Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Tema:

ARQUITECTURA DE MICROSERVICIOS PARA LA INTEGRACIÓN DE RECURSOS DE INFORMACIÓN EN LA APLICACIÓN WEB HELLOTOOL DEDICADA A LA AUTOMATIZACIÓN DE PROCESOS INTERNOS EN ORGANIZACIONES

Autor:

Andrés Camilo Romero Rodríguez

Pamplona, Norte De Santander

Noviembre 2021

Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Trabajo de grado presentado para optar al título de Ingeniero de Sistemas

Tema:

ARQUITECTURA DE MICROSERVICIOS PARA LA INTEGRACIÓN DE RECURSOS DE INFORMACIÓN EN LA APLICACIÓN WEB HELLOTOOL DEDICADA A LA AUTOMATIZACIÓN DE PROCESOS INTERNOS EN ORGANIZACIONES

Autor:

Andrés Camilo Romero Rodríguez

Director:

PhD. William Mauricio Rojas Contreras

Pamplona, Norte De Santander

Noviembre 2021

DEDICATORIA

Dedicado a mi familia y todas las personas que aportaron

a lo largo de este tiempo a mi

sueño de ser ingeniero, sé haga realidad**.**

Resumen

En la actualidad el software está teniendo un cambio importante en el aspecto en como es desarrollado, como se escala y como se despliega, pasando del modelo de desarrollo más usado conocido como monolito a la arquitectura de microservicios, este tipo de arquitectura de microservicios permite un desarrollo mucho más ágil y más escalable a comparación con el modelo tradicional monolito.

El poder integrarse y consumir data de un sistema externo ya sea una Database o una Api es un proceso muy complicado y en muchos casos confuso por la configuración de este proceso, por lo consecuente se diseño e implemento un microservicio el cual permite realizar estas funcionalidades de integrarse y consumir data externa , este microservicio se desarrolló con el uso de la herramienta Spring Boot especial para el desarrollo Backend en Java o Kotlin , esta herramienta se especializa en el desarrollo de aplicaciones basadas en microservicios y se usará como lenguaje de programación Kotlin que usa una máquina virtual de java, el objetivo del diseño e implementación de este microservicio de integración es poder solucionar dicho problema facilitando el uso y manipulación de la información externa.

En el siguiente proyecto se describe el diseño e implementación de un microservicio el cual permite integrarse a diferentes sistemas de información ya sea una API o una DATABASE, al final resultara un modelo de arquitectura de microservicios más conveniente para dicho microservicio de integración.

Abstract

At present, software is having a big change in how it is developed, how it is scaled and how it is deployed, going from the most used development model, known as monolith to the microservices architecture, this type of microservices architecture allows a lot of development. more agile and more scalable compared to the traditional monolith model.

Being able to integrate and consume data from an external system, be it a database or an API, is a very complicated process and in many cases confusing due to the configuration of this process, therefore a microservice will be designed and implemented which allows these functionalities to be carried out. to integrate and consume external data, this microservice will be developed with the use of the special Spring Boot tool for backend development in Java or Kotlin, this tool specializes in the development of applications based on microservices and will be used as a Kotlin programming language that uses a java virtual machine, the objective of the design and implementation of this integration microservice is to be able to solve this problem by facilitating the use and manipulation of external information.

In the next project we will observe the design and implementation of a microservice which allows integrating different information systems either an API or a DATABASE, in the end we will obtain a more convenient microservice architecture model for said integration microservice.

Tabla de contenido

Lista de tabla

Tabla de ilustraciones

CAPÍTULO 1

1 Descripción del proyecto

1.1 Planteamiento del problema

En HelloGuru se está desarrollando una herramienta llamada HelloTool esta misma es de tipo web, la cual permitirá desarrollar aplicaciones internas, seguras, escalables y de una manera rápida, la arquitectura de la herramienta está orientada a MICROSERVICIOS, un microservicio importante de la herramienta es el microservicio de INTEGRATION el cual permitirá consumir información externa ya sea desde una API ya sea para múltiples endpoints de diferentes APIs externas, ya sea una DATABASE para diferentes bases de datos externas para la cual hay una conexión por cada una de estas bases de datos, o un WEBHOOK todo esto para la manipulación de la información desde las aplicaciones creadas con la herramienta HelloTool.

El problema que tiene la herramienta HelloTool es que en el momento no cuenta con dicha funcionalidad de poder tener integraciones con una API, DATABASE, o un WEBHOOK, limitando al usuario a desarrollar sus aplicaciones internas con data a nivel interna de la misma herramienta, por lo cual se ha decidido desarrollar el microservicio de INTEGRATION que consiste en poder acoplarse a todos estos tipos de datasources externos que con dichas instancias de los diferentes datasources permite la funcionalidad primeramente de consumir información externa y segundo de poder manipular esta información y ver los diferentes cambios en los datasources externos, esto evito al usuario hacer migraciones de información desde sus datasources internos de las diferentes organizaciones a la herramienta HelloTool, ya que con una simple integración consumen y manipulan la información.

1.2 Justificación

La integración con fuentes de información externas (API, DATABASE, WEBHOOK) no es tan simple para el usuario común ya que el propósito de la herramienta HelloTool es poder permitirle a los usuarios desarrollar aplicaciones para procesos internos en organizaciones de una manera rápida, por lo cual en la primera versión de la herramienta HelloTool la funcionalidad de poder tener integraciones con múltiples fuentes de información externa facilito que el usuario de esta herramienta pueda desarrollar las aplicaciones de una manera mucho más rápida, escalables y seguras, atrayendo así a muchos clientes para la herramienta ya que se evitaron todo el proceso de hacer las múltiples migraciones de su información y mediante integraciones obtener y procesar toda la información requerida sin tener que trasladar la data de un hospedador a otro, ahorrando tiempo en desarrollo, perdida de información y seguridad de la misma.

1.3 Delimitaciones

1.3.1 Objetivo general

- Diseñar una arquitectura de microservicios para la integración de información en modelos de negocios de diferentes organizaciones a través del producto HelloTool

1.3.2 Objetivos específicos

- Construir un estado del arte de la arquitectura de microservicios e integración de información.
- Desarrollar el proceso de ingeniería de requerimientos para el microservicio de integración de información del producto HelloTool.
- Diseñar una arquitectura de software del microservicio de integración de información del producto HelloTool.
- Implementar el microservicio de integración de información del producto HelloTool.
- Implementar test unitarios para comprobar el funcionamiento correcto del microservicio de integración de información del producto HelloTool

1.4 Metodología

En la elaboración de este proyecto se implemento el método basado en investigación aplicada, ya que toda la información y conocimiento generado se aplicó en todo el proceso de desarrollo de la herramienta HelloTool, se estudió el conocimiento ya existente sobre el tema y se aplicó dicho conocimiento para el desarrollo y cumplimiento de los objetivos establecidos en el proyecto.

Este proyecto se elaboró teniendo en cuenta las siguientes etapas:

La primera etapa consistió en investigar y estudiar el tema para tener pertinencia del mismo, se consultaron referentes nacionales e internacionales toda la información relacionada a la arquitectura de microservicios y la implementación de los mismos, se hizo un análisis de las herramientas que serían usadas en todo el proceso de desarrollo e implementación.

La segunda etapa se basó en aplicar todo el conocimiento consultado y generado, usando tecnología y estándares de desarrollo actuales para generar un producto a la vanguardia exigido por todos los estándares de desarrollo de software actuales.

En la tercera etapa se implementó y se realizaron sus respectivos test o lo previamente desarrollado, se implementó en un servidor de ambiente de desarrollo y se le desarrollaron y aplicaron sus respectivos tests unitarios y test de integración.

La última etapa consistió en la entrega del producto desarrollado y validado por todo el equipo de desarrollo y su respectivo VP de ingeniería encargado de la supervisión del cumplimiento de todo lo pactado en el proyecto.

CAPÍTULO 2

2 Marco teórico y estado del arte

2.1 Marco conceptual

2.1.1 API

En el mundo del desarrollo de software es muy común el desarrollo y consumo de APIs por su fácil uso, teóricamente una Api es una interfaz de programación de aplicaciones la cual permite el transporte de información entre diversos sistemas externos o internos de las organizaciones.

Las APIs son muy conocidas e implementadas en el entorno WEB por que es ahí donde más uso tienen para soportar el intercambio de información dentro de organizaciones y entre organizaciones, generalmente se refieren como APIs sobre la web, esta cualidad que tienen las APIs ofrece un conjunto de funcionalidades montadas en un servidor web para su uso por diversas aplicaciones.

Una API se usa específicamente para la interacción con un sistema de una organización sin la necesidad de saber cómo es su estructura interna o la tecnología en la cual está desarrollado tal sistema.

Las Apis son implementadas por lo general siguiendo el modelo arquitectónico REST (Representational State Transfer) , las APIs que siguen este tipo de topología son conocidas como RESTful APIs, una característica relevante de las APIs es la utilización de los estándares HTTP/HTTPS, lo cual implica que la implementación de las APIs en el servidor o de las aplicaciones tipo cliente no tienen una forma específica o una manera concreta de ser implementadas.

Una de las ventajas de una API REST es que puede ser implementada usando cualquier tipo de lenguaje de programación como por ejemplo siendo de uso más común : Java, .Net, Python, Ruby, o Node.Js, por lo general en este tipo de lenguajes de programación las APIs tienden a estar bien documentadas para su implementación, otra característica muy importante es el uso del protocolo HTTP como función de acceso y del uso de las URIs, lo cual aporta una separación entre cada petición y respuesta de datos, esto otorga la implementación de servicios más eficientes.

2.1.2 Database

Una Database es una agrupación o conjunto de datos lo cual estos datos pueden pertenecer a un contexto específico, el cual permite almacenar grandes volúmenes de información de manera sistemática, por lo general las bases de datos suelen ser implementadas en diversos tipos de sistemas para tener persistencia de la información y posteriormente ser consultada con usos o para funcionalidades específicas de un sistema.

2.1.3 Webhook

Un webhook es muy usado en el desarrollo web, ya que permite hacer callbacks personalizados, estos llamados son respuestas de peticiones HTTP establecidas por el usuario, los webhook en teoría son de un grado muy simple pero la potencia que ofrece estos a la hora de integrarse con información externa de manera instantánea es muy importante. La manera en que se implementan un webhook es muy fácil ya que consiste en un llamado HTTP POST, este llamado es enviado a una URL especificada por el usuario en respuesta de algún evento , los webhook también aportan la comunicación entre un servidor y otro es decir de servidor a

servidor, este tipo de comunicaciones suelen ser simples y eficaces por no ser conexiones de largas duraciones.

En resumen un WEBHOOK implica que una aplicación pueda recibir diferentes eventos en tiempo real, solo se requiere una conexión HTTP con el uso de una solicitud POST lo cual permite integrarse con cualquier arquitectura existente.

2.1.4 Microservicios

Los microservicios tienen dos orientaciones: el primero es de tipo arquitectónico y el segundo como un estilo de desarrollar software. Con la implementación de los microservicios un producto software se divide en elementos mucho más pequeños e independientes entre sí , a estos elementos se les denomina microservicios, a diferencia de la manera tradicional de arquitectura y desarrollo monolítica en la cual todo se encuentra desarrollado y compilado en un solo lugar , los microservicios funcionan de manera conjunta para llevar a cabo una tarea o una función específica.

Una gran ventaja de este tipo de arquitectura de microservicios es que el producto software se puede escalar de una manera mucho más fácil a comparación con otro tipos de arquitecturas ya que el desarrollo de cada servicio se realiza sin afectar el funcionamiento del resto de servicios, otra ventaja es que el producto software queda de manera distribuida por lo cual cada servicio es desplegado de manera individual y por último la comunicación entre los diferentes servicios se realiza a través de API facilitando que cada servicio tenga su grupo de desarrolladores eliminando la necesidad de que el desarrollador tenga que entender todo el código del software.

En resumen la arquitectura de microservicios tiene las siguientes ventajas :

- Despliegue individual de cada servicio del software
- Facilita las actualizaciones ya que solo se necesita modificar el servicio que se desea modificar y por esto no se de tiene el funcionamiento de toda la aplicación.
- Permite la utilización de múltiples lenguajes de programación de acuerdo a los requerimientos del proyecto
- La escalabilidad que aporta esta arquitectura ya que se encuentra dividido el software en múltiples servicios
- Facilita la implementación de los test y las pruebas y reduce los errores de programación
- La entrega continua que se tiene de cara con el cliente

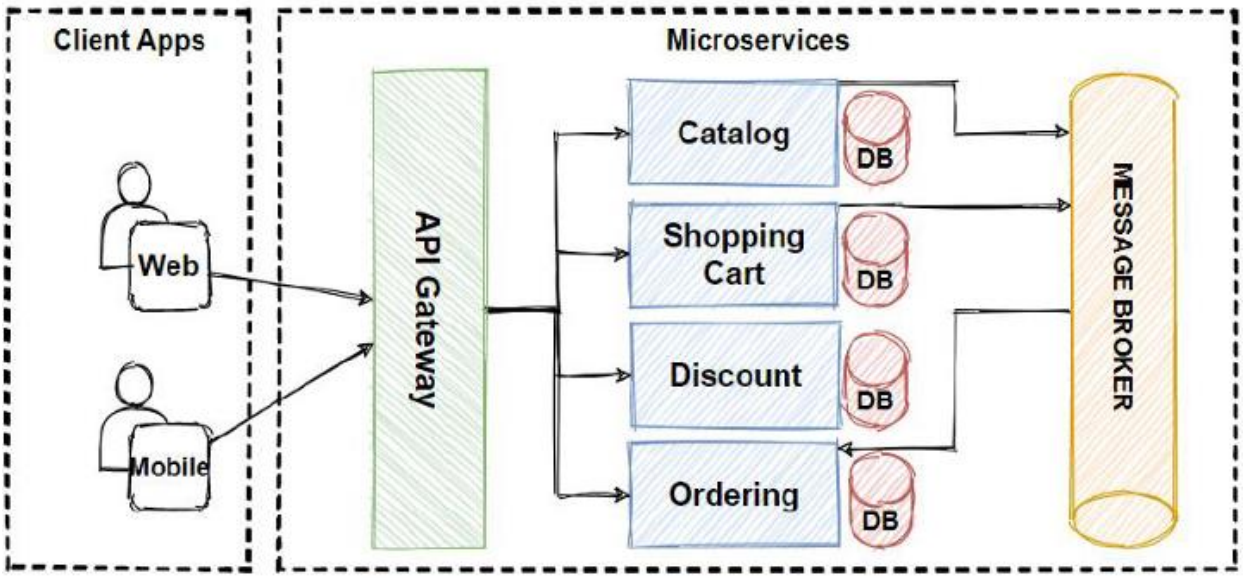

Ilustración 1 arquitectura de microservicios

(ICHI.PRO, 2021)

2.1.5 Arquitectura Monolítica

La arquitectura monolítica ha sido por mucho tiempo la manera de desarrollar software más común o tradicional para los desarrolladores más veteranos o con más experiencia, de manera muy simple la arquitectura Monolítica es aquella en que todas las funcionalidades de un software quedan de manera integradas en un sola parte sujetos a un solo programa, la ventaja de este tipo de arquitectura monolítica es que es mucho más fácil de desarrollar los programas el despliegue y la ejecución son mucho mas fácil a comparación con la arquitectura de microservicios ya que el consumo en servidores de los microservicios es mucho más alto que la de un monolito.

La gran desventaja que tiene este tipo de arquitectura monolítica es la escalabilidad del software y la dificultad para los desarrolladores ya que necesitan entender todo el código del software para procesos de optimización o features nuevos, estos problemas hacen que los desarrolladores dejen este tipo de arquitectura monolítica y adopten la arquitectura de microservicios aunque su facilidad y bajo coste siguen siendo muy interesante para desarrollos con bajo requerimientos.

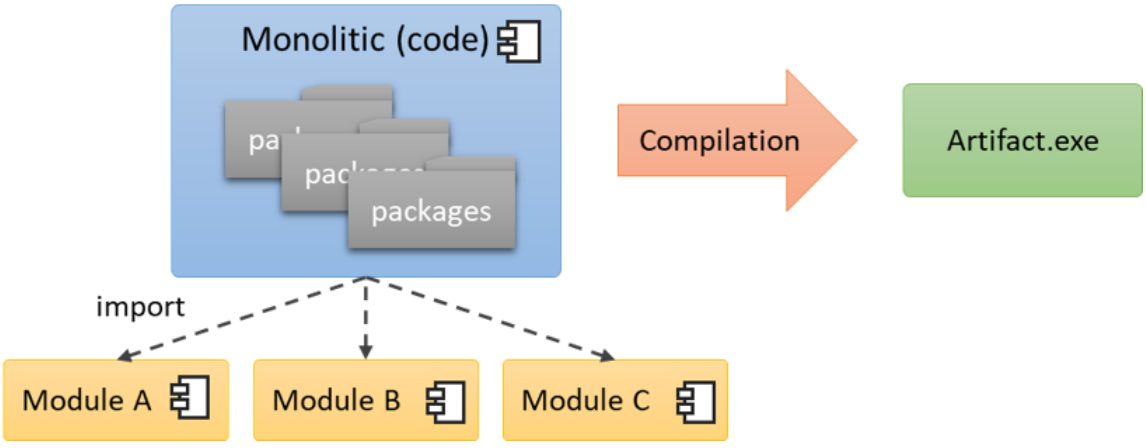

Ilustración 2 arquitectura monolítica

(Scribbr, 2020)

2.1.6 Arquitectura de Microservicios

La implementación de una buena arquitectura de microservicios otorga una serie de ventajas el cual permite desplegar una aplicación relativamente grande como conjuntos de pequeñas aplicaciones (microservicios) que se pueden desarrollar , desplegar , escalar de forma individual.

Existen múltiples patrones de implementación de arquitecturas de microservicios, pero existen 3 topologías que son las mas destacadas, comunes o populares, primero basadas en API REST (Representational State Transfer) o en español (Transferencia de Estado Representacional), la segunda basada en aplicaciones REST, y por último centralizada de mensajería.

2.1.7 Topología API REST

Este tipo de topología consiste en componentes de servicios que tienen un modulo o más, que realizan funciones bien específicas e independientes de los demás servicios, en este tipo de arquitectura cada componente de servicio se accede mediante una interfaz basada en REST conectada a través de una API basado en WEB y desplegada por separado, este tipo de arquitectura es muy recomendable para sitios WEB que exponen servicios por separados y de cierta manera autónomos a través de algún tipo de API.

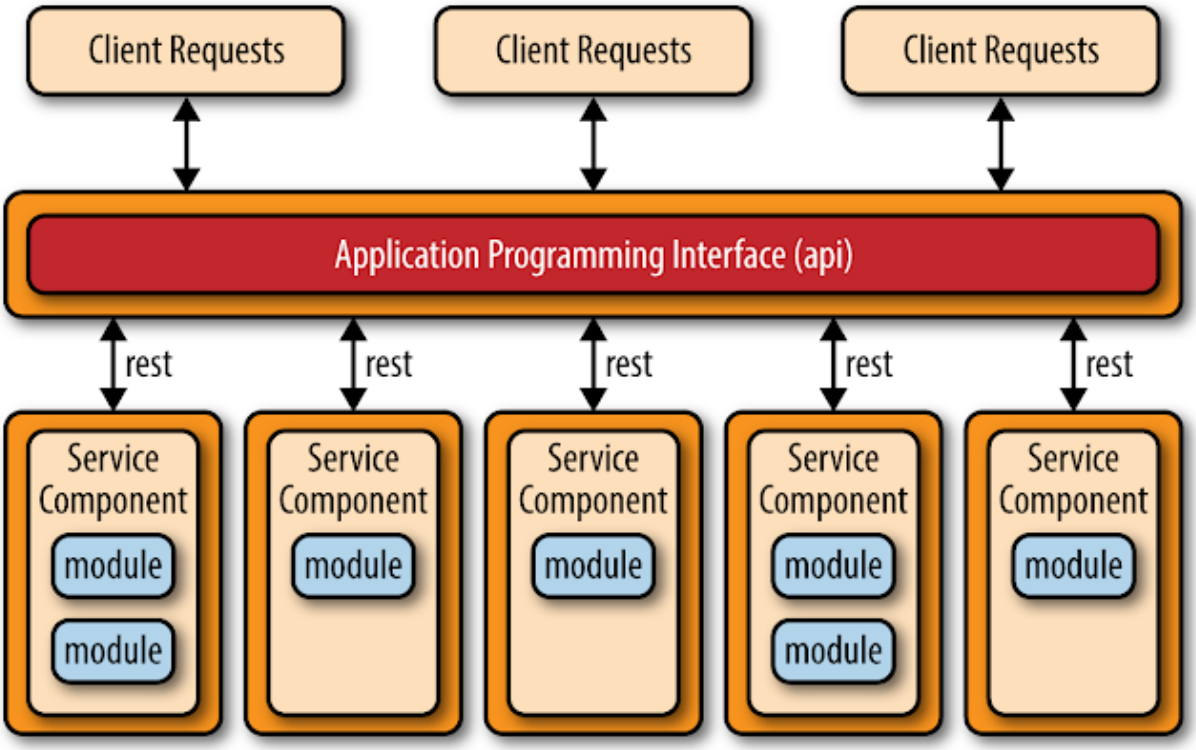

Ilustración 3 topología API REST

(Goette, 2017)

2.1.8 Ventajas y contras de la arquitectura de microservicios

VENTAJAS	**CONTRAS**
Los microservicios otorgan una gran versatilidad ya que permite la implementación de múltiples tecnologías y lenguajes de programación.	El uso de esta arquitectura de microservicios implica que cada microservicio este por separado uno de otro esto conlleva a que el uso de test globales sean un poco más complicados con respecto a la arquitectura

	monolítica.
Proporcionan una entrega continua y facilidad de escalar con aplicaciones de terceros.	Es muy importante llevar el control de los microservicios existentes ya que entre más existan microservicios para dar solución es mucho más complicado llevar el control e integrarlos.
La manera en que se despliega cada microservicio por individual lo cual encaja muy bien dentro de las metodologías ágiles de desarrollo.	Implementar este tipo de arquitectura de microservicios implica tener un buen equipo de desarrolladores con un nivel alto de experiencia.
Las funcionalidades desarrolladas bajo este tipo de arquitectura de microservicios otorgan mejoras o actualizaciones rápidas y continuas para cada funcionalidad.	Implica llevar una gestión del control de versiones muy detallada.
El mantenimiento del desarrollo por lo general es más simplificado y barato , permitiendo que las mejoras se puedan hacer a un microservicio en específico sin afectar el funcionamiento normal de la aplicación.	Una de las grandes desventajas que implica usar este tipo de arquitectura de microservicios es que suele ser muy costosa de implementar debido al alto costo en hardware y licenciamiento de terceros.
La fácil integración de diversas funcionalidades que han sido desarrolladas	

por terceros.	
Un proyecto basado en la topología de microservicios evoluciona de una forma mucho más natural y tiene una gestión del desarrollo muy fácil ya que los diferentes desarrolladores no necesitan conocer toda la estructura del proyecto	

Tabla 1 ventajas y contras de la arquitectura de microservicios

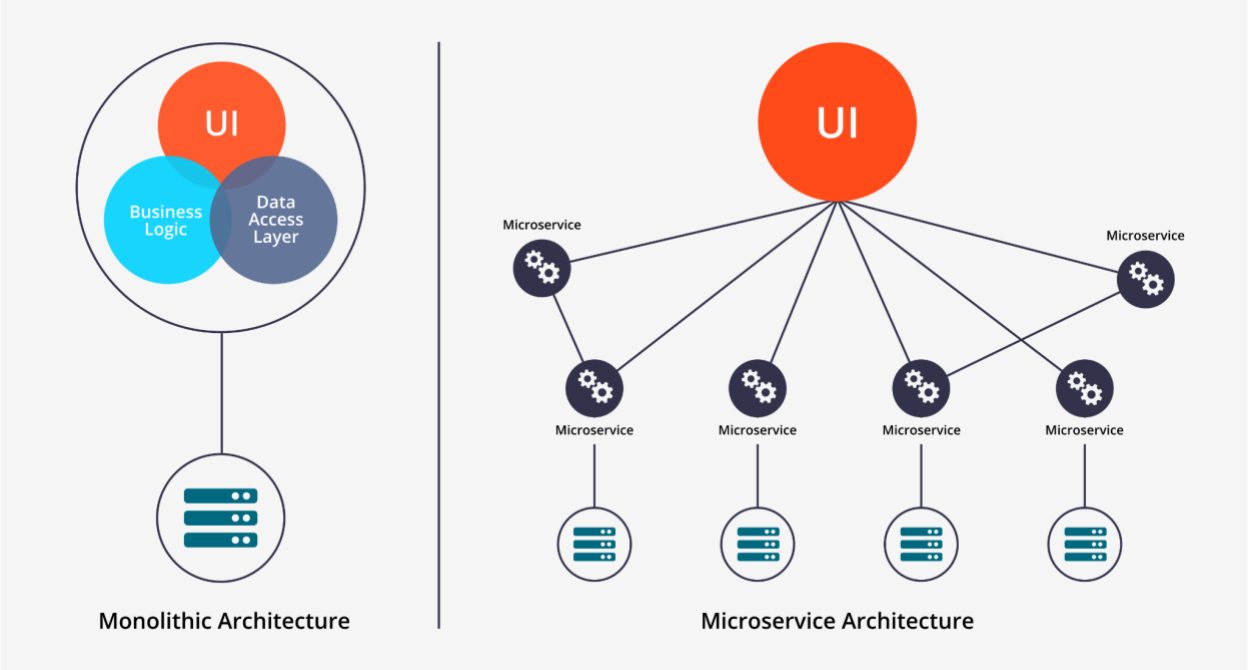

Ilustración 4 arquitectura monolítica vs arquitectura de microservicios

(mrirfanto, 2018)

2.1.9 Ventajas y contras de la arquitectura tradicional monolítica.

VENTAJAS	**CONTRAS**
Este tipo de arquitectura monolítica otorga un fácil desarrollo debido a que todo está en un mismo componente, para un equipo de tamaño pequeño es mucho fácil iniciar un nuevo proyecto y colocarlo en producción rápidamente	Anclado a un Stack tecnológico: Debido a que todo el software es una sola pieza, implica que utilice el mismo Stack tecnológico para absolutamente todo, lo que impide que aprovechemos todas las tecnologías disponibles.
Fácil de escalar: Solo es necesario instalar la aplicación en varios servidores y ponerla detrás de un balanceador de carga.	Escalado Monolítico: Escalar una aplicación Monolítica implica escalar absolutamente toda la aplicación, donde se agotan recursos para funcionalidades que quizás no necesitan ser escaladas.
Pocos puntos de fallo: El hecho de no depender de nadie más, mitiga gran parte de los errores de comunicación, red, integraciones, etc. Prácticamente los errores que pueden salir son por algún bug del programador, pero no por factores ajenos.	El tamaño sí importa: sin albur, las aplicaciones monolíticas son fácil de operar con equipo pequeños, pero a medida que la aplicación crece y con ello el equipo de desarrollo, se vuelve cada vez más complicado dividir el trabajo sin afectar funcionalidad que otro miembro del equipo también está moviendo.

Autónomo: Las aplicaciones monolíticas se caracterizan por ser totalmente autónomas (autosuficientes), lo que les permite funcionar independientemente del resto de aplicaciones.	Versión tras versión: Cualquier mínimo cambio en la aplicación implica realizar una compilación de todo el artefacto y con ello una nueva versión que tendrá que ser administrada.
Performance: Las aplicaciones Monolíticas son significativamente más rápidas debido que todo el procesamiento lo realiza localmente y no requieren consumir procesos distribuidos para completar una tarea.	Si falla, falla todo: A menos que se tenga alta disponibilidad, si la aplicación Monolítica falla, falla todo el sistema, quedando totalmente inoperable.
Fácil de probar: Debido a que es una sola unidad de código, toda la funcionalidad está disponible desde el inicio de la aplicación, por lo que es posible realizar todas las pruebas necesarias sin depender de nada más.	Es fácil perder el rumbo: Por la naturaleza de tener todo en un mismo módulo es fácil caer en malas prácticas de programación, separación de responsabilidades y organización de las clases del código.
	Puede ser abrumador: En proyectos muy grandes, puede ser abrumador para un nuevo programador hacer un cambio en el sistema.

Tabla 2 ventajas y contras de la arquitectura monolítica

(Scribbr, 2020)

2.1.11 Topología basada en REST

Este tipo de topología o arquitectura se diferencia de la topología API REST ya que las solicitudes del cliente llegan a través de pantallas de aplicaciones empresariales muy comunes basadas en web o en clientes, en lugar de a través de una capa API.

La capa de interfaz de usuario de la aplicación se despliega de manera individual como una aplicación web, está accede de forma remota a componentes de servicio desplegados por separado a través de simples interfaces basadas en REST.

Los servicios en este tipo de topología o arquitectura se diferencia de los servicios de la topología basada en API-REST en que los servicios por lo general tienen la tendencia de ser más grandes. Este tipo de topología es muy recomendable implementar en aplicaciones de tamaño y complejidad relativamente pequeñas o medianas.

2.1.12 Topología de Mensajería central

Este tipo de topología o arquitectura es muy similar a la arquitectura basada en REST se diferencia en el punto que en vez de usar un REST para el acceso la topología de mensajería centralizada usa un intermediario de mensajes centralizado, el intermediario usado para la mensajería centralizada que se encuentra en este tipo de topologías no realiza ninguna orquestación, transformación o enrutamiento, es sólo un transporte para acceder a los diferentes servicios que se encuentran en remoto.

Este tipo de arquitectura de mensajería centralizada se encuentra muy comúnmente en aplicaciones de negocios mucho más grandes o aplicaciones que requieren un control más refinado sobre la capa de transporte entre la interfaz de usuario y cada servicio.

Los beneficios de este tipo de arquitectura o topología sobre la simple topología basada en REST son dispositivos avanzados de colas, mensajería asíncrona, monitoreo, manejo de errores y mejor balanceo de carga y escalabilidad

2.2 Tecnologías implementadas y usadas en la etapa del desarrollo

2.2.1 Kotlin

En el año 2011 la compañía JetBrains dio a conocer el proyecto, Kotlin es un lenguaje de programación el cual tiene un tipado estático este mismo es ejecutado sobre una máquina virtual de java, fue desarrollado por JetBrains en Rusia , este nombre proviene de la isla de kotlin la cual se encuentra en la región de San Petersburgo.

En el mes de enero del 2016 fue nombrado el lenguaje del mes por la revista Dr.Dobb' Journal, Kotlin fue diseñado para ser implementado junto con java , y es dependiente de muchas de las bibliotecas y clases de Java, en la actualidad Kotlin es muy conocido por su gran uso en el desarrollo de aplicaciones para ANDROID pero también tiene un muy buen uso en el desarrollo de microservicios con SPRING BOOT.

Una gran ventaja de este lenguaje de programación es que permite emplear todo el conocimiento de JAVA , esto permite escribir mucho menos código y más sencillo de entender.

El gigante Google ha impulsado el uso de este lenguaje en el desarrollo de aplicaciones móviles tratando de dejar a un lado el uso de JAVA y promoviendo más el uso de Kotlin en sus proyectos para ANDROID.

Ilustración 5 Kotlin

(Cupertino, 2020)

2.2.3 Spring Boot

Primero hay que diferenciar entre Spring Boot y Spring Framework, spring boot viene siendo una extensión de Spring Framework y forma parte de este framework , en la actualidad Spring Boot es muy usado en todo el mundo que concierne a JAVA.

Spring Boot nace de la necesidad de desarrollar aplicaciones de manera rápida y fácil, con el objetivo de que los desarrolladores no escriban las mismas configuraciones repetitivamente esto implica que los diversos desarrolladores se enfoquen más en el desarrollo de la lógica de negocio, este objetivo se cumple ya que Spring Boot posee una serie de bibliotecas que dichas colecciones ya traen una lista de dependencias preconfiguradas que se necesitan para iniciar una funcionalidad en específico, Spring Boot otorga a los desarrolladores una lista grande de configuraciones automáticas por defecto que permite que un aplicativo web pueda ser construido , desplegado de una manera muy rápida, sencilla y segura, este tipo de configuraciones pueden ser personalizadas de acuerdo a los requerimientos del sistema o del proyecto a construir.

Características adicionales de Spring Boot:

- Posee todas las características de Spring Framework.
- Implementa el principio de diseño de Abierto-Cerrado, donde esta es cerrada a modificación, pero abierta a extensión cuando se desee tener un mayor control de sus componentes.
- Posee servidores de aplicaciones y contenedores de Servlet embebido.
- Eliminar la necesidad de configurar la aplicación por medio de código o XML. Para ello utiliza ficheros .properties o .yml.
- Es una gran opción para crear aplicaciones basadas en microservicios.
- Posee configuraciones automáticas para diferentes Frameworks como son Spring Security, Spring Batch y otros.
- Ofrece un amplio soporte a diferentes tecnologías como son bases de datos relaciones y no relacionales, almacenamiento en caché, mensajería, procesamiento por lotes y más
- Ofrece métricas de la aplicación por medio de una librería llamada Actuator, utilizada para exponer información importante sobre la aplicación que se encuentra en ejecución a través del monitoreo.

Tabla 3 características adicionales de Spring Boot

(Ramírez Pérez, 2020)

Ilustración 6 Spring Boot

(Guerrero, 2019)

2.2.4 PostgreSQL

PostgreSQL es un gestor de bases de datos tipo relacional orientado a objetos , también es de código abierto lanzado en 1996 y desde entonces es uno de los sistemas de gestión de base de datos más usados entre toda la comunidad de desarrolladores, es de código abierto y como los múltiples proyectos de código abierto que existen no es manejado por una empresa o por una persona si no por una comunidad muy grande desarrolladores que trabajan sin ánimos de beneficiarse económicamente, PostgreSql es muy versátil ya que otorga una flexibilidad y permite el uso de varios tipos de lenguajes como el PL/pgSQL, PL/Tcl, PL/Perl

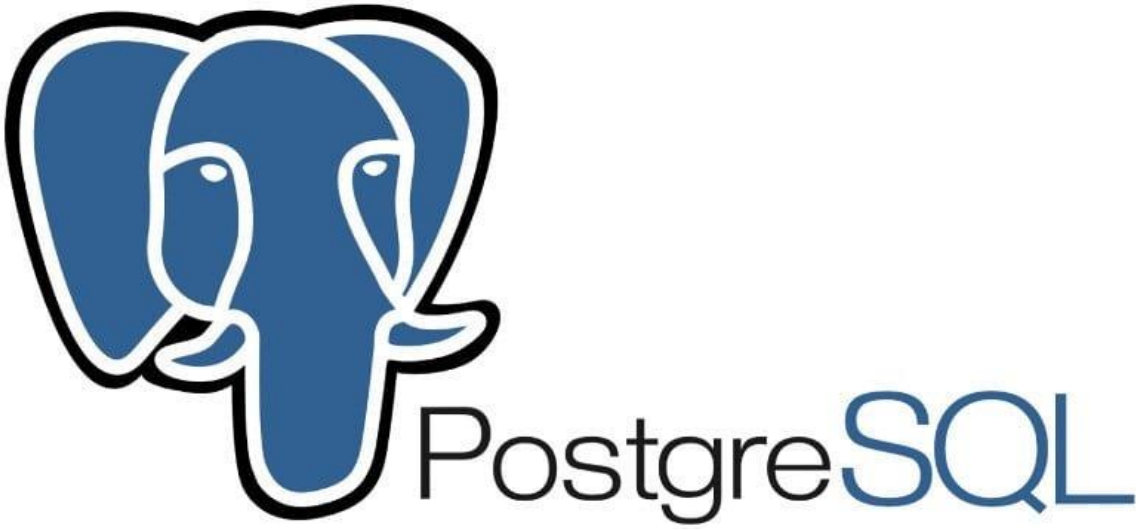

Ilustración 7 PostgreSQL

(Scribbr, 2019)

2.2.5 IntelliJ IDEA

Intellij IDEA es un IDE un entorno de desarrollo integrado para elaborar diferentes tipos de programas informáticos. Fue desarrollado por JetBrains , este IDE se encuentra en dos versiones la versión comercial y la versión para la comunidad, la primer versión de este IDE fue en enero del 2001 en aquel tiempo solo soportaba el desarrollo para JAVA en la actualidad este IDE soporta muchos lenguajes de programación como : Java, Clojure, Rust, Dart, Erlang, Go, Groovy, Haxe, Perl, Scala, Kotlin y entre otros muchos más.

Este IDE permite integrarnos con una gran variedad de plugins y herramientas externas como para el control de versiones con Git, hacer Debugger, implementar tests unitarios y tests de integración entre otras funcionalidades que brinda este IDE.

Ilustración 8 IntelliJ IDEA

(JET BRAINS, 2021)

2.2.6 Postman

Postman nace como una herramienta con el propósito principalmente para crear peticiones sobre APIs y testear dichas APIs de forma simple y fácil de usar, a continuación ciertas características que posee la herramienta.

- Crear Peticiones, permite crear y enviar peticiones http a servicios REST mediante un interfaz gráfico. Estas peticiones pueden ser guardadas y reproducidas a posteriori.
- Definir colecciones, mediante Postman se puede agrupar las APIs en colecciones. En estas colecciones se puede definir el modelo de autentificación de las APIs para que se añadan en cada petición. De igual manera se puede ejecutar un conjunto de test, así como definir variables para la colección.
- Gestionar la Documentación, genera documentación basada en las API y colecciones que se han creado en la herramienta. Además esta documentación se puede hacer pública.
- Entorno Colaborativo, permite compartir las API para un equipo entre varias personas. Para ello se apoya en una herramienta colaborativa en Cloud.

- Genera código de invocación, dado un API es capaz de generar el código de invocación para diferentes lenguajes de programación: C, CURL, C#, Go, Java, JavaScript, NodeJS, Objective-C, PHP, Python, Ruby, Shell, Swift.
- Establecer variables, con Postman se pueden crear variables locales y globales que posteriormente se utilizan dentro de las invocaciones o pruebas.
- Soporta Ciclo Vida API management, desde Postman se puede gestionar el ciclo de vida del API Management, desde la conceptualización del API, la definición del API, el desarrollo del API y la monitorización y mantenimiento del API.
- Crear mockups, mediante Postman, se puede crear un servidor de mockups o sandbox para que se puedan testear nuestras API antes de que estas estén desarrolladas.

(Cuervo, 2019)

Ilustración 9 Postman

(Ludim, 2018)

2.3 Estado del arte

2.3.1 Internacional

Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web

En la actualidad, a nivel empresarial y tanto en el sector privado como público se realiza desarrollo de software para suplir las necesidades de automatización de procesos internos, este desarrollo ha seguido las tendencias impuestas por la plataforma, lenguajes de programación o por la experiencia del área de desarrollo, lo cual deviene en la implantación de sistemas de construcción tradicional o monolítico. El proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software monolítica, misma que ha sido adoptada por el uso de un lenguaje de programación específico para construcción de aplicaciones web empresariales; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas. (Lopez & Maya, 2017)

Arquitectura de Microservicios para Plataformas de Integración

La integración de sistemas heterogéneos se apoya comúnmente en Plataformas de Integración (PI). Estas plataformas consisten en infraestructura especializada, que provee mecanismos para resolver incompatibilidades entre sistemas con el fin de posibilitar su comunicación. En este ámbito, el avance y la expansión de la computación en la nube ha generado nuevos escenarios y requerimientos sobre las PIs en términos de escalabilidad y eficiencia, entre otros. Por otro lado, la arquitectura de microservicios ha surgido recientemente

impulsada por la industria, y está ganando creciente popularidad. Esta posee ventajas como el escalamiento independiente y la mantenibilidad que podrían beneficiar a las PIs en distintos escenarios. Por tanto, resulta de interés explorar las ventajas, desafíos y alternativas que introduce la arquitectura de microservicios en estas plataformas. Esta tesis estudia la aplicabilidad de la arquitectura de microservicios para la construcción de PIs. Para esto se analiza, por un lado, el impacto de utilizar dicha arquitectura en PIs, determinando escenarios en los cuales es propicia su aplicación. Por otro lado, se proponen alternativas de arquitectura y diseño para la construcción de PIs basadas en microservicios, las cuales son evaluadas en base a distintos factores. En relación al análisis de impacto, se plantea una metodología para determinar cómo es afectada una PI en base a sus atributos de calidad, obteniéndose del análisis dos resultados principales. Por un lado, se determina cómo impacta aplicar una arquitectura de microservicios en la calidad de la plataforma. Por otro, se determina un conjunto de características de los escenarios de integración (p. ej. cantidad alta de sistemas a integrar), que permite identificar si un escenario se beneficia al utilizar una PI basada en microservicios. En relación a las propuestas de arquitectura y diseño, se presenta una propuesta principal aplicable a diversos escenarios, así como variantes aplicables a contextos específicos. La propuesta principal se evalúa en base a tres factores: i) patrones y buenas prácticas de microservicios, ii) atributos de calidad de la PI, y iii) la construcción de un prototipo. Se concluye en este trabajo que la arquitectura de microservicios es aplicable para la construcción de PIs en escenarios con determinadas características y que las propuestas de arquitectura planteadas siguiendo este enfoque son técnicamente viables. (Nebel, 2018)

Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura

La arquitectura de microservicios, o microservicios es un nuevo concepto que ha crecido en popularidad en los últimos años en la comunidad del desarrollo de sistemas software. A diferencia de los sistemas monolíticos, donde la lógica de negocio se encuentra en un único proceso con dependencias fuertes entre las unidades, los microservicios pretenden desacoplar la lógica de negocio en servicios independientes que se comunican entre ellos con un mecanismo ligero. Estos servicios pueden estar implementados en diferentes lenguajes de programación y usar distinta tecnología de almacenamiento. Son fácilmente escalables y desplegados en un sistema distribuido debido a su débil acoplamiento entre componentes y a su carácter distribuido. La aparición del término microservicios ha originado "ríos de tinta" sobre este concepto y su aplicación. Entre otras cuestiones, algunos autores hablan ya de un estilo arquitectónico de microservicios. El objetivo de este trabajo de investigación es revisar toda la literatura que existe desde el año 2014, fecha en la que apareció el artículo de Martin Fowler: Microservices, (Fowler & Lewis, 2014) que traten específicamente sobre la arquitectura de microservicios, es decir, aquellos que presenten un nuevo modelo arquitectónico o arquitectura de referencia, desligado del concepto SOA. Por lo tanto, este estudio intenta dilucidar: a) si existe tal estilo y b) cómo está definido en términos de sus elementos constituyentes y restricciones topológicas y de comunicación, así como el lenguaje que lo describe. Mediante la revisión de la literatura basada en la búsqueda sistemática en bases de datos científicas, se han seleccionado aquellos artículos que responden a las preguntas de investigación previamente definidas. Una vez realizada la extracción y síntesis de los datos sobre los estudios primarios seleccionados se concluye que actualmente no hay estilos arquitectónicos propios ni lenguajes descriptivos asociados a los microservicios. Debido a las características particulares, en el que un sistema se divide en

numerosos servicios independientes y autónomos, resulta complicado tener una visión completa de la arquitectura. No obstante, este diseño es cada vez más utilizado por distintas empresas y organizaciones, con el fin de abordar una mayor complejidad empresarial con reglas de negocio cambiantes. Por lo tanto, se muestra la necesidad de definir un estilo arquitectónico que sirva de referencia en la construcción de sistemas informáticos reutilizables basados en microservicios. (Gomez, 2018)

2.3.2 Nacional

Arquitectura de microservicios

Para dar comienzo a este documento es importante iniciar por conocer el contexto de una manera muy breve, en él surgió el término microservicio y cómo fue madurando para que el día de hoy sea utilizado con el fin de describir un estilo arquitectural, como lo son las llamadas arquitecturas de microservicios. La primera pregunta es de dónde nace el término microservicio; pues bien, de acuerdo con Martin Fowler (Ingeniero de sistemas británico, autor de múltiples publicaciones, y orador público en diseño de software empresarial) el término microservice (microservicio en español) fue discutido en un taller para arquitectos de software cerca de la ciudad de Venecia en mayo del 2011, se utilizó para describir un estilo arquitectural que muchos habían empezado a explorar recientemente. Más tarde, en mayo de 2012, este mismo grupo decide optar por el nombre de microservicios como el término más apropiado para describir sus experiencias. Algo que se debe tener en cuenta es que aunque el término puede ser relativamente nuevo, especialmente a partir del 2014 cuando se convirtió en uno de los términos más populares, atrayendo mucha atención, como una nueva manera de pensar en la estructuración de las aplicaciones, su idea base ha existido por mucho tiempo; adopta, aparentemente, los mismo

ideales de SOA (Service Oriented Architecture), tanto que para muchos no es fácil ver una diferencia real. Sin embargo, es importante recalcar que durante los últimos años el término ha sido utilizado para describir un modo particular para diseñar y estructurar software como un conjunto de servicios desplegables independientemente. Y aunque a pesar de su auge no existe una definición precisa de este estilo arquitectural, sí cuenta con una serie de características que giran alrededor de la organización, su lógica del negocio, despliegues automáticos, control descentralizado del lenguaje de programación y datos.

Con el fin de facilitar el entendimiento de este estilo arquitectural de microservicios, es importante tener como referencia el estilo monolítico, este último se centra dotándolo muy superficialmente en construir una aplicación como una unidad en la que se construyen usualmente aplicaciones empresariales, como una unión de tres partes: la parte cliente, la parte encargada del manejo de datos y la parte servidor que contiene la lógica de dominio de la aplicación. El emplear un estilo monolítico en aplicaciones puede tener resultados exitosos, sin embargo, el paso del tiempo ha generado que más personas se sienten frustradas, puesto que los cambios constantes en el modelo de negocio (lo que se transforma en cambios de requerimientos) causan que se tenga que modificar toda la aplicación haciéndola más grande, lo que lleva a que mantener este tipo de aplicaciones modulares se vuelva más complicado. Otro tema muy importante sobre todo en esta época es el permitir crecer con cierta funcionalidad del sistema, para solucionar esto, se incrementa la capacidad de toda la aplicación mas no del segmento que realmente lo requiere. Para hacer frente a esta problemática, la arquitectura de microservicios establece construir una aplicación como un conjunto de servicios, donde cada uno de estos es independiente del otro y hasta pueden ser escritos en lenguajes diferentes y mantenidos por equipos diferentes. Lo anterior permite que el problema de escalar se solucione

incrementando el procesamiento necesario para el servicio específico que lo esté requiriendo y no de otros que no lo necesitan. (Barrios C., 2017)

Un acercamiento a los microservicios

El desarrollo basado en microservicios es una tendencia emergente que surge de las necesidades de la industria de software, para mejorar la escalabilidad y flexibilidad de las aplicaciones web y que hoy en día se reconoce como un nuevo modelo de arquitectura conocido como microservicios. Generalmente, las aplicaciones web tradicionales siguen una arquitectura por capas, en las que se hace separación lógica de la solución en tres capas: la interfaz de usuario, la lógica de la aplicación y el sistema de gestión de datos; sin embargo, el despliegue de la solución se realiza como una unidad monolítica que se ejecuta en un solo espacio de direcciones, generando problemas ante la demanda de aplicaciones con servicios especializados que requieren manejo de grandes volúmenes de datos integrados con enfoques IoT y con requerimientos de procesamiento distribuido. Las arquitecturas basadas en microservicios proponen una arquitectura en la que cada funcionalidad de negocio se descompone en servicios web altamente cohesivos que pueden ser desplegados, evolucionados y escalados de manera independiente. Este enfoque trae beneficios como la flexibilidad, escalabilidad y productividad del equipo de trabajo, pero también conlleva nuevos retos que están siendo enfrentados, como la seguridad, el desempeño y la mantenibilidad de la solución. Uno de los aspectos más importantes para tener en cuenta en el desarrollo de este tipo de aplicaciones es el alineamiento organizacional a través de un enfoque de desarrollo orientado al dominio (DDD). (Gomez S. & Cano, 2018)

CAPÍTULO 3

3 Análisis preliminar

Este capítulo se centra en el análisis de las herramientas que ayudan al desarrollo de microservicios y a su vez observar y determinar cada una de sus características de dichas herramientas con el objetivo de recolectar toda la información y tomar una mejor decisión de acuerdo a los requerimientos de la empresa para el desarrollo de microservicios, no falta recalcar que dentro del mercado y del mundo del desarrollo de software hay una gran variedad de tecnología en la cual se pueden desarrollar microservicios, también se entrará a observar las diferentes herramientas que ayudan a la gestión de microservicios y el control de versiones de los mismos.

3.1 Spring Boot Starters

Los Starters son un conjunto de descriptores de las dependencias de spring boot que pueden ir incluidas en su aplicación, esto obtiene una especie de puerta el cual da acceso a todo lo que ofrece spring y a todo lo que llegue a necesitar su aplicación sin tener que buscar el código por defecto copiar y pegar, estos starters lo que permite es cargar mediante dependencias la tecnología que necesite o se requiere.

3.2 JPA

JPA es una API que otorga JAVA para implementar un Framework Object Relational Mapping es decir un (ORM), el cual permite manipular la base de datos mediante objetos, es

decir JPA es el encargado de convertir los objetos de JAVA en este caso los de kotlin en instrucciones para el manejador de la base de datos (MDB).

3.3 Jackson-module-kotlin

Jackson es una librería la cual sirve para serializar y deserializar JSON, en este caso se usó el módulo de kotlin de jackson para parsear diccionarios y listas como JSON o viceversa , este módulo es de gran utilidad ya que permite estructurar la data de acuerdo al response que se quiere en específico en los diferentes endpoints que llevaría el microservicio.

3.4 Control de versiones y Repositorio

3.4.1 SmartGit

SmartGit es un cliente gráfico de Git con soporte para SVN y pull request para GitHub y BitBucket. Tiene versiones para Windows, macOS y Linux. Actualmente (febrero 2020) está en la versión 19.1.

Características:

- El cliente tiene la misma interfaz de usuario para Windows, macOS y Linux.
- Oculta la complejidad de Git para el usuario.
- Permite realizar acciones como merge y commit de forma gráfica.
- Incluye cliente de línea de comandos para Git (Windows y macOS).
- Git-Flow.
- Cliente SSH.
- Comparador de archivos.

- Combinador de archivos (resuelve conflictos de sincronización).
- La licencia cuesta US$ 59 (la licencia más reducida) al año.
- En algunas páginas dice que hay una versión gratuita para uso no comercial, pero en la página oficial no hay información sobre dicha versión.

(Mayanga, 2020)

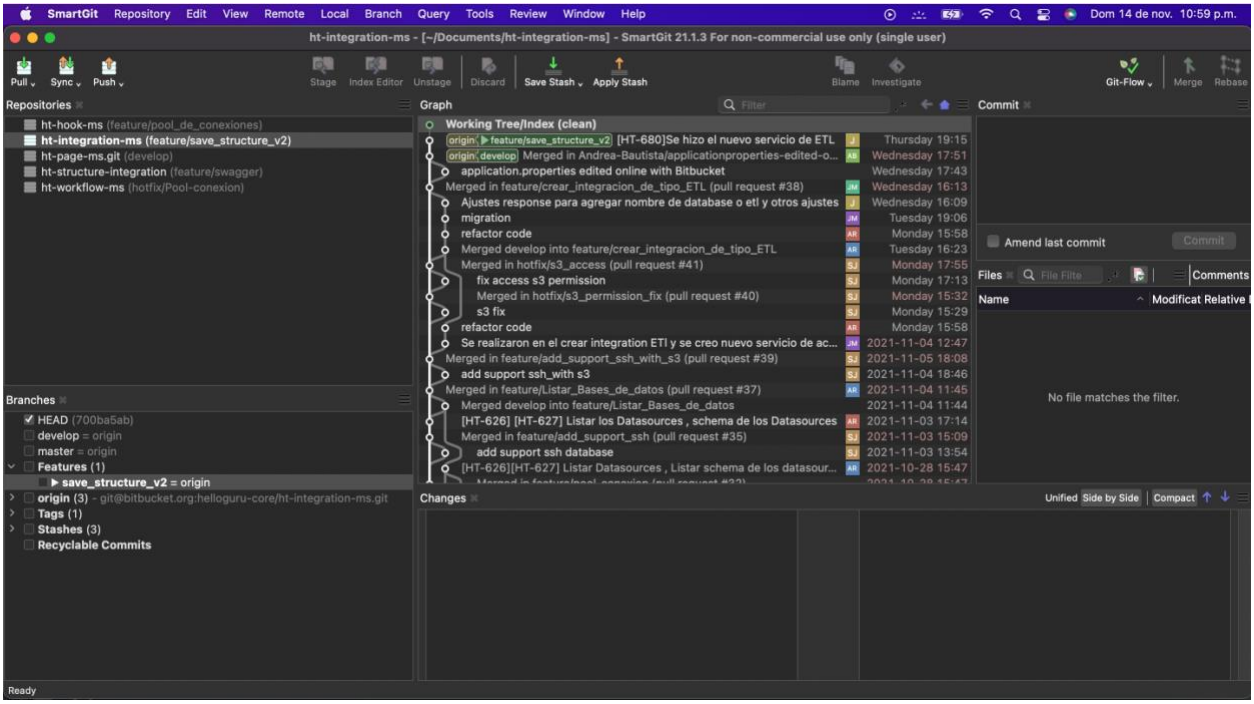

Ilustración 10 SmartGit

3.4.2 Bitbucket

Bitbucket es una plataforma basada en web el cual permite gestionar los proyectos los cuales los sistemas de versiones Mercurial y Git, esta plataforma ofrece tanto el plan comercial como el plan gratuito, fue lanzado en el 2008 por Atlassian y viene tanto en la versión en inglés como en la versión en español, este así mismo ofrece diversas integraciones como con Trello, Slack, Amazon web Services, Google Cloud Platform, npm y diversos sistemas, también ofrece la opción de establecer la integración continua y todos los pipelines.

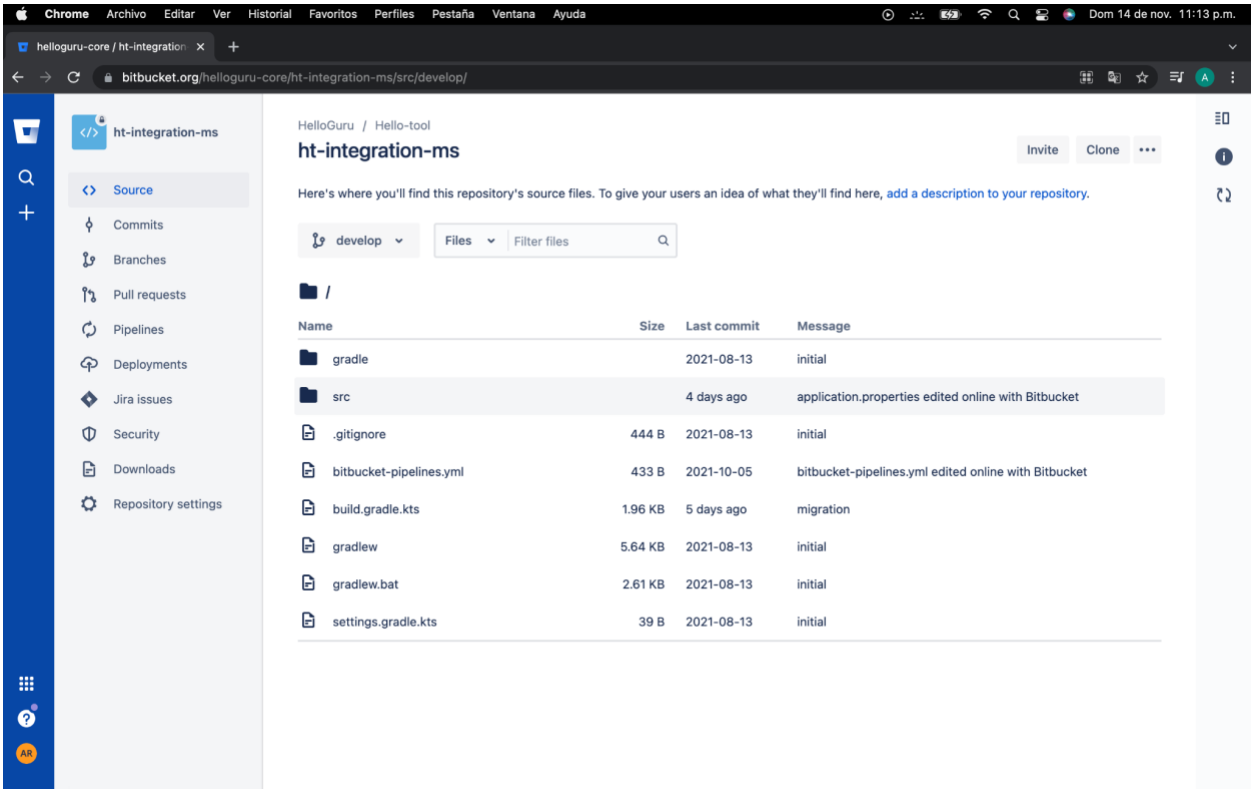

Ilustración 11 Bitbucket

3.4.3 Jira

Jira es una herramienta WEB la cual permite la gestión de las tareas de un proyecto, permite el seguimiento de errores e incidencias, esta herramienta también permite organizar los flujos de trabajo y los estados de avance de un proyecto, dicha herramienta fue desarrollada por la compañía Atlassian

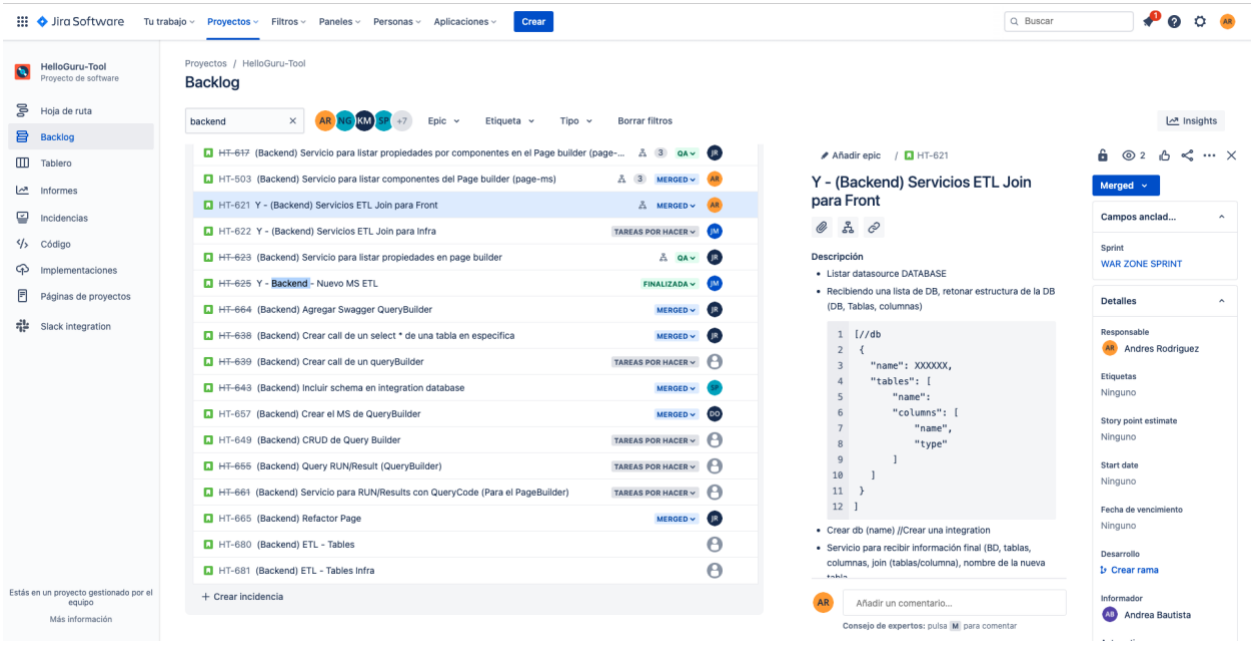

Ilustración 12 Jira

CAPÍTULO 4

4. Requerimientos de Ingeniería para el microservicio de Integración

En este capítulo se llevo acabo todo el proceso de ingeniería para establecer los requerimientos para el desarrollo del microservicio de integración de la HelloTool con el objetivo de definir con claridad todos los posibles casos de uso del flujo de integración de los diferentes tipos de data sources.

4.1 Casos de uso

Los casos de uso otorgan una mejor observabilidad de las funcionalidades de la aplicación, con esto se podrá observar y analizar los requerimientos funcionales y no funcionales, por lo tanto, en esta sección detallo los flujos que se implementaron.

4.2 Flujo general del microservicio

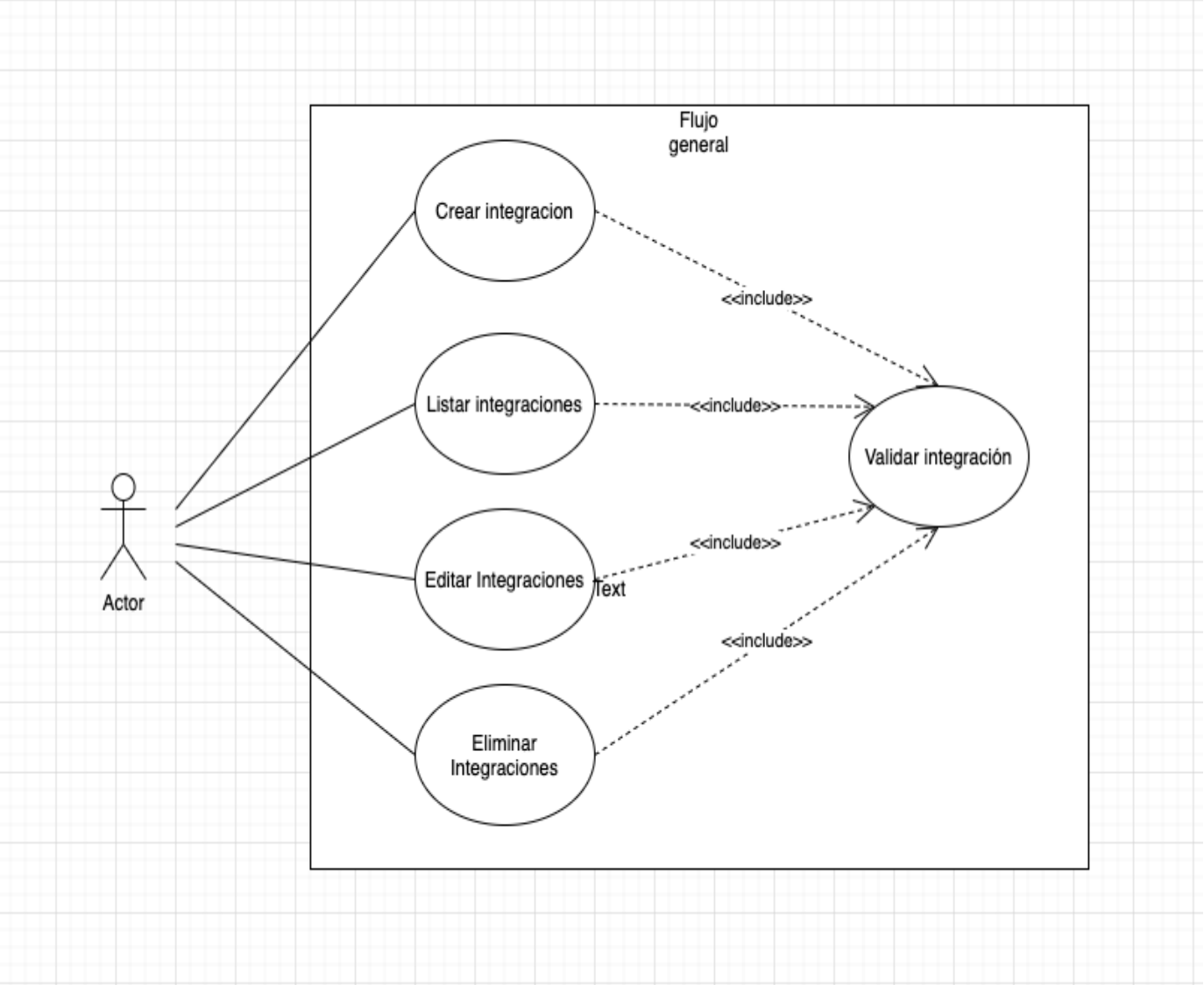

Ilustración 13 flujo general

4.3 Diagrama de clases

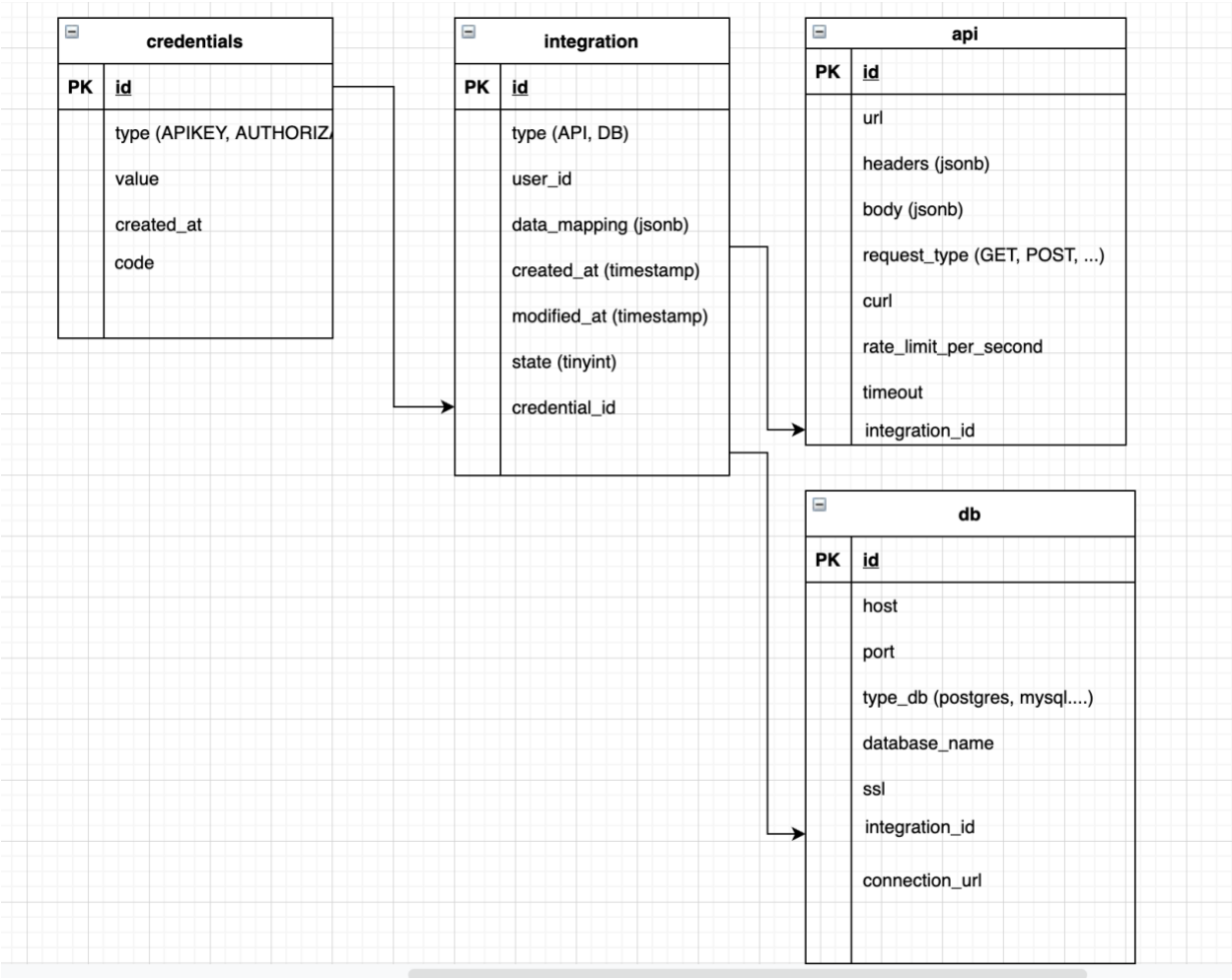

Ilustración 14 diagrama de clases

4.4 Diagramas de casos de uso

4.4.1 Crear integración tipo Database

Nombre del caso de uso	Crear Integración tipo Database
Actor participante	Indiciado por el usuario
Condición inicial	1. El usuario presiona el botón create new integration
Flujo de eventos	1. El usuario presiona el botón create new integration y se desplegará un listado con el tipo de integración que quiere el usuario. 2. El usuario selecciona el tipo de integración Database 3. Se despliega un formulario con todos los campos que se requieren 4. Se llena el campo Name, Host, Port, Database Name, User y Password 5. El usuario presiona el botón create 6. Se habilita el botón test connection 7. Se verifica si la conexión fue exitosa o

	no
Condición de salida	1. Saldrá un popup de succes o failed si la conexión fue exitosa o no
Requerimientos especiales	1. Todos los campos deben ser registrados 2. Los datos de conexión deben ser correctos para una correcta integración

Tabla 4 caso de uso crear integración tipo database

4.4.2 Crear integración tipo Api

Nombre del caso de uso	Crear Integración tipo Api
Actor participante	Indiciado por el usuario
Condición inicial	1. El usuario presiona el botón create new integration
Flujo de eventos	1. El usuario presiona el botón create new integration y se desplegará un listado con el tipo de integración que quiere el usuario. 2. El usuario selecciona el tipo de

	integración API 3. Se despliega un formulario con todos los campos que se requieren 4. Se llena el campo Api Name,Api Url,el request type si es GET,POST,DELETE o UPDATE, los headers y body son opcionales 5. El usuario presiona el botón create
Condición de salida	1. Saldrá un popup de succes
Requerimientos especiales	1. Todos los campos deben ser llenados 2. Los datos de conexión deben ser correctos para una correcta integración

Tabla 5 caso de uso crear integración tipo api

4.4.3 Listar Integraciones

Nombre del caso de uso	Listar Integraciones
Actor participante	Indiciado por el usuario
Condición inicial	1. El usuario presiona el botón data sources en el navbar

Flujo de eventos	1. El usuario presiona el botón datasource en navbar y lo redirige al listado de las integraciones
Condición de salida	1. Listado con la información de las integraciones.
Requerimientos especiales	1. si no existen integraciones saldrá un mensaje el cual indica que no hay integraciones

Tabla 6 caso de uso listar integraciones

4.4.4 Eliminar Integraciones

Nombre del caso de uso	Listar Integraciones
Actor participante	Indiciado por el usuario
Condición inicial	1. El usuario presiona el botón menú en una integración específica
Flujo de eventos	1. El usuario presiona el botón menú en una integración específica el cual despliega un menú con la opción de editar y eliminar

	2. El usuario selecciona la opción de eliminar 3. se despliega un popup con la opción de confirmar la eliminación 4. el usuario confirma la eliminación 5. Se refresca el listado de integraciones
Condición de salida	1. El listado con la información de las integraciones se refresca
Requerimientos especiales	1. Si el usuario no confirma el popup con la opción de eliminar, la integración no se eliminará.

Tabla 7 caso de uso eliminar integración

4.5 Diagramas de secuencias

4.5.1 Crear integración

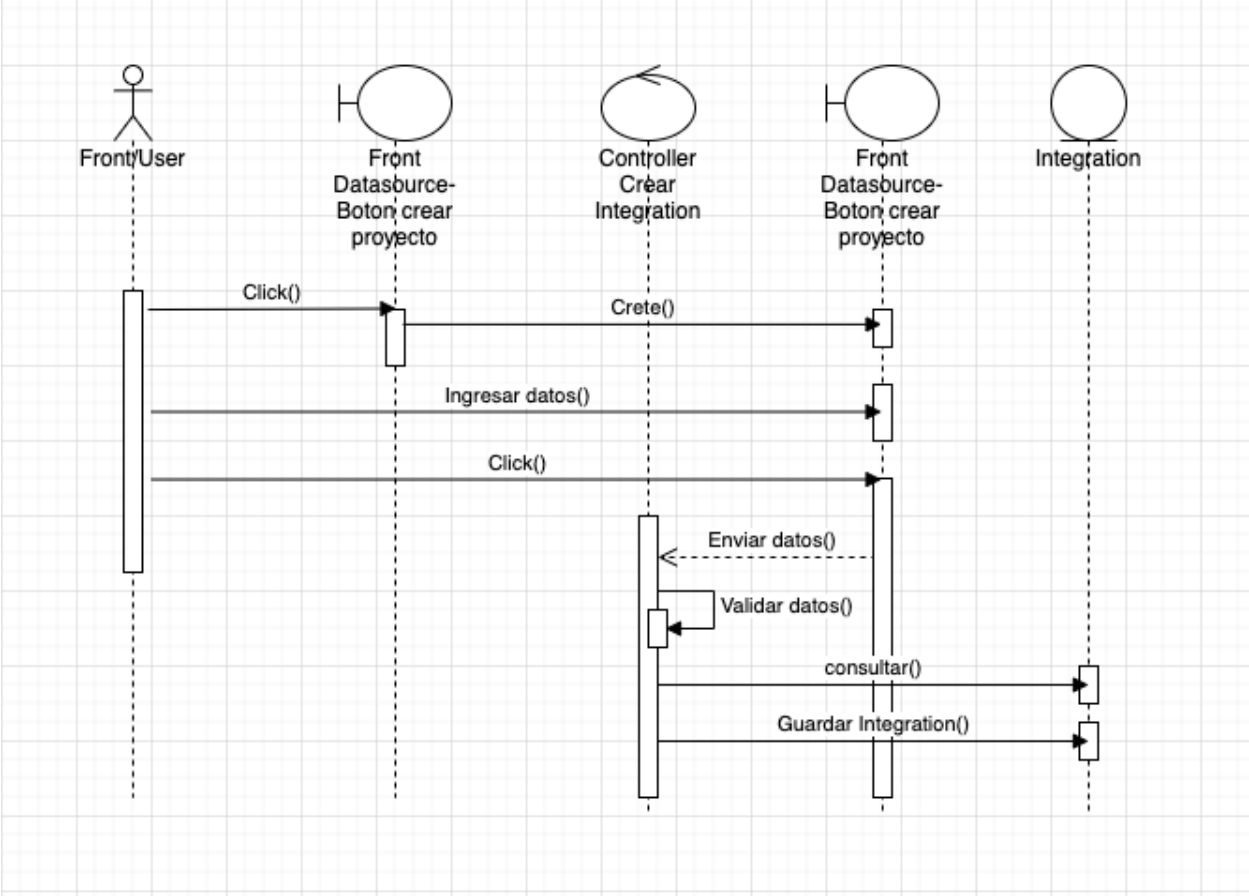

Ilustración 15 diagrama de secuencia crear integración

4.5.2 Listar Integraciones

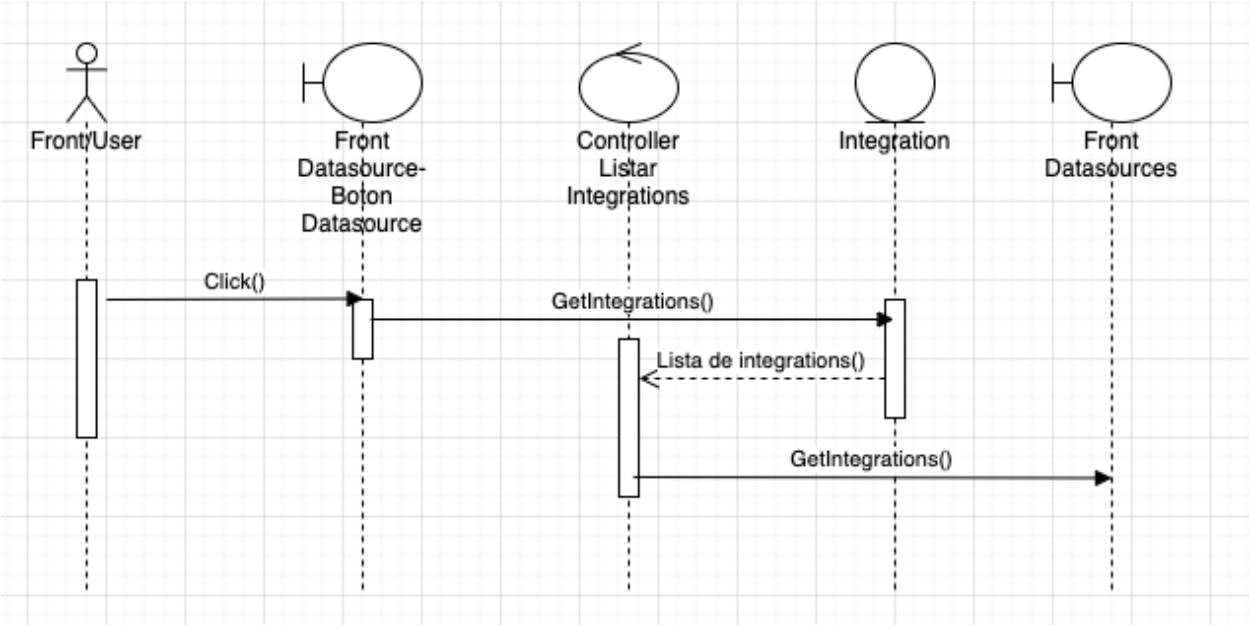

Ilustración 16 diagrama de secuencia listar integraciones

4.5.3 Eliminar Integración

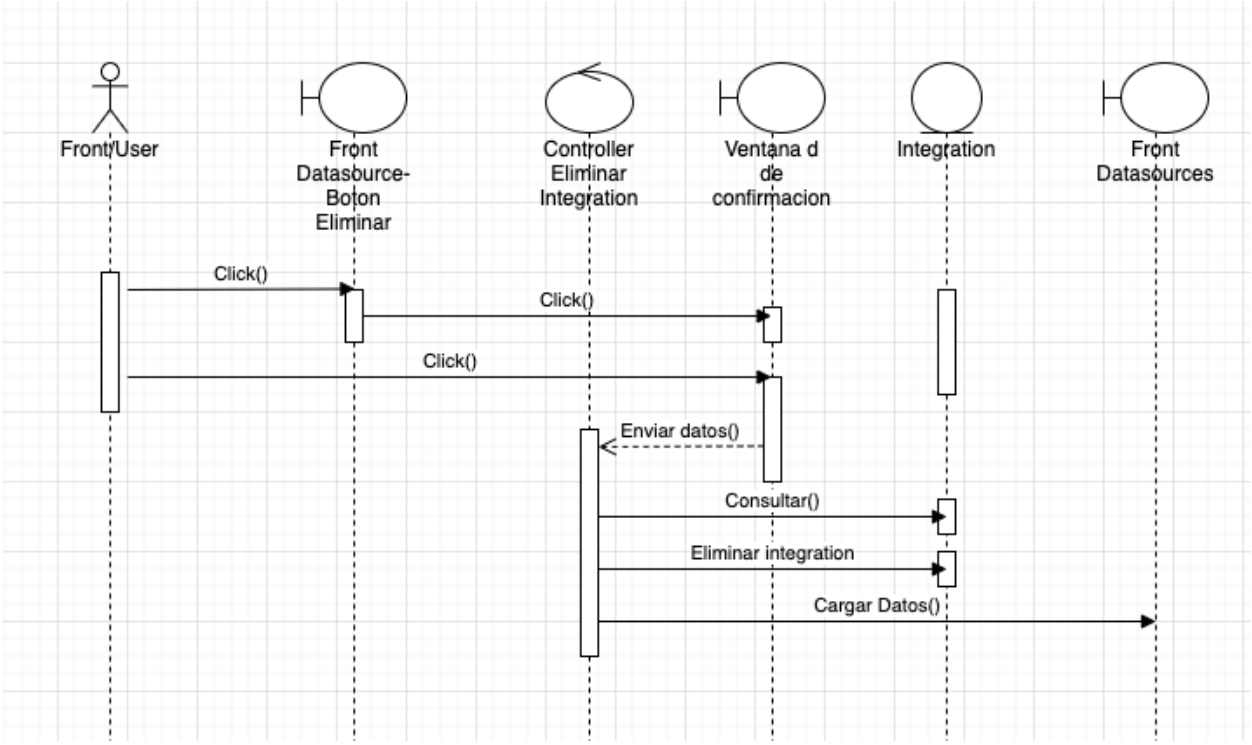

Ilustración 17 diagrama de secuencia eliminar integración

4.6 Funcionalidades básicas

4.6.1 Crear Integración tipo DATABASE

Caso de prueba	Crear integración tipo Database
Descripción	Este flujo es el encargado de crear una integración y guardarlo de manera persistente en la base de datos, luego el usuario es enviado a la pantalla “Data sources”, donde podrá ver todas las integraciones disponibles al igual que eliminarlas.

	Contiene un panel con los tipos de integración a escoger y un formulario donde el usuario debe llenar los campos y contiene dos botones uno para “continuar” con la integración y el otro para cancelar”.
Funcionalidad/característica	Crear integración tipo Database
Datos y acciones de entrada	El usuario selecciona el tipo de integración en este caso de tipo database , y debe registrar los campos de nombre, host, puerto, el nombre de la base de datos, username y password y por último el botón de crear.
Resultado esperado (Datos de salida)	Después de clickear el botón de crear lo llevará al panel de test de conexión para validar si la conexión fue exitosa y por último es redirigido a la vista de “Data sources”
Requerimientos de ambiente de pruebas	Ninguna especificación requerida
Procedimientos especiales requeridos	● El usuario debe registrar todos los campos del formulario y de manera correcta para que exista una conexión exitosa. ● Si el usuario presiona cancelar no se

	realiza ninguna acción, lo dirige de nuevo al panel para escoger el tipo de integración a realizar.

Tabla 8 funcionalidad básica crear integracion tipo database

4.6.1.2 Seguimiento

Resultado obtenido	Al realizar el llenado correcto del formulario y si los datos de conexión están bien , se realizará el test de conexión y posteriormente se refrescará la lista de Datasources
Estado	Aceptado
Última fecha de estado	08/09/2021
Observaciones	En la ejecución del servicio crear integración del microservicio no hubo ningún problema, se obtuvieron los resultados esperados.

Tabla 9 seguimiento funcionalidad básica crear integración tipo database

3.6.1.3 Imagen

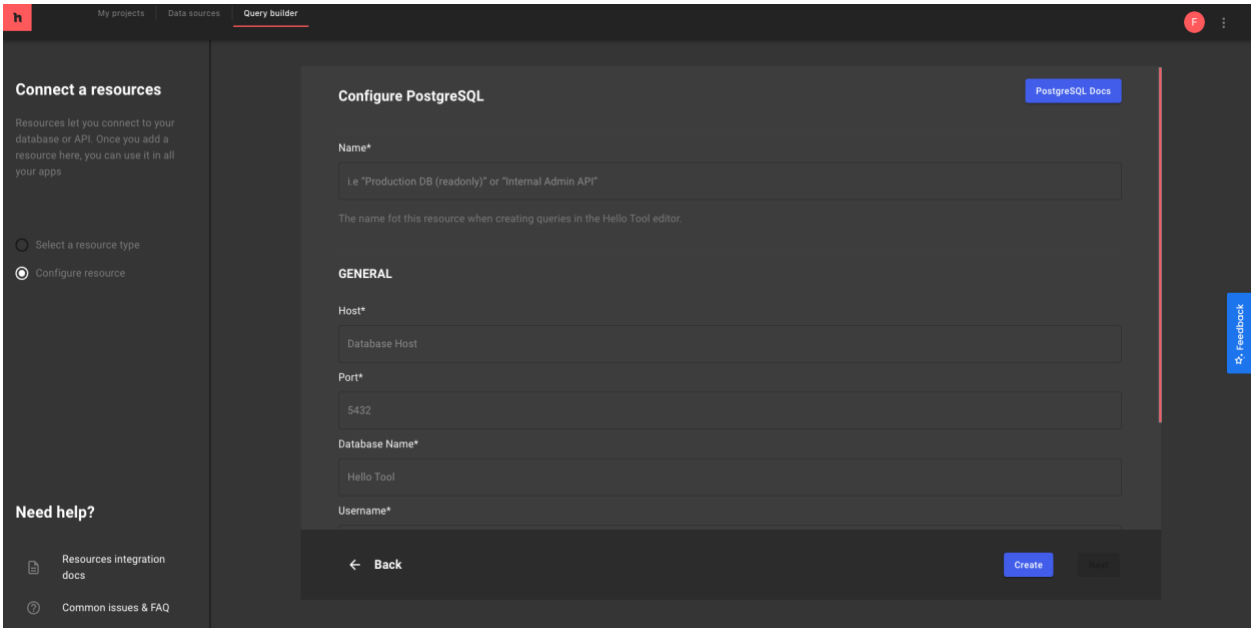

Ilustración 18 crear integración tipo database

4.6.2 Crear integración tipo API

Caso de prueba	Crear integración tipo Database
Descripción	Este flujo es el encargado de crear una integración tipo API y guardarla de manera persistente en la base de datos, luego el usuario es enviado a la pantalla “Data sources”, donde puede ver todas las integraciones disponibles al igual que eliminarlas. Contiene un panel con los tipos de

	integración a escoger y un formulario donde el usuario debe llenar los campos y contiene dos botones uno para “continuar” con la integración y el otro para cancelar”.
Funcionalidad/característica	Crear integración tipo Database
Datos y acciones de entrada	El usuario selecciona el tipo de integración en este caso de tipo API , y selecciona la credencial, api url, api name, seleccionar el tipo de request, headers opcional, body opcional, y por último el botón de crear.
Resultado esperado (Datos de salida)	Después de clickear el botón de crear se envía el request y por último es redirigido a la vista de “Data sources”
Requerimientos de ambiente de pruebas	Ninguna especificación requerida
Procedimientos especiales requeridos	● El usuario debe llenar todos los campos del formulario y de manera correcta para que la petición se realice de manera correcta. ● Si el usuario presiona cancelar no se

	realizará ninguna acción, lo llevará de nuevo al panel para escoger el tipo de integración a realizar.

Tabla 10 funcionalidad básica crear integración tipo api

4.6.2.1 Seguimiento

Resultado obtenido	Al realizar el llenado correcto del formulario y si los datos están correctos la petición se realizará de manera correcta.
Estado	Aceptado
Última fecha de estado	08/09/2021
Observaciones	En la ejecución del servicio crear integración del microservicio no hubo ningún problema, se obtuvieron los resultados de acuerdo a los requerimientos.

Tabla 11 seguimiento funcionalidad básica crear integración tipo api

4.6.2.2 Imágenes

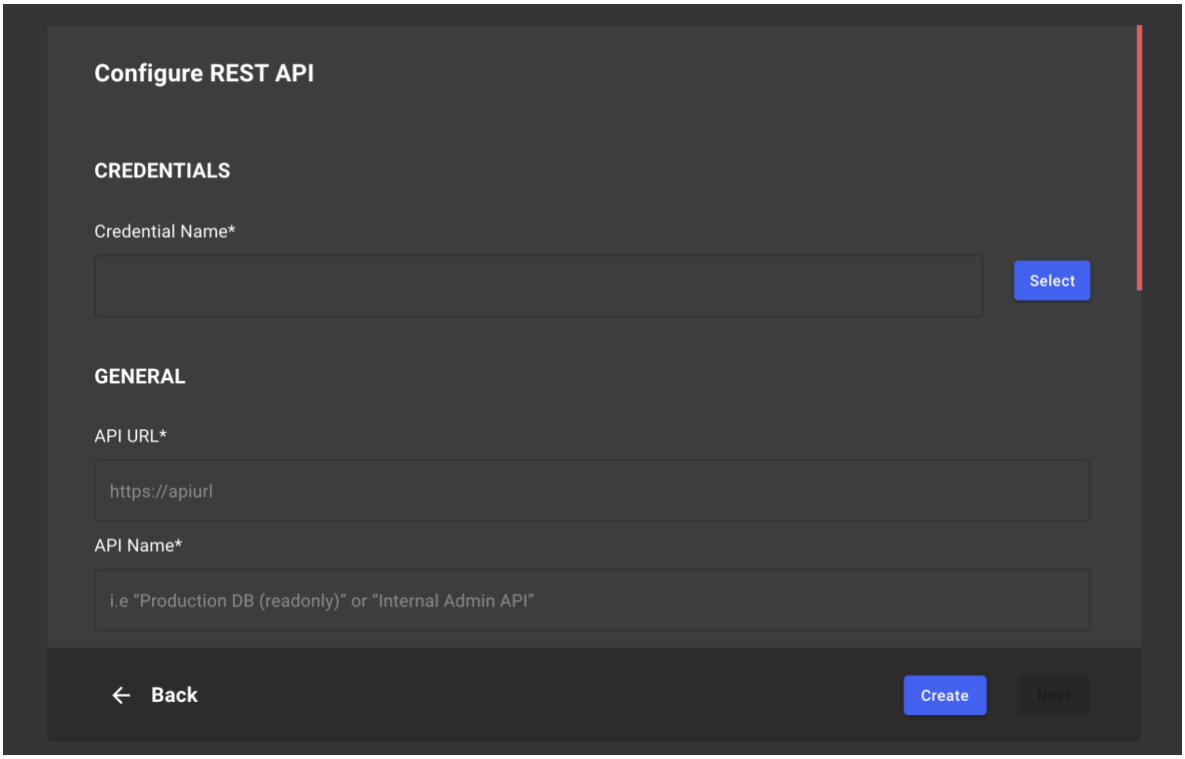

Ilustración 19 crear integración tipo api parte 1

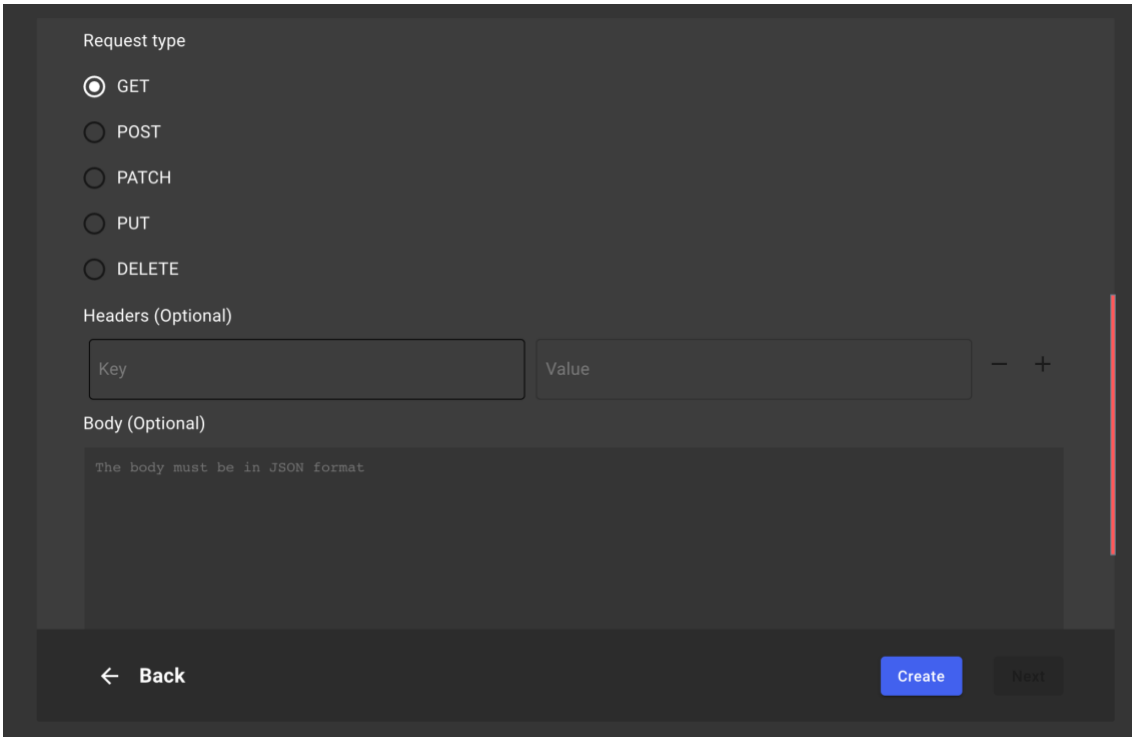

Ilustración 20 crear integración tipo api parte 2

4.6.3 Listar integraciones

Caso de prueba	Listar integraciones
Descripción	Este flujo es el encargado de mostrarle al usuario todas las integraciones creadas en la vista “Data sources”
Funcionalidad/característica	Listar integraciones
Datos y acciones de entrada	El usuario clickea en el navbar el

	botón de “Data sources” y se ocurre una acción un llamado a bases de datos el cual trae todas las integraciones.
Resultado esperado (Datos de salida)	Una lista con todas las integraciones del usuario
Requerimientos de ambiente de pruebas	Ninguna especificación requerida
Procedimientos especiales requeridos	• Si el usuario no tiene creada ningún tipo de integración responderá con una lista vacía.

Tabla 12 funcionalidad básica listar integraciones

4.6.3.2 Seguimiento

Resultado obtenido	Al realizar el proceso correcto , se cargará la lista de integraciones en la vista “Data Sources”
Estado	Aceptado

Última fecha de estado	08/09/2021
Observaciones	En la ejecución del servicio listar integraciones del microservicio no hubo ningún problema, se obtuvieron los resultados de acuerdo a los requerimientos.

Tabla 13 seguimiento funcionalidad básica listar integraciones

4.6.3.3 Imagen

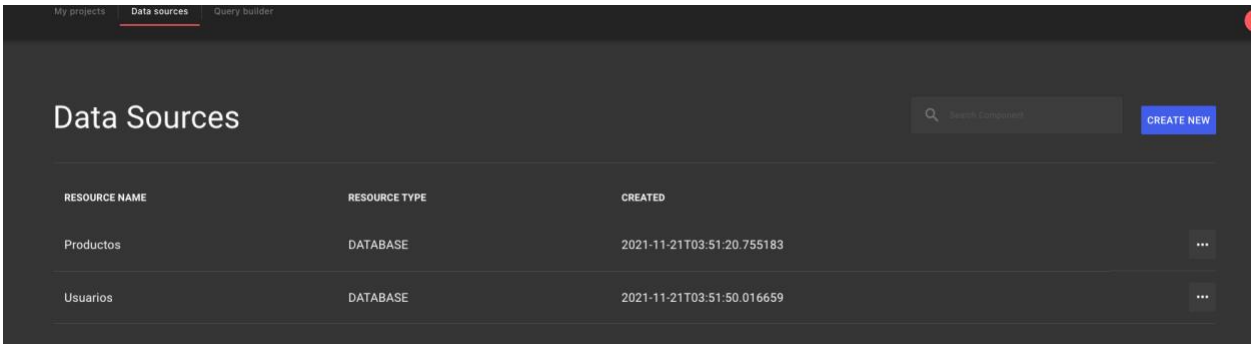

Ilustración 21 listar integraciones

4.6.4 Eliminar Integración

Caso de prueba	Eliminar integración
Descripción	Este flujo de eliminar una integración en específica que corresponde al usuario, el usuario solo debe seleccionar la opción del menu por cada registro en la vista “Data sources”
Funcionalidad/característica	Eliminar integración
Datos y acciones de entrada	El usuario clickear la opción eliminar del menú solo debe confirmar la eliminación del registro , si confirma este registro se eliminará de la base de datos.
Resultado esperado (Datos de salida)	Una lista con las integraciones actualizadas donde ya no estará la integración eliminada previamente.
Requerimientos de ambiente de pruebas	Ninguna especificación requerida

Procedimientos especiales requeridos	● Si el usuario clickea el botón cancelar no se ejecuta ninguna acción interna y el registro permanecerá en la base de datos

Tabla 14 funcionalidad básica eliminar integración

4.6.4.1 Seguimiento

Resultado obtenido	Al usuario presionar el botón eliminar en el panel y el usuario confirme la eliminación se refrescará la lista de integraciones en la vista de “Data sources”
Estado	Aceptado
Última fecha de estado	08/09/2021
Observaciones	En la ejecución del servicio eliminar integración del microservicio no hubo ningún problema, se obtuvieron los resultados de acuerdo a los requerimientos.

Tabla 15 seguimiento funcionalidad básica eliminar integración

4.6.4.2 Imagen

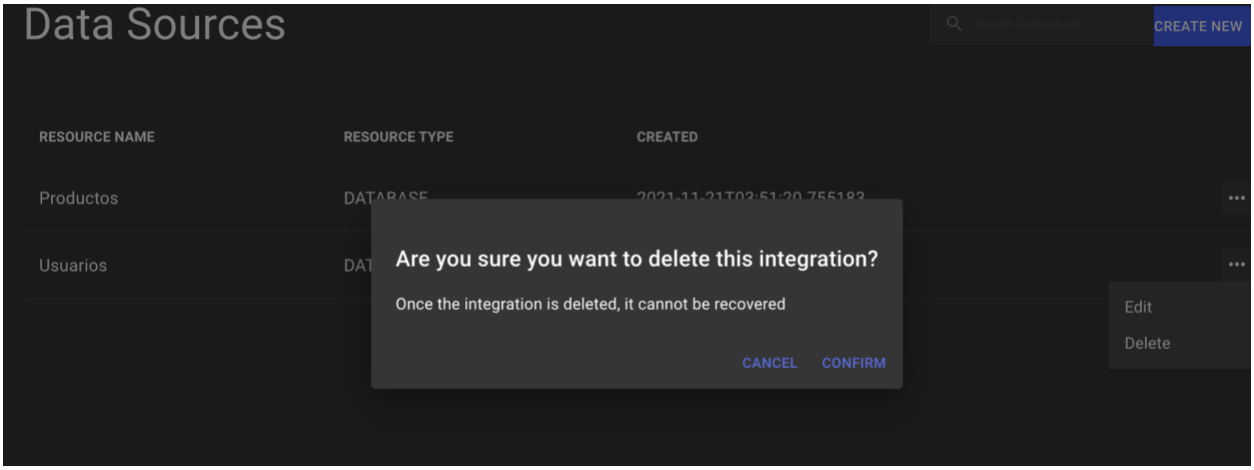

Ilustración 22 eliminar integración

CAPÍTULO 5

5 Arquitectura del microservicio Integration

En este capítulo se describe la arquitectura del microservicio de integración de la herramienta Hello Tool que se usó.

5.1 Arquitectura REST API

Según los requerimientos establecidos para el microservicio integration , se concluyó que la mejor arquitectura para dicho microservicio es la de REST API ya que dicha arquitectura se apoya totalmente en el estándar HTTP y esto permitio desarrollar una API desde el punto de vista del backend, esto permitió que desde la parte de frontend se pudo usar los endpoints desarrollados, un ejemplo de llamado a un endpoint sería: **host/microservicioIntegration/integration/**,este ejemplo sería para un endpoint tipo GET , y así respectivamente con los diferentes endpoints.

5.2 Repository

El repository es una clase de tipo “interface” que usa el marcador stereotype de @Repository propia de spring para indicar que es un repositorio o un objeto de acceso de datos, este tipo de anotación tiene la funcionalidad de traducción automática ejemplo: cuando sucede una excepción en el @Repository, ya hay un controlador para esa excepción y no es necesario agregar un bloque try-catch.

Esta clase también usa la etiqueta @Query el cual permite realizar las consultas en este SQL para las diversas consultas que se usaron en el desarrollo.

```
package com.helloguru.lego.repository

import com.helloguru.lego.model.Integration
import org.springframework.data.jpa.repository.JpaRepository
import org.springframework.data.jpa.repository.Query
import org.springframework.stereotype.Repository
import java.util.UUID

@Repository
interface IntegrationRepository : JpaRepository<Integration, Long> {
    @Query("SELECT i FROM Integration AS i WHERE i.code = :code AND i.userId = :userId")
    fun getByCode(code: UUID, userId: UUID): Integration?

    @Query("SELECT i FROM Integration  AS i WHERE  i.userId = :userId")
    fun getByUserId(userId: UUID): List<Integration>

    @Query("SELECT i FROM Integration  AS i WHERE  i.type = :type AND i.userId = :userId")
    fun getByType(type: String, userId: UUID): List<Integration>
}
```

Ilustración 23 repository

5.3 Model

En esta clase se definió todos los campos que lleva los diferentes modelos del microservicio Integration, en las clases se usaron diferente etiquetas como @Table con esta etiqueta se le dio nombre a la tabla, @Entity con esta etiqueta se definió la entidad, la etiqueta @Id que se usó para definir su primary key, la etiqueta @Column que se usó para definir las diferentes columnas de la tabla integration y las etiquetas de relación @OneToOne, todas estas etiquetas lo que permitieron fue que cuando el proyecto se compilo y ejecuto las tablas y sus campos se crearon de forma automática en la base de datos que en este caso se encuentra alojada en postgresql.

Aparte de esta clase funcionar para crear las tablas en bases de datos el otro objetivo importante es que permitió instanciar las diferentes clases como objetos para toda la manipulación de información. Este tipo de modelo aplico para las demás tablas, Database, Api y Webhook.

```
@Table(name = "integration")
@Entity
@TypeDef(name = "json", typeClass = JsonType::class)
class Integration {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    var id: Long = 0

    @Column(name = "code", unique = true)
    var code: UUID = UUID.randomUUID()

    @Column(name = "name")
    var name: String = ""

    @Column(name = "type")
    var type: String = ""

    @Type(type = "json")
    @Column(name = "schema", columnDefinition = "json")
    var schema: Map<String, Any>? = null

    @Column(name = "user_id")
    var userId: UUID? = null

    @Column(name = "project_id")
    var projectId: UUID? = null
```

Ilustración 24 modelo

5.4 Service

En esta clase es donde se definieron las diferentes funcionalidades del microservicio como por ejemplo: Crear integración, listar las integraciones, eliminar las integraciones y otras funcionalidades adicionales del microservicio.

Se usaron las etiquetas de @Service para indicar que esta clase es un servicio, la etiqueta de @Autowired la cual permite llamar otros servicios dentro del mismo microservicio o llamar a los repositorios donde se encuentran los diferentes llamados a bases de datos.

Este es un ejemplo de una función para listar todas las integraciones.

```
fun findAll(userId: UUID): List<IntegrationResponse> {
    val listIntegration = integrationRepository.getByUserId(userId)
    val listIntegrationResponse = mutableListOf<IntegrationResponse>()
    var nameSource: String? = null
    listIntegration.forEach { integration ->
        when (integration.type) {
            IntegrationType.DATABASE.name -> {
                val datasource = databaseRepository.getByIntegrationId(integration.id)
                nameSource = datasource.name
            }
            IntegrationType.ETL.name -> {
                val datasource = databaseRepository.getByIntegrationId(integration.id)
                nameSource = datasource.name
            }
        }
        listIntegrationResponse.add(IntegrationMapper.getIntegrationResponse(integration, urlWebhook: null, nameSource))
    }
    return listIntegrationResponse
}
```

Ilustración 25 service

5.5 Controller

En esta clase lo que se hace principalmente es llamar a los servicios que fueron desarrollados y que puedan ser ejecutados a través de endpoints de tipo GET, POST , DELETE, UPDATE O PATCH, generando una url por cada endpoint en el controlador, que posteriormente cada uno de estos endpoints se consumieron desde el Frontend para cada respectiva funcionalidad del microservicio. En esta clase se usan las etiquetas @RestController para indicar que la clase es de tipo controlador y que en ella hay diferentes endpoints que son consumidos vía HTTP como una REST API.

A continuación un ejemplo del controlador de la clase integration.

```kotlin
@RestController
@RequestMapping(Routes.INTEGRATION_MAIN_ROUTE)
class IntegrationController(private val integrationService: IntegrationService) {

    @Autowired
    lateinit var integrationValidator: IntegrationValidator

    @GetMapping
    @ApiOperation(tags = ["integration-controller"], value = "List all integrations",
        response = IntegrationResponse::class, responseContainer = "List")
    @ApiResponses(
        value = [
            ApiResponse(code = 200, message = "Ok"),
            ApiResponse(code = 400, message = "bad request"),
            ApiResponse(code = 404, message = "Not found")
        ]
    )
    fun list(
        @RequestHeader(value = "x-user-id", required = false) encoding: String?,
        @RequestHeader(value = "x-user-type", required = false) type: String?,
        request: HttpServletRequest
    ): List<IntegrationResponse> {
        val userId = request.getHeader(Headers.xUserId)
        request.getHeader(Headers.xUserType)

        return integrationService.findAll(UUID.fromString(userId))
    }
```

Ilustración 26 controller

5.6 Descripción de la arquitectura

Se puede observar que la arquitectura usada en este microservicio desarrollado en Kotlin y apoyado de Spring maneja una arquitectura REST API la cual consiste en un repositorio que es el encargado de toda la interacción en cuanto a consultas con la base de datos, se compone de un modelo el cual sirve como clase para poder manipular todos los objetos, también está constituido de una clase servicio que es la encargada de manipular todos los objetos estableciendo una conexión al repositorio o a los diferentes servicios que existen , por último esta el controlador que es el encargado de ejecutar todos estos servicios, estos servicios se ejecutaron previamente definiendo unos endpoints ya sean de tipo GET, POST , DELETE, UPDATE O PATCH y al final todo este conjunto de endpoints forman una API.

CAPÍTULO 6

6 Implementaciones y validaciones

En este Capítulo esta presente toda la implementación del microservicio de integración y sus diferentes funcionalidades como lo son crear integración de tipo Api, integración de tipo Api, listar todas las integraciones, poder eliminar cada integración tanto del punto de vista del backend como ya desplegadas y funcionando con el frontend, también están los respectivos tests para cada funcionalidad en especifica.

6.1 Crear integración

La funcionalidad crear integración es de tipo Post recibe un body el cual esta conpuesto de un campo nombre el cual contiene el nombre de la integración, el tipo el cual guarda el tipo de integración que se quiere realizar, dependiendo del tipo recibe otro request el cual contiene el nombre, url, tipo, timeout y las request por segundo, al pegarle al endpoint de crear integración se evidencio que respondio un status 200 se ejecuto en 1413 ms y un response con toda la información de la integración.

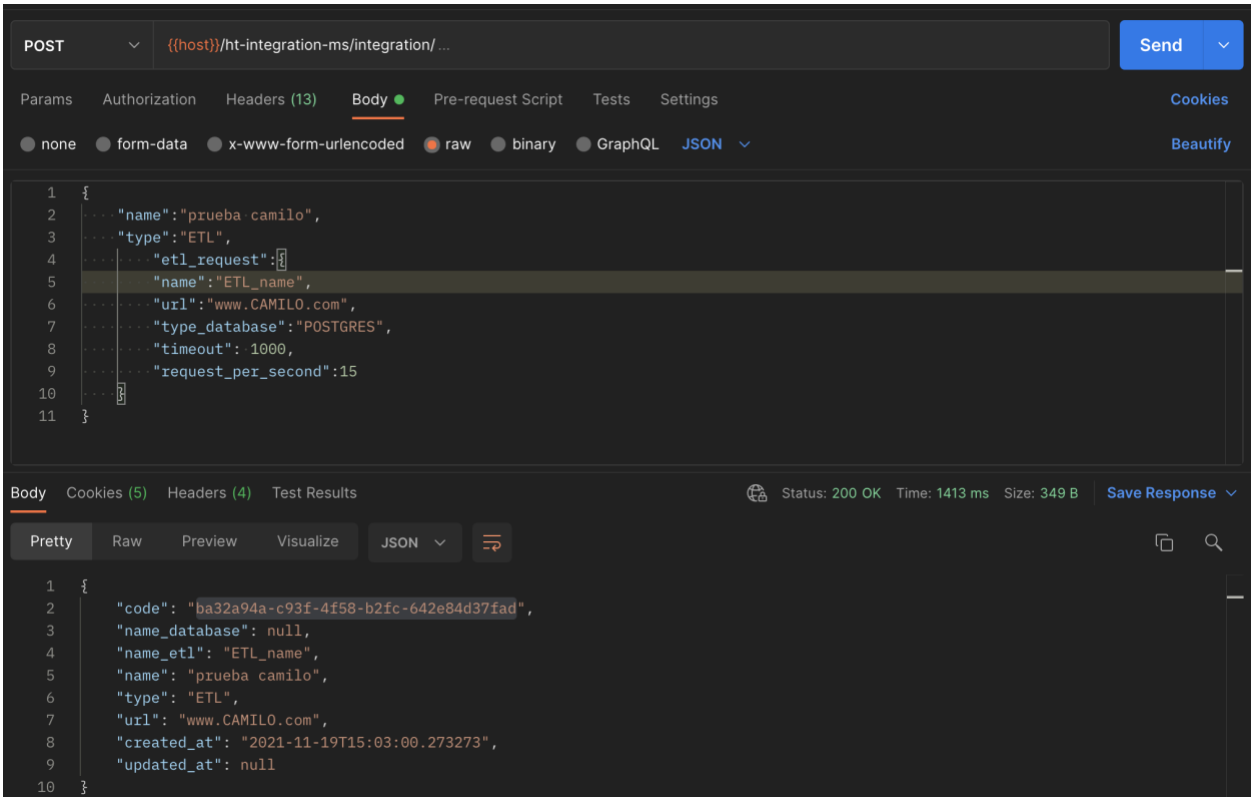

Ilustración 27 crear integración desde postman

6.2 Listar integraciones

Esta funcionalidad muestra todas las integraciones creadas se evidencio al momento de pegarle al endpoint este respondió un status 200 y se ejecuto en 698 ms con un response de una lista la cual contiene todas las integraciones y su información.

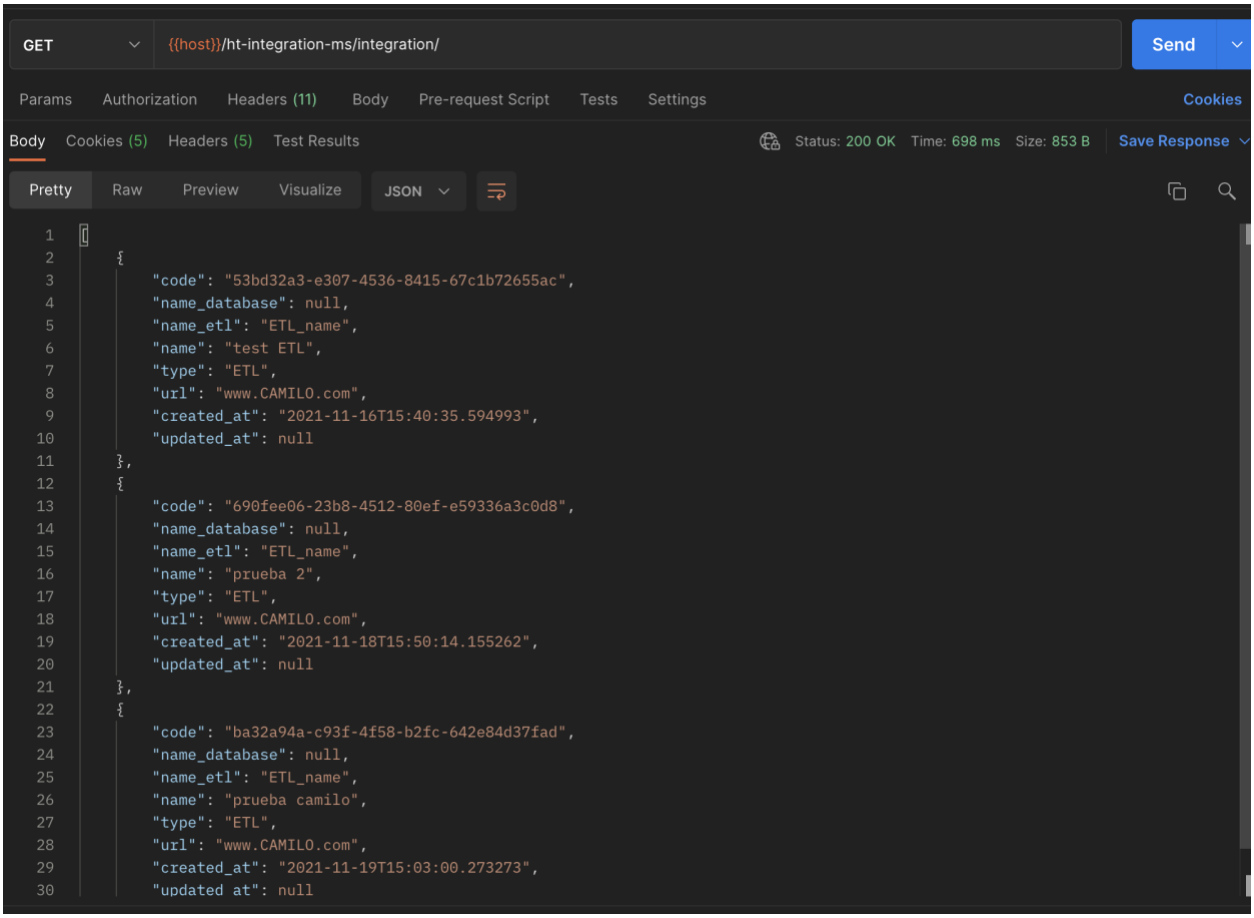

Ilustración 28 listar integraciones desde postman

6.3 Eliminar integraciones

Esta funcionalidad recibe como parámetro el código de la integración la cual se quiere eliminar al consumir este endpoint se evidencio una respuesta de un status 200 , se ejecuto en 77 ms y esta funcionalidad no tiene un response

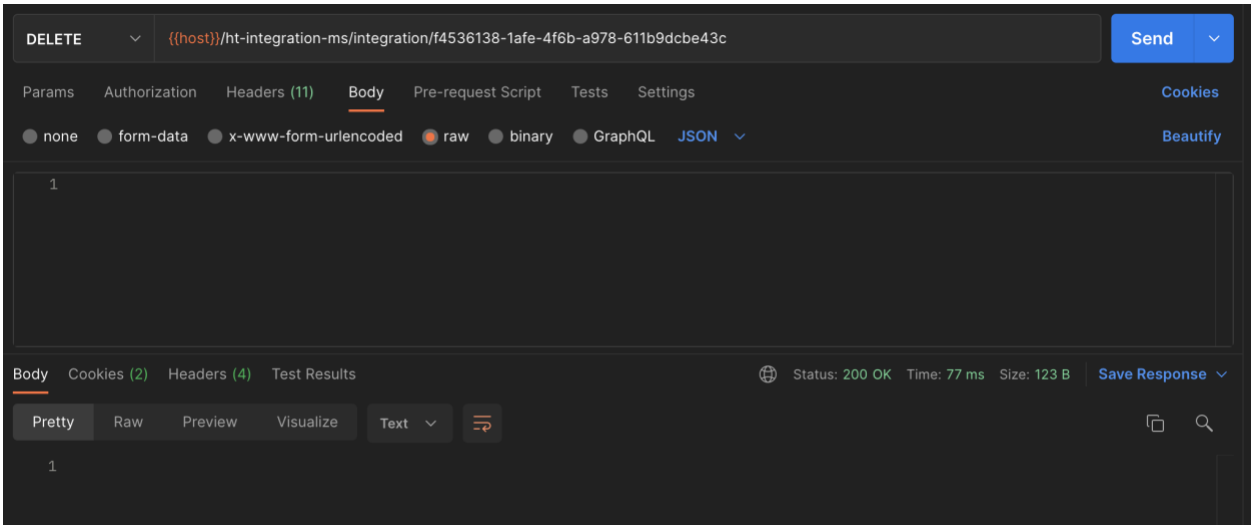

Ilustración 29 eliminar integración desde Postman

6.4 Test de crear integración

Este test corresponde a la funcionalidad de crear integración, se implemento y se evidencio que el test paso con éxito y se ejecuto en 3 segundos y 950 ms

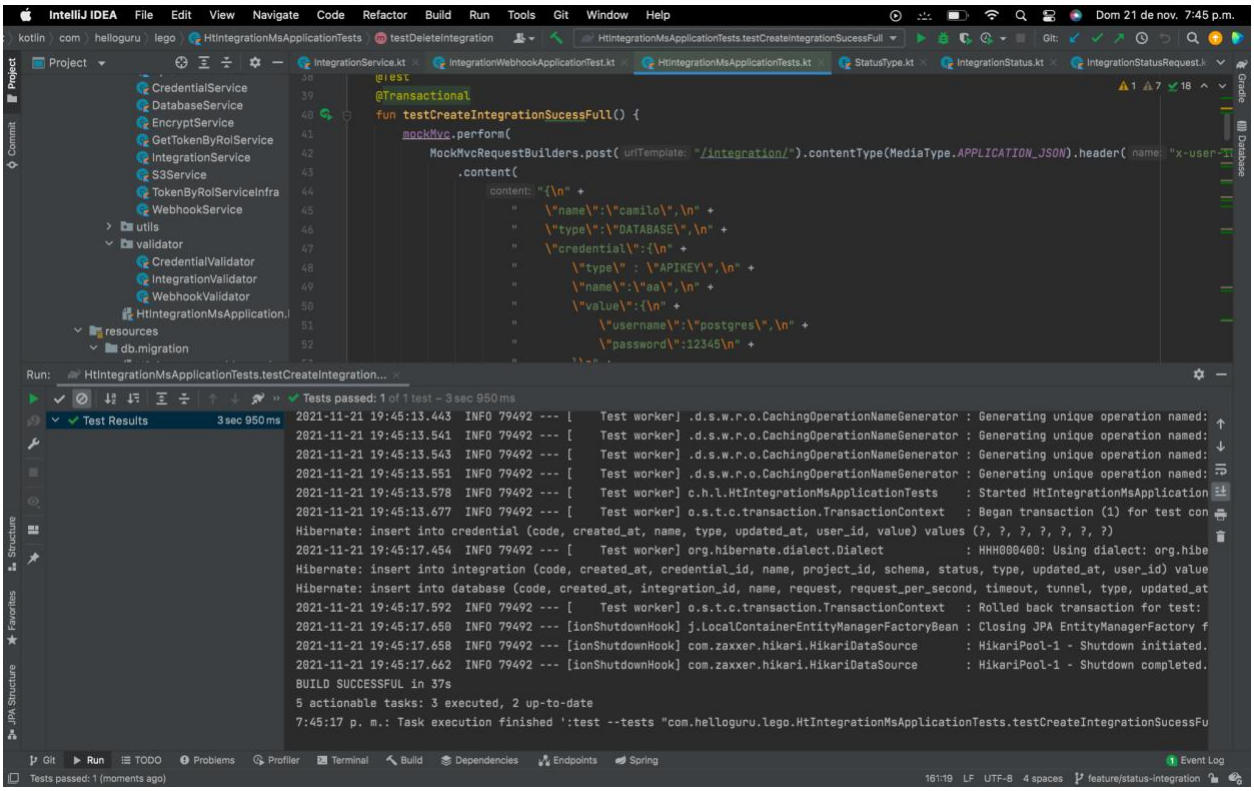

Ilustración 30 test crear integración

6.5 Test de listar integración

Este test corresponde a la funcionalidad de listar integraciones, se implemento y se evidencio que el test paso con éxito y se ejecuto en 942 ms

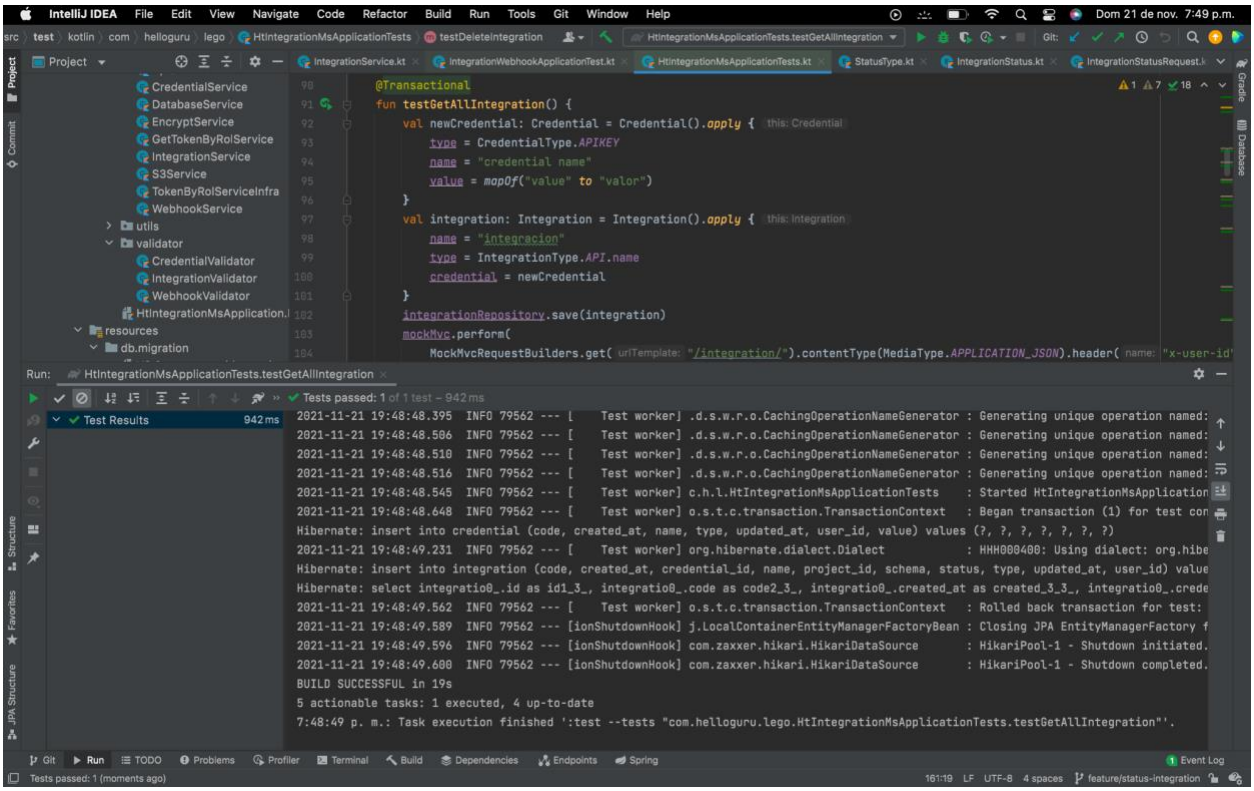

Ilustración 31 test listar integración

6.6 Test de eliminar integración

Este test corresponde a la funcionalidad de eliminar integración, se implemento y se evidencio que el test paso con éxito y se ejecuto en 1 segundo y 464 ms

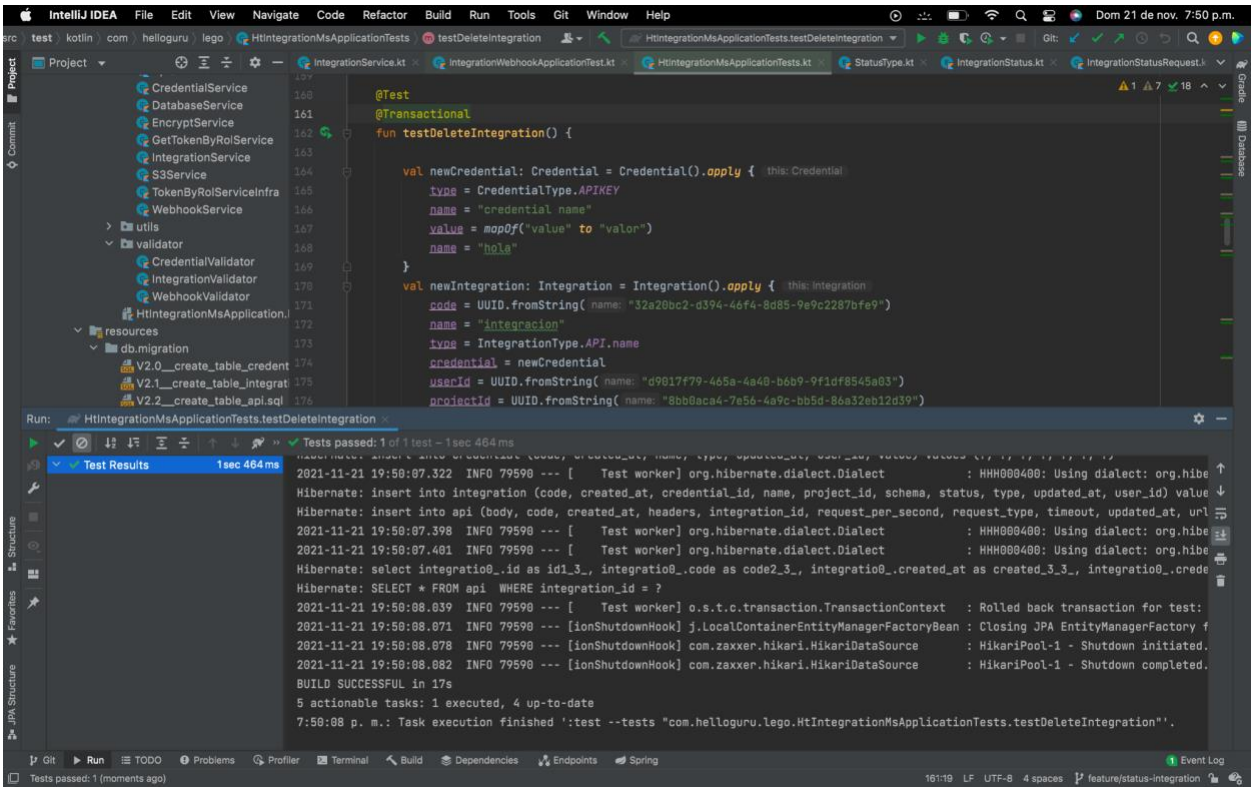

Ilustración 32 test eliminar integración

CAPÍTULO 7

7. Conclusiones

De acuerdo con los resultados se puede observar que la arquitectura de microservicios en este caso REST API es muy recomendable para proyectos de mediano y grande impacto por la facilidad de poder escalar estos mismos.

Manejar este tipo de arquitecturas garantiza un desarrollo ágil y de entregas continuas ya que este tipo de arquitectura REST API permite segmentar todo en pequeños servicios o módulos de diversas funcionalidades

Realizar cambios en las funcionalidades o agregar nuevos features al aplicativo se realiza de una manera muy fácil ya que la escalabilidad que otorga la arquitectura de microservicios es muy grande.

Implementar una arquitectura de microservicios es mucho más costosa de desarrollar e implementar en comparación con otro tipo de arquitecturas , ya que se requieren un cierto número de desarrolladores.

La arquitectura monolítica es recomendable para proyectos de bajo impacto ya que la implementación de este tipo de arquitectura es mucho más económica en cuanto a presupuestos de desarrollo.

Bibliografía

Goette, E. (22 de julio de 2017). *El patrón de arquitectura microservicios.* Obtenido de El patrón de arquitectura microservicios: https://4.bp.blogspot.com/-kfcr4PdRB2M/WW1eFY3_xdI/AAAAAAAAMSE/sNhKbuTPkZQozuJDlMw1Xa4Q_sa3U8H-gCLcBGAs/s640/sapr_0402-bb1171cdbd95fa7aec2601b6d1c44aa6.png

ICHI.PRO. (07 de 09 de 2021). *Arquitectura de microservicios.* Obtenido de ICHI.PRO: https://ichi.pro/es/arquitectura-de-microservicios-48295257297962

mrirfanto. (19 de 04 de 2018). *Software Architecture(Monolithic and Microservice) - KuGen.* Obtenido de Medium: https://medium.com/kugen/software-architecture-monolithic-and-microservice-a9ed178ed954

Scribbr. (2020 de 06 de 10). *Arquitectura de Monolítico.* Obtenido de scribbr: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico

Ramirez Perez, S. (01 de 06 de 2020). *Estudio del framework spring, spring boot y microservicios.* Obtenido de ebuah: https://ebuah.uah.es/xmlui/bitstream/handle/10017/45107/TFM_Ramirez_Perez_2020.pdf?sequence=1&isAllowed=y

Cupertino, A. (2020 de 01 de 01). *Creando una entidad JPA.* Obtenido de Codemonkey: https://codemonkeyjunior.wordpress.com/2019/07/06/kotlin-creando-una-entidad-jpa/

Guerrero, N. (01 de 08 de 2019). *Spring boot starter.* Obtenido de Programa en linea: https://www.scribbr.es/detector-de-plagio/generador-apa/new/webpage/

Scribbr. (09 de 10 de 2019). *Liberada la nueva versión de PostgreSQL 12 y estas son sus novedades.* Obtenido de Scrubbr: https://ubunlog.com/liberada-la-nueva-version-de-postgresql-12-y-estas-son-sus-novedades/

JET BRAINS. (01 de 06 de 2021). *Jet Brains.* Obtenido de IntelliJ IDEA: https://www.jetbrains.com/es-es/idea/download/#section=mac

Cuervo, V. (14 de 02 de 2019). *Que es Postman.* Obtenido de Arquitectoit: https://www.arquitectoit.com/postman/que-es-postman/

Ludim, L. (15 de 06 de 2018). *Postman.* Obtenido de Medium: https://ludim.medium.com/actualizando-postman-en-ubuntu-b5a58db0579a

Lopez, D., & Maya, E. (17 de 07 de 2017). *Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.* Obtenido de https://documentos.redclara.net/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

Nebel, A. (01 de 10 de 2018). *Arquitectura de Microservicios para Plataformas de Integración.* Obtenido de https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/20586/1/tm-nebel.pdf

Gomez, E. A. (01 de 06 de 2018). *Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura.* Obtenido de http://msde.etsisi.upm.es/wp-content/uploads/2019/07/Arquitecturas-para-Microservicios.pdf

Mayanga, C. (03 de 12` de 2020). *Que es SmartGIT.* Obtenido de Carlos Mayanga: https://www.carlosmayanga.com/2020/02/que-es-smartgit-requisito-de-empleo_19.html

Barrios Contreras, D. A. (04 de 06 de 2017). *Arquitectura de microservicios.* Obtenido de Revistas universidad distrital: https://revistas.udistrital.edu.co/index.php/tia/article/download/9687/pdf/63352

Gomez suarez, K. T., & Cano, A. F. (24 de 07 de 2018). *Un acercamiento a los microservicios.* Obtenido de Repositorio unac: http://repository.unac.edu.co/bitstream/handle/11254/959/Un%20acercamiento%20a%20los%20microservicios.pdf?sequence=1&isAllowed=y