UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Microservicio Web para la interacción con el estándar ISO-8583: una implementación open source.
Autores: Ignacio Julián Brocchi
Director: Patricia Bazán
Codirector: Matías Pagano
Carrera: Licenciatura en Sistemas

Resumen

El avance tecnológico de los últimos tiempos en el área de e-commerce y servicios digitales, donde se realizan pagos con tarjetas de crédito y débito, impulsó la necesidad de poder integrar servicios bancarios con distintas plataformas digitales. Éstas suelen alimentarse de APIs que proveen la funcionalidad necesaria para realizar estas operaciones bancarias. Dichas APIs, para poder exponer estas funcionalidades, consumen servicios bancarios que utilizan transacciones bajo el estándar ISO 8583.
El estándar ISO 8583 definido por la International Organization for Standardization define un formato de mensaje y un flujo de comunicación para que diferentes sistemas interactúen entre sí, a través de transacciones electrónicas realizadas por poseedores de tarjetas de crédito.
El objetivo de este trabajo es desarrollar un proyecto open source para facilitar el uso de un servicio que implemente el estándar ISO 8583 a través de un microservicio desarrollado en un lenguaje moderno como lo es Golang, siendo una alternativa a desarrollos existentes pagos o basados en otros lenguajes, y ofreciendo una solución desacoplada y sin restricciones de lenguaje de programación.

Palabras Clave

ISO 8583 - Microservicio - Golang - GoKit - Docker - API Rest

Conclusiones

En este trabajo se implementa un estándar utilizado en transacciones con tarjetas de crédito y débito como un microservicio encapsulado que pueda ser reutilizado por cualquier organización.
Debido al dominio en el cual se usa este tipo de estándares, el entorno bancario, no hay una forma pública de ver funcionando el desarrollo en un entorno productivo. Esto mismo también implica que no existe información pública de quienes implementan el estándar en sus servicios ni de cómo lo usan.
Sin embargo, es clara la necesidad de contar con un servicio que implemente el estándar como de manera desacoplada, genérica, con un lenguaje moderno y de código abierto, habida cuenta la gran cantidad de plataformas digitales que operan con tarjetas de crédito y débito y la interoperabilidad requerida entre ellas a nivel global.

Trabajos Realizados

- Análisis de la estructura y funcionamiento del estándar ISO 8583.
- Análisis de una solución para entornos con microservicios, lenguaje de programación Golang y biblioteca GoKit, virtualización con Docker.
- Desarrollo de microservicio capaz de convertir información en formato JSON a una transacción ISO 8583 y viceversa.
- Despliegue en entorno de prueba con cliente Mock para poder probar el funcionamiento del microservicio.
- Desarrollo de página web para consumir el microservicio de prueba y obtener resultados mejor presentados.

Trabajos Futuros

A futuro el trabajo podría ser continuado agregando soporte para las versiones posteriores del estándar ISO 8583. Actualmente existe una configuración base para la primera versión del estándar publicada en 1987, por lo que se podría agregar las configuraciones para las versiones de 1993 y 2003, permitiendo seleccionar cuál versión se desea utilizar.
Agregar soporte a gRPC ya que el microservicio solo recibe peticiones HTTP.
Permitir a los que utilicen el microservicio elegir de qué forma se quiere codificar el bitmap, ofreciendo EBCDIC (Extended Binary Coded Decimal Interchange Code) como opción adicional a Hexadecimal representado en ASCII la cual se utilizó en este trabajo. Esta codificación puede ser implementada también para los campos de datos del mensaje, el estándar no especifica cómo codificar puntualmente los datos de un mensaje por lo que podría ser una funcionalidad posible de agregar en el futuro.
Otra mejora aplicable a futuro es poder elegir una configuración personalizada para cada campo del mensaje ISO8583, ya que las organizaciones no están obligadas por el estándar a respetar las especificaciones de los campos.

Fecha de la presentación: Octubre 2022

Microservicio Web para la interacción con el estándar ISO-8583: una implementación open source.

Autor: Ignacio Julián Brocchi
Directora: Patricia Bazan
Codirector: Matias Pagano

Índice

1. Introducción

En el contexto de la pandemia causada por el COVID-19, los procesos que impulsan a las empresas a integrar la tecnología digital en todos los aspectos del negocio, se han acelerado. Hubo una transformación en la manera de realizar transacciones comerciales. El nuevo contexto provocó que surja la necesidad de llegar a más usuarios desde cualquier lugar y cualquier dispositivo, como también la necesidad de poder realizar operaciones bancarias con tarjetas ya sea para realizar compras o pagar los servicios ofrecidos.
En este mismo sentido se potenció el uso de plataformas digitales que operan utilizando transacciones con tarjetas, como por ejemplo: 1- plataformas de e-commerce, que operan como tiendas on line, 2- plataformas de streaming, que ofrecen sus servicios digitales mediante membresía, 3- billeteras digitales, aplicaciones móviles para operar financieramente sin una cuenta en un banco pero respaldada por tarjetas de crédito o débito o 4 - servicios de homebanking, plataformas y aplicaciones donde pueden operar con sus cuentas y tarjetas ingresando a una banca virtual, pagar servicios y realizar transferencias sin necesidad de asistir presencialmente a una sucursal del banco.
Esta transformación digital requiere, a su vez, de tecnologías habilitantes para la construcción de un software que permita la interoperabilidad de distintos servicios.
El estándar ISO 8583 (*ISO 8583:1993 - Financial Transaction Card Originated Messages — Interchange Message Specifications*, n.d.) definido por la International Organization for Standardization define un formato de mensaje y un flujo de comunicación para que diferentes sistemas interactúen entre sí, a través de transacciones electrónicas realizadas por poseedores de tarjetas de crédito.
Si bien el estándar ISO involucra principalmente a la banca, alcanza a todas las organizaciones que utilizan medios de pago electrónicos, ya que para poder realizar operaciones con tarjeta de crédito deben integrar sus servicios con servicios bancarios que lo implementan.
Así, plataformas digitales mencionadas anteriormente, suelen alimentarse de APIs que proveen la funcionalidad necesaria para realizar estas operaciones bancarias. Dichas APIs, para poder exponer estas funcionalidades, consumen servicios bancarios que utilizan transacciones bajo el estándar ISO 8583.
En base a lo explicado anteriormente, el objetivo será desarrollar un proyecto open source para facilitar el uso de un servicio que implemente el estándar ISO-8583 a través de una API Rest (*REST API*, 2021) desarrollada en Golang (*The Go Programming Language*, n.d.) como alternativa a desarrollos existentes pagos o basados en otros lenguajes.
Golang, un lenguaje que se adapta a la arquitectura donde se va a aplicar, rápido de replicar bajo demanda y con la posibilidad de tener varias réplicas en funcionamiento a bajo costo dado que es un lenguaje liviano, lo que lo hace óptimo para su funcionamiento en la nube (Malik, 2019). Una gran comunidad soporta a Golang (*Go - DEV Community*, n.d.) e invierte en su crecimiento, así como también grandes empresas reconocidas lo utilizan y retroalimentan, entre ellas, Google, Uber, Netflix, Paypal, American Express, Mercadolibre, Docker, Kubernetes, Dropbox, Twitch, Twitter.
Los desarrollos propuestos para alcanzar el objetivo son:

- Obtener un amplio conocimiento del funcionamiento del protocolo de transacción financiera ISO 8583. Estudio teórico de los aspectos más relevantes que se deben tomar en cuenta para la implementación en Golang del estándar.
- Utilizar el lenguaje de programación Golang para implementar el protocolo de transacción financiera ISO 8583.
- Desarrollar un microservicio que convierta una petición JSON (*JSON.org*, n.d.) al estándar ISO 8583 y una respuesta ISO 8583 a JSON.
- Realizar pruebas para verificar el correcto funcionamiento.

Implementar un servicio que utilice el estándar no es tarea fácil, ISO 8583 define una transacción compuesta por un conjunto de campos donde cada uno tiene una ubicación, corresponde a información específica y debe cumplir ciertas restricciones de longitud y de tipo. Dado esto es que surge la necesidad de desarrollar un microservicio intermediario que haga más sencilla esta integración.
Las tecnologías más utilizadas en en el ámbito bancario, como por ejemplo antiguas versiones de Java (*Java | Oracle*, n.d.) o servicios basados en SOAP (*Simple Object Access Protocol*, n.d.) ya no son tan populares dado que las APIs REST (*What Is REST*, 2022) ha ganado presencia gracias a que funciona sobre HTTP (*RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2)*, n.d.), soporta más formatos de datos y no sólo XML (*XML Essentials - W3C*, n.d.) y, al utilizar REST con JSON puede ser consumido desde navegadores y fácil de integrar con sitios web existentes.
Las APIs REST que reciben y envían información en formatos como XML o JSON deben comunicarse con un servicio bancario que implemente el estándar ISO 8583, por lo que es necesario hacer una conversión de un formato de datos como JSON a una trama donde la información debe representarse de forma específica.
Teniendo en cuenta que la pandemia aceleró la transformación digital donde muchas empresas y organizaciones, entre ellas los bancos, buscan digitalizar sus procesos para poder incrementar su llegada a los usuarios, ofreciendo la oportunidad de realizar todo tipo de operaciones desde la comodidad de su casa, a través de cualquier dispositivo personal. En este sentido, contar con una implementación moderna, como componente atómico, para operar con el estándar ISO 8583 mejorará su uso por parte de la comunidad de desarrolladores involucrados en soluciones de índole financiera, comercial y bancaria.

ISO 8583 (1987)

ISO 8583 (*ISO 8583*, n.d.) es un estándar internacional para transacciones financieras originadas en tarjetas de crédito y débito. Define el formato del mensaje y el flujo de comunicación para que diferentes sistemas puedan interactuar. Las transacciones incluyen compras, extracciones, depósitos, reintegros, reversos, consultas de saldo, pagos y transferencias entre cuentas.
Si bien el estándar define un formato para la transacción, deja lugar a que la organización que lo utilice modifique el formato para adecuarse más a sus necesidades.

Una mensaje ISO 8583 consta de 3 partes:

1. MTI (Message Type Indicator) o Indicador de Tipo de Mensaje
2. Bitmaps que representan qué campos se utilizaron en el mensaje. Puede ser 1 o más.

3. Campos del mensaje, donde se encuentran los datos.

Las dificultades de implementación se hacen presentes al momento de crear o leer un mensaje en el formato que el estándar ISO 8583 indica, desde una aplicación desarrollada en cualquier lenguaje. La creación del mensaje implica transformar información comprensible por los humanos en una cadena de bytes, y por lo tanto, la lectura de un mensaje sería el camino inverso, convertir una cadena de bytes en información legible nuevamente. Para poder hacer estas conversiones, el estándar explica cómo debe crearse cada sección del mensaje, incluyendo tamaño de la sección, formato de los datos, descripción de los campos internos, entre otros. En el Anexo 1 se explica detalladamente cómo se debe presentar la información en cada sección.
Conociendo las dificultades de trabajar con mensajes bajo el estándar ISO 8583, cada aplicación que se desarrolle dentro de una organización que interactúe con dichos mensajes debería implementar toda la lógica de conversión, para poder crear y leer la información que intercambian, haciendo que la aplicación sea bastante más grande y compleja.
Por otro lado, como se mencionó anteriormente, el mensaje ISO 8583 es finalmente una cadena de bytes. Para poder enviar y recibir dichas cadenas de bytes se utiliza TCP (*RFC 793: Transmission Control Protocol*, n.d.), se genera una conexión y se envía el mensaje como trama de un servicio a otro. Implementar esta funcionalidad también agrega complejidad a cada aplicación que se desarrolle, ya que también debe gestionar una conexión TCP que dependiendo el entorno de la aplicación puede ser complejo y/o inseguro.

2. Soluciones existentes y comparación con la propuesta

Actualmente hay soluciones desarrolladas para facilitar el manejo de transacciones bajo el estándar ISO 8583 e incluso ofrecen utilidades para realizar pruebas. Se analizaron las soluciones enumeradas más adelante con el fin de tener una visión de cómo se utilizan y qué alcance tienen las utilidades que ofrecen.

2.1. Moov-io

Moov-io (*Moov-Io/iso8583: A Golang Implementation to Marshal and Unmarshal Iso8583 Message.*, n.d.) es un paquete para Go que facilita el manejo de tramas ISO8583, con objetivo a futuro de convertirse en API. Dicho paquete ofrece estructuras y funciones para crear un mensaje ISO 8583 que luego se traduce en una trama en formato ASCII o Hexadecimal.
Para utilizarlo se debe crear una estructura de configuración en la que campo por campo de la trama se debe indicar el largo y formato del mismo.
Una vez configurado, se puede utilizar la estructura Message que el paquete ofrece para crear un mensaje ISO8583 indicando posición y valor, la cual a través de una función se convierte a una transacción ISO 8583.
Moov-io provee paquetes fáciles de usar para crear transacciones ISO 8583 y utilidades para el envío y recepción de la misma, sin embargo, aún es un paquete para Go que se debe importar (aún no se llegó a su objetivo de ser una API) y se debe incorporar al desarrollo la funcionalidad que provee para poder utilizarlo generando gran dependencia y limitando únicamente su uso a desarrollos en Go. Esto lleva a que el código que se

desarrolle esté fuertemente acoplado al paquete de Moov-io, lo cual puede ser un problema dado que para incorporar las mejoras del paquete en sus nuevas versiones pueda llevar a tener que realizar cambios en el desarrollo. Por otro lado, como ventaja se puede ver que recibe soporte y se continúa trabajando en su mejora.
Por otra parte, Moov-io tiene la ventaja de existir hace más tiempo, su primera versión v0.0.1 fue publicada el 17 de julio del 2020 y varias personas llevan aportando al proyecto desde entonces, sin embargo, por sus características de despliegue, provoca una alta cohesión y acoplamiento con la solución en la cual se debe incorporar.

2.2. Licklider

Licklider es una compañía de Software que ofrece soluciones en el ámbito clínico y bancario.
Dicha empresa expone un parser web (*Licklider - Free ISO8583 Parser*, n.d.) donde se puede introducir una trama ISO8583 de forma manual y la convierte en información legible o a formato JSON, indicando qué campos de la trama son utilizados y sus respectivos valores.
Licklider es sólo una interfaz web que ofrece un conversor online. Es útil para hacer pruebas pero no ofrece ninguna funcionalidad para incorporar a un proyecto de desarrollo.

2.3 MOFAX

MOFAX es un paquete Go que provee funcionalidad para codificar y decodificar transacciones ISO 8583. El paquete no fue terminado de desarrollar y no recibe actualizaciones desde Febrero del 2018. No es una solución funcional al problema planteado en este trabajo.

2.4 Nuestra Propuesta

Nuestra propuesta en este trabajo da una alternativa independiente del lenguaje con el que se esté desarrollando. Al ser un microservicio es posible incorporarlo a cualquier solución con una arquitectura de microservicios, sin tener que incorporar paquetes o código externo al código de la aplicación, ya que sólo se le debe enviar una petición HTTP para utilizarlo y prácticamente todos los lenguajes ofrecen las herramientas para poder hacerlo. De esta manera se elimina la dependencia y el acople del código con paquetes externos a la aplicación que se está desarrollando.
El estándar ISO 8583 si bien especifica cómo debe utilizarse, deja cierta libertad a las organizaciones para hacer su propia versión del mismo, por lo que, si dentro de una organización se decide modificar cómo operar un campo del mensaje, los cambios en los desarrollos serían mínimos ya que la modificación principal debería hacerse en el microservicio de la solución propuesta en este trabajo, quedando la tarea simple de modificar cómo se envía la información en las aplicaciones que lo utilizan.
Por otro lado, al estar desarrollado en un lenguaje liviano como Go, el componente desplegado no hace un consumo notable de recursos, por lo que también no sería costoso replicar el microservicio ante una sobrecarga de peticiones.
En conclusión, la solución propuesta es un microservicio eficiente y fácil de mantener que

puede ser incorporado a cualquier arquitectura de microservicios.

2.5 Comparación

Habiéndose analizado algunas de las soluciones existentes se puede realizar una comparación con el trabajo propuesto donde se pueden ver los puntos que se tuvieron en cuenta y se buscó mejorar.
El mecanismo de integración que utiliza cada solución se planteó como una problemática, ya que si genera mucha dependencia o el desarrollo resultante se ve muy acoplado a la solución externa, en algún futuro podría haber problemas de compatibilidad o incluso dejar de funcionar.
Por otro lado, se tuvo en cuenta la importancia del lenguaje de programación utilizado, dado que algunas soluciones sólo se pueden integrar si se está utilizando un lenguaje específico, limitando así su utilización.
Finalmente, la comunidad de desarrolladores le dan uso y soporte a las soluciones publicadas asegurando su funcionamiento y fidelidad, este punto es importante a la hora de comparar las soluciones existentes, dando prioridad a las soluciones mantenidas ya que estas tienen menos riesgo de desaparecer o presentar problemas a futuro, además de seguir actualizándose para mantenerse vigentes y aprovechar lo que las tecnologías van mejorando y perfeccionando.
En la Tabla 1 - Características comparativas de soluciones existentes, se analiza cada solución y el trabajo desarrollado comparando los puntos que se consideraron más importantes en cada uno.

Tabla 1 - Características comparativas de soluciones existentes

Características	Moov -io	Licklider	MOFAX	Solución propuesta
Mecanismo de integración	Importación de código	No posee	Importación de código	Microservicios
Dependencia del lenguaje	Alta	No aplica	Alta	Baja
Actividad de la comunidad que lo utiliza	Alta	No aplica	Baja	Aún no medible

Este trabajo da una alternativa a los problemas de integración y acople de código ya que no es necesario agregar ni importar ningúna biblioteca, incluyendo que no existe limitaciones con el lenguaje de programación que se utilice por ser un componente externo.
Finalmente, la desventaja de la solución propuesta en este trabajo, en comparación con las otras soluciones, es que, al ser un microservicio, solo se puede integrar a una arquitectura de microservicios, acotando su utilización a soluciones implementadas con dicha arquitectura.
A continuación se explican las tecnologías que se utilizaron para el desarrollo y el por qué fueron seleccionadas en base a sus características.

3. Tecnologías utilizadas

En esta sección se describen las tecnologías que se decidió utilizar para desarrollar el trabajo. Cada tecnología será explicada por separado para luego en la sección 4 abordar cómo se utilizó en la implementación.
Las tecnologías que se seleccionaron para el desarrollo fueron elegidas por su utilización con microservicios y para explicarlas, se debe explicar primero qué es un microservicio.

Los microservicios (Lewis & Fowler, 2014) son una arquitectura de software donde el software está compuesto por pequeños servicios independientes que trabajan en conjunto para llevar a cabo una tarea.
Para explicar la arquitectura de microservicios es útil hacer una comparación con la arquitectura monolítica, una aplicación monolítica construida como una sola unidad.
Usualmente las aplicaciones se construyen en 3 partes: una interfaz de usuario, una base de datos y una aplicación en el servidor donde se encuentra la lógica de dominio, y opera tanto la información que se encuentra en la base de datos como la que se envía hacia la interfaz de usuario.

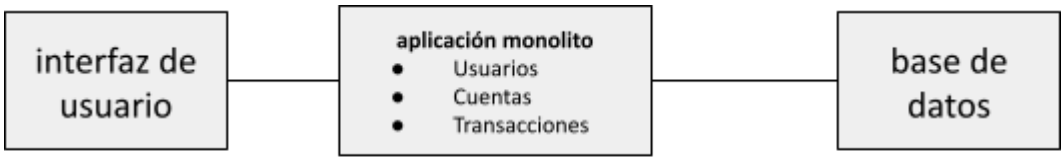


Figura 1 - Arquitectura monolítica

Ésta aplicación es un monolito, una sóla unidad con toda la funcionalidad. Por ejemplo, si una aplicación bancaria se viera como monolito (Figura 1 - Arquitectura monolítica) la misma aplicación proveería la posibilidad de operar con Usuarios, Cuentas y Transacciones.
Una desventaja de esta arquitectura es que si se debe realizar un cambio en la forma en que se opera con un Usuario, se debe volver a desplegar toda la aplicación, incluso sin haber hecho modificaciones en las funcionalidades de Cuentas y Transacciones, lo que hace que sea costoso aplicar cambios al desarrollo.
Además, a medida que la aplicación crece, se torna más complicado sostener la modularidad de la aplicación y que un cambio a un módulo solo afecte a ese módulo.
Por otro lado, la frecuencia en que se crean Usuarios seguramente sea menor a la frecuencia en que se crean Cuentas, que también debe ser menor a la frecuencia en la que se crean Transacciones, por lo tanto, cada funcionalidad recibe distintas cargas, pero al ser una sola aplicación, si la aplicación se sobrecarga por Transacciones, una operación con Usuarios podría demorarse, por lo tanto si se desea atender la sobrecarga de Transacciones escalando la aplicación (crear una réplica idéntica para poder soportar más peticiones), se debe escalar el monolito completo ocupando recursos en funcionalidades que no lo necesitan,en este caso, las de Usuarios y Cuentas.

Si la misma aplicación fuese creada con microservicios, cada funcionalidad podría ser un microservicio distinto (Figura 2 - Arquitectura microservicios).

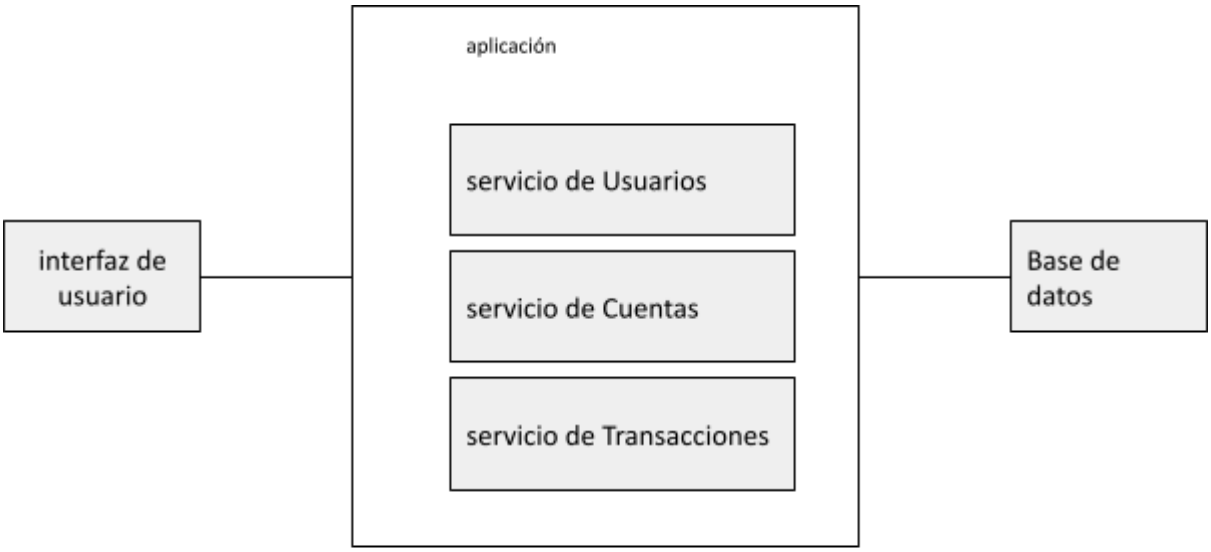


Figura 2 - Arquitectura microservicios

Esto quiere decir que existirían tres microservicios, uno exclusivamente para Usuarios, otro exclusivamente para Cuentas, y uno más, exclusivamente para Transacciones, por lo tanto quedarían las funcionalidades separadas en módulos totalmente independientes pero que en conjunto ofrecerían la misma funcionalidad que la aplicación monolítica ofrece.
Con esta arquitectura, si se realiza un cambio sobre cómo se opera con los Usuarios, el resto de la aplicación no se vería afectada y solo debería volver a desplegarse el microservicio de Usuarios, una tarea más simple ya que el riesgo de romper la modularización ya no existe. También, siguiendo el ejemplo de la sobrecarga en una arquitectura monolítica, si ocurre una carga alta en el microservicio de Transacciones, los servicios de Usuarios y Cuentas no se verían afectados, así cómo también se puede realizar un escalado del microservicio de Transacciones para soportar la demanda.
Estos microservicios, a su vez, pueden estar escritos en distintos lenguajes y ser manejados por distintos equipos.
A continuación, se describirán las tecnologías elegidas para desarrollar este trabajo y con las que es posible crear un microservicio con las características que se explicaron al comienzo de esta sección.

3.1 Go-lang

El lenguaje elegido es Go, un lenguaje publicado en 2009 con el objetivo de ser un lenguaje simple y fácil de aprender.
Es un lenguaje tipificado estáticamente que se percibe como un lenguaje interpretado y tipificado dinámicamente. Ofrece un manejo de paquetes basado en URIs que al compilarse una aplicación genera un binario pequeño sin dependencias, logrando una mejor velocidad de ejecución.
Go está diseñado para, entre otras cosas, desarrollar aplicaciones web escalables y

seguras, sin perder simplicidad y legibilidad logrando crear código confiable y fácil de mantener (*Go for Web Development*, 2019).
La biblioteca estándar incluye paquetes para necesidades comunes como servidores y clientes HTTP y de alto rendimiento, también provee funcionalidades para el manejo de JSON y XML, y una variedad de funcionalidades de seguridad y cifrado.
Go compila rápidamente en código de máquina, cuenta con un *garbage collecto*r y el poder de aplicar *reflection* en tiempo de ejecución.
Una de las principales características de este lenguaje es que provee una librería nativa de concurrencia, lo que hace eficiente el manejo de los hilos de ejecución e incluye herramientas para detectar *race conditions* en tiempo de ejecución.
Go nativamente tiene paquetes que dan soporte para las tecnologías más recientes, como HTTP , bases de datos como MySQL (*MySQL*, n.d.), MongoDB (*MongoDB: La Plataforma De Datos Para Aplicaciones*, n.d.) y ElasticSearch (*Elasticsearch: El Motor De Búsqueda Y Analítica Distribuido Oficial*, n.d.), y estándares de encriptación como TLS 1.3 (*TLS - Seguridad De La Capa De Transporte*, n.d.).
Las aplicaciones web de Go se ejecutan de forma nativa en Google App Engine (*Google App Engine*, n.d.) y Google Cloud Run (*Cloud Run: Container to Production in Seconds*, n.d.), para facilitar el escalado, o en cualquier entorno, nube o sistema operativo gracias a la portabilidad de Go.

3.1.1 Go kit

Go kit (*Go Kit - A Toolkit for Microservices*, n.d.) es una herramienta para crear microservicios en Go, su objetivo es facilitar el desarrollo de aplicaciones en sistemas distribuidos - conjunto de componentes de software, la función que cumple cada componente y que interrelación hay entre los mismos- , y así poder dar prioridad a desarrollar la lógica de negocio.
Go kit es una colección de paquetes Go que ayudan a crear microservicios robustos, confiables y fáciles de mantener, provee funcionalidades que la biblioteca nativa de Go no ofrece. Los paquetes de Go kit exponen estructuras y funciones con funcionalidad que se repiten en cada microservicio, ahorrando así tener que escribir bloques de código que serían necesarios para construir un microservicio utilizando la librería nativa de Go, de esta manera, el desarrollo mínimo de un microservicio - el código mínimo para levantar un microservicio que reciba peticiones y responda - resulta más rápido y menos complejo.

Go kit también propone diseño y arquitectura para los servicios que lo utilicen.
Los servicios con Go kit se distribuyen en 3 capas:
1. Transport
2. Endpoint
3. Service (Business Logic)

Una petición que reciba un microservicio entra por la capa 1, baja hasta la capa 3 y la respuesta recorrería el camino inverso, subiendo desde la capa 3 hasta la capa 1.

Transporte
Cuando se construyen microservicios basados en sistemas distribuidos, dichos microservicios se comunican entre sí utilizando transportes concretos como HTTP, gRPC (*GRPC - Official Site*, n.d.), Thrift (*Apache Thrift*, n.d.), y net/rpc (*Rpc Package - Net/rpc*, n.d.). Un microservicio puede soportar ambos transportes o más.

Endpoint
Un Endpoint es como una acción en un controlador, envuelve una funcionalidad de la capa de servicio, como si fuera un adapter, con una interfaz de petición y respuesta. Si el microservicio implementa dos transportes, como por ejemplo HTTP y gRPC, es posible que tenga dos métodos para enviar peticiones al mismo Endpoint.

Service
Los servicios son donde toda la lógica de negocio es implementada. Un microservicio generalmente une múltiples Endpoints. En Go kit, los servicios se modelan como interfaces, y las implementaciones de esas interfaces contienen la lógica de negocio. La capa de servicio no debe tener ningún conocimiento de qué Endpoint o Transporte ,o qué codificación y decodificación de petición y respuesta se está utilizando en el microservicio.

Middlewares
Go kit propone la utilización de middlewares - capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones - para incorporar funcionalidad sin incluir código que no corresponde a la capa en la que se implementa.
Un middleware puede envolver un Endpoint - componente funcional del backend - para agregar control de la cantidad de peticiones que recibe, o hacer balance de carga, también un middleware puede envolver un servicio y agregar funcionalidad para la toma de métricas de negocio o guardar algún registro de la información que se opera. Así como las capas tienen un objetivo, cada middleware debe satisfacer una sola necesidad y en caso de tener más de uno, los middleware se pueden encadenar envolviendo un componente y luego a ellos mismos.

Microservicio con Gokit
Uniendo estos conceptos, un microservicio con Go Kit estaría modelado como una cebolla, en capas (Varghese, n.d.). Estas capas están agrupadas por los tres dominios. La capa más profunda es donde se encuentra la definición del servicio y donde está implementada toda la lógica de negocio. La capa intermedia es la de endpoint donde cada método del servicio es abstraído en un Endpoint. Finalmente, la capa exterior es la capa de transporte, donde los Endpoints están vinculados a transportes concretos como HTTP.

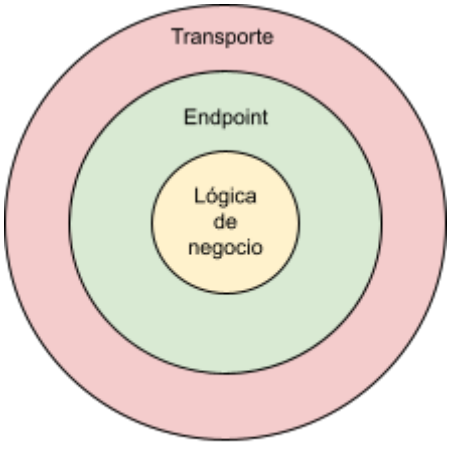


Figura 3 - Capas Gokit

El núcleo contiene la lógica de negocio y define en una interfaz de servicio y creando una implementación concreta de la misma. Luego, se implementan los middlewares para proveer funcionalidad adicional, como guardar registro de ciertas operaciones, datos analíticos; cualquier información que se deba conocer sobre el dominio de negocio.
Go kit provee middlewares para las capas de Endpoint y de Transporte, para limitar las peticiones que recibe, middleware de cortocircuito, balanceo de carga y seguimiento, generalmente funcionalidades que no se relacionan con el dominio de negocio.

Go kit alienta el diseño de los microservicios como un conjunto de componentes que interactúan entre sí, incluyendo varios middlewares con un solo propósito específico. Todos estos componentes se instancian y conectan en la función main de la aplicación, imponiendo así el tiempo de vida de cada componente al tiempo de vida del main, y obliga también a que el alcance de cada componente se encuentra también dentro de la función main, dejando así solo la posibilidad de asignar dependencias entre componentes a través de parámetros explícitos en sus constructores.

La arquitectura de Go kit es bastante simple (Figura 3 - Capas Go kit), pero realmente poderosa, lo que lo hace atractivo para todo tipo de casos de uso, pero al momento en que un microservicio crece, hasta incluso dejar de ser un microservicio y pasar a ser un pequeño monolito, Go kit empieza a ser engorroso, ya que al incorporar lógica de negocio habría que incorporar nuevos endpoints y nuevos métodos en la capa de transporte, que también se deben instanciar en la función main de la aplicación, volviéndose más complejo de mantener.

3.2 Docker

Docker (*Docker Website*, n.d.) es una plataforma que permite empaquetar software en unidades estandarizadas llamadas *imágenes* que incluyen todo lo necesario, incluyendo sus bibliotecas, herramientas de sistema y código, para que el software se ejecute de forma rápida e independiente del entorno.
Las *imágenes* de Docker se convierten en *contenedores* (*What Is a Container?*, n.d.) cuando se ejecutan, y gracias a su portabilidad estos pueden ser utilizados tanto en máquinas locales, máquinas virtuales o en la nube y asegura que funcionen de forma uniforme. De este modo, múltiples *contenedores* pueden ejecutarse en la misma máquina y compartir el Kernel - núcleo - del Sistema Operativo con otros contenedores, cada uno ejecutándose de forma aislada como un proceso en espacio de usuario.
Un contenedor Docker, a diferencia de una máquina virtual, no requiere incluir un sistema operativo independiente. En su lugar, se basa en las funcionalidades del kernel, utiliza el aislamiento de recursos (CPU, la memoria, el bloque E / S, red, etc.) y namespaces separados para aislar la vista de una aplicación del sistema operativo. Adicionalmente, gracias a que no necesita incluir un sistema operativo completo y solo incluye lo necesario para que la aplicación pueda ejecutarse, una imagen de docker ocupa solo una cantidad de megabytes, lo que lo hace eficiente en cuanto a espacio y aporta a su portabilidad.
También, la utilización de contenedores favorece el control de recursos, asignando a cada contenedor ciertos límites que debe respetar en el consumo de memoria, uso de CPU y E/S, muy útil en arquitecturas distribuidas.
Para construir una aplicación con Docker, se necesita crear un archivo llamado *Dockerfile*.

Dockerfile

Un *Dockerfile* (*Dockerfile Reference*, n.d.) es un archivo de texto que contiene las instrucciones para construir una *imagen*.
Para crear una *imagen* se debe partir de otra imagen base. Con la instrucción FROM se indica cual imagen base se va a utilizar, de la cual hereda las dependencias necesarias para la compilación de la aplicación, que puede ser obtenida del repositorio remoto que ofrece Docker y que descarga automáticamente al momento de la construcción de la imagen de la aplicación. El siguiente paso es la creación de un directorio, a través de la instrucción WORKDIR, dentro de la imagen que se está construyendo y que será el destino para las próximas instrucciones que se ejecuten. Aquí es donde se descargan las dependencias y luego se copia el código de la aplicación.
El último paso es la compilación, se indica a través de la instrucción RUN el comando necesario para compilar la aplicación que dará como resultado un ejecutable. Una vez compilada la aplicación, mediante la instrucción CMD se indica el comando que ejecutará la aplicación cuando se utilice la imagen como contenedor.
Finalmente un Dockerfile ser verá parecido a esto:

FROM *unaImagenBase*
WORKDIR */unDirectorio*
RUN *comandoDeCompilación*
CMD *comandoDeEjecución*

Las tecnologías fueron elegidas con el propósito de desarrollar un microservicio eficiente, por esto es que se utilizó Go como lenguaje, que como se explicó anteriormente, es muy útil para crear microservicios y permite crear aplicaciones independientes, ya que incorpora sus dependencias en el binario compilado, logrando así evitar problemas futuros de dependencias externas. Por otro lado, para poder ser desplegado en una arquitectura de microservicios, se pensó en que la aplicación pueda desplegarse como un contenedor, para esto se eligió Docker. Utilizando la imagen de la aplicación, se puede construir el contenedor para ser desplegado en cualquier clúster en el que sea compatible sin necesidad de hacer cambios en la arquitectura.
En conclusión, la aplicación resultante de este trabajo es un componente eficiente e independiente que pueda incorporarse a una arquitectura existente sin costo alguno.

4. Trabajo

Se desarrolló un microservicio capaz de recibir un mensaje en formato JSON, equivalente a un mensaje del estándar ISO 8583 (versión 1987), convertir dicho mensaje según la especificaciones del estándar y luego enviarlo al servicio que corresponda. Una vez obtenida la respuesta en formato ISO 8583, se convierte a formato JSON y se expone como respuesta del microservicio.

4.1 Microservicio en Golang

El microservicio escrito en Golang expone una API para acceder a la funcionalidad de conversión y comunicación con el servicio que utiliza el estándar.
La arquitectura interna está basada en cómo Go-kit sugiere, dividida en capas (Figura 3 - Capas Gokit) que abstraen la solución de las decisiones técnicas:

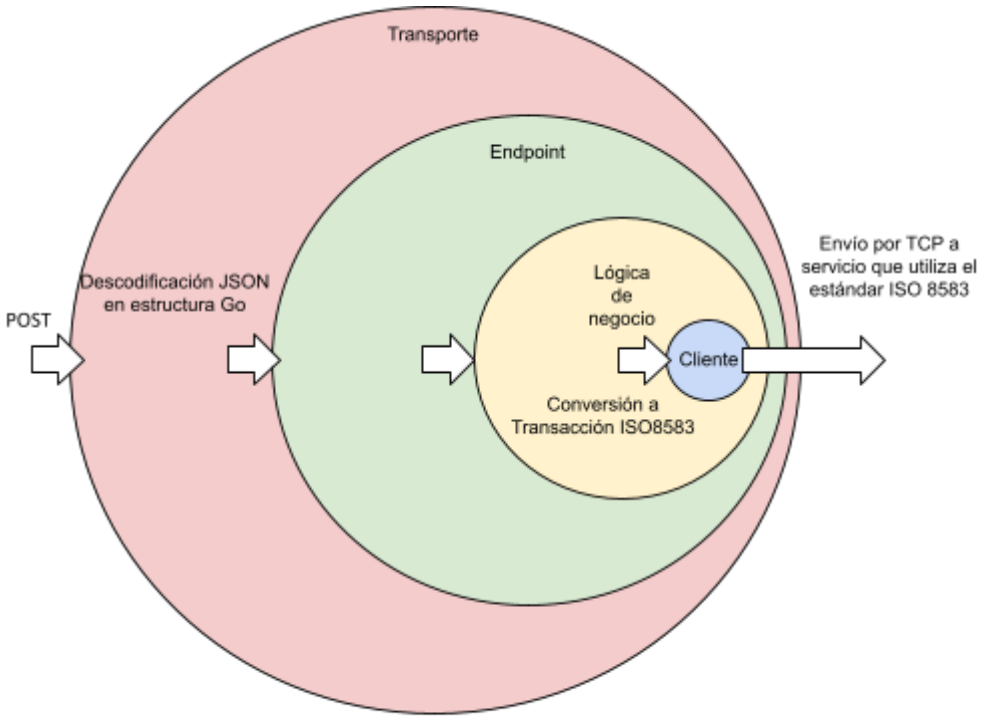


Figura 4 - Microservicio - Flujo petición

En la Figura 4 - Microservicio - Flujo petición se puede ver cómo una petición llega al microservicio y cómo atraviesa las capas para finalmente enviar una petición convertida en una transacción ISO 8583. La capa *Transport* (o de Transporte) es la primera, la cual implementa HTTP para aceptar una petición POST con los datos en formato JSON. En esta capa también se hace la descodificación de la petición HTTP en una estructura Golang que representa un mensaje ISO 8583 y también la codificación de dicha estructura Golang a la respuesta HTTP. Realizada la conversión, la petición continua a la capa *Endpoint*.
La capa *Endpoint*, expone la funcionalidad central del microservicio para poder ser accedida desde la capa *Transport*, recibe la petición y la traslada a la capa *Service*.
La capa de *Business Logic* (Lógica de negocio) o de *Service* (Servicio), recibe de la capa *Endpoint* el mensaje, se hace la conversión de la estructura Golang a trama ISO8583,y se envía el mensaje al componente *Client* el cual se ocupa de enviar la trama al servicio al que se desea acceder, espera su respuesta y luego envía la respuesta a la capa *Service* que realiza la conversión de trama ISO8583 a estructura Golang, ésta es enviada a la capa *Endpoint* y luego a la capa *Transport* donde el microservicio responde en formato JSON con

los datos que obtuvo del servicio.

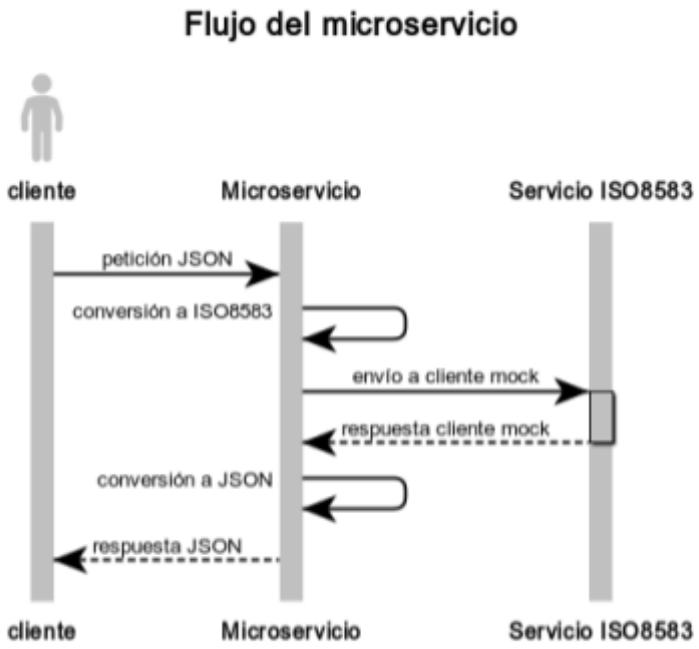


Figura 5 - Diagrama de una petición al microservicio

En la Figura 5 se puede ver cómo un *cliente*, el cual puede ser por ejemplo, otro servicio, envía una petición al microservicio y cómo este envía la petición ISO 8583 al servicio que lo implementa, y luego el flujo de respuesta hasta llegar al usuario una respuesta en formato JSON.

4.1.1 Despliegue con Docker

Para ejecutar el microservicio se configuró un Dockerfile[1] con los pasos necesarios para crear una imagen funcional.
La imagen base utilizada es golang:1.16-alpine . (*Golang - Official Image*, n.d.) Esta imagen para aplicaciones Go está basada en Alpine Linux (*Alpine Linux*, n.d.) , una distribución de Linux eficiente, pequeña y simple.
El siguiente es el Dockerfile desarrollado para esta solución:

```
1. FROM golang:1.16-alpine AS build
2. RUN mkdir -p /app
3. WORKDIR /app
4. COPY . .
5. RUN go build -mod=vendor -ldflags '-s -w' -o /api-iso-to-json main.go
6. EXPOSE 8080
7. CMD [ "/api-iso-to-json" ]
```

Siguiendo la numeración de las líneas, se explica su comportamiento en el Dockerfile:

[1] (“Build your Go image”)

1. Se indica la imagen base.

2. Se crea el directorio donde se va a ubicar la aplicación

3. Se selecciona el nuevo directorio como directorio de trabajo.

4. Se copia el código de la aplicación

5. Se compila el código. Se utiliza go build para compilar una aplicación Go, y con la opción -mod=vendor se indica que debe descargar las dependencias. Luego la opción -ldflags '-s -w' da como resultado binarios sin recursos necesarios para hacer *debug* de la aplicación, logrando así binarios más pequeños. La última opción -o /api-iso-to-json le da nombre al ejecutable resultante. Finalmente, el parámetro main.go es donde se encuentra la función *main* de la aplicación.

6. Para que una petición pueda llegar del exterior al interior del contenedor se debe exponer un puerto, el microservicio funciona sobre el puerto 8080, por lo cual se expone el mismo. Luego al ejecutar el contenedor se debe indicar de qué puerto a qué puerto se debe mapear, por ejemplo se puede indicar que el puerto local 9090 corresponda al 8080 del contenedor.

7. Con la instrucción CMD se indica que el ejecutable api-iso-to-json se debe iniciar cuando se instancia la imagen como contenedor.

Para crear la imagen, desde línea de comandos se ejecuta

```
docker build --pull --rm -f "Dockerfile" -t iso8583tojson:latest "."
```

Luego para ejecutar localmente se utiliza el comando

```
docker run --name=iso2json -p 8080:8080 iso8583tojson
```

4.2 Conversión de JSON a trama ISO8583

Como se mencionó anteriormente, la primera conversión se hace en la capa de transporte donde se decodifica la petición HTTP en una estructura que representa un mensaje ISO 8583. Por ejemplo, el siguiente JSON se recibe en el Body:

```
{
        "mti": "0200",
        "fields": {
                "2": "4321123443211234",
                "3": "000000",
                "4": "000000012300",
                "7": "0304054133",
                "11": "001205",
                "14": "0205",
                "18": "5399",
                "22": "022",
                "25": "00",
                "35": "2312312332",
```

```
            "37": "206305000014",
            "41": "29110001",
            "42": "1001001",
            "49": "840"
        }
}
```

Y su información debe ser cargada en una estructura como la siguiente:

```
type Iso8583 struct {
   Mti    string
   Fields map[int]string
}
```

El valor del campo ‘mti’ se asigna al campo ‘Mti’ de la estructura, y luego cada campo del objeto ‘fields’ con clave numérica (los cuales corresponden a las posiciones del mensaje ISO8583) se mapea dentro del campo ‘Fields’ de la estructura, utilizando como clave del mapa la clave del campo JSON y al valor del mapa el valor del campo JSON.

Luego en la capa de servicio, se construye el mensaje ISO 8583 en base a la estructura obtenida de la capa de transporte.
El primer paso es generar el bitmap que indica qué posiciones del mensaje son utilizadas. El bitmap se puede representar como un arreglo de 64 o 128 posiciones, donde el valor 0 indica que esa posición no se utiliza y el valor 1 indica que si. Una vez creado el bitmap, se lo codifica en Hexadecimal y se obtiene su equivalente string.
Para el request de ejemplo sabemos que los campos que se utilizan son el bitmap es el siguiente:

0111001000 1001000100 0100100000 0000101000 1100000010 0000000000 0000

que luego se codifica a Hexadecimal tomando el valor:

7224448028C08000

Luego se procesa cada campo según su configuración específica, se concatenan uno detrás del otro obteniendo el mensaje ISO 8583 como un string único que representa todos los campos de datos del mensaje.
Finalmente se concatena el valor del MTI, el bitmap convertido a hexadecimal y los datos, formando un sólo string para luego ser convertido en una cadena de bytes, el mensaje ISO 8583 resultante es el siguiente:

```
02007224448028C08000164321123443211234000000000000012300030405413300120502055399
022001023123123322063050000142911000 1001001840
```

Donde 0200 corresponde al MTI, 7224448028C08000 corresponde al Bitmap y 164321123443211234000000000000012300030405413300120502055399022001023123123322063 0500001429110001 1001001840 corresponde a los datos.

4.3 Envío y recepción

Una vez obtenido el mensaje como cadena de bytes, el siguiente paso es el envío, el cual se realiza a través de TCP en el componente *client* (o cliente).
Para realizar el envío, al principio de la cadena de bytes se suman 4 bytes que especifican el largo del mensaje.
Se establece la conexión, se envían los datos:

```
0135010072244480 28C0800016432112344321123400000000000001230003040541330012050205
5399022001023123123322063050000 1429110001 1001001840
```

donde 0135 representa el largo del mensaje.

Luego del envío se espera la respuesta del servicio.
Para la recepción, el proceso es inverso. Se leen los primeros 4 bytes para obtener el largo del mensaje y luego se lee el resto de los bytes que corresponden al mensaje.

4.4 Conversión de trama ISO8583 a JSON

El mensaje de respuesta es recibido por la capa de servicio desde el cliente y se inicia el proceso de conversión de trama ISO 8583 a JSON.
El primer paso es extraer el mensaje en una estructura Go que representa el mensaje ISO 8583. El primer campo, el MTI, se extrae al leer los primeros 4 bytes del mensaje. Luego se deben extraer el Bitmap, o los Bitmaps en caso de haber más de uno. Para esto se extraen los siguientes 16 bytes de datos en Hexadecimal y se decodifican en 64 valores binarios. Una vez obtenido el primer Bitmap, se valida si la primera posición tiene valor ‘1’ para extraer los próximos 16 bytes del segundo Bitmap.
Finalmente se procede a extraer los datos, recorriendo el Bitmap generado se van recuperando en orden los datos del mensaje. En base a la posición indicada en el Bitmap como activa, se extraen los bytes correspondientes según la configuración indicada para esa posición. Por ejemplo, en el siguiente mensaje:

```
0210 7224448028C08000 1643211234432112340000000000000123000304054133001205020553 99
0220010231231233220630500001429110001 1001001840
```

El bitmap 7224448028C08000 convertido a binario: 0111001000 1001000100 0100100000 0000101000 1100000010 0000000000 0000 indica con el valor 1 que la segunda posición está utilizada, por lo tanto debe ser leído de forma que el valor sea el correcto.
El campo 2, correspondiente al “primary account number” es de largo variable, con un máximo de 19 caracteres numéricos y largo expresado con 2 bytes. Por lo tanto para leer el largo del campo 2 se deben extraer 2 bytes desde el byte 16 del mensaje (luego del Bitmap). Se extraen los 2 bytes y se resuelve que el largo del campo es de 16, por lo que el siguiente paso es extraer desde la posición 18 (luego del Bitmap y los dos bytes del largo) los próximos 16 bytes, obteniendo así el valor 4321123443211234.
Luego para leer el campo 3, correspondiente al “processing code”, el mismo es de largo fijo con un tamaño de 6, por lo tanto simplemente se leen 6 posiciones luego del campo 2.

02107224448028C0800016432112344321123400000000000000123000304054133001205020553990220010231231233220630500001429110001 1001001840

Para el campo 4 , el cual corresponde al “amount”, su largo es fijo y el tamaño es de 12, por lo que al igual que con el campo 3 , se leen las 12 posiciones siguientes al campo anterior. Este proceso se repite para cada campo hasta obtener toda la información dentro de la estructura Go de respuesta dando como resultado una estructura con la información de la siguiente tabla:

Id Field	Type	Usage	Value
2	n..19	Primary account number (PAN)	<4321123443211234>
3	n 6	Processing code	<000000>
4	n 12	Amount, transaction	<000000012300>
7	n 10	Transmission date & time	<0304054133>
11	n 6	System trace audit number (STAN)	<001205>
14	n 4	Expiration date	<0205>
18	n 4	Merchant type, or merchant category code	<5399>
22	n 3	Point of service entry mode	<022>
25	n 2	Point of service condition code	<00>
35	z..37	Track 2 data	<2312312332>
37	an 12	Retrieval reference number	<206305000014>
41	ans 8	Card acceptor terminal identification	<29110001>
42	ans 15	Card acceptor identification code	< 1001001>
49	a or n 3	Currency code, transaction	<840>

Luego, se envía la respuesta a la capa *Endpoint* y luego a la capa *Transport* donde la estructura se codifica en una respuesta JSON y es enviada al usuario o sistema que inició la petición.

4.5 Tests

Se realizaron tests unitarios de cada función de la aplicación asegurando su correcto funcionamiento aislado. Para esto se utilizan *mocks* - objetos que simulan ser un objeto específico - para simular una petición a la aplicación, así cómo también distintos *mocks*, con los valores esperados, para poder comparar los valores de retorno de las funciones testeadas.

Finalmente se testean de forma íntegra la conversión de una petición a un mensaje ISO 8583 y viceversa, corroborando que ambos caminos funcionen correctamente validando la funcionalidad de todos los componentes involucrados trabajando en conjunto.
Para realizar los test se utilizó el paquete “testing” (*Testing Package - Testing*, n.d.) propio de Go y un paquete externo llamado “testify” (*Testify Package - Github.com/stretchr/testify*, n.d.) que provee funciones útiles como *asserts* - función para evaluar valores - para comparar los valores de retorno esperados con los obtenidos de las funciones.

```
ok      api-iso8583-to-JSON/internal/iso8583    (cached)        coverage: 100.0% of statements
ok      api-iso8583-to-JSON/internal/service    (cached)        coverage: 100.0% of statements
```

Figura 6 - Cobertura de los tests

Se buscó el más alto porcentaje de cobertura de todas las funciones implicadas en la conversión, en la Figura 6 - Cobertura de los tests, se puede observar el resultado de la ejecución de los tests indicando 100% de cobertura en cada paquete que contiene los archivos donde se encuentran las funciones que participan en la conversión.

4.6 Decisiones tomadas

La primera versión del estándar ISO 8583 fue publicada en 1987, con el avance de las tecnologías de pago se debió hacer revisiones que llevaron a la publicación de nuevas versiones en 1993 y 2003. En este trabajo se decidió utilizar la versión del estándar de 1987 dejando un desarrollo funcional base para que en el futuro se pueda incorporar soporte a las siguientes versiones.
Por otro lado, el sitio web de la ISO sólo ofrece la versión 2003 a un precio aproximado de 200 dólares americanos, por lo que para realizar el trabajo hubo que recurrir a Wikipedia (*ISO 8583*, n.d.) y algunos otros sitios web que exponen información sobre el estándar.

Entre las decisiones técnicas se priorizó que el microservicio reciba peticiones HTTP en formato JSON.
Por otro lado, se eligió entre las opciones que el estándar sugiere, codificar el Bitmap del mensaje ISO 8583 a Hexadecimal representado en ASCII, ocupando así 16 bytes en el mensaje que corresponden a los 16 caracteres.
Los campos de largo fijo se auto completan con 0 a la izquierda para los tipos numéricos y con espacios en blanco a la izquierda para los alfanuméricos evitando así errores de longitud en los campos.

4.7 Casos de uso y entorno de prueba

La solución desarrollada está pensada para entornos bancarios donde se deba utilizar el estándar ISO8583 para enviar información de transacciones y se deba integrar esta funcionalidad con otro conjunto de microservicios.
La dificultad de simular un entorno bancario del que no hay mucha información ni opciones influyó notablemente en cómo se creó el entorno de prueba. Se debía exponer el microservicio de forma que pueda recibir una petición, por lo que se tuvo que crear un

formulario web que envíe la petición y un backend *mock* que reciba el mensaje ISO 8583 y de una respuesta.

Se desplegó el microservicio en Heroku -plataforma para desplegar aplicaciones de forma gratuita- para poder tener un entorno de prueba. Lamentablemente Heroku no acepta conexiones TCP, por lo tanto se tuvo que crear un componente *mock* interno que simula la lectura de una petición y retorna una respuesta, en gran parte, copiada de la petición. De esta forma se pueden enviar peticiones al microservicio y obtener respuestas del mismo como si estuviese funcionando en un entorno productivo, como se ve en la Figura 7 - Microservicio - Flujo petición cliente Mock, atravesando por toda la funcionalidad que el microservicio ofrece.

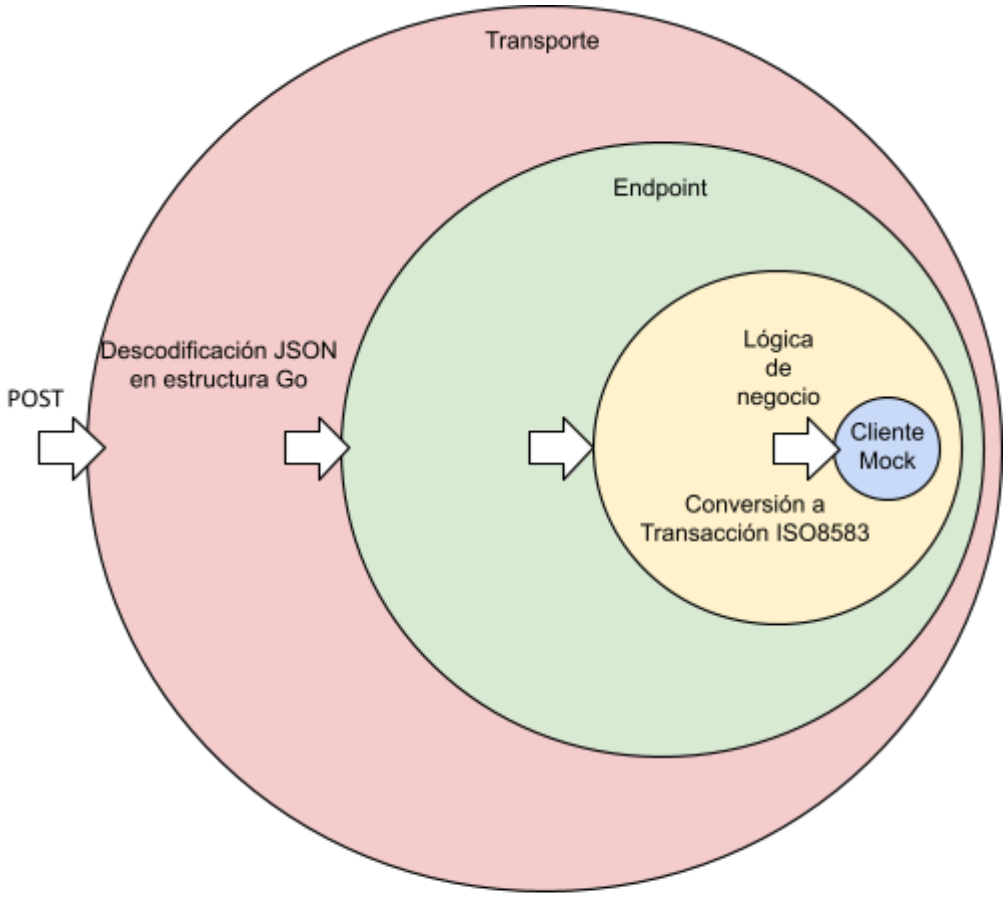
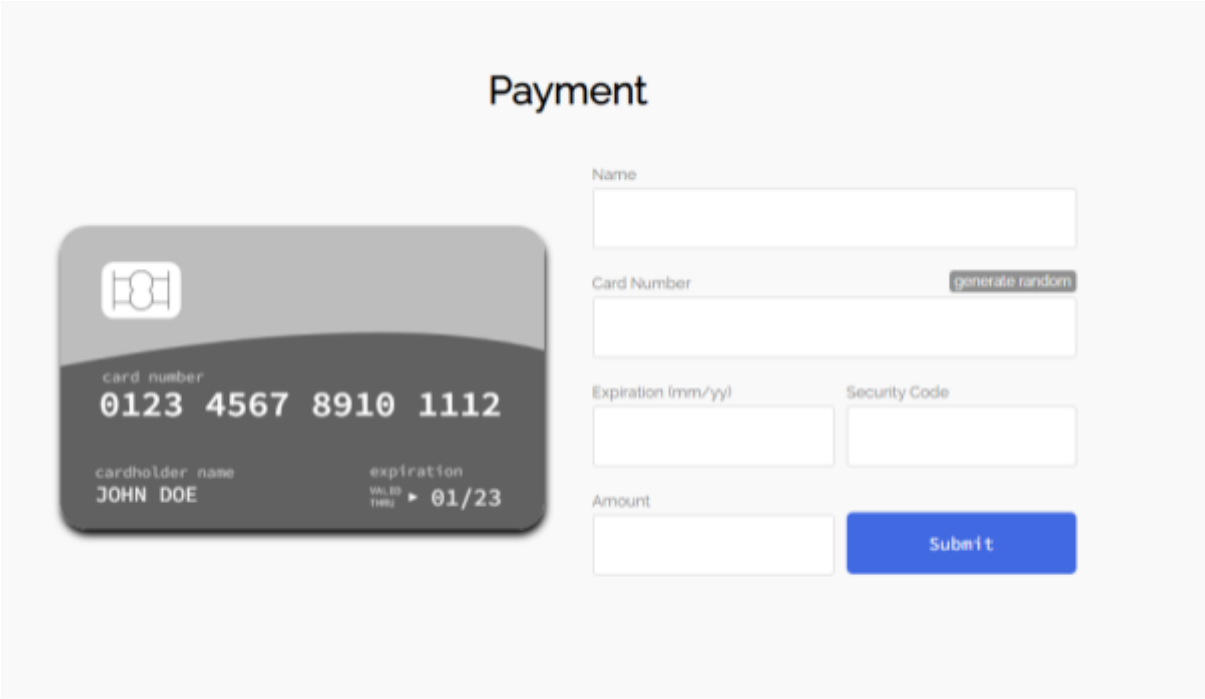


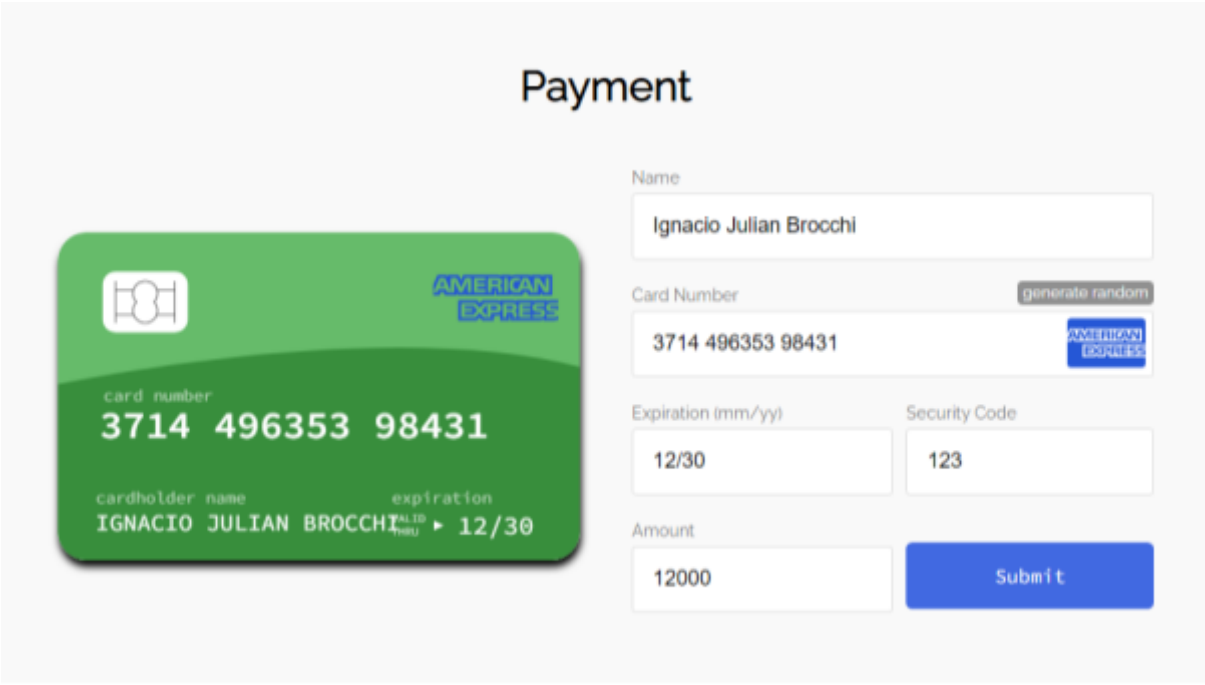
Figura 7 - Microservicio - Flujo petición cliente MOCK

Por otro lado, para poder explicar mejor cómo podría utilizarse el microservicio desarrollado, se creó la página web Front Payment https://front-payment.herokuapp.com/ que simula una pantalla de pago online, la cual solicita al usuario los datos de una tarjeta para realizar un pago.



Entorno de prueba - Figura 1 - Formulario

Se completan los datos del formulario [Entorno de prueba - Figura 1 - Formulario] indicando un Nombre en el campo *Name*, un número de tarjeta en *Card Number* , el cual puede ser generado con el botón gris *generate random*, una fecha de 4 números en *Expiration* , un número de 3 dígitos en *Security Code* y un monto en el campo *amount*.



Entorno de prueba - Figura 2 - Formulario completo

Al presionar el botón *Submit* los datos cargados en el formulario [Entorno de prueba - Figura 2 - Formulario completo] la información se envía al microservicio y este responde con la autorización de la operación. Recibida la respuesta del microservicio [Entorno de prueba - Figura 3 - Respuesta del Microservicio], ésta se muestra debajo del formulario. En primer lugar se encuentra la respuesta JSON del microservicio seguido también por la cadena generada por la petición, como también la generada por la respuesta antes de ser convertida a la respuesta del microservicio.

Entorno de prueba - Figura 3 - Respuesta del Microservicio

La página se creó con HTML y la funcionalidad de enviar el formulario al microservicio se desarrolló en Javascript y también se desplegó en Heroku para poder ser utilizada desde internet. Se descargó un template de una pantalla de pagos [https://codepen.io/quinlo/pen/YONMEa] y se le realizó las modificaciones necesarias para que obtenga la funcionalidad esperada. El formulario solicita un número de tarjeta de crédito, un código de seguridad, fecha de vencimiento de la tarjeta y un monto. La información ingresada en el formulario es procesada y utilizada para crear la petición en formato JSON respetando la estructura que el microservicio acepta.

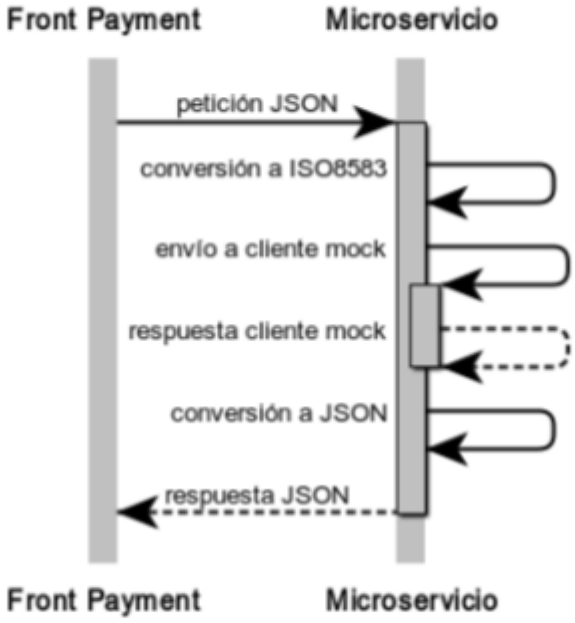


Figura 8 - Flujo del entorno de prueba

La petición se envía al microservicio desde la web Front Payment - Figura 8 - Flujo del entorno de prueba - y se espera por la respuesta, cuando ésta es obtenida, se visualiza debajo del formulario los mensajes ISO8583 que fueron generados por el microservicio para ser envíados al siguiente servicio, así como también el mensaje de respuesta que se obtiene en ISO 8583, visualizandose explícitamente las conversiones que se realizaron y la respuesta en formato JSON.

5. Conclusiones y trabajos futuros

En este trabajo se implementa un estándar utilizado en transacciones con tarjetas de crédito y débito como un microservicio encapsulado que pueda ser reutilizado por cualquier organización.
Debido al dominio en el cual se usa este tipo de estándares, el entorno bancario, no hay una forma pública de ver funcionando el desarrollo en un entorno productivo. Esto mismo también implica que no existe información pública de quiénes implementan el estándar en sus servicios ni de cómo lo usan.
Sin embargo, es clara la necesidad de contar con un servicio que implemente el estándar como de manera desacoplada, genérica, con un lenguaje moderno y de código abierto, habida cuenta la gran cantidad de plataformas digitales que operan con tarjetas de crédito y débito y la interoperabilidad requerida entre ellas a nivel global.
El desarrollo de este trabajo contribuye con la documentación detallada del estándar ISO 8583 y construye una solución de dicho estándar en el lenguaje Golang. El microservicio implementado se encuentra dockerizado para facilitar su despliegue.
En su primera versión, el desarrollo permite ser continuado agregando funcionalidades útiles fuera del alcance planteado inicialmente.
En primer lugar, se podría agregar soporte para las siguientes versiones del estándar ISO

8583. Actualmente existe una configuración base para la primera versión del estándar publicada en 1987, por lo que se podría agregar las configuraciones para las versiones de 1993 y 2003, permitiendo seleccionar cuál versión se desea utilizar. El código actual puede procesar mensajes de estas versiones sin necesidad de modificaciones adicionales en la funcionalidad de conversión, sólo es necesario agregar la configuración de los campos para cada versión.
Por otro lado, se puede agregar soporte a gRPC ya que el microservicio solo recibe peticiones HTTP. Go-kit en su paquete ofrece herramientas para gRPC que facilitan su desarrollo. Cómo se vio en la sección 3.1.1, gracias a la arquitectura en capas que propone Go-kit, sólo habría que agregar gRPC a la capa de transporte y que utilice el endpoint existente para la conversión.
En cuanto a la conversión que ocurre dentro del microservicio, podría agregarse la posibilidad de elegir de qué forma se quiere codificar el bitmap, ofreciendo EBCDIC (Extended Binary Coded Decimal Interchange Code) como opción adicional a Hexadecimal representado en ASCII la cual se utilizó en este trabajo. Esta codificación puede ser implementada también para los campos de datos del mensaje, el estándar no especifica cómo codificar puntualmente los datos de un mensaje por lo que podría ser una funcionalidad que se puede agregar en el futuro.
Otra mejora aplicable a futuro es poder elegir una configuración personalizada para cada campo del mensaje ISO8583, el cual si bien se define en el estándar, se podría utilizar un archivo de configuración con la información de cada campo indicando sus características, tipo, largo y tamaño fijo o variable, e incluso si se define como tamaño fijo, se podría indicar cómo autocompletar el campo si no se envió información suficiente. Esto aportaría una flexibilidad importante ya que las organizaciones no están obligadas por el estándar a respetar las especificaciones de los campos.

6. Repositorios remotos

Microservicio: https://github.com/nachobrocchi1/api-iso8583-to-JSON
Front Payment (entorno de prueba): https://github.com/nachobrocchi1/front-payment

Referencias

Alpine Linux. (n.d.). Alpine Linux. Retrieved April 9, 2022, from https://alpinelinux.org/about/

Apache Thrift. (n.d.). Apache Thrift. Retrieved April 9, 2022, from https://thrift.apache.org/

Build your Go image. (n.d.). Docker Documentation. Retrieved April 9, 2022, from https://docs.docker.com/language/golang/build-images/

Cloud Run: Container to production in seconds. (n.d.). Google Cloud. Retrieved April 9, 2022, from https://cloud.google.com/run?hl=es

Dockerfile reference. (n.d.). Docker Documentation. Retrieved May 5, 2022, from https://docs.docker.com/engine/reference/builder/

Docker Website. (n.d.). Docker: Home. Retrieved May 5, 2022, from https://www.docker.com/

Elasticsearch: El motor de búsqueda y analítica distribuido oficial. (n.d.). Elastic. Retrieved April 9, 2022, from https://www.elastic.co/es/elasticsearch/

Go - DEV Community. (n.d.). DEV Community . Retrieved April 9, 2022, from https://dev.to/t/go

Go for Web Development. (2019, October 4). The Go Programming Language. Retrieved April 9, 2022, from https://go.dev/solutions/webdev

Go kit - A toolkit for microservices. (n.d.). Go kit - A toolkit for microservices. Retrieved April 9, 2022, from https://gokit.io/

Golang - Official Image. (n.d.). Docker Hub. Retrieved April 9, 2022, from https://hub.docker.com/_/golang

Google App Engine. (n.d.). Google App Engine. Retrieved April 9, 2022, from https://appengine.google.com/

The Go Programming Language. (n.d.). The Go Programming Language. Retrieved April 9, 2022, from https://go.dev/

gRPC - Official Site. (n.d.). gRPC.io. Retrieved April 9, 2022, from https://grpc.io/

ISO 8583. (n.d.). Wikipedia. Retrieved September 13, 2022, from https://es.wikipedia.org/wiki/ISO_8583

ISO 8583:1993 - Financial transaction card originated messages — Interchange message specifications. (n.d.). ISO. Retrieved April 9, 2022, from https://www.iso.org/standard/15871.html

Java | Oracle. (n.d.). Java | Oracle. Retrieved September 13, 2022, from https://www.java.com/

JSON.org. (n.d.). JSON.org. Retrieved September 5, 2022, from https://www.json.org/json-en.html

Lewis, J., & Fowler, M. (2014, 3 25). *Microservices*. Martin Fowler. Retrieved May 1, 2022, from https://martinfowler.com/articles/microservices.html

Licklider - Free ISO8583 Parser. (n.d.). Licklider - Free ISO8583 Parser. Retrieved September 5, 2022, from https://licklider.cl/services/financial/iso8583parser/

Malik, R. (2019, October 4). *Go for Cloud & Network Services*. The Go Programming Language. Retrieved April 9, 2022, from https://go.dev/solutions/cloud

MongoDB: La Plataforma De Datos Para Aplicaciones. (n.d.). MongoDB: La Plataforma De Datos Para Aplicaciones | MongoDB. Retrieved April 9, 2022, from https://www.mongodb.com/es

moov-io/iso8583: A golang implementation to marshal and unmarshal iso8583 message. (n.d.). GitHub - moov-io/iso8583. Retrieved September 5, 2022, from https://github.com/moov-io/iso8583

MySQL. (n.d.). MySQL. Retrieved April 9, 2022, from https://www.mysql.com/

REST API. (2021, April 6). IBM. Retrieved September 11, 2022, from https://www.ibm.com/cloud/learn/rest-apis

RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2). (n.d.). RFC Editor. Retrieved April 9, 2022, from https://www.rfc-editor.org/rfc/rfc7540.html

RFC 793: Transmission Control Protocol. (n.d.). RFC Editor. Retrieved September 5, 2022, from https://www.rfc-editor.org/rfc/rfc793

rpc package - net/rpc. (n.d.). go.pkg.dev. Retrieved April 9, 2022, from https://pkg.go.dev/net/rpc

Simple Object Access Protocol. (n.d.). 1 Simple Object Access Protocol Overview. Retrieved September 11, 2022, from https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm

testify package - github.com/stretchr/testify. (n.d.). Go Packages. Retrieved September 12, 2022, from https://pkg.go.dev/github.com/stretchr/testify@v1.7.0

testing package - testing. (n.d.). Go Packages. Retrieved September 12, 2022, from https://pkg.go.dev/testing

TLS - Seguridad de la capa de transporte. (n.d.). Wikipedia. Retrieved April 9, 2022, from https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte

Varghese, S. (n.d.). *Go Microservices with Go kit: Introductio*. Go Microservices with Go kit: Introductio. Retrieved April 9, 2022, from https://shijuvar.medium.com/go-microservices-with-go-kit-introduction-43a757398183

What is a Container? (n.d.). Docker. Retrieved April 9, 2022, from https://www.docker.com/resources/what-container/

What is REST. (2022, April 7). What is REST - REST API Tutorial. Retrieved September 13, 2022, from https://restfulapi.net/

XML Essentials - W3C. (n.d.). World Wide Web Consortium (W3C). Retrieved September 11, 2022, from https://www.w3.org/standards/xml/core

ANEXO 1 - Composición de un mensaje ISO 8083

MTI

Es un campo numérico de 4 dígitos que clasifica la función del mensaje. Cada dígito tiene un significado y un dominio de valores.

Posición 1 - Versión de ISO 8583
La primera posición especifica la versión del estándar ISO 8583 que se utiliza en el mensaje. Por ejemplo, el valor 0200 se utiliza para indicar que la transacción que se envía es una solicitud de autorización financiera, la cual puede provenir de una terminal de pago o un e-commerce.

Valor	Versión
0xxx	ISO 8583 - 1987
1xxx	ISO 8583 - 1993
2xxx	ISO 8583 - 2003

Posición 2 - Clase de mensaje
La segunda posición del MTI indica el propósito del mensaje.

Valor	Clase	Uso
x1xx	Autorización	Determina la existencia de fondos disponibles, obtiene una aprobación pero no se toma en cuenta para la conciliación. Dual message system (DMS - el primer mensaje solicita aprobación y espera el segundo mensaje que solicita la liquidación)
x2xx	Financiero	Determina la existencia de fondos, obtiene aprobación e impacta directamente en la cuenta. Single message system.
x3xx	Manejo de archivos	Usado por hot-cards, TMS y otros intercambios.
x4xx	Reverso y mensajes de reintegro	x4x0 o x4x1 - Revierte una acción previamente autorizada x4x2 o x4x3 - Reintegra una transacción financiera previa
x5xx	Conciliación	Mensaje de información de liquidación.
x6xx	Administrativo	Transmite avisos administrativos. Usado para mensajes de

		error.
x7xx	Fee Collection	
x8xx	Manejo de Red	Usado para intercambio seguro de claves, echo test y otras funciones de red
x9xx	Reservado por el estándar	

Posición 3 - Función del mensaje

La tercera posición del MTI es la que representa la función del mensaje. Ésta define cómo se procesa dentro del sistema. Existen dos tipos de función de mensajes: Las peticiones (requests) son mensajes end-to-end, petición y respuesta, con timeout y rollbacks. El siguiente tipo son los mensajes de aviso (advises), mensajes point-to-point con transmisión garantizada.

Valor	**Función**	
xx0x	Petición	Solicitud del adquirente al emisor para realizar una acción; el emisor puede aceptar o rechazar
xx1x	Respuesta a la Petición	Respuesta del emisor
xx2x	Aviso	Aviso de que se ha llevado a cabo una acción; el receptor sólo puede aceptar, no rechazar
xx3x	Respuesta al Aviso	Respuesta a un aviso
xx4x	Notificación	Notificación de un evento; el receptor sólo puede aceptar, no rechazar
xx5x	Respuesta a la Notificación	Respuesta a la notificación
xx8x	Reservado por el estándar	
xx9x	Reservado por el estándar	

Posición 4 - Origen del mensaje

La posición 4 del MTI indica el origen del mensaje dentro de la cadena de pago.

Valor	**Significado**
xxx0	Adquirente
xxx1	Adquirente Repetición

xxx2	Emisor
xxx3	Emisor Repetición
xxx4	Otros
xxx5	Otros Repetición

Una vez comprendida cada posición, un MTI especifica qué debe hacer un mensaje y cómo se transmite a través de la red. Si bien las implementaciones del ISO 8583 interpretan de distinta manera el MTI, algunos usos comunes son:

MTI	**Significado**	**Uso**
0100	Petición de Autorización	Petición de autorización desde una terminal para una operación con tarjeta
0110	Respuesta de Autorización	Petición de respuesta de autorización a una terminal para una operación con tarjeta
0120	Aviso de Autorización	
0121	Repetición de Aviso de Autorización	
0130	Respuesta al Aviso de Autorización	Confirma la recepción del aviso de autorización
0200	Petición financiera del adquirente	Petición de fondos, desde una terminal (punto de venta) o cajero automático (ATM)
0210	Respuesta a la petición financiera	El emisor responde a la petición de fondos
0220	Aviso financiero del adquirente	Usado para completar una transacción iniciada con una petición de autorización
0221	Repetición del Aviso financiero del adquirente	Si se cumple el tiempo de espera de la respuesta al aviso
0230	Respuesta al Aviso financiero del adquirente	Confirma la recepción del aviso financiero
0320	Actualización de archivo	Aviso de actualización o transferencia de archivo
0330	Respuesta a la Actualización de archivo	Respuesta al Aviso de actualización o transferencia de archivo
0400	Petición de reversión	Revertir una transacción
0410	Respuesta de reversión	Transacción revertida

0420	Aviso de reversión	
0430	Respuesta al aviso de reversión	Respuesta a la reversión de una transacción

Bitmaps

Dentro de un mensaje ISO8583, luego del campo MTI se encuentran los Bitmaps que indican qué campos de la sección de datos se utilizaron en el mensaje. En otras palabras, un campo se considera presente solo cuando su correspondiente bit en el bitmap está activo. Por ejemplo, 1000 0010 significa que los campos 1 y 7 están presentes en el mensaje mientras que 2,3,4,5,6 y 8 no.
Un bitmap puede ser representado como 8 bytes de datos en binario o como 16 caracteres en hexadecimal representado en ASCII o EBCDIC (Extended Binary Coded Decimal Interchange Code, codificación de 8 bits utilizada en mainframes IBM). Un mensaje contiene al menos un bitmap, llamado *primary bitmap*, el cual indica cuáles de los campos entre 1 y 64 están presentes. La presencia de un segundo bitmap está indicada por el primer bit del primer bitmap. Si está presente, el segundo bitmap indica cuáles campos entre 65 y 128 están presentes. La versión de 1987 define hasta 128 campos, mientras que las versiones posteriores permiten un tercer bitmap incrementando la cantidad de campos a 192.

Dado el bitmap 7224448028C08000

Número de bit	0	10	20	30	40	50	60
	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234
Bitmap	0111001000	1001000100	0100100000	0000101000	1100000010	0000000000	0000

Los campos presentes serían: 2 - 3 - 4 - 7 - 11 - 14 - 18 - 22 - 25 - 35 - 37 - 41 - 42 - 49

Datos

Son los campos individuales donde viaja la información de la transacción. Hay 128 campos de datos especificados en el ISO8583 de 1987 y hasta 192 en las versiones posteriores.
Cada campo de datos tiene un significado y formato específico, también el estándar incluye algunos campos de propósito general, o específicos de un país, que varían en cada implementación.
Cada campo es descrito en un formato estándar que define el contenido permitido del campo y el largo del mismo.

Tipos de datos de campo	
Abreviación	**Contenido del campo**
a	Alfanumérico con blancos
n	Sólo valores numéricos

x+n	Valores de Monto. El primer byte puede ser ‘C’ que indica positivo o ‘Credit’ y ‘D’ indica negativo o ‘Debit’, seguido por el monto numérico.
s	Caracteres especiales
an	Alfanumérico
as	Alfanumérico y caracteres especiales
ns	Numérico y caracteres especiales
ans	Alfabético, numérico y caracteres especiales
b	Binario
z	Tracks 2 y 3 especificados en ISO/IEC 7813 e ISO/IEC 4909 respectivamente
. .. ...	Indicadores de largo de campo, cada punto indica un dígito
x xx xxx	Campos de largo fijo, o largo máximo en caso de campos de largo variable. Donde ‘x' es un valor numérico.

Tipos de largo de campo	
Abreviación	**Tipo**
Fixed	Largo fijo
LVAR	Un dígito para el largo, donde 0 < L < 10
LLVAR	Dos dígitos para el largo, donde 0 < L < 100
LLLVAR	Tres dígitos para el largo, donde 0 < L < 1000

Un campo de largo variable puede ser comprimido o ASCII dependiendo del formato del mensaje. Si el tipo de dato de un campo es numérico, estará comprimido, si por ejemplo el largo es 87 se representará por un byte ‘87x’, si es ASCII serán dos bytes ‘38x’ y ‘37x’

Campo	**Tipo**	**Uso**
1	b 64	Bitmap
2	n ..19	PAN - Primary account Number
3	n 6	Processing Code
4	n 12	Amount Transaction
5	n 12	Settlement Amount

Campo	Tipo	Uso
6	n 12	Cardholder Billing Amount
7	n 10	Transmission date & time
8	n 8	Cardholder billing fee
9	n 8	Conversion rate, settlement
10	n8	Conversion rate, cardholder billing
11	n 6	System Trace Audit Number (STAN)
12	n 6	Local Transaction Time (hhmmss)
13	n 4	Local Transaction Date (MMdd)
14	n 4	Expiration Date
15	n 4	Settlement date
16	n 4	Currency Conversion Date
17	n 4	Capture Date
18	n 4	Merchant Type
19	n 3	Acquiring institution (Country Code)
20	n 3	PAN extended (Country Code)
21	n 3	Forwarding institution (country code)
22	n 3	Point of service entry mode
23	n 3	Application PAN sequence number
24	n 3	Function code (ISO 8583:1993), or network international identifier (NII)
25	n 2	Point of service condition code
26	n 2	Point of service capture code
27	n 1	Authorizing identification response length
28	x+n 8	Amount, transaction fee
29	x+n 8	Amount, settlement fee
30	x+n 8	Amount, transaction processing fee

Campo	Tipo	Uso
31	x+n 8	Amount, settlement processing fee
32	n ..11	Acquiring institution identification code
33	n ..11	Forwarding institution identification code
34	ns ..28	Primary account number, extended
35	z ..37	Track 2 data
36	n ...104	Track 3 data
37	an 12	Retrieval reference number
38	an 6	Authorization identification response
39	an 2	Response code
40	an 3	Service restriction code
41	ans 8	Card acceptor terminal identification
42	ans 15	Card acceptor identification code
43	ans 40	Card acceptor name/location (1–23 street address, –36 city, –38 state, 39–40 country)
44	an ..25	Additional response data
45	an ..76	Track 1 data
46	an ...999	Additional data (ISO)
47	an ...999	Additional data (national)
48	an ...999	Additional data (private)
49	a or n 3	Currency code, transaction
50	a or n 3	Currency code, settlement
51	a or n 3	Currency code, cardholder billing
52	b 64	Personal identification number data
53	n 16	Security related control information
54	an ...120	Additional amounts
55	ans ...999	ICC data – EMV having multiple tags

Campo	Tipo	Uso
56	ans ...999	Reserved (ISO)
57	ans ...999	Reserved (national)
58	ans ...999	
59	ans ...999	
60	ans ...999	Reserved (national) (e.g. settlement request: batch number, advice transactions: original transaction amount, batch upload: original MTI plus original RRN plus original STAN, etc.)
61	ans ...999	Reserved (private) (e.g. CVV2/service code transactions)
62	ans ...999	Reserved (private) (e.g. transactions: invoice number, key exchange transactions: TPK key, etc.)
63	ans ...999	Reserved (private)
64	b 64	Message authentication code (MAC)
65	b 1	Extended bitmap indicator
66	n 1	Settlement code
67	n 2	Extended payment code
68	n 3	Receiving institution country code
69	n 3	Settlement institution country code
70	n 3	Network management information code
71	n 4	Message number
72	n 4	Last message's number
73	n 6	Action date (YYMMDD)
74	n 10	Number of credits
75	n 10	Credits, reversal number
76	n 10	Number of debits
77	n 10	Debits, reversal number
78	n 10	Transfer number
79	n 10	Transfer, reversal number

Campo	Tipo	Uso
80	n 10	Number of inquiries
81	n 10	Number of authorizations
82	n 12	Credits, processing fee amount
83	n 12	Credits, transaction fee amount
84	n 12	Debits, processing fee amount
85	n 12	Debits, transaction fee amount
86	n 16	Total amount of credits
87	n 16	Credits, reversal amount
88	n 16	Total amount of debits
89	n 16	Debits, reversal amount
90	n 42	Original data elements
91	an 1	File update code
92	an 2	File security code
93	an 5	Response indicator
94	an 7	Service indicator
95	an 42	Replacement amounts
96	b 64	Message security code
97	x+n 16	Net settlement amount
98	ans 25	Payee
99	n ..11	Settlement institution identification code
100	n ..11	Receiving institution identification code
101	ans ..17	File name
102	ans ..28	Account identification 1
103	ans ..28	Account identification 2
104	ans ...100	Transaction description

Campo	Tipo	Uso
105	ans ...999	Reserved for ISO use
106	ans ...999	
107	ans ...999	
108	ans ...999	
109	ans ...999	
110	ans ...999	
111	ans ...999	
112	ans ...999	Reserved for national use
113	ans ...999	
114	ans ...999	
115	ans ...999	
116	ans ...999	
117	ans ...999	
118	ans ...999	
119	ans ...999	
120	ans ...999	Reserved for private use
121	ans ...999	
122	ans ...999	
123	ans ...999	
124	ans ...999	
125	ans ...999	
126	ans ...999	
127	ans ...999	
128	b 64	Message authentication code