollo de Softtek que presta servicio en TEIC cuenta con testers (encargados de probar el software), pero no realiza ningún control ni auditoría en la calidad de los datos almacenados. Es por esto que el cliente ha solicitado implementar una plataforma que realice estas acciones para que así sea capaz de analizar las bases de datos, encontrar inconsistencias y enviar alertas periódicas cuando alguno no cumpla las condiciones deseadas.

1.1 Instituciones involucradas

Softtek

Fundada en 1982, es una compañía mexicana dedicada al desarrollo de software y otros servicios relacionados con las Tecnologías de la Información y la Comunicación (TIC).

Techint Ingeniería & Construcción (TEIC)

Es una empresa perteneciente al grupo Techint con más de 75 años de experiencia en el mercado, desarrolla soluciones integrales de gestión de proyectos, ingeniería, suministros y construcción en los segmentos de mercado: refinamiento del petróleo crudo y al procesamiento y purificación del gas natural, energía, minería, plantas petroquímicas e industriales, y obras civiles tanto de arquitectura como infraestructura.

1.2 Objetivos

Uno de los objetivos de la PPS es diseñar y desarrollar un sistema que permita mejorar los avisos tempranos de anomalías de calidad de datos y las integraciones entre sistemas que los utilizan para disparar acciones. Para ellos, se desea establecer metas medibles y realistas para implementar vía aprendizaje automático (machine learning o ML) una detección automática de anomalías.

Esto se debe a que ML funciona mejor para esta tarea ya que resulta más oportuno que la detección manual, es adaptable a los cambios y tiene la capacidad de gestionar fácilmente grandes conjuntos de datos. De esta manera, se busca analizar mayor información, en menor tiempo, de manera automática y anticipándose a posibles errores que impactan de forma negativa en las decisiones que tome la organización.

Otro fin que se busca es llevar a cabo una Gobernanza de datos eficiente. Es decir, implementación de una serie de prácticas, normas y/o políticas, que garantiza la disponibilidad, calidad, coherencia, confiabilidad y seguridad de estos en el seno de la empresa. De este modo y tal como se anticipó anteriormente, tener una buena gestión de datos permitirá tomar las mejores decisiones para el negocio, lo que resulta en un aumento de la productividad y de la eficiencia operacional y, consecuentemente, en un incremento en los ingresos de la organización.

Por último, se busca impactar en la satisfacción del cliente, brindándole una solución que le permita reducir pérdidas derivadas de malas decisiones, minimizar costos y maximizar los beneficios aumentando así, el nivel de rentabilidad del negocio.

1.3 Tareas para ejecutar

El desarrollo del proyecto se llevará a cabo durante cuatro meses. El mismo será trabajado bajo metodologías ágiles, dividido en iteraciones o sprints donde la mayoría será de dos semanas de duración cada uno. durante cada uno se realizará refinamiento de las tareas, plannings (reuniones para definir las

tareas a realizar durante el sprint), retrospectivas, test por parte del products owner (los dueños del producto y quienes lo aprueban) y análisis de los requerimientos a desarrollar durante el mismo. A medida que son implementados y antes de ser puestos en producción, cada funcionalidad tendrá que ser testeada en un ambiente de QAS que cumpla las mismas condiciones que el de producción, para validar el correcto funcionamiento y luego pueda ser liberado.

- Configuración y alcance: reunirse con el Product Owner quien es el líder del equipo de Data y Analytics para comprender la naturaleza del proyecto y definir el alcance de este. El mismo no se realizará solo al comienzo, sino que a medida que se avance en cada iteración se incluirán reuniones con el mismo para alcanzar una retroalimentación más eficaz. También se debe crear el repositorio y configurar los entornos de este.
- Mockup UX: se implementará un mapa del flujo de valor que permita visualizar el proceso a lo largo de la cadena de valor desde que se solicita un elemento mínimo hasta que se entrega el software al cliente. Luego, se deberán generar y mostrar al usuario las interfaces con las que interactuara en el sistema.
- Desarrollo de Backend: durante dicha etapa no solo se crearán las funcionalidades básicas y las tablas necesarias en la base de datos. Sino que también se definirán las reglas de calidad y se ejecutarán las mismas.
- Ejecución de calidad: las reglas definidas deben ser ejecutadas en las distintas fuentes de datos, en base a los resultados obtenidos se deberán diseñar acciones correctivas.
- Implementación de acciones correctivas: se deben desarrollar activades o procedimientos reformatorios para cuando los resultados no sean satisfactorios. También se implementan pruebas de aceptación por parte del Pruduct Owner para determinar si acepta o no el software.
- Desarrollo del modelo de machine Learning: se debe utilizar patrones de detección de anomalías para desarrollar un modelo que permita encontrar las mismas y generar las alertas. Además, se deben ejecutar las acciones correctivas desarrolladas en la etapa anterior.

- MLOPS: se debe definir el modelo de MLOPS que se implementará en la plataforma una vez finalizada la PPS.

1.4 Cronograma de trabajo

Tareas	Sprint						
	N° 1	N° 2	N° 3	N° 4	N° 5	N° 6	N° 7
Configuración y Alcance							
Definición del Scope y Alcance							
Definición de estructura de BD							
Configurar Entornos (Repositorios, Accesos, Usuarios)							
Mockup UX							
Desarrollo de frontend							
Entendimiento de Customer Experience							
Definición del value stream							
Desarrollo del Backend							
Creación de tablas en BD							
Desarrollo del core del aplicativo: definición de reglas de calidad.							
Ejecución de reglas							
Quality execution							
Ejecución de reglas contra la fuente de datos							
Obtención de resultados							
Diseño de acciones correctivas							
Implementación de acciones correctivas							
Visualización de resultados							

Desarrollo de actividades correctivas							
Pruebas de aceptación							
Desarrollo del modelo de machine Learning							
Investigación de patrones de anomalías							
Desarrollo del modelo de anomalías							
ejecución de acciones correctivas							
MLOPS							
Definición del modelo MLOPS a futuro							
Conclusión							
Conclusiones sobre el proyecto							
Redacción de informe final PPS							

1.5 Conceptos generales

Una dimensión de calidad de datos es un conjunto de atributos que representan un aspecto de calidad a alto nivel. Los principales son: Exactitud (Accuracy), Completitud (Completeness), Frescura (Freshness), Consistencia (Consistency) y Unicidad (Uniqueness). Cada una de estas dimensiones, tienen aspectos diferentes los cuales se miden de diferentes maneras. En otras palabras, un factor representa un aspecto particular de una dimensión de calidad, y cada uno de estos, se puede medir de diferente manera. Por ejemplo, la exactitud se puede medir a través de la correctitud sintáctica o de la precisión y a su vez, dependiendo del tipo de dato para analizar, ambos factores se medirán a través de métricas y métodos distintos.

Para decidir de qué forma medir un factor de calidad, es decir, qué aspecto de una dimensión es necesario mensurar y con qué tipo de granularidad (columna, celda, tupla, tabla o base de datos) se utiliza una métrica como instrumento.

Cuando ya se la tiene determinada, se crea un método de medición que la pone en marcha y así se obtiene un valor cualitativo. El siguiente paso es generar una métrica instanciada; esta asocia la métrica a aplicar y el método a un modelo de datos determinado. También en este momento se define la granularidad de la medición.

Para el presente proyecto se tomará una columna, la que se relacionará con el método de medición que se debe utilizar. De esta manera se obtiene el siguiente diagrama de jerarquía:

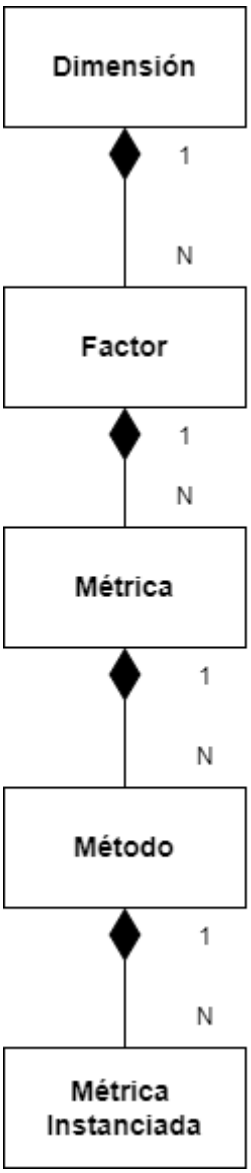

Figura 1- Diagrama de Jerarquías
Fuente: Elaboración propia basada en la práctica

Una vez que se establece qué se quiere medir, dónde, cómo y con qué, se deben crear reglas de calidad que asocien una métrica instanciada a valores que debe cumplir el dato. Puede ser que una regla necesite ninguno, uno o más argumentos para realizar la medición; esto dependerá del método que ponga en funcionamiento la métrica instanciada.

Cuando se ejecuten aquellas reglas (paso 1 de la figura 3), se deben tomar acciones en función de los resultados obtenidos que devuelvan las bases de datos analizadas (paso 2). Para ello, cuando estas encuentren datos que no

cumplan con las condiciones definidas y esperadas, se generarán y almacenarán alertas (paso 3) para indicar los problemas presentes en los datos. En caso de que haya una alerta anterior, se verifica el estado de la misma, si está cerrada, se abre una nueva, si sigue abierta, entonces no se realizan acciones.

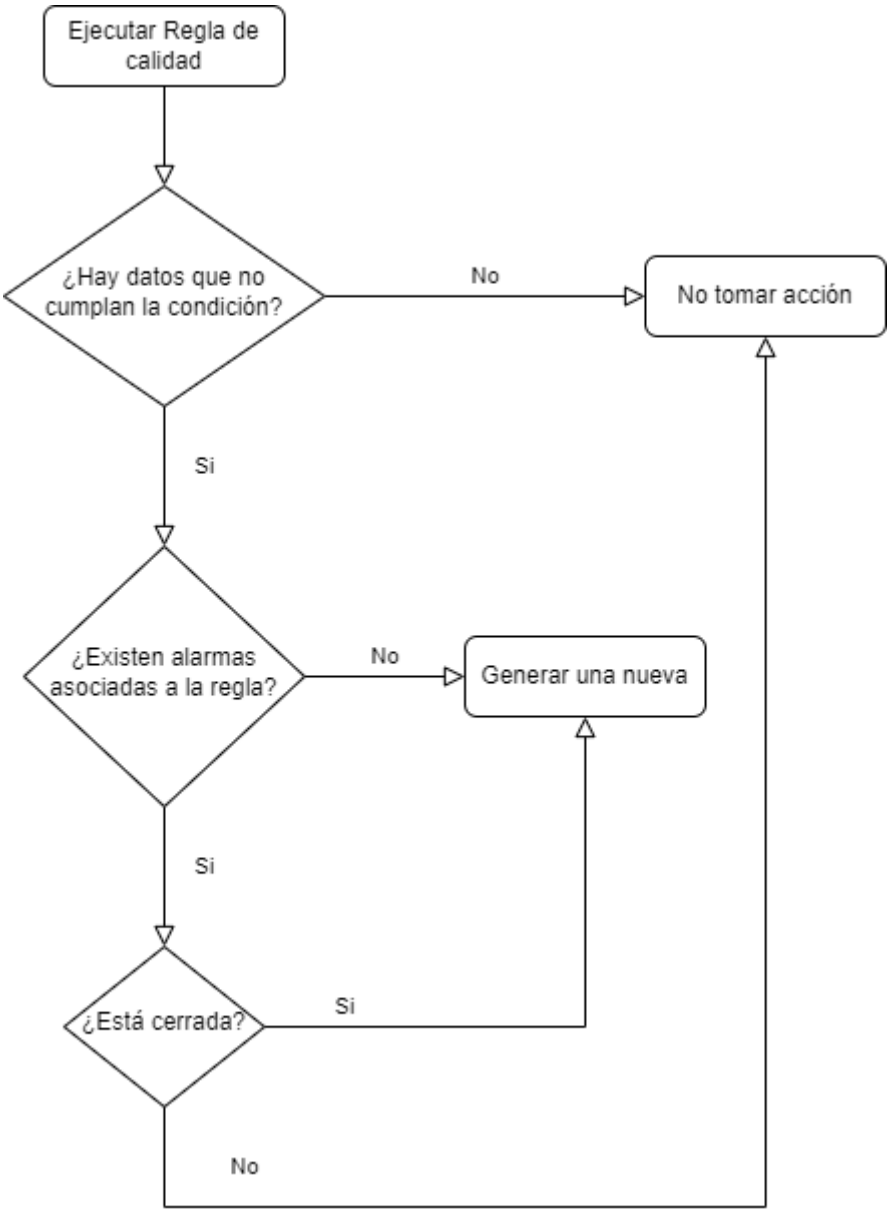

Figura 2- Diagrama de flujo: generación de alertas
Fuente: Elaboración propia basada en la práctica

La información sobre estas alertas es la metadata del sistema y es la que permite realizar un seguimiento y mantener un historial de la calidad de los datos.

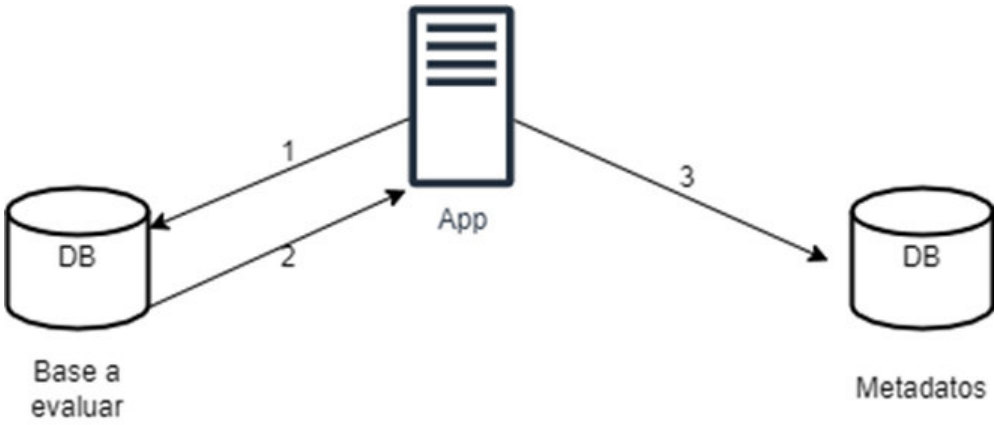

Figura 3- Generación de metadata
Fuente: Elaboración propia basada en la práctica

2. Desarrollo

2.1 Tecnologías utilizadas

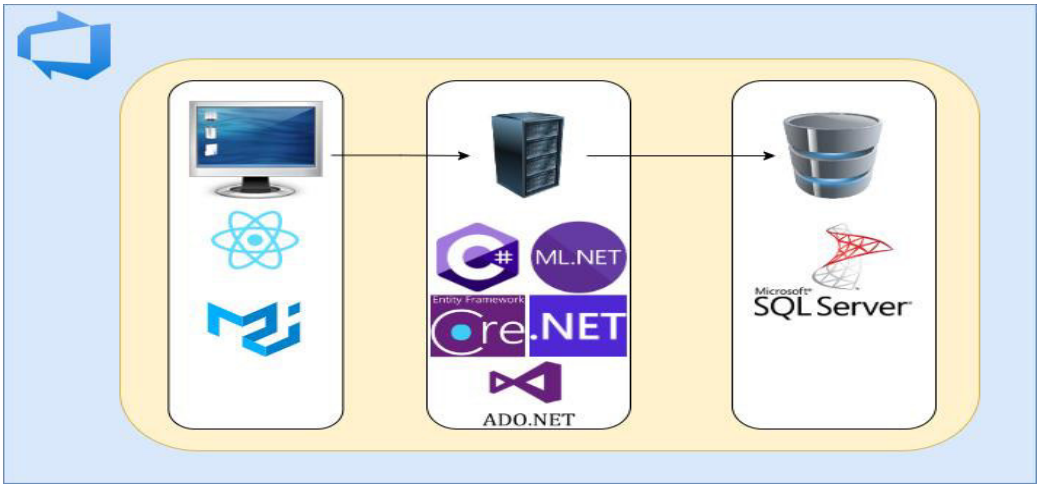

Figura 4 -Tecnologías utilizadas en cada parte de la aplicación
Fuente: Elaboración propia basada en la práctica

2.1.1 Lenguaje C#

Figura 5 - Logo Lenguaje de Programación C#
Fuente: https://cdnlogo.com/logo/c_760.html

Es un lenguaje de programación multiparadigma, simple, moderno, orientado a objetos y con seguridad de tipos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

C# fue desarrollado tomando lo mejor de los lenguajes C y C++, a la vez que se le agregaron funcionalidades de Java que permiten tener una sintaxis más avanzada que sus antecesores. Esto permite que el trabajar con este lenguaje sea familiar para los programadores de C, C++, Java.

2.1.2 Net 5

Figura 6 - Logo Framework Net 5
Fuente: https://es.m.wikipedia.org/wiki/Archivo:.NET_Logo.svg

Se trata de un marco de trabajo (framework) para el desarrollo de software que fue lanzado por Microsoft con la finalidad de fusionar su amplio catálogo de productos, que va desde sus múltiples sistemas operativos hasta herramientas de desarrollo. Al tratarse de una plataforma de propósito general, se pueden realizar tanto desarrollos web, como programas de escritorio o aplicaciones para dispositivos móviles. Además, favorece el desarrollo en multiplataforma, como por ejemplo el que una misma aplicación pueda correr indistintamente en los diferentes sistemas operativos móviles como IOS o Android garantizando la comunicación entre los diferentes dispositivos. Al mismo tiempo, se pueden escribir aplicaciones .NET en lenguajes como C#, F# o Visual Basic.

2.1.3 Microsoft SQL Server

Figura 7 - Logo Sistema de gestión SQL Server
Fuente: https://www.abd.es/soluciones/infraestructura-red/sql-server-logo/

Sistema de gestión de base de datos relacional, desarrollado por la empresa Microsoft que permite almacenar y posteriormente acceder a los datos de forma rápida y estructurada. El mismo está desarrollado como un servidor que para otras aplicaciones.

Al igual que otros programas RDBMS, éste se basa en SQL, un lenguaje de programación estandarizado que los administradores de bases de datos (DBA) y otros profesionales utilizan para gestionar las bases de datos y consultar los datos que contienen. SQL Server está vinculado a Transact-SQL (T-SQL), una implementación de SQL de Microsoft que añade un conjunto de extensiones de programación propias al lenguaje estándar.

El componente principal de Microsoft SQL Server es el motor de base de datos, que controla el almacenamiento, procesamiento y seguridad de los datos. Incluye un motor relacional que procesa los comandos y las consultas; y un motor de almacenamiento que gestiona los archivos de la base de datos, las tablas, las páginas, los índices, los búferes de datos y las transacciones. El motor de base de datos también crea y ejecuta procedimientos almacenados, desencadenantes, vistas y otros objetos de la base de datos.

2.1.4 ADO.NET

Figura 8 - Logo ADO.NET
Fuente: https://raima.com/es/how-to-create-a-database-using-ado-net/

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para programadores de .NET Framework. Proporciona un amplio conjunto de componentes para crear aplicaciones distribuidas de intercambio de datos. Es una parte integral de .NET, que brinda acceso a datos relacionales, XML y de aplicaciones. Es decir, incluye proveedores de datos para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente, se colocan en un objeto DataSet de ADO.NET para exponerlos al usuario de manera sencilla, se combinan con datos de múltiples fuentes o se pasan entre niveles.

2.1.5 Entity Framework Core

Figura 9 - Logo Entity Framework Core
Fuente: https://www.fixedbuffer.com/entity-framework-core-3-0-que-novedades-nos-trae

Entity Framework (EF) es un marco de mapeo de objetos / relaciones (ORM). Es un tipo de biblioteca de acceso a datos que intenta hacer que esta tarea sea más natural para los desarrolladores. Así, a la hora de acceder a datos, en lugar de utilizar otro lenguaje (generalmente SQL), un ORM permite que se pueda utilizar los paradigmas habituales de la programación orientada a objetos: clases y objetos. En lugar de pensar en tablas y relaciones, se piensa en objetos y propiedades. Es una mejora de ADO.NET que brinda a los desarrolladores un

mecanismo automatizado para acceder y almacenar los datos en la base de datos.

Entity Framework Core es la nueva versión de EF después de EF 6.x. Es una versión de código abierto, ligera, extensible y multiplataforma de la tecnología de acceso a datos de Entity Framework.

2.1.6 React

Figura 10 - Logo React
Fuente: https://es.wikipedia.org/wiki/React#/media/Archivo:React.svg

Biblioteca JavaScript es de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. Esta tecnología permite el desarrollo de interfaces de usuario de forma sencilla, esto es posible mediante componentes interactivos y reutilizables.

React está basado en un paradigma llamado programación orientada a componentes en el que cada componente es una pieza con la que el usuario puede interactuar. Estas se crean usando una sintaxis llamada JSX permitiendo escribir HTML (y opcionalmente CSS) dentro de objetos JavaScript.

Dichos componentes son reutilizables y se combinan para crear otros mayores hasta configurar una web completa.

Esta es la forma de tener HTML con toda la funcionalidad de JavaScript y el estilo gráfico de CSS centralizado y listo para ser abstraído y usado en cualquier otro proyecto.

2.1.7 MUI

Figura 11 - Logo Material UI
Fuente: https://worldvectorlogo.com/es/logo/material-ui-1

Es una biblioteca robusta, personalizable y accesible de componentes React básicos y avanzados listos para ser utilizados (tablas, formularios, ventanas, botones, etc.) lo que permite desarrollar aplicaciones React más rápido al ahorrar código y tiempos de desarrollo. Las principales ventajas que tiene esta librería son:

- Sus componentes prearmados funcionan de forma aislada. Es decir, no interfieren en la configuración de otros.
- MUI no depende de otras librerías. Ya que no se basa en ninguna hoja de estilo global (como CSS).
- Tiene una fácil personalización. Ya que se pueden modificar el procesamiento de cada componente hasta la última clase.

Además, su documentación ofrece una página de demostración individual para probar cada componente en (y así definir si conviene o no implementarlos).

2.1.8 Azure DevOps

Figura 12 - Logo Azure Devops
Fuente: https://actualizatech.com/cartflows_step/azure-devops-2/

Azure DevOps es un conjunto de herramientas y servicios que permite que los equipos planifiquen el trabajo, colaboren en el desarrollo de código y creen y pongan en funcionamiento aplicaciones ayudando así en la administración del ciclo de vida de un proyecto de software. Admite una cultura

colaborativa y un conjunto de procesos que reúnen a programadores, administradores de proyectos y colaboradores para realizar sistemas. También permite a las organizaciones crear y mejorar productos a un ritmo más rápido que con los enfoques tradicionales de desarrollo. Además, soporta cualquier lenguaje y plataforma de programación. Con lo que cualquier programador, sea cual sea su lenguaje, puede utilizarla.

Algunos de los servicios utilizados son:

- Azure Repos: aporta repositorios Git o Team Foundation Version Control (TFVC) para el control de código fuente.
- Azure Pipelines: proporciona servicios de compilación y lanzamiento para admitir la integración y la entrega continuas de las aplicaciones.
- Azure Boards: ofrece un conjunto de herramientas Agile para respaldar la planificación y el seguimiento del trabajo, los defectos del código y los problemas mediante los métodos Kanban y Scrum.
- Azure Test Plans: facilita varias herramientas para probar aplicaciones, incluidas pruebas manuales/exploratorias y pruebas continuas.
- Azure Artifacts: permite a los equipos compartir paquetes como Maven, npm, NuGet y otros tanto de fuentes públicas como privadas e integrar el uso compartido de paquetes en los pipelines.

2.1.9 ML.Net

Figura 13 -Logo ML.NET
Fuente: https://en.wikipedia.org/wiki/ML.NET#/media/File:Mldotnet.svg

ML.NET es un marco de aprendizaje automático gratuito, de código abierto y multiplataforma para la plataforma de desarrollo .NET. Permite entrenar, crear y enviar modelos de aprendizaje automático personalizados mediante C# para una variedad de escenarios de Machine Learning. También

incluye funciones como el aprendizaje automático automatizado (AutoML) y herramientas como ML.NET CLI y ML.NET Model Builder (no utilizadas en este proyecto ya que todavía están en desarrollo), que facilitan la integración del aprendizaje automático en una aplicación.

2.1.10 Elección de las tecnologías

Si bien el proyecto podría ser realizado con otras tecnologías igual de eficaces que las utilizadas, el cliente eligió que se trabajará con un ecosistema de aplicaciones (stack) conformado por: C#, SQL Server, React y Devops por dos motivos:

- Todo el equipo de desarrollo conoce y trabaja con ellas. Esto posibilita que en el futuro el mantenimiento, la escalabilidad y nuevos evolutivos sean fáciles de llevar a cabo sin la necesidad de contratar nuevos recursos.
- Con estas herramientas se programó la mayoría de las aplicaciones que tiene en uso TEIC y en todas se cumplió el rendimiento y seguridad esperado por el cliente.

También, se utilizó Material UI frente a otras alternativas como Ant Design o Grommet por diferentes motivos:

- Posee la mayoría de los componentes que son necesarios para la aplicación y permite una personalización de los mismos sencilla.
- Es fácil agregar estilos CSS adicionales en estos.
- Ofrece la posibilidad de probar y personalizar los componentes en un ambiente individual utilizando la plataforma Sandbox para luego agregarlos al proyecto.
- Es muy popular y hay muchos foros donde ya se respondieron o resolvieron problemas típicos de la implementación de algunos componentes.

Además, se optó por NET 5 y no la versión 6 por dos motivos:

- Fue liberada a fines del 2021 y disponible en modo de prueba, por lo tanto, alguna funcionalidad nueva podría no funcionar correctamente.

- Net 5 se lanzó a finales del 2020, entonces, al momento de resolver un problema o error, hay mucha más información y respuestas de esta versión.

Respecto a Entity Framework Core, se eligió a este frente a otros ORM (como por ejemplo EF6) por lo siguiente:

- Es el recomendado por Microsoft y en la mayoría de las documentaciones oficiales, foros y tutoriales se utiliza este.
- Tiene nuevas características que no se implementaron en EF6.
- Es más sencilla y modular a la vez que tiene mayor rendimiento y consume menos recursos.
- Es una plataforma diseñada para .NET.

Por último, si bien ML.NET actualmente es un framework con algoritmos más básicos, limitados y menos eficientes que los desarrollados para Python o R y hay poca información en foros, se lo utilizó porque al ser desarrollado para .Net es totalmente compatible; lo que facilita una rápida implementación y, además, se liberan y mejoran más algoritmos a medida que surgen nuevas versiones.

2.2 Metodología Scrum

Es un proceso para desarrollar software incrementalmente en entornos complejos donde los requisitos no están claros o cambian con mucha frecuencia. Por lo tanto, un equipo que lo implementa puede realizar entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan a los interesados del proyecto o por la urgencia y prioridad que ellos le planteen al equipo. Esto hace que Scrum sea muy usado en la actualidad porque tiene características que encajan con el desarrollo de software y las nuevas formas de gestionar las empresas. Este es menos burocrático y está más orientado a la productividad, ya que le quita importancia a la documentación de los proyectos.

Los principales beneficios que aporta aplicarlo son:

- Comunicación
- Trabajo en equipo

- Flexibilidad
- Proveer software funcionando de manera incremental

Debido a la organización y metodología de trabajo establecida y llevada a cabo por Softtek y TEIC, los desarrollos de software se llevan a cabo con una adaptación de Scrum. Por lo tanto, algunos de los componentes que intervienen en el proyecto son:

- Backlog: es una pizarra de tareas, con las funcionalidades y errores a desarrollar o corregir durante cada intervalo de trabajo. Hay múltiples aplicaciones que realizan esta función. Para este proyecto se utiliza el tablero que ofrece Azure Boards.
- Equipos de desarrollo: además del programador encargado de realizar la aplicación, se cuenta con la participación de un arquitecto de software, quien trabaja de manera cross en los demás desarrollos que le brinda Softtek al cliente y siempre está disponible para realizar propias de la definición y diseño de alto nivel de la arquitectura del sistema. También hay dos referentes de tecnología, uno por el lado del Backend (desarrollo de la API) y otro por el Frontend. El trabajo de ellos es realizar una revisión de código a partir de cada funcionalidad nueva y asesorar al desarrollador en consultas técnicas que este tenga. También será labor de este último, reunirse con el cliente para realizar la planificación de las próximas mejoras, funcionalidades o errores o corregir.
- Product Owner (PO): Representante del cliente que usará el software. Se concentra en la parte de negocio y él es responsable del ROI del proyecto (entregar un valor superior al dinero invertido). Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las prioriza de forma regular.
- Sprints: son intervalos de trabajo, los cuales serán de dos semanas y los mismos contarán con diferentes etapas, en el que el resultado será un incremento o mejora al desarrollo hecho. Por otro lado, al comienzo de cada uno, se realizará una reunión para el planeamiento de las tareas a ejecutar durante el sprint. Con ello, se completará el backlog. Para la selección de las tareas, se tendrá fuertemente en cuenta la decisión y

opinión de los interesados en la implementación del proyecto (stakeholders).

- Reuniones de revisión: al final de cada sprint, se deben analizar los resultados obtenidos, puntos, tareas o actitudes a mejorar o llevar a cabo. De este modo, se tendrán en cuenta las acciones que se deberán mantener debido a los resultados positivos e indagar qué se debe incorporar o adoptar para la siguiente iteración. El resultado de todo este trabajo es una release, es decir, un producto incremental que puede ser mostrado al cliente y usuario final con sus respectivas mejoras y funcionalidades nuevas.

3 Metodología de desarrollo

Como se explicó en el punto anterior, al comienzo de cada sprint se definirán las tareas a realizar durante el mismo, dándole un valor basado en esfuerzo a cada una. En ese momento se podrán asignar las mismas o ir tomándolas durante el transcurso. Existe también, bajo esta metodología, la posibilidad de refinar los requerimientos para tener una mejor definición de estos y facilitar su desarrollo. Por ejemplo, una reunión de refinamiento contará, mínimamente, con un representante comercial y un desarrollador, donde se intentará definir con el mayor detalle posible un requerimiento o funcionalidad.

3.1 Gestión de ramas

En Git, las ramas son espacios o entornos independientes que un desarrollador utiliza para trabajar de forma autónoma con otros. Esto le permite trabajar sobre un proyecto sin modificar o borrar el conjunto de archivos originales de este, dando así flexibilidad a un equipo para desarrollar un sistema de manera más organizada y segura.

Para administrar la gestión de las ramas del repositorio, en Azure DevOps Repos se definió la siguiente estrategia:

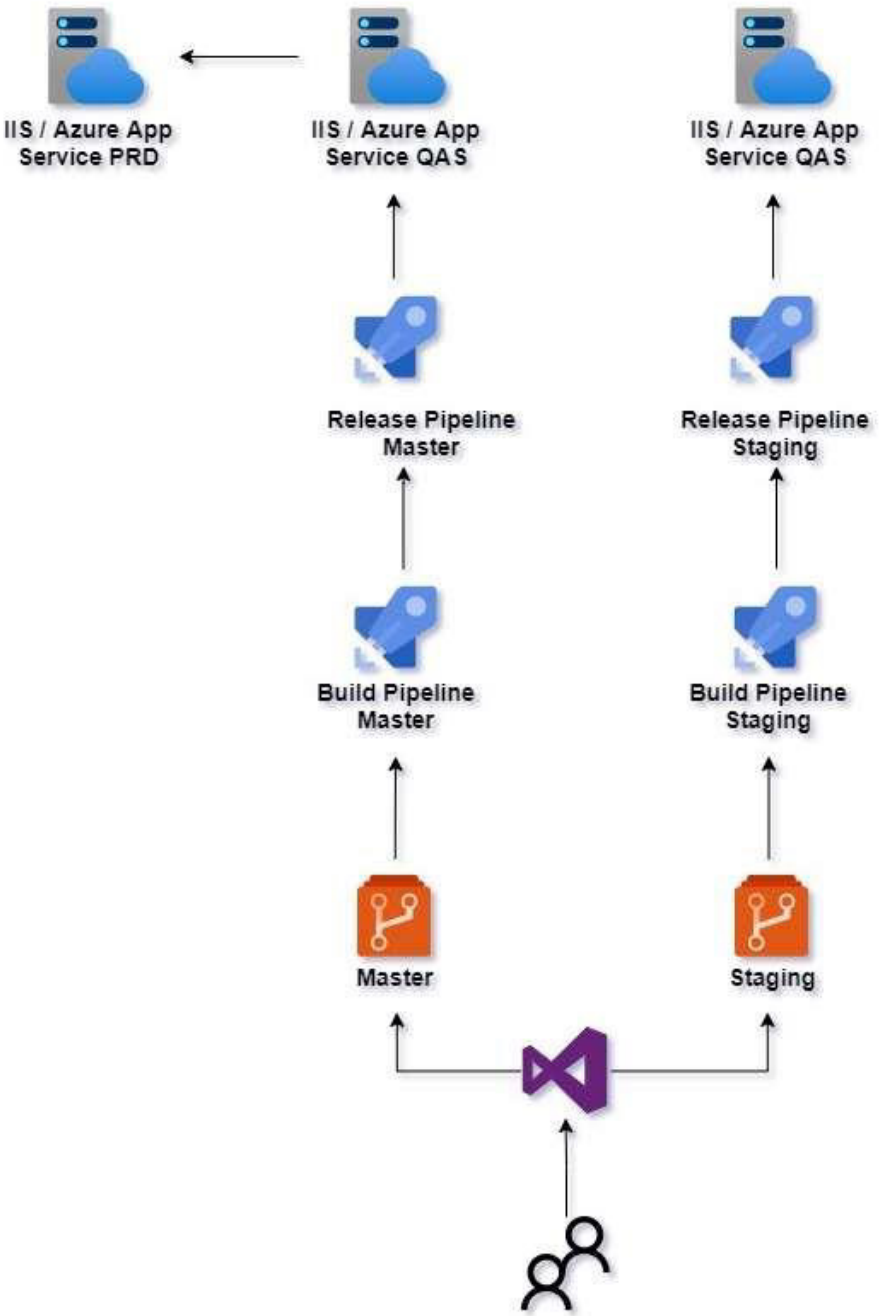

Figura 14- Ramas Master y Staging
Fuente: Elaboración propia basada en la práctica

Con esta configuración, se obtienen dos ventajas: que los evolutivos se implementen sobre la rama Staging que asegura que ningún error permitiría que esos cambios vayan directamente al entorno productivo, ya que la rama tiene impacto solo hasta QAS. Y, además, en caso de necesitar corregir un error (bug) en el código, este puede realizarse en una rama basada en Master y, una vez

llevada a cabo, impactar los cambios sobre esta sin importar si hay evolutivos en desarrollo ya que los mismos, van por otra rama y, por lo tanto, no se pierde el avance realizado.

Una vez definido qué tareas o requerimientos tomar, se creará una rama a partir de Master con la nomenclatura {Evo}_{nombredelatarea} por ejemplo: Evo_VistaAlarmas. En la misma se comenzará con la programación, despejando dudas de negocio con el Product Owner, y las técnicas, con el arquitecto o los referentes de tecnología del equipo.

Una vez desarrollada y testeada la funcionalidad correspondiente en el entorno local del desarrollador, se subirá la rama utilizada (con sus cambios correspondientes) al repositorio de Azure y se solicitará al propietario de la rama Staging que ésta incorpore dichos cambios (pull request). Dependiendo del impacto, tamaño e importancia del evolutivo, dicha petición puede ser analizada por uno o varios referentes de tecnología del equipo, en lo que se denomina 'una revisión de código'. La misma sirve para analizar posibles mejoras, sugerencias en el código desarrollado de manera individual, verificar que se cumplan las buenas prácticas y, por último, garantizar que se tenga una doble validación de lo que se pone en funcionamiento.

3.2 Pipeline de CI/CD

Un pipeline de CI / CD es el conjunto completo de procesos que se ejecutan cuando se realiza un pull request en un proyecto. Las canalizaciones abarcan sus flujos de trabajo, que coordinan sus trabajos, y todo esto se define en el archivo de configuración del proyecto.

En Azure, un pipeline combina la integración continua (CI) y la entrega continua (CD) para probar y compilar el código y enviarlo a cualquier destino (entorno productivo, de pruebas, etc.).

La CI es la práctica utilizada por los equipos de desarrollo para automatizar la fusión y el código de prueba. La implementación de CI ayuda a detectar errores al principio del ciclo de desarrollo, lo que los hace menos costosos de corregir. Las pruebas automatizadas se ejecutan como parte del

proceso de CI para garantizar la calidad. Los artefactos se producen a partir de sistemas de CI y se alimentan de procesos de lanzamiento para impulsar implementaciones frecuentes.

La entrega continua (CD) es un proceso mediante el cual el código se crea, prueba e instala en uno o más entornos de prueba y producción. La implementación y las pruebas en múltiples entornos aumentan la calidad. Los sistemas de CI producen artefactos (artifacts) desplegables, incluidas la infraestructura y las aplicaciones. Los procesos de lanzamiento automatizados consumen estos artefactos para lanzar nuevas versiones y correcciones a los sistemas existentes. Los sistemas de monitoreo y alerta se ejecutan continuamente para impulsar la visibilidad de todo el proceso de CD.

3.3 Implementación de cambios

Luego de ser aprobada cada solicitud, se construye automáticamente un pipeline. Una vez que termine la construcción del mismo, este es liberado en el entorno de pruebas (QAS) sin solicitar aprobación manual y cuando su ejecución termina, los cambios pueden verse en este ambiente.

Cuando el código nuevo se encuentre en QAS, el Product Owner procederá a probar los casos de uso que crea correspondiente, pudiendo realizar pruebas punta a punta o los casos particulares (al no contar con un equipo de testing serán fundamentales las pruebas hechas por los desarrolladores). Una vez que se obtiene el visto bueno por parte de éste, el desarrollador deberá preparar el pasaje para el entorno productivo. Para ello, debe actualizar los cambios de su rama local Staging (realizando un pull en su equipo). Una vez hecho esto, debe volver a subir los cambios de la misma (realizar un push de estos), hacer un pull request a la rama Master. Nuevamente se creará otro pipeline y luego, se lo liberará y otra vez se detendrá en QAS. Para pasar estos cambios a PRD, se deberá enviar un correo electrónico al arquitecto de software del equipo, adjuntando la autorización del Product Owner y estableciendo el horario para realizar el pasaje.

Cuando se necesite realizar una corrección de un error (fix de un bug), los cambios se deben realizar sobre la rama master; así, el pasaje se realiza con las autorizaciones nombradas anteriormente, pero en menor tiempo.

4 Estructura de la aplicación

Al usar la plantilla de “Aplicación Asp.Net Core con React”, se genera una estructura por defecto para los directorios del proyecto, que permite facilitar la conexión entre el Backend y el Frontend. Sin embargo, para una mejor escalabilidad y mantenibilidad en el código, se debió agregar más carpetas y una capa más.

La Capa DataQuality.Web es la generada por el proyecto que contiene los directorios principales que le dan funcionalidad al sistema. Mientras que la de DataQuality.DataAccess es la que tiene la configuración y migración de la base de datos.

En la siguiente figura se observa la configuración final de directorios y archivos.

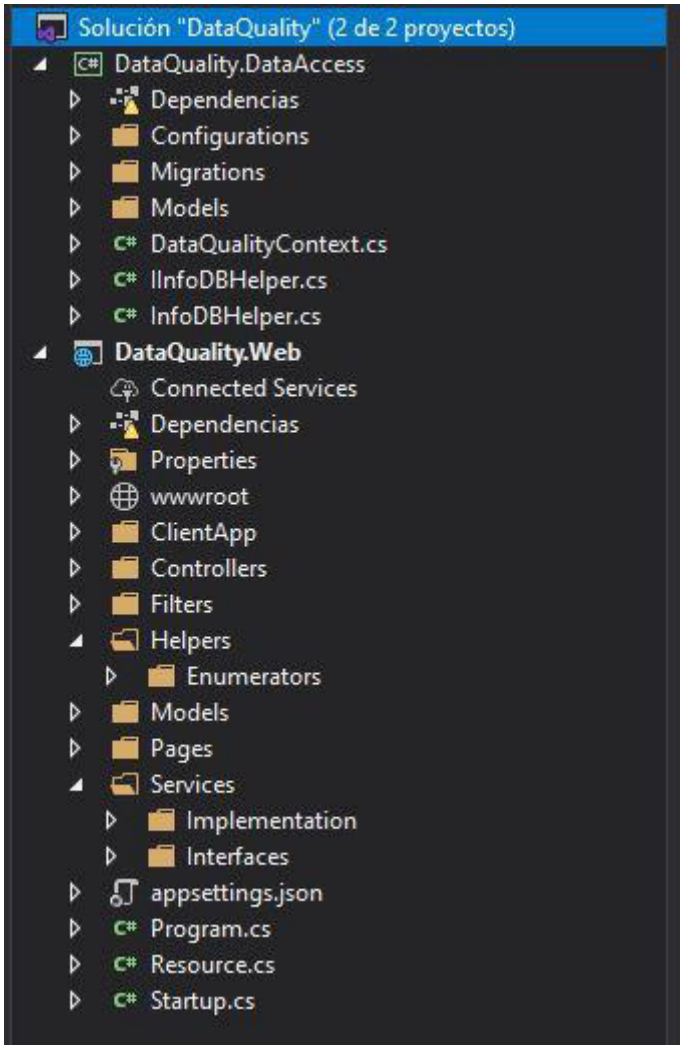

Figura 15- Estructura de la aplicación
Fuente: Elaboración propia basada en la práctica

4.1 Backend

Los directorios y archivos dentro de la Web son:

- wwwroot: contiene archivos estáticos que son necesarios en tiempo de ejecución. Estos pueden ser: imágenes, bundles de scripts y CSS, HTML, etc.
- Controllers: contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, ejecutar una regla de calidad, búsqueda de información, etc. Su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino

servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

- Filters: engloba archivos que permiten filtrar las excepciones, registrarlas (loguearlas) y notificar de las mismas por correo a los desarrolladores y personalizar los mensajes de error en pantalla para los usuarios.
- Helpers: contiene clases que ayudan o simplifican diferentes procesos de una aplicación. Estos pueden ser de diferente tipo y funcionalidad, por ejemplo, resolver la autenticación de un usuario o, como en el caso de los Enumerators, que permiten asignar valores fijos (constantes) que serán utilizados en la aplicación y estos se representan en el sistema como un número entero almacenado en la BD, de esta manera, se simplifica el diseño de esta, al evitar crear campos para una descripción asociada a dicho valor.
- Models: tiene los DTOs que son clases sencillas que cuentan con un conjunto de propiedades que permiten almacenar datos. Estos archivos permiten:

    Reducir la cantidad de información que se transfieren entre capas.
    Múltiples operaciones para extraer la información.
    Adaptar los datos a una salida específica.
- Pages: es generada por la plantilla y trae vistas para cuando ocurren errores al ejecutar localmente (en la computadora del desarrollador) la aplicación.
- Services, contiene dos subdirectorios. Interfaces donde se definen las mismas y sus métodos e Implementation donde se implementan las mismas. Estas interfaces permiten definir un "contrato del comportamiento" sobre el que se puede estar seguros de que, las clases que las instrumenten lo van a cumplir. Mientras que, en sus correspondientes implementaciones, se rellenan cada uno de los métodos definidos anteriormente. Es decir, en las interfaces se escriben la firma de los métodos de una clase. Y en cada uno de los servicios que utilicen dichas interfaces se rellenan los métodos con las funcionalidades necesarias.

- .appSettings.json: almacena información de configuración de la aplicación personalizada, como cadenas de conexión de base de datos, rutas de acceso de archivo, direcciones URL de servicio Web XML o cualquier otra información de configuración personalizada para una aplicación.
- Program.cs: se define el código de inicio de la aplicación, por lo tanto, se configuran los servicios requeridos por la aplicación.
- Statup.cs: su objetivo en los proyectos de ASP.NET Core es configurar los servicios que se van a utilizar en la aplicación, así como la serie de objetos de tipo middleware que conformarán el pipeline de ejecución.

Los directorios y archivos dentro de acceso de datos (Data Access) son:

- Configurations: contiene una clase por cada tabla de la base de datos. En cada uno de estos, se definen las relaciones entre las tablas y se establecen las características de los campos de las mismas. Se separan por clase para que la configuración y mantenimiento sea más sencillo al quitarle complejidad a DataQualtityContext.
- Migrations: contiene clases que son generadas automáticamente por Entity Framework cada vez que se realiza una migración en la base de datos.
- Models: contiene las clases que son modeladas en la aplicación (factores, alertas, reglas de calidad, etc). En cada una de ellas se establecen de qué tipo de datos serán las propiedades de cada una.
- DataQualtityContext.cs: esta clase, perteneciente al paquete de EntityFrameWorkCore, permite entre otras cosas configurar las entidades (tablas) de la base de datos, configurar los índices compuestos de las tablas y acceder a los datos de dichas tablas mediante código C#.
- IInfoDBHelper.cs es una interfaz donde se definen todas las funcionalidades que va implementar InfoDBHelper
- InfoDBHelper.cs: es una clase ayuda y simplifica todos los procesos de conexión entre los servicios y las bases de datos mediante ADO.NET. Para ello implementa los métodos definidos en la interfaz anterior.

4.2 Frontend

Todos los directorios que dan funcionalidad al mismo están dentro de ClientApp. Los mismos son:

- Node_Modules: directorio que se crea en la carpeta raíz de un proyecto en React (en este caso, en la carpeta ClientApp) cuando se instalan paquetes o dependencias mediante npm. De esta forma, desde el código Javascript se puede importar paquetes externos instalados mediante npm, teniéndolos en el proyecto local (no dependen de una ruta externa al proyecto) y sin necesidad de manipularlos manualmente.
- Public: contiene archivos estáticos como index.html, archivos de biblioteca JavaScript, imágenes y otros activos, etc. que no se desea que sean procesados por el webpack el cual es una herramienta de compilación que permite añadir en un archivo todas las dependencias a los elementos que forman parte del proyecto de desarrollo. Los archivos de esta carpeta se copian y pegan ya que están directamente en la carpeta de compilación. Solo los archivos dentro de ella pueden ser referenciados desde el HTML.
- .env: React permite configurar variables de entorno en este fichero y, gracias a ello, se puede disponer de estos valores en todo el proyecto de una forma sencilla.
- .env.production: permite disponer de sobreescritura de dichas variables para el entorno de producción.
- .gitignore: archivo de texto que le dice a Git qué archivos o carpetas ignorar en un proyecto.
- Package.json: archivo con formato JSON dónde se definen información acerca del proyecto (nombre, versión, etc.) además de listar los paquetes de los que depende (con sus respectivas versiones). De esa forma, centraliza todas las dependencias y facilita trabajar en equipo en un proyecto. Ya que permite que, al clonar el repositorio y ejecutar el comando npm install en la computadora de otro desarrollador, se descarguen automáticamente los paquetes más actualizados a la fecha.

- Package-lock.json: archivo similar al anterior. Evita ese comportamiento general de actualizar versiones; de modo que cuando alguien clona el repositorio y ejecuta npm install en su equipo, npm examinará package-lock.json e instalará la versión exacta de los paquetes que se habían instalado inicialmente.
- Readme.md: archivo generado por la plantilla. Guarda información acerca de otros archivos en un directorio, es una forma de documentación del programa, usualmente en un archivo de texto plano.
- Src: carpeta principal del frontend ya que albergará todos los componentes de React.
    - assets: carpeta pública que contiene archivos estáticos que, en este caso, no van a cambiar.tiene un subdirectorio llamado icons, que guarda los archivos necesarios para utilizar iconos en el proyecto.
    - components: directorio que se crea para guardar todos los componentes básicos que se utilizarán en otros más complejos. Por ejemplo, cartel de confirmación, menú simple, notificaciones, etc.
    - constants: aquí se almacenan archivos js que contienen variables de tipo texto con valores constantes que son utilizados en distintos componentes. De esta manera, cuando se quiere cambiar un valor, solo se cambia una vez.
    - Pages: aquí se guardan todas las vistas principales del proyecto y los componentes más grandes que estas utilizan (principalmente las tablas que éstas implementan). para mantener el orden también hay subdirectorios con el nombre de la vista.
    - store: aquí se almacenan archivos JS con las configuraciones de las peticiones a los diferentes controladores que tiene la API. También, se pueden guardar configuraciones extras para el comportamiento del estado de la aplicación.
    - styles:aquí se definen estilos a ser utilizados en casi toda la aplicación. Para ello, hay dos archivos:

- Layout.js: definen los estilos que implementan los componentes almacenados en la carpeta components
- material-icons.css: hace referencia a los iconos almacenados la carpeta assets/icons y le define estilos generales para que todos tengan el mismo aspecto

- Utilities: crea archivos JS que contienen funciones específicas que son utilizadas en diferentes componentes. Por ejemplo, dar formato hora y fecha, realizar validaciones, etc.
- App.css: contiene estilos específicos para App.jsx
- App.jsx: componente principal de la aplicación. Es el encargado de cargar a todos los demás y renderizarlos (mostrarlos) en la página principal index.html (la cual está en la carpeta public)
- index.css define estilos de apariencia específicos para la página anterior.
- index.js: tiene la configuración necesaria para que apps.jsx pueda renderizar correctamente los componentes en index.html

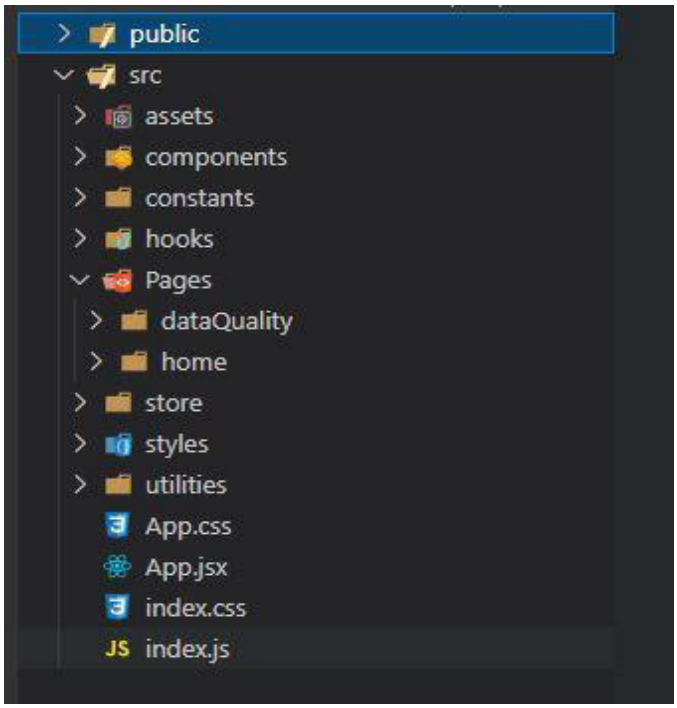

Figura 16- Estructura del frontend
Fuente: Elaboración propia basada en la práctica

5 Arquitectura

5.1 Flux

Flux es un patrón de diseño o forma de "arquitectura" de una aplicación web. En concreto, es la forma en la que se manejan los datos o estado de la aplicación (datos de un usuario, datos recogidos a partir de un API REST o webservice, etc.)

Flux propone una arquitectura en la que el flujo de datos es unidireccional. Es decir, los datos viajan desde la vista por medio de acciones y llegan a un Store desde el cual se actualizará la vista de nuevo. Esto se debe a que teniendo un único camino, y un sitio donde se almacena el estado de la aplicación, es más sencillo depurar errores y saber qué está pasando en cada momento.

Al igual que un patrón MVC está formado por un Modelo, una Vista y un Controlador. Pero, además hay otros actores:

- Vista: componentes web, pequeñas partes de código que se pueden mover y reutilizar en una página.
- Store: semejante al modelo de la aplicación. Guarda los datos/estado de la aplicación y en Flux puede haber varias. No hay métodos en el Store que permitan modificar los datos en ella, eso se hace a través de dispatchers y acciones.
- Acciones: una acción es simplemente un objeto JavaScript que indica una intención de realizar algo y que lleva datos asociados si es necesario.
- Dispatcher: las anteriores son enviadas por la vista a un dispatcher que se encarga de disparar o propagarlas hasta el Store. Un dispatcher no es más que un mediador entre el Store o Stores y las acciones. Sirve para desacoplar el store de la vista ya que así no es necesario conocer qué store maneja una acción concreta.

  En resumen, el flujo es el siguiente:

  1. La vista envía, mediante un evento, una acción con la intención de realizar un cambio en el estado

2. La acción contiene el tipo y los datos (si los hubiere) y es enviada al dispatcher.
3. El dispatcher propaga la acción al Store y se procesa en orden de llegada.
4. El Store recibe la acción y dependiendo del tipo recibido, actualiza el estado y notifica a las vistas de ese cambio.
5. La vista recibe la notificación y se actualiza con los cambios.

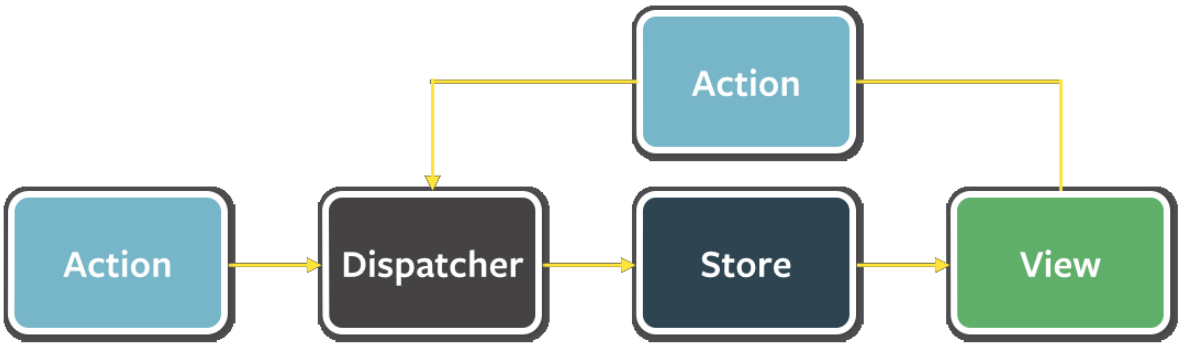

Figura 17- Flujo de datos utilizando Flux
Fuente: https://facebook.github.io/flux/docs/in-depth-overview

6 Modelo de la BD

En este proyecto, la API será montada sobre una base de datos SQL Server, con el siguiente diagrama de entidad – relación (forma visual de ver las relaciones entre las tablas)

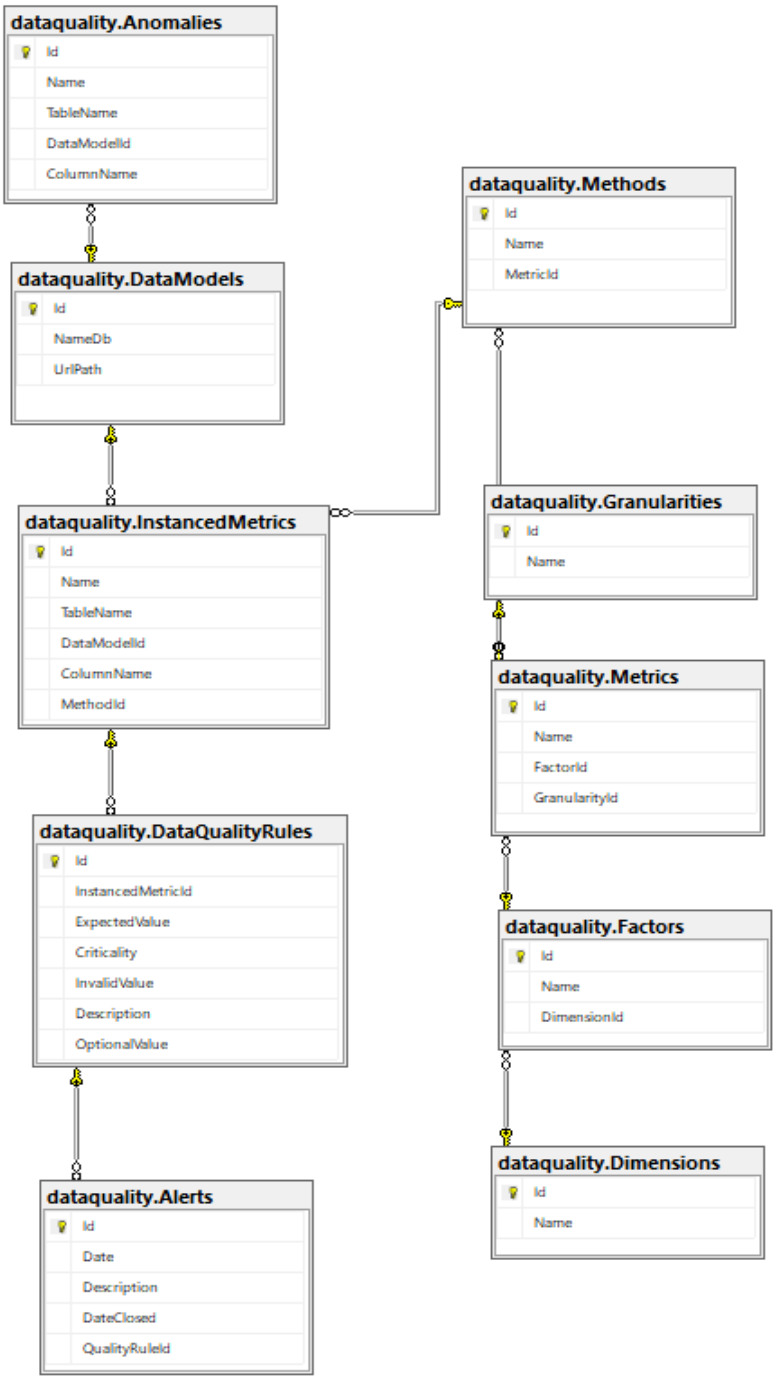

Figura 18- Diagrama de base de datos
Fuente: Elaboración propia basada en la práctica

- Instaced Metrics: entidad principal de la aplicación, la cual asigna un método de medición a un conjunto de datos (también llamado Data Model) que va a ser medible. Es decir, a una columna de una tabla de cualquier base de datos o Data Warehouse que el cliente esté interesado en medir

la calidad de sus datos. Para ello, se guarda un código único (Id) para identificar la métrica instanciada, un nombre (Name) y el id de identificación del método de medición (MethodId). Respecto al recurso medible, los mismos van a ser identificados a través de un DataModelId, a la vez que se guarda el nombre de la tabla (TableName) y nombre de la columna (ColumnName).

- DataModels: almacenará su código Id, el nombre de la Base de datos (NameDb) y la dirección del servidor que la contiene (UrlPath).
- Methods:, guarda un Id, un nombre (Name) y el Id de la métrica que va a utilizar (MetricId)
- Metrics: al igual que las anteriores, contiene un Id y un Nombre, a la vez que contiene el Id del factor de calidad que mide (FactorId) y el Id del tipo de granularidad con la que asociará la medida (para el alcance de este proyecto, la granularidad siempre será una columna).
- Factors: contiene un Id, un nombre y el Id de la dimensión a la que referencia (DimensionId).
- DataQualitRules: otra entidad principal ya que le asigna valores a una métrica instanciada. Para ello, guarda su Id, un valor esperado en el dato que mide (ExpectedValue), uno invalido (InvalidValue), uno opcional (OpcionalValue), el Id de la métrica instanciada que utiliza (InstancedMetricId), la criticidad de la regla (Criticality) y una descripción opcional (Description).
- Alerts: contiene su Id, la fecha en la que se generó (Date), una descripción que informa si está abierta o cerrada (Description), la fecha de cierre (DateClosed) y el Id de la regla de calidad asociada (QualityRuleId)
- Tanto Granularities como Dimensions tienen un Id y un nombre que las identifican.
- Anomalies: archiva los conjuntos de datos (llamado también Data Model) que van a ser analizados en búsqueda de puntos de cambios inesperados. Para ser identificadas, se guarda (Id) , un nombre y respecto al recurso analizable, los mismos van a ser identificados a través de un

DataModelId, a la vez que se conserva el nombre de la tabla (TableName) y nombre de la columna (ColumnName).

7 Vistas

7.1 Login

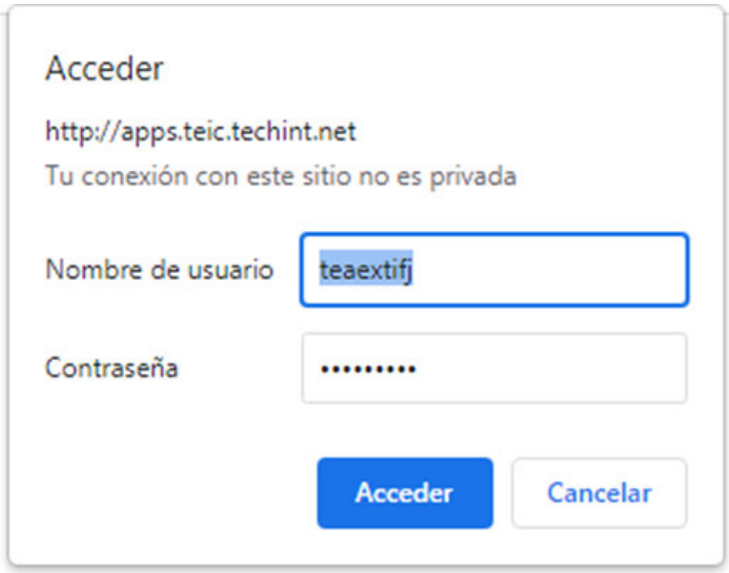

Figura 19- Pantalla de Login
Fuente: Aplicación en ambiente de producción

7.2 Barra de navegación

Como se muestra en la ilustración 20, las pantallas se agrupan con una forma similar a la de un árbol de nodos. La finalidad que busca no sólo es generar un orden sino también mejorar la experiencia del usuario al tener las tres pantallas más utilizadas con mejor accesibilidad.

Para ordenarlas, se eligió que las que estén relacionadas de forma directa a la medición estén juntas, siendo hijas del menú Medición (Measurement). Dado que Información General no tiene nada que ver con estas, se la ubicó al mismo nivel jerárquico que las vistas del tablero, las reglas y las alertas.

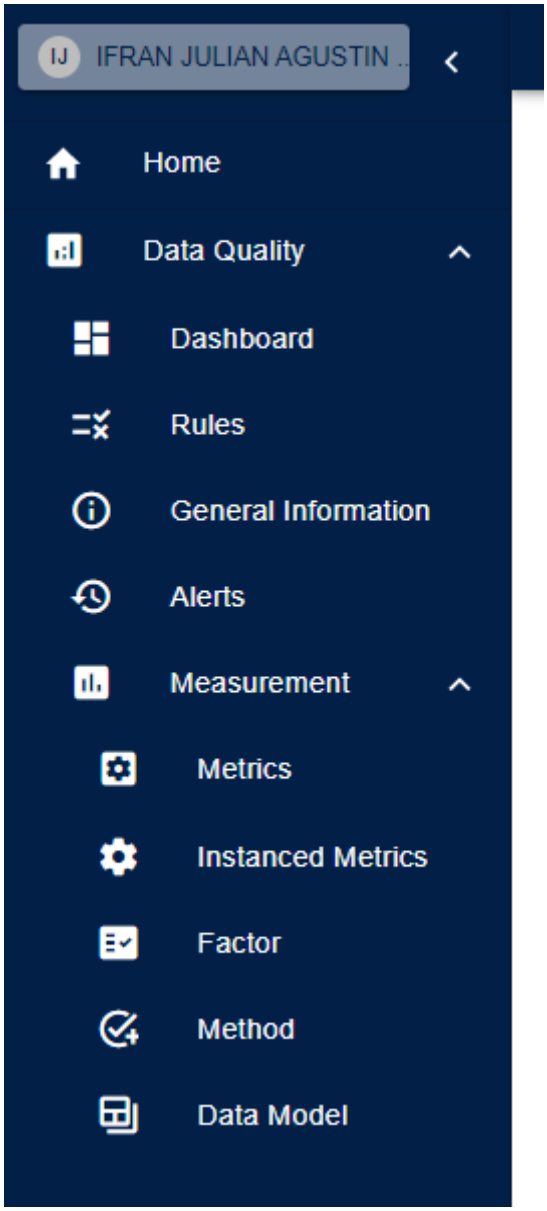

Figura 20- Barra de navegación
Fuente: Aplicación en ambiente de pruebas

7.3 Métricas instanciadas

Esta vista permite dar de alta, modificar o eliminar las métricas que se asignan a un recurso. Además, la tabla cuenta con botones que permiten buscar, seleccionar qué columnas ver, fijar filtros por columna y actualizar los datos (estas funcionalidades van a estar disponible en todas las tablas de la aplicación, ya que son propias de la librería MUI).

Como se puede ver en la figura 22, el formulario solicita el nombre de la métrica instanciada y lista las bases de datos cargadas en la vista de modelos de datos para que el usuario seleccione una. Con la finalidad de que el usuario no deba escribir el nombre de la tabla y columna, está el botón “Get Tables” que lista las tablas que contiene la BD seleccionada. Además, una vez elegida una, automáticamente se listan las columnas que tiene la misma y, por último, se muestran los métodos de medición cargados previamente para que el usuario elija uno.

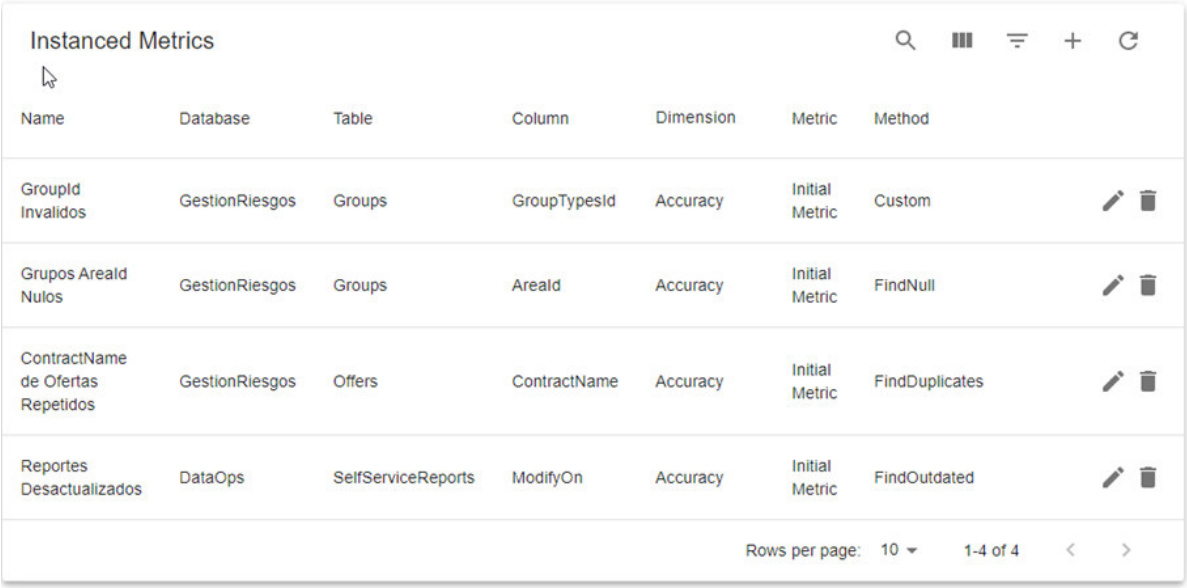

Figura 21- Vista de métricas instanciadas
Fuente: Aplicación en ambiente de pruebas

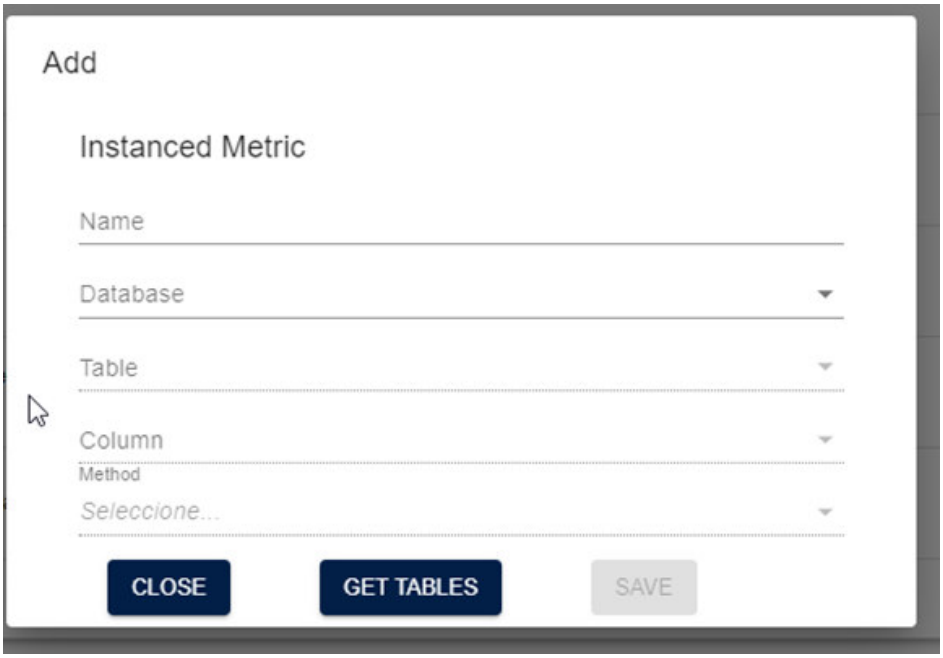

Figura 22- Formulario de creación de métricas instanciadas
Fuente: Aplicación en ambiente de pruebas

7.4 Modelos de datos

Aquí se agregan, modifican o eliminan las bases de datos que serán analizadas con la aplicación.En el formulario se solicita el nombre de la BD y la dirección del servidor, ya que, por motivos de seguridad, no se piden las credenciales de acceso a la misma. En cambio, el administrador de los servidores previamente debe guardar manualmente en cada uno de las BDs, el appPoolUser del servidor donde está almacenada la aplicación. En caso de no hacerlo, se notificará por pantalla de un error al usuario.

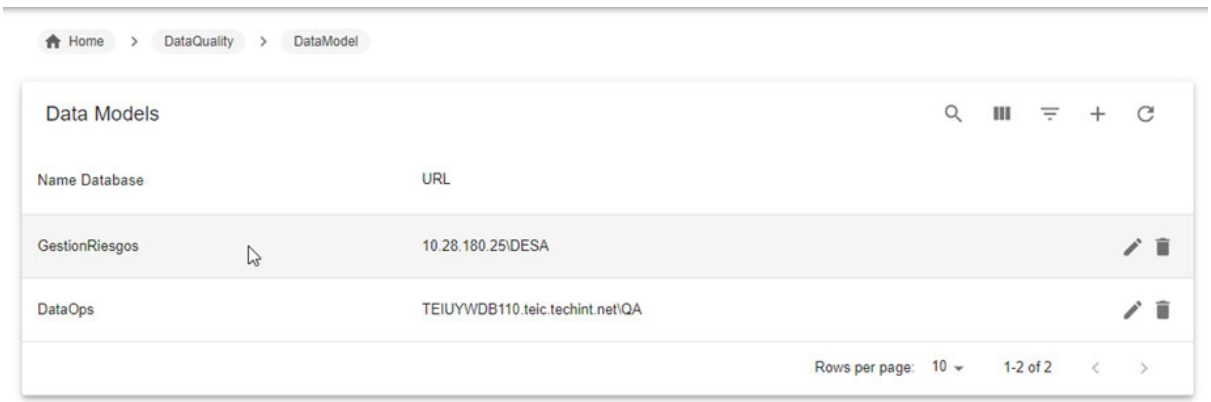

Figura 23- Vista de modelo de datos
Fuente: Aplicación en ambiente de pruebas

Figura 24- Formulario de alta de modelos de datos
Fuente: Aplicación en ambiente de pruebas

7.5 Métodos

Como se puede ver en la figura 25, en una tabla se visualizan los métodos de medición disponibles para utilizar, pero no se permite generar nuevos ni editarlos. Esto se debe a que los mismos deben ser solicitados por los usuarios y agregados por los desarrolladores.Así, se vuelve más fácil la implementación de estos desde el backend y, de ser necesario, se pueden realizar modificaciones en los formularios existentes.

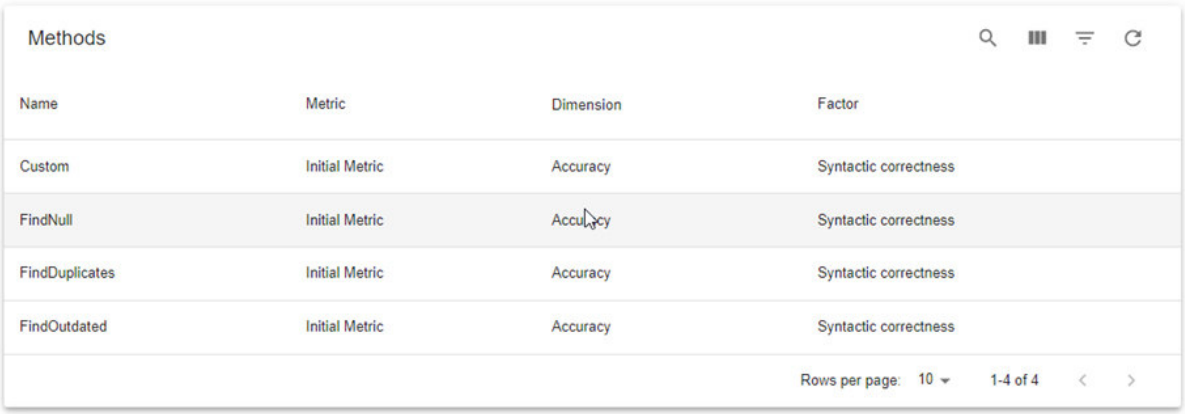

Figura 25- Vista de métodos
Fuente: Aplicación en ambiente de pruebas

7.6 Métricas

En esta pantalla los usuarios podrán crear, actualizar y eliminar las métricas de medición que luego serán vinculadas a una base de datos. Al momento de agregar una nueva o editar una existente, el formulario solicita un nombre para esta y ofrece la lista de factores y tipos de granularidad para que el usuario seleccione. Además, al momento de listarse en la tabla, también se muestra qué tipo de dimensión del dato se quiere cuantificar.

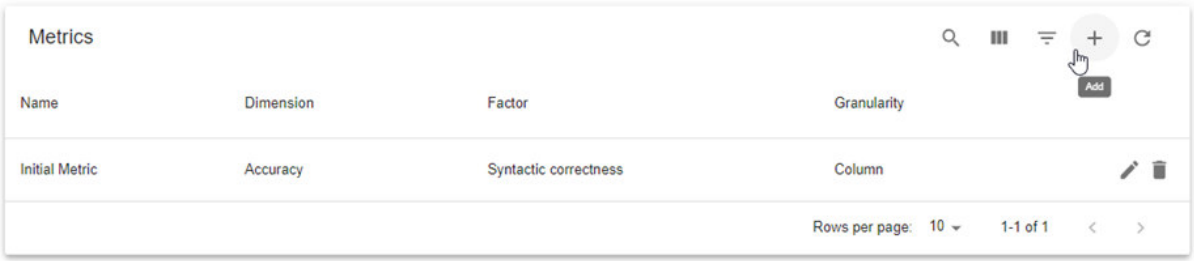

Figura 26- Vista de métricas
Fuente: Aplicación en ambiente de pruebas

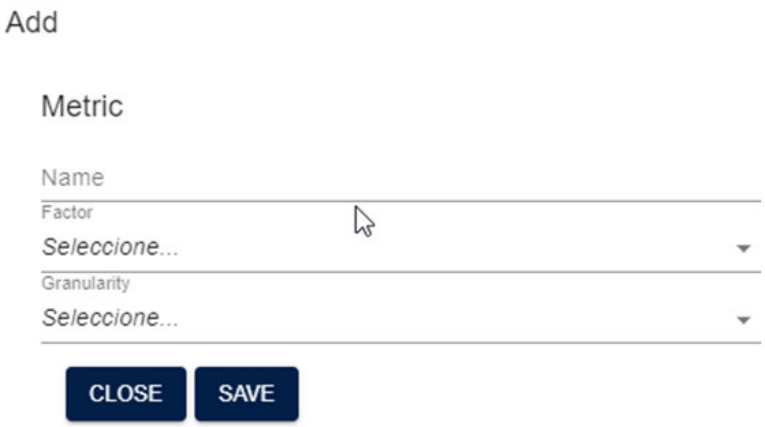

Figura 27- Formulario de creación de métricas
Fuente: Aplicación en ambiente de pruebas

7.7 Factores

Similar a los casos anteriores, se utiliza una tabla que permite crear, editar o eliminar los factores que serán implementados por las métricas. Para ello, se

debe ingresar un nombre para el factor y, a continuación, se muestra una lista de tipos de dimensiones (ya definidas) para que el usuario seleccione.

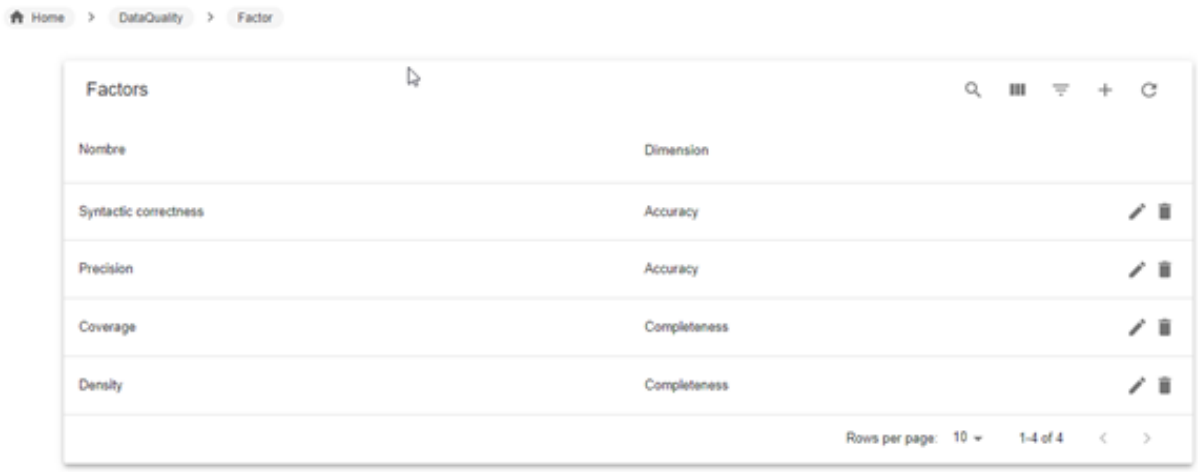

Figura 28- Vista de factores
Fuente: Aplicación en ambiente de pruebas

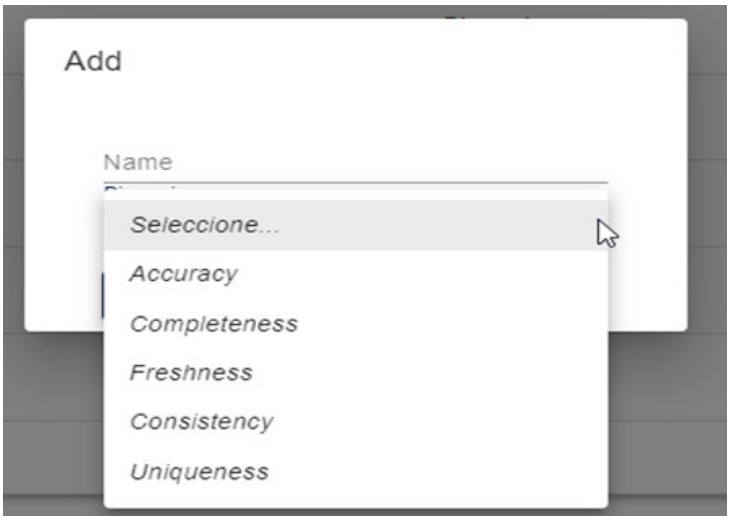

Figura 29- Formulario de creación de factores
Fuente: Aplicación en ambiente de pruebas

7.8 Reglas de calidad

Como se puede ver en la ilustración 30, además de tener las operaciones típicas de crear, editar y eliminar, esta tabla agrega un comportamiento distinto:

- Si una regla no necesita de valores ingresados por el usuario (esperado, opcional o invalido), se completa la celda con “NA” (No aplica) y al ubicar el mouse sobre el mismo, se le comunica al usuario que ésta no requiere de ese tipo de valor.
- Dependiendo del nivel de criticidad de la misma, se pinta un icono con los colores de un semáforo (alta: colorado, media: amarillo y baja: verde)

Para dar de alta o editar una regla, se despliega todas las métricas instanciadas cargadas para que se seleccione una. Con el fin de evitar confusiones y mejorar la experiencia de usuario, una vez elegida la métrica instanciada se verifica automáticamente el tipo de método que esta tenga asociada para pedir sólo los valores que él mismo necesite (esperado, invalido u opcional). Es decir, el formulario tiene un comportamiento dinámico, ya que en vez de dejar campos en blanco o por defecto,van a ser ocultados lo que simplifica su diseño y además evita posibles errores.

También, despliega las opciones de criticidad y permite de forma opcional ingresar una descripción de la regla.

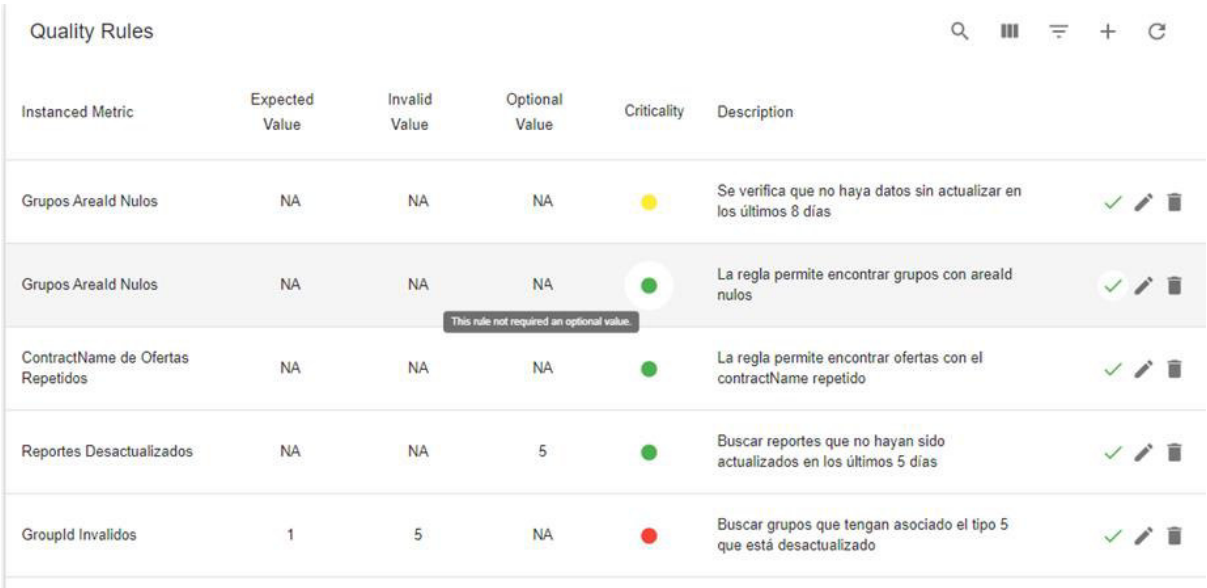

Figura 30- Vista de reglas de calidad
Fuente: Aplicación en ambiente de pruebas

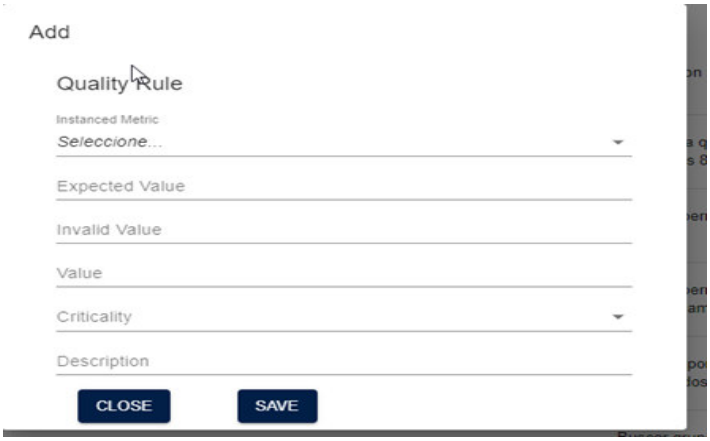

Figura 31- Formulario de creación de reglas de calidad
Fuente: Aplicación en ambiente de pruebas

7.9 Alertas

Una vez que se ejecutan reglas de calidad si se encuentran datos que incumplan a la misma, en esta pantalla se mostrarán las alertas correspondientes.

Como se puede ver en la figura 32, luego de generarse una alerta, si el problema presentado se soluciona y la regla de calidad vuelve a correrse, ésta cambia de estado agregando la fecha de cierre. Además, está disponible la opción de eliminar (así se encuentre abierta o cerrada)

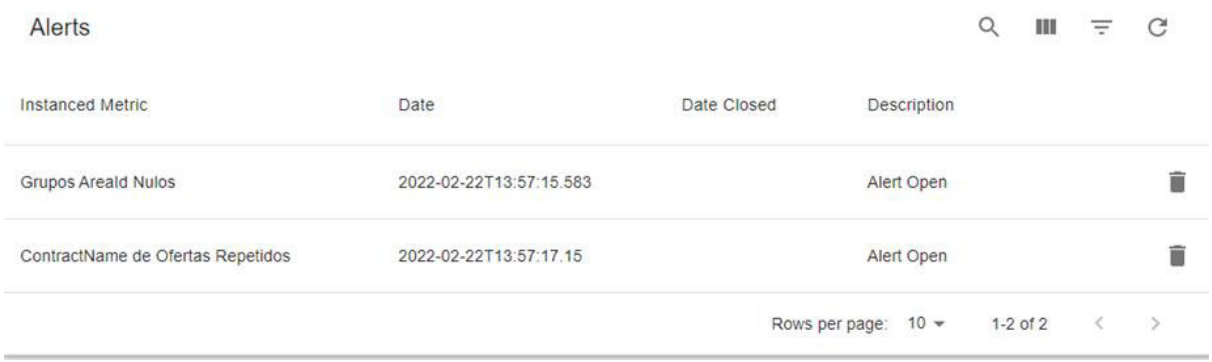

Figura 32- Vista de alertas
Fuente: Aplicación en ambiente de pruebas

7.10 Tablero

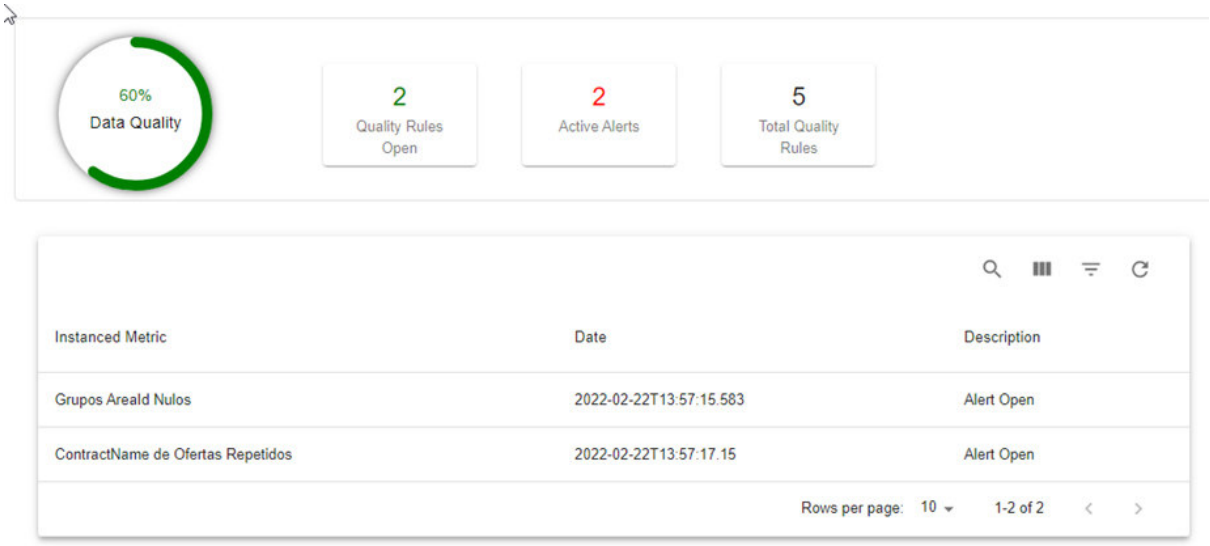

Figura 33- Vista del tablero
Fuente: Aplicación en ambiente de pruebas

Esta vista tiene dos áreas principales y permite visualizar, de forma rápida y sencilla, el estado de los datos, reglas y alertas del sistema.

En las tarjetas se indica cuántas alertas hay activas, es decir, que resta la implementación de una acción correctiva que solucione el error en el origen de los datos. También se indica el total de reglas definidas por los usuarios. Por último, en base al siguiente cálculo se grafica la calidad de los datos:

$$\frac{Reglas - Reglas\ abiertas}{Reglas} * 100$$

Respecto a la tabla inferior, la misma tiene una apariencia similar a la pantalla de Alertas, a excepción de que sólo muestra las que están activas.

7.11 Información General

Dado que la aplicación no cuenta con un manual de usuario, se agregó una vista que contiene un glosario donde se explica brevemente los conceptos más utilizados. La finalidad de esta es introducir a los usuarios del equipo de datos (que reciben las alertas) sobre qué factores de calidad se buscan cumplir con el uso de la aplicación y explicar la relación entre factor - método - métrica.

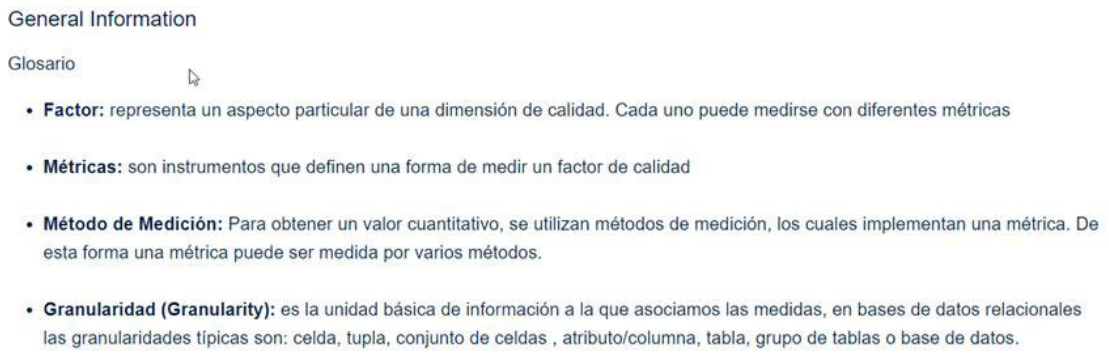

Figura 34- Vista del Glosario
Fuente: Aplicación en ambiente de pruebas

7.12 Detección de anomalías

Esta pantalla es muy similar a la de Modelo de datos salvo que solicita y muestra dos campos más: tabla y columna, a la vez que agrega un botón de ejecución que permite analizar en búsqueda de picos que puedan representar datos incorrectos.

Se muestran en la figura 35 todos los recursos cargados para que se ejecuten de a uno por vez.

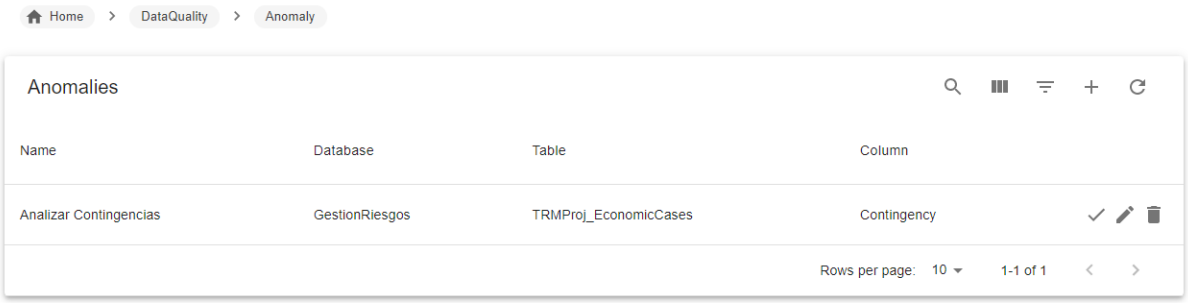

Figura 35- Vista de anomalías
Fuente: Aplicación en ambiente de pruebas

Al momento de la ejecución, de forma transparente para el usuario, se envía el origen de datos correspondiente a los algoritmos de ML para que busquen picos repentinos pero temporales que difieren significativamente de la mayoría de los valores. Una vez finalizado el análisis, se abre un modal (ilustración 36) que contiene una tabla que muestra los resultados correspondientes al análisis realizado:

- Alert: Si el resultado obtenido es una alerta, se pinta de colorado; en caso contrario, de verde
- Value: se muestra el valor analizado
- pValue: se indica la probabilidad de que el dato sea una anomalía.

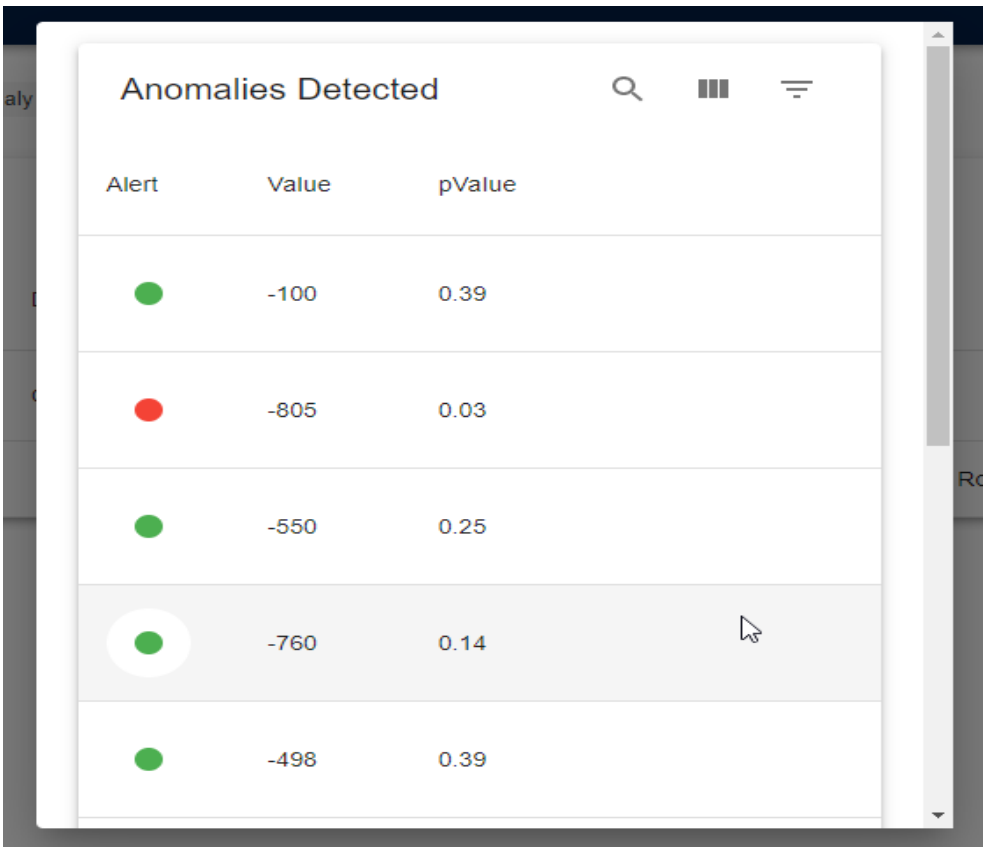

Figura 36- Modal con resultado de análisis de búsqueda de anomalías
Fuente: Aplicación en ambiente de pruebas

7.13 Acciones correctivas

Si al ejecutarse reglas, se encuentran datos que incumplen los valores esperados, además de generarse una alerta, también se notificará por email a un responsable del equipo de Data y Analytics definido por el product owner del proyecto.

En el mismo se notifica cuál regla de calidad fue ejecutada y se envía el enlace correspondiente para que se pueda acceder directamente.

Al conocer cuál es la métrica instanciada implicada, el usuario puede conocer en qué recurso, se encontraron datos que incumplen los valores esperados y pueden seguir las acciones correspondientes definidas por el equipo. Como, por ejemplo, auditar si hay errores en la aplicación u ocurrió un error humano en la carga de datos.

Figura 37- Email de notificación por alerta generada
Fuente: Aplicación en ambiente de pruebas

Este procedimiento es similar al buscar anomalías. Si se encuentran datos que generen posibles alertas, también se notificará por email al mismo usuario. En este caso, se informa el nombre de ésta, en qué base de datos, en cuál tabla y columna ocurrió.

Dado que encontrar datos fuera del rango esperado no significa que sea un error, el procedimiento a seguir por parte del equipo de Data y Analytics es diferente. En este caso, el usuario deberá buscar anomalías nuevamente y, una vez que los datos sean identificados, deberá trabajar con su equipo para definir si es un error y en ese caso proceder como si fuera una alerta o, en el caso que no lo sea, desestimar a la misma.

Figura 38- Email de notificación por anomalías encontradas
Fuente: Aplicación en ambiente de pruebas

8 Interacciones en el Backend

En los siguientes casos se detallan los pasos más significativos de las interacciones que se realizan en el Backend (principalmente donde interactúan el InfoDbService y el InfoDbHelper). El hecho de que no se expliquen todos los pasos es para que sean claros, evitar repeticiones y simplificar las descripciones.

8.1 Consultas a las bases de datos

Si bien Entity Framework permite contar y mapear una base de datos de forma sencilla a una aplicación para realizar consultas, se necesita que un desarrollador realice estas configuraciones. Por lo tanto, cada vez que se quiera agregar un modelo de datos nuevo, se deben destinar horas de trabajo para el backend sumado a las desventajas que generaría el crecimiento de la aplicación cada vez que se conecta a una nueva base.

Para evitar dichas complicaciones y poder conectarse de forma automática a las bases cargadas y poder realizar las consultas necesarias, se decidió utilizar ADO.NET en el proyecto.

De esta manera, se utiliza lenguaje SQL en el código para conectarse a las bases, ejecutar las consultas (queries) y los resultados obtenidos son

enviados a la lógica de negocio de la aplicación. Estas consultas son definidas en InfoDbHelper y es este quien se conecta a todas las bases de datos y luego de ejecutar las queries, procesa la respuesta y se la envía a la lógica de negocio de la aplicación. En la imagen siguiente, se muestra como es el proceso para una vista determinada.

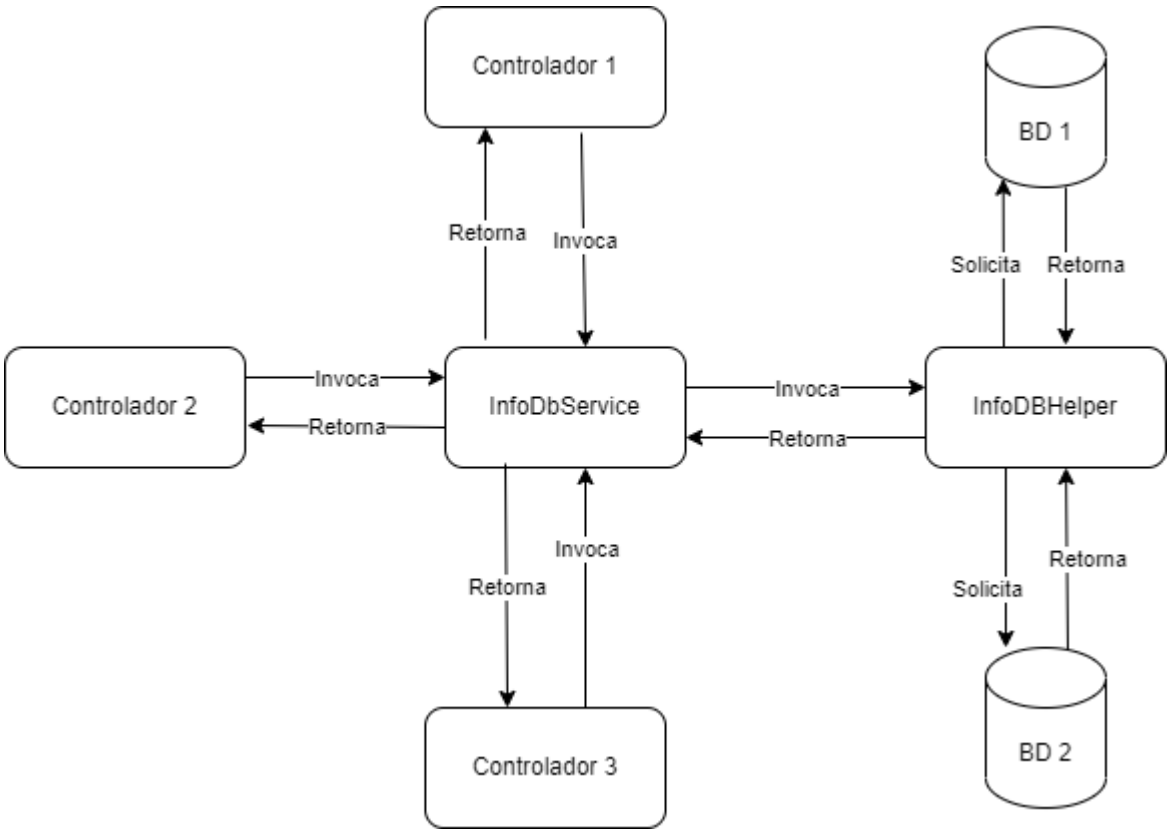

Figura 39- Proceso de conexión de múltiples orígenes de datos
Fuente: Elaboración propia basada en la práctica

8.2 Dar de alta modelos de datos

Al momento de dar de alta bases de datos (que luego serán analizadas), se debe validar que el usuario tenga acceso a las mismas. El proceso es el siguiente (según figura 40):

1. Desde la vista, el usuario ingresa el nombre de la base y la url del servidor, le solicita la acción TestConnection a DataModelController.
2. Este mismo, atiende la solicitud y llama a un método homónimo de InfoDbService.

3. Dicho servicio tiene la responsabilidad de llamar InfoDBHelper.
4. Como se aclaró anteriormente, este tiene definida una función que intenta conectarse al modelo de datos. Para ello, aparte del nombre y de la ruta del servidor, se necesita un usuario y contraseña. Como no sería seguro ni práctico que para cada consulta a la base el usuario deba ingresar estas credenciales y, mucho menos aún, guardarlas en base, la autenticación se realiza con una identidad de grupo de aplicaciones (Application Pool Identity).En las aplicaciones web que se ejecutan bajo IIS, esta identidad es un usuario que se va a utilizar para ejecutar el proceso de dicha aplicación. Por lo tanto, si en una base de datos, se autoriza a este usuario, una aplicación que se encuentre en dicho servidor web podrá tener acceso a la misma, sin necesitar ingresar usuario y contraseña. Esto es porque cuando se usa la identidad del grupo de aplicaciones con seguridad integrada (al momento de conectarse a la base), las conexiones a SQL Server usan esta identidad para autenticar mientras que el motor SQL valida si el mismo es un usuario autorizado en la base, si no lo es, rechaza la conexión a la misma. En resumen, en el método TestConnection de InfoDBHelper se utiliza la seguridad integrada para conectarse a la base de datos.
5. Luego, el motor de la base responde si la conexión es exitosa o no.
6. Dependiendo de dicha respuesta, la función TestConnection retorna verdadero o falso al InfoDbService.
7. Este servicio le transmite dicha respuesta al controlador.
8. Entonces, este último se encarga de procesar y enviar la respuesta a la vista de Modelo de datos.
9. La cual procesa y muestra la respuesta por pantalla.

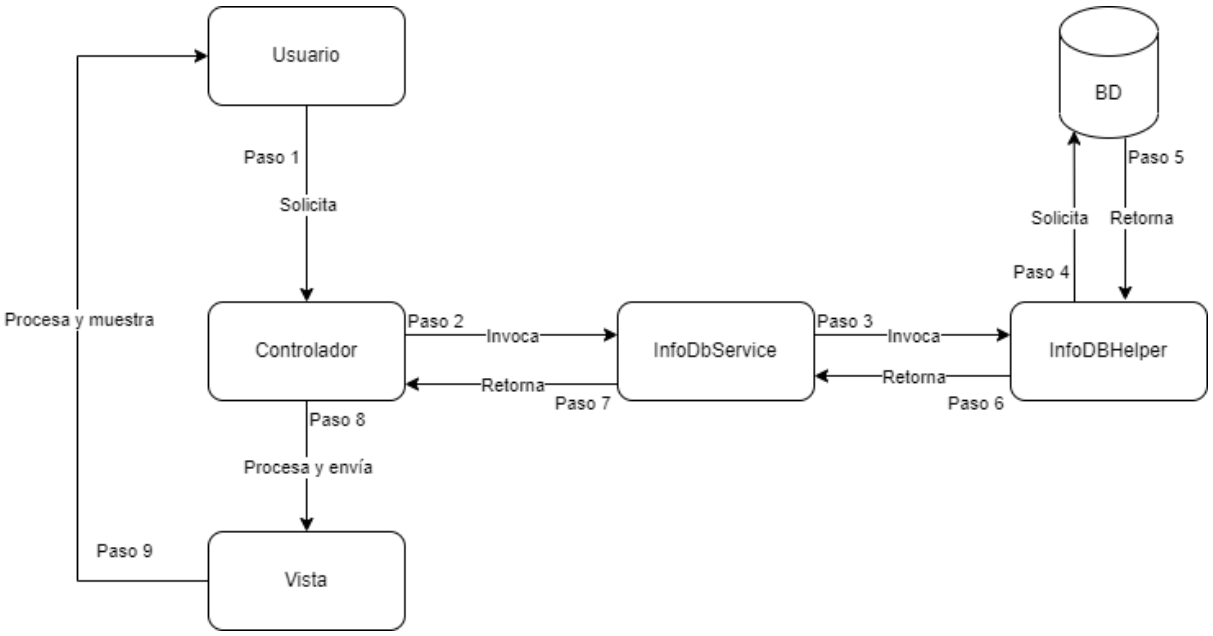

Figura 40 - Interacciones
Fuente: Elaboración propia basada en la práctica

8.3 Crear métricas instanciadas

Para agregar o editar estas, una vez que el usuario elige entre las bases cargadas, se debe buscar cuáles son las tablas y columnas que tiene la misma. Se diseñó el siguiente proceso (según figura 40) para optimizar los tiempos de la consulta, traer solo los registros necesarios y que esta operación sea totalmente transparente para el usuario, :

1. Desde la vista de métrica instanciada, el usuario selecciona la base que quiere medir y presiona el botón 'Get Tables', éste solicita la acción ListTablesFromDb a DataModelController.
2. A su vez, llama a la función homónima que tiene InfoDbService.
3. El servicio tiene la responsabilidad de llamar InfoDBHelper. Este método pide conectarse a la base de datos y ver las tablas que contiene la misma. Para ello, además del nombre y de la ruta del servidor, se necesita un usuario y contraseña.
4. Igual que la función TestConnection, en el método ListTablesFromDb del InfoDBHelper, se utiliza la seguridad integrada para conectarse a la base de datos.

5. Ante la consulta, el motor de la base responde si la conexión es exitosa o no.
6. Dependiendo si la conexión es exitosa y a su vez, si la BD tiene tablas creadas, dicha función retorna una lista con los nombres de estas o una lista vacía al InfoDbService.
7. Este servicio, le reenvía la lista al controlador.
8. DataModelController es el encargado de transformar y enviar la respuesta a la vista.
9. Esta pantalla, procesa y muestra la respuesta en su formulario listando las tablas para que el usuario elija una.

8.4 Crear reglas de calidad

Al momento de dar de alta reglas, dependiendo del método que vaya a implementar, se personaliza el formulario (desde el Backend para que sea transparente para el usuario) y así evitar errores en la generación de estas. El proceso es el siguiente (según figura 41):

1. Al presionar el botón de agregar nuevo, se le solicita la acción GetAll a IntancedMetricController.
2. A su vez, este controlador llama a dicha función de IntancedMetricService.
3. Éste tiene la responsabilidad de conectarse a la base de datos de la aplicación y traer todas las métricas instanciadas cargadas.
4. Para esa consulta también se pide que se traiga los datos del método que tenga asociada cada métrica. Esta información es retornada por la base de datos al servicio en forma de lista.
5. Por lo tanto, IntancedMetricService hace de “pasamanos” y le reenvía esta lista al controlador.
6. IntancedMetricController procesa la información, generando una lista de InstancedMetricDTO (con el fin de enviar solo la información que la misma necesite) y se la envía a la vista de reglas de calidad.
7. Ésta debe procesar y mostrar la respuesta por pantalla, listando las instancias métricas ya cargadas para que el usuario elija una.

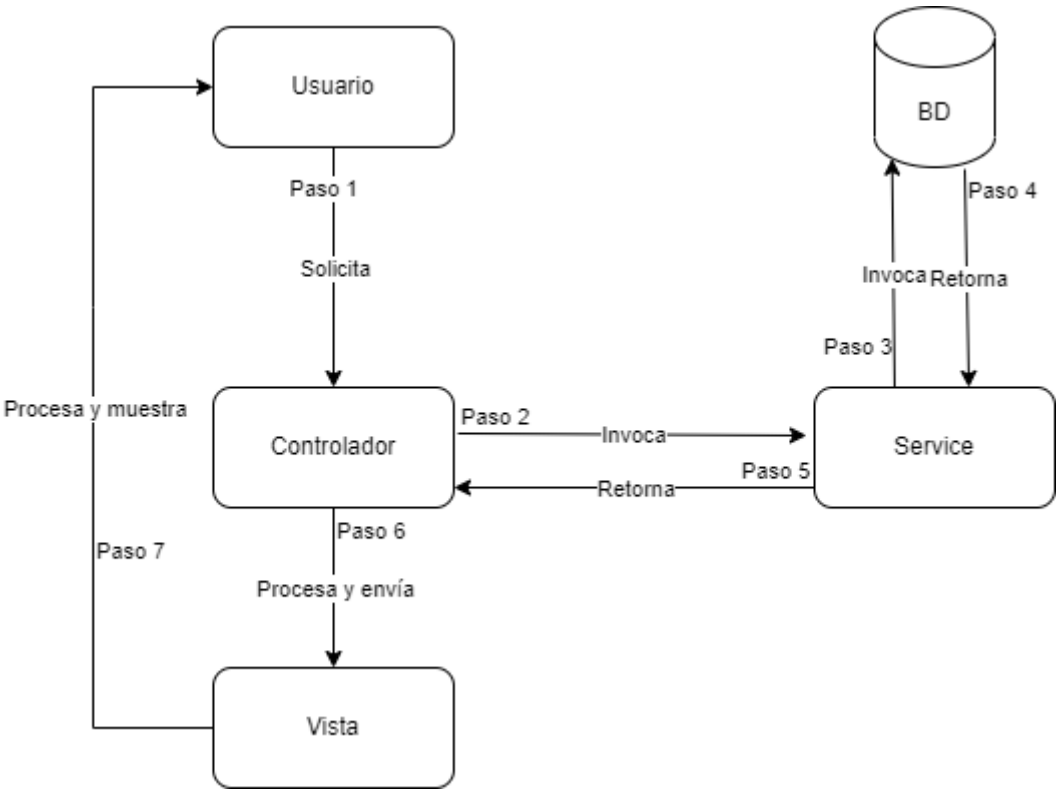

Figura 41 - Interacciones
Fuente: Elaboración propia basada en la práctica

8.5 Crear métricas

Para dar de alta métricas, se lleva a cabo el siguiente proceso (según figura 41):

1. Al presionar el botón de agregar nuevo, se le solicita la acción GetAll a FactorController.
2. A su vez éste llama a dicho método de FactorService.
3. Este tiene la responsabilidad de conectarse a la base de datos y traer todos los factores guardados en el sistema.
4. Para dicha operación, también se requiere que traiga los datos de la dimensión que tiene asociada dicho factor. Esta información es retornada por la base de datos al servicio.
5. Una vez que InstancedMetricService recibe la respuesta, la reenvía en forma de lista al controlador.

6. Aquí, él mismo procesa la información generando una lista de FactorDTO y se la envía a la vista de métricas.
7. Esta procesa la información recibida y muestra la respuesta por pantalla listando los factores para que el usuario elija una.

8.6 Creación de factores, métodos y visualización de alertas

El procedimiento anterior es similar cuando se crean, editan o eliminan factores, métodos y alertas. En todos los casos, los componentes participantes son sus propios servicios, controladores, DTOs y vistas (a excepción de la pantalla de factores donde se necesita que se traigan las dimensiones y por lo tanto se utilizan los suyos). Por lo tanto, para cada pantalla interactúan los siguientes:

- Factor: DimensionController, DimensionService y DimensionDTO.
- Métodos: MethodController, Methodervice y MethodDTO.
- Alertas: AlertController, AlertService y AlertDTO.

8.7 Crear o cerrar alertas (almacenamiento de los metadatos)

La creación de alertas ocurre cuando una regla de calidad es ejecutada por un usuario y la misma encuentra datos en la base que no tienen el formato o valor que debería. Por lo tanto, para generar o cerrar alertas se lleva a cabo el siguiente proceso (según figura 42):

1. Desde la vista de reglas de calidad el usuario selecciona cuál quiere ejecutar y presiona el botón ‘Execute Rule’, quien solicita la acción ExecuteRule a RuleExecutionController y le manda como argumento un QualityRuleDTO.
2. Este es procesado por el controller para obtener la regla de calidad asociada y, a su vez, llama al método homónimo de RuleExecutionService y le envía el Id de la regla como argumento.

3. Una vez recibida la regla, dicho servicio llama a la función GetById de IntancedMetricService pasando el Id de la instancia métrica como argumento.
4. Este servicio tiene la responsabilidad de conectarse a la base de datos para obtener la métrica instanciada que tiene ese Id.
5. Dicha información es retornada por la BD a IntancedMetricService.
6. Este actúa de “pasamanos” por lo tanto, despacha dichos datos al RuleExecutionService.
7. A continuación, el servicio verifica el método que tiene la métrica instanciada y se comunica a la función homónima que tiene InfoDbService.
8. Quien a su vez tiene la responsabilidad de llamar a InfoDBHelper.
9. Todos los métodos de medición son similares en su funcionamiento ya que intentan conectarse a la base de datos y ejecutar una consulta con una condición establecida que deben cumplir los datos, (la cual varía según el método). Para ello, además del nombre, de la ruta del servidor, de los argumentos necesarios (o no) para cada método, se necesita un usuario y contraseña. Por este motivo, se utiliza la seguridad integrada para conectarse a la base de datos.
10. Si la conexión es exitosa, se realiza la consulta y luego el motor de la base retorna la cantidad de registros que no cumplieron la condición de calidad definida al InfoDBHelper.
11. A su vez, éste le reenvía dicha cantidad a InfoDbService.
12. El mismo la transmite a RuleExecutionService, quien verifica si la respuesta es mayor a cero o no y, en base al resultado, tiene dos opciones posibles:

- Si la respuesta es mayor a 0:
    13. Llama al método GetByQualityRule de AlertService y le envía como parámetro el id de la regla de calidad ejecutada.
    14. Este tiene la responsabilidad de conectarse a la base de datos y consultar si hay alertas asociadas a ese Id.

15. En caso de que las haya, la base le retornará la última generada. Si no, le devolverá un valor nulo.
    - Si no hay alertas generadas, este servicio, creará una nueva utilizando el método InsertAsync, pasándole esta última como argumento. Para eso, AlertService tiene la responsabilidad de conectarse nuevamente a la base de datos y agregar esta alerta en la tabla correspondiente.
    - Si hay alguna alerta asociada a ese Id, se verifica el estado. Si está abierta, no se toman acciones; , pero si está cerrada, se creará una nueva, utilizando la función InsertAsync.

- Si la respuesta es igual a cero:

  Se repiten los pasos 13 y 14 para buscar alertas existentes. A diferencia del caso anterior, si la hay, se verifica su estado; y si está abierta, se debe cerrar. Por lo tanto, este servicio utiliza el método Update enviando dicha alerta como parámetro.

16. Dependiendo de si estas operaciones fueron satisfactorias o no, RuleExecutionService le responde Verdadero o Falso al controlador.
17. El mismo es el encargado de dirigir dicha respuesta a la vista de reglas de calidad.
18. Esta última procesa y muestra la respuesta por pantalla indicando si la regla corrió o no satisfactoriamente.

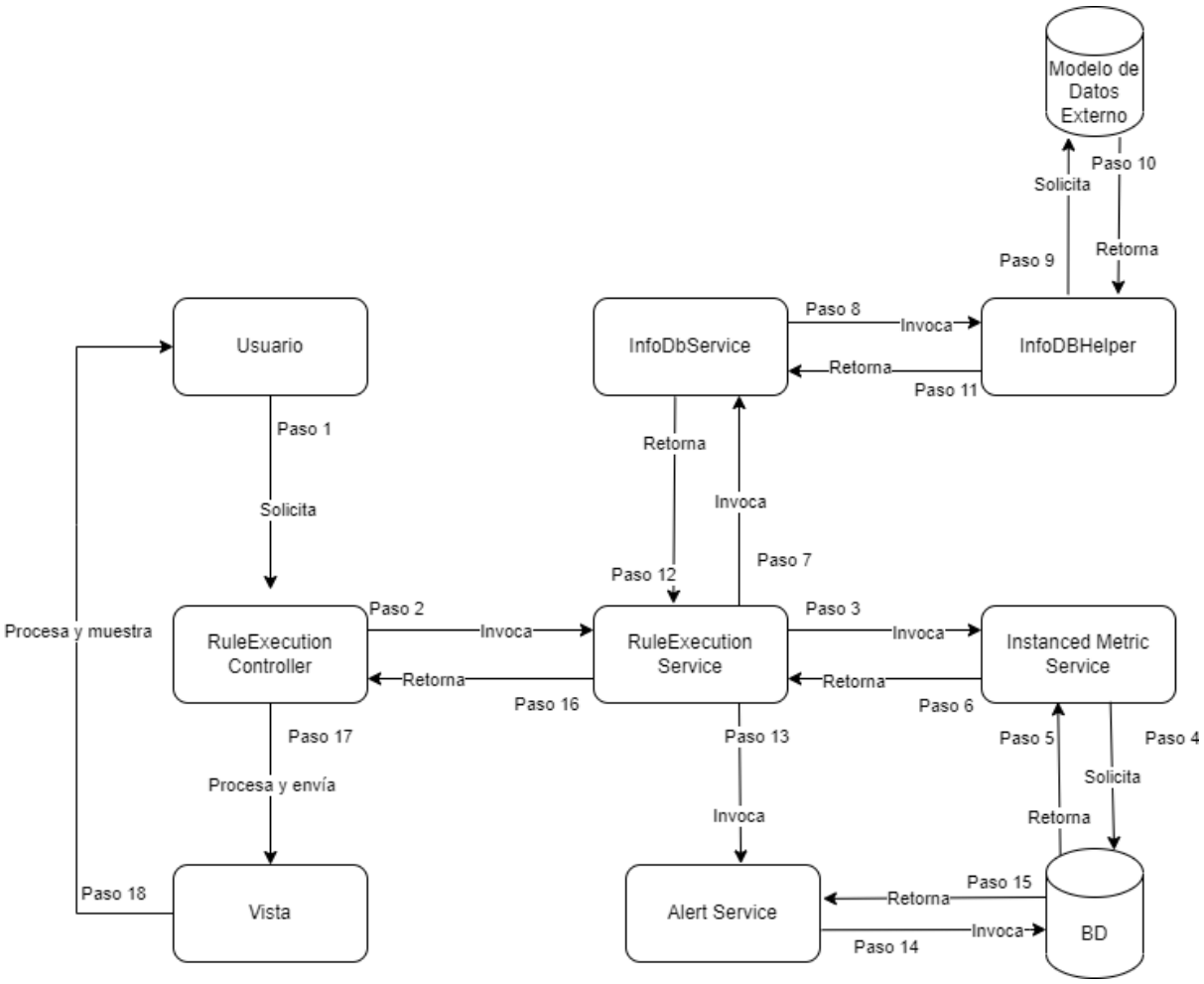

Figura 42 - Generación de una alerta
Fuente: Elaboración propia basada en la práctica

8.8 Ver tablero

Es una pantalla sencilla donde sólo se pueden visualizar alertas abiertas y el total de reglas de calidad. Para ello se realiza un proceso similar al descrito anteriormente en 8.4. Es decir, para consultar las reglas de calidad previamente cargadas, se comienza desde esta vista y se utiliza el controlador y servicio de reglas de calidad para comunicarse con la base de datos. Y esta, a su vez, devuelve la información solicitada.

También y al mismo tiempo, por medio del controlador y servicio de alertas, la vista llega a la BD pero en este caso para visualizar cuáles están activas.

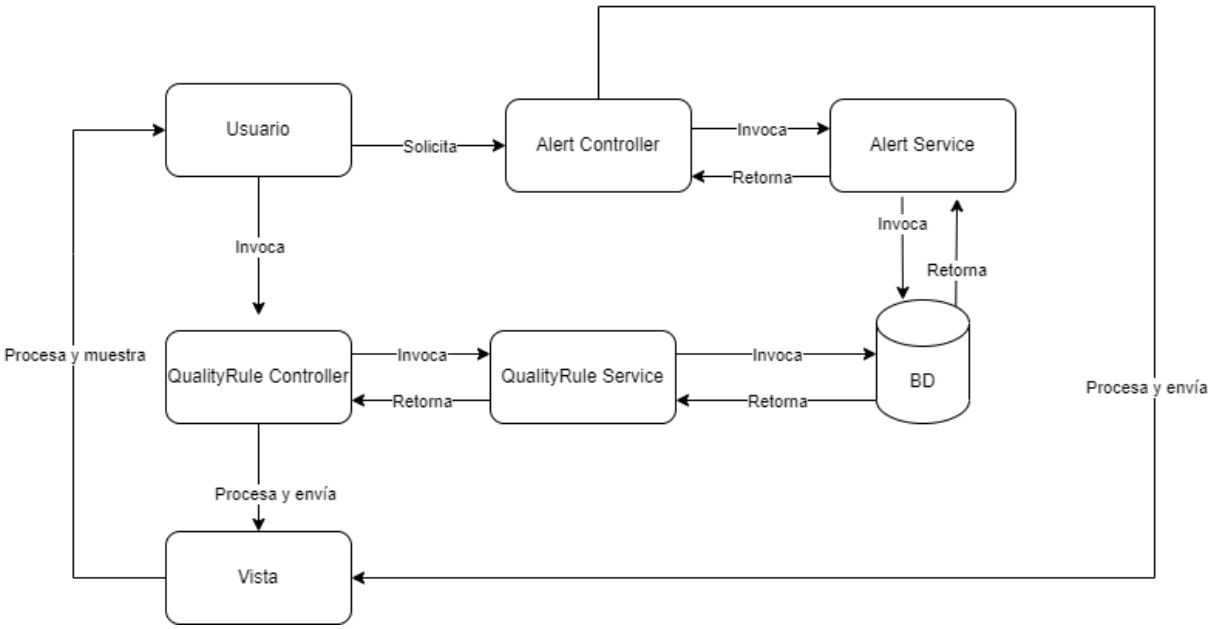

Figura 43 - Cargar tablero
Fuente: Elaboración propia basada en la práctica

8.9 Autenticación

Para el login de usuarios, se utilizó la ofrecida por Windows (también llamada NTLM) que permite identificar un inicio de sesión con credenciales suministradas por el sistema operativo Windows del equipo que realiza la conexión. Se utilizó por tres motivos:

- El servidor web se hace responsable de esta tarea ahorrando tiempo de desarrollo y posibles errores al implementar las funcionalidades necesarias.
- Como la aplicación es interna de la compañía (un sitio que solo se accede desde la intranet), los usuarios pueden usar sus nombres de usuario y contraseñas estándar de Windows al acceder.
- La autenticación de Windows generalmente es muy robusta dado que utiliza un mecanismo de seguridad basado en certificados. Los inicios de sesión autenticados de Windows pasan un acceso token en lugar de un nombre y una contraseña.

8.10 Detección de anomalías

La detección de anomalías marca comportamientos o eventos inesperados o poco habituales. Proporciona pistas sobre dónde buscar problemas y ayuda a responder la pregunta "¿es extraño esto?". Es decir, es el proceso que detecta valores atípicos de datos de series temporales, puntos en determinada serie temporal de entrada donde el comportamiento no es lo que se esperaba o es "extraño".

Hay dos tipos de anomalías de series temporales que se pueden detectar:

- Los picos indican ráfagas temporales de comportamiento anómalo en el sistema.
- Los puntos de cambio indican el inicio de cambios persistentes con el tiempo en el sistema.

En ML.NET, los algoritmos de detección de picos de IID o de puntos de cambio de IID son adecuados para los conjuntos de datos independientes y distribuidos de manera idéntica. Se supone que los datos de entrada son una secuencia de puntos de datos que se muestran independientemente de una distribución fija.

A diferencia de otros modelos, las transformaciones del detector de anomalías de series temporales funcionan directamente en los datos de entrada, es decir, no necesita los de entrenamiento para generar la transformación. Sin embargo, necesita de un esquema (origen de datos), que se proporciona mediante una vista de estos generada a partir de una lista de los que se quieren analizar.

Para cualquiera de estos tipos de anomalías, el proceso de modelo de entrenamiento y compilación es el mismo. La principal diferencia es el algoritmo de detección específico que se utiliza.

8.10.1 Detección de picos

El objetivo de la detección de picos es identificar ráfagas repentinas pero temporales que difieren significativamente de la mayoría de los valores de datos de las series temporales. Es importante detectar estos elementos, eventos u observaciones poco frecuentes y sospechosas de manera oportuna con el fin de

minimizarlos. El siguiente enfoque puede utilizarse para detectar una variedad de anomalías, como interrupciones, ciberataques o contenido web viral. La siguiente imagen es un ejemplo de los picos de un conjunto de datos de serie temporal:

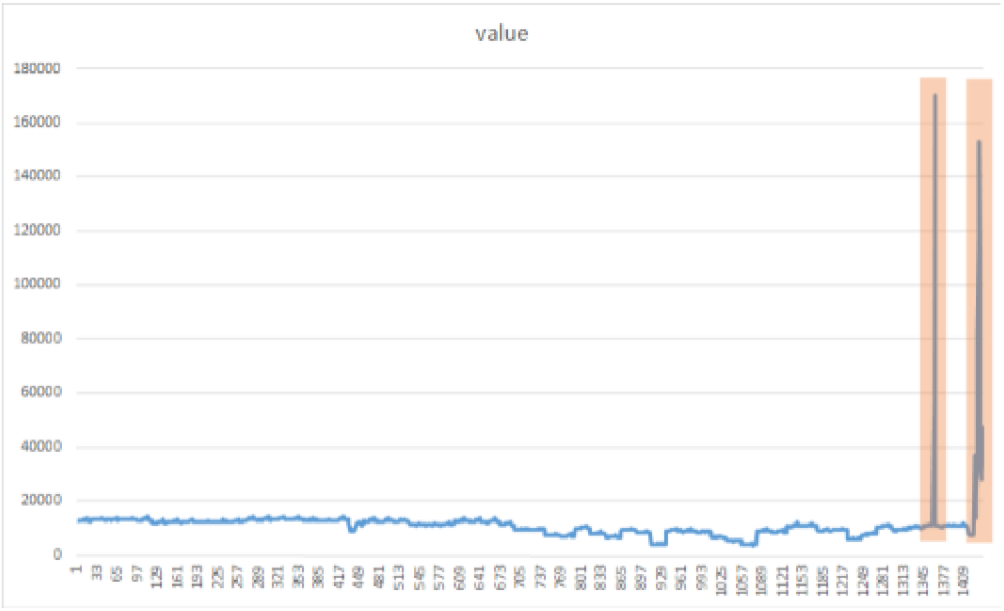

Figura 44 - Dos picos encontrados en una serie temporal
Fuente: https://docs.microsoft.com/es-es/dotnet/machine-learning/tutorials/sales-anomaly-detection

El resultado de la detección depende de los siguientes parámetros:

- Confidencialidad: determina la sensibilidad del modelo a los picos. A menor confianza, hay más probabilidades de que el algoritmo detecte picos "más pequeños", es decir, encuentre más anomalías. La documentación recomienda utilizar una confidencialidad de 95
- Longitud histórica: este parámetro es un porcentaje de todo el conjunto de datos y es resultado de dividir al número de registros que tiene el origen de datos (base de datos, archivos de tipo .CSV, XLSX, etc). Cuanto menor sea este porcentaje, con más rapidez se olvida el modelo de los picos grandes anteriores.

La documentación sugiere que la confidencialidad sea igual a 95 mientras que la longitud sea calculada al realizar el cociente entre el número de registros y cuatro.

Una vez realizada la detección de anomalías, por cada registro analizado (punto de dato) se devuelven los siguientes datos:

- Alerta: indica una alerta de pico para determinado registro. Devuelve cero en caso de no encontrar ninguno; en caso contrario retorna uno.
- Valor: muestra el número analizado.
- Probabilidad: cuando más se acerque este valor a cero, es más probable que el punto de datos sea una anomalía.

8.11 Crear anomalías

Al momento de dar de alta anomalías, una vez que el usuario elige entre las bases cargadas, se debe buscar cuáles son las tablas y columnas que tiene la misma. Para optimizar los tiempos de la consulta, traer sólo los registros necesarios y que esta operación sea totalmente transparente para el usuario, se diseñó el siguiente proceso (según figura 40):

1. Desde la vista de anomalías el usuario selecciona la base que quiere medir y presiona el botón 'Get Tables', este solicita la acción ListTablesFromDb a DataModelController.
2. Este, a su vez, llama a la función homónima de InfoDbService.
3. Cuando el servicio recibe la petición, tiene la responsabilidad de llamar InfoDBHelper. Este método pide conectarse a la base de datos y ver las tablas que contiene la misma. Para ello, además del nombre y de la ruta del servidor, se necesita un usuario y contraseña.
4. En el método ListTablesFromDb del InfoDBHelper se utiliza la seguridad integrada para conectarse a la base de datos.
5. Luego de consultar a la misma, el motor de la base responde cuál es el estado de la conexión.
6. Dependiendo si la conexión es exitosa y a su vez, si la base tiene tablas creadas, dicha función retorna una lista con los nombres de estas o una lista vacía al InfoDbService.
7. El servicio, recibe la lista y a continuación la despacha al controlador.

8. Éste es el encargado de transformar y enviar la respuesta a la vista de métrica instanciada.
9. Una vez que la recibe, la procesa y muestra la respuesta por pantalla listando las tablas para que el usuario elija una.

8.12 Ejecutar búsqueda de anomalías

Para buscar y mostrar alertas de datos inusuales se lleva a cabo el siguiente proceso (según figura 45):

1. Desde la vista de anomalías, el usuario selecciona cuál quiere ejecutar y presiona el botón correspondiente. Esta acción llama a la funcion ExecuteAnomaly del AnomalyExecutionController y le envía como argumento un AnomalyDTO que es procesado por el controller para obtener la anomalía asociada.
2. Este controlador llama al método homónimo que tiene AnomalyExecutionService y le envía la anomalía que se guardó del paso anterior.
3. A partir de los datos recibidos, este servicio obtiene el Id del modelo de datos asociado a la anomalía y llama al método GetById de DataModelService enviando este código como argumento.
4. Una vez que recibe el Id, este tiene la responsabilidad de conectarse a la base de datos y traer todos los datos del modelo correspondiente.
5. Dicha información es retornada por la base de datos a DataModelService.
6. Este actúa como un “pasamanos” y por lo tanto, despacha dicha respuesta al AnomalyExecutionService.
7. Con los datos del modelo a analizar y los de la anomalía, el servicio crea una consulta SQL y se la envía a la función GetAllValuePredictions del AnomalyService.
8. Una vez que recibe dicha consulta, éste llama al método CreateDataTable de InfoDbService.
9. A continuación, el mismo se comunica con la función homónima que contiene InfoDBHelper.

10. Todas las detecciones anticipadas de anomalías son similares en su funcionamiento, ya que intentan conectarse a la base de datos y ejecutar una consulta con una condición establecida que deben cumplir los datos. Para ello, además del nombre, de la ruta del servidor se necesita un usuario y contraseña. Además, se utiliza la seguridad integrada para conectarse a la base de datos.
11. Al consultarla, si la conexión es exitosa, el motor de la base retorna todos los registros que cumplieron la consulta realizada al InfoDBHelper.
12. A continuación, se reenvían los datos a InfoDbService (paso 12).
13. Una vez más, este actúa de pasamanos y los transmite a AnomalyService.
14. Cuando recibe los datos, el servicio los procesa buscando picos de cambio y genera una lista de predicciones de valores que es enviada al AnomalyExecutionService.
15. Este último, hace lo mismo con el AnomalyExecutionController al trasladar dicha información en forma de lista.
16. El controlador, es el encargado de transformar los datos recibidos en una lista de AnomalyResponseDTO y enviarla a la vista de anomalías detectadas.
17. Por último, se procesa la lista y se muestra la respuesta por pantalla listando los datos examinados para que el usuario vea posibles alertas.

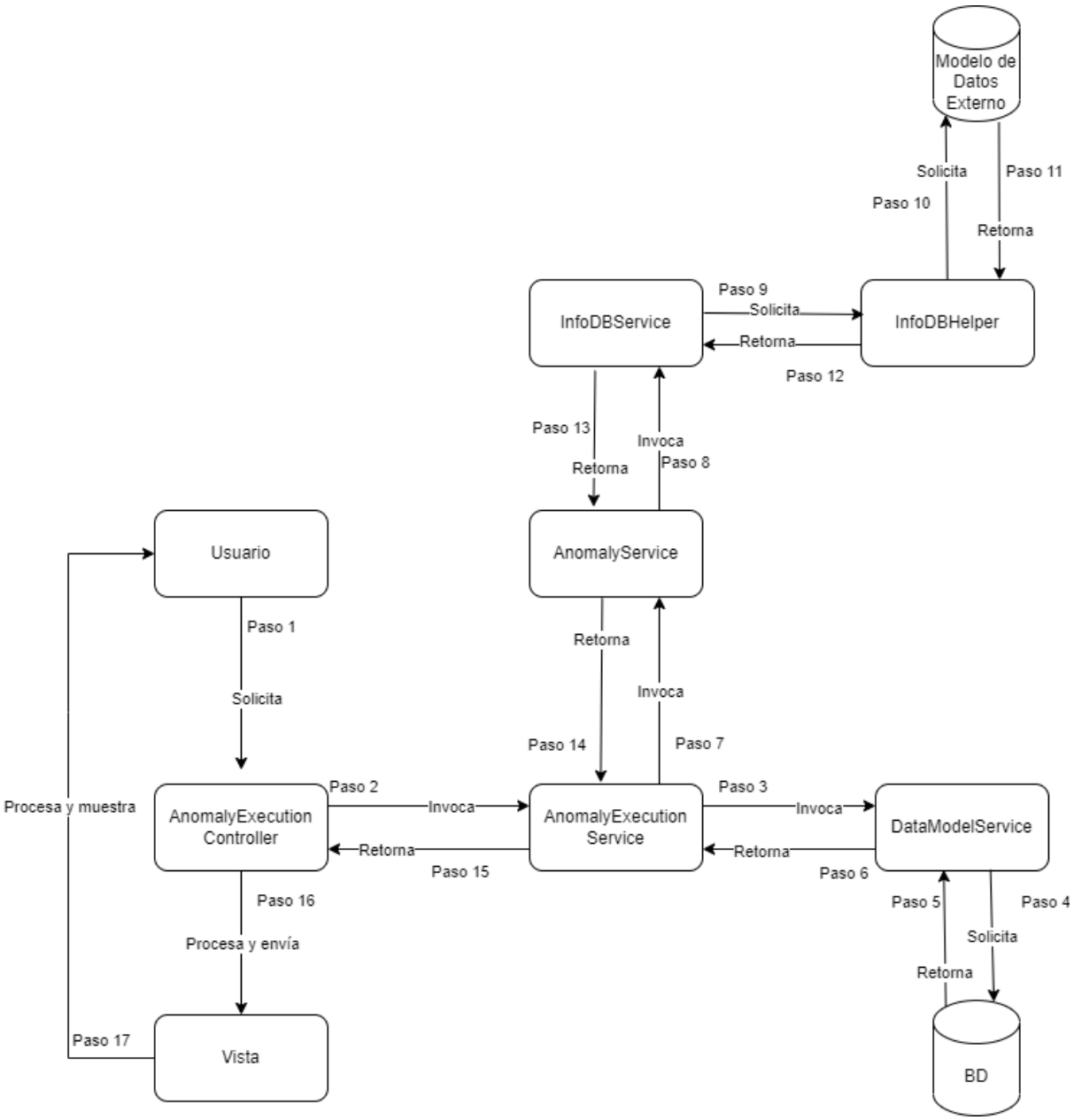

Figura 45 - Detección de anomalías
Fuente: Elaboración propia basada en la práctica

8.13 Acciones correctivas

Al ejecutar reglas o buscar anomalías (como se describe en los puntos 9.8 y 9.13 respectivamente) el controlador, una vez que obtiene la respuesta del servicio, valida si se generó una alerta (en el caso de que se haya corrido una regla) o si hay datos que representen una posible anomalía antes de procesarla y enviarla a la vista. En cualquiera de estos casos, gracias al DTO que recibe,

obtiene la información necesaria y se la envía a EmailService. Este es el encargado de armar y enviar el correo correspondiente. Una vez que lo hace, el controller continua con el procesamiento y le envía la respuesta a la vista correspondiente.

9. Futuras mejoras

Algunos evolutivos que pueden mejorar la experiencia del usuario es la implementación de métodos de medición desde la aplicación, ya que actualmente la misma se realiza solo por el desarrollador y se guarda desde la base de datos. Esto implica dos cuestiones: los tiempos lógicos de desarrollo, pruebas, e implementación en producción y además que estos estén sujetos a la disponibilidad del programador.

Otro caso similar ocurre con las reglas de ejecución y el método asociado a esta. A medida que el cliente solicite la implementación nuevos, se debe verificar que el formulario existente lo soporte y si no es así se debe modificar el mismo agregando más lógica tanto en Backend como Frontend.

Por último, un evolutivo que podría solucionar la problemática planteada anteriormente y además brindar una respuesta rápida a la necesidad del cliente sin la intervención del programador es agregar una pantalla, donde los usuarios puedan escribir una consulta SQL y automáticamente se mida la calidad de los datos del resultado.

9.1 Limitaciones del modelo actual de inteligencia artificial

El hecho de que ML.NET sea un marco de trabajo nuevo implica algunas desventajas. Entre ellas, que sus algoritmos sean básicos y que haya muy pocos desarrollos no oficiales. Esto genera que escaseen los debates e interacciones sobre estos temas en foros de desarrolladores y que lo poco que hay sean copias textuales de la documentación oficial. Por este motivo, las anomalías que se pueden detectar con este proyecto tienen algunas limitaciones como las siguientesr:

- No se puede analizar toda una tabla, ya que el algoritmo soporta de a una columna por vez. Esto genera que en la vista de anomalías se deba crear una por cada columna de la tabla que se quiere buscar detección de picos.
- Este modelo no tiene posibilidad de ser entrenado con datos previos.
- En el caso de requerir el análisis de una columna formada por los resultados de una consulta SQL avanzada; la misma deberá ser solicitada por los usuarios y llevarla a cabo por los desarrolladores en el backend.

Sumado a lo anterior, y a diferencia de otros lenguajes como Python o R, C# tiene características que vuelve muy difícil la implementación o desarrollo de modelos de Machine Learning

9.2 Definición del modelo MLOps

El alcance del proyecto le permitió al cliente “entrar al mundo de los datos y la inteligencia artificial”. Pero, para lograr una mejora en los algoritmos que vaya a necesitar, deberá continuar con la implementación de un modelo de MLOps que permita asegurar el éxito en el desarrollo de estos.

9.2.1 ¿Qué es MLOps?

MLOps significa Operaciones de aprendizaje automático. Es una función central de la ingeniería de aprendizaje automático, centrada en optimizar el proceso de llevar los modelos de aprendizaje automático a producción y luego mantenerlos y monitorearlos. Además, es de colaboración que a menudo incluye científicos de datos, ingenieros de desarrollo y TI.

9.2.2 ¿Cuál es el uso de MLOps?

MLOps se utiliza para crear y garantizar la calidad de soluciones de inteligencia artificial y aprendizaje automático mediante la implementación de prácticas de integración e distribución continuas (CI/CD). Es decir, se utiliza para

que los científicos de datos y los ingenieros de aprendizaje automático puedan colaborar y aumentar el ritmo del desarrollo y producción de estos modelos de aprendizaje.

9.2.3 ¿Por qué se necesita MLOps?

La producción del aprendizaje automático es difícil. El ciclo de vida del aprendizaje automático consta de muchos componentes complejos, como la ingesta de datos, la preparación de datos, el entrenamiento de modelos, el ajuste de modelos, la implementación de modelos, la supervisión de modelos, la explicabilidad y mucho más. También requiere colaboración y transferencias entre equipos, desde ingeniería de datos hasta ciencia de datos e ingeniería de ML. Naturalmente, se requiere un riguroso rigor operativo para mantener todos estos procesos sincrónicos y trabajando en tándem.

Esto hace que el desarrollo de modelos de aprendizaje automático sea inherentemente experimental y las fallas sean parte del proceso. Por lo tanto, la disciplina aún está evolucionando y se entiende que, a veces, incluso un modelo de ML exitoso puede no funcionar de la misma manera al día siguiente.

MLOps abarca la experimentación, la iteración y la mejora continua del ciclo de vida del aprendizaje automático.

9.2.4 ¿Cuáles son los beneficios de MLOps?

La gestión de estos sistemas a escala no es una tarea fácil, y existen numerosas brechas que deben solucionarse. Una buena política de MLOps resuelve desafíos como:

- La escasez de científicos de datos para desarrollar e implementar herramientas escalables.
- Aceptar rápidamente cambios de los objetivos permitiendo aportar valor lo antes posible.

- Salvar las brechas de comunicación entre los equipos técnicos, de datos y de negocio debido a un flujo de trabajo conocido.
- Facilitar la gestión de riesgos en la implementación de Modelos ML.

Las MLOps garantizan que los modelos se puedan implementar reiteradamente y monitorear de forma continua permitiendo lo siguiente:

- Implementar más modelos con mayor rapidez con procesos automatizados.
- Acelerar el tiempo de creación de valor con una rápida entrega de modelos.
- Optimizar la productividad a través de la colaboración y la reutilización de modelos.
- Reducir el riesgo de perder tiempo y dinero en modelos que nunca se incorporan a la producción.
- Monitorear y actualizar continuamente modelos a medida que los datos evolucionan con el tiempo.

9.2.5 ¿Cuáles son los componentes de MLOps?

El alcance de MLOps en proyectos de aprendizaje automático puede ser tan enfocado o expansivo como lo requiera el proyecto. La mayoría de las veces se implementan los principios de MLOps en lo siguiente:

- Análisis exploratorio de datos (EDA):
- Preparación de datos e ingeniería de características
- Entrenamiento y ajuste de modelos
- Revisión y gobernanza del modelo
- Inferencia y servicio de modelos
- Monitoreo de modelos
- Reentrenamiento automatizado de modelos

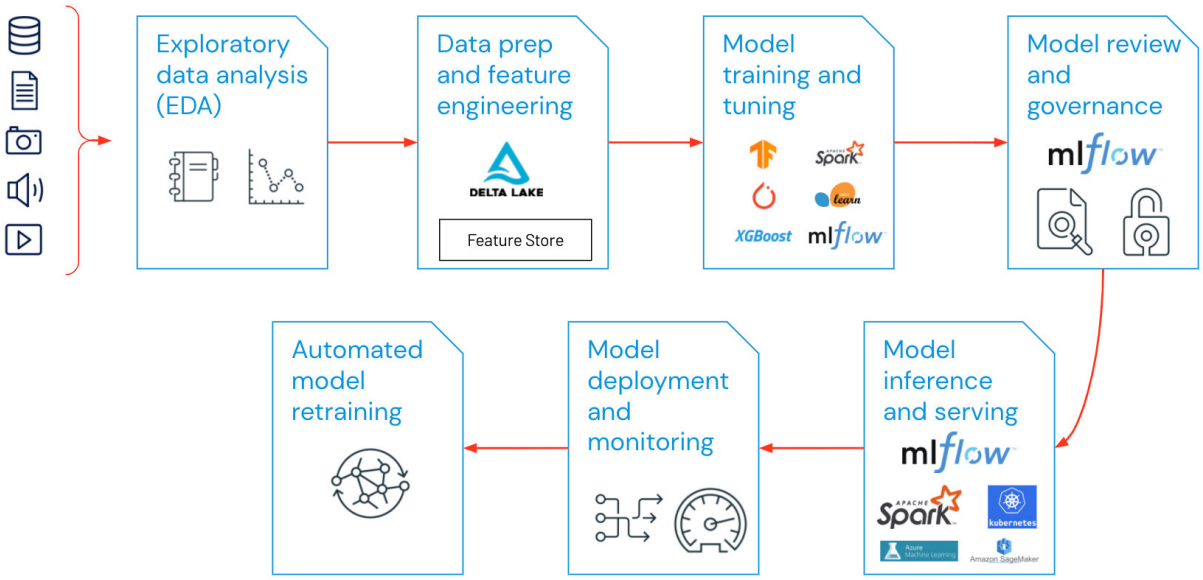

Figura 46 - Componentes de MLOps
Fuente: https://databricks.com/glossary/mlops

9.2.6 ¿Cómo es el proceso?

Las MLOps son compatibles con la entrega rápida de modelos a medida. Una versión resumida del proceso de MLOps incluye los siguientes pasos:

- Crear: esto implica la preparación de datos, la ingeniería de características, la creación de modelos y las pruebas.
- Administrar: después de que se crean los modelos, a menudo se colocan en un repositorio de modelos que se puede auditar y cuenta con control de versiones para promover la reutilización en toda la organización.
- Implementar: este paso implica exportar el modelo o el pipeline, implementarlo e integrarlo en los sistemas y las aplicaciones empresariales de producción.
- Monitorear: se requiere un monitoreo continuo para garantizar un rendimiento óptimo. A medida que los datos evolucionan, se puede volver a entrenar el modelo o se puede reemplazar el existente por uno nuevo.

9.2.7 ¿Qué es una plataforma MLOps?

Es la encargada de proporcionar a los científicos de datos y a los ingenieros de software un entorno de colaboración que facilite la exploración iterativa de datos, las capacidades de trabajo conjunto en tiempo real para el seguimiento de experimentos, la ingeniería de funciones y la gestión de modelos, así como la transición, implementación y supervisión de modelos controlados. Un MLOps automatiza los aspectos operativos y de sincronización del ciclo de vida del aprendizaje automático.

9.2.8 Plataformas en el mercado

Algunas de las plataformas disponibles son:

- Amazon SageMaker (https://aws.amazon.com/es/sagemaker/)
- Azure Machine Learning (https://azure.microsoft.com/en-us/services/machine-learning)
- Vertex: (https://cloud.google.com/vertex-ai)
- Metaflow: (https://metaflow.org)
- MLflow: (https://mlflow.org)

10 Conclusiones

En base a los resultados obtenidos luego de las primeras semanas de la puesta en producción, se llegó a la conclusión de que dos de los tres objetivos propuestos en esta PPS fueron alcanzados.

El primer objetivo consistió en diseñar y desarrollar un sistema que permita mejorar los avisos tempranos de anomalías de calidad de datos y para ello debía implementarse un modelo de aprendizaje automático. Si bien este llevó más tiempo del estipulado (ya que no se estimaron bien los tiempos de aprendizaje de las tecnologías seleccionadas) y las limitaciones que tiene ML.NET, se logró cumplir con el mismo.

El segundo objetivo era el desarrollo y puesta en práctica de una serie de prácticas, normas y/o políticas, que garantizara la disponibilidad, calidad, coherencia, confiabilidad y seguridad de los datos. Si bien se trabajó en la

detección temprana y en acciones correctivas generales (como el uso de notificaciones automáticas a los usuarios), la implementación de normas y políticas correctivas no se logró porque el Product Owner prefirió utilizar parte del tiempo estimado que le fue asignado para corregir y mejorar la performance de la aplicación. Por lo tanto, no se cumplió ya que en el futuro, el cliente deberá brindar más especificaciones (y más detalle) sobre las políticas que quiere poner en práctica para que estas acciones correctivas puedan ser mejoradas, personalizadas y acompañadas con procedimientos correspondientes.

Por último, se buscaba impactar en la satisfacción del cliente brindándole una solución que le permita reducir pérdidas derivadas de malas decisiones. Este objetivo fue cumplido ya que no sólo las pruebas de aceptación fueron realizadas satisfactoriamente, sino que, además, se pactó para el futuro la implementación de un modelo de MLOps que permita mejorar el modelo existente.

En resumen, el proyecto permitió que el cliente obtuviera una nueva herramienta de análisis y, además, lo acercó hacia el desarrollo propio de un modelo de aprendizaje automático, el cual permitirá mejorar avisos tempranos de anomalías para reducir pérdidas derivadas de malas decisiones, minimizar costos y maximizar rentabilidad del negocio.

10.1 Reflexión sobre la práctica profesional supervisada como espacio de formación

En mi opinión siento que la práctica profesional supervisada aportó mucho valor a mi experiencia, tanto como estudiante como desarrollador en una empresa de sistemas. Gracias a ella, y sin trabajar como analista funcional, se me permitió de forma individual administrar, gestionar, organizar y desarrollar un proyecto de inicio a fin con la finalidad de alcanzar los objetivos planteados en la práctica en el tiempo y forma establecidos. También fortaleció mis habilidades blandas y capacidades de interacción con otros sectores de la empresa cliente.

En cuanto a la redacción del informe, fue un gran desafío y por sobre todo, muy interesante ya que si bien los alumnos contamos con algo de experiencia gracias a materias como Ingeniería de Software I, Proyecto de Software y

Administración de Proyectos, ninguno de esos informes tuvo el volumen y magnitud de este.

Finalmente, considero que la PPS es trascendental en nuestra carrera ya que no sólo aporta un gran valor agregado a nuestro desarrollo como profesionales de la informática, sino que también nos permite aplicar todos los conocimientos y habilidades adquiridos a lo largo de estos años.

11 Bibliografía

ADO.NET. Recuperado de https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/. Consulta julio de 2022.

Análisis exploratorio de datos. Recuperado de https://www.ibm.com/es-es/cloud/learn/exploratory-data-analysis. Consulta julio de 2022.

Azure DevOps. Recuperado de https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops. Consulta julio de 2022.
Arquitectura Flux. Recuperado de https://facebook.github.io/flux/. Consulta julio de 2022.

C#. Recuperado de https://docs.microsoft.com/en-us/dotnet/csharp/. Consulta julio de 2022.

Clerici Florencia - Fernández Beatriz (2019). *Herramienta para la evaluación de calidad de datos.* Informe final de Proyecto de Grado. Tutor: Adriana Marotta. Instituto de Computación Facultad de Ingeniería - Universidad de la República Montevideo - Uruguay

Entity Framework. Recuperado de https://docs.microsoft.com/en-us/ef/. Consulta julio de 2022.

NET 5. Recuperado de https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-5. Consulta julio de 2022.

MLOp. Recuperado de https://ml-ops.org/.Consulta julio de 2022.

MLOp. Recuperado de https://www.plainconcepts.com/es/que-es-mlops/.Consulta julio de 2022.

MLOp. Recuperado de https://databricks.com/glossary/mlops.Consulta julio de 2022.

ML.NET. Recuperado de https://docs.microsoft.com/en-us/dotnet/machine-learning/. Consulta julio de 2022.

React. Recuperado de https://es.reactjs.org/docs/getting-started.html. Consulta julio de 2022.