



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO E IMPLEMENTACIÓN DE UN MÓDULO GENÉRICO PARA LA
INTEGRACIÓN DE MEDIOS DE PAGO ELECTRÓNICOS EN EL FRAMEWORK
MOQUI

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

TOMÁS IGNACIO RODRÍGUEZ DIAS

PROFESOR GUÍA:
ANDRÉS MUÑOZ ÓRDENES

PROFESOR CO-GUÍA:
JENS HARDINGS PERL

MIEMBROS DE LA COMISIÓN:
JOSÉ URZÚA REINOSO
CECILIA BASTARRICA PIÑEYRO

SANTIAGO DE CHILE
2025

Resumen

El presente informe describe el diseño, desarrollo e implementación de un módulo de integración de medios de pago electrónicos en el *framework* empresarial Moqui, utilizado por la empresa Moit SpA. Este *framework* de código abierto ofrece un conjunto extensible de herramientas para construir soluciones empresariales, pero no incluye de forma nativa una solución general para pagos electrónicos con proveedores modernos. Frente a la creciente demanda de soluciones seguras y confiables de pago en línea en Latinoamérica, se desarrolló un módulo basado en la pasarela MercadoPago, con el objetivo de establecer una base reutilizable para futuras integraciones de otros proveedores como PayPal o WebPay Plus.

Durante la fase de investigación se analizó en profundidad la arquitectura de Moqui y su modelo de datos (UDM), así como la capa de servicios estándar (USL). Se identificaron entidades clave como *Payment*, *PaymentMethod*, *PaymentGatewayConfig*, *PaymentApplication* y *PaymentGatewayResponse*, y se revisaron los servicios asociados (*authorize#Payment*, *capture#Payment*, entre otros). Asimismo, se exploró el mecanismo *SystemMessage* para la integración de mensajes externos, como las confirmaciones de pago.

El desarrollo contempló dos enfoques de integración: una implementación independiente del núcleo financiero del ecosistema Moqui, tipo *plug and play*, y una integración acoplada a Mantle, el módulo de artefactos de negocio estándar de Moqui. En ambos casos, se empleó el servicio Checkout Pro de MercadoPago, lo cual evitó la necesidad de construir una interfaz propia de pago y permitió concentrar los esfuerzos en la lógica de *backend*. Se utilizaron las bibliotecas oficiales de MercadoPago y se definieron entidades y servicios personalizados en Moqui para representar clientes, sesiones de pago y transacciones, además de gestionar el flujo completo de pago, incluyendo la validación segura de notificaciones asíncronas mediante *webhooks* con firma HMAC-SHA256.

La solución resultante permite gestionar pagos electrónicos a través de MercadoPago de forma funcional y adaptable. La versión independiente ofrece una alternativa simple y rápida de desplegar, mientras que la integración con Mantle asegura una alineación completa con la arquitectura nativa de Moqui.

Este trabajo sienta una base sólida para incorporar nuevas pasarelas de pago en el futuro y proporciona una plantilla replicable para desarrollos similares en el ecosistema Moqui.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi familia, por su apoyo incondicional y por brindarme siempre la posibilidad de estudiar, crecer y seguir adelante.

También agradezco sinceramente a los docentes que han sido parte fundamental de mi formación, no solo académica, sino también personal y profesional.

Finalmente, gracias a mis amigos y compañeros por su compañía, sus palabras de aliento y los momentos compartidos a lo largo de este camino.

Tabla de Contenido

1. Introducción	1
1.1. Moit	1
1.2. El <i>framework</i> Moqui	1
1.3. Motivación del proyecto	2
1.4. Objetivos	2
1.4.1. Objetivo General	2
1.4.2. Objetivos Específicos	3
2. Preparación	4
2.1. Estudio del <i>framework</i> Moqui	4
2.2. Análisis de los módulos Mantle	5
2.3. Relación de Investigación con Objetivo general	8
2.4. Revisión de implementaciones existentes	8
2.4.1. Patrones reutilizables	9
2.5. Estudio de proveedores de pago	10
2.5.1. Modalidades de integración	10
2.5.2. Estudio de su API y SDK	11
2.5.3. Otros: Webpay Plus y PayPal	12
2.5.4. Comparación general de modelos de <i>hosted checkout</i>	12
3. Diseño, Desarrollo e Implementación	14
3.1. Módulo Independiente	14

3.1.1. Modelo de Datos (Entidades)	15
3.1.2. Servicios Principales	18
3.2. Configuración autenticación Webhook	28
3.2.1. Configuración para probar Webhook	31
3.3. Adaptación/Integración con Mantle	32
3.3.1. Arquitectura general	32
3.3.2. Modelo de Datos (Entidades)	34
3.3.3. Servicios Principales	36
3.3.4. Prueba de funcionamiento usando MarbleERP	40
4. Conclusión	44
4.1. Retrospectiva	44
4.2. Trabajo a futuro	45
Bibliografía	46
ANEXO A. Configuraciones Mercado Pago	48
ANEXO B. Código de la implementación independiente de Mantle	51
B.1. MoquiConf.xml	51
B.2. entity/MoitMPPaymentEntities.xml	51
B.3. data/MoitPaymentSetup.xml	54
B.4. service/moit/payments/MercadoPagoServices.xml	54
B.4.1. getOrCreate#MercadoPagoCustomer Service	54
B.4.2. create#MercadoPagoCheckout Service	57
B.4.3. process#MercadoPaymentEvent Service	61
B.4.4. consume#MercadoPagoWebhookEvent Service	62
ANEXO C. Código de la implementación integrada a Mantle	65
C.1. MoquiConf.xml	65

C.2. entity/CheckoutEntityExt.xml	65
C.3. data/MPinitialEnumerations.xml	66
C.4. data/MPinitialGatewayData.xml	66
C.5. data/MPsetupWebhook.xml	67
C.6. service/moit/payments/MercadoPagoServices.xml	67
C.7. service/moit/payments/MercadoPagoCore.xml	68
C.7.1. create#MercadoPagoCheckout Service	68
C.7.2. process#MercadoPagoPaymentEvent Service	71
C.8. service/moit/payments/MercadoPagoServices.xml	75
C.8.1. consume#MercadoPagoWebhookEvent Service	75
C.8.2. authorize#MercadoPago Service	77
C.8.3. capture#MercadoPago Service	79
C.8.4. refund & release#MercadoPago templates	80

Índice de Tablas

2.1. Comparación entre Checkout Pro (Mercado Pago), Webpay Plus (Transbank) y PayPal Checkout	13
--	----

Índice de Ilustraciones

2.1. Diagrama de entidades UDM de Mantle relacionadas a método de pago . . .	6
2.2. Diagrama de entidades UDM de Mantle relacionadas a pagos	7
3.1. Diagrama ER del modelo de datos interno para MercadoPago (entidades clave y relaciones).	16
3.2. Diagrama de flujo de pago	33
3.3. Configuración de Store en MarbleERP.	41
3.4. Detalle de orden en MarbleERP.	41
3.5. Autorización de pago en MarbleERP.	42
3.6. Formulario de pago en MercadoPago.	42
3.7. Pago exitoso en MercadoPago.	42
3.8. Historial de estados de pago en MarbleERP.	43
3.9. Historial de respuestas de la pasarela en MarbleERP.	43
A.1. Configuración de la integración con Mercado Pago	48
A.2. Credenciales de Mercado Pago	49
A.3. Configuración de webhook de Mercado Pago	50

Capítulo 1

Introducción

1.1. Moit

Moit¹ es una empresa dedicada a la implementación de soluciones empresariales utilizando principalmente el *framework* de código abierto Moqui. A lo largo del tiempo, Moit ha desarrollado diversos componentes, mejoras y contribuciones a la comunidad Moqui, posicionándose como uno de los referentes regionales en esta tecnología.

Dentro de sus soluciones destacan Moit ERP (integrado con el Servicio de Impuestos Internos de Chile), Moit DTE (gestión y control de documentos tributarios electrónicos) y Moit Remuneraciones (gestión de personas e integración con Previred). Estas son soluciones complejas integradas con Moqui que utilizan sus componentes de manera completa y aprovechando muchas de las funcionalidades del ecosistema.

1.2. El *framework* Moqui

Moqui es un *framework* de código abierto para la construcción de aplicaciones empresariales, escrito en Java. Este *framework* provee un ecosistema completo que incluye un motor de entidades, un motor de servicios y componentes de negocio predefinidos, agrupados bajo el nombre *Mantle Business Artifacts*. Su diseño modular y extensible permite desarrollar sistemas robustos y adaptables a diversas necesidades de negocio.

Entre sus componentes más destacados se encuentran el *Universal Data Model* (UDM), que define un conjunto de entidades comunes para representar clientes, productos, pagos, facturas, entre otros; y la *Universal Service Library* (USL), que contiene servicios reutilizables para tareas empresariales típicas como la gestión de pedidos, inventario o contabilidad. Además, Moqui proporciona un sistema de pantallas (*screens*) para la interfaz de usuario y mecanismos de integración con tecnologías modernas, como servicios REST (*Representational State Transfer*), mensajería y conectores externos.

¹Moit SpA (ver <https://moit.cl>).

1.3. Motivación del proyecto

La transformación digital ha impulsado a las empresas a adoptar nuevos canales de venta y atención, priorizando plataformas digitales y medios no presenciales. En este contexto, el comercio electrónico ha crecido significativamente, tanto por su conveniencia como por la necesidad de adaptarse a escenarios como la pandemia mundial, que aceleró la adopción de soluciones digitales.

Este crecimiento, sin embargo, trae consigo desafíos técnicos importantes, entre los cuales destaca la implementación de medios de pago electrónicos seguros, eficientes y confiables. La diversidad de proveedores de pago y los distintos estándares de seguridad que manejan estos, imponen barreras técnicas que dificultan la integración directa en sistemas empresariales.

Moqui ofrece cierta infraestructura para el manejo de pagos mediante entidades genéricas y servicios como autorización, captura y reembolsos. Sin embargo, carece de un módulo genérico que permita integrar de manera eficiente pasarelas de pago modernas que operan bajo el esquema de *hosted checkout*, como PayPal, Webpay Stripe o MercadoPago. Este tipo de integración permite cumplir con normativas de seguridad como PCI-DSS, ya que delega la captura de datos sensibles a la plataforma del proveedor de pago.

En este contexto, surge la necesidad de desarrollar un módulo específico que extienda las capacidades de Moqui para integrar medios de pago electrónicos utilizados tanto a nivel Latinoamericano como Internacional. El presente trabajo de memoria tiene como objetivo diseñar, desarrollar e implementar un módulo de integración de pagos electrónicos en Moqui, compatible con proveedores de pago modernos y que sirva como base flexible para integraciones futuras.

El módulo propuesto busca facilitar tanto el proceso de pago (incluyendo el redireccionamiento seguro y la autorización de transacciones) como la recepción de confirmaciones mediante notificaciones tipo *webhook*. Asimismo, se espera que este desarrollo sirva como plantilla replicable para futuras integraciones con otros proveedores, contribuyendo así a la expansión de Moqui como plataforma de comercio electrónico adaptable y segura.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar, desarrollar e implementar un módulo para la integración de medios de pago electrónicos en el framework Moqui, incorporando distintas pasarelas de pago, de modo que las integraciones futuras sean más sencillas y directas.

1.4.2. Objetivos Específicos

- **Estudiar el *framework* Moqui:** Comprender la estructura general del *framework*, el desarrollo de componentes, el uso de servicios, sintaxis para scripts, definiciones en XML y Groovy.
- **Analizar el módulo Mantle:** Examinar en detalle las entidades existentes relacionadas a pagos (*Payment*, *PaymentMethod*, *PaymentGatewayConfig*, etc.), los servicios disponibles (*authorize#Payment*, *capture#Payment*, etc.), y los mecanismos de extensión como enumeraciones, configuraciones y mensajes del sistema.
- **Revisar implementaciones existentes:** Estudiar módulos ya desarrollados por la comunidad Moqui que integren pasarelas de pago, para identificar patrones reutilizables y buenas prácticas.
- **Estudiar proveedores de pago:** Estudiar el proveedor de pago a integrar, considerando su disponibilidad en Latinoamérica, compatibilidad con *hosted checkout*, documentación de sus APIs y SDKs, y mecanismos de seguridad.
- **Diseñar la arquitectura del módulo:** Estructurar el nuevo módulo considerando carpetas, servicios, entidades, credenciales, puntos de integración, y mecanismos de configuración y extensibilidad.
- **Desarrollar servicios de integración con el proveedor de pago:** Implementar los servicios necesarios para crear sesiones de pago, redireccionar al usuario al *checkout* del proveedor, y manejar *callbacks* y notificaciones (*webhooks*).
- **Implementar un módulo independiente a Mantle:** Crear una versión del módulo que funcione sin depender del modelo de datos de Mantle, ideal para integraciones más simples o casos de uso específicos fuera del ecosistema estándar.
- **Integrar el módulo con Mantle (UDM y USL):** Adaptar el módulo para que sea compatible con la arquitectura estándar de Moqui, utilizando las entidades, servicios y flujos definidos en Mantle para que funcione como una extensión completamente integrada.
- **Documentar el módulo desarrollado:** Elaborar documentación técnica detallada y completa que describa el diseño, la instalación, los servicios implementados, el flujo de integración, y cómo extender el módulo para futuras pasarelas de pago.

Capítulo 2

Preparación

En este capítulo se abordarán las investigaciones, estudios y análisis previo al desarrollo del módulo. Estas actividades son relevantes para familiarizarse con las herramientas y tecnologías que se utilizaron. Además para la implementación de este trabajo fue necesario entender el *framework* Moqui y sus componentes (especialmente Mantle).

2.1. Estudio del *framework* Moqui

Moqui se caracteriza por ser un *framework* altamente modular y configurable para la construcción de aplicaciones empresariales. Técnicamente, Moqui está escrito en Java e incluye un motor basado en Groovy (lenguaje dinámico que corre en la JVM) para la definición de lógica de negocio de manera flexible. El *framework* provee una infraestructura integral de tres capas (datos, lógica de negocio y presentación) que facilita el desarrollo de sistemas tipo ERP de manera modular y con mínima programación manual (tiene muchas funcionalidades por defecto). Entre sus componentes principales se encuentran un motor de entidades (para gestión de base de datos), un motor de servicios (para la lógica de negocio orientada a servicios) y un motor de pantallas (para la interfaz de usuario basada en componentes reutilizables).

Un concepto clave en Moqui es el de componente. Un componente es un módulo de software en su mayor parte autocontenido que agrupa artefactos relacionados (entidades, servicios, pantallas, datos de ejemplo, etc.) siguiendo una estructura predefinida. La creación de funcionalidades en Moqui usualmente implica desarrollar un nuevo componente o añadir artefactos a componentes existentes. Cada componente respeta una convención de organización en directorios: por ejemplo, el directorio *entity* contiene definiciones de entidades de datos (en archivos XML, donde se pueden tanto crear entidades o extender entidades ya existentes), *service* contiene definiciones de servicios (también en XML, posiblemente con implementaciones embebidas o referenciadas), *screen* agrupa las pantallas de interfaz de usuario y *data* puede incluir datos de carga inicial o de ejemplo. Esta estructura estándar facilita la integración de componentes en el *runtime* de Moqui, ya sea colocándolos en el directorio global de componentes o declarándolos en la configuración (*moqui.conf*) correspondiente. Moqui

detecta y carga automáticamente estos artefactos al iniciar la aplicación, creando tablas a partir de las entidades definidas.

En el desarrollo con Moqui se hace un uso variado de definiciones en XML (similar a los tags de HTML) y de scripts Groovy para implementar la lógica (dentro de las definiciones de servicios se puede usar Groovy en formato *inline*, facilitando el desarrollo ya que no hay necesidad de recompilar el código). Las entidades se definen mediante archivos XML declarativos que describen los campos, tipos de datos y relaciones; a partir de estas definiciones, el *framework* genera el esquema de base de datos. Los servicios se declaran también en XML, siguiendo una nomenclatura verbo#Sustantivo (por ejemplo, *create#Invoice* o *authorize#Payment*), e incluyen la definición de sus parámetros de entrada/salida y el tipo de implementación. En resumen, este estudio del *framework* Moqui aporta dando las bases para entender como estructurar el desarrollo del proyecto: creando componentes modulares, definiendo entidades de datos y servicios en XML, y utilizando scripts Groovy cuando se requiere lógica personalizada más allá de lo que ofrecen las definiciones declarativas.

Ejemplo de definición de servicio en XML, obtenido del tutorial de Moqui *Framework* [7]:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <services xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation=
4     "http://moqui.org/xsd/service-definition-2.1.xsd">
5     <service verb="create" noun="Tutorial" type="entity-auto">
6         <in-parameters>
7             <auto-parameters include="all"/>
8         </in-parameters>
9         <out-parameters>
10            <auto-parameters include="pk"
11                required="true"/>
12        </out-parameters>
13        <actions>
14            ... Include logic here
15        </actions>
16    </service>
17</services>
```

2.2. Análisis de los módulos Mantle

Mantle Business Artifacts es la colección estándar de artefactos de negocio que complementa al Moqui *Framework*, proporcionando funcionalidades genéricas de nivel empresarial. Como se mencionó, Mantle se divide principalmente en UDM (*Universal Data Model*) y USL (*Universal Service Library*). En esta sección profundizamos en aquellos aspectos de Mantle relevantes para la integración de pagos electrónicos.

El *Universal Data Model* (UDM) de Mantle se basa en los conceptos encontrados en *The Data Model Resource Book* por Len Silverston. Estas entidades están diseñadas a partir de patrones comunes en sistemas ERP (buscando dar soporte a todas las ERP posibles, dando

una solución general). Las principales entidades Mantle relacionadas pagos son: *Payment*, *PaymentMethod*, *PaymentGatewayConfig* y *PaymentGatewayResponse*. La entidad *Payment* (pago) representa una transacción de pago realizada o pendiente, registrando información como el monto, la moneda, el estado del pago y referencias a otras entidades (por ejemplo, al pedido o factura que se está pagando, y al cliente que realiza el pago). La entidad *PaymentMethod* (método de pago) describe el medio o forma de pago utilizada; por ejemplo, puede representar una tarjeta de crédito, una cuenta de PayPal, un monedero electrónico u otras formas. *PaymentMethod* almacena datos relevantes según el tipo de método (como los últimos dígitos y fecha de expiración de una tarjeta, un *token* de pago seguro, etc.) y está vinculada con la parte (cliente) que es propietaria de ese método. Por su parte, la entidad *PaymentGatewayConfig* (configuración de pasarela de pago) contiene la configuración para conectar con un proveedor o *gateway* de pagos externo (por ejemplo, PayPal, Stripe, etc.). En esta entidad se especifica el tipo de pasarela (un identificador de tipo enumerado) y parámetros necesarios para la integración (como servicios, secretos, entre otros). Típicamente, una *PaymentGatewayConfig* se asocia a una tienda o contexto de tienda en línea mediante la entidad *ProductStorePaymentGateway*, indicando que esa tienda usará determinada pasarela para procesar cierto tipo de pagos. Finalmente, la entidad *PaymentGatewayResponse* (respuesta de la pasarela) registra los detalles de la respuesta obtenida al invocar un servicio de la pasarela externa. Cada vez que se realiza una operación de pago a través de un *gateway*, los códigos de autorización, códigos de respuesta, mensajes y cualquier dato relevante devuelto por el proveedor se almacenan en un *PaymentGatewayResponse*. Esta entidad generalmente se vincula al *Payment* correspondiente (mediante *paymentId*) y opcionalmente al *PaymentMethod* utilizado, sirviendo como registro histórico y de auditoría de la comunicación con la pasarela de pago.

La figura 2.1 obtenida de la documentación Moqui [7], muestra la extensibilidad de la entidad *PaymentMethod* y su relación al *PartyId* de el usuario.

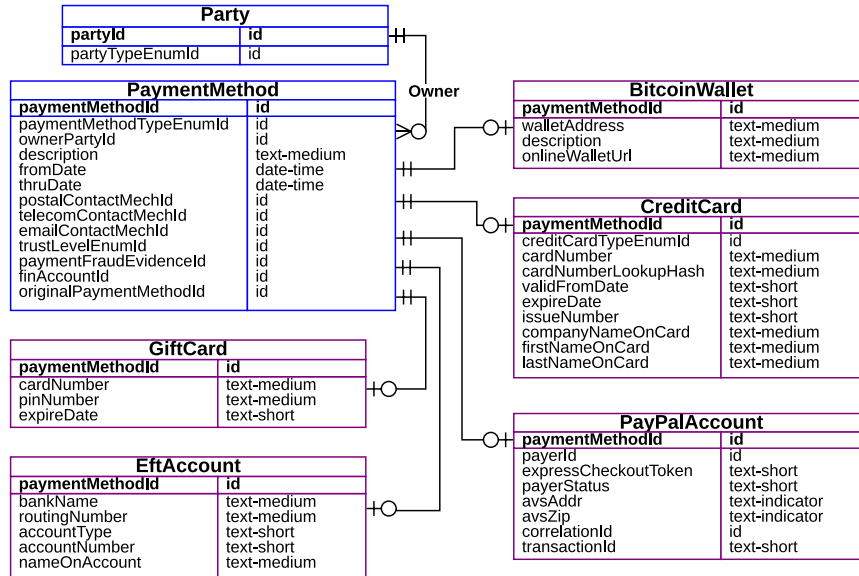


Figura 2.1: Diagrama de entidades UDM de Mantle relacionadas a método de pago

La figura 2.2 adquirida de la documentación Moqui [7] presenta de manera detallada la relación entre las entidades *Invoice*, *Payment*, *PaymentMethod* y *OrderPart*. También se puede ver cómo se aplican los pagos a las facturas mediante la entidad *PaymentApplication*.

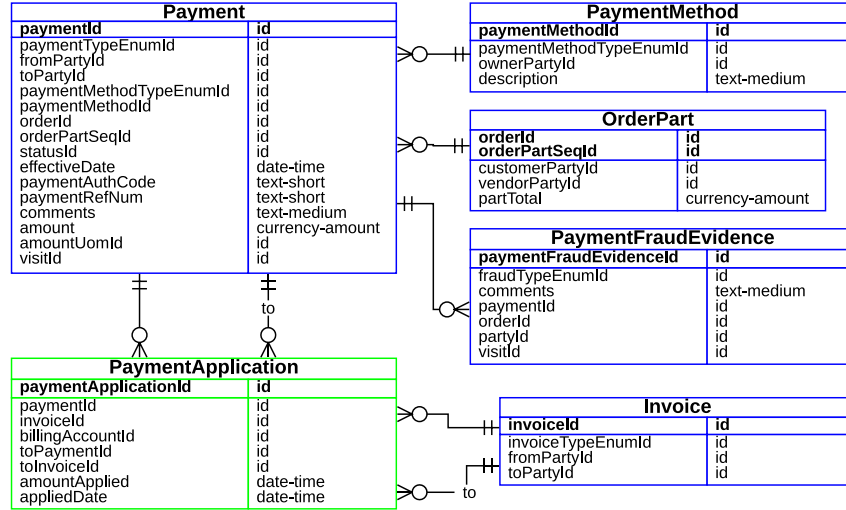


Figura 2.2: Diagrama de entidades UDM de Mantle relacionadas a pagos

La *Universal Service Library* (USL) por su parte define servicios que operan sobre el UDM para implementar procesos de negocio. Algunos ejemplos: creación de órdenes, facturación, contabilización, y también procesamiento de pagos. Revisando los servicios existentes en *mantle-usl*, se encontraron servicios como *authorize#Payment*, *capture#Payment* y *refund#Payment*, entre otros. El servicio *authorize#Payment* gestiona una solicitud de autorización de fondos: típicamente utilizado en pagos con tarjeta de crédito, envía los datos requeridos al *gateway* para confirmar que el medio de pago es válido y reservar el monto (sin capturarlo aún). Si la autorización es aprobada, el *framework* registra el pago con un estado indicativo (por ejemplo, autorizado) y almacena los códigos de autorización en un *PaymentGatewayResponse*. El servicio *capture#Payment* se emplea para realizar la captura (cobro) efectiva de un pago previamente autorizado, haciendo que el dinero reservado sea transferido. Esta separación entre autorización y captura sigue las mejores prácticas en procesamiento de pagos y permite, por ejemplo, ajustar el monto antes de capturar o manejar tiempos entre reserva y cobro. Todos estos servicios están diseñados siguiendo principios orientados a servicios (SOA) y actúan como interfaces abstractas: es decir, la implementación concreta internamente puede variar según el tipo de pasarela configurada. Por ejemplo, *authorize#Payment* evaluará que *PaymentGatewayConfig* o método de pago esté asociado al pago en curso y delegará la operación a la integración específica de ese proveedor (sea invocando una API REST, SDK o mecanismo propio de la pasarela). Así, Mantle ofrece un punto único de entrada para la aplicación (los servicios de pago unificados), mientras se conectan con el servicio externo correspondiente según la configuración.

2.3. Relación de Investigación con Objetivo general

Dado que el objetivo de este trabajo es incorporar una nueva integración de pasarela de pagos, resulta fundamental aprovechar los mecanismos de extensión que ofrece Mantle para este propósito. Uno de los principales mecanismos son las *enumeraciones* (*Enumeration*), utilizadas en Moqui para clasificar diversos conceptos. Por ejemplo, tipos de método de pago, tipos de operación de pago, tipos de *gateway*, estados de pago, entre otros. La extensibilidad en este caso radica en la posibilidad de definir nuevos valores enumerados dentro de componentes propios del proyecto, ampliando así las opciones existentes sin alterar el núcleo del sistema. En el contexto de pagos, esto implica la necesidad de agregar identificadores para la nueva pasarela, como un nuevo valor en *PaymentGatewayTypeEnum*, además de asegurarse de contar con los valores apropiados en *PaymentMethodTypeEnum*. Dichas enumeraciones permiten que el sistema reconozca y clasifique adecuadamente la información proveniente de una pasarela personalizada.

Otro mecanismo relevante es el uso de configuraciones personalizadas mediante la entidad *PaymentGatewayConfig* y sus entidades relacionadas, las cuales permiten almacenar los parámetros necesarios para la conexión con proveedores externos. En caso de ser necesario, también es posible extender entidades mediante la funcionalidad de *extensions* en las definiciones de entidad, incorporando campos adicionales requeridos para la integración.

Finalmente, destaca el uso del mecanismo de *System Message*, que Moqui proporciona para integraciones asíncronas entre sistemas. A través de la entidad *SystemMessage* y sus servicios asociados, es posible enviar, recibir y almacenar mensajes provenientes de sistemas externos de forma estandarizada, incluyendo funcionalidades como reintentos y registro de estado. En el contexto de pasarelas de pago, este mecanismo resulta especialmente útil para manejar notificaciones del proveedor, como las notificaciones IPN o *webhooks* que informan sobre el estado de un pago. Para ello, el *framework* permite definir configuraciones de sistemas remotos (*SystemMessageRemote*) con credenciales o *tokens* compartidos, además de exponer *endpoints* REST genéricos para recibir eventos de manera segura.

En conjunto, estos mecanismos proporcionan una base sólida y flexible sobre la cual construir integraciones con nuevas pasarelas de pago, asegurando que estas puedan ser implementadas de manera modular, segura y sin modificar el código central del *framework*. Esto es posible tanto para integraciones usando Mantle como para integraciones que no lo usen.

2.4. Revisión de implementaciones existentes

Para orientar el desarrollo de la integración de pago, se revisaron implementaciones existentes de pasarelas realizadas por la comunidad de Moqui. En particular, se analizó en profundidad el módulo de integración con Stripe (*coarchy/stripe*) disponible en GitHub. Este componente provee la funcionalidad para procesar pagos a través de esas plataformas y sirven como referencia para identificar patrones reutilizables/buenas prácticas de diseño.

El módulo de Stripe presenta una implementación consolidada y conveniente como refe-

rencia. Desarrollado por la empresa Coarchy, este componente integra Moqui con los servicios de Stripe aprovechando el producto *Stripe Checkout*. El componente crea nuevas enumeraciones para poder implementar este método de pago (*PaymentMethodType*, *PaymentInstrument* y también agrega el correspondiente *PaymentGatewayConfig* para configurar. Stripe fomenta el uso de su *checkout* alojado para mayor seguridad, y el módulo *stripe* adopta esta modalidad. En la revisión del repositorio, se observó que el componente define servicios propios dentro de *StripeServices.xml*, entre ellos uno llamado *create#CheckoutFromSalesOrder*. Este servicio encapsula la lógica de generar una sesión de *Checkout* de Stripe a partir de una orden de venta en Moqui: envía los detalles del pedido (montos, descripción, URL de éxito y cancelación, etc.) a la API de Stripe para crear una sesión de pago, y como resultado obtiene una *checkoutUrl*. En la carpeta de entity se puede notar que se extiende la entidad *OrderPart* para agregar los campos *checkoutId* y *checkoutUrl*. Tras guardar la información de *checkout*, la aplicación Moqui luego redirige al usuario a esa URL segura de Stripe donde el cliente ingresa sus datos de pago. Una vez completado el pago en Stripe, el usuario vuelve al *successUrl* del comerciante. La integración se encarga de confirmar el estado del pedido en Moqui al recibir notificaciones del *webhook* de Stripe. Un aspecto importante es como el módulo gestiona las credenciales y notificaciones: se apoya en *SystemMessage* para registrar un *SystemMessageRemote* con la configuración del *endpoint* de Stripe (incluyendo la clave secreta para validar *webhooks*). Así, cuando Stripe envía una notificación (por ejemplo, evento *checkout.session.completed*), esta ingresa a Moqui a través de un servicio genérico REST de *SystemMessage*, el cual valida la firma del mensaje y lo guarda en la cola de mensajes del sistema. Luego, el componente *stripe* contiene lógica para procesar esos mensajes: actualizar el *Payment* y el *Order* correspondientes, marcar el pedido como pagado y eventualmente generar la transacción financiera interna. Este diseño demuestra una integración custom, desacoplada y segura con la pasarela de pago, adhiriendo a las recomendaciones tanto de Moqui como del proveedor externo. Es importante notar que esta solución si bien se integra hasta cierto punto con Mantle, terminó siendo una solución personalizada para su compañía ya que solo son pagos de *OrderParts* y no un pago general.

2.4.1. Patrones reutilizables

- **Uso de la abstracción de Mantle:** Este módulo utiliza parte de las entidades y servicios nativos de Mantle en lugar de crear esquemas paralelos. Esto significa registrar un *PaymentGatewayConfig* específico para la pasarela (con su tipo enumerado correspondiente) y conectar la lógica adecuándose al servicio del proveedor. Esta práctica permite que los pagos queden registrados de forma consistente usando de manera correcta el ecosistema de Moqui, permitiendo que el resto de componentes (facturación, envíos, etc.) funcionen sin cambios. Sin embargo es importante crear una integración a Mantle más general para que se pueda usar en otros proyectos.
- ***Checkout* hospedado para seguridad:** Se observa que las integraciones más recientes favorecen el modelo de *hosted checkout*, donde el cliente es redirigido a la página del proveedor para ingresar datos sensibles (tarjeta, contraseña, etc.). Tanto PayPal como Stripe ofrecen esta opción (PayPal mediante su flujo *Express Checkout*, Stripe mediante *Stripe Checkout*). Esto minimiza el manejo de información confidencial por parte de la aplicación Moqui y mejora la confianza del usuario en la transacción.

- **Manejo de notificaciones asíncronas:** Una buena práctica identificada es utilizar el mecanismo de *SystemMessage* para procesar notificaciones de la pasarela (*webhooks*). En lugar de confiar solo en la respuesta inmediata a la petición de pago (que podría fallar o no contener toda la información). Los componentes están preparados para recibir confirmaciones en segundo plano y obtener el estado de los pagos. Esto aumenta la robustez ante eventuales retrasos o problemas de conectividad, ya que las notificaciones pueden reintentarse y quedan registradas en la cola de mensajes de Moqui hasta ser procesadas exitosamente.
- **Documentación y datos de prueba:** Se noto que los módulos investigados proporcionan archivos README con instrucciones. Incluir documentación clara y detallada es una buena práctica para validar la integración y permitir que otros la desplieguen correctamente. En este trabajo se planea hacer lo mismo, ofreciendo instrucciones para configuración de las credenciales de MercadoPago y pruebas de pago en ambiente de sandbox.

2.5. Estudio de proveedores de pago

Existen múltiples proveedores y pasarelas de pago electrónico en el mercado, cada uno con diferentes características, costos y niveles de complejidad de integración. Entre las opciones considerados inicialmente estuvieron: PayPal, Transbank (WebPay, popular en Chile), y Mercado Pago. Si bien lo ideal es implementar múltiples proveedores, está la posibilidad que como parte de este estudio el esfuerzo se concentre en implementar un solo proveedor. Por lo tanto es fundamental elegir al adecuado para implementar primero. De las 3 opciones se eligió Mercado Pago como proveedor inicial para este componente por las siguientes razones:

En primer lugar, cobertura geográfica: Mercado Pago tiene una fuerte presencia en Latinoamérica, con soporte en países como Argentina, Brasil, Chile, Colombia, México, Perú, entre otros. Dado que Moit desarrolla soluciones principalmente para Chile y la región, esta opción resulta conveniente. Integrar Mercado Pago agrega valor a los clientes de Moit ya que pueden ofrecer métodos de pago familiares para sus consumidores (por ejemplo, pago en cuotas con tarjetas locales, etc.), condición que otras pasarelas internacionales no siempre soportan.

En segundo lugar, funcionalidad e información de integración: La plataforma Mercado Pago provee un conjunto de APIs REST documentadas, así como SDKs en varios lenguajes (incluyendo Java, .NET, Node.js, Python, etc.), lo que permite que el desarrollo sea directo. El SDK de Java es relevante para la integración con Moqui (entorno Java/Groovy), además ofrece diferentes modalidades de integración para pagos.

2.5.1. Modalidades de integración

- **Checkout Pro:** es una integración tipo *hosted checkout*, donde el usuario es enviado a la página de pago segura de Mercado Pago para completar los datos de su tarjeta o método de pago. Luego es llevado de vuelta al sitio del comercio. Esta modalidad es

la más sencilla de implementar, pues Mercado Pago maneja la interfaz de pago y la seguridad de los datos sensibles.

- **Checkout API (custom checkout):** en esta modalidad, la aplicación del comercio controla completamente la experiencia de pago, recopilando los datos de tarjeta en su propia interfaz (usando herramientas como la biblioteca JavaScript de Mercado Pago para tokenizar la tarjeta) y luego enviando la transacción vía API. Requiere mayor trabajo y cumplir con estándares de seguridad (PCI compliance) ya que el manejo de datos sensibles recae en el comercio, pero ofrece una experiencia más integrada al sitio.

Para el alcance de este trabajo, se decidió implementar la integración usando Checkout Pro, aprovechando la rapidez de puesta en marcha y la existencia de un flujo ya diseñado por Mercado Pago. Esto permite centrarse en la comunicación entre Moqui y Mercado Pago (creación de la transacción y recepción de notificación) sin tener que desarrollar una interfaz de pago completa desde cero.

2.5.2. Estudio de su API y SDK

Credenciales y autenticación: Para utilizar la API de Mercado Pago es necesario crear una cuenta de desarrollador y registrar una aplicación. Esto proporciona dos claves principales: el *Access Token* (token secreto del lado servidor) y la *Public Key* (clave pública para integraciones del lado cliente). Mercado Pago tiene un modo *sandbox* dedicado para el desarrollo, que permite procesar pagos de prueba sin mover dinero real, utilizando credenciales de prueba diferenciadas. En el anexo se puede ver la figura A.2 con las credenciales de prueba.

Creación de una preferencia de pago: El paso central del flujo de pago con Checkout Pro es la creación de una *Preferencia de Pago* (Payment Preference). Una preferencia es un objeto que se envía a Mercado Pago con los detalles de la transacción. Esta incluye, descripción del producto o servicio, monto a cobrar, moneda, correo del pagador (opcional), identificador externo (número de orden en el sistema local por ejemplo), URLs de retorno (éxito, fallo, pago pendiente) y URL de notificación (webhook). Al crear exitosamente una preferencia, la API devuelve un ID único de preferencia junto con una URL a la cual redirigir al usuario para que realice el pago. En el SDK de Java, esto se simplifica con clases y métodos dedicados. Para crear preferencias en este caso se usa la clase *Preference* donde se setean los atributos mencionados.

Notificaciones (Webhooks): Para recibir resultado de un pago, existen dos mecanismos complementarios: la redirección final del usuario (por ejemplo, Mercado Pago redirige a la URL de éxito o error tras completar el pago) y las notificaciones silenciosas al backend (webhooks). Las notificaciones son esenciales para confiabilidad, pues aseguran que el sistema sepa el estado del pago aunque el usuario cierre la página antes de volver, por ejemplo. Mercado Pago permite configurar en la preferencia una *notification_url* (A.3) cuando el estado de la transacción cambia (pago aprobado, pendiente, rechazado, etc.), se envía una solicitud HTTP POST a esa URL con información del evento. El servidor debe exponer un servicio para recibir estas notificaciones, verificar su autenticidad (es importante validar que provienen

de Mercado Pago mediante firma) y luego actualizar el registro correspondiente del pago en el sistema local.

Manejo de pagos: Los pagos en Mercado Pago pueden tener varios estados (*approved*, *pending*, *in_process*, *rejected*, etc.). Se prestó atención a cómo manejar cada caso. Por ejemplo, *pending* significa que el usuario quizá no completó el pago o está en proceso, *approved* es el éxito definitivo, *rejected* indica fallo (podría reintentarse). El webhook puede enviarse múltiples veces o Mercado Pago puede reintentar notificar si el servidor no respondió correctamente. Por tanto, la implementación debía ser idempotente (propiedad de una operación que, al ejecutarse repetidamente con los mismos parámetros, produce siempre el mismo resultado que si se ejecutara una sola vez) y robusta frente a notificaciones duplicadas.

2.5.3. Otros: Webpay Plus y PayPal

Además de Mercado Pago, durante el análisis se revisaron otras pasarelas de pago como Webpay Plus y PayPal para evaluar su funcionamiento e integración. A continuación se presenta un resumen breve de cómo operan estos servicios de *checkout*. Ambos tienen funcionamiento similar a Mercado Pago Checkout Pro, pero con algunas diferencias:

Webpay Plus (Transbank): Es la pasarela de pago más utilizada en Chile y opera bajo un modelo de *hosted checkout*. El comercio inicia la transacción enviando los datos del pago a la API (los detalles de la compra monto, orden, etc.), recibiendo un *token* y una URL de redirección al formulario seguro de Webpay. Tras autorizar el pago, el usuario vuelve al sitio del comercio, donde se consulta el resultado definitivo mediante una nueva llamada a la API. Además, Webpay puede enviar notificaciones *webhook* para confirmar el estado del pago en segundo plano. La integración requiere certificados digitales, un código de comercio y puede probarse en un entorno *sandbox*.

PayPal (servicio de Checkout): PayPal es una pasarela de pago internacional que permite transacciones con cuentas PayPal o tarjetas de crédito/débito mediante un esquema de *hosted checkout*. El comercio crea una orden vía API usando sus credenciales (*Client ID* y *Secret*), y redirige al usuario a la página de PayPal para aprobar el pago. Una vez completado, el usuario vuelve al sitio del comercio, y este debe confirmar la transacción ejecutando una llamada final a la API. Además, PayPal ofrece mecanismos asíncronos como *webhooks* o *IPN* para asegurar el registro del estado del pago incluso si el redireccionamiento falla. También cuenta con un entorno *sandbox* para pruebas.

2.5.4. Comparación general de modelos de *hosted checkout*

Webpay Plus, PayPal y Checkout Pro de Mercado Pago comparten un modelo similar de integración denominado *hosted checkout*, donde el usuario es redirigido a un entorno seguro del proveedor para completar el pago, antes de regresar al sitio del comercio. La tabla 2.1 muestra un resumen de las comparaciones.

Tabla 2.1: Comparación entre Checkout Pro (Mercado Pago), Webpay Plus (Transbank) y PayPal Checkout

Aspecto	MercadoPago (Checkout Pro)	WebpayPlus (Transbank)	PayPal (Checkout)
Cobertura	Latinoamérica	Local (Chile)	Global (menos común en Chile)
Medios de pago	Tarjeta, débito, crédito, saldo MP, efectivo	Tarjeta de crédito y débito chilena	Tarjeta, cuenta PayPal y otros métodos según región
Notificaciones asíncronas	IPN y Webhooks	IPN y Webhooks	IPN y Webhooks
Experiencia usuario	Fluida, interfaz personalizable y local	Formal y fija, página segura de Transbank	Conocida a nivel global, puede requerir autenticación PayPal (popup integrado)
Facilidad de integración	Alta, con SDKs y REST bien documentados	Media, requiere certificados, aprobación de comercio y pruebas	Alta, con SDKs y documentación robusta

Capítulo 3

Diseño, Desarrollo e Implementación

Durante la etapa de desarrollo se optó por enfocar los esfuerzos en la integración de un único proveedor de pagos, MercadoPago Checkout Pro. Esta decisión se fundamentó en la investigación previa, donde se evaluaron diversas alternativas (como Webpay Plus y PayPal), concluyendo que MercadoPago ofrecía una mejor combinación de facilidad de integración, documentación disponible, funcionalidad y potencial de extensión dentro del ecosistema Moqui. En este capítulo se detallan las dos implementaciones desarrolladas para integrar MercadoPago en Moqui: (1) un módulo independiente a Mantle de tipo *plug & play*, y (2) una solución integrada con Mantle. Se describen las arquitecturas de cada enfoque, las entidades y servicios implementados, configuraciones realizadas, fragmentos de código y diagramas para facilitar la comprensión. Notar que el desarrollo se realizó en un repositorio en Gitlab, donde se utilizaron ramas para cada *feature* específica. Al terminar el desarrollo se dejó la solución independiente en la rama *dev* y la integrada con Mantle en *feature/checkout-payments-mantle-integration*.

3.1. Módulo Independiente

Este componente de Moqui, provee una integración plug-and-play con MercadoPago Checkout Pro para procesar pagos en línea. Está diseñado específicamente para facilitar la integración de pagos con MercadoPago, con soporte de Webhook seguro (HMAC SHA-256, se extiende soporte para la firma particular que utiliza MercadoPago) y mínima configuración. Este componente no utiliza las entidades ni servicios de Mantle (permitió enfocarse en el desarrollo de la integración con MercadoPago, sin tener que preocuparse por la lógica de Mantle). Incluye las bibliotecas necesarias de MercadoPago (SDK Java v2.2.0). Para su correcto funcionamiento se deben definir las siguientes variables de entorno (o propiedades) en la configuración de Moqui B.1:

- *mercado_pago_access_token*: Token de acceso a la API de MercadoPago.
- *mercado_pago_currency*: Código de moneda (por ejemplo, USD).
- *mercado_pago_success_url*: URL de retorno tras pago exitoso.
- *mercado_pago_failure_url*: URL de retorno tras pago fallido o cancelado.

3.1.1. Modelo de Datos (Entidades)

El archivo *MoitMPPaymentEntities.xml* define las entidades usadas para registrar información de MercadoPago. Las principales son:

- **MercadoPagoCustomer:** vincula un *partyId* interno con el ID de cliente de MercadoPago (*mpCustomerId*). Se usa para mapear usuarios entre ambos sistemas. Este *partyId* es general, es decisión del desarrollador con que entidad/campo se va a relacionar. Esta entidad no es necesaria para Checkout Pro, pero se implementó para futuras extensiones.
- **MercadoPagoCheckout:** representa una sesión de pago iniciada (checkout). Llave primaria *internalCheckoutId*, un identificador interno para crear relaciones necesarias. Almacena el *partyId* de la tabla MercadoPagoCustomer, una referencia externa *externalReference* (por ejemplo número de orden), el monto total *amount*, moneda *currency*, y un campo *status* (p. ej. "init", "approved"). El campo *mpPreferenceId* es el identificador de la preferencia de pago generada por MercadoPago.
- **MercadoPagoPayment:** registra cada transacción de pago. Llave primaria *mpPaymentId*. Incluye el *internalCheckoutId* asociado, el *paymentStatus* (estado del pago), el monto de la transacción y moneda.
- **MercadoPagoPlan:** define un plan de pago recurrente (suscripción). Llave primaria *mpPlanId*. Contiene razón (*reason*), frecuencia de facturación (*frequency*, *frequencyType*), monto por periodo (*transactionAmount*, *currency*), repeticiones, día de facturación y posible periodo de prueba gratuito. Es importante notar que este plan no es necesario para Checkout Pro, pero se implementó para futuras extensiones.
- **MercadoPagoSubscription:** representa una suscripción activa. Llave primaria *subscriptionId*. Incluye el *partyId* (cliente interno), el *mpSubscriptionId*, refiere a un *mpPlanId*, establece monto y frecuencias recurrentes, fechas de inicio/fin, estado (*authorized*, *paused*, *cancelled*, etc.) y si está activa. Es importante notar que este plan no es necesario para Checkout Pro, pero se implementó para futuras extensiones.

Por ejemplo, la entidad *MercadoPagoCheckout* se define como sigue (el resto estará en el Anexo B.2):

```
1 <entity entity-name="MercadoPagoCheckout"
2   package-name="moit.payments" has-stamp="true"
3   enable-audit-log="true">
4   <field name="internalCheckoutId" type="id" is-pk="true"/>
5   <field name="mpPreferenceId" type="text-long"
6     description="Preference ID from Mercado Pago"/>
7   <field name="partyId" type="id" required="true"/>
8   <field name="externalReference" type="text-short"
9     description="Reference to order, invoice, etc."/>
10  <field name="amount" type="currency-amount"/>
11  <field name="currency" type="text-short"/>
12  <field name="status" type="text-short"/>
```

```

13         description="pending , expired , completed"/>
14 </entity>

```

Las relaciones entre entidades son: por ejemplo, un *MercadoPagoCustomer* puede estar asociado a varias *MercadoPagoCheckout* y *MercadoPagoSubscription*, y un *MercadoPagoCheckout* puede tener varios *MercadoPagoPayment* (esto ocurre cuando durante el checkout se rechaza el pago, y se vuelve a intentar, resultando en 2 pagos para un mismo checkout).

En la figura 3.1 se puede ver el diagrama entidad relación de la solución independiente a Mantle.

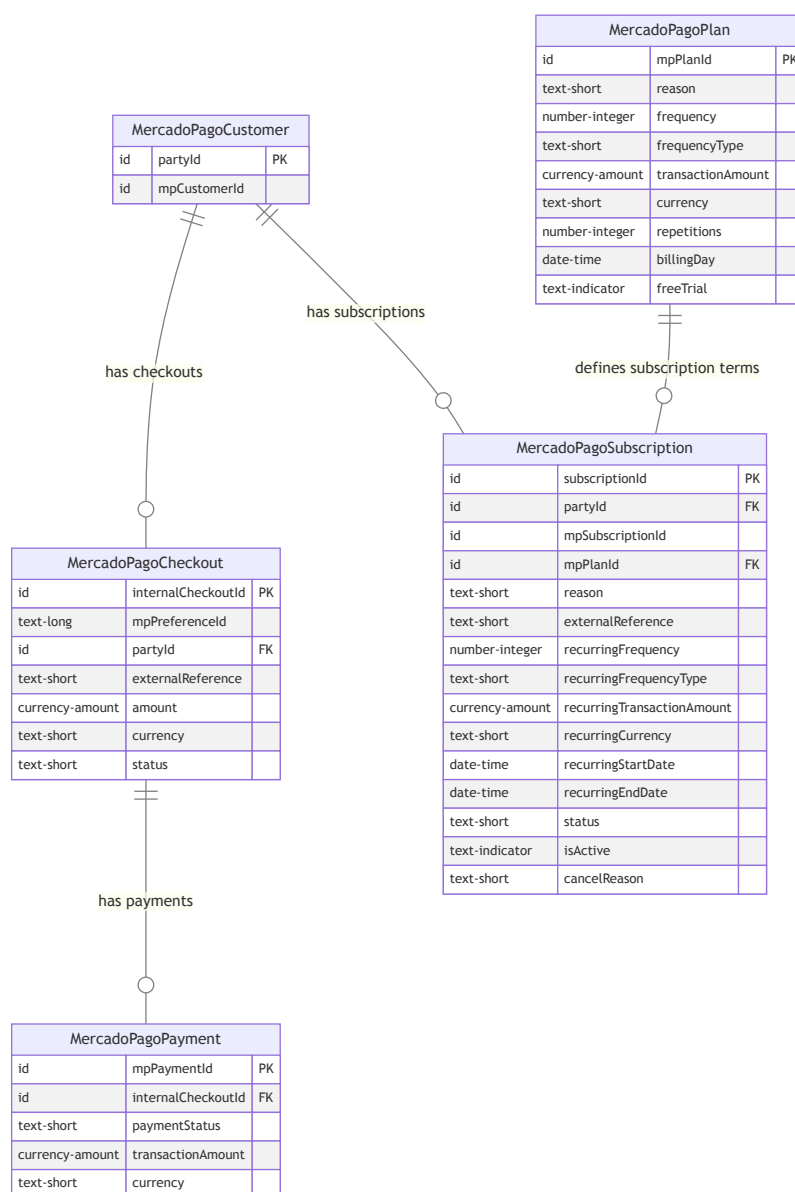


Figura 3.1: Diagrama ER del modelo de datos interno para MercadoPago (entidades clave y relaciones).

Elecciones de diseño

Al diseñar el modelo de datos, se consideró utilizar *mpCustomerId* y *mpPreferenceId* como llaves primarias. Sin embargo, esta opción se descartó porque no es posible asociar una preferencia (*preference*) a un cliente (*customer*) de forma única: sólo se pueden incluir datos como nombre, correo electrónico y otros atributos de perfil, pero no existe un vínculo foráneo obligatorio entre preferencias y clientes. Además, al momento de recibir las confirmaciones de pago, no resultaba posible recuperar el valor de *mpPreferenceId*, lo que imposibilitaba su uso como llave primaria.

Para resolver esta limitación, se introdujo un nuevo campo *internalCheckoutId* que actúa como llave primaria de las preferencias de Checkout Pro. Este identificador interno se almacena en la metadata de la preferencia, lo que permite relacionar de manera confiable cada Checkout con su correspondiente registro de pago en el sistema.

Asimismo, aunque las entidades *MercadoPagoSubscription* y *MercadoPagoPlan* no son estrictamente necesarias para el flujo de Checkout Pro, se implementaron en base de los campos expuestos por la API de MercadoPago. Así, queda preparado el componente para futuras extensiones.

Datos Set-up de componente

En el archivo *data/MoitPaymentSetup.xml* se inicializan los datos semilla necesarios para el manejo de webhooks de Mercado Pago. Concretamente, se definen:

- Una nueva entrada en la enumeración *SystemMessageAuthType* para HMAC SHA-256 (*SmatMercadoPago*). (Más adelante se explica cómo se configura este nuevo auth type en Moqui)
- El tipo de mensaje *SystemMessageType MercadoPagoWebhook*, que asocia las notificaciones entrantes al servicio *consume#MercadoPagoWebhookEvent*.
- La fuente remota *SystemMessageRemote MercadoPago*, configurada con el método de autenticación HMAC (*SmatMercadoPago*), el secreto compartido y el nombre del encabezado (*x-signature*).

A continuación se muestra el fragmento XML de configuración: (archivo entero en el Anexo B.3)

```
1 <!-- ===== System Message Types (for Webhook Handling)
   ===== -->
2 <moqui.basic Enumeration enumId="SmatMercadoPago"
   description="Mercado Pago HMAC SHA-256"
   enumTypeId="SystemMessageAuthType"/>
3
4
5 <!-- ===== System Message Types (for Webhook Handling)
   ===== -->
```

```

6 <moqui.service.message.SystemMessageType
    systemMessageTypeId="MercadoPagoWebhook"
7     consumeServiceName=
8     "moit.payments.MercadoPagoServices."
9     consume#MercadoPagoWebhookEvent"
10    description="MercadoPago Webhook Events"
11    contentType="application/json"/>
12
13 <moqui.service.message.SystemMessageRemote
    systemMessageRemoteId="MercadoPago"
14     description="MercadoPago Payment System"
15     messageAuthEnumId="SmatMercadoPago"
16     sharedSecret="SharedSecret"
17     authHeaderName="x-signature"/>

```

3.1.2. Servicios Principales

La implementación define varios servicios Moqui en *MercadoPagoServices.xml* para manejar la lógica de pagos y webhooks. Todos los servicios fueron implementados usando xml y groovy inline. Se intentó que el código groovy relacionado a MercadoPago, quedara compilado (así solo se importaba el código) pero esto obstaculizó demasiado el desarrollo, ya que para cada cambio se tenía que recompilar todo el código. Además implementarlo inline facilita la lectura del código, y permite ver los cambios en tiempo real (mejorando la experiencia de desarrollo). Todos los servicios que usan el sdk de MercadoPago, utilizan el *accessToken* de configuración.

getOrCreate#MercadoPagoCustomer

Este servicio se encarga de obtener/crear un cliente en Mercado Pago asociado a un *partyId*: primero obtiene el *accessToken* de configuración y valida que *partyId*, *payerEmail*, *payerName* y *payerSurname* estén presentes; luego busca localmente en la entidad *moit.payments.MercadoPagoCustomer* un registro con ese *partyId* usando moqui xml. Si existe, retorna su *mpCustomerId* con una variable de retorno *isNew=false*. En caso contrario, configura el SDK de Mercado Pago y realiza una búsqueda remota por correo electrónico donde si encuentra un cliente, lo reutiliza, y si no, invoca la API para crearlo, marcando *isNew=true*. En caso de que ocurra un error, este se registra y se devuelve. Finalmente, se devuelve (solo *isNew* y *mpCustomerId*).

Este es el código groovy inline del servicio, cuando se crea un customer (el código completo se encuentra en el Anexo B.4.1):

```

1 <service verb="getOrCreate" noun="MercadoPagoCustomer">
2     <description>Get or create a Mercado Pago customer, linked to a
        partyId</description>
3     <in-parameters>
4         <parameter name="partyId" required="true"/>

```

```

5      <parameter name="payerEmail" required="true"/>
6      <parameter name="payerName" required="true"/>
7      <parameter name="payerSurname" required="true"/>
8  </in-parameters>
9  <out-parameters>
10     <parameter name="mpCustomerId"/>
11     <parameter name="isNew"/>
12 </out-parameters>
13 <actions>
14 ...
15     def custClient = new CustomerClient()
16     def resultCustomer = null
17     def isNew = false
18
19     // 1. Try to find existing customer by email (PLEASE NOTE:
20     //     EMAIL IS UNIQUE, SO NO USERS CAN HAVE THE SAME EMAIL OTHER
21     //     WISE THIS WILL FAIL)
22     try {
23         def searchReq = MPSearchRequest.builder()
24             .filters([email: payerEmail])
25             .limit(10)
26             .offset(0)
27             .build()
28         def searchResp = custClient.search(searchReq)
29         resultCustomer = searchResp.results?.find { it.email ==
30             payerEmail }
31         if (resultCustomer) {
32             logger.info("Found existing MercadoPago customer:
33                 ${resultCustomer.id}")
34             isNew = false
35         }
36     } catch (MPApiException e) {
37         System.out.println("Search failed, continuing to create:
38             ${e.message}")
39         logger.warn("Search failed, continuing to create:
40             ${e.message}", e)
41     }
42
43     // 2. Create only if not found
44     if (!resultCustomer) {
45         try {
46             resultCustomer = custClient.create(
47                 CustomerRequest.builder()
48                     .email(payerEmail)
49                     .firstName(payerName)
50                     .lastName(payerSurname)
51                     .build()
52             )
53             isNew = true
54             logger.info("Created new MercadoPago customer:

```

```

        ${resultCustomer.id}")
49     } catch (MPApiException e) {
50         System.out.println("Error creating MercadoPago
            customer: ${e.message}")
51         def resp = e.apiResponse
52         def status = resp?.status
53         def content = resp?.getContent()
54         def json = new JsonSlurper().parseText(content ?:
            '{}')
55         def desc = json.cause?.getAt(0)?.description
56
57         logger.error("MP API error (status: ${status}):
            ${content}", e)
58         ec.context.errorMessage = "MP Error ${status}: ${desc
            ?: content}"
59         return
60     }
61 }
62 ec.context.mpCustomerId = resultCustomer?.id
63 ec.context.isNew = isNew
64
65 ...
66 </service>

```

create#MercadoPagoCheckout

Este servicio crea una sesión de pago en Mercado Pago y registra su información básica en entidad dedicada. Primero recupera desde la configuración el *accessToken*, la moneda (*mercado_pago_currency*) y las URLs de retorno. Se valida que todas estas variables base estén definidas. Después se revisa que las input variables obligatorias *partyId*, *externalReference* y *payingItems* no estén vacías. Luego se itera sobre *payingItems* (cada uno con *id*, *title*, *unitPrice* y *quantity*) para calcular el monto total y construir la lista de *PreferenceItemRequest*. Luego genera un nuevo registro en *moit.payments.MercadoPagoCheckout* con *internalCheckoutId* (UUID), estado “init”, monto y datos básicos. Utilizando el SDK de Mercado Pago se arma un *PreferenceRequest* con *externalReference*, los ítems, las *backUrls*, *autoReturn*=“*approved*” y metadatos que incluyen el *internalCheckoutId*. Opcionalmente agrega datos del pagador si se proporcionan *payerEmail*, *payerName* y *payerSurname*. Tras llamar a *PreferenceClient.create(...)* obtiene la URL de inicio de pago (*initPoint*) y el *mpPreferenceId*. Con estos datos actualiza el registro local cambiando el estado y guardando el *mpPreferenceId*. Finalmente retorna *success=true*, *checkoutUrl* (la URL a la que debe redirigirse al cliente) y *checkoutId* (el *internalCheckoutId* generado).

Este servicio en particular, fue diseñado para ser invocado desde otro servicio donde el desarrollador puede definir los ítems a pagar, y el servicio se encarga de crear el checkout y retornar la url de pago (de manera más general). (el código completo se encuentra en el Anexo B.4.2)

```

1 <!-- Create Checkout -->
2 <service verb="create" noun="MercadoPagoCheckout">
3     <description>Create Mercado Pago checkout and store
4         it</description>
5     <in-parameters>
6         <parameter name="payingItems" type="List" required="true"/>
7         <parameter name="payerEmail"/>
8         <parameter name="payerName"/>
9         <parameter name="payerSurname"/>
10        <parameter name="externalReference" required="true"/>
11        <parameter name="partyId" required="true"/>
12    </in-parameters>
13    <out-parameters>
14        <parameter name="success"/>
15        <parameter name="checkoutUrl"/>
16    </out-parameters>
17
18    <script language="groovy"><![CDATA[
19        ...
20        // EXAMPLE OF PAYING ITEMS
21        /*def itemsList = [
22            [id: "item1", title: "Test Item 1", unitPrice:
23                "15.00", quantity: 1],
24            [id: "item2", title: "Test Item 2", unitPrice:
25                "20.00", quantity: 2]
26        ]
27        */
28        def itemsList = payingItems
29        if (!itemsList || !(itemsList instanceof List) ||
30            itemsList.isEmpty()) {
31            logger.error("Missing or invalid 'payingItems'")
32            ec.message.addError("Parameter 'payingItems' must be a
33                non-empty List")
34            ec.context.success = false
35            return
36        }
37
38        MercadoPagoConfig.setAccessToken(accessToken)
39
40        def totalAmount = BigDecimal.ZERO
41        logger.info("Creating checkout for items: ${itemsList}")
42        def itemsReq = itemsList.collect {
43            def unitPrice = (it.unitPrice as BigDecimal)
44            def quantity = (it.quantity as Integer)
45            totalAmount += unitPrice * quantity
46            PreferenceItemRequest.builder()
47                .id(it.id as String)
48                .title(it.title as String)
49                .quantity(quantity)

```

```

46         .currencyId(mercadoPagoCurrency)
47         .unitPrice(unitPrice)
48         .build()
49     }
50
51     def backUrls = PreferenceBackUrlsRequest.builder()
52     .success(mercadoPagoSuccessUrl)
53     .pending(mercadoPagoPendingUrl)
54     .failure(mercadoPagoFailureUrl)
55     .build()
56
57
58     def v =
59         ec.entity.makeValue("moit.payments.MercadoPagoCheckout")
60     v.partyId = partyId
61     v.externalReference = externalReference
62     v.amount = totalAmount
63     v.currency = mercadoPagoCurrency
64     v.status = "init"
65     v.internalCheckoutId = UUID.randomUUID().toString()
66     v.create()
67     def internalCheckoutId = v.internalCheckoutId.toString()
68
69     // Here we can also add
70     notificationUrl("https://your-webhook-url.com")
71     // For now, we will use global webhook in Developer
72     Dashboard
73
74     def prefBuilder = PreferenceRequest.builder()
75     .externalReference(externalReference)
76     .items(itemsReq)
77     .backUrls(backUrls)
78     .autoReturn("approved")
79     .metadata([
80         "checkout_id": internalCheckoutId
81     ])
82
83     def email = payerEmail?.trim()
84     def name = payerName?.trim()
85     def surname = payerSurname?.trim()
86
87     if (email && name && surname) {
88         // Build a payer object with email/name/surname
89         prefBuilder.payer(
90             PreferencePayerRequest.builder()
91             .email(email)
92             .name(name)
93             .surname(surname)
94             .build()

```

```

93         )
94         logger.info("Using payer info: ${name} ${surname},
95                     ${email}")
96     }
97     if (!(email && name && surname)) {
98         logger.warn("Incomplete payer info; proceeding without
99                     payer details")
100     }
101     def pref = new
102         PreferenceClient().create(prefBuilder.build())
103     def url = pref.getInitPoint()
104
105     v.mpPreferenceId = pref.getId()
106     v.status = "init"
107     v.update()
108     ec.context.checkoutUrl = url
109     ec.context.checkoutId = internalCheckoutId
110     ec.context.success = true
111     ...

```

consume#MercadoPagoWebhookEvent

Este servicio consume las notificaciones webhook de Mercado Pago recibidas a través de un *SystemMessage* mediante el parámetro *systemMessageId*. Primero recupera el registro *SystemMessage* correspondiente para extraer el JSON crudo en *messageText*. A continuación, parsea el payload con *JsonSlurper* y verifica que el tipo del evento sea *type="payment"*. De ser así usando *PaymentClient.get(paymentId)*, obtiene los detalles de la transacción (ID, metadata, estado, monto y moneda). Con estos datos construye el mapa de parámetros *{mpPaymentId, internalCheckoutId, paymentStatus, transactionAmount, currency}* y llama de forma síncrona al servicio *process#MercadoPagoPaymentEvent* para actualizar en la base de datos el registro *MercadoPagoPayment* y el estado de la entidad *MercadoPagoCheckout*. En caso de errores en la API o en el parsing, registra la excepción y retorna sin interrumpir el flujo global de mensajes. (Servicio completo en el Anexo B.4.4)

```

1 <!-- ===== Webhook Receiver ===== -->
2 <service verb="consume" noun="MercadoPagoWebhookEvent"
3     authenticate="anonymous-all">
4     <description>Handle MercadoPago webhook
5         notifications</description>
6     <in-parameters>
7         <parameter name="systemMessageId" required="true"/>
8     </in-parameters>
9     <actions>
10
11         <!-- Get the SystemMessage record to access the payload -->

```

```

10      <entity-find-one
        entity-name="moqui.service.message.SystemMessage"
        value-field="systemMessage">
11          <field-map field-name="systemMessageId"
            from="systemMessageId"/>
12      </entity-find-one>
13
14      ...
15
16      <!-- Parse JSON if needed -->
17      <script language="groovy"><![CDATA[
18          ...
19          def jsonSlurper = new JsonSlurper()
20          def payloadData = jsonSlurper.parseText(webhookPayload)
21
22          // Access specific fields from the webhook
23          def eventType = payloadData.type
24          def event = payloadData.action
25
26          logger.info("Webhook event type: ${eventType}")
27
28          MercadoPagoConfig.setAccessToken(accessToken)
29
30          if (eventType.equals("payment")) {
31              def params = null
32              try {
33                  def paymentId = payloadData.data?.id as Long
34                  def payment = new PaymentClient().get(paymentId)
35                  logger.info("Payment metadata:
36                      ${payment.getMetadata()}")
37                  params = [
38                      mpPaymentId: paymentId,
39                      internalCheckoutId:
40                          payment.getMetadata().get("checkout_id"),
41                      paymentStatus: payment.getStatus(),
42                      transactionAmount: payment.getTransactionAmount(),
43                      currency: payment.getCurrencyId()
44                  ]
45              } catch (MPApiException e) {
46                  def apiResp = e.getApiResponse()
47                  logger.error("MercadoPago API error (status
48                      ${apiResp.getStatusCode()}):
49                      ${apiResp.getContent()}", e)
50                  return
51              } catch (Exception e) {
52                  logger.error("Unexpected error fetching payment
53                      ${paymentId}: ${e.message}", e)
54                  return
55              }
56          }
57      ]></script>

```



```

52         logger.info("Params: ${params}")
53         ec.service.sync().name("moit.payments
54         .MercadoPagoServices
55         .process#MercadoPagoPaymentEvent")
56         .parameters(params).call()
57         return
58     }
59     else {
60         logger.warn("Unsupported webhook type: ${eventType}")
61         return
62     }
63     ]]> </script>
64 </actions>
65 </service>

```

process#MercadoPagoPaymentEvent

Este servicio se encarga de procesar los eventos de tipo pago recibidos de Mercado Pago. A partir de los parámetros de entrada *mpPaymentId*, *internalCheckoutId*, *paymentStatus*, *transactionAmount* y *currency*. Primero se busca en la entidad *MercadoPagoPayment* un registro existente con el mismo *mpPaymentId*; si no lo encuentra, crea uno nuevo asignándole *mpPaymentId*, *internalCheckoutId*, *paymentStatus*, *transactionAmount* y *currency*. En caso contrario, actualiza los campos *paymentStatus*, *transactionAmount* y *currency* del registro existente. A continuación, obtiene la sesión de pago correspondiente en la entidad *MercadoPagoCheckout* mediante *internalCheckoutId*. Si el estado almacenado difiere de *paymentStatus*, actualiza el campo *status* y guarda el cambio, registrando en el log la transición de estado. (Servicio completo en el Anexo B.4.3)

```

1 <service verb="process" noun="MercadoPagoPaymentEvent">
2   <in-parameters>
3     <parameter name="mpPaymentId" required="true"/>
4     <parameter name="internalCheckoutId"/>
5     <parameter name="paymentStatus" required="true"/>
6     <parameter name="transactionAmount" required="true"/>
7     <parameter name="currency" required="true"/>
8   </in-parameters>
9   <actions>
10    <!-- Update or create payment record -->
11    <entity-find-one
12      entity-name="moit.payments.MercadoPagoPayment"
13      value-field="paymentEntity">
14      <field-map field-name="mpPaymentId" from="mpPaymentId"/>
15    </entity-find-one>
16    <if condition="!paymentEntity">
17      <entity-make-value
18        entity-name="moit.payments.MercadoPagoPayment"
19        value-field="newPayment"/>
20      <set field="newPayment.mpPaymentId" from="mpPaymentId"/>

```

```

17     <set field="newPayment.internalCheckoutId"
        from="internalCheckoutId"/>
18     <set field="newPayment.paymentStatus" from="paymentStatus"/>
19     <set field="newPayment.transactionAmount"
        from="transactionAmount"/>
20     <set field="newPayment.currency" from="currency"/>
21     <entity-create value-field="newPayment"/>
22 </if>
23 <else>
24     <set field="paymentEntity.paymentStatus"
        from="paymentStatus"/>
25     <set field="paymentEntity.transactionAmount"
        from="transactionAmount"/>
26     <set field="paymentEntity.currency" from="currency"/>
27     <entity-update value-field="paymentEntity"/>
28 </else>
29 <!-- Update checkout status -->
30 <entity-find-one
        entity-name="moit.payments.MercadoPagoCheckout"
        value-field="checkoutEntity">
31     <field-map field-name="internalCheckoutId"
        from="internalCheckoutId"/>
32 </entity-find-one>
33 <if condition="checkoutEntity">
34     <if condition="checkoutEntity.status != paymentStatus">
35         <set field="checkoutEntity.status" from="paymentStatus"/>
36         <entity-update value-field="checkoutEntity"/>
37         <log message="Checkout ${internalCheckoutId} status
            updated to ${paymentStatus}"/>
38     </if>
39 </if>
40 </actions>
41 </service>

```

Servicio de ejemplo: create#MercadoPagoCheckoutInvoice

Este servicio sirve de ejemplo para saber como utilizar los servicios ya creados de Mercado Pago. Utiliza una entidad ya existente como una factura e implementa la lógica para crear un checkout de Mercado Pago a partir de la información de la factura. Recibe como entrada el *invoiceId* y consulta los *InvoiceItem* asociados para recopilar su *id*, descripción (o un texto por defecto), monto y cantidad. Convirtiéndolos en la lista *payingItems* que necesita el servicio que crea el checkout. Luego, asigna en el contexto los parámetros necesarios (*payingItems*, *externalReference* con el mismo *invoiceId* y *partyId* del usuario actual) y delega la creación del checkout al servicio *create#MercadoPagoCheckout*, mapeando el contexto de entrada y salida. Finalmente, deja en *checkoutUrl* la URL resultante y registra en los logs la generación del checkout para la factura. Demostrando la utilidad de este componente plug and play.

```

1 <!-- ===== EXAMPLE OF CHECKOUT SERVICE USAGE ===== -->

```

```

2 <service verb="create" noun="MercadoPagoCheckoutInvoice"
  type="inline" authenticate="anonymous-all">
3   <description>Generate a Mercado Pago Checkout for a given
    invoice</description>
4   <in-parameters>
5     <parameter name="invoiceId" type="String" required="true"/>
6   </in-parameters>
7   <out-parameters>
8     <parameter name="checkoutUrl" type="String"/>
9   </out-parameters>
10  <actions>
11  <script language="groovy"><![CDATA[
12    // Fetch invoice items
13    def invoiceItems =
14      ec.entity.find("mantle.account.invoice.InvoiceItem")
15      .condition("invoiceId", invoiceId).list()
16
17    // Build list of maps
18    def itemsList = invoiceItems.collect { item ->
19      [
20        id: item.invoiceItemSeqId,
21        title: item.description ?: "Invoice Item
22          ${item.invoiceItemSeqId}",
23        unitPrice: (item.amount ?: BigDecimal.ZERO).toString(),
24        quantity: (item.quantity ?: BigDecimal.ONE).intValue()
25      ]
26    }
27
28    // Validate
29    if (!itemsList) {
30      ec.message.addError("Invoice has no items")
31      return
32    }
33    context.payingItems = itemsList
34    context.externalReference = invoiceId
35    context.partyId = ec.user.userId
36  ]]></script>
37
38  <!-- Call your Checkout service -->
39  <service-call name="moit.payments.MercadoPagoServices
40    .create#MercadoPagoCheckout"
41    in-map="context" out-map="context"/>
42
43  <log message="Mercado Pago checkout created for invoice
    ${invoiceId}: ${checkoutUrl}"/>
44  </actions>
45 </service>

```

Notar que si está configurado el webhook correctamente, al llegar los eventos de pago

de esta factura, se actualizará el estado del *MercadoPagoCheckout* y creará su respectivo *MercadoPagoPayment*.

3.2. Configuración autenticación Webhook

Mientras se intentaba implementar la autenticación de los webhooks de Mercado Pago, se vio que la implementación de Moqui no era suficiente. Su tipo de autenticación más similar HMAC SHA-256 con time stamp, no funcionaba dado a que MercadoPago tiene su propia autenticación customizada. Para habilitar la verificación de las notificaciones webhook de Mercado Pago, es necesario extender la el código Moqui *WebFacadeImpl.groovy* con un manejador específico para el *SystemMessageAuthType* *SmatMercadoPago*. El flujo de autenticación implementado contempla:

1. **Parseo de cabecera:** se extraen los valores *ts* (timestamp) y *v1* (firma) del header *x-signature*.
2. **Generación de template:** se construye la cadena en el formato **id:[data.id];request-id:[x-request-id];ts:[ts]**; incluyendo solo los campos presentes.
3. **Verificación HMAC:** con el *sharedSecret* configurado se genera la firma HMAC SHA-256 de dicha cadena y se compara con el valor *v1* entrante.
4. **(Opcional) Restricción temporal:** aunque está disponible para comparar que el timestamp venga dentro de un rango de 5 minutos, se omite en pruebas por inconsistencias en el entorno de pruebas de MercadoPago (timestamps son inconsistentes con la documentación).

Extensión de WebFacadeImpl.groovy

A continuación, el fragmento que debe incluirse dentro de la cláusula que maneja los *SystemMessageAuthType*:

```
1 // This code should be present in WebFacadeImpl.groovy
2 } else if ("SmatMercadoPago".equals(messageAuthEnumId)) {
3     // Validate Mercado Pago x-signature HMAC header using
      sharedSecret
4     String authHeaderName = (String)
      systemMessageRemote.authHeaderName
5     String sharedSecret = (String) systemMessageRemote.sharedSecret
6
7     String headerValue = request.getHeader(authHeaderName ?:
      "x-signature")
8     String xRequestIdHeader = request.getHeader("x-request-id")
9     // Log request details
10    logger.info("Request details:")
11    logger.info("Headers:")
```

```

12     request.getHeaderNames().asIterator().each { headerName ->
13         logger.info("${headerName}:
14             ${request.getHeader(headerName)}}")
15     }
16     logger.info("Body: ${messageText}")
17
18
19     if (!headerValue) {
20         logger.warn("System message receive Mercado Pago verify no
21             header ${authHeaderName} value found, for remote
22             ${systemMessageRemoteId}")
23         response.sendError(HttpServletResponse.SC_FORBIDDEN, "No
24             signature header ${authHeaderName} found for remote
25             system ${systemMessageRemoteId}")
26         return
27     }
28
29     // Parse x-signature header: ts=...,v1=...
30     String timestamp = null;
31     String incomingSignature = null;
32     String[] headerValueList = headerValue.split(",") // split on
33         comma
34     for (String headerValueItem : headerValueList) {
35         String key = headerValueItem.split("=")[0].trim()
36         if ("ts".equals(key))
37             timestamp = headerValueItem.split("=")[1].trim()
38         else if ("v1".equals(key))
39             incomingSignature = headerValueItem.split("=")[1].trim()
40     }
41
42     if (!timestamp || !incomingSignature) {
43         logger.warn("System message receive Mercado Pago invalid
44             x-signature format for remote ${systemMessageRemoteId}")
45         response.sendError(HttpServletResponse.SC_FORBIDDEN,
46             "Invalid signature format for remote system
47             ${systemMessageRemoteId}")
48         return
49     }
50
51     // Get data.id from query params
52     String dataId = request.getParameter("data.id") ?: ""
53
54     // mercado pago signature template
55     id:[data.id_url];request-id:[x-request-id_header];ts:[ts_header];
56     // If any of the values are missing, we must remove the value
57     from the signature template
58     StringBuilder manifestBuilder = new StringBuilder()
59     if (dataId)
60         manifestBuilder.append("id:").append(dataId).append(";")

```

```

50     if (xRequestIdHeader) manifestBuilder.append("request-id:")
51         .append(xRequestIdHeader).append(";")
52     manifestBuilder.append("ts:").append(timestamp).append(";")
53     String signatureTextToVerify = manifestBuilder.toString()
54
55
56     Mac hmac = Mac.getInstance("HmacSHA256")
57     hmac.init(new
58         SecretKeySpec(sharedSecret.getBytes(StandardCharsets.UTF_8),
59             "HmacSHA256"))
60
61     // NOTE: if this fails try with "ISO-8859-1"
62     byte[] hash =
63         hmac.doFinal(signatureTextToVerify.getBytes(StandardCharsets.UTF_8));
64
65     StringBuilder signatureBuilder = new StringBuilder()
66     for (byte b : hash) {
67         signatureBuilder.append(Integer.toString((b & 0xff) +
68             0x100, 16).substring(1))
69     }
70     String generatedSignature = signatureBuilder.toString()
71
72     if (!incomingSignature.equals(generatedSignature)) {
73         logger.warn("System message receive HMAC verify header
74             value ${incomingSignature} calculated
75             ${generatedSignature} did not match for remote
76             ${systemMessageRemoteId}")
77         response.sendError(HttpServletResponse.SC_FORBIDDEN, "HMAC
78             verify failed for remote system
79             ${systemMessageRemoteId}")
80     }
81     return
82 }
83
84 //Here Optionally we can add time restrictions (5 min for
85     example)
86 // ...
87
88 // login anonymous if not logged in
89 eci.userFacade.loginAnonymousIfNoUser()
90 }

```

Opcional: Restricción temporal

Si se desea agregar una restricción temporal, se puede agregar el siguiente código:

```

1 // THIS VALIDATION WORKS WHEN SIMULATING USING MERCADO PAGO
2 // SIMULATE BUTTON
3
4 // HOWEVER MERCADO PAGO IS HAVING ISSUES WITH THE TIMESTAMP, WHILE
5 // USING TEST CREDENTIALS, AND TEST ACCOUNT

```

```

3 // THIS IS WHY WE ARE NOT USING THIS VALIDATION FOR NOW
4 long tsLong = Long.parseLong(timestamp)
5
6 // Detect and convert seconds to milliseconds if needed (Mercado
  pago has inconsistencies)
7 if (tsLong < 1_000_000_000_000L) {
8     logger.warn("Mercado Pago ts (${tsLong}) is in seconds;
      converting to milliseconds")
9     tsLong *= 1000
10 } else {
11     logger.info("Mercado Pago ts (${tsLong}) is already in
      milliseconds")
12 }
13 Timestamp incomingTimestamp = new Timestamp(tsLong)
14
15 // Add 10 seconds to now timestamp to allow for clock skew (10
  seconds = 10000 milliseconds = 10*1000)
16 Timestamp nowTimestamp = new
  Timestamp(eci.user.nowTimestamp.getTime() + 10000)
17 // If timestamp was not sent in past 5 minutes, reject message (5
  minutes = 300000 milliseconds = 5*60*1000)
18 Timestamp beforeTimestamp = new Timestamp(nowTimestamp.getTime() -
  300000)
19 if (!incomingTimestamp.before(nowTimestamp) ||
  !incomingTimestamp.after(beforeTimestamp) ){
20     logger.warn("System message receive HMAC invalid incoming
      timestamp where before timestamp ${beforeTimestamp} <
      incoming timestamp ${incomingTimestamp} < now timestamp
      ${nowTimestamp}" )
21     response.sendError(HttpServletResponse.SC_FORBIDDEN, "HMAC
      timestamp verification failed")
22     return
23 }

```

3.2.1. Configuración para probar Webhook

Para validar localmente las notificaciones de Mercado Pago sin desplegar en un servidor público, se utilizó **ngrok** para exponer temporalmente el endpoint de Moqui. A continuación se describen los pasos:

1. **Instalar ngrok** Descarga y descomprime ngrok según tu sistema operativo desde <https://ngrok.com/> y añade el ejecutable a tu PATH.
2. **Iniciar ngrok** Abre una terminal y ejecuta (ya que Moqui usa el puerto 8080):

```
1 ngrok http 8080
```

Esto creará dos URLs públicas, por ejemplo:

- `https://abcd1234.ngrok.app`
- `http://abcd1234.ngrok.app`

3. **Actualizar Webhook en Mercado Pago** En el panel de desarrollador de Mercado Pago, edita la URL de notificaciones (*webhook URL*) para que apunte a:

`https://abcd1234.ngrok.app/rest/sm/MercadoPagoWebhook/MercadoPago`

(reemplaza `abcd1234.ngrok.app` por tu subdominio `ngrok`).

De esta manera al completar el checkout usando las credenciales y tarjetas de prueba de Mercado Pago, se puede verificar que el pago se procesa correctamente (el evento de pago se recibe en Moqui, autentica que proviene de Mercado Pago y actualiza las entidades correspondientes).

3.3. Adaptación/Integración con Mantle

Al concluir la implementación del componente independiente a Mantle, se adaptó para integrarlo con la lógica, los servicios y las entidades de Mantle. Tener el módulo independiente funcionando simplificó notablemente la integración de nueva versión del módulo. Esto se debe a que se superó el principal obstáculo: disponer de un SDK de MercadoPago operativo y que los servicios que lo utilizan funcionen correctamente. Solo quedaba enfocarse en utilizar correctamente las entidades y servicios de Mantle. Resultando en un módulo que se integra de forma correcta con el ecosistema de Moqui.

Para su correcto funcionamiento, se deben definir las siguientes propiedades (en *Moqui-Conf.xml* C.1):

Notar que `currency` ahora es manejado por el flujo de Mantle.

- *mercado_pago_access_token*: Token de acceso a la API de MercadoPago.
- *mercado_pago_success_url*: URL de retorno tras pago exitoso.
- *mercado_pago_pending_url*: URL de retorno cuando el pago queda pendiente.
- *mercado_pago_failure_url*: URL de retorno tras pago fallido o cancelado.

3.3.1. Arquitectura general

La implementación sigue la arquitectura de pagos de Mantle USL, aprovechando los servicios estándar (*PaymentServices.authorize#Payment*, *capture#Payment*, *refund#Payment*, etc.) y las configuraciones de pasarela de pago. En concreto, se crea una entrada en *PaymentGatewayConfig* (p. ej. *MercadoPagoCheckoutPro*) con los nombres de servicio definidos

en este componente para autorizar y capturar pagos. Dicha configuración se asocia al *ProductStorePaymentGateway* de la tienda correspondiente (por ejemplo, *DemoStore* y el *paymentInstrument PiMercadoPago*). El flujo general consiste en crear un registro de *Payment* en Mantle (al crear la orden o factura) con status *PmntPromised* y luego invocar el servicio *authorize#MercadoPago* para iniciar el pago en MercadoPago dejando el payment en status *PmntAuthorized*. Tras el pago, los webhooks de MercadoPago son manejados mediante mensajes de sistema (*SystemMessage*, igual que en el módulo independiente), que invocan el servicio *consume#MercadoPagoWebhookEvent* para procesar el resultado y actualizar el pago interno usando Mantle (finalmente dejando al payment en status *PmntDelivered*).

Este es el flujo general de pago:

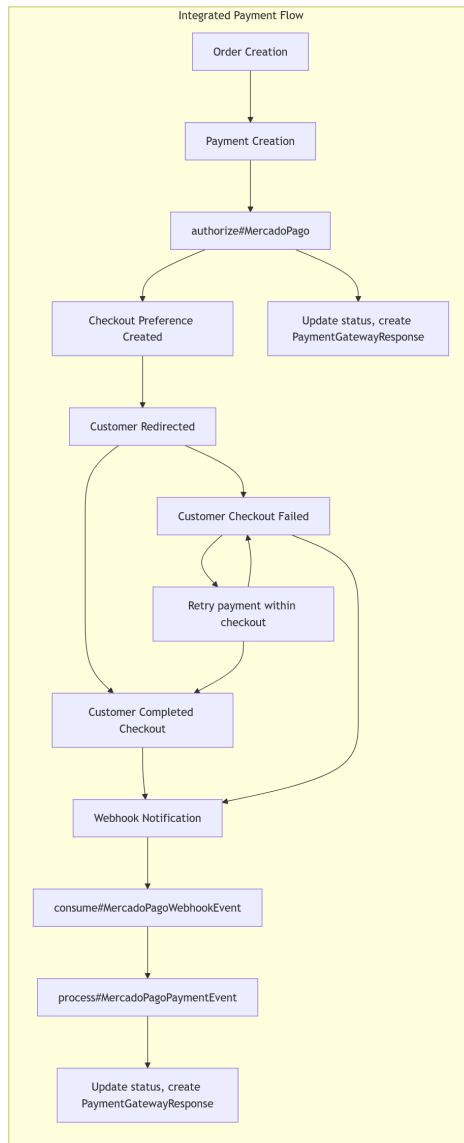


Figura 3.2: Diagrama de flujo de pago

3.3.2. Modelo de Datos (Entidades)

En la integración con Mantle no se crearon nuevas entidades específicas de MercadoPago, sino que se utilizan y extienden las existentes en Mantle. Las principales entidades involucradas son:

- **PaymentGatewayConfig**: define la configuración de la pasarela de pagos para MercadoPago (tipo *PgtMercadoPago*), indicando/mapeando los servicios *authorize*, *capture*, etc., implementados por este componente.
- **ProductStorePaymentGateway**: vincula el instrumento de pago (*PiMercadoPago*) y la tienda (por ejemplo, *DemoStore*) con la configuración de gateway (*MercadoPagoCheckoutPro*).
- **Payment**: entidad principal que representa el pago interno. El campo *paymentId* de *Payment* se usa como *externalReference* al crear la preferencia de pago en MercadoPago (i.e. se deja definido el *paymentId* en la metadata de la preferencia). Tras el pago, este registro se actualiza con el estado correspondiente (p. ej. *PmntDelivered*, *PmntDeclined*, etc.).
- **PaymentGatewayResponse**: almacena la respuesta de la pasarela ante cada operación (autorización, captura, etc.). Se extiende esta entidad para incluir la URL de checkout generada por MercadoPago. Esto se hizo para poder redirigir al usuario a su correspondiente hosted checkout (así en la screen se puede obtener la url de el *PaymentGatewayResponse*).

La extensión de la entidad *PaymentGatewayResponse* se define así (archivo completo en el Anexo C.2):

```
1 <extend-entity entity-name="PaymentGatewayResponse"
   package="mantle.account.method">
2 <field name="checkoutUrl" type="text-long" description="Checkout
   URL de MercadoPago"/>
3 </extend-entity>
```

Elecciones de diseño

Se decidió aprovechar al máximo la infraestructura de Mantle. Por ello, el flujo de pago interno se basa en la entidad *Payment*, y las respuestas del gateway se registran en *PaymentGatewayResponse*. No se implementó un equivalente a *MercadoPagoCustomer*, ya que Mantle ya asocia los pagos a clientes internos (*partyId*) mediante su modelo estándar (también el servicio de Checkout Pro no utiliza/maneja los customers). Además, los webhooks de MercadoPago se manejan de la misma manera que en el componente independiente, mediante el sistema de mensajes de Moqui, usando HMAC SHA-256 para validar la firma customizada (*SystemMessageAuthType SmatMercadoPago*). Al utilizar el ecosistema de Moqui es

necesario adherirse a la interfaz de servicios (i.e. *authorize#Payment*, *capture#Payment*, *refund#Payment*, etc.) de Mantle. Esto permite que Mantle ocupe este nuevo método de pago sin necesidad de modificar el código de Mantle.

Datos Set-up de componente

El componente incluye datos semilla para configurar las entidades de Mantle necesarias. Se utiliza la misma configuración the Moqui SystemMessage de el módulo independiente para manejar los webhooks de MercadoPago. Sin embargo en la carpeta data ahora también se necesitan las enumeraciones y configuración de pasarela de pago. Estos datos son fundamentales para que el componente funcione correctamente, ya que son los que habilitan la integración con Mantle. Se define el instrumento de pago *PiMercadoPago*, el gateway *PgtMercadoPago*. Además se define la tienda *DemoStore* y la configuración de la pasarela de pago *MercadoPagoCheckoutPro*. Con este setup Mantle reconoce el nuevo método de pago y lo muestra como una opción de pago. (archivos completos en el Anexo *MPsetupWebhook.xml* C.5, *MPinitialGatewayData.xml* C.4 y *MPinitialEnumerations.xml* C.3)

```
1 <entity-facade-xml type="seed">
2   <!-- Payment enums for Mercado Pago checkout -->
3   <moqui.basic Enumeration enumTypeId="PaymentInstrument"
4     enumId="PiMercadoPago" description="Mercado Pago Checkout
5     Pro"/>
6
7   <moqui.basic Enumeration enumTypeId="PaymentGatewayType"
8     enumId="PgtMercadoPago" description="Mercado Pago"/>
9   <moqui.basic Enumeration enumTypeId="PaymentMethodType"
10     enumId="PmtMercadoPagoAccount" description="Mercado Pago
11     Account"/>
12
13   <mantle.product.store.ProductStore productStoreId="DemoStore"/>
14
15   <mantle.account.method.PaymentGatewayConfig
16     paymentGatewayConfigId="MercadoPagoCheckoutPro"
17     paymentGatewayTypeEnumId="PgtMercadoPago"
18     description="Mercado Pago Checkout Pro"
19     authorizeServiceName=
20       "moit.payments.MercadoPagoServices
21       .authorize#MercadoPago"
22     captureServiceName=
23       "moit.payments.MercadoPagoServices
24       .capture#MercadoPago"/>
25
26   <mantle.product.store.ProductStorePaymentGateway
27     productStoreId="DemoStore"
28     paymentInstrumentEnumId="PiMercadoPago"
29     paymentGatewayConfigId="MercadoPagoCheckoutPro"/>
30 </entity-facade-xml>
```

3.3.3. Servicios Principales

La lógica de pago se implementa con varios servicios Moqui que interactúan con la API de MercadoPago dentro del flujo de Mantle. Para esto se separaron los servicios en dos grupos: (1) servicios auxiliares que utiliza MercadoPago *MercadoPagoCore.xml* y (2) servicios principales que adhieren a la interfaz estándar de Mantle *MercadoPagoServices.xml*. Los servicios auxiliares son los que se encargan de crear la preferencia de pago, el checkout y el procesamiento de los eventos de pago. Mientras que los servicios principales cuando es necesario se invocan los servicios auxiliares.

Servicios Core

Para el desarrollo de estos servicios core se utilizaron como base los servicios desarrollados en el módulo independiente. Donde se adaptaron para que se integren con Mantle.

- **create#MercadoPagoCheckout:** Recupera de configuración el *accessToken* y las URLs de retorno (*success*, *pending*, *failure*), valida los parámetros obligatorios (*payinItems*, *externalReference*, *currency*) y calcula el total de la orden. Este no crea ningún registro en ninguna entidad de Mantle. Solo se crea la preferencia de pago, y el checkout url. Crea un *PreferenceRequest* (ítems, backUrls, metadatos, payer) y lo envía usando el SDK de Mercado Pago.

Al recibir la respuesta, retorna *checkoutUrl*, *mpPreferenceId* y *success=true*. Este servicio se invoca desde el servicio *authorize#MercadoPago* de Mantle. El código de este servicio se basó en el servicio *create#MercadoPagoCheckout* del módulo independiente.

La diferencia es que en este caso no se interactúa con ninguna entidad, solamente crea la sesión de checkout y retorna la información de la preferencia de pago para posteriormente ser usada en el servicio *authorize#MercadoPago*. (servicio checkout completo en el Anexo C.7.1)

- **process#MercadoPagoPaymentEvent:** *mpPaymentId*, *externalReference*, *paymentStatus*, *transactionAmount* y *currency*. Luego busca el *Payment* de Mantle usando *externalReference* que en este caso es el *paymentId* que indica el *Payment* de Mantle asociado a la preferencia de pago (checkout). Se valida que el *Payment* exista, levantando un error en caso contrario.

En el proceso se mapea el estado de Mercado Pago (*paymentStatus*) a los estados estándar de Mantle (*PmntDelivered*, *PmntAuthorized*, *PmntDeclined*, *PmntVoid*). En caso de que el status de Mercado Pago se desconozca, se mantiene el status actual del *Payment*.

Para tener idempotencia se verifica que el status mapeado de Mercado Pago no sea el mismo que el status actual del *Payment*. Si es así, no se actualiza el *Payment*. Si el status es distinto se crea un *PaymentGatewayResponse* para auditar la operación (tiene toda la información del evento de pago).

Finalmente se actualiza el payment *mantle.account.payment.Payment* con el status nuevo. Un desafío particular fue manejar cuando el checkout es rechazado pero dentro de la

misma sesión de checkout el usuario vuelve a pagar. Cuando el pago es rechazado MercadoPago manda un evento de pago con status *rejected*. Cuando este evento se procesa el Payment queda con status *PmntDeclined*. Si el usuario intenta pagar de nuevo de manera exitosa, MercadoPago envía un evento de pago con estado *approved*. La lógica de mantle no permite actualizar un status de payment de *PmntDeclined* a *PmntDelivered*, por lo tanto cuando este caso ocurre se actualiza el Payment a status *PmntAuthorized* antes de actualizarlo a *PmntDelivered*. Cualquier otro cambio de status se actualiza directamente.

Este servicio se invoca desde *consume#MercadoPagoWebhookEvent* de Mantle. El código completo de este servicio se puede ver en el anexo C.7.2.

consume#MercadoPagoWebhookEvent

Este servicio consume las notificaciones webhook de Mercado Pago recibidas a través de un *SystemMessage* mediante el parámetro *systemMessageId*. Hace lo mismo que en el módulo independiente, obtiene los detalles de la transacción (ID, metadata, estado, monto y moneda).

Con estos datos construye el mapa de parámetros *{mpPaymentId, externalReference, paymentStatus, transactionAmount, currency}* y llama de forma síncrona al servicio *process#MercadoPagoPaymentEvent* integrado a Mantle para actualizar en la base de datos el registro *Payment* y crear su *PaymentGatewayResponse* correspondiente. En caso de errores en la API o en el parsing, registra la excepción y retorna sin interrumpir el flujo global de mensajes. El código completo de este servicio se puede ver en el anexo C.8.1.

authorize#MercadoPago

Este servicio implementa el contrato *PaymentServices.authorize#Payment*. Localiza el *Payment* (levanta un error si no existe) y su *PaymentGatewayConfig*.

Obtiene información de la orden para crear la preferencia de pago. Construye dinámicamente la lista de ítems (Si no hay orden asociada asume que es un pago de una factura) que necesita el servicio *create#MercadoPagoCheckout* para generar la preferencia de pago y el checkout url. Con la lista creada se invoca a el servicio auxiliar para crear la sesión de pago. A continuación si la creación de la sesión es exitosa, registra un *PaymentGatewayResponse* con *PgoAuthorize*, código de respuesta provisional y URL de checkout (en el campo extendido *checkoutUrl*), dejando preparado el flujo de redirección del cliente.

(Servicio completo en el Anexo C.8.2)

```
1 <service verb="authorize" noun="MercadoPago">
2   <implements
3     service="mantle.account.PaymentServices.authorize#Payment"/>
4   <actions>
5     <!-- Get payment details -->
```

```

6      <entity-find-one entity-name="mantle.account.payment.Payment"
      value-field="payment"/>
7      <if condition="payment == null"><return error="true"
      message="Payment ${paymentId} not found"/></if>
8      ...
9      <!-- Get gateway configuration and access token -->
10     <entity-find-one
      entity-name="mantle.account.method.PaymentGatewayConfig"
      value-field="gatewayConfig">
11         <field-map field-name="paymentGatewayConfigId"/>
12     </entity-find-one>
13     ...
14     <script language="groovy"><![CDATA[
15         // Build list of maps
16         def itemsList = [[
17             id: "payment_${paymentId}",
18             title: payment.orderId ? "Order Payment
      #${payment.orderId}" : "Invoice Payment",
19             unitPrice: payment.amount,
20             quantity: 1
21         ]]
22         // Validate
23         if (!itemsList || itemsList.isEmpty()) {
24             ec.message.addError("List for checkout creation has no
      items")
25             return
26         }
27
28         ec.context.itemsList = itemsList
29
30     ]]></script>
31
32     <!-- Call your Checkout service -->
33     <service-call
      name="moit.payments.MercadoPagoServices.create#MercadoPagoCheckout"
34         in-map="[payingItems:itemsList,
      externalReference:paymentId,
      currency:payment.amountUomId]"
      out-map="checkoutResult"/>
35     ...
36     <!-- Extract results -->
37     <set field="preferenceId"
      from="checkoutResult.mpPreferenceId"/>
38     <set field="checkoutUrl" from="checkoutResult.checkoutUrl"/>
39     ...
40     <!-- Create PaymentGatewayResponse record for preference
      creation -->
41     <service-call
      name="create#mantle.account.method.PaymentGatewayResponse"
      out-map="context"

```

```

42         in-map="[paymentGatewayConfigId:paymentGatewayConfigId,
                    paymentOperationEnumId:'PgoAuthorize',
43         paymentId:paymentId,
                    paymentMethodId:payment.paymentMethodId,
44         amount:payment.amount,
                    amountUomId:payment.amountUomId,
45         referenceNum:preferenceId,
                    approvalCode:preferenceId,
46         responseCode:'pending', reasonMessage:'Checkout
                    preference created, redirect customer to
                    checkout URL',
47         checkoutUrl: checkoutUrl,
48         transactionDate:ec.user.nowTimestamp,
49         resultSuccess:'Y', resultDeclined:'N',
                    resultError:'N']"/>
50     ...
51     </actions>
52 </service>

```

capture#MercadoPago

Este servicio implementa *mantle.account.PaymentServices.capture#Payment*. Dado que en Checkout Pro la captura se realiza vía webhook, este servicio comprueba si ya existe un *PaymentGatewayResponse*. Si lo encuentra, retorna sin errores, y si no, informa que la captura sólo puede completarse mediante notificación de webhook, manteniendo así la consistencia con el modelo asíncrono de Mercado Pago. (Servicio completo en el Anexo C.8.3)

refund#MercadoPago

Este servicio no está completo, sin embargo deja la base para la implementación de reembolsos (dejando comentado el código necesario para la implementación real). Obtiene el *Payment*, simula la llamada a la API de Mercado Pago para reembolsos que genera un identificador de reembolso y finalmente crea un *PaymentGatewayResponse* con *PgoRefund*, registrando monto, estado y referencia del reembolso, listo para extenderse con la integración real de la API. (Template completo en el Anexo C.8.4)

release#MercadoPago

Este servicio no está completo, sin embargo deja la base para la implementación de anulaciones (dejando comentado el código necesario para la implementación real). Para el caso de Checkout Pro solo se pueden anular pagos que se encuentren en estado pending. Por lo tanto primero obtiene el *Payment*, verifica que su estado permita anulación (por ejemplo, *PmntAuthorized*), simula la llamada de cancelación a Mercado Pago (plantilla comentada) y crea un *PaymentGatewayResponse* con *PgoRelease*. Si la anulación es exitosa, actualiza el

estado del *Payment* a *PmntVoid*, completando el ciclo de autorización sin captura. (Template completo en el Anexo C.8.4)

3.3.4. Prueba de funcionamiento usando MarbleERP

Marble es una aplicación ERP para minoristas y mayoristas con una funcionalidad operativa y de gestión integral. Este componente esta construido sobre Mantle USL, Mantle UDM y SimpleScreens. Al usar Mantle de manera completa esta aplicación es buena para probar la integración de MercadoPago con Mantle. Para esto se necesita adaptar parte de código de SimpleScreens para que se redirija al checkout de MercadoPago. Esto se hizo de manera que cuando el usuario haga click en authorize payment se redirija al checkout de MercadoPago.

```
1 <transition name="gatewayAuthorize">
2   <service-call name="mantle.account.PaymentServices
3     .authorize#SinglePayment" out-map="authResult"/>
4   <actions>
5     <if condition="authResult.paymentGatewayResponseId">
6       <entity-find-one entity-name="mantle.account.method
7         .PaymentGatewayResponse"
8         value-field="gatewayResponse">
9         <field-map field-name="paymentGatewayResponseId"
10           from="authResult.paymentGatewayResponseId"/>
11       </entity-find-one>
12       <if condition="gatewayResponse?
13         .checkoutUrl?.contains('http')">
14         <set field="checkoutUrl"
15           from="gatewayResponse.checkoutUrl"/>
16       </if>
17     </if>
18   </actions>
19   <conditional-response url="{checkoutUrl}" url-type="plain"
20     condition="checkoutUrl"/>
21   <default-response url="."/>
22 </transition>
```

Con este código incorporado a SimpleScreens se puede probar la integración de Mantle con MarbleERP.

Estos son los pasos para probar la integración de Mantle con MarbleERP:

1. Configuración de Store

El componente de MarbleERP trae datos semillas para probar la aplicación. En específico trae productos, stores, descuentos, etc. En este caso para poder probar la integración de Mantle con MarbleERP se necesita agregar el nuevo método de pago a la store de prueba. Esto se logra navegando a la store de prueba y agregando el nuevo método de pago. 3.3

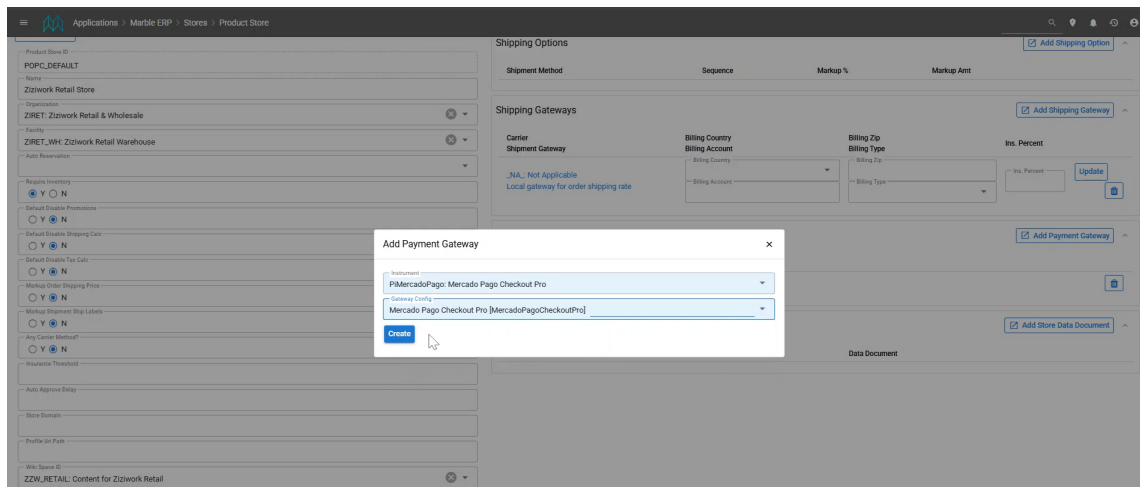


Figura 3.3: Configuración de Store en MarbleERP.

2. Creación de Consumer y Order

Luego se debe crear un consumer y una sales order asociada a él. Una vez en la pantalla que muestra la información de la order, se deben seleccionar los productos. Finalmente se debe crear un payment para la order. Este Payment debe crearse con el instrumento de pago *PiMercadoPago* y tendrá status *PmntPromised*. La vista Order Detail se puede ver en la figura 3.4.

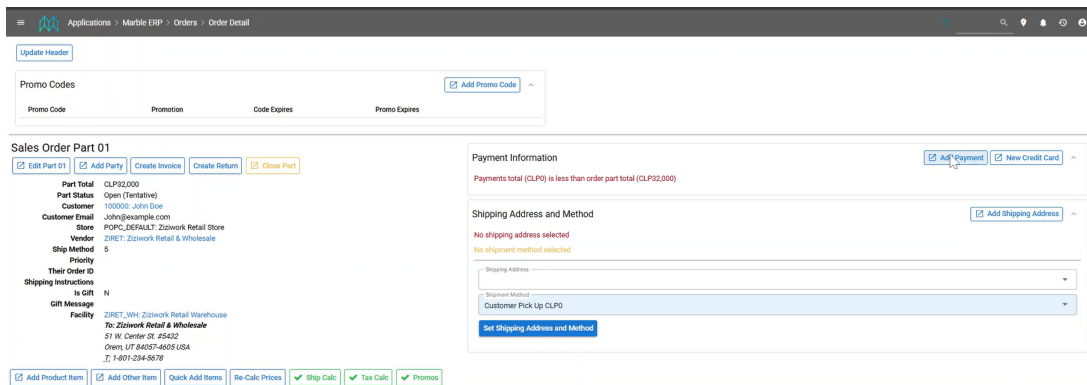


Figura 3.4: Detalle de orden en MarbleERP.

3. Autorización de Payment

Después se debe autorizar el payment. Esto se logra haciendo click en el botón de gateway authorize. Tras hacer click el usuario será redirigido al checkout de MercadoPago. Donde podrá ingresar los datos de la tarjeta de prueba de MercadoPago. Una vez que el pago es exitoso, el usuario será redirigido al success url seteado en *MoquiConf.xml*. El botón de autorización se puede ver en la figura 3.5. También se puede ver el proceso de pago en MercadoPago en las figuras 3.6 y 3.7.

Payment Information [Add Payment](#) [New Credit Card](#) ^

Mercado Pago Checkout Pro (Invoice Payment) CLP32,000 - Status: Promised [Update](#) [Gateway Authorize](#)

[Edit Payment 100000](#)

Figura 3.5: Autorización de pago en MarbleERP.

Subtotal

Completa los datos de tu tarjeta de crédito o débito

Número de tarjeta
5416 7526 0258 2580

Nombre del titular
APRO

Vencimiento
11/30

Código de seguridad
123

Detalles del pago
Order Payment #100000 \$ 32.000

[Volver](#) [Continuar](#)

Figura 3.6: Formulario de pago en MercadoPago.

¡Listo! Tu pago ya se acreditó
Operación #1324038106

Pagaste 1x \$ 32.000 (Total \$ 32.000)
Mastercard **** 2580 Mastercard Crédito
En el resumen de tu tarjeta verás el cargo a nombre de
Mercadopago*fake

[← Volver al sitio](#) En 1 segundos te llevaremos a .

Figura 3.7: Pago exitoso en MercadoPago.

4. Verificación de pago

Una vez que el pago es exitoso, el usuario al ingresar a la pantalla específica del Payment debe poder ver que el status del payment es *PmntDelivered* 3.8 como también todos los GatewayResponses asociados al Payment 3.9.

Status History			^
2025-07-13 00:31		Delivered	
2025-07-13 00:30 by John Doe [john.doe]		Authorized	
2025-07-13 00:28 by John Doe [john.doe]		Promised	

Figura 3.8: Historial de estados de pago en MarbleERP.

Gateway Responses					^
Gateway Config	Ref Num		Success	AVS	
Operation	Approval		Decline	CdV	
Amount	Resp Code		Error		
TX Date	Reason				
Message	Status				
Mercado Pago Checkout Pro					
Capture	1324038106		Y		
CLP32,000	1324038106		N	N/A	
2025-07-13 00:31	success		N	N/A	
Payment approved by Mercado Pago					
Mercado Pago Checkout Pro					
Authorize	2428398404-1df62ed4-46df-4a72-a261-458062cbab59		Y		
CLP32,000	2428398404-1df62ed4-46df-4a72-a261-458062cbab59		N	N/A	
2025-07-13 00:30	pending		N	N/A	
Checkout preference created, redirect customer to checkout URL					

Figura 3.9: Historial de respuestas de la pasarela en MarbleERP.

Capítulo 4

Conclusión

4.1. Retrospectiva

El objetivo principal de este proyecto fue diseñar, desarrollar e implementar un módulo para la integración de medios de pago electrónicos en el *framework* Moqui, con el propósito de facilitar futuras integraciones con distintas pasarelas. En este contexto, se logró una integración completa con MercadoPago Checkout Pro, lo cual representa un avance significativo y cumple con el núcleo del objetivo planteado. Los resultados obtenidos han sido exitosos, estableciendo una base sólida y extensible que permitirá incorporar nuevos proveedores con mayor facilidad en el futuro.

Durante el desarrollo, se adquirió un entendimiento profundo de la arquitectura del *framework* Moqui y del módulo Mantle, en particular respecto a las entidades y servicios relacionados con el procesamiento de pagos. Se construyeron dos versiones funcionales del módulo: una independiente y otra acoplada a Mantle, ambas diseñadas para manejar transacciones electrónicas mediante MercadoPago Checkout Pro.

El proceso de implementación presentó diversos desafíos técnicos y conceptuales. En primer lugar, el desconocimiento inicial sobre el funcionamiento general de los medios de pago electrónicos requirió una etapa previa de investigación para comprender sus flujos y modelo de datos. Obtener las credenciales y configurar cuentas de desarrollador con MercadoPago también supuso un obstáculo adicional, debido a las validaciones requeridas y la escasa orientación disponible para entornos de prueba.

Desde el punto de vista técnico, programar en Moqui representó una curva de aprendizaje compleja. La sintaxis propia del *framework*, su enfoque basado en archivos XML para definir servicios, entidades y pantallas, dificultaron el inicio del desarrollo (la documentación fue difícil de digerir). En varias ocasiones surgieron errores inesperados y poco descriptivos, cuya resolución implicó un proceso de exploración y prueba para comprender completamente la estructura interna del *framework*.

Asimismo, la complejidad de Mantle y su limitada documentación de alto nivel exigieron un esfuerzo considerable para comprender e integrar correctamente nuevas funcionalidades

sin afectar la lógica existente. La integración con MercadoPago también presentó desafíos, especialmente en lo relativo a la validación de *webhooks* y su firma basada en *timestamp*, cuya documentación oficial resultó ambigua e inconsistente. Esto generó incertidumbre al momento de verificar notificaciones asíncronas. Se abordaron retos específicos en la autenticación de eventos entrantes, debiendo implementar una verificación robusta de firmas HMAC-SHA256, considerando una validación específica que solo aplica para MercadoPago (utiliza una validación muy particular que se tenía que agregar al *framework*).

A pesar las dificultades encontradas durante desarrollo del proyecto, se estableció una infraestructura sólida y funcional que demuestra la viabilidad de integrar pasarelas de pago modernas en Moqui. Se ganó experiencia práctica en el uso del SDK de MercadoPago (SDKs de pago en general), construcción de componentes Moqui, interacción con Mantle UDM/USL y con la validación HMAC-SHA256 usando *sharedSecret*. El desarrollo de este módulo fue una experiencia llena de aprendizajes tanto técnicos (implementación de servicios, configuraciones y entidades), como conceptuales (mayor entendimiento acerca flujos de pago/facturas).

4.2. Trabajo a futuro

El diseño modular del componente, junto con su documentación técnica, permite extender su funcionalidad con mayor facilidad. Proveedores como WebpayPlus (Transbank), PayPal o incluso Stripe podrían ser integrados utilizando la misma/similar arquitectura, aprovechando los servicios y entidades ya definidos en este proyecto.

Una mejora sugerida sería modificar el código de los servicios para que queden dentro de una clase (en carpeta *scripts*), evitando la duplicación de código. Esto no se hizo debido a que obstaculizaba el desarrollo del módulo. Sin embargo como los servicios quedaron funcionales, sería una útil mejora a lo implementado.

Este proyecto sienta las bases para futuras integraciones de medios de pago dentro del ecosistema Moqui, simplificando y acelerando el desarrollo de soluciones relacionadas a pagos electrónicos. En particular, se dejó una plantilla replicable para futuras integraciones de pasarelas de pago.

Bibliografía

- [1] Coarchy. Stripe integration for moqui (github). <https://github.com/coarchy/stripe>, 2025. Último acceso: 14 de Junio de 2025.
- [2] MercadoPago Developers. Api reference - mercadopago developers. <https://www.mercadopago.cl/developers/es/reference>, 2025. Último acceso: 14 de Junio de 2025.
- [3] Transbank Developers. Webpay plus - documentación oficial. <https://www.transbankdevelopers.cl/documentacion/webpay-plus>, 2025. Último acceso: 14 de Junio de 2025.
- [4] GrowERP. Moqui paypal integration (github). <https://github.com/growerp/moqui-paypal>, 2025. Último acceso: 14 de Junio de 2025.
- [5] Ngrok. Ngrok - public urls for localhost. <https://ngrok.com/>, 2025. Último acceso: 14 de Junio de 2025.
- [6] PayPal. Paypal developer portal. <https://developer.paypal.com>, 2025. Último acceso: 14 de Junio de 2025.
- [7] Moqui Project. Moqui framework - documentación oficial. <https://www.moqui.org/m/docs/framework/>, 2025. Último acceso: 14 de Junio de 2025.
- [8] Moit SpA. Sitio oficial de moit. <https://moit.cl>, 2025. Último acceso: 14 de Junio de 2025.

ANEXOS

ANEXO A

Configuraciones Mercado Pago

Crear aplicación

Completa los siguientes datos y obtén tus credenciales para integrarte con Mercado Pago.

¿Dudas para comenzar a integrar? [Conocer primeros pasos](#)

Configuraciones básicas

Nombre de la aplicación

PROD TESTING ✓

Elige un nombre para identificar tus aplicaciones con más facilidad. 12 / 50

¿Qué tipo de solución de pago vas a integrar?

☒ Pagos online

☐ Pagos presencial

¿Estás usando una plataforma de e-commerce?

☐ Sí

☒ No


¿Qué producto estás integrando? [Conocer sobre los productos](#)

CheckoutPro

Modelo de integración: Opcional

Selecciona una opción

☒ Autorizo el uso de mis datos personales conforme a la [Declaración de Privacidad](#) y certifico que mi cuenta usa las herramientas de Mercado Pago de acuerdo a los [Términos y Condiciones](#).

☒ No soy un robot  reCAPTCHA
Privacidad * Términos

Crear aplicación

Figura A.1: Configuración de la integración con Mercado Pago

Credenciales de producción

Las credenciales de producción son un conjunto de llaves que sirven para recibir pagos reales en tiendas online y otras aplicaciones. [Accede a la documentación](#) para conocer más acerca de las credenciales.

Public Key

APP_USR-0c9c1a7a-9bab-4959-9b41-150229079318

Access Token


.....

Client ID

4323591994202905

Client Secret

.....

 Comparte las credenciales con un desarrollador

Si alguien te está ayudando a integrar los productos de Mercado Pago, puedes compartir las credenciales de tu aplicación con esa persona de forma segura. Puedes deshacer esa acción eliminando la cuenta que recibió la persona a la que le compartiste y renovando las credenciales.

[Compartir credenciales](#)

Figura A.2: Credenciales de Mercado Pago

[← Volver a Notificaciones](#)

Configurar notificaciones Webhooks

Define los eventos y la URL a la que se enviarán. [Saber más](#)

Modo de prueba

Modo productivo

URL de producción

https://2d13-191-113-148-173.ngrok-free.app/rest/sm/MercadoPagoWebhook/MercadoPago

Eventos recomendados para integraciones con CheckoutPro

☒ Pagos

☐ Vinculación de aplicaciones

☐ Alertas de fraude

☐ Reclamos

☐ Contracargos

☐ Órdenes comerciales

☐ Envíos (Mercado Pago)

Otros eventos

☐ Planes y suscripciones

☐ Integraciones Point

☐ Delivery (proximity marketplace)

☐ Wallet Connect

☐ Card Updater

☐ Order (Mercado Pago)

Clave secreta ⓘ

.....

🗖

🔄

Guardar configuración

Simular notificación

🔄 Restablecer

Figura A.3: Configuración de webhook de Mercado Pago

ANEXO B

Código de la implementación independiente de Mantle

B.1. MoquiConf.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <moqui-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation=
4     "http://moqui.org/xsd/moqui-conf-3.xsd">
5
6     <default-property name="mercado_pago_access_token"
7         value="TEST-example"/>
8     <default-property name="mercado_pago_public_key"
9         value="TEST-example"/>
10    <default-property name="mercado_pago_currency" value="USD"/>
11    <default-property name="mercado_pago_success_url" value=
12    "https://localhost:8080/moit-payments/payments/success"/>
13    <default-property name=
14    "mercado_pago_pending_url" value=
15    "https://localhost:8080/moit-payments/payments/pending"/>
16    <default-property name=
17    "mercado_pago_failure_url" value=
18    "https://localhost:8080/moit-payments/payments/failure"/>
19 </moqui-conf>
```

B.2. entity/MoitMPPaymentEntities.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <entities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation=
```

```

4      "http://moqui.org/xsd/entity-definition-3.xsd">
5
6
7      <entity entity-name="MercadoPagoCustomer"
8          package-name="moit.payments" has-stamp="true"
9          enable-audit-log="true">
10
11          <field name="partyId" type="id" is-pk="true"/>
12          <field name="mpCustomerId" type="id" required="true"/>
13
14          <relationship type="many"
15              related="moit.payments.MercadoPagoCheckout"
16              short-alias="checkoutCustomer">
17          <key-map field-name="partyId"/>
18          </relationship>
19
20          <relationship type="many"
21              related="moit.payments.MercadoPagoSubscription"
22              short-alias="subscriptionCustomer">
23          <key-map field-name="partyId"/>
24          </relationship>
25
26      </entity>
27
28      <entity entity-name="MercadoPagoCheckout"
29          package-name="moit.payments" has-stamp="true"
30          enable-audit-log="true">
31
32          <field name="internalCheckoutId" type="id" is-pk="true"/>
33          <field name="mpPreferenceId" type="text-long"
34              description="Preference ID from Mercado Pago"/>
35          <field name="partyId" type="id" required="true"/>
36          <field name="externalReference" type="text-short"
37              description="Reference to order, invoice, etc."/>
38          <field name="amount" type="currency-amount"/>
39          <field name="currency" type="text-short"/>
40          <field name="status" type="text-short"
41              description="pending, expired, completed"/>
42      </entity>
43
44      <entity entity-name="MercadoPagoPayment"
45          package-name="moit.payments" has-stamp="true"
46          enable-audit-log="true">
47
48          <field name="mpPaymentId" type="id" is-pk="true"/>
49          <field name="internalCheckoutId" type="id"/>
50          <field name="paymentStatus" type="text-short"
51              description="pending, approved, rejected, etc."/>
52          <field name="transactionAmount" type="currency-amount"/>
53          <field name="currency" type="text-short"/>
54
55          <relationship type="one"
56              related="moit.payments.MercadoPagoCheckout"

```

```

39         short-alias="paymentCheckout">
40         <key-map field-name="internalCheckoutId"/>
41     </relationship>
42 </entity>
43
44 <entity entity-name="MercadoPagoPlan"
45     package-name="moit.payments" has-stamp="true"
46     enable-audit-log="true">
47     <field name="mpPlanId" type="id" is-pk="true"/>
48     <field name="reason" type="text-short" required="true"/>
49     <field name="frequency" type="number-integer"
50         required="true"/>
51     <field name="frequencyType" type="text-short"
52         required="true"/>
53     <field name="transactionAmount" type="currency-amount"
54         required="true"/>
55     <field name="currency" type="text-short" required="true"/>
56     <field name="repetitions" type="number-integer"/>
57     <field name="billingDay" type="date-time"/>
58     <field name="freeTrial" type="text-indicator"/>
59
60     <relationship type="many"
61         related="moit.payments.MercadoPagoSubscription"
62         short-alias="planSubscriptions">
63         <key-map field-name="mpPlanId"/>
64     </relationship>
65 </entity>
66
67 <entity entity-name="MercadoPagoSubscription"
68     package-name="moit.payments" has-stamp="true"
69     enable-audit-log="true">
70     <field name="subscriptionId" type="id" is-pk="true"/>
71     <field name="partyId" type="id" required="true"/>
72     <field name="mpSubscriptionId" type="id" required="true"/>
73     <field name="mpPlanId" type="id"/>
74     <field name="reason" type="text-short"/>
75     <field name="externalReference" type="text-short"/>
76     <field name="recurringFrequency" type="number-integer"/>
77     <field name="recurringFrequencyType" type="text-short"/>
78     <field name="recurringTransactionAmount"
79         type="currency-amount"/>
80     <field name="recurringCurrency" type="text-short"/>
81     <field name="recurringStartDate" type="date-time"/>
82     <field name="recurringEndDate" type="date-time"/>
83     <field name="status" type="text-short"
84         description="authorized, paused, cancelled, etc."/>
85     <field name="isActive" type="text-indicator" default="Y"/>
86     <field name="cancelReason" type="text-short"/>
87 </entity>

```

77 </entities>

B.3. data/MoitPaymentSetup.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-facade-xml type="seed">
3
4     <!-- ===== System Message Types (for Webhook Handling)
5         ===== -->
6     <moqui.basic Enumeration enumId="SmatMercadoPago"
7         description="Mercado Pago HMAC SHA-256"
8         enumTypeId="SystemMessageAuthType"/>
9
10    <!-- ===== System Message Types (for Webhook Handling)
11        ===== -->
12    <moqui.service.message.SystemMessageType
13        systemMessageTypeId="MercadoPagoWebhook"
14        consumeServiceName="moit.payments.MercadoPagoServices
15            .consume#MercadoPagoWebhookEvent"
16        description="MercadoPago Webhook Events"
17        contentType="application/json"/>
18
19    <moqui.service.message.SystemMessageRemote
20        systemMessageRemoteId="MercadoPago"
21        description="MercadoPago Payment System"
22        messageAuthEnumId="SmatMercadoPago"
23        sharedSecret="SharedSecret"
24        authHeaderName="x-signature"/>
25 </entity-facade-xml>
```

B.4. service/moit/payments/MercadoPagoServices.xml

B.4.1. getOrCreate#MercadoPagoCustomer Service

```
1 <service verb="getOrCreate" noun="MercadoPagoCustomer">
2 <description>Get or create a Mercado Pago customer, linked to a
3     partyId</description>
4 <in-parameters>
5     <parameter name="partyId" required="true"/>
6     <parameter name="payerEmail" required="true"/>
7     <parameter name="payerName" required="true"/>
8     <parameter name="payerSurname" required="true"/>
```

```

8 </in-parameters>
9 <out-parameters>
10     <parameter name="mpCustomerId"/>
11     <parameter name="isNew"/>
12 </out-parameters>
13 <actions>
14     <!-- Get access token -->
15     <set field="accessToken" from=
16         "org.moqui.util.SystemBinding
17         .getPropOrEnv('mercado_pago_access_token')"/>
18     <if condition="!accessToken">
19         <return error="true" type="danger" message="No MercadoPago
20             access token found in configuration"/>
21     </if>
22     <if condition="!partyId || !payerEmail || !payerName ||
23         !payerSurname">
24         <return error="true" type="danger" message="Missing required
25             input parameter(s)"/>
26     </if>
27     <!-- 1. Check if customer already exists locally -->
28     <entity-find-one
29         entity-name="moit.payments.MercadoPagoCustomer"
30         value-field="existingCustomer">
31         <econdition field-name="partyId" from="partyId"/>
32     </entity-find-one>
33     <if condition="existingCustomer != null">
34         <set field="mpCustomerId" from="existingCustomer.mpCustomerId"/>
35         <set field="isNew" value="false"/>
36         <return/>
37     </if>
38     <!-- 2. If not found, create via MercadoPago API -->
39     <script language="groovy"><![CDATA[
40         import com.mercadopago.MercadoPagoConfig
41         import com.mercadopago.client.customer.CustomerClient
42         import com.mercadopago.client.customer.CustomerRequest
43         import com.mercadopago.net.MPSearchRequest
44         import com.mercadopago.exceptions.MPApiException
45         import groovy.json.JsonSlurper
46         import org.slf4j.LoggerFactory
47
48         def logger = LoggerFactory
49             .getLogger("moit.payments.services
50             .getOrCreate#MercadoPagoCustomer")
51         MercadoPagoConfig.setAccessToken(accessToken)
52         def custClient = new CustomerClient()
53         def resultCustomer = null
54         def isNew = false

```

```

53
54 // 1. Try to find existing customer by email (PLEASE NOTE:
    EMAIL IS UNIQUE, SO NO USERS CAN HAVE THE SAME EMAIL OTHER
    WISE THIS WILL FAIL)
55 try {
56   def searchReq = MPSearchRequest.builder()
57     .filters([email: payerEmail])
58     .limit(10)
59     .offset(0)
60     .build()
61   def searchResp = custClient.search(searchReq)
62   resultCustomer = searchResp.results?.find { it.email ==
    payerEmail }
63   if (resultCustomer) {
64     logger.info("Found existing MercadoPago customer:
        ${resultCustomer.id}")
65     isNew = false
66   }
67   } catch (MPApiException e) {
68     System.out.println("Search failed, continuing to create:
        ${e.message}")
69     logger.warn("Search failed, continuing to create:
        ${e.message}", e)
70   }
71
72 // 2. Create only if not found
73 if (!resultCustomer) {
74   try {
75     resultCustomer = custClient.create(
76       CustomerRequest.builder()
77         .email(payerEmail)
78         .firstName(payerName)
79         .lastName(payerSurname)
80         .build()
81     )
82     isNew = true
83     logger.info("Created new MercadoPago customer:
        ${resultCustomer.id}")
84   } catch (MPApiException e) {
85     System.out.println("Error creating MercadoPago customer:
        ${e.message}")
86     def resp = e.apiResponse
87     def status = resp?.status
88     def content = resp?.getContent()
89     def json = new JsonSlurper().parseText(content ?: '{}')
90     def desc = json.cause?.getAt(0)?.description
91
92     logger.error("MP API error (status: ${status}):
        ${content}", e)

```



```

93         ec.context.errorMessage = "MP Error ${status}: ${desc
           ?: content}"
94         return
95     }
96 }
97 ec.context.mpCustomerId = resultCustomer?.id
98 ec.context.isNew = isNew
99
100 // Create and set only the two required fields
101 def ev =
102     ec.entity.makeValue("moit.payments.MercadoPagoCustomer")
103     ev.set("partyId", partyId)
104     ev.set("mpCustomerId", ec.context.mpCustomerId)
105
106 // Persist to DB
107 ev.create()
108
109 ]]></script>
110 <return/>
111 </actions>
112 </service>

```

B.4.2. create#MercadoPagoCheckout Service

```

1 <!-- Create Checkout -->
2 <service verb="create" noun="MercadoPagoCheckout">
3     <description>Create Mercado Pago checkout and store
4         it</description>
5     <in-parameters>
6         <parameter name="payingItems" type="List" required="true"/>
7         <parameter name="payerEmail"/>
8         <parameter name="payerName"/>
9         <parameter name="payerSurname"/>
10        <parameter name="externalReference" required="true"/>
11        <parameter name="partyId" required="true"/>
12    </in-parameters>
13    <out-parameters>
14        <parameter name="success"/>
15        <parameter name="checkoutUrl"/>
16    </out-parameters>
17    <actions>
18
19        <!-- Config -->
20        <set field="accessToken" from=
21            "org.moqui.util.SystemBinding
22            .getPropOrEnv('mercado_pago_access_token')"/>
23        <if condition="!accessToken">
24            <set field="success" value="false"/>

```

```

24         <return error="true" message="No MercadoPago access token
           configured"/>
25     </if>
26     <set field="mercadoPagoCurrency" from=
27     "org.moqui.util.SystemBinding
28     .getPropOrEnv('mercado_pago_currency')"/>
29     <if condition="!mercadoPagoCurrency">
30         <set field="success" value="false"/>
31         <return error="true" message="No MercadoPago currency
           configured"/>
32     </if>
33     <set field="mercadoPagoSuccessUrl" from=
34     "org.moqui.util.SystemBinding
35     .getPropOrEnv('mercado_pago_success_url')"/>
36     <if condition="!mercadoPagoSuccessUrl">
37         <set field="success" value="false"/>
38         <return error="true" message="No MercadoPago success URL
           configured"/>
39     </if>
40     <set field="mercadoPagoPendingUrl" from=
41     "org.moqui.util.SystemBinding
42     .getPropOrEnv('mercado_pago_pending_url')"/>
43     <if condition="!mercadoPagoPendingUrl">
44         <set field="success" value="false"/>
45         <return error="true" message="No MercadoPago pending URL
           configured"/>
46     </if>
47     <set field="mercadoPagoFailureUrl" from=
48     "org.moqui.util.SystemBinding
49     .getPropOrEnv('mercado_pago_failure_url')"/>
50     <if condition="!mercadoPagoFailureUrl">
51         <set field="success" value="false"/>
52         <return error="true" message="No MercadoPago failure URL
           configured"/>
53     </if>
54     <if condition="!partyId || !externalReference || !payingItems">
55         <return error="true" type="danger" message="Missing
           required input parameter(s)"/>
56     </if>
57
58     <script language="groovy"><![CDATA[
59         import com.mercadopago.MercadoPagoConfig
60         import com.mercadopago.client.preference.*
61         import com.mercadopago.exceptions.MPEException
62         import org.slf4j.LoggerFactory
63
64         def logger = LoggerFactory
65         .getLogger("moit.payments
66         .create#MercadoPagoCheckout")
67         // EXAMPLE OF PAYING ITEMS

```

```

68     /*def itemList = [
69         [id: "item1", title: "Test Item 1", unitPrice:
70             "15.00", quantity: 1],
71         [id: "item2", title: "Test Item 2", unitPrice:
72             "20.00", quantity: 2]
73     ]
74 */
75 def itemList = payingItems
76 if (!itemList || !(itemList instanceof List) ||
77     itemList.isEmpty()) {
78     logger.error("Missing or invalid 'payingItems'")
79     ec.message.addError("Parameter 'payingItems' must be a
80         non-empty List")
81     ec.context.success = false
82     return
83 }
84
85 MercadoPagoConfig.setAccessToken(accessToken)
86
87 def totalAmount = BigDecimal.ZERO
88 logger.info("Creating checkout for items: ${itemList}")
89 def itemsReq = itemList.collect {
90     def unitPrice = (it.unitPrice as BigDecimal)
91     def quantity = (it.quantity as Integer)
92     totalAmount += unitPrice * quantity
93     PreferenceItemRequest.builder()
94         .id(it.id as String)
95         .title(it.title as String)
96         .quantity(quantity)
97         .currencyId(mercadoPagoCurrency)
98         .unitPrice(unitPrice)
99         .build()
100 }
101
102 def backUrls = PreferenceBackUrlsRequest.builder()
103     .success(mercadoPagoSuccessUrl)
104     .pending(mercadoPagoPendingUrl)
105     .failure(mercadoPagoFailureUrl)
106     .build()
107
108 def v =
109     ec.entity.makeValue("moit.payments.MercadoPagoCheckout")
110 v.partyId = partyId
111 v.externalReference = externalReference
112 v.amount = totalAmount
113 v.currency = mercadoPagoCurrency
114 v.status = "init"
115 v.internalCheckoutId = UUID.randomUUID().toString()
116 v.create()

```

```

113     def internalCheckoutId = v.internalCheckoutId.toString()
114
115
116     // Here we can also add
117     notificationUrl("https://your-webhook-url.com")
118     // For now, we will use global webhook in Developer
119     Dashboard
120
121     def prefBuilder = PreferenceRequest.builder()
122     .externalReference(externalReference)
123     .items(itemsReq)
124     .backUrls(backUrls)
125     .autoReturn("approved")
126     .metadata([
127         "checkout_id": internalCheckoutId
128     ])
129
130     def email = payerEmail?.trim()
131     def name = payerName?.trim()
132     def surname = payerSurname?.trim()
133
134     if (email && name && surname) {
135         // Build a payer object with email/name/surname
136         prefBuilder.payer(
137             PreferencePayerRequest.builder()
138             .email(email)
139             .name(name)
140             .surname(surname)
141             .build()
142         )
143         logger.info("Using payer info: ${name} ${surname},
144             ${email}")
145     }
146
147     if (!(email && name && surname)) {
148         logger.warn("Incomplete payer info; proceeding without
149             payer details")
150     }
151
152     def pref = new
153         PreferenceClient().create(prefBuilder.build())
154     def url = pref.getInitPoint()
155
156     v.mpPreferenceId = pref.getId()
157     v.status = "init"
158     v.update()
159     ec.context.checkoutUrl = url
160     ec.context.checkoutId = internalCheckoutId
161     ec.context.success = true

```

```

158     ]]> </script>
159     </actions>
160 </service>

```

B.4.3. process#MercadoPaymentEvent Service

```

1 <service verb="process" noun="MercadoPagoPaymentEvent">
2 <in-parameters>
3   <parameter name="mpPaymentId" required="true"/>
4   <parameter name="internalCheckoutId"/>
5   <parameter name="paymentStatus" required="true"/>
6   <parameter name="transactionAmount" required="true"/>
7   <parameter name="currency" required="true"/>
8 </in-parameters>
9 <actions>
10  <!-- Update or create payment record -->
11  <entity-find-one entity-name="moit.payments.MercadoPagoPayment"
12    value-field="paymentEntity">
13    <field-map field-name="mpPaymentId" from="mpPaymentId"/>
14  </entity-find-one>
15  <if condition="!paymentEntity">
16    <entity-make-value
17      entity-name="moit.payments.MercadoPagoPayment"
18      value-field="newPayment"/>
19    <set field="newPayment.mpPaymentId" from="mpPaymentId"/>
20    <set field="newPayment.internalCheckoutId"
21      from="internalCheckoutId"/>
22    <set field="newPayment.paymentStatus" from="paymentStatus"/>
23    <set field="newPayment.transactionAmount"
24      from="transactionAmount"/>
25    <set field="newPayment.currency" from="currency"/>
26    <entity-create value-field="newPayment"/>
27  </if>
28  <else>
29    <set field="paymentEntity.paymentStatus" from="paymentStatus"/>
30    <set field="paymentEntity.transactionAmount"
31      from="transactionAmount"/>
32    <set field="paymentEntity.currency" from="currency"/>
33    <entity-update value-field="paymentEntity"/>
34  </else>
35  <!-- Update checkout status -->
36  <entity-find-one
37    entity-name="moit.payments.MercadoPagoCheckout"
38    value-field="checkoutEntity">
39    <field-map field-name="internalCheckoutId"
40      from="internalCheckoutId"/>
41  </entity-find-one>
42  <if condition="checkoutEntity">

```

```

34     <if condition="checkoutEntity.status != paymentStatus">
35         <set field="checkoutEntity.status" from="paymentStatus"/>
36         <entity-update value-field="checkoutEntity"/>
37         <log message="Checkout ${internalCheckoutId} status updated
           to ${paymentStatus}"/>
38     </if>
39 </if>
40 </actions>
41 </service>

```

B.4.4. consume#MercadoPagoWebhookEvent Service

```

1 <!-- ===== Webhook Receiver ===== -->
2 <service verb="consume" noun="MercadoPagoWebhookEvent"
   authenticate="anonymous-all">
3     <description>Handle MercadoPago webhook
       notifications</description>
4     <in-parameters>
5         <parameter name="systemMessageId" required="true"/>
6     </in-parameters>
7     <actions>
8         <set field="accessToken" from="org.moqui.util.SystemBinding
9           .getPropOrEnv('mercado_pago_access_token')"/>
10        <if condition="!accessToken">
11            <return error="true" message="No MercadoPago access
12              token configured"/>
13        </if>
14        <!-- Get the SystemMessage record to access the payload -->
15        <entity-find-one
16            entity-name="moqui.service.message.SystemMessage"
17            value-field="systemMessage">
18            <field-map field-name="systemMessageId"
19              from="systemMessageId"/>
20        </entity-find-one>
21
22        <!-- Extract the JSON payload -->
23        <set field="webhookPayload"
24            from="systemMessage.messageText"/>
25        <log message="Received MercadoPago Webhook with payload:
26          ${webhookPayload}"/>
27
28        <!-- Parse JSON if needed -->
29        <script language="groovy"><![CDATA[
30            import groovy.json.JsonSlurper
31            import org.slf4j.LoggerFactory
32            import com.mercadopago.client.payment.PaymentClient
33            import com.mercadopago.MercadoPagoConfig
34            import com.mercadopago.exceptions.MPApiException

```

```

29
30     def logger = LoggerFactory
31         .getLogger("moit.payments
32         .consume#MercadoPagoWebhookEvent")
33     def jsonSlurper = new JsonSlurper()
34     def payloadData = jsonSlurper.parseText(webhookPayload)
35
36     // Access specific fields from the webhook
37     def eventType = payloadData.type
38     def event = payloadData.action
39
40     logger.info("Webhook event type: ${eventType}")
41
42     MercadoPagoConfig.setAccessToken(accessToken)
43
44     if (eventType.equals("payment")) {
45     def params = null
46     try {
47         def paymentId = payloadData.data?.id as Long
48         def payment = new PaymentClient().get(paymentId)
49         logger.info("Payment metadata:
50             ${payment.getMetadata()}")
51         params = [
52             mpPaymentId: paymentId,
53             internalCheckoutId:
54                 payment.getMetadata().get("checkout_id"),
55             paymentStatus: payment.getStatus(),
56             transactionAmount: payment.getTransactionAmount(),
57             currency: payment.getCurrencyId()
58         ]
59     } catch (MPApiException e) {
60         def apiResp = e.getApiResponse()
61         logger.error("MercadoPago API error (status
62             ${apiResp.getStatusCode()}):
63             ${apiResp.getContent()}", e)
64         return
65     } catch (Exception e) {
66         logger.error("Unexpected error fetching payment
67             ${paymentId}: ${e.message}", e)
68         return
69     }
70
71     logger.info("Params: ${params}")
72     ec.service.sync().name("moit.payments
73     .MercadoPagoServices.process#MercadoPagoPaymentEvent")
74         .parameters(params).call()
75     return
76 }
77 else {
78     logger.warn("Unsupported webhook type: ${eventType}")

```

```
74         return
75     }
76     ]]> </script>
77     </actions>
78 </service>
```

ANEXO C

Código de la implementación integrada a Mantle

C.1. MoquiConf.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <moqui-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation=
4     "http://moqui.org/xsd/moqui-conf-3.xsd">
5
6     <default-property name="mercado_pago_access_token"
7         value="accessToken"/>
8     <default-property name="mercado_pago_success_url" value=
9         "https://localhost:8080/moit-payments/payments/success"/>
10    <default-property name="mercado_pago_pending_url" value=
11        "https://localhost:8080/moit-payments/payments/pending"/>
12    <default-property name="mercado_pago_failure_url" value=
13        "https://localhost:8080/moit-payments/payments/failure"/>
14 </moqui-conf>
```

C.2. entity/CheckoutEntityExt.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <entities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation=
4     "http://moqui.org/xsd/entity-definition-3.xsd">
5
6     <!-- ===== Payment Gateway Response ===== -->
7
```

```

8      <extend-entity entity-name="PaymentGatewayResponse"
      package="mantle.account.method">
9          <field name="checkoutUrl" type="text-long"
            enable-audit-log="update"
10             description="Checkout URL from payment gateway"/>
11      </extend-entity>
12 </entities>

```

C.3. data/MPinitialEnumerations.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-facade-xml type="seed">
3
4     <!-- Payment enums for Mercado Pago checkout -->
5     <moqui.basic Enumeration enumTypeId="PaymentInstrument"
      enumId="PiMercadoPago" description="Mercado Pago Checkout
      Pro"/>
6     <moqui.basic Enumeration enumTypeId="PaymentGatewayType"
      enumId="PgtMercadoPago" description="Mercado Pago"/>
7     <moqui.basic Enumeration enumTypeId="PaymentMethodType"
      enumId="PmtMercadoPagoAccount" description="Mercado Pago
      Account"/>
8
9 </entity-facade-xml>

```

C.4. data/MPinitialGatewayData.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-facade-xml type="seed">
3
4     <mantle.product.store.ProductStore productStoreId="DemoStore"/>
5
6     <mantle.account.method.PaymentGatewayConfig
      paymentGatewayConfigId="MercadoPagoCheckoutPro"
7       paymentGatewayTypeEnumId="PgtMercadoPago"
8       description="Mercado Pago Checkout Pro"
9       authorizeServiceName=
10        "moit.payments.MercadoPagoServices.authorize#MercadoPago"
11       captureServiceName=
12        "moit.payments.MercadoPagoServices.capture#MercadoPago"/>
13
14
15     <mantle.product.store.ProductStorePaymentGateway
      productStoreId="DemoStore"

```

```

16         paymentInstrumentEnumId="PiMercadoPago"
17         paymentGatewayConfigId="MercadoPagoCheckoutPro"/>
18
19 </entity-facade-xml>

```

C.5. data/MPsetupWebhook.xml

C.6. service/moit/payments/MercadoPagoServices.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-facade-xml type="seed">
3
4     <!-- ===== System Message Types (for Webhook Handling)
5         ===== -->
6     <moqui.basic Enumeration enumId="SmatMercadoPago"
7         description="Mercado Pago HMAC SHA-256"
8         enumTypeId="SystemMessageAuthType"/>
9
10    <!-- ===== System Message Types (for Webhook Handling)
11        ===== -->
12    <moqui.service.message.SystemMessageType
13        systemMessageTypeId="MercadoPagoWebhook"
14        consumeServiceName=
15            "moit.payments.MercadoPagoServices
16            .consume#MercadoPagoWebhookEvent"
17        description="MercadoPago Webhook Events"
18        contentType="application/json"/>
19
20    <moqui.service.message.SystemMessageRemote
21        systemMessageRemoteId="MercadoPago"
22        description="MercadoPago Payment System"
23        messageAuthEnumId="SmatMercadoPago"
24        sharedSecret="sharedSecret"
25        authHeaderName="x-signature"/>
26
27 </entity-facade-xml>

```

C.7. service/moit/payments/MercadoPagoCore.xml

C.7.1. create#MercadoPagoCheckout Service

```
1 <!-- Create Checkout -->
2 <service verb="create" noun="MercadoPagoCheckout">
3     <description>Create Mercado Pago checkout and store
        it</description>
4     <in-parameters>
5         <parameter name="payingItems" type="List" required="true"/>
6         <parameter name="payerEmail"/>
7         <parameter name="payerName"/>
8         <parameter name="payerSurname"/>
9         <parameter name="externalReference" required="true"/>
10        <parameter name="currency" required="true"/>
11    </in-parameters>
12    <out-parameters>
13        <parameter name="success"/>
14        <parameter name="checkoutUrl"/>
15        <parameter name="mpPreferenceId"/>
16    </out-parameters>
17    <actions>
18        <set field="accessToken" from=
19            "org.moqui.util.SystemBinding
20            .getPropOrEnv('mercado_pago_access_token')"/>
21        <if condition="!accessToken">
22            <return error="true" message="No MercadoPago access token
                configured"/>
23        </if>
24        <set field="mercadoPagoSuccessUrl" from=
25            "org.moqui.util.SystemBinding
26            .getPropOrEnv('mercado_pago_success_url')"/>
27        <if condition="!mercadoPagoSuccessUrl">
28            <set field="success" value="false"/>
29            <return error="true" message="No MercadoPago success URL
                configured"/>
30        </if>
31        <set field="mercadoPagoPendingUrl" from=
32            "org.moqui.util.SystemBinding
33            .getPropOrEnv('mercado_pago_pending_url')"/>
34        <if condition="!mercadoPagoPendingUrl">
35            <set field="success" value="false"/>
36            <return error="true" message="No MercadoPago pending URL
                configured"/>
37        </if>
38        <set field="mercadoPagoFailureUrl" from=
39            "org.moqui.util.SystemBinding
40            .getPropOrEnv('mercado_pago_failure_url')"/>
41        <if condition="!mercadoPagoFailureUrl">
```

```

42         <set field="success" value="false"/>
43         <return error="true" message="No MercadoPago failure URL
           configured"/>
44     </if>
45     <if condition="!accessToken || !currency || !externalReference
           || !payingItems">
46         <return error="true" type="danger" message="Missing
           required input parameter(s)"/>
47     </if>
48
49     <script language="groovy"><![CDATA[
50         import com.mercadopago.MercadoPagoConfig
51         import com.mercadopago.client.preference.*
52         import com.mercadopago.exceptions.MPApiException
53         import org.slf4j.LoggerFactory
54
55         def logger = LoggerFactory.getLogger("moit.payments
56             .create#MercadoPagoCheckout")
57         // EXAMPLE OF PAYING ITEMS
58         /*def itemsList = [
59             [id: "item1", title: "Test Item 1", unitPrice:
60                 "15.00", quantity: 1],
61             [id: "item2", title: "Test Item 2", unitPrice:
62                 "20.00", quantity: 2]
63         ]
64         */
65         try {
66             def itemsList = payingItems
67             if (!itemsList || !(itemsList instanceof List) ||
68                 itemsList.isEmpty()) {
69                 logger.error("Missing or invalid 'payingItems'")
70                 ec.message.addError("Parameter 'payingItems' must be a
71                     non-empty List")
72                 ec.context.success = false
73                 return
74             }
75
76             MercadoPagoConfig.setAccessToken(accessToken)
77
78             def totalAmount = BigDecimal.ZERO
79             logger.info("Creating checkout for items: ${itemsList}")
80             def itemsReq = itemsList.collect {
81                 def unitPrice = (it.unitPrice as BigDecimal)
82                 def quantity = (it.quantity as Integer)
83                 totalAmount += unitPrice * quantity
84                 PreferenceItemRequest.builder()
85                     .id(it.id as String)
86                     .title(it.title as String)
87                     .quantity(quantity)
88                     .currencyId(currency)

```

```

85         .unitPrice(unitPrice)
86         .build()
87     }
88
89     def backUrls = PreferenceBackUrlsRequest.builder()
90         .success(mercadoPagoSuccessUrl)
91         .pending(mercadoPagoPendingUrl)
92         .failure(mercadoPagoFailureUrl)
93         .build()
94
95
96
97     // Here we can also add
98     notificationUrl("https://your-webhook-url.com")
99     // For now, we will use global webhook in Developer
100    Dashboard
101
102    def prefBuilder = PreferenceRequest.builder()
103        .externalReference(externalReference)
104        .items(itemsReq)
105        .backUrls(backUrls)
106        .autoReturn("approved")
107
108
109
110    def email = payerEmail?.trim()
111    def name = payerName?.trim()
112    def surname = payerSurname?.trim()
113
114
115
116    if (email && name && surname) {
117        // Build a payer object with email/name/surname
118        prefBuilder.payer(
119            PreferencePayerRequest.builder()
120                .email(email)
121                .name(name)
122                .surname(surname)
123                .build()
124        )
125        logger.info("Using payer info: ${name} ${surname},
126                    ${email}")
127    }
128
129
130    if (!(email && name && surname)) {
131        logger.warn("Incomplete payer info; proceeding without
132                    payer details")
133    }
134
135
136
137    def pref = new
138        PreferenceClient().create(prefBuilder.build())
139    def url = pref.getInitPoint()
140
141
142

```

```

130     ec.context.checkoutUrl = url
131     ec.context.mpPreferenceId = pref.getId()
132     ec.context.success = true
133     logger.info("Created checkout (MP pref: ${pref.getId()}) -
        URL: ${url}")
134     } catch (MPApiException e) {
135     logger.error("MercadoPago API error: ${e.message}", e)
136     ec.context.success = false
137     return
138     } catch (Exception e) {
139     logger.error("Error creating checkout: ${e.message}", e)
140     ec.context.success = false
141     return
142     }
143     ]]></script>
144     </actions>
145 </service>

```

C.7.2. process#MercadoPagoPaymentEvent Service

```

1 <service verb="process" noun="MercadoPagoPaymentEvent">
2 <in-parameters>
3     <parameter name="mpPaymentId" required="true"/>
4     <parameter name="externalReference" required="true"/>
5     <parameter name="paymentStatus" required="true"/>
6     <parameter name="transactionAmount" required="true"/>
7     <parameter name="currency" required="true"/>
8 </in-parameters>
9 <actions>
10     <log level="info" message="Processing Mercado Pago payment
        event: MP Payment ID ${mpPaymentId}, Status:
        ${paymentStatus}"/>
11
12     <!-- Find the Mantle Payment using external_reference (which
        should be the paymentId) -->
13     <entity-find-one entity-name="mantle.account.payment.Payment"
        value-field="payment">
14     <field-map field-name="paymentId" from="externalReference"/>
15     </entity-find-one>
16
17     <if condition="!payment">
18     <log level="error" message="No Mantle Payment found for
        external reference: ${externalReference}"/>
19     <return error="true" message="Payment not found for external
        reference: ${externalReference}"/>
20     </if>
21

```

```

22     <log level="info" message="Found Mantle Payment
        ${payment.paymentId} with status ${payment.statusId}"/>
23
24     <!-- Map Mercado Pago status to Mantle status -->
25     <if condition="paymentStatus == 'approved'">
26     <then>
27         <set field="mantleStatus" value="PmntDelivered"/>
28         <set field="responseCode" value="success"/>
29         <set field="reasonMessage" value="Payment approved by
            Mercado Pago"/>
30         <set field="resultSuccess" value="Y"/>
31         <set field="resultDeclined" value="N"/>
32     </then>
33     <!-- this is added for better understanding of the payment
        status, since after checkout mantleStatus is already set to
        PmntAuthorized -->
34     <else-if condition="paymentStatus == 'pending'">
35     <then>
36         <set field="mantleStatus" value="PmntAuthorized"/>
37         <set field="responseCode" value="pending"/>
38         <set field="reasonMessage" value="Payment pending in
            Mercado Pago (User will be notified via email)"/>
39         <set field="resultSuccess" value="Y"/>
40         <set field="resultDeclined" value="N"/>
41     </then>
42     </else-if>
43     <else-if condition="paymentStatus == 'in_process'">
44     <then>
45         <set field="mantleStatus" value="PmntAuthorized"/>
46         <set field="responseCode" value="pending"/>
47         <set field="reasonMessage" value="Payment pending in
            Mercado Pago (User will be notified via email)"/>
48         <set field="resultSuccess" value="Y"/>
49         <set field="resultDeclined" value="N"/>
50     </then>
51     </else-if>
52     <else-if condition="paymentStatus == 'rejected'">
53     <then>
54         <set field="mantleStatus" value="PmntDeclined"/>
55         <set field="responseCode" value="declined"/>
56         <set field="reasonMessage" value="Payment rejected by
            Mercado Pago"/>
57         <set field="resultSuccess" value="N"/>
58         <set field="resultDeclined" value="Y"/>
59     </then>
60     </else-if>
61     <else-if condition="paymentStatus == 'cancelled'">
62     <then>
63         <set field="mantleStatus" value="PmntVoid"/>
64         <set field="responseCode" value="cancelled"/>

```



```

65         <set field="reasonMessage" value="Payment cancelled"/>
66         <set field="resultSuccess" value="N"/>
67         <set field="resultDeclined" value="N"/>
68     </then>
69 </else-if>
70 <else>
71     <then>
72         <log level="warn" message="Unknown Mercado Pago status:
           ${paymentStatus}, keeping current status"/>
73         <set field="mantleStatus" from="payment.statusId"/>
74         <set field="responseCode" value="unknown"/>
75         <set field="reasonMessage" value="Unknown Mercado Pago
           status: ${paymentStatus}"/>
76         <set field="resultSuccess" value="N"/>
77         <set field="resultDeclined" value="N"/>
78     </then>
79 </else>
80 </if>
81
82 <log level="info" message="Mapped MP status '${paymentStatus}'
   to Mantle status '${mantleStatus}'"/>
83
84 <!-- Create PaymentGatewayResponse for all status changes -->
85 <if condition="mantleStatus != payment.statusId">
86 <service-call
   name="create#mantle.account.method.PaymentGatewayResponse"
   out-map="gatewayResponse"
87       in-map="[paymentGatewayConfigId:payment.paymentGatewayConfigId,
88               paymentOperationEnumId: 'PgoCapture',
89               paymentId:payment.paymentId,
90               paymentMethodId:payment.paymentMethodId,
91               amount:transactionAmount,
92               amountUomId:currency,
93               referenceNum:mpPaymentId,
94               approvalCode:(paymentStatus == 'approved' ?
                             mpPaymentId : null),
95               responseCode:responseCode,
96               reasonMessage:reasonMessage,
97               transactionDate:ec.user.nowTimestamp,
98               resultSuccess:resultSuccess,
99               resultDeclined:resultDeclined,
100              resultError:'N']"/>
101 </if>
102 <!-- Handle approved payments specially -->
103 <if condition="paymentStatus == 'approved' & payment.statusId != 'PmntDelivered'">
104 <then>
105     <!-- Check if payment was previously declined and reset to
         authorized -->
106     <if condition="payment.statusId == 'PmntDeclined'">

```

```

107     <then>
108         <!-- Reset directly to Authorized -->
109         <service-call
110             name="update#mantle.account.payment.Payment"
111             in-map="[paymentId:payment.paymentId,
112                 statusId: 'PmntAuthorized']"/>
113         <!-- Refresh payment object -->
114         <entity-find-one
115             entity-name="mantle.account.payment.Payment"
116             value-field="payment">
117             <field-map field-name="paymentId"
118                 from="payment.paymentId"/>
119             </entity-find-one>
120         </then>
121     </if>
122
123     <!-- Now update to delivered -->
124     <service-call name="update#mantle.account.payment.Payment"
125         in-map="[paymentId:payment.paymentId,
126             paymentRefNum:mpPaymentId,
127             statusId: 'PmntDelivered',
128             effectiveDate:ec.user.nowTimestamp]"/>
129
130     <log level="info" message="Payment ${payment.paymentId}
131         captured successfully via webhook"/>
132 </then>
133 <else-if condition="mantleStatus != payment.statusId">
134     <then>
135         <!-- Update status for other status changes -->
136         <service-call name="update#mantle.account.payment.Payment"
137             in-map="[paymentId:payment.paymentId,
138                 statusId:mantleStatus]"/>
139
140         <log level="info" message="Payment ${payment.paymentId}
141             status updated to ${mantleStatus}"/>
142     </then>
143 </else-if>
144 <else>
145     <then>
146         <log level="info" message="No status change needed for
147             Payment ${payment.paymentId}"/>
148     </then>
149 </else>
150 </if>
151 </actions>
152 </service>

```

C.8. service/moit/payments/MercadoPagoServices.xml

Usa los servicios de C.7.1 y C.7.2

```
1
2 <!-- Include core services from separate file -->
3 <service-include verb="create" noun="MercadoPagoCheckout"
4     location="component://moit-payments/service/moit/payments
5         /MercadoPagoCore.xml"/>
6 <service-include verb="process" noun="MercadoPagoPaymentEvent"
7     location="component://moit-payments/service/moit/payments
8         /MercadoPagoCore.xml"/>
```

C.8.1. consume#MercadoPagoWebhookEvent Service

```
1 <!-- ===== Webhook Receiver ===== -->
2 <service verb="consume" noun="MercadoPagoWebhookEvent"
3     authenticate="anonymous-all">
4     <description>Handle MercadoPago webhook
5         notifications</description>
6     <in-parameters>
7         <parameter name="systemMessageId" required="true"/>
8     </in-parameters>
9     <actions>
10         <set field="accessToken" from=
11             "org.moqui.util.SystemBinding
12             .getPropOrEnv('mercado_pago_access_token')"/>
13         <if condition="!accessToken">
14             <return error="true" message="No MercadoPago access
15                 token configured"/>
16         </if>
17         <!-- Get the SystemMessage record to access the payload -->
18         <entity-find-one
19             entity-name="moqui.service.message.SystemMessage"
20             value-field="systemMessage">
21             <field-map field-name="systemMessageId"
22                 from="systemMessageId"/>
23         </entity-find-one>
24
25         <!-- Extract the JSON payload -->
26         <set field="webhookPayload"
27             from="systemMessage.messageText"/>
28         <log message="Received MercadoPago Webhook with payload:
29             ${webhookPayload}"/>
30
31         <!-- Parse JSON if needed -->
32         <script language="groovy"><![CDATA[
33             import groovy.json.JsonSlurper
```

```

26 import org.slf4j.LoggerFactory
27 import com.mercadopago.client.payment.PaymentClient
28 import com.mercadopago.MercadoPagoConfig
29 import com.mercadopago.exceptions.MPApiException
30
31 def logger = LoggerFactory
32     .getLogger("moit.payments
33     .consume#MercadoPagoWebhookEvent")
34 def jsonSlurper = new JsonSlurper()
35 def payloadData = jsonSlurper.parseText(webhookPayload)
36
37 // Access specific fields from the webhook
38 def eventType = payloadData.type
39 def event = payloadData.action
40
41 logger.info("Webhook event type: ${eventType}")
42
43 MercadoPagoConfig.setAccessToken(accessToken)
44
45 if (eventType.equals("payment")) {
46     def params = null
47     try {
48         def paymentId = payloadData.data?.id as Long
49         def payment = new PaymentClient().get(paymentId)
50         logger.info("Payment metadata:
51             ${payment.getMetadata()}")
52         params = [
53             mpPaymentId: paymentId,
54             externalReference: payment.getExternalReference(),
55             paymentStatus: payment.getStatus(),
56             transactionAmount: payment.getTransactionAmount(),
57             currency: payment.getCurrencyId()
58         ]
59     } catch (MPApiException e) {
60         def apiResp = e.getApiResponse()
61         logger.error("MercadoPago API error (status
62             ${apiResp.getStatusCode()}):
63             ${apiResp.getContent()}", e)
64         return
65     } catch (Exception e) {
66         logger.error("Unexpected error fetching payment
67             ${paymentId}: ${e.message}", e)
68         return
69     }
70
71     logger.info("Params: ${params}")
72     ec.service.sync()
73         .name("moit.payments.MercadoPagoServices
74             .process#MercadoPagoPaymentEvent")
75         .parameters(params).call()

```

```

72         return
73     }
74     else {
75         logger.warn("Unsupported webhook type: ${eventType}")
76         return
77     }
78     ]]> </script>
79 </actions>
80 </service>

```

C.8.2. authorize#MercadoPago Service

```

1 <service verb="authorize" noun="MercadoPago">
2 <implements
3     service="mantle.account.PaymentServices.authorize#Payment"/>
4 <actions>
5     <log level="info" message="Starting Mercado Pago authorization
6         for Payment ${paymentId}"/>
7
8     <!-- Get payment details -->
9     <entity-find-one entity-name="mantle.account.payment.Payment"
10         value-field="payment"/>
11     <if condition="payment == null"><return error="true"
12         message="Payment ${paymentId} not found"/></if>
13
14     <log level="info" message="Payment amount: ${payment.amount},
15         currency: ${payment.amountUomId}"/>
16
17     <!-- Get gateway configuration and access token -->
18     <entity-find-one
19         entity-name="mantle.account.method.PaymentGatewayConfig"
20         value-field="gatewayConfig">
21     <field-map field-name="paymentGatewayConfigId"/>
22     </entity-find-one>
23
24     <!-- Get order information for preference creation -->
25     <if condition="payment.orderId">
26     <entity-find-one entity-name="mantle.order.OrderHeader"
27         value-field="orderHeader">
28         <field-map field-name="orderId" from="payment.orderId"/>
29     </entity-find-one>
30     </if>
31
32     <log level="info" message="Creating Mercado Pago checkout
33         preference for payment ${paymentId}"/>
34
35     <script language="groovy"><![CDATA[
36         // Build list of maps

```

```

28   def itemList = [[
29       id: "payment_${paymentId}",
30       title: payment.orderId ? "Order Payment
31           #${payment.orderId}" : "Invoice Payment",
32       unitPrice: payment.amount,
33       quantity: 1
34   ]]
35   // Validate
36   if (!itemList || itemList.isEmpty()) {
37       ec.message.addError("List for checkout creation has no
38           items")
39       return
40   }
41   ec.context.itemList = itemList
42   ]]></script>
43
44   <!-- Call your Checkout service -->
45   <service-call name="moit.payments.MercadoPagoServices
46       .create#MercadoPagoCheckout"
47       in-map="[payingItems:itemList,
48           externalReference:paymentId,
49           currency:payment.amountUomId]"
50       out-map="checkoutResult"/>
51
52   <!-- Validate checkout creation -->
53   <if condition="!checkoutResult.success">
54       <return error="true" message="Failed to create Mercado Pago
55           checkout preference"/>
56   </if>
57
58   <!-- Extract results -->
59   <set field="preferenceId" from="checkoutResult.mpPreferenceId"/>
60   <set field="checkoutUrl" from="checkoutResult.checkoutUrl"/>
61
62   <if condition="!preferenceId || !checkoutUrl">
63       <return error="true" message="Invalid response from Mercado
64           Pago checkout service"/>
65   </if>
66
67   <log level="info" message="Mercado Pago checkout URL created:
68       ${checkoutUrl}"/>
69   <log level="info" message="Preference ID: ${preferenceId}"/>
70
71   <!-- Create PaymentGatewayResponse record for preference
72       creation -->
73   <service-call
74       name="create#mantle.account.method.PaymentGatewayResponse"
75       out-map="context"

```

```

67         in-map=[paymentGatewayConfigId:paymentGatewayConfigId,
68             paymentOperationEnumId: 'PgoAuthorize',
69             paymentId:paymentId,
70             paymentMethodId:payment.paymentMethodId,
71             amount:payment.amount,
72             amountUomId:payment.amountUomId,
73             referenceNum:preferenceId,
74             approvalCode:preferenceId,
75             responseCode:'pending', reasonMessage:'Checkout
76             preference created, redirect customer to
77             checkout URL',
78             checkoutUrl: checkoutUrl,
79             transactionDate:ec.user.nowTimestamp,
80             resultSuccess:'Y', resultDeclined:'N',
81             resultError:'N']"/>
82
83     <log level="info" message="PaymentGatewayResponse created with
84         ID: ${paymentGatewayResponseId}"/>
85     <log level="info" message="Mercado Pago authorization completed
86         for Payment ${paymentId}"/>
87 </actions>
88 </service>

```

C.8.3. capture#MercadoPago Service

```

1 <!-- ===== Capture Service ===== -->
2 <service verb="capture" noun="MercadoPago">
3     <implements
4         service="mantle.account.PaymentServices.capture#Payment"/>
5     <actions>
6         <log level="info" message="Starting Mercado Pago capture for
7             Payment ${paymentId}"/>
8
9         <!-- Get payment details -->
10        <entity-find-one entity-name="mantle.account.payment.Payment"
11            value-field="payment"/>
12        <if condition="payment == null"><return error="true"
13            message="Payment ${paymentId} not found"/></if>
14
15        <log level="info" message="Payment reference number:
16            ${payment.paymentRefNum}"/>
17
18        <!-- For hosted checkout, capture usually happens automatically
19            via webhook -->
20        <!-- This service checks if payment was already captured -->
21        <log level="info" message="Checking if Mercado Pago payment was
22            already captured via webhook"/>

```

```

17 <!-- Check latest gateway response for this payment -->
18 <entity-find
    entity-name="mantle.account.method.PaymentGatewayResponse"
    list="responseList">
19     <econdition field-name="paymentId"/>
20     <econdition field-name="paymentOperationEnumId"
        value="PgoCapture"/>
21     <order-by field-name="-transactionDate"/>
22 </entity-find>
23
24 <if condition="responseList">
25     <log level="info" message="Payment ${paymentId} already
        captured via webhook"/>
26     <set field="paymentGatewayResponseId"
        from="responseList[0].paymentGatewayResponseId"/>
27 <else>
28     <!-- No manual capture possible - return message indicating
        webhook-only -->
29     <log level="warn" message="Payment ${paymentId} not yet
        captured - waiting for webhook notification"/>
30     <return message="Mercado Pago Checkout Pro payments can
        only be captured via webhook notifications"/>
31 </else></if>
32 <log level="info" message="Mercado Pago capture completed for
        Payment ${paymentId}"/>
33 </actions>
34 </service>

```

C.8.4. refund & release#MercadoPago templates

```

1 <!-- TODO: Refund Service is not implemented yet, but might be
    necessary for future use -->
2 <!-- ===== Refund Service ===== -->
3 <service verb="refund" noun="MercadoPago">
4     <implements
        service="mantle.account.PaymentServices.refund#Payment"/>
5     <actions>
6         <log level="info" message="Starting Mercado Pago refund for
            Payment ${paymentId}, amount: ${amount}"/>
7
8         <!-- Get payment details -->
9         <entity-find-one entity-name="mantle.account.payment.Payment"
            value-field="payment"/>
10        <if condition="payment == null"><return error="true"
            message="Payment ${paymentId} not found"/></if>
11
12        <log level="info" message="Refunding MP Payment:
            ${payment.paymentRefNum}"/>

```



```

13
14 <!-- TODO: MERCADO PAGO API INTEGRATION POINT -->
15 <!-- Replace this section with actual Mercado Pago refund API
    call -->
16 <!--
17 Required Mercado Pago API call:
18 1. POST /v1/payments/{payment.paymentRefNum}/refunds
19 2. Body: { "amount": amount } (optional, full refund if not
    specified)
20 3. Extract refund.id from response
21 4. Handle partial vs full refunds
22 -->
23
24 <!-- Simulated refund creation - REPLACE WITH ACTUAL API CALL
    -->
25 <set field="refundId"
    value="MP_REFUND_${paymentId}_${ec.user.nowTimestamp.time}"/>
26
27 <log level="info" message="Mercado Pago refund created with ID:
    ${refundId}"/>
28
29 <!-- Create refund response -->
30 <service-call
    name="create#mantle.account.method.PaymentGatewayResponse"
    out-map="context"
31     in-map="[paymentGatewayConfigId:paymentGatewayConfigId,
32             paymentOperationEnumId:'PgoRefund',
33             paymentId:paymentId,
34             paymentMethodId:payment.paymentMethodId,
35             amount:amount ?: payment.amount,
36             amountUomId:payment.amountUomId,
37             referenceNum:refundId, approvalCode:refundId,
38             responseCode:'success', reasonMessage:'Refund
    processed successfully',
39             transactionDate:ec.user.nowTimestamp,
40             resultSuccess:'Y', resultDeclined:'N',
41             resultError:'N']"/>
42
43 <log level="info" message="Mercado Pago refund completed for
    Payment ${paymentId}"/>
44 </actions>
45 </service>
46
47 <!-- TODO: Release Service is not implemented yet, but might be
    necessary for future use -->
48 <service verb="release" noun="MercadoPago">
49     <implements
        service="mantle.account.PaymentServices.release#Payment"/>
50 </actions>

```

```

47     <log level="info" message="Starting Mercado Pago release (void)
        for Payment ${paymentId}"/>
48
49     <!-- Get payment details -->
50     <entity-find-one entity-name="mantle.account.payment.Payment"
        value-field="payment"/>
51     <if condition="payment == null"><return error="true"
        message="Payment ${paymentId} not found"/></if>
52
53     <log level="info" message="Releasing MP Payment:
        ${payment.paymentRefNum}"/>
54
55     <!-- Check if payment can be cancelled (only pending/authorized
        payments) -->
56     <if condition="payment.statusId not in ['PmntPromised',
        'PmntAuthorized']">
57         <log level="warn" message="Cannot release Payment
            ${paymentId} with status ${payment.statusId} - only
            pending/authorized payments can be cancelled"/>
58         <return error="true" message="Cannot release payment in
            status ${payment.statusId}. Only pending or authorized
            payments can be cancelled."/>
59     </if>
60
61     <!-- TODO: MERCADO PAGO API INTEGRATION POINT -->
62     <!-- Replace this section with actual Mercado Pago cancellation
        API call -->
63     <!--
64     Note: For hosted checkout, cancellation might not be possible
        after customer interaction
65     Required Mercado Pago API call:
66     1. PUT /v1/payments/{payment.paymentRefNum}
67     2. Body: { "status": "cancelled" }
68     3. This only works for pending payments, not approved ones
69     -->
70
71     <!-- Simulate cancellation - REPLACE WITH ACTUAL API CALL -->
72     <set field="cancelSuccess" value="true"/>
73     <set field="cancelMessage" value="Payment cancelled
        successfully"/>
74
75     <log level="info" message="Mercado Pago payment cancelled
        successfully"/>
76
77     <!-- Create release response -->
78     <service-call
        name="create#mantle.account.method.PaymentGatewayResponse"
        out-map="context"
79         in-map="[paymentGatewayConfigId:payment.paymentGatewayConfigId,
80             paymentOperationEnumId: 'PgoRelease',

```

```

81         paymentId:paymentId,
            paymentMethodId:payment.paymentMethodId,
82         amount:payment.amount,
            amountUomId:payment.amountUomId,
83         referenceNum:payment.paymentRefNum,
            approvalCode:payment.paymentRefNum,
84         responseCode:'success', reasonMessage:cancelMessage,
85         transactionDate:ec.user.nowTimestamp,
86         resultSuccess:(cancelSuccess ? 'Y' : 'N'),
87         resultDeclined:'N',
88         resultError:(cancelSuccess ? 'N' : 'Y')]"/>
89
90     <!-- Update payment status to void if cancellation was
        successful -->
91     <if condition="cancelSuccess">
92         <service-call name="update#mantle.account.payment.Payment"
93             in-map="[paymentId:paymentId,
                    statusId:'PmntVoid']"/>
94         <log level="info" message="Payment ${paymentId} status
            updated to PmntVoid"/>
95     </if>
96
97     <log level="info" message="Mercado Pago release completed for
        Payment ${paymentId}"/>
98     </actions>
99 </service>

```
