ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

Máster en Big Data: Tecnología y Analítica Avanzada
2020/2021

Trabajo Fin de Máster

"Diseño e implementación de un Feature Store"

Raúl Gómez Serrano

Tutor

Jesús Vicente García Hernández

Fecha de entrega: 29/06/2021

Fecha de defensa: 06/07/2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

DISEÑO E IMPLEMENTACIÓN DE UN FEATURE STORE

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/21 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Raúl Gómez Serrano Fecha: 29/ 06/ 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jesús Vicente García Hernández Fecha: 29/ 06/ 2021

Vº Bº del Coordinador de Proyectos

Fdo.: Carlos Morrás Ruiz-Falcó Fecha: 29/ 06/ 2021

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Raúl Gómez Serrano

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Diseño e implementación de un Feature Store,

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que

pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 29 de junio de 2021

ACEPTA

Fdo

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

RESUMEN

En el presente documento se desarrolla el proceso de trabajo del diseño y la implementación de un Feature Store (1).

Para ello, en primer lugar, se realiza un estudio del contexto actual, en el que se refleja que las necesidades de almacenamiento de variables para la posterior implementación de modelos son cada vez mayores, haciéndose patente la necesidad de desarrollar nuevas herramientas que ayuden en esta labor. Además, se definen los objetivos, tanto a nivel de implementación como de adquisición de conocimiento por parte del alumno.

También se define la gestión del proyecto, indicando cuál es la organización del equipo de trabajo, la planificación a lo largo de los seis meses y un presupuesto teórico de cuál habría sido el coste del proyecto.

Más adelante, se lleva a cabo un análisis de muchas de las alternativas existentes en el mercado que podrían competir con la herramienta a desarrollar, obteniendo una serie de elementos básicos que debe tener y comprobando si cada una de estas alternativas cumple con dichos elementos. También se lleva a cabo un profundo estudio de las diferentes alternativas tecnológicas existentes para cada una de las piezas a implementar.

Después, se realiza el análisis del sistema, donde especificando los casos de uso y los requisitos que deberá cumplir el sistema. Es aquí donde se define la funcionalidad del sistema, desde su despliegue mediante contenedores, las funciones que permitirán interactuar con el mismo y el modelo de datos necesario.

Más adelante, se plantea un diseño del sistema, estudiado cuál es la forma idónea de implementar el sistema analizado previamente. Este diseño determina qué componentes se deben desarrollar y de qué manera implementarán las funcionalidades necesarias, indicando las librerías y decisiones tomadas.

A continuación, tras haber implementado del sistema, se realiza una batería de pruebas que permita comprobar que el resultado obtenido en cada una de ellas es el esperado.

Finalmente, se desarrolla una conclusión, donde se plantean los principales problemas que se han encontrado, se listan posibles mejoras que realizar a la herramienta y se plantean alternativas para obtener beneficio económico de este proyecto.

ABSTRACT

This document develops the workflow of designing and implementing a Feature Store (1).

To do this, first of all, a previous study of the current context will be carried out, in which it is verified that the needs of storage of variables for the subsequent implementation of models are increasing, becoming evident the need to develop new tools that help in this work. In addition, the objectives are defined, both at the level of implementation and acquisition of knowledge by the student.

It also defines the management of the project, indicating what is the organization of the work team, the planning over the six months and a theoretical budget of what the cost of the project would have been.

Next, an in-depth analysis of many of the existing alternatives on the market that could compete with the tool to be developed is carried out, obtaining a series of basic elements that it must have and checking if each of these alternatives complies with those elements. There is also a study of the different technological alternatives that exist for each of the pieces to be implemented.

A system analysis is then performed, specifying the use cases and system requirements. It is here that the functionality of the system is defined, from its deployment through containers, the functions that will allow interaction with it and the necessary data model.

Later, a design of the system is proposed, studied what is the correct way to implement the analyzed system. This design determines which components should be developed and how they will implement the necessary functionalities, indicating the libraries and decisions made.

Then, after having carried out the implementation of the system, a battery of tests are proposed which allows to verify the results obtained are the expected ones.

Finally, a conclusion is developed, where the main problems encountered are discussed, possible improvements to be made to the application and alternatives are proposed to take economic advantage of this project.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El presente documento tiene como objetivo mostrar el desarrollo de un trabajo de fin de máster de Big Data: Tecnología y Analítica Avanzada. Se trata de un trabajo donde se ponen en práctica algunos de los conocimientos adquiridos durante el máster, principalmente relacionados con la rama de ingeniería de datos. Además, se adquieren conocimientos nuevos relacionados con la gestión de proyectos, uso de tecnologías cloud, orquestación y organización de procesos y despliegue de contenedores para la virtualización de la herramienta.

El trabajo consiste en el diseño e implementación de un Feature Store. Dentro del ciclo de vida de los datos, en un contexto donde las técnicas de Machine Learning o Inteligencia Artificial son primordiales para la empresa, disponer de un sistema de gobierno de las características o features empleadas para entrenar los modelos resulta clave. No sólo facilita la gobernanza y reproducibilidad de modelos, sino que redunda en eficiencias asociadas a la convergencia de uso de esas características, limitando las duplicidades o, peor aún, la disparidad de obtención de los mismos conceptos por distintos equipos.

El Feature Store, más allá de un simple repositorio de datos preparados y listos para ser usados, es el catálogo de características empleables por distintos colectivos en una misma organización donde toda información asociada a conjuntos de datos dispone de registro sobre su descripción, distintas versiones empleadas durante la vida útil del conjunto y el perfil de los datos, de modo que cualquier incidente relativo a la calidad de estos pueda ser informado.

Más allá, haciendo un uso consciente de estos metadatos, la capacidad de automatización redunda en un ecosistema más eficiente, pudiendo así planificar la preparación de datos o alertas de modo que el impacto en los equipos de ingeniería resulte mínimo, a la par que se agiliza la autonomía de los equipos dedicados a la construcción de modelos.

Para el desarrollo del documento se han tenido en cuenta guías relacionadas con la realización de Trabajos de Fin de Grado/Máster (2). También se ha tomado como ejemplo el TFG realizado por el alumno en sus estudios de grado (3), con el objetivo de definir de forma clara los apartados a realizar y el contenido de los mismos. Varios de los apartados del presente trabajo han sido reutilizados de dicho TFG, si bien se han realizado los cambios necesarios para adaptarlos al sistema a desarrollar.

1.1. Contexto social actual.

Hoy en día el gobierno y el uso de los datos es clave para cualquier compañía que quiera mejorar su rendimiento y obtener mayores beneficios. Conforme pasan los años, la cantidad de datos que se manejan en el mundo es cada vez mayor, siendo la previsión para 2025 de ciento ochenta y un zettabytes (10^{21} bytes) de datos manejados:

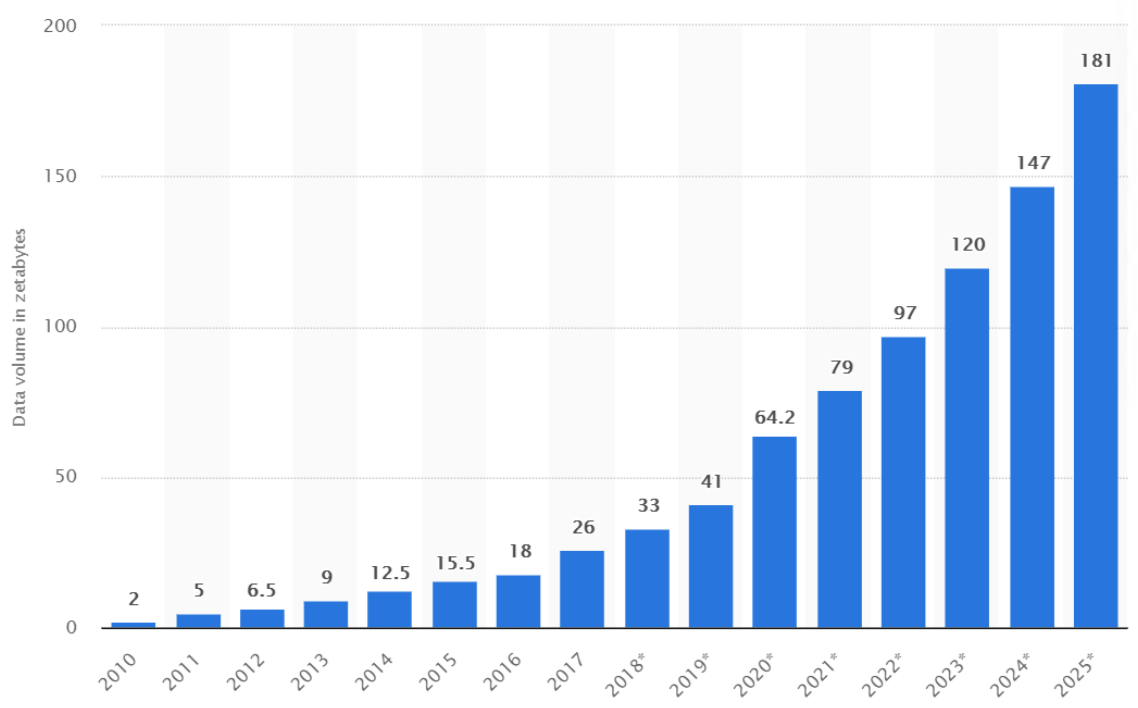

Ilustración 1: Volume of data managed worldwide from 2010 to 2025 (4)

Dentro de estas necesidades surge el Big Data como concepto, cantidades masivas de datos que, por su complejidad, velocidad de crecimiento o dificultad para ser capturados, necesitan de técnicas y aplicaciones informáticas no tradicionales para su procesamiento y almacenado. Con una buena gestión de estos datos, es posible obtener información que permita dirigir o guiar las decisiones de una empresa.

Desde la aparición del Big Data, ha evolucionado mucho, siendo cada vez mayores las necesidades que se generan en el sector. Una de ellas, es la comunicación entre los distintos elementos involucrados en la vida de los datos. Desde la obtención de los mismos, su transformación, su utilización en modelos, la visualización de los resultados de forma clara y precisa, hasta la toma de decisiones en base a ellos. Se trata de procesos largos, que pasan por muchas fases, y en las que la comunicación entre ellas es clave para obtener un producto final, es decir, información, con la máxima calidad posible. Es por ello, que, entre muchas soluciones alcanzadas, se ha definido la idea de Feature Store.

Este concepto ha sido utilizado y desarrollado por múltiples empresas, como Google, Uber, Amazon, Netflix, etc. Estas compañías cuentan con una cantidad masiva de datos, muchos de los cuales deben ser procesados en tiempo real.

A raíz de estas necesidades, se han desarrollado también soluciones parcialmente open source, como Hopsworks o Feast, que servirán de ejemplo e inspiración para este trabajo.

Sin embargo, no cuentan con todas las funcionalidades deseables que debería tener un Feature Store, tal y como se explicará en posteriores apartados.

Por todo ello, este proyecto pretende solventar una de estas necesidades de comunicación, como es proveer a los científicos de datos, es decir, las personas encargadas de definir y ejecutar los modelos, con una plataforma que les ayude a la gobernanza y cálculo de las variables o características necesarias. Con esta herramienta, se reduce notablemente el proceso de obtención y limpieza de los datos a ser utilizados, siendo tan sólo necesario acceder al repositorio global con las features ya definidas. Además, el trabajo pretende mejorar las capacidades de otras herramientas similares y proveer a las compañías de un producto completo, con todas las características necesarias para su despliegue.

1.2. Motivación del Trabajo.

El presente trabajo tiene como motivación profundizar en el mundo de la ingeniería de datos y aumentar el conocimiento en esta rama del Big Data. Siendo los estudios previos del alumno relacionados con el mundo de la informática, esta es la rama con más relación a estos, por lo que supone un impulso extra de interés en la profundización de la materia.

Además, al adquirir cierta experiencia dentro de la empresa y ver las necesidades que existen, se descubre que esta herramienta es un motor muy potente para impulsar el uso del Big Data en las compañías, aportando un punto de partida donde gobernar los datos y realizar una buena gestión de los mismos. Esta visión es la misma que se comparte en la empresa donde se realiza el desarrollo del TFM, impulsora de esta idea y con un gran interés es las nuevas tecnologías y metodologías que surgen alrededor de este campo.

Por ello, con la motivación de desarrollar una carrera profesional relacionada con los datos, y más concretamente en el perfil de ingeniero de datos, se decide desarrollar el proyecto de un Feature Store, para adquirir conocimientos y experiencia real que sirvan en un futuro laboral.

1.3. Objetivos.

Por lo tanto, los objetivos del proyecto son:

- Desarrollar un Feature Store tomando como referencia los ya existentes.
- Aplicar los conocimientos adquiridos a lo largo del máster, tanto a nivel de programación como a nivel de documentación y gestión de proyectos.
- El almacenamiento versionado e histórico de las features.
- La capacidad de definir nuevas features, de forma que sean calculadas de forma periódica y almacenadas en un repositorio común.
- La capacidad de actualizar las definiciones de las features.
- La posibilidad de obtener los valores de las features, tanto las más actuales como en un momento temporal concreto.
- La posibilidad de agrupar las features en grupos con características comunes.
- La posibilidad de deshabilitar features una vez no sea necesario su cálculo periódico.

1.4. Estructura del documento.

El presente documento está formado de los siguientes capítulos:

- **Introducción**: en este apartado se muestra una primera visión del proyecto, en el que se definen las ideas principales en cuanto al contexto social y la motivación detrás de este trabajo. Además, se incluyen los objetivos que se quieren conseguir con el desarrollo de la herramienta y se describe la estructura del proyecto.
- **Definición y acrónimos**: este apartado define los términos más utilizados y técnicos del documento, así como la explicación de los acrónimos empleados.
- **Gestión del proyecto**: se especifican las tareas necesarias para desarrollar el proyecto, y la organización de cada una de ellas. Además, se incluye un apartado para explicar el marco regulador y las leyes que aplican al proyecto.
- **Estado del arte**: se realiza un estudio del mercado actual, analizando posibles competidores de la herramienta propuesta y finalizando con una propuesta del sistema que se va a implementar.
- **Análisis del sistema**: se realiza un análisis previo a la implementación del sistema, especificando los casos de uso y los requisitos que se han detectado a lo largo de las distintas iteraciones.
- **Diseño del sistema**: en este apartado se desarrolla la herramienta definida en el apartado de análisis. Se explica el proceso que se ha llevado a cabo en el desarrollo y el funcionamiento del sistema, así como la arquitectura y el modelo de datos empleado.
- **Validación del sistema**: se realizan una serie de tests o pruebas que permitan comprobar el correcto funcionamiento y que la herramienta final cumple con los objetivos definidos en el análisis del sistema.
- **Conclusiones**: se exponen las conclusiones, mejoras futuras e idas de comercialización obtenidas tras el desarrollo de la herramienta y de la documentación.
- **Bibliografía**: en este capítulo se desglosan todos los artículos, páginas web, y recursos en los que se apoyado el alumno como ayuda para el desarrollo del trabajo.
- **Abstract**: parte escrita en inglés que resume en pocas páginas los elementos principales del trabajo.

1.5. Definiciones y acrónimos.

1.5.1. Definiciones.

Feature: cada una de las variables que componen el dataset que es utilizado para entrenar un modelo de machine learning.

Machine Learning: se trata de un campo de las ciencias de la computación que tiene como finalidad es el desarrollo de técnicas que permitan aprender a los ordenadores.

Open source: se trata de un modelo de desarrollo de software que se basa en la colaboración abierta entre todas las personas que quieran participar. En muchas ocasiones también hace referencia a poder adquirir el software de forma gratuita.

Data driven: se dice de una compañía en la que impera el uso de datos y la toma de decisiones se basa en la información que aportan esos datos.

On-premise: se dice cuando la instalación del software se realiza en las instalaciones o infraestructura de la empresa, siendo responsable de la máquina donde se encuentre, su seguridad, mantenimiento, etc.

Cloud: se trata de servidores remotos, mantenidos por otra compañía, y en los que el usuario no debe preocuparse de su mantenimiento.

Batch: también conocido como sistema por lotes, se trata de la ejecución de un programa sin el control de un usuario.

1.5.2. Acrónimos.

Acrónimo	Significado
API	Application Programing Interface
AWS	Amazon Web Service
CSV	Comma Separated Values
DAG	Grafo Acíclico Dirigido
DSL	Domain Specific Language
GB	Gigabyte
GDPR	Reglamento General de Protección de Datos
HTTP	Protocolo de transferencia de hipertexto

ICAI	Instituto Católico de Artes e Industrias
IDE	Entorno de Desarrollo Integrado
IU	Interfaz de Usuario
IVA	Impuesto sobre el Valor Añadido
JSON	JavaScript Object Notation
LOPD	Ley Orgánica de Protección de Datos
ML	Machine Learning
MPP	Massively Parallel Processing
OS	Operating System
RAM	Random Memory Acces
SDK	Software Development Kit
SQL	Structured Query Language
SSH	Secure Shell
TFM	Trabajo de Fin de Master

Tabla 1: Acrónimos

2. GESTIÓN DEL PROYECTO

2.1. Metodología.

El presente proyecto se desarrolla en base a una adaptación de la metodología Agile (5).

Dicha metodología permite desarrollar proyectos relacionados con la ingeniería del software de forma iterativa e incremental, donde los requisitos del producto y las soluciones propuestas evolucionan según las necesidades del proyecto. Dado que el trabajo es realizado por el alumno de forma individual, no existe un equipo de trabajo completo, con scrum master y otros elementos propios de la metodología. Es por ello que se ha utilizado una adaptación.

En esta metodología se han empleado las siguientes características:

- Desarrollo del trabajo de forma individual y remota por parte del alumno.
- Reuniones periódicas con el tutor en la empresa, siendo más frecuentes en el comienzo del proyecto para definir las tareas a acometer y menos frecuentes según fue posible realizar un trabajo más autónomo por parte del alumno.
- Reuniones con los jefes de departamento para realizar un seguimiento de los avances y encontrar un equilibrio entre las necesidades de la empresa y las soluciones propuestas por el alumno.
- La utilización de la herramienta de administración de tareas Jira para la definición de tareas a acometer y tiempo empleado en las mismas. Esta herramienta es muy útil para visualizar el progreso de un proyecto y repartir el trabajo entre los miembros del equipo. Sin embargo, dada la naturaleza individual del trabajo, se ha utilizado como seguimiento por parte de los jefes del proyecto, así como para definir el alcance y posibles tareas a acometer, aunque no todas se hayan llevado a cabo finalmente.

En la elaboración de este proyecto se han tenido en cuenta todos los estándares de calidad y software existentes en el momento de desarrollo del mismo, por lo que se respetarán todas las normas y requisitos establecidos.

2.2. Organización.

Dada la naturaleza del presente proyecto, cuyo objetivo es el desarrollo de un trabajo de fin de máster, la organización se compone por un solo individuo, que toma la posición de los distintos roles que tendrían que constituir el equipo de trabajo de desarrollo de un producto software.

En este apartado se establecen la estimación de recursos iniciales del proyecto y la organización del trabajo utilizada.

2.2.1. Estimación de recursos iniciales.

Para el desarrollo de este proyecto serán necesarios los siguientes recursos:

2.2.1.1. Recursos hardware.

- Un ordenador portátil.

2.2.1.2.Recursos software.

- Un entorno de desarrollo integrado (IDE).
- Un software de despliegue de aplicaciones dentro de contenedores.
- Una base de datos.
- Una herramienta de administración de bases de datos.
- Un software de orquestación de procesos.
- Herramientas de diseño.
- Herramienta de escritura de documentos.

2.2.1.3.Recursos personales.

El alumno será la única persona involucrada en el desarrollo de este proyecto, con la asistencia del tutor y otros compañeros de la empresa, que guiarán los pasos a seguir y aportarán conocimiento y experiencia previas en desarrollo software.

2.2.2. Organización del trabajo.

En este apartado se analiza la organización del trabajo con el objetivo de que el resultado final del proyecto sea de calidad y se haga de forma estructurada.

En este caso, el papel de cliente lo desempeñan los jefes de departamento, mientras que el jefe de proyecto es el propio alumno. Mediante comunicación vía mensajería y videollamadas, el cliente y el jefe de proyecto discuten y negocian los aspectos más relevantes del sistema y su desarrollo.

El papel de analista es, de nuevo, desempeñado por el alumno, encargado de la definición de los requisitos y necesidades del sistema conforme se avanza en el proyecto y se definen las soluciones. No obstante, a lo largo de las diferentes iteraciones y reuniones con los encargados de la empresa se han ido refinando dichos requisitos para cumplir con las necesidades que se le han pedido al alumno. Además, se realiza un análisis de la estructura del sistema.

El alumno desempeña también el papel de diseñador, tomando las decisiones de cómo resolver el problema y encontrando soluciones para desarrollarlo, aunque siempre guiado por los conocimientos de los miembros de la empresa, que aconsejan en todo momento cuáles son las mejores tecnologías para afrontar el proyecto. Además, también realiza las pruebas necesarias para asegurar que el sistema diseñado funciona tal y como se describe en el apartado de análisis.

Por último, la puesta en producción del sistema también será tarea del alumno, siendo el encargado del posterior mantenimiento del mismo y de la resolución de fallos.

2.3. Planificación.

La planificación se va a llevar a cabo mediante un diagrama de Gantt en la herramienta Jira. Debido a la metodología ágil del proyecto, este diagrama ha sufrido modificaciones a lo largo del mismo, por lo que el diagrama mostrado a continuación es el resultado final de las distintas iteraciones o sprints:

Ilustración 2: Diagrama de Gantt

Como se puede observar en la captura anterior, el proyecto se ha desarrollado a lo largo de seis meses.

Las primeras tareas que se realizaron fueron las de definición y análisis. Esta fase fue muy importante, pues supuso la comprensión de las características del proyecto, la investigación de aplicaciones similares para conocer su enfoque hacia el problema y para determinar las piezas base que se iban a tener que desarrollar.

Una vez se tenía este conocimiento, el siguiente paso era realizar el diseño de la arquitectura, determinando qué aspectos se iban a implementar y cuáles se dejaban fuera del proyecto.

Mientras se realizaba esta tarea también se realizó el diseño de metadatos, una de las tareas más complejas y lentas, pues a cada avance en el proyecto había que replantear de cero el enfoque que se había hecho.

Las funciones de operación y orquestación son las que más tiempo han llevado, es decir, la implementación del código, asegurando que todo funcionase correctamente y realizando un aprendizaje paralelo de todas las librerías y tecnologías involucradas, de las que no se tenía conocimiento previo.

Los motores de transformación también son una parte importante, porque definen qué operaciones se pueden realizar y de qué forma se realizan.

Finalmente, el DSL es otra pieza clave para poder definir las features con los metadatos.

2.4. Marco regulador (6).

En el desarrollo de una herramienta que almacena datos se deben tener en cuenta diferentes restricciones y leyes que puedan afectar al proyecto, tanto de carácter económico, como operativo, técnico o legal.

En primer lugar, con respecto a las restricciones económicas, la empresa donde se desarrolla el proyecto es la encargada de aportar la remuneración en formato de beca y tiene fondos suficientes para cubrir esta necesidad, por lo que no existe un límite económico que pueda afectar. No obstante, se debe contar con los recursos hardware necesarios, como es un ordenador portátil, también aportado por la empresa. En cuanto al software, los programas utilizados son de uso gratuito, por lo que no suponen un gasto adicional en el proyecto.

Con respecto a las restricciones de carácter técnico y operativo, se encuentran la falta de conocimiento previo de las herramientas que se van a utilizar y de la naturaleza del sistema que se va a desarrollar, siendo completamente nueva para el alumno. Esto implica un estudio previo a la fase de implementación para aprender dichas herramientas, con la finalidad de asegurar un desarrollo de calidad. Además, aunque los lenguajes de programación utilizados han sido vistos previamente por el alumno, muchas de las novedades como las librerías utilizadas son cuestiones que se deben abordar.

En referencia a las restricciones legales del proyecto, se debe sujetar a la Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPD) y al Reglamento de Protección de Datos (GDPR). La herramienta puede guardar información personal como nombres, correos, etc., por lo que se debe asegurar un correcto almacenamiento de los datos para evitar así accesos externos a los permitidos. Por ello, se deben estudiar las alternativas de almacenamiento que garanticen esta seguridad.

Finalmente, se debe contar con las licencias de todas las herramientas y productos que se utilicen a lo largo del desarrollo del sistema, así como imágenes utilizadas libres de derechos de autor y código libre obtenido en línea o de otros autores.

2.5. Entorno socioeconómico.

Una vez que se conoce la planificación del proyecto y el marco regulador para desarrollarlo, es necesario estimar el coste de este. Dado que el trabajo se ha realizado dentro de una empresa en formato de prácticas remuneradas, se indicarán las horas empleadas en el proyecto y el salario percibido durante las mismas.

2.5.1. Estimación de costes no personales.

Es necesario analizar el hardware y herramientas utilizadas para calcular el coste de estas:

En primer lugar, se analizan las herramientas software empleadas, y en caso de que alguna fuese de cobro, se incluiría su precio. Los softwares utilizados son los siguientes:

- Sistema operativo: Windows 10.
- Sistema de gestión de entornos Python: Anaconda.
- Entrono de desarrollo integrado: Visual Studio.
- Edición de código: Notepad++ y Visual Studio.
- Navegador web: Google Chrome.
- Herramienta de creación de contenedores: Docker Desktop (7).
- Herramienta de orquestación y monitorización: Apache Airflow (8).
- Base de datos: PostgreSQL (9) y Snowflake (10).
- Gestor de bases de datos: DBeaver (11).
- Herramienta de escritura de documentación: Microsoft Word.
- Diseño de diagramas e ilustraciones: draw.io.
- Herramienta de recortes: aplicación por defecto del ordenador.

Todas y cada una de estas herramientas son de carácter gratuito o tienen una prueba gratuita limitada mayor al tiempo utilizado por el alumno. De estas herramientas, la única que podría incurrir en gastos es la base de datos cloud Snowflake, pero cuenta con un período de prueba gratuito de treinta días, tiempo suficiente para realizar las pruebas y desarrollos necesarios.

Componente software	Precio
Windows 10	0,00€
Anaconda	0,00€
Visual Studio	0,00€
Notepad++	0,00€
Google Chrome	0,00€
Docker Desktop	0,00€
Airflow	0,00€
PostgreSQL	0,00€
Snowflake	0,00€
Draw.io	0,00€
Microsoft Word	0,00€
TOTAL	**0,00€**

Tabla 2: Estimación de costes software

Con respecto a los componentes hardware, se ha utilizado el ordenador portátil proporcionado por la empresa. El precio de este dispositivo se especifica en la siguiente tabla:

Componente hardware	Precio
Ordenador portátil Lenovo ThinkPad con procesador Intel i7, 32GB de RAM y 512GB de almacenamiento SSD.	1299,99€
TOTAL	**1299,99€**

Tabla 3: Estimación de costes hardware

2.5.2. Estimación de costes personales.

Tal y como se ha comentado anteriormente, el trabajo se ha desarrollado en empresa, dedicando un total de ciento quince días laborales y cinco horas diarias de trabajo. En base a estos datos, se estima el número de horas.

2.5.2.1. Estimación de horas.

En primer lugar, se definen cuántos roles componen el equipo de trabajo, y se calculan el número de horas que dedica cada uno:

Cargo	Nombre	Horas de trabajo
Jefe de proyecto	Raúl Gómez Serrano	80
Analista	Raúl Gómez Serrano	130
Diseñador	Raúl Gómez Serrano	160
Programador	Raúl Gómez Serrano	180
Gestor de pruebas	Raúl Gómez Serrano	25
TOTAL		**575**

Tabla 4: Estimación horas equipo de trabajo

2.5.2.2.Estimación del presupuesto personal.

En función de las horas calculadas anteriormente y de las tareas que son necesarias para realizar el proyecto se calculan los sueldos de los componentes del equipo. Este sueldo se establece en función de la remuneración recibida en la empresa:

Tareas	Encargado	Horas de trabajo	€/h
Gestión del proyecto	Jefe de proyecto	30	7,50€/h
Planificación	Jefe de proyecto	10	7,50€/h
Introducción	Jefe de proyecto	5	7,50€/h
Estudio de viabilidad	Jefe de proyecto	15	7,50€/h
Análisis del sistema	Analista	130	7,50€/h
Diseño del sistema	Diseñador	160	7,50€/h
Implementación del sistema	Programador	180	7,50€/h
Pruebas del sistema	Gestor de pruebas	25	7,50€/h
Documentación	Jefe de proyecto	20	7,50€/h

Tabla 5: Estimación de horas y precio por tarea

Finalmente, el presupuesto por el personal será el siguiente:

Cargo	Nombre	Horas de trabajo	Sueldo
Jefe de proyecto	Raúl Gómez Serrano	80	600,00€
Analista	Raúl Gómez Serrano	130	975,00€
Diseñador	Raúl Gómez Serrano	160	1200,00€
Programador	Raúl Gómez Serrano	180	1350,00€
Gestor de pruebas	Raúl Gómez Serrano	25	187,50€
		TOTAL	4312,50€

Tabla 6: Estimación sueldo del personal

2.5.3. Estimación de costes indirectos.

Para el desarrollo de este proyecto serán necesarios el alquiler de una oficina o lugar de reunión para el equipo y los gastos extra que esto supone, como luz, agua, etc.. Teniendo en cuenta la duración de seis meses del proyecto, los presupuestos son los siguientes:

Concepto	Gasto por mes	Gasto total
Alquiler del local	1750,00€	10500,00€
Factura eléctrica	230,00€	1380,00€
Factura del agua	65,00€	390,00€
Factura de internet	115,00€	690,00€
Factura telefónica	120,00€	720,00€
Limpieza y mantenimiento	350,00€	2100,00€
TOTAL		15780,00€

Tabla 7: Estimación de costes indirectos

Ya que se trata de un proyecto de carácter académico que se ha desarrollado en remoto, no se ha incurrido en gastos indirectos.

2.5.4. Margen de riesgo y beneficio.

Concepto	Porcentaje	Gasto total
Costes no personales		1299,99€
Costes personales		4312,50€
Costes indirectos		15780,00€
Coste total del proyecto		21392,49€
Margen de beneficio	20%	+4278,50€
Margen de riesgo	20%	+4278,50€
IVA	21%	+4492,42€
PRESUPUESTO TOTAL		34441,91€

Tabla 8: Presupuesto total del proyecto

El presupuesto total del proyecto es treinta y cuatro mil cuatrocientos cuarenta y un euros con noventa y un céntimos.

3. ESTADO DEL ARTE

3.1. Crítica al estado del arte.

El mercado de soluciones y herramientas Big Data es muy amplio hoy en día, siendo cada vez mayor debido a la gran demanda por parte de las compañías de convertirse en data driven.

Entre todas estas herramientas, el término Feature Store surge en los últimos años como un sistema capaz de ayudar a las compañías a alcanzar ese objetivo de ser manejadas por la información que aportan los datos, siendo un elemento centralizador y con múltiples herramientas de gestión. Sin embargo, tal y como se verá en los siguientes apartados, las soluciones propuestas que se encuentran hoy en día en el mercado no son lo suficientemente maduras y completas como para que supongan un gran cambio dentro de las compañías. Es por ello que se plantea este proyecto, con el objetivo de desarrollar un Feature Store completo, con todas las funcionalidades necesarias para ser implantado en una empresa y comenzar a ser útil desde el principio.

3.2. Estudio del mercado actual.

En este apartado se muestran todas las alternativas que existen en el mercado, que cada vez son mayores y han ido aumentando incluso en el transcurso del desarrollo de este proyecto. De cada una de ellas se comentarán los aspectos más relevantes para finalmente realizar una comparación en formato tabular que permita una rápida visualización de las capacidades y carencias de cada una de ellas.

3.2.1. Herramientas competidoras.

3.2.1.1. Hopsworks (12).

Se trata de una plataforma open-source para el almacenamiento, gobierno e histórico de features, disponible tanto on-premise como en las plataformas cloud AWS y Azure. Se trata de una herramienta muy completa, con interfaz de usuario para el descubrimiento de features y reglas de calidad para notificar posibles errores en los campos. La ingesta de datos puede ser tanto batch como streaming. Sin embargo, se trata de un sistema de almacenamiento de características con algunas funcionalidades extra, pero no cuenta con un sistema de procesamiento que permita calcular las features de forma periódica, así como definir su método de cálculo. Carece por tanto de una de las características más interesantes de un feature store.

3.2.1.2. Feast (13).

Se trata de otra plataforma similar, también open-source, que permite el almacenamiento de las features. No cuenta con interfaz de usuario, por lo que todas las funcionalidades se ejecutan a través de un SDK que implementa distintas funciones. Cuenta con almacenamiento tanto online como offline y la posibilidad de ingestar datos batch y

streaming. De nuevo, no cuenta con la posibilidad de definir las features y planificar el cálculo de estas de forma periódica.

3.2.1.3. Amazon SageMaker (14).

Plataforma muy similar a las anteriores desarrollada por Amazon y de software libre. Permite a los desarrolladores y a los científicos de datos preparar, crear, entrenar e implementar modelos de forma sencilla. Cuenta con muchas características interesantes que complementan el almacén de las features. Funciona en gran medida con el ecosistema de Amazon, utilizando AWS Batch para el cálculo periódico y Amazon Kinesis para el procesado en tiempo real. Sin embargo, este procesado se realiza y orquesta mediante otra herramienta, Data Wrangler, también de Amazon. Por ello, no constituye una herramienta completa con las características deseables de un feature store.

3.2.1.4. Tecton (15).

Se trata de una plataforma muy completa, que no sólo cuenta con las funcionalidades de almacenamiento de features, sino que también permite la definición de nuevas características de forma sencilla en base a un esquema. De esta forma, se añade una nueva funcionalidad muy relevante, como es el cálculo periódico o en tiempo real de datos en crudo obtenidos del exterior para su posterior volcado en el almacenamiento global. Se trata por lo tanto de la primera alternativa completa de los requisitos básicos que se esperan en un Feature Store. Sin embargo, la herramienta es de pago, por lo que puede no llegar a ser la mejor de las alternativas.

3.2.1.5. Vertex AI (16).

Se trata del Feature Store desarrollado por Google, que toma Google Cloud como almacenamiento. No obstante, se trata de una solución muy poco madura, que se encuentra en desarrollo, y que aún no cuenta con la posibilidad de calcular las features. Por lo tanto, se trata de un simple almacenamiento o repositorio global de características, sin completar el resto de las funciones necesarias.

3.2.1.6. Michelangelo Pallette (17).

Se trata del Feature Store desarrollado por Uber. Esta plataforma es muy completa, permitiendo todo el procesado de features que se generan en tiempo real en todo el mundo con los diferentes servicios de la compañía, tanto en el sistema de taxis como en el reparto de comida a domicilio. Sin embargo, no es de acceso libre, por lo que no supone una alternativa viable a la hora de implementar un Feature Store en una compañía diferente.

3.2.2. Características deseables.

Una vez comparadas todas las alternativas, podemos identificar una serie de características deseables que debe tener un Feature Store:

- **Domain Specific Language (DSL)**: Deberá disponer de un lenguaje de dominio de sintaxis sencilla que permita definir a grandes rasgos las operaciones a realizar y los pormenores de estas de cara a disponibilidad y frecuencia.
- **Acceso centralizado**: Deberá plantear una arquitectura que permita el rápido acceso de las features y de forma única.
- **Serving**: El sistema permitirá interactuar con él de tal forma que sea sencillo obtener las features calculadas en el mismo o implementar unas nuevas a través del DSL.
- **Versionado y time-travel**: Dado que las features forman parte del proceso iterativo de los data scientist durante la fase de Discovery, el Feature Store deberá disponer de mecanismos de versionado de las features así como de time-travel de cara a garantizar su reproducibilidad.
- **Monitorización**: Dado que son procesos automatizados sobre datos "en movimiento" deberá existir mecanismo que monitorice o informe tanto de la evolución de los datos como de la potencial modificación de la calidad de las features consolidadas en el almacén.

3.2.3. Comparativa final.

En base a estas características deseables y la comparativa de cada herramienta, se puede observar en forma de tabla una visión global de todo lo comentado:

Elemento a comparar	HopsWorks	Feast	SageMaker	Tecton	Vertex AI
Source	Batch y Stream	Batch y Stream	Batch y Stream	Batch y Stream	Batch y Stream
Procesos ETL	No	No	No	Sí	No
DSL	No	No	No	Sí	No
Serving	Sí	Sí	Sí	Sí	Sí
Time travel	Sí	Sí	Sí	Sí	Sí
Monitorización	Sí	No	Sí	Sí	No

Tabla 9: Comparativa de Feature Stores del mercado

3.3. Propuesta.

Por lo tanto, viendo los resultados obtenidos en la comparativa anterior, podemos determinar que no existe ninguna herramienta lo suficientemente completa que abarque las necesidades detectadas. La falta de procesamiento por parte de la mayoría de ellas, así como el coste de algunas otras hace que no sean las alternativas más completas.

Por ello, se propone realizar el desarrollo de un Feature Store propio, realizado desde cero, adaptado a las necesidades actuales y con las tecnologías más modernas posibles. El objetivo principal del proyecto es el desarrollo de un producto mínimo viable, que permita sentar una base de lo que en un futuro podría ofrecerse a las compañías en forma de sistema completo.

Este sistema debe contar con la principal carencia estudiada en las alternativas del mercado, como es el procesado de los datos, para evitar así problemas relacionados con la comunicación entre los encargados del procesado de datos y los encargados de la creación y ejecución de modelos.

3.4. Estudio de las alternativas de diseño.

Tras un estudio del mercado actual y los posibles competidores, se ha obtenido una propuesta genérica de los requisitos con los que se trabajará en el sistema. En este apartado, en primer lugar, se describen cada uno de los componentes del sistema, para posteriormente realizar un estudio de las distintas alternativas que existen para cada uno de ellos. Para ello, se tiene en cuenta el siguiente diagrama de la arquitectura de un Feature Store, que muestra de forma simple los principales elementos que lo componen:

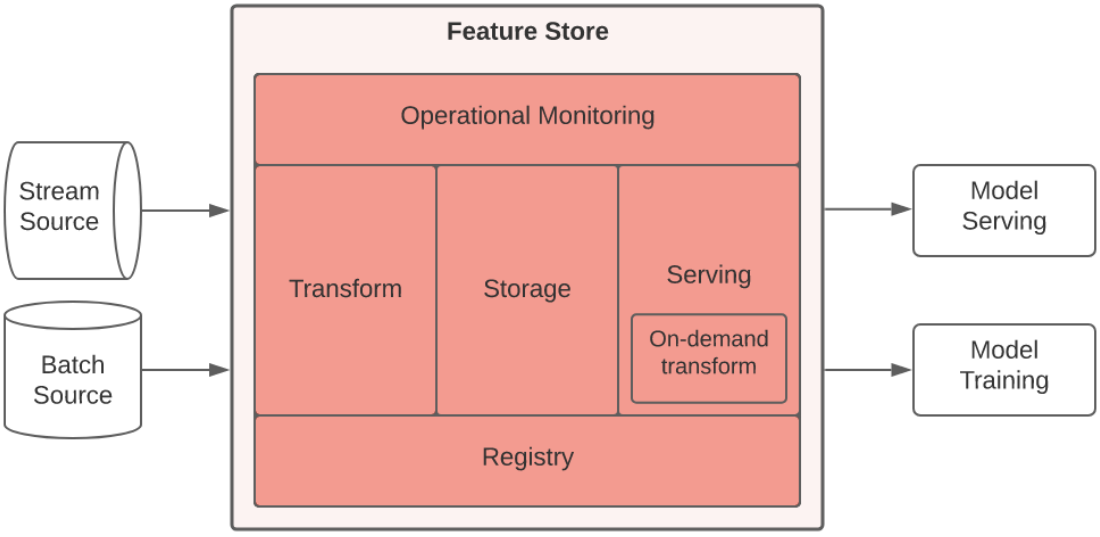

Ilustración 3: Arquitectura principal de un Feature Store (18)

- **Batch Source**: se trata del origen de los datos en batch, es decir, un sistema de almacenamiento que contiene los datos en crudo que serán necesarios para calcular las features. Este sistema de almacenamiento podrá verse actualizado cada cierto período de tiempo, siendo necesario actualizar también las features que se nutren del mismo. El sistema debe ser capaz de almacenar información de

forma ordenada, ser accesible por la pieza de procesamiento y ofrecer las medidas necesarias para asegurar la disponibilidad y seguridad de los datos en todo momento.

- **Stream Source**: se trata del origen de los datos en tiempo real, es decir, la llegada de eventos que se están generando en ese mismo momento Estos datos es posible que no pasen por un almacenamiento previo a la transformación de los mismos en features, por lo que habrá que proveer de un sistema de procesado potente que no suponga un cuello de botella.
- **Procesado**: una vez se encuentran los datos en crudo almacenados, es necesario transformarlos para generar las features. Las transformaciones determinan las operaciones que será posible realizar con los datos de origen para generar las features. Puede ser suma de columnas, cálculo de medias, períodos de tiempo… Estas transformaciones podrán ser definidas desde un DSL o IU.
- **Almacenamiento**: Una vez procesadas y creadas las features, será necesario almacenarlas para tenerlas siempre disponibles.
- **Registro/Metadatos**: Un elemento clave del feature store son los metadatos, los cuales determinan cómo se estructura la información, qué agrupaciones lógicas existen, qué transformaciones se han realizado, registro histórico de features, etc. Esta información debe almacenarse junto con los datos, de tal forma que sea fácil relacionarlos. Además, sería importante incluir funcionalidades de indexador de búsqueda, siendo posible acceder a determinadas features o grupos en función de búsquedas de texto. De esta forma, sería posible obtener metadatos sin tener un conocimiento completo de las features existentes, pudiendo acceder a categorías de features. Sin esta funcionalidad, sería necesario implementar mecanismos de cacheado de la información.
- **Serving**: una vez se tienen almacenadas las features, es necesario que puedan ser accedidas y utilizadas de forma sencilla por parte del equipo de Data Science. A través del mismo SDK o IU, se debe poder realizar peticiones a las features existentes. Así mismo, sería interesante la posibilidad de pedir features on demand, es decir, crearlas sobre la marcha, sin la necesidad de que se encuentren ya en el almacenamiento. Para ello, sería interesante dividir en dos las maneras en que se accede a los datos. Por un lado, una API, en algún lenguaje de alto nivel conocido por la mayoría de los Data Science, como python. Esta API tendría la funcionalidad de realizar la petición de las features y feature groups en base a su fecha de creación. También sería un punto a través del cual definir otras features y sus metadatos. Por otro lado, existiría un a IU, con la funcionalidad de mostrar las features y sets existentes de forma sencilla e intuitiva, así como punto donde comprobar la calidad de las mismas.
- **Monitorización/Scheduler**: debe existir un elemento capaz de organizar el desencadenamiento de los procesos de transformación y calidad en base a las reglas y periodicidades definidas en los metadatos de las features.
- **Orquestación**: para que todos los procesos mencionados anteriormente funcionen de forma lógica y ordenada debe existir un orquestador, un elemento que indique

qué procesos se ejecutan y en qué orden. Este elemento puede ir estrechamente relacionado con la monitorización y organización/scheduler.

Una vez definidos los elementos principales, se procede a realizar una comparativa de las alternativas que existen en cada categoría.

3.4.1. Batch Source.

El almacenamiento de origen donde se encuentren los datos en crudo puede ser muy variado, por lo que el objetivo es que el resto de las piezas sean lo suficientemente adaptables como para que se puedan conectar a distintas fuentes de datos. Este origen pueden ser datos almacenados en distintos proveedores de cloud, ficheros de datos como CSV o JSON, almacenamientos onpremise, como Oracle, MySQL, PostgresSQL, etc.

A continuación, se muestra en formato tabular algunas de las alternativas y si cumplen o no una serie de características:

	SnowFlake	Azure Data Lake Store / S3	Oracle	Azure Synapse Analytics	ADLS + Delta Lake	ADLS + Apache Iceberg
Procesado	Sí	No	Sí	Sí	No	No
Estructura de datos	Tabular	Ficheros	Tabular	Tabular y ficheros	Ficheros	Ficheros
Versionado de datos	Sí	No	No	No	Sí	Sí
Escalado horizontal	Sí	Sí	No	Sí	Sí	Sí
Índices	Sí	No	Sí	Sí	No	No
Desventajas	Únicamente en nube	Únicamente en nube	No tiene escalado horizontal	Únicamente en nube	Únicamente en nube	Únicamente en nube

Tabla 10: Comparativa batch source

3.4.1. Procesado Batch.

Para realizar el procesado periódico de datos se plantean las siguientes alternativas:

- **Framework de procesamiento de datos**: se trata de soluciones específicas pensadas para el procesamiento distribuido de datos. Estas tecnologías permiten reducir notablemente el tiempo que se tarda en procesar un conjunto muy grande de datos mediante la paralelización de los procesos.

- **Procesamiento interno del almacenamiento**: en muchas ocasiones, el procesamiento que es capaz de llevar a cabo la propia base de datos donde se encuentren almacenados los datos es más que suficiente. El simple hecho de realizar una consulta que transforma los datos y genera unos nuevos calculados es suficiente para conseguir las features. Por ello, se plantea como una alternativa interesante, principalmente porque se evita el movimiento de datos entre plataformas, lo cual es muy poco eficiente. Sin embargo, no vale cualquier almacenamiento, pues idealmente debería contar con procesado en paralelo que permita agilizar el tiempo empleado.

	Frameworks de procesamiento	Procesamiento interno del almacenamiento
Procesado en paralelo	Sí	Algunos
Tipo de procesado	Batch/Streaming	Batch
Plataforma	Onpremise/Cloud	Onpremise/Cloud
Capacidades nativas de ML	Muchas alternativas	Pocas alternativas
Ventajas	Herramientas más completas, con mayores capacidades y específicas para el procesamiento.	Almacenamiento y procesamiento juntos, lo que implica menor latencia y menor complejidad.
Desventajas	No cuentan con almacenamiento propio, mayor latencia y mayor complejidad.	No siempre cuenta con procesado paralelo, menores capacidades y procesado sólo en batch.

Tabla 11: Comparativa procesado batch

3.4.1.1. Alternativas framework procesamiento de datos.

	Spark	Flink	Kafka Streams
Procesado en paralelo	Sí	Sí	Sí
Procesado	Batch/Streaming	Batch/Streaming	Streaming
Lenguaje	SparkSQL/PySpark	Java, Scala, Python, SQL	Java, Scala
Plataforma	Onpremise/Cloud	Cloud	Onpremise
Capacidades nativas de ML	Librerías de ML	No	No

Ventajas	Mayor control sobre las transformaciones.	Integración con múltiples conectores a distintos sources y sinks de datos.	Permite el procesado de datos tanto en streaming como en batch.
Desventajas	Lenguaje complicado (necesidad de un DSL), almacenamiento aparte.	No cuenta con sistema de almacenamiento propio.	Permitiría procesado batch pero no es eficiente.

Tabla 12: Comparativa frameworks procesamiento de datos

3.4.1.2. Alternativas procesamiento interno almacenamiento.

	Snowflake, AWS, Azure, Google Cloud
Procesado en paralelo	Sí
Procesado	Batch
Lenguaje	SQL, Java, Python, PowerShell
Plataforma	Cloud
Capacidades nativas de ML	Librerías de ML limitadas.
Ventajas	Lenguaje sencillo, sistema de almacenamiento junto a procesado y time-travel dentro de la plataforma.
Desventajas	Necesidad de tener los datos en crudo en esta plataforma. Casi no cuenta con capacidades de ML, haciendo difícil generar features en base a las salidas de modelos.

Tabla 13: Comparativa procesamiento interno almacenamiento

3.4.2. Almacenamiento.

Es la pieza principal del sistema, donde se encontrarán los valores y metadatos de todas las features, con su histórico y con las definiciones de los procesos mediante los que se calculan. Es importante que sea un almacenamiento escalable fácilmente, de rápido acceso y que cuente con la seguridad necesaria para que no sea posible su ataque desde el exterior.

Las principales alternativas disponibles son las distintas soluciones cloud del mercado, como Amazon Web Service, Google Cloud, Microsoft Azure, Snowflake, etc. Todas ellas

son almacenamientos escalables y que cuentan con procesamiento paralelo de datos muy eficiente, por lo que la comparativa se realizaría a nivel económico más que de capacidad.

3.4.1. Resgistro/Metadatos.

Se trata del almacenamiento de los metadatos de las features, por lo que la cantidad de datos no será tan elevada como en el almacenamiento de las features en sí. Por ello, es posible utilizar una base de datos local, desplegada como servicio, sin incurrir en gastos económicos como en la nube. Cualquier alternativa es válida para este componente, con alternativas como MySQL, PostgreSQL, MariaDB, etc. La elección de una u otra es mera preferencia o conocimiento previo, aunque son muy similares entre sí.

3.4.1. Serving.

Para la parte de descubrimiento de features y de interacción con el sistema se plantean dos posibilidades.

En primer lugar, un SDK o API que permita realizar llamadas a las distintas funciones del sistema e interactuar con él. Son soluciones más tecnológicas, donde se necesita cierto conocimiento informático, pero que permiten devolver los resultados de múltiples formas. Esto permitiría, por ejemplo, obtener los valores de una serie de features directamente como un dataset, que podría ser utilizado directamente para entrenar un modelo.

Por otro lado, una interfaz de usuario es más amigable para usuarios poco expertos en tecnologías o lenguajes de programación, siendo mucho más visuales e interactivas. Sin embargo, cuentan con dos desventajas principales. La primera, que supone un desarrollo extra muy costoso, que alargaría el proyecto y restaría tiempo de piezas más importantes. Por otro lado, es más complicado utilizar los valores o datos obtenidos directamente con el entrenamiento de un modelo.

3.4.1. Orquestación/Monitorización/Scheduler.

Para la organización de las diferentes piezas del feature store se plantean las siguientes posibilidades:

- **Airflow**: esta plataforma open source permite la gestión de flujo de trabajo a través del lenguaje de programación Python. Mediante gráficos acíclicos dirigidos (DAG), se gestiona la orquestación de las diferentes tareas, definidas en scripts de Python. El propio Airflow es el encargado de gestionar la programación y la ejecución de las tareas. Además, se puede definir la ejecución de los DAG en función de períodos de tiempo o de eventos externos, lo que es de gran ayuda para el desencadenado de tareas que se necesitan en la monitorización y el scheduler.
- **Apache NiFi**: esta herramienta open source permite la gestión del flujo de trabajo desde una interfaz gráfica sencilla también a través de DAGs. Es altamente configurable, permite realizar un seguimiento del flujo de datos de principio a fin y es seguro en cuanto a tolerancia a pérdidas y protocolos de comunicación segura (SSH, HTTP, etc.). Como desventaja encontramos que no es demasiado

configurable, por lo que si las herramientas de flujo que contiene no son suficientes para el desarrollo del feature store supondría una limitación.

- **Apache Oozie**: se trata de otra herramienta basada en DAGs para la gestión del flujo de trabajo. Está altamente relacionado con Hadoop, por lo que funciona de forma óptima con los elementos que componen los clústeres de esta plataforma. El lanzamiento de los procesos también se puede programar en función de períodos de tiempo o de acciones concretas externas.

	Airflow	NiFi	Oozie
DAGs	Sí	Sí	Sí
Lenguaje	Python	Java	Java/JS
Interfaz	Sí	Sí	Sí
Periodicidad	Sí	Sí	Sí
Eventos externos	Sí	En pruebas	Sí

Tabla 14: Comparativa Orquestación/Monitorición/Scheduler

3.5. Valoración y selección de las alternativas de diseño.

Una vez analizadas las diferentes alternativas de diseño se deben valorar todas las opciones y elegir las óptimas.

Con respecto al almacenamiento, teniendo en cuenta las preferencias de la empresa y siendo una plataforma en auge, se ha decidido utilizar Snowflake. Este almacenamiento no sólo permitirá guardar los datos en crudo, las features versionadas y crecer de forma escalable, sino que además cuenta con procesado interno en paralelo (MPP), por lo que serviría también como sistema de procesado de datos.

Con respecto a los metadatos, evitando el uso de la nube y de gastos innecesarios, se ha decidido realizar un despliegue local de PostgreSQL, siendo una base de datos conocida por el alumno y con mucha documentación disponible.

Con respecto al orquestador y scheduler, se ha tomado la decisión de utilizar Airflow, herramienta muy completa que permite crear DAGs para ejecutar todo tipo de procesos y ejecutarlos tanto de forma periódica como bajo demanda. Además, a través de la interfaz de usuario es posible monitorizar de forma sencilla si las operaciones están funcionando correctamente.

Finalmente, con respecto a la parte de serving, se ha decidido desarrollar una API, la llamada a una serie de funciones con las que será posible interactuar con todo el sistema. El desarrollo de un SDK es algo más complejo, así como la interfaz de usuario, por lo que se han dejado como posibles mejorar del sistema.

4. ANÁLISIS DEL PROBLEMA

4.1. Especificaciones del proyecto.

Las restricciones que se definen en el desarrollo del sistema son las siguientes:

- El idioma en que se desarrollará el sistema será el castellano.
- El proyecto se desarrollará con el distribuidor de Python Anaconda.
- Se utilizará Airflow como orquestador y monitorizado de los procesos.
- Los lenguajes de programación utilizados serán Python y SQL.
- Se utilizará Snowflake como almacenamiento masivo de features.
- El sistema utilizará PostgreSQL como almacenamiento de metadatos.
- La herramienta necesita de conexión a internet para funcionar correctamente.

4.2. Especificación de casos de uso.

Se definen los flujos de acciones que el usuario puede realizar mientras interactúa con el sistema.

Existen dos escenarios distintos en función del tipo de acción que lleve a cabo el usuario:

- **Escenario 1**: la feature ya se encuentra disponible en el sistema.
- **Escenario 2**: la feature no se encuentra en el sistema.

4.2.1. Definición de casos de uso.

La definición de los casos de uso se realiza en formato tabular con la siguiente forma:

Identificador	CU-XX
Nombre	
Descripción	
Precondiciones	
Pasos	
Resultado	

Tabla 15: Definición de casos de uso

Donde los apartados de cada tabla definen:

- **Identificador**: Se trata de un valor alfanumérico que se compone por las letras mayúsculas CU, que hacen referencia a “Caso de Uso”, y un número de compuesto por dos cifras que permiten identificar de forma única cada caso de uso.
- **Nombre**: nombre descriptivo.

- **Objetivo**: describe el objetivo que se quiere conseguir.
- **Precondiciones**: lista de condiciones previas al caso de uso que se deben cumplir para su correcto funcionamiento.
- **Pasos**: describe los pasos que hay que realizar para llevar a cabo la acción.
- **Resultado**: lista de condiciones posteriores al caso de uso que se deben cumplir.

4.2.1.1. Casos de uso de features ya existentes.

Identificador	CU-00
Nombre	Listar las features
Objetivo	Listar todas las features disponibles en el sistema
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema
Pasos	• Se realiza la llamada a la función que devuelve el listado
Resultado	• El usuario tendrá acceso a lista de features disponibles en el sistema.

Tabla 16: Listar las features CU-00

Identificador	CU-01
Nombre	Listar grupo de features
Objetivo	Listar todas las features disponibles en una agrupación lógica.
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema • Haber asociado dichas features a un grupo
Pasos	• Se determina el nombre del grupo que se quiere obtener. • Se realiza la llamada a la función que devuelve el listado de features del grupo
Resultado	• El usuario recibirá un listado con las features pertenecientes al grupo indicado.

Tabla 17: Listar grupo de features CU-01

Identificador	CU-02
Nombre	Deshabilitar una feature
Objetivo	Hacer que una feature deje de ser funcional y no se calcule de forma periódica.
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema • Que la feature se encuentre habilitada
Pasos	• Se determina el nombre de la feature que se quiere deshabilitar • Se realiza la llamada a la función que deshabilita la feature indicada.
Resultado	• Los metadatos mostrarán la feature como deshabilitada y el dag de Airflow correspondiente estará pausado.

Tabla 18: Deshabilitar una feature CU-02

Identificador	CU-03
Nombre	Actualizar una feature
Objetivo	Actualizar los metadatos de una feature
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema
Pasos	• Se determina el nombre de la feature que se quiere actualizar. • Se determina el metadato o metadatos que se quieren actualizar. • Se realiza la llamada a la función que actualiza los metadatos indicados
Resultado	• Los metadatos de la feature se habrán actualizado y se habrá generado un nuevo dag de Airflow que refleje los cambios.

Tabla 19 Actualizar una feature CU-03

Identificador	CU-04
Nombre	Lanzar la ejecución de una feature
Objetivo	Ejecutar una feature bajo demanda.

Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema
Pasos	• Se determina el nombre de la feature que se quiere ejecutar bajo demanda. • Se realiza la llamada a la función que ejecuta el dag de Airflow de la feature seleccionada.
Resultado	• El dag de la feature se ejecutará en el momento e insertará los valores calculados en el sistema.

Tabla 20: Lanzar la ejecución de una feature CU-04

Identificador	CU-05
Nombre	Obtener últimos datos
Objetivo	Obtener un listado con los últimos valores calculados para una feature.
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema • Haber ejecutado una feature
Pasos	• Se realiza la llamada a la función que obtiene los datos de la feature seleccionada.
Resultado	• El usuario obtendrá a los valores actualizados de la feature.

Tabla 21: Obtener últimos datos CU-05

Identificador	CU-06
Nombre	Obtener datos en fecha
Objetivo	Obtener un listado con los valores calculados para una feature en una fecha concreta.
Precondiciones	• Tener la herramienta desplegada • Haber introducido features en el sistema • Que existan datos para la fecha seleccionada
Pasos	• Se realiza la llamada a la función que obtiene los datos de la feature seleccionada en la fecha indicada.
Resultado	• El usuario tendrá acceso a los valores de la feature en la fecha indicada en la llamada.

Tabla 22: Obtener datos en fecha CU-06

4.2.1.2.Casos de uso de uso de features no existentes.

Identificador	CU-07
Nombre	Añadir una nueva feature calculable.
Objetivo	Agregar una nueva feature al sistema con los datos existentes en el almacenamiento en crudo.
Precondiciones	• Tener la herramienta desplegada • Que no exista la feature que se quiere agregar.
Pasos	• Determinar el nombre de la feature. • Determinar la descripción. • Determinar la periodicidad con la que se va a calcular. • Determinar de qué tipo es el valor del cálculo. • Determinar quién es el encargado o dueño. • Determinar de qué esquema proceden los datos. • Determinar de qué tablas vienen los datos. • Determinar de qué campos proceden los datos. • Determinar qué operaciones se van a realizar entre los campos. • Determinar qué condiciones deben cumplir los datos. • Determinar qué uniones entre tablas se deben realizar. • Realizar la llamada la función que inserta una nueva feature en el sistema en base a la definición y metadatos establecidos.
Resultado	• Se creará un dag en Airflow con las características indicadas en la definición. • Se ejecutará el dag de forma periódica agregando valores nuevos a esa feature. • Se agregarán los demás metadatos de la feature.

Tabla 23: Añadir una nueva feature calculable CU-07

Identificador	CU-08
Nombre	Añadir una nueva feature no calculable.
Objetivo	Agregar una nueva feature al sistema con datos que no existen en el almacenamiento en crudo.
Precondiciones	• Que no exista la feature que se quiere agregar. • Que no existan los datos de los que se calcula la feature.
Pasos	• Determinar qué datos van a ser necesarios. • Determinar dónde se encuentran almacenados esos datos. • Determinar cómo se van a llevar esos datos al almacenamiento del Feature Store. • Implementar las funcionalidades necesarias para disponibilizarlos. • Determinar el nombre de la feature. • Determinar la descripción. • Determinar la periodicidad con la que se va a calcular. • Determinar de qué tipo es el valor del cálculo. • Determinar quién es el encargado o dueño. • Determinar de qué esquema proceden los datos. • Determinar de qué tablas vienen los datos. • Determinar de qué campos proceden los datos. • Determinar qué operaciones se van a realizar entre los campos. • Determinar qué condiciones deben cumplir los datos. • Determinar qué uniones entre tablas se deben realizar. • Realizar la llamada la función que inserta una nueva feature en el sistema en base a la definición y metadatos establecidos.
Resultado	• Se crearán nuevas tablas y campos en el almacenamiento en crudo para disponibilizar los datos en el sistema. • Se creará un dag en Airflow con las características indicadas en la definición. • Se ejecutará el dag de forma periódica agregando valores nuevos a esa feature. • Se agregarán los demás metadatos de la feature.

Tabla 24: Añadir una nueva feature no calculable CU-08

Identificador	CU-09
Nombre	Añadir nuevas operaciones.
Objetivo	Agregar nuevas operaciones para el cálculo de features que no se encontraban disponibles en el sistema.
Precondiciones	• Que no existan las operaciones que se quieren agregar.
Pasos	• Determinar qué operaciones se quieren disponibilizar. • Modificar el código para que se puedan realizar dichas operaciones
Resultado	• El sistema tendrá disponibles nuevas operaciones para realizar el cálculo de las features.

Tabla 25: Añadir nuevas operaciones CU-09

4.3. Especificación de requisitos software.

En este apartado se describen los requisitos que se deben cumplir. Estos requisitos se han establecido a lo largo de las iteraciones realizadas y de los distintos sprints. Para la definición de los mismos, se parte de los escenarios que han sido descritos anteriormente. Se pueden identificar dos tipos de requisitos software para el proyecto:

- **Requisitos funcionales**: describen las funciones del sistema software o sus componentes.
- **Requisitos no funcionales**: describen restricciones y obligaciones del desarrollo software.

En la tabla a continuación se describe el formato que tendrán cada uno de ellos:

Identificador	YYY-XX
Nombre	
Descripción	
Necesidad	
Dependencias	

Tabla 26: Definición de requisitos

Donde los campos de cada tabla definen:

- **ID**: Valor alfanumérico compuesto por dos elementos:
  - YYY: siglas que identifican los dos tipos de requisitos. Estas siglas pueden ser:
    - RF: requisito funcional.
    - RNF: requisito no funcional.

- **Nombre**: nombre descriptivo.
- **Descripción**: describe con más detalle el objetivo del requisito.
- **Necesidad**: define la necesidad que tiene el requisito en el sistema. Hay tres tipos:
    - Esencial: es muy necesario para el sistema.
    - Deseable: tiene importancia moderada para el sistema.
    - Opcional: es poco importante o prescindible para el sistema.
- **Dependencias**: listado de requisitos de los cuales depende el requisito descrito en la tabla. Ayuda a mejorar mejorar la trazabilidad y conocer el orden de las implementaciones. En caso de no existir el recuadro se mantendrá vacío.

4.3.1. Requisitos funcionales.

Identificador	RF-00
Nombre	Almacenamiento en crudo
Descripción	El sistema contará con un almacenamiento donde se guardarán los datos en crudo en base a los cuales se calcularán las features.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	

Tabla 27: Almacenamiento en crudo RF-00

Identificador	RF-01
Nombre	Almacenamiento de metadatos.
Descripción	El sistema contará con un almacenamiento de metadatos.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-04

Tabla 28: Almacenamiento de metadatos RF-01

Identificador	RF-02
Nombre	Almacenamiento de features
Descripción	El sistema contará con un almacenamiento para guardar los valores de las features calculadas.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-00, RF-01

Tabla 29: Almacenamiento de features RF-02

Identificador	RF-03
Nombre	Domain Specific Language
Descripción	El sistema contará con un DSL para definir las features
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	

Tabla 30: Domain Specific Language RF-03

Identificador	RF-04
Nombre	Software Development Kit
Descripción	El sistema contará con un SDK a través del cual interactuar con el mismo
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01, RF-04

Tabla 31: Software Development Kit RF-04

Identificador	RF-05
Nombre	Orquestador
Descripción	El sistema contará con un orquestador de procesos.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 32: Orquestador RF-05

Identificador	RF-06
Nombre	Scheduler
Descripción	El sistema contará con un scheduler que lance los procesos de forma periódica.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-06

Tabla 33: Scheduler RF-06

Identificador	RF-07
Nombre	Monitorización
Descripción	El sistema contará con una herramienta de monitorización de procesos.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-06

Tabla 34: Monitorización RF-07

Identificador	RF-08
Nombre	Listar features
Descripción	El sistema será capaz de listar las features creadas en el mismo.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01, RF-06

Tabla 35: Listar features RF-08

Identificador	RF-09
Nombre	API funciones
Descripción	La interacción con las funciones del sistema se realizará vía API.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 36: API funciones RF-09

Identificador	RF-10
Nombre	Listar grupos
Descripción	El sistema será capaz de listar las features pertenecientes a un grupo
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01, RF-06

Tabla 37: Listar grupos RF-10

Identificador	RF-11
Nombre	Crear grupos
Descripción	El sistema será capaz de crear nuevas agrupaciones lógicas de features
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 38: Crear grupos RF-11

Identificador	RF-12
Nombre	Crear features
Descripción	El sistema será capaz de crear nuevas features en base al DSL implementado.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01, RF-06

Tabla 39: Crear features RF-12

Identificador	RF-13
Nombre	Ejecutar feature
Descripción	El sistema será capaz de ejecutar una feature bajo demanda.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-06

Tabla 40: Ejecutar feature RF-13

Identificador	RF-14
Nombre	Deshabilitar feature
Descripción	El sistema será capaz de deshabilitar una feature.
Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 41 Deshabilitar feature RF-14

Identificador	RF-15
Nombre	Actualizar descripción
Descripción	El sistema será capaz de actualizar la descripción de una feature
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 42: Actualizar descripción RF-15

Identificador	RF-16
Nombre	Actualizar periodicidad
Descripción	El sistema será capaz de actualizar la periodicidad de una feature.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 43: Actualizar periodicidad RF-16

Identificador	RF-17
Nombre	Actualizar dueño
Descripción	El sistema será capaz de actualizar el dueño de una feature.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 44: Actualizar dueño RF-17

Identificador	RF-18
Nombre	Actualizar condición.
Descripción	El sistema será capaz de actualizar la condición de la query de la transformación.
Necesidad	☐ Esencial ☑ Deseable ☐ Opcional
Dependencias	RF-01

Tabla 45: Actualizar condición RF-18

4.3.2. Requisitos no funcionales.

Identificador	RNF-00		
Nombre	Despliegue Docker		
Descripción	El sistema debe ser capaz de desplegar mediante contenedores Docker.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Dependencias			

Tabla 46: Despliegue Docker RNF-00

Identificador	RNF-01		
Nombre	Contenedor PostgreSQL		
Descripción	En el sistema se desplegará un contenedor de PostgreSQL.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Dependencias	RNF-00		

Tabla 47: Contenedor PostgreSQL RNF-01

Identificador	RNF-02		
Nombre	Contenedor Airflow		
Descripción	En el sistema se desplegará un contenedor de Airflow.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Dependencias	RNF-00		

Tabla 48: Contenedor Airflow RNF-02

Identificador	RNF-03		
Nombre	Contenedor Feature Store		
Descripción	En el sistema se desplegará un contenedor con la funcionalidad del Feature Store.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Dependencias	RNF-00		

Tabla 49: Contenedor Feature Store RNF-03

Identificador	RNF-04
Nombre	Función PostgreSQL
Descripción	El contenedor PostgreSQL se utilizará para almacenar los metadatos
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RNF-01

Tabla 50: Función PostgreSQL RNF-04

Identificador	RNF-05
Nombre	Función Airflow orquestador
Descripción	Se utilizará Airflow como orquestador de procesos.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RNF-02

Tabla 51: Función Airflow orquestador RNF-05

Identificador	RNF-06
Nombre	Función Airflow monitorización
Descripción	Se utilizará Airflow para monitorizar los procesos.
Necesidad	☐ Esencial ☐ Deseable ☑ Opcional
Dependencias	RNF-02

Tabla 52: Función Airflow monitorización RNF-06

Identificador	RNF-07
Nombre	Función Airflow scheduler
Descripción	Se utilizará Airflow como scheduler de procesos.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	RNF-02

Tabla 53: Función Airflow scheduler RNF-07

Identificador	RNF-08
Nombre	Lenguaje Python
Descripción	Se utilizará Python como lenguaje de programación.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	

Tabla 54: Lenguaje Python RNF-08

Identificador	RNF-09
Nombre	Almacenamiento features
Descripción	Se utilizará Snowflake como almacenamiento de los valores de las features
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	

Tabla 55: Almacenamiento features RNF-09

Identificador	RNF-10
Nombre	Almacenamiento datos en crudo
Descripción	Se utilizará Snowflake como almacenamiento de los datos en crudo mediante los que se calculan las features.
Necesidad	☑ Esencial ☐ Deseable ☐ Opcional
Dependencias	

Tabla 56: Almacenamiento datos en crudo RNF-10

Identificador	RNF-11
Nombre	API Postman
Descripción	Se utilizará Postman como herramienta para realizar las llamadas a la API.
Necesidad	☐Esencial ☐ Deseable ☑ Opcional
Dependencias	

Tabla 57: API Postman RNF-11

4.4. Elaboración del modelo de datos.

Los datos que se van a alamcenar en el sistema se pueden dividir en tres grandes grupos: metadatos, datos en crudo y features versionadas. Los metadatos hacen referencia a toda la información que acompaña a una feature, como es su nombre, descripción, periodicidad con la que se calcula, etc. Los datos en crudo son los datos en base a los que se calcularán las features o características. Por último, las features versionadas son el conjunto de valores, con fecha y hora, que toma cada una de ellas, y que serán los datos con los que se entrenen los modelos.

Encontramos una diferenciación entre los tres grupos y por tanto necesidades de almacenamiento distintos.

Los metadatos, aunque pueden ser elevados, no alcanzarán cantidades muy altas, por lo que la base de datos donde se almacenen no tendrá necesidad de ser escalable. Es por ello que se ha decidido almacenarlos en una base de datos PostgreSQL. Este sistema, de tipo relacional, es una muy utilizada, por lo que será sencillo encontrar información y documentación de cómo utilizarla.

Por otro lado, los datos en crudo pueden suponer un almacenamiento mucho más elevado. A mayor cantidad de orígenes de información (otras bases de datos, CSV, JSON, etc.), mayor será la cantidad de datos que habrá que almacenar para calcular posteriormente con ellos las features. Es por ello que deben contar con un almacenamiento escalable, donde sea sencillo añadir miles o millones de registros nuevos sin tener problemas de espacio. Por lo tanto, la mejor alternativa se encuentra en la nube, y tal y como se indicó en el estudio de alternativas, se ha decidido utilizar Snowflake.

Finalmente, con respecto a los valores de las features calculadas y versionadas, estas pueden ser cantidades muy elevadas, en función de los datos de origen y de la periodicidad con la que se calculen. Por lo tanto, también se almacenarán en Snowflake, para asegurar así un escalado sencillo de la información.

5. DISEÑO DEL SISTEMA

5.1. Objetivo

En este capítulo se resuelve la cuestión planteada en el apartado anterior. Para ello, se define tanto la arquitectura que deberá tener el sistema que se ha planteado como su funcionamiento, con la finalidad de comprender cómo se ha implementado la herramienta desarrollada.

5.2. Arquitectura del sistema

Tal y como se indicó en el Estado del Arte, los elementos básicos para construir un feature store son:

- El origen de datos, tanto batch como en tiempo real, la transformación de los datos
- El almacenamiento y registro histórico de features
- Una capa de serving para interactuar con el sistema
- Todo ello orquestado y monitorizado.

En base a todos estos componentes, y siguiendo las recomendaciones y necesidades de la empresa, se ha alcanzado la siguiente arquitectura:

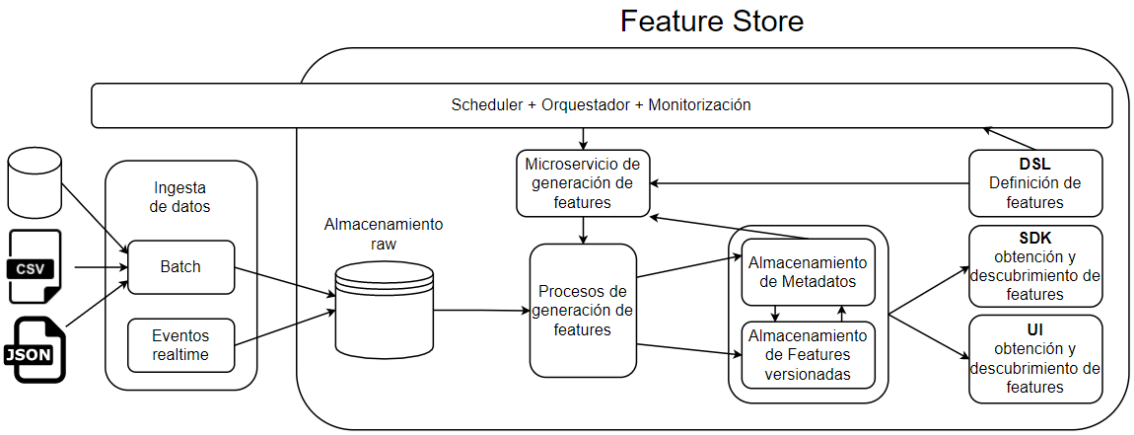

Ilustración 4: Arquitectura del sistema

Como se puede observar, existen muchos orígenes de datos posibles, como CSV, JSON, otras bases de datos, etc. Esos datos pueden ser extraídos de dichos orígenes mediante procesos batch y volcados a un almacenamiento raw o de datos en crudo. Lo mismo ocurre con los eventos real-time. Una vez estos datos están disponibles en el almacenamiento en crudo, es posible calcular features con ellos. Mediante procesos de generación de features, los datos son transformados según se encuentren definidos y se almacenarán como features versionadas (histórico). Para conectar a los diferentes orígenes y lanzar dichos procesos se utilizará un microservicio de generación de features. Además de los valores calculados, existe un almacenamiento de metadatos, donde se encontrará toda la información adicional, incluida la definición de la transformación.

Como se puede observar, se ha optado por un modelo ELT (Extract, Load y Transform). Esto se debe a que la complejidad asociada a realizar el procesado de los datos al vuelo según se obtienen de los orígenes es muy elevada, ya que supondría un nivel de generalización y adaptación a las diferentes fuentes fuera del alcance de un proyecto como este. Por ello, se ha optado por tener un almacenamiento en crudo, que permita almacenar todos los datos utilizados para el cálculo de las features con un formato tabular similar, de tal forma que sea accesible con simples consultas en el lenguaje SQL.

Como capa de serving encontramos tres elementos principales. En primer lugar, un DSL, con el que poder definir las features de forma sencilla. Este lenguaje será muy similar a SQL, siendo muy conocido por todo el mundo y sencillo de utilizar. Además, permite entender fácilmente como ocurrirá la transformación de los datos en crudo en features.

Para interactuar con el sistema encontramos dos alternativas. Por un lado, un SDK, una librería con todas las funciones disponibles, como listar las features, obtener valores, insertar nuevas features en conjunto con el DSL, etc. Este SDK permite obtener los valores y almacenarlos como datasets, siendo muy fácil de combinar con otras librerías para construir directamente los modelos. Por otro lado, podría existir una interfaz de usuario, mucho más amigable pero más limitada en cuanto a funcionalidades.

Finalmente, existirá una capa común a todo el sistema que permita orquestar los procesos, desde la extracción de los datos de los orígenes, su volcado en el almacenamiento en crudo, su transformación en formato de features y su serving a los usuarios del sistema. Esta capa será además la encargada de realizar estos procesos de forma periódica y de proporcionar una monitorización de los mismos para comprobar que todo funcione correctamente.

En esta arquitectura se han utilizado las tecnologías elegidas en el análisis del sistema, pero se han ido modificando en varias fases con el objetivo de probar el sistema en una fase más sencilla para ir aumentando la dificultad del mismo poco a poco. A continuación, se describen las diferentes fases:

5.2.1. Fase uno

En esta primera fase se ha utilizado la arquitectura que se muestra a continuación:

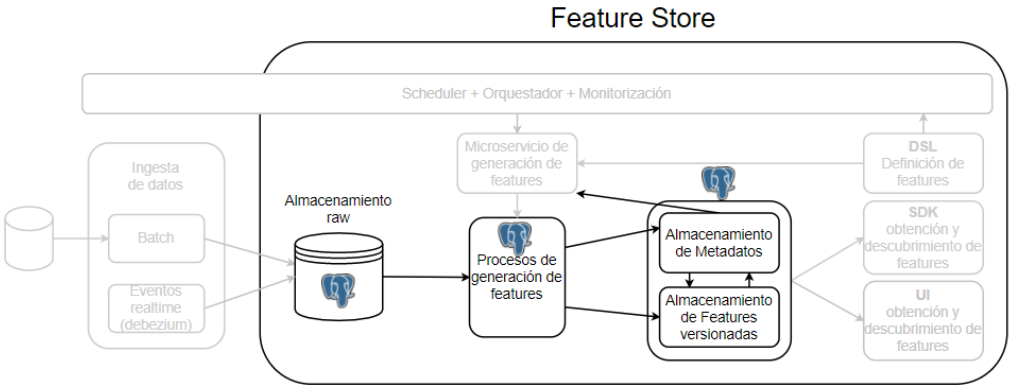

Ilustración 5: Arquitectura fase uno

Tal y como se puede observar en la imagen, en primer lugar, se ha decidido desarrollar la mayor parte del sistema en torno a la base de datos PostreSQL. En esta se han insertado una serie de datos de forma manual en la base de datos y mediante las propias capacidades de esta se han transformado en features ya calculadas. De la misma forma, se introducen los metadatos en base a los cuales calcular las features de forma manual, sin utilizar un DSL ni un SDK para interactuar con el sistema. Tampoco se orquestan los procesos, por lo que las transformaciones se lanzan a mano, realizando una serie de tests unitarios para comprobar que todo funcione correctamente. Esta primera fase también permite definir el modelo de datos, probando las diferentes necesidades y asegurando que se cubren todos los aspectos necesarios para que funciones el sistema.

5.2.2. Fase dos

En esta segunda fase se han añadido capacidades relacionadas con la interacción del sistema:

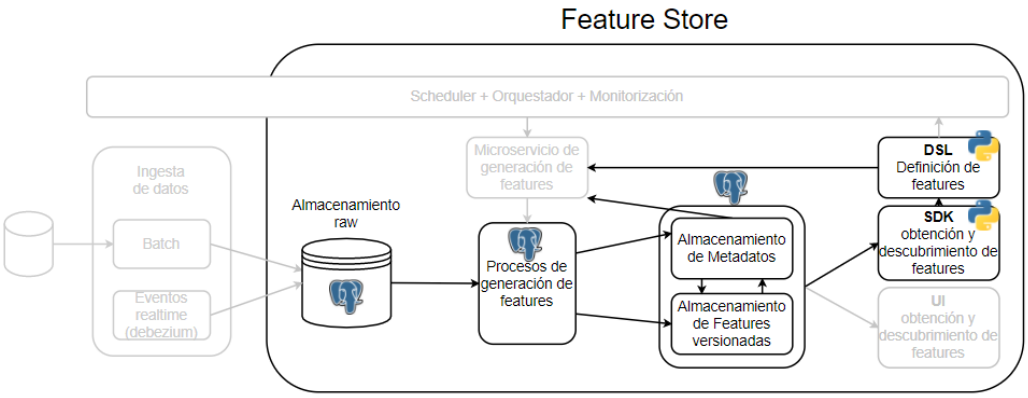

Ilustración 6: Arquitectura fase 2

Como se puede observar, se han añadido los componentes del SDK y el DSL para poder definir las features e interactuar con el sistema de forma externa. En esta fase se realizan adaptaciones en el modelo de datos y se realizan tests unitarios para comprobar que todo funciona correctamente.

5.2.3. Fase tres

En esta última fase se añaden las capacidades de orquestación y se modifica la base de datos a Snowflake:

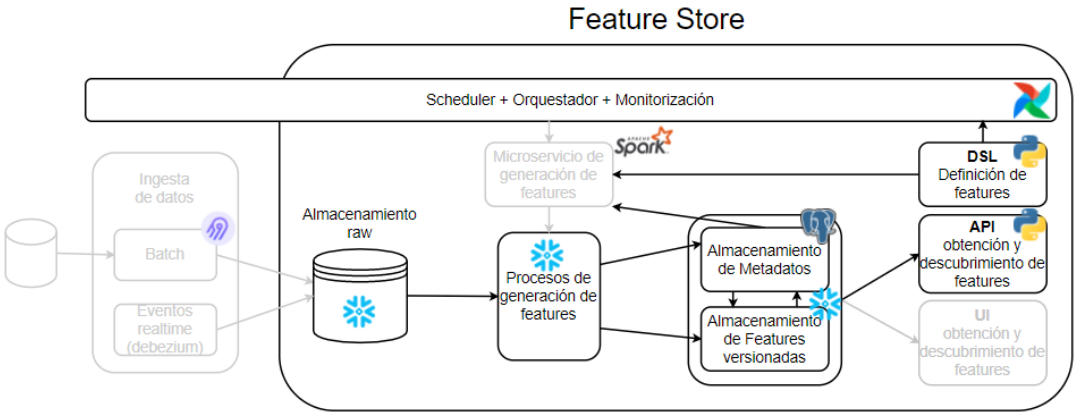

Ilustración 7: Arquitectura fase 3

Tal y como se comentaba en la elección de tecnologías, el almacenamiento en crudo y de features versionadas pasa a ser Snowflake, así como el motor que procesa los datos, debido a su capacidad de cómputo en paralelo. Como orquestador, scheduler y monitorización se utiliza Airflow, que permite, a través de los dags, indicar qué acciones se deben llevar a cabo y en qué momento.

Además, cabe destacar que para interactuar con el sistema se utiliza una API en vez de un SDK, y mediante llamadas a la misma se pueden realizar las distintas funciones implementadas.

El resto de los componentes, los que quedan con un sombreado más claro, no se han implementado por falta de tiempo en el proyecto. Sin embargo, se han dejado reflejadas algunas tecnologías que podrían servir para esos propósitos, como son Airbyte para mover los datos de los diferentes orígenes al almacenamiento en crudo y Spark como microservicio de generación de features que permitiría generalizar el sistema y conectar a diferentes bases de datos y adaptarse mejor a las necesidades de diferentes compañías.

5.3. Diseño de prototipos del sistema.

En este apartado se describen los diferentes prototipos diseñados para el sistema. Estos prototipos hacen referencia a la parte lógica de la aplicación, es decir, al código implementado.

Para la presentación de cada prototipo se empleará una tabla con este formato:

Identificador	PT-XX	
Nombre		
Librerías		
Funciones	Nombre y parámetros	Descripción

Tabla 58: Definición de prototipos

Donde los campos de cada tabla definen:

- **Identificador**: Valor alfanumérico compuesto por dos elementos:
  - PT: siglas que corresponden a la palabra “prototipo”.
  - XX: valor numérico entre 00 y 99.
- **Nombre**: nombre descriptivo del prototipo
- **Librerías:** las librerías utilizadas.
- **Funciones:** métodos que tenga el código. Para cada una de ellas se indicará el nombre y una descripción de su principal objetivo.

5.3.1. Docker-Compose.

Este módulo es el encargado de inicializar todo el sistema. En él se definen todos los contenedores Docker que se van a desplegar, junto con los puertos donde serán visibles y otros parámetros.

Los contenedores que se despliegan son los siguientes:

- Airflow: permitirá realizar la orquestación, scheduler y monitorización de los procesos de generación de features.
- PostgreSQL: permitirá desplegar una base de datos donde almacenar los metadatos asociados a cada feature.
- Feature Store: permitirá desplegar la pieza principal del sistema, donde se encontrarán las funciones necesarias para interactuar y utilizar el sistema.

En este módulo no hay nada destacable en cuanto a la implementación, ya que se trata de un fichero Docker-compose.yml estándar del que se puede obtener mucha información en la documentación. Sin embargo, se puede destacar la necesidad de crear carpetas compartidas entre los distintos contenedores en la máquina donde se despliegue, haciendo posible la comunicación de metadatos y dags de Airflow entre ellos.

5.3.2. Feature Store.

En este módulo se encuentran todas las funciones que permiten el funcionamiento del sistema. Estas funciones permiten realizar los requisitos establecidos en el Análisis del Sistema, como listado de features, inserción de nuevas, modificación de metadatos, etc.

Para realizar estas operaciones, se cuenta con las siguientes librerías y métodos:

Identificador	PT-01	
Nombre	Funciones del sistema	
Librerías	• os • shutil • datetime • pyscopg2 • pyscopg2.extensions: ISOLATION_LEVEL_AUTOCOMMIT • sqlalchemy • requests	
Funciones	init	Inicializa el esquema de metadatos y Snowflake
	init_schema(credentials)	Realiza la conexión a PostgreSQL y crea el esquema y estructura de tablas de los metadatos.

	init_snowflake(credentials)	Realiza la conexión a Snowflake y crea la tabla necesaria.
	create_set(feature_set_name, description)	Crea un set o agrupación lógica de features.
	insert_feature(definition)	Crea una nueva feature en el sistema en base a la definición realizada con el DSL.
	get_features	Lista todas las features del sistema.
	get_feature(feature_name, date)	Devuelve los datos de la feature en la fecha indicada. Si no se establece fecha, devuelve los últimos datos.
	get_set(set_name)	Devuelve las features pertenecientes al set indicado.
	update_metadata(definition)	Actualiza los metadatos de una feature en base a una nueva definición realizada con el DSL.
	disable_feature(feature_name)	Deshabilita una feature para que ya no se calcule más.
	trigger_feature(feature_name)	Lanza la transformación de una feature bajo demanda.

Tabla 59: Funciones del sistema PT-01

A continuación, se explican con más detalle cada una de las funciones:

- **init**: llama a las funciones init_schema e init_snowflake, pasándoles las credenciales necesarias para la inicialización de las dos bases de datos.
- **init_schema**: inicializa el schema de metadatos dentro de la base de datos PostgreSQL. Para ello, se realiza la conexión al contenedor desplegado con esta base de datos con las credenciales obtenidas por parámetro. Además, se crea el esquema de metadatos y las tablas correspondientes. Para facilitar las posteriores pruebas, también se insertan una serie de metadatos de ejemplo para poder utilizar el sistema de forma inmediata.
- **init_snowflake**: se realiza la conexión a una base de datos Snowflake, que se debe haber creado con anterioridad, mediante las credenciales recibidas por parámetro. Además, se crea una tabla que almacenará todos los valores de las features que se calculen a lo largo del tiempo.
- **create_set**: esta función crea un nuevo set o grupo dentro del sistema. Para ello, se inserta en la tabla correspondiente un nuevo registro con el nombre y

descripción proporcionados en la llamada del método y se establece la fecha de creación del mismo a la fecha y hora actual del sistema.

- **insert_feature**: esta función permite crear una nueva feature en el sistema. Para ello, recibirá una definición realizada a través del DSL, en este caso, un fichero del tipo JSON con el formato de un esquema definido, en el que se especifican las características de la feature, como nombre, periodicidad con la que se calcula, grupo al que pertenece, y forma en que se calcula, es decir, la query con la que se construye. Para ello, se inserta en las diferentes tablas de metadatos la información obtenida en la definición para finalmente realizar la llamada a otro módulo llamado "create_dag". Este módulo, que se explicará más adelante, permite crear el dag de Airflow correspondiente a la feature introducida en el sistema para que se calcule de forma periódica.
- **get_features**: este método recorre las features almacenadas en el sistema y devuelve el nombre de todas ellas en formato de lista.
- **get_feature**: esta función recibe por parámetros el nombre de una fecha y tiene la posibilidad de recibir una fecha también. En base a estos parámetros se realiza una llamada a la base de datos de Snowflake donde se encuentran almacenados los valores de las features, y se devuelven aquellos que corresponden con la indicada. En caso de indicar fecha, se devolverán los datos de ese día, pero en caso de no indicarla, se devolverán los datos de la última ejecución registrada.
- **get_set**: devuelve el nombre de todas las features que pertenecen a la agrupación lógica que se epsecifica por parámetros.
- **update_metadata**: esta función recibe por parámetros la definición de una nueva feature. Primero se comprueba que dicha feature existe, para, en caso contrario, realizar la llamada a la función de inserción. Si existe, se procede a comparar los datos almacenados actualmente en el sistema para comprobar cuáles son los que se desean actualizar y cuáles permanecen igual, de tal forma que sea posible insertar nuevos registros para esos metadatos. En caso de que la actualización afecte a la forma en que se calcula la feature, se realizará también la llamada a la creación de un nuevo dag de Airflow que refleje el cambio. La finalidad de realizar inserciones de metadatos nuevos y no actualizar los ya existentes se debe a la necesidad de historificación, ya que los nuevos metadatos irán asociados a una nueva fecha de actualización.
- **disable_feature**: este método permite deshabilitar una feature del sistema, de tal forma que se deje de calcular de forma periódica. Para ello, se modifica su estado en los metadatos del sistema y se realiza una llamada a la API de Airflow que permita modificar el estado del dag correspondiente de activo a pausado.
- **trigger_feature**: esta función permite realizar una llamada al dag de Airflow que calcula la transformación de la feature bajo demanda, realizando esa transformación al momento sin tener que esperar a su planificación correspondiente. Para ello, se realiza una llamada a la API de Airflow para lanzar el dag en ese momento.

5.3.3. Dag Creator.

Las funciones definidas en este módulo permiten crear los dags de Airflow que ejecutarán de forma periódica las transformaciones necesarias para calcular las distintas featutes.

A continuación, se muestra un ejemplo de cómo sería la definición en formato JSON de una feature a través del DSL:

```
{
   "name": "prueba_demo",
   "description": "Feature de prueba para comprobar el funcionamiento del sistema",
   "periodicity": "*/10 * * * *",
   "type": "Float",
   "owner": "1",
   "set_name": "group_orders",
   "schema": "SNOWFLAKE_SAMPLE_DATA.TPCH_SF10",
   "select": [
      {"pref": "",
      "operation": "",
      "fields": [{"name": "C_CUSTKEY", "table": "CUSTOMER"}]
      },
      {"pref": "SUM",
      "operation": "",
      "fields": [{"name": "O_TOTALPRICE", "table": "ORDERS"}]
      }],
   "where": "",
   "join": "CUSTOMER.C_CUSTKEY=ORDERS.O_CUSTKEY"
}
```

Para realizar estas operaciones se cuenta con las siguientes librerías y funciones:

Identificador	PT-02	
Nombre	Funciones de creación de un dag	
Librerías	• os • sys	
Funciones	get_tables(definition)	Construye el string de texto correspondiente a las tablas de la query.
	get_fields(definition)	Construye el string de texto correspondiente a los campos de la query.
	get_where(definition)	Construye el string de texto correspondiente a las condiciones de la query.
	get_group_by(definition)	Construye el string de texto correspondiente a la sentencia de agrupación de la query.
	create_dag(definition)	Crea un nuevo dag con la información de la query que se ha definido en el DSL.

Tabla 60: Funciones de creación de un dag PT-02

A continuación, se explican con mayor detalle cada uno de los métodos:

- **get_tables( )**: esta función es la encargada de crear el string de texto correspondiente a las tablas que compondrán la query de la transformación, es decir, la sentencia "FROM". Para ello, se recorre la definición de la feature, y se insertan en una lista todas las tablas distintas que se detecte que son necesarias. Finalmente, se concatenan dichas tablas con el schema donde se encuentran y se construye la cadena de texto.
- **get_fields( )**: este método permite construir la cadena de texto de los campos que son necesarios en la query de transformación de la feature, así como las operaciones correspondientes. Para ello, se recorre la definición de la feature y se agrupan los campos cuyas operaciones coincidan. A cada campo se le concatena el nombre de la tabla del que proviene y se crea el string final con el "SELECT".
- **get_where()**: esta función permite construir la cadena de texto con las condiciones de la query que realiza la transformación. Dentro de estas condiciones encontramos dos tipos. Por un lado, las condiciones de ejecución, como puede ser indicar que un campo deba ser mayor que una constante. Por otro lado, encontramos la condición de "join" entre tablas en caso de que sea necesario, donde se indica que el campo de una tabla debe ser igual al campo de otra tabla. Tanto las condiciones como el join son opcionales, por lo que el string final podrá tener uno, otro, ambos, o ninguno de ellos. Finalmente, se devuelve el string con el "WHERE" indicado.
- **get_group_by( )**: en este método se crea la cadena de texto correspondiente a la sentencia "GROUP BY" de la query. En todos los casos el campo por el que se agrupará es el identificador de la feature, por lo que basta con saber cuál es para
- **create_dag( )**: este método es el que permite crear el dag de Airflow correspondiente a la definición realizada. Estos dags son ficheros de Python que se deben colocar en una carpeta concreta dentro del sistema de Airflow llamada "dags". Por ello, para crearlo, se crea un fichero con esta extensión que contenga toda la información necesaria y se coloca en la carpeta indicada. Todos los dags se basan en la misma plantilla, en la que se indica el nombre, que corresponde con el de la feature más un identificador, la priodicidad y algunos otros metadatos necesarios. Finalmente, el elemento que realizará la transformación será un Snowflake Operator. Esta función, que se importa desde una librería, permite realizar la llamada a la base de datos de Snowflake inicializada al levantar el sistema y ejecutar la query definida en el DSL. Este dag cuenta con un solo paso, el de ejecutar la query, pero si se implementase la extracción de datos de distintos orígenes para su posterior volcado en el almacenamiento en crudo, habría un paso previo para tomar los datos de ese origen y actualizar el almacenamiento raw para calcular la feature con los últimos datos disponibles.

5.3.4. API

Este módulo es el que contiene todas las funciones que permiten construir la API con la que interactuar con el sistema. Esta API se ha construido en sustitución del SDK, que podría haberse realizado de forma sencilla creando un paquete con las funciones definidas anteriormente. No obstante, se ha decidido utilizar esta opción como investigación por parte del alumno, que nunca había realizado nada similar y le permite adquirir este conocimiento.

Para realizar estas operaciones se cuenta con las siguientes librerías y funciones:

Identificador	PT-03	
Nombre	Funciones de llamadas a la API	
Librerías	• Json • Flask • Featurestore (se importan las funciones del Feature Store)	
Funciones	get_features	Llama a la función para obtener las features. Es de tipo GET y devuelve en formato de lista las features obtenidas.
	new_feature	Llama a la función para insertar una nueva feature. Es de tipo POST, recibe en el cuerpo de la llamada un JSON con la definición de la feature y devuelve si el resultado fue correcto o no.
	get_set	Llama a la función que devuelve las features de un grupo. Es de tipo GET y devuelve una lista con las features obtenidas.
	get_feature	Llama a la función que devuelve el valor de una feature. Es de tipo GET y recibe en el cuerpo de la llamada el nombre y la fecha de la feature que se quiere obtener. Devuelve una lista con los identificadores y valores correspondientes.

	update_feature	Llama a la función que actualiza la definición de una feature. Es de tipo POST y recibe en el cuerpo de la llamada la nueva definición que se quiere actualizar. Devuelve si el proceso ha ido bien o no.
	disable	Llama a la función que deshabilita una feature. Es de tipo POST y recibe en el cuerpo de la función la feature que se quiere deshabilitar. Devuelve si todo ha ido correctamente o no.
	trigger	Llama a la función que lanza la transformación de una feature. Es de tipo POST y devuelve si todo ha ido correctamente o no.

Tabla 61: Funciones de llamadas a la API PT-03

5.4. Interacción de prototipos y casos de uso.

Mediante la siguiente tabla se observa la trazabilidad que existe entre los escenarios planteados y los prototipos definidos:

	CU-00	CU-01	CU-02	CU-03	CU-04	CU-05	CU-06	CU-07	CU-08	CU-09
PT-00	X	X	X	X	X	X	X	X	X	X
PT-01	X	X	X	X	X	X	X	X	X	X
PT-02			X	X	X			X		
PT-03	X	X	X	X	X	X	X	X	X	X

Tabla 62: Interacción de prototipos y casos de uso

Como se puede observar, para todos los casos de uso son necesarios tres de los prototipos, ya que conciernen a el despliegue del sistema, las funciones disponibles y la interacción con el sistema. Con el tercer prototipo, el encargado de crear los dags de Airflow, sólo están relacionados cuatro de los casos de uso.

5.5. Modelo de datos.

Una definida la arquitectura y las necesidades del modelo de datos se puede elaborar de forma completa atendiendo a todas las necesidades identificadas en el problema:

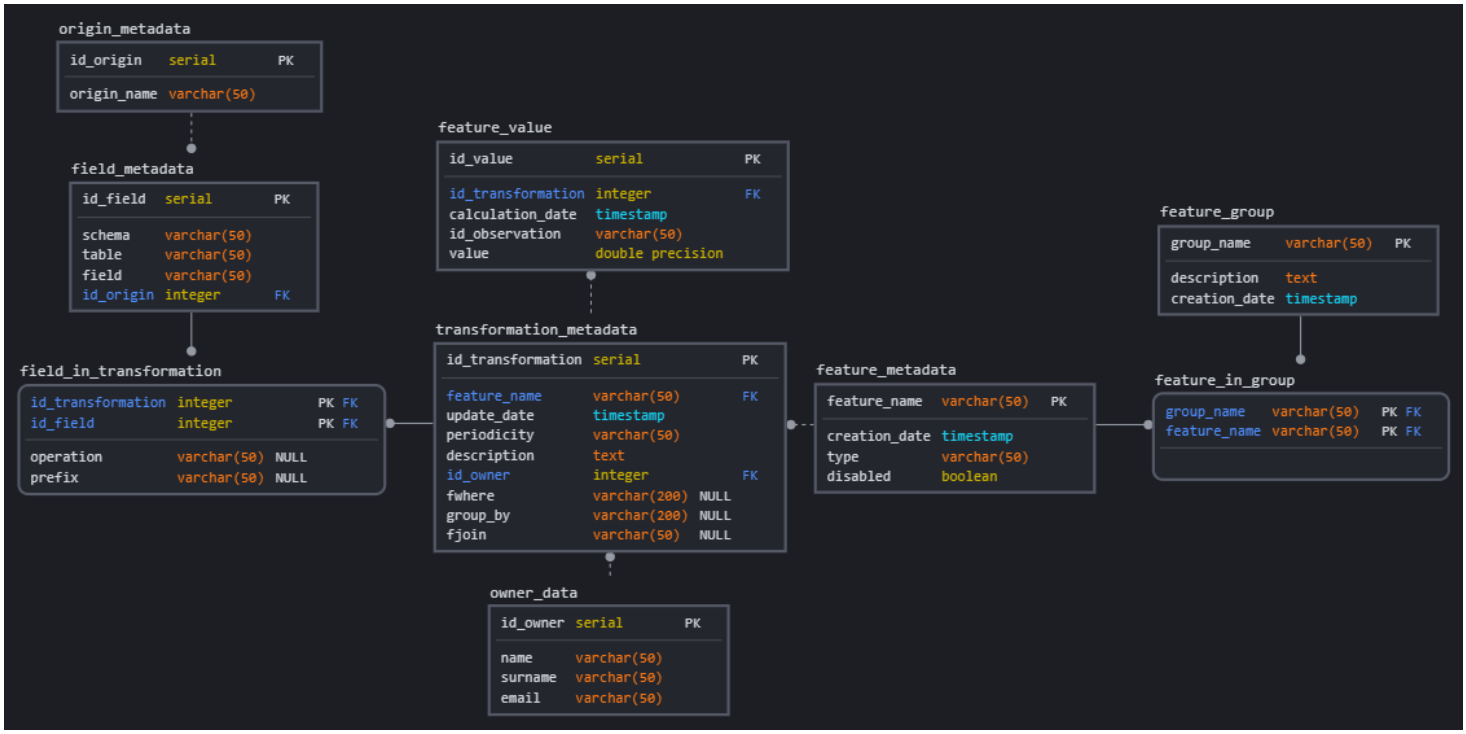

Ilustración 8: Modelo de datos

En esta imagen podemos observar el modelo de datos diseñado. La tabla principal y alrededor de la cual se construye todo el modelo es *feature_metadata*:

Tabla	feature_metadata	
Descripción	Almacena los metadatos básicos y no variables de las features	
Campos	feature_name	Es el nombre de la feature, y será único en el sistema. Permite identificar una feature de forma unívoca.
	creation_date	Fecha de creación de la feature.
	type	Tipo de feature que se almacena. Puede ser integer, float, boolean, etc.
	disabled	Identifica si la feature se encuentra activa o por el contrario ha sido deshabilitada y por tanto ya no se calcula.

Tabla 63: Tabla feature_metadata

A la derecha de esta tabla encontramos otras dos, *feature_group* y *feature_in_group*. La primera, almacena la información relacionada con las agrupaciones lógicas de features, mientras que la segunda permite conectar la tabla de features con la de grupos.

Tabla	feature_group	
Descripción	Almacena los metadatos de los grupos o sets de features.	
Campos	group_name	Nombre único que identifica al grupo
	description	Descripción aportada por el usuario que crea el grupo.
	creation_date	Fecha de creación del grupo.

Tabla 64: Tabla feature_group

Por otro lado, a la izquierda de la tabla principal, encontramos las tablas de los metadatos que pueden variar a lo largo del tiempo y de los que se quiere mantener un histórico. En primer lugar, encontramos la tabla *transformation_metadata*:

Tabla	transformation_metadata	
Descripción	Almacena los metadatos variables de una feature y establece una fecha para mantener un histórico de dichos cambios.	
Campos	id_transformation	Clave subrogada que permite identificar de forma unívoca los metadatos de una de las posibles transformaciones que tenga una feature a lo largo del tiempo.
	feature_name	Nombre de la feature a la que pertenecen los metadatos.
	update_date	Fecha de actualización de los metadatos. Permite llevar el registro histórico de las modificaciones realizadas. En caso de modificar algún metadato, se establecerá un nuevo id_transformation (clave subrogada que permite conectar con otras tablas de forma sencilla) y mediante el nombre y la fecha de actualización se identifica realmente de qué feature se trata y en qué momento del tiempo.

	periodicity	Periodicidad con la que se calculará la feature en los procesos batch. Este dato será el que reciba el scheduler para saber cada cuánto actualizar los valores de la feature.
	description	Descripción aportada por el usuario sobre la feature.
	id_owner	Identificador del dueño de la feature, que será el encargado de asegurar que se calcula de forma correcta y de dar acceso a los demás usuarios. Con relación a este identificador encontramos la tabla inferior a esta, donde se almacena el nombre, apellidos y correo electrónico del dueño para que sea posible mandarle un aviso en caso de fallo.
	fwhere	Sentencia "where" de la query de cálculo de la feature, es decir, la condición que debe cumplir alguno o algunos campos para realizar el cálculo.
	group_by	Sentencia "group by" de la query de cálculo de la feature.
	fjoin	Sentencia "join" de la query de cálculo de la feature. Este dato podría ser omitido en caso de que estuviese establecido cómo se unen todas las tablas del almacenamiento en crudo. Sin embargo, esto es una tarea muy compleja y larga, por lo que no se ha tenido en cuenta y se ha simplificado almacenando esta sentencia.

Tabla 65: Tabla transformation_metadata

A la izquierda de esta tabla encontramos otras tres colocadas en vertical una encima de otra. Estas son *origin_metadata*, *field_metadata* y *field_in_transformation*. La primera de ellas es muy sencilla y actualmente no tiene información real. La idea de esa tabla es almacenar toda la información del origen de los datos en crudo, es decir, el origen desde el cuál serán volcados al repositorio común de datos. Esto permitiría identificar si se trata de otra base de datos, un CSV, un JSON, cómo se conecta a ese origen, como se mapean esos datos hacia el destino de datos en crudo, etc. Actualmente, esta funcionalidad no se encuentra implementada, por lo que no se ha entrado en más detalles y simplemente se ha dejado mencionada la idea.

La siguiente tabla debajo de esta (*field_metadata*), almacena todos los campos existentes en el almacenamiento en crudo. De esta manera, es posible saber qué información está disponible para ser utilizada para crear nuevas features y qué campos se están utilizando ahora mismo:

Tabla	field_metadata	
Descripción	Almacena los campos existentes en el almacenamiento en crudo.	
Campos	id_field	clave para identificar de forma única un campo.
	schema	Esquema dentro del cual se encontrará el campo. Dentro de la base de datos en crudo, habrá muchos esquemas con varias tablas cada uno y cada esquema hará referencia a un origen de datos.
	table	Tabla donde se encuentra el campo.
	field	Nombre del campo.
	id_origin	Clave foránea para identificar futuro origen de esos datos.

Tabla 66: Tabla field_metadata

Además, encontramos la tabla *field_in_transformation*, que permite identificar, para un id de transformación, es decir, para una feature con unos metadatos en una fecha y hora concreta, qué campos se ven involucrados en ese cálculo y de qué manera, indicando tanto el prefijo (SUM, AVG, MIN, MAX) típico del lenguaje SQL, como la operación (multiplicación, suma, resta, división). De esta forma, campos pertenecientes a una transformación o cálculo con prefijo y operaciones comunes se relacionan de forma sencilla para establecer que forman parte de la misma operación. Actualmente se tiene limitado el cálculo a que todos los campos pertenecientes a la transformación deben tener el mismo prefijo y operación, pero la complejidad de los cálculos puede ser adaptada actualizando el código.

Finalmente, en la parte superior central encontramos la tabla *feature_value*. Esta tabla permite almacenar todos los valores de las features que se han calculado a lo largo del tiempo. Aunque se muestre dentro del mismo modelo de datos y exista una relación con otra tabla, realmente se encuentra separada. Esta tabla se almacena en Snowflake, en la nube, mientras que todas las anteriores se almacenan de forma local en PostreSQL:

Tabla	feature_value	
Descripción	Almacena los valores de todas las features.	
Campos	id_value	Identificador único de ese valor.
	id_transformation	Identificador de la transformación, que permite saber a qué feature hace referencia ese valor y en qué momento del tiempo. Con las relaciones a las demás tablas es posible saber qué cálculo y qué campos estaban asociados a ese valor en ese momento concreto, quién era el dueño de la feature en el momento en el que se obtuvo ese valor, qué condiciones existían, con qué frecuencia se calculaba, etc. En resumen, con este campo podemos relacionar un valor concreto con todos los metadatos necesarios de la feature en un momento concreto.
	calculation_date	Fecha y hora en la que se realizó el cálculo.
	id_observation	Identificador de la observación del valor. Permite saber a qué hace referencia un valor. Por ejemplo, si la feature que se está calculando es el volumen de ventas por empleado diariamente, para cada uno de los empleados se generará un valor de esa feature cada día. El id de observación hace referencia al identificador del trabajador, de tal forma que es sencillo saber para un trabajador, en una fecha, qué volumen de ventas tuvo. En caso de que la feature sean temperaturas medidas en una serie de sensores, el id de observación hará referencia al identificador único que tenga el sensor. De esta forma, la tabla es común para todas las features y es posible generalizar lo suficiente como para englobarlas a todas. Si se quiere saber a qué hace referencia el identificador de observación, basta con mirar los metadatos de esa transformación y tirar del hilo hasta obtenerlo.

	value	El valor de una feature en un momento concreto y para un identificador concreto.

Tabla 67: Tabla feature_value

Para usar la base de datos hay que tener en consideración dos aspectos importantes. En primer lugar, parte de la información se almacena en una base de datos en la nube, lo que significa que es necesaria conexión a internet para realizar las operaciones sobre ella. Por otro lado, hay que tener en cuenta cuándo se está accediendo a la nube y cuándo a la base de datos local, pues las conexiones son distintas.

6. VALIDACIÓN DEL SISTEMA

En este apartado se lleva a cabo la validación del correcto funcionamiento de la herramienta, comprobando que cumpla con los requisitos y estableciendo si los resultados que se han obtenido son los previstos. Para ello se define un plan de pruebas.

6.1. Plan de pruebas.

En este apartado se define el plan de pruebas a través de tablas con el siguiente formato:

Identificador	PR-YY
Nombre	
Finalidad	
Precondición	
Pasos	
Resultado previsto	
Resultado logrado	
Postcondición	

Tabla 68: Definición del plan de pruebas

Donde los campos de cada tabla definen:

- **Identificador**: Valor alfanumérico compuesto por dos elementos:
  - PR: siglas que corresponden a la palabra “prueba”.
  - XX: valor numérico entre 00 y 99.
- **Nombre**: nombre descriptivo de la prueba.
- **Finalidad**: breve descripción del objetivo del test.
- **Precondición**: situación del sistema antes de realizarla.
- **Pasos**: la acción o acciones que se deben realizar para llevar a cabo la prueba.
- **Resultado previsto**: resultado que se obtiene de la realización del test. Puede ser:
  - Correcto: la prueba ha funcionado correctamente.
  - Error: la prueba no ha funcionado como se esperaba y o a ocurrido un error.
- **Resultado logrado:** resultado real obtenido tras realizar la prueba. También puede ser correcto o erróneo.
- **Postcondición:** situación del sistema tras realizar el test.

6.1.1. Plan de pruebas.

Identificador	PR-00
Nombre	Desplegar la herramienta.
Finalidad	Desplegar la herramienta para que los contenedores y conexiones estén disponibles
Precondición	No haber desplegado aún la herramienta.
Pasos	Ejecutar el comando “docker-compose up --build” en el terminal.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Los contenedores estarán levantados y las conexiones establecidas, siendo posible interactuar con el sistema

Tabla 69: Desplegar la herramienta PR-00

Identificador	PR-01
Nombre	Desplegar la herramienta sin conexión a internet.
Finalidad	Desplegar el Feature Store sin contar con conexión a internet.
Precondición	No haber desplegado aún la herramienta y tener desconectada la conexión a internet.
Pasos	Ejecutar el comando “docker-compose up --build” en el terminal.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se levantarán los contenedores, pero no se establecerá la conexión con Snowflake y se mostrará por tanto un error.

Tabla 70: Desplegar la herramienta sin conexión a internet PR-01

Identificador	PR-02
Nombre	Insertar una feature
Finalidad	Insertar una nueva feature no existente en el sistema.

Precondición	Haber desplegado la herramienta y no haber introducido esa feature previamente.
Pasos	Realizar la llamada a la API con el JSON que incluye los metadatos a ser introducidos.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se habrán introducido los metadatos, se habrá creado el dag y se recibirá una respuesta de que todo ha funcionado correcta,emte.

Tabla 71: Insertar una feature PR-02

Identificador	PR-03
Nombre	Insertar una feature existente
Finalidad	Insertar una nueva feature que ya exista en el sistema.
Precondición	Haber desplegado la herramienta y haber introducido esa feature previamente.
Pasos	Realizar la llamada a la API con el JSON que incluye los metadatos de una feature cuyo nombre ya se encuentre en el sistema.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de error avisando que dicha feature ya se encuentra en el sistema.

Tabla 72: Insertar una feature existente PR-03

Identificador	PR-04
Nombre	Insertar una feature incorrecta
Finalidad	Insertar una nueva feature cuyo esquema del DSL no sea correcto.
Precondición	Haber desplegado la herramienta.
Pasos	Realizar la llamada a la API con el JSON que incluye los metadatos de una feature con un formato incorrecto.

Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de error avisando que el formato indicado no es el correcto.

Tabla 73: Insertar una feature incorrecta PR-04

Identificador	PR-05
Nombre	Listar las features
Finalidad	Listar todas las features que se encuentran en el sistema.
Precondición	Haber desplegado la herramienta y haber introducido features previamente.
Pasos	Realizar la llamada a la API para listar todas las features.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se devolverá una lista con todas las features del sistema.

Tabla 74: Listar las features PR-05

Identificador	PR-06
Nombre	Listar features con el sistema vacío.
Finalidad	Listar las features del sistema sin haber introducido ninguna con anterioridad.
Precondición	Haber desplegado la herramienta y no haber introducido features previamente.
Pasos	Realizar la llamada a la API para listar las features.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se devolverá una lista vacía.

Tabla 75: Listar features con el sistema vacío PR-06

Identificador	PR-07
Nombre	Listar features de un grupo
Finalidad	Listar todas las features pertenecientes a un grupo o set del sistema.
Precondición	Haber desplegado la herramienta, haber creado un grupo y haber introducido features en ese grupo.
Pasos	Realizar la llamada a la API para listar las features de un grupo.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se devolverá la lista con las features pertenecientes al grupo.

Tabla 76: Listar features de un grupo PR-07

Identificador	PR-08
Nombre	Listar features de un grupo vacío.
Finalidad	Listar todas las features pertenecientes a un grupo o set del sistema que no contenga features.
Precondición	Haber desplegado la herramienta, haber creado un grupo y no haber introducido features en ese grupo.
Pasos	Realizar la llamada a la API para listar las features de un grupo.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se devolverá una lista vacía.

Tabla 77: Listar features de un grupo vacío PR-08

Identificador	PR-09
Nombre	Obtener valores de una feature.
Finalidad	Obtener los últimos valores registrados en el sistema para una feature.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.

Pasos	Realizar la llamada a la API para obtener los valores de la feature cuyo nombre se indica en la llamada.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se devolverá una lista con los identificadores y los valores de la feature seleccionada.

Tabla 78: Obtener valores de una feature PR-09

Identificador	PR-10
Nombre	Obtener valores de una feature inexistente.
Finalidad	Obtener los últimos valores registrados en el sistema para una feature que no existe en el sistema.
Precondición	Haber desplegado la herramienta y no haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API para obtener los valores de la feature cuyo nombre se indica en la llamada.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se devolverá un mensaje de error indicando que la feature solicitada no se encuentra en el sistema.

Tabla 79: Obtener valores de una feature inexistente PR-10

Identificador	PR-11
Nombre	Obtener valores de una feature con fecha.
Finalidad	Obtener los valores registrados en el sistema para una feature en una fecha determinada.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API para obtener los valores de la feature cuyo nombre y fecha se indican en la llamada.
Resultado previsto	Correcto
Resultado logrado	Correcto

Postcondición	Se devolverá una lista con los identificadores y los valores de la feature seleccionada en la fecha indicada.

Tabla 80: Obtener valores de una feature con fecha PR-11

Identificador	PR-12
Nombre	Obtener valores de una feature con fecha inexistente.
Finalidad	Obtener los valores registrados en el sistema para una feature en una fecha para la que no existen valores.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API para obtener los valores de la feature cuyo nombre y fecha se indican en la llamada.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se devolverá un mensaje de error indicando que la fecha seleccionada no es válida.

Tabla 81: Obtener valores de una feature con fecha inexistente PR-12

Identificador	PR-13
Nombre	Actualizar una feature.
Finalidad	Actualizar los metadatos de una feature existente en el sistema.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API con el JSON que define los nuevos metadatos.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se mostrará un mensaje de que la actualización ha funcionado correctamente.

Tabla 82: Actualizar una feature PR-13

Identificador	PR-14
Nombre	Actualizar la transformación de una feature.
Finalidad	Actualizar los metadatos de una feature que afectan a la transformación u operación con la que se calcula.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API con el JSON que define la nueva operación.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se mostrará un mensaje de que la actualización ha funcionado correctamente y se creará un dag de Airflow nuevo para cambiar el método de cálculo.

Tabla 83: Actualizar la transformación de una feature PR-14

Identificador	PR-15
Nombre	Actualizar una feature inexistente.
Finalidad	Actualizar los metadatos de una feature que no existe en el sistema.
Precondición	Haber desplegado la herramienta y no haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API con el JSON que define los nuevos metadatos.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se mostrará un mensaje de que en vez de actualizar la feature, esta no existía en el sistema y por tanto se ha introducido en él.

Tabla 84: Actualizar una feature inexistente PR-15

Identificador	PR-16
Nombre	Actualizar una feature con formato erróneo.
Finalidad	Actualizar los metadatos de una feature con un esquema de metadatos erróneo.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API con el JSON incorrecto que define los nuevos metadatos.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de que la actualización no se ha llevado a cabo porque el formato no es el correcto.

Tabla 85: Actualizar una feature con formato erróneo PR-16

Identificador	PR-17
Nombre	Deshabilitar una feature.
Finalidad	Deshabilitar una feature para que deje de calcularse con la periodicidad indicada.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API para deshabilitar la feature.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se mostrará un mensaje de que ha funcionado correctamente, se habrán actualizado los metadatos correspondientes y se habrá pausado el dag de Airflow de la feature.

Tabla 86: Deshabilitar una feature PR-17

Identificador	PR-18
Nombre	Deshabilitar una feature inexistente.
Finalidad	Deshabilitar una feature que no se encuentre en el sistema.
Precondición	Haber desplegado la herramienta y no haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API para deshabilitar la feature.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de que la feature no se encuentra en el sistema.

Tabla 87: Deshabilitar una feature inexistente PR-18

Identificador	PR-19
Nombre	Lanzar una ejecución
Finalidad	Lanzar la transformación de una feature bajo demanda.
Precondición	Haber desplegado la herramienta y haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API que lanza la ejecución del dag de Airflow de una feature en el momento.
Resultado previsto	Correcto
Resultado logrado	Correcto
Postcondición	Se mostrará un mensaje de que ha funcionado y se habrá ejecutado el dag, insertando nuevos datos en la tabla de valores.

Tabla 88: Lanzar una ejecución PR-19

Identificador	PR-20
Nombre	Lanzar una ejecución de feature inexistente.
Finalidad	Lanzar la transformación de una feature que no se encuentra en el sistema.
Precondición	Haber desplegado la herramienta y no haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API que lanza la ejecución del dag de Airflow de una feature en el momento.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de error indicando que la feature solicitada no existe.

Tabla 89: Lanzar una ejecución de feature inexistente PR-20

Identificador	PR-21
Nombre	Insertar una feature con operaciones no soportadas
Finalidad	Insertar una feature en el sistema que tenga operaciones o cálculos no soportados por el sistema.
Precondición	Haber desplegado la herramienta y no haber introducido la feature en el sistema.
Pasos	Realizar la llamada a la API que inserta una nueva feature.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de error indicando que las operaciones no están soportadas.

Tabla 90: Insertar una feature con operaciones no soportadas PR-21

Identificador	PR-22
Nombre	Insertar una feature con datos no disponibles.
Finalidad	Insertar una feature en el sistema cuyos datos en crudo no se encuentren en el almacenamiento de datos en crudo.
Precondición	Haber desplegado la herramienta, no haber introducido la feature en el sistema y que no existan los datos en el almacenamiento en crudo.
Pasos	Realizar la llamada a la API que inserta una nueva feature.
Resultado previsto	Error
Resultado logrado	Error
Postcondición	Se mostrará un mensaje de error indicando que los campos solicitados para la operación no se encuentran en el sistema.

Tabla 91: Insertar una feature con datos no disponibles PR-22

7. CONCLUSIONES

Para finalizar este proyecto, es necesario comentar los aspectos más relevantes que se han observado a lo largo del desarrollo del mismo.

En primer lugar, con respecto a los problemas encontrados, cabe destacar el trabajo de investigación previo a comenzar con la implementación. La falta de conocimiento sobre el concepto de Feature Store y de todo lo que este conlleva supuso un punto de partida complicado para entender el objetivo. Por otro lado, el uso de tecnologías completamente nuevas para el alumno ha supuesto muchas horas de investigación y pruebas para familiarizarse con su uso y poder sacarles el máximo partido posible. Entre estas herramientas, cabe destacar Docker y Airflow. La primera ha supuesto problemas relacionados con la gran cantidad de documentación sobre ella, que, en ocasiones, lejos de ayudar a encontrar los problemas, suponía un mayor tiempo de búsqueda tratando de encontrar la solución al error obtenido. Por otro lado, con respecto a Airflow, los problemas han ido principalmente también relacionados con Docker, en cuanto el despliegue del contenedor y su comunicación con el resto de los servicios del sistema.

Por otro lado, el uso de Snowflake no ha supuesto un problema, ya que tiene una interfaz muy intuitiva y es muy parecido a otros gestores de bases de datos. Además, habiendo realizado la primera iteración del proyecto en PostgreSQL, sólo fue necesario adaptar algunas conexiones y consultas para integrarlo.

Con respecto a las librerías de Python utilizadas, no hay ninguna destacable que haya supuesto un problema, pero todas han tenido un proceso de aprendizaje de las mismas y de investigación para conocer sus potenciales.

Cabe destacar que todos estos problemas no han sido resueltos el cien por cien por completo por el alumno, ya que la asistencia que se ofrecía por parte de la empresa para resolver dudas y acelerar los procesos de aprendizaje ha sido de gran ayuda.

También cabe comentar los conocimientos adquiridos, que han sido muchos a lo largo de los seis meses de proyecto. En primer lugar, destacaría todo el concepto alrededor del Feature Store, y la introducción a metodologías como DevOps, MLOps, etc. También en tecnologías nuevas nunca vistas por el alumno, como son Docker, Airflow y Snowflake, que actualmente son muy utilizadas y necesarias para cualquier puesto de trabajo en el sector, dando un gran aporte al currículum del alumno para su desempeño futuro. También se ha obtenido un gran conocimiento de cómo afrontar un proyecto de principio a fin, desde las fases de reunión con el cliente para determinar qué se quiere realizar hasta la finalización de todos los componentes.

Por otro lado, es necesario comentar la importancia de este trabajo. Así como se considera que las prácticas en empresa son muy necesarias para una mejor compresión del mundo laboral y adquirir una visión distinta al trabajo teórico realizado en las clases, el Trabajo de Fin de Máster tiene una función similar.

La mayor parte de los trabajos realizados durante el máster están muy acotados y pensados para llegar a un objetivo concreto y encontrar unos problemas específicos por el camino. Sin embargo, en este proyecto queda patente cómo es la realidad de un ingeniero de datos, tanto a nivel de formación, siendo necesario un constante aprendizaje de las nuevas tecnologías que surgen en el mercado, como a nivel de soluciones, teniendo un problema complejo delante y teniendo que buscar soluciones a ello sin haberse enfrentado previamente a algo similar.

Además, muchos de los recursos utilizados son nuevos para el alumno, y aunque las asignaturas cursadas son de gran ayuda para comprender muchos aspectos, es necesaria una labor de investigación y aprendizaje individuales para sacar adelante las ideas y diseños planteados.

7.1. Mejoras del sistema.

En este apartado se quieren comentar algunos aspectos planteados que no se han podido implementar pero que podrían mejorar el sistema implementado.

En primer lugar, cabe destacar la falta de tiempo en el proyecto y el carácter individual, lo que ha hecho que muchas de las piezas o ideas que se presentan en el análisis del sistema finalmente no se hayan podido llevar a cabo. Por ello, faltan elementos como el microservicio de generación de features mediante spark, que permitiría generalizar mucho más el sistema. También sería muy importante añadir una herramienta de extracción de datos como Airbyte, para volcar los datos de los diferentes orígenes al almacenamiento en crudo. Finalmente, sería interesante desarrollar tanto el SDK como la interfaz de usuario para interactuar con el sistema.

Por otro lado, la herramienta sólo está pensada para la creación de features en batch, pero una parte muy importante de los datos y cada vez mayor es la información en tiempo real. La obtención de datos en streaming es muy necesaria en un sistema como este, pues completaría la segunda pierna de las capacidades del mismo.

Además, habría que realizar mejoras generales en el sistema, para cumplir con todos los requisitos esperados en el mismo y que sea algo más que un producto mínimo viable.

Finalmente, sería interesante desplegar la herramienta en un entorno productivo, con datos de prueba, pero en gran cantidad y con un grupo de personar interactuando con él para obtener conclusiones y mejorar que podrían realizarse.

7.2. Ideas de comercialización del sistema.

En este apartado se van a comentar algunas ideas que permitirían sacar beneficio de la aplicación.

La principal forma en la que se podría obtener beneficio económico y que la empresa donde se ha realizado el trabajo tiene, es la venta de la herramienta para ser desplegada y mantenida en distintas empresas.

El concepto de Feature Store cada está más extendido en el mundo del Big Data, y cada vez son más las compañías que se interesan por ello. Tal y como se vio en el estado del arte, no existen opciones en el mercado suficientemente buenas como para satisfacer las necesidades de las empresas. Esto hace que un despliegue personalizado para cada caso de uso y compañía cobre una gran ventaja con respecto a los competidores.

8. BIBLIOGRAFÍA

1. Feature Store for Machine Learning – Consultado en enero de 2021.
https://docs.featurestore.org/
2. Guías TFG – Consultado en marzo de 2021.
https://uc3m.libguides.com/TFG/escribir
http://personales.upv.es/fjabad/pfc/comoEscribir.pdf
3. TFG realizado por el alumno en el curso 2019/2020 – Consultado en junio de 2021
https://e-archivo.uc3m.es/handle/10016/32955
4. Evolución del uso de datos – Consultado en febrero de 2021
https://www.statista.com/statistics/871513/worldwide-data-created/
5. Metodología Agile – Consultado en enero de 2021
https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software
6. Aspectos jurídicos – Consultado en febrero de 2021
https://ec.europa.eu/info/law/law-topic/data-protection_es
7. Documentación Docker Desktop – Consultado en marzo de 2021
https://docs.docker.com/get-started/overview/
8. Documentación Apache Airflow – Consultado en abril de 2021
https://airflow.apache.org/docs/
9. Documentación PostgreSQL – Consultado en febrero de 2021
https://www.postgresql.org/docs/
10. Documentación Snowflake – Consultado en abril de 2021
https://docs.snowflake.com/en/
11. Documentación DBeaver – Consultado en febrero de 2021
https://dbeaver.com/docs/wiki/
12. Documentación Hopsworks – Consultado en enero de 2021
https://hopsworks.readthedocs.io/en/stable/
13. Documentación Feast – Consultado en enero de 2021
https://docs.feast.dev/

14. Documentación Amazon SageMaker – Consultado en enero de 2021
https://docs.aws.amazon.com/sagemaker/index.html

15. Documentación Tecton – Consultado en abril de 2021
https://docs.tecton.ai/v1/

16. Documentación Vertex AI – Consultado en junio de 2021
https://cloud.google.com/vertex-ai/docs

17. Documentación Michelangelo Pallette – Consultado en enero de 2021
https://eng.uber.com/michelangelo-machine-learning-platform/

18. Arquitectura principal de un Feature Store, Tecton – Consultado en febrero de 2021
https://www.tecton.ai/blog/what-is-a-feature-store/

9. EXTENDED ABSTRACT

This section aims to show the student's English knowledge and provide an overview of the entire document. Therefore, the most important parts of each chapter are explained.

9.1. Introduction

The objective of this document is to show the evolution of a master's degree in Big Data: Technology and Advanced Analytics. It is a work in which many of the expertise acquired during the master's degree are put into practice, mainly related to the branch of data engineering. In addition, new knowledge related to project management, use of cloud technologies, orchestration and organization of processes and deployment of containers for the virtualization of the tool are acquired.

The job is to design and implement a Feature Store. Within the life cycle of the data, in a context where machine learning or Artificial Intelligence techniques are essential for the company, having a governance system of the characteristics or features used to train the models is key. Not only does it facilitate the governance and reproducibility of models, but it also results in efficiencies associated with the convergence of the use of these characteristics, limiting duplications or, worse, the disparity of obtaining the same concepts by different teams.

The Feature Store, beyond a simple repository of data prepared and ready to be used, is the catalog of features employable by different groups in the same organization where all information associated with data sets has a record of their description, different versions used during the useful life of the set and the profile of the data, so that any incident related to the quality of these can be reported.

Furthermore, by making conscious use of this metadata, the automation capacity results in a more efficient ecosystem, thus being able to plan the preparation of data or alerts so that the impact on the engineering teams is minimal, while speeding up the autonomy of the teams dedicated to the construction of models.

For making this the document, examples and guides relating to the completion of End-of-Degree/Master's Degree Works have been considered (2). It has also been taken as an example the TFG carried out by the student in his studies of degree (3), with the aim of defining in a clear way the sections to be carried out and the content of the same. Several of the sections of this work have been reused from this TFG, although the necessary changes have been made to adapt them to the system to be developed.

9.1.1. Current social context.

Today governance and the use of data is key for any company that wants to improve its performance and make greater profits. As the years go by, the amount of data being

handled in the world is increasing, with the forecast for 2025 being one hundred and eighty-one zettabytes (1021 bytes) of data handled:

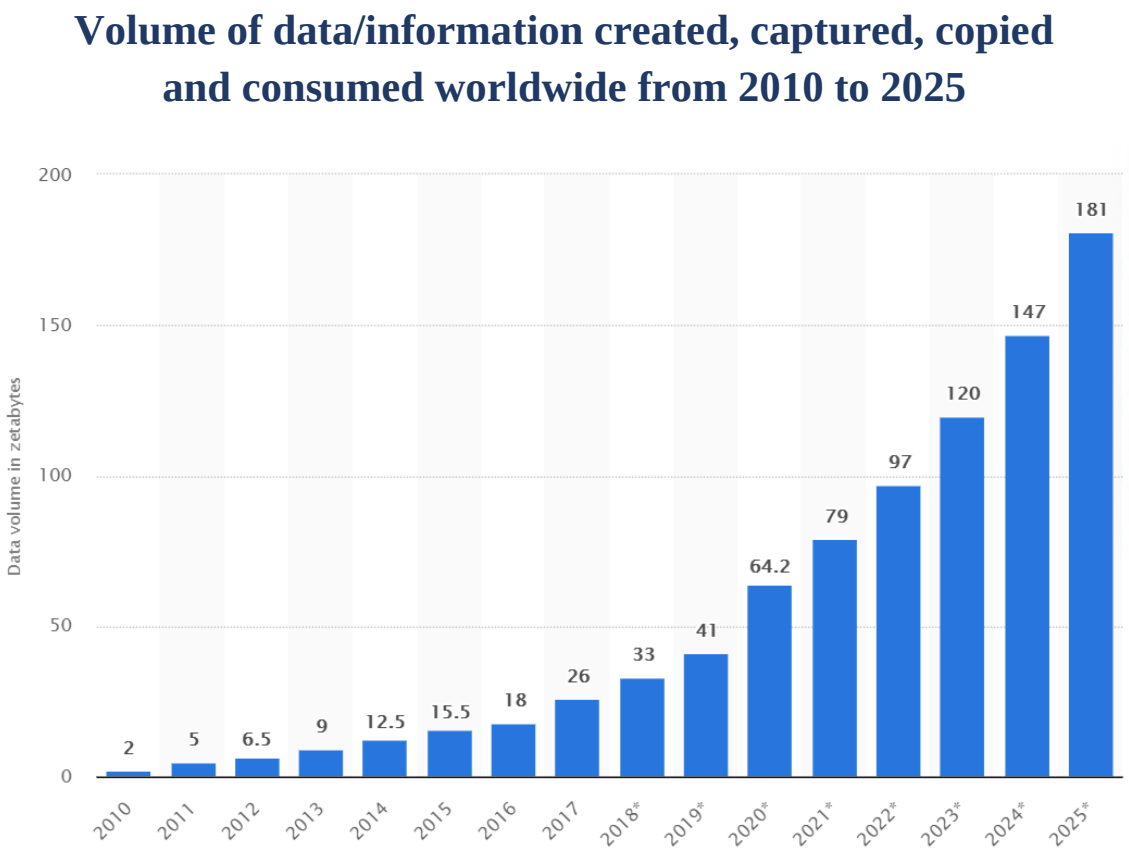

Illustration 9: Volume of data managed worldwide from 2010 to 2025 (4)

Within these needs arises Big Data as a concept, massive amounts of data that, due to its complexity, speed of growth or difficulty to be captured, need non-traditional computer techniques and applications for its processing and storage. With a good management of this data, it is possible to obtain information that allows to direct or guide the decisions of a company.

Since the appearance of Big Data, it has evolved a lot, being increasing the needs that are generated in the sector. One of them is the communication between the different elements involved in the life of the data. From obtaining them, their transformation, their use in models, the visualization of the results in a clear and precise way, to the decision-making based on them. These are long processes, which go through many phases, and in which communication between them is key to obtaining a final product, that is, information, with the highest possible quality. That is why, among many solutions reached, the idea of Feature Store has been defined.

This concept has been used and developed by multiple companies, such as Google, Uber, Amazon, Netflix, etc. These companies have a massive amount of data, much of which must be processed in real time.

As a result of these needs, partially open-source solutions have also been developed, such as Hopsworks or Feast, which will serve as an example and inspiration for this work. However, they do not have all the desirable features that a Feature Store should have, as will be explained in later sections.

Therefore, this project aims to solve one of these communication needs, which is to provide data scientists, that is, the people in charge of defining and executing the models, with a platform that helps them to the governance and calculation of the variables or characteristics needed. With this tool, the process of obtaining and cleaning the data to be used is significantly reduced, being only necessary to access the global repository with the features already defined. In addition, the work aims to improve the capabilities of other similar tools and provide companies with a complete product, with all the necessary features for its deployment.

9.1.2. Work motivation.

The present work has as motivation to deepen in the world of data engineering and increase knowledge in this branch of Big Data. Being the previous studies of the student related to the world of information, this is the branch with more relation to these, so it supposes an extra impulse of interest in the deepening of the matter.

In addition, by acquiring some experience within the company and see the needs that exist, it is discovered that this tool is a very powerful engine to promote the use of Big Data in companies, providing a starting point where to govern the data and perform a good management of them. This vision is the same that is shared in the company where the development of the TFM is carried out, promoter of this idea and with a great interest is the new technologies and methodologies that arise around this field.

Therefore, with the motivation of developing a professional career related to data, and more specifically in the profile of data engineer, it is decided to develop the project of a Feature Store, to acquire knowledge and real experience that will serve in a future work.

9.1.3. Objectives.

Therefore, the objectives of the project are:

- Develop a Feature Store using existing ones as a reference.
- Apply the knowledge acquired throughout the master's degree, both at the programming level and at the level of documentation and project management.
- The versioned and historical storage of the features.
- The ability to define new features, so that they are calculated periodically and stored in a common repository.
- The ability to update feature definitions.
- The possibility of obtaining the values of the features, both the most current and at a specific time.
- The possibility to group features into groups with common characteristics.

- The possibility to disable features once its periodic calculation is not necessary.

9.2. State of art

9.2.1. Critical to the state of art.

The market for Big Data solutions and tools is very broad today, being increasing due to the great demand by companies to become data driven.

Among all these tools, the term Feature Store emerges in recent years as a system capable of helping companies achieve that goal of being managed by the information provided by the data, being a centralizing element and with multiple management tools. However, as will be seen in the following sections, the proposed solutions that are currently on the market are not mature and complete enough to represent a great change within companies. That is why this project is proposed, with the aim of developing a complete Feature Store, with all the necessary functionalities to be implemented in a company and begin to be useful from the beginning.

9.2.2. Study of the current market.

This section shows all the alternatives that exist on the market, which are increasing and have been increasing even during the development of this project. Of each of them the most relevant aspects will be discussed to finally make a comparison in tabular format that allows a quick visualization of the capabilities and shortcomings of each of them.

9.2.3. Competing tools.

9.2.3.1. Hopsworks (12).

Itis an open-source platform for the storage, governance and historical of features, available both on-premises and on the AWS and Azure cloud platforms. It is a very complete tool, with user interface for the discovery of features and quality rules to report possible errors in the fields. Data ingestion can be both batch and streaming. However, it is a feature storage system with some extra functionality, but it does not have a processing system that allows you to calculate the features periodically, as well as define their calculation method. It therefore lacks one of the most interesting features of a feature store.

9.2.3.2. Feast (13).

Itis about another similar platform, also open source, which allows the storage of features. It does not have a user interface, so all the functionalities are executed through an SDK that implements different functions. It has both online and offline storage and the possibility of ingesting batch and streaming data. Again, it does not have the possibility to define the features and plan the calculation of these on a regular basis.

9.2.3.3. Amazon SageMaker (14).

Plataforma very similar to the previous ones developed by Amazon and free software. It enables developers and data scientists to easily prepare, create, train, and deploy models. It has many interesting features that complement the feature store. It works heavily with the Amazon ecosystem, using AWS Batch for periodic computation and Amazon Kinesis for real-time processing. However, this processing is done and orchestrated using another tool, Data Wrangler, also from Amazon. Therefore, it is not a complete tool with the desirable features of a feature store.

9.2.3.4. Tecton (15).

It is a very complete platform, which not only has the functionalities of storing features, but also allows the definition of new features in a simple way based on a schema. In this way, a new very relevant functionality is added, such as the periodic or real-time calculation of raw data obtained from abroad for later dump in global storage. It is therefore the first complete alternative to the basic requirements expected in a Feature Store. However, the tool is paid, so it may not turn out to be the best of the alternatives.

9.2.3.5.Vertex AI (16).

This is the Feature Store developed by Google, which takes Google Cloud as storage. However, it is a very poor solution, which is in development, and which does not yet have the possibility of calculating the features. Therefore, it is a simple storage or global repository of features, without completing the rest of the necessary functions.

9.2.3.6. Michelangelo Pallette (17).

It's about the Feature Store developed by Uber. This platform is very complete, allowing all the processing of features that are generated in real time around the world with the different services of the company, both in the taxi system and in the delivery of food at home. However, it is not freely accessible, so it is not a viable alternative when deploying a Feature Store in a different company.

9.2.4. Desirable features.

Once all the alternatives have been compared, we can identify several desirable features that a Feature Store should have:

- **Domain Specific Language** (DSL): You must have a simple syntax domain language that allows you to define in broad outline the operations to be performed and the details of these in terms of availability and frequency.
- **Centralized access**: You must propose an architecture that allows the fast access of the features and in a unique way.
- **Serving**: The system will allow you to interact with it in such a way that it is easy to obtain the features calculated in it or to implement new ones through the DSL.

- **Versioning and time-travel**: Since the features are part of the iterative process of the data scientist during the Discovery phase, the Feature Store must have mechanisms for versioning the features as well as time-travel to guarantee their reproducibility.
- **Monitoring**: Since they are automated processes on data "in motion" there must be a mechanism that monitors or reports both the evolution of the data and the potential modification of the quality of the features consolidated in the warehouse.

9.2.5. Final comparison.

Based on these desirable characteristics and the comparison of each tool, you can see in the form of a table a global view of everything mentioned:

Item to compare	HopsWorks	Feast	SageMaker	Tecton	Vertex AI
Source	Batch y Stream	Batch y Stream	Batch y Stream	Batch y Stream	Batch y Stream
ETL Processes	No	No	No	Yes	No
DSL	No	No	No	Yes	No
Serving	Yes	Yes	Yes	Yes	Yes
Time travel	Yes	Yes	Yes	Yes	Yes
monitoring	Yes	No	Yes	Yes	No

Table 92: Comparison of Feature Stores in the market

9.2.6. Proposal.

Therefore, seeing the results obtained in the previous comparison, we can determine that there is no sufficiently complete tool that covers the detected needs. The lack of processing by most of them, as well as the cost of some others makes them not the most complete alternatives.

Therefore, it is proposed to carry out the development of its own Feature Store, made from scratch, adapted to current needs and with the most modern technologies possible. The main objective of the project is the development of a minimum viable product, which will allow us to lay a foundation of what could be offered to companies in the future in the form of a complete system.

This system must have the main deficiency studied in the alternatives of the market, such as the processing of data, to avoid problems related to communication between those in charge of data processing and those in charge of the creation and execution of models.

9.3. Valuation and selection of design alternatives.

Once the design alternatives have been analyzed, the different options must be evaluated, and the optimal ones chosen.

About storage, considering the company's preferences and being a booming platform, it has been decided to use Snowflake. This storage will not only allow to save the raw data, the versioned features and grow in a scalable way, but also has internal parallel processing (MPP), so it would also serve as a data processing system.

Regarding metadata, avoiding the use of the cloud and unnecessary expenses, it has been decided to make a local deployment of PostgreSQL, being a database known by the student and with a lot of documentation available.

Regarding the orchestrator and scheduler, the decision has been made to use Airflow, a very complete tool that allows you to create DAGs to run all kinds of processes and run them both periodically and on demand. In addition, through the user interface it is possible to easily monitor whether the operations are working correctly.

Finally, about the serving part, it has been decided to develop an API, the call to a series of functions with which it will be possible to interact with the whole system. The development of an SDK is somewhat more complex, as well as the user interface, so they have been left as possible to improve the system.

9.4. System analisys

In this chapter, a system analysis is performed with the objective of defining the main functionalities of the system. To do this, the use cases are first defined, describing the steps to perform different actions. Based on this study, the requirements are developed, which mark how the tool should be developed and what functionalities it should include.

9.5. System desing

The objective of this section is to solve the problem raised in the analysis of the system. To do this, both the architecture of the system and its operation are explained, in order to understand how the developed application behaves.

9.5.1. System arquitecture

As indicated in the State of the Art, the basic elements to build a feature store are:

- El data source, both batch and real-time, the transformation of the data
- El storage and historical record of features
- Una layer of serving to interact with the system
- Everything is orchestrated and monitored.

Based on all these components, and following the recommendations and needs of the company, the following architecture has been achieved:

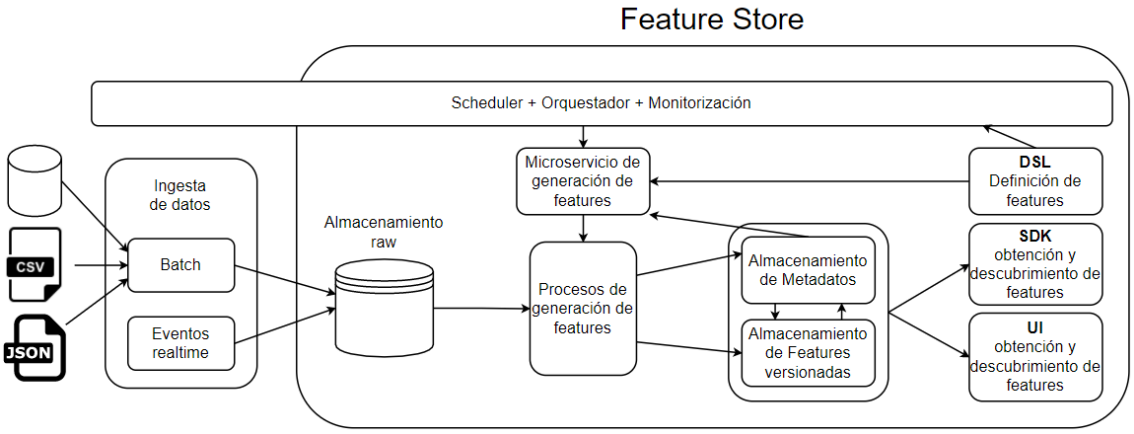

Figure 10: System Architecture

As you can see, there are many possible data sources, such as CSV, JSON, other databases, and so on. That data can be extracted from these sources by batch processes and dumped to raw or raw data storage. The same goes for real-time events. Once this data is available in raw storage, it is possible to calculate features with it. Through feature generation processes, the data is transformed as defined and will be stored as versioned features (historical). To connect to the different sources and launch these processes, a feature generation microservice will be used. In addition to the calculated values, there is a metadata store, where you will find all the additional information, including the definition of the transformation.

As you can see, we have opted for an ELT model (Extract, Load and Transform). This is because the complexity associated with processing the data on the fly as it is obtained from the sources is very high, since it would imply a level of generalization and adaptation to the different sources outside the scope of a project like this. Therefore, it has been chosen to have a raw storage, which allows to store all the data necessary for the calculation of the features with a similar tabular format, in such a way that it is accessible with simple queries in the SQL language.

As a serving layer we find three main elements. First, a DSL, with which you can define the features in a simple way. This language will be very similar to SQL, being well known by everyone and simple to use. In addition, it allows you to easily understand how the transformation of raw data into features will occur, using sums, means, minimums, etc.

To interact with the system, we find two alternatives. On the one hand, an SDK, a library with all the available functions, such as listing the features, obtaining values, inserting new features in conjunction with the DSL, etc. This SDK allows you to obtain the values

and store them as datasets, being very easy to combine with other libraries to directly build the models. On the other hand, there could be a user interface, much friendlier but more limited in terms of functionalities.

Finally, there will be a layer common to the whole system that allows to orchestrate the processes, from the extraction of the data from the sources, its dump in the raw storage, its transformation in feature format and its serving to the users of the system. This layer will also be responsible for carrying out these processes periodically and providing monitoring of them to verify that everything works correctly.

9.6. System validation

This section tests the system to verify that everything is working properly. This is done using tables that show the different tests and the result of the tests. With this chapter, the designed system is verified to be of quality and meet the requirements set out in the system analysis.

These tests note that the tool works correctly in most cases, but that there are some aspects that have not been properly covered and that will need to be fixed.

9.7. Conclusion

To finish this project, it is important to comment on the most relevant aspects that have been observed in the development of the same.

First, about the problems encountered, it is worth noting the research work prior to starting with the implementation. The lack of knowledge about the concept of feature store and all that it entails was a complicated starting point to understand the objective. On the other hand, the use of completely new technologies for the student has meant many hours of research and tests to familiarize themselves with their use and be able to make the most of them. These tools include Docker and Airflow. The first has involved problems related to the large amount of documentation on it, which, sometimes, far from helping to find the problems, meant a longer search time trying to find the solution to the error obtained. On the other hand, regarding Airflow, the problems have been mainly also related to Docker, in terms of the deployment of the container and its communication with the rest of the system services.

On the other hand, the use of Snowflake has not been a problem since it has a very intuitive interface and is very similar to other database managers. In addition, having performed the first iteration of the project in PostgreSQL, it was only necessary to adapt some connections and queries to integrate it.

For the rest, both the use of programming languages and Google Maps APIs have been quite simple. The large amount of documentation and examples that exist on the Internet have allowed us to work with them in an agile and correct way.

Regarding the Python libraries used, there is no notable one that has been a problem, but all have had a process of learning them and researching them to know their potentials.

It should be noted that all these problems have not been completely solved by the student, since the assistance offered by the company to solve doubts and accelerate the learning processes has been of great help.

It is also worth mentioning the knowledge acquired, which has been many throughout the six months of the project. First, I would highlight the whole concept around the Feature Store, and the introduction to methodologies such as DevOps, MLOps, etc. Also in new technologies never seen by the student, such as Docker, Airflow and Snowflake, which are currently widely used and necessary for any job in the sector, giving a great contribution to the student's curriculum for their future performance. It has also gained a great knowledge of how to face a project from start to finish, from the phases of meeting with the client to determine what you want to do to the completion of all the components.

On the other hand, it is necessary to comment on the importance of this work. Just as it is considered that internships in companies are essential for a better understanding of the world of work and acquire a different vision to the theoretical work of the subjects, the master's degree Final Project has a similar function.

Most of the work done during the master's degree is very limited and designed to reach a specific objective and find specific problems along the way. However, in this project it is evident how the reality of a data engineer is, both at the level of training, being necessary a constant learning of the new technologies that arise in the market, and at the level of solutions, having a complex problem in front of it and having to look for solutions to it without having previously faced something similar.

In addition, many of the resources used are new to the student, and although the subjects taken are of great help to understand many aspects, individual research and learning is necessary to carry out the ideas and designs proposed.

Therefore, it is of great importance for the student to be able to carry out this type of work to demonstrate that the knowledge acquired is sufficient and to prove to himself that he is qualified for the labor market.

9.8. System improvements.

In this section we want to comment on some ideas that have not been able to be implemented but that could improve the system implemented.

First, it is worth noting the lack of time in the project, which has meant that many of the pieces or ideas presented in the analysis of the system have finally not been able to be carried out. Therefore, there are missing elements such as the microservice of generating features through spark, which would allow the system to be much more generalized. It would also be very important to add a data extraction tool like Airbyte, to dump data from different sources into raw storage. Finally, it would be interesting to develop both the SDK and the user interface to interact with the system.

On the other hand, the tool is only designed for the creation of features in batch, but a very important part of the data and increasing is the information in real-time. Obtaining data in streaming is very necessary in a system like this, as it would complete the second leg of the capabilities of the same.

In addition, general improvements should be made to the system, to meet all the requirements expected in it and to make it more than just a minimum viable product.

Finally, it would be interesting to deploy the tool in a productive environment, with test data, but in large quantities and with a group of people interacting with it to obtain conclusions and improve that could be carried out.

9.9. System marketing ideas.

In this section we will discuss some ideas that would allow us to benefit from the application.

The main way in which you could obtain economic benefit and that the company where the work has been done, is the sale of the tool to be deployed and maintained in different companies.

The concept of Feature Store is becoming more widespread in the world of Big Data, and more and more companies are interested in it. As seen in the state of the art, there are no options in the market good enough to meet the needs of companies. This makes a custom deployment for each use case and company gain a huge advantage over competitors.