UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

TIA
Tecnología, Investigación y Academia

Arquitectura de microservicios

Microservice architecture

Diego Alfonso Barrios Contreras[1]

Para citar este artículo: González, I. (2018). Market Cart App: aplicación móvil para la gestión de compra de víveres en línea. *TIA*, 6(1), pp. 36-46.

Resumen

En este documento se presentará una introducción a las arquitecturas de microservicios, término que cada día toma más relevancia, liderado por personajes de la talla de Martin Fowler. Se verán características referentes a los microservicios, así como su comparación con arquitecturas monolíticas, con el fin de facilitar la comprensión; por último, se analizará un caso de estudio que tiene como objetivo mejorar la competitividad para tiendas de barrio en Colombia, para ello, se tendrá en cuenta cómo se podría iniciar con la definición de un microservicio para este producto.

Palabras clave: arquitectura, ley de Conwad, microservicio, monolito, NOSQL, RDBM.

Abstract

TThis present document will present an introduction to the microservice architectures, term which takes more relevance every day and is leaded by characters of the size of Martin Fowler. We will see features related to microservices, as well as their comparison with monolithic architectures, in order to facilitate understanding. Finally, we will analyze a case study which aims to improve competitiveness for local stores in Colombia. For that, we will consider how we could start to define a microservice for this product.

Keywords: architecture, Conwad's law, microservice, monolith, NOSQL, RDBM.

Artículo de investigación

Fecha de recepción:
22-11-2015
Fecha de aceptación:
04-06-2017

ISSN: 2344-8288
Vol. 6 No. 1
Enero - junio 2018
Bogotá-Colombia

[1] Ingeniero de Sistemas. Especialista en Ingeniería de Software, Universidad Distrital Francisco José de Caldas. Arquitecto de aplicaciones y soluciones, Aldeamo. Correo electrónico: diegobarrios.c@gmail.com

¿QUÉ ES?

Para dar comienzo a este documento es importante iniciar por conocer el contexto —de una manera muy breve— en el 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 [1].

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 [2].

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 —denotándolo 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. La Figura 1 muestra cuál es la estrategia para escalar una aplicación que está construida bajo este estilo.

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. En la Figura 2 se muestra cómo se ejecutaría el escalamiento en una arquitectura de este tipo.

EJEMPLO

A continuación, se analizará un ejemplo tomado del blog de Ngix [3], donde será posible apreciar cómo luciría una aplicación bajo un estilo monolítico y cómo sería esta misma aplicación bajo un estilo de microservicios. El ejemplo consiste en realizar una aplicación para el manejo de solicitud de taxis,

que compita con aplicaciones como Uber [9] y Hailo [10]. Después de un análisis se plantea una solución modular bajo una arquitectura hexagonal, como se puede apreciar en la Figura 3.

En el núcleo de la aplicación se encuentra la lógica de negocio, la cual esta implementada por diferentes módulos; alrededor del núcleo se encuentran los adaptadores que se encargan de comunicarse con el exterior (por ejemplo componentes de accesos a base de datos, componentes de manejo de mensajes, web, entre otros).

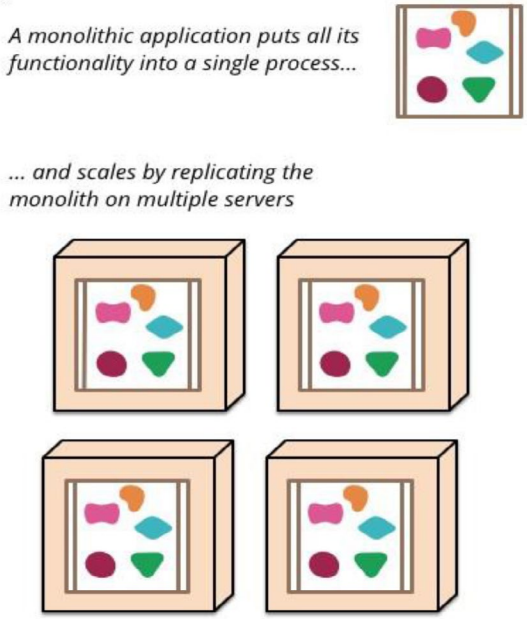

Figura 1. Escalabilidad en una aplicación monolítica

Fuente: [2].

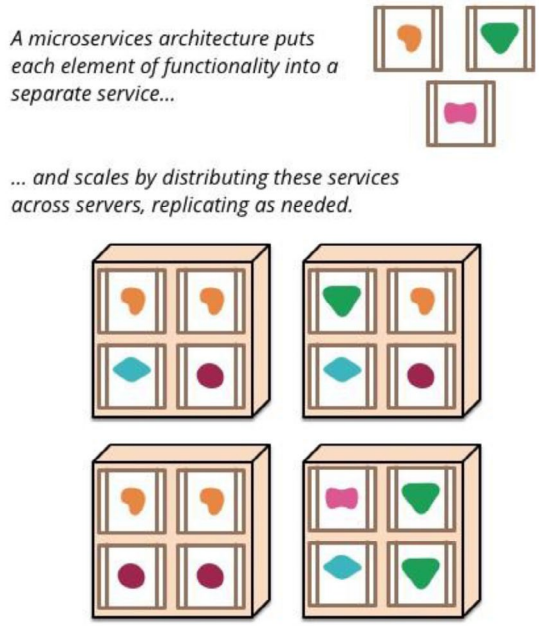

Figura 2. Escalabilidad en una aplicación con microservicios

Fuente: [2].

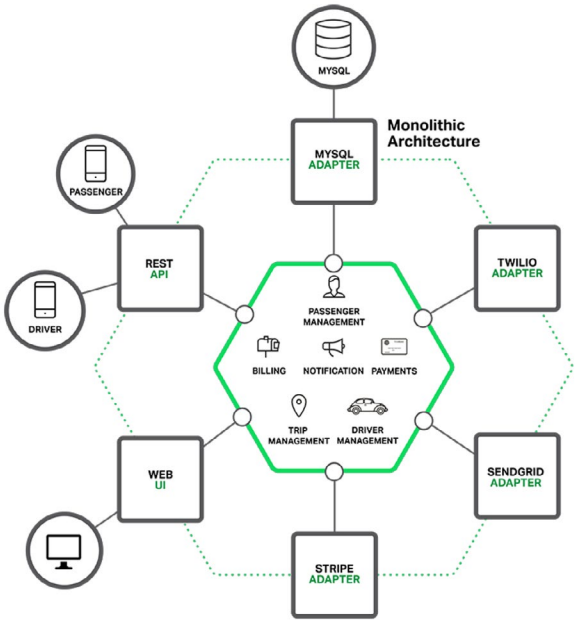

Figura 3. Arquitectura Hexagonal

Fuente: [4].

Aunque la aplicación es modular, es empaquetada y desplegada como un monolito, el formato de empaquetado depende mucho del lenguaje en el que estén construidas, por ejemplo, muchas aplicaciones Java se empaquetan como un archivo WAR.

Este tipo de aplicaciones cuentan con múltiples ventajas, por ejemplo, el hecho de que son simples de desarrollar, pues cuentan con IDE especializados para esto. También son simples para ejecutar pruebas sobre ellas, así como en su despliegue, pues básicamente consiste en un copiado del archivo empaquetado al servidor.

Desafortunadamente, este enfoque simple tiene grandes limitaciones. Aplicaciones exitosas tienden a crecer y con el tiempo se convierten en inmensas aplicaciones, lo que ocasiona que con cada iteración se adicionen más funcionalidades y con el tiempo se conviertan en monolitos monstruosos, no solo en tamaño sino en complejidad. Adicionalmente, afectan el proceso del despliegue, puesto que con aplicaciones más grandes se requiere más tiempo para iniciar y pone un obstáculo adicional en el despliegue continuo. También es muy probable que el equipo no pueda entenderla completamente (salvo algunas pocas personas). Como respuesta a esto, muchas organizaciones, como Amazon, eBay y Netflix, han solucionado este problema adoptando lo que es hoy conocido como arquitectura de microservicios.

Continuando con el ejemplo, cada servicio que se genere tendrá su propia arquitectura hexagonal y se interconectaran con otros, tal como se ve en la Figura 4.

Cada área funcional es implementada ahora por su propio microservicio. Más aun, la web es divida en un conjunto de aplicaciones web más simples, permitiendo obtener las ventajas que se plantea en una arquitectura de este tipo.

El utilizar este enfoque de arquitectura de microservicios tiene un impacto significativo entre la aplicación y la base de datos; en lugar de compartir una base de datos para múltiples servicios, cada servicio tiene su propio esquema de datos, permitiendo escoger el modelo más conveniente dependiendo de la necesidad del servicio. Comúnmente se puede dar duplicidad

de información entre los esquemas, pero tener un esquema por cada servicio es esencial si se quieren los beneficios de los microservicios, dado que asegura el bajo acoplamiento. En la Figura 5 se visualiza la arquitectura de base de datos de la aplicación, en tanto que son independientes, permite que cada base de datos sea diferente y pueda usar enfoques (como NoSQL) y diferentes tecnologías.

CARACTERÍSTICAS

- Pequeño y enfocado en hacer una sola cosa bien [5].

El código fuente crece frecuentemente con cada adición de una nueva funcionalidad, lo que con el tiempo se convierte en un problema pues cada vez es más difícil saber dónde se debe hacer un cambio.

Los microservicios intentan aislar la especialidad de una lógica de negocio específica en un solo lugar, lo que permite disminuir la tentación de hacer crecer mucho el componente.

La pregunta recurrente es ¿qué tan pequeño es pequeño? La cantidad de líneas de código es problemática, pues cada lenguaje se expresa de manera diferente. Jon Evans en RealEstate.com.au en Australia caracteriza los microservicios como algo que pueda ser reescrito en dos semanas.

Un factor adicional a definir para responder qué tan pequeño, es que tan bien se alinea a las estructuras del equipo. Si el código base es muy grande para ser manejado por un equipo pequeño, es un indicador de que puede ser dividido.

- Organizado en torno a la lógica de negocio [2].

Cuando se busca dividir una aplicación en partes a menudo la administración se enfoca en la capa tecnológica, separando los equipos de UI, servidor y base de datos. Cuando los equipos están separados necesitan seguir un protocolo para el manejo de los tiempos y presupuestos de otros equipos, lo que ocasiona que el equipo intente separar su trabajo del de los otros equipos. Lo

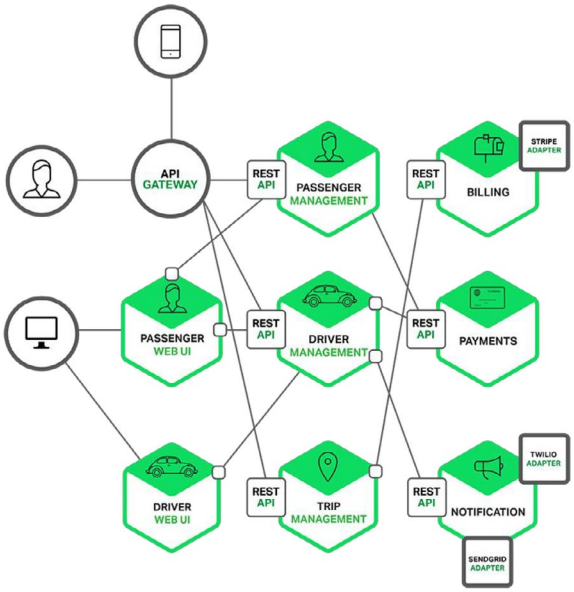

Figura 4. Arquitectura de microservicios

Fuente: [4].

que es un claro ejemplo de la ley de Conway —que dice que cualquier organización que diseña sistemas, producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización—, la cual podemos ver representada en la Figura 6.

El enfoque de microservicios es diferente, pues divide las organizaciones alrededor de la lógica de negocio. Esta organización es interdisciplinar pues mezcla la UI, lógica de negocio y base de datos, lo que conlleva a que el equipo tenga diferentes habilidades, la Figura 7 permite ver cómo se representaría la ley de Conway a este estilo.

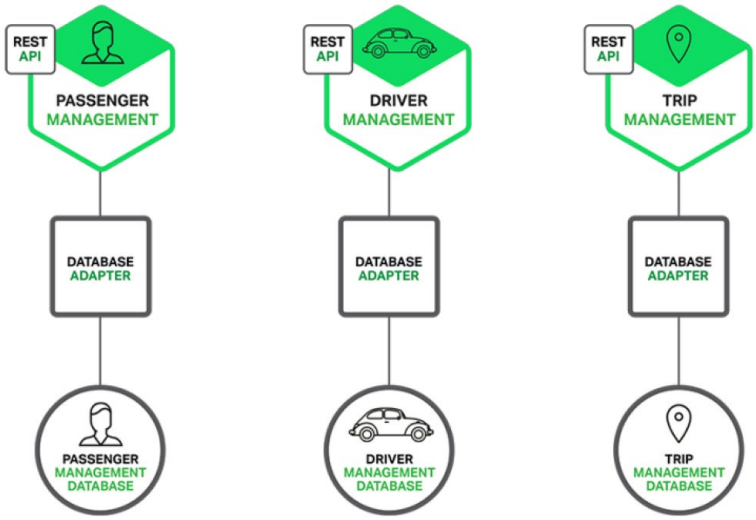

Figura 5. Arquitectura de datos de la aplicación

Fuente: [4].

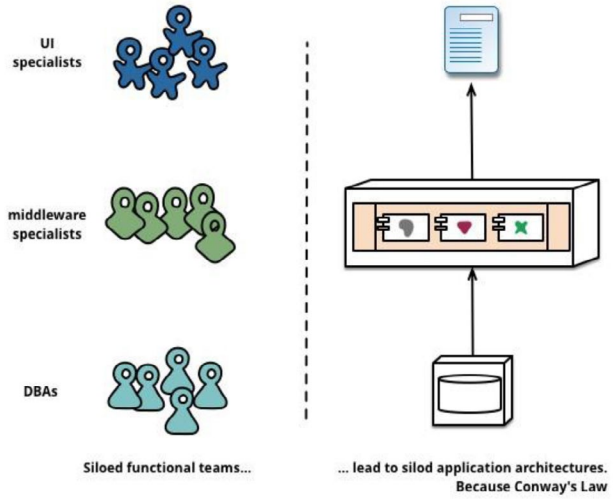

Figura 6. Ley de Conway

Fuente: [2].

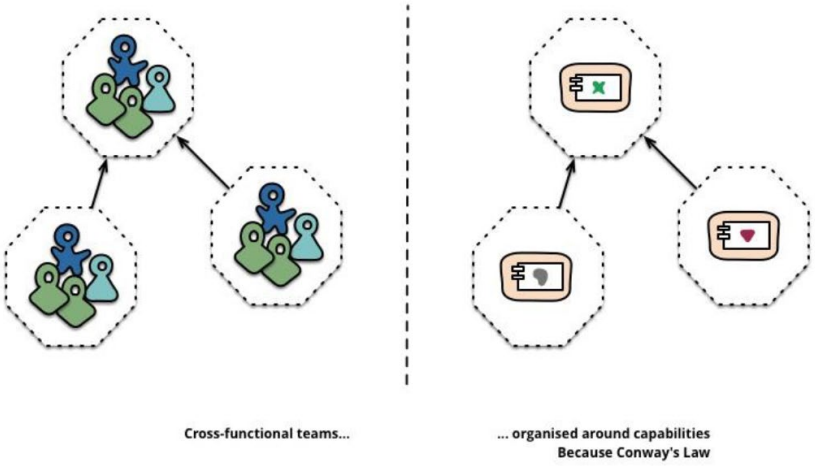

Figura 7. Conway vista en microservicios

Fuente: [2].

- Autonomía [5].

Los microservicios son una entidad separada. Deben ser desplegados como un servicio aislado en infraestructuras que se enfocan en ofrecer plataformas como servicio (PAAS, por sus siglas en inglés) o deben estar en su propio sistema. Todas las comunicaciones entre servicios son llamados de red para reforzar la separación y evitar los peligros del estrecho acoplamiento.

- Productos no proyectos [2].

La mayoría de los desarrollos usan los modelos de proyectos en los cuales el objetivo es desarrollar piezas de *software* que una vez terminadas son consideradas completas. Una vez terminado el *software*, es manejado por el encargado a una empresa o área de mantenimientos y el equipo se disuelve. Los promotores de microservicios intentan evitar este modelo con la idea que un equipo debería manejar un producto a lo larga del ciclo de vida de este. Como inspiración se toma la idea de Amazon que dice: "lo construyes, lo ejecutas", donde se empodera al equipo a que tome la completa responsabilidad sobre el *software* en producción.

Lo anterior no quiere decir que en un enfoque monolítico no pueda ser aplicada esta ideología, solo que con servicios más pequeños es más sencillo lograr una familiaridad entre el equipo desarrollador y el producto.

- Diversidad de tecnología [2].

Dado que cada microservicio es una unidad de despliegue independiente, se cuenta con una considerable libertad en las elecciones de tecnología que se puedan tomar. Los microservicios pueden usar diferentes lenguajes, librerías o almacenamiento de datos, con lo que la elección se basa en cuál es el problema por resolver, y con base en ello escoger la mejor herramienta, o a un nivel más granular la librería correcta, pues usualmente en una aplicación monolítica todos los módulos deben usar la misma versión.

No se debe sobrepasar el conocimiento que puede manejar la organización, pues puede ocurrir que el que exista demasiada diversidad tecnológica se convierta en un problema más que en una solución; por ello, se recomienda tener límites en cuanto a las opciones que se puedan tener.

Otro punto muy importante es el poder experimentar. Es muy importante con aplicaciones monolíticas hacer un cambio de tecnología resulta casi imposible, mientras que si esto se desea hacer con un microservicio es mucho más sencillo dado su tamaño y control.

MANEJO DE DATOS

Teniendo en cuenta las características de los microservicios enunciadas anteriormente, se puede decir que estos abren una gran gama de posibilidades con respecto a qué se puede o no utilizar incluso en el planteamiento de la solución para el modelado de la información de una aplicación, la tecnología o herramienta usar entre otras interrogantes. Por ello, este apartado se enfoca en dar un pequeño abre bocas sobre los dos sistemas de bases de datos más usados en el mundo, las bases de datos relacionales y las bases de datos NoSQL.

Bases de datos relacionales (RDBMS)

Informáticamente, una base de datos es un sistema formado por un conjunto de datos almacenados en memorias masivas que permiten acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos. Definiéndolas formalmente, se puede tomar como un conjunto exhaustivo (en su modelización del mundo real) de datos estructurados, fiables y homogéneos, organizados independientemente de su utilización y de su implementación en máquina, accesibles en tiempo real, compartibles por usuarios concurrentes que tienen necesidades de información diferentes y no predecibles en el tiempo [6]. Entonces, una base de datos relacional es una base de datos donde todos los datos visibles al usuario están organizados estrictamente como tablas de valores y donde todas las operaciones de la base de datos operan sobre estas tablas.

Estas bases de datos son percibidas por los usuarios como una colección de relaciones normalizadas de diversos grados que varían con el tiempo. Los sistemas relacionales son importantes porque ofrecen muchos tipos de procesos de datos como: simplicidad y generalidad, facilidad de uso para el usuario final, períodos cortos de aprendizaje y las consultas de información se especifican de forma sencilla.

Bases de datos NoSQL

Hacen referencia a aquellas bases de datos que no se ajustan al modelo de bases de datos relacionales y a sus características, por ejemplo, los que no tienen esquemas, no usan SQL ni permiten *joins*, tampoco garantizan la propiedad ACID (*atomicity*, *consistency*, *isolation*, *durability*); sin embargo, en contraposición manejan un concepto denominado BASE (*basically available, soft-state y eventual consistency*). Se basan en escalar horizontalmente, consumo alto de la memoria principal del computador, resuelven problemas de manejo de altos volúmenes de información y la inmensa cantidad de consultas y transacciones diarias [7].

Aunque existen docenas de bases de datos, principalmente están dentro de las tres categorías que se mencionan a continuación [8].

- Modelo de documentos: almacena la información en documentos. Estos documentos usualmente usan una estructura similar a un JSON. Proveen una manera intuitiva y natural de modelar información, alineada con la programación orientada a objetos, donde cada documento es un objeto. La noción de esquema es dinámica dado que cada documento puede contener diferentes campos, esta flexibilidad puede ayudar a modelar información sin estructurar; adicionalmente, las bases de datos de documentos ofrecen robustez en las consultas que los desarrolladores esperan de las bases de datos relacionales.
- Modelo de grafos: usan estructuras con nodos, aristas y propiedades para representar datos. Básicamente la información es modelada como una red de relaciones entre elementos específicos. Su principal atractivo es que hace sencillo el modelar y navegar entre las relaciones de las entidades de una aplicación. Bases de datos de grafos son útiles en casos donde relaciones de desplazamiento son el centro de la aplicación, como navegación entre las conexiones de una red social.

- Modelo llave-valor y columnas amplias: son las bases de datos típicas, desde una perspectiva del modelado de datos. Cada ítem en la base de datos es almacenado con un nombre de atributo junto con su valor; sin embargo, el valor es oculto para el sistema, los datos son solo accedidos por medio del nombre de atributo (llave). Son usados usualmente por aplicaciones que solo requieren obtener datos por una única llave. El atractivo de estos sistemas es su desempeño y escalabilidad, el cual puede ser altamente optimizado debido a la simplicidad de los patrones de acceso a datos y el ocultamiento de datos en sí.

CASO DE ESTUDIO

Antes de describir el caso de estudio, es importante mencionar que el problema se describirá en el contexto de comercio en Colombia; con esto en mente, se plantea la necesidad de ofrecer a las tiendas de barrio (negocios que están enfocados a la venta al detal, su tamaño es pequeño y son utilizados en su mayoría para consumos menores), una herramienta que les permita competir con las grandes superficies —que día a día se interesan en estar más cerca de los usuarios mediante versiones exprés de sus megatiendas— y les ofrezca a los clientes un valor agregado que les facilite el consumo de productos.

La herramienta permite solicitar pedidos a las tiendas de barrio desde una plataforma web o móvil, donde se encontrarán las tiendas, historial de pedidos, descripción de ellos, recordatorios de pedidos hechos, sugerencias de compra basado en sus gustos, entre otros.

Con el objetivo de desarrollar una arquitectura de microservicios, construida de manera iterativa e incremental, se plantea iniciar por el análisis y diseño de la solicitud de pedidos por parte de los clientes. Para ello la funcionalidad a nivel de negocio es descrita por las siguientes dos historias de usuario:

- Como cliente deseo poder solicitar productos a la tienda desde mi computador, para facilitar mi comunicación con esta.
- Como tendero deseo aceptar pedidos de mis clientes por la herramienta para no perder ningún pedido.

Teniendo como base las historias anteriores, el objetivo será diseñar el microservicio de administración de pedidos excluyendo las demás funcionalidades así como la interfaz; para este propósito se planta una solución simple con un modelo de tres capas (tan solo para este microservicio, dado que los otros pueden tener arquitecturas diferentes), la cual se describe en la Figura 8.

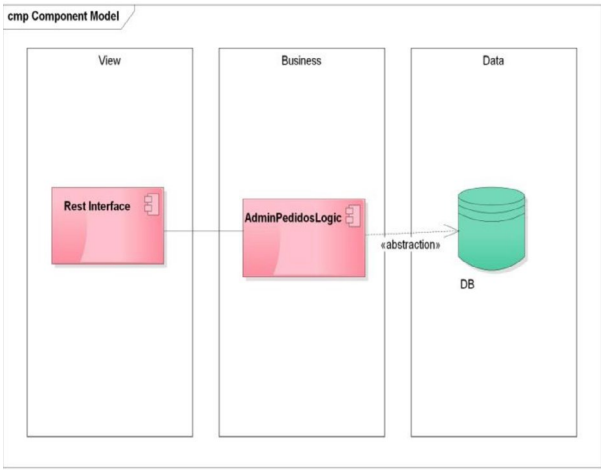

Figura 8. Modelo de capas

Fuente: elaboración propia.

Como puede observarse con este sencillo diseño el objetivo ahora es escoger el lenguaje, así como el modelo de datos; por ejemplo, se podría escoger desarrollar la lógica de negocio de este microservicio con PHP, utilizando un *microframework* como Phalcon [11].

Para el modelo de datos se seleccionará un modelo relacional, el cual puede verse representado en la Figura 9; en este diseño es posible observar que al enfocarse en solo este servicio, e diseño puede ser más sencillo y enfocado en resolver tan solo este problema. Una vez el servicio madure, se podrá replantear nuestro diseño al punto incluso de cambiar nuestro modelo relacional por uno NoSQL.

CONCLUSIONES

Como se pudo apreciar a lo largo del documento, las arquitecturas de microservicios buscan dar solución a las nuevas demandas de desarrollo que el mercado está requiriendo. A alto nivel, estas arquitecturas se centran en resolver un problema dividiéndolo en sus subsistemas. Cada uno de ellos es un microservicio de la solución. Cada uno es resuelto, de manera independiente, lo que permite explotar características como autonomía, diversidad tecnológica, un tamaño más pequeño (en comparación a elaborar un monolito con toda la lógica de negocio), entre otras.

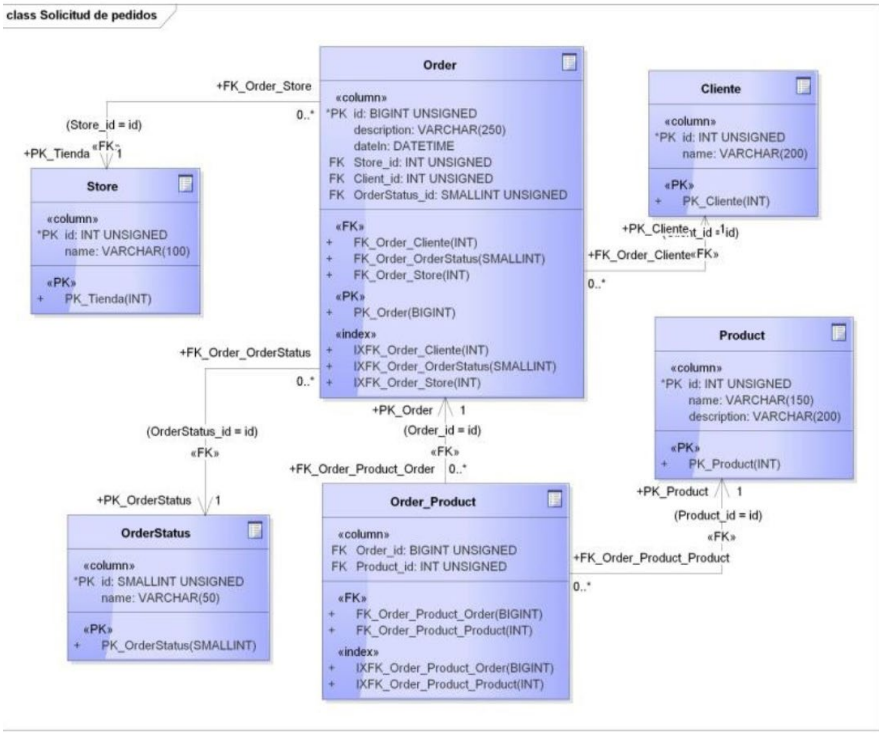

Figura 9. Modelo de datos

Fuente: elaboración propia.

Es importante tener en cuenta que, como cualquier propuesta arquitectónica, no es perfecta y debe ser analizada dependiendo del proyecto. Pero esta arquitectura se perfila como una opción muy interesante para el desarrollo de la ingeniería.

REFERENCIAS

[1] EMag. (2015). Architectures you Always Wondered about Lessons Learnt from Adopting Microservices at eBay, Google, Gilt, Hailo and Nearform. *EMag*, *31*.

[2] Fowler, M. (2014). *Microservices*. Recuperado de http://martinfowler.com/articles/ microservices.html

[3] Mauro, T. (2015). Adopting Microservices at Netflix: Lessons for Architectural Design. Recuperado de https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[4] Richardson, C. (2015). Introduction to Microservices. Recuperado de https://www.nginx.com/blog/introduction-to-microservices/

[5] Newman, S. (2015). *Building Microservices*. Newton (Massachusetts): O'Reilly Media.

[6] Date, C. (2001). *Introducción a los sistemas de bases de datos*. México: Prentice Hall.

[7] Díaz, W. (2013). *Bases de datos NOSQL: llegaron para quedarse*. Recuperado de http://basesdedatosnosql.blogspot.com.co/

[8] MongoDB. (2015). *Top 5 Considerations when Evaluating NOSQL Databases*. Recuperado de https://www.ascent.tech/wp-content/uploads/documents/mongodb/10gen-top-5-nosql-considerations-february-2015.pdf

[9] Uber. (s.f.). Uber. Recuperado de https://get.uber.com

[10] Hailo. (s.f.). Hailo. Recuperado de https://www.hailoapp.com

[11] Phalcon. (s.f.). Phalcon. Recuperado de https://phalconphp.com/en/MartinFowler