Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS

Julio César Osorio Boror
Luis Fernando Morales Mejicanos
Asesorado por el Ing. José Alfredo González Díaz

Guatemala, abril de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JULIO CESAR OSORIO BOROR
LUIS FERNANDO MORALES MEJICANOS
ASESORADO POR EL ING. JOSÉ ALFREDO GONZÁLEZ DÍAZ

AL CONFERÍRSELES EL TÍTULO DE

INGENIEROS EN CIENCIAS Y SISTEMAS

GUATEMALA, ABRIL DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. William Estuardo Escobar Argueta
EXAMINADOR	Ing. William Samuel Guevara Orellana
SECRETARIA	Inga. Lesbia Magalí Herrera López

Julio César Osorio Boror

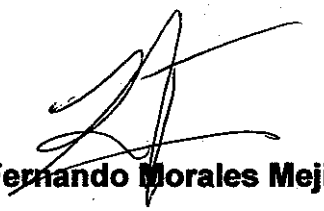
HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presentamos a su consideración nuestro trabajo de graduación titulado:

DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS

Tema que nos fuera asignado por la dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha de noviembre de 2015.

Julio César Osorio Boror



Luis Fernando Morales Mejicanos

Guatemala, 29 de Febrero de 2016

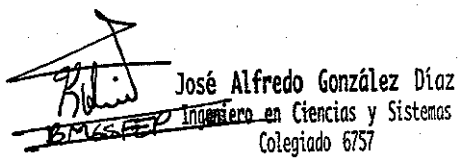
Ingeniero
Marlon Pérez Turk
Director
Escuela de Ciencias y Sistemas
Facultad de Ingeniería

Respetable Ingeniero Pérez

Por este medio hago de su conocimiento que he revisado el trabajo de graduación de los estudiantes **Julio César Osorio Boror**, identificado con el número de carné **201020177** y **Luis Fernando Morales Mejicanos** con el número de carné **201020573**, titulado: **"DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICRO SERVICIOS"**, y a mi criterio el mismo cumple con los objetivos propuestos para su elaboración de acuerdo al protocolo presentado.

Sin otro particular, me suscribo de usted,

Atentamente,



José Alfredo González Díaz
Ingeniero en Ciencias y Sistemas

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 9 de Marzo del 2016

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación de los estudiantes **JULIO CÉSAR OSORIO BOROR** con carné **201020177**, y **LUIS FERNANDO MORALES MEJICANOS** con carné **201020573**, titulado: **"DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS"****, realizado por los estudiantes JULIO CÉSAR OSORIO BOROR y LUIS FERNANDO MORALES MEJICANOS, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Turk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 18 de abril de 2016

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref.DTG.D.180.2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **DESARROLLO DE UNA API PARA EL ANÁLISIS DE INVERSIONES Y CRÉDITOS A TRAVÉS DE FACTORES DE INTERÉS COMPUESTO, IMPLEMENTANDO MICROSERVICIOS**, presentado por los estudiantes universitarios: **Julio César Osorio Boror y Luis Fernando Morales Mejicanos**, y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERÍA

Guatemala, abril de 2016

/cc

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
SECRETARIA	Inga. Lesbia Magalí Herrera López

Luis Fernando Morales Mejicanos

ACTO QUE DEDICO A:

Dios	Por ser el sostén de mi vida.
Mi madre	Sandra Boror, por todo el amor que me has brindado, por tu apoyo incondicional y por tus sabios consejos que llenan mi alma.
Mi padre	Julio Osorio (q. e. p .d), por los hermosos recuerdos que nunca olvidaré.
Mi hermana	Jenniffer Osorio, por tu apoyo y comprensión en cada momento de mi vida.
Mis abuelos	Isabel García y Rafael Boror, por todo su amor y sus cuidados que me ayudaron a alcanzar mis metas.
Mi sobrino	Evan Andrés Osorio, por permitirme presenciar el milagro de la vida y ser para ti un ejemplo a seguir.
Mi novia	Argentina Flores, por todo tu amor y apoyo, por cambiar mi vida y por todos los hermosos momentos que hemos vivido.

Mis tíos Maynor Boror y Verónica Morales, por darme su aliento para seguir adelante.

Mis amigos Ángel Cajas, José Avalos, Eduardo Herrera y Luis Morales, por su apoyo a lo largo de todo este trayecto.

Julio César Osorio Boror

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Casa de estudios que me ha ayudado en mi formación, no solo profesional, también personal, y por ser mi segundo hogar.
Mi compañero de trabajo de graduación	Luis Fernando Morales Mejicanos, por haber trabajado en conjunto en este proyecto de graduación.
Mi familia	Por todo su amor y su apoyo que me dio todas las fuerzas necesarias para salir adelante.
Mis amigos y compañeros de la Facultad de Ingeniería	Por todos los momentos compartidos, el apoyo recibido y las nuevas experiencias que enriquecieron el camino para culminar esta meta.

Julio César Osorio Boror

ACTO QUE DEDICO A:

Dios Porque es quien me da el don de la vida, me inspira a seguir adelante y me permite crecer intelectual y emocionalmente.

Mis padres Irma Leticia Mejicanos y Carlos Humberto Morales, quienes me han dado la vida y me han acompañado en cada etapa de la misma, brindándome el apoyo necesario para cada decisión que he tomado.

Mi hermana Alejandra Morales, con quien he compartido mi vida desde que tengo memoria y, con ello, todos los momentos significativos de la misma.

Mis abuelos José María Faraón Mejicanos (q. e. p. d.) y Martha Leonor Jol, quienes siempre han estado pendientes de mí, dándome su amor incondicional.

Mis tíos Erik Rolando, Jorge René, Mayra Elizabeth, Juan Tomás, Dilma Yanet y Claudia Mariné Mejicanos, por su cariño y constante apoyo en mi vida.

Luis Fernando Morales Mejicanos

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Casa de estudios que me dio la oportunidad de estudiar en sus aulas, además de formarme intelectual y académicamente, moldeando en mí un pensamiento crítico y analítico, enfocado al servicio de la sociedad guatemalteca.
Facultad de Ingeniería	Por brindarme los conocimientos técnicos y científicos necesarios para cumplir con esta meta.
Mi asesor, mis tutores y catedráticos	Por compartir con nosotros sus conocimientos y por dedicar su tiempo a la noble vocación de la enseñanza.
Mi compañero de trabajo de graduación	Julio César Osorio, por la labor realizada en conjunto para convertir en una realidad el proyecto que se describe en nuestro trabajo.
Mis madrinas de graduación	Mi madre, Irma Leticia y mi tía, Dilma Yanet Mejicanos, por su asesoría y grandes consejos profesionales y de la vida misma.

Mi familia	A quienes también he dedicado este acto, a mis padres, mi hermana, mis abuelos y mis tíos, por ser el núcleo social que me ha permitido forjar ideas y desarrollarme como ser humano.
Mi novia	Narda Pacay, por darme aliento en los momentos más difíciles y críticos y, sobre todo, por su amor y cariño.
Mis amigos de la Escuela de Ciencias y Sistemas	Katerin Amaya, Jorge Rubio, Otto Anaya, Kevin Godínez, Kevin Cardona, Andrés Cardona, José Ávalos, Eduardo Herrera, Ángel Cajas, Daniel Godínez, Diego Solis, por brindarme su amistad, su apoyo y por las alegrías y momentos compartidos durante nuestros días de estudiantes universitarios.
Mis amigos de la Facultad	Diego Seisdedos, Giullian Locón, Eduardo Barrios, Edgar Guzmán, por brindarme su amistad durante nuestra vida universitaria, a pesar de no compartir la misma carrera.
Mis amigos de toda la vida	Markko Barahona, Rolando Méndez y Luis Vela, por su invaluable amistad durante casi toda la vida.

Luis Fernando Morales Mejicanos

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

API — *Application programming interface*, conjunto de funciones, subrutinas, protocolos y herramientas utilizadas para la construcción de productos de software.

Aplicación — Se refiere a un programa o producto de software que brinda utilidad al usuario final, sea entretenimiento o productividad.

Archivo binario — Archivo que contiene información codificada en lenguaje máquina, generalmente código binario, el cual es comprendido únicamente por una computadora con una arquitectura específica.

Base de datos — Repositorio centralizado de información que contiene datos relevantes sobre diversas temáticas de manera persistente.

Biblioteca — Conjunto de utilidades, funciones, herramientas y subrutinas compiladas en forma de archivo binario o código fuente, que extienden la funcionalidad de los programas que las incluyen.

Branch	División lógica de un repositorio distribuido que contiene una copia del historial del mismo y en la que se realizan cambios sin afectar el producto final.
Chacrka	Motor JavaScript presente en los navegadores de sitios web desarrollados por Microsoft Corporation para el sistema operativo Windows.
Commit	Operación de confirmación de cambios realizados en un repositorio almacenado en un sistema de control de versiones.
Cron	Administrador de procesos en segundo plano, que ejecuta procesos o secuencias de estos a intervalos programables.
Función de primera clase	En ciencias de la computación, se trata a una función como de primera clase cuando esta puede ser pasada como parámetro de otra función.
Google Chrome	Navegador multiplataforma de sitios web desarrollado por Google Inc.
GPL	*General public license*, licencia de software libre creada por la Free Software Foundation. Especifica que el software con esa licencia por ningún motivo viola ninguna de las libertades del software libre y de su filosofía.

HTTP	*Hypertext transfer protocol* protocolo de aplicación del modelo OSI de redes que se encarga de la transferencia de hipertexto. Es el más utilizado en internet.
Internet	Red de computadoras de área global, creada a partir de redes de menor tamaño, cuyo origen reside en la cooperación de dos universidades estadounidenses. Actualmente es la red más grande del mundo.
JSON	*JavaScript object notation*, formato ligero que permite la representación de objetos y datos con la notación de objetos del lenguaje JavaScript.
Mipymes	Acrónimo relacionado al sector de las micro, pequeñas y medianas empresas.
Multiparadigma	Se dice que un lenguaje de programación es multiparadigma cuando este soporta varios paradigmas de programación conocidos.
NO-SQL	*Not only SQL*, es un paradigma en el que bases de datos que difieren de la utilización de SQL como lenguaje de consulta son catalogadas.
Objeto	Ejemplificación general de datos de una clase, que posee características propias y métodos para manipular su comportamiento.

Plataforma Término usado normalmente para referirse a una arquitectura de hardware específica. El término también es utilizado para sistemas operativos o para el conjunto de ambos.

Plugin Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Polling Operación llevada a cabo por un cliente a un servidor de manera constante para verificar el estado de algún recurso.

Programa Conjunto de instrucciones ordenadas lógicamente que permiten realizar una tarea o trabajo específico dentro de un ordenador o computadora.

Pull request Acción llevada a cabo en un repositorio remoto para la revisión de cambios previos antes de un *merge*.

RDBMS Sistema de gestión de bases de datos relacionales, son sistemas de software encargados administrar, operar físicamente y abstraer lógicamente el contenido de bases de datos que siguen el modelo relacional.

Release Distribución de la versión final en una etapa del ciclo de vida de un software.

Repositorio	Sitio centralizado donde se mantiene almacenada, de forma ordenada, información digital, como bases de datos o archivos.
REST	*Representational state transfer* o transferencia de estado representacional, arquitectura de software basada en la transferencia y comunicación por medio de hypermedia.
Rinho	Motor JavaScript presente en el navegador web multiplataforma de Mozilla, Firefox.
SOA	*Software oriented architecture*, arquitectura basada en la separación de procesos de negocio en forma de servicios independientes.
SOAP	*Simple object access protocol,* provee los mecanismos para la ejecución de llamadas a procedimientos remotos entre programas, y establece eficientemente las comunicaciones de tipo programa a programa.
Software	Término genérico que designa al conjunto de programas de distinto tipo, sean sistemas operativoso aplicaciones diversas, que hacen posible operar un equipo de cómputo.

Spidermonkey	Motor JavaScript escrito en el lenguaje de programación Java, presente en el navegador web Firefox.
TIR	Tasa interna de retorno o tasa interna de rendimiento.
TMAR	Tasa mínima atractiva de retorno.
V8	Motor JavaScript que ejecuta código de dicho lenguaje del lado servidor en Node.js y de lado cliente en el navegador Google Chrome.
VPN	Valor presente neto, también conocido como valor actual neto.
Web service	Es un método de interconexión entre dos o más sistemas remotos, proveído generalmente a través de internet.
XML	*Extensible markup language,* metalenguaje de etiquetado, simple y estricto, adecuado a usos determinados. En la práctica, corresponde a un estándar que permite a diferentes aplicaciones interactuar con facilidad a través de la red.

RESUMEN

El desarrollo de la API planteada se basó en la necesidad de la realización de un sistema automatizado para el análisis financiero de proyectos, enfocado especialmente para el acoplamiento de dicha actividad a los sistemas informáticos de instituciones que brindan apoyo a nuevas empresas.

Implementando el patrón de diseño de arquitectura de microservicios, la API fue segmentada en varios servicios atómicos, aislados e independientes que proveen de una funcionalidad específica al sistema. El conjunto de estos microservicios conforman un servicio transparente para los clientes del mismo, gracias a la implementación de un servicio Gateway que se encarga de exponer las rutas necesarias.

La API provee a los clientes la capacidad de crear usuarios, que serán dueños de cada proyecto de inversión que deseen evaluar, dentro de los cuales es posible crear operaciones y realizar evaluaciones de factibilidad.

Cada solicitud de evaluación realizada en un proyecto implica que la API realizará una petición asíncrona para el cálculo del valor presente neto y del análisis de beneficio/costo, cálculos llevados a cabo a partir de ecuaciones y factores de interés compuesto propuestos dentro de la ingeniería económica.

Los microservicios que conforman la composición de la API son entregados como contenedores de Docker, con lo que se garantiza la ejecución de los mismos en un ambiente aislado y controlado.

OBJETIVOS

General

Desarrollar un servicio de análisis financiero para la evaluación de viabilidad económica de proyectos que requieran inversión o crédito implementado en la nube.

Específicos

1. Desarrollar una interfaz de programación de aplicaciones (API) orientada a la realización de análisis financieros.

2. Implementar modelos matemáticos para el cálculo de factores de interés compuesto para el análisis de viabilidad financiera.

3. Aislar las funcionalidades principales del servicio proveído por la API, utilizando el patrón de diseño de arquitectura de microservicios.

4. Proveer una API que facilite la creación de herramientas para análisis financieros precisos y actualizados.

5. Realizar una herramienta tecnológica para el análisis financiero con indicadores y variables acordes a la realidad económica de Guatemala.

INTRODUCCIÓN

En Guatemala, la creación de micro, pequeñas y medianas empresas crece considerablemente cada año, sin embargo, la sobrevivencia de la mayoría de ellas tiende a peligrar en los primeros años debido a múltiples factores, siendo uno de los más impactantes la falta de previsión financiera y la mala toma de decisiones en cuanto a inversiones riesgosas y créditos poco accesibles. Existen instituciones, tanto estatales como privadas, que dedican recursos al apoyo de nuevos proyectos de emprendimiento, pero carecen de asesoría en cuanto a análisis financiero se refiere, imposibilitando la prevención de fracasos económicos.

Una de las maneras de realizar análisis financieros futuros es la utilización de la ingeniería económica, que provee de métodos matemáticos para el cálculo de variables que indican el valor de inversiones, créditos y ganancias estimadas futuras, lo cual es altamente utilizado para la realización de análisis financieros. Para esto se utilizan variables como el valor presente neto y el análisis de beneficio/costo.

A través de la API remota desarrollada, toda institución de atención a nuevos proyectos de emprendimiento podrá incluir la realización de análisis financieros a sus sistemas, ya que provee la lógica necesaria para alcanzar dicho objetivo. La API está desarrollada utilizando el patrón de microservicios, el cual separa y aísla las funciones principales del sistema en servicios atómicos, independientes y agnósticos, lo que le da mantenibilidad, escalabilidad y mejor rendimiento con el pasar del tiempo.

1. CONCEPTOS DE LA INGENIERÍA ECONÓMICA

La ingeniería económica es el conjunto de técnicas matemático-financieras utilizadas para la evaluación de alternativas de inversión. Conlleva el soporte de toma de decisiones de manera financiera para propuestas de soluciones de ingeniería. Surge de la necesidad de controlar los bienes financieros después del crecimiento industrial que explota a partir de los años 50.

Su principal campo de estudio y mejoramiento es el control del valor del dinero, el cual es variable a través del tiempo. La utilización correcta de las técnicas planteadas por la ingeniería económica permitirá la obtención de beneficios en inversiones, evitar quiebras prematuras, estimar costos de operación, entre otros, utilizando el continuo cambio de valor del dinero.

Entre los indicadores estudiados y herramientas utilizadas por la ingeniería económica es posible encontrar tasas de interés, inflación, flujos de efectivo, cálculo de factores de equivalencia, además de métodos de análisis financieros como el valor presente neto, la tasa interna de retorno y el análisis de beneficio/costo.

1.1. Inflación

Es el fenómeno económico que describe la pérdida de valor adquisitivo del dinero a través del tiempo, situación dada por el aumento generalizado en los precios para la adquisición de bienes y servicios en el mercado.

El cálculo de la inflación es obtenido a partir de la variación anual en el índice de precios del consumidor (IPC) de un país, el cual engloba los precios de un grupo de productos como la canasta básica.

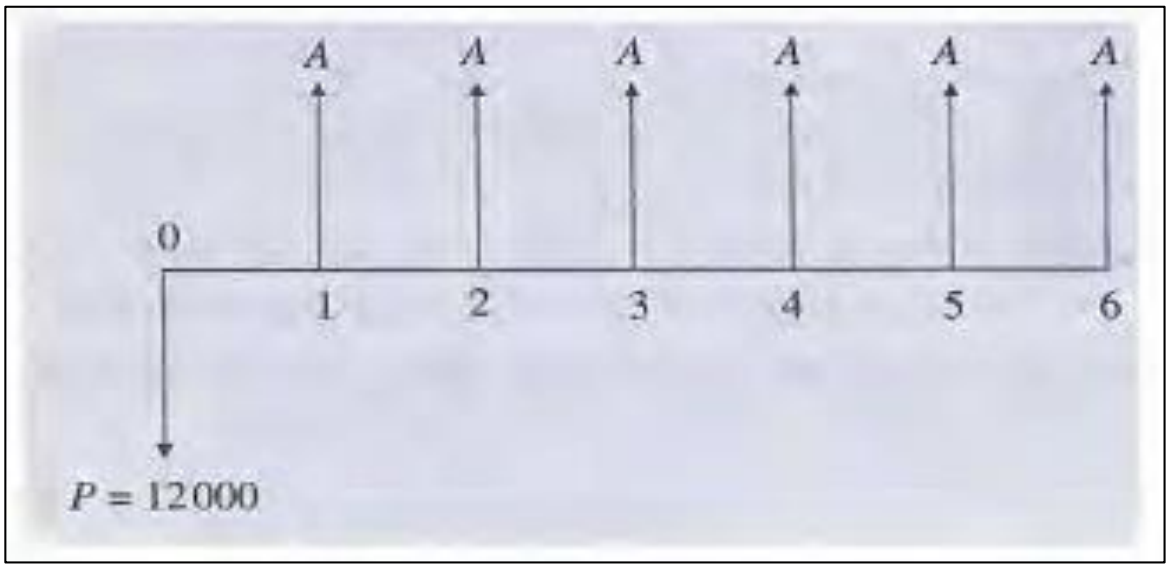
La inflación es una de las variables controladas por el Banguat en Guatemala, la cual es posible consultar de forma abierta en sus sistemas de información.

1.2. Flujo de efectivo

Es la representación de entradas y salidas de bienes financieros y monetarios a través del tiempo. Los flujos de efectivo se construyen a partir del punto de vista del inversionista, ya que son de apoyo gráfico para él.

Representa los movimientos por medio de flechas, cuya dirección depende directamente del tipo de operación que se realice, sea ingreso o egreso monetario. Si un ingreso se registra, este tendrá automáticamente un signo positivo, mientras que si se realiza un egreso, este será negativo.

Figura 1. **Ejemplo de flujo de efectivo**



Fuente: BARCA, Gabriel. *Fundamentos de ingeniería económica*. p. 13

1.3. Interés

Es un índice que mide el rendimiento y la rentabilidad de ahorros e inversiones, el cual es altamente utilizado para medir el costo de crédito. En concepto, es lo que se paga por el uso del dinero ajeno; es la ganancia de valor del dinero a través del tiempo.

1.3.1. Tasa de interés

Es el porcentaje utilizado para el cálculo de los intereses a cobrar o pagar en una operación monetaria a través del tiempo.

En la banca y el mercado financiero existen dos tipos de tasa de interés. Ambos tipos de tasas de interés pueden ser expresados de dos maneras.

1.3.1.1. Tasa de interés pasiva

Es el porcentaje pagado por instituciones bancarias a los depositarios del capital. Es conocida por ser una tasa de interés de riesgo, generalmente utilizada para manipular productos financieros de ahorro y de plazo fijo.

El Banco de Guatemala proporciona el valor de la tasa de interés pasiva como una variable económica utilizada como referencia por las instituciones bancarias, conocida como tasa de interés líder de política monetaria.

1.3.1.2. Tasa de interés activa

Es el porcentaje cobrado por instituciones bancarias y crediticias, dependiente de la tasa de interés pasiva (o tasa de interés líder) y de la inflación de la moneda del país.

Para las entidades que prestan servicios crediticios y financieros, esta tasa de interés es conocida como TMAR, y generalmente es posible calcularla a partir de la siguiente ecuación:

$$TMAR = i + f + fi \qquad [Ec. 1]$$

Donde

i: tasa de interés pasiva o líder de política monetaria
f: inflación

El factor fi se interpreta como el interés operado a la inflación monetaria.

1.3.1.3. Tasa de interés nominal

Es la tasa de interés aparente en la que se expresa únicamente el período durante el que se aplica el interés. Generalmente, es utilizada por entidades bancarias para dar a conocer a grandes rasgos la tasa de interés con la que trabajan, tanto pasiva como activa.

1.3.1.4. Tasa de interés efectiva

Es la tasa de interés periódica (generalmente anual) real, la cual se hace efectiva en períodos más pequeños, dependiendo del período de capitalización o pago de intereses que se posea.

En el caso que el período de pago y el de capitalización de intereses sea el mismo, la tasa de interés nominal es la tasa de interés efectiva.

La conversión de tasa nominal a real o efectiva, está dada por la siguiente ecuación:

$$i_{efectiva} = \left[\left(1 + \frac{i}{n}\right)^{n} - 1\right] \qquad [Ec. 2]$$

Donde

$i_{efectiva}$: tasa de interés real
i: tasa interés nominal anual
n: período menor a un año sobre el que se paga el interés

En el caso en el que el período de pago sea menor al período de capitalización, la tasa efectiva a aplicar es la siguiente:

$$i_{efectiva} = \frac{i}{n} \qquad [Ec. 3]$$

Donde

$i_{efectiva}$: tasa de interés efectiva a obtener
i: tasa de interés nominal
n: cantidad de períodos en los que se dividirá la tasa

1.3.2. Interés simple

Es el tipo de interés calculado a partir de una cantidad monetaria fija, conocida como capital inicial. El cálculo no incluye los intereses pagados o a pagar en un rango específico de tiempo.

Es posible calcularlo de la siguiente manera:

$$F = P_o + P_0 ni \qquad [Ec. 4]$$

Donde

F: valor futuro calculado a partir del interés simple
P_0: capital inicial presente
n: período sobre el que se calculará el interés
i: es la tasa de interés a aplicar

1.3.3. Interés compuesto

Es el interés calculado a partir de cantidades variables a través del tiempo, las cuales son calculadas a través del interés generado sobre los intereses calculados previamente sobre un capital inicial.

El cálculo de interés compuesto proporciona un valor futuro más elevado para una determinada cantidad de capital inicial que el interés simple, esto debido a que calcula el precio de la utilización del dinero y de los intereses ganados por el mismo a través del tiempo.

El interés compuesto está dado por:

$$F = P_o(1 + i)^n \qquad [Ec. 5]$$

Donde

F: valor futuro del capital inicial aplicando interés compuesto
P_o: capital inicial
i: tasa de interés efectiva para el cálculo
n: período en el que se aplicará el interés

1.4. Factores de equivalencia

Los diversos usos generados por el interés compuesto han definido una serie de escenarios en los que es necesario calcular más allá del valor del capital inicial a futuro. Para ello, se han definido factores de equivalencia, los cuales permiten calcular el valor de pagos únicos, pagos de serie uniforme y

pagos con comportamiento gradiente, tanto para calcular el valor presente como el futuro.

Para facilitar la lectura de los factores de equivalencia se tiene la siguiente notación:

$$Factor = (nombre\ del\ factor, i, n)$$

Donde el nombre del factor está definido por el tipo de pago original dividido entre el período al que se desea llegar, i es la tasa de interés y n el período en el que se aplicará el factor.

A continuación se definen los principales factores de equivalencia.

1.4.1. Factor futuro dado presente

Es el factor básico dentro de los factores de equivalencia del interés compuesto. Permite la obtención del valor que alcanzará el capital inicial al final del período especificado, utilizando tanto el valor del mismo como la tasa de interés.

El factor futuro dado presente está representado por la ecuación del interés compuesto $[Ec. 5]$ y se simplifica con la notación:

$$F = P_o(F/P, i, n) \qquad [Ec. 6]$$

Donde

F: valor futuro del capital inicial
P_0: capital inicial en cuestión

F/P: nombre del factor futuro dado presente
i: tasa de interés
n: el período en el que se aplicará el factor

1.4.2. Factor presente dado futuro

Es el factor encargado de obtener el valor presente de un valor futuro a través del cálculo inverso del interés compuesto. Está representado por la siguiente ecuación:

$$P = F(1+i)^{-n} = F\ (P/F, i, n) \qquad [Ec. 7]$$

Donde

P: valor presente a obtener con el factor
F: el valor futuro
F/P: indica que se utilizará el factor presente dado futuro
i: tasa de interés
n: período en el que se aplicará el factor

1.4.3. Factor de recuperación de capital por pagos uniformes

Es el factor de interés compuesto que permite fraccionar el capital inicial en una serie de pagos uniformes, de forma equitativa cronológicamente, en un período específico.

Para los productos crediticios, es uno de los factores más utilizados, ya que permite la división de un valor monetario específico de forma equitativa, sin incurrir en pérdida por el mal cálculo de intereses.

El factor está dado por la siguiente ecuación:

$$A = P\left[\frac{i(1+i)^n}{(1+i)^n - 1}\right] = P\ (A/P, i, n) \qquad [Ec. 8]$$

Donde

A: representa la anualidad o pago uniforme
P: valor presente dado
A/P: nombre del factor anualidad dado presente
i: tasa de interés a aplicar al factor
n: cantidad de tiempo en la que se realizará el pago uniforme

1.4.4. Factor presente dada una serie uniforme

Es el factor de interés compuesto inverso del factor de recuperación de capital por pagos uniformes. Permite obtener el valor del capital inicial a partir de una serie de pagos uniformes llevados a cabo durante un período específico.

El factor está dado por la siguiente ecuación:

$$P = A\left[\frac{i(1+i)^n}{(1+i)^n - 1}\right]^{-1} = A\ (P/A, i, n) \qquad [Ec. 9]$$

Donde

P: valor del capital inicial a obtener
A: valor de los pagos uniformes a realizar
i: tasa de interés

n: tiempo en el que se encuentran distribuidos los pagos uniformes
P/A: nombre del factor presente dado anualidad

1.4.5. Factores de gradiente aritmético

Es una serie de flujo de efectivo que aumenta o disminuye de manera uniforme a través del tiempo.

En cada factor gradiente, el cambio está dado por la siguiente ecuación:

$$G = \frac{\Delta P}{n-1} \qquad [Ec. 10]$$

Donde

G: valor del incremento uniforme que se dará a través del tiempo
ΔP: cambio dado entre el valor inicial del factor y el último
n: tiempo en el que se llevará a cabo el cálculo del factor

En la siguiente tabla se muestran los factores gradientes existentes.

Tabla I. **Factores de gradiente aritmético**

Nombre	Ecuación	Notación corta
Presente dado gradiente	$P = \frac{G}{i}\left[\frac{(1+i)^n - 1}{i\,(1+i)^n} - \frac{n}{(1+i)^n}\right]$ $[Ec. 11]$	$P = G(P/G, i, n)$
Futuro dado gradiente	$F = G\left[\frac{1}{i}\left(\frac{(1+i)^n - 1}{i} - n\right)\right]$ $[Ec. 12]$	$F = G(F/G, i, n)$
Serie dado gradiente	$A = G\left[\frac{1}{i} - \frac{n}{(1+i)^n - 1}\right]$ $[Ec. 13]$	$A = G(A/G, i, n)$

Fuente: elaboración propia.

Donde

G: valor de incremento gradiente
F: valor futuro a obtener
P: valor presente a obtener a partir del gradiente
A: valor del pago uniforme a calcular
n: tiempo en el que se aplicará el gradiente
i: tasa de interés

Las ecuaciones mostradas con anterioridad reflejan únicamente el valor de cada factor de equivalencia dado el gradiente. Para calcular el total de cada tipo de capital, se deberán aplicar las ecuaciones mostradas a continuación.

Tabla II. **Ecuaciones del cálculo total de capital**

Valor a obtener	**Ecuación**
Presente total	$P = P_A + P_G \quad [Ec. 14]$
Serie uniforme total	$A_t = A + A_G \quad [Ec. 15]$
Futuro total	$F = F_A + F_G \quad [Ec. 16]$

Fuente: elaboración propia.

Donde

P: valor presente a obtener

P_A: valor del presente calculado a partir de la serie uniforme que posee el gradiente aritmético

P_G: valor del presente obtenido a partir del gradiente aritmético

A_t: valor de la serie uniforme total

A: valor base para el gradiente aritmético

A_G: valor de la serie uniforme obtenido a través del factor gradiente

F: valor futuro a obtener

F_A: valor del futuro a partir de la base uniforme del pago gradiente aritmético

F_G: valor futuro obtenido a partir del factor de gradiente aritmético

1.5. Valor presente neto

Es uno de los métodos utilizados para el análisis financiero, el cual consiste en el traslado de los valores futuros de las operaciones monetarias de inversión y crédito hacia el presente, cuando se inicia el análisis de dichas operaciones.

Cuando se trasladan cantidades de presente a futuro, estas ganan intereses, por lo que en el proceso inverso en cálculo del VPN, se utiliza una tasa de descuento, ya que permite conocer el valor de las operaciones futuras sin intereses en el presente, a las que se les conoce como flujos descontados.

El valor presente neto puede ser calculado como la sumatoria de los valores de las operaciones de series uniformes y gradientes convertidas a futuras, las cuales son trasladadas al presente utilizando factores de equivalencia de interés compuesto. Se representa con la siguiente ecuación:

$$VPN = P + \sum_{j=1}^{m} FNE_j(P/F, i_j, n_j) \qquad [Ec. 17]$$

Donde

VPN: valor presente neto a calcular.
P: capital inicial o movimiento de apertura del flujo de efectivo.
j: número de operación correspondiente.
m: cantidad de operaciones a analizar.
FNE_j: valor futuro neto equivalente, el cual puede ser un valor futuro simple o uno calculado a partir de otros factores de equivalencia, como serie uniforme o gradiente aritmético.
P/F: nombre del factor de equivalencia a aplicar.
i_j: tasa de interés aplicada a la operación j.
n_j: período en el que se encuentra la operación j.

1.5.1. Análisis del VPN

El VPN sin análisis no es más que un número obtenido luego de realizar operaciones matemáticas que implican tasas de interés y tiempos. Sin embargo, su correcta utilización y análisis permite dar a conocer si un proyecto a futuro es rentable o no.

El análisis es posible realizarlo basado en el resultado obtenido con el VPN de la siguiente manera:

- $VPN > 0$ indica que la inversión o proyecto a analizar es conveniente para su ejecución.

- $VPN = 0$ indica que se encuentra en un punto de equilibrio, por lo que el proyecto o inversión analizada no obtendrá ni pérdidas ni ganancias.

- $VPN < 0$ indica que la inversión o proyecto no es conveniente para su ejecución.

En el caso de analizar el VPN de gastos, se debe tomar el gasto cuyo VPN se acerque más a 0.

1.6. Tasa interna de retorno

También llamada tasa interna de rendimiento, indica al inversionista el equilibrio económico de un proyecto, es decir, cuando el VPN es igual a 0, por lo tanto, no generaría ni pérdidas ni ganancias. Generalmente es comparada con la tasa del inversionista o TMAR.

La obtención de la tasa interna de retorno se da gracias a operaciones de interpolación lineal de la siguiente manera:

- Si VPN > 0 entonces se deberá decrecer la tasa de interés en cantidades razonables (no más de un 5 %) hasta encontrar un VPN < 0.

- Si VPN < 0 entonces se deberá incrementar la tasa de interés en cantidades razonables (no más de un 5 %) hasta encontrar un VPN >0.

En cualquiera de los dos casos, la TIR está dada por la siguiente ecuación:

$$TIR = \frac{\Delta i VP^{+}}{VP^{+} + |VP^{-}|} + i_{menor} \qquad [Ec. 18]$$

Donde

TIR: tasa interna de retorno a obtener.
Δi: diferencia entre las tasas de interés seleccionadas.
VP^{+}: valor presente de los ingresos en el flujo de efectivo.
VP^{-}: valor presente de los egresos en el flujo de efectivo.
i_{menor}: menor tasa de interés seleccionada para el análisis.

En el caso que el VPN sea igual a 0, la TIR es la tasa interés utilizada para su cálculo.

1.6.1. Análisis de la TIR

Para realizar el análisis de la tasa interna de retorno, se debe comparar su valor con la tasa líder activa, de inversionista o TMAR. El análisis con base en el resultado de la TIR es el siguiente:

- TMAR ≥ TIR indica que el proyecto o inversión es aceptable

- TMAR < TIR indica que la inversión o proyecto no es conveniente

1.7. Análisis de beneficio/costo

Es un método que evalúa las alternativas de inversión a través del cálculo de los valores de operaciones trasladados a presente de ingresos y egresos y la relación existente entre ambos.

Para realizar el análisis, es necesario calcular una relación entre beneficio sobre costo (B/C). El beneficio se obtiene a partir del cálculo del valor presente de ingresos y el costo de los egresos.

La interpretación de la relación B/C es la siguiente:

- B/C ≥ 1 indica que la inversión en un proyecto es justificable.

- B/C < 1 indica que la inversión no se justifica, por lo tanto, no se debe continuar con el proyecto.

2. INTERFACES DE PROGRAMACIÓN DE APLICACIONES

Se define como API, o interfaz de programación de aplicaciones, al conjunto de subrutinas, funciones y procedimientos que simplifican y encapsulan tareas específicas y transparentes de funcionamiento para su implementación dentro de uno o más productos de software. Una API deberá ofrecer servicios que podrán ser consumidos por componentes de software sin importar lo que estos hagan, a través de un medio de comunicación estandarizado.

Generalmente, cuando se lleva a cabo la creación de una API, se realiza por medio de dos formas. La primera implica la creación de un paquete de software que es incrustado en un programa más complejo para consumir su funcionalidad a nivel de código fuente, a lo que se le llama API integrada como módulo. La segunda forma, implica el consumo de servicios con una funcionalidad específica por medio de protocolos de red.

2.1. API integrada como módulo

Este concepto hace referencia a la inclusión de una interfaz de programación consumible a nivel de código fuente, como una dependencia o biblioteca dentro de un producto de software. La API describe y prescribe el comportamiento deseado de la biblioteca o dependencia instanciada, siendo la descripción de los métodos a utilizar el medio de comunicación con otros componentes dependientes de ella.

La inclusión de una API a un proyecto de software agrega rutinas, estructuras de datos, clases e interfaces al código fuente de los programas en las que se agregan, con el objetivo de llevar a cabo diversas tareas, como el manejo de hardware, acceso a bases de datos, manejo de interfaz gráfica, entre otras.

Cada API puede ser desarrollada para lenguajes y tecnologías específicas. A continuación se describen algunas formas en las que se pueden presentar.

2.1.1. Descripción de bibliotecas del sistema

Una API en ocasiones, describe el comportamiento de bibliotecas compartidas del sistema operativo por medio de interfaces. Es la forma más conocida de implementación, ya que se ha utilizado desde los inicios de la computación moderna.

Las bibliotecas del sistema son archivos binarios que brindan rutinas y subrutinas específicas al funcionamiento del sistema operativo y del software que convivirá dentro del mismo. Sin embargo, la funcionalidad de estos archivos no puede ser accedida sin una interfaz para su utilización en un lenguaje de alto nivel.

Las interfaces que describen el funcionamiento de las bibliotecas del sistema pueden variar en cuanto al lenguaje de programación destino. Permiten la creación de programas que utilicen, de formas dinámicas y diferentes, la biblioteca descrita en cuestión. En algunas ocasiones, exponen el comportamiento de más de una biblioteca.

Algunos ejemplos de API de este tipo son: la interfaz de programación de la biblioteca GTK+, la cual es expuesta en varios lenguajes tales como Python, Ruby, C/C++, entre otros; la API de Android que expone la funcionalidad de las bibliotecas que controlan tanto el hardware, como los servicios del sistema operativo.

2.1.2. Comunicación con software de terceros

Utilizando el mismo concepto de descripción de bibliotecas del sistema, esta forma permite la descripción de servicios de software de terceros, abstrayendo y encapsulando la forma de comunicación entre dos o más sistemas de software diferentes con objetivos específicos.

Algunos ejemplos de este tipo de utilización de interfaces son: conectores a bases de datos (JDBC, ADODB, OLEDB, conectores nativos), conectores con redes sociales, inclusiones de aplicaciones dentro de programas específicos (Google Maps JavaScript API), motores JavaScript (Google Chrome V8), entre otros.

2.1.3. Inclusión de módulos a nivel de código fuente

Implica la simplificación de tareas agregando módulos de código fuente como dependencia de implementación dentro de un producto de software. Permiten a los desarrolladores realizar tareas genéricas que no se encuentran dentro de la lógica de negocio de un programa, como la creación y manipulación de estructuras de datos genéricas, creación de registro de sucesos (*logs*), simplificación de la utilización de protocolos de comunicación, entre otros.

Algunos ejemplos de la implementación de estas son: Java Collections Framework, módulos de node.js, System Collections Generic Namespace del .net Framework, bibliotecas HTTP, entre otros.

2.1.4. Descripción de la funcionalidad de un *framework*

Las interfaces de programación de aplicaciones integradas también están altamente relacionadas con *frameworks*, lo que permite la extensión de la funcionalidad de un lenguaje o tecnología de desarrollo para un fin específico.

Algunos ejemplos de *frameworks* que exponen su operación por medio de interfaces integradas son: express.js, hapi y loopback para node.js, Spring Framework, Hibernate y Google Web Toolkit para Java, entre otros.

2.2. API como RPC

RPC hace referencia a la técnica de realizar una llamada local en un programa y ejecutarla en algún lugar fuera del contexto de este, ya sea un servidor remoto o una aplicación diferente.

Existen diversas tecnologías de RPC, algunas necesitan de algún tipo de definición de su interfaz, otras están basadas en la transmisión de datos binarios.

Algunas tecnologías de RPC están atadas por completo a un protocolo de comunicación específico.

2.2.1. SOAP

Es un protocolo de intercambio de información estructurada, el cual es utilizado ampliamente en la creación de *web services* a través de redes de computadoras. SOAP utiliza XML como medio de representación de los datos que transporta entre solicitud y respuesta de los *web services*.

2.2.1.1. Especificación

Define un *framework* de mensajería, el cual está conformado por los siguientes elementos:

- Modelo de procesamiento: define las reglas del protocolo, identificando los roles de cada nodo SOAP.

- Modelo extensible: provee una serie de elementos que permiten hacer de SOAP un protocolo extensible, permitiendo la definición de esquemas de datos más complejos que los ofrecidos por defecto. Expone el mecanismo anterior y el siguiente para exponer características.

- *Binding framework*: describe como los servicios representados a través de SOAP intercambian mensajes.

- Construcción del mensaje: describe los bloques que cada mensaje de SOAP debe contar.

2.2.1.2. Mensajes

Un mensaje SOAP está conformado por los siguientes bloques:

- *Envelop*: identifica que el documento XML transportado es un mensaje SOAP.

- *Header*: contiene la información de los encabezados de la petición o respuesta (tipo de codificación, versión de SOAP, entre otros).

- *Body*: es el bloque que contiene los datos de la petición realizada o de la respuesta del servicio.

- *Fault*: contiene mensajes de error y estados de la transacción.

SOAP utiliza un *namespace* por defecto para los *envelops*. En este se encuentran declarados los elementos necesarios para la construcción de bloques de mensajes SOAP.

Figura 2. **Mensaje de solicitud de SOAP**

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Fuente: W3C School. http://www.w3schools.com/xml/xml_soap.asp. Consulta: 30 de diciembre de 2015.

Figura 3. **Mensaje de respuesta de SOAP**

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Fuente: W3C School. http://www.w3schools.com/xml/xml_soap.asp. Consulta: 30 de diciembre de 2015.

2.2.1.3. WSDL

Es un lenguaje de interfaz basado en XML utilizado para describir la funcionalidad esperada por parte de un *web service*. Está relacionado altamente con el protocolo SOAP, ya que describe la funcionalidad de servicios que utilizan este protocolo.

Describe los servicios por medio de rutas específicas, a través de las cuales los clientes pueden acceder. Las rutas permiten que los servicios sean descubiertos a través de internet.

En SOA, el WSDL representa el contrato por el cual se estandarizará el comportamiento de los servicios a ofrecer por parte de una organización.

2.3. API REST

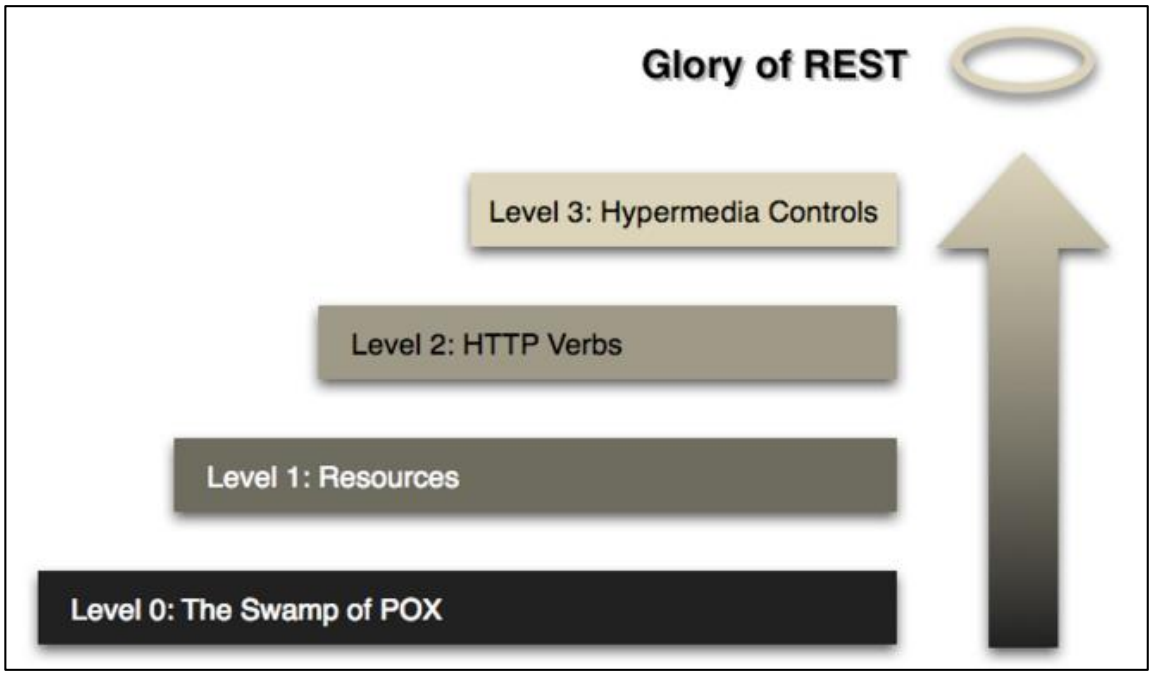
Es un estilo de arquitectura inspirado por la web que utiliza los verbos o métodos del protocolo a nivel de aplicación más conocido en internet, el HTTP. Aprovecha por completo las características de este protocolo para utilizarlo a nivel de aplicación y no solamente como transporte (tal a como SOAP se comporta).

El objetivo de REST no es solamente la exposición de operaciones (a como otros mecanismos lo hacen) sino en el manejo de recursos. Un recurso puede ser cualquier elemento del que un servicio pueda tener conocimiento propio. Las operaciones, que generalmente eran expuestas en protocolos como SOAP, son reemplazadas por los verbos de HTTP, lo que permite una interacción más natural y estandarizada entre consumidores e interfaces de programación de aplicaciones.

2.3.1. Richardson Maturity Model

Es un modelo creado por Leonard Richardson para mejorar la calidad en la creación de interfaces de programación que utilizan REST como modelo arquitectónico. Introduce tres niveles para alcanzar la madurez completa de una API RESTful.

Figura 4. **Richardson Maturity Model**



Fuente: FOWLER, Martin. *Richardson Maturity Model: steps toward the glory of REST.* http://martinfowler.com/articles/richardsonMaturityModel.html. Consulta: 30 de diciembre de 2015.

2.3.1.1. Nivel 0

En este nivel, se tiene HTTP como medio de transporte y no de aplicación, por lo que la implementación de los servicios es más parecida a un RPC como los descritos con anterioridad.

Toda aquella API que exponga operaciones específicas sin identificar recursos y que utilice HTTP como medio de transporte y no como parte de la definición de la arquitectura, es una API que se encuentra en el nivel 0 de REST.

2.3.1.2. Nivel 1

Indica que toda API RESTful deberá exponer los recursos involucrados en la funcionalidad del servicio, en lugar de exponer rutas a operaciones.

Por ejemplo, una API de mapas permite realizar operaciones como obtener puntos en el mapa, buscar mapas en específico, eliminar mapas, entre otros. Esto implicaría que la API tenga varias rutas para realizar esas operaciones. Sin embargo, ¿qué sucedería si la API, además de mapas, también operara sobre rutas de navegación? El problema escalaría en la cantidad de rutas o *endpoints* que se deberían exponer. Al implementar el nivel 1, solamente se deberían tener *endpoints* asociados a los recursos, que en este caso serían los mapas y las rutas de navegación.

2.3.1.3. Nivel 2

REST es un modelo de arquitectura basado en las cualidades que brinda el protocolo HTTP, una API RESTful deberá realizar un uso correcto de los métodos o verbos del protocolo.

A continuación se detalla el uso correcto de algunos de los verbos de HTTP dentro de REST:

Tabla III. **Verbos del protocolo HTTP y su utilización en REST**

Verbo	**Utilización**
GET	Obtención de recursos, ya sean específicos o en colección.
POST	Creación de nuevos recursos.
PUT	Modificación o creación de nuevos recursos (cuando es permitido).
DELETE	Eliminación de un recurso en específico o de una colección entera.

Fuente: elaboración propia.

Con REST es posible ampliar la cantidad de verbos de HTTP a utilizar, sin embargo, esto no es del todo recomendable, ya que generalmente son verbos que se encuentran en desarrollo o que son simples extensiones o derivaciones de los anteriormente mencionados.

Además de los verbos correctos, el nivel 2 también implica utilizar de forma correcta, los códigos de respuesta de HTTP.

2.3.1.4. Nivel 3

Es el último nivel para alcanzar la verdadera madurez en cuanto a la implementación de servicios RESTful. En este nivel, se profundiza en un concepto llamado HATEOAS (*hypertext as the engine of application state*).

El objetivo de este nivel es reducir el nivel de dependencia de los clientes de una API hacia los *endpoints* de la misma, creando *links* basados en los recursos expuestos y la relación que estos poseen con otros recursos.

Generalmente, estos *links* son creados utilizando representaciones estándar de hipermedia, dentro de los encabezados de las respuestas o en las respuestas mismas.

3. ARQUITECTURA DE MICROSERVICIOS

Es una arquitectura de desarrollo de aplicaciones que está enfocada en la construcción de servicios autónomos, los cuales se comunican de forma integral para transmitir información relevante para la aplicación que se esté desarrollando. A estos servicios autónomos se les conoce como microservicios.

Los microservicios son autónomos, se enfocan en una sola funcionalidad, que se ejecutan en su propio proceso y que son desplegados independientemente de otros servicios. Además, están desarrollados en diferentes lenguajes de programación y con una variada infraestructura entre ellos.

Entre las ventajas que posee utilizar una arquitectura basada en microservicios está la posibilidad de distribuirlos en varios sistemas, ya que al momento de desplegar un microservicio, se puede replicar en todos los sistemas que se destinen para tal, agruparlos por funcionalidades y tener un mejor control de los recursos.

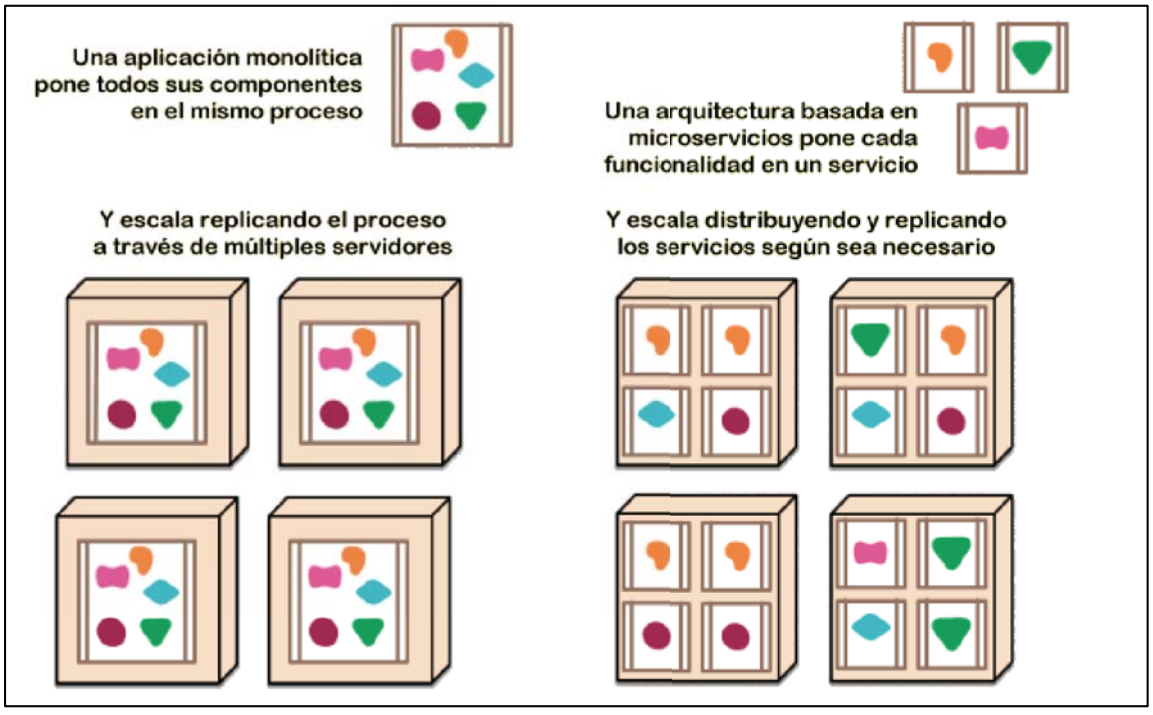
En comparación con la arquitectura monolítica, destaca que cada elemento funcional de la aplicación se encuentra unida en un único proceso, a comparación de la arquitectura de microservicios que divide las funcionalidades en procesos separados y aislados.

Además, se puede utilizar diferentes tecnologías para cada microservicio. Es decir, se pueden aprovechar diferentes enfoques para la construcción de microservicios, ya que al ser independientes entre sí, no están atados al uso de

una tecnología en concreto. Ese es uno de los riesgos que el uso de microservicios mitiga en comparación a otras arquitecturas, ya que la adopción de una nueva tecnología no implica reestructurar los demás servicios.

Otra de las características de esta arquitectura es el aislamiento que tiene cada servicio, en donde, si se encuentra un error o algún problema, este puede ser aislado para que no afecte a los demás servicios y puedan seguir trabajando con normalidad, si es que el problema no afecta a múltiples servicios.

Figura 5. **Comparativa entre arquitectura monolítica y microservicios**



Fuente: MONTORO, Sergio. *Microservicios*. http://lapastillaroja.net/2014/07/microservicios/. Consulta: 30 de diciembre de 2015.

3.1. Modelación de microservicios

Para modelar microservicios se aplican algunas características del paradigma tradicional de servicios, como el acoplamiento que tiene cada servicio, la cohesión entre ellos, bajo qué contexto trabaja, el modelo que define al servicio, los módulos y servicios con los que trabajará y, por último, el modelo de negocio que tendrá el servicio.

3.1.1. Acoplamiento

Dentro de los principios de SOA se encuentra el principio de acoplamiento débil, el cual se refiere al acoplamiento entre servicios en una arquitectura orientada a servicios, dictando que cada miembro de la arquitectura debe contener un acoplamiento débil para interactuar con el resto de los miembros.

En este acoplamiento débil se debe mantener una independencia entre el servicio y otros servicios externos, es decir, sus consumidores. Las modificaciones que recibe no deben alterar el comportamiento de los servicios que lo consumen y la forma en la que se comunican.

El mismo servicio debe ser independiente de sí mismo, es decir de no depender de la tecnología que lo implementa, de las reglas de negocio, de otros servicios y cualquier otro factor interno.

3.1.2. Cohesión

Se refiere a que cada servicio debe realizar únicamente la tarea para la que fue diseñado y cualquier otra tarea complementaria debe de realizarla otro

servicio. Es decir, cada servicio realizará exclusivamente una tarea y no se desviará del objetivo de la misma.

Si existieran tareas similares o con un comportamiento parecido, se deben unir para facilitar el mantenimiento y lograr encapsular cualquier riesgo sobre la misma.

3.1.3. Contexto acotado

Se deben definir límites entre las diferentes partes de un modelo para definir qué información manejará, cuál será su significado y definir las relaciones entre cada contexto. Esto, con el fin de unificar los esfuerzos de cada parte del modelo y que no existan contradicciones en el mismo, evitando redundancia de funcionalidades dentro de la arquitectura.

3.1.4. Modelos ocultos y compartidos

Cada parte del modelo puede compartirse entre varios contextos y tener un significado diferente en cada uno, para lo cual cada contexto debe compartir únicamente la información necesaria para los contextos relacionados.

3.1.5. Módulos y servicios

Cada contexto acotado del modelo puede ser definido como un módulo dentro del sistema y convertirse en un nuevo microservicio, siendo los límites del mismo las funcionalidades del servicio y las relaciones con otros servicios.

3.1.6. Descomposición prematura

Si el contexto no está bien definido o abarca más de lo que debería, no se debe separar inmediatamente, ya que lo primordial es definir claramente el objetivo del mismo y las relaciones entre los actores de la arquitectura, con el fin de minimizar el riesgo de cambios futuros o de tener servicios con un alto acoplamiento y baja cohesión.

3.1.7. Reglas del negocio

La definición de los contextos del modelo sobre el que se trabaja debe estar basada en las reglas del negocio, ya que se debe analizar cómo fluye la información para definir las relaciones y qué información se requiere para construir correctamente cada parte del modelo.

3.2. Integración de microservicios

Es una de las características más importantes de la arquitectura. Debe facilitar la implementación de las características que se definieron al modelar cada microservicio y mantener su autonomía.

Existen algunas formas de realizar esta integración, entre las cuales se definen las siguientes.

3.2.1. Manejo de los cambios

Se refiere a que los cambios no deben interferir con el funcionamiento de los servicios o aplicaciones de los clientes, ni con los servicios internos con los

que se relaciona. El manejo de los cambios está altamente relacionado con los principios de débil acoplamiento y de alta cohesión entre servicios.

Por ejemplo, el agregar un valor a una respuesta no debe afectar el funcionamiento, tanto a los clientes como al negocio.

3.2.2. Servicios agnósticos

El agnosticismo de los servicios implica que estos sean independientes, en cuanto a funcionalidad y tecnología, de otros miembros de la arquitectura.

Cuando un servicio es agnóstico puede ser reutilizado muchas veces en diferentes contextos, aún si estos se encuentran dispersos a lo largo de contextos muy distintos entre sí, permitiendo así la reusabilidad del mismo, lo que agrega valor al desarrollo de la tarea.

La reusabilidad dentro de la ingeniería del software está altamente relacionada a cuan genérica es la tarea que se está reutilizando. Para construir una arquitectura orientada a servicios (SOA o microservicios), cada servicio deberá realizar tareas atómicas y genéricas para el negocio que se desea modelar con la arquitectura.

3.2.3. Sencillez de los servicios

Para cada servicio, este debe ser sencillo de utilizar para el cliente, promoviendo el uso libre de cualquier tecnología que el cliente desea utilizar.

También puede construirse una biblioteca o módulo cliente para estandarizar la adopción de los servicios prestados por la empresa u

organización, pero esto puede causar problemas de incompatibilidad con alguna tecnología y, a su vez, incrementa el acoplamiento de los servicios.

3.2.4. Esconder detalles de implementación

La lógica de cada servicio debe ser encapsulada dentro de sí misma. Cada cliente accederá a él por medio de un contrato estandarizado o por un protocolo de comunicación establecido y acordado previamente sin necesidad de conocer el comportamiento interno del servicio.

Los clientes no deben depender sobre la implementación interna de los servicios de la organización, ya que al momento de requerir un cambio, este afectará al cliente y llevaría a un manejo equivoco de los cambios, aumentando el acoplamiento de los servicios y sus consumidores. Además, esto hará al servicio dependiente de los clientes y será menos proclive a realizarle cambios.

3.3. Despliegue de microservicios

Desplegar un microservicio o un conjunto de ellos difiere con el despliegue tradicional de una aplicación monolítica, el cual es un proceso de pasos lineales, mientras que con los microservicios, se necesitan aplicar uno o incluso varios pasos a la vez.

3.3.1. Integración continua

La idea principal de la integración continua es mantener todo los elementos sincronizados, analizando que todo el código nuevo se integre con el código existente, verificando que el código integrado pase las pruebas de compilación y de verificación.

Entre algunos de los elementos que se utilizan para realizar esta integración, se encuentran los artefactos, contar con un repositorio central para almacenar las distintas versiones de código, disponer de un módulo de diferentes pruebas para verificar el código y todo debe ser realizado mediante un flujo automatizado.

3.3.2. Control de versiones de artefactos

Dentro de la gestión de la configuración, el control de versiones es un mecanismo para mantener múltiples versiones y el historial de modificación de artefactos, archivos de código fuente, documentación, *scripts* de construcción y pruebas de proyectos de software.

Permite a los equipos de desarrollo saber quién, cuándo y porqué se realizó un cambio dentro del proyecto.

La finalidad principal es proveer un control dentro del desarrollo de software colaborativo, ya que permite que equipos de desarrollo permitan la evolución de un producto de software a través del espacio y el tiempo.

Es de vital importancia para el despliegue de microservicios contar con un sistema de control de versiones, con el objetivo de llevar el control del desarrollo de cada microservicio, manejando las versiones con las que se desplegará cada uno.

3.3.3. Tuberías de despliegue

Son flujos de trabajo que permiten ordenar los distintos elementos de la integración continua.

Para mantener un flujo de trabajo ordenado, se elaboran diferentes etapas de compilación. En cada etapa se puede monitorear el funcionamiento del servicio obteniendo retroalimentación para futuras revisiones y despliegues.

3.3.4. Artefactos

Son los elementos tangibles dentro del ciclo de desarrollo de software, entre los que se incluyen diagramas, modelos, documentos y versiones de programas en desarrollo, generalmente empacados en un contenedor.

Estos artefactos dependen de herramientas de configuración automática para ser desplegados y ejecutados, debido a que al no depender de una tecnología en específico, se requiere llevar el control de las herramientas necesarias para el despliegue y ejecución, considerando que pueden ser varias las tecnologías a configurar.

3.3.5. Imágenes

Uno de los grandes problemas de las herramientas de configuración automática es el tiempo en ejecutar un *script* de despliegue o ejecución, el cual conlleva instalar una nueva aplicación para el ambiente de trabajo o revisar que esta ya se encuentre instalada en el sistema.

Al momento de realizar esta comprobación, surgen problemas si todos los recursos de algún servidor están siendo utilizados o si se tiene que realizar este proceso varias veces al día.

Para mitigar este tipo de problemas, se utilizan herramientas de virtualización, las cuales pueden generar una imagen de sistemas con las

herramientas básicas para ejecutar los microservicios y luego generar una instancia de esa imagen para desplegarlo.

3.4. Pruebas de microservicios

Las pruebas automatizadas son un recurso utilizado en el desarrollo de software para evaluar el funcionamiento del código de una manera controlada, comparando los resultados obtenidos con los resultados esperados, generalmente realizadas en ambientes de prueba.

Para las pruebas realizadas sobre una arquitectura de microservicios, se adopta una estrategia enfocada en cubrir cada modelo del microservicio y como estos se relacionan entre sí, ya que esto permite validar cada capa del sistema, corrigiendo las fallas en cada capa hasta cubrir con la totalidad del servicio.

Para cada componente de un microservicio se necesita aplicar una estrategia diferente de pruebas que valide el funcionamiento del mismo. Por ejemplo, utilizar la virtualización de software, lo cual permitirá realizar pruebas individuales en cada servicio sin esperar al despliegue de otros servicios dependientes.

3.4.1. Pruebas unitarias

El propósito de esta prueba es verificar el funcionamiento del microservicio sin depender de otros componentes, comprobando una única funcionalidad a la vez, evaluando el funcionamiento del núcleo del microservicio para garantizar el correcto funcionamiento desde las capas internas del mismo.

Brinda una rápida retroalimentación sobre la funcionalidad que se está evaluando, facilitando la búsqueda de errores y la resolución de problemas de diseño de código, permitiendo reestructurar el código mientras se está desarrollando

Estas pruebas están orientadas a probar la arquitectura tecnológica en lugar de las reglas de negocio.

3.4.2. Pruebas de integración

Luego que el funcionamiento individual de cada uno de los microservicios haya sido verificado, se necesita probar las interacciones entre los mismos, por medio de las pruebas de integración.

Estas pruebas deben realizarse utilizando la arquitectura integrada a los servicios dependientes. Esto garantiza que el sistema está funcionando correctamente y las dependencias entre servicios tienen un comportamiento esperado.

3.4.3. Pruebas de punto a punto

Son las pruebas que evalúan el funcionamiento de sistema como un todo, guiadas a través de la interfaz gráfica o por medio de otros procesos como la carga de un archivo, imitando la interacción del usuario con el sistema.

Estas pruebas incluyen verificaciones sobre el flujo de proceso y la interacción con las fuentes de datos, siendo necesario realizar más pruebas a fondo, dependiendo del número de servicios que estén siendo analizados para garantizar que el sistema cumpla con los requerimientos propuestos.

3.4.4. Alcance de las pruebas

Delimitar el alcance de las pruebas de microservicios depende de la naturaleza de los microservicios y de cómo interactuarán entre sí, ya que eso permite enfocar los esfuerzos y detectar de una forma más eficiente los posibles fallos.

Para las pruebas de microservicios enfocados en consumir los recursos de una fuente de datos, se debe comprobar que estos datos solamente sean accesibles desde el alcance del servicio, creando para ello puntos de acceso, desde los cuales se presenta la información y esta es obtenida por medio de las diferentes llamadas a otros servicios.

3.5. Seguridad de la infraestructura

La seguridad es un tema crítico al implementar microservicios, ya que maneja uno de los activos más importantes con la que cuenta una organización, y esta es la información. Para cada elemento que conforma un microservicio se pueden definir medidas de protección, entre las que se incluye la seguridad de la infraestructura, de cómo fluye la información y de quién o quiénes tienen acceso a los distintos elementos de la implementación.

3.5.1. Autenticación y autorización

Entre las principales bases para definir el control de acceso a las personas o elementos que interactuarán con los recursos de los microservicios, se utiliza la autenticación y la autorización.

El proceso de autenticación inicia cuando una persona o un elemento confirma quien dice ser, ya sea por medio de credenciales de acceso, como el usuario y la contraseña, o alguna otra forma similar, y luego son validadas por una autoridad, enviando una confirmación o rechazo sobre dicha validación.

En el proceso de autorización se decide si la persona o elemento tiene acceso a uno o varios recursos de los microservicios, que puede ser información, algún servicio o funcionalidad específica.

Al momento de autenticarse se puede consultar la información con la cual hizo el proceso y decidir qué se le permite hacer en el sistema. Estos permisos le son concedidos generalmente por el administrador del sistema de forma manual o de forma automática por medio de las herramientas diseñadas para ese fin.

3.5.2. API Keys

La autenticación a nivel de negocio es parte de las características de cualquier sistema de software. Sin embargo, el identificar un acceso a recursos y entidades de negocio no significa conocer de dónde provienen las solicitudes realizadas al sistema.

La arquitectura de microservicios, generalmente, es utilizada en la estructura interna de una API remota. Para cualquier microservicio, es importante reconocer al cliente que le está realizando una solicitud, por lo que la autenticación a nivel de negocio o aplicación no es significativa para el contexto del microservicio como tal.

Por ello, es necesario implementar un mecanismo de identificación de los clientes de la arquitectura de microservicios. Uno de estos mecanismos es la autenticación por API Key.

A través de un API Key es posible asignar una clave única e inequívoca a un cliente específico de un sistema conformado por aplicaciones monolíticas o microservicios. Entre los clientes más comunes se encuentran aplicaciones móviles, sitios web, otros microservicios, entre otros.

En el mercado de servicios en la nube, los proveedores de interfaces de programación de aplicaciones y *web services* proveen API Keys para delimitar y autorizar las acciones de sus clientes, utilizando métricas como la cantidad de accesos a los recursos, de instancias de un cliente, de instalaciones de un software específico, entre otras.

4. NECESIDAD DEL ANÁLISIS DE VIABILIDAD FINANCIERA

La creación de nuevas empresas, micro, pequeña y mediana, en Guatemala, siempre ronda por un alto número cada año. De acuerdo a estadísticas del Banco Mundial, solo en 2012 y 2013 se registraron alrededor de 5 000 y 4 000 empresas nuevas respectivamente. Estas presentan retos más allá de lo económico para su sobrevivencia, entre ellos se identifican factores como la inestabilidad política, el crimen, los impuestos, la administración tributaria, el financiamiento laboral, el pago de servicios, entre otros.

El acceso a crédito es uno de los principales problemas que enfrentan las micro y pequeñas empresas nuevas, ya que generalmente muchas de ellas son rechazadas (30 % aproximadamente de las primeras y 15 % de las segundas) o no logran cumplir con las deudas adquiridas con bancos y entidades financieras.

De acuerdo a estadísticas del Registro Mercantil de Guatemala, en 2015 fueron cerradas alrededor de 2 000 empresas.

4.1. Análisis financiero para créditos personales y privados

Además de las nuevas empresas que se han creado en Guatemala, los créditos personales han ido en aumento, ya que las cifras de los créditos hasta junio de 2015 han aumentado en un 10,5 % con respecto a junio de 2014, con una cantidad de Q 50,9 millardos frente a los Q 46,1 millardos del año anterior.

De manera similar, los créditos privados han aumentado en un 12 % al incrementarse desde Q 128 481 millones en 2014 hasta Q 143 995 millones en 2015.

4.2. Análisis financiero para micro y pequeñas empresas

La mayoría de las nuevas micro y pequeñas empresas no realizan una planificación adecuada de acuerdo a los créditos y gastos que deberán adquirir, omitiendo variables importantes como la inflación, las tasas de interés aplicadas a créditos, el tipo de cambio, entre otros.

El análisis financiero permite a los nuevos empresarios realizar estimaciones sobre el capital necesario para la inicialización de sus proyectos y de la factibilidad económica que estos tendrán a futuro.

También permite seleccionar alternativas de inversión o crédito, indicando cuál de los proveedores de servicios crediticios en el mercado financiero es el adecuado para el inicio de operaciones de la empresa.

Existen entidades estatales que brindan apoyo a micro y pequeñas empresas, las cuales podrían brindar un servicio de análisis financiero para garantizar la sobrevivencia de las mismas.

4.3. Adquisición de activos y patrimonio

La mayoría de bienes inmuebles y vehículos son adquiridos por medio de crédito. Algunas personas individuales y micro y pequeñas empresas realizan compras de activos por medio de productos financieros, sin embargo, las adquisiciones peligran por falta de pago de cuotas o por el cese de actividades

de la empresa. Para ello, un análisis financiero permitirá seleccionar la mejor forma de adquirir un bien o si es factible su compra.

4.4. Propuesta de solución tecnológica

Es posible realizar el análisis financiero de manera automatizada, sin embargo, existen muy pocas soluciones tecnológicas e informáticas que presten ese servicio a instituciones dedicadas al fomento del crecimiento y sobrevivencia de micro y pequeñas empresas.

El análisis financiero puede ser prestado tanto por instituciones estatales como privadas. Entre las instituciones estatales que apoyan al sector de Mipymes en Guatemala se encuentran: Viceministerio de las Mipymes del Ministerio de Economía, Programa Nacional de Competitividad, Instituto Técnico de Capacitación y Productividad, Escuela Nacional Central de Agricultura, Consejo Nacional de Ciencia y Tecnología, entre otras.

Entre los recursos disponibles de manera electrónica de cada dependencia no se encuentra un servicio de análisis financiero para micro y pequeños nuevos empresarios, por lo que la creación de un servicio remoto para el desarrollo del mismo puede ser de ayuda para estas instituciones y para los nuevos emprendedores.

4.4.1. Características de la solución tecnológica

Para prestar el servicio de análisis financiero, se debe de ir más allá del simple desarrollo de un sitio web o una aplicación para cada institución o empresa que de soporte a nuevos proyectos de emprendimiento. Por ello, se

propone la creación de un servicio remoto, cuyo acceso sea a través de una API RESTful.

La razón principal para ofrecer este servicio como una API RESTful es debido a la facilidad de integración que podrán tener las instituciones o empresas al mismo, ya que, por definición, una API encapsula la funcionalidad interna de un servicio y expone solamente el modo de interacción con sus clientes. Además, se plantea con el estilo de arquitectura RESTful debido a la facilidad de utilizar elementos de hipermedia para la comunicación.

La solución tecnológica contará con las siguientes características principales:

Tabla IV. **Características del servicio de análisis financiero**

Característica	**Descripción**
Identificación de usuarios dentro de la plataforma	Los usuarios quienes solicitarán un análisis financiero de sus proyectos tendrán la capacidad de identificarse de manera inequívoca dentro del sistema a través de la institución o empresa que le provea el servicio.
División de los análisis financieros por medio de proyectos	Cada usuario podrá crear diferentes proyectos de inversión, los cuales se traducirán en análisis financieros para cada uno. Con ello, los usuarios podrán evaluar la factibilidad de sus proyectos antes de ejecutarlos.
Obtención de variables económicas	Los usuarios podrán obtener variables económicas actuales, como tasas de interés e inflación.

Continuación de la tabla IV.

API RESTful pública	El servicio será prestado de manera remota por medio de una API RESTful, la cual podrá ser consumida por cualquier aplicación web, móvil, de escritorio o cualquier otro tipo de solución tecnológica.

Fuente: elaboración propia

5. PROCESO DE DESARROLLO DE LA API

La ingeniería de software agrega un enfoque sistémico, cuantificable y disciplinado al desarrollo de software en general, así como el resto de las etapas del ciclo de vida del mismo. Los procesos de desarrollo de cualquier producto de software pueden variar dependiendo de la naturaleza que se tenga, del objetivo que se desea cumplir y de los interesados en el mismo.

En el caso de la API RESTful propuesta, el proceso de desarrollo deberá realizarse a partir de la concepción de la idea de la creación de un sistema de análisis financiero, posterior a ello, deberán definirse los procesos propios, el dominio de la API y su lógica de negocio.

Una vez definido lo anterior, es posible generar una definición, análisis y diseño de los servicios que prestará la API. Esto ayudará a identificar posibles dominios de servicio individuales para la creación de la arquitectura de microservicios que tendrá. Posterior a ello, será posible la definición de modelos de datos y de elementos tecnológicos para el desarrollo.

5.1. Definición de lógica de negocio y arquitectura

Antes de desarrollar cualquier tipo de componente de software, es necesario tener claro el objetivo y el alcance que tendrá el mismo. Para ello se define la lógica de negocio a cubrir, los procesos clave necesarios y la arquitectura del sistema, la cual estará basada en microservicios.

En la ingeniería de software se utilizan distintas herramientas para documentar, analizar y diseñar la lógica de negocio de un producto de software. Una de las principales herramientas es UML.

UML es un lenguaje gráfico que provee diagramas cuyo objetivo principal es dejar claro el alcance de un sistema de software a partir de la visualización de su arquitectura, la especificación de sus componentes principales, la construcción y la documentación de los mismos.

Para definir la lógica de negocio del sistema, se utilizarán los diagramas UML de casos de uso y de secuencia.

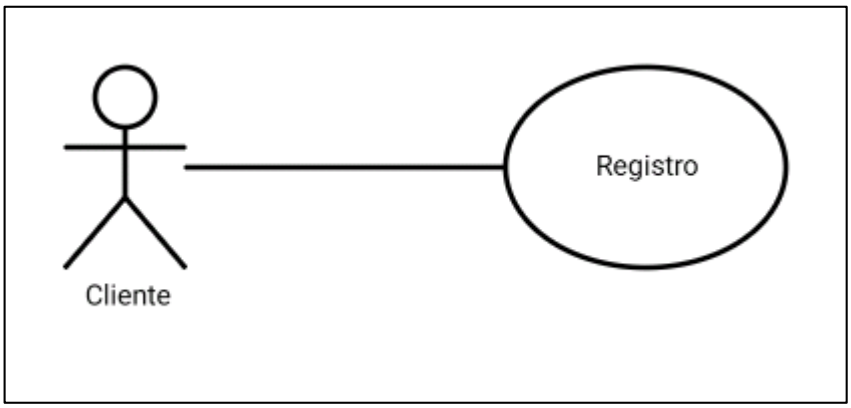
5.1.1. Diagramas de casos de uso

Se utilizarán para definir la interacción entre los clientes del sistema y el sistema en sí. Modela lo que cada cliente puede realizar a través de las interfaces expuestas por el sistema. A continuación se muestran los diagramas de caso de uso del sistema con su respectiva descripción de caso de uso.

5.1.1.1. Registro

El caso de uso de registro permitirá definir la funcionalidad del registro de nuevos dueños de proyectos o usuarios a nivel de la API. Todo cliente de la misma podrá registrar nuevos usuarios dueños.

Figura 6. **Diagrama de caso de uso de registro**



Fuente: elaboración propia, empleando DrawExpress.

Tabla V. **Definición caso de uso de registro**

Caso de uso	Registro
Actores	Cliente de la API
Propósito	Permitir el registro de usuarios dueños de proyectos a cada cliente de la API.
Resumen	Toda aplicación cliente de la API deberá tener la capacidad de crear usuarios dueños de proyectos dentro del sistema de análisis financiero expuesto por la API.
Precondiciones del flujo principal	Los clientes deberán poder acceder a una interfaz agnóstica para la inserción de nuevos usuarios.

Continuación de la tabla V.

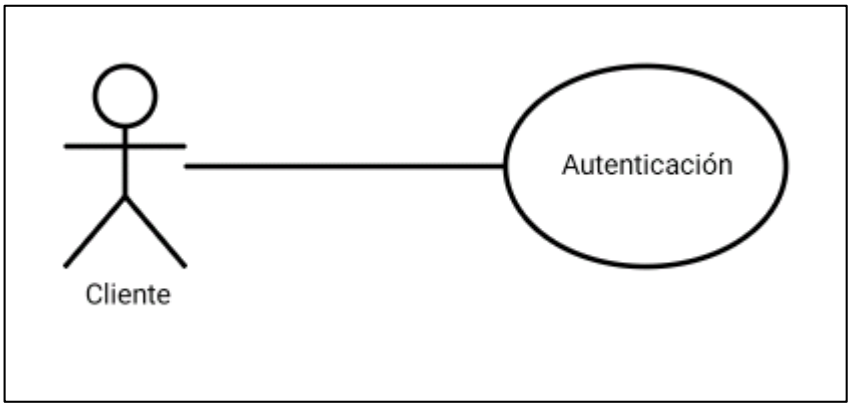
Flujo principal	Cualquier aplicación cliente ingresará los datos del usuario a insertar siendo estos: • Nombre de usuario • Correo electrónico • Contraseña
Excepciones	E1 – No se cumple con los parámetros requeridos. E2 – Ya existe un usuario dueño con la información ingresada.

Fuente: elaboración propia

5.1.1.2. Autenticación

Cada cliente deberá autenticar a sus usuarios, por lo que será necesario que la API posea un mecanismo para identificar a los usuarios dueños de proyectos que acceden a sus recursos, independientemente del cliente que consuma la API.

Figura 7. **Diagrama de caso de uso de autenticación**



Fuente: elaboración propia, empleando DrawExpress.

Tabla VI. **Definición de caso de uso de autenticación**

Caso de uso	Autenticación
Actores	Cliente de la API
Propósito	Permitir el acceso a recursos únicamente a usuarios autenticados.
Resumen	Toda aplicación cliente de la API deberá tener la capacidad de autenticar a los usuarios dueños de proyectos para acceder a recursos dentro del sistema de análisis financiero expuesto por la API.
Precondiciones del flujo principal	Los clientes deberán poder acceder a una interfaz agnóstica para la autenticación de usuarios existentes.

Continuación de la tabla VI.

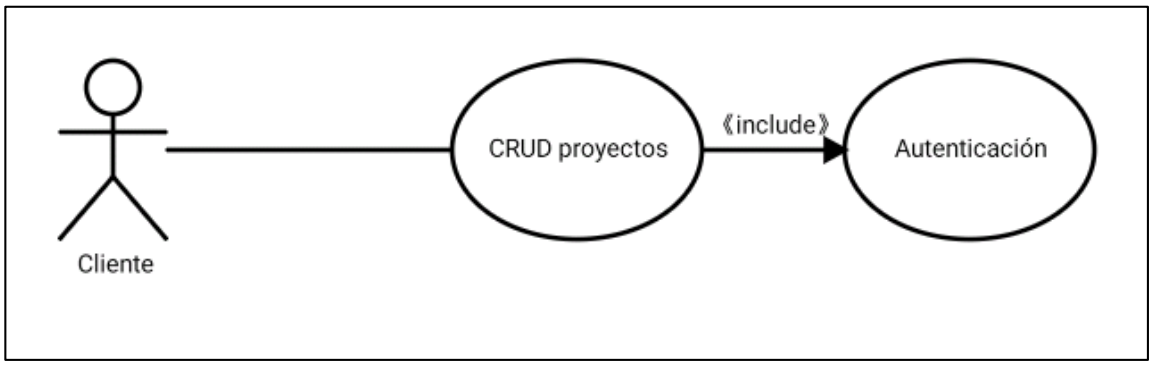
Flujo principal	Cualquier aplicación cliente ingresará los datos de autenticación de un usuario dueño para obtener autorización a acceso a recursos, los cuales son: • Nombre de usuario y contraseña • Correo electrónico y contraseña.
Excepciones	E3 – No se cumple con los parámetros requeridos. E4 – El nombre de usuario, correo electrónico o contraseña no se encuentra registrado en el sistema.

Fuente: elaboración propia.

5.1.1.3. Creación, lectura, modificación y eliminación de proyectos

A través de los clientes de la API cada usuario podrá crear, acceder, modificar y eliminar sus propios proyectos con la debida autenticación y autorización.

Figura 8. **Caso de uso CRUD de proyectos**



Fuente: elaboración propia, empleando DrawExpress.

Tabla VII. **Definición de caso de uso CRUD de proyectos**

Caso de uso	CRUD proyectos
Actores	Cliente de la API
Propósito	Permitir el acceso, creación, actualización y eliminación de proyectos.
Casos de uso relacionados	Autenticación
Resumen	Los usuarios dueños podrán realizar tareas de creación, actualización, acceso y eliminación de sus proyectos
Precondiciones del flujo principal	Para acceder a cualquier proyecto, los clientes deberán brindar el código de autorización del usuario que desea acceder.

Continuación de la tabla VII.

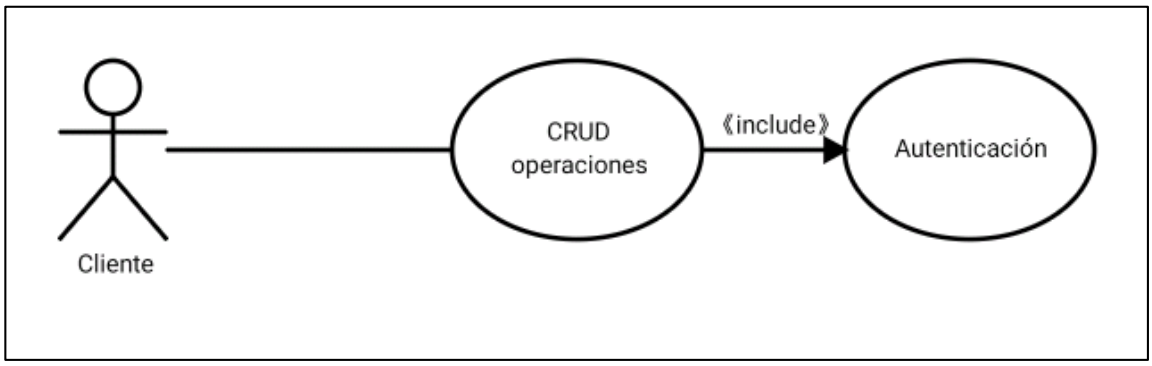
Flujo principal	Cualquier aplicación cliente podrá crear, modificar, acceder y eliminar proyectos de un usuario autenticado.
Excepciones	E4 – No se cumple con los parámetros requeridos. E5 – No se tiene la autorización necesaria para realizar la operación. E6 – El proyecto a obtener no existe.

Fuente: elaboración propia.

5.1.1.4. Creación, lectura, modificación y eliminación de operaciones

Las operaciones a realizar para cada proyecto podrán ser creadas, accedidas, modificadas y eliminadas por los usuarios dueños de las mismas a través cualquier cliente de la API.

Figura 9. **Caso de uso CRUD de operaciones**



Fuente: elaboración propia, empleando DrawExpress.

Tabla VIII. **Definición del caso de uso CRUD operaciones**

Caso de uso	CRUD operaciones
Actores	Cliente de la API
Propósito	Permitir el acceso, creación, actualización y eliminación de operaciones.
Casos de uso relacionados	Autenticación
Resumen	Los usuarios dueños podrán realizar tareas de creación, actualización, acceso y eliminación de las operaciones de sus proyectos.
Precondiciones del flujo principal	Para acceder a cualquier operación, los clientes deberán brindar el código de autorización del usuario que desea acceder.

Continuación de tabla VIII.

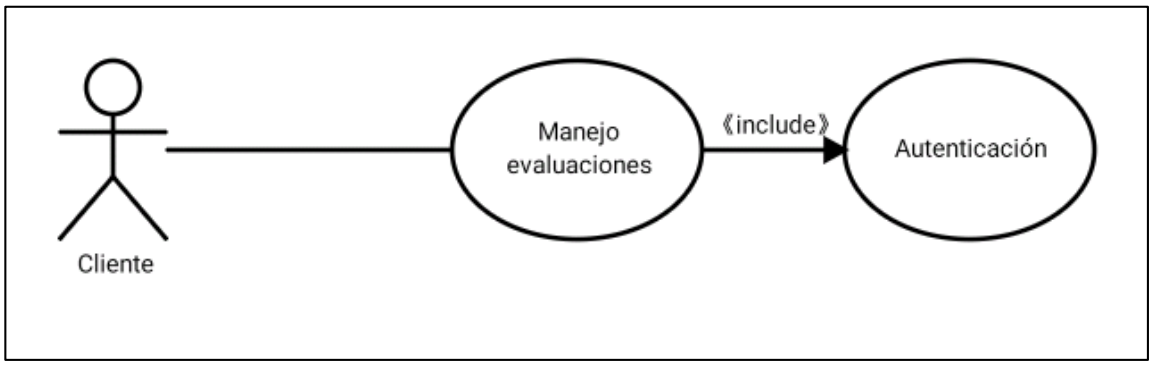
Flujo principal	Cualquier aplicación cliente podrá crear, modificar, acceder y eliminar operaciones de un usuario autenticado.
Excepciones	E7 – No se cumple con los parámetros requeridos. E8 – No se tiene la autorización necesaria para realizar la operación. E9 – La operación a obtener no existe.

Fuente: elaboración propia.

5.1.1.5. Creación, lectura y eliminación de evaluaciones

Cada usuario dueño de proyectos podrá crear, acceder y eliminar evaluaciones de viabilidad financiera para cada proyecto. La creación de evaluaciones dependerá de la existencia de un proyecto y podrá tomar más tiempo para responder que los demás casos de uso.

Figura 10. **Caso de uso de manejo de evaluaciones**



Fuente: elaboración propia, empleando DrawExpress.

Tabla IX. **Definición caso de uso de manejo de evaluaciones**

Caso de uso	Manejo evaluaciones
Actores	Cliente de la API
Propósito	Permitir el acceso, creación y eliminación de evaluaciones.
Casos de uso relacionados	Autenticación
Resumen	Los usuarios dueños podrán realizar tareas de creación, acceso y eliminación de las evaluaciones de sus proyectos.
Precondiciones del flujo principal	Para acceder a cualquier evaluación, los clientes deberán brindar el código de autorización del usuario que desea acceder, además de contar con al menos un proyecto.

Continuación de tabla IX.

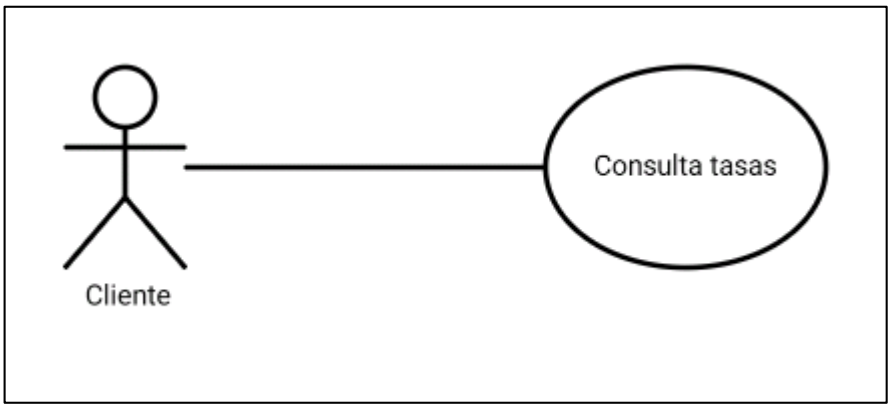
Flujo principal	Cualquier aplicación cliente podrá crear, acceder y eliminar evaluaciones de un usuario autenticado.
Excepciones	E10 – No se tiene la autorización necesaria para realizar la evaluación. E11 – La evaluación a obtener no existe.

Fuente: elaboración propia.

5.1.1.6. Consulta de tasas de interés

Cada cliente de la API podrá obtener datos actualizados y oficiales proveídos por el Banguat a través de la consulta de tasas de interés. Para este caso de uso, los clientes de la API no necesitan autenticar usuarios para su consumo.

Figura 11. **Caso de uso de consulta de tasas**



Fuente: elaboración propia, empleando DrawExpress.

Tabla X. **Definición de caso de uso de consulta de tasas**

Caso de uso	Consulta tasas
Actores	Cliente de la API
Propósito	Permitir la consulta de las tasas líderes proveídas por el Banguat.
Resumen	A través de cualquier cliente, es posible consultar las tasas líderes de interés del Banguat.
Precondiciones del flujo principal	Ninguna
Flujo principal	Cualquier aplicación cliente podrá consultar las tasas de interés que provee el Banguat desde la API.
Excepciones	E12 – No se poseen tasas

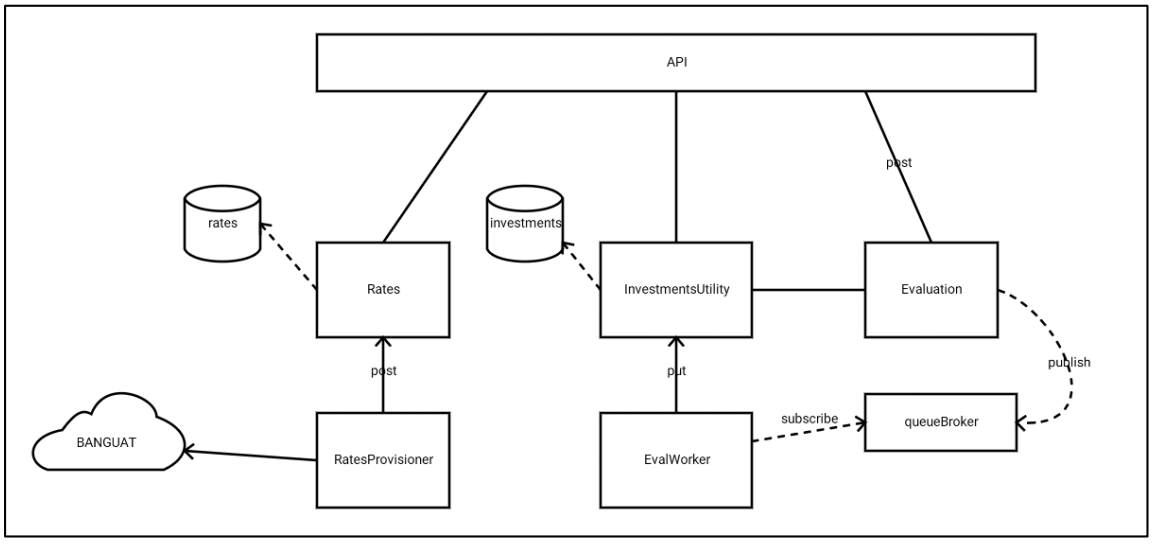
Fuente: elaboración propia.

5.1.2. Diagrama de componentes

Permite conocer la arquitectura externa de un sistema en cuanto a interacciones internas entre componentes. Para el caso de la API a desarrollar, este diagrama permitirá la identificación de los microservicios que compondrán la arquitectura del sistema.

A continuación, se muestra el diagrama de componentes de la arquitectura.

Figura 12. **Arquitectura de microservicios de la API**



Fuente: elaboración propia, empleando DrawExpress

En la siguiente tabla se muestra la descripción de cada uno de los componentes, tanto dependencias de almacenamiento como microservicios.

Tabla XI. **Descripción de componentes de la arquitectura**

Componente	**Descripción**
API Gateway	Servicio que utiliza el patrón de API Gateway para exponer los servicios de la API.
InvestmentsUtility	Servicio encargado de manejar el acceso a los datos persistentes relacionados a la lógica del negocio.
Evaluation	Servicio encargado de crear y obtener evaluaciones.
Queue broker	Servicio interno manejador de colas de mensajes que almacenará evaluaciones pendientes.
EvaluationWorker	Proceso encargado de procesar las evaluaciones pendientes que se encuentran en la cola de mensajes.
Investments	Base de datos persistente que almacenará la información relacionada a la lógica del negocio.
Rates	Servicio encargado de obtener las tasas líderes de interés almacenadas en un registro temporal de tasas.
Rates (Base de datos)	Base de datos que almacenará información temporal sobre tasas de interés.
Rates Polling	Proceso encargado de consumir los servicios web del Banguat para obtener las tasas de interés propuestas por dicha entidad.

Fuente: elaboración propia.

5.1.3. Diagramas de secuencia

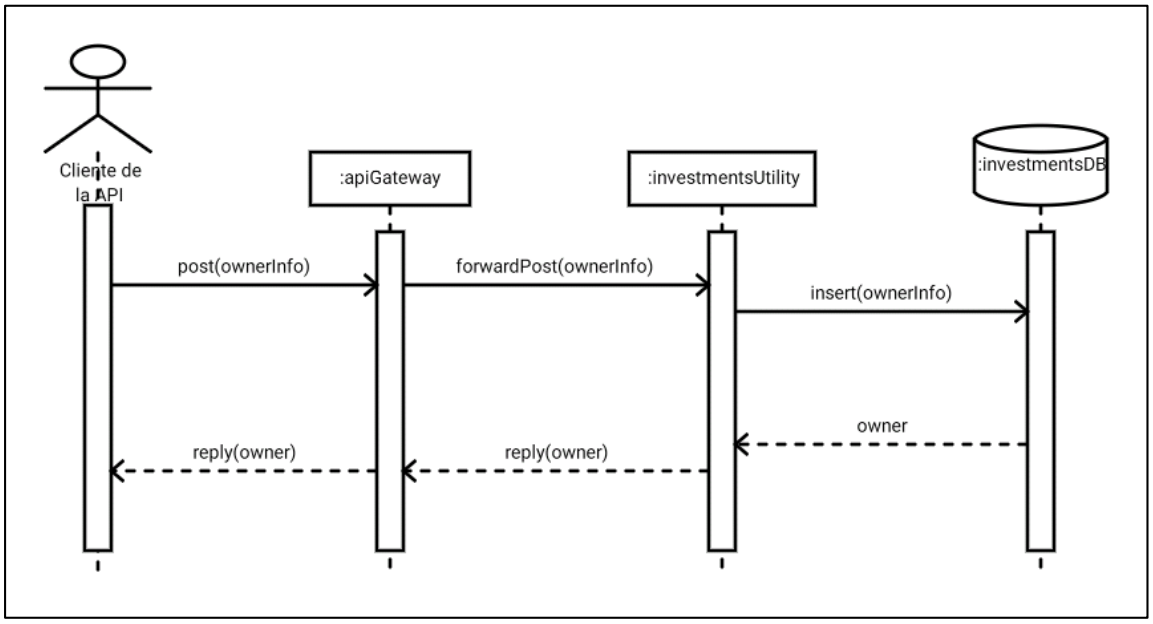
Dentro de UML, los lenguajes de secuencia son utilizados como diagramas de comportamiento, permitiendo observar la lógica de procesos principales relacionados al sistema que se está desarrollando.

A continuación se muestran los principales procesos de la API modelados como diagramas de secuencias.

5.1.3.1. Proceso de registro de usuarios

De acuerdo a los casos de uso, los clientes de la API tendrán la capacidad de registrar nuevos usuarios, los cuales serán dueños de los proyectos cuyo objetivo es conocer su viabilidad financiera. El siguiente diagrama muestra la secuencia del proceso.

Figura 13. **Diagrama de secuencia de registro de usuarios dueños**



Fuente: elaboración propia, empleando DrawExpress.

El proceso toma el siguiente flujo:

- El cliente de la API solicita al servicio API Gateway la inserción de un nuevo usuario por medio de un POST de HTTP.

- El servicio API Gateway redirige su solicitud y la envía al microservicio de utilidades.

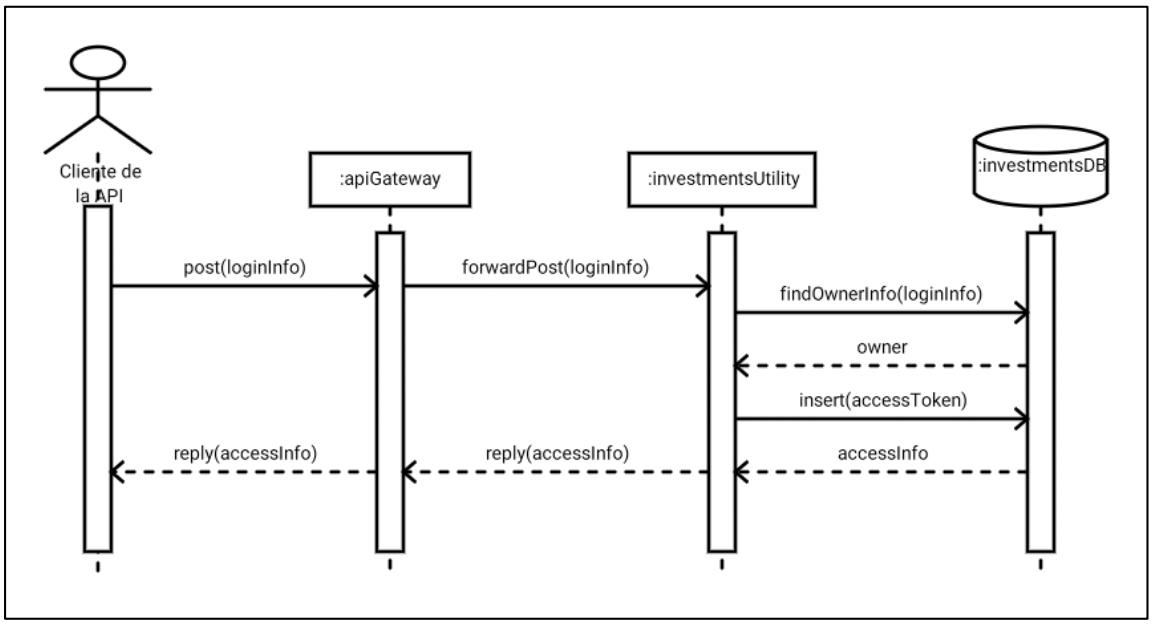
- El servicio de utilidades inserta en la base de datos la información del usuario.

- La respuesta de cada servicio se da en el orden inverso en el que fueron invocados.

5.1.3.2. Proceso de autenticación de usuarios dueños

Para acceder a los recursos proveídos por la API, será necesaria la utilización de un mecanismo de autenticación que permita identificar de manera inequívoca a los usuarios dueños de proyectos financieros. Dicho mecanismo estará regulado por el servicio de utilidades, a través del siguiente proceso.

Figura 14. **Diagrama de secuencia de autenticación de usuarios**



Fuente: elaboración propia, empleando DrawExpress

El proceso de autenticación responde al flujo de actividades definidas a continuación:

- El cliente de la API solicita al servicio API Gateway una autenticación de un usuario dueño en específico por medio de un POST de HTTP.

- El servicio API Gateway redirige la solicitud al servicio de utilidades.

- El servicio de utilidades solicita la información del usuario a la base de datos y valida que coincida con la información proveída a través de la solicitud.

- El servicio de utilidades genera un *token* de acceso para el usuario y lo inserta en la base de datos.

- La respuesta del flujo es la información de inicio de sesión, la cual incluye el *token* generado.

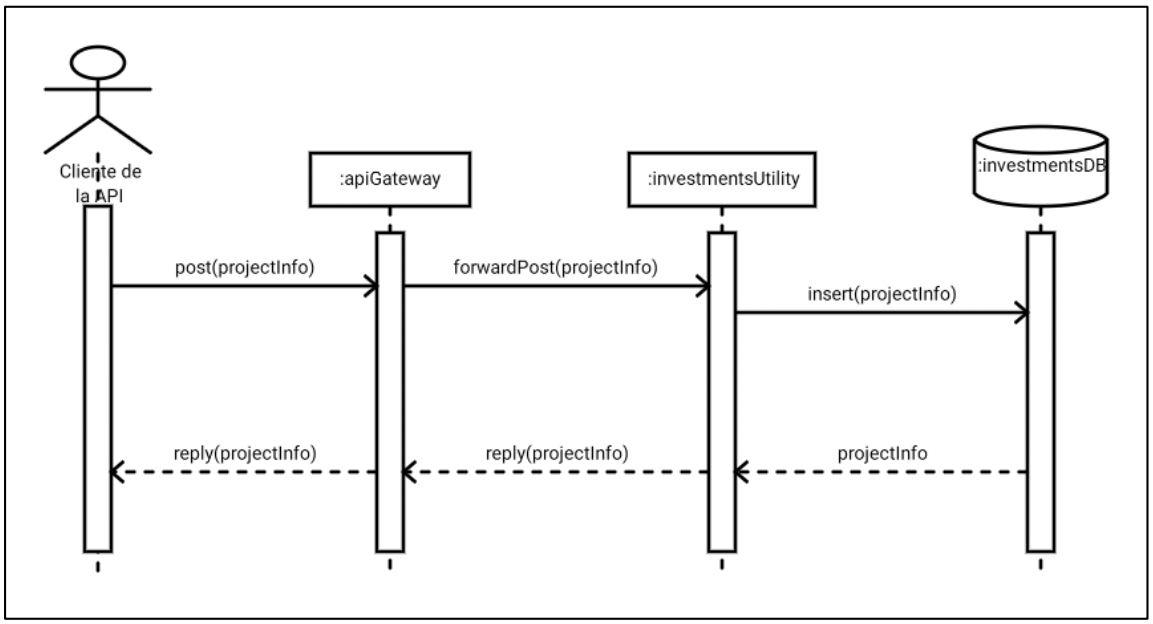
5.1.3.3. Proceso de creación de proyectos

Para cada usuario registrado dentro de la API, la base de la lógica son los proyectos, ya que estos tendrán asociada la información necesaria para la realización de evaluaciones de viabilidad económica.

Un proyecto es un conjunto de movimientos financieros u operaciones, las cuales determinan una proyección de gastos e ingresos futuros de las que dependerá la viabilidad del mismo.

En el siguiente diagrama se muestra el flujo de actividades del proceso de creación de proyectos por medio de un diagrama de secuencias.

Figura 15. **Diagrama de secuencias de creación de proyectos**



Fuente: elaboración propia, empleando DrawExpress

El proceso está compuesto por las siguientes actividades:

- El cliente de la API solicita la creación de un proyecto al servicio API Gateway con la respectiva información sobre el objeto en cuestión.

- El servicio API Gateway redirige la solicitud al servicio de utilidades para la inserción del proyecto.

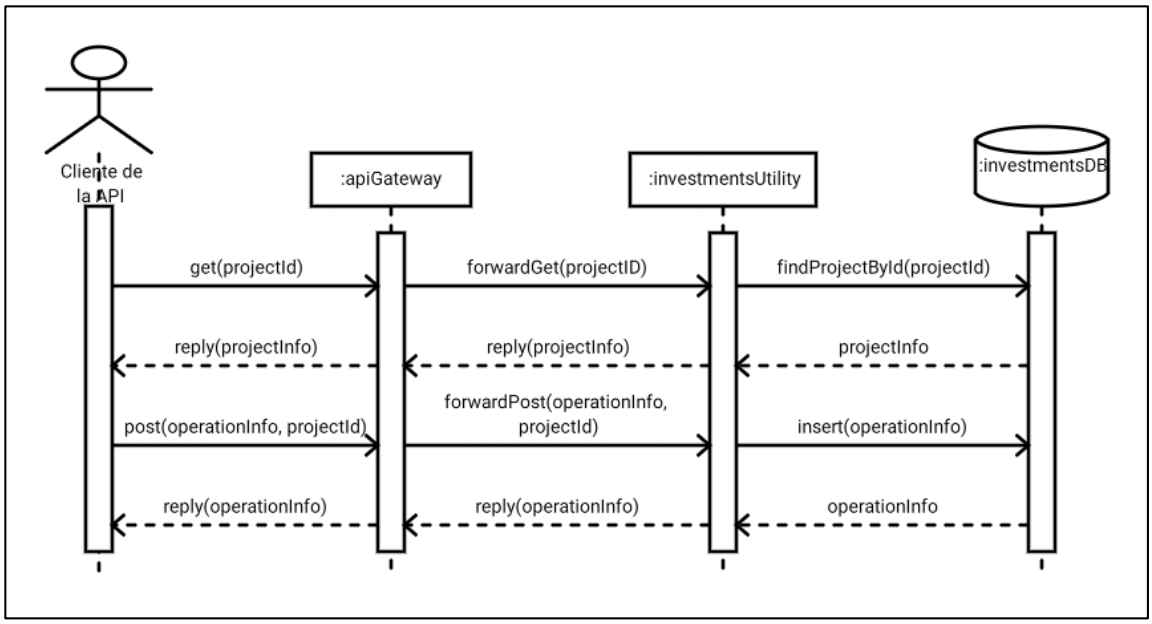
- El servicio de utilidades inserta en la base de datos el proyecto en cuestión.

- La información retornada a los clientes es el nuevo elemento insertado.

5.1.3.4. Proceso de creación de nuevas operaciones

Las operaciones son los elementos de un proyecto que representan un movimiento en un flujo de efectivo. Estas pueden representar ingresos o egresos económicos en un punto específico. A continuación se muestra el diagrama de secuencia para el proceso de creación de las mismas.

Figura 16. **Diagrama de secuencia de creación de nuevas operaciones**



Fuente: elaboración propia, empleando DrawExpress.

El proceso está conformado en las siguientes actividades:

- El cliente de la API solicita la información de un proyecto en específico al servicio API Gateway, el cual, a su vez, lo solicita con el servicio de utilidades que obtiene el proyecto desde la base de datos.

- Con la información del proyecto recibida, el cliente de la API solicita al servicio API Gateway la creación de una operación nueva.

- El servicio API Gateway redirige la solicitud POST de HTTP al servicio de utilidades, el cual se encarga de almacenar la información de la operación insertada.

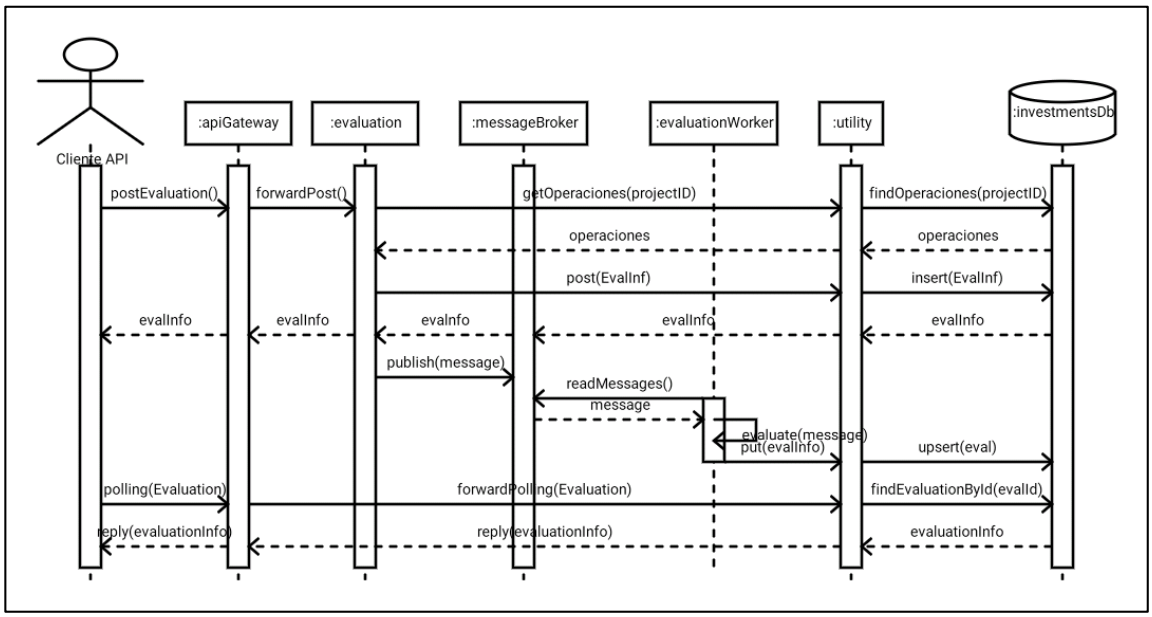
- Existe una confirmación de la nueva operación, la cual viaja en el orden inverso en el que viajo la solicitud de creación.

5.1.3.5. Proceso de evaluación de proyectos

Cada proyecto contará con operaciones financieras que se distribuirán a través del tiempo. Sin embargo, un proyecto sin su evaluación financiera no es de utilidad para quien desea hacer una estimación de la factibilidad del mismo, por ello, es necesario realizar evaluaciones sobre los proyectos basadas en las operaciones financieras ingresadas.

Del proceso de evaluación, los usuarios dueños de proyectos podrán obtener estimaciones sobre sus proyectos con el análisis del valor presente neto y el análisis de beneficio/costo. A continuación se presenta el diagrama de secuencias para el proceso de evaluación de proyectos.

Figura 17. **Diagrama de secuencias del proceso de evaluación**



Fuente: elaboración propia, empleando DrawExpress.

El proceso de evaluación de proyectos es un proceso asíncrono, lo que implica que los clientes de la API no obtendrán el resultado de la misma al instante que lo solicitan. Las actividades que se toman en cuenta en el proceso son:

- El cliente de la API le solicita al servicio API Gateway la evaluación de un proyecto por medio de un POST. Dicha solicitud será redirigida al servicio de evaluaciones.

- El servicio de evaluaciones solicitará al servicio utilitario la información del proyecto a evaluar, con esto se incluye la última evaluación realizada, la fecha de última modificación del proyecto y las evaluaciones.

- El servicio de evaluaciones decidirá realizar o no una nueva evaluación. Si decide realizarla, solicita la inserción de una nueva evaluación al servicio de utilidades.

- El servicio de utilidades responde con la información de una nueva evaluación pendiente de procesar.

- El servicio de evaluaciones inserta, en una cola de mensajes, la información requerida para realizar el cálculo del valor presente neto y el análisis de beneficio/costo.

- Existe un proceso que está suscrito a la cola de mensajes, este lee de manera periódica el ingreso de mensajes nuevos.

- Al encontrar un mensaje, el proceso realiza el cálculo de la evaluación requerida y solicita su inserción al servicio utilitario.

- El cliente de la API que solicitó al principio del proceso la evaluación del proyecto no obtendrá el resultado de manera inmediata, sino hasta que la evaluación haya cambiado de estado. Para ello, el cliente de la API deberá estar consultando la información de la evaluación de manera constante.

5.2. Definición, análisis y diseño de servicios

La arquitectura de microservicios permitirá a la API el encapsulamiento de funciones específicas dentro de servicios atómicos. Por ello, tanto los procesos y la lógica de negocio, deberán ser separados en diversos

microservicios. A continuación se detalla la definición, el análisis y el diseño propuesto para cada uno de ellos.

5.2.1. Servicio de utilidades

La obtención de datos persistentes es uno de los grandes retos en la construcción de interfaces de programación de aplicaciones y servicios, ya que, generalmente, esta estará dirigida por el modelo lógico de datos que se desea acceder y por la tecnología de la misma.

Para proveer los datos de manera homóloga, consistente y transparente, se ha propuesto el desarrollo de un microservicio especializado en la obtención de los datos a partir de las fuentes de datos persistentes.

Este servicio se comunicará con la fuente de datos y será el único medio por el cual la información será obtenida desde los datos persistentes.

5.2.1.1. Características del servicio

Para el desarrollo del servicio de utilidades, se contará con las siguientes características o requerimientos:

- Autenticación y autorización de usuarios y clientes

Cada uno de los clientes del API deberá ser registrado, a su vez, cada vez que un cliente realice una petición a la API, esta deberá ser autenticada y se debe verificar la identidad de quien la realizó.

- Definición de los modelos de datos

Además del modelo de autenticación, se deberá definir el modelo de datos del negocio, reflejando la lógica del negocio en los datos persistentes de manera transparente. Gracias a ello, se tienen los siguientes modelos:

  - Definición del modelo de proyectos.
  - Definición del modelo de operaciones.
  - Definición del registro de modificación de los proyectos.
  - Definición del registro de las últimas evaluaciones realizadas a los proyectos.

- Definición de relaciones entre los modelos de datos

Permitirá crear y conocer las relaciones necesarias entre las entidades del modelo de datos que manejará el microservicio

- Creación de restricciones de acceso a los modelos de datos

Con base en la autenticación brindada, los recursos deberán ser accedidos únicamente por clientes autenticados y dichos recursos deben estar relacionados con el cliente relacionado.

- Creación de *endpoints* adicionales

De ser necesario, se crearán nuevos *endpoints* para la manipulación de los datos relacionados a los modelos.

- Inclusión del nivel 3 de madurez de REST

Implica agregar controles de hipermedia al contenido de cada recurso obtenido.

- Creación de imagen del servicio

Permitirá crear un entregable del micro servicio de forma aislada, el cual se ejecutará por medio de virtualización a nivel de sistema operativo.

5.2.1.2. División de las características

En la tabla XII se muestra la división de las características por actividades de desarrollo. Cada actividad tendrá un identificador, al cual se podrá hacer referencia desde el sistema de gestión de versiones en el momento en el que se desarrolle.

Tabla XII. **Actividades de desarrollo del servicio de utilidades**

Identificador	**Nombre del servicio**	**Descripción**
IU1	InvestmentsUtility	Creación de mecanismo de autenticación.
IU2	InvestmentUtility	Definición del modelo de proyectos.

Continuación de la tabla XII.

IU3	InvestmentUtility	Definición del modelo de operaciones
IU4	InvestmentUtility	Definición del registro de modificación de proyectos
IU5	InvestmentUtility	Definición del registro de últimas evaluaciones
IU6	InvestmentUtility	Definición de relaciones
IU7	InvestmentUtility	Creación de restricciones
IU8	InvestmentUtility	*Endpoints* adicionales
IU9	InvestmentUtility	Adición de links de hipermedia
IU10	InvestmentUtility	Creación de pruebas unitarias
IU11	InvestmentUtility	Creación de imagen del servicio

Fuente: elaboración propia.

5.2.2. Servicio de evaluaciones

Para las evaluaciones es necesario plantear un microservicio especializado, ya que se realizarán operaciones complejas que van más allá del simple CRUD que ofrece el servicio de utilidades.

Las operaciones relacionadas a la ingeniería económica son operaciones matemáticas complejas que en muchas ocasiones requieren un tiempo de procesamiento mayor a los *timeouts* definidos para la obtención de recursos, debido a que cada proyecto puede tener una cantidad ilimitada de operaciones definidas de distinto tipo. Por ello, el servicio de evaluación deberá ser asíncrono, permitiendo que los clientes de la API no esperen más de lo estipulado por cada petición del tipo *request/response.*

El desarrollo de este servicio deberá llevarse a cabo en dos partes:

- Servicio publicador de trabajos
- Proceso subscrito

Cada parte del servicio se deberá realizar de forma separada, como microservicios separados e independientes. Ambos componentes de software deberán llevar el control de colas de mensajes, uno publicando y otro recibiendo.

5.2.2.1. Características del servicio publicador

Para el desarrollo del servicio publicador, se contará con las siguientes características o requerimientos:

- Obtención de peticiones de creación

El servicio deberá obtener todas las peticiones de creación de evaluaciones. No deberá omitir un solo parámetro de creación.

- Obtención de información de proyectos

Para cada evaluación, el servicio deberá obtener todas las operaciones que posee un proyecto, su última fecha de modificación y su última evaluación.

- Creación de nuevas evaluaciones

De ser necesario, se deberá crear una nueva evaluación, cuyo objeto almacenado por el servicio utilitario sea devuelto al cliente, mientras se envía un mensaje a la cola de mensajes con el detalle del proyecto y las operaciones del mismo.

- Omisión de nueva creación y obtención de última evaluación

Si la creación no es necesaria (la última modificación fue realizada antes de la última evaluación), no se incluirá un mensaje en la cola, sino se retornará la última evaluación realizada sobre el proyecto.

5.2.2.2. Características del subscriptor

El subscriptor o *worker* del servicio evaluador será el proceso encargado de extraer de la cola de mensajes el detalle necesario para realizar la evaluación de un proyecto. A continuación se enlistan sus características:

- Lectura de la cola

El *worker* deberá escuchar siempre a un sistema manejador de colas, esperando nuevos mensajes que contendrán los detalles de cada proyecto para crear evaluaciones.

- Ejecución de cálculos

Una vez obtenido el mensaje, el *worker* deberá realizar los cálculos sobre la viabilidad financiera del proyecto, utilizando procedimientos de ingeniería económica.

- Actualización de la evaluación

El *worker* deberá solicitar una modificación en el estado de la evaluación en concreto. Esta modificación es realizada finalmente por el servicio utilitario.

5.2.2.3. División de las características

En la siguiente tabla se muestra la división de las características por actividades de desarrollo. Cada actividad tendrá un identificador, al cual se podrá hacer referencia desde el sistema de gestión de versiones en el momento en el que se desarrolle. Además, el identificador tendrá un prefijo dependiente de la aplicación que se desarrolle para el servicio.

Tabla XIII. **Actividades de desarrollo del servicio de evaluaciones**

Identificador	Nombre de la aplicación	Descripción
EV1	Evaluation	Creación y definición de arquitectura interna.
EV2	Evaluation	Mecanismo de obtención de última evaluación.
EV3	Evaluation	Mecanismo de creación de evaluaciones.
EV4	Evaluation	Creación de mensajes para la cola
EW1	EvaluationWorker	Creación de subscriptor de la cola.
EW2	EvaluationWorker	Implementación de lógica de ingeniería económica.
EW3	EvaluationWorker	Actualización de evaluaciones.
EW4	EvaluationWorker	Creación de la imagen del servicio.

Fuente: elaboración propia.

5.2.3. API Gateway

En la arquitectura de microservicios, cada servicio expone una ruta a través de la cual es accesible por cualquier cliente que desee consumirlo. Sin embargo, esto supone problemas para los clientes del sistema, puesto que se deben conocer todas las rutas de cada servicio a consumir. La solución se plantea con el patrón de API Gateway que propone la creación de un servicio que conozca todas las rutas de los demás servicios a los que los clientes accederán de manera directa, sin necesidad de conocer la ubicación de cada uno.

El patrón en sí brinda un nivel de abstracción para los clientes del sistema, sin embargo, coloca un salto más en la arquitectura, por lo que su implementación debe ser cuidadosa y ligera.

5.2.3.1. Características para el desarrollo del servicio API Gateway

Para el desarrollo del servicio API Gateway se tienen las siguientes características:

- Manejo de rutas por servicio/recurso a consumir

El servicio API Gateway deberá identificar a que servicio o recurso se pretende acceder desde un cliente, respetando los verbos de HTTP por los que se consuman.

- Manejo de verbos

El servicio en sí deberá realizar tareas de traducción de peticiones basado en los verbos de HTTP, debido a que cada verbo invoca un comportamiento distinto dentro de cada microservicio.

- Estandarización de errores

Cada microservicio posee una forma específica para mostrar los errores, sin embargo, para dar abstracción a los clientes, el servicio API Gateway deberá estandarizar la manera en la que cada error se mostrará.

- Exposición gráfica de rutas

A través de una aplicación web sencilla, el servicio API Gateway, deberá mostrar qué rutas expone a los clientes, de qué verbos dispone para cada servicio/recurso y cómo debe consumirse cada uno.

5.2.3.2. División de características

En la tabla XIV se muestra la división de las características por actividades de desarrollo. Cada actividad tendrá un identificador, al cual se podrá hacer referencia desde el sistema de gestión de versiones en el momento en el que se desarrolle.

Tabla XIV. **Actividades de desarrollo del servicio API Gateway**

Identificador	Nombre de la aplicación	Descripción
GW1	API Gateway	Manejo de rutas por servicio
GW2	API Gateway	Manejo de verbos HTTP
GW3	API Gateway	Estandarización de respuestas
GW4	API Gateway	Exposición gráfica de rutas.

Fuente: elaboración propia.

5.2.4. Servicio de obtención de tasas de interés

Para obtener información sobre las diferentes tasas de interés que maneja Banguat, se necesita de una API que obtenga estos datos actualizados periódicamente, que los almacene y tenga un historial de tasas y que provea una API que despliegue esta información.

5.2.4.1. Características de servicio de obtención de tasas

Para el desarrollo del servicio de operaciones CRUD se tienen las siguientes características:

- Manejo de rutas por recurso a consumir

La API Rates cuenta con distintas rutas de acceso a la información de las tasas de cambio, teniendo en cuenta que el servicio devolverá la información más reciente a la que se tenga acceso por defecto.

- Manejo de verbos

El servicio deberá realizar tareas de traducción de peticiones basado en los verbos de HTTP, debido a que cada verbo invoca un comportamiento distinto dentro de cada microservicio.

5.2.4.2. Características de servicio de obtención de datos de Banguat

Para el desarrollo del servicio de obtención de datos de Banguat se tienen las siguientes características:

- Servicio de *polling*

Este servicio es el encargado de consumir los recursos obtenidos por medio del *web service* del Banguat, el cual obtiene la información y la almacena en la base de datos local.

- Proceso *cron*

Este es el encargado de ejecutar periódicamente el servicio de *polling*, para obtener la información más actualizada de las diferentes tasas de interés ejecutándose una vez al día.

5.2.4.3. División de características

En la siguiente tabla se muestra la división de las características por actividades de desarrollo. Cada actividad tendrá un identificador, al cual se podrá hacer referencia desde el sistema de gestión de versiones en el momento en el que se desarrolle.

Tabla XV. **Actividades de desarrollo de la API Rates**

Identificador	Nombre de la aplicación	Descripción
RA1	Rates	Operaciones de consumo de datos.
RA2	Rates	Consumo de *web services* del Banguat.
RA3	Rates	Creación de proceso Cron.
RA4	Rates	Creación de la imagen Docker.

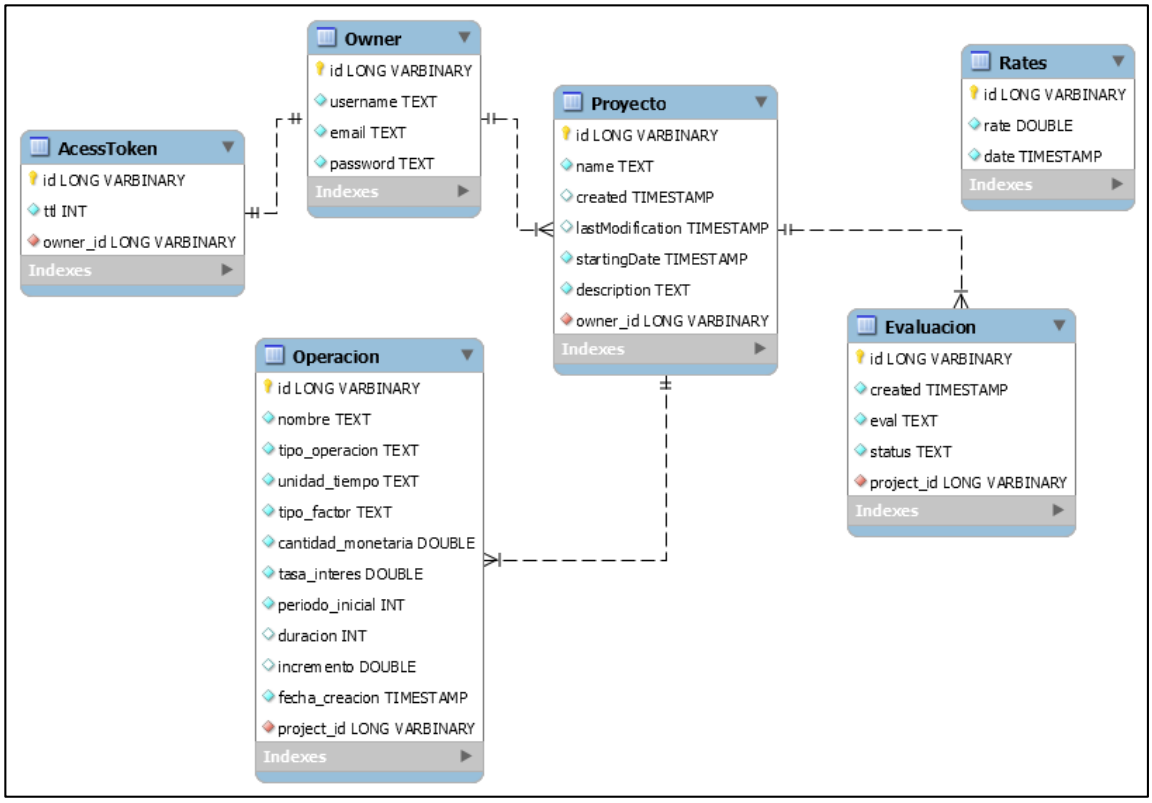
Fuente: elaboración propia.

5.3. Definición de modelos de datos

Una de las principales fases en el desarrollo de microservicios es la de la definición de modelos de datos, los cuales están atados directamente a los procesos propios de cada servicio.

Para la API a desarrollar, el modelo de datos que se adecúa de mejor forma a la lógica de negocio es el mostrado a continuación.

Figura 18. **Modelo lógico de datos**



Fuente: elaboración propia, empleando MySQL Workbench.

El modelo propuesto representa entidades que interactúan en cada proceso realizado por la API. Por conveniencia y entendimiento, se representa como un modelo relacional de base de datos. Sin embargo, en la implementación este puede variar.

Las entidades del modelo se describen a continuación.

Tabla XVI. **Descripción de las entidades del modelo**

Tabla	**Descripción**
AccessToken	Encargada del almacenamiento de las claves de acceso a los recursos de cada usuario dueño de proyecto. La información almacenada permite la autorización y autenticación de acciones referentes a los recursos de la API.
Owner	Almacena la información de cada usuario dueño dentro del sistema. Los usuarios almacenados son a nivel de negocio, no de acceso a la API.
Proyecto	Representa la parte central del negocio, ya que el objetivo del mismo es el análisis financiero de proyectos. Almacena la información referente a cada proyecto.
Operación	Las operaciones están contenidas dentro de los proyectos e indican el movimiento de activos monetarios cuyo valor es fundamental para el análisis financiero de cada proyecto. Almacena datos referentes a cada operación, pues es necesario conocer varios aspectos, tales como el tipo de operación, la tasa de interés, entre otros.
Evaluación	Todo análisis financiero de un proyecto deberá registrarse de manera histórica, por lo que la entidad de evaluaciones será la encargada de modelar la estructura de almacenamiento de información de cada evaluación realizada.

Continuación de la tabla XVI.

Rates	Para proveer de datos sobre la tasa líder de política económica en Guatemala, será necesario almacenar un caché de la información obtenida del Banco de Guatemala, por lo que será necesaria una entidad para ello. Esta entidad no estará relacionada con las demás.

Fuente: elaboración propia.

5.4. Definición de elementos tecnológicos

Al definir el contexto de la API y de los microservicios que la componen, es posible determinar las herramientas tecnológicas que se utilizarán para el desarrollo de los mismos. Debido a las características identificadas para cada microservicio y sus dependencias, es de suma importancia la correcta selección de plataformas, lenguajes, sistemas de gestión de bases de datos, entre otras.

5.4.1. Definición de lenguajes de programación y plataformas

Cada plataforma de ejecución y lenguaje de programación juega un papel determinante en el desarrollo de software, ya que las características, ventajas y desventajas afectan directamente el resultado final, el cual es un producto de software que provee un servicio o cumple con una tarea.

Un lenguaje de programación, por definición, es un lenguaje formal cuyo objetivo es la realización de procesos específicos realizados por una computadora.

Existen lenguajes de programación cuya existencia está destinada a un paradigma o estilo de programación, tal es el caso de los lenguajes utilizados para la creación de programas, aplicaciones y servicios basados en la web. En el caso de los microservicios, se buscan lenguajes cuya interpretación y ejecución garanticen alta disponibilidad, atomicidad, agnosticismo y un control de eventos.

La arquitectura de microservicios permite la utilización de múltiples lenguajes de programación, sin embargo, esto no quiere decir que sea forzosamente necesario tener todos los servicios en diferentes lenguajes, sino basar el código fuente de cada uno en un lenguaje que cubra sus necesidades.

Las plataformas de ejecución muchas veces influyen en el comportamiento de los lenguajes de programación a utilizar, ya que cada una es dependiente del sistema en el que se ejecuta, por lo que provee recursos diferentes de acuerdo a cada implementación.

Para llevar a cabo el desarrollo de los microservicios, se ha seleccionado la siguiente combinación entre plataforma y lenguaje de programación.

5.4.1.1. JavaScript

Es un lenguaje de programación caracterizado por ser liviano, dinámico, débilmente tipificado, interpretado y con funciones de primera clase. Es altamente conocido por brindar funcionalidad de *scripting* a páginas web para mejorar la interacción entre usuario y sitio.

Se basa en prototipos, lo que permite utilizar múltiples paradigmas de programación, como programación orientada a eventos, imperativa, orientada a objetos y funcional. Su sintaxis está basada en C++ y Java, sin embargo, no quiere decir que sea la implementación interpretada de este último.

Al ser multiparadigma, JavaScript puede funcionar como un lenguaje procedural o uno orientado a objetos. Esto último es posible agregando métodos y atributos de manera dinámica en tiempo de ejecución a objetos vacíos o predefinidos, lo contrario a la declaración estática de clases tradicional en la mayoría de lenguajes de programación orientados a objetos.

El poseer funciones de primera clase permite a JavaScript tratar a las mismas como objetos en tiempo de ejecución, por lo que es posible pasar una función como argumento de otra, característica que permite la ejecución asíncrona de funciones.

5.4.1.1.1. Motores JavaScript

El comportamiento de JavaScript está dictado estrictamente por la implementación de sus intérpretes. Estos son llamados JavaScript *engines* o motores JavaScript.

Cada motor JavaScript está descrito como una biblioteca del sistema, la cual permite instanciar una máquina virtual cuyo objetivo es dotar de memoria a una aplicación escrita en JavaScript y traducir el código fuente de la misma a código máquina.

Algunos de los motores JavaScript más conocidos son: V8 de Google Chrome, Chackra, Rinho, Spidermonkey, entre otros.

5.4.1.2. Node.js

Es un entorno de ejecución multiplataforma para el desarrollo de aplicaciones web *server-side*. Los programas ejecutados por Node.js están escritos en JavaScript, por lo que permiten la ejecución de tareas asíncronas en un solo hilo de ejecución. Internamente, implementan el JavaScript *engine* V8 de Google Chrome.

Está diseñado para construir aplicaciones de red altamente escalables, está inspirado en la Event Machine del lenguaje Ruby y Twisted de Python, con la diferencia que presenta el ciclo de eventos como parte del lenguaje y no como un *framework* más.

La unidad básica de todo programa ejecutado en Node.js son los módulos, los cuales son objetos que son cargados una única vez en la memoria principal de V8, por lo que si se tiene un *script* que se ejecute antes de la exportación del módulo, este se ejecutará una única vez.

A diferencia de la ejecución de aplicaciones web con lenguajes como PHP y Python que inician una instancia del servidor con cada petición recibida, las aplicaciones de Node.js generalmente contienen un servidor web (el cual es inicializado como un objeto dentro de la lógica del programa) que ejecutará una única instancia sin importar la cantidad de solicitudes que reciba.

Otra de las características de Node.js es el manejo de *sockets* TCP y de *sockets* web, permitiendo establecer conexiones entre servidor y cliente más duraderas que con solo las peticiones de HTTP.

5.4.2. *Frameworks* a utilizar en el desarrollo

La implementación de *frameworks* en el desarrollo de software permite extender las características de un lenguaje de programación, una plataforma de ejecución o ambos. En el caso de la API a desarrollar, se utilizarán los siguientes *frameworks* de Node.js.

5.4.2.1. Loopback

Es un *framework* ODM y ORM que permite la exposición de los modelos generados a partir de una fuente de datos como interfaces de programación. Se basa en Express.js y permite mapear datos a objetos desde conexiones con fuentes de datos RDBMS y NO-SQL, como DB2, MySQL, MongoDB, Redis, Oracle, Postgres, entre otros.

Al basarse en Express.js, Loopback posee características de registro de objetos durante diversas fases de la ejecución de la API por medio de *middlewares*. Además de ello, permite al desarrollador controlar el comportamiento de los modelos de datos utilizando *hooks* y métodos remotos, agregando lógica de negocio antes, durante y después del acceso a la fuente de datos.

Por cada modelo creado, por defecto, Loopback crea los métodos para crear, leer, modificar y eliminar instancias por medio de los verbos POST, GET, PUT y DELETE de HTTP respectivamente, por lo que expone un API RESTful de nivel de madurez 2 del modelo de Richardson.

5.4.2.2. Mongoose

Es un *framework* de modelado de objetos de MongoDB diseñada para trabajar de forma asíncrona, por medio de esquemas, que incluye la manipulación de datos almacenados, creación de modelos, validación, casteo y facilita la construcción de consultas sobre la base de datos, sin necesidad de realizar configuraciones complejas o que requieran de una gran cantidad de trabajo.

Es uno de los *frameworks* más populares y completos que existen, gracias a su gama de herramientas disponibles, siendo posible combinarlo con alguna herramienta de ruteo para crear los métodos básicos para configurar los verbos HTTP, al igual que Loopback. Traduce la información que se encuentra almacenada en la base de datos a objetos JavaScript y viceversa.

5.4.2.3. Joi

Es una biblioteca utilizada para la descripción y validación de objetos JavaScript, por medio de esquemas similares a los utilizados por Mongoose. Puede validar datos desde valores escalares como cadenas de texto, números, hasta datos más complejos como expresiones regulares, objetos anidados y arreglos.

Joi permite que la información que ingresa al sistema tenga la estructura correcta. Verificar este tipo de entradas ayuda a que el programa que se esté desarrollando sea más estable y fiable.

5.4.2.4. Hapi JS

Es un *framework* web para Node.js enfocado específicamente en crear lógica de aplicación reusable. Hapi utiliza el manejo de rutas de forma similar a otros *frameworks*, con la diferencia que las rutas en realidad son objetos altamente configurables que permiten manejar las peticiones de diversas maneras dependiendo de la configuración que tengan. También cuenta con una gran cantidad de *plugins* que enriquecen el comportamiento del *framework*.

Además de soportar los diferentes tipos de contenido de hipermedia, Hapi soporta la conexión de clientes por medio de *streaming* utilizando módulos que extienden la funcionalidad del *framework,* como Wreck.

Hapi posee su propio *framework* para la creación de pruebas, Lab, el cual permite verificar el comportamiento del software creado en un ambiente controlado por experimentos.

5.4.3. Sistema de gestión de base de datos MongoDB

Algunos de los microservicios planteados requieren de persistencia de la información que utilizan para su operación.

Para lograr abstraer la lógica de almacenamiento físico de datos persistentes, se debe utilizar un sistema de gestión de base de datos que se acople a las necesidades de los microservicios que lo requieran. Sin embargo, para la selección del mismo deben considerarse características como: atomicidad, persistencia, alta disponibilidad, escalabilidad, integridad, entre otras.

MongoDB es un sistema de gestión de base de datos NO-SQL documental, el cual sustituye el concepto de almacenamiento lógico de datos at través de tablas y relaciones por documentos JSON con esquema dinámico. Físicamente, los documentos son almacenados en un formato binario de JSON llamado BSON.

Entre las principales características de MongoDB se encuentran:

- Indexado de cualquier campo, sin importar el tipo de dato que este posea.

- Escalabilidad y alta disponibilidad, por medio de configuraciones flexibles para replicación y balanceo de carga respectivamente.

- Posee un *framework* para realizar agregaciones, lo que simplifica la escritura de consultas complejas que requieran de operaciones de agregado por medio de una tubería de instrucciones.

- Utiliza JavaScript de lado servidor para manejo de los datos, lo que permite una interacción más natural entre aplicaciones debido a que utiliza métodos para realizar las operaciones.

Gracias a las características de MongoDB y por poseer soporte nativo para JavaScript, este sistema de gestión de base de datos será utilizado en el desarrollo de los microservicios que requieran persistencia de datos.

5.4.4. Sistema de gestión de colas de mensajes RabbitMQ

Las colas de mensajes permiten la interacción remota entre múltiples procesos de forma asíncrona. Uno de los protocolos más utilizados para la interacción por medio de colas de mensajes es el AMQP, el cual es un protocolo estándar abierto de la capa de aplicación del modelo OSI de redes.

RabbitMQ es un sistema intermediario de mensajes de código libre que utiliza AMQP como protocolo de comunicación. Permite separar las los diferentes mensajes que se publican en su servicio en colas específicas, las cuales son escuchadas o leídas por procesos *workers*. Posee bibliotecas cliente para la mayoría de los lenguajes de programación más utilizados y dentro de sus características es posible encontrar:

- Balanceo de carga para el envío de mensajes.
- *Broadcast* de mensajes a varios suscriptores.
- Permite la utilización de acuse de recibido para los mensajes en las diferentes colas existentes.
- Está construido sobre un *framework* de *clustering* y *failover* llamado Open Telecom Platform.

Debido a que el servicio de evaluaciones es un servicio asíncrono planteado con el patrón *publish/subscriber*, se ha seleccionado RabbitMQ como intermediario de mensajes debido a sus características de integridad de envío y recepción de mensajes y de integración con diferentes lenguajes de programación.

5.4.5. Plataforma de ejecución de los microservicios

Para aislar los procesos que dan vida a los microservicios planteados y reducir la cantidad de instancias de servidores o máquinas virtuales, la API con arquitectura de microservicios será entregada por medio de contenedores.

Los contenedores son un set de operaciones estándares que ejecutan un proceso aislado en un ambiente controlado de ejecución. Para ejecutar un contenedor, previamente se debe poseer una imagen del proceso que se desea ejecutar, la cual es construida en formato binario a partir de paquetes base y el código ejecutable o código fuente de la aplicación que se ejecutará dentro del contenedor.

5.4.5.1. Docker

Para llevar a cabo este objetivo, se utilizará Docker como demonio controlador de contenedores.

Docker es un motor de código abierto que automatiza el proceso de entrega de aplicaciones y gestiona ambientes aislados y controlados utilizando contenedores. Está diseñado para proveer un ambiente ligero y rápido en el que se ejecuta el código fuente de múltiples aplicaciones como si estas estuviesen aisladas.

Posee una arquitectura de cliente servidor, en el que el cliente es utilizado para la creación de imágenes y la ejecución de contenedores desde una terminal de comandos, mientras que el servidor se encarga de procesar las solicitudes y manejar el demonio que controla el motor de contenedores.

La utilización de Docker como mecanismo de virtualización es conocida como virtualización a nivel de kernel, ya que, internamente, los contenedores comparten el kernel del equipo anfitrión que ejecuta el motor, evitando que cada servicio o aplicación consuma más recursos de los necesarios al momento de implementarse una virtualización tradicional.

El aislamiento de contenedores es completo, ya que cada uno poseerá su propio directorio raíz, paquetes, listado de *hostnames*, dirección IP interna, puerto expuesto, entre otros.

Docker puede ser ejecutado únicamente en un equipo anfitrión con kernel Linux con arquitectura x64, lo que limita su uso únicamente a distribuciones del sistema operativo GNU/Linux de 64 bits.

5.5. Gestión de la configuración

Es el proceso por medio del que todos los artefactos relevantes para un sistema de software y sus relaciones, se almacenan, recuperan, identifican de manera única y se modifican, llevando el control sobre el historial evolutivo de sistema como tal.

Define una serie de pasos que deberán tomarse en cuenta para el desarrollo de un nuevo producto de software, planteando buenas prácticas para el control de los cambios a nivel de desarrollo, como para la entrega de los artefactos.

5.5.1. Sistema de control de versiones

Conocido también como control de fuentes, es un mecanismo para mantener múltiples revisiones de un artefacto, sea de código fuente, o de archivos de configuración, cuyos cambios serán registrados a través del tiempo.

Los objetivos principales en la utilización de un sistema de control de versiones son los siguientes:

- Almacenar de manera persistente e histórica los artefactos que componen o configuran un producto de software.

- Registrar mediante revisiones y versiones la historia del cambio de un producto de software.

- Proveer un flujo de trabajo.

- Centralizar el trabajo de equipos de múltiples personas.

Existen dos tipos de sistemas de control de versiones:

- Centralizados: almacenan todo el código fuente en un solo servidor de control de versiones, los desarrolladores obtienen únicamente los archivos que modificarán. Un ejemplo de un sistema centralizado es SVN.

- Distribuidos: distribuyen una copia completa del repositorio central, lo que permite a los desarrolladores realizar cambios en más de un archivo. Entre los sistemas distribuidos se encuentran Git y Mercurial.

5.5.1.1. Git

Es un sistema de control de versiones distribuido que provee una serie de operaciones y características que lo hacen ser uno de los sistemas de control de versiones preferidos para la creación de software por parte de equipos de trabajo y comunidades de software libre.

Está enfocado principalmente en la agilidad, la integridad de los datos almacenados y en el soporte de flujos de trabajo no lineales.

Dentro de las principales características de Git se encuentran la facilidad del trabajo por medio de ramas o *branches*, las cuales registran cambios sin afectar la fuente original hasta que se realiza un *merge* para incluir los mismos.

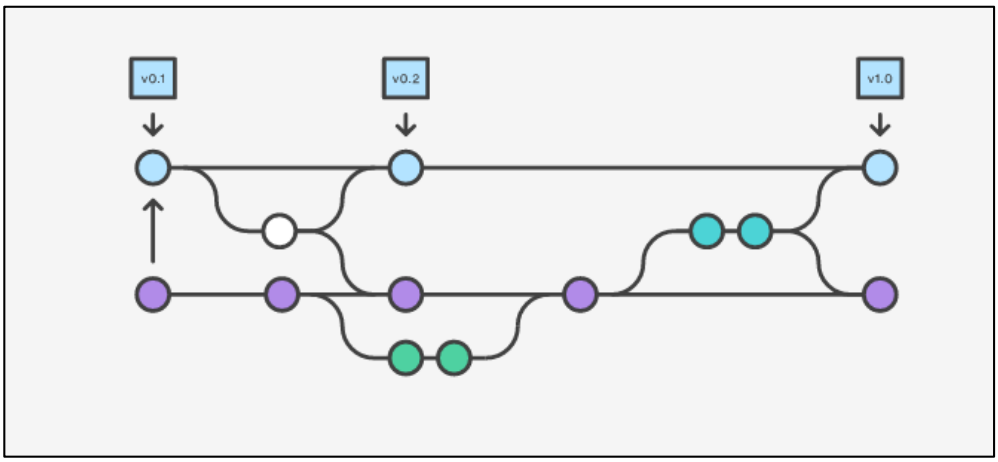
5.5.1.2. Gitflow

Es un flujo de trabajo que define un modelo estricto manejo de ramas o *branches* diseñado alrededor de la liberación de un proyecto de software.

Utiliza un repositorio centralizado como medio de comunicación entre los distintos desarrolladores, los cuales trabajan de manera local con un clon o copia del mismo.

Gitflow exige la existencia de dos *branches* históricas obligatorias: Master y Development. La primera contendrá la historia de los *releases* estables del programa, sobre ella se realizarán los *tags* o etiquetas para identificar las distintas versiones. La segunda es la encargada de almacenar la historia estable e inestable del programa, hacia ella, los desarrolladores incluirán sus cambios hasta que se realice un *release*.

Figura 19. **Flujo de trabajo de Gitflow**



Fuente: ATLASSIAN. *Gitflow workflow.* https://es.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. Consulta: 10 de enero de 2016.

A continuación se describe el flujo de trabajo normal que se debe seguir utilizando Gitflow:

- Las ramas Master y Develop contienen toda la historia del desarrollo del producto de software.

- Se solicita la creación de nuevas características para el programa, por lo que un desarrollador crea una nueva rama, llamada *feature branch* desde la rama Develop.

- El desarrollador trabaja sobre su rama los cambios solicitados.

- Los cambios son sometidos a revisión de código por medio de un *pull request* antes que se efectúe un *merge* en la rama Develop.

- La rama es mezclada por medio de un *merge* en la rama Develop, adicionalmente se crea una rama de *release* conocida como *release branch*.

- Si la rama de *release* funciona de manera esperada, se realiza un *merge* tanto en la rama Develop como en Master, posterior a ello se crea una nueva etiqueta indicando la nueva versión del producto.

Para el desarrollo de la API se utilizará Gitflow como flujo de trabajo para el desarrollo de todos los microservicios involucrados.

6. DOCUMENTACIÓN DE LA UTILIZACIÓN DE LA API

La API desarrollada puede ser accedida a través del protocolo HTTP, si se desea probar el consumo de la misma es posible realizarse con clientes HTTP como el programa PostMan.

El sistema permite a los clientes de la API la creación de usuarios, los cuales serán dueños de los proyectos a los que se le harán las evaluaciones financieras respectivas. Cada proyecto estará compuesto por operaciones, las cuales serán de ingreso o egreso.

La representación de los datos obtenidos de la API posee formato JSON, por lo que el tipo de contenido de cada solicitud deberá ser de aplicación JSON.

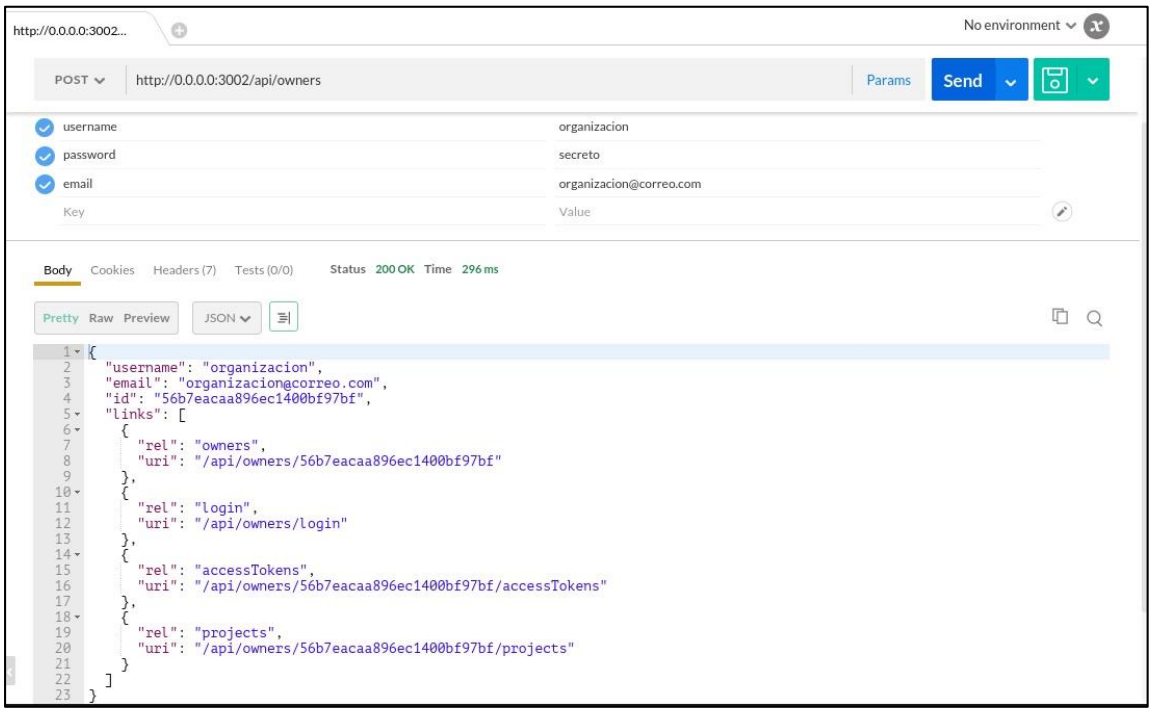
A continuación se muestra una guía para la utilización de la API.

6.1. Acceso a usuarios y clientes de la API

Para crear proyectos es necesario crear usuarios dueños u *owners*. La API provee una ruta especial para ello.

Se necesita tener un usuario, el cual requiere de un nombre de usuario, *username*, de una contraseña, *password* y de un correo electrónico, *email*. La solicitud se realiza a http://api.lmjyo.org/api/owners.

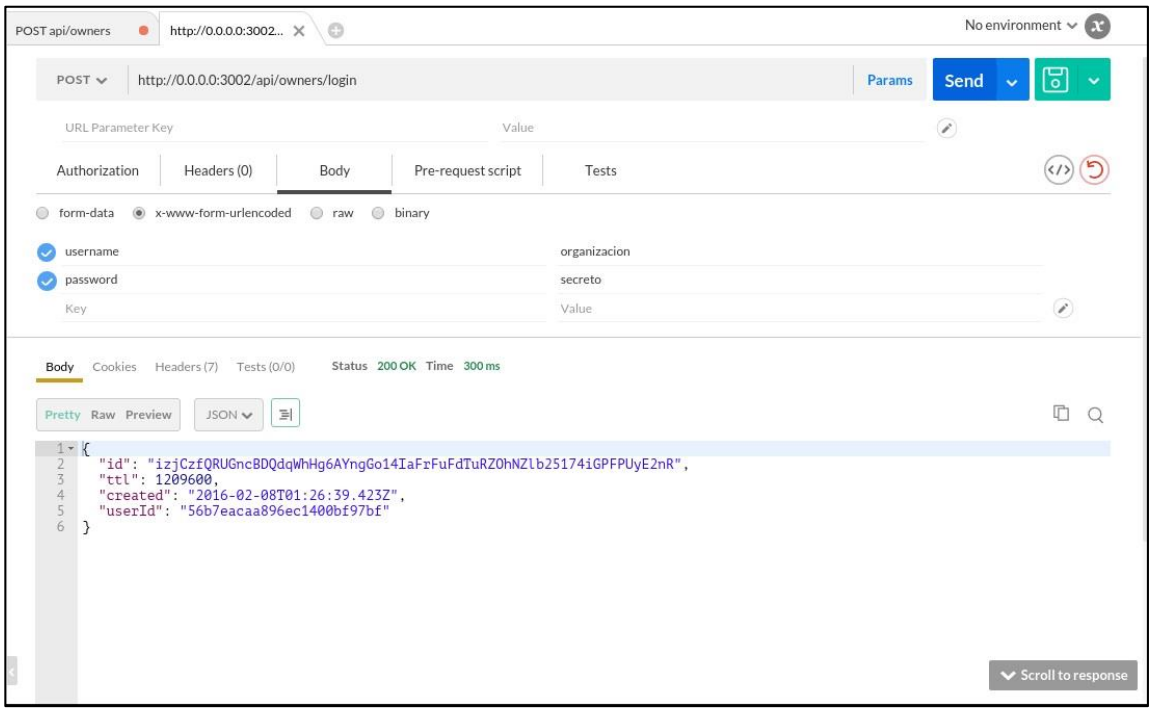
Figura 20. **Solicitud POST para la creación de un usuario**



Fuente: elaboración propia, empleando PostMan.

Una vez el usuario dueño es creado, puede generar una clave de acceso o autorización. En el contexto de la API, el cliente realiza un *login* para el usuario dueño por medio de una solicitud POST a http://api.lmjyo.org /api/owners/login.

Figura 21. **Solicitud POST para la autenticación del usuario al API**



Fuente: elaboración propia, empleando PostMan.

Una vez autenticado, el usuario obtiene un *token* de autorización, el cual cada cliente de la API debe manejar como un encabezado HTTP de autorización.

Figura 22. ***Header* de autenticación**

Fuente: elaboración propia, empleando PostMan.

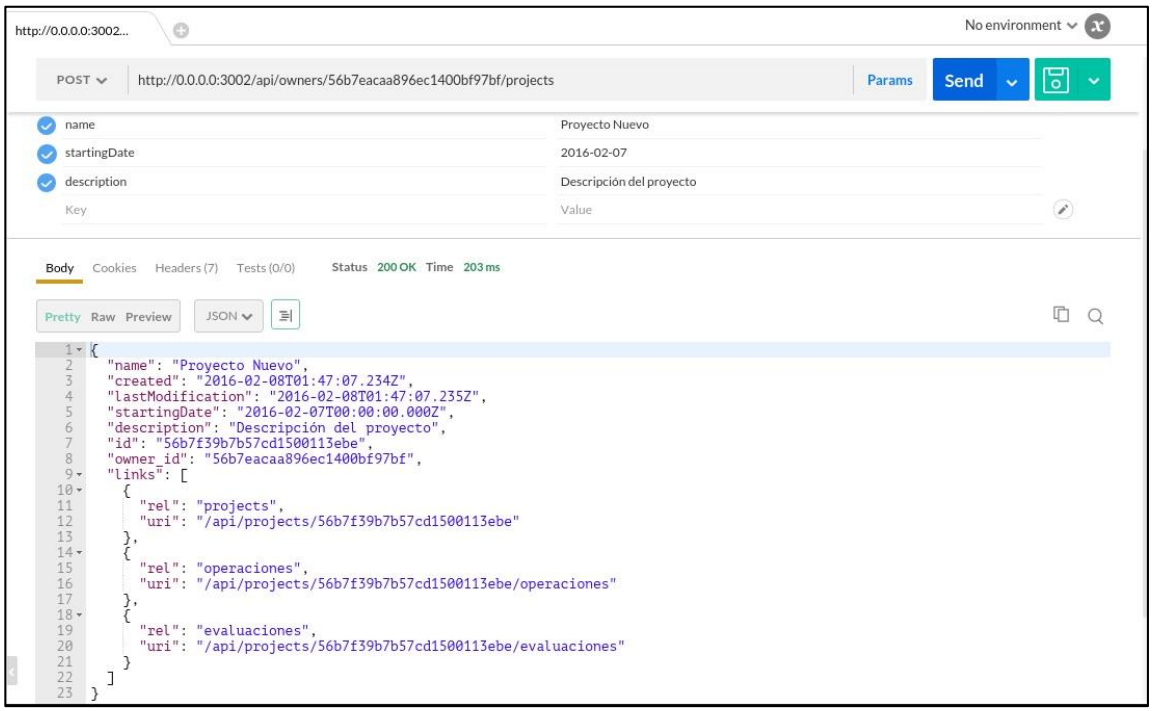
Para cada solicitud que realice el cliente sobre cualquier recurso del usuario autenticado, se deberá incluir este encabezado.

6.2. Creación, obtención, modificación y eliminación de proyectos

Cada usuario podrá acceder a los recursos que le corresponden. El principal activo dentro del sistema para un usuario es el proyecto. Los usuarios pueden crear proyectos, modificarlos, obtenerlos y eliminarlos siempre y cuando sean de su propiedad.

Se necesita de un nombre, una fecha de inicio del proyecto y una descripción del mismo.

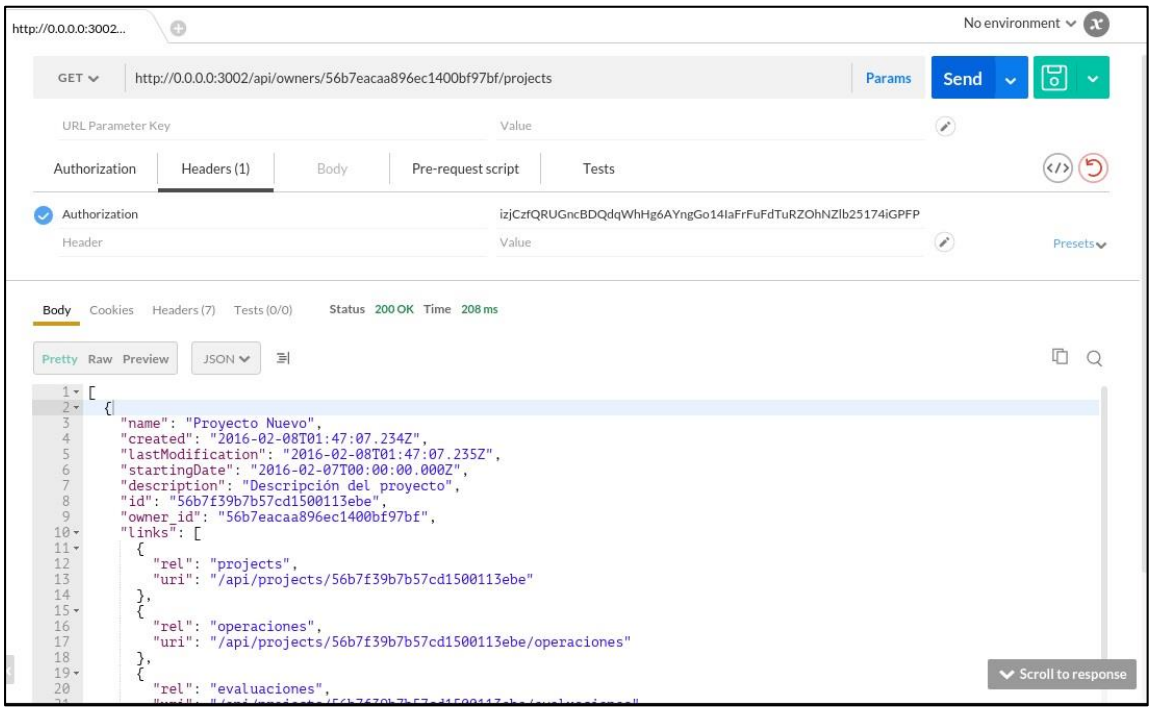
Figura 23. **Solicitud POST para la creación de un proyecto**



Fuente: elaboración propia, empleando PostMan.

El usuario dueño puede acceder a todos los proyectos que posee por medio de una solicitud GET a la ruta http://api.lmjyo.org/api/owners/*id*/projects. Incluirá todos los proyectos que el usuario identificado con el *id* especificado en la ruta ha creado.

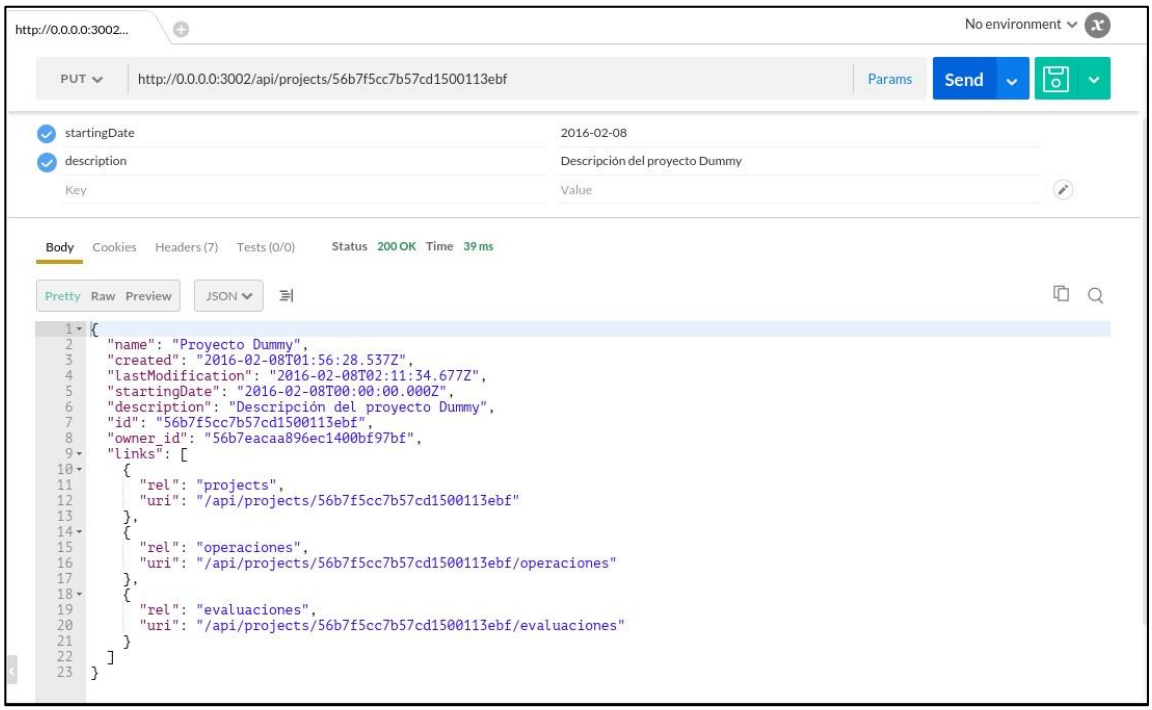
Figura 24. **Solicitud GET para obtener todos los proyectos**



Fuente: elaboración propia, empleando PostMan.

Para modificar un proyecto, simplemente debe seguirse uno de los enlaces proveídos al momento de obtener los proyectos para acceder a la ruta http://api.lmjyo.org/api/projects/*id*. Los usuarios podrán realizar modificaciones al proyecto por medio de una solicitud PUT.

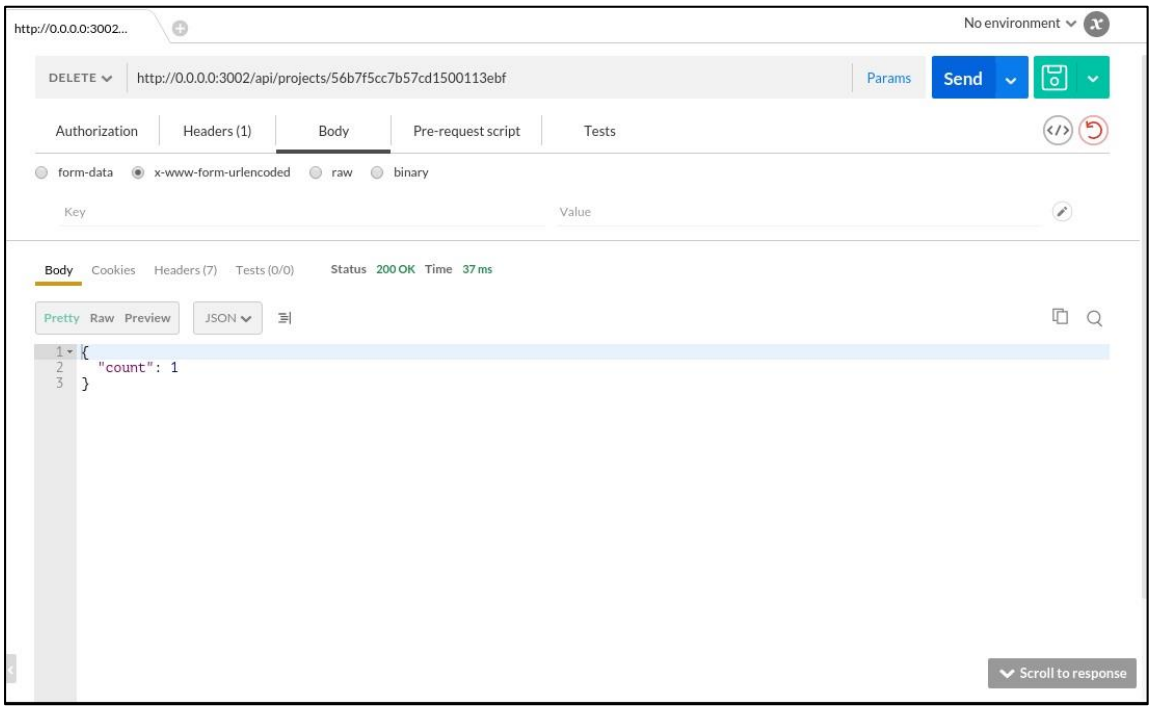
Figura 25. **Solicitud PUT para actualizar un proyecto**



Fuente: elaboración propia, empleando PostMan.

Los usuarios dueños también podrán eliminar los proyectos que han creado por medio del método HTTP DELETE, realizando la solicitud a la misma ruta http://api.lmjyo.org/api/projects/*id.*

Figura 26. **Solicitud DELETE para eliminar un proyecto**



Fuente: elaboración propia, empleando PostMan.

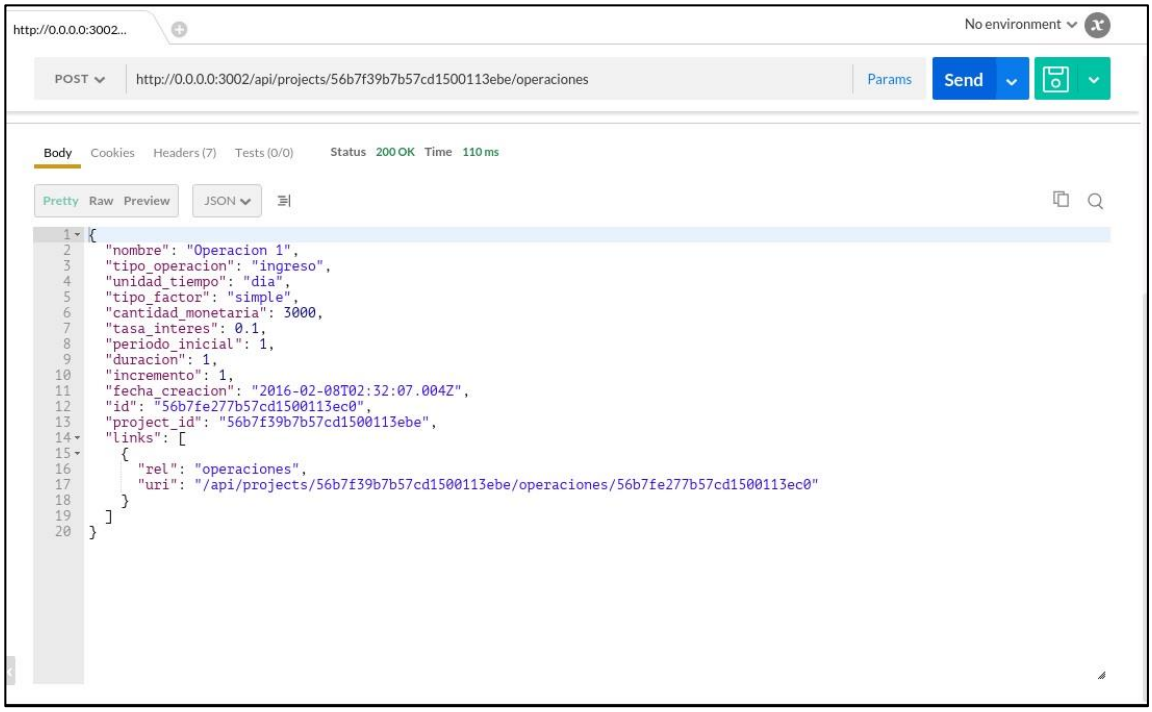
6.3. Creación, obtención, modificación y eliminación de operaciones

Las operaciones representan operaciones financieras que se realizarán a partir del inicio de la ejecución del proyecto. Cada proyecto deberá poseer operaciones para poder ser evaluado, por lo que las operaciones son los elementos que le agregan valor a los proyectos.

Para la creación de una operación se necesita la siguiente información: nombre, el tipo de operación, la unidad de tiempo, el tipo de factor, la cantidad en unidades monetarias, la tasa de interés y el periodo inicial. La duración y el incremento dependerán del tipo de factor que se aplique a la operación.

Se realiza la solicitud a http://api.lmjyo.org/api/projects/id/operaciones en donde se coloca el *id* del proyecto sobre el cual se creara la operación.

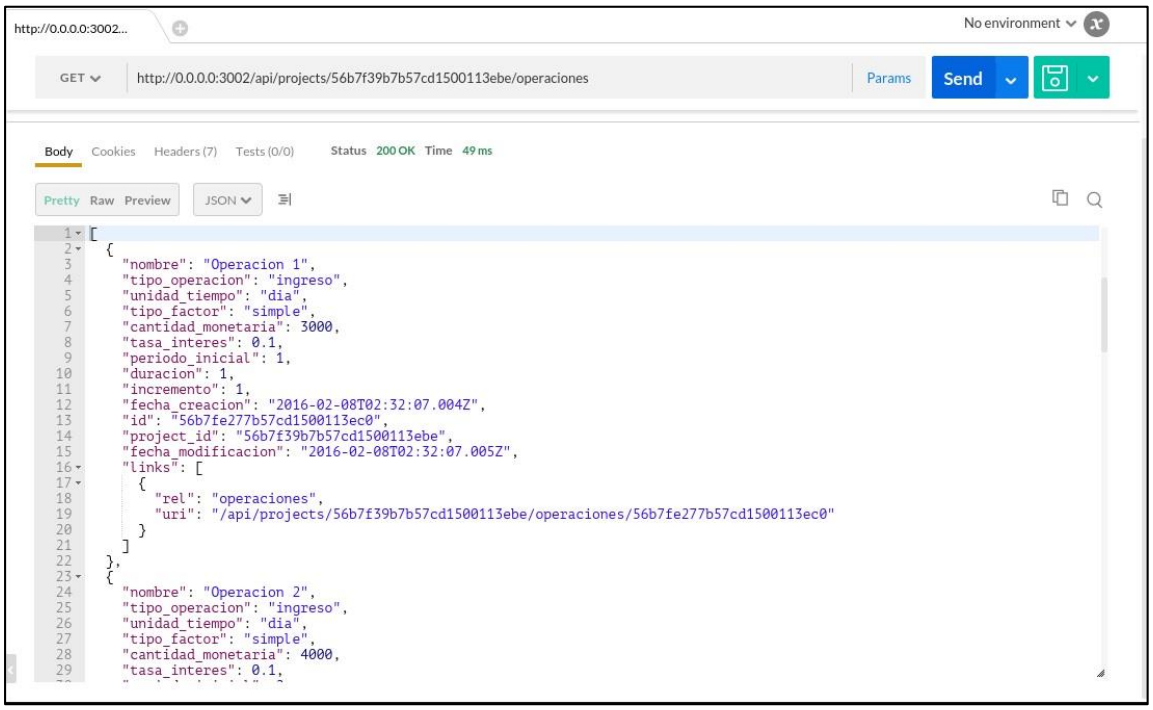
Figura 27. **Solicitud POST para la creación de operaciones**



Fuente: elaboración propia, empleando PostMan.

Los usuarios podrán obtener una operación en específico a partir del *id* de la misma.

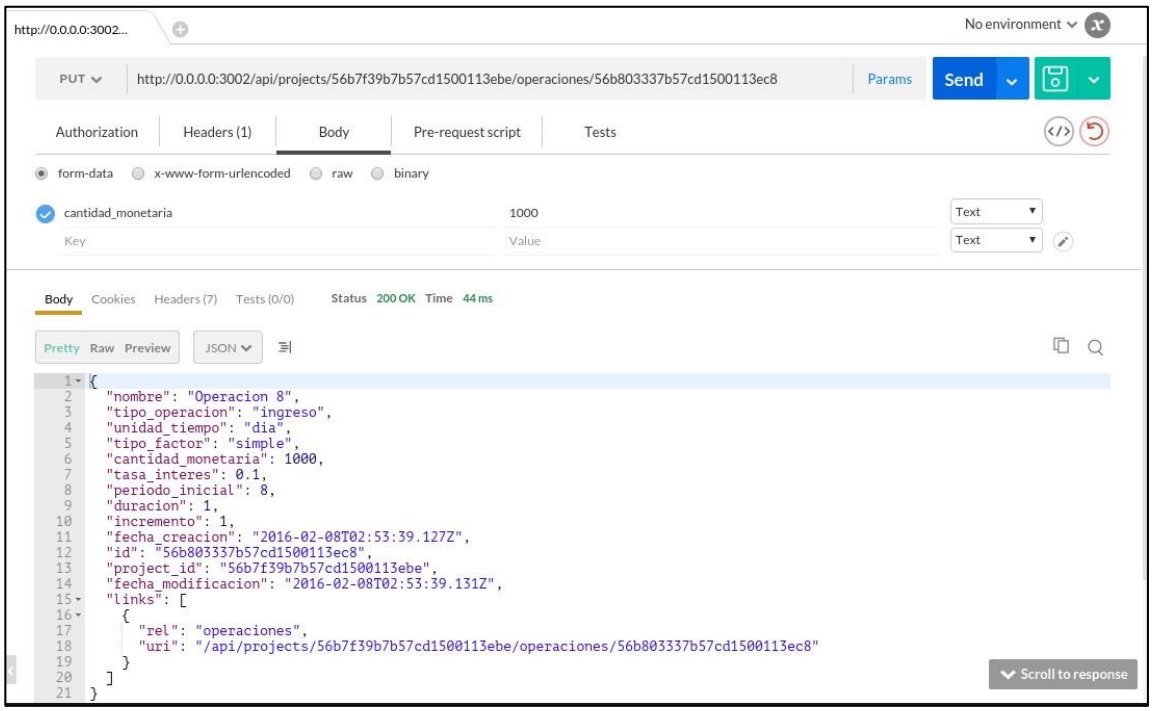
Figura 28. **Solicitud GET para obtener una operación**



Fuente: elaboración propia, empleando PostMan.

Los usuarios pueden actualizar la información contenida en alguna operación, por medio de una operación PUT sobre la ruta http://api.lmjyo.org/api/projects/id_proyecto/operaciones/id, donde se utiliza el *id* del proyecto y el *id* de la operación.

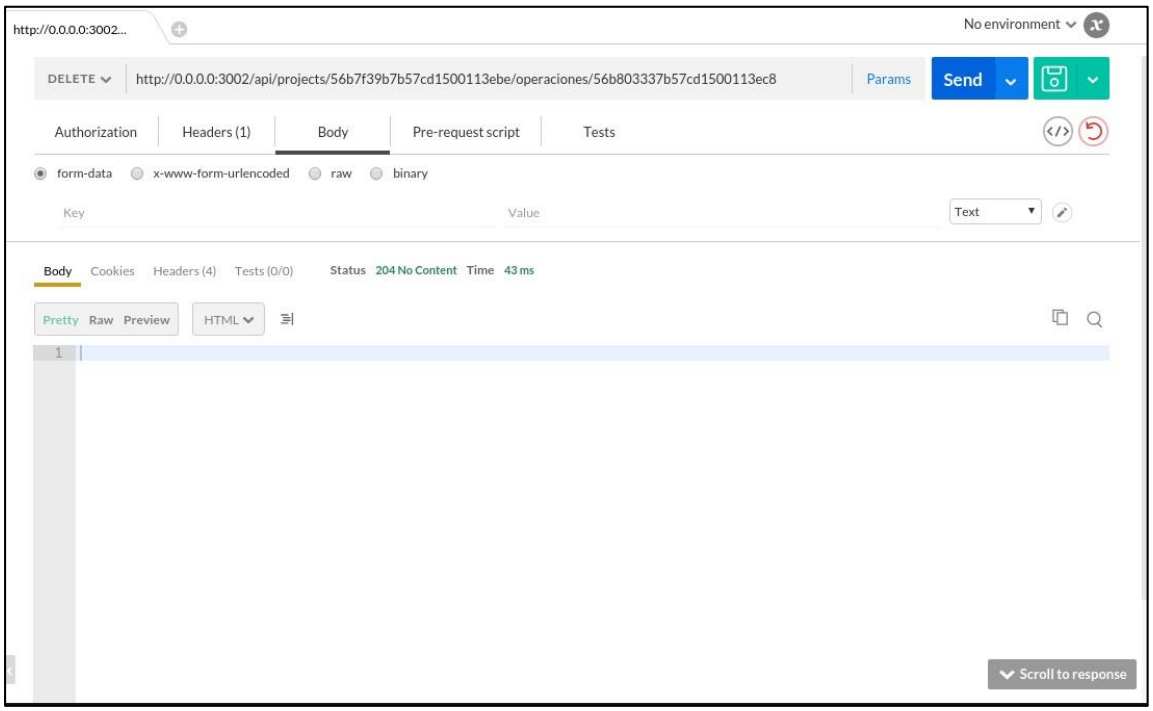
Figura 29. **Solicitud PUT para la actualización de una operación**



Fuente: elaboración propia, empleando PostMan.

Al igual que con las operaciones para los proyectos, se cuenta con una ruta para realizar eliminaciones de operaciones, la cual es http://api.lmjyo.org/api/projects/id_project/operaciones/id, donde se coloca el *id* del proyecto y se coloca el *id* de la operación a eliminar. Cabe aclarar que únicamente se eliminará la operación y no el proyecto en sí.

Figura 30. **Solicitud DELETE para la eliminación de una operación**

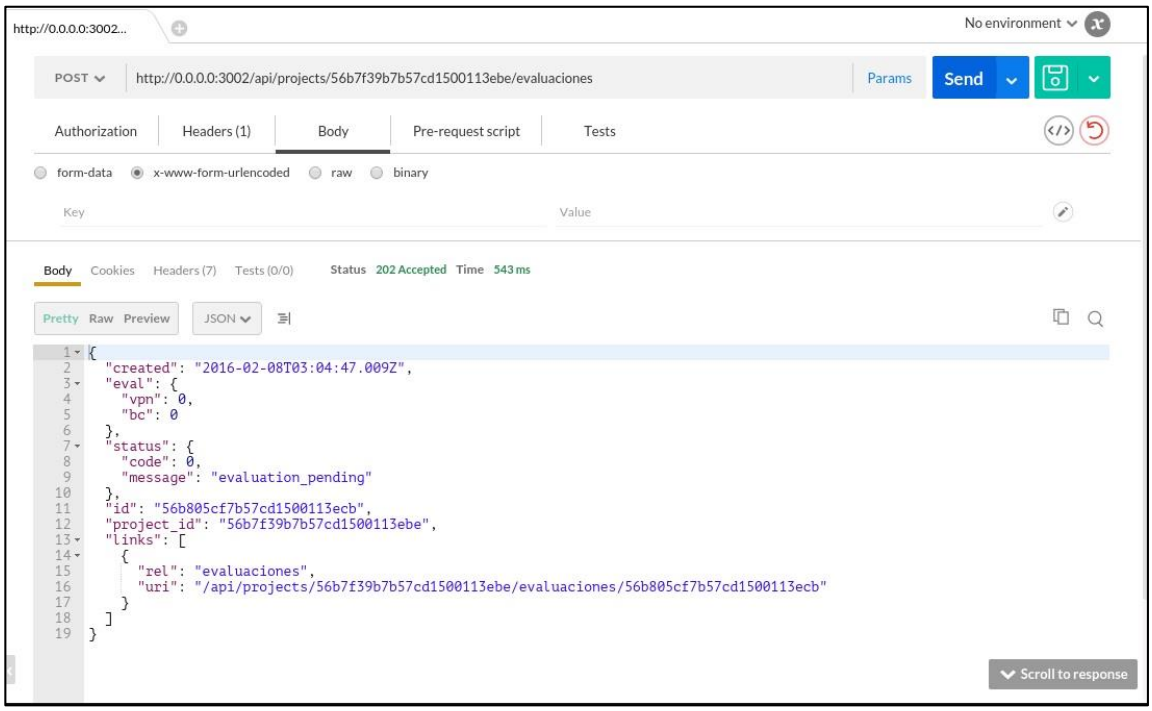


Fuente: elaboración propia, empleando PostMan.

6.4. Evaluación de proyectos

Al momento de tener varias operaciones dentro de un proyecto, se puede realizar una evaluación, la cual se hará utilizando las fórmulas de ingeniería económica para el cálculo de VPN y del análisis de beneficio/costo.

Figura 31. **Solicitud POST para la creación de una evaluación**

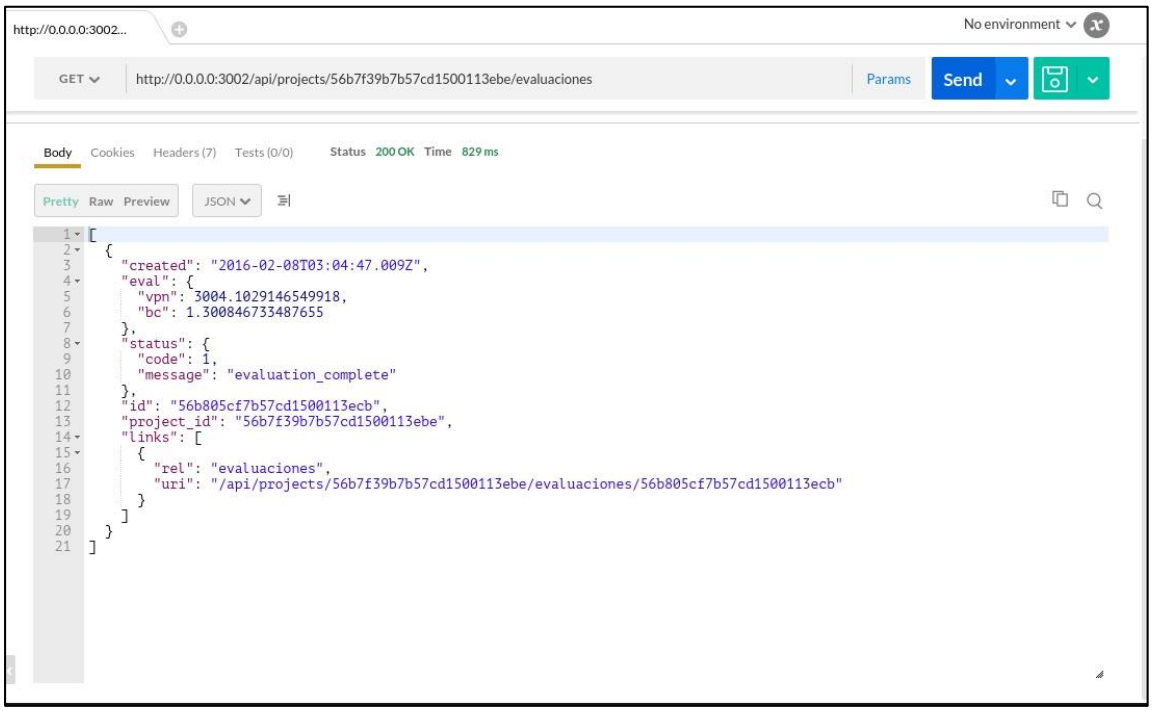


Fuente: elaboración propia, empleando PostMan.

Se realiza la solicitud POST para crear una nueva evaluación a http://api.lmjyo.org/api/projects/id_project/evaluaciones.

En cada evaluación se almacena el estado de la misma. Dicho estado cambia al ser procesada por la API, pero al momento de crearse la evaluación, está aún no ha sido procesada y por lo tanto no devolverá la información en el momento en que se realiza la solicitud.

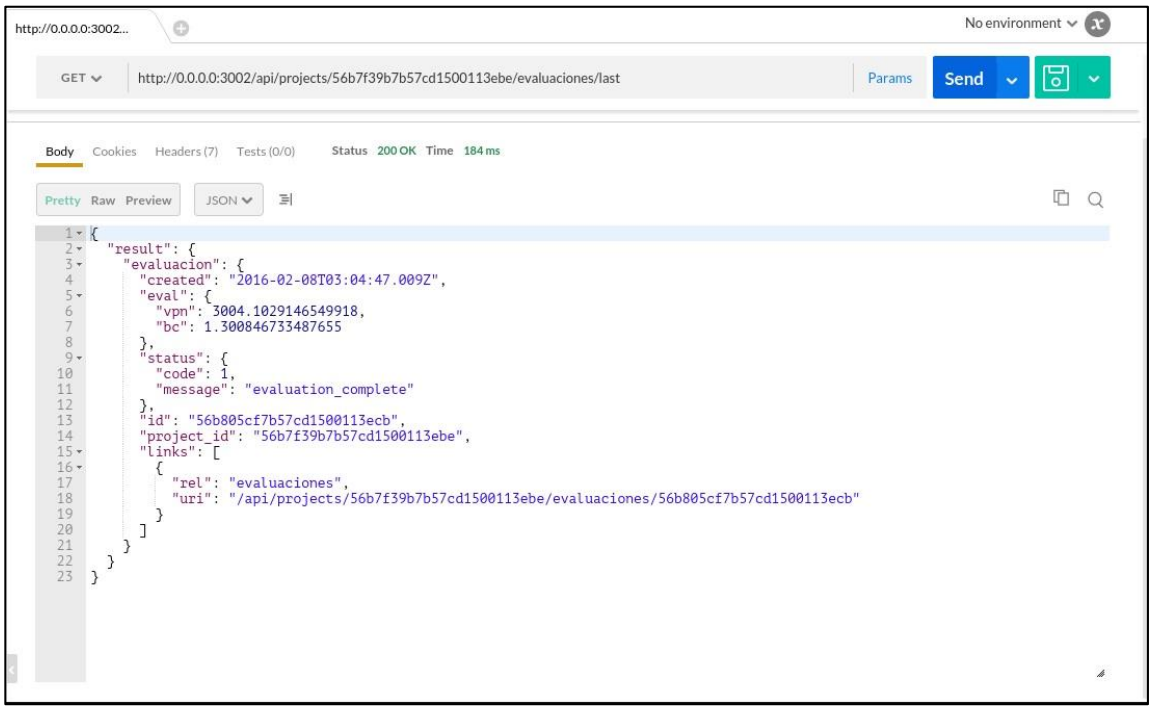
Figura 32. **Solicitud GET para verificar el estado de la evaluación**



Fuente: elaboración propia, empleando PostMan.

La ruta para obtener la última evaluación procesada es http://api.lmjyo.org/api/projects/id_project/evaluaciones/last, la cual garantiza que el proyecto cuenta con la información de las evaluaciones actualizadas, teniendo en cuenta que si se trabaja con base en una evaluación antigua, se pueden tomar otras decisiones que no se adaptan a la realidad.

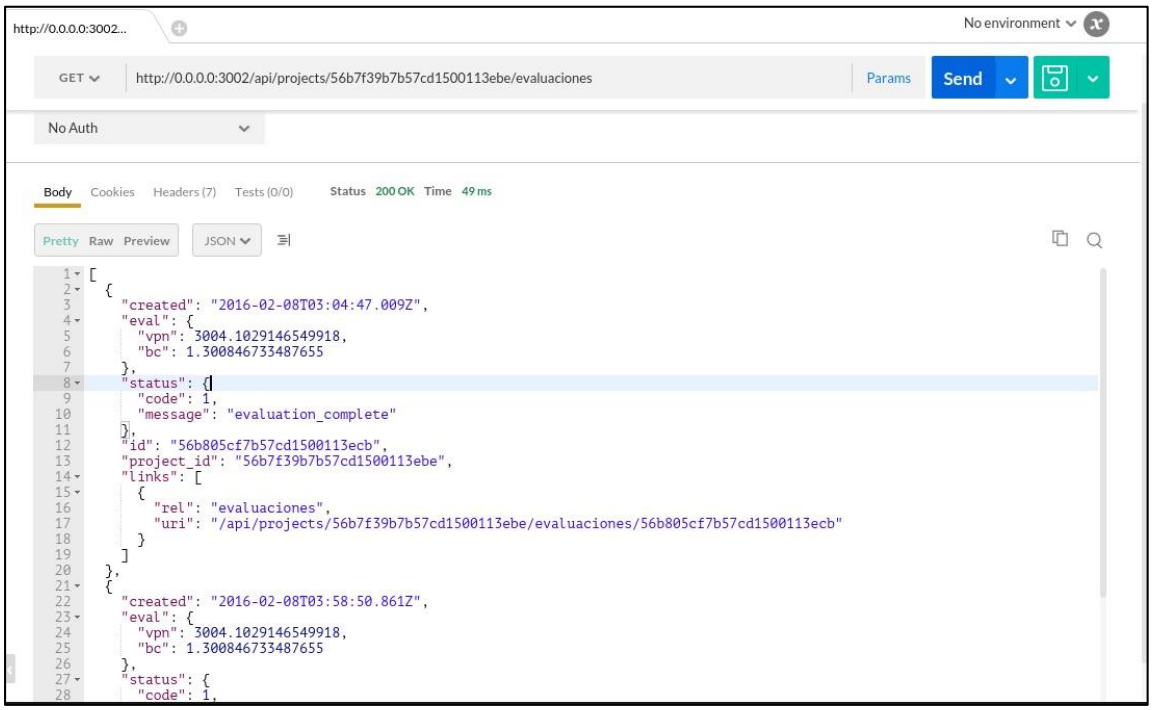
Figura 33. **Solicitud GET para obtener la última evaluación**



Fuente: elaboración propia, empleando PostMan.

Si se actualiza alguna operación dentro del proyecto, se puede volver a crear una nueva evaluación por medio de la ruta destinada para ello. Esto garantiza que únicamente al momento de realizarse estos cambios se pueda crear una nueva evaluación, para no sobrecargar al proyecto de evaluaciones idénticas.

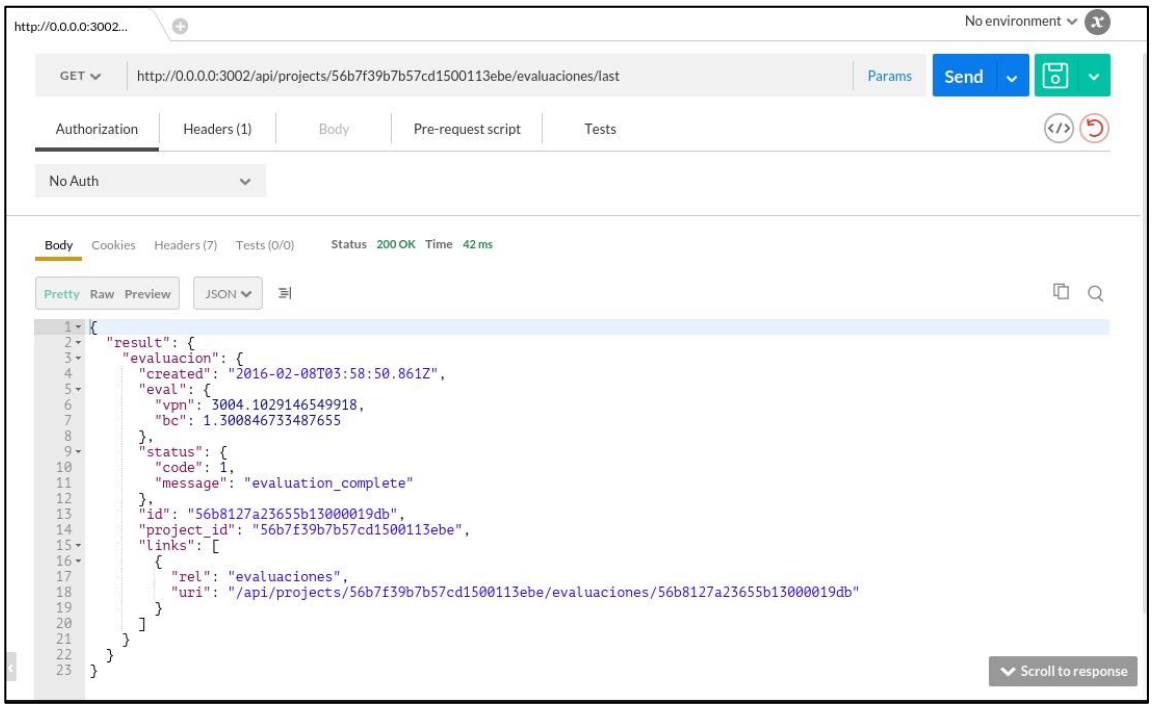
Figura 34. **Solicitud GET para obtener todas las evaluaciones**



Fuente: elaboración propia, empleando PostMan.

En lugar de devolver la primera evaluación creada, se regresa la última evaluación que ha sido trabajada. Esta es la evaluación con los datos de las operaciones actualizados y es la que se recomienda utilizar para la toma de decisiones.

Figura 35. **Solicitud GET para obtener la última evaluación procesada**

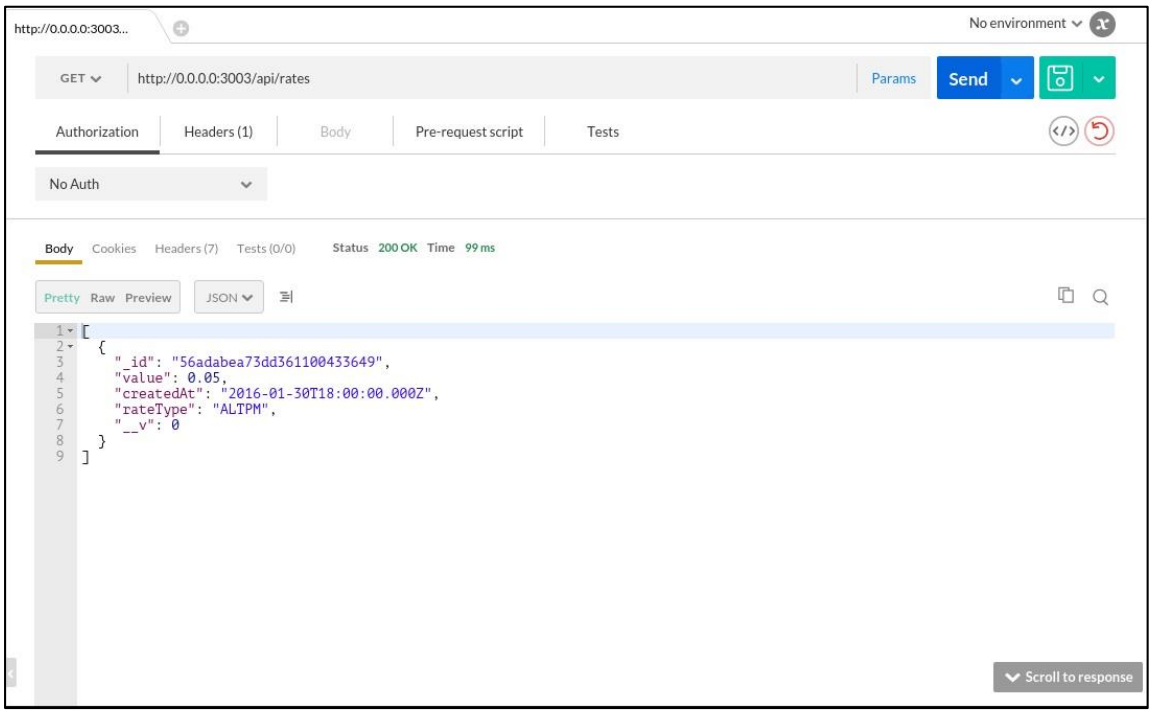


Fuente: elaboración propia, empleando PostMan.

6.5. Obtención de tasas de interés

Para obtener un listado de todas las tasas de interés que se encuentran en el servicio, se realiza una solicitud GET a http://api.lmjyo.org/api/rates la cual devuelve el id de la tasa, el tipo y el valor, en valores entre 0 a 1 donde 1 representa el 100 %.

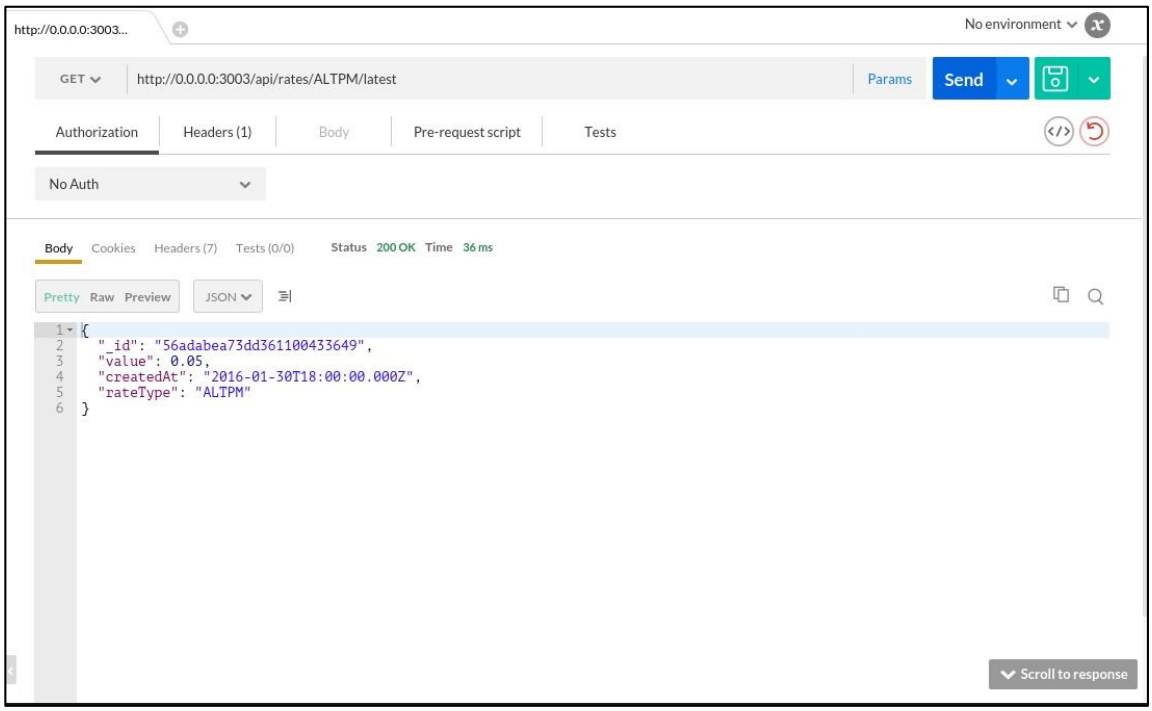
Figura 36. **Solicitud GET para obtener todas las tasas de interés**



Fuente: elaboración propia, empleando PostMan.

Ya que existen distintos tipos de tasas de interés, se cuenta con una ruta especializada en obtener la tasa de interés más reciente, dependiendo del tipo que sea. La ruta es http://api.lmjyo.org/api/rates/id_tipo/latest y se obtiene utilizando una solicitud GET.

Figura 37. **Solicitud GET para obtener la tasa de interés actualizada**

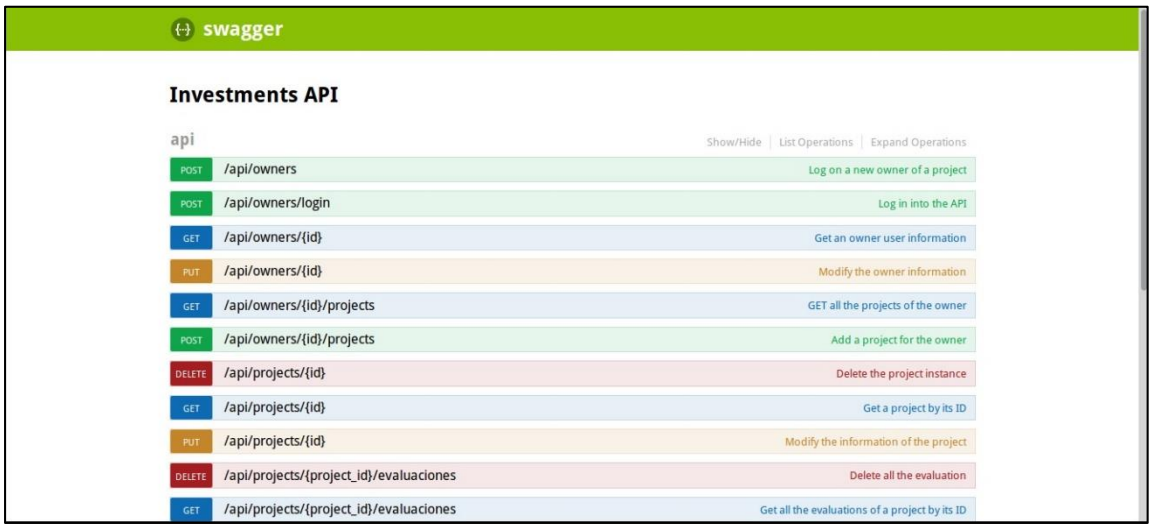


Fuente: elaboración propia, empleando PostMan.

6.6. Documentación de la API

Este servicio brinda información acerca de cada una de las diferentes rutas que la API utiliza, presentando la documentación en cada una de ellas. Este puede ser accedido desde http://api.lmjyo.org/documentation utilizando un navegador web.

Figura 38. **Documentación de todas las rutas de la API**



Fuente: elaboración propia, empleando el módulo Hapi Swagger.

6.7. Código fuente y licencia

El código fuente de la API se encuentra de manera pública en el sitio web Github en la organización https://github.com/lmyjo. En ella se pueden encontrar los repositorios de cada microservicio que conforma la API.

Para contribuir al desarrollo del conocimiento colectivo de la humanidad, cada proyecto ha sido liberado bajo la licencia GPL versión 3, la cual está presente en cada repositorio de la organización. La especificación de la licencia puede ser encontrada en un archivo llamado LICENSE en cada proyecto.

CONCLUSIONES

1. Se desarrolló una API para la estimación del valor presente neto y del análisis de beneficio/costo de proyectos, con base en los conocimientos teóricos de la ingeniería económica y de los técnicos sobre desarrollo de soluciones de software con arquitectura de servicios.

2. La API desarrollada cuenta con un servicio asíncrono encargado del cálculo de los factores presente dado futuro, presente dado anualidad y presente dado gradiente aritmético, utilizando el modelo matemático de cada uno para el cálculo del valor presente neto y del análisis de beneficio/costo.

3. La API desarrollada está dividida en microservicios que se ejecutan en ambientes aislados y controlados. Cada uno cumple una tarea específica y fundamental.

4. Se desarrolló una API pública, la cual provee servicios que facilitan la creación de herramientas tecnológicas, como aplicaciones web, móviles y de escritorio, para la realización de análisis financieros de viabilidad de proyectos a través de cálculo del valor presente neto y el análisis de beneficio/costo.

5. Se desarrolló una API que permite la obtención de variables relacionadas a tasas de interés directamente del Banco de Guatemala, a través del microservicio de Rates.

RECOMENDACIONES

1. Dar a conocer la API de análisis financiero con instituciones que brindan apoyo a las nuevas micro y pequeñas empresas, para facilitar la realización de un análisis financiero que sea de ayuda para la sobrevivencia de las mismas.

2. Implementar procesos de entrega continua para facilitar la liberación de nuevas versiones de los microservicios que componen a la API.

3. Utilizar y ampliar el conocimiento sobre herramientas de creación de *clusters* para realizar una entrega continua efectiva.

4. Publicar los contenedores de la API en un servicio de infraestructura o plataforma en la nube que soporte una fácil instalación de los mismos.

BIBLIOGRAFÍA

1. ATLASSIAN. *Gitflow workflow.* [en línea]. <https://es.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow/>. [Consulta: 1 de diciembre de 2015].

2. BARCA, Gabriel. *Fundamentos de la ingeniería económica.* 4a. ed. México: McGraw-Hill Interamericana, 2007. 593 p.

3. Centro de Investigaciones Económicas Nacionales. *Micro, pequeñas y medianas empresas en Guatemala*. Guatemala: CIEN, 2010. 48 p.

4. ERL, Thomas. *SOA, principles of service design.* Estados Unidos: Prentince Hall, 2008. 608p.

5. FOWLER, Martin. *Richardson Maturity Model*. [en línea]. <http://martinfowler.com/articles/richardsonMaturityModel.html>. [Consulta: 1 de diciembre de 2015].

6. GUERRERO, Alba. *Apuntes del curso de ingeniería económica 1*. Guatemala: USAC, 2013. 50 p.

7. MDN. *About JavaScript*. [en línea]. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. [Consulta: 30 de diciembre de 2015].

8. MORALES, Ricardo. *Apuntes del curso de análisis y diseño de sistemas 2*. Guatemala: USAC, 2013. 45 p.

9. NEWMAN, Sam. *Building microservices: designing fine-grained systems.* Estados Unidos: O'Reilly Media, 2015. 473 p.

10. RICHARDSON, Chris. *Building microservices: using an API gateway*. [en línea]. <https://www.nginx.com/blog/building-microservices-using-an-api-gateway/>. [Consulta: 10 de diciembre de 2015].

11. _____. *Microservices patterns*. [en línea]. <http://microservices.io/patterns/>. [Consulta: 10 de diciembre de 2015].

12. STRONGLOOP. *Loopback documentation*. [en línea]. <https://docs.strongloop.com/display/public/LB/LoopBack>. [Consulta: 12 de diciembre de 2015].

13. TURNBULL, James. *The docker book*. Estados Unidos: Turnbull, 2014. 258 p.

APÉNDICE

Apéndice 1. Códigos de estado de HTTP

Código	Descripción
1xx – Información	
100	Continuar: indica que el servidor ha recibido encabezados y el cliente puede enviar el cuerpo del mensaje.
101	Conmutación de protocolos: indica que el cliente solicitó al servidor el cambio en los protocolos y el servidor está enterado de ello.
102	Procesando: indica que el servidor está procesando la petición y el cliente no excede el límite de espera.
103	Punto de control: indica que el servidor reanudara una petición POST o PUT abortada previamente.
2xx – Éxito	
200	OK: respuesta estándar para respuestas HTTP exitosas.
201	Creado: indica que un recurso ha sido creado de forma exitosa.
202	Aceptado: indica que una petición ha sido aceptada con éxito pero que aún debe ser procesada.
203	Información de no autoría: el servidor respondió de manera exitosa la petición pero el resultado es de otro servidor.
204	Sin contenido: el servidor ha procesado de forma exitosa la petición pero no ha retornado ningún contenido.
205	Contenido de reinicio: retorna una respuesta sin contenido y solicita que el cliente reinicie la carga del documento.

Continuación del apéndice 1

206	Contenido parcial: retorna solamente una parte del recurso, siendo utilizado para reiniciar una descarga o dividir una descarga en varios flujos.
207	Múltiple respuesta: retorna un mensaje XML que contiene múltiples códigos de estado, dependiendo del número de sub solicitudes realizadas.
208	Previamente reportado: se ha notificado previamente sobre el listado de elementos DAV.
3xx – Redirección	
300	Múltiples elecciones: Indica diferentes opciones para presentar la información.
301	Movido permanentemente: Todas las solicitudes realizadas deben ser redirigidas a la nueva localización del recurso.
302	Encontrado: el recurso solicitado se encuentra en otra localización y se redirige a la nueva localización.
303	Mirar otro: el recurso solicitado se encuentra en otra localización y se indica la URI en donde se encuentra.
304	No modificado: indica que el recurso, a pesar de haberse solicitado una modificación, quedó sin cambio alguno.
305	Uso de proxy: el recurso es requerido es accesible solamente a través de un proxy.
307	Redirigido temporalmente: indica que el recurso accedido no se encuentra en su ruta temporalmente.
308	Redirigido permanentemente: indica que el recurso no se encuentra en su ruta y no será accesible a través de ella nuevamente.

Continuación del apéndice 1

4xx – Errores de usuario	
400	Solicitud mala: la solicitud que se realizó no se hizo por el método adecuado o incluye errores de sintaxis en el protocolo.
401	No autorizado: el recurso es accesible, sin embargo el usuario que lo requiere no está autorizado para obtenerlo o hace falta una autenticación.
403	Prohibido: el acceso al recurso está prohibido por el servidor, aún con autenticación.
404	No encontrado: el recurso solicitado no ha sido encontrado.
405	Método no aceptado: el método HTTP empleado no es aceptado por el servidor.
406	No aceptado: el recurso solicitado no acepta las condiciones colocadas en los encabezados.
407	Autenticación por medio de proxy requerida: el usuario que está solicitando el recurso debe autenticarse por medio de un proxy.
408	Tiempo de espera agotado para solicitud: el servidor agotó su tiempo de espera para recibir una solicitud.
415	Tipo de media no aceptado: indica que el servidor no acepta el tipo de media que está enviando el cliente.
422	Entidad no procesable: el servidor no puede procesar la entidad que el cliente está intentando ingresar.
5xx – Errores de servidor	
500	Error interno del servidor: mensaje genérico que indica que algo interno no esperado estropeó el funcionamiento del servicio.

Continuación del apéndice 1

501	No implementado: el servidor no reconoce el método por el que se realizó la solicitud.
502	Mala puerta de enlace: el servidor actuaba como puerta de enlace o proxy y no pudo responder de manera adecuada a una solicitud debido a una respuesta inválida.
503	Servicio no disponible: indica que el servicio solicitado se encuentra fuera de línea.
504	Tiempo de respuesta de puerta de enlace agotado: el servidor actúa como proxy o puerta de enlace y excede en el tiempo de respuesta.

Fuente: elaboración propia.