UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso para Alumnos con Práctica Profesional Supervisada

TÍTULO: Creacion de Gema utilizando Mercado Pago

AUTOR/A: Matias Adrian Delle Donne

DIRECTOR/A ACADÉMICO: Christian Rofriguez

DIRECTOR/A PROFESIONAL: Damian Grimberg

CODIRECTOR/A ACADÉMICO: -

CARRERA: Licenciatura en Informatica

RESUMEN

Banda Invitada es una plataforma innovadora diseñada para unir a diversos actores dentro del ámbito de la música. La plataforma ofrece un servicio de suscripción con pagos regulares a través de Mercado Pago. Este proyecto marcó un punto de inflexión, ya que identificamos que las soluciones utilizadas en proyectos anteriores no eran fácilmente adaptables para su reutilización en iniciativas futuras. En respuesta a esta necesidad, se propuso el desarrollo de una librería en Ruby, con el fin de extender las funcionalidades de Mercado Pago. Esta librería no solo se encargaría de gestionar las notificaciones Webhook, sino que también optimizará los procesos en los proyectos vigentes.

Palabras Claves

Ruby, Ruby on Rails, Weebhook, IPN, Ngrok, Gema, Mercado Pago, SDK, Mixins, Bundler, DRY

Conclusiones

Se propuso desarrollar una librería para mejorar un sistema de suscripciones, hemos podido refactorizar código de manera significativa, lo que ha permitido a desarrolladores de diferentes proyectos adoptar la creación de gemas en pos de no repetir código

Trabajos Realizados

Se logro crear una librería valiosa para un sistema de suscripciones, pudiendo manejar un gran volumen de notificaciones sin comprometer el rendimiento del sistema. Aplicando una técnica que permite incluir el comportamiento en una clase a partir de un módulo para extender las capacidades de la librería de Mercado Pago y proporcionar funcionalidades adicionales para el manejo de notificaciones webhook

Trabajos Futuros

Explorar la posibilidad de agregar características adicionales, como opciones de pago personalizables, integración con otras pasarelas de pago o manejo de notificaciones IPN.
Liberar la gema a la comunidad de desarrolladores y proporcionarle soporte a quien utilice nuestra gema.

Fecha de la presentación: Junio 2024

Creación de Gema utilizando Mercado Pago

Objetivo

Desarrollar una nueva librería en ruby, que simplifique el uso de la API de Mercado Pago. El objetivo es el de extender la funcionalidad de la librería existente, desarrollada y mantenida por Mercado Pago, implementando las funcionalidades ausentes y que son propias en los sistemas de suscripciones. De esta forma, se satisfacen las necesidades específicas del entorno laboral en el que me desempeño y además, luego de verificar la correcta implementación en diversos proyectos, su liberación a la comunidad de desarrolladores publicándose en un repositorio de gemas de acceso restringido por personal de la empresa. Será luego decisión de la propia empresa, si desea liberar este trabajo como software libre utilizando alguna licencia específica.

Introducción

El propósito de este trabajo, es el de describir el proceso por medio del cuál hemos detectado la importancia y necesidad de crear una gema en Ruby que extiende otra ya existente para aprovechar mejor la API de Mercado Pago. Esta necesidad, tiene a su vez dos desencadenantes cruciales en un equipo de desarrollo. Por un lado, contribuye al crecimiento y fortalecimiento del producto final por incorporar piezas de código independientes, reutilizables y con un ciclo de vida propio. Por otro lado, satisface una necesidad específica del negocio, dado que provee de una mejora sustancial al aprovechar código en múltiples proyectos, lo cuál significa menor tiempo para ofrecer una solución que es frecuente en los clientes de la empresa en la que trabajo.

La génesis en este proyecto se encuentra en una necesidad identificada durante el desarrollo, relacionada con la utilización de funciones específicas proporcionadas por la API de Mercado Pago. Estas funcionalidades aún no estaban disponibles en las librerías existentes diseñadas para el lenguaje de programación que estábamos utilizando en el proyecto. La necesidad de abordar esta carencia impulsó la creación de este trabajo. Inicialmente, se concebía como una solución limitada al proyecto en curso. Sin embargo, a medida que avanzaba el desarrollo, se convirtió en un proyecto emocionante con el objetivo de crear una librería y de esta forma contribuir a la comunidad de desarrolladores de la empresa. El propósito central de esta librería es proporcionar una solución que permita la reutilización de las funcionalidades desarrolladas en múltiples proyectos con características similares.

El mayor desafío, no radica en únicamente describir la implementación de una librería donde, si bien es código, se debe respetar cierto formato en este tipo de proyectos. Por el contrario se pone en valor la importancia de haber detectado una porción de código que diferentes equipos, en distintos proyectos, implementan libremente, repitiendo trabajo que se traduce en tiempo. A su vez, esta librería, al mantenerse como tal, tendrá lanzamientos periódicos que mejoren sus prestaciones e incluso solucionando potenciales errores que serán aplicables en múltiples proyectos de una forma más ordenada y segura.

Además, se deja el código fuente de esta solución bajo una licencia MIT[1], lo que permitirá en caso de ser liberada como software libre, su accesibilidad, contribuyendo al crecimiento y enriquecimiento de la comunidad de desarrolladores de nuestro país y Latinoamérica.

[1] https://opensource.org/license/mit

Proyecto: Banda Invitada

Banda Invitada[2] es una innovadora plataforma que tiene como objetivo unir a diferentes actores del mundo de la música, como artistas, productores, programadores de bandas y dueños de espacios para conciertos, en un solo lugar. La esencia de este proyecto es facilitar la organización y promoción de eventos musicales al permitir que tanto músicos como anfitriones publiquen sus shows en la plataforma y los compartan con una comunidad apasionada por la música en vivo. Cabe destacar que el cliente ya disponía de una plataforma desarrollada, y solicitó un rediseño con el fin de mejorar la experiencia de sus usuarios. Esta nueva versión de la plataforma fue desarrollada integralmente por la empresa en la cuál me desempeño y actualmente, se encuentra operando en un entorno productivo plenamente funcional.

Los músicos pueden aprovechar esta plataforma para dar a conocer sus propuestas artísticas y postularse para formar parte de los eventos. Esto brinda una oportunidad única para artistas emergentes o aquellos que buscan expandir su presencia en el escenario, ya que podrán conectar con anfitriones en busca de talento fresco.

Por otro lado, los anfitriones de eventos pueden utilizar el portal desarrollado para descubrir y seleccionar a los artistas que mejor se adapten a su visión y público. Esto simplifica el proceso de planificación y promoción de shows, fomentando la diversidad y la colaboración en la escena musical.

Además, la plataforma ofrece un servicio de suscripciones que requiere pagos regulares. Con el fin de garantizar transacciones seguras y una experiencia de usuario fluida, se ha integrado Mercado Pago como proveedor de pagos. Esta integración permite a los usuarios realizar transacciones confiables, ofreciendo al mismo tiempo una amplia variedad de opciones de pago.

Las transacciones a través de Mercado Pago se limitan únicamente al pago de suscripciones, las cuales pueden ser mensuales, trimestrales o anuales.

En esencia, Banda Invitada representa mucho más que una plataforma, es un ecosistema colaborativo que impulsa la música en vivo al conectar talento con oportunidades. Ahora en pleno funcionamiento se ha fortalecido este puente digital entre músicos y organizadores de eventos. Esto no solo amplía horizontes para artistas emergentes, sino que también simplifica la planificación para anfitriones, fomentando la diversidad y la creación de experiencias musicales inolvidables. Esto representa un paso más hacia el objetivo principal del proyecto: contribuir al crecimiento y la diversificación continuos de la vibrante industria musical.

[2] https://www.bandainvitada.com/

Tecnología

Como se mencionó en secciones anteriores, el proyecto Banda Invitada se ha desarrollado utilizando el lenguaje Ruby[3] sobre el framework Rails[4]. A continuación, introduciremos toda la tecnología de base implicada en el proyecto.

El lenguaje ruby

Ruby fue creado por Yukihiro Matsumoto (conocido por el seudónimo Matz) en Japón a partir de 1993. Matz mantuvo Ruby prácticamente para sí mismo hasta 1995, cuando lo liberó al público. Rápidamente, Ruby comenzó a ganar seguidores en el país natal de Matz, Japón, en los años siguientes, y finalmente obtuvo reconocimiento en el resto del mundo de la programación a partir del año 2000. A partir de ese momento, Ruby ha ganado popularidad, especialmente debido a la popularidad del framework de desarrollo de aplicaciones web Ruby on Rails.

Ruby es un lenguaje de programación orientado a objetos interpretado. Cuando decimos que es interpretado, queremos decir que el código fuente de Ruby se compila mediante un intérprete en el momento de la ejecución (similar en este sentido a JavaScript y PHP). Esto contrasta con los lenguajes compilados como C o C++, donde el código se compila previamente en un formato binario dirigido a ejecutarse en una arquitectura específica de microprocesador.

Existe una desventaja propia de un lenguaje interpretado, que es la velocidad, ya que el código fuente debe interpretarse en tiempo de ejecución, lo que significa que se ejecuta más lentamente que una aplicación compilada equivalente. Sin embargo, Ruby anula esta desventaja con sus ventajas, como la portabilidad y expresividad, que lo hacen una opción atractiva para los desarrolladores que valoran la flexibilidad y la facilidad de expresión en su código.

La principal ventaja de los lenguajes interpretados es que son portables en múltiples plataformas de sistemas operativos y arquitecturas de hardware. Una aplicación compilada, por otro lado, solo se ejecutará en el sistema operativo y hardware para el que se haya compilado. Por ejemplo, puede tomar una aplicación Ruby y ejecutarla sin modificaciones en un sistema Intel con Linux, un sistema Intel con Windows, un sistema Intel con Mac OS X o incluso un sistema PowerPC con Mac OS o Linux. Para lograr esto con una aplicación en C o C++, necesitaría compilar el código en cada uno de los 5 sistemas diferentes y poner a disposición cada imagen binaria. Con Ruby, solo proporciona el código fuente.

Otra ventaja de ser interpretado es que podemos escribir y ejecutar código Ruby en tiempo real directamente en el intérprete de Ruby, lo que se conoce como bucle de lectura-evaluación-impresión (REPL[5], por sus siglas en inglés).

---

[3] https://www.ruby-lang.org/es/
[4] https://rubyonrails.org/
[5] https://www.rubyguides.com/2018/12/what-is-a-repl-in-ruby/

En un REPL, el usuario ingresa una o más expresiones (en lugar de una unidad de compilación completa) y el REPL las evalúa y muestra los resultados. El nombre bucle lectura-evaluación-impresión proviene de los nombres de las funciones primitivas Lisp que implementan esta funcionalidad:

- La función de lectura acepta una expresión del usuario y la analiza en una estructura de datos en la memoria.
- La función de evaluación toma esta estructura de datos interna y la evalúa.
- La función de impresión toma el resultado obtenido por eval y lo imprime para el usuario.

Además de las consideraciones sobre Ruby como lenguaje interpretado, es importante mencionar que existen diferentes máquinas virtuales de Ruby. Estas máquinas virtuales son:

1. **MRI (Matz 's Ruby Interpreter)**: Esta es la implementación de referencia de Ruby, también conocida como CRuby. Fue desarrollada por el creador de Ruby, Yukihiro Matsumoto, y es ampliamente utilizada. MRI interpreta y ejecuta el código Ruby de manera eficiente, pero tiene ciertas limitaciones de rendimiento en aplicaciones de alto rendimiento debido a su arquitectura de un solo subproceso.

2. **JRuby:** JRuby es una implementación de Ruby que se ejecuta en la máquina virtual de Java (JVM). Esto significa que puedes utilizar Ruby en entornos Java y aprovechar la interoperabilidad con las bibliotecas y el ecosistema de Java. JRuby también puede ofrecer un mejor rendimiento en ciertos casos que MRI debido a la JVM y su capacidad para manejar la concurrencia.

3. **Rubinius:** Rubinius es una implementación de Ruby que se centra en la ejecución en paralelo y la concurrencia. Utiliza una máquina virtual escrita en C++ y Ruby, y se esfuerza por ofrecer un rendimiento óptimo. Aunque no es tan comúnmente utilizado como MRI o JRuby, es una opción interesante para ciertas aplicaciones.

4. **TruffleRuby:** TruffleRuby es una implementación experimental de Ruby que se ejecuta en la máquina virtual GraalVM. Tiene como objetivo mejorar el rendimiento y la eficiencia de Ruby a través de técnicas de compilación y optimización avanzadas. Puede ser una opción interesante para aplicaciones de alto rendimiento.

La Popularidad de Ruby

En primer lugar, Ruby es un lenguaje de programación muy intuitivo y limpio. Esto hace que aprender Ruby sea una tarea menos desafiante que aprender algunos otros lenguajes. Ruby también es un gran lenguaje de propósito general. Se puede utilizar para escribir scripts de la misma manera que usaría Perl y se puede utilizar para crear aplicaciones independientes a gran escala basadas en GUI. Sin embargo, la utilidad de Ruby no se detiene allí. Ruby también es excelente para servir páginas web, generar contenido de páginas web dinámicas y sobresale en tareas de acceso a bases de datos.

No solo Ruby es intuitivo y flexible, sino que también es extensible, lo que permite agregar nuevas funcionalidades a través de la integración de librerías de terceros o incluso bibliotecas creadas a medida.

Y, por supuesto, al ser un lenguaje interpretado, Ruby es portable. Una vez que se ha desarrollado una aplicación en Ruby, funcionará igual de bien en plataformas compatibles con Ruby, como Linux, UNIX, Windows y Mac OS X.

Gemas: las librerías de Ruby

En el contexto de Ruby, el término "gema" alude a una unidad de software que encapsula una funcionalidad específica y se convierte en un pilar fundamental en el ecosistema de desarrollo de aplicaciones Ruby.

Estas gemas pueden conceptualizarse como librerías de código reutilizables que simplifican y agilizan el proceso de construcción de aplicaciones en Ruby, permitiendo a los desarrolladores acceder a un vasto conjunto de herramientas previamente desarrolladas para abordar una amplia gama de tareas. Desde funciones elementales, como la manipulación de cadenas y operaciones matemáticas, hasta soluciones más especializadas, como librerías para la interacción con bases de datos o frameworks de desarrollo web, las gemas enriquecen el lenguaje Ruby y amplían sus capacidades, proporcionando una base sólida para la creación de aplicaciones robustas y funcionales.

En términos concretos, una gema típicamente incluye varios componentes fundamentales que respaldan su utilidad y eficacia:

1. **Código Ruby:** En el corazón de cada gema reside el código Ruby que implementa las funcionalidades específicas que ofrece. Este código actúa como el motor que impulsa las capacidades de la gema y proporciona a los desarrolladores un conjunto de herramientas listo para su uso.

2. **Archivos de Configuración:** Los archivos de configuración acompañan a la gema, proporcionando información detallada sobre cómo instalar y configurar la gema en un proyecto. Esto garantiza que su integración en una aplicación Ruby sea un proceso ordenado y efectivo.

3. **Documentación:** La documentación es un recurso esencial para los desarrolladores. Ofrece descripciones detalladas y ejemplos de uso que permiten a quienes trabajan con la gema comprender su funcionalidad y aplicarla de manera efectiva en sus propios proyectos.

4. **Pruebas Automatizadas:** La robustez y confiabilidad de una gema se demuestran a través de un conjunto de pruebas automatizadas. Estas pruebas garantizan que la gema funcione de manera coherente y resista cambios o actualizaciones sin interrupciones en su rendimiento.

5. **Metadatos:** Los metadatos contienen información crucial sobre la gema, como su versión, el nombre del autor y otros detalles importantes que facilitan su identificación y gestión en proyectos.

En este contexto, RubyGems[6] emerge como un sistema oficial de gestión de paquetes diseñado específicamente para Ruby. RubyGems actúa como un repositorio centralizado que facilita tanto la instalación como la distribución de gemas, simplificando significativamente el proceso de desarrollo de aplicaciones en Ruby. Gracias a RubyGems, los desarrolladores pueden buscar, instalar y gestionar eficientemente las gemas necesarias para sus proyectos, lo que se traduce en una mayor productividad y fluidez en el desarrollo de software Ruby.

Acompañando a RubyGems, Bundler[7] se convierte en una herramienta crucial en el desarrollo de aplicaciones Ruby al manejar sus dependencias, asegurando que el proyecto tenga acceso a las versiones precisas de las gemas requeridas. Su papel central radica en mantener la consistencia y la portabilidad del código en distintos entornos de desarrollo, lo que contribuye a un proceso ordenado y fiable. El Gemfile, propio de Bundler, es clave para definir las dependencias y versiones de las gemas de manera clara y legible, permitiendo a los desarrolladores especificar configuraciones importantes para la gestión de dependencias. Este enfoque garantiza que el proyecto acceda exactamente a las gemas necesarias, manteniendo la uniformidad y la capacidad de reproducir el entorno de desarrollo de manera confiable.

Por último, Semver[8], la abreviatura de "Semantic Versioning" (Versionamiento Semántico), establece una convención de versionamiento que permea tanto Ruby como otras comunidades de desarrollo de software. Basada en un formato de tres números (X.Y.Z), donde X denota la versión principal, Y la versión secundaria y Z la versión de revisión, Semver se convierte en una herramienta valiosa para los desarrolladores al ofrecer una comprensión inmediata del impacto de una actualización en una gema o biblioteca. Esto simplifica significativamente la gestión de dependencias y previene conflictos de compatibilidad, contribuyendo a un flujo de desarrollo más eficiente y menos propenso a errores. En el contexto de Ruby, esta convención de versionamiento se integra de manera orgánica para garantizar la estabilidad y compatibilidad de las gemas en el desarrollo de aplicaciones.

Aunque RubyGems, Bundler y Semantic Versioning son pilares fundamentales para la eficiencia y estabilidad del desarrollo en Ruby, existen las gemas privadas.

Estas gemas, se almacenan en repositorios específicos, como Gemfury[9], Sonatype Nexus[10] o en repositorios privados de Git[11], también permiten salvaguardar secciones de código confidencial, como algoritmos propietarios, lógica comercial específica o incluso datos sensibles, evitando su exposición inadvertida.

---

[6] https://guides.rubygems.org/what-is-a-gem/
[7] https://bundler.io/
[8] https://semver.org/lang/es/
[9] https://gemfury.com/
[10] https://www.sonatype.com/
[11] https://git-scm.com/

Este enfoque no solo garantiza la seguridad de información crítica, sino que también preserva la integridad de la propiedad intelectual dentro del entorno de desarrollo.

Ruby on Rails

Ruby on Rails o simplemente Rails, es un framework de desarrollo de aplicaciones web escrito en el lenguaje de programación Ruby creado el año 2004 por David Heinemeier Hanss. Está diseñado para facilitar la programación de aplicaciones web al hacer suposiciones sobre lo que cada desarrollador necesita para comenzar. Le permite escribir menos código y lograr más que muchos otros lenguajes y marcos. Los desarrolladores experimentados de Rails también informan que hace que el desarrollo de aplicaciones web sea más divertido.

Rails en sí es una gema que requiere un conjunto de otras gemas, esto puede verse en Rubygems.

Librerías de Rails

Dentro del de Rails, existen "gemas" que abarcan una amplia gama de funcionalidades, desde el manejo de autenticación y autorización hasta la integración con servicios externos y la optimización del rendimiento.

- **Active Job**[12]**:** Permite declarar trabajos y hacer que se ejecuten en una variedad de servidores de cola. Estos trabajos pueden ser de todo, desde limpiezas programadas periódicamente hasta cargos de facturación y envíos por correo. En realidad, cualquier cosa que pueda dividirse en pequeñas unidades de trabajo y ejecutarse en paralelo. El punto principal es garantizar que todas las aplicaciones de Rails tengan una infraestructura de trabajo en su lugar.

- **Action Cable**[13]**:** Integra websockets con Rails, permitiendo escribir funciones en tiempo real en Ruby con el mismo estilo y forma que el resto de su aplicación Rails, sin dejar de ser eficaz y escalable. Es una oferta completa que proporciona un framework de JavaScript del lado del cliente y un framework de Ruby del lado del servidor. Tiene acceso a todo su modelo de dominio escrito con Active Record o su ORM de elección.

- **Action Mailbox**[14]**:** Enruta los correos electrónicos entrantes a buzones de correo similares a controladores para procesarlos en Rails. Estos correos electrónicos entrantes son enrutados de forma asincrónica mediante **Active Job** a uno o varios buzones de correo dedicados, que son capaces de interactuar directamente con el resto de su modelo de dominio.

[12] `https://api.rubyonrails.org/classes/ActiveJob.html`
[13] `https://api.rubyonrails.org/classes/ActionCable.html`
[14] `https://api.rubyonrails.org/classes/ActionMailbox.html`

- **Action Mailer**[15]**:** Se utiliza para gestionar el envío de correos electrónicos desde una aplicación Rails. Permite a los desarrolladores generar y enviar correos electrónicos de manera programática, lo que es especialmente útil para enviar notificaciones, confirmaciones, boletines y otros tipos de mensajes por correo electrónico desde una aplicación web.

- **Action Text**[16]**:** Permite la incorporación y manipulación sencilla de contenido enriquecido, como imágenes y videos, en aplicaciones web

- **Action View**[17]**:** Es el responsable del templating de vistas y su renderización.Integra helpers que asisten en la construcción de respuestas.

- **Active Model**[18]**:** Es quien provee módulos con funcionalidades similares a las presentes en AR. De esta forma, Action Pack puede utilizar Active Models como objetos AR de forma indiferente dando independencia del ORM que podría ser personalizado.

- **Active Record**[19]**:** Es un framework para la abstracción de las bases de datos. Un ORM que representa la M de MVC.

- **Active Storage**[20]**:** Simplifica la subida y gestión de archivos en proveedores de cloud mediante referencias. Se integra fácilmente con AWS S3, Google Cloud Storage, Microsoft Azure Storage, almacenando las referencias correspondientes en AR. Soporta además un servicio principal y otros como espejos para ofrecer redundancia. Puede usarse localmente para deployment locales o desarrollo, pero su foco principal está en storage en la nube.

- **Active Support**[21]**:** Representan las extensiones a Ruby y clases que proveen mayor funcionalidad.

- **Bundler**[22]**:** Es un gestor de dependencias.

- **Railties**[23]**:** Es el núcleo de Rails proporcionando varias funciones para extender Rails y/o modificar el proceso de inicialización.

La filosofía Rails incluye dos principios rectores principales:

1. **Don't repeat yourself (DRY):** DRY es un principio de desarrollo de software que establece que *"Cada conocimiento debe tener una representación única, inequívoca*

[15] https://api.rubyonrails.org/classes/ActionMailer.html
[16] https://api.rubyonrails.org/classes/ActionText.html
[17] https://api.rubyonrails.org/classes/ActionView.html
[18] https://api.rubyonrails.org/classes/ActiveModel.html
[19] https://api.rubyonrails.org/classes/ActiveRecord.html
[20] https://api.rubyonrails.org/classes/ActiveStorage.html
[21] https://api.rubyonrails.org/classes/ActiveSupport.html
[22] https://bundler.io/
[23] https://api.rubyonrails.org/classes/Rails/Railtie.html

y autorizada dentro de un sistema". Al no escribir la misma información una y otra vez, nuestro código es más fácil de mantener, más extensible y tiene menos errores.

2. **Convención sobre configuración:** Es un paradigma de diseño de software utilizado por frameworks de software que intenta reducir la cantidad de decisiones que un desarrollador que utiliza el framework debe tomar sin perder necesariamente flexibilidad y los principios de DRY.

Rails incluye todo lo necesario para crear aplicaciones web respaldadas por una base de datos de acuerdo con el patrón Modelo-Vista-Controlador (MVC).

Comprender el patrón MVC es clave para comprender Rails. MVC divide tu aplicación en tres capas: Modelo, Vista y Controlador, cada una con una responsabilidad específica.

Modelos en Rails

Esta capa representa el modelo de dominio (como Cuenta, Producto, Persona, Publicación, etc.) y encapsula la lógica empresarial específica de tu aplicación. En Rails, las clases de modelo respaldadas por bases de datos se derivan de ActiveRecord::Base[24]. Active Record[25] te permite presentar los datos de las filas de la base de datos como objetos y enriquecer estos objetos de datos con métodos de lógica empresarial. Aunque la mayoría de los modelos de Rails están respaldados por una base de datos, los modelos también pueden ser clases Ruby ordinarias o clases Ruby que implementan un conjunto de interfaces proporcionadas por el módulo Active Model[26].

Vistas en Rails

Esta parte del modelo MVC se compone de "plantillas" que son responsables de proporcionar representaciones adecuadas de los recursos de tu aplicación. Las plantillas pueden tener una variedad de formatos, pero la mayoría de las plantillas de vista son HTML con código Ruby incrustado (archivos ERB). Las vistas generalmente se representan para generar una respuesta del controlador o para generar el cuerpo de un correo electrónico. En Rails, la generación de vistas es manejada por Action View[27].

Controladores en Rails

En este nivel se encuentra la capa responsable de manejar las solicitudes HTTP entrantes y proporcionar una respuesta adecuada. Por lo general, esto significa devolver HTML, pero los controladores de Rails también pueden generar XML, JSON, PDF, vistas específicas para dispositivos móviles y más. Los controladores cargan y manipulan modelos y representan plantillas de vista para generar la respuesta HTTP apropiada. En Rails, las solicitudes entrantes son enrutadas por Action Dispatch hacia un controlador adecuado, y las clases de controlador se derivan de ActionController::Base. Action Dispatch y Action Controller se agrupan juntos en Action Pack[28].

---

[24] https://api.rubyonrails.org/classes/ActiveRecord/Base.html
[25] https://api.rubyonrails.org/files/activerecord/README_rdoc.html
[26] https://api.rubyonrails.org/files/activemodel/README_rdoc.html
[27] https://api.rubyonrails.org/files/actionview/README_rdoc.html
[28] https://api.rubyonrails.org/files/actionpack/README_rdoc.html

Entre los productos más exitosos construidos con Ruby on Rails se encuentran Basecamp, GitHub, Shopify, Twitch, Dribbble, Kickstarter, Hulu, Soundcloud, entre otros.

La Doctrina Rails

El ascenso significativo de Ruby on Rails a la prominencia se debe, en gran medida, a su adopción en un momento de innovación tecnológica. Sin embargo, las ventajas tecnológicas tienden a disminuir con el tiempo, y el simple hecho de estar en el lugar adecuado en el momento oportuno no es suficiente para mantener la relevancia a largo plazo. Por lo tanto, es necesario proporcionar una explicación más profunda de cómo Ruby on Rails no solo continúa siendo relevante, sino que también amplía su influencia y su comunidad. Mi propuesta se centra en el papel crucial y constante desempeñado por su doctrina, que ha sido objeto de controversia y evolución a lo largo de la última década.

Esta doctrina, aunque ha experimentado cambios con el tiempo, aún encuentra sus raíces en las ideas fundamentales de sus fundadores. No pretendo afirmar la originalidad exclusiva de estas ideas. La principal contribución de Ruby on Rails radica en la capacidad de reunir y nutrir una comunidad cohesionada en torno a un conjunto diverso de conceptos revolucionarios acerca de la programación y los programadores.

Después de esta introducción, es relevante destacar que la Doctrina Rails[29] se compone de nueve pilares fundamentales:

1. **Optimización para la satisfacción del programador:** Enfocarse en la comodidad y la productividad de los desarrolladores.

2. **Convención sobre configuración:** Promover la configuración automática basada en convenciones predefinidas.

3. **La carta al estilo omakase:** Ofrecer elecciones preseleccionadas para simplificar la toma de decisiones.

4. **Ningún paradigma:** No restringir a un paradigma de programación en particular.

5. **Exaltar el código hermoso:** Fomentar la escritura de código limpio y elegante.

6. **Proporcionar cuchillos afilados:** Suministrar herramientas poderosas y precisas.

7. **Valorar los sistemas integrados:** Favorecer la integración efectiva de componentes y bibliotecas.

8. **Progreso sobre estabilidad**: Priorizar la mejora continua sobre la estabilidad inmutable.

[29] https://rubyonrails.org/doctrine/es

9. **Construir una gran tienda de campaña**: Crear una comunidad acogedora y diversa.

Estos pilares representan la base de la filosofía que ha impulsado y sigue impulsando a Ruby on Rails como un marco de desarrollo web de renombre en el panorama tecnológico actual.

El problema

Al enfrentarnos al desafío de gestionar sistemas de pagos y suscripciones en nuestros proyectos, generalmente optamos por utilizar Mercado Pago debido a su amplia aceptación en la sociedad. Además, contamos con experiencias previas de su integración en nuestro entorno laboral.

Sin embargo, este proyecto fue un punto de inflexión porque notamos que otras soluciones implementadas en proyectos anteriores, no eran fáciles de reutilizar en proyectos futuros. Otros proyectos solamente abordaban la gestión de notificaciones específicas, mientras que otros gestionaban todos los tipos de notificaciones sin ofrecer funcionalidades adicionales. Además, en ninguno de los casos se contemplaban soluciones para posibles nuevas notificaciones por parte de Mercado Pago.

Cómo funciona Mercado Pago

Mercado Pago, ofrece a los desarrolladores una API robusta y flexible que les permite integrar sus servicios de pago en aplicaciones web y móviles.

Para poder utilizar este servicio, como desarrollador es necesario tramitar las credenciales requeridas para la interacción de forma autorizada. Este proceso se realiza desde un portal Mercado Pago Developers[30] donde se deben crear entidades que denominan aplicaciones y que serán la base de nuestras pruebas. Básicamente, al crear una cuenta en este portal, es posible como desarrollador crear configuraciones de integración para cada aplicación. Estas aplicaciones son una entidad que nos permite entonces describir cada aplicación y a su vez configurar cada entorno donde correrá: producción, pruebas, producción, etc. De esta forma, cada desarrollador se autogestiona sus ambientes.

Aplicaciones en Mercado Pago

En el contexto de Mercado Pago, una aplicación es una entidad que representa una aplicación o plataforma en su sistema. Al crear una aplicación en Mercado Pago, se obtienen credenciales y configuraciones específicas que permiten interactuar con su API de manera segura y controlada.

La creación de una aplicación en Mercado Pago es crucial por varias razones:

1. **Seguridad:** Al crear una aplicación, Mercado Pago proporciona claves de acceso, como el client_id y el client_secret, necesarias para autenticar las solicitudes a su

[30] `https://www.mercadopago.com.ar/developers/es`

API. Esto asegura que solo las aplicaciones autorizadas puedan realizar operaciones en nombre de los usuarios.

2. **Control y seguimiento:** La plataforma de Mercado Pago permite realizar un seguimiento detallado de las transacciones y pagos realizados a través de la aplicación. Esto es fundamental para mantener un registro de las operaciones y resolver posibles problemas.

3. **Configuración personalizada:** Es posible ajustar la aplicación según las necesidades específicas del proyecto, como las notificaciones de pagos, las URL de retorno y otros parámetros relevantes.

4. **Facilita la escalabilidad:** Una vez registrada la aplicación en Mercado Pago, es posible ampliar las operaciones y ofrecer servicios de pago de forma más eficiente. Esto es especialmente útil en el desarrollo de plataformas de comercio electrónico o aplicaciones que manejan transacciones financieras.

A continuación se describe el proceso de gestión de aplicaciones desde su portal.

Autogestión de aplicaciones en el portal de Mercado Pago

La aplicación de Mercado Pago ofrece una interfaz completa y organizada que simplifica la configuración y el monitoreo de sus integraciones. Las principales secciones de para una aplicación serían:

1. **Información General:** En esta sección Mercado Pago ofrece un espacio con información clave como el nombre y User ID, número de aplicación y que tipo de Pagos Online utiliza (Link de Pago o Suscripciones).

2. **Notificaciones:** Aquí se ofrece información sobre transacciones en tiempo real sobre pagos, cancelaciones, devoluciones y otros eventos importantes.

   Existen dos tipos de notificaciones disponibles que, una vez configuradas, permiten que nuestro desarrollo sea notificado acerca del estado de las suscripciones. Estas notificaciones pueden ser de dos tipos por el momento, **Webhooks**[31] o **IPN (Instant Payment Notification)**[32].

   i. **Webhooks:** Es una forma de comunicación entre dos aplicaciones diferentes para anoticiar sobre eventos que ocurren en una de ellas a la otra. Generalmente, la comunicación se hace por HTTP, más específicamente POST. Esto permite que la aplicación receptora procese los datos y realice acciones específicas en respuesta al evento recibido. Desde la documentación de Mercado Pago tendremos distintos tipos de notificaciones y acciones. No hay garantía de entrega. La aplicación receptora del webhook debe manejar posibles fallas de entrega

[31] https://www.mercadopago.com.ar/developers/es/docs/your-integrations/notifications/webhooks
[32] https://www.mercadopago.com.ar/developers/es/docs/your-integrations/notifications/ipn

y reintentos, si es necesario. En base a seguridad depende de cómo se implementan, a menudo se utilizan tokens o firmas HMAC[33] para verificar la autenticidad de las solicitudes entrantes.

ii. **IPN:** Es un método de comunicación en sistemas web que se utiliza específicamente en el contexto de pagos en línea, como en plataformas de comercio electrónico. En este caso se notifica a la aplicación vendedora sobre el estado de una transacción, como por ejemplo la confirmación de pago o reembolso, para que pueda tomar acciones correspondientes. Se considera más confiable en comparación con los Webhooks, ya que los servicios de pago generalmente realizan varios intentos para entregar el IPN, garantizando así la notificación de la transacción. Generalmente, requiere autenticación mediante un mecanismo basado en claves o certificados SSL[34], proporcionados por la plataforma de pagos, para garantizar que solo las notificaciones legítimas sean aceptadas por la aplicación vendedora.

3. **Prueba:** En esta sección, Mercado Pago proporciona no solo las credenciales de prueba necesarias para integrar una aplicación, sino también la opción de crear usuarios de prueba y utilizar tarjetas de prueba.

4. **Credenciales de Producción:** Dentro de esta sección, Mercado Pago almacena las credenciales requeridas para iniciar la recepción de pagos o transacciones reales.

Con objeto de clarificar los tipos de notificaciones ofrecidos por la plataforma, a continuación explicamos más en detalle cada uno de ellos.

Notificaciones de tipo Webhook

Siendo una pieza esencial en la integración de sistemas, los webhooks facilitan la comunicación asincrónica entre aplicaciones.

En Mercado Pago, se encuentran disponibles diversos tipos de notificaciones webhook. Estas notificaciones abarcan eventos donde una plataforma envía a endpoints específicos configurados por los usuarios, informando sobre cambios significativos en transacciones, pagos y otros eventos relevantes.

[33] https://cloud.google.com/storage/docs/authentication/hmackeys
[34] https://www.cloudflare.com/es-es/learning/ssl/what-is-an-ssl-certificate/

Tipo de notificación	Acción	Descripción
`payment`	`payment.created`	Creación de pagos
`payment`	`payment.updated`	Actualización de pago
`mp-connect`	`application.deauthorized`	Desvinculación de cuenta
`mp-connect`	`application.authorized`	Vinculación de cuenta
`subscription_preapproval`	`created - updated`	Suscripción
`subscription_preapproval_plan`	`created - updated`	Plan de suscripción
`subscription_authorized_payment`	`created - updated`	Pago recurrente de una suscripción
`point_integration_wh`	`state_FINISHED`	Intento de pago finalizado
`point_integration_wh`	`state_CANCELED`	Intento de pago cancelado
`point_integration_wh`	`state_ERROR`	Ocurrió un error al procesar el intento de pago
`delivery`	`delivery.updated`	Datos de envío y actualización de pedidos
`delivery_cancellation`	`case_created`	Solicitud de cancelación de envío
`topic_claims_integration_wh`	`updated`	Reclamos hechos por las ventas

Cada tipo de notificación cumple propósitos distintos y resulta crucial para mantener actualizado un sistema con respecto a las actividades financieras y transacciones en una cuenta de Mercado Pago.

Ante un evento en Mercado Pago, se enviará un mensaje HTTP POST a cada endpoint configurado con datos que pueden usarse por el receptor como considere necesario. En general, se actualiza el estado ante determinada situación.

Notificaciones IPN

En el ámbito específico de Mercado Pago también existen las notificaciones IPN que juegan un papel fundamental al automatizar procesos relacionados con transacciones, pagos y actualizaciones de estado.

Estas notificaciones comprenden eventos que la plataforma envía a URLs específicas configuradas por los usuarios, informando sobre cambios significativos en transacciones, pagos y otros eventos relevantes.

Campo	Descripción
`topic`	Identifica cuál es el recurso, puede ser `payment`, `chargebacks`, `merchant_order` o `point_integration_ipn`.
`id`	Es un identificador único del recurso notificado.

Cada tipo de notificación IPN tiene propósitos específicos, siendo el campo "topic" el más relevante, ya que permite identificar distintos tipos de recursos. Esta diferenciación es fundamental, especialmente en comparación con los webhooks, donde los tipos y acciones ya están separados de manera más clara.

Flujos de pago

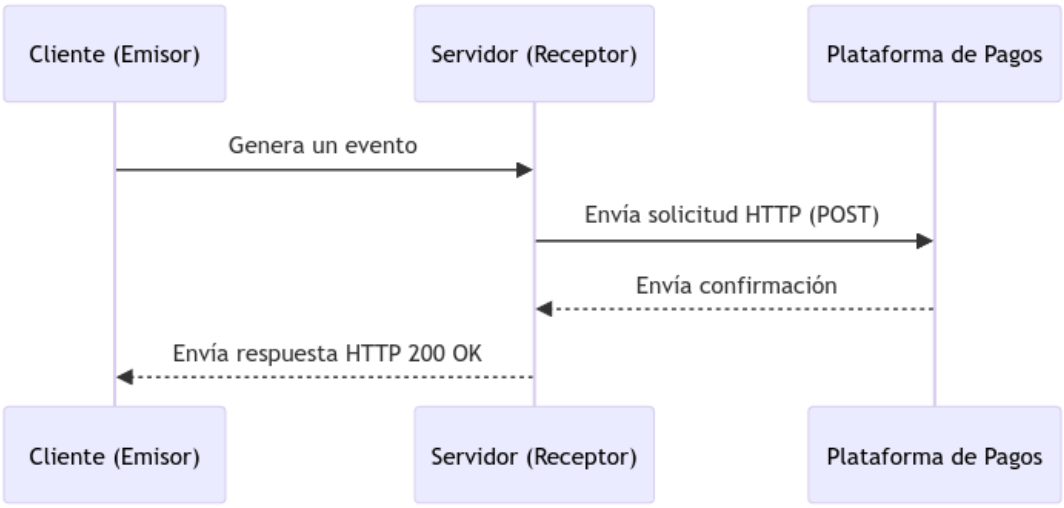
En los siguientes gráficos se presentan detalladamente los procesos de pago al integrar Mercado Pago con una aplicación. Se hace énfasis en que la elección entre notificaciones IPN o Webhooks es irrelevante, ya que ambos métodos ofrecen herramientas esenciales para gestionar notificaciones de manera eficiente.

Lo interesante del siguiente análisis radica en comprender de qué forma una aplicación desarrollada en cualquier lenguaje o framework, interactúa con las API de Mercadopago. Los escenarios son deterministicos, pero dependen de la aprobación final del pago o no. A continuación veremos cada caso.

Flujo Exitoso

Este diagrama de secuencia representa el flujo de interacción entre un cliente (emisor), un servidor (receptor) y una plataforma de pagos.

- **Cliente (Emisor):** El proceso comienza con el cliente generando un evento, que puede ser una acción como realizar una compra en línea o solicitar un servicio.
- **Servidor (Receptor):** Actúa como receptor de la solicitud del cliente, recibe la información del evento a través de una solicitud HTTP, generalmente POST.
- **Plataforma de Pagos:** Una vez que el servidor recibe la solicitud del cliente, la reenvía a la plataforma de pagos para su procesamiento.Se encarga de verificar la transacción, autorizar el pago y llevar a cabo cualquier otra acción necesaria para completar la solicitud.
- **Confirmación:** Después de procesar la solicitud, la plataforma de pagos envía una confirmación al servidor para indicar que la transacción ha sido completada con éxito.
- **Respuesta al Cliente:** Finalmente, el servidor envía una respuesta HTTP 200 OK de vuelta al cliente para indicar que la solicitud ha sido procesada correctamente. Esta respuesta puede incluir información adicional sobre el estado de la transacción o cualquier otra acción realizada.



Flujo no exitoso con reintentos

Este diagrama de secuencia representa el flujo de eventos en un sistema de pagos en línea, con especial atención a cómo se manejan los reintentos y las notificaciones de pausa de suscripción.

- **Cliente (Emisor):** Se inicia un evento de pago en el sistema.

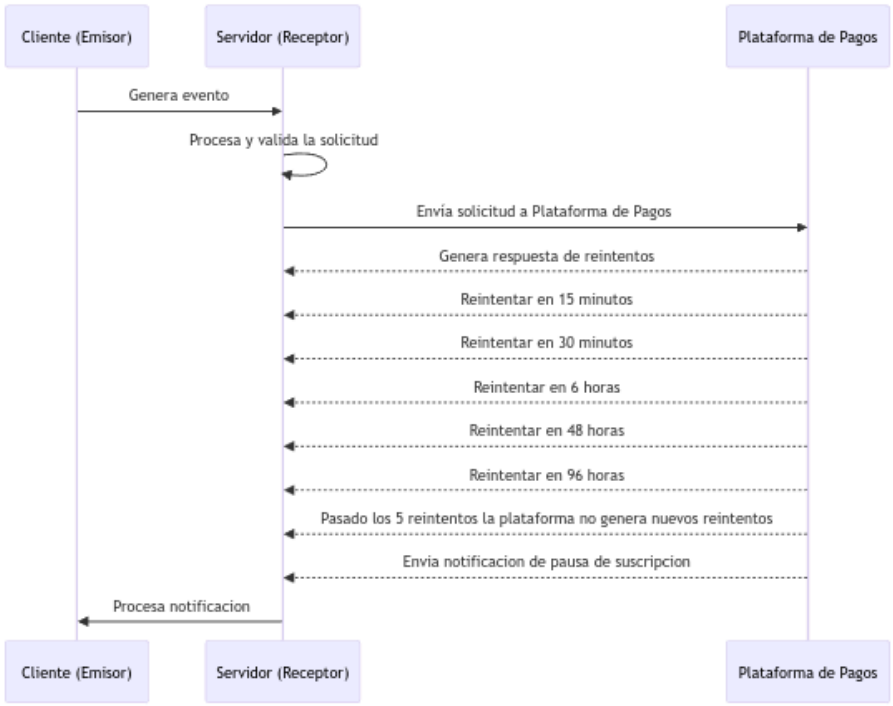
- **Servidor (Receptor):** Recibe y procesa las solicitudes de pago enviadas por el cliente. Durante este proceso, el servidor valida y verifica la solicitud de pago antes de enviarla a la plataforma de pago. Si la validación es exitosa, el servidor envía el evento a la plataforma de pago. En caso contrario, notifica al cliente sobre la falla en alguna validación.

- **Plataforma de Pagos:**Facilita la transacción financiera entre el cliente y el servidor. Después de que el servidor haya procesado y validado la solicitud de pago, la envía a la plataforma de pagos para su procesamiento.

- **Generación de Reintentos:** Si la plataforma de pagos detecta un problema durante el procesamiento de la solicitud de pago, como un error de comunicación o un fallo temporal, generará una respuesta de reintentos. Esta respuesta incluye una serie de reintentos programados en intervalos específicos, como 15 minutos, 30 minutos, 6 horas, 48 horas y 96 horas, que indican cuándo se volverá a intentar procesar la solicitud.

- **Notificación al Cliente:** Después de haber agotado los reintentos programados (en este caso, 5 reintentos), la plataforma de pagos no generará nuevos reintentos. En este punto, es responsabilidad del servidor notificar al cliente que su transacción no

pudo ser procesada por la plataforma de pagos. Esta situación podría deberse a la falta de fondos suficientes en la cuenta del cliente.



Dada la estructura y los requerimientos del proyecto, se tomó la decisión de emplear notificaciones del tipo Webhook en lugar de las notificaciones IPN. Esta elección se sustenta en las reglas de negocio establecidas, las cuales dictaminan que las notificaciones webhook ofrecen una gama más amplia de tipos de notificaciones disponibles, proporcionando una mayor flexibilidad y capacidad de adaptación a los diferentes eventos y necesidades específicas del sistema.

SDK de Mercado Pago

Una de las características distintivas de Mercado Pago es su robusta infraestructura tecnológica, respaldada por una variedad de Software Development Kits (SDK) que proporcionan a desarrolladores y empresas, herramientas para integrar sus servicios de pago en una amplia variedad de aplicaciones y plataformas. Estos SDK, diseñados con un enfoque en la simplicidad, la seguridad y la escalabilidad.

Además de proporcionar una interfaz sencilla para interactuar con las APIs de Mercado Pago, éstas ofrecen una serie de ventajas adicionales que impulsan la eficiencia y reducen costos y serían:

- **Mayor eficiencia y reducción de costos:** Se adopta una herramienta lista para usar y homologada por Mercado Pago.

- **Instalación optimizada:** Se ofrece una fácil instalación a sistemas de gestión de paquetes, como Composer, Gradle, Maven y NPM. Agilizando la integración en algún proyecto y en tiempos de desarrollo.

- **Construcción de requisiciones simplificadas:** Se reduce la complejidad optimizando el desarrollo y permitiendo una implementación eficaz, sin errores.

- **Seguridad en el tratamiento de datos:** Se fortalece la seguridad en el tratamiento de los datos, utilizando las mejores prácticas de seguridad para proteger la información en su envío y recepción.

Los SDKs de Mercado Pago están disponibles en distintos lenguajes de programación, siendo estos: PHP, Java, NodeJS, .NET, Python, Ruby y Go

SDK	Versiones compatibles	Descarga	Documentación
PHP 3.0.0	PHP 8.2 o superior	Packagist	GitHub
Java 2.1.14	Java 1.8 o superior	Maven	GitHub
NodeJS 2.0.0	Node.js 12 o superior	NPM	GitHub
Ruby 2.2.0	Ruby 2.3+	Gem	GitHub
.NET 2.3.3	.NET Standard 2.0+ .NET Core 2.0+ .NET Framework 4.6.1+	NuGet	GitHub
Python 2.2.1	Python 3+	PyPI	GitHub
Go 1.0.0	Go 1.18+	Go	GitHub

A continuación explicamos más en detalle del SDK escrito en Ruby.

Gema de Mercado Pago

Es esencial destacar que el repositorio de GitHub de la gema de Mercado Pago en Ruby[35] es un proyecto de código abierto. Esto significa que está abierto a recibir contribuciones de la comunidad de desarrolladores. Los usuarios tienen la posibilidad de enviar pull requests con mejoras, nuevas funcionalidades o correcciones de errores. El equipo de mantenimiento del proyecto revisará estas solicitudes y las aprobará o rechazará en función de su pertinencia y calidad.

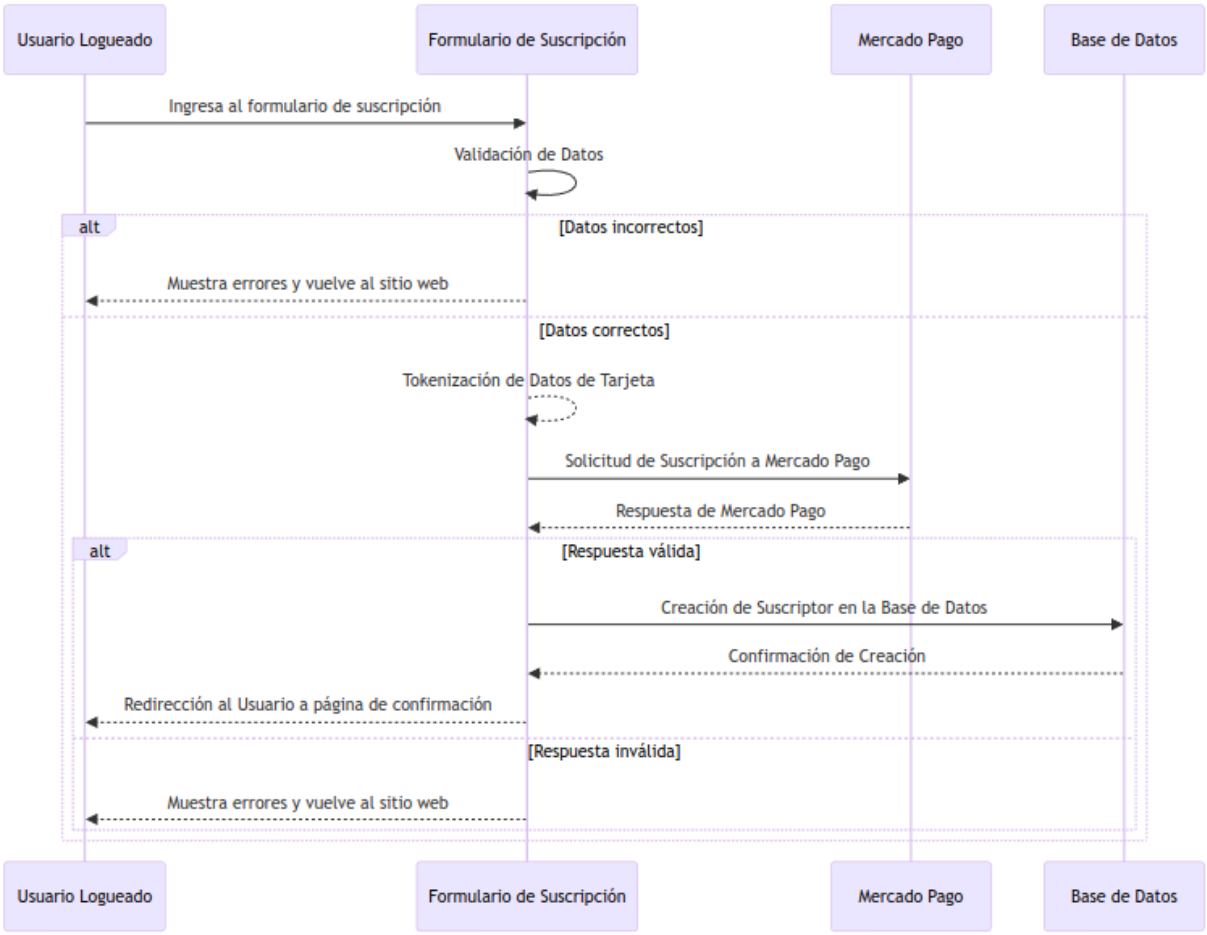
Además, este modelo de desarrollo colaborativo permite que los usuarios reporten problemas o bugs encontrados en la gema. Los desarrolladores pueden abrir issues en el repositorio de GitHub para informar sobre errores, problemas de rendimiento o sugerencias de mejoras. Esto fomenta una comunidad activa que trabaja en conjunto para mejorar y mantener la calidad del proyecto.

[35] `https://github.com/mercadopago/sdk-ruby`

Es relevante mencionar que la gema se encuentra actualmente en su versión 2.0. Esta última versión se lanzó hace 3 años. Los usuarios pueden acceder a la documentación y las instrucciones de instalación más recientes en el repositorio de GitHub para garantizar que estén utilizando la versión más actualizada y optimizada de la gema.

Cómo se utilizó la librería en el proyecto banda invitada

A continuación mostramos un flujo detallado que describe cómo un usuario logueado ingresa sus datos, se valida, realiza la suscripción a través de Mercado Pago, y recibe una confirmación de suscripción exitosa. Cada paso se evalúa para manejar posibles errores, garantizando una experiencia fluida para el usuario.



1. **Usuario Logueado (User):** El proceso comienza con un usuario que ya está logueado en la plataforma y desea suscribirse a un servicio.

2. **Formulario de Suscripción (Form):** El usuario ingresa al formulario de suscripción donde proporciona los datos necesarios para completar la suscripción.

3. **Validación de Datos:** Se validan los datos ingresados por el usuario para garantizar el formato correcto.
    a. Si los datos son incorrectos, el formulario muestra errores y vuelve al sitio web (paso alternativo).

   b. Si los datos son correctos, el proceso continúa.

4. **Tokenización de Datos de Tarjeta:** Para garantizar la seguridad de los datos de la tarjeta del usuario, el formulario procede a tokenizar estos datos.

5. **Solicitud de Suscripción a Mercado Pago (MP):** Luego del formulario se realiza una solicitud con los datos necesarios para que Mercado Pago pueda crear una suscripción.

6. **Respuesta de Mercado Pago:** La solicitud y los datos se procesan en Mercado Pago y, según la respuesta, se siguen dos cursos de acción diferentes
   a. Si la respuesta de Mercado Pago es válida, el proceso continúa.
   b. Si la respuesta de Mercado Pago es inválida, el formulario muestra errores y vuelve al sitio web (paso alternativo).

7. **Creación de Suscriptor en la Base de Datos (DB):** Si la respuesta de Mercado Pago es válida, se crea un registro de suscriptor en la base de datos de tu plataforma.

8. **Confirmación de Creación:** La base de datos envía una confirmación al formulario para indicar que la creación del suscriptor ha sido exitosa.

9. **Redirección al Usuario:** Se redirige al usuario a una página de confirmación donde se le informa que la suscripción se ha completado exitosamente.

Presentado el flujo, puede observarse que existen dos posibles respuestas de la API de Mercado pago: la exitosa y la inválida. Ambas se representan como una llamada desde Mercado pago hacia nuestra aplicación por medio de Webhooks. A continuación entonces presentamos las integraciones necesarias para recibir tales endpoints en nuestro desarrollo.

Configuración de Webhooks en un proyecto Rails

Para que nuestro desarrollo reciba los webhook desde Mercado Pago, es necesario configurar un endpoint HTTP en nuestra aplicación Rails, lo cuál se traduce en crear un controlador y una ruta dedicada para esta tarea.

El código genérico de un controlador encargado de gestionar el webhook para Mercado Pago puede ser implementado de la siguiente manera:

```ruby
class WebhookMercadoPagoController < ApplicationController
  def process_webhook
    WebhookMercadoPagoService.new(params).call

    render status: :ok, json: params.to_json
  end
end
```

Además, es necesario agregar una ruta en Rails que redirige al controlador mencionado cuando se recibe un método HTTP POST llamado a la ruta /webook_mercado_pago:

```
Rails.application.routes.draw do
 post 'webhook_mercado_pago', to:
'webhook_mercado_pago#process_webhook'
end
```

El objeto utilizado en el controlador, invoca al método #call del objeto, siguiendo la filosofía de middlewares en Ruby basada en otra gema llamada Rack[36]. En este objeto se manipula la lógica de negocio en base a los tipos de notificaciones que se reciban::

```
class WebhookMercadoPagoService
  def initialize(response)
    @type = response[:webhook_mercado_pago][:type]
    @action = response[:webhook_mercado_pago][:action]
  end

  def call
    Rails.logger.info @response.to_json
    case @type
    when 'payment'
      # Realiza una acción específica si es crear o actualizar
    when 'subscription_preapproval_plan'
      # Realiza una acción específica si es crear o actualizar
    when 'subscription_preapproval'
      # Realiza una acción específica si es crear o actualizar
    when 'subscription_authorized_payment'
      # Realiza una acción específica si es crear o actualizar
    else
      Rails.logger.debug "La acción es: #{@type}"
    end
  end
end
```

Observaciones acerca de la experiencia con la API de Mercadopago

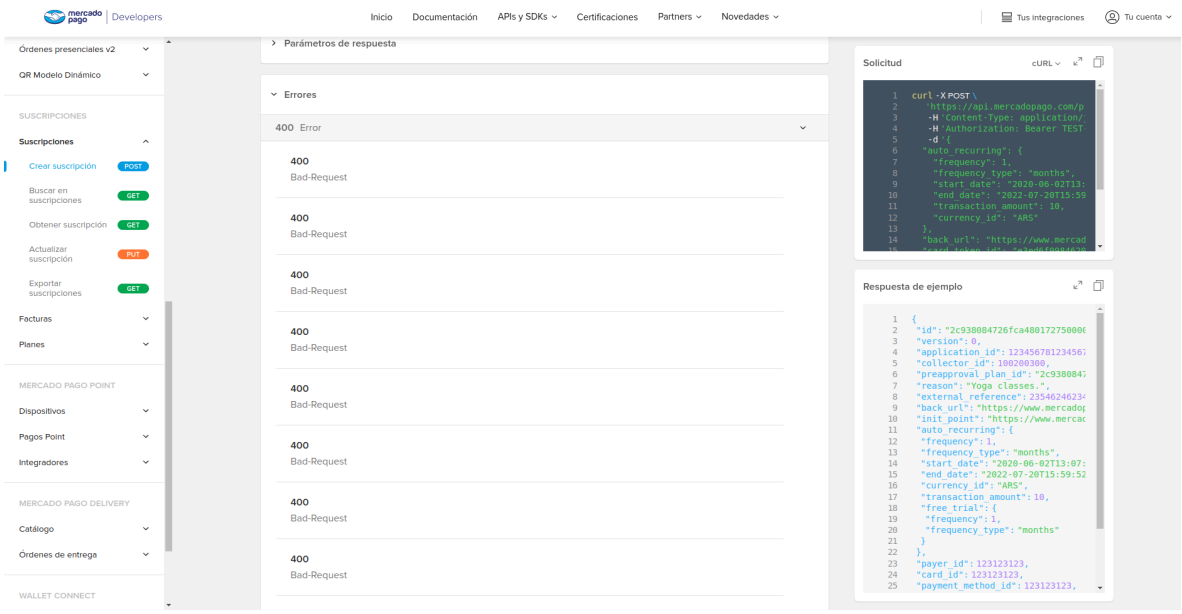
A continuación describimos la experiencia de nuestro equipo de trabajo en relación a las implementaciones de la librería mencionada en diferentes proyectos.

[36] https://github.com/rack/rack

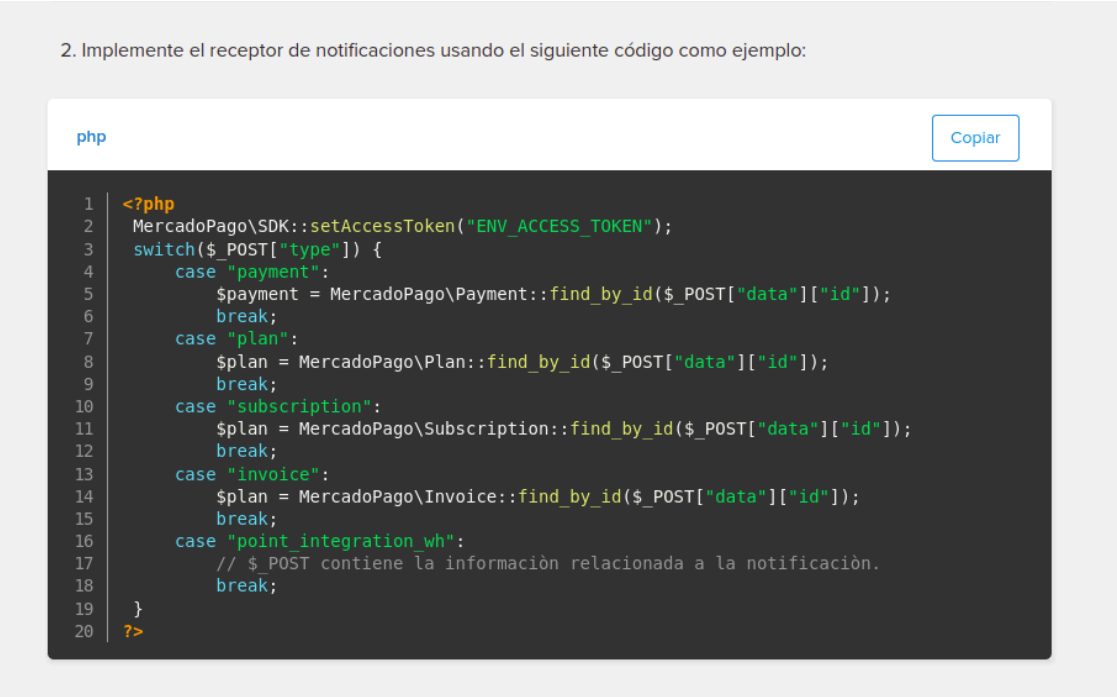
Escasa documentación de Mercado Pago

A pesar de la robustez y flexibilidad que ofrece la API de Mercado Pago, es importante destacar algunas limitaciones y desafíos que los desarrolladores pueden encontrar durante el proceso de integración. Estas limitaciones las encontramos en torno a la documentación, la disponibilidad de ejemplos de uso para ciertos lenguajes y el soporte técnico proporcionado:

1. **Documentación incompleta:** En algunos casos, la documentación proporcionada por Mercado Pago puede estar incompleta o no detallar suficientemente los endpoints disponibles y sus parámetros. Esto puede dificultar la comprensión y la implementación adecuada de la API.



2. **Falta de ejemplos para algunos lenguajes:** Aunque Mercado Pago ofrece soporte para varios lenguajes de programación, es posible que algunos de ellos carezcan de ejemplos de código y casos implementados. Esto puede representar un obstáculo para los desarrolladores que trabajan en un entorno específico. Ej: Receptor de notificaciones solamente para PHP



2. Implemente el receptor de notificaciones usando el siguiente código como ejemplo:

php

Copiar

```php
<?php
 MercadoPago\SDK::setAccessToken("ENV_ACCESS_TOKEN");
 switch($_POST["type"]) {
     case "payment":
         $payment = MercadoPago\Payment::find_by_id($_POST["data"]["id"]);
         break;
     case "plan":
         $plan = MercadoPago\Plan::find_by_id($_POST["data"]["id"]);
         break;
     case "subscription":
         $plan = MercadoPago\Subscription::find_by_id($_POST["data"]["id"]);
         break;
     case "invoice":
         $plan = MercadoPago\Invoice::find_by_id($_POST["data"]["id"]);
         break;
     case "point_integration_wh":
         // $_POST contiene la informaciòn relacionada a la notificaciòn.
         break;
 }
?>
```

3. **Escasez de soluciones técnicas:** En ciertos casos, el soporte técnico brindado por Mercado Pago puede no proporcionar suficientes soluciones a problemas técnicos específicos que los desarrolladores puedan encontrar durante la integración. Esto puede requerir que los desarrolladores busquen soluciones por su cuenta, lo que podría retrasar el proceso de desarrollo.

Problemas asociados a la gestión de proyectos

Los problemas en la gestión de proyectos pueden surgir por diversas razones, y uno de ellos se asocia al desaprovechamiento del trabajo repetitivo. Si el mismo problema se aborda diferente en cada proyecto, entonces damos lugar a la falta de coherencia y consistencia. Reutilizar código en la implementación de tecnologías o prácticas específicas, maximiza el ROI, mejora la comunicación del equipo y minimiza el tiempo de solución de problemas.

Desde la perspectiva de la empresa, se observó que varios equipos de desarrollo tomaban decisiones divergentes respecto a la implementación de los webhooks en sus implementaciones con la librería de Mercadopago en proyectos Ruby on Rails. Esta situación no solo genera inconsistencias en el código, sino que también contraviene con el principio DRY.

Además de lo antes expuesto, las soluciones actuales no son fácilmente mantenibles. A menudo, las soluciones implementadas que no reutilizan código probado y robusto, promueven arquitecturas inadecuadas que no se adaptan y crecen con las necesidades del proyecto a largo plazo. Esto no solo generaba dificultades en la gestión y mantenimiento del código, sino que también limitaba la capacidad de la empresa para escalar y evolucionar de manera eficiente. Otros problemas derivan de la velocidad en el desarrollo, afectada por implementar múltiples veces la misma funcionalidad en diferentes proyectos.

Es importante abordar estos problemas con un enfoque integral que incluya una revisión exhaustiva de las prácticas de desarrollo, la estandarización de los procesos y la implementación de herramientas y metodologías que promuevan la coherencia, la escalabilidad y la facilidad de mantenimiento en todos los aspectos del proyecto. Esto puede implicar la adopción de buenas prácticas de diseño, la capacitación del personal y la implementación de procesos de revisión de código y pruebas automatizadas para garantizar la calidad y coherencia del código en todo momento.

Problemas con los ambientes de desarrollo

Al desarrollar utilizando la API de Mercado Pago, surge una dificultad para identificar los pagos recurrentes de todas las suscripciones activas. Específicamente, el desafío reside en poder recibir las llamadas HTTP POST correspondientes a las notificaciones generadas por Mercado Pago. En general los desarrolladores trabajan detrás de una red LAN que no admite a los servidores de Mercado pago, invocar un webhook que está corriendo en el escritorio de un desarrollador en redes privadas.

Esta complicación pudimos superarla utilizando una herramienta que simplifica el armado de túneles HTTP desde la máquina del desarrollador a servidores públicos que exponen directamente lo que el desarrollador está trabajando. Dicha herramienta es Ngrok[37].

Ngrok

Ngrok es una herramienta útil en el desarrollo de aplicaciones web, ya que posibilita la exposición de un servidor web local a través de una dirección URL pública accesible desde Internet. Este recurso resulta especialmente útil durante las fases de prueba, cuando se requiere que un servicio o aplicación ejecutada en un entorno local sea accesible desde el exterior, como sucede con las notificaciones webhook en Mercado Pago.

Mediante Ngrok, creamos túneles que permiten el acceso a un servidor web local o a una aplicación en desarrollo desde cualquier parte del mundo, a través de una URL pública generada por la herramienta. Esto facilita enormemente el proceso de desarrollo y prueba al brindar una manera rápida y sencilla de compartir y validar el funcionamiento de la aplicación en diferentes entornos.

Esto es beneficioso en varios escenarios, incluyendo:

1. **Pruebas locales:** Permite a los desarrolladores probar sus aplicaciones web en un entorno controlado antes de implementarlas en un servidor público.

2. **Colaboración:** Facilita la colaboración entre equipos de desarrollo al proporcionar un acceso fácil a aplicaciones locales sin necesidad de desplegarlas en un servidor compartido.

[37] https://ngrok.com/docs/what-is-ngrok/

3. **Depuración remota:** Poder depurar problemas en aplicaciones web al permitir a los desarrolladores y colaboradores externos acceder al entorno de desarrollo en tiempo real.

Solución propuesta

Para enfrentar los desafíos encontrados en la gestión de proyectos, tales como la labor repetitiva y la falta de mantenibilidad, se ha creado una gema que amplía las capacidades de la gema de Mercado Pago para manejar notificaciones webhook de manera estructurada y escalable. Estas funcionalidades no estaban disponibles en las bibliotecas existentes diseñadas para el lenguaje de programación utilizado en el proyecto.

Así es como en el proyecto de Banda Invitada, hemos detectado y asumido el desafío de completar en una nueva gema, las funcionalidades ausentes en las bibliotecas existentes de Mercado Pago para ruby. Por lo tanto, nos comprometimos a crear una solución a medida que pudiera satisfacer nuestras necesidades específicas y proporcionar una base sólida para el manejo de notificaciones webhook que sea reutilizable en otros proyectos con necesidades similares.

Gema Webhook MP

El resultado de haber identificado la necesidad de crear una nueva librería, se consolidó en el desarrollo de una gema que es una extensión de la biblioteca original de Mercado Pago. Esta gema provee las funcionalidades necesarias para el manejo de notificaciones por medio de webhooks de una manera estructurada y eficiente, que estandariza la forma en que se implementa esta funcionalidad. Esta extensión se logró aprovechándose de la funcionalidad de un Mixin[38] en Ruby que simplifica la metaprogramación[39].

Mixins en Ruby

Un Mixin en Ruby es una técnica que permite incluir el comportamiento en una clase a partir de un módulo. Sin entrar en los detalles técnicos de cómo se implementan, podemos compararlos con las interfaces existentes en lenguajes Orientados a Objetos como Java o PHP, donde diferentes clases deben respetar cierto prototipo de funciones. Los mixins, permiten a diferencia de las interfaces, la posibilidad de programar código que es mezclado (mixed in) las clases que incluyen el módulo. Esto reduce la cantidad de código y promueve DRY. Además es una estrategia de metaprogramación que agrega funcionalidades de forma implícita a clases existentes en ruby.

En el contexto de la gema Webhook MP, se utiliza un Mixin para extender las capacidades de la gema de Mercado Pago y proporcionar funcionalidades adicionales para el manejo de notificaciones webhook.

---

[38] `https://ruby-doc.com/docs/ProgrammingRuby/html/tut_modules.html`
[39] `https://www.shakacode.com/blog/metaprogramming-in-ruby/`

Funcionalidad de la gema

Una vez que el Mixin se ha integrado en la biblioteca de Mercado Pago, la gema Webhook MP asume la responsabilidad de procesar las notificaciones provenientes de Mercado Pago. Esto implica validar la autenticidad de las notificaciones, analizar los datos recibidos y ejecutar acciones correspondientes según el tipo de evento notificado.

Además, la gema gestiona todas las etapas del ciclo de vida de una notificación webhook, desde su recepción inicial hasta la ejecución de acciones en función del contenido de la notificación. Este enfoque simplifica la integración de la funcionalidad de notificaciones webhook en aplicaciones Ruby on Rails, liberando a los desarrolladores de la complejidad subyacente del procesamiento de eventos de Mercado Pago.

El sistema de notificaciones desempeña un papel fundamental al mantener a nuestros usuarios actualizados en tiempo real sobre el estado de sus transacciones, proporcionándoles una experiencia de usuario que garantiza transparencia en las actividades relacionadas a sus pagos. Esta capacidad de ofrecer información instantánea y relevante contribuye significativamente a la satisfacción del cliente y a la reputación de nuestra plataforma.

La evolución de esta gema desde sus primeras líneas de código hasta convertirse en un componente esencial de un sistema de suscripciones en un entorno de producción es notable. En la actualidad, este sistema de suscripciones es utilizado por varios usuarios que realizan pagos de forma regular, confiando en nuestra solución para gestionar sus transacciones financieras de manera segura y eficiente.

Además, es de destacar que otros proyectos están sumándose a incorporar la gema de Webhook MP, centralizando así la forma en que varios proyectos manejan las suscripciones de forma coherente dentro de nuestra empresa. Por su parte, cabe mencionar que el ciclo de vida de la gema ahora pasa a ser independiente de cada proyecto, permitiendo a los demás proyectos incorporar mejoras de forma mucho más simple dado que se gestiona como dependencia.

Resultados

Una vez implementada, la gema no solo cumplió con nuestras expectativas debido a la simplificación en su adopción cambiando el código del proyecto por el provisto por la gema, sino que también las superó al ofrecer un enfoque centrado en el rendimiento. Esta capacidad nos permitió manejar un gran volumen de notificaciones sin comprometer el rendimiento del sistema, lo que resultó ser fundamental para mantener nuestras aplicaciones en funcionamiento de manera eficiente incluso en momentos de alta carga.

Además, al integrar esta gema en otros proyectos, hemos podido refactorizar código de manera significativa, lo que ha permitido a desarrolladores de diferentes proyectos adoptar la creación de gemas en pos de no repetir código y mejorar la mantenibilidad de nuestras aplicaciones. El resultado se ha manifestado como un aumento en la productividad del equipo y ha sentado las bases para futuras mejoras y expansiones de nuestros sistemas.

Conclusiones

Este proyecto nos ha permitido no solo crear una gema valiosa para nuestra empresa y solucionar el problema puntual de un sistema de suscripciones productivo, sino también crecer como equipo de desarrollo enfrentando desafíos técnicos y conceptuales. Hemos aprendido que la colaboración, la perseverancia y la atención al detalle son fundamentales en el desarrollo de software de calidad. Además, hemos ganado experiencia en los procesos de pago en línea y la importancia de mantener la seguridad y confidencialidad de los datos financieros de nuestros usuarios.

Reflexionando sobre esta experiencia de integrar una aplicación y desarrollar una gema, se aprendieron lecciones invaluables. Experimentado de primera mano la importancia de la documentación clara y la modularidad en el desarrollo de software. Además, el proceso de desarrollo de la gema nos ha llevado a reflexionar sobre la importancia de la calidad del código y prácticas de desarrollo sostenibles. Esta reflexión nos ha llevado a adoptar un enfoque más proactivo hacia la calidad de software en nuestros proyectos futuros.

Reconocemos el valor de compartir nuestro trabajo con otros desarrolladores y permitir que lo utilicen, lo modifiquen y lo mejoren según sus necesidades. Abrir la gema a la comunidad no solo podría llevar a una mayor adopción y contribución por parte de otros desarrolladores, sino que también podría enriquecer nuestra propia comprensión y habilidades al colaborar con una comunidad diversa y talentosa. Esta reflexión no solo nos invita a considerar los principios del código abierto, sino que también a cómo podemos integrarlos en nuestros procesos de desarrollo en el futuro.

Trabajos a Futuro

Mirando hacia el futuro, siempre hay espacio para la mejora y el crecimiento. Algunos de los trabajos a futuro que consideramos incluyen:

1. **Expansión de características:** Explorar la posibilidad de agregar características adicionales, como opciones de pago personalizables, integración con otras pasarelas de pago o manejo de notificaciones IPN.

2. **Documentación completa:** Continuar mejorando y ampliando la documentación de nuestra gema para que otros desarrolladores puedan utilizarla de manera más efectiva.

3. **Seguridad continua:** Mantenernos actualizados con las mejores prácticas de seguridad y asegurarnos de que nuestra solución siga siendo segura y resistente a las amenazas.

4. **Soporte de comunidad:** Proporcionar soporte a la comunidad de desarrolladores que utilice nuestra gema, respondiendo a preguntas y resolviendo problemas. Recibir feedback de otros desarrolladores para futuros desarrollos.

Bibliografía

Recursos en línea

- RubyGems (s.f.) *What is a gem?* https://guides.rubygems.org/what-is-a-gem/
- Mixins (s.f.) *Ruby mixin* https://ruby-doc.com/docs/ProgrammingRuby/html/tut_modules.html
- Ruby Lang (s.f.) https://www.ruby-lang.org/es/
- Ruby on Rails (s.f.) *Rails Api* https://api.rubyonrails.org/
- Ruby on Rails (s.f.) *Ruby on Rails - Framework* https://rubyonrails.org/
- Ruby on Rails (s.f.) *La Doctrina de Rails* https://rubyonrails.org/doctrine
- Ruby on Rails (s.f.) *Ruby on Rails - Guía* https://guides.rubyonrails.org/
- Mercado Pago API (s.f.) *Introduccion* https://www.mercadopago.com.ar/developers/es/reference
- Mercado Pago (s.f.) *Notificaciones* https://www.mercadopago.com.ar/developers/es/docs/your-integrations/notifications
- Mercado Pago (s.f.) *Notificaciones: Webhook* https://www.mercadopago.com.ar/developers/es/docs/your-integrations/notifications/webhooks
- Mercado Pago (s.f.) *Notificaciones: IPN* https://www.mercadopago.com.ar/developers/es/docs/your-integrations/notifications/ipn
- Mercado Pago (s.f.) *Documentaciones* https://www.mercadopago.com.ar/developers/es/docs
- Open Source Interactive (s.f.) *The MIT License* https://opensource.org/license/mit
- Ngrok (s.f.) *What is ngrok?* https://ngrok.com/docs/what-is-ngrok/
- Developer Mozilla (s.f.) *MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web* https://developer.mozilla.org/es/docs/Glossary/MVC
- Firmas HMAC (s.f.) *Claves HMAC* https://cloud.google.com/storage/docs/authentication/hmackeys?hl=es-419#:~:text=Una%20clave%20HMAC%20es%20un,servicio%20autorizan%20una%20solicitud%20determinada.
- SSL (s.f.) *¿Qué es un certificado SSL?* https://www.cloudflare.com/es-es/learning/ssl/what-is-an-ssl-certificate/
- Rack (s.f.) *Rack - A modular Ruby web server interface* https://github.com/rack/rack

Libros

Libros sobre Webhooks

- Van Rousselt, Rick. "Incoming webhooks". En Pro *Microsoft Teams Development*, 243–74. Berkeley, CA: Apress, 2020. DOI: 10.1007/978-1-4842-6364-8_13.
- Boonstra, Lee. "Creating Fulfillment Webhooks". En *The Definitive Guide to Conversational AI with Dialogflow and Google Cloud*, 203–67. Berkeley, CA: Apress, 2021. DOI: 10.1007/978-1-4842-7014-1_10.
- Biswas, Nabendu. "Using Webhooks at the Site". En *Advanced Gatsby Projects*, 133–47. Berkeley, CA: Apress, 2021. DOI: 10.1007/978-1-4842-6640-3_4.

Libros sobre Ruby on Rails

- "Practical Object-Oriented Design in Ruby: An Agile Primer" (Addison-Wesley Professional Ruby). Disponible en: PDF
- Pine, C. (2018). Aprenda a programar (2.ª edición).
- Olsen, R. (2011). Rubí elocuente.
- Flanagan, D., & Matsumoto, Y. (2008). El lenguaje de programación Ruby.
- Hartl, M. (2015). Tutorial de Ruby on Rails (3.ª edición).