

Universidad de Pamplona
Facultad de Ingenierías y Arquitectura
Ingeniería de Sistemas

**DISEÑO DE MICROSERVICIOS PARA REALIZAR REPORTE SOBRE UNA BASE
DE DATOS POSTGRES MEDIANTE UNA INTERFAZ GRÁFICA PARA CONSULTAS
SIN SQL**

Presenta
Steban Jose Castro Perez

Pamplona, Norte de Santander
2022

Universidad de Pamplona
Facultad de Ingenierías y Arquitectura
Ingeniería de Sistemas

**DISEÑO DE MICROSERVICIOS PARA REALIZAR REPORTE SOBRE UNA BASE
DE DATOS POSTGRES MEDIANTE UNA INTERFAZ GRÁFICA PARA CONSULTAS
SIN SQL**

Trabajo de grado

Presenta

Steban Jose Castro Perez

Para obtener el título de

Ingeniero de Sistemas

Director

PhD. William Mauricio Rojas Contreras

Pamplona, Norte de Santander

2022

Dedicatoria

Le dedico el resultado de este trabajo de grado a mis padres que gracias a su esfuerzo y dedicación han sido mi principal motivación para alcanzar las metas que me propongo en la vida. A mis amigos cercanos que siempre me han brindado su ayuda incondicional. A mi director de trabajo de grado por su orientación y dedicación a la hora de realizar este trabajo de grado. Por último agradecer a dos personas (Andrea y David) que admiro y de las cuales he aprendido directamente.

Resumen

El presente trabajo tiene como finalidad el diseño de una serie de microservicios que se desarrollarán en los lenguajes de programación Go, JavaScript (node.js) y Kotlin; la elección de dichos lenguajes se debe a ventajas orientadas al desarrollo y al uso de las características de la arquitectura orientada a microservicios, donde cada microservicio puede ser implementado de manera individual para crear una herramienta no-code cuyo objetivo es generar consultas SQL sin la necesidad de usar Native Query, llevándolo a una interfaz gráfica que le facilite a un usuario, independientemente de su nivel de conocimiento en programación, realizar reportes basado en sus necesidades, siendo esta interfaz parecida en aspecto y usabilidad a herramientas tipo hoja de cálculo. Para lograr dicha meta se realizará un análisis a la experiencia de usuario con interfaces gráficas para lograr la accesibilidad que se requiere, junto a un análisis de patrones de arquitectura y el planteamiento de un stack tecnológico. (Acá no se si es stack tecnológico o stack de tecnología o si es como está escrito)

Palabras clave: Bases de datos, Microservicios, SQL, Query builder, Arquitectura de software.

Abstract

The purpose of this work is the design of a series of microservices that will be developed in the programming languages go, JavaScript (node.js) and Kotlin; The choice of these languages is due to development-oriented advantages and the use of the characteristics of the microservice-oriented architecture, where each microservice can be implemented individually to create a no-code tool whose objective is to generate SQL queries without the need of using Native Query, taking it to a graphical interface that makes it easier for a user, regardless of their level of knowledge in programming, to make reports based on their needs, this interface being similar in appearance and usability to spreadsheet-type tools, to achieve this objective, It will carry out an analysis of the user experience with graphical interfaces to achieve the accessibility that is required, together with an analysis of architectural patterns and the approach of a technology stack.

Keywords: Databases, Microservices, SQL, Query builder, Software architecture.

Tabla de contenido

Planteamiento del problema	11
Justificación	12
Objetivos	13
General	13
Específicos	13
Alcances y limitaciones	14
Alcances	14
Limitaciones	14
Marco Teórico	15
Arquitectura de Software	15
Patrones de diseño	16
Patrones de arquitectura	16
Patrón por capas	17
Patrón cliente-servidor	18
Patrón modelo-vista-controlador (MVC)	19
Patrón orientado a microservicios	20
Lenguajes de programación	21
Kotlin	21
Go	21
Javascript	22
Frameworks y Librerías	23
Spring boot	23

Echo	24
Node.js	24
Express	24
Knex.js	25
React	25
Next.js	25
Bases de datos	25
PostgreSQL	25
Otras herramientas	26
Docker	26
Kubernetes	26
Herramientas no-code	26
Estado del arte	27
Investigaciones internacionales	27
Investigaciones nacionales	29
Herramientas	30
Herramientas no-code sobre base de datos	30
Herramientas low-code sobre de datos	31
Análisis de requerimientos	33
Comparativa entre herramientas	33
Query builder	33
Reportes	37
Requerimientos funcionales	38

	8
Requerimientos no funcionales	38
Justificación de solución propuesta	39
Arquitectura de microservicios	39
Ventajas	39
Desventajas	39
Comparativa entre la arquitectura de microservicios y monolítica	40
Ventajas de una arquitectura monolítica contra una arquitectura de microservicios	40
Desventajas de una arquitectura monolítica contra una arquitectura de microservicios	41
Diseño de microservicios	42
Análisis de dominio	42
Definición de contexto acotado	44
Definición de entidades, agregados y servicios	45
Identificación de microservicios	48
Diagramas de casos de uso	51
Diagramas de secuencia	57
Definición de microservicios	62
Microservicio: User-MS	62
Microservicio: Integration-MS	63
Microservicio: Runner-MS	64
Microservicio: Query-Builder-MS	65
Microservicio: Query-MS	69

	9
Microservicio: Public-API	70
Definición de servicio frontend	71
Infraestructura	77
Docker Compose	77
Kubernetes	79
Componentes de kubernetes	79
Recursos de Kubernetes	81
Digital Ocean	85
Servicios	85
Escalado Automático	86
Presupuesto	87
Conclusiones	88
Recomendaciones	89
Bibliografía	90

Lista de Figuras

<i>Ilustración 1.</i>	17
<i>Ilustración 2.</i>	18
<i>Ilustración 3.</i>	19
<i>Ilustración 4.</i>	20
<i>Ilustración 5.</i>	31
<i>Ilustración 6.</i>	32
<i>Ilustración 7.</i>	34
<i>Ilustración 8.</i>	35
<i>Ilustración 9.</i>	36
<i>Ilustración 10.</i>	37
<i>Ilustración 11.</i>	40
<i>Ilustración 12.</i>	42
<i>Ilustración 13.</i>	43
<i>Ilustración 14.</i>	45
<i>Ilustración 15.</i>	47
<i>Ilustración 16.</i>	48
<i>Ilustración 17.</i>	50
<i>Ilustración 18.</i>	51
<i>Ilustración 19.</i>	52
<i>Ilustración 20.</i>	57
<i>Ilustración 21.</i>	58
<i>Ilustración 22.</i>	59
<i>Ilustración 23.</i>	60

	11
<i>Ilustración 24.</i>	60
<i>Ilustración 25.</i>	61
<i>Ilustración 26.</i>	62
<i>Ilustración 27.</i>	63
<i>Ilustración 28.</i>	65
<i>Ilustración 29.</i>	66
<i>Ilustración 30.</i>	67
<i>Ilustración 31.</i>	68
<i>Ilustración 32.</i>	68
<i>Ilustración 33.</i>	69
<i>Ilustración 34.</i>	71
<i>Ilustración 35.</i>	72
<i>Ilustración 36.</i>	72
<i>Ilustración 37.</i>	73
<i>Ilustración 38.</i>	73
<i>Ilustración 39.</i>	74
<i>Ilustración 40.</i>	74
<i>Ilustración 41.</i>	75
<i>Ilustración 42.</i>	75
<i>Ilustración 43.</i>	76
<i>Ilustración 44.</i>	78
<i>Ilustración 45.</i>	79
<i>Ilustración 46.</i>	83

Ilustración 47.

84

Ilustración 48.

86

Ilustración 49.

87

Lista de Tablas

<i>Tabla 1.</i>	53
<i>Tabla 2.</i>	53
<i>Tabla 3.</i>	54
<i>Tabla 4.</i>	55
Tabla 5.	55
<i>Tabla 6.</i>	56

Planteamiento del problema

En el modelo empresarial se le asigna a un departamento el manejo de datos relevantes con el cual se prevé obtener métricas de diferentes parámetros que generan información valiosa para la toma de decisiones y conocer el estado actual de la empresa, esta situación obliga a una ardua capacitación en cuanto conocimiento sobre manejo de bases de datos, de lo cual se desglosan problemas como, el tiempo de aprendizaje, la necesidad de conocimientos previos sobre informática, la sintaxis SQL y la lógica que conlleva.

Además, existen conceptos como el pool de conexiones que gestiona las sesiones para el acceso de la base de datos, estas son limitadas y un mal manejo de las mismas termina en un pool sin la capacidad de atender solicitudes nuevas, convirtiéndose en un bloqueo de acceso hasta no terminar o cerrar sesiones abiertas.

Por otro lado, una vez se obtienen datos tratados, estos aún no representan por sí mismos información valiosa, dado que necesitan un manejo sobre un contexto para que tengan sentido, tal como puede ser crear gráficas las cuales muestran lo que normalmente los datos en bruto no reflejan, lo que conlleva a una exportación de datos para luego ser tratados, siendo en algunos casos imposible hacer un tratado de los mismos en SQL.

Justificación

Considerando el problema expuesto anteriormente se crea la necesidad de una solución que permita disminuir la complejidad de escribir consultas SQL de manera significativa y teniendo en cuenta la flexibilidad de generar las mismas ya que está orientado a personas sin conocimiento en este campo, además, junto con una solución centralizada se debe tener en cuenta el uso adecuado de las conexiones a las bases de datos con el fin de tener disponibilidad y evitar saturar el pool de conexiones, la solución debe ser sencilla de usar, siendo de vital importancia dar una experiencia de usuario donde sin importar el nivel de conocimientos se pueda entender, como en el caso particular de Excel, el cual es un entorno que permite su uso independientemente del nivel de documentación del sujeto.

Por lo tanto, el objetivo principal del diseño de los microservicios, es generar una herramienta no-code de consultas a bases de datos postgres orientado principalmente en la interfaz gráfica con la finalidad de crear reportes, teniendo en cuenta que el público objetivo de las herramientas no-code es principalmente usuarios con pocos o nulos conocimientos de programación, por ello el énfasis sobre la experiencia de usuario con la interfaz gráfica, además dicho conjunto de microservicios serán fundamentales para mantener un rendimiento óptimo considerando que tienen la particularidad de escalar de manera individual y siendo cada uno de ellos responsables de un dominio de la herramienta, para crear en general una buena experiencia de usuario.

Objetivos

General

Diseñar una serie de microservicios enfocados en la gestión de bases de datos relacionales (postgres) con el fin de crear una herramienta no-code con la capacidad de generar consultas mediante una interfaz gráfica sin necesidad de escribir SQL y ejecutarlas para obtener reportes.

Específicos

- Construir el estado del arte de trabajos similares en cuanto a la gestión de base de datos por medio de constructores de consultas SQL.
- Analizar e identificar los requerimientos básicos para la generación de consultas por medio de una interfaz gráfica dirigida a un usuario sin conocimientos en SQL en el contexto de herramientas no-code.
- Diseñar la infraestructura de los microservicios para la gestión de las bases de datos incluyendo el stack tecnológico.

Alcances y limitaciones

Alcances

1. Ser capaz de crear integraciones de las bases de datos en la herramienta no-code con bases de datos SQL Postgres.
2. Generar consultas SQL sin necesidad de usar Native Queries ni la estructura característica de la misma.
3. Una arquitectura y un stack tecnológico que sea escalable.

Limitaciones

1. Solo será compatible con bases de datos Postgres.
2. Funcionará solo con bases de datos expuestas a internet, no a redes privadas.

Marco Teórico

Arquitectura de Software

El concepto de arquitectura de software suele ser difícil de entender dado que el significado de "software" es abstracto, a diferencia del hardware o cualquier otro objeto que sea tangible, en donde se puede ver cada uno de sus componentes. Ahora cuando se habla de arquitectura es más sencillo debido a que es una palabra del común que se puede entender como el diseño de una estructura que une todas las piezas para que una construcción pueda ser llevada a cabo, al unir estas dos palabras como concepto puede llegar a ser confuso ya que se entiende como el diseño de algo abstracto, este término surgió en los años 60 y fue ganando popularidad hasta que en la década de los 80 se integró en el desarrollo de software, según el Instituto de Ingeniería de Software se define como:

“La arquitectura de software de un sistema representa las decisiones de diseño relacionadas con la estructura y el comportamiento general del sistema. La arquitectura ayuda a las partes interesadas a comprender y analizar cómo el sistema logrará cualidades esenciales como modificabilidad, disponibilidad y seguridad.” (Software Architecture | Software Engineering Institute, n.d.)

Entonces dada la definición del Instituto de Ingeniería de Software se puede decir que la arquitectura de software es el diseño de desarrollo para analizar si un enfoque producirá una solución aceptable a todos los requerimientos, ya sea para un sistema nuevo o un sistema heredado, además de la identificación temprana de riesgos de diseño.

Patrones de diseño

Cuando se habla de arquitectura de software es inherente hablar sobre los patrones de diseño, gracias a que a la hora de diseñar un arquitectura de software hay dos formas de afrontar dicho reto, uno es creando desde cero la solución, y la otra es buscar soluciones ya propuestas por otras personas que sean útiles para resolver nuestro problema, en el caso de buscar soluciones propuestas se encuentran: los patrones de diseño que son una forma reutilizable de resolver problemas comunes a nivel de código y después de dichos patrones de diseño se tienen también los patrones de arquitectura.

Patrones de arquitectura

Los patrones arquitectónicos son aquellos que tienen un nivel de abstracción mayor con respecto a los patrones de diseño en la arquitectura del software debido a que no especifican cómo se escribirá el código, pero sí cómo debe estar organizado y tienen en cuenta los elementos que serán importantes para aplicación, son problemas comunes con soluciones reutilizables, los más populares en la actualidad son:

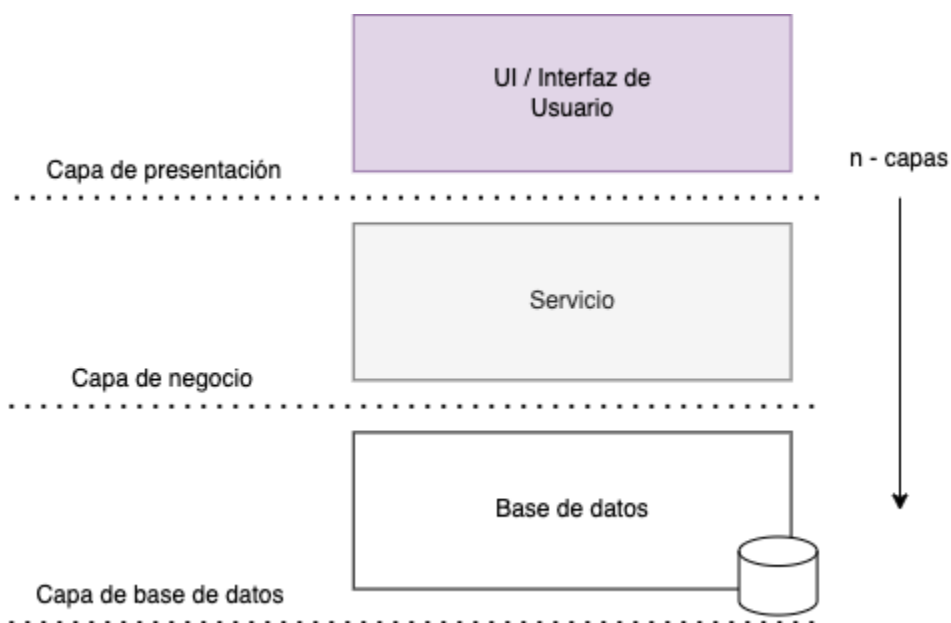
Patrón por capas

Es un patrón comúnmente usado para soluciones las cuales se descomponen en un flujo de procesos donde cada capa tiene un rol en la aplicación junto a su responsabilidad, estas capas se dividen de manera horizontal pueden ser tantas como se requieran y están conectadas entre ellas, la implementación más común de este patrón es dividirlo en 3 capas las cuales son:

- Capa de presentación (encargada de la interfaz gráfica del usuario).
- Capa de negocio (donde viven las reglas y lógica de negocio para el procesamiento de datos).
- Capa de base de datos (se ocupa de la persistencia de datos).

Ilustración 1.

Diagrama de ejemplo de un patrón por capas

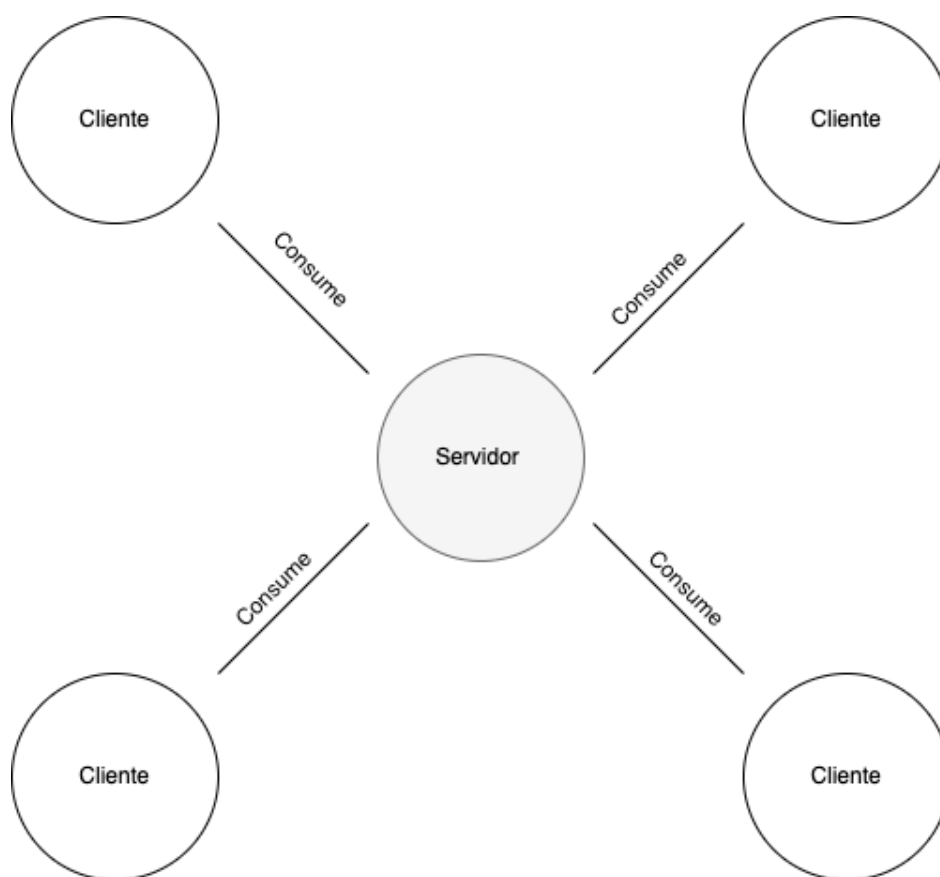


Patrón cliente-servidor

Siendo uno de los patrones más conocidos compuesto por dos componentes, un servidor y los clientes que se conectan al mismo para hacer solicitudes. En esta arquitectura, el Cliente y el Servidor son desarrollados como dos aplicaciones diferentes, de tal forma que cada una puede ser desarrollada de forma independiente, dando como resultado dos aplicaciones separadas, las cuales pueden ser construidas en tecnologías diferentes, pero siempre respetando el mismo protocolo de comunicación para establecer comunicación.

Ilustración 2.

Diagrama de ejemplo de un patrón cliente-servidor



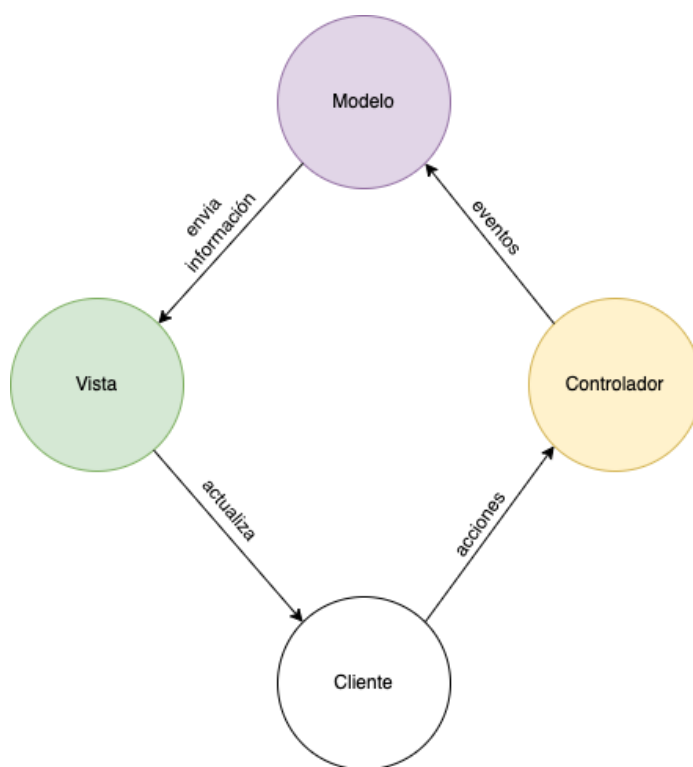
Patrón modelo-vista-controlador (MVC)

En este patrón la aplicación se divide en 3 componentes fundamentales:

- Modelo: Encargado de los datos de la aplicación y el acceso a ellos, así como también es el responsable de la lógica de negocio.
- Vista: En este componente se encuentra la interfaz del usuario (UI), en donde se obtienen datos enviados directamente por el controlador y representados de forma que el usuario pueda usar.
- Controlador: Asume un rol de escuchar eventos que se ejecutan por acciones del usuario para enviar órdenes hacia el modelo.

Ilustración 3.

Diagrama de ejemplo de un patrón modelo-vista-controlador



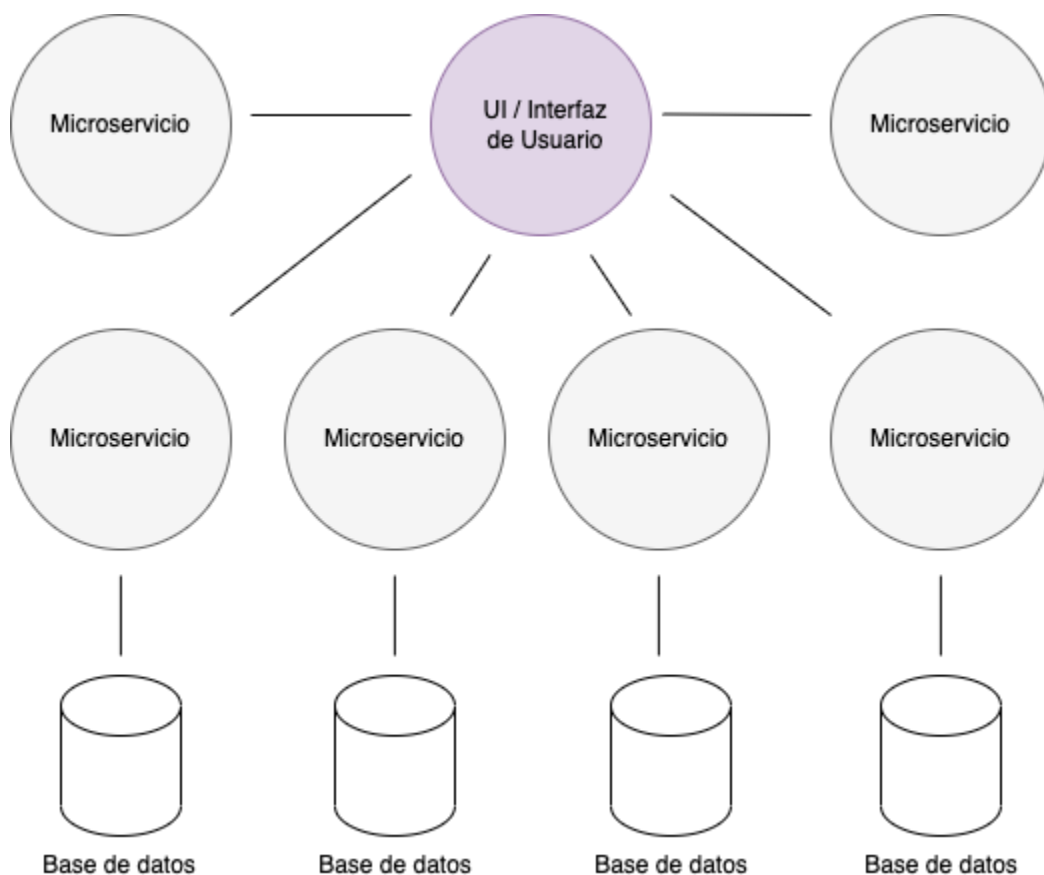
Patrón orientado a microservicios

Es un estilo de patrón de arquitectura en el cual se divide en una serie de pequeños servicios, que luego se ejecutan de manera independiente, por ello pueden estar escritos en cualquier lenguaje de programación y su código puede ser implementado sin que afecte a los demás. Por otro lado, tienen la característica de comunicarse entre ellos, por ejemplo, a través de peticiones HTTP por medio de API, cada microservicio se corresponde con un área de la aplicación. En el libro “Building Microservices” caracterizan a un microservicio como

“algo que a nivel de código podría ser reescrito en dos semanas” (Newman, 2015, 280).

Ilustración 4.

Diagrama de ejemplo de un patrón orientado a microservicios.



Lenguajes de programación

Kotlin

“Es un nuevo lenguaje de programación dirigido a la plataforma Java. Kotlin es conciso, seguro, pragmático y se centra en la interoperabilidad con el código Java. Puede ser usado en casi todos los lugares donde se usa Java hoy en día: para el desarrollo del lado del servidor, Android aplicaciones y mucho más. Funciona muy bien con todas las bibliotecas Java existentes y frameworks y se ejecuta con el mismo nivel de rendimiento que Java.” (Kotlinlang.org, s.f.)

Ventajas:

- Tiene una fácil curva de aprendizaje gracias a su sintaxis sencilla.
- Orientado a objetos y funcional.
- Compatibilidad con librerías de java siendo un lenguaje con una larga trayectoria.

Desventajas:

- Usa JVM que es una de las razones por las cuales java consume muchos recursos en comparación con otros lenguajes de programación.

Go

“Go es un lenguaje de programación expresivo, conciso, limpio y eficiente. Sus mecanismos de concurrencia facilitan la escritura de programas que aprovechan al máximo las máquinas multinúcleo y en red, mientras que su novedoso sistema de tipos permite la construcción de programas modulares y flexibles. Go compila rápidamente en código de máquina, pero tiene la comodidad de la recolección de elementos no utilizados (garbage collector) y el poder de la reflexión en tiempo de ejecución (run-time reflection). Es un lenguaje

compilado rápido, tipificado estáticamente que se siente como un lenguaje interpretado y tipificado dinámicamente”. (Go.dev, s.f.)

Ventajas:

- Es muy rápido y eficiente con la memoria, tiene un rendimiento parecido a C.
- Soporte de concurrencia implementado de forma sencilla.
- Librerías estándar sobre todo para HTTP.

Desventajas:

- Orientado a objetos (parcialmente).
- Manejo de errores, se tiene que validar en cada función que pueda tenerlo y validación constantemente hace que el código sea más extenso.

Javascript

“JavaScript (JS) es un lenguaje de programación ligero, interpretado o compilado justo a tiempo con funciones de primera clase. Si bien es más conocido como el lenguaje de secuencias de comandos para páginas web, muchos entornos que no son de navegador también lo utilizan, como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje dinámico basado en prototipos, multiparadigma, de un solo subproceso, que admite estilos orientados a objetos, imperativos y declarativos (por ejemplo, programación funcional).” (Developer.mozilla.org, s.f.)

Ventajas:

- Es rápido, ligero, simple y popular.

- Versátil ya que es un lenguaje que puede ser usado en múltiples aplicaciones no solo en el navegador, puedes construir una aplicación de principio a fin solo con el.

Desventajas:

- No es compatible en todos los navegadores de manera uniforme.
- Solo usa un hilo de ejecución, especialmente problemático con implementaciones backend en términos de carga.

Frameworks y Librerías

Spring boot

“Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación. Un elemento clave de Spring es el soporte de infraestructura a nivel de aplicación: Spring se enfoca en la "plomería" de las aplicaciones empresariales para que los equipos puedan enfocarse en la lógica de negocios a nivel de aplicación, sin ataduras innecesarias a entornos de implementación específicos.” (Spring.io, s.f.)

Características:

- Estable, popular y sencillo.
- Compatible con kotlin.
- Se configura automáticamente y también configura bibliotecas de terceros siempre que sean compatibles.

Echo

"Echo es un framework para Go orientado a la creación de API RESTful de alto rendimiento, siendo extensible y minimalista, contando con soporte a HTTP/2, además de tener un enrutador con asignación de memoria dinámica cero que prioriza rutas de manera inteligente." (Echo.labstack.com, s.f.)

Características:

- Ligero, y muy rápido
- Escalable.

Node.js

“Node.js es un entorno de tiempo de ejecución de JavaScript multiplataforma y de código abierto que se ejecuta en un solo proceso, sin crear un nuevo hilo para cada solicitud. Node.js proporciona un conjunto de primitivas de E/S asíncronas en su biblioteca estándar que evitan que el código JavaScript se bloquee.” (Node.js.dev, s.f.)

Express

Framework de aplicaciones web back-end para crear API RESTful con Node.js rápido y popular, diseñado para tener una curva de aprendizaje sencilla y ahorrar muchas líneas de código gracias a su simplicidad en su implementación.

Características:

- Código abierto con una gran comunidad activa.
- Sencillo de implementar.
- Versátil y compatible con muchas librerías de terceros gracias a su popularidad.

Knex.js

Una de las librerías más populares en JavaScript para generar consultas SQL para PostgreSQL, Cockroach DB, MSSQL, MySQL, MariaDB, SQLite3, Better-SQLite3, Oracle y Amazon Redshift diseñado para ser flexible, portátil y divertido de usar.

React

Librería de JavaScript diseñada para construir interfaces gráficas de usuarios basadas en componentes reutilizables.

Next.js

Framework de JavaScript construido sobre React, facilitando la configuración y utilización de librerías complementarias para React, y optimiza varias características, algunas de ellas son el SSR (Server Side Rendering) que es la posibilidad de renderizar el HTML del lado del servidor, rutas dinámicas sin necesidad de volver a cargar la página desde 0 entre otros.

Bases de datos

PostgreSQL

“PostgreSQL es un poderoso sistema de base de datos relacional de objetos de código abierto con más de 35 años de desarrollo activo que le ha valido una sólida reputación por su confiabilidad, robustez de funciones y rendimiento.” (Postgresql.org, s.f.)

Otras herramientas

Docker

“Docker es un conjunto de productos de plataforma como servicio que utilizan la virtualización a nivel del sistema operativo para entregar software en paquetes denominados contenedores.” (Contenedores de Docker | ¿Qué es Docker? | AWS, s. f.)

Kubernetes

“Kubernetes es un sistema de orquestación de contenedores de código abierto para automatizar la implementación, el escalado y la administración de software.” (¿Qué es Kubernetes?, 2022)

Herramientas no-code

El término “no-code” hace referencia a herramientas y plataformas que permiten crear y ejecutar software sin necesidad de codificar.

Ejemplos de algunas herramientas/plataformas:

- Webflow: herramienta no-code diseñada para la creación de sitios web responsivos, a partir de una interfaz gráfica con numerosas opciones de configuración para cada componente.
- Bubble: herramienta no-code orientada al desarrollo de web apps implementando una gran variedad de opciones para crear funcionalidades complejas, acceso a bases de datos y todo a través de una interfaz gráfica drag and drop (arrastra y suelta).

Estado del arte

Investigaciones internacionales

- **(Ochoa Paz, Luis Alberto, 2010) Desarrollo de una interfaz gráfica para la creación de sentencias SQL.** “Este trabajo trata sobre la creación de un proyecto de software libre, con la finalidad de crear una aplicación que sea capaz de guiar de forma gráfica, rápida e intuitiva al usuario a través del proceso de creación de consultas SQL para la base de datos postgresSQL.”

El enfoque del mismo es más estructural con el fin de presentar la información necesaria para realizar consultas, como por ejemplo nombres de las tablas existentes, de una forma amigable que les evitará tener que memorizar la estructura de un modelo de datos, para poder posteriormente consultarlo, además de guiarlos de forma correcta entre los elementos constitutivos del lenguaje SQL, lo cual evitará que hagan uso de conceptos erróneos de su estructura sintáctica y semántica.

- **(Tony Russell-Rose, Shokrane, Farhad, 2020) Designing the Structured Search Experience: Rethinking the Query-Builder Paradigm.** “En este artículo, se presenta un nuevo enfoque en el que los usuarios expresan conceptos como objetos en un lienzo visual y los manipulan para articular relaciones. Esto ofrece una experiencia de usuario (UX) más intuitiva que elimina muchas fuentes de error, hace que la semántica de consulta sea más transparente y ofrece nuevas formas de colaborar, compartir y optimizar las estrategias de búsqueda y las mejores prácticas.” En el presente trabajo se habla de la importancia que aporta

que profesionales de diferentes áreas tengan la facilidad de realizar búsquedas de una forma clara y precisa.

- **(Vera-Rivera, Astudillo y Gaona, 2019) Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación** “Los microservicios son un enfoque arquitectónico y organizativo del desarrollo de software en el que las aplicaciones están compuestas por pequeños servicios independientes que se comunican a través de un API bien definida y protocolos ligeros. En este trabajo se realizó una revisión de literatura para identificar los desafíos y tendencias de investigación presentes en las fases del proceso de desarrollo y en la gestión de atributos de calidad de este tipo de aplicaciones.” Investigación a la arquitectura orientada a microservicios ya que es una tendencia popular que se ha visto en los últimos años que consiste en muchos servicios independientes cada uno es un contenedor que puede tener réplicas, las cuales ofrecen una alta disponibilidad y son muy útiles para solventar problemas relacionados con un alto tráfico de usuarios y de datos al distribuir la carga en diferentes réplicas y ser fácilmente escalables.
- **(Roig Hervás, 2021) Tecnologías Low-Code y No-Code: Un caso práctico para estudiar su potencial y limitaciones.** “Las tecnologías Low-Code y No-Code constituyen una de las propuestas más interesantes en el ámbito del desarrollo de software. Se han hecho populares por la promesa de que, con ellas, podrías realizar un proyecto software sin necesidad de conocimiento técnico y en un tiempo reducido.” En este análisis se tiene como principal objetivo analizar dichas tecnologías con el fin de descubrir sus ventajas y desventajas. Para ello se

llevan a cabo unos retos en los que se intenta desarrollar una serie de funcionalidades comunes y específicas que se presentan en el desarrollo de software, donde se encuentran que como todo este tipo de herramientas tiene limitaciones y en la búsqueda de facilitar conceptos e implementaciones siempre tiene como costo final un menor control sobre lo que se quiere realizar, en el mismo se argumenta que si bien esto es una desventaja en este tipo de tecnología solo se evidencia cuando la complejidad de un proyecto es alta, pero resulta muy eficaz para los desarrollos más comunes.

Investigaciones nacionales

- **(Ortiz Montoya, 2021) Microservicio para identificar preferencias de clientes en los canales digitales de Bancolombia.** “En este trabajo de grado estudia cómo la eficiencia de gasto y experiencia del cliente en canales digitales, en este caso Bancolombia, puede mejorar mediante servicios en arquitecturas basadas en microservicios, donde se puede centralizar todo el tráfico de diferentes fuentes tales como aplicaciones móviles y aplicaciones web para luego ser expuestas en una API, permitiendo así un consumo sencillo y ágil del servicio.”

Herramientas

En lo que respecta a herramientas no code se encuentran dos vertientes que se clasifican en no code y low code, en ambas se mantiene la premisa de desarrollar software de una manera sencilla y eficiente, una de las más grandes diferencias entre ellas es que las herramientas low code son diseñadas para tener un nivel de complejidad más alto y en algunas de ellas te permiten codificar trozos de pseudocódigo o código y requieren una mayor comprensión de conocimiento técnico, mientras que las no code están diseñadas para resolver problemas más generales y menos específicos, en otras palabras apuntan a diferentes públicos aunque haya similitudes entre ellas. (Meijers Jesse, 2018).

Herramientas no-code sobre base de datos

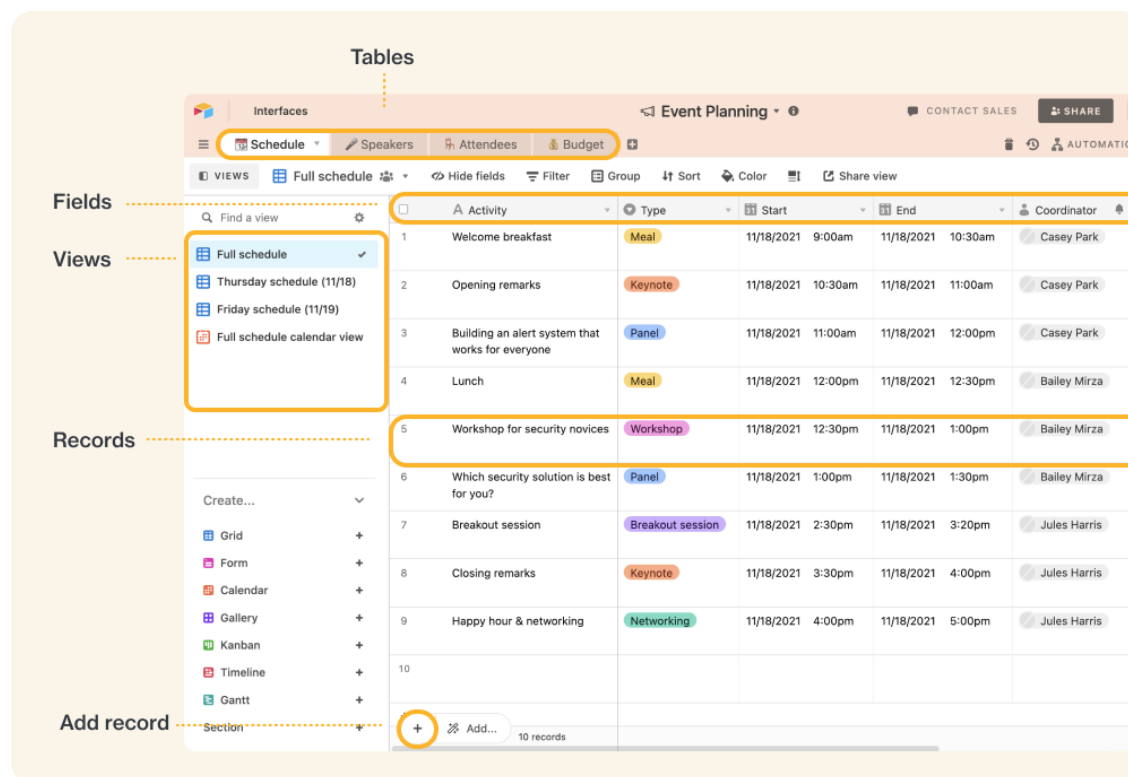
Airtable: “Es una plataforma que facilita la creación de aplicaciones potentes y personalizadas usando un query builder sencillo, pero con variedad de opciones. Estas herramientas pueden optimizar casi cualquier proceso, flujo de trabajo o proyecto y lo mejor de todo, puede crearlas sin tener que aprender a escribir una sola línea de código.” (Airtable.com, s.f.)

En esta herramienta se hace referencia al query builder cómo visitas (views) un concepto heredado de las hojas de cálculo (spreadsheet)

“En una hoja de cálculo tradicional, todos los usuarios ven el contenido de sus hojas de la misma manera; si desea ocultar o congelar columnas, filtrar, ocultar campos y agrupar dicha información.” (Airtable.com, s.f.)

Ilustración 5.

Airtable, vista general del query builder de la herramienta. (Airtable.com, s.f.)



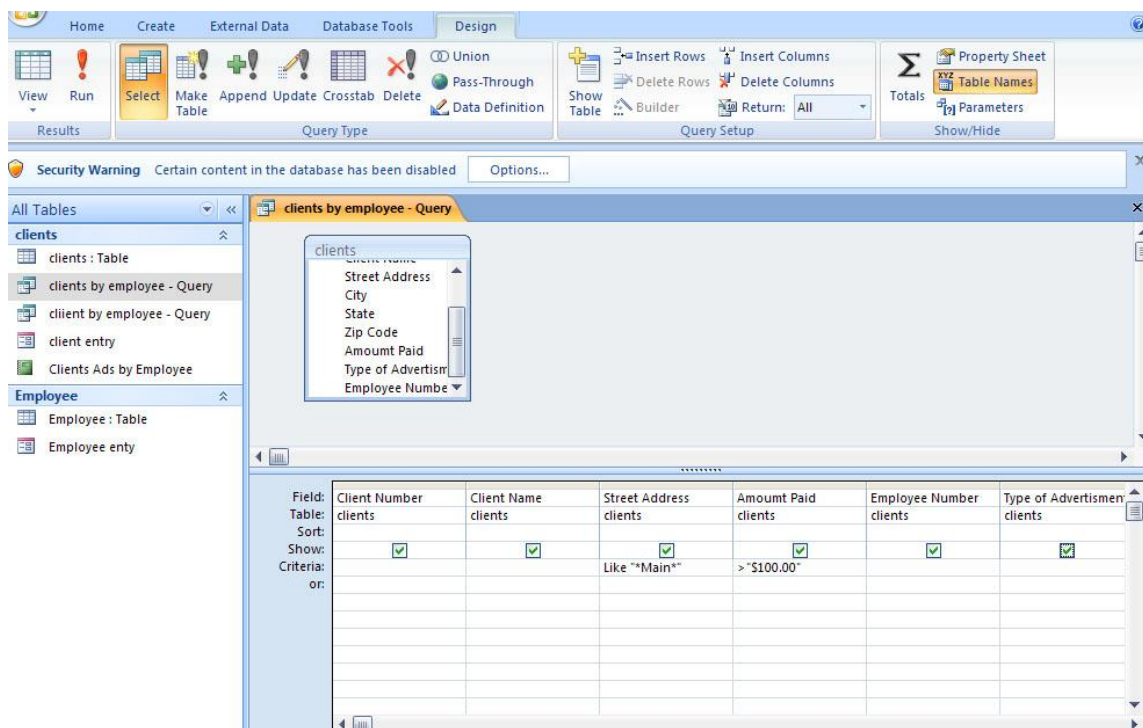
Herramientas low-code sobre de datos

Access: “Es una herramienta fácil de usar para crear aplicaciones comerciales, desde plantillas o desde cero. Con sus herramientas de diseño completas e intuitivas, Access puede ayudarte a crear aplicaciones atractivas y altamente funcionales en una cantidad mínima de tiempo.” (Microsoft.com. s.f.)

En esta herramienta el query builder usa la nomenclatura de SQL para hacer referencia a todo lo que conforma una query como es select, from, where, entre otros más haciendo requerido tener un conocimiento acerca de lenguaje de consultas para usarlo a través de la interfaz gráfica y además de ofrecer la posibilidad de ejecutar SQL nativo.

Ilustración 6.

Access, vista general del query builder de la herramienta. (Wikibooks, 2009)



Análisis de requerimientos

Teniendo en cuenta las herramientas mencionadas en el estado de arte y las dos vertientes de no-code donde se concluyó que dichas herramientas son orientadas para públicos diferentes, además de afrontar el mismo problema puntual de consultar información a una base de datos mediante una interfaz gráfica desde dos perspectivas distintas con el fin de crear una propuesta de software tomando en consideración las dos partes, se usa la técnica de levantamiento de los requerimientos en este proyecto por medio del análisis de documentación de ambas herramientas, que consiste en obtener la información sobre los requerimientos funcionales y requerimientos no funcionales de software con la ayuda de la documentación de un producto que ya está elaborado.

Comparativa entre herramientas

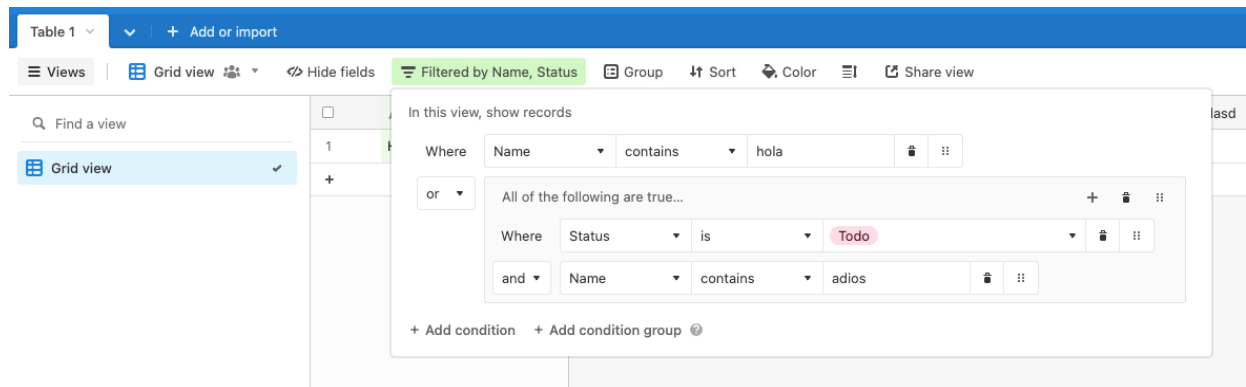
Considerando que los requerimientos serán levantados usando el análisis de documentación es pertinente hacer una comparativa entre las herramientas mencionadas en el marco teórico para tener presente cada perspectiva de las mismas en ciertos aspectos específicos.

Query builder

Este es el aspecto más diferente de ambas herramientas, en el caso de airtable se tiene un concepto más orientado a una hoja de cálculo teniendo este aspecto característico además de usar una nomenclatura de fields (campos) siendo estas las columnas, filters (filtros) siendo estos los condicionales del where.

Ilustración 7.

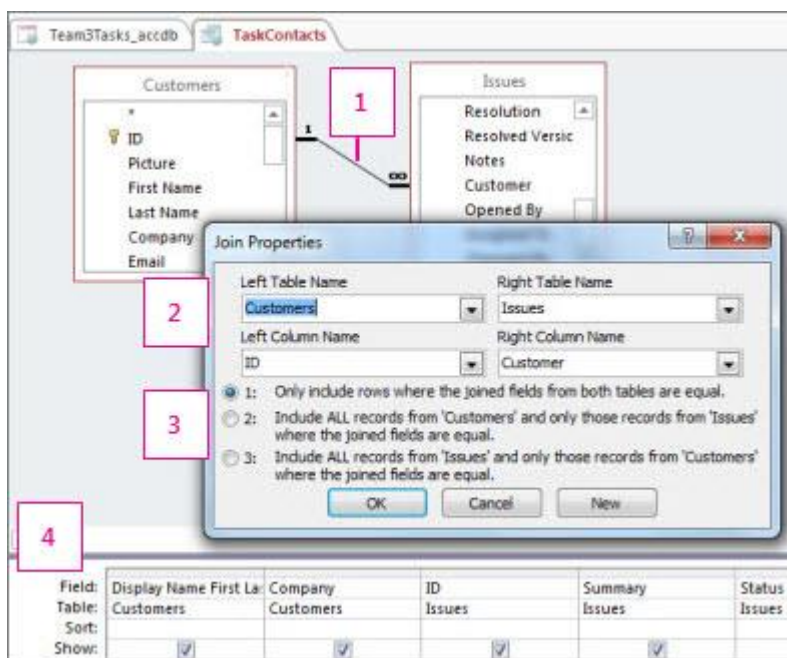
Airtable, toolbar con opciones Hide fields, Filters, Group, Sort para filtrar información.



Se usan palabras mucho más descriptivas para un usuario con pocos o nulos conocimientos sobre SQL en el lado de access se tiene una nomenclatura clásica de un lenguaje de consultas.

Ilustración 9.

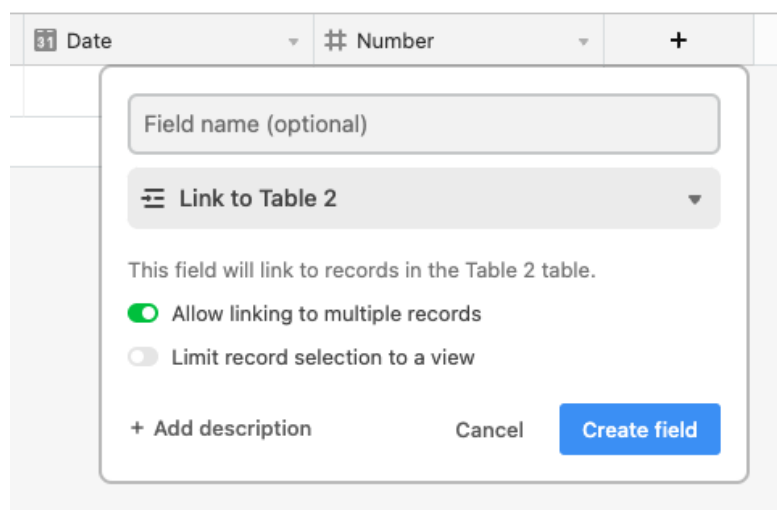
Access, joins entre tablas. (Microsoft, s.f.)



Por otro lado, se tiene airtable que implementa el join de tablas a partir de una nueva columna que hace un vínculo a la siguiente vista de la tabla referenciada a partir de la llave primaria de manera automática sin posibilidad de modificarla.

Ilustración 10.

Airtable, joins entre tablas.



En vista de ambas implementaciones para el caso de Access se tiene más control para poder unir información entre tablas, aunque abre la puerta a problemas conocidos como son Cross joins de manera involuntaria, y en el caso de airtable se tiene un join más controlado pero sin posibilidades de modificar la condición del join crea limitaciones, por lo cual una implementación del lado de Access sería más adecuada para este proyecto, teniendo en cuenta tener una lista de joins predeterminados basados en el esquema de la base de datos (tablas, columnas, tipos de datos, primary key, foreign key).

Reportes

En este aspecto se encuentran soluciones en ambas herramientas con una gran cantidad de similitudes basadas en queries y siendo representadas por gráficas que usan la información de las columnas y funciones agregativas.

Requerimientos funcionales

En este tipo de requerimientos se encuentra todo aquello que define el comportamiento específico del sistema, que se definen a continuación basado en la comparativa:

- Registrarse e iniciar sesión en la herramienta.
- Crear una integración de su base de datos con la plataforma.
- Consultar las integraciones existentes con sus bases de datos.
- Generar consultas de información mediante las integraciones con la base de datos.
- Consultar una lista de joins predeterminados basados en el esquema de la base de datos.
- Generar reportes con gráficas a partir de las consultas.

Requerimientos no funcionales

En este tipo de requerimientos se representan características y/o limitaciones del sistema, que se definen a continuación basado en la comparativa:

- La interfaz gráfica del query builder debe tener una buena usabilidad.
- Soporte solo a bases de datos PostgreSQL.

Justificación de solución propuesta

Arquitectura de microservicios

La arquitectura de microservicios se basa principalmente en una serie de pequeños servicios, que luego se ejecutan de manera independiente, por ello pueden estar escritos en cualquier lenguaje de programación y se comunican normalmente por API REST.

Ventajas

- Es una arquitectura muy escalable a nivel individual de servicios o en general.
- Al ser independientes se pueden implementar en cualquier lenguaje de programación, por lo cual se pueden optimizar para tareas específicas.
- Es más sencillo organizar los grupos de trabajo por microservicios o dominio de la aplicación.
- Los despliegues son individuales permitiendo deshacerse de dependencias entre equipos.

Desventajas

- Mayor consumo de recursos, por consecuencia de que cada servicio es una instancia que consume recursos por lo cual suele ser una arquitectura costosa.
- Cuando se tienen una gran variedad de servicios transportar información es complicado ya que se debe garantizar que todos los servicios tienen las firmas sincronizadas, por lo cual un cambio de las mismas suele ser difícil de gestionar.
- La división de una aplicación en microservicios suele llevar muchas más horas de planificación que en otros tipos de arquitectura.

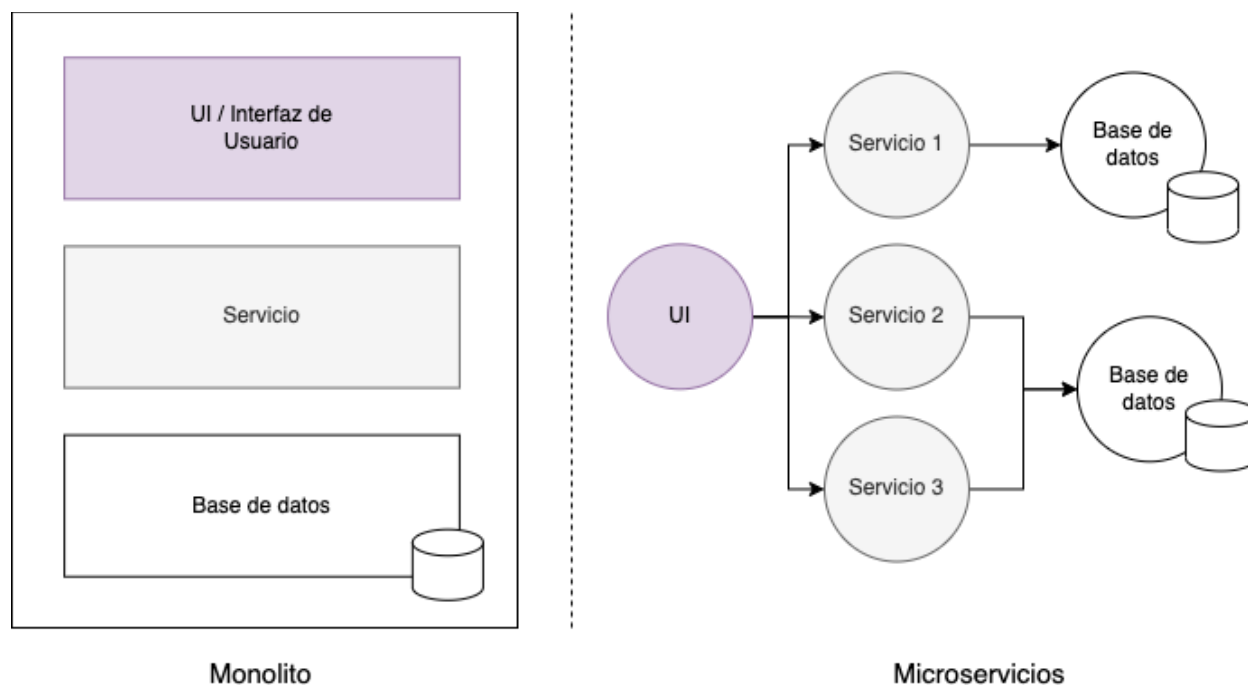
Comparativa entre la arquitectura de microservicios y monolítica

“Una arquitectura monolítica es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones.”

(Chandler Harris, s.f.)

Ilustración 11.

Comparativa entre la arquitectura de microservicios y monolítica.



Ventajas de una arquitectura monolítica contra una arquitectura de microservicios

- El desarrollo es más sencillo ya que solo se tiene una aplicación.
- El rendimiento, al ser una aplicación con una API que brinda el mismo servicio que muchas APIs como es el caso de los microservicios.
- El consumo de recursos se reduce mucho en comparación de los microservicios, por lo cual es una arquitectura muy barata.

Desventajas de una arquitectura monolítica contra una arquitectura de microservicios

- La velocidad de desarrollo se reduce exponencialmente a medida que el monolito crece, el desarrollo se vuelve más complejo y lento.
- La disponibilidad es vital en cualquier aplicación, en el caso de los monolitos cualquier error en algún módulo puede afectar este aspecto.
- La escalabilidad al estar todo dentro de un mismo desarrollo no se puede escalar individualmente, por lo cual se escala de manera vertical, generando un costo elevado y limitando lo escalable que puede ser en un futuro.
- La flexibilidad de un monolito está limitada por las tecnologías que se utilizan en él.

Considerando ambos tipos de arquitecturas, la arquitectura de microservicios se acopla mejor a este proyecto por ser una aplicación compleja que busca ser escalable y aprovechar la compatibilidad de diferentes lenguajes de programación para hacer más sencillo el desarrollo en conjunto.

Diseño de microservicios

Para diseñar la arquitectura de microservicios del proyecto se usó el Diseño dirigido por dominio (DDD) el cual consiste en proporcionar un framework que tiene como finalidad construir un conjunto de microservicios bien diseñados.

El DDD tiene dos fases distintas, estratégica y táctica. En el DDD estratégico, está definiendo la estructura a gran escala del sistema y ayuda a garantizar que su arquitectura permanezca enfocada en las capacidades de negocio. El DDD táctico proporciona un conjunto de patrones de diseño para crear un modelo de dominio en ellos se encuentran entidades, colecciones y servicios de dominio. (Domain analysis for microservices - Azure Architecture Center, 2022)

Ilustración 12.

Pasos para definir microservicios usando DDD.

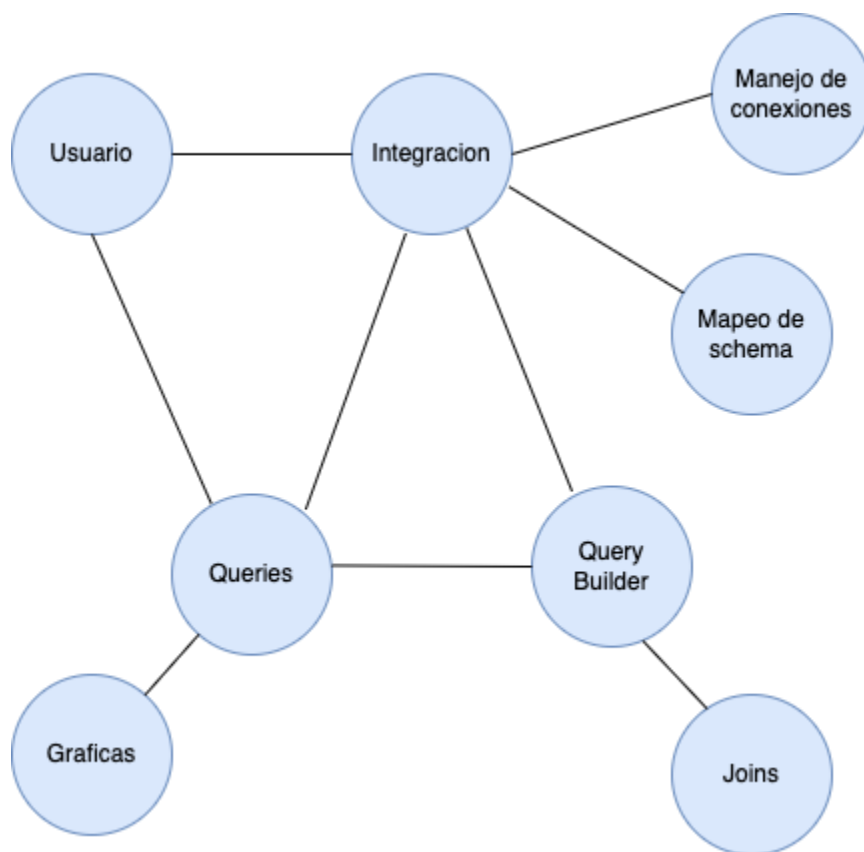


Análisis de dominio

En un comienzo mediante un análisis inicial de dominio se mapean todas las funciones de negocio y la manera en cómo se conectan, no es necesario usar ningún formalismo para diseñar el diagrama.

Ilustración 13.

Diagrama de análisis inicial de dominio



- **Usuario:** permite administrar la información del usuario y se encarga de las sesiones del usuario.
- **Integración:** es la funcionalidad que persiste y maneja la información referente a las bases de datos de los usuarios.
- **Manejo de conexiones:** se encarga de mantener conexiones activas para comunicarse con las bases de datos de las integraciones.
- **Mapeo de schema:** escanea la base de datos del usuario en busca de conocer la estructura de las tablas y las relaciones que tienen entre ellas.

- **Queries:** administra las queries guardadas por el usuario junto a la integración que pertenece generada por el query builder.
- **Query builder:** transforma el input generado del usuario con el UI (campo, filtros, grupos, orden) a SQL con el fin de ser ejecutado y obtener un resultado.
- **Joins:** genera un listado de unión de información entre tablas predeterminadas basadas en el esquema de la integración o en condiciones asignadas por el usuario.
- **Gráficas:** genera reportes con gráficas basado en la estructura de la query generada por el usuario.

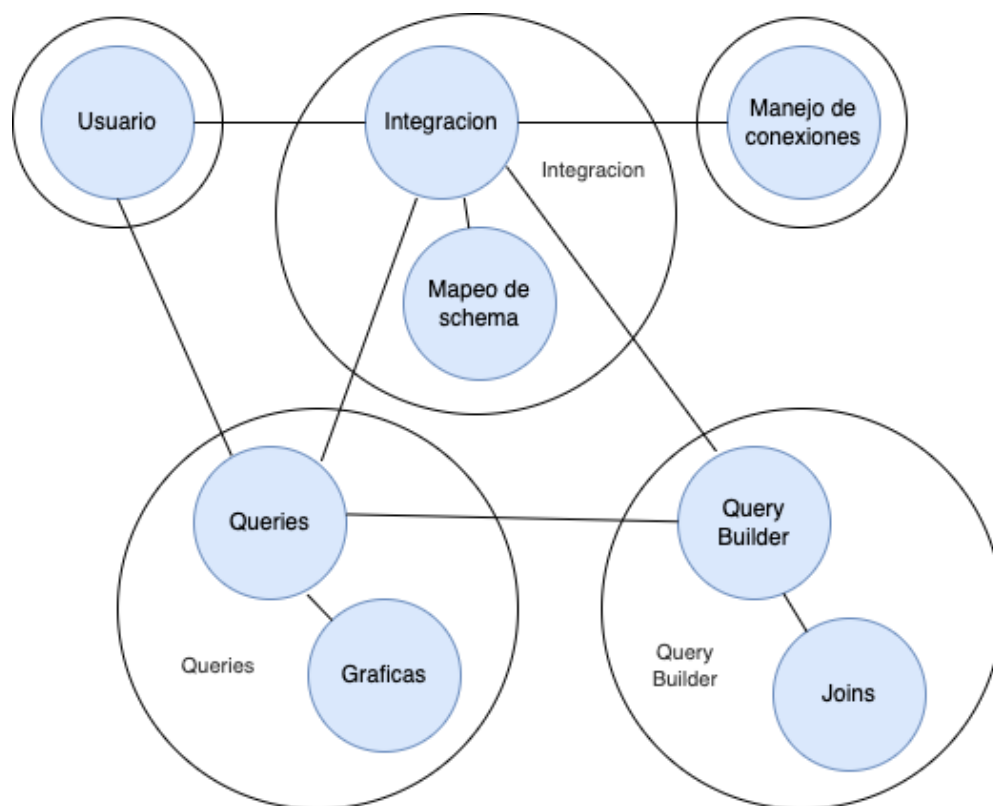
Definición de contexto acotado

En este punto de DDD aún no se ha definido nada sobre la implementación o las tecnologías que se usarán, pero sí un contexto inicial de la aplicación, en el siguiente paso que se realiza es definir los contextos acotados. Para generar dichos contextos acotados se agruparán las funcionalidades del diagrama anterior que tienen dependencias directas, para generar modelos de dominios menos acoplados.

En la figura 14 se observan dependencias directas como en el caso de la integración y el mapeo de schema ya que su contexto a nivel de datos tiene relación directa por lo cual se puede asignar la responsabilidad de mapear el schema a la integración, existen otros dos casos en los cuales se presenta el mismo problema los cuales son las queries con las gráficas y el query builder con los joins.

Ilustración 14.

Diagrama de análisis de dominio con contexto acotado



Definición de entidades, agregados y servicios

Una vez construido el diagrama de análisis de dominio con contexto acotado como se muestra en la Figura 15, se concluye la fase estratégica y entra en juego la fase táctica del DDD en donde se definen los modelos de dominio con mayor precisión mediante patrones.

Dentro de los patrones se encuentran las entidades, objetos de valor, agregados, servicios de dominio y aplicación, y eventos de dominio que se definen de la siguiente manera:

Entidades: Una entidad es un objeto único. Por ejemplo, un auto que tiene una matrícula que lo identifica, un usuario con una identificación.

Objetos de valor: Son objetos los cuales no tienen una identidad única, definen los valores de un atributo. Por ejemplo: un tipo de moneda, color, fecha.

Agregados: Se define como un límite de consistencia alrededor de una o más entidades.

Exactamente una entidad en un agregado es la raíz. La búsqueda se realiza utilizando el identificador de la entidad raíz. (Domain analysis for microservices - Azure Architecture Center, 2022)

Una acotación sobre los agregados es que no son y no deben ser un reflejo de la base de datos, es un error común intentar representar las relaciones de tablas que componen un modelo de entidad-relación, incluso si existe alguna relación entre estos conceptos se recomienda modelar la relación entre los objetos de dominio y los objetos agregados manualmente sin ninguna referencia a la base de datos.

Servicios de dominio y aplicación: un servicio es un objeto que implementa lógica sin necesidad de mantener algún estado, la distinción entre servicios de dominio y de aplicación recae principalmente en el tipo de lógica que encapsula. Por ejemplo, un servicio de dominio se usa para modelar el comportamiento de las entidades que contiene, mientras que el servicio de aplicación la funcionalidad técnica como puede ser la autenticación de usuarios.

Eventos de dominio: estos eventos se usan para notificar a otras partes del sistema que se ejecute alguna acción o evento del dominio, por ende, no aplica acciones como “insertar un registro en una tabla”, hace referencia a una acción de dominio como puede ser “la orden fue enviada”.

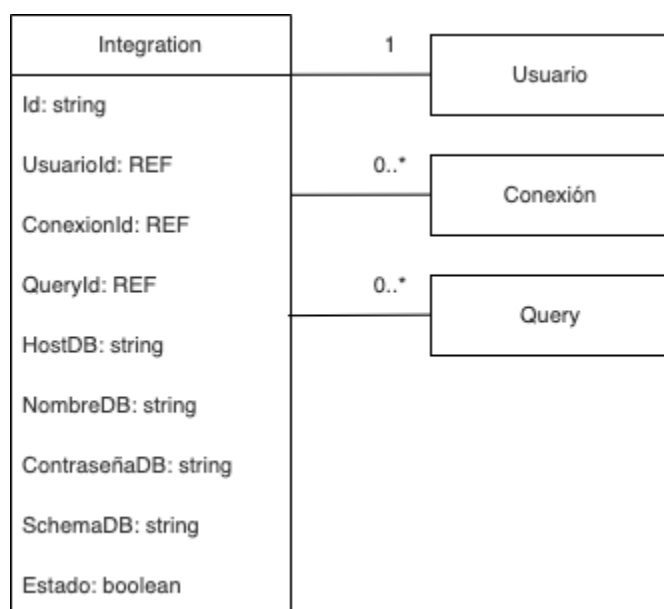
En el proyecto las entidades identificadas son:

- Integración
- Usuario
- Conexión
- Query

Representadas con sus objetos de valor y agregados en la figura 16.

Ilustración 15.

Diagrama con entidades, objetos de valor, y agregados



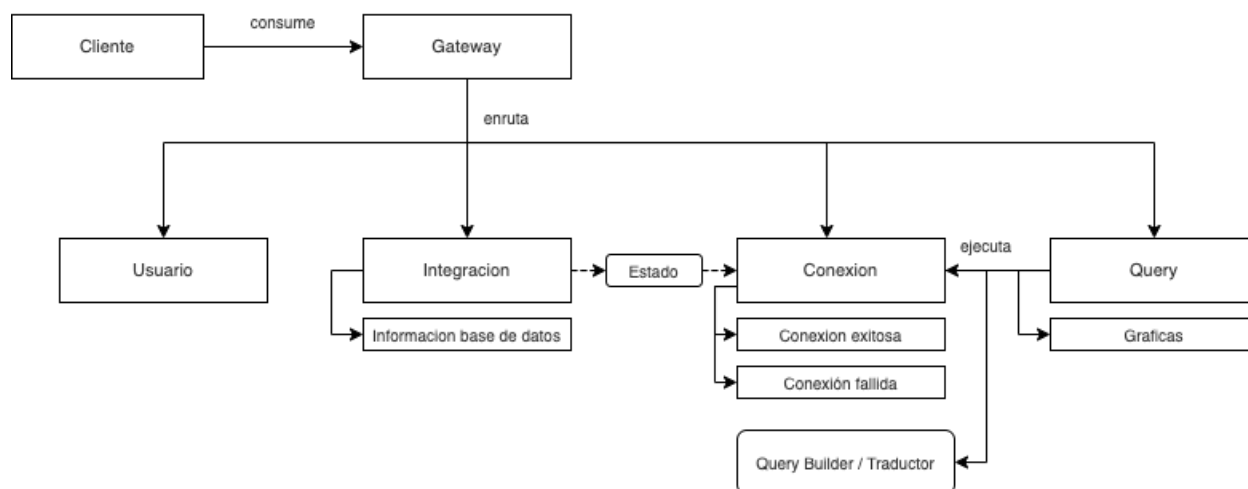
Los eventos de dominio que se definieron son los siguientes:

- Mientras se pueda establecer conexión con una integración su estado será true y en caso contrario false.
- Cuando se crea una integración y se establezca la conexión se mapeara un esquema.
- Una query puede ejecutarse siempre y cuando la conexión sea exitosa.

Además de aplicar el patrón enrutador por puerta de enlace (Gateway Routing) que permite dirigir solicitudes a varias instancias de un servicio desde un mismo endpoint, para fines de balanceo de carga y/o disponibilidad.

Ilustración 16.

Diagrama con eventos de dominio, y servicios de dominio y aplicación



Identificación de microservicios

El siguiente paso es identificar los microservicios considerando los diagramas anteriores donde se identifican los contextos y dominios, en dichos diagramas se identificaron cuatro agregados que son usuario, integración, conexión y query, y dos servicios de dominio que son gateway y query builder.

Existen pautas para identificar microservicios las cuales se realizan de diferentes maneras una de ellas es validando los contextos acotados ya que dichos microservicios no deben abarcar más de un contexto delimitado, en caso de encontrar un microservicio que combine diferentes modelos de dominio es una señal de que se debería volver a refinar el análisis de dominio, en el caso de

este proyecto el contexto acotado fue igual al resultado del modelo de dominio por lo cual se consideran buenos candidatos.

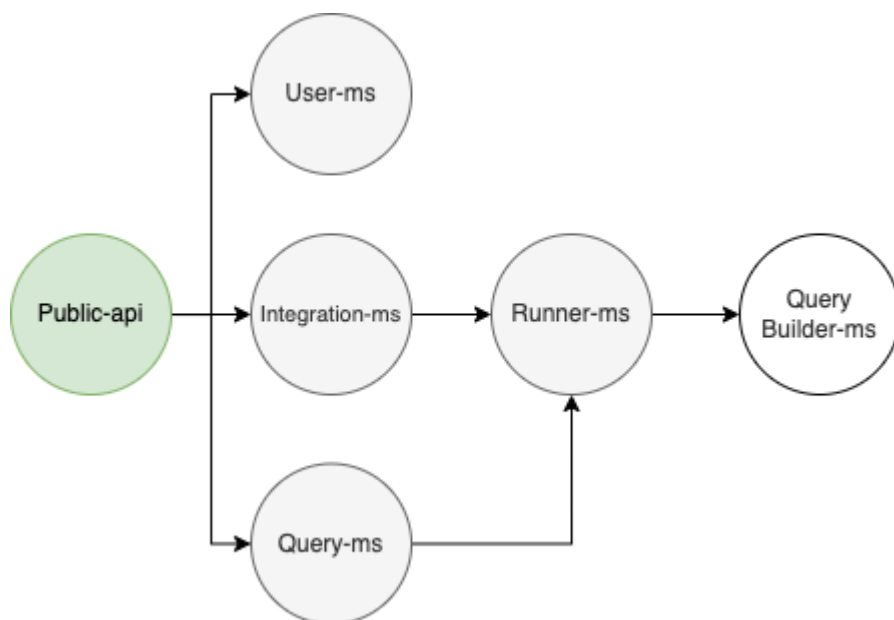
Normalmente los agregados son candidatos por defecto a ser microservicios porque derivan directamente de los requisitos de negocio y en su definición suelen estar poco acoplados. Los servicios de dominio también son buenos candidatos para microservicios por ser operaciones sin estado en múltiples agregados, suelen tener la responsabilidad de un proceso que involucra a varios microservicios. Con estas pautas se tiene un enfoque para derivar microservicios del modelo de dominio. (Identify microservice boundaries - Azure Architecture Center, s. f.)

En este proyecto para los nombres de los microservicios se definieron en inglés, por lo cual es pertinente hacer una equivalencia de los nombres con los que se usaron en el modelo de dominio:

- **Public-api:** Es un servicio de dominio llamado “Gateway” definido por el patrón enrutador por puerta de enlace.
- **User-ms:** Es el agregado llamado “Usuario” tiene el dominio de la información de los usuarios.
- **Integration-ms:** Es el agregado llamado “Integración” tiene el dominio de la información de las bases de datos de los usuarios.
- **Query-ms:** Es el agregado llamado “Query” tiene el dominio de la información de las consultas que genera el usuario basado en una integración.
- **Runner-ms:** Es el agregado llamado “Conexión” tiene la responsabilidad de manejar el pool de conexiones hacia las bases de datos de las integraciones, además de ejecutar las consultas SQL.
- **Query-builder-ms:** Es un servicio de dominio llamado “Query builder” encargado de la construcción de las consultas SQL a partir de la solicitud que genera el frontend.

Ilustración 17.

Diagrama de microservicios a partir del modelo del dominio.

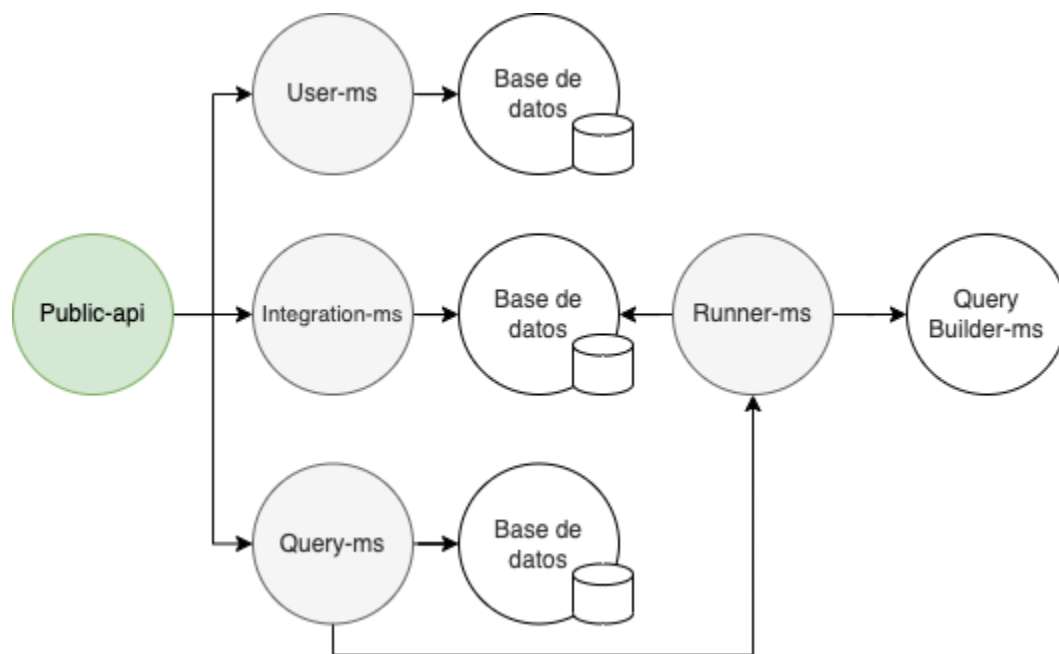


Una vez se han definido los microservicios se realizan consideraciones para administrar los datos en la arquitectura propuesta, debido a que cada microservicio puede manejar sus propios datos, la consistencia de datos es un objetivo importante a tener en cuenta.

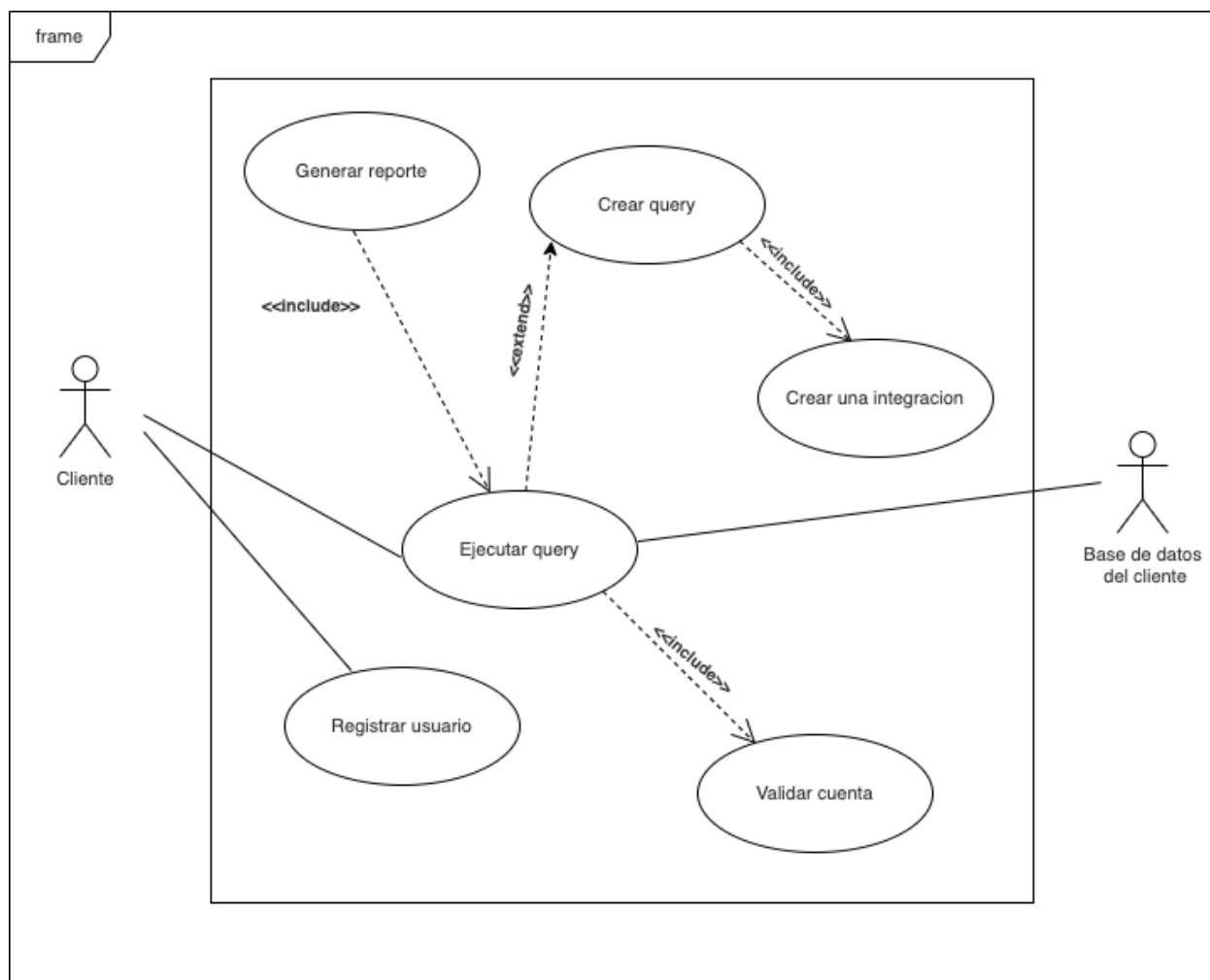
En este punto se le asignan bases de datos a los microservicios que tengan un dominio el cual requiera la persistencia de datos, pueden existir microservicios con bases de datos compartidas, pero debe haber un microservicio responsable principal y los demás pueden tener un rol de lectura, en lo posible este tipo de práctica debe ser evitado ya que limita la capacidad de optimización para cada servicio particular y puede ser considerado como una mala práctica. En este proyecto se identificó una necesidad de aplicar esta práctica ya que existen dos dominios con diferente contexto, pero con información en común, para llevar a cabo esta práctica cada servicio tendrá un rol asignado, en el caso de **integration-ms** el rol de escritura (responsable principal) y **runner-ms** el rol de lectura.

Ilustración 18.

Diagrama de microservicios con bases de datos.

**Diagramas de casos de uso**

Cuando se modela un diagrama de casos de uso se representa la forma en la cual el usuario interactúa con la aplicación, además de definir límites e interacciones con actores externos como pueden ser servicios de terceros, en el proyecto un actor externo son las bases de datos de los usuarios las cuales son las integraciones.

Ilustración 19.*Flujo general del sistema.*

Con el diagrama de casos de uso se define el comportamiento de cada caso de uso, para ello se usa normalmente una tabla en donde se especifican las condiciones iniciales, el flujo de eventos, la condición de salida y requerimientos especiales.

Tabla 1.**Especificación del caso de uso “Registrar usuario”**

Nombre del caso de uso	Registrar Usuario
Actor participante	Iniciado por el usuario
Condición inicial	El usuario al cargar la aplicación tendrá una pantalla de inicio de sesión.
Flujo de eventos	1. El usuario hace click en el botón de registrarse. 2. Se muestra una nueva pantalla donde se le piden los datos básicos (nombre, email y contraseña). 3. El usuario rellena los campos. 4. Se oprime el botón de registrarse
Condición de salida	Cargará la página de inicio con las opciones principales.
Requerimientos especiales	1. Si el usuario no llena los datos del formulario de registro no se podrá registrar. 2. Si el correo ya fue usado tampoco podrá ser registrado.

Tabla 2.**Especificación del caso de uso “Crear una integración”**

Nombre del caso de uso	Crear una integración
Actor participante	Iniciado por el usuario
Condición inicial	1. El usuario al cargar la aplicación en la pantalla de inicio con una sesión activa.
Flujo de eventos	2. El usuario hace click en el botón de crear integración. 3. Se muestra una nueva pantalla donde se le piden los datos básicos de la integración (nombre de la integración,

	host, puerto, usuario, contraseña, y nombre de la base de datos). 4. El usuario rellena los campos. 5. Se oprime el botón de crear integración.
Condición de salida	Cargará la página de inicio con las opciones principales y abajo un listado de últimas integraciones creadas.
Requerimientos especiales	1. Si el usuario no llena los datos del formulario de registro no se podrá crear la integración. 2. Si no se logra conectar con la base de datos, no se podrá crear la integración.

Tabla 3.

Especificación del caso de uso “Validar cuenta”

Nombre del caso de uso	Validar cuenta
Actor participante	Iniciado por el usuario
Condición inicial	El usuario al cargar la aplicación tendrá una pantalla de inicio de sesión.
Flujo de eventos	1. Se muestra la pantalla donde se le piden los datos de sesión (email y contraseña). 2. El usuario rellena los campos. 3. Se oprime el botón de iniciar sesión
Condición de salida	Cargará la página de inicio con las opciones principales.
Requerimientos especiales	1. Si el usuario escribe una contraseña incorrecta o el email no existe no se podrá iniciar sesión.

Tabla 4.*Especificación del caso de uso “Crear una query”*

Nombre del caso de uso	Crear una query
Actor participante	Iniciado por el usuario
Condición inicial	El usuario una vez haya ejecutado una query y se muestran los resultados.
Flujo de eventos	1. Oprime el botón de guardar query.
Condición de salida	Se guarda la query en la base de datos.
Requerimientos especiales	1. Antes de guardar la query debe ser ejecutada con éxito.

Tabla 5.*Especificación del caso de uso “Ejecutar query”*

Nombre del caso de uso	Ejecutar query
Actor participante	Iniciado por el usuario
Condición inicial	El usuario al cargar la aplicación tendrá una pantalla de inicio con las funcionalidades.
Flujo de eventos	1. El usuario selecciona la integración con la cual desea realizar la query. 2. Se genera una pantalla nueva donde se listan las queries existentes, y arriba un botón de crear query. 3. Se oprime el botón de crear query. 4. Se genera una pantalla con el query builder. 5. Se muestra un dropdown en donde se selecciona una tabla a la cual consultar la información. 6. El usuario selecciona la tabla a consultar en el dropdown. 7. Oprime el botón ejecutar query. 8. Se despliega una tabla con la información consultada de la base de

	datos.
Condición de salida	Cargará la información suministrada por la integración del usuario (base de datos).
Requerimientos especiales	1. Si no se selecciona una tabla a consultar no se podrá ejecutar la query.

Tabla 6.

Especificación del caso de uso “Generar reporte”

Nombre del caso de uso	Generar reporte
Actor participante	Iniciado por el usuario
Condición inicial	El usuario una vez generado la query en la pantalla de query builder, con el dropdown de la tabla que desea consultar previamente seleccionada y ejecutada la query.
Flujo de eventos	1. Se oprime el botón generar reporte. 2. Se abre una nueva pantalla en la cual se le pide al usuario 2 fuentes de datos (columnas de la tabla consultada) con un dropdown para cada una y otro dropdown para el tipo de gráfica. 3. El usuario selecciona las fuentes de datos y el tipo de gráfica.
Condición de salida	Se genera una gráfica de reporte con la información seleccionada.
Requerimientos especiales	1. Se debe rellenar toda la información (fuentes de datos y tipo de gráfica) para generar la gráfica de reporte correspondiente en caso contrario no se podrá visualizar el reporte.

Diagramas de secuencia

Un diagrama de secuencia es usado para describir cómo y en qué orden interactúa un grupo de objetos entre sí, su principal función es ayudar a comprender los requisitos del sistema y documentar el proceso que se está modelando.

Ilustración 20.

Diagrama de secuencia de “validar cuenta”

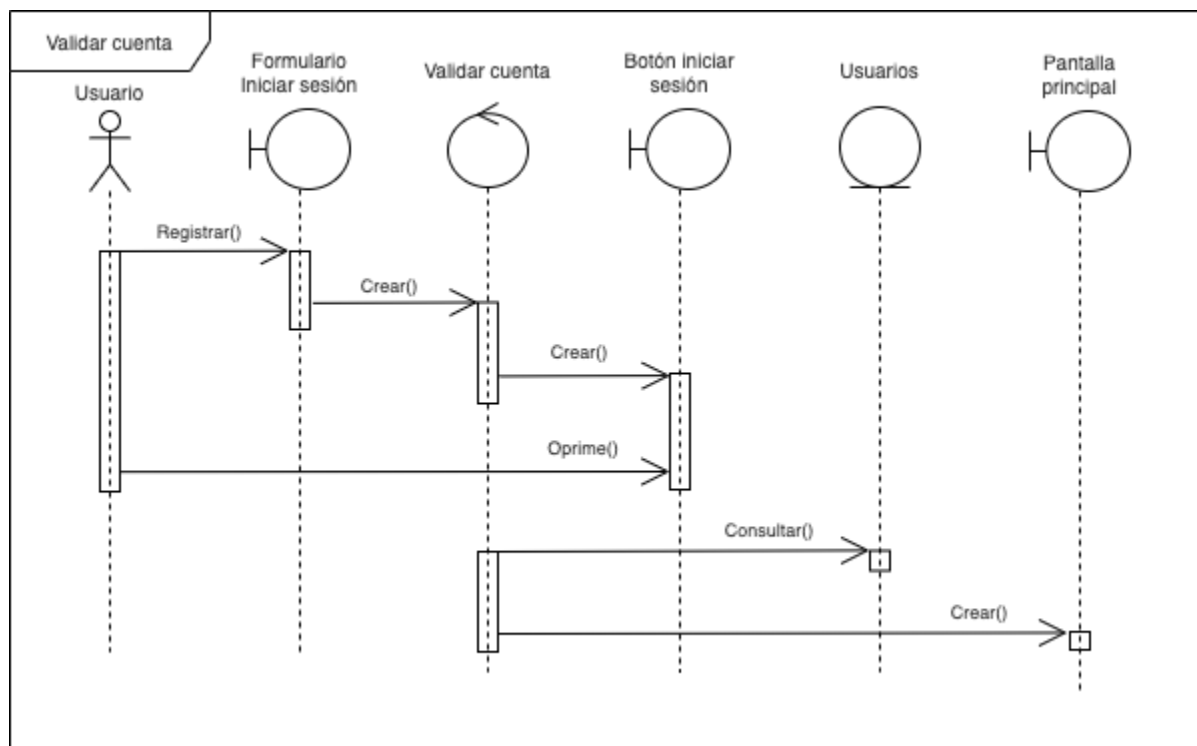


Ilustración 21.

Diagrama de secuencia de “registrar usuario”

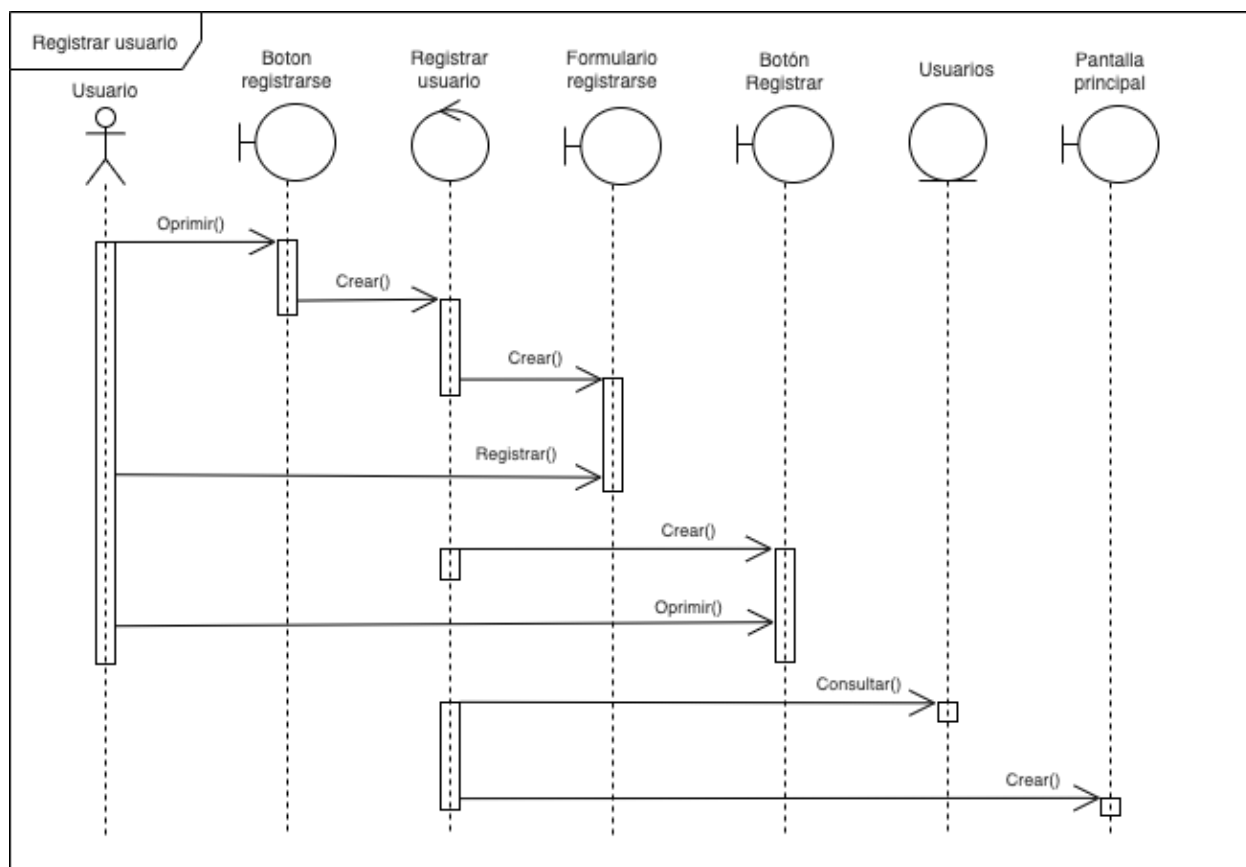


Ilustración 22.

Diagrama de secuencia de “crear integración”

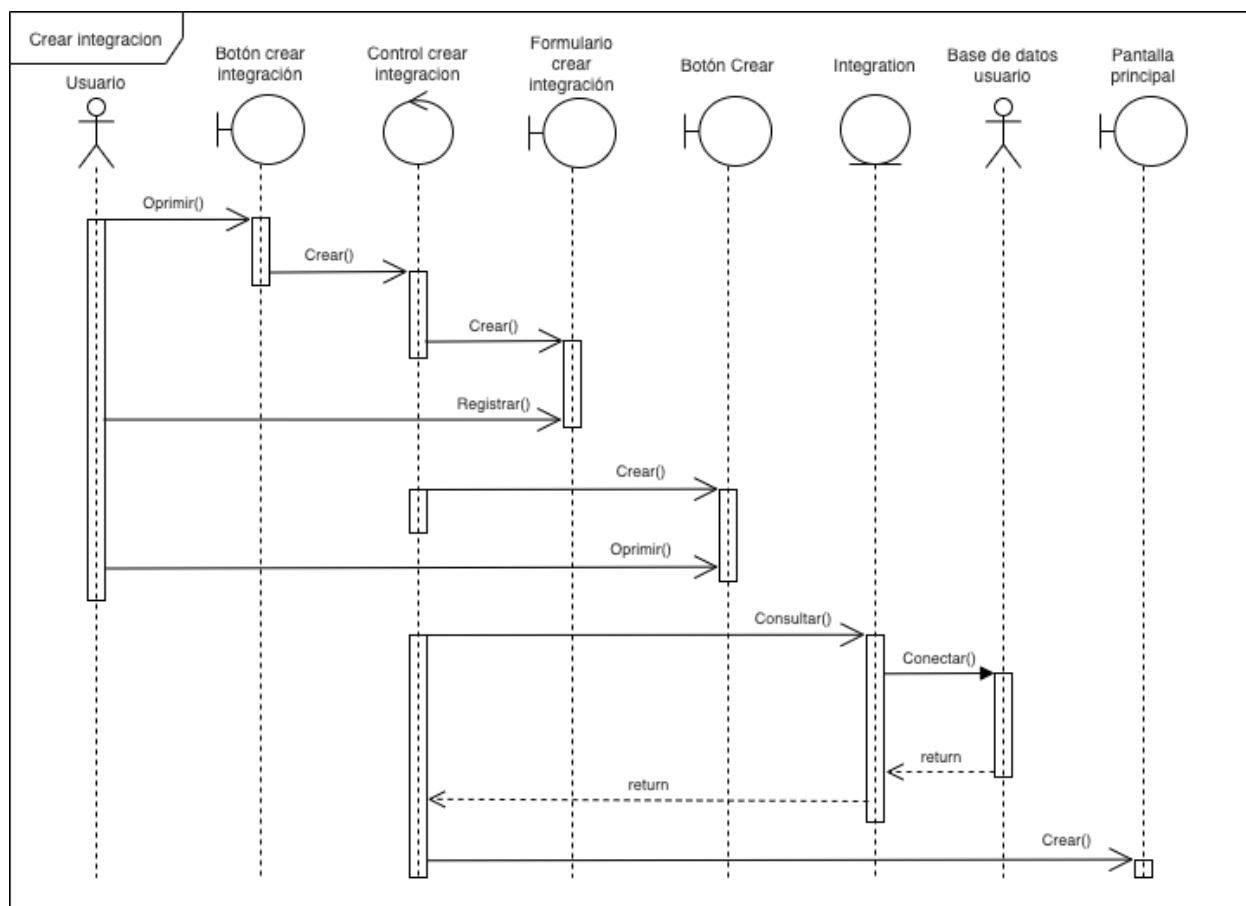


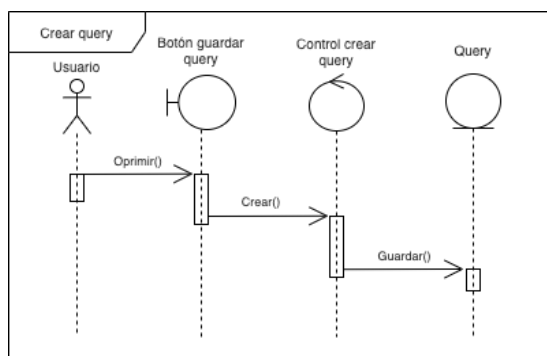
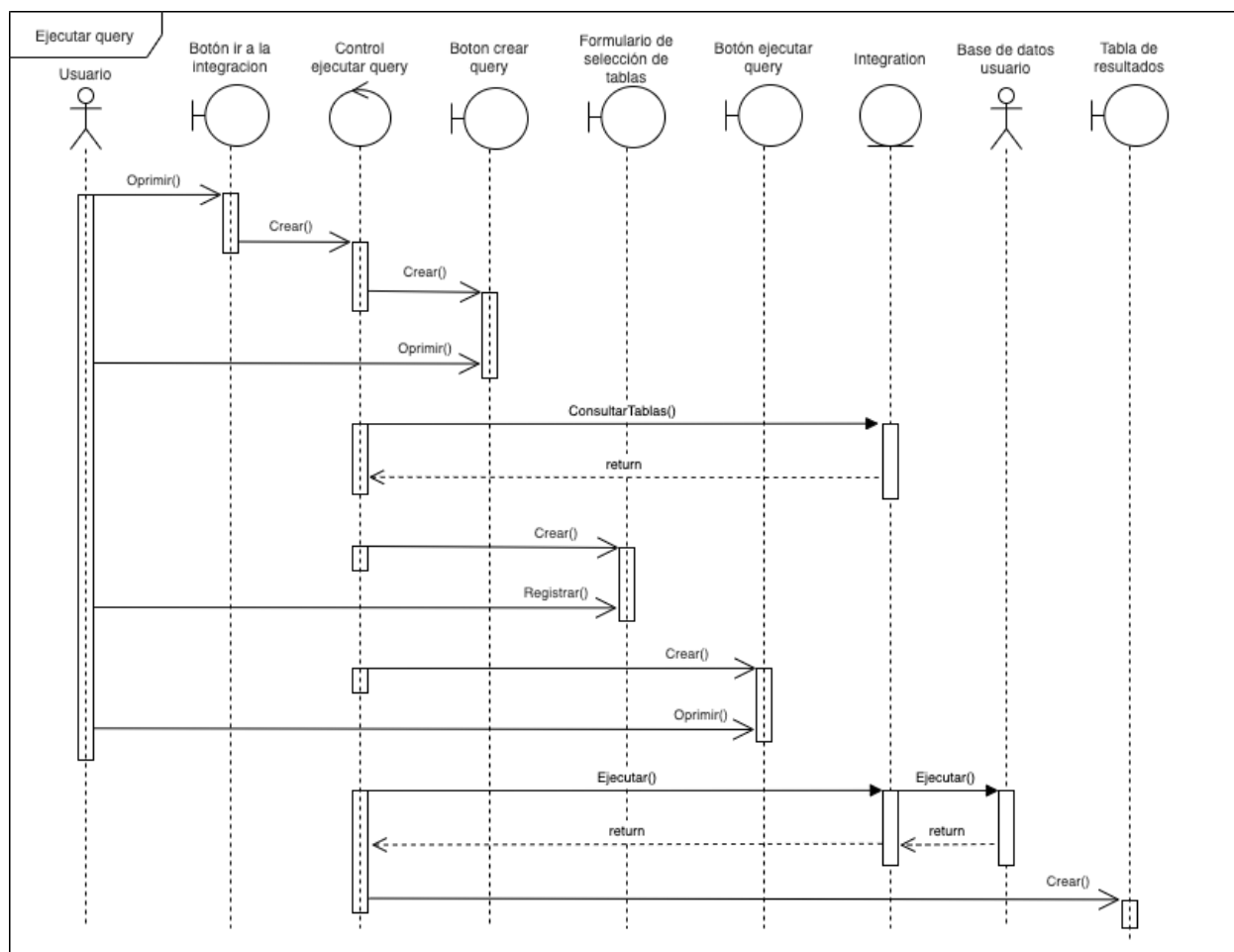
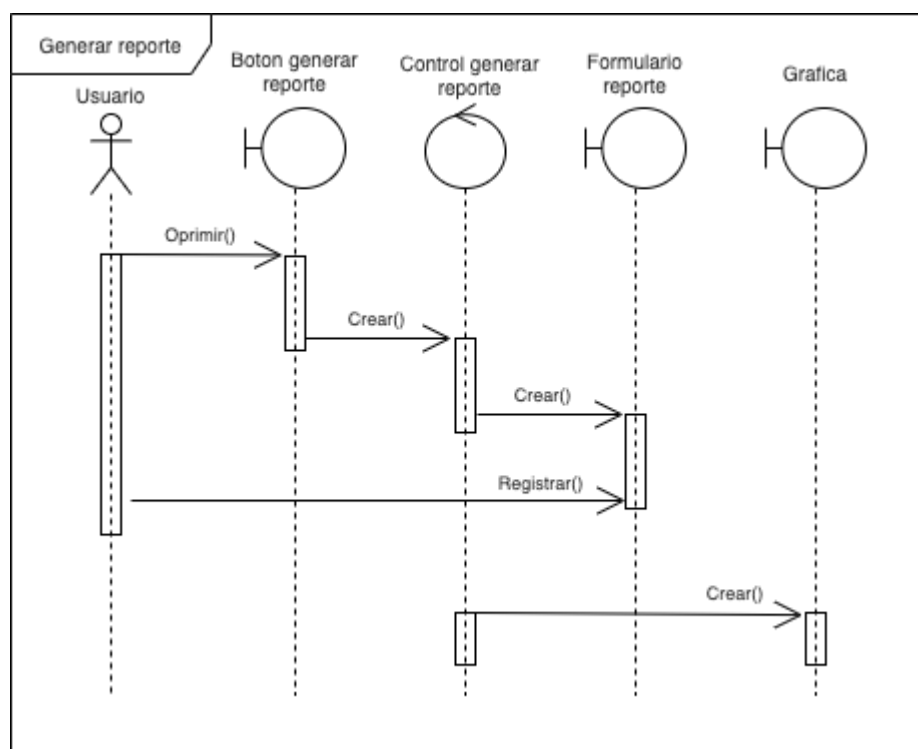
Ilustración 23.*Diagrama de secuencia de “crear query”***Ilustración 24.***Diagrama de secuencia de “ejecutar query”*

Ilustración 25.

Diagrama de secuencia de “generar reporte”



Definición de microservicios

Microservicio: User-MS

Es un microservicio escrito en go usando el framework de echo, encargado de administrar el dominio de los usuarios en donde se almacenan todos los datos básicos.

Ilustración 26.

Modelo entidad relación de User-MS

user	
PK	<u>id SERIAL</u>
UK	code UUID
	first_name VARCHAR
	last_name VARCHAR
	email VARCHAR
	password VARCHAR
	created_at TIMESTAMP
	updated_at TIMESTAMP

Endpoints

- Crear usuario:
POST /user-ms/v1/user
- Obtener información del usuario por código:
GET /user-ms/v1/user/:code
- Obtener información del usuario por email y contraseña:
POST /user-ms/v1/login

Microservicio: Integration-MS

Es un microservicio escrito en kotlin usando el framework spring, encargado de administrar el dominio de las integraciones en donde se almacena la información de la integración como son los datos de conexión a las bases de datos de los clientes y los esquemas de las mismas, también encargado de los servicios CRUD de las integraciones.

Ilustración 27.

Modelo entidad relación de Integration-MS

integration	
PK	<u>id SERIAL</u>
UK	code UUID
	name VARCHAR
	status VARCHAR
	user_code UUID
	host VARCHAR
	port VARCHAR
	db_name VARCHAR
	db_username VARCHAR
	db_password VARCHAR
	schema JSON
	ssl BOOLEAN
	checksum VARCHAR(255)
	created_at TIMESTAMP
	updated_at TIMESTAMP

Endpoints

- Crear integración:
POST /integration-ms/v1/integration
- Listado de integraciones:
GET /integration-ms/v1/integration
- Obtener una integración por código:
GET /integration-ms/v1/integration/:code
- Obtener el esquema de una integración por código:
GET /integration-ms/v1/integration/:code/schema
- Actualizar una integración por código:
PATCH /integration-ms/v1/integration/:code
- Eliminar una integración por código:
DELETE /integration-ms/v1/integration/:code

Microservicio: Runner-MS

Es un microservicio escrito en kotlin usando el framework spring, tiene el dominio de conexión con las bases de datos de los clientes siendo un administrador de pool de conexiones y encargado de ejecutar consultas SQL, hace uso de la información del microservicio Integration-Ms con el rol de lectura.

Endpoints

- Ejecutar una query SQL:
POST /runner-ms/v1/runner
- Analizar el esquema de una base de datos:
POST /runner-ms/v1/runner/schema
- Ejecutar una query JSON:
POST /runner-ms/v1/runner/query

Microservicio: Query-Builder-MS

Es un microservicio escrito en javascript con node.js usando el framework express.js, se encarga de traducir la request que se genera en el frontend en JSON a SQL con ayuda de la librería knex.js.

Endpoints

- Traducir una query JSON a SQL:
POST /query-builder-ms/v1/query

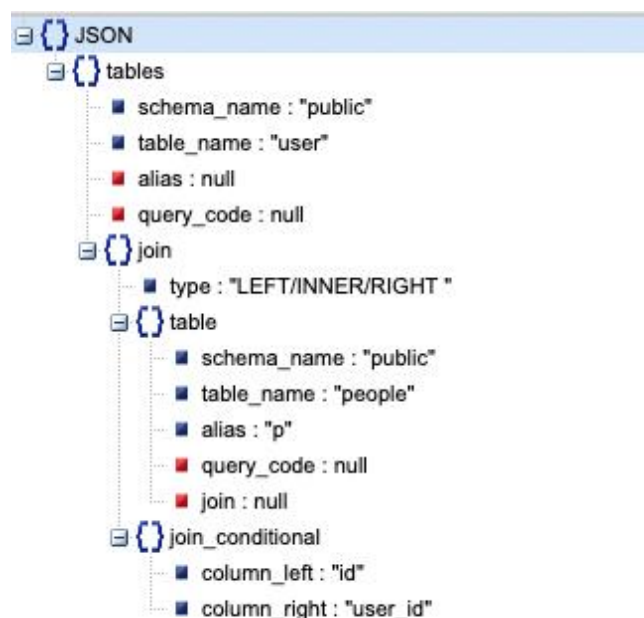
Estructura de las consultas en JSON

La estructura para representar consultas en formato JSON está constituido de 5 objetos que se genera desde el frontend con ayuda la interfaz gráfica:

- “tables”, es un objeto donde se referencia la información de la tabla o query de la integración a modo de subquery a consultar y dentro del mismo otro objeto join en donde se requiere un tipo el cual es el tipo de JOIN y otros dos objetos más, uno es una copia exacta el objeto raíz de “tables” por lo cual tendrá otro objeto join con el cual seguir haciendo joins o en caso contrario valor nulo para detenerse, y el otro la condición para el join, es equivalente al FROM de SQL.

Ilustración 28.

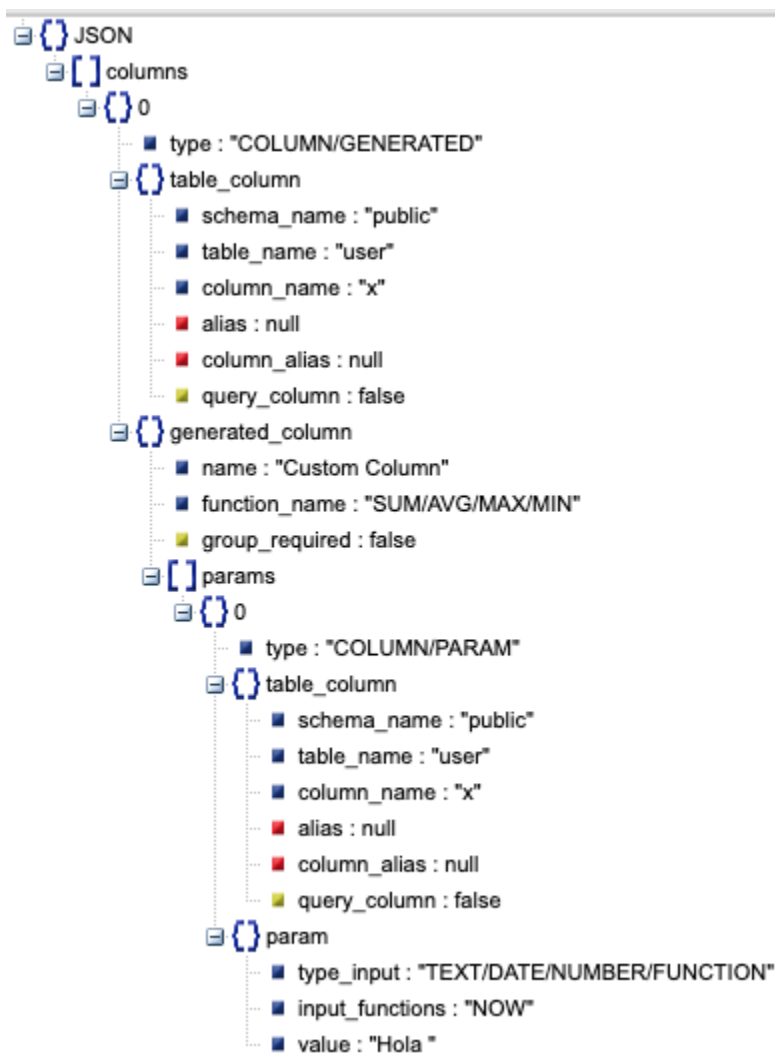
Objeto “tables” de la estructura de consultas en JSON.



- “columns”, es una lista de objetos que referencian a columna de una tabla (como en el objeto tables también puede ser de una query de la integración) o una función generada, cada objeto tiene el tipo de columna con ella se determina que objeto de dentro se debe rellenar en caso de ser columna de una tabla debe ser “table_column” en donde se hace referencia la información de dicha columna, en caso contrario de ser un tipo de columna GENERATED, el objeto a rellenar es “generated_column” que dentro de se encuentra un atributo tipo el cual decide la función además de una array de parámetros los cuales pueden ser de tipo columna o parámetro ingresado por el usuario, es equivalente al SELECT en SQL.

Ilustración 29.

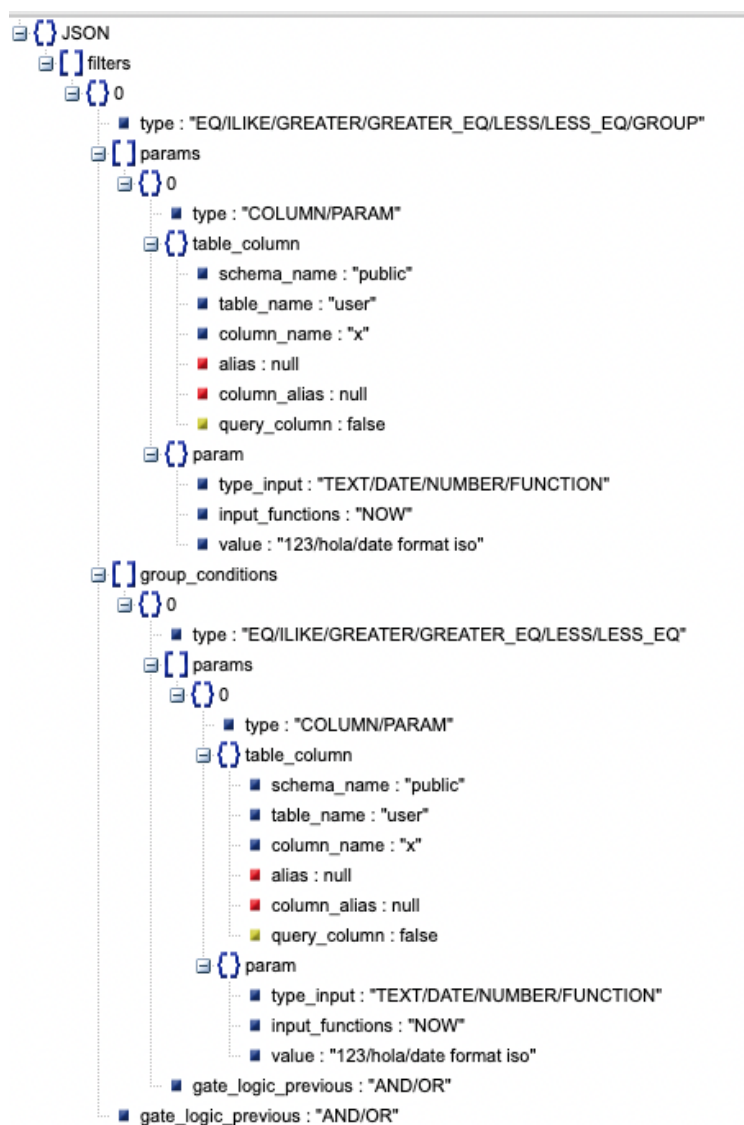
Lista de objetos “columns” de la estructura de consultas en JSON.



- “filters” es la lista de objetos que representan una condición booleana con las columnas de la base de datos y entradas del usuario, que se dividen en dos tipos, grupo de condiciones (condiciones booleanas agrupadas con un paréntesis) o condiciones individuales siendo esto determinado por el tipo en caso de ser GROUP, de otra manera es el tipo de condicional a aplicar. Además del tipo de condición tenemos parámetros que como en el caso de los objetos de “columns” hacen referencias a columnas o entradas del usuario, es equivalente al WHERE de SQL.

Ilustración 30.

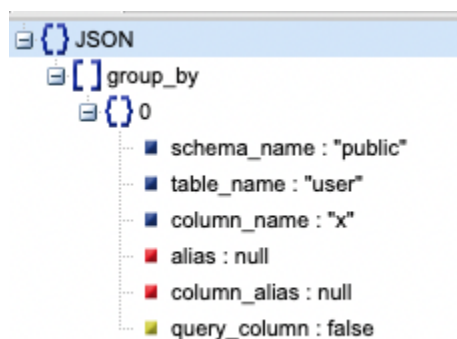
Lista de objetos “filters” de la estructura de consultas en JSON.



- “group_by” es una lista de objetos que representan una columna como en el caso de los objetos de “columns”, equivalente a GROUP BY en SQL.

Ilustración 31.

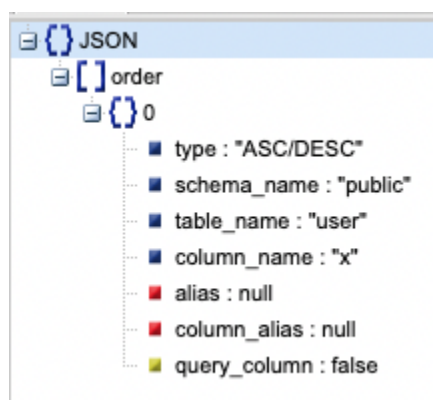
Lista de objetos “group_by” de la estructura de consultas en JSON.



- “order” es una lista de objetos que representan una columna como en el caso de los objetos de “columns” solo con la particularidad de tener un campo extra que determina si se ordena de manera descendente o ascendente, equivalente a SORT en SQL.

Ilustración 32.

Lista de objetos “orden” de la estructura de consultas en JSON.



Microservicio: Query-MS

Es un microservicio escrito en kotlin usando el framework spring, encargado de administrar el dominio de las queries en donde se almacena la información relacionada a las queries que se generan en SQL y JSON generadas por el frontend y ademas las gráficas.

Ilustración 33.

Modelo entidad relación de Query-MS

query	
PK	<u>id SERIAL</u>
UK	code UUID
	name VARCHAR
	status VARCHAR
	sql VARCHAR
	json JSON
	schema JSON
	created_at TIMESTAMP
	updated_at TIMESTAMP
	checksum VARCHAR(255)
	integration_code UUID

Endpoints

- Crear una query:
POST /query-ms/v1/query
- Obtener listado de queries por una lista de códigos:
POST /query-ms/v1/query/list
- Obtener un listado de queries por el código de la integración:
GET /query-ms/v1/query/integration/:integration_code
- Obtener una query por el código de la integración y código de la query:
GET /query-ms/v1/query/:query_code/integration/:integration_code

- Actualizar una query por el código de la integración y código de la query:
PATCH /query-ms/v1/query/:query_code/integration/:integration_code
- Eliminar una query por el código de la integración y código de la query:
DELETE /query-ms/v1/query/:query_code/integration/:integration_code

Microservicio: Public-API

Es un microservicio escrito en go usando el framework de echo, encargado de exponer los endpoints de los microservicios y la autenticación con la aplicación mediante JWT (Json Web Token), el único endpoint que no requiere la autenticación es crear usuario.

Endpoints

- Crear usuario:
POST /public-api/v1/user
- Obtener información del usuario por código:
GET /public-api/v1/user/:code
- Autenticarse con el usuario por email y contraseña:
POST /public-api/v1/login
- Crear integración:
POST /public-api/v1/integration
- Listado de integraciones:
GET /public-api/v1/integration
- Obtener una integración por código:
GET /public-api/v1/integration/:code
- Obtener el esquema de una integración por código:
GET /public-api/v1/integration/:code/schema
- Actualizar una integración por código:
PATCH /public-api/v1/integration/:code
- Eliminar una integración por código:
DELETE /public-api/v1/integration/:code

- Ejecutar una query JSON:
POST /public-api/v1/query/run
- Crear una query:
POST /public-api/v1/query
- Obtener un listado de queries por el código de la integración:
GET /public-api/v1/integration/integration_code/queries
- Obtener una query por el código de la integración y código de la query:
GET /public-api/v1/integration/:integration_code/query/:query_code
- Actualizar una query por el código de la integración y código de la query:
PATCH /public-api/v1/integration/:integration_code/query/:query_code
- Eliminar una query por el código de la integración y código de la query:
DELETE /public-api/v1/integration/:integration_code/query/:query_code

Definición de servicio frontend

Servicio desarrollado con javascript con el framework next.js de la librería react.js

Ilustración 34.

Pantalla principal del proyecto



Ilustración 35.

Pantalla para crear integraciones

 QueryBuilder

Create a new integration
Postgres Integration

Integration name

Name integration

Database info

Host

Host database

Username

Username database

Password

Password database

Database name

Name database

Port

Port database

Create integration

Ilustración 36.

Pantalla listada de queries de una integración

 QueryBuilder

Queries of integration "Otra integracion"
Name database: integration-ms-db

Welcome Steban! 

Create a new query

Name	Status	Updated at
List clients by revenue every month of year 2021	 SUCCESSFULLY	Friday 19 Aug 5:26PM

Ilustración 37.

Pantalla del query builder

The screenshot shows the QueryBuilder interface. At the top, there's a header with the QueryBuilder logo and a user profile 'Welcome Steban!'. Below the header, there's a navigation bar with 'Query' and 'Chart' tabs. The main area is divided into a left sidebar and a large central workspace. The sidebar contains a 'Main table source:' section with a dropdown menu showing 'integration' and an 'Alias*' field with the value 'i'. The central workspace is empty, and at the bottom, there's a status bar that says 'Ready for execute query...'.

Ilustración 38.

Interfaz para filtrar en el query builder

The screenshot shows the QueryBuilder interface with the filter configuration screen. The left sidebar is the same as in the previous screenshot. The central workspace is now filled with a filter configuration area. It starts with the text 'In this view, show records'. Below this, there's a 'Where' clause with a dropdown menu showing 'i.id', a comparison operator '>', a value '3', and a checkbox 'x'. Below this, there's a nested filter configuration area with an 'and' operator, an 'or' operator, and two filter conditions: 'i.name contains hola' and 'i.name contains adios'. Each condition has a checkbox 'x' and a menu icon. At the bottom, there's an 'Add filter' button and a '+ Add group filter' button.

Ilustración 39.

Interfaz para ocultar o mostrar columnas en el query builder

 QueryBuilder

Query	Chart			
Data in use 		Hide fields	Filter	Group
Main table source: i 		☐ i.updated_at  ☐ i.id  ☐ i.code  ☐ i.user_code  ☐ i.port  ☐ i.ssl  ☐ i.schema  ☐ i.created_at  ☐ i.name  ☐ i.status  ☐ i.host  ☐ i.checksum  ☐ i.db_name  ☐ i.db_username  ☐ i.db_password  Hide all Show all		
Table integration 				
Alias* i				

Ilustración 40.

Interfaz para agrupar campos y columnas agregadas en el query builder

 QueryBuilder

Query	Chart					
Data in use 		Hide fields	Filter	Group	Sort	
Main table source: i 		In this view, group by i.port  x  + Add group by field and added columns aggregate i.id  Count  x  + Add aggregate column function				
Table integration 						
Alias* i						

Ilustración 41.

Interfaz para ordenar por campos en el query builder

 QueryBuilder

Query	Chart					
Data in use		Hide fields	Filter	Group	Sort	
Main table source: i		In this view, sorted by i.id ▼ DESC ▼ x ≡ + Add Sort by				
Table						
integration ▼						
Alias*						
i						

Ilustración 42.

Pantalla del query builder con resultados

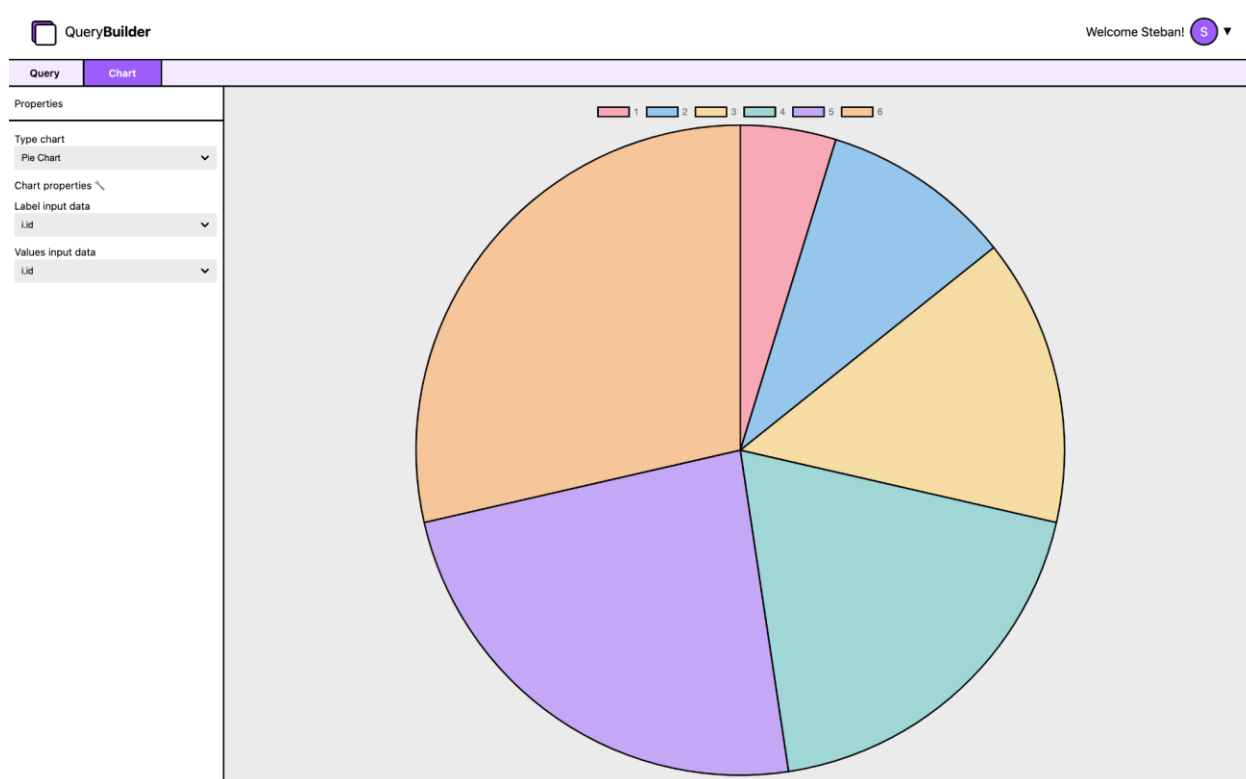
 QueryBuilder Welcome Steban! 

Query	Chart											Run Query	Save Query																																																														
Data in use		Hide fields	Filter	Group	Sort																																																																						
Main table source: i		<table border="1"> <thead> <tr> <th>i.updated_at</th> <th>i.id</th> <th>i.code</th> <th>i.user_code</th> <th>i.port</th> <th>i.ssl</th> <th>i.schema</th> <th>i.created_at</th> <th></th> </tr> </thead> <tbody> <tr> <td>2022-08-14T22:09:01.722+00:00</td> <td>1</td> <td>47324f04-98bf-4fd7-9096-b5686a610115</td> <td>2092ed6b-a8ab-49f3-982c-62264e09f728</td> <td>5432</td> <td>false</td> <td>[object Object]</td> <td>2022-08-14T22:09:01.722+00:00</td> <td>Paym</td> </tr> <tr> <td>2022-08-16T20:33:08.946+00:00</td> <td>2</td> <td>7f2437e7-3a55-40cf-b550-9741a2b7f368</td> <td>d66e7e49-bb5a-46b4-97b4-f10e1134eeb6</td> <td>5432</td> <td>false</td> <td>[object Object]</td> <td>2022-08-16T20:33:08.946+00:00</td> <td>Nuev</td> </tr> <tr> <td>2022-08-18T05:16:10.454+00:00</td> <td>3</td> <td>5a283d6d-db5c-4a06-af77-50d8f044ffc0</td> <td>93eb16b6-4785-40d3-a3ec-d63fe5e3a349</td> <td>5432</td> <td>false</td> <td>[object Object]</td> <td>2022-08-18T05:16:10.454+00:00</td> <td>Hola j</td> </tr> <tr> <td>2022-08-18T05:16:51.771+00:00</td> <td>4</td> <td>77f2f66d-a880-4f17-90c6-f042f095f167</td> <td>93eb16b6-4785-40d3-a3ec-d63fe5e3a349</td> <td>5432</td> <td>false</td> <td>[object Object]</td> <td>2022-08-18T05:16:51.771+00:00</td> <td>Hola i</td> </tr> <tr> <td>2022-08-19T20:53:53.436+00:00</td> <td>5</td> <td>f7a122d6-0fae-4d77-88a3-35b49c4e1a04</td> <td>d66e7e49-bb5a-46b4-97b4-f10e1134eeb6</td> <td>5432</td> <td>false</td> <td>[object Object]</td> <td>2022-08-19T20:53:53.436+00:00</td> <td>Otra i</td> </tr> <tr> <td>2022-09-11T23:41:42.151+00:00</td> <td>6</td> <td>d5a8d9e1-0f5f-4401-8189-73c3cb0ebf48</td> <td>d66e7e49-bb5a-46b4-97b4-f10e1134eeb6</td> <td>5438</td> <td>false</td> <td>[object Object]</td> <td>2022-09-11T23:41:42.150+00:00</td> <td>Paym</td> </tr> </tbody> </table>											i.updated_at	i.id	i.code	i.user_code	i.port	i.ssl	i.schema	i.created_at		2022-08-14T22:09:01.722+00:00	1	47324f04-98bf-4fd7-9096-b5686a610115	2092ed6b-a8ab-49f3-982c-62264e09f728	5432	false	[object Object]	2022-08-14T22:09:01.722+00:00	Paym	2022-08-16T20:33:08.946+00:00	2	7f2437e7-3a55-40cf-b550-9741a2b7f368	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5432	false	[object Object]	2022-08-16T20:33:08.946+00:00	Nuev	2022-08-18T05:16:10.454+00:00	3	5a283d6d-db5c-4a06-af77-50d8f044ffc0	93eb16b6-4785-40d3-a3ec-d63fe5e3a349	5432	false	[object Object]	2022-08-18T05:16:10.454+00:00	Hola j	2022-08-18T05:16:51.771+00:00	4	77f2f66d-a880-4f17-90c6-f042f095f167	93eb16b6-4785-40d3-a3ec-d63fe5e3a349	5432	false	[object Object]	2022-08-18T05:16:51.771+00:00	Hola i	2022-08-19T20:53:53.436+00:00	5	f7a122d6-0fae-4d77-88a3-35b49c4e1a04	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5432	false	[object Object]	2022-08-19T20:53:53.436+00:00	Otra i	2022-09-11T23:41:42.151+00:00	6	d5a8d9e1-0f5f-4401-8189-73c3cb0ebf48	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5438	false	[object Object]	2022-09-11T23:41:42.150+00:00	Paym
i.updated_at	i.id	i.code	i.user_code	i.port	i.ssl	i.schema	i.created_at																																																																				
2022-08-14T22:09:01.722+00:00	1	47324f04-98bf-4fd7-9096-b5686a610115	2092ed6b-a8ab-49f3-982c-62264e09f728	5432	false	[object Object]	2022-08-14T22:09:01.722+00:00	Paym																																																																			
2022-08-16T20:33:08.946+00:00	2	7f2437e7-3a55-40cf-b550-9741a2b7f368	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5432	false	[object Object]	2022-08-16T20:33:08.946+00:00	Nuev																																																																			
2022-08-18T05:16:10.454+00:00	3	5a283d6d-db5c-4a06-af77-50d8f044ffc0	93eb16b6-4785-40d3-a3ec-d63fe5e3a349	5432	false	[object Object]	2022-08-18T05:16:10.454+00:00	Hola j																																																																			
2022-08-18T05:16:51.771+00:00	4	77f2f66d-a880-4f17-90c6-f042f095f167	93eb16b6-4785-40d3-a3ec-d63fe5e3a349	5432	false	[object Object]	2022-08-18T05:16:51.771+00:00	Hola i																																																																			
2022-08-19T20:53:53.436+00:00	5	f7a122d6-0fae-4d77-88a3-35b49c4e1a04	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5432	false	[object Object]	2022-08-19T20:53:53.436+00:00	Otra i																																																																			
2022-09-11T23:41:42.151+00:00	6	d5a8d9e1-0f5f-4401-8189-73c3cb0ebf48	d66e7e49-bb5a-46b4-97b4-f10e1134eeb6	5438	false	[object Object]	2022-09-11T23:41:42.150+00:00	Paym																																																																			
Table																																																																											
integration ▼																																																																											
Alias*																																																																											
i																																																																											

Executed in 243.4 ms.

Ilustración 43.

Pantalla de generar reporte con una gráfica.



Infraestructura

Una vez definidos los microservicios junto a sus dominios y funcionalidades, se plantea una infraestructura la cual permita ejecutar todo el proyecto.

Cada microservicio es construido como una imagen con Docker a través de un Dockerfile; Un Dockerfile es un documento de texto que contiene una secuencia de instrucciones a ser ejecutadas con el fin de cumplir los procesos necesarios para ejecutar dichos microservicios y ensamblar una imagen.

Las imágenes en Docker son usadas para instanciar máquinas virtuales usando la secuencia de instrucciones que se especifican en dicha imagen con el fin de ejecutar el código, a estas instancias de imágenes se les llama contenedores. Es un concepto similar a la programación orientada a objetos donde una imagen sería una clase, y la instancia de clase sería un contenedor.

Docker Compose

Docker también tiene una herramienta llamada “compose” la cual usa manifiestos llamados “docker compose” en formato YAML que permiten instanciar múltiples contenedores con la posibilidad de comunicarse entre ellos con una red virtual, además de provisionar volúmenes (discos duros) virtuales para persistir información en ellos.

Ilustración 44.

Ejemplo de `docker-compose.yml` con el microservicio `user-ms` y una base de datos `postgres`.

```
1  version: '3.6'
2  > services:
3  >   user-ms-tesis-new:
4  >     image: user-ms-tesis-new
5  >     ports:
6  >       - "8081:8080"
7  >     depends_on:
8  >       - postgres
9  >     environment:
10 >       - SERVER_PORT=8080
11 >       - DB_HOST=postgres
12 >       - DB_PORT=5432
13 >       - DB_NAME=user-ms-db
14 >       - DB_USER=user-ms
15 >       - DB_PASSWORD=secret
16
17 >   postgres:
18 >     image: postgres
19 >     restart: always
20 >     environment:
21 >       - POSTGRES_DB=user-ms-db
22 >       - POSTGRES_USER=user-ms
23 >       - POSTGRES_PASSWORD=secret
24
25 >     ports:
26 >       - "5432:5432"
27
```

Con esta herramienta se puede tener todos los microservicios corriendo en simultáneo cada uno en un contenedor con sus recursos asignados, el problema más grande de tener nuestro sistema corriendo con `docker compose` recae directamente en que no escala, se pueden hacer scripts para hacer que se modifique estos archivos `YAML` con el fin de escalarlos pero no es una forma óptima ni recomendada para solucionar el problema de orquestar los contenedores.

Kubernetes

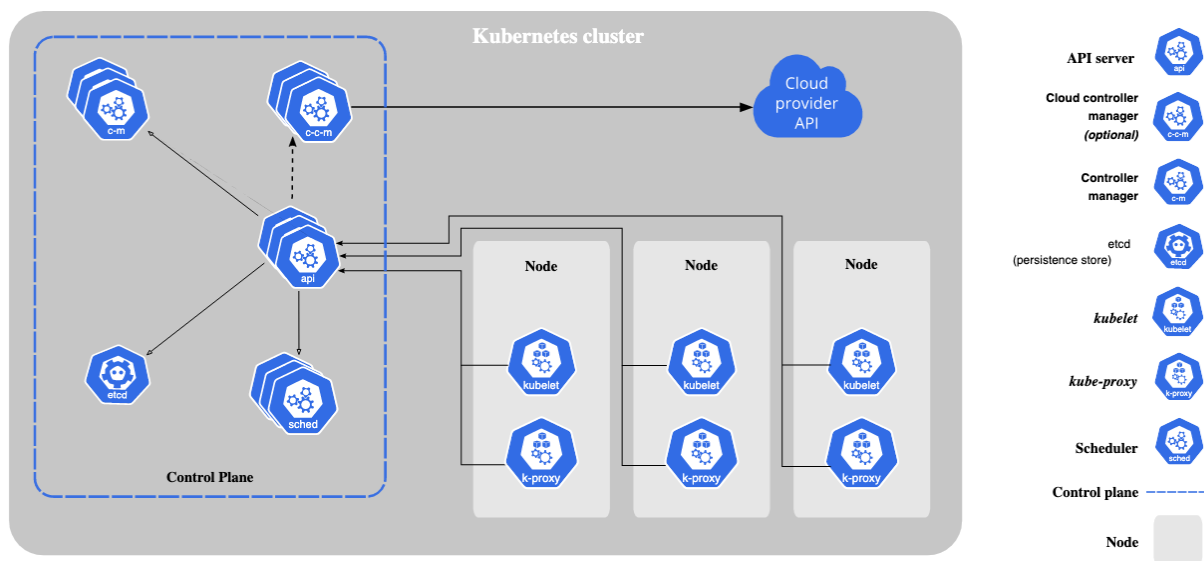
Kubernetes es un orquestador de contenedores que facilita la automatización y la configuración de manera declarativa mediante manifiestos YAML con una gran cantidad de herramientas y servicios.

Componentes de kubernetes

Cuando se habla de kubernetes realmente se hace referencia a un cluster que está formado de muchos nodos o al menos uno, que es donde se alojan los pods los cuales son componentes donde se ejecutan las imágenes de docker en otras palabras los contenedores y el plano de control (control plane) que es el encargado de administrar los nodos.

Ilustración 45.

Componentes de kubernetes. (Kubernetes Components, 2022)



Componentes de plano de control (Control Plane)

Los componentes que conforman el plano de control toman decisiones globales sobre el clúster (por ejemplo, la planificación) detectan y responden a eventos del clúster, como la creación de un nuevo pod cuando la propiedad réplicas de un controlador no se cumple. (*Kubernetes Components*, 2022)

- **kube-apiserver:** El servidor de la API de Kubernetes válida y configura los datos para los objetos de la API, que incluyen pods, servicios, controladores de replicación y otros.
- **etcd:** Es la base de datos de kubernetes que usa clave-valor para persistir toda la configuración e información sobre el cluster.
- **kube-scheduler:** Su función principal es supervisar los pods a ser creados y tomar la decisión de seleccionar en cuál nodo será ejecutado teniendo en cuenta los recursos disponibles, restricciones y dependencias.
- **kube-controller-manager:** Encargado de ejecutar los controladores de nodos, replicación, endpoints, token y cuentas de servicios.
- **cloud-controller-manager:** Es una comunicación para interactuar con proveedor de la nube.

Componentes de Nodo (Node)

- **kubelet:** Es un agente que se ejecuta en cada nodo del cluster y su funcionalidad es asegurarse que los contenedores estén corriendo en un pod y estén funcionando correctamente.
- **kube-proxy:** Encargado de las reglas de red y el reenvío de conexiones.

Recursos de Kubernetes

Recursos a nivel de carga de trabajo:

- **Pod:** es un conjunto de contenedores en ejecución en el cluster.
- **Deployment:** es una implementación declarativa con la cual se pueden definir implementaciones para crear nuevos Pods y ReplicaSets o eliminar implementaciones existentes y adoptar recursos con nuevas implementaciones.
- **ReplicaSet:** el propósito es mantener un conjunto estable de pods de réplica ejecutándose en un momento dado. Como tal, a menudo se usa para garantizar la disponibilidad de un número específico de Pods idénticos.
- **StatefulSet:** se usa para administrar aplicaciones con estado, al igual que un Deployment, un StatefulSet administra Pods que se basan en una especificación de contenedor idéntica. A diferencia de un Deployment, un StatefulSet mantiene una identidad fija para cada uno de sus pods.
- **PersistentVolume:** es una pieza de almacenamiento en el clúster que ha sido aprovisionada por un administrador o aprovisionada dinámicamente mediante clases de almacenamiento.

- **DaemonSet:** garantiza que todos (o algunos) nodos ejecuten una copia de un Pod, es normalmente para ejecutar agentes de monitoreo y recolección de información.
- **Job:** crea uno o más pods y continuará reintentando la ejecución de los pods hasta que un número específico de ellos finalice con éxito.
- **CronJob:** ejecuta un trabajo periódicamente en un horario determinado, escrito en formato Cron.

Recursos a nivel de red:

- **Service:** Expone una aplicación que se ejecuta en el clúster detrás de un único endpoint, incluso cuando la carga de trabajo se divide en varios pods con ip distintas.
- **LoadBalancer:** Tiene la misma particularidad de un Service pero con la diferencia de exponerlo fuera del cluster.
- **Ingress:** Hace que un servicio de red HTTP (o HTTPS) esté disponible mediante un mecanismo de configuración con reconocimiento de protocolo, que comprende conceptos web como URI, nombres de host, rutas y más. El concepto Ingress le permite asignar tráfico a diferentes backends según las reglas que defina a través de la API de Kubernetes.

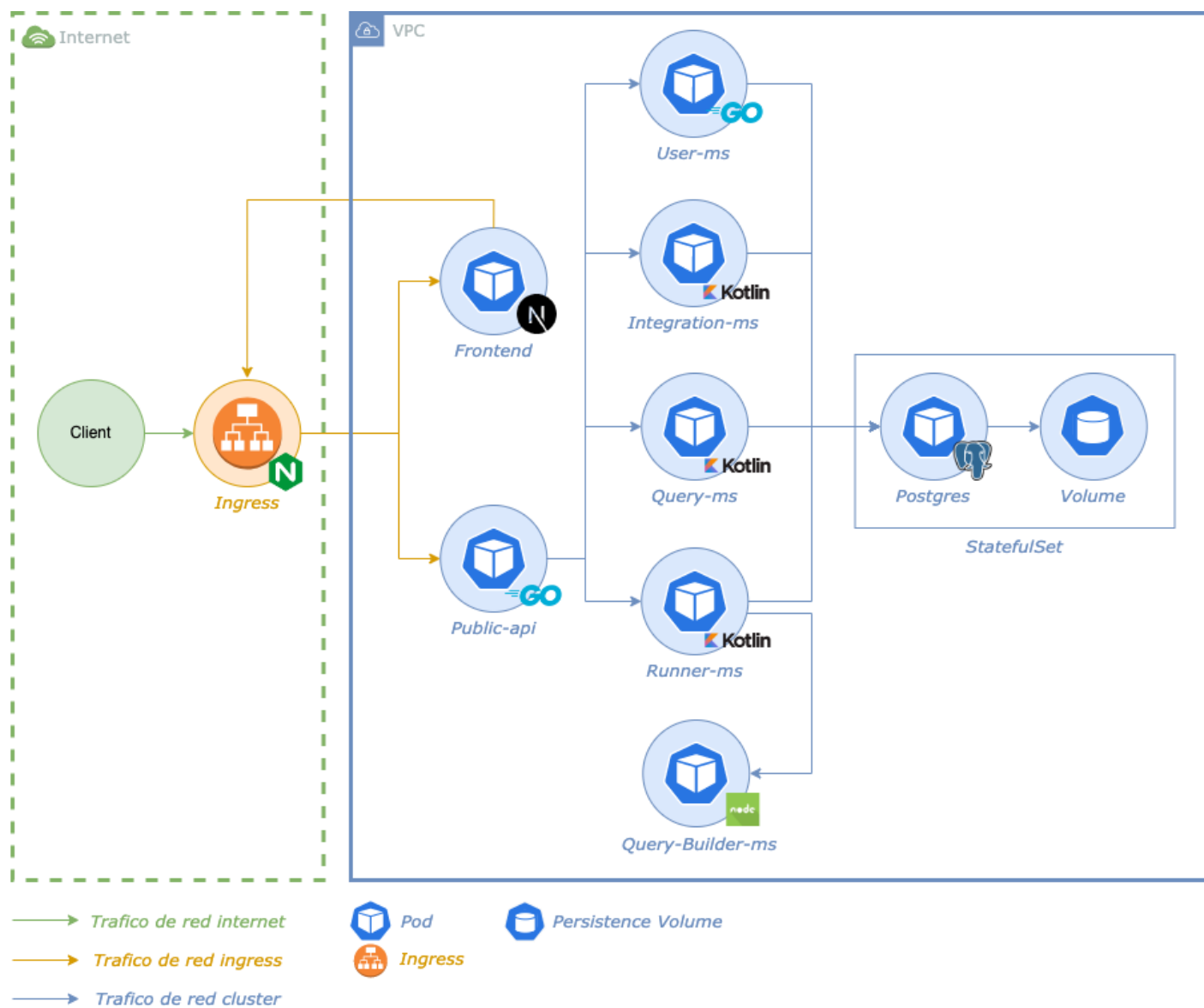
Ilustración 46.

Ejemplo de un manifiesto de un Deployment y un Service

```
327   apiVersion: apps/v1
328   kind: Deployment
329   metadata:
330     name: frontend
331   spec:
332     replicas: 1
333     selector:
334       matchLabels:
335         role: frontend
336     template:
337       metadata:
338         labels:
339           role: frontend
340       spec:
341         containers:
342         - name: frontend
343           image: stebancastro/frontend-tesis:0.0.1-amd64
344           readinessProbe:
345             httpGet:
346               path: /
347               port: 3000
348             initialDelaySeconds: 5
349             periodSeconds: 10
350           ports:
351             - containerPort: 3000
352
353   ---
354   apiVersion: v1
355   kind: Service
356   metadata:
357     name: frontend-svc
358   spec:
359     ports:
360     - port: 80
361       targetPort: 3000
362     selector:
363       role: frontend
```

Ilustración 47.

Infraestructura propuesta con kubernetes.



Digital Ocean

DigitalOcean es un proveedor de alojamiento en la nube que ofrece servicios de computación en la nube e infraestructura como servicio (IaaS).

Servicios

Kubernetes: DigitalOcean Kubernetes (DOKS) es un servicio de Kubernetes administrado que le permite implementar clústeres de Kubernetes sin las complejidades de manejar el plano de control y la infraestructura en contenedores. (*DigitalOcean Kubernetes | DigitalOcean Documentation*, s. f.)

Volumen: Los volúmenes almacenan datos en hardware que está separado de los pods, lo que reduce las posibilidades de pérdida de datos debido a fallas de hardware. (*Volume Features | DigitalOcean Documentation*, s. f.)

Load Balancer: Los balanceadores de carga de DigitalOcean son un servicio de balanceo de carga de red altamente disponible y completamente administrado. Brinda la flexibilidad de cambiar su infraestructura de back-end sin afectar la disponibilidad de sus servicios, lo que permite escalar sin problemas, implementar implementaciones, grandes rediseños de arquitectura y más. (*Load Balancers Features | DigitalOcean Documentation*, s. f.)

Escalado Automático

DigitalOcean Kubernetes proporciona un Cluster Autoscaler (CA) que ajusta automáticamente el tamaño de un clúster de Kubernetes agregando o eliminando nodos en función de la capacidad del clúster para programar pods.

Ilustración 48.

Autoscaler de DigitalOcean para kubernetes.

Adjust node count ×

Select a scaling type

☐ Fixed size ☒ Autoscale [?](#)

[Autoscaling](#) is best for unpredictable and mission critical workloads. Used in conjunction with [horizontal pod autoscaling \(HPA\)](#).

Node count range

Minimum nodes Maximum nodes

— 1 + to — 5 +

Autoscaled node pools require at least one node.

MONTHLY RATE

\$12 - \$60/month
(\$0.02 - \$0.09/hour)
Basic - 1vCPU 2 GB RAM

Close Save

Conclusiones

- De acuerdo a los objetivos planteados para el presente proyecto, se desarrolló exitosamente una web app con una arquitectura de microservicios la cual es escalable, además de tener como foco principal a las consultas SQL a través de interfaces gráficas usando como referencias herramientas no-code y low-code.
- Con el uso de la arquitectura de microservicios se logró usar diferentes lenguajes de programación para cada microservicio pensando principalmente en facilitar el desarrollo de cada funcionalidad.
- Tras el análisis de requerimientos basado en las herramientas no-code y low-code estudiadas en este proyecto, se crearon criterios de aceptación basados en la usabilidad de la interfaz web tomando como modelo dichas herramientas.
- En virtud de lo estudiado, referente al diseño dirigido por dominio se pudo identificar de manera sencilla e intuitiva la definición de los microservicios realizados en el proyecto a partir de los requerimientos planteados.
- Como se ha podido observar en estudios anteriores, es de vital importancia una buena usabilidad en la interfaz gráfica, particularmente cuando hablamos de constructores de consultas por que hace que la semántica de la consulta sea más transparente.
- En conclusión, el desarrollo de arquitecturas basadas en microservicios permite una gran versatilidad a la hora de desarrollar y escalar de manera automática una infraestructura.

Recomendaciones

Con base en los resultados obtenidos se recomienda considerar o tomar en cuenta los siguientes aspectos de cara a futuros proyectos o investigaciones de la misma línea:

- Explorar la posibilidad de generar columnas a partir de funciones como CONCAT, DATE PART, y condicionales etc.
- Integrar la compatibilidad con integraciones en bases de datos que no están expuestas en internet como por ejemplo en redes privadas o otro tipo de autenticación especial.
- Implementar un gestor de reportes con el fin de guardar los reportes en la aplicación a modo de archivos que puedan ser accedidos en cualquier momento.
- Agregar un diagrama de entidad relación dentro de la aplicación con el fin de realizar consultas sin conocer previamente la estructura de la base de datos.

Bibliografía

airtable. (s.f.). *airtable*. Obtenido de <https://www.airtable.com/guides/start/what-is-airtable>

aws.amazon. (s.f.). *aws.amazon*. Obtenido de <https://aws.amazon.com/es/docker/>

developer.mozilla. (29 de Noviembre de 2022). *developer.mozilla*. Obtenido de <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Digitalocean. (16 de Noviembre de 2021). *Digitalocean*. Obtenido de <https://docs.digitalocean.com/products/kubernetes/>

Digitalocean. (10 de Enero de 2022). *Digitalocean*. Obtenido de <https://docs.digitalocean.com/products/networking/load-balancers/details/features/>

Digitalocean. (5 de Abril de 2022). *Digitalocean*. Obtenido de <https://docs.digitalocean.com/products/volumes/details/features/>

echo.labstack. (s.f.). *echo.labstack*. Obtenido de <https://echo.labstack.com/>

Go. (s.f.). *Go.dev*. Obtenido de <https://go.dev/doc/>

Harris, C. (s.f.). *atlassian*. Obtenido de <https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith>

Kotlin. (16 de Noviembre de 2022). *Kotlinlang*. Obtenido de <https://kotlinlang.org/docs/getting-started.html>

Kubernetes. (17 de Julio de 2022). *Kubernetes*. Obtenido de <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>

Kubernetes. (24 de Octubre de 2022). *Kubernetes*. Obtenido de

<https://kubernetes.io/docs/concepts/overview/components/>

Meijers, J. (s.f.). *tiggre*. Obtenido de <https://www.triggre.com/blog/the-difference-between-low-code-and-no-code-platforms>

Microsoft. (4 de Junio de 2022). *Microsoft*. Obtenido de <https://learn.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis>

microsoft. (s.f.). *microsoft*. Obtenido de <https://www.microsoft.com/en-us/microsoft-365/access>

microsoft. (s.f.). *microsoft*. Obtenido de <https://support.microsoft.com/en-us/office/join-tables-and-queries-3f5838bd-24a0-4832-9bc1-07061a1478f6>

Microsoft. (s.f.). *Microsoft*. Obtenido de <https://learn.microsoft.com/en-us/azure/architecture/microservices/model/microservice-boundaries>

Newman, S. (2014). *Building Microservices*. O'Reilly Media. Obtenido de <https://www.oreilly.com/library/view/building-microservices/9781491950340/>

nodejs. (s.f.). *nodejs*. Obtenido de <https://nodejs.dev/en/learn/>

Paez, L. A. (10 de Febrero de 2010). Desarrollo de una interfaz gráfica para la creación de sentencias SQL. Universitat Oberta de Catalunya. Obtenido de <https://openaccess.uoc.edu/handle/10609/1120>

postgresql. (s.f.). *postgresql*. Obtenido de <https://www.postgresql.org/>

Software Engineering Institute. (4 de Diciembre de 2021). *Software Engineering Institute*. Obtenido de <https://www.sei.cmu.edu/our-work/software-architecture/>

spring. (s.f.). *spring*. Obtenido de <https://spring.io/projects/spring-framework>

Wikibooks. (2009). *Wikibooks*. Obtenido de

https://en.wikibooks.org/wiki/Microsoft_Office/Create_queries_for_a_database