Software Architecture Document (SAD)

AllConnect Market

Plataforma Multicanal Unificada para la Gestión de Compras

Proyecto AS 2025-30

Versión:	2.0
Fecha:	23 de noviembre de 2025
Departamento:	Ingeniería de Sistemas
Facultad:	Ingeniería

Autores:

Nicolas Camacho - camachoa.nicolas@javeriana.edu.co
Sara Albarracín - saalbaracin@javeriana.edu.co
Alejandro Caicedo - caicedo_alejandro@javeriana.edu.co
Alejandro Pinzón - alejandro_pinzon@javeriana.edu.co
Pontificia Universidad Javeriana
Bogotá, Colombia

Índice

1. Objetivo

El presente documento tiene como propósito ofrecer una visión detallada de la arquitectura del sistema allConnectMarket, abordando aspectos clave como los atributos de calidad, la arquitectura de alto nivel y los factores de riesgo y restricciones asociados. Se establecerá una estructura clara del sistema, alineada con sus objetivos y requisitos arquitectónicos, tanto funcionales como no funcionales.

A lo largo del documento, se analizarán las decisiones arquitectónicas tomadas, justificando su elección y evaluando su impacto en el desarrollo del proyecto. La arquitectura se presenta siguiendo el modelo de vistas 4+1 de Philippe Kruchten, que permite examinar el sistema desde múltiples perspectivas complementarias. Además, se incluirán representaciones gráficas para facilitar la comprensión de la estructura del sistema, y se definirán los pasos a seguir para asegurar que la arquitectura se mantenga alineada con los objetivos estratégicos de allConnectMarket.

Este documento servirá como guía fundamental para el equipo de desarrollo, los arquitectos de software, los stakeholders del proyecto y cualquier persona involucrada en la implementación y mantenimiento de la plataforma, proporcionando una comprensión profunda de cómo los diferentes componentes del sistema interactúan para alcanzar los objetivos de negocio establecidos.

2. Contexto del Sistema

La plataforma multicanal allConnectMarket nace como respuesta a una problemática actual en el comercio electrónico: la fragmentación de las experiencias de compra. Actualmente, los clientes que desean adquirir bienes físicos, contratar servicios profesionales y acceder a contenidos digitales deben recurrir a diferentes plataformas, cada una con su propio catálogo, métodos de pago y políticas de envío o reserva. Esta fragmentación genera diversos inconvenientes que afectan tanto a consumidores como a proveedores.

Entre los principales problemas identificados se encuentra la duplicidad de procesos, donde el usuario debe registrar sus datos múltiples veces y realizar pagos separados en diferentes plataformas. Adicionalmente, la baja personalización resulta inevitable al no existir un perfil unificado que permita generar recomendaciones y promociones consistentes. Los riesgos operativos también se incrementan cuando la disponibilidad y los precios no se actualizan en tiempo real, lo que puede derivar en ventas no cumplidas y una experiencia de usuario deficiente. Finalmente, los costos elevados de integración representan una barrera significativa, ya que incorporar cada nueva línea de negocio implica altos costos de desarrollo y mantenimiento.

Esta situación afecta directamente la fidelización de clientes, incrementa la complejidad operativa y reduce las oportunidades de ventas multicanal. Por tal motivo, surge la necesidad de diseñar e implementar una solución centralizada, escalable y segura que integre múltiples verticales de negocio bajo un ecosistema unificado, basado en patrones arquitectónicos que soporten integración ágil, escalabilidad horizontal y comunicación en tiempo real.

El desarrollo de allConnectMarket permitirá consolidar la experiencia del usuario en un solo flujo de compra, generar recomendaciones inteligentes mediante análisis de comportamiento y machine learning, gestionar la disponibilidad de productos y servicios en tiempo real, mejorar la eficiencia operativa gracias a la integración con sistemas externos de logística, facturación y marketing, y facilitar la incorporación de nuevos proveedores o verticales de negocio sin necesidad de reestructurar el sistema completo.

3. Atributos de Calidad

Los atributos de calidad son características esenciales de un sistema de software que determinan su comportamiento y desempeño más allá de sus funcionalidades principales. Estos atributos permiten evaluar el sistema en términos de factores como eficiencia, seguridad, mantenibilidad y escalabilidad, asegurando que la aplicación cumpla con los requerimientos tanto funcionales como no funcionales establecidos para el proyecto.

En arquitectura de software, cada decisión conlleva trade-offs inherentes, lo que implica que mejorar un atributo de calidad puede afectar negativamente a otro. Por ejemplo, aumentar la seguridad del sistema mediante mecanismos de cifrado extremo a extremo puede impactar el rendimiento al requerir mayor capacidad de procesamiento. De manera similar, optimizar la escalabilidad horizontal mediante una arquitectura de microservicios puede incrementar la complejidad operativa del sistema. De esta manera, el diseño arquitectónico debe encontrar un balance adecuado entre estos atributos, alineándose con los objetivos del sistema y las necesidades prioritarias del negocio y los usuarios.

3.1. Descripción de Atributos de Calidad

Para estructurar la evaluación de los atributos de calidad en allConnectMarket, se utilizará el marco de referencia de la norma ISO 25010, que define diferentes categorías de atributos de calidad, cada una con subcaracterísticas específicas. En el contexto de esta plataforma multicanal, se han identificado los siguientes atributos como los más relevantes para garantizar el éxito del sistema.

La **funcionalidad** evalúa si el sistema proporciona las funciones necesarias para cumplir con los objetivos del usuario de manera precisa y completa. En allConnectMarket,

esto implica garantizar que todas las operaciones relacionadas con el catálogo multicanal, el carrito de compras unificado, el procesamiento de pagos y la generación de recomendaciones funcionen correctamente y de forma integrada.

El **desempeño** analiza el uso eficiente de los recursos y el tiempo de respuesta del sistema al ejecutar sus funciones. Dado el requisito crítico de mantener latencias inferiores a 2 segundos en la carga de catálogos, este atributo resulta fundamental para proporcionar una experiencia de usuario fluida, especialmente durante picos de tráfico como eventos de ventas especiales o promociones.

La **usabilidad** determina la facilidad con la que los usuarios pueden interactuar con el sistema y aprender a utilizar sus funcionalidades. Para una plataforma que integra múltiples tipos de productos y servicios, la interfaz debe ser intuitiva y permitir que usuarios con diferentes niveles de experiencia tecnológica puedan completar sus compras sin fricción.

La **fiabilidad** examina la estabilidad del sistema y su capacidad para operar sin fallos o interrupciones prolongadas. En un entorno de comercio electrónico donde las transacciones financieras están involucradas, la pérdida de datos o la caída del sistema durante un proceso de checkout puede resultar en pérdidas económicas significativas y daño a la reputación de la plataforma.

La **seguridad** garantiza la protección de los datos del usuario, la prevención de accesos no autorizados y la resistencia a ataques maliciosos. Este atributo es crítico dado que allConnectMarket maneja información sensible como datos de tarjetas de crédito, información personal de usuarios y transacciones financieras, requiriendo cumplimiento estricto con estándares como PCI DSS.

La **mantenibilidad** evalúa la facilidad con la que el sistema puede ser actualizado, corregido y adaptado a nuevas necesidades sin comprometer su estabilidad. Dada la naturaleza evolutiva del comercio electrónico y la necesidad de incorporar nuevos proveedores y verticales de negocio, este atributo es esencial para la sostenibilidad a largo plazo del proyecto.

La **escalabilidad** determina la capacidad del sistema para manejar un crecimiento en la cantidad de usuarios o volumen de datos sin degradar su rendimiento. Con el requisito de soportar más de 10,000 usuarios concurrentes, la arquitectura debe permitir escalado horizontal eficiente tanto de servicios como de infraestructura de datos.

Finalmente, la **disponibilidad** mide la capacidad del sistema para estar operativo y accesible en todo momento, minimizando los tiempos de inactividad. Con un SLA objetivo del 99.9 %, el sistema debe contar con mecanismos de redundancia, failover automático y recuperación ante desastres que garanticen la continuidad del servicio.

3.2. Atributos de Calidad en allConnectMarket

A continuación, se describe cómo cada uno de estos atributos de calidad se aplican específicamente en el contexto de allConnectMarket y las decisiones arquitectónicas tomadas para satisfacerlos.

En términos de **funcionalidad**, allConnectMarket debe garantizar que todas las funciones necesarias para la gestión de un ecosistema multicanal sean implementadas de manera completa y precisa. Esto incluye la administración de productos físicos con inventarios en tiempo real, la gestión de servicios profesionales con sistemas de reserva y disponibilidad de proveedores, el manejo de suscripciones digitales con licenciamiento y control de acceso, la implementación de un motor de recomendaciones basado en inteligencia artificial que analice el comportamiento de navegación y compras, y la integración con múltiples pasarelas de pago que soporten diferentes métodos de transacción. La funcionalidad del sistema debe estar alineada con las necesidades del usuario, asegurando que los resultados obtenidos sean relevantes y útiles para completar transacciones de manera eficiente.

El **desempeño** es crítico para proporcionar una experiencia fluida a los usuarios en un entorno de alta concurrencia. El sistema debe ser capaz de procesar consultas al catálogo multicanal en menos de 2 segundos, incluso cuando se integran datos de múltiples proveedores externos con protocolos heterogéneos. El procesamiento de checkout, que incluye validación de inventario, cálculo de pricing con reglas de negocio complejas, autorización de pagos y generación de notificaciones, debe completarse en menos de 3 segundos para mantener la confianza del usuario. Además, el motor de recomendaciones debe generar sugerencias personalizadas en menos de 1 segundo utilizando algoritmos de machine learning previamente entrenados y almacenados en cache. Para lograr estos objetivos de desempeño, se implementa una estrategia cache-first con Redis distribuido, se utiliza procesamiento asíncrono mediante colas de mensajes para operaciones no críticas, y se aplica particionamiento de datos para distribuir la carga de consultas entre múltiples réplicas de base de datos.

En cuanto a **usabilidad**, dado que la plataforma está dirigida a usuarios finales que pueden tener diferentes niveles de experiencia con comercio electrónico, la interfaz debe ser clara y fácil de navegar. El carrito unificado debe presentar de manera comprensible los diferentes tipos de ítems (productos físicos con opciones de envío, servicios con fechas de reserva, contenidos digitales con tipos de licencia), permitiendo que los usuarios gestionen sus compras sin confusión. Las representaciones visuales de recomendaciones, estados de orden y confirmaciones de pago deben ser inmediatamente comprensibles, minimizando la necesidad de soporte técnico. La navegación entre secciones debe requerir la menor cantidad de pasos posible, y el proceso de checkout debe optimizarse para reducir el

abandono de carritos.

La **fiabilidad** implica que el sistema debe ser capaz de operar de manera continua y sin errores críticos que afecten transacciones en curso. La pérdida de datos de órdenes o información de pago es inaceptable, por lo que se implementan mecanismos de respaldo automático con retención de al menos 30 días, replicación síncrona de datos transaccionales críticos, y procedimientos de recuperación point-in-time que permitan restaurar el estado del sistema en caso de fallos. Además, el sistema debe ser resistente a fallos inesperados de proveedores externos mediante circuit breakers que detecten servicios degradados, mecanismos de retry con backoff exponencial para solicitudes fallidas, y estrategias de degradación graceful que permitan continuar operaciones con funcionalidad reducida cuando ciertos proveedores no estén disponibles.

La **seguridad** de allConnectMarket debe garantizar la protección integral de los datos mediante cifrado extremo a extremo utilizando TLS 1.3 para datos en tránsito y AES-256 para datos en reposo. El cumplimiento PCI DSS se logra mediante tokenización de tarjetas de crédito que evita almacenar información sensible directamente, delegación de procesamiento de pagos a pasarelas certificadas, y auditoría completa de todas las transacciones financieras con trazabilidad de cada operación. El control de acceso se implementa mediante autenticación multifactor (MFA) para usuarios y administradores, políticas de autorización basadas en roles con permisos granulares, y sesiones con tokens JWT que expiran automáticamente. La seguridad en la integración con proveedores se asegura mediante VPN tunnels cifrados para comunicaciones sensibles, validación estricta de certificados SSL, y aislamiento de red entre diferentes capas del sistema.

Para garantizar **mantenibilidad**, el sistema está diseñado de manera modular siguiendo principios de Arquitectura Orientada a Servicios (SOA). Cada servicio empresarial (CatalogManagementService, OrderProcessingService, CustomerManagementService, RecommendationService, NotificationService) encapsula responsabilidades específicas y puede ser actualizado independientemente. La implementación de nuevas funcionalidades, como agregar soporte para un nuevo método de pago o integrar un proveedor adicional, debe realizarse sin afectar la estabilidad del sistema mediante el uso de adaptadores e interfaces bien definidas. La documentación del código y la arquitectura se mantiene actualizada en repositorios centralizados, y se siguen convenciones de código estandarizadas para permitir una rápida comprensión por parte de nuevos desarrolladores.

La **escalabilidad** del sistema se aborda mediante una arquitectura distribuida que permite escalado horizontal de servicios críticos. El despliegue en Kubernetes con Horizontal Pod Autoscaler (HPA) permite que servicios como CatalogManagementService y OrderProcessingService escalen automáticamente de 2 a 6 instancias basándose en métricas de CPU y tasa de peticiones. El ESB cluster con 3 nodos balanceados permite distri-

buir la carga de orquestación sin crear cuellos de botella. La base de datos PostgreSQL utiliza arquitectura primary-replica con 1 nodo primario para escrituras y 2 réplicas para lecturas, permitiendo escalado de consultas. El cache distribuido Redis Cluster con 3 nodos permite almacenar catálogos, recomendaciones y sesiones con alta disponibilidad. El message broker ActiveMQ configurado en cluster con particionamiento permite procesar miles de mensajes por segundo de manera concurrente.

Finalmente, la **disponibilidad** es un atributo fundamental dado el requisito de SLA del 99.9 %, lo que permite un downtime máximo de aproximadamente 43 minutos al mes. Para alcanzar este objetivo, el sistema implementa redundancia en todos los componentes críticos mediante load balancer en modo active-standby con health checks cada 5 segundos, ESB cluster de 3 nodos con failover automático, BPEL Engine cluster stateful con base de datos compartida que permite continuar workflows en diferentes nodos, PostgreSQL con failover automático mediante Patroni que promociona réplicas a primario en caso de fallo, y Redis Cluster con replicación que mantiene datos de cache disponibles incluso si un nodo falla. El despliegue multi-zona en Kubernetes distribuye pods en 3 availability zones diferentes, garantizando que fallos de infraestructura en una zona no afecten la disponibilidad global. Se implementan procedimientos de disaster recovery con respaldo diario completo, backups incrementales cada 6 horas, replicación asíncrona a región secundaria para recuperación geográfica, RTO (Recovery Time Objective) de 1 hora y RPO (Recovery Point Objective) de 15 minutos.

3.3. Priorización de Atributos de Calidad

Para garantizar que allConnectMarket cumpla con sus objetivos y ofrezca una experiencia estable y confiable, es esencial priorizar los atributos de calidad en función de su impacto en el sistema. La siguiente clasificación justifica la relevancia de cada atributo dentro del contexto de la plataforma.

3.3.1. Prioridad Alta

La **funcionalidad** se considera de prioridad alta porque representa el valor fundamental que allConnectMarket ofrece a sus usuarios. Sin una implementación completa y precisa de las operaciones de catálogo multicanal, carrito unificado, procesamiento de pagos y generación de recomendaciones, el sistema no cumpliría su propósito básico. Cualquier falla en estos procesos afectaría directamente la utilidad del sistema y la confianza de los usuarios en la plataforma.

La **fiabilidad** es crítica dado que los registros de transacciones y datos de usuarios representan información valiosa tanto para el negocio como para los clientes. El sistema

debe ser capaz de operar de manera continua sin errores que puedan comprometer la integridad de las órdenes procesadas o los datos almacenados. La pérdida de registros de compras o fallos en la recuperación de información pueden afectar negativamente tanto la experiencia del usuario como la viabilidad del negocio. Los mecanismos de respaldo y recuperación ante fallos son fundamentales para garantizar la persistencia de los datos a lo largo del tiempo.

La **disponibilidad** también se clasifica como prioridad alta, dado que los usuarios pueden necesitar acceder a la plataforma para realizar compras en cualquier momento del día. Para un sistema de comercio electrónico, la indisponibilidad significa pérdida directa de ingresos y daño a la reputación. El sistema debe minimizar los tiempos de inactividad y contar con una infraestructura capaz de manejar solicitudes de acceso sin interrupciones. La redundancia en los servicios y estrategias de recuperación ante fallos son necesarias para garantizar que los usuarios siempre puedan interactuar con la aplicación cuando lo requieran.

La **seguridad** en este contexto es esencial debido a la naturaleza sensible de los datos que gestiona la aplicación. La información relacionada con transacciones financieras, datos personales y preferencias de compra debe estar protegida mediante controles de acceso estrictos para evitar accesos no autorizados. El cumplimiento con estándares como PCI DSS no es opcional sino un requisito regulatorio. Adicionalmente, es crucial implementar mecanismos de autenticación robustos y monitoreo de actividad para detectar y mitigar posibles amenazas de seguridad. Un incidente de seguridad podría tener consecuencias catastróficas tanto legales como reputacionales.

3.3.2. Prioridad Media

El **desempeño** del sistema es importante para garantizar una experiencia de usuario fluida, pero puede tolerarse cierta variabilidad siempre que se mantengan los umbrales críticos. La carga de catálogos en menos de 2 segundos y el procesamiento de checkout en menos de 3 segundos son objetivos deseables que mejoran significativamente la experiencia, pero pequeñas desviaciones ocasionales durante picos de carga pueden ser aceptables si se cuenta con mecanismos de comunicación apropiados al usuario. El uso de cache y procesamiento asíncrono permite mitigar la mayoría de problemas de desempeño sin requerir inversión excesiva en infraestructura.

La **usabilidad** es fundamental para garantizar que los usuarios puedan interactuar con la aplicación sin dificultades significativas. Dado que el público objetivo incluye usuarios con diferentes niveles de experiencia tecnológica, la interfaz debe ser eficiente y directa, evitando elementos que dificulten la navegación innecesariamente. Las funciones principales, como la navegación del catálogo, la gestión del carrito y la visualización de estados de

orden, deben ser accesibles con la menor cantidad de pasos posible. Sin embargo, aspectos más avanzados de usabilidad, como personalización de la interfaz o funcionalidades de accesibilidad extendidas, pueden implementarse progresivamente sin comprometer la operación básica del sistema.

3.3.3. Prioridad Baja

La **escalabilidad** es importante en el largo plazo para garantizar que se pueda soportar un crecimiento en la cantidad de usuarios sin degradar su rendimiento. No obstante, en el contexto inicial del proyecto, el enfoque principal debe estar en la estabilidad y confiabilidad del sistema antes de optimizarlo para manejar volúmenes masivos de usuarios concurrentes. La arquitectura está diseñada de manera que permita la expansión futura mediante escalado horizontal de servicios y base de datos, pero esta expansión puede realizarse de manera incremental conforme crezca la demanda real, sin requerir cambios arquitectónicos disruptivos en las fases iniciales.

La **mantenibilidad** es relevante para la evolución de la aplicación, ya que un diseño modular facilita la implementación de mejoras y la corrección de errores sin afectar la estabilidad del sistema. Sin embargo, en la fase inicial del proyecto, este atributo puede gestionarse progresivamente siempre que se sigan buenas prácticas de desarrollo y se mantenga documentación actualizada. La inversión en refactorización extrema o implementación de patrones complejos de mantenibilidad puede posponerse hasta que el sistema demuestre su viabilidad en producción y se identifiquen los puntos reales de complejidad que requieren atención.

4. Escenarios de Calidad

Los escenarios de calidad permiten validar que las decisiones arquitectónicas tomadas efectivamente cumplen con los atributos de calidad priorizados. Cada escenario define una situación específica que el sistema debe manejar, describiendo el estímulo, el artefacto afectado, el ambiente en el que ocurre, la respuesta esperada del sistema y las métricas con las que se evalúa el éxito. A continuación, se presentan los escenarios más relevantes para allConnectMarket.

4.1. Escenario QA-01: Escalabilidad durante Black Friday

Atributo	Escalabilidad
Descripción	El sistema debe soportar picos masivos de tráfico durante eventos de ventas sin degradar el servicio.
Fuente del estímulo	Evento de ventas masivas (Black Friday)
Estímulo	15,000 usuarios concurrentes accediendo simultáneamente durante 4 horas
Artefacto	Web tier, ESB cluster, service tier, base de datos
Ambiente	Operación normal del sistema con demanda extraordinaria
Respuesta	Kubernetes HPA escala Web tier de 3 a 10 pods. Service tier escala de 2 a 6 pods por servicio. ESB cluster distribuye carga entre 3 nodos. Redis cache absorbe 80 % de lecturas. PostgreSQL read replicas manejan consultas. Message broker procesa 5000 mensajes/segundo.
Medida de Respuesta	Latencia catálogo ¡2s (p95). Latencia checkout ¡3s (p95). Disponibilidad ¿99.9 %. Sin degradación de servicio. 0 errores relacionados con capacidad.

4.2. Escenario QA-02: Disponibilidad ante Fallo de Nodo ESB

Atributo	Disponibilidad
Descripción	El sistema debe mantener operatividad ante fallos de infraestructura crítica.
Fuente del estímulo	Fallo de hardware
Estímulo	ESB Node 1 cae completamente (crash)

Artefacto	ESB cluster
Ambiente	Operación con carga normal (5000 usuarios concurrentes)
Respuesta	Load balancer detecta fallo en health check (5 segundos). Tráfico redirigido automáticamente a ESB Node 2 y 3. BPEL Engine continúa workflows desde estado persistido. Peticiones en-vuelo se reencolan en MessageBroker. Kubernetes reinicia ESB Node 1 en 2 minutos. Tráfico se redistribuye al recuperarse Node 1.
Medida de Respuesta	Downtime: 0 segundos (failover transparente). Peticiones perdidas: 0 (requeue automático). Tiempo de recuperación: ¡5 segundos. SLA mantenido: 99.9 %.

4.3. Escenario QA-03: Performance en Actualización de Inventario Tiempo Real

Atributo	Performance
Descripción	El sistema debe propagar actualizaciones de inventario a todos los usuarios en menos de 1 segundo.
Fuente del estímulo	Proveedor actualiza inventario
Estímulo	Proveedor publica evento "Stock agotado producto X"
Artefacto	Event messaging layer
Ambiente	1000 usuarios navegando catálogo simultáneamente
Respuesta	CallbackHandler recibe webhook del proveedor (¡100ms). EventPublisher publica a MessageQueue (¡50ms). EventConsumer procesa evento (¡100ms). ProductRepository actualiza BD (¡200ms). CacheService invalida cache (¡50ms). EventBus notifica a 1000 WebUIs vía WebSocket (¡500ms). UI actualiza disponibilidad sin reload.
Medida de Respuesta	Latencia end-to-end: ¡1 segundo. Propagación a todos los usuarios: ¡500ms. 0 ventas de productos agotados.

4.4. Escenario QA-04: Seguridad en Procesamiento de Pago PCI DSS

Atributo	Seguridad
Descripción	El sistema debe procesar pagos cumpliendo estrictamente PCI DSS sin almacenar datos sensibles.
Fuente del estímulo	Usuario realiza pago con tarjeta de crédito
Estímulo	Usuario ingresa datos de tarjeta de crédito en checkout
Artefacto	PaymentGatewayFacade, EncryptionService
Ambiente	Checkout normal
Respuesta	WebUI cifra datos en browser (TLS 1.3). API Gateway valida certificado SSL. ESB invoca PaymentService (SOAP sobre HTTPS). EncryptionService cifra E2E con AES-256. CreditCardAdapter tokeniza tarjeta (PCI DSS). Datos nunca se almacenan sin cifrar. SecurityService audita transacción. Logs no contienen información sensible.
Medida de Respuesta	100 % de datos cifrados en tránsito y reposo. Tokenización exitosa (no se almacenan tarjetas). Auditoría completa registrada. Cumplimiento PCI DSS validado. 0 brechas de seguridad.

4.5. Escenario QA-05: Resiliencia ante Proveedor con Alta Latencia

Atributo	Resiliencia
Descripción	El sistema debe mantener disponibilidad y experiencia de usuario aceptable incluso cuando proveedores externos presentan alta latencia.
Fuente del estímulo	Sistema externo lento
Estímulo	Proveedor tarda 45 segundos en responder consulta de inventario
Artefacto	IntegrationService, AsyncRequestManager, CircuitBreaker
Ambiente	Usuario navegando catálogo

Respuesta	Usuario solicita catálogo. CatalogManagementService retorna cache inmediatamente (¡2s). AsyncRequestManager envía petición a proveedor en background. Sistema no espera respuesta (asíncrono). Usuario ve catálogo sin bloqueo. CallbackHandler detecta timeout (60s). Sistema reintenta con backoff exponencial. Si 3 fallos consecutivos: CircuitBreaker abre circuito. Cuando proveedor responde finalmente: EventBus actualiza UI en tiempo real.
Medida de Respuesta	Latencia percibida por usuario: ¡2s (cache). Disponibilidad sistema: 99.9 % (no afectada). Actualización en background: transparente. Circuit breaker activa tras 5 fallos consecutivos. Sistema degradado gracefully (sin crash).

4.6. Escenario QA-06: Modificabilidad - Nuevo Proveedor con gRPC

Atributo	Modificabilidad
Descripción	El sistema debe permitir incorporar proveedores con nuevos protocolos sin modificar servicios existentes.
Fuente del estímulo	Equipo de desarrollo
Estímulo	Incorporar proveedor que usa protocolo gRPC (no soportado previamente)
Artefacto	ProtocolAdapterFactory, IProtocolAdapter
Ambiente	Sistema en producción
Respuesta	Desarrollador crea gRPCAdapter implementando IProtocolAdapter. Desarrollador registra adapter en ProtocolAdapterFactory. Admin registra proveedor en ProviderRepository (protocol=grpc). IntegrationService detecta protocolo automáticamente y crea gRPCAdapter. Sistema comienza a integrar con proveedor transparentemente. Sin cambios en servicios enterprise existentes. Sin recompilación de otros componentes. Sin downtime del sistema.

Medida de Respuesta	Tiempo de desarrollo: ¡16 horas. Líneas de código modificadas: ¡500 (solo nuevo adapter). Componentes afectados: 1 (ProtocolAdapterFactory). Downtime para despliegue: 0 (hot deploy). Testing: unit + integration tests completados.

5. Arquitectura

5.1. Descripción de la Plataforma

La plataforma allConnectMarket ha sido diseñada como un ecosistema integrado que centraliza la experiencia de compra multicanal bajo una arquitectura empresarial robusta y escalable. La decisión de adoptar un enfoque de Arquitectura Orientada a Servicios (SOA) tradicional con Enterprise Service Bus (ESB) como elemento central responde a la necesidad de integrar múltiples sistemas heterogéneos, tanto internos como externos, garantizando interoperabilidad, reutilización de servicios y gobernanza centralizada.

A través de una interfaz web responsive accesible desde desktop y dispositivos móviles, junto con aplicaciones móviles nativas y un panel administrativo especializado, los usuarios podrán acceder a diversas funcionalidades que incluyen la navegación de catálogos multicanal que integran productos físicos, servicios profesionales y contenidos digitales; la gestión de un carrito unificado capaz de manejar ítems con reglas de negocio diferenciadas; el procesamiento de checkout con múltiples métodos de pago cumpliendo estándares PCI DSS; la visualización de recomendaciones personalizadas generadas por motores de machine learning; y el seguimiento de órdenes con notificaciones multicanal.

La decisión de estructurar allConnectMarket bajo principios SOA tradicionales responde a varios factores críticos del contexto del proyecto. En primer lugar, la necesidad de integrar proveedores externos con protocolos heterogéneos (SOAP, HTTPS, RPC, gRPC) requiere un middleware capaz de mediar y transformar comunicaciones de manera centralizada. El ESB proporciona esta capacidad de mediación de protocolos, transformación de mensajes y aplicación de políticas de manera consistente. En segundo lugar, los workflows de negocio complejos que involucran múltiples servicios, como el procesamiento de checkout que requiere validación de inventario, cálculo de pricing, autorización de pago, notificación a proveedores y generación de facturas, se benefician de la orquestación centralizada que proporciona el BPEL Engine. Finalmente, la gobernanza empresarial requerida para garantizar seguridad, cumplimiento normativo y calidad de servicio se facilita mediante la centralización de políticas en el ESB.

Esta estructura busca maximizar la disponibilidad y usabilidad de la aplicación, ase-

gurando que los usuarios puedan acceder a sus funcionalidades sin interrupciones y sin necesidad de configuraciones complejas. Al priorizar la accesibilidad multi-dispositivo y la integración transparente con proveedores, la plataforma se convierte en una herramienta útil y práctica para consumidores finales, alineándose con las expectativas de experiencia de usuario del comercio electrónico moderno.

5.2. Decisiones Arquitectónicas

La arquitectura de allConnectMarket adopta un enfoque basado en Arquitectura Orientada a Servicios (SOA) tradicional, combinando servicios empresariales coarse-grained que optimizan la reutilización y la gobernanza del sistema. Se ha diseñado una arquitectura distribuida pragmática que balancea los beneficios de modularidad y separación de responsabilidades con la simplicidad operativa necesaria para un proyecto académico con recursos limitados.

Se priorizó la independencia de cada componente mediante contratos WSDL bien definidos y comunicación exclusivamente a través del ESB, asegurando la escalabilidad y la fiabilidad del sistema a largo plazo. Esta decisión arquitectónica fundamental se basa en varios principios y patrones establecidos que se describen a continuación.

5.2.1. Arquitectura Basada en SOA Tradicional con ESB

La aplicación sigue los principios de la arquitectura SOA tradicional, lo que permite dividir la funcionalidad en servicios empresariales independientes alineados con dominios específicos del negocio. Este enfoque garantiza una mayor modularidad y facilita la evolución del sistema sin afectar otros componentes. Entre sus principales ventajas se encuentran la flexibilidad y mantenibilidad, ya que cada servicio empresarial puede actualizarse o reemplazarse sin afectar el funcionamiento global del sistema mediante el uso de versionado de contratos WSDL y despliegues independientes.

La escalabilidad eficiente se logra porque los servicios pueden escalarse de manera independiente en función de la demanda. Por ejemplo, CatalogManagementService puede requerir más instancias durante navegación intensiva de catálogos, mientras que OrderProcessingService necesita escalado durante eventos de checkout masivos. Esta granularidad en el escalado optimiza el uso de recursos computacionales.

El desacoplamiento y resiliencia se fortalecen mediante la independencia de cada servicio empresarial, lo que reduce el impacto de fallos en el sistema. Si IntegrationService experimenta problemas con un proveedor específico, el resto de servicios continúan operando normalmente. El ESB actúa como punto de absorción de fallos, implementando circuit breakers y retry policies de manera centralizada.

Este enfoque sigue las recomendaciones establecidas por Richards y Ford en "Fundamentals of Software Architecture", quienes enfatizan que la elección de una arquitectura de servicios debe considerar la relación entre escalabilidad y complejidad operativa. Según estos autores, una segmentación excesiva en microservicios puede generar sobrecarga en la comunicación entre servicios y aumentar la dificultad en la gestión. En este sentido, la arquitectura de allConnectMarket busca un equilibrio entre modularidad y simplicidad operativa, garantizando independencia entre servicios sin introducir una fragmentación innecesaria que afecte el desempeño y la mantenibilidad.

El componente central de esta arquitectura es el Enterprise Service Bus (ESB), implementado mediante Apache ServiceMix o WSO2 ESB, que actúa como middleware de integración centralizado. El ESB proporciona ruteo inteligente de mensajes basado en contenido y destino, transformación de protocolos entre SOAP, REST, MQ y JMS, orquestación de servicios mediante invocación de BPEL workflows, gestión de transacciones distribuidas con soporte para compensación, aplicación centralizada de políticas de seguridad, y mediación entre servicios heterogéneos con diferentes tecnologías.

Complementando al ESB, el sistema implementa un registro UDDI (Universal Description, Discovery and Integration) basado en Apache jUDDI que permite la publicación de contratos WSDL por parte de todos los servicios empresariales, el descubrimiento dinámico de servicios mediante búsquedas por taxonomía y categoría, el binding en tiempo de ejecución que permite flexibilidad en la localización de servicios, y el versionado de contratos para permitir evolución de servicios sin romper clientes existentes.

El BPEL Engine, implementado con Apache ODE, proporciona la capacidad de ejecución de workflows BPEL complejos que coordinan múltiples servicios empresariales, coordinación stateful de procesos de larga duración como el procesamiento de órdenes que puede involucrar aprobaciones manuales, manejo automático de compensación cuando transacciones fallan parcialmente, y persistencia de estado que permite reinicio de workflows tras fallos de infraestructura.

5.2.2. Consideraciones de Diseño y Mitigación de Limitaciones

Para abordar las limitaciones inherentes a esta arquitectura y mejorar tanto el rendimiento como la experiencia del usuario, se implementaron varias estrategias que optimizan la disponibilidad, eficiencia y respuesta del sistema, minimizando latencias y asegurando una interacción fluida con la aplicación.

El ESB con balanceo de carga interno se configura en un cluster de 3 nodos que distribuyen el tráfico de manera eficiente entre los microservicios empresariales. Gracias a las capacidades de balanceo de carga interno del ESB, se evita la necesidad de un Load Balancer externo adicional, reduciendo la complejidad arquitectónica y los puntos

de fallo. Cada nodo del ESB cluster mantiene sincronización de configuración mediante almacenamiento compartido, lo que garantiza consistencia en políticas de seguridad y ruteo.

La replicación y alta disponibilidad en bases de datos se implementa mediante un sistema híbrido de almacenamiento. PostgreSQL actúa como base de datos transaccional con configuración Primary-Replica, donde un nodo primario maneja todas las escrituras y dos réplicas sincronizan datos para manejar lecturas. El failover automático mediante Patroni garantiza que si el nodo primario falla, una réplica es promovida automáticamente en menos de 30 segundos. Complementariamente, MongoDB se utiliza para almacenar datos semi-estructurados de proveedores y cache de consultas complejas, beneficiándose de su flexibilidad de esquema para manejar la heterogeneidad de formatos de diferentes proveedores.

El procesamiento asíncrono con proveedores se gestiona mediante un sistema de colas basado en ActiveMQ que desacopla temporalmente las peticiones a sistemas externos. Cuando un usuario consulta el catálogo, CatalogManagementService retorna datos desde cache inmediatamente mientras envía peticiones asíncronas a proveedores en background. El AsyncRequestManager coordina estas peticiones, asignando requestId únicos para tracking, estableciendo timeouts configurables por proveedor, implementando reintentos automáticos con backoff exponencial, y manejando callbacks cuando proveedores responden.

La contenedorización con Docker permite despliegues consistentes y escalables de todos los servicios. Cada servicio empresarial, el ESB cluster, el BPEL Engine y los componentes de infraestructura se ejecutan en contenedores Docker orquestados por Kubernetes. Esto permite la ejecución de múltiples instancias de los microservicios en diferentes entornos sin conflictos de configuración, facilita rolling updates sin downtime mediante estrategias de despliegue blue-green, y proporciona aislamiento de recursos mediante límites de CPU y memoria por contenedor.

El monitoreo y observabilidad en tiempo real se implementa mediante la integración de Grafana y Prometheus para supervisar el estado y rendimiento de los servicios en producción. Se recopilan métricas clave como latencia de respuesta por operación SOAP, uso de CPU y memoria por servicio, disponibilidad de microservicios con alertas automáticas, throughput del ESB medido en mensajes procesados por segundo, y tasa de error en integraciones con proveedores. Adicionalmente, se implementa tracing distribuido mediante Jaeger que permite seguir una petición a través de múltiples servicios, facilitando el diagnóstico de problemas de latencia.

La gestión de autenticación externa se delega a Keycloak implementando OAuth 2.0 y OpenID Connect. En lugar de manejar credenciales dentro del sistema, se implementa

Single Sign-On (SSO) que permite a usuarios autenticarse una vez y acceder a todos los servicios, autenticación multifactor (MFA) configurable para diferentes niveles de seguridad, federación de identidad que podría permitir login con cuentas de Google o Facebook en el futuro, y gestión centralizada de roles y permisos. Esto no solo mejora la seguridad al delegar la responsabilidad de almacenar credenciales a un proveedor especializado, sino que reduce la carga operativa en los microservicios empresariales.

Con estas estrategias, la arquitectura de allConnectMarket logra mitigar riesgos relacionados con disponibilidad, escalabilidad y seguridad, asegurando un funcionamiento óptimo y confiable del sistema incluso en condiciones adversas o de alta demanda.

5.3. Vistas Arquitectónicas (Modelo 4+1)

A continuación se presentan las vistas arquitectónicas del sistema siguiendo el modelo 4+1 de Philippe Kruchten. Cada vista examina la arquitectura desde una perspectiva diferente, proporcionando una comprensión completa del sistema.

5.3.1. Vista Lógica

La vista lógica presenta la descomposición modular del sistema en capas jerárquicas, mostrando los servicios empresariales, sus responsabilidades y las dependencias entre módulos.

[IMAGEN: Diagrama de Vista Lógica]

Inserte aquí la imagen PNG de la vista lógica que muestra la arquitectura en capas con ESB, servicios empresariales, capa de dominio, acceso a datos e infraestructura.

La vista lógica organiza el sistema en 10 capas principales que se describen a continuación desde la capa superior hasta la inferior.

La **Capa de Enterprise Service Bus** actúa como el backbone de integración del sistema, conteniendo cuatro componentes fundamentales. El ESB proporciona ruteo inteligente de mensajes, transformación de protocolos y aplicación de políticas de seguridad. El UDDI actúa como registro centralizado de servicios WSDL permitiendo descubrimiento dinámico. El BPEL Engine orquesta workflows complejos que coordinan múltiples servicios empresariales. El MessageBroker proporciona colas persistentes para comunicación asíncrona con garantía de entrega.

La **Capa de Presentación** incluye la WebUI como interfaz web responsive, la MobileApp para dispositivos móviles, el AdminPanel para monitoreo y gestión, y el UIController que maneja sesiones y autenticación MFA. Todos los componentes de esta capa se comunican exclusivamente con el ESB mediante peticiones SOAP.

La **Capa de Servicios Empresariales** contiene los servicios coarse-grained que encapsulan lógica de negocio compleja. El CatalogManagementService gestiona el catálogo multicanal integrando productos físicos, servicios profesionales y contenidos digitales. El OrderProcessingService coordina todo el proceso de checkout incluyendo validación, pricing y creación de órdenes. El CustomerManagementService maneja autenticación, perfiles y historial de usuarios. El RecommendationService genera sugerencias personalizadas mediante algoritmos de machine learning. El NotificationService coordina el envío de alertas por email, SMS y push notifications. Todos estos servicios publican contratos WSDL en UDDI y son invocados exclusivamente a través del ESB.

La **Capa de Servicios Compartidos** contiene servicios transversales reutilizables por múltiples servicios empresariales. El PaymentService maneja procesamiento de pagos cumpliendo PCI DSS. El IntegrationService coordina la comunicación asíncrona con proveedores externos. El SecurityService proporciona autenticación, autorización, cifrado y auditoría. Estos servicios también exponen contratos WSDL y se invocan vía ESB.

La **Capa de Dominio** contiene las entidades y value objects del modelo de negocio. La entidad Order representa órdenes con ítems, estado y información de pago. CartItem es un value object que encapsula productos, servicios o contenidos digitales en el carrito. Product, DigitalContent y ProfessionalService representan los diferentes tipos de ítems comercializados. User mantiene perfil, historial de compras y navegación. Provider almacena configuración de integración con proveedores externos.

La **Capa de Pasarela de Pagos** es crítica para cumplimiento PCI DSS. El PaymentGatewayFacade orquesta el procesamiento de pagos. El EncryptionService proporciona cifrado extremo a extremo. Los adaptadores CreditCardAdapter, DigitalWalletAdapter y BankTransferAdapter implementan la interfaz IPaymentAdapter para diferentes métodos de pago.

La **Capa de Integración con Proveedores** maneja la complejidad de comunicación asíncrona con sistemas externos. El ProviderIntegrationFacade expone operaciones WSDL para solicitar inventario, precios y disponibilidad. El AsyncRequestManager coordina peticiones asíncronas con tracking de requestId. El ProtocolAdapterFactory crea dinámicamente adaptadores según el protocolo del proveedor. Los adaptadores HTTPSAdapter, SOAPAdapter, RPCAdapter y gRPCAdapter implementan IProtocolAdapter para diferentes protocolos. El ProviderDataTranslator realiza traducción bidireccional entre esquemas de proveedores y el modelo canónico.

La **Capa de Mensajería de Eventos** proporciona actualización en tiempo real. El EventBus permite publicación y suscripción a eventos. El EventPublisher notifica cambios de inventario, precios y estados de orden. El EventConsumer procesa eventos actualizando cache y base de datos. El MessageQueue garantiza entrega de mensajes con persistencia.

La **Capa de Acceso a Datos** abstrae el almacenamiento persistente. El OrderRepository gestiona persistencia de órdenes. El ProductRepository maneja catálogo y actualizaciones de inventario. El UserRepository almacena usuarios, perfiles e historial. El ProviderRepository mantiene configuración de proveedores. DatabaseConnection proporciona conexiones a PostgreSQL con pooling.

La **Capa de Infraestructura** contiene servicios transversales. El Logger registra eventos, métricas y auditoría. El MonitoringService recolecta métricas de latencia, concurrencia y disponibilidad. El CacheService proporciona cache distribuido con Redis. El ConfigurationManager gestiona configuración de servicios, proveedores y políticas de escalado.

Las reglas de dependencia estrictas garantizan la integridad arquitectónica. Toda comunicación entre servicios pasa obligatoriamente por el ESB, nunca directamente. Los servicios empresariales publican contratos WSDL en UDDI para descubrimiento. El BPEL Engine orquesta workflows complejos invocando servicios vía ESB. Las capas superiores solo dependen de capas inmediatamente inferiores. La capa de dominio es independiente de infraestructura. No se permiten dependencias circulares entre módulos.

5.3.2. Vista de Desarrollo

La vista de desarrollo describe la organización del código fuente, las tecnologías utilizadas y las convenciones de desarrollo que guían la implementación del sistema.

[IMAGEN: Diagrama de Vista de Desarrollo]

Inserte aquí la imagen PNG de la vista de desarrollo que muestra la estructura de módulos, paquetes y dependencias del proyecto.

El sistema se organiza en módulos independientes que corresponden a los servicios empresariales y componentes de infraestructura identificados en la vista lógica. Cada servicio empresarial se implementa como un proyecto independiente con su propio ciclo de vida de desarrollo y despliegue.

El stack tecnológico seleccionado balancea madurez, soporte comunitario y alineación con principios SOA tradicionales. Para el ESB se utiliza Apache ServiceMix o WSO2 ESB, ambos con amplio soporte para SOAP, WSDL, transformación de mensajes y orquestación BPEL. El UDDI Registry se implementa con Apache jUDDI siguiendo el estándar UDDI v3. El BPEL Engine utiliza Apache ODE cumpliendo la especificación WS-BPEL 2.0. El Message Broker puede ser Apache ActiveMQ o RabbitMQ, ambos proporcionando garantía de entrega y persistencia de mensajes.

Los servicios empresariales se implementan en Java 17 con Spring Boot, aprovechando

el ecosistema maduro de SOA en la plataforma Java. El framework Apache CXF se utiliza para generación de contratos WSDL y stubs de cliente SOAP. Spring Boot proporciona inyección de dependencias, gestión de transacciones y configuración externalizada.

El frontend web se desarrolla con React 18 y TypeScript para proporcionar una Single Page Application moderna con componentes reutilizables. Las aplicaciones móviles utilizan React Native permitiendo compartir código entre iOS y Android. La base de datos transaccional es PostgreSQL 15 proporcionando garantías ACID, escalabilidad mediante replicación y soporte para datos JSON cuando se requiere flexibilidad de esquema. El cache distribuido utiliza Redis Cluster para almacenar sesiones, catálogos y recomendaciones en memoria con alta disponibilidad.

El motor de recomendaciones se implementa en Python 3.11 utilizando scikit-learn para algoritmos de machine learning, ejecutándose como un servicio separado invocado por RecommendationService. El monitoreo se realiza con Prometheus para recolección de métricas y Grafana para visualización y alertas. El logging centralizado utiliza ELK Stack (Elasticsearch, Logstash, Kibana) para agregación, búsqueda y análisis de logs de todos los servicios.

La contenedorización se realiza con Docker y la orquestación con Kubernetes, permitiendo despliegues consistentes, escalado automático y gestión declarativa de infraestructura. El CI/CD se implementa con Jenkins o GitLab CI para automatización de build, testing y despliegue. El API Gateway utiliza Kong o WSO2 API Manager para rate limiting y autenticación en la capa de presentación. La autenticación se delega a Keycloak para SSO y MFA con soporte OAuth 2.0 y OpenID Connect.

Las convenciones de código siguen estándares establecidos. Para Java se utiliza Google Java Style Guide. Para TypeScript se sigue Airbnb TypeScript Style Guide. El naming es consistente con sufijos como *Service.java para servicios empresariales, *Repository.java para acceso a datos, *Adapter.java para adaptadores de protocolo y *Facade.java para fachadas de integración.

El testing se estructura en múltiples niveles. Los unit tests utilizan JUnit 5 y Mockito para pruebas de componentes aislados. Los integration tests emplean Spring Boot Test y Testcontainers para validar interacción entre servicios. Se establece un coverage mínimo del 80 % para servicios críticos. Los contract tests validan que los contratos WSDL publicados son respetados por implementaciones.

5.3.3. Vista de Procesos

La vista de procesos describe los aspectos dinámicos del sistema, incluyendo concurrencia, distribución de carga, comunicación entre procesos y sincronización.

[IMAGEN: Diagrama de Vista de Procesos]

Inserte aquí la imagen PNG de la vista de procesos que muestra los flujos principales del sistema, incluyendo búsqueda de catálogo, checkout y actualización de inventario.

El sistema maneja múltiples procesos concurrentes que requieren coordinación cuidadosa para garantizar consistencia y rendimiento. Los procesos principales se describen mediante flujos detallados que muestran la interacción entre componentes.

El proceso de búsqueda en catálogo multicanal comienza cuando un usuario envía una búsqueda desde la WebUI, la cual genera una petición SOAP al ESB. El ESB consulta el UDDI para descubrir el endpoint actual de CatalogManagementService y rutea la petición mediante transformación de protocolo si es necesario. CatalogManagementService verifica primero el CacheService con un TTL de 5 minutos. Si hay cache válido, retorna inmediatamente al usuario vía ESB en menos de 2 segundos. Si el cache expiró, inicia un flujo asíncrono en paralelo.

Mientras el usuario recibe la respuesta desde cache, CatalogManagementService invoca IntegrationService a través del ESB para actualizar datos de proveedores. IntegrationService delega a AsyncRequestManager que crea peticiones asíncronas para los N proveedores relevantes, cada una con un requestId único. El ProtocolAdapterFactory crea adaptadores apropiados según la configuración de cada proveedor. Los adaptadores se conectan concurrentemente a los endpoints de proveedores usando HTTPS, SOAP o RPC según corresponda.

El CallbackHandler espera respuestas de proveedores mediante webhooks o polling activo. Cuando las respuestas llegan, ProviderDataTranslator las convierte al esquema canónico del sistema. ProductRepository actualiza la base de datos con nueva información de inventario y precios. EventPublisher publica un evento de actualización al MessageQueue. Los EventConsumers suscritos procesan el evento invalidando cache relevante. Finalmente, WebUI recibe una notificación en tiempo real vía WebSocket y actualiza la interfaz sin recargar la página.

El proceso de checkout BPEL orquestado es el más complejo del sistema debido a la coordinación de múltiples servicios. Comienza cuando el usuario confirma checkout desde WebUI, generando una petición SOAP al ESB. El ESB invoca el BPEL Engine con el workflow ÇompleteCheckout"que coordina los siguientes pasos secuenciales.

En el paso 1, el BPEL Engine invoca OrderProcessingService.validateCart() vía ESB para validar stock de productos físicos consultando proveedores, validar disponibilidad de servicios profesionales verificando calendarios y validar licencias de contenidos digitales. Si alguna validación falla, el workflow se aborta inmediatamente con compensación.

En el paso 2, se invoca OrderProcessingService.calculatePricing() que aplica reglas de pricing diferenciadas por tipo de ítem, calcula impuestos con tasas específicas por

categoría y región, aplica descuentos y promociones según reglas de negocio configuradas, y genera el total final con desglose detallado.

El paso 3 crítico invoca PaymentService.processPayment() vía ESB. EncryptionService cifra datos sensibles con AES-256. CreditCardAdapter valida PCI DSS, tokeniza la tarjeta y solicita autorización a la pasarela de pago. Si el pago falla, el BPEL Engine ejecuta compensación automáticamente, liberando reservas de inventario y notificando al usuario del error.

Si el pago es exitoso, el paso 4 invoca OrderProcessingService.createOrder() que persiste la orden en OrderRepository con estado PENDING y registra la transacción de pago. El paso 5 notifica a proveedores invocando IntegrationService que envía órdenes vía AsyncRequestManager y actualiza el estado a PROCESSING.

El paso 6 envía confirmaciones al cliente mediante NotificationService.sendConfirmation() que coordina email con detalles de orden, SMS con código de confirmación y push notification a aplicación móvil. Finalmente, el paso 7 invoca servicios externos como LogisticsAPIClient para programar envíos de productos físicos, BillingAPIClient para generar factura electrónica y MarketingAPIClient para tracking de conversión.

El BPEL Engine retorna una respuesta agregada vía ESB a WebUI, consolidando el resultado de todas las operaciones. Todo el workflow debe completarse en menos de 30 segundos, con timeout configurable. Si ocurre algún fallo parcial, el motor BPEL ejecuta compensación automática usando transacciones SAGA.

El proceso de actualización en tiempo real mediante eventos es fundamental para mantener consistencia. Cuando un proveedor actualiza su inventario, envía una notificación mediante webhook al CallbackHandler. El CallbackHandler valida la autenticidad del webhook mediante signature verification y parsea el payload del proveedor. ProviderDataTranslator convierte los datos al formato canónico del sistema.

EventPublisher publica un evento ÏnventoryUpdated.ªl MessageQueue con detalles del producto y nueva disponibilidad. El MessageQueue garantiza entrega at-least-once a todos los subscribers mediante persistencia de mensajes. Múltiples instancias de EventConsumer procesan el evento concurrentemente, cada una actualizando ProductRepository en su base de datos local. CacheService invalida las entradas de cache relacionadas con el producto actualizado.

MonitoringService registra métricas de latencia del flujo completo. EventBus notifica a todas las instancias de WebUI conectadas mediante WebSocket. Las interfaces de usuario actualizan la disponibilidad del producto en tiempo real sin necesidad de que el usuario recargue la página. Todo el flujo desde webhook hasta actualización de UI completa en menos de 1 segundo en el percentil 95.

La gestión de concurrencia se realiza mediante pools de threads dimensionados cuida-

dosamente. El ESB mantiene un pool de 200 threads para manejo de peticiones SOAP concurrentes. Cada servicio enterprise mantiene un pool de 100 threads para procesamiento de lógica de negocio. IntegrationService tiene un pool dedicado de 50 threads exclusivamente para comunicación con proveedores, evitando que latencias externas bloqueen threads de servicios internos.

AsyncRequestManager mantiene una cola de 1000 peticiones pendientes con priorización basada en antigüedad y criticidad. MessageQueue se particiona en 10 particiones para permitir procesamiento paralelo de eventos sin contención. La base de datos PostgreSQL mantiene un connection pool de 50 conexiones reutilizables. Redis Cluster distribuye datos en 3 nodos con replicación para alta disponibilidad.

El manejo de fallos implementa múltiples estrategias de resiliencia. El Circuit Breaker se activa tras 5 fallos consecutivos, abriendo el circuito por 30 segundos durante los cuales las peticiones fallan inmediatamente sin intentar contactar al servicio degradado. El patrón Retry implementa 3 intentos con backoff exponencial de 1, 5 y 15 segundos. Los timeouts se configuran en 60 segundos para proveedores externos y 30 segundos para servicios internos.

La compensación BPEL permite reversión automática de transacciones distribuidas cuando algún paso falla. Por ejemplo, si el pago falla tras reservar inventario, el BPEL Engine invoca automáticamente operaciones de compensación que liberan las reservas. La Dead Letter Queue captura mensajes que no pueden procesarse tras 3 reintentos para análisis posterior y reprocesamiento manual si es necesario.

5.3.4. Vista Física (Despliegue)

La vista física describe el mapeo de componentes software a infraestructura física, la topología de red y las estrategias de escalabilidad y alta disponibilidad.

[IMAGEN: Diagrama de Vista Física/Despliegue]

Inserte aquí la imagen PNG de la vista física que muestra la arquitectura de despliegue en Kubernetes con todos los nodos, réplicas y conexiones de red.

La arquitectura de despliegue de allConnectMarket se estructura en múltiples capas de red con diferentes niveles de acceso y seguridad. En la capa externa, un CDN (Cloudflare o Akamai) sirve contenido estático como imágenes, CSS y JavaScript, proporcionando además protección contra ataques DDoS mediante análisis de tráfico y filtrado automático.

El tráfico dinámico pasa a través de un Load Balancer (NGINX o HAProxy) configurado en modo active-standby para alta disponibilidad. Este componente realiza terminación SSL con certificados TLS 1.3, rate limiting por IP para prevenir abuso, health checks cada

5 segundos a los servicios backend y ruteo basado en path a diferentes servicios.

La capa web consiste en pods de Kubernetes que ejecutan WebUI, MobileApp backend y AdminPanel. El Horizontal Pod Autoscaler (HPA) monitorea CPU y tasa de peticiones por segundo, escalando automáticamente de 3 a 10 pods según la demanda. Cada pod tiene asignados 2 CPU cores y 4 GB de RAM.

El API Gateway (Kong o WSO2 API Manager) se ubica entre la capa web y el ESB, proporcionando autenticación y autorización mediante integración con Keycloak, rate limiting por usuario y endpoint, transformación de peticiones REST a SOAP cuando es necesario, y logging detallado de todas las peticiones para auditoría.

El ESB cluster consta de 3 nodos desplegados en diferentes availability zones para tolerancia a fallos. Cada nodo ESB tiene asignados 8 CPU cores y 16 GB de RAM, suficiente para manejar transformación de mensajes y aplicación de políticas. El BPEL Engine cluster comparte almacenamiento persistente para estado de workflows, permitiendo que cualquier nodo continúe un workflow iniciado en otro nodo si este falla.

La capa de servicios empresariales despliega cada servicio (CatalogManagementService, OrderProcessingService, CustomerManagementService, RecommendationService, NotificationService) en pods independientes. El HPA configura diferentes rangos de escalado según la criticidad del servicio. OrderProcessingService escala de 3 a 8 pods dado que maneja checkouts críticos. CatalogManagementService escala de 2 a 6 pods. RecommendationService escala de 2 a 5 pods. Cada pod de servicio tiene 4 CPU cores y 8 GB de RAM.

El ActiveMQ cluster proporciona mensajería asíncrona con 3 brokers configurados en cluster con replicación. Los mensajes se persisten en storage compartido garantizando durabilidad. Las colas se particionan para permitir procesamiento paralelo de eventos. Cada broker tiene 4 CPU cores y 16 GB de RAM.

La capa de datos utiliza PostgreSQL en configuración Primary-Replica. El nodo primario maneja todas las escrituras y tiene 8 CPU cores, 32 GB de RAM y 1 TB de almacenamiento SSD. Dos réplicas síncronas manejan lecturas y tienen la misma especificación. Patroni gestiona el failover automático, promoviendo una réplica a primario si detecta fallo del nodo principal en menos de 30 segundos.

Redis Cluster se configura con 3 nodos master, cada uno con su propia réplica para alta disponibilidad. Los datos se particionan automáticamente entre los masters usando consistent hashing. Cada nodo Redis tiene 4 CPU cores, 16 GB de RAM y 100 GB de almacenamiento SSD para persistencia opcional.

La capa de observabilidad incluye Prometheus con 4 CPU cores y 16 GB de RAM para recolección de métricas de todos los servicios. Grafana visualiza las métricas y genera alertas configurables. El ELK Stack con 4 CPU cores y 16 GB de RAM y 500 GB de storage

para logs centraliza los logs de todos los servicios permitiendo búsqueda y análisis.

La estrategia de escalabilidad horizontal se implementa mediante Kubernetes HPA que monitorea múltiples métricas. Para la capa web, si la CPU supera el 70 % o las RPS (requests per second) superan 1000, se agregan pods hasta el máximo de 10. Para servicios empresariales, cada servicio tiene umbrales personalizados. OrderProcessingService escala agresivamente durante eventos de ventas. CatalogManagementService escala basándose en cache miss rate.

El ESB cluster escala manualmente agregando nodos según análisis de tendencias de carga. Actualmente configurado con 3 nodos, puede extenderse a 5 nodos para eventos especiales. La base de datos PostgreSQL escala lecturas agregando réplicas adicionales, actualmente con 2 réplicas, escalable a 5 sin cambios arquitectónicos. El escalado vertical del nodo primario permite hasta 64 CPU cores y 128 GB de RAM.

La alta disponibilidad se garantiza mediante redundancia en todos los niveles. El Load Balancer en modo active-standby con health checks cada 5 segundos garantiza failover en menos de 10 segundos. El ESB cluster de 3 nodos con balanceo automático permite que 2 nodos manejen toda la carga si uno falla. El BPEL Engine cluster stateful continúa workflows desde estado persistido sin pérdida de progreso.

El MessageBroker cluster de 3 brokers con replicación garantiza entrega de mensajes incluso si 1 broker falla. PostgreSQL con failover automático vía Patroni promociona réplica a primario en menos de 30 segundos. Redis Cluster con replicación mantiene datos disponibles incluso si un nodo master falla, promoviendo su réplica automáticamente.

El despliegue multi-zona de Kubernetes distribuye pods en 3 availability zones diferentes, garantizando que fallos de infraestructura en una zona (como pérdida de energía o red) no afecten la disponibilidad global del sistema. Los pod disruption budgets configurados garantizan que siempre haya al menos 2 pods de cada servicio crítico corriendo durante actualizaciones.

La distribución geográfica se estructura con una región principal en us-east-1 (N. Virginia) que maneja todo el tráfico de producción. Una región secundaria en eu-west-1 (Irlanda) actúa como sitio de disaster recovery con replicación asíncrona de datos. El CDN tiene puntos de presencia en más de 150 ubicaciones globalmente, garantizando latencias inferiores a 50ms en América del Norte, inferiores a 100ms en Europa e inferiores a 150ms en América Latina.

La seguridad en red se implementa mediante segmentación estricta. La DMZ pública contiene únicamente la capa web con acceso desde internet. La subnet privada contiene servicios empresariales, ESB y base de datos sin acceso directo desde internet. El firewall permite únicamente puertos 80 y 443 desde internet hacia el load balancer. Kubernetes Network Policies segmentan tráfico entre pods permitiendo únicamente comunicaciones

autorizadas.

El cifrado protege datos en tránsito mediante TLS 1.3 en todas las comunicaciones externas e internas sensibles, y datos en reposo mediante AES-256 para bases de datos y almacenamiento de archivos. Los túneles VPN establecen comunicaciones seguras con proveedores externos que requieren acceso directo, utilizando IPSec o WireGuard con autenticación mutua basada en certificados.

El disaster recovery implementa backup diario completo de PostgreSQL con retención de 30 días, backups incrementales cada 6 horas con retención de 7 días, y replicación asíncrona a región secundaria con lag máximo de 1 hora. El RTO (Recovery Time Objective) establecido es de 1 hora para restauración de operaciones en región secundaria, y el RPO (Recovery Point Objective) es de 15 minutos representando la máxima pérdida de datos aceptable. El failover a región secundaria es manual requiriendo aprobación de management, excepto en caso de desastre completo de región primaria donde puede activarse automáticamente.

6. Riesgos

Los riesgos identificados en allConnectMarket se agrupan en tres categorías principales que pueden afectar de forma directa o indirecta la calidad, estabilidad y éxito del sistema.

6.1. Riesgos de Producto

Los riesgos de producto se relacionan con la calidad, funcionalidad, seguridad y rendimiento del sistema final.

En términos de seguridad, existen vulnerabilidades potenciales en la gestión de autenticación y autorización. Errores en la validación de tokens JWT podrían permitir accesos no autorizados a datos de órdenes o información de usuarios. La gestión deficiente del almacenamiento de datos sensibles, como información de tarjetas de crédito incluso tokenizadas o registros de transacciones, sin cifrado adecuado representa un riesgo crítico. Las vulnerabilidades en la configuración del ESB, como exposición de endpoints administrativos o políticas de seguridad mal configuradas, podrían comprometer la integridad del sistema completo.

Respecto al rendimiento, la latencia en el procesamiento de transacciones debido a la arquitectura distribuida con múltiples saltos entre ESB, servicios empresariales y bases de datos podría degradar la experiencia de usuario. La sobrecarga del sistema durante picos de peticiones simultáneas, especialmente en fechas como Black Friday, puede saturar el ESB convirtiéndolo en cuello de botella. Los cuellos de botella en la comunicación

asíncrona con proveedores, como saturación de colas de mensajes o timeouts excesivos, pueden afectar la actualización de inventarios.

La disponibilidad del sistema enfrenta riesgos de interrupciones del servicio debido a errores en la configuración de Kubernetes, problemas de red entre pods o fallos en la sincronización del ESB cluster. El tiempo de inactividad por mantenimientos no planificados de componentes críticos como PostgreSQL o el BPEL Engine puede exceder el presupuesto de downtime del SLA del 99.9 %. La dependencia de servicios externos como Keycloak para autenticación o pasarelas de pago significa que la caída de estos servicios afecta directamente la capacidad de usuarios para acceder al sistema o completar compras.

6.2. Riesgos de Proceso

Los riesgos de proceso están asociados al desarrollo, integración y mantenimiento del sistema.

En mantenibilidad, el aumento en la complejidad del código base especialmente por la integración de múltiples tecnologías (ESB, BPEL, SOAP, REST, microservicios) puede dificultar la localización y corrección de errores. Los cambios o actualizaciones en componentes del ESB o framework SOAP como Apache CXF pueden generar incompatibilidades o introducir nuevos errores que requieren actualizaciones coordinadas en múltiples servicios. La falta de documentación técnica clara sobre la arquitectura SOA, especialmente los workflows BPEL y las políticas de ruteo del ESB, puede dificultar futuras modificaciones o el traspaso del proyecto a nuevos desarrolladores.

Los riesgos relacionados con la implementación incluyen la posible falta de experiencia del equipo con herramientas como BPEL Engine, configuración avanzada de ESB o diseño de servicios SOAP, lo que puede ocasionar errores de diseño o implementación de antipatrones. La disponibilidad limitada de tiempo puede afectar la cobertura de pruebas, especialmente pruebas de integración end-to-end que validen workflows BPEL completos, la implementación de casos borde como manejo de fallos parciales en transacciones distribuidas, o la entrega oportuna de funcionalidades clave como el motor de recomendaciones o integración con nuevos proveedores.

6.3. Riesgos de Proyecto

Los riesgos de proyecto corresponden a factores externos o limitaciones generales que pueden afectar el cumplimiento de los objetivos.

En recursos financieros, las restricciones presupuestarias pueden impedir acceder a licencias empresariales de ESB como WSO2 ESB que ofrecen soporte comercial y funcionalidades avanzadas, suscripciones avanzadas de servicios de autenticación como Auth0 o

Keycloak con SLA garantizado, o mayor capacidad de cómputo en la nube para despliegue de entornos de staging y producción robustos.

La disponibilidad de recursos presenta limitaciones en el uso de herramientas de monitoreo empresariales como Dynatrace o New Relic que podrían proporcionar observabilidad superior pero tienen costos prohibitivos. La falta de acceso a entornos de prueba replicables que simulen producción, especialmente para probar escalabilidad bajo carga de miles de usuarios concurrentes, limita la validación de atributos de calidad críticos.

Las limitaciones de hardware representan que las máquinas virtuales disponibles para despliegue de desarrollo y pruebas pueden no contar con suficiente capacidad de procesamiento, memoria o almacenamiento. Esto afecta directamente la capacidad de ejecutar un cluster ESB completo con 3 nodos, múltiples instancias de servicios empresariales y bases de datos replicadas localmente, forzando simplificaciones en el ambiente de desarrollo que pueden ocultar problemas que solo aparecerán en producción.

7. Restricciones

Las restricciones son limitaciones específicas que afectan la capacidad del sistema para cumplir con ciertos requisitos o estándares y deben ser consideradas durante el desarrollo.

7.1. Cumplimiento Normativo

El sistema debe cumplir con regulaciones legales y normativas vigentes relacionadas con protección de datos y privacidad. En Colombia, la Ley 1581 de 2012 establece el marco general para el tratamiento de datos personales, requiriendo consentimiento explícito de usuarios para recolección y procesamiento de información personal, políticas de privacidad claras y accesibles que expliquen qué datos se recolectan y cómo se utilizan, mecanismos para ejercer derechos de acceso, rectificación y eliminación de datos personales, y medidas de seguridad apropiadas para proteger información sensible.

El cumplimiento PCI DSS (Payment Card Industry Data Security Standard) es obligatorio dado que el sistema procesa pagos con tarjetas de crédito. Esto limita significativamente el diseño de la arquitectura de pagos, requiriendo tokenización de tarjetas para evitar almacenar números completos, segregación de red con la pasarela de pagos aislada de otros componentes, auditoría completa de accesos a datos de pago con logs inmutables, pruebas de penetración anuales realizadas por entidades certificadas, y uso exclusivo de protocolos de cifrado aprobados como TLS 1.2 o superior.

7.2. Restricciones Tecnológicas

La integración con proveedores externos está limitada por los protocolos que estos soportan. Aunque el sistema implementa adaptadores para HTTPS, SOAP, RPC y gRPC, algunos proveedores pueden usar protocolos propietarios o versiones antiguas que requieren adaptación especial. La latencia de respuesta de proveedores es completamente externa al control del sistema, lo que obliga a implementar estrategias de cache agresivas y procesamiento asíncrono.

La base de datos empresarial compartida, aunque simplifica la gestión de transacciones distribuidas y reduce complejidad operativa comparada con el patrón de base de datos por servicio de microservicios puros, crea acoplamiento a nivel de esquema que debe gestionarse cuidadosamente mediante versionado de esquemas y migraciones coordinadas.

7.3. Restricciones de Recursos

El proyecto cuenta con recursos limitados de infraestructura para ambientes de desarrollo y pruebas. Los ambientes de staging no pueden replicar completamente la escala de producción, limitando la capacidad de validar comportamiento bajo carga real de 10,000+ usuarios concurrentes. Las pruebas de carga deben realizarse con herramientas open source como JMeter o Gatling que, aunque efectivas, requieren mayor esfuerzo de configuración y análisis comparadas con soluciones comerciales.

El equipo de desarrollo tiene expertise limitado en tecnologías SOA tradicionales como BPEL y ESB, ya que la industria ha migrado mayormente a microservicios y arquitecturas cloud-native. Esto requiere inversión en capacitación y puede resultar en curva de aprendizaje prolongada que afecte los tiempos de desarrollo. Sin embargo, esta restricción también representa una oportunidad de aprendizaje de patrones arquitectónicos empresariales aún relevantes en contextos de integración con sistemas legacy.

8. Referencias

- Bass, L., Clements, P., Kazman, R. (2012). *Software Architecture in Practice*. 3rd Edition. Addison-Wesley.
- Clements, P. et al. (2010). *Documenting Software Architectures: Views and Beyond*. 2nd Edition. Addison-Wesley.
- Erl, T. (2008). *SOA: Principles of Service Design*. Prentice Hall.
- Kruchten, P. (1995). *The 4+1 View Model of Architecture*. IEEE Software, Vol. 12, No. 6.

- Richards, M. and Ford, N. (2023). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, Inc.
- ISO/IEC 25010:2011. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*.