ULPGC Universidad de Las Palmas de Gran Canaria | Escuela de Ingeniería Informática

Moonshot Analytics

Memoria Trabajo Final de Grado

Autor: Borja Álvarez Medina

Titulación: Ingeniería Informática

Tutorizado por:

- José Juan Hernández Cabrera
- Íñigo Aramburu Martínez de Estíbaliz

Fecha de presentación: Junio/2022

Agradecimientos

A mi familia, especialmente a mis padres.

Por haberme apoyado en todo momento, por cada uno de sus consejos, por su motivación constante, por los valores transmitidos desde pequeño que me han permitido llegar a donde estoy y ser quien soy ahora y, sobre todo, por su amor y cariño.

A mis amigos.

Por su apoyo incondicional en los buenos y malos momentos durante todo este tiempo.

Resumen

Este proyecto pretende desarrollar un módulo relacionado con el ámbito de la inteligencia empresarial para la plataforma de la empresa Moonshot Innovation S.L. denominado "Moonshot Analytics", que dé solución a problemáticas actuales de falta de información que afectan a los clientes de la empresa.

Como solución se plantea desarrollar un módulo que funcione como una herramienta de extracción, análisis y visualización con gráficas de los datos que se encuentran guardados en los eventos que ocurren en un ecosistema digital y almacenados dentro del data lake del ecosistema.

Se utilizan varios conceptos del mundo del BI como pueden ser la construcción de una arquitectura BI a partir de data warehouses y data marts; técnicas de business intelligence como procesos ETL para la parte de extracción, transformación y carga de datos de diferentes fuentes; análisis multidimensional para el modelado de datos y data visualization con herramientas software de BI para su visualización.

El objetivo es ayudar al propietario de un ecosistema en la toma de decisiones sobre su negocio aportando información de su ecosistema, mediante un conjunto de dashboards para conocer la actividad de su ecosistema y si este está alineado con los objetivos de innovación de su empresa.

Palabras clave: Java, Angular, Docker, PostgreSQL, Metabase, inteligencia empresarial, Business Intelligence, dashboard, KPI, proceso ETL, análisis multidimensional, data lake, data mart, data warehouse, ecosistema digital, innovación, data visualization.

Abstract

This project aims to develop a module related to the field of business intelligence for the platform of the company Moonshot Innovation S.L. called "Moonshot Analytics", which provides a solution to the current lack of information problems that affect the company's customers.

This module will work as a tool for extracting, analysing and visualizing data stored in the events that occur in a digital ecosystem and that are stored within the ecosystem's data lake.

This project uses several concepts from the BI world such as: BI architecture elements like data warehouses and data marts; BI techniques like ETL processes for the extraction, transformation and loading data from different sources; multidimensional analysis for data modelling and data visualization with BI software tools for visualization.

The main objective is to help ecosystem's owners to make decisions about their business by providing information about their ecosystem, through a set of dashboards to know the activity of their ecosystem and whether it is aligned with the innovation goals of their company.

Key words: Java, Angular, Docker, PostgreSQL, Metabase, Business Intelligence, dashboard, KPI, ETL process, multidimensional analysis, data lake, data mart, data warehouse, digital ecosystem, innovation, data visualization.

Índice General

Contenido

1 Introducción. ... 11
1.1 Objetivos iniciales. ... 12
1.2 Competencias específicas cubiertas. ... 13
2 Estado de la cuestión. ... 14
2.1 Moonshot Innovation S.L. ... 14
2.1.1 Plataforma de Moonshot. ... 14
2.1.2 Arquitectura de Moonshot. ... 14
2.1.3 Gestión de la información. ... 15
2.2 Planteamiento del problema. ... 16
2.3 Business Intelligence. ... 17
2.3.1 Arquitectura de BI. ... 17
2.3.2 Procesos ETL. ... 18
2.3.3 Análisis multidimensional. ... 21
2.4 Data visualization. ... 24
3 Implementación de la solución. ... 25
3.1 Arquitectura BI. ... 25
3.2 Análisis multidimensional. ... 26
3.2.1 Tablas de hechos y dimensiones. ... 26
3.3 Moonshot Analytics. ... 29
3.3.1 Proceso ETL. ... 31
3.3.2 Arquitectura del proyecto. ... 36
3.4 Data visualization. ... 37
4 Proceso de desarrollo. ... 46
4.1 Metodología. ... 46
4.2 Tecnologías. ... 48
4.3 Iteraciones. ... 50
4.4 Desarrollo del proyecto. ... 52
4.4.1 Iteración 0. ... 52
4.4.2 Iteración 1. ... 52
4.4.3 Iteración 2. ... 55
4.4.4 Iteración 3. ... 57

4.4.5 Iteración 4. 59
5 Conclusiones. 62
5.1 Resultados 62
5.2 Contribuciones. 64
5.3 Experiencia personal. 65
5.4 Trabajos futuros. 66
6 Bibliografía 67
7 Anexo 69
7.1 GLOSARIO DE TÉRMINOS. 69

Índice de figuras.

Figura 1 Arquitectura modular Moonshot [2] 15
Figura 2 Proceso ETL [8] 18
Figura 3 Esquema estrella [12] 22
Figura 4 Esquema copo de nieve [12] 23
Figura 5 Arquitectura BI Moonshot 26
Figura 6 Ejemplo análisis multidimensional 1 27
Figura 7 Modelo tabla de hechos que mide la actividad de un miembro en un momento determinado 27
Figura 8 Modelo tabla de dimensión de miembro 28
Figura 9 Modelo tabla de dimensión de tiempo 28
Figura 10 Ejemplo tabla de hechos tras proceso ETL 28
Figura 11 Archivo de configuración docker-compose.yaml 29
Figura 12 Fichero de inicialización de PostgreSQL 30
Figura 13 Servicios ETL implementados 31
Figura 14 Configuración tareas SchedulerService 32
Figura 15 Estructura servicio ETL para eventos 33
Figura 16 DatLakeExtractService (extraer por tipo de evento) 34
Figura 17 Ejemplo 1 DatalakeTransformService (Transformación) 35
Figura 18 Ejemplo 2 DatalakeTransformService (Transformación) 35
Figura 19 DimEvent repository (carga de datos) 36
Figura 20 Arquitectura software final Moonshot Analytics (resumida) 37
Figura 21 Ecosystem dashboard 38
Figura 22 Actors dashboard 39
Figura 23 Direct chats dashboard 39
Figura 24 Events dashboard 40
Figura 25 Follow-up room dashboard (Parte 1) 40
Figura 26 Follow-up room dashboard (Parte 2) 41
Figura 27 Members Dashboard 41
Figura 28 Taxonomies Dashboard (Technologies & Business Models) 42
Figura 29 Taxonomies Dashboard (Industries & Social Innovations) 42
Figura 30 Project dashboard (Parte 1) 43
Figura 31 Project dashboard (Parte 2) 43
Figura 32Public channel dashboard 44
Figura 33 Wall post dashboard (Parte 1) 44
Figura 34Wall post dashboard (Parte 2) 45
Figura 35 Proceso iterativo e incremental 47
Figura 36 Ejemplo análisis multidimensional: nº de actores creados y borrados 53
Figura 37 Arquitectura proyecto (iteración 1) 53
Figura 38 Estructura proyecto iteración 1 54
Figura 39 ActorEtlService 54
Figura 40 Interfaz gráfica editor de consultas de Metabase 55
Figura 41 Versión inicial dashboard mi ecosistema 55
Figura 42 Ejemplo análisis multidimensional: Actividad de una publicación 56

Figura 43 Wall post dashboard (Iteración 2) 56
Figura 44 Ejemplo código front-end proyecto..... 57
Figura 45 Prototipo front-end del proyecto 57
Figura 46 Ejemplo análisis multidimensional: taxonomías de un actor 58
Figura 47 Ejemplo código: proceso ETL taxonomías de un actor..... 59
Figura 48 Scheduler Job: Ejecutar procesos ETL 60
Figura 49 Scheduler Job: insertar fechas año actual 60
Figura 50 Resultados análisis SonarLint..... 60
Figura 51 Estructura final del proyecto..... 61

Índice de tablas.

Tabla 1 Diferencias Ralph Kimball VS Bill Inmon 18
Tabla 2 Diferencias esquema estrella vs copo de nieve [12] 23
Tabla 3 Iteraciones del proyecto 50

1 Introducción.

En la era actual conocida como la ‘era digital’ o ‘era de la información’, existen miles y miles de empresas compitiendo constantemente entre ellas y, solo aquellas más competitivas y preparadas consiguen el éxito ofreciendo a los clientes sus bienes y servicios con una mejor calidad y/o a un mejor coste. Con este objetivo en mente, las organizaciones tratan de comprender mejor los deseos y necesidades de sus clientes mediante la recopilación y análisis de datos comerciales que proporcionen información sobre el negocio, , sus productos y sus procesos internos para ser más eficientes y abaratar costes.

Una de las claves para el éxito puede resumirse en el concepto de innovación empresarial y, por eso, cada vez es más frecuente ver como estas empresas tratan de fomentar este aspecto de diferentes maneras con el fin de seguir siendo viables y competitivas. Partiendo de esta premisa, nacen los ecosistemas de innovación con el fin de generar y potenciar las relaciones entre distintas personas y empresas con el fin de estimular ideas innovadoras con las que obtener sinergias positivas para impulsar sus negocios.

Otro pilar fundamental para las empresas es la obtención, análisis y aplicación de la información empresarial mediante el uso del Business Intelligence, es decir obtener información de grandes volúmenes de datos tanto internos como externos a la empresa para generar conocimiento empresarial con el que poder mejorar sus productos, sus servicios, sus procesos internos o cualquier aspecto relacionado con el entorno de la empresa.

Debido a esto, surge la idea que desarrollará este trabajo de aplicar el uso de técnicas de BI dentro de los ecosistemas de innovación con el fin de apoyar a los gestores de innovación de un ecosistema en su toma de decisiones para orientar los procesos internos de innovación de su empresa en su ecosistema.

Con el fin de facilitar la comprensión de algunos términos utilizados en esta memoria se acompaña con un glosario de términos explicados según [1] en el Anexo de esta memoria.

1.1 Objetivos iniciales.

El objetivo principal del trabajo es desarrollar un producto software de BI, dirigido a los gestores de innovación de las empresas clientes que son propietarias de un ecosistema en la plataforma, con el que poder obtener conocimiento a partir de la extracción y análisis de la información de los usuarios y de las actividades que realizan en la aplicación y en los distintos ecosistemas, mostrando los resultados de una manera fácil e intuitiva. Como resultado, los propietarios de los ecosistemas pueden mejorar su toma de decisiones y la propia empresa de Moonshot puede llevar un control más exhaustivo, gestionar de una mejor manera su aplicación tomando mejores decisiones para implementar nuevas funcionalidades o mejorar las existentes que más están siendo utilizadas por la comunidad de usuarios.

Otros objetivos secundarios que pretende alcanzar el trabajo son:

- **A nivel personal:** aprender nuevas tecnologías para tratar grandes cantidades de datos, ahora que el Business Intelligence se considera un aspecto clave y competitivo dentro del entorno empresarial, aprender sobre nuevos conceptos y técnicas relacionados con dicho ámbito y mostrar los análisis efectuados utilizando librerías de gráficos de datos y softwares de BI.

- **A nivel académico:** descubrir nuevos indicadores del nivel de emprendimiento e innovación de los procesos internos empresariales dentro de los ecosistemas de la aplicación y cómo estos se relacionan entre sí.

- **A nivel empresarial:** ser una herramienta útil con la que poder identificar tecnologías innovadoras, mejorar la toma de decisiones e incluso canalizar de una manera más eficiente la inversión de capital a proyectos innovadores.

- **A nivel de producto:** identificar una nueva propuesta de valor que sea relevante para los usuarios y haga más atractivo nuestro producto frente a la competencia.

1.2 Competencias específicas cubiertas.

Competencia específica cubierta vinculada a la mención de ingeniería del software presentada en el TFT01: IS01. Esta competencia se define como:

> "Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software".

Este trabajo cumple con dicha competencia puesto que se desarrolla un proyecto software que cumple con los requisitos especificados por el cliente, además de utilizar metodologías de la ingeniería del software como las metodologías ágiles para el desarrollo del producto que se comentan en el apartado 4.1 y la aplicación de teorías, principios y prácticas de la ingeniería del software como el uso de patrones de diseño.

2 Estado de la cuestión.

2.1Moonshot Innovation S.L.

Como se explicó en la introducción, este proyecto consiste en crear un nuevo módulo de BI por petición de la empresa Moonshot Innovation S.L. para luego agregarlo a su plataforma. Para conocer mejor el problema que se plantea y la solución a este, primero hay que contextualizar el entorno que rodea al proyecto explicando en qué consiste la empresa, cuál es la arquitectura y cómo se estructura su plataforma para ubicar dónde reside el nuevo módulo que se ha desarrollado y sobre qué elementos se basa este módulo para realizar sus funciones. También hay que conocer cómo se gestiona y almacena la información que esta plataforma genera sobre la que se fundamentará todo el proyecto para realizar los procesos de BI correspondientes.

2.1.1 Plataforma de Moonshot.

Moonshot es una plataforma digital de Big Data orientada a eventos que permite crear ecosistemas digitales para empresas con el fin de fomentar la innovación empresarial. Estos ecosistemas de innovación buscan promover la cooperación y conexión entre diferentes stakeholders (empresas, startups, inversores, expertos, universidades, etc.) con la finalidad de mejorar los procesos empresariales internos de innovación y que estos estén alineados con los objetivos estratégicos del propietario de dicho ecosistema. En otras palabras, la tecnología de Moonshot se encarga de conectar entre sí a los diferentes actores dentro de un ecosistema y conectar la innovación con la industria y la inversión.

2.1.2 Arquitectura de Moonshot.

La arquitectura de la plataforma se basa en una arquitectura modular orientada a eventos o event sourcing, donde cada módulo es independiente de otro y, por tanto, pueden añadirse o eliminarse módulos sin que estos afecten al funcionamiento de la plataforma. Como se observa en la figura 1, concretamente este nuevo módulo corresponde al módulo de Dashboard que se encuentra en la capa de datos de la arquitectura de la plataforma y se basa en la información que generan los módulos de creación de ecosistemas de Community y Core del nivel inferior.

El módulo de Community comprende las funcionalidades relacionadas con la interacción entre usuarios de una red social, donde existen varios tipos de chats, programación de eventos y un muro de publicaciones. También se encarga, del manejo de actores y miembros que son los usuarios que integran un ecosistema.

El módulo de Core comprende las funcionalidades básicas de todo ecosistema entre las cuales se encuentran el mapa de las taxonomías que mapean las tecnologías, modelos

de negocio, industrias e innovaciones sociales en las que se pueden clasificar los actores de un ecosistema.

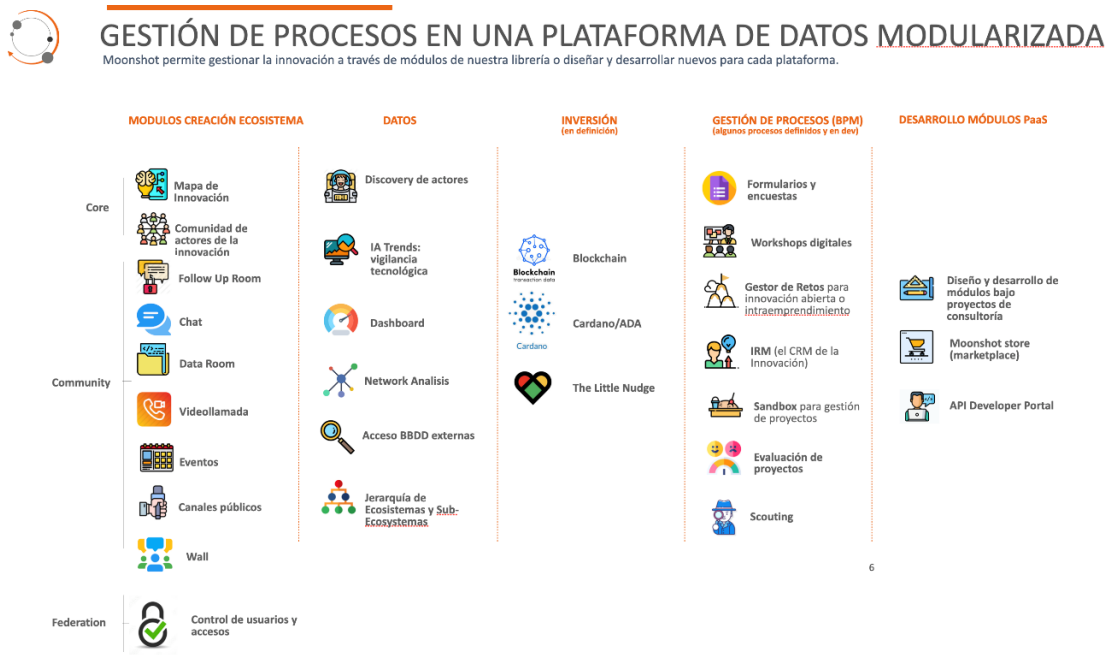


Figura 1 Arquitectura modular Moonshot [2]

2.1.3 Gestión de la información.

Como se explicó en el apartado anterior, la arquitectura de la plataforma es una arquitectura modular orientada a eventos o event sourcing, esto quiere decir que cualquier acción que ocurre en un ecosistema de la plataforma queda registrada dentro de un evento que guarda una amplia variedad de información acerca de la acción que ha ocurrido. Cada uno de estos ecosistemas posee un repositorio de datos o data lake que es el encargado de almacenar estos eventos, para que luego los distintos módulos de la plataforma puedan guardar nuevos eventos generados o acceder a estos eventos y en base a su información desempeñar sus funciones.

El data lake de cada ecosistema consiste en ser un repositorio de datos a nivel de sistema de ficheros, los cuales están jerarquizados en varios niveles siendo el primer nivel de directorio el tipo de evento que almacena, en segundo nivel el año en que ocurre el evento y, como último nivel, el mes en el que ocurre. Este último nivel contiene todos los eventos ocurridos en ese intervalo de tiempo en ficheros de texto ordenados por día y hora de creación de más antiguo a más reciente. De la misma manera, cada fichero contiene toda la información relacionada con el evento que ha ocurrido en el ecosistema como un objeto JSON, es decir, mediante el uso de pares de clave-valor.

2.2 Planteamiento del problema.

Actualmente, Moonshot presenta un producto básico a sus clientes con los módulos correspondientes al primer nivel de su arquitectura expuesta en la figura 1, que son los módulos básicos de Community, Core y Federation pertenecientes a la capa de "Módulos de creación de ecosistemas" para la creación de dichos ecosistemas digitales, la creación y manejo de usuarios que los integran y los distintos medios por los que los usuarios pueden interactuar entre ellos. Aparte de los módulos básicos, cuenta con un módulo perteneciente al nivel de "Gestión de procesos (BPM)" que es el módulo de Gestión de retos para innovación abierta o intraemprendimiento, el cual permite a los gestores de los ecosistemas crear diversos retos y competiciones dentro de la plataforma para fomentar la innovación.

A partir del uso de estos módulos, los gestores de los ecosistemas se quejan de la falta de una herramienta de visualización de información por parte de la plataforma para conocer la evolución en el tiempo y el estado actual de su ecosistema. Este aspecto es crucial para los clientes pues determina si el gasto de la inversión que realiza su empresa, en hacer uso de la plataforma para crear un ecosistema digital que mejore sus procesos internos de innovación, está siendo bien invertido.

Sumado al problema anterior, los mismos gestores también denuncian otro problema muy relacionado que afecta actualmente a varios ecosistemas. Se trata de la falta de dinamización de los ecosistemas de la plataforma por lo que los gestores precisan que la plataforma les muestre información acerca de la actividad que realizan sus usuarios en el ecosistema. Su objetivo es poder evaluar si se están cumpliendo los objetivos estratégicos de innovación de su empresa, y en caso de no cumplirlo, poder tomar mejores medidas que solucionen dicho problema.

Por parte de la empresa, se plantea la ausencia de un módulo todavía sin desarrollar que se encuentra en la capa de "Datos" de su arquitectura que dé solución a los problemas expuestos anteriormente y que sirva para atraer nuevos clientes con los que obtener nuevas fuentes de financiación. Por tanto, se necesita una herramienta cuya tarea principal sea el análisis de los datos que genera un ecosistema y su correspondiente visualización mediante gráficas que aporte información del ecosistema de manera fiable y sencilla a los gestores de este.

En conclusión, la plataforma de la empresa padece de algunos problemas comunes que enfrentan las empresas que no aplican tecnologías de BI a sus negocios y que se explicarán en el siguiente apartado.

2.3 Business Intelligence.

La inteligencia de negocios o BI, por sus siglas en inglés Business Intelligence, se define como la combinación de tecnología, herramientas y procesos que coleccionan, integran, analizan y presentan la información de un negocio. [3]

La inteligencia de negocios debe ser parte de la estrategia empresarial puesto que permite optimizar la utilización de recursos, mejorar los beneficios, monitorear el cumplimiento de los objetivos de la empresa y la capacidad de tomar buenas decisiones para obtener mejores resultados.

¿Qué ventajas aporta a las empresas el uso del Business Intelligence? [4]

La principal ventaja del Business Intelligence es que permite a las empresas entender las fuerzas que moldean los mercados y los negocios, favoreciendo así que puedan adelantarse a la competencia al tiempo que satisfacen las necesidades de los consumidores. Dichas ventajas son:

- Mejora en la toma de decisiones de negocio y la eficiencia operativa.
- Aumento de la satisfacción de clientes y empleados.
- Reducción de costes y aumento de ingresos.
- Elaborar informes, analizar o planificar con mayor rapidez y precisión.
- Obtener datos de mayor calidad y alcanzar mayores ventajas competitivas.

2.3.1 Arquitectura de BI.

Existen muchas metodologías de diseño y construcción de un data warehouse, sin embargo, existen dos grandes metodologías muy populares entre ellas: Ralph Kimball y Bill Inmon. La mayor diferencia entre ambos reside en su sentido de construcción del data warehouse que se explica a continuación, aunque presentan también otras diferencias que se recogen en la tabla 1: [5]

- **Bottom-up (Kimball)**: defiende un modelo modular para construir el data warehouse, donde cada departamento crea su propio data mart de manera independiente. El data warehouse se completa integrando los diferentes data marts.
- **Top-down (Inmon)**: defiende comenzar construyendo primero el data warehouse de la empresa para luego descender a sus diferentes departamentos y crear su data mart, para ello necesita un mayor tiempo de diseño previo para conocer toda la estructura de la empresa y sus procesos.

	Bill Inmon (Top-Down)	Ralph Kimball (Bottom-Up)
Construcción.	Primero data warehouse, luego data marts.	Primero data marts, luego data warehouse.
Modelado de datos.	Relacional: 3ª forma normal. Poco eficiente para análisis.	Dimensional: esquema estrella. Diseñado para análisis de datos.
Actualización de datos.	Continua e integrada.	De forma asíncrona e independiente por data mart.
Carga y transformación de datos.	Proceso unificado.	Cada data mart se encarga de sus procesos.
Mantenimiento y escalabilidad.	Complejo.	Depende de cada departamento y puede escalar agregando data marts.

Tabla 1 Diferencias Ralph Kimball VS Bill Inmon

2.3.2 Procesos ETL.

El término ETL proviene de las siglas en ingles Extract-Transform-Load correspondiente a cada una de las tres fases que integran dicho proceso. Se define como el proceso que permite extraer datos desde múltiples fuentes, transformarlos en un nuevo formato para que puedan ser analizados, y finalmente, cargarlos en un Data Warehouse, una base de datos u otro sistema; con el objetivo de analizar dicha información por parte de las organizaciones para apoyar sus procesos de negocio utilizando diferentes herramientas de analíticas o reportes. [6], [7]

Todos los pasos de este proceso se reflejan en la figura 2.

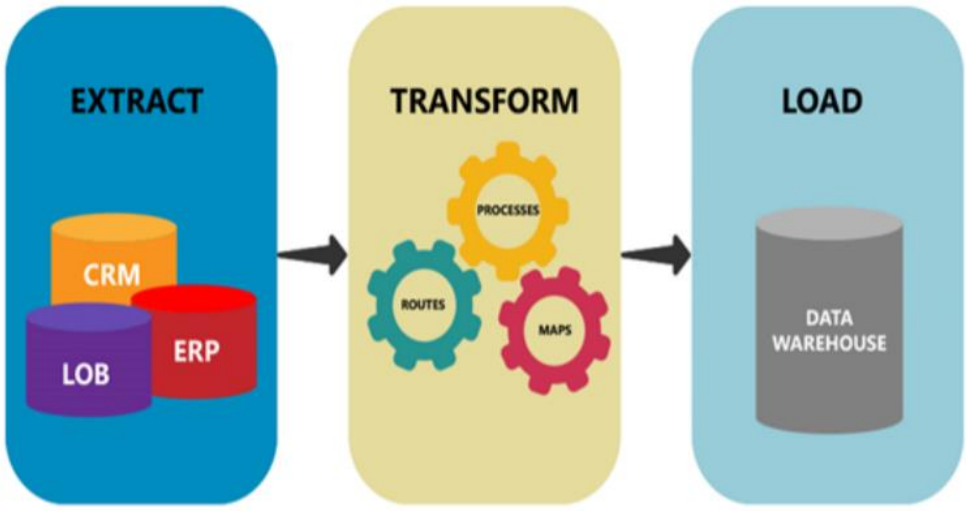


Figura 2 Proceso ETL [8]

Fases de un proceso ETL. [9]

Cada una de estas fases se definen a continuación:

- **Extracción**: se obtiene la información de una o varias fuentes heterogéneas como pueden ser bases de datos, APIs o ficheros de texto y se convierte a un formato adecuado para el proceso de transformación. Un aspecto importante a

tener en cuenta durante esta fase es que debe de causar el menor impacto posible en el sistema origen, puesto que podría ralentizar e incluso colapsar dicho sistema si al extraerse una gran cantidad de datos se produce cualquier pérdida de información. Por eso, se recomienda programar este tipo de operación en las franjas horarias cuando el impacto sea nulo o muy bajo para el rendimiento del sistema.

El proceso de extracción consta de los siguientes pasos:

1) Extraer la información desde el/los sistemas de origen.

2) Analizar la información, realizando una revisión previa que verifique que la información extraída cumple con los criterios esperados.

3) Convertir los datos al formato requerido para iniciar la siguiente fase.

Según las necesidades y los requerimientos de la organización se puede realizar la extracción de datos de una fuente de uno u otro modo:

- **Extracción total** (Full extract): consiste en extraer todos los datos de las fuentes de datos origen, como por ejemplo todos los registros de todas las tablas de una base de datos.

- **Extracción incremental** (Incremental extract): consiste en realizar un procesamiento por lotes para extraer aquellos datos que han sido agregados o modificados desde la última operación.

- **Notificación de actualización** (Update notification): consiste en extraer aquellos datos a medida que se produce alguna actualización cuando se realiza alguna operación de insert, delete o update.

- **Transformación**: se manipula la información extraída del paso anterior aplicando una serie de funciones o reglas de negocio para reformatear y limpiar los datos con el fin de homogeneizar la información de diferentes fuentes bajo un mismo formato. Adicionalmente, en esta fase se incluye la validación y el rechazo de los datos extraídos si no son válidos para su posterior integración.

  A continuación, se enumeran las acciones más habituales de transformación:
  - Agregación.
  - División de una columna en varias.
  - Conversión de unidades.
  - Codificar valores libres.
  - Obtener nuevos valores calculados.
  - Selección de columnas (Filtrado).
  - Traducción.
  - Ordenación.
  - Estandarización.

- Eliminación de duplicados.

- **Carga**: se almacenan los datos provenientes de la fase anterior en el repositorio de información destino como puede ser un data warehouse o una base de datos, con el objetivo de analizar dichos datos para apoyar un proceso de negocio. Esta información se almacena en formato de tablas de dos tipos, las cuales componen la estructura de la base de datos objetivo: las tablas de dimensión y las tablas de hechos.

  Suele considerarse la fase de cuello de botella de todo el proceso ETL por el gran volumen de datos que se debe de manejar y el consumo de tiempo que este proceso conlleva.

  Existen dos formas para desarrollar la fase de carga de datos diferenciándose según el tiempo de carga y el nivel de consistencia de los datos y, cuya elección final corresponde al dueño del proceso ETL:

  - **Acumulación simple:** se realiza un resumen de todas las transacciones comprendidas en un intervalo de tiempo seleccionado y se transporta el resultado como una única transacción hacia el Data Warehouse. Es la forma más sencilla y común de realizar el proceso de carga de datos, pero tiene el inconveniente de que ante un accidente o problema se pierde la consistencia de los datos, pudiéndose dar el caso de tenerse que repetir toda la carga. Sin embargo, la duración del tiempo de carga es menor.

  - **Rolling:** se almacena la información resumida a distintos niveles de forma más escalonada y segura, correspondientes a distintas agrupaciones de la unidad de tiempo o diferentes niveles jerárquicos en alguna o varias dimensiones. Este proceso sería el más recomendable si se busca mantener varios niveles de granularidad de los datos y si el volumen de datos a cargar es importante, puesto que permite retomar el proceso de carga desde un punto concreto sin necesidad de repetir todo el proceso en caso de que ocurra un fallo.

Posibles fallos vinculados a un proceso ETL.

Como todo proceso en la informática, los procesos ETL no están exentos de fallos durante su ejecución y, por tanto, se deben tener en cuenta para tratar de evitarlos o en su defecto mitigarlos. A continuación, se muestran algunos de estos fallos que pueden ocurrir y sus posibles soluciones para minimizarlos:

- **Existencia de campos o valores nulos**: durante el proceso de extracción se desconoce el tipo de datos que se extraen y la información que estos contienen, pudiendo contener varios campos con valores nulos que pueden generar excepciones en las siguientes fases. No obstante, este fallo también puede ocurrir durante el proceso de carga por lo que puede generar excepciones a la hora de cargar los datos en el sistema objetivo como, por ejemplo, al tratar de introducir

un valor nulo dentro de una columna de una tabla de una base de datos SQL que presenta una restricción de tipo "NOT NULL".

Para solucionar este fallo es conveniente analizar los datos que se van a extraer previamente para identificar cada una de las posibles excepciones y realizar procesos de transformación que eliminen la existencia de valores nulos por algún valor por defecto para minimizar los errores que se puedan producir en la fase de la carga de datos.

- **Tablas de referencia inexistentes**: durante el proceso de carga puede ocurrir que no se encuentre la tabla sobre la que se ejecutará las operaciones de inserción o tablas referenciadas mediante sus claves ajenas.

Es recomendable para los dos puntos anteriores manejar un sistema de log de errores que permita trazar los fallos que ocurren para ponerles solución. Su uso permite garantizar que se diseñen procesos robustos que minimicen los fallos anteriormente descritos.

2.3.3 Análisis multidimensional.

El análisis multidimensional representa una forma intuitiva y lógica de analizar la información, ya que los datos se organizan en dimensiones que permiten ofrecer un contexto sirviendo, por tanto, para ver más en profundidad todas las variables reagrupando, organizando y ordenando los datos para dar una visión más compleja y clara de la situación. También se podría definir como: la capacidad de contextualizar una variable o variables (medidas) a través del empleo de perspectivas (dimensiones) entendiendo las medidas como números y las dimensiones generalmente como alfanuméricas. [10]

El análisis multidimensional permite a los usuarios visualizar datos desde diferentes perspectivas, ofreciendo respuestas ágiles a problemas concretos y con un alto nivel de detalle en cada análisis, pero también un acceso directo, eficaz y rápido a la información. En definitiva, ayuda a los gerentes a tener una mejor perspectiva de su negocio ayudando a mejorar la toma de decisiones para conseguir resultados óptimos a través de un análisis pormenorizado de los indicadores claves de rendimiento (KPIs).

Normalmente se usa en los sistemas de procesamiento analítico en línea (OLAP) que permiten analizar volúmenes de datos mediante la utilización de estructuras dimensionales, llamadas cubos OLAP.

Base de datos multidimensional. [11]

Las bases de datos multidimensionales se diferencian de las bases de datos relacionales por la forma en la que guardan y procesan la información, es decir, la diferencia radica en la estructura que forman las tablas que componen la base de datos. En las bases de datos multidimensionales los datos se ven como "cubos de información".

Estos cubos de información añaden una nueva dimensión a las tablas de las bases de datos tradicionales, ya que se encuentran formados por dos componentes:

- **Tablas de dimensiones**: representan aquello que se quiere almacenar con respecto a un tema concreto.

- **Tablas de hechos:** tabla que almacena las medidas -valores numéricos que van a permitir establecer las relaciones entre las distintas dimensiones del cubo OLAP- y las claves que la relaciona con las tablas de las dimensiones.

La estructura de una base de datos multidimensional puede seguir diferentes esquemas entre los cuales encontramos: [12]

- **Esquema estrella:** es la estructura más empleada en las bases de datos multidimensionales, como se puede apreciar en la figura 3 está formada por una tabla de hechos rodeada de una o varias tablas de dimensión. En este esquema se desnormaliza las dimensiones, es decir, existen campos referentes a una subdimensión que se encuentran repetidos en esta tabla.

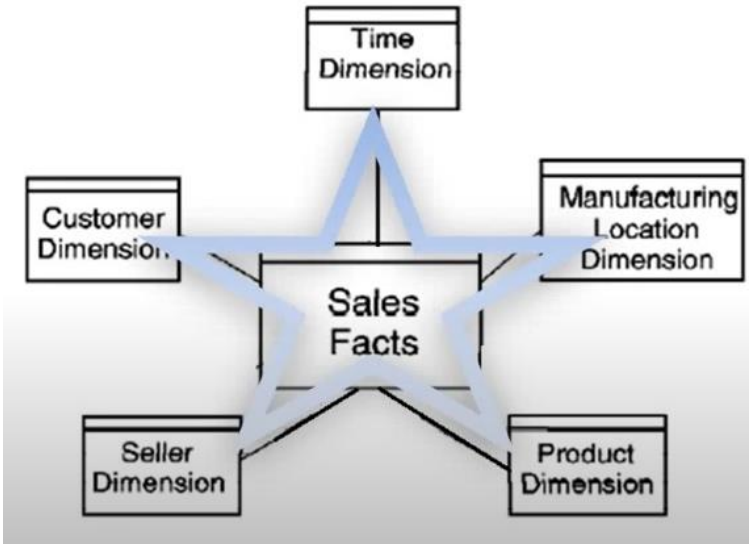


Figura 3 Esquema estrella [12]

- **Esquema copo de nieve:** está formada por una tabla de hechos rodeada de varias tablas de dimensiones que a su vez están compuestas de otras tablas de dimensiones como refleja la figura 4. En este esquema las dimensiones se encuentran normalizadas, es decir, existen campos referentes a una subdimensión que no se encuentran repetidos en esta tabla, pero están referenciados por clave ajena a otra tabla que contiene ese valor solo una vez. Debido a esto, su principal ventaja frente al modelo estrella es que reduce la redundancia y ahorra espacio de almacenamiento.

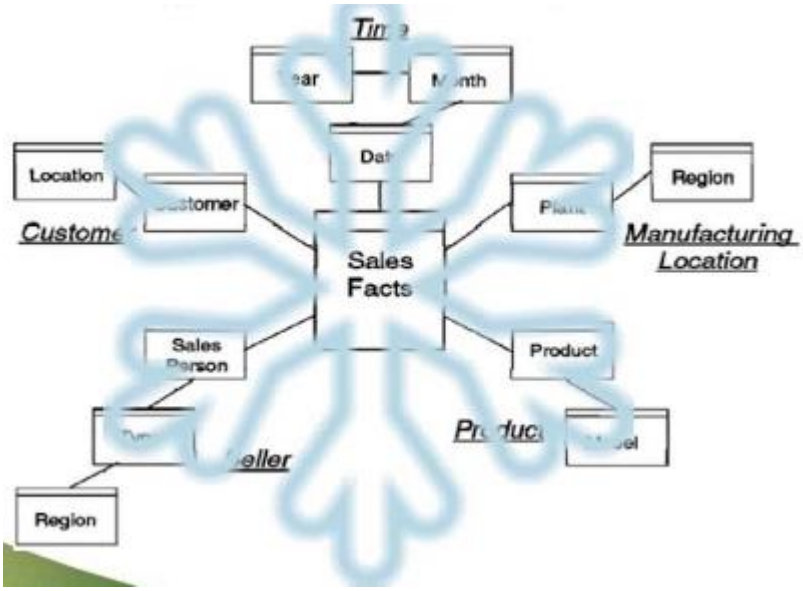


Figura 4 Esquema copo de nieve [12]

Se debe utilizar el modelo estrella siempre que sea posible, salvo cuando exista una tabla de dimensión muy grande que degrade el tiempo de las consultas, en ese caso se empleará el modelo copo de nieve. Adicionalmente, se presenta en la tabla 2 otras diferencias entre los dos modelos que también pueden influir en su elección:

	Estrella	**Copo de Nieve**
Entendimiento del modelo	Sencillo	Mayor dificultad
Número de tablas	Menor	Mayor
Complejidad de la consulta	Baja	Alta
Desempeño de las consultas y procesamiento del cubo	Rápida	Lenta

Tabla 2 Diferencias esquema estrella vs copo de nieve [12]

2.4Data visualization.

La visualización de datos o Data visualization es una forma de comunicación que representa información densa y compleja de manera gráfica. De esta manera, los resultados visuales son diseñados para facilitar la comparación de datos y usarlo para contar una historia, ya que ambos pueden ayudar a los usuarios en su toma de decisiones. [13]

La visualización de datos permite expresar datos de diferentes tipos y tamaños: desde pequeños puntos de datos hasta grandes conjuntos de datos con múltiples variables. Para ello, se basa en tres principios:

I. **Precisión**: priorizar la precisión, la claridad y la integridad de los datos permite presentar la información sin distorsionar que necesita el usuario.
II. **Utilidad**: ayudar a los usuarios a navegar en sus datos ofreciendo un contexto y enfatizando en que la exploración y la comparación de sus datos sea fácil y asequible.
III. **Escalabilidad**: Adaptar la visualización de datos a diferentes tamaños de dispositivos, al mismo tiempo que anticiparse a las posibles necesidades de los usuarios con respecto a la complejidad y modalidad de sus datos.

Existen una gran variedad de gráficos como pueden ser los gráficos de barras, de tarta o histogramas que permiten representar los datos de diversas maneras según las necesidades del cliente. No solo el tipo de gráfico es lo único importante dentro de la visualización de datos, conceptos como el comportamiento y el estilo que se aplica a estos pueden realizar grandes diferencias a la hora de entender mejor la información que se muestra. Incluso a la hora de crear dashboards que muestren dicha información, el diseño y la disposición de los gráficos que lo componen pueden influir en la toma de decisiones de los usuarios.

A raíz de esto, existen diversas técnicas y consejos dentro del ámbito del data visualization como, por ejemplo: el uso de la teoría del color o el uso de leyendas, etiquetas y anotaciones dentro de los gráficos para contextualizar los datos, que orientan en la construcción de las diversas representaciones gráficas que componen un dashboard.

3 Implementación de la solución.

Para resolver los varios problemas que presenta la empresa expuestos en el apartado 2.2, se desarrolla un nuevo módulo de Business Intelligence para la plataforma de la empresa que aporte las soluciones correspondientes a cada uno de estos problemas. Dicho módulo debe ser lo más independiente posible para que su integración posterior con la plataforma sea sencilla.

Se ha decidido implementar un módulo de BI porque los problemas que presenta la plataforma de la empresa son problemas comunes que afrentan las empresas que no aplican dichas soluciones a sus negocios. En este caso concreto son los siguientes:

- Los gestores de los ecosistemas cuentan con datos dentro de los eventos del data lake de su ecosistema, pero esos datos no poseen ningún valor. Esto se debe a que no ofrecen información relevante puesto que no existe una herramienta de análisis que extraiga y presente la información que necesitan estos gestores para operar correctamente.

- Debido a la carencia de información del ecosistema se produce una falta de agilidad por parte de cada gestor para adaptarse a las diferentes necesidades de su ecosistema.

Este módulo sienta las bases para construir una arquitectura de Business Intelligence para la empresa que solucione esos problemas que actualmente padece. También debe ser fácilmente escalable y adaptable, puesto que en el futuro debe poder integrar diferentes fuentes de datos con los que poder trabajar según las necesidades de cada ecosistema.

3.1 Arquitectura BI.

Para comenzar una solución BI para un negocio, lo primero que se debe realizar es un estudio acerca de la estructura del negocio como pueden ser los departamentos que componen a una empresa y sus fuentes de información, para poder plantear la arquitectura BI correspondiente: de dónde se extraen los datos, dónde y cómo se almacenan luego los datos si en un repositorio central de toda la organización (data warehouse) o por departamentos (data mart).

Para implementar esa arquitectura, se decidió utilizar la metodología de Kimball para la construcción de un data warehouse, pues proporciona un enfoque de menor a mayor muy versátil que ayuda a la implementación de un data warehouse. Es acorde a nuestra empresa porque permite implementar pequeños data marts empleando pocos recursos y tiempo. Por tanto, se construye la arquitectura desde abajo hacia arriba, primero se construyen los data marts para luego integrarlos poco a poco en un gran almacén de datos o data warehouse.

A partir de la estructura de la plataforma reflejada en la figura 1, se pueden identificar dos principales data marts: el data mart de Community y el data mart de Core, los cuales

se integrarán para conformar el data warehouse de la empresa como se muestra en la figura 5. Una vez planteada la estructura inicial, se puede establecer que información se encargará de almacenar cada uno de los data marts y el data warehouse.

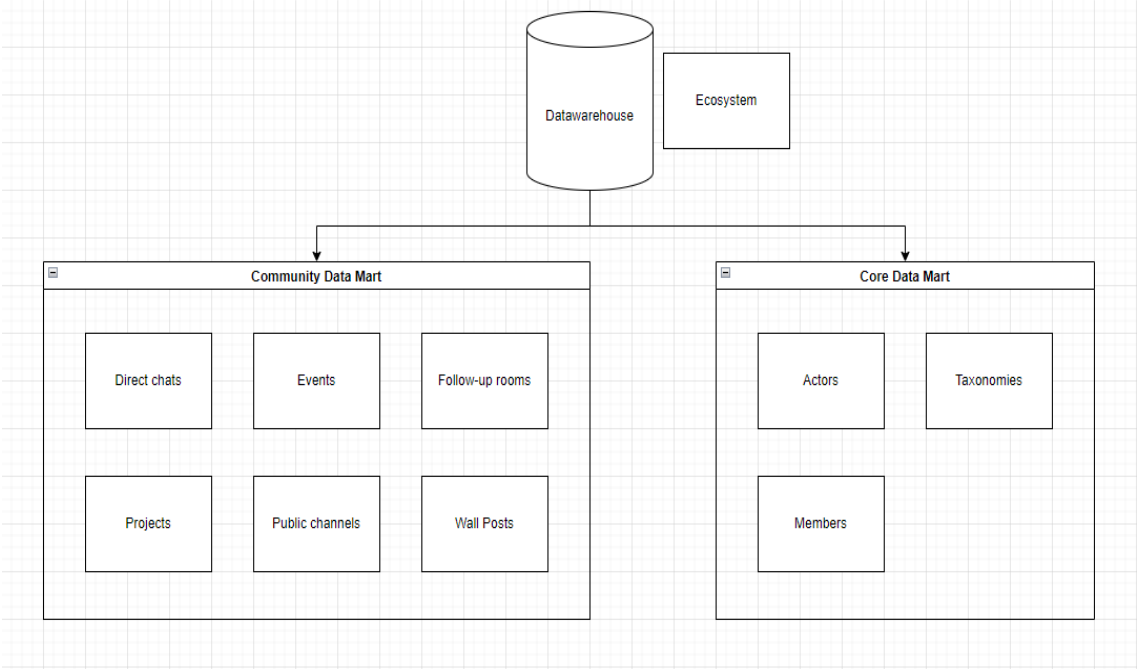


Figura 5 Arquitectura BI Moonshot

3.2 Análisis multidimensional.

Para realizar el análisis multidimensional de los datos, se emplea el esquema estrella expuesto en el apartado 2.3 por diversas razones:

- Es el modelo más común y sencillo de implementar.
- Es el modelo recomendable para tareas de análisis, puesto que el coste de las consultas es bajo y es de fácil lectura.
- Es el modelo que más se ajusta a la estructura de datos que vamos a construir, pues nuestros datos no cuentan con varias subdimensiones, por tanto, el número de tablas es muy reducido.

3.2.1 Tablas de hechos y dimensiones.

Una vez se conocen los KPIs que desea medir el cliente se debe hacer su transcripción al modelo multidimensional comenzando por la tabla de hechos, identificando las métricas que debe medir y las dimensiones asociadas necesarias para calcular esas métricas.

En la figura 6, se muestra un ejemplo de una tabla de hechos que mide el KPI "Actividad de un miembro" especificado por un cliente, situada en el centro y rodeada por las

tablas de dimensión que contienen los datos referentes al miembro que realizó dichas interacciones y la fecha y hora en la que ocurrieron.

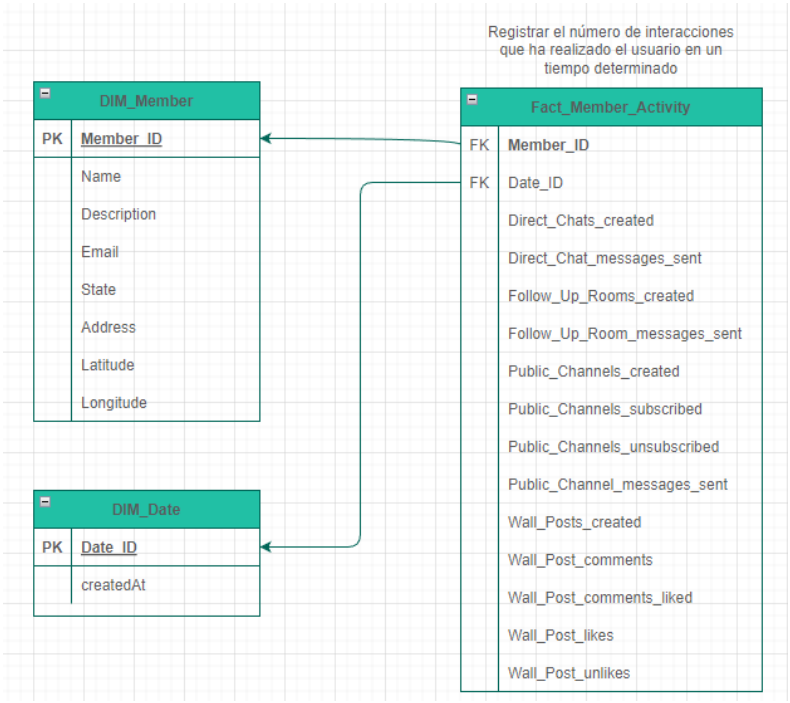


Figura 6 Ejemplo análisis multidimensional 1

Cada una de estas tablas tiene su modelo correspondiente dentro del proyecto, véanse la figura 7 correspondiente al modelo de la tabla de hechos y las figuras 8 y 9 como modelo de las tablas de dimensiones del ejemplo para la ejecución de su proceso ETL.

```java
public class FtMemberActivity {

    private final String memberId;
    private final LocalDateTime createdAt;
    private int directChatsCreated;
    private int directChatMessagesSent;
    private int followUpRoomsCreated;
    private int followUpRoomMessagesSent;
    private int publicChannelsCreated;
    private int publicChannelMessagesSent;
    private int publicChannelsSubscribed;
    private int publicChannelsUnsubscribed;
    private int wallPostsCreated;
    private int wallPostComments;
    private int wallPostCommentsLiked;
    private int wallPostLikes;
    private int wallPostUnlikes;
```

Figura 7 Modelo tabla de hechos que mide la actividad de un miembro en un momento determinado

```java
public class DimMember {

    private String id;
    private String state;
    private String name;
    private String address;
    private double latitude;
    private double longitude;
    private String email;
    private String about;
```

Figura 8 Modelo tabla de dimensión de miembro

```java
public class DimTime {

    private LocalDateTime createdAt;
```

Figura 9 Modelo tabla de dimensión de tiempo

Por último, en la figura 10 se muestra la tabla de hechos y sus valores dentro del data warehouse una vez se han realizado el análisis multidimensional para su estructura y rellenado con datos mediante el proceso ETL.

Member ID	Date ID	Direct Chats Created	Direct Chat Messages Sent	Follow Up Rooms Created	Follow Up Room Messages Sent	Public Channels Created	Public Channel Messages Sent	Public Channels Subscribed	Public Channels Uns
60b53ce23a28bb702667cbe6	12406	2	12	0	2	0	0	0	0
60b545953a28bb702667cc0f	12406	1	1	1	2	0	0	0	0
60b53ce23a28bb702667cbe6	12407	1	5	0	0	0	0	0	0
60b549db3a28bb702667cc2d	12422	1	1	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12570	1	4	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12737	1	1	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12738	5	0	0	1	0	0	0	0
60b53ce23a28bb702667cbe6	12778	2	4	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12780	2	7	0	0	0	0	0	0
60c9dd45cd028255201ad096	12781	2	0	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12906	1	17	0	0	0	0	0	0
60b53ce23a28bb702667cbe6	12922	1	12	0	0	0	0	0	0

Figura 10 Ejemplo tabla de hechos tras proceso ETL

3.3Moonshot Analytics.

Una vez realizados los pasos previos para construir la arquitectura BI y el análisis multidimensional de los datos que conformarán la estructura de las tablas del data warehouse hay que pasar al desarrollo del módulo que implementará la solución.

Lo primero que hay que destacar del módulo desarrollado es que utiliza Docker como plataforma de contenerización de aplicaciones para poder descargar las imágenes de PostgreSQL y Metabase para su uso. Por tanto, para poder hacer uso de estas aplicaciones se necesita de un archivo de configuración docker-compose.yaml (véase la figura 11) que permita descargar, crear y configurar los ajustes de las imágenes de ambos programas en nuestra máquina. Aparte de este archivo de configuración inicial, también se utiliza un script de SQL llamado "init-database.sql" que se ejecuta cuando se lanza el contenedor de Docker de la máquina de PostgreSQL y permite inicializar la base de datos con algunos valores previos como tipos de datos enumerados o la tabla de dimensión del tiempo, como se refleja en la figura 12.

```yaml
version: '3.1'

services:

  postgres:
    image: postgres:latest
    restart: always
    container_name: PostgreSQL
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres
    ports:
      - 5432:5432
    volumes:
      - C:\tmp\moonshot\postgres\data:/data/db
      - ./init-database.sql:/docker-entrypoint-initdb.d/init-database.sql

  metabase:
    image: metabase/metabase:latest
    container_name: "Metabase"
    restart: always
    environment:
      JAVA_TIMEZONE: "Europe/Madrid"
      MB_DB_TYPE: postgres
      MB_DB_DBNAME: postgres
      MB_DB_PORT: 5432
      MB_DB_USER: postgres
      MB_DB_PASS: postgres
      MB_DB_HOST: postgres
      MB_DB_FILE: /metabase-data/metabase.db
    ports:
      - 3000:3000
    volumes:
      - C:\tmp\moonshot\metabase\data:/metabase-data
    depends_on:
      - postgres
```

Figura 11 Archivo de configuración docker-compose.yaml

```
SET DATESTYLE TO 'European';

CREATE TYPE STATE AS ENUM ('ACCEPTED', 'PENDING', 'REJECTED');

CREATE TYPE ROOM_STATE AS ENUM ('OPEN', 'CLOSED', 'CREATED', 'DELETED');

CREATE TYPE ACTION AS ENUM ('CREATED', 'DELETED');

CREATE TYPE ACTOR_TYPE AS ENUM (
    'Startup',
    'Company',
    'Expert',
    'Person investor',
    'Organization investor',
    'Mentor',
    'Public entity',
    'Research group',
    'Talent',
    'University',
    'Ngo',
    'Hub',
    'Cluster');

CREATE TYPE STARTUP_STAGE AS ENUM(
    'None',
    'Got deck',
    'Business plan',
    'First prototype',
    'Initial interest',
    'Got beta',
    'Virality scalability',
    'Fixed beta',
    'Started invoice',
    'Running business');

CREATE TYPE TRL_STAGE AS ENUM(
    'None',
    'Basic principles observed',
    'Technology concept formulated',
    'Experimental proof of concept',
    'Technology validated in lab',
    'Technology validated in relevant environment',
    'Technology demonstrated in relevant environment',
    'System prototype demonstration in operational environment',
    'System complete and qualified',
    'Actual system proven in operational environment');

CREATE TYPE PROTECTION_METHOD AS ENUM (
    'None',
    'Patents',
    'Trade secrets',
    'Computer algorythms',
    'Design',
    'Database',
    'Trademark',
    'Copyright',
    'Industrial design');

CREATE TYPE EVENT_TYPE AS ENUM (
    'Not defined',
    'Face to Face',
    'Online',
    'Both');

CREATE TABLE DIM_DATE (
    Date_ID serial,
    CreatedAt TIMESTAMP NOT NULL,
    PRIMARY KEY (Date_ID));
```

Figura 12 Fichero de inicialización de PostgreSQL

3.3.1 Proceso ETL.

Una vez estén en funcionamiento los contenedores de Docker, el módulo ya puede realizar sus funciones con normalidad, y es por ello por lo que avanzamos al siguiente paso: el proceso ETL.

Este módulo realiza un procesamiento en lote para ejecutar cada uno de los procesos ETL puesto que la mayoría de los datos se relacionan con los actores y miembros de un ecosistema y, por tanto, es necesario cargar primero dentro de nuestra base de datos relacional aquellas tablas cuya clave primaria sea clave ajena en otra tabla para evitar futuras excepciones. Es por ello, por lo que primero se deben de cargar las tablas de dimensiones (contienen claves primarias) y luego las tablas de hechos (contienen claves ajenas).

Para realizar cada uno de los pasos del proceso ETL (extracción, transformación y carga) de los datos del data lake de cada ecosistema a su data mart, se crean varios servicios que se encargan de manejar cada una de estas fases para cada entidad (figura 13):

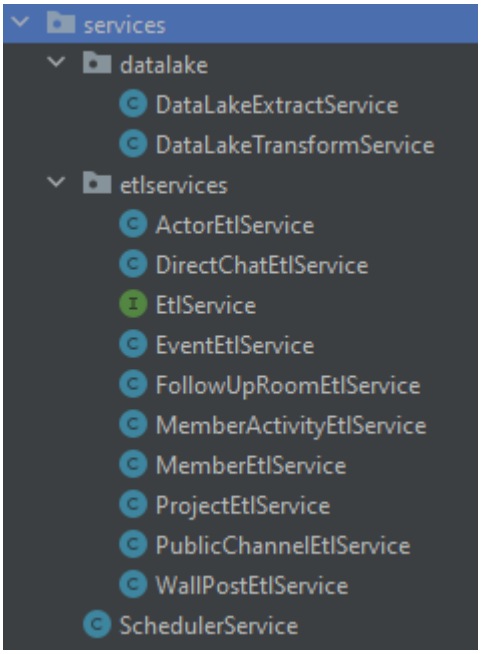
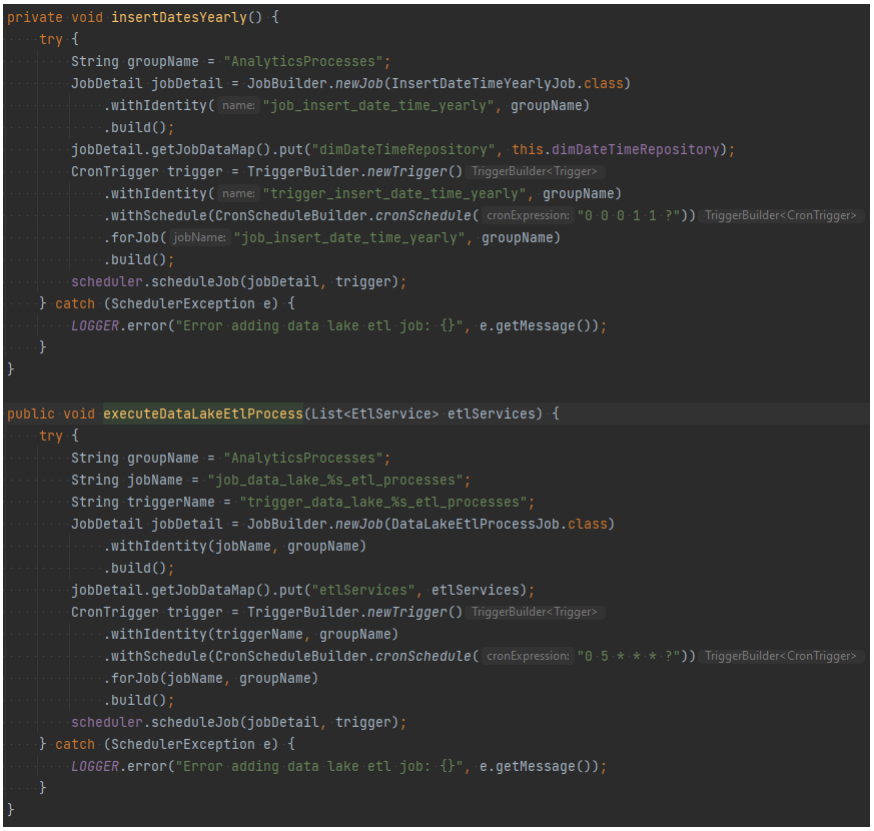


Figura 13 Servicios ETL implementados

- **SchedulerService**: se encarga de ejecutar dos tareas periódicas cuya configuración se muestra en la figura 14:
  - La primera consiste en insertar a principio de cada año las fechas y horas de principio a fin de este en la tabla de la dimensión del tiempo, cuyo código se muestra en la figura 49.
  - La segunda tarea consiste en avisar cada hora a las 05 minutos a los etlservices para que ejecuten sus procesos ETL, cuyo código se muestra en la figura 48.



```
private void insertDatesYearly() {
    try {
        String groupName = "AnalyticsProcesses";
        JobDetail jobDetail = JobBuilder.newJob(InsertDateTimeYearlyJob.class)
            .withIdentity( name: "job_insert_date_time_yearly", groupName)
            .build();
        jobDetail.getJobDataMap().put("dimDateTimeRepository", this.dimDateTimeRepository);
        CronTrigger trigger = TriggerBuilder.newTrigger() TriggerBuilder<Trigger>
            .withIdentity( name: "trigger_insert_date_time_yearly", groupName)
            .withSchedule(CronScheduleBuilder.cronSchedule( cronExpression: "0 0 0 1 1 ?")) TriggerBuilder<CronTrigger>
            .forJob( jobName: "job_insert_date_time_yearly", groupName)
            .build();
        scheduler.scheduleJob(jobDetail, trigger);
    } catch (SchedulerException e) {
        LOGGER.error("Error adding data lake etl job: {}", e.getMessage());
    }
}

public void executeDataLakeEtlProcess(List<EtlService> etlServices) {
    try {
        String groupName = "AnalyticsProcesses";
        String jobName = "job_data_lake_%s_etl_processes";
        String triggerName = "trigger_data_lake_%s_etl_processes";
        JobDetail jobDetail = JobBuilder.newJob(DataLakeEtlProcessJob.class)
            .withIdentity(jobName, groupName)
            .build();
        jobDetail.getJobDataMap().put("etlServices", etlServices);
        CronTrigger trigger = TriggerBuilder.newTrigger() TriggerBuilder<Trigger>
            .withIdentity(triggerName, groupName)
            .withSchedule(CronScheduleBuilder.cronSchedule( cronExpression: "0 5 * * * ?")) TriggerBuilder<CronTrigger>
            .forJob(jobName, groupName)
            .build();
        scheduler.scheduleJob(jobDetail, trigger);
    } catch (SchedulerException e) {
        LOGGER.error("Error adding data lake etl job: {}", e.getMessage());
    }
}
```

Figura 14 Configuración tareas SchedulerService

- **Etlservice**: es el servicio que se encarga de la realización de todo el proceso ETL en lote por entidad, primero utiliza el servicio de extracción para obtener los eventos que desea analizar, luego utiliza el servicio de transformación para convertir los eventos extraídos en el nuevo modelo de datos que maneja y, por último, utiliza el repositorio correspondiente para realizar la fase de carga de esos nuevos datos dentro de su data mart. A modo de ejemplo, se muestra la estructura de uno de estos servicios etl en la figura 15.

  Cada uno se ejecuta cuando se inicia la aplicación para reflotar los datos dentro del data lake por si ha habido posibles cambios desde la última vez que se ejecutó la aplicación principal de la plataforma para así mantener los datos constantemente actualizados, y a partir de ahí se ejecuta cada vez que se dispara la alarma del SchedulerService para realizar un nuevo proceso ETL de los eventos creados en la última hora.

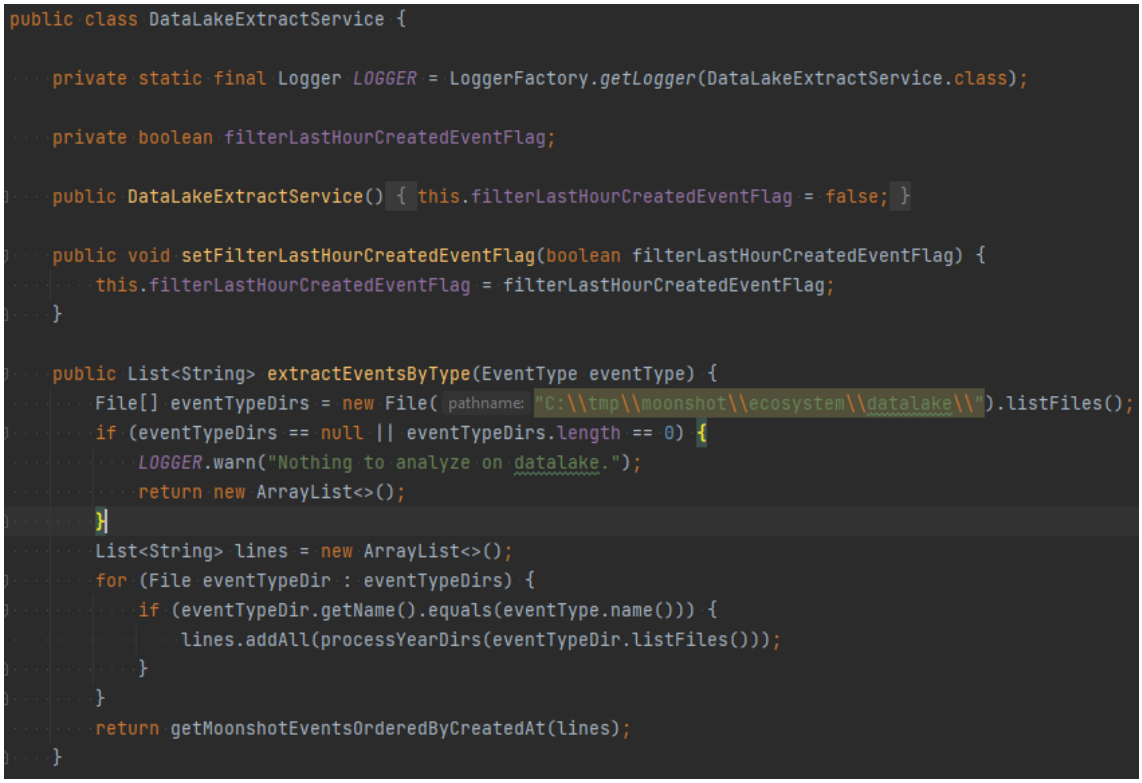
```java
public EventEtlService(
    DataLakeExtractService dataLakeExtractService,
    DataLakeTransformService dataLakeTransformService,
    DimEventRepository dimEventRepository,
    FtEventRepository ftEventRepository
) {
    this.dataLakeExtractService = dataLakeExtractService;
    this.dataLakeTransformService = dataLakeTransformService;
    this.dimEventRepository = dimEventRepository;
    this.ftEventRepository = ftEventRepository;
    executeEtlProcess();
}

@Override
public void executeEtlProcess() {
    LOGGER.info("ETL event events process starting...");
    this.etlEventCreateEvents();
    this.etlEventUpdateEvents();
    this.etlEventDeleteEvents();
    LOGGER.info("ETL event events process finished.");
}

private void etlEventCreateEvents() {
    List<String> events = this.dataLakeExtractService.extractEventsByType(EVENT_CREATE_EVENT);
    List<DimEvent> eventsCreated = new ArrayList<>();
    List<FtEvent> ftEventsCreated = new ArrayList<>();
    for (String event : events) {
        DimEvent eventCreated = this.dataLakeTransformService.getEventCreated(event);
        if (eventCreated != null) {
            DimTime createdAt = this.dataLakeTransformService.getDateTimeCreatedAt(event);
            eventsCreated.add(eventCreated);
            ftEventsCreated.add(createFactEvent(eventCreated.getId(), createdAt, Action.CREATED));
        }
    }
    this.dimEventRepository.insertMany(eventsCreated);
    this.ftEventRepository.insertMany(ftEventsCreated);
}
```

Figura 15 Estructura servicio ETL para eventos

- **DataLakeExtractService (Extracción)**: se encarga de buscar en el data lake del ecosistema todos los eventos cuyo tipo se pasa por parámetro, filtrando o no los eventos según un flag interno que en su primera ejecución busca todos los eventos en el data lake (inactivo), después solo busca aquellos que han sido creados en la última hora (activo) y, por último, devolver los eventos encontrados ordenados por fecha de creación del evento para su posterior procesamiento como se resume en la figura 16.



```
public class DataLakeExtractService {

    private static final Logger LOGGER = LoggerFactory.getLogger(DataLakeExtractService.class);

    private boolean filterLastHourCreatedEventFlag;

    public DataLakeExtractService() { this.filterLastHourCreatedEventFlag = false; }

    public void setFilterLastHourCreatedEventFlag(boolean filterLastHourCreatedEventFlag) {
        this.filterLastHourCreatedEventFlag = filterLastHourCreatedEventFlag;
    }

    public List<String> extractEventsByType(EventType eventType) {
        File[] eventTypeDirs = new File( pathname: "C:\\tmp\\moonshot\\ecosystem\\datalake\\").listFiles();
        if (eventTypeDirs == null || eventTypeDirs.length == 0) {
            LOGGER.warn("Nothing to analyze on datalake.");
            return new ArrayList<>();
        }
        List<String> lines = new ArrayList<>();
        for (File eventTypeDir : eventTypeDirs) {
            if (eventTypeDir.getName().equals(eventType.name())) {
                lines.addAll(processYearDirs(eventTypeDir.listFiles()));
            }
        }
        return getMoonshotEventsOrderedByCreatedAt(lines);
    }
```

Figura 16 DatLakeExtractService (extraer por tipo de evento)

- **DataLakeTransformService (Transformación)**: se encarga de transformar los datos de los eventos extraídos en la fase previa a un nuevo modelo de datos como el expuesto en la figura 15 utilizando varias técnicas de transformación expuestas en el apartado 2.3.2. Como muestra la figura 16, en este paso también se establecen unas series de restricciones que deben cumplir los datos extraídos para que puedan ser considerados datos válidos.

```java
public DimEvent getEventCreated(String event) {
    try {
        EventCreateEvent eventCreateEvent = fromJson(event, EventCreateEvent.class);
        if (isBlank(eventCreateEvent.getOwnerMemberId())) {
            LOGGER.info("Invalid owner member id while parsing event create event: {}", eventCreateEvent);
            return null;
        }
        DimEvent eventCreated = transformAndCheckEventConstraints(eventCreateEvent.toEvent());
        if (eventCreated != null) {
            eventCreated.setOwnerMemberId(eventCreateEvent.getOwnerMemberId());
            LOGGER.info("Event created: {}", event);
            return eventCreated;
        }
    } catch (MoonshotException e) {
        LOGGER.error("Error while getting event created from event");
    }
    return null;
}

public DimEvent getEventUpdated(String event) {
    try {
        LOGGER.info("Event updated: {}", event);
        EventUpdateEvent eventUpdateEvent = fromJson(event, EventUpdateEvent.class);
        return transformAndCheckEventConstraints(eventUpdateEvent.toEvent());
    } catch (MoonshotException e) {
        LOGGER.error("Error while getting event updated from event.");
    }
    return null;
}
```

Figura 17 Ejemplo 1 DatalakeTransformService (Transformación)

```java
private DimEvent transformAndCheckEventConstraints(Event event) {
    if (isBlank(event.getId())) {
        LOGGER.info("Invalid id while parsing event: {}", event);
        return null;
    }
    DimEvent dimEvent = new DimEvent();
    dimEvent.setId(event.getId());
    dimEvent.setEventDateTime(LocalDateTimeUtils.truncateToHours(event.getDate()));
    dimEvent.setTitle(event.getTitle());
    dimEvent.setDescription(event.getDescription());
    Address address = transformAndCheckAddressConstraints(event.getAddress());
    dimEvent.setAddress(address.getFormatted());
    dimEvent.setLatitude(address.getLat());
    dimEvent.setLongitude(address.getLng());
    if (!isBlank(dimEvent.getAddress()) && !event.isOnline()) {
        dimEvent.setEventType("Face to Face");
    } else if (isBlank(dimEvent.getAddress()) && event.isOnline()) {
        dimEvent.setEventType("Online");
    } else if (!isBlank(dimEvent.getAddress()) && event.isOnline()) {
        dimEvent.setEventType("Both");
    } else {
        dimEvent.setEventType("Not defined");
    }
    return dimEvent;
}
```

Figura 18 Ejemplo 2 DatalakeTransformService (Transformación)

- **Repositorio de la entidad (Carga):** utilizando el patrón Repository, el repositorio de la entidad del etlService se encarga de generar un batch de la operación correspondiente de insertMany, updateMany o deleteMany de los datos obtenidos del paso anterior. Se ejecuta una sola consulta de este modo para minimizar el impacto de rendimiento en la base destino (en este caso PostgreSQL) en lugar de muchas consultas SQL pequeñas como se detalla en la figura 19.

```java
public class DimEventRepository {

    private static final Logger LOGGER = LoggerFactory.getLogger(DimEventRepository.class);

    private final PostgresConfig postgresConfig;

    public DimEventRepository(PostgresConfig postgresConfig) { this.postgresConfig = postgresConfig; }

    public void insertMany(List<DimEvent> eventsCreated) {
        try {
            Connection connection = this.postgresConfig.getConnection();
            PreparedStatement insertEventQuery = connection.prepareStatement(
                sql: "INSERT INTO dim_events(Event_id, Owner_Member_id, Event_date_id, Event_Type, Title, " +
                    "Description, Address, Latitude, Longitude) " +
                    "VALUES (?, ?, (SELECT date_id from dim_date WHERE createdat = ?), CAST(? as event_type), " +
                    "?, ?, ?, ?, ?);");
            eventsCreated.forEach(event -> {
                try {
                    insertEventQuery.setString( parameterIndex: 1, event.getId());
                    insertEventQuery.setString( parameterIndex: 2, event.getOwnerMemberId());
                    insertEventQuery.setObject( parameterIndex: 3, event.getEventDateTime());
                    insertEventQuery.setString( parameterIndex: 4, event.getEventType());
                    insertEventQuery.setString( parameterIndex: 5, event.getTitle());
                    insertEventQuery.setString( parameterIndex: 6, event.getDescription());
                    insertEventQuery.setString( parameterIndex: 7, event.getAddress());
                    insertEventQuery.setDouble( parameterIndex: 8, event.getLatitude());
                    insertEventQuery.setDouble( parameterIndex: 9, event.getLongitude());
                    insertEventQuery.addBatch();
                } catch (SQLException e) {
                    LOGGER.error("Error processing query to insert new event: ", e);
                }
            });
            LOGGER.info("Executing event insert batch updates:");
            this.postgresConfig.executePreparedStatementBatchUpdate(insertEventQuery);
        } catch (SQLException e) {
            LOGGER.error("Error creating query to insert new event: ", e);
        }
    }
```

Figura 19 DimEvent repository (carga de datos)

3.3.2 Arquitectura del proyecto.

En lo que respecta a la arquitectura software del proyecto, se muestra una versión simplificada de la misma en la figura 20, puesto que todos los servicios ETL expuestos anteriormente en la figura 13 presentan una estructura casi similar, solo diferenciándose en los tipos de eventos que analizan, para los cuales poseen un método concreto por tipo de evento, y los repositorios que utilizan para cargar los datos dentro de sus data marts

correspondientes, pero el flujo de trabajo es el mismo para todos tal cual se ha descrito en el anterior apartado con más lujo de detalles.

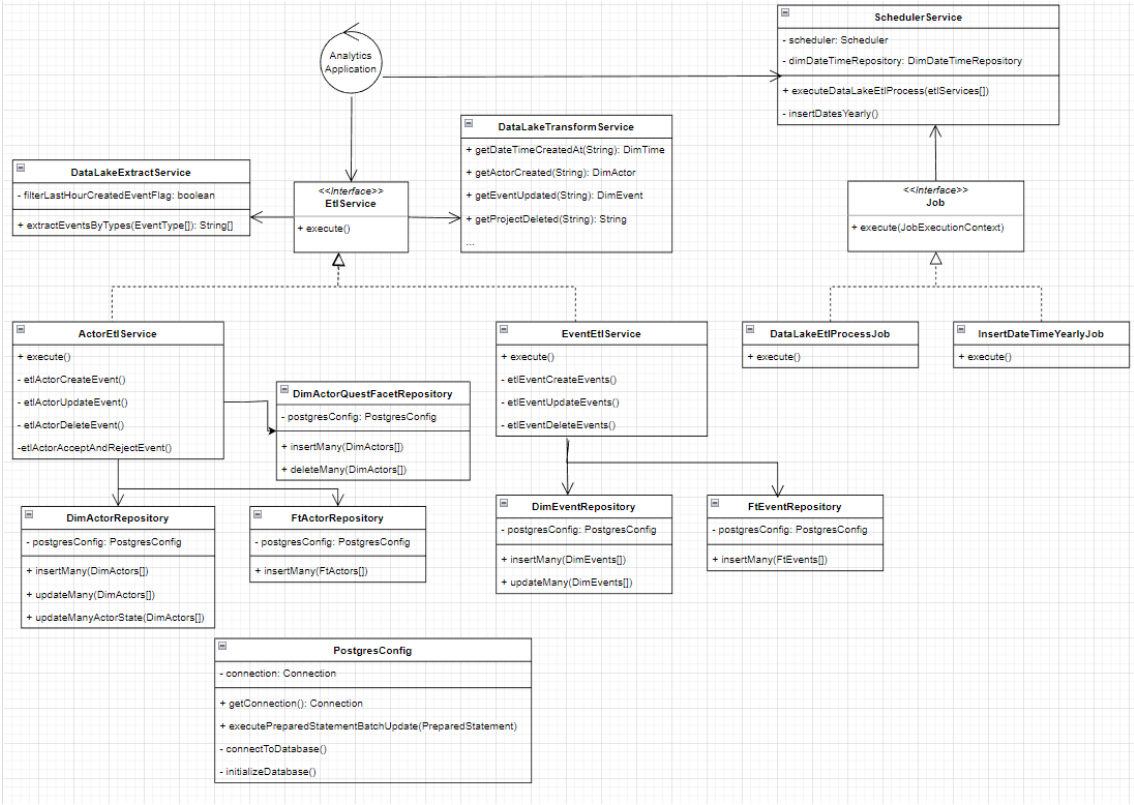


Figura 20 Arquitectura software final Moonshot Analytics (resumida)

Mediante el uso del patrón Repository cada entidad cuenta con un modelo de datos que se corresponde con la tabla alojada en la base de datos y posee un único repositorio especializado que interactúa solo con esa tabla. Además, nos permite crear una interfaz entre el módulo y el tipo de base de datos que se usa, por lo que si en el futuro la empresa necesitase cambiar la tecnología que utiliza para su data warehouse solo habría que sustituir en los repositorios la configuración actual de PostgreSQL por la nueva tecnología de bases de datos elegida.

3.4Data visualization.

Después de la elaboración de una arquitectura de BI con sus correspondientes data marts y data warehouse; del modelado de los datos mediante las tablas de dimensiones y de hechos; y del desarrollo del proceso ETL de los datos de la plataforma mediante el desarrollo de un nuevo módulo; ya solo queda mostrar el resultado final del proyecto que percibe el cliente siendo el más importante por ser la solución visible a sus problemas, el apartado gráfico de los datos.

Para mostrar estos datos, se utiliza Metabase como software de BI para generar los dashboards y las gráficas que se mostrarán a los gestores de los ecosistemas y que miden cada uno de los KPIs especificados por ellos.

Existen un total de 10 dashboards, los cuales están compuestos de varias gráficas que se enumeran a continuación:

- **Ecosystem dashboard (Figura 21):** presenta el estado actual de un ecosistema a su propietario.
    1. Nº de actores en el ecosistema.
    2. Nº de miembros en el ecosistema.
    3. Nº de actores pendientes de aceptar.
    4. Nº de posts en el ecosistema.
    5. Nº de eventos programados en el ecosistema.
    6. Nº de proyectos en el ecosistema.
    7. Distribución de actores en el ecosistema por tipo de actor.
    8. Nº de salas por tipo de salas (DC, FUR & PC).

Figura 21 Ecosystem dashboard

- **Actors Dashboard (figura 22):** muestra los gráficos que responden a los KPIs relacionados con los actores de un ecosistema.
    1. Nº de actores creados en la última hora.
    2. Evolución del nº de actores en el ecosistema
    3. 10 actores más activos en el ecosistema según sus interacciones con la comunidad mediante publicaciones en el muro de la plataforma o sus interacciones en los distintos tipos de chats.

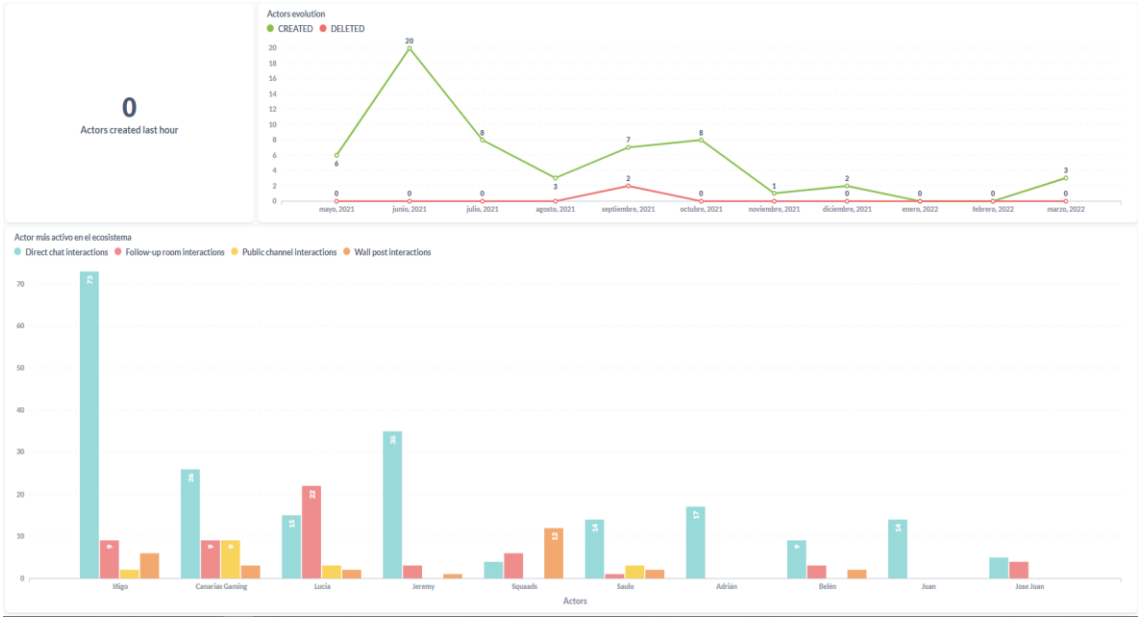


Figura 22 Actors dashboard

- **Direct chat dashboard (figura 23):** muestra los gráficos que responden a los KPIs relacionados con los chats de un ecosistema.
    1. Nº de direct chats creados en la última hora.
    2. Evolución de los direct chats.
    3. Top 10 actores más activos en los direct chats (creación de chats y envío de mensajes).

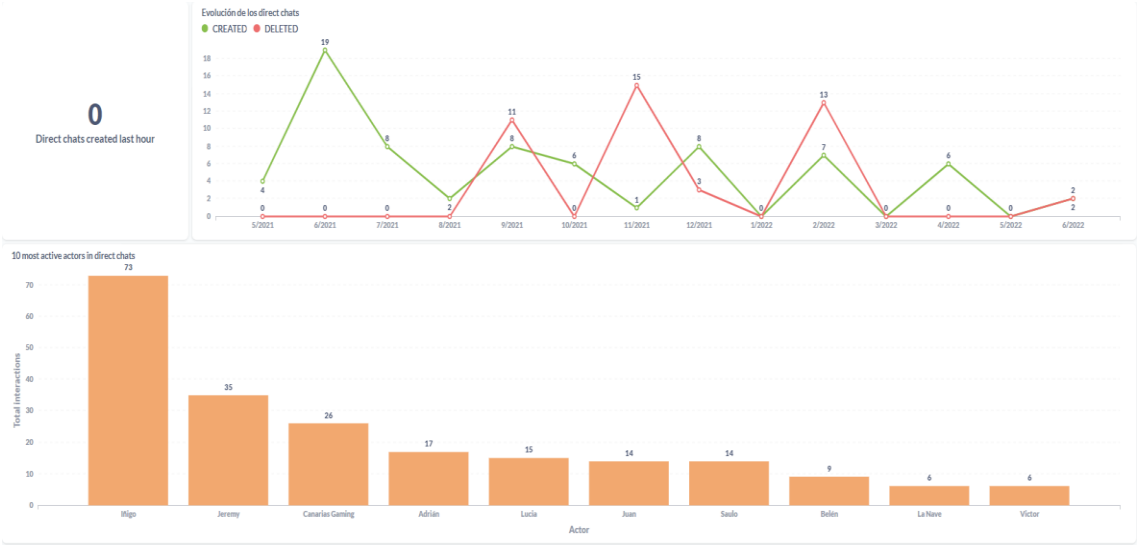


Figura 23 Direct chats dashboard

- **Events dashboard (figura 24):** muestra los gráficos que responden a los KPIs relacionados con los eventos organizados de un ecosistema. Cada evento puede ser de tipo presencial, online, ambos o sin definir.

1. Nº de eventos creados en la última hora.
2. Eventos en el ecosistema por tipo.
3. Eventos programados por tipo.
4. Evolución de eventos.
5. Eventos que ocurren en el mes actual por tipo.

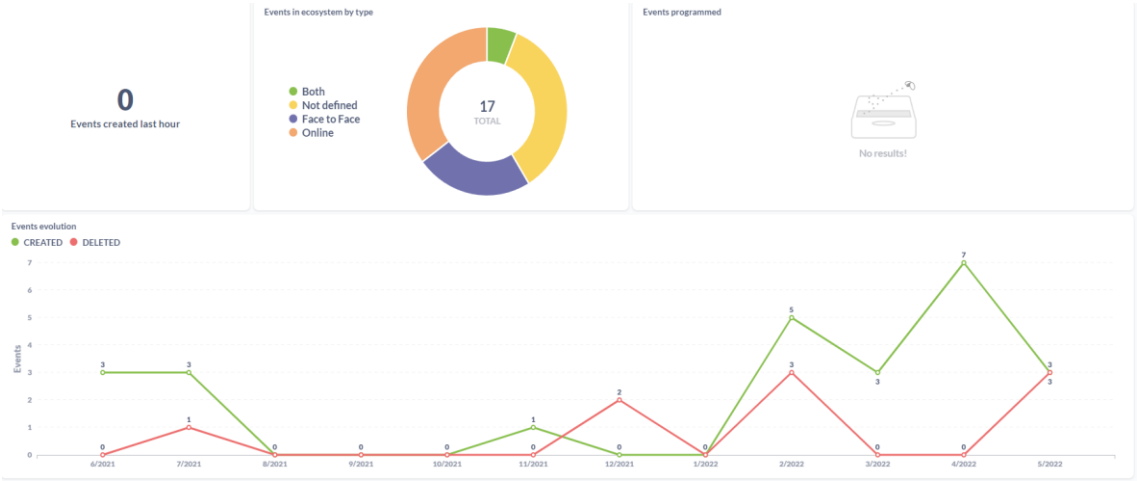


Figura 24 Events dashboard

- **Follow-up rooms dashboard:** (figuras 25 y 26) muestran los gráficos que responden a los KPIs relacionados con las salas de trabajo de un ecosistema.
    1. Nº de follow-up rooms creados en la última hora.
    2. Nº de follow-up rooms abiertas y cerradas.
    3. Evolución de las follow-up rooms.
    4. Media de participantes nuevos por follow-up room.
    5. Nº de follow-up rooms con actividad semanal.

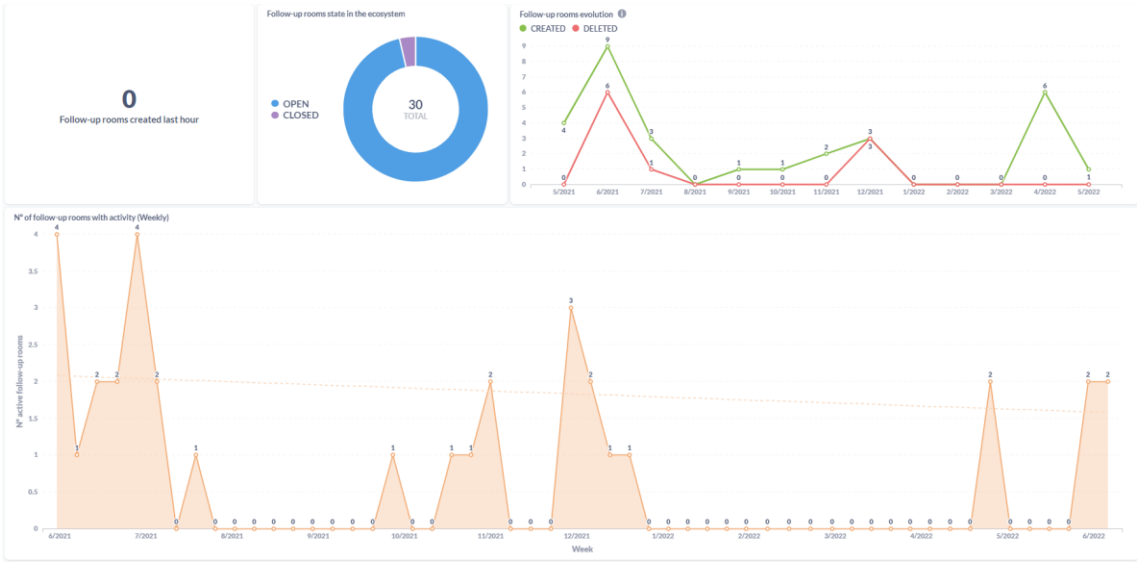


Figura 25 Follow-up room dashboard (Parte 1)

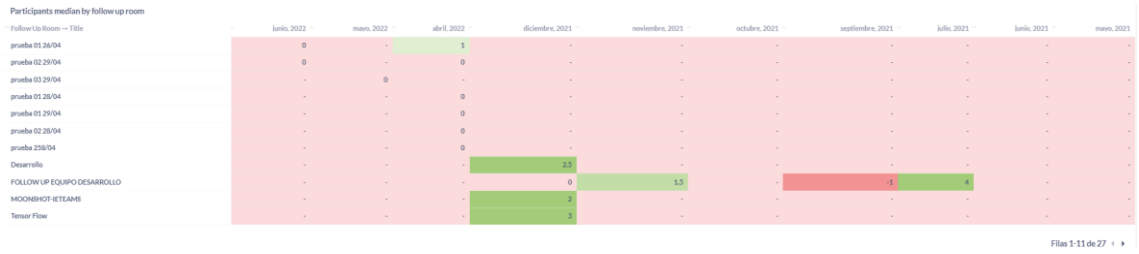


Figura 26 Follow-up room dashboard (Parte 2)

- **Member dashboard:** (figura 27) muestra los gráficos que responden a los KPIs relacionados con los miembros de un ecosistema.
    1. Nº de miembros creados en la última hora.
    2. Evolución del nº de miembros.
    3. Top 10 miembros más activos según sus interacciones en las salas.

Figura 27 Members Dashboard

- **Taxonomies dashboard:** (figuras 28 y 29) muestra los gráficos que responden a los KPIs relacionados con las taxonomías que ofertan y demandan los actores de un ecosistema. Existen 4 tipos de taxonomías:
    1. 60 tecnologías más ofertadas por actores.
    2. 60 tecnologías más demandadas por actores.
    3. Distribución de business models.
    4. Indicador top 15 industrias relacionadas con actores.
    5. Indicador top 15 social innovations relacionadas con actores.

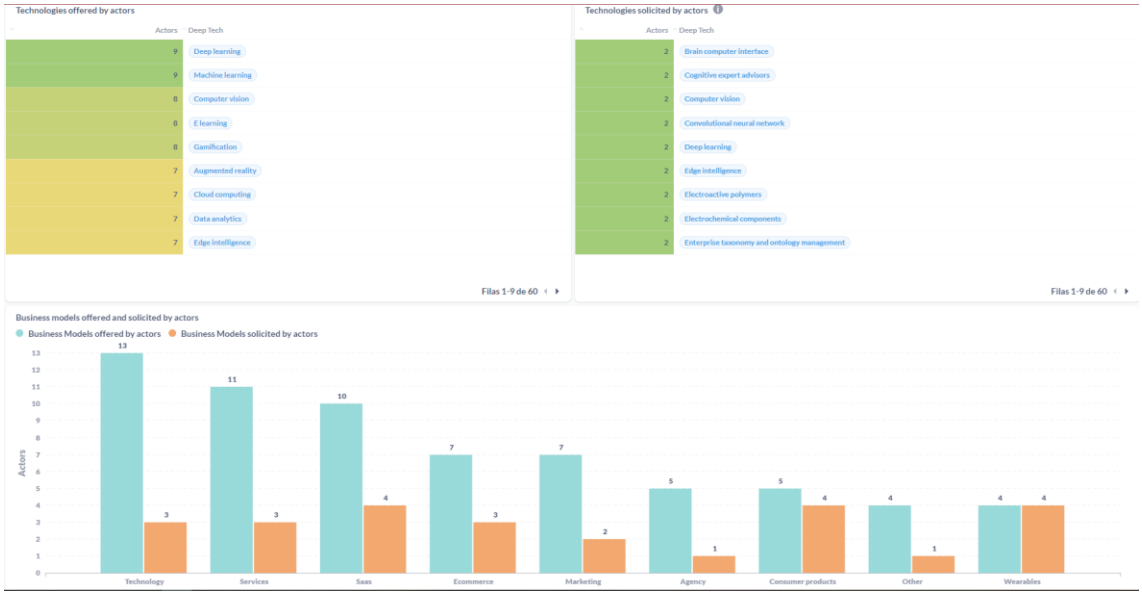


Figura 28 Taxonomies Dashboard (Technologies & Business Models)

Figura 29 Taxonomies Dashboard (Industries & Social Innovations)

- **Projects dashboard:** (figuras 30 y 31) muestran los gráficos que responden a los KPIs relacionados con los proyectos de un ecosistema, así como las taxonomías que demandan.
    1. Nº de proyectos creados en la última hora.
    2. Nº de proyectos creados y borrados.
    3. Evolución de los proyectos.
    4. Proyectos creados por actores.
    5. Indicador de tecnologías por proyectos.
    6. Indicador de industrias por proyectos.
    7. Indicador de business models por proyectos.

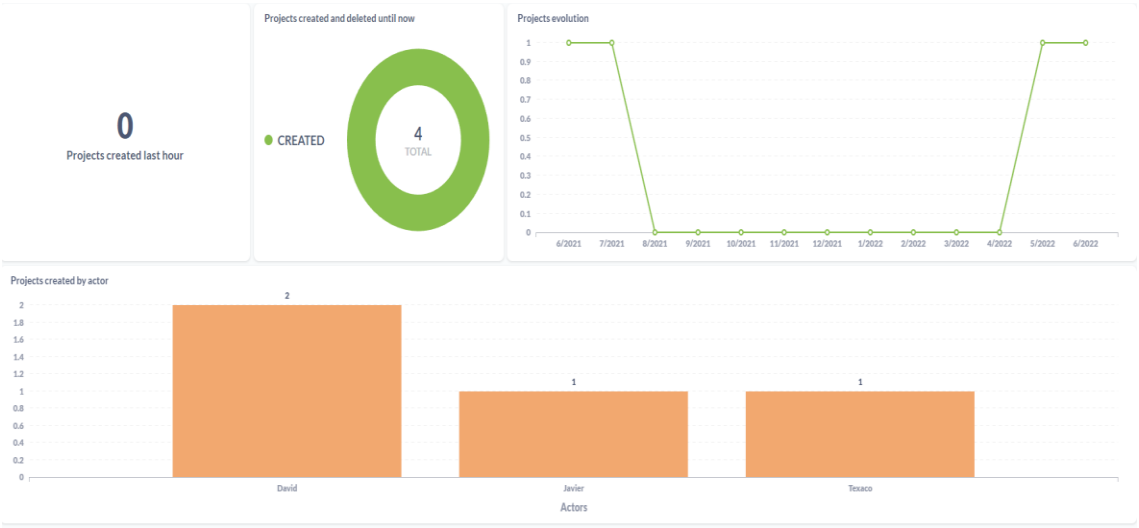


Figura 30 Project dashboard (Parte 1)

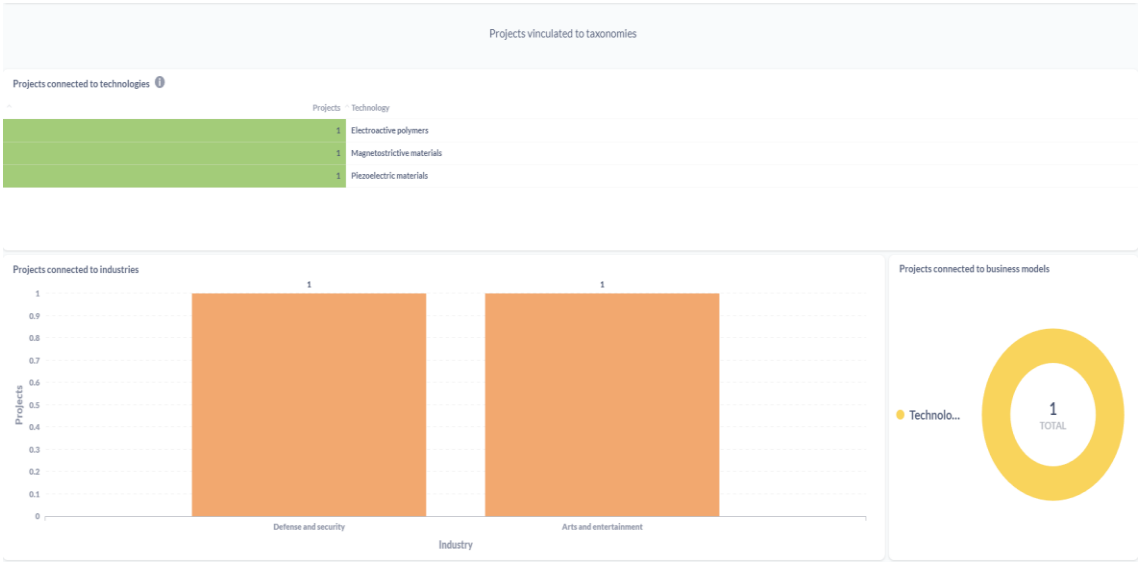


Figura 31 Project dashboard (Parte 2)

- **Public channel dashboard:** (figura 32) muestra los gráficos que responden a los KPIs relacionados con los canales públicos de un ecosistema.
    1. Nº de canales públicos creados en la última hora.
    2. Evolución de los canales públicos.
    3. Top 10 canales públicos más activos del ecosistema.

Figura 32Public channel dashboard

- **Wall post dashboard:** (figuras 33 y 34) muestran los gráficos que responden a los KPIs relacionados con las publicaciones en el muro de la comunidad de un ecosistema.
    1. Nº de wall posts creados en la última hora.
    2. Evolución de los wall posts.
    3. Top 10 wall posts con más likes.
    4. Top 10 wall posts con más comentarios.
    5. Nº de wall posts creados y borrados entre los miembros más activos.
    6. Top 10 miembros más activos en los wall posts (likes, dislikes, comentarios, likes a comentarios).
    7. Top 10 wall posts con más actividad (likes, dislikes, comentarios).

Figura 33 Wall post dashboard (Parte 1)

Figura 34Wall post dashboard (Parte 2)

4 Proceso de desarrollo.

4.1Metodología.

Metodologías ágiles.

Con el fin de alcanzar los objetivos expuestos en el apartado 1.1, se propone utilizar metodologías ágiles de desarrollo en lugar de metodologías clásicas con el fin de evitar las desventajas en el proceso de desarrollo que presentan estas últimas. Por tanto, al utilizar metodologías ágiles seguimos cuatro principios definidos en el *Manifiesto Ágil* como se explica en [14]:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Desarrollar software que funcione en vez de conseguir una buena documentación.

III. La colaboración con el cliente más que la negociación de un contrato.

IV. Responder a los cambios para obtener una ventaja competitiva más que seguir estrictamente un plan.

Desarrollo iterativo e incremental.

Para el desarrollo de este proyecto se ha empleado un modelo de desarrollo ágil conocido como "Desarrollo iterativo e incremental". Este modelo permite desarrollar un producto software siguiendo un proceso de desarrollo cíclico en el que periódicamente, en cada iteración, se va liberando una nueva versión del producto mediante prototipos, aumentando el número de funcionalidades (incremental) y presentando una mejoría en calidad con respecto a la anterior (iterativo).

Según [14], cada una de estas iteraciones se divide en varias fases (obsérvese en la figura 35) compuesta por una fase de planificación, en la que se planifica la actual iteración en base a los requisitos de los clientes; una fase de análisis y diseño de la solución para luego realizar su posterior implementación; la cual una vez terminada se desplegará la versión del producto con las nuevas funcionalidades y se entregará a los clientes; para luego terminar con las fases de test y evaluación de la solución entregada y volver a comenzar el ciclo con la siguiente iteración.

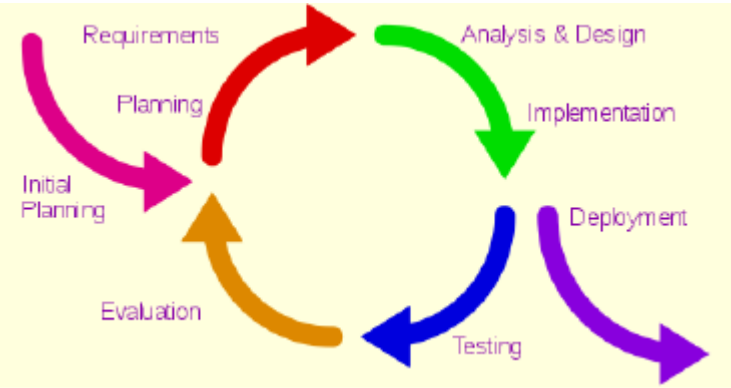


Figura 35 Proceso iterativo e incremental

Razones principales para su elección.

1. Es adecuado para proyectos de pequeño tamaño como es en este caso el desarrollo de un nuevo módulo para la plataforma modular de la empresa.

2. Está basado en el enfoque de las metodologías ágiles lo cual permite que el desarrollo del software sea incremental, sencillo y adaptable ante posibles cambios en las preferencias del cliente.

3. Al final de cada iteración se entrega una versión más estable, de mayor calidad y con nuevas funcionalidades del producto software que puede ser utilizada y evaluada por los clientes.

Participantes.

- **Desarrollador**: es el encargado de analizar, diseñar y desarrollar el producto software desde el inicio hasta el final de cada iteración. Representado por el autor del proyecto: Borja Álvarez Medina.

- **Product Owner**: representante de los clientes y stakeholders cuya función es obtener los KPIs generales y específicos de un ecosistema que desean medir los clientes para comunicarlos al equipo de desarrollo y aportar retroalimentación del producto desarrollado en cada iteración. Representado por David Suriol.

- **Tutor académico y de empresa**: tutores del proyecto encargados de guiar y de supervisar el aspecto técnico del producto software desarrollado en cada iteración. Representado por José Juan como tutor académico e Iñigo Aramburu como tutor de empresa.

- **Clientes**: usuarios finales de la aplicación de la empresa entrevistados por el Product Owner para obtener los KPIs que desean medir de su ecosistema y que luego confeccionarán los dashboards elaborados.

4.2 Tecnologías.

A la hora de desarrollar el proyecto ha sido necesario utilizar distintos tipos de herramientas y lenguajes de programación, los cuales muchos de ellos han sido elegidos porque la plataforma de la empresa está construida con esa tecnología y utiliza los lenguajes de programación que se explicarán a continuación:

Lenguajes de programación:

- **Backend**: se encuentra desarrollado en Java versión 17 del SDK.

- **Frontend**: está desarrollado utilizando Angular 12 como framework, el cual hace uso de distintos lenguajes de programación como HTML, CSS y Typescript.

- **Base de datos**: el lenguaje utilizado es SQL (Structured Query Language) ya que PostgreSQL se considera una base de datos SQL.

Herramientas:

- **Docker**: es una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente empaquetando software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Este proyecto utiliza Docker para generar dos contenedores encargados de manejar las instancias de PostgreSQL y Metabase. [15]

- **IntelliJ IDEA Ultimate Edition**: es el entorno de desarrollo integrado (IDE) desarrollado por Jetbrains seleccionado para el desarrollo del proyecto puesto que soporta los lenguajes de programación descritos en el punto anterior, además de presentar diversas funcionalidades que facilitan el desarrollo del proyecto como la compatibilidad con sistemas de control de versiones (VCS) como GIT, compatibilidad con Docker y soporte para base de datos como PostgreSQL. [16]

- **GitHub**: es una plataforma de hospedaje de código para el control de versiones y la colaboración utilizando el sistema de control de versiones de GIT y es donde se encuentra el repositorio del código de nuestro proyecto. [17]

- **GIT**: sistema de control de versiones utilizado para el seguimiento y control de nuestro proyecto.

- **Metabase**: es una herramienta de visualización de datos e inteligencia empresarial de código abierto que se utilizará para crear cuadros de mando a partir de múltiples fuentes de datos y que permitirá al propietario de un ecosistema aprender y tomar decisiones en base a la información generada en su ecosistema.

  Se ha elegido esta herramienta de BI sobre otras de la misma índole por presentar una curva de aprendizaje baja gracias a la documentación y ejemplos que se

aportan en su página web [18]. Otro motivo para su elección es que ofrece una versión gratuita que cumple con todas las funcionalidades que necesita el proyecto para una versión básica como, por ejemplo, la creación y exportación de cuadros de comando para incrustarlos en la aplicación web de la empresa, sin necesidad de que la empresa incurra en gastos extras.

- **PostgreSQL**: es un sistema de gestión de base de datos relacional orientado a objetos y de código abierto que ha ganado en los últimos 30 años una fuerte reputación por su confianza, robustez y rendimiento. [19]

  Se ha elegido este sistema de base de datos relacional que utilizará Metabase por la cantidad de ejemplos de uso que se encuentran en la documentación oficial de Metabase, sumado a que permite realizar procesos BI (como construir tablas dinámicas) que otras bases de datos como MongoDB no pueden soportar.

4.3 Iteraciones.

En cuanto al número de iteraciones, el proceso de desarrollo consta de cuatro iteraciones las cuales tienen un periodo de duración aproximado de un mes, cuya planificación temporal y objetivos principales se muestran en la tabla 3.

Nº de iteración	Fecha inicio	Fecha Fin	Objetivos
0	01/12/2021	31/12/2021	Estudio de documentación y tecnologías
1	01/01/2022	31/01/2022	Prototipo inicial
2	01/02/2022	28/02/2022	Estadísticas generales
3	01/03/2022	31/03/2022	Estadísticas detalladas
4	01/04/2022	30/04/2022	Ampliación y mejora de funcionalidades

Tabla 3 Iteraciones del proyecto

A pesar de que la última iteración termina el 30 de abril de 2022, es importante destacar que este proyecto continúa en desarrollo hasta la actualidad puesto que debe adaptarse a los cambios en los datos y la constante evolución de la plataforma para la que luego se integrará este módulo.

Cada una de las iteraciones de la uno a la cuatro sigue el esquema de la figura 35, en la que cada fase de la iteración se desarrolla de la siguiente manera:

- **Planificación:** en esta fase el Product Owner se reúne con los clientes para obtener los KPIs específicos que desean medir de su ecosistema mediante una entrevista, y una vez obtenidos se realiza la reunión del inicio de la iteración para establecer los objetivos de esta.

- **Análisis y diseño:** en esta fase el desarrollador analiza y clasifica los KPIs comunicados por el Product Owner que son posibles de medir según la información que posee la plataforma y se procede a su análisis multidimensional para elaborar la estructura de tablas de la base de datos. Luego, se procede a analizar como extraer la información de las fuentes de datos para diseñar el proceso ETL correspondiente.

- **Implementación:** en esta fase el desarrollador implementa el proceso ETL diseñado en la fase anterior para extraer, transformar y cargar los datos en los data marts, a partir del cual mediante consultas SQL y el uso de Metabase poder crear las gráficas que mejor representen la información.

- **Test y validación:** en esta fase el desarrollador comprueba que los datos de salida -los valores de los gráficos- son los datos esperados después de realizar el proceso ETL. Además, en esta misma fase se procede a validar el producto desarrollado durante la iteración en cuanto al proceso de obtención y análisis de datos, la implementación software desarrollada y el apartado visual de los gráficos elaborados mediante una reunión del desarrollador con los tutores y el Product Owner.

- **Documentación:** etapa final de la iteración en la que el desarrollador documenta lo realizado durante la iteración como los KPIs elegidos para desarrollar durante la iteración; análisis multidimensional efectuado; desarrollo del proceso ETL correspondiente; consultas SQL y uso de Metabase para elaborar el gráfico correspondiente a cada KPI seleccionado; y por último los resultados de la fase de validación.

Por cada iteración normalmente se realizan varias reuniones con los tutores y el Product Owner. Por lo general se realizan 3 reuniones por iteración, una al principio de cada iteración para planificar la iteración; la segunda a mediados de la iteración para revisar el progreso de los objetivos actuales, revisar si hay cambios en los requisitos y para la resolución de posibles dudas; y la última, al final de la iteración para evaluar el producto entregado. También se realizan reuniones aparte de las anteriores con alguno de los dos tutores para preguntar dudas con respecto a algún apartado técnico relacionado con la iteración en curso del proyecto.

Hay que destacar que también se han realizado varias reuniones con clientes y potenciales clientes de la empresa para mostrar el progreso de la plataforma y el roadmap que va a seguir la empresa para los meses futuros poniendo de ejemplo el desarrollo de este módulo de analíticas.

4.4Desarrollo del proyecto.

4.4.1 Iteración 0.

Para el desarrollo de este proyecto, se realiza una iteración cero para realizar un estudio previo a la implementación de la solución sobre los temas relacionados con este trabajo debido a la falta de conocimientos en este ámbito de la informática, cuyos conceptos están expuestos en el apartado 2.3, así como de las tecnologías que se van a emplear para la solución que se especifican en el apartado 4.2.

En esta iteración se confeccionó el archivo de configuración de Docker expuesto anteriormente en la imagen 11 y el script de inicialización de PostgreSQL de la imagen 12, para poder hacer uso de las tecnologías que se utilizarán en las iteraciones posteriores.

Durante dicha iteración, se realizaron varias reuniones con los tutores para la explicación de los conceptos, la resolución de dudas asociados al respecto y cómo iba a ser la estructura del proyecto que se iba a desarrollar en las futuras iteraciones.

4.4.2 Iteración 1.

Objetivos: construir un prototipo inicial que extraiga los datos del data lake, los transforme a un modelo dimensional y los cargue dentro de la base de datos objetivo. Generar un dashboard general del estado del ecosistema.

El Product Owner comunica los KPIs básicos que se desea medir en un ecosistema. A partir de ahí se elabora una lista con los KPIs que se pueden realizar en base a los datos de la plataforma que se encuentran en el data lake para una versión inicial resultando en la siguiente lista:

KPIs:

1) Nº de actores en el ecosistema.
2) Nº de actores pendientes de aceptar en el ecosistema.
3) Nº de miembros en el ecosistema.
4) Nº de chats en el ecosistema.
5) Nº de salas en el ecosistema.
6) Nº de canales públicos en el ecosistema.
7) Nº de publicaciones en el ecosistema.
8) Nº de proyectos en el ecosistema.
9) Límite de almacenamiento (no se puede realizar).
10) Facturación (no se puede realizar)

Desarrollo:

Se desarrolla el análisis multidimensional para cada uno de esos KPIs como se puede observar en el ejemplo de la figura 36 correspondiente al KPI 1 de la lista anterior.

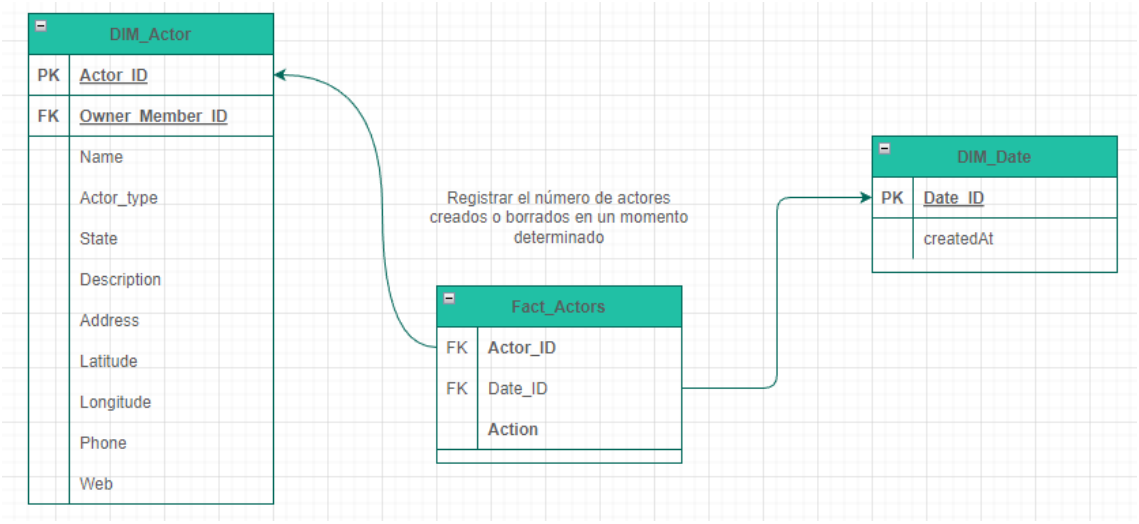


Figura 36 Ejemplo análisis multidimensional: nº de actores creados y borrados

Luego, se desarrolló la arquitectura básica del módulo que se puede observar una versión resumida de esta en la figura 37, compuesta por 3 servicios encargados de realizar cada uno las diferentes partes de un proceso ETL de forma que realizase un procesamiento en lotes, una entidad por cada data mart encargada de controlar su proceso ETL de manera independiente utilizando los 3 servicios anteriores descritos como refleja la figura 39. Por último, se creó la configuración necesaria para conectarse a la base de datos de PostgreSQL y se crearon dos repositorios encargados ejecutar las consultas para insertar los datos resultantes de cada proceso ETL dentro de las tablas de dimensión y de hechos de cada entidad. La figura 38 muestra la estructura del proyecto resultado de la implementación de todo lo expuesto anteriormente en la primera iteración.

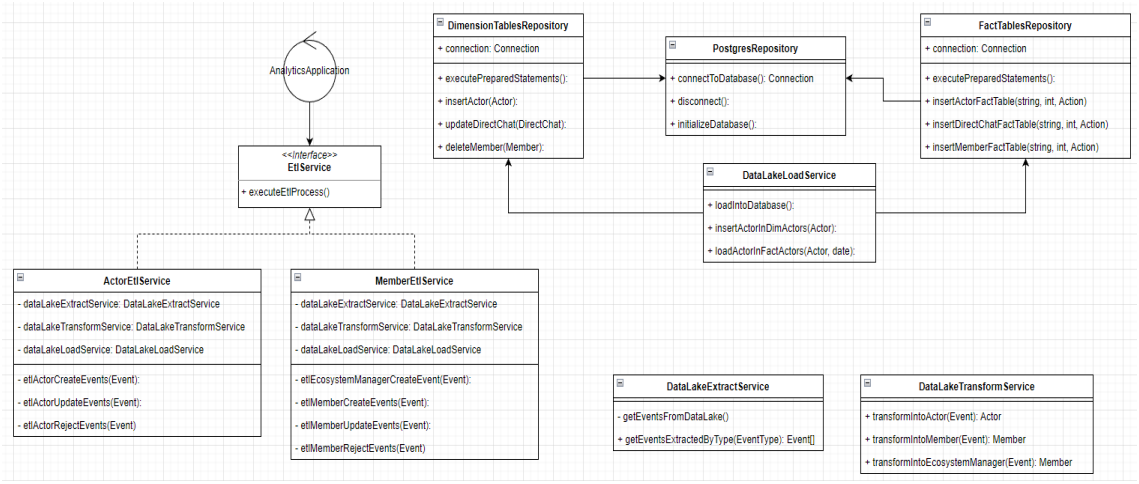


Figura 37 Arquitectura proyecto (iteración 1)

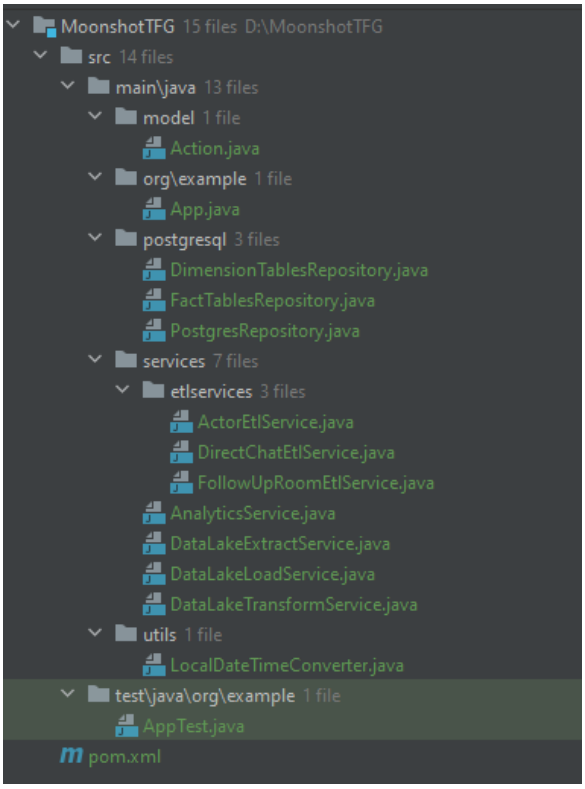


Figura 38 Estructura proyecto iteración 1

```java
public ActorEtlService(
    DataLakeExtractService dataLakeExtractService,
    DataLakeTransformService dataLakeTransformService,
    DataLakeLoadService dataLakeLoadService) {
        this.dataLakeExtractService = dataLakeExtractService;
        this.dataLakeTransformService = dataLakeTransformService;
        this.dataLakeLoadService = dataLakeLoadService;
}

public List<PreparedStatement> executeEtlProcess() {
    List<PreparedStatement> preparedStatements = new ArrayList<>();
    preparedStatements.addAll(this.etlActorCreateEvents());
    preparedStatements.addAll(this.etlActorAcceptEvents());
    preparedStatements.addAll(this.etlActorRejectEvents());
    preparedStatements.addAll(this.etlActorUpdateEvents());
    preparedStatements.addAll(this.etlActorDeleteEvents());
    return preparedStatements;
}

private List<PreparedStatement> etlActorCreateEvents() {
    LOGGER.info("ETL actor create events process starting...");
    List<PreparedStatement> preparedStatements = new ArrayList<>();
    List<Event<?>> eventsExtractedByType = dataLakeExtractService.getEventsExtractedByType(ACTOR_CREATE_EVENT);
    eventsExtractedByType.forEach(event -> {
        Actor actorCreated = dataLakeTransformService.transformIntoActor(event);
        preparedStatements.add(dataLakeLoadService.dimActorsLoadNewActor(actorCreated));
        LocalDateTime createdAt = event.getCreatedAt().toLocalDateTime();
        preparedStatements.add(dataLakeLoadService.factActorsLoadNewActor(actorCreated, createdAt));
    });
    LOGGER.info("ETL actor create events process finished.");
    return preparedStatements;
}
```

Figura 39 ActorEtlService

Una vez los datos se encontrasen dentro de la base de datos destino, en un principio se utilizó el editor que proporciona Metabase (véase figura 40) para generar las gráficas que midiesen los KPIs requeridos pues era la forma más sencilla de generar las representaciones gráficas.

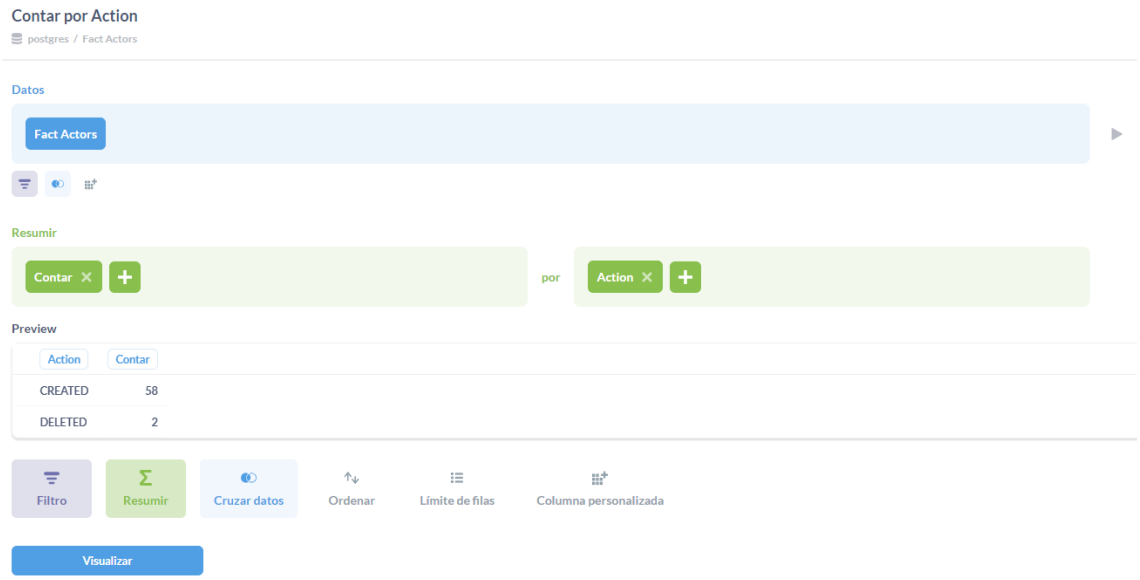


Figura 40 Interfaz gráfica editor de consultas de Metabase

Resultado:

Una vez realizadas todas las gráficas correspondientes a los KPIs descritos inicialmente muchas de estas se guardan dentro del dashboard que muestra el estado actual del ecosistema que puede observarse en la figura 41.

Figura 41 Versión inicial dashboard mi ecosistema

4.4.3 Iteración 2.

Objetivo: añadir estadísticas generales a cada data mart y generar sus dashboards.

Tras validar el proceso anterior, se procedió a realizar la reunión de planificación del nuevo sprint con el fin de analizar y clasificar nuevos KPIs básicos para cada data mart, los cuales se recogen en la siguiente lista:

KPIs:

1) Nº de cada entidad de la plataforma creada en la última hora.

2) Evolución en el tiempo de cada entidad de la plataforma.
3) Top 10 canales públicos más activos en un ecosistema.
4) Distribución de las salas abiertas y cerradas.
5) Media de participantes nuevos por sala de trabajo.
6) Distribución de los actores en el ecosistema por tipo de actor.
7) Proyectos creados por actores.
8) Top 10 publicaciones con más likes.
9) Top 10 publicaciones con más comentarios.

Desarrollo:

Siguiendo el mismo proceso descrito en la anterior iteración se procedió a realizar el análisis multidimensional de los KPIs como muestra el ejemplo de la imagen 42 correspondiente a los KPIs 7 y 8, para luego elaborar el proceso ETL y la creación de las gráficas y dashboards, en este caso el dashboard de las publicaciones como muestra la figura 43.

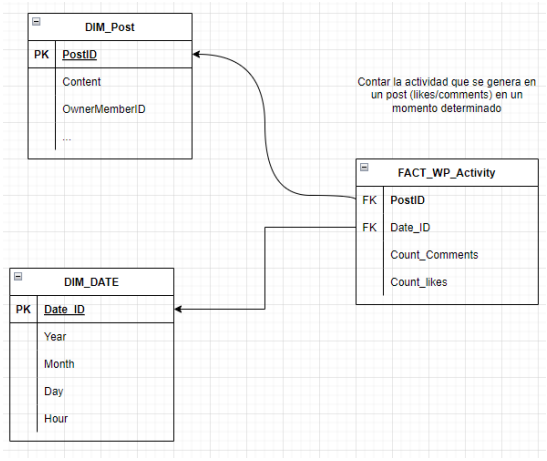


Figura 42 Ejemplo análisis multidimensional: Actividad de una publicación

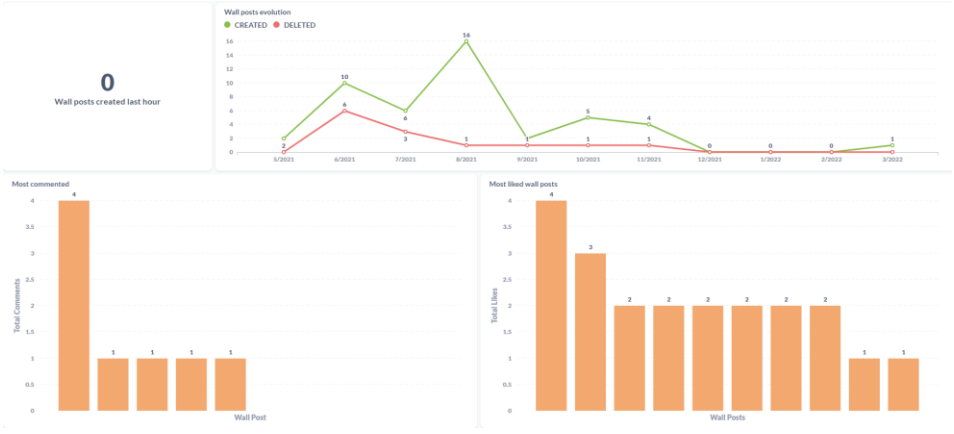


Figura 43 Wall post dashboard (Iteración 2)

Resultado:

Al final de esta iteración, se desarrolló un front-end para el módulo con un diseño muy parecido a la plataforma de la empresa con el fin de incrustar los dashboards elaborados mediante un iframe con la url pública proporcionada por Metabase en las dos iteraciones realizadas (figura 44). Mediante diversas reuniones, se mostró a los clientes los

dashboards elaborados utilizando este prototipo tal y como se muestra en la figura 45 y se les preguntó por su opinión y posibles sugerencias de mejoras para las siguientes iteraciones.

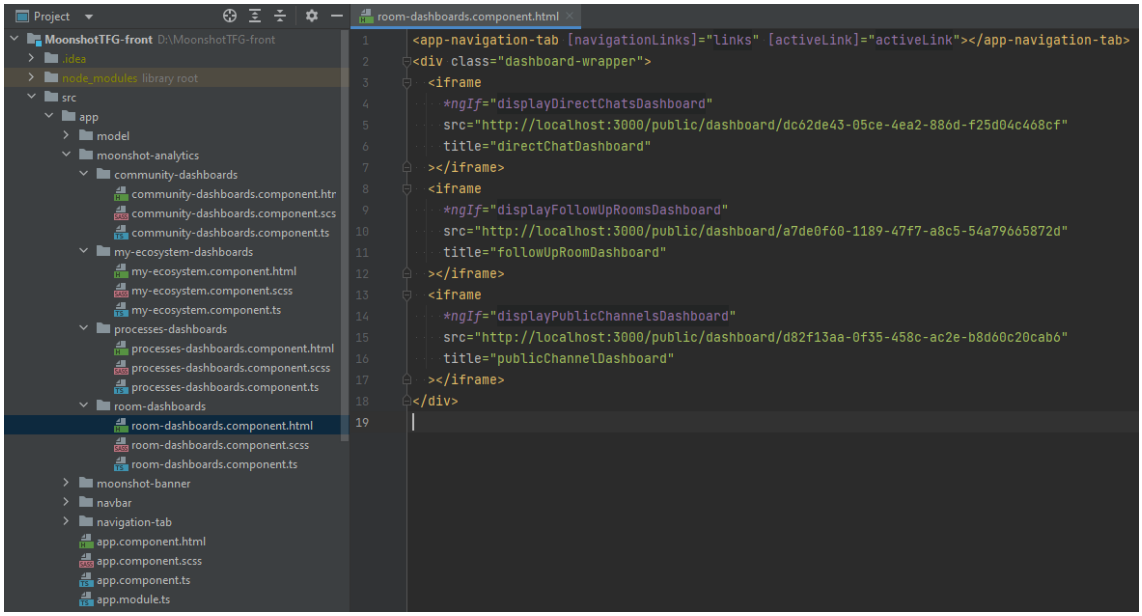


Figura 44 Ejemplo código front-end proyecto

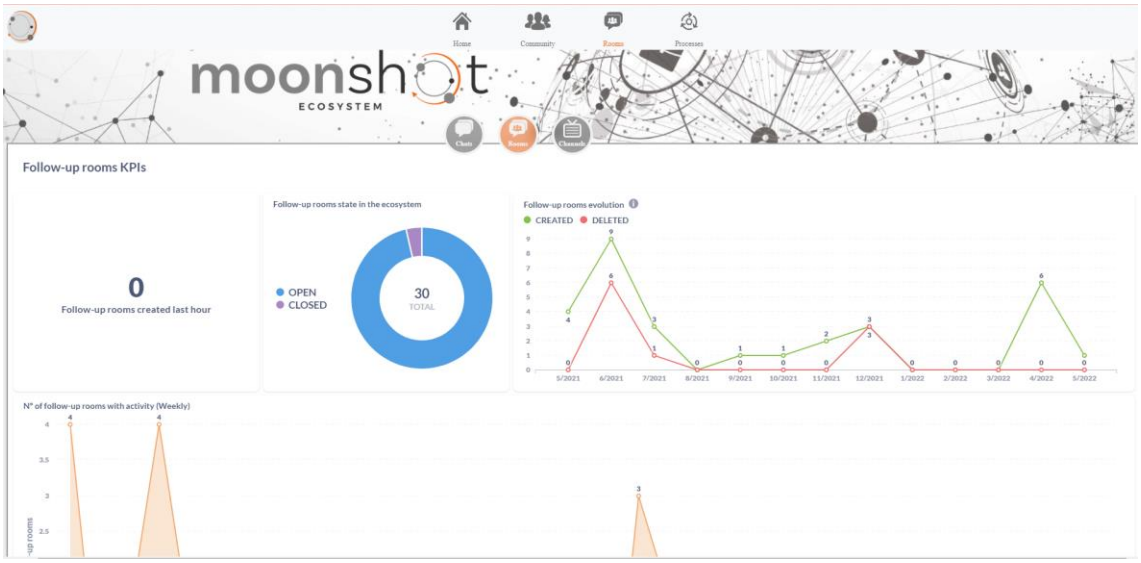


Figura 45 Prototipo front-end del proyecto

4.4.4 Iteración 3.

Objetivo: añadir estadísticas detalladas a cada data mart dependiendo de los KPIs y mejorar el apartado visual.

Para esta iteración se repite el proceso de las dos anteriores: reunión de planificación de la iteración; clasificación de los KPIs; análisis multidimensional de los KPIs elegidos (figura 46); desarrollo del proceso ETL como ejemplifica la figura 47 y la elaboración de las gráficas que se añadirán a los dashboards.

En este caso, la lista de KPIs que se quieren analizar proceden de las reuniones con los clientes al final de la anterior iteración:

KPIs:

1) 10 actores más activos del ecosistema.
2) Top 60 tecnologías demandadas y ofertadas por actores.
3) Top 10 industrias demandadas y ofertadas por actores.
4) Top 10 innovaciones sociales ofertadas y demandadas por actores.
5) Indicador de business models demandados y ofertados por actores.
6) Eventos programados por tipo de evento.
7) Eventos en el mes actual por tipo de evento.
8) Nº de salas de trabajo con actividad semanal.
9) Indicadores de taxonomías relacionadas con proyectos.

Desarrollo:

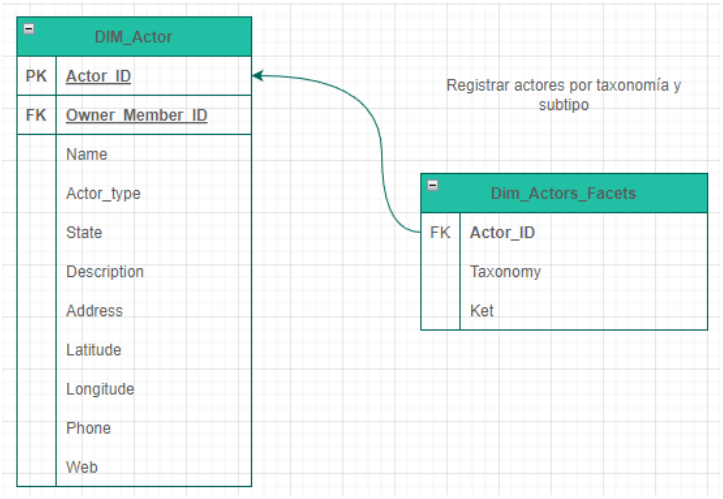


Figura 46 Ejemplo análisis multidimensional: taxonomías de un actor

Figura 47 Ejemplo código: proceso ETL taxonomías de un actor

También en esta iteración se aplicaron técnicas de data visualization para mejorar el apartado visual, como el uso de una misma paleta de colores para los mismos tipos de gráficas en los data marts como son los que representan las figuras 26 y 27, tras las opiniones recogidas fruto de las reuniones con los clientes.

Resultado:

Por tanto, al final de la iteración el producto presentó una estructura mejor y más homogénea visualmente junto con nuevos KPIs específicos requeridos por los clientes.

4.4.5 Iteración 4.

Objetivo: mejorar la arquitectura del proyecto y ampliar algunas funcionalidades.

En esta iteración a partir de la reunión de planificación, se decidió no desarrollar nuevos dashboards ni analizar nuevos KPIs puesto que se debía mejorar la arquitectura del proyecto, corregir bugs, realizar refactorizaciones de código y ampliar algunas funcionalidades.

Desarrollo:

Con respecto a la arquitectura, se implementó el patrón Repository creando así un repositorio específico para cada una de las tablas de dimensiones y hechos elaboradas en las iteraciones anteriores al mismo tiempo que se crearon sus modelos correspondientes dentro del proyecto.

Se añadieron nuevas funcionalidades como tener un servicio de tipo Scheduler que automatizase la ejecución de las tareas de los procesos ETL de todos los servicios ETL

creados hasta el momento (figura 48), así como la inserción de las fechas y horas cada vez que se cambie de año dentro de la tabla de dimensión del tiempo (figura 49).

```java
public class DataLakeEtlProcessJob implements Job {

    private static final Logger LOGGER = LoggerFactory.getLogger(DataLakeEtlProcessJob.class);

    @Override
    public void execute(JobExecutionContext jobExecutionContext) {
        LOGGER.info("Executing data lake ETL process job...");
        List<EtlService> etlServices = (List<EtlService>) jobExecutionContext
            .getMergedJobDataMap()
            .get("etlServices");
        etlServices.forEach(EtlService::executeEtlProcess);
        LOGGER.info("Data lake ETL process job finished.");
    }
}
```

Figura 48 Scheduler Job: Ejecutar procesos ETL

```java
public class InsertDateTimeYearlyJob implements Job {

    private static final Logger LOGGER = LoggerFactory.getLogger(InsertDateTimeYearlyJob.class);

    @Override
    public void execute(JobExecutionContext jobExecutionContext) {
        int currentYear = LocalDateTime.now().getYear();
        String startDate = String.format("%d/01/01 00:00:00", currentYear);
        String endDate = String.format("%d/12/31 23:00:00", currentYear);
        DimDateTimeRepository dimDateTimeRepository = (DimDateTimeRepository) jobExecutionContext
            .getMergedJobDataMap()
            .get("dimDateTimeRepository");
        LOGGER.info("Inserting new date times between {} and {} into database.", startDate, endDate);
        dimDateTimeRepository.insertMany(LocalDateTimeUtils.toLocalDateTimeList(startDate, endDate));
    }
}
```

Figura 49 Scheduler Job: insertar fechas año actual

Se refactorizó código utilizando el plugin de Sonarlint para eliminar código duplicado, diversos code smells, posibles fallas de seguridad… con el fin de aumentar la calidad del producto entregado. En la figura 50 se muestra el último análisis efectuado al proyecto en el que se recogen 9 code smells, de los cuales 7 tienen un nivel de major code smell vinculado al uso de cadenas de caracteres literales repetidas en lugar de usar constantes y los 2 restantes son clasificados como minor code smell. Por tanto, se puede concluir que el módulo desarrollado presenta un código bastante limpio que facilite su mantenimiento.

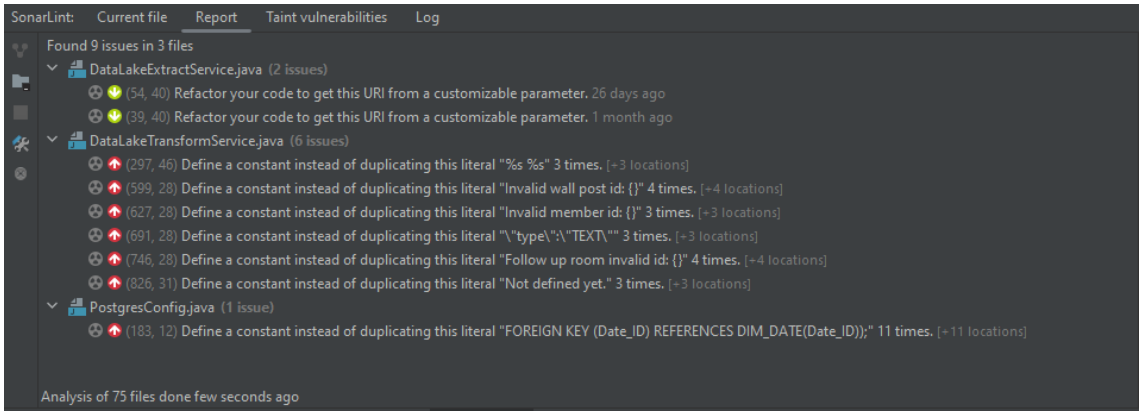


Figura 50 Resultados análisis SonarLint

Resultados:

Como resultado de la iteración, ocurrieron grandes cambios puesto que se reestructuró la estructura del proyecto como se observa en la figura 51 y su arquitectura software final en la figura 20, para adaptarlo de mejor forma a la arquitectura BI de la solución. Del mismo modo, se añadieron nuevas funcionalidades y se mejoró la calidad del código para aumentar el valor del producto.

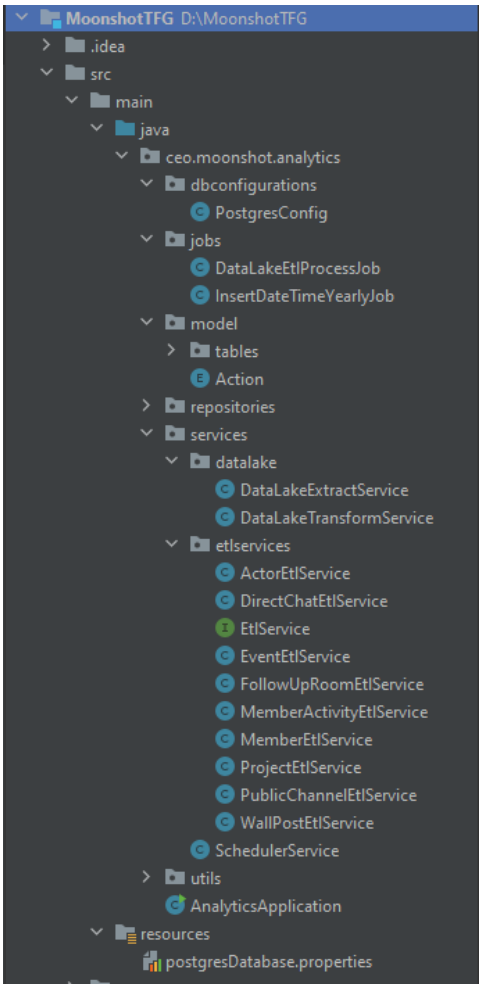


Figura 51 Estructura final del proyecto

5 Conclusiones.

5.1 Resultados.

El presente trabajo tuvo como objetivo solucionar los problemas que sufrían los propietarios de un ecosistema sobre la falta de información del estado y la actividad que se genera en un ecosistema por parte de la plataforma de la empresa. Esto dificultaba enormemente el gestionar el crecimiento y la dinamización de un ecosistema puesto que no había ninguna herramienta que proporcionase información útil que ayudase a los gestores de este en su toma de decisiones.

Para resolver estos problemas, se propuso implementar una solución de BI al modelo de negocio de la plataforma mediante el desarrollo de un módulo independiente cuyo principal objetivo es extraer información de los eventos almacenados en cada data lake de un ecosistema, para luego analizarlos y poder generar una base de conocimiento que ofrecer a los propietarios del ecosistema mediante herramientas de visualización como dashboards para conocer su ecosistema.

Primero se realizó un estudio de la arquitectura BI que necesitaría la empresa para imlpementar la solución para construir los data marts y el data warehouse en función de la estructura de la plataforma, después se analizaron los datos fuentes que se encontraban en el data lake de un ecosistema y los KPIs que requerían medir los clientes mediante varias entrevistas, para los cuales se aplicó un análisis multidimensional con el que construir el modelo de datos que conformarían los data marts y data warehouse.

Posteriormente, mediante varias iteraciones se desarrolló el módulo cuya tarea es realizar las fases de un proceso ETL: extraer los datos del data lake; transformar esos datos en su nuevo modelo dimensional; y cargar esos nuevos datos en cada uno de los data marts correspondientes; con una cierta periodicidad para tener los datos constantemente actualizados.

Por último, una vez cargados los datos dentro de los data marts y el data warehouse con los procesos ETL, se construyeron las gráficas y dashboards que muestran la información requerida por los clientes mediante el uso de herramientas y técnicas de visualización de datos.

Como resultado de todo este proceso, se ha obtenido un nuevo módulo escalable y adaptable que pueda responder a cada una de las diferentes necesidades actuales y futuras de los clientes al igual que sienta las bases para una solución de BI para la plataforma en constante evolución de la empresa. Además, se ha conseguido desarrollar un proyecto con implicaciones reales que una vez finalizado se ha lanzado al mercado y ha obtenido bastante retroalimentación por parte de los usuarios de la plataforma en forma de una buena aceptación y de posibles mejoras.

Como limitaciones del proyecto solo queda por reseñar la falta de automatización de tests del módulo desarrollado puesto que al no utilizar un framework de desarrollo como

Spring que facilite estas tareas, iba a ser muy costoso en tiempo y forma crear las configuraciones necesarias para la automatización de tests utilizando PostgreSQL como nueva base de datos.

5.2 Contribuciones.

Al finalizar este trabajo, se puede afirmar que los objetivos principales como secundarios inicialmente planteados en el apartado 1.1 han sido alcanzados con éxito desde distintas perspectivas:

- **A nivel personal:** ha contribuido a obtener unos conocimientos de una rama de la informática (Business Intelligence) que era desconocida puesto que no se cursó ninguna asignatura relacionada con este ámbito en la carrera e, incluso, a despertar el interés por esta misma como especialización del perfil profesional de ingeniero informático.

- **A nivel profesional:** ha contribuido a obtener y mejorar mis conocimientos sobre las diversas tecnologías y herramientas utilizadas; los procesos seguidos; la realización de un proyecto real; y como resultado obtener experiencia real que puede plasmarse en el currículum profesional.

- **A nivel académico:** ha contribuido a descubrir nuevos indicadores del nivel de emprendimiento e innovación dentro de los procesos internos empresariales dentro de los ecosistemas de la aplicación y cómo estos se relacionan entre sí.

- **A nivel empresarial:** ha contribuido a añadir un mayor valor a la empresa frente a sus competidores, así como capaz de atraer a nuevos potenciales clientes y ser una herramienta que permite mejorar la toma de decisiones para con su producto.

- **A nivel de producto:** ha contribuido a implementar nuevas funcionalidades que solucionan un problema que demandaban los gestores de innovación de los ecosistemas, con lo cual el producto de la empresa mejora su calidad y presenta un valor añadido para los clientes.

5.3 Experiencia personal.

En un primer momento consideraba bastante duro el desarrollo de este TFG por mi desconocimiento y falta de experiencia de varias de las herramientas que se iban a utilizar, al igual que mi amplio desconocimiento de muchos conceptos relacionados con el Business Intelligence, pero tenía bastante interés e ilusión por el concepto del mismo lo que me permitió sobrellevarlo.

En las fases iniciales del proyecto, dicho desconocimiento fue la mayor barrera al progreso del mismo puesto que necesité de casi un mes para comprender todos los conceptos teóricos e intentar ponerlos en prácticas. Lo mismo ocurrió con algunas tecnologías para las cuales tenía un mínimo conocimiento adquirido en las prácticas como Docker o ninguno como Metabase o PostgreSQL y, por tanto, tuve que dedicar también bastante tiempo a estudiar su documentación oficial para resolver los continuos problemas que surgieron durante el desarrollo.

He de destacar la labor de ambos tutores, pues a pesar de dejarme más de lo que me hubiera gustado en un principio a mi libre albedrío para desarrollar la solución, siendo yo mismo quien se enfrentase a los problemas y los resolviese por su cuenta, siempre estaban dispuestos a ayudar resolviéndome cualquier duda y sugiriendo posibles mejoras con el fin de que aprendiese y el de desarrollar un buen trabajo que creo que se ha realizado.

Mirando en retrospectiva, ha terminado en una experiencia desafiante y gratificante a partes iguales pues, a pesar de las dificultades que expuse anteriormente, con esfuerzo y dedicación he superado los distintos problemas que han surgido durante el desarrollo del proyecto. Como resultado, considero que he ampliado mis conocimientos en el uso de algunas tecnologías y metodologías, soy más autodidacta y he descubierto una rama de la informática que me ha llamado la atención como posible especialización de cara al futuro.

5.4Trabajos futuros.

Como se ha explicado en varias partes de la memoria, este TFG consiste en desarrollar un nuevo módulo que sienta las bases para una arquitectura de Business Intelligence para una plataforma modular ya existente. Por tanto, el trabajo futuro más importante e inmediato que presenta este proyecto es su implementación real en la plataforma de la empresa.

Esta implementación conllevará algunos cambios extras en su configuración para añadirlo a la plataforma y adaptarlo para que funcione con el resto de los módulos. Algunos de estos cambios pueden ser: la configuración de los contenedores de Docker utilizados (Metabase y PostgreSQL) para que funcionen en un entorno de desarrollo y de producción; la creación de un contenedor propio de Docker como módulo para su despliegue en la plataforma; y una vez hechos estos cambios, las tareas de despliegue a producción.

Además, se debe destacar que la plataforma Moonshot es una herramienta que se encuentra en constante evolución y cambio, por lo que existen una multitud de futuras líneas de trabajo que pueden repercutir en este nuevo módulo mediante cambios en la estructura de los datos almacenados, nuevas funcionalidades o nuevos módulos que generen datos susceptibles para ser analizados.

Incluso si se eliminase el factor de evolución constante de la plataforma, el módulo desarrollado sienta unas bases mínimas que se pulirán con el paso del tiempo y por ello presenta el mismo potencial de cambio y evolución descrito anteriormente debido a que:

- Se pueden desarrollar nuevos análisis en función de los datos ya almacenados en el data warehouse o integrando nuevos datos desde otras fuentes.

- Se pueden desarrollar nuevos análisis en función de nuevos requerimientos por parte de los clientes que no habían sido relevantes anteriormente.

- Se pueden mejorar los procesos de análisis existentes y la arquitectura del proyecto para ser más eficiente y presentar una alta mantenibilidad.

- Se pueden mejorar los análisis existentes explorando otras vías de representación que aporten mayor conocimiento.

Por todo lo expuesto anteriormente, este trabajo es un módulo básico inicial que soluciona un problema actual pero que presenta una alta capacidad para evolucionar y adaptarse para dar solución a posibles problemas futuros.

6 Bibliografía

[1] Metabase, «Metabase,» [En línea]. Available: https://www.metabase.com/glossary. [Último acceso: Junio 2022].

[2] Moonshot Innovation S.L., *Arquitectura modular Moonshot,* Madrid, 2021.

[3] Oracle, «Oracle.com,» [En línea]. Available: https://www.oracle.com/business-analytics/business-intelligence/. [Último acceso: 2022].

[4] Iberdrola, «Iberdrola,» [En línea]. Available: https://www.iberdrola.com/innovacion/que-es-business-intelligence. [Último acceso: Junio 2022].

[5] G. R. Rivadera, 2010. [En línea]. Available: https://www.ucasal.edu.ar/htm/ingenieria/cuadernos/archivos/5-p56-rivadera-formateado.pdf. [Último acceso: Enero 2022].

[6] SAS, «Sas,» [En línea]. Available: https://www.sas.com/en_us/insights/data-management/what-is-etl.html. [Último acceso: Junio 2022].

[7] ETL Tools.org, «Etl Tools,» [En línea]. Available: https://www.etltools.org/. [Último acceso: Junio 2022].

[8] Astera, «Astera,» 4 Febrero 2020. [En línea]. Available: https://www.astera.com/es/tipo/blog/proceso-etl-y-pasos/. [Último acceso: 2022].

[9] PowerData, «Blog PowerData,» Junio 2013. [En línea]. Available: https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/bid/288859/procesos-etl-extracci-n-transformaci-n-carga. [Último acceso: Enero 2022].

[10] Prometeus Global Solutions, «Prometeus GS,» 11 Enero 2021. [En línea]. Available: https://prometeusgs.com/analizando-y-explorando-los-datos-en-detalle-para-que-sirve-el-analisis-de-datos-multidimensionales/. [Último acceso: Junio 2022].

[11] Ayudaley, «Ayudaleyprotecciondatos,» [En línea]. Available: https://ayudaleyprotecciondatos.es/bases-de-datos/multidimensionales/. [Último acceso: Diciembre 2021].

[12] S. Group, «Youtube,» Noviembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=lkTJ_XgOC7U&ab_channel=SmartbaseGroup. [Último acceso: Diciembre 2021].

[13] Material Design, «Material Design,» [En línea]. Available: https://material.io/design/communication/data-visualization.html. [Último acceso: Marzo 2022].

[14] "Ciclo de vidas y metodologías", apuntes de Ingeniería del Software 1, Escuela de Ingeniería Informática, ULPGC, 2019.

[15] Amazon, «Amazon Aws,» [En línea]. Available: https://aws.amazon.com/es/docker/. [Último acceso: Junio 2022].

[16] JetBrains, «JetBrains,» [En línea]. Available: https://www.jetbrains.com/es-es/idea/features/. [Último acceso: Junio 2022].

[17] GitHub, «GitHub Docs,» [En línea]. Available: https://docs.github.com/es/get-started/quickstart/hello-world. [Último acceso: Diciembre 2021].

[18] Metabase, «Metabase,» [En línea]. Available: https://www.metabase.com/learn/getting-started/getting-started. [Último acceso: Diciembre 2021].

[19] PostgreSQL, «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/ . [Último acceso: Diciembre 2021].

7 Anexo.

7.1GLOSARIO DE TÉRMINOS.

- **Agregación**: es el acto de resumir datos mediante una función matemática (ej: calcular la media de los valores de una columna de una tabla), con el fin de obtener un único resultado para medir nuevas métricas a partir de muchos datos que por sí solos no aportan ningún significado.
- **Application Program Interface (API)**: interfaz que define como otros programas pueden comunicarse con la aplicación.
- **Atributo**: propiedad que describe o identifica a una entidad.
- **Base de datos analítica**: es una base de datos optimizada para realizar pocas pesadas operaciones de análisis en vez de muchas pequeñas transacciones.
- **Base de datos relacional**: base de datos que almacena la información de una manera tabular, es decir, almacena la información en tablas mediante filas y columnas estableciendo relaciones entre diferentes tablas para almacenar información estructurada más compleja.
- **Business Intelligence (BI)**: uso de estrategias y herramientas que sirven para obtener conocimiento a partir de la transformación de información interna y externa que posee una empresa, con el objetivo de mejorar el su proceso de toma de decisiones.
- **Clave ajena**: atributo de una tabla que hace referencia a la clave primaria de otra tabla de la base de datos estableciendo una relación entre ambas.
- **Clave primaria**: valor único que identifica una fila dentro de una tabla de la base de datos.
- **Colección**: Conjunto de elementos -preguntas, modelos, dashboards, subcolecciones- almacenados juntos con un objetivo empresarial.
- **Cuadro de comando o Dashboard**: herramienta de visualización de datos que contiene textos y gráficos importantes recogidos y agrupados en una sola pantalla, como por ejemplo el salpicadero de un coche, siendo fáciles de leer. Proveen una vista centralizada y detallada de KPIs y otras métricas de negocio como pueden ser desde el estado general de la empresa hasta el éxito de un proyecto específico.

- **Data lake**: lugar donde se almacena información estructurada o no de diferentes fuentes, usualmente en ficheros o blobs.
- **Data mart**: es un repositorio de información similar a un data warehouse pero orientado a un área o departamento concreto dentro de una organización.
- **Data model**: patrón para organizar y etiquetar información con el fin de facilitar la búsqueda de información.
- **Data warehouse**: es una base de datos diseñada y utilizada específicamente para procesos de *Business Intelligence* que contiene información estructurada sin analizar de diferentes fuentes de datos.
- **Dimensión**: atributo cualitativo de un dato que contiene información descriptiva (ej: nombre, categoría de un producto, o un país).
- **ETL**: operación común en los sistemas de procesamiento de datos cuyo procedimiento consisten en leer datos de varias fuentes de datos (Extract), transformar esa información realizando diferentes operaciones como agrupar, calcular, resumir… (Transform) y guardarlo en otro sistema de almacenamiento de datos (Load).
- **Key Performance Indicator (KPI)**: métrica que muestra el progreso de un objetivo personal o empresarial importante.
- **Medida**: atributo numérico de un dato, es decir es un campo que contiene información cuantitativa que puede ser calculada o agregada (ej: precio de un objeto, cantidad de objetos) pero que por sí sola no ofrece información relevante.
- **Métrica**: es un tipo de medida resultante de aplicar cálculos a las medidas de un dato que nos ofrece información relevante (ej: media de precio de un producto en un mes).
- **Tabla dinámica (pivot table)**: herramienta de visualización de datos que permite resumir filas y columnas de una tabla y rotar las columnas (el pivote) para visualizar los resúmenes de distintas maneras. Son muy útiles para analizar datos que dependen de múltiples dimensiones como el tiempo, la localización o la categoría de un producto.