Universidad ORT

Facultad de Ingeniería

Arquitectura de software

Obligatorio TePagoYa

Rodrigo Martínez - 154505

Ángel Díaz – 161761

Grupo: N7A

Docentes: Mathias Fonseca, Andrés Calviño

Noviembre 2018

ÍNDICE

1. Introducción

Este documento presenta información sobre la arquitectura y características del sistema TePagoYa.

El documento se estructura en dos secciones: una primera sección donde se identifican y especifican los requerimientos funcionales, no funcionales y restricciones; una segunda sección enfocada al detalle de la arquitectura, donde se utilizan distintas vistas para representar los aspectos más relevantes del sistema y reflejar las decisiones tomadas a nivel de diseño de la solución.

Es importante destacar que la siguiente lectura se elaboró para un público familiarizado con leguaje técnico referente a desarrollo de software.

1.1. Propósito

El propósito del documento es proveer una especificación completa de la arquitectura de TePagoYa, dejando una idea clara de las decisiones tomadas.

2. Antecedentes

2.1. Propósito del sistema

El propósito del sistema es desarrollar un *backend* en *NodeJS* que permita simplificar lo más posible la integración de los siguientes sistemas:

- Comercio.
- Gateway.
- Red.
- Emisor.

La propuesta es crear una herramienta (TePagoYa) que provea la funcionalidad de actuar de intermediario entre los cuatro diferentes actores resolviendo la mayor parte posible los desafíos de integración, como pueden ser: los diferentes formatos de fechas y números, campos obligatorios y opcionales, formatos de representación de datos, etc.; permitiendo que la comunicación sea siempre hacia y desde el nuevo sistema, identificando el actor de destino de cada comunicación.

2.2. Requerimientos significativos de Arquitectura

Se describen a continuación los requerimientos: funcionales, no funcionales y restricciones que llevaron a la arquitectura desarrollada.

2.2.1 Resumen de Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
REQF01: Compra en comercio	El comercio debe poder procesar una transacción de compra (enviada por un consumidor). Para esto debe decidir en función de la categoría del producto, a qué gateway enviar la transacción.	comercio
REQF02: Compra en Gateway	El gateway debe poder procesar una transacción de compra (enviada por un comercio). Para esto debe decidir en función de la tarjeta a qué gateway enviar la transacción.	gateway
REQF03: Compra en Red	La red debe poder procesar una transacción de compra (enviada por un gateway). Para esto debe decidir en función de la tarjeta a qué emisor enviar la transacción.	Red
REQF04: Análisis de Fraude	Para el análisis de fraude, se revisa la cantidad de transacciones que hizo esa tarjeta en las últimas 72 horas y si	Red

	supera un límite determinado la transacción se rechaza.	
REQF05: Compra en emisor	El emisor debe poder procesar una transacción de compra (enviada por la red).	emisor
REQF06: Control Saldo Suficiente	Cuando la red envía una transacción a ser aprobada en el emisor, este debe validar que el saldo de la cuenta del titular sea suficiente para el monto a aprobar.	emisor
REQF07: Días de devolución	El emisor podrá definir la cantidad máxima de días que pueden transcurrir entre que se realiza una compra y la devolución de la misma.	emisor
REQF08: Devolución	Deberá ser posible realizar la devolución de cualquier compra realizada previamente que no haya superado el límite de días definido en el emisor.	comercio/tePagoYa/ gateway /red/emisor
REQF09: Chargeback	El usuario puede realizar un desconocimiento de compra en su emisor. El emisor deberá informar al comercio de este evento para que ejecute las acciones administrativas pertinentes.	comercio/ tePagoYa / gateway / red /emisor
REQF10: Cierre de Lotes	El comercio le solicita diariamente al gateway la transacción de cierre de lotes, en donde se informa al máximo detalle los movimientos de dinero que hubo en su cuenta en ese día.	gateway
REQF11: Registro de Interfaz	Para cada prestador de servicios podrá registrar una “interfaz de servicio” que permita acceder a su funcionalidad.	tePagoYa
REQF12: Especificar formatos	Los prestadores de servicios podrán especificar propiedades sobre la forma de usar cada interfaz (por ejemplo, el formato de las estructuras de datos que se intercambian por la interfaz, o el tipo de comunicación).	tePagoYa
REQF13: Localización de Interfaces	Para cualquier consumidor de servicios de debe poder localizar una interfaz de un servicio registrado.	tePagoYa
REQF14: Validez de tarjeta de crédito.	Controlar validez de la tarjeta. Para que una tarjeta sea válida, debe haber sido emitida por el emisor, pasar el algoritmo de Luhn, no estar vencida, ni bloqueada (por falta de pago), ni denunciada (por robo o pérdida).	emisor
REQF15: ChargeBack	Se debe poder hacer un Chargeback de hasta 6 meses atrás	emisor

Tabla 2.1 – Requerimientos funcionales

2.2.2 Resumen de Requerimientos No Funcionales

ID Atributo	Descripción	Atributo de calidad	Actor
REQNF01: Selección de Gateway	La selección del gateway debe ser configurable sin necesidad de modificar el código de los comercios.	Modificabilidad	comercio
REQNF02: Tiempos de identificación de red	La identificación de la red debe realizarse en tiempos menores a 10 ms en promedio bajo cargas de 1000 solicitudes por minuto.	Eficiencia	gateway
REQNF03: Configuración de límites de compras	El límite de operaciones debe poder ser alterado en tiempo de ejecución sin necesidad de interrumpir la operación de la red.	Modificabildad	red

REQNF04: Tiempos de identificación de emisor	La identificación del emisor debe realizarse en tiempos menores a 10 ms en promedio bajo cargas de 1000 solicitudes por minuto.	Eficiencia	red
REQNF05: Tiempo de procesamiento del cierre de lotes.	Esta transacción debe responder en un tiempo promedio menor a 50 ms bajo cargas de 5000 rpm.	Eficiencia	gateway
REQNF06: Cumplimiento de PCI	Toda aplicación debe cumplir con los estándares de seguridad de PCI, pudiendo demostrar y justificar a cada momento el nivel de cumplimiento.	Seguridad	comercio/ tePagoYa / gateway / red /emisor/ autenticador **
REQNF07: Mecanismo de integración de aplicaciones	Toda aplicación que se registra en TePagoYa puede actuar como prestador de servicios y/o consumidor de servicios.	Modificabildad	tePagoYa
REQNF08: Gestión de errores y fallas	El sistema debe proveer suficiente información, de alguna forma, que permita conocer el detalle de las tareas que se realizan. En particular, en el caso de ocurrir una falla o cualquier tipo de error.	Disponibilidad	comercio/ tePagoYa / gateway / red /emisor
REQNF09: Autenticación de operaciones hacia prestadoras de servicios	Para toda aplicación “prestadora de servicios” que se registre en TePagoYa, se requiere que todas sus operaciones publicadas deben ser invocadas únicamente a través de TePagoYa.	Seguridad	comercio/gateway/red/emisor
REQNF10: Posibilidad de cambiar la herramienta de Autenticación y de reutilización.	Desarrollar una herramienta que pueda ser reutilizada en otros aplicativos, y permitir cambiar la misma generando el menor impacto posible en el sistema ya implementado.	Modificabilidad	autenticador **
REQNF11: Autenticación de operaciones hacia TePagoYA	Toda invocación a TePagoYa debe provenir de una aplicación consumidora de servicios legítima.	Seguridad	autenticador **
REQNF12: Configuración Hora de cierre de lote	El sistema permitir cerrar los lotes a una hora acordada	Modificabilidad	gateway

Tabla 2.2 – Requerimientos no funcionales

** autenticador: representa el aplicativo encargado de manejar las validaciones del sistema.

2.2.3 Resumen de Restricciones

ID Restricción	Descripción
RES1: Backend	La implementación del Backend debe desarrollarse en Node js utilizando las tecnologías vistas en el curso.
RST2: APIs	Las APIs expuestas por los servicios de comercios deben ser REST.
RST3: GIT	Todo el código fuente, documentación, archivos de configuración o cualquier otro artefacto referido al desarrollo de los prototipos debe gestionarse en el repositorio GIT asignado.

Tabla 2.3 – Restricciones

3. Documentación de la arquitectura

En las secciones de este capítulo se documenta la arquitectura del sistema por medio de distintas vistas (Módulos, Componentes y Conectores, Asignación).

- **Vista de Módulos**: se detallan los módulos identificados en las distintas aplicaciones desarrolladas, los cuales fueron agrupados de acuerdo a sus responsabilidades. Para dicha representación se utilizaron dos vistas:
    - **Vista de Descomposición:** muestra el detalle de los módulos del sistema, para tener un contexto de la solución.
    - **Vista de Uso del aplicativo tePagoYa:** muestra los módulos internos al aplicativo y dependencias existentes.
- **Vista de Componentes y Conectores**: se detallan los elementos que tienen presencia en tiempo de ejecución, tales como: procesos, servidores, BD, etc, y las interacciones entre ellos representadas como conectores.
- **Vista de Asignación/Despliegue:** se detalla el mapeo entre los elementos de software y de hardware para tener claridad cómo desplegar los distintos aplicativos a nivel servidor.

3.1. Vistas de Módulos

3.1.1 Vistas de Descomposición

3.1.1.1 Representación primaria

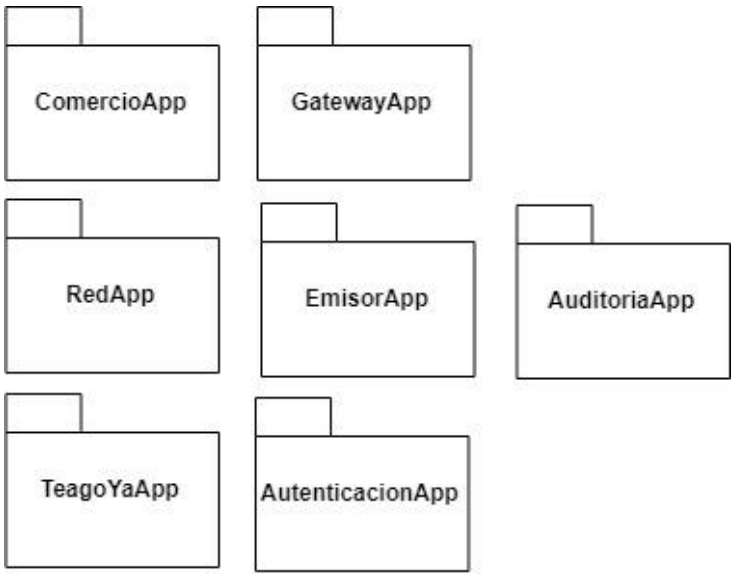

Ilustración 3.1 – Diagrama de Módulos

3.1.1.2 Catálogo de elementos

Elemento	Responsabilidades
ComercioApp	Agrupa la lógica de negocio del comercio. Este ofrece las funcionalidades especificadas en la tabla de requerimientos bajo el actor Comercio. Entre ellas recibir una transacción, seleccionar el gateway en función de su categoría y enviar dicha transacción a TepagoYa para que continúe el flujo hasta el gateway.
GateWayApp	Agrupa la lógica de negocio del gateway. Este ofrece las funcionalidades especificadas en la tabla de requerimientos bajo el actor gateway. Entre ellas recibir una transacción, seleccionar la red en función del número de tarjeta y enviar dicha transacción a la red seleccionada por medio de tePagoYa. Además ofrece el servicio de cerrar lotes para un comercio específico.
RedApp	Agrupa la lógica de negocio de la red. Este ofrece las funcionalidades especificadas en la tabla de requerimientos bajo el actor red. Entre ellas recibir una transacción, seleccionar el emisor en función de la tarjeta y enviar dicha transacción al emisor seleccionado por medio de tePagoYa. Además realiza controles de prevención de fraude utilizando el histórico de transacciones realizadas por una tarjeta en las ultimas 72hrs.
EmisorApp	Agrupa la lógica de negocio del emisor. Este ofrece las funcionalidades especificadas en la tabla de requerimientos bajo el actor emisor. Entre ellas recibir una transacción, controlar la validez de la tarjeta, el saldo de la cuenta para el monto de la transacción.
TePagoYA	Agrupa la lógica de negocio de tePagoYa. Este ofrece las funcionalidades especificadas en la tabla de requerimientos bajo el actor tePagoYa. Es el responsable de la interoperabilidad entre todos los actores del sistema.

AutenticacionApp	Agrupa la lógica referente a la autenticación. Responsable de resolver la autenticación entre los distintos actores por medio de tokens.
AuditoriaApp	Agrupa la lógica referente a la gestión de errores y log en el procesamiento de cada transacción de los distintos actores.

Tabla 3.1 – Catálogo de elementos

3.1.1.3 Decisiones de diseño

Como se puede visualizar en la Ilustración 3.1 se desarrolló un aplicativo por cada actor de forma de separar las responsabilidades y agrupar la lógica de forma independiente. Las comunicaciones se dan entre los actores (Comercio, Gateway, Red y Emisor) hacía y desde TePagoYa, de forma de que este último tenga la responsabilidad de la interoperabilidad entre los actores.

Como se puede observar, fue necesario desarrollar dos aplicativos “ajenos” a los actores mencionados:

1. **auditoriaApp**: Gestiona los errores y log de los aplicativos, permitiendo tener la trazabilidad de las tareas que se realizan. Por cada error o actividad que se realiza, el sistema guarda el dato del evento en un repositorio en la nube, permitiendo así tener un diagnóstico rápido y preciso sobre la causa de los errores y posibilidad de contar con información respecto a los eventos del sistema. Fue necesario realizar un aplicativo independiente para cumplir con los requerimientos no funcionales: REQNF08 y para aplicar la táctica de Mantener un Seguimiento de Auditoria. En el repositorio se podrán encontrar entradas del tipo error en general y log.

   Además, se aplicó la táctica de Detección de Excepciones y Manejo de Excepciones por medio de bloques de código try/catch, de forma que el sistema pueda sobrellevar las fallas, seguir prestando servicios y al mismo tiempo, desplegar información que sirva de ayuda para entender las causas de la excepción.

2. **autenticacionApp**: Gestiona la autenticación de las operaciones. Al iniciar cada aplicativo actor, se realiza una *request* a autenticacionApp solicitando el *login*, si

el usuario y la contraseña enviados son válidas se retorna un token con expiración 24 hrs. Este es enviado en el *header* de cada petición hacia tePagoYa y desde ésta al resto de los actores, por tanto al recibir el *token*, cada actor valida el mismo en el aplicativo y consulta el rol asociado a dicho usuario.

Por medio de la validación del rol de cada aplicativo se tiene certeza que toda invocación a TepagoYa proviene de una aplicación consumidora de servicios y que esta sea legítima (registrada en la base de datos). Además, se valida que las peticiones que se realizan a los actores sean únicamente invocaciones realizadas por tePagoYa.

Por temas de seguridad, la contraseña es *hasheada* previo a la persistencia en el aplicativo, promoviendo así el REQNF06 y aplicando la táctica de Encriptación de Datos. Por motivos de alcance, no se *hashea* en el los aplicativos de los actores, quedando expuesta y sin seguridad en una variable de entorno.

Por medio del aplicativo se aplicaron las tácticas de Identificación, Autenticación y Autorización de Actores, cumpliendo con el REQNF06, REQNF09, REQNF10 y REQNF11.

Para este *realease*, no se implementa el servicio de "refresque" del *token*, por tanto una vez que el *token* haya expirado, es necesario iniciar nuevamente cada aplicativo actor.

Además, para un próximo release, se podría implementar la táctica de Bloquear Computadoras frente a un escenario donde un aplicativo tiene varios intentos de logueo fallido, y post bloqueo, aplicar la táctica de Informar Actores.

Dado que todas las aplicaciones consumen servicios de TePagoYa de una forma similar, se desarrollaron los módulos pensando en la abstracción, se buscó desarrollar servicios más generales en lugar de especializados para cada tipo de consumidor, por tanto se utilizó la táctica de Abstraer Servicios Comunes.

3.1.2 Vista de Uso tePagoYa

3.1.2.1 Representación primaria

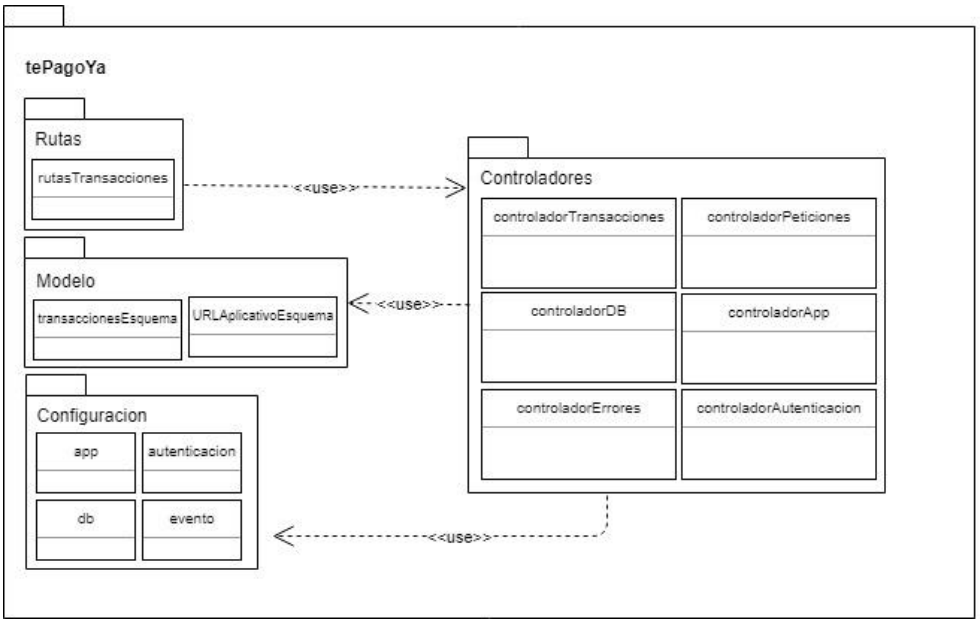

Ilustración 3.2 – Diagrama de Uso del aplicativo tePagoYa

3.1.2.2 Catálogo de elementos

Elemento	Responsabilidades
Rutas	Contiene las rutas de acceso al aplicativo, las URLs que permiten acceder a las distintas funcionalides y así atender las peticiones que llegan a la raíz del servidor.
Rutas.transacciones	Contiene las rutas de acceso a las funcionalidades referente a las transacciones.
Modelo	Contiene los modelos utilizados en el esquema de base de datos.
Modelo.transaccionesEsquema	Contiene el esquema de datos de las transacciones.
Modelo.URLAplicativoEsquema	Contiene el esquema de datos utilizado para el registro
Controladores	Contiene los controladores necesarios para acceder a los datos solicitados por medio de las Rutas.
Controladores.controladorPeticiones	Contiene la lógica referente a las peticiones a los distintos aplicativos.
Controladores.controladorDB	Contiene la lógica referente a las funcionalidades para la manipulación de la base de datos.
Controladores.controladorErrores	Contiene la lógica referente a la gestión de los errores.
Controladores.controladorAutenticacion	Contiene la lógica referente a la autenticación.
Controladores.controladorTransacciones	Contiene la lógica referente al manejo de todo lo referente a las transacciones.
Controladores.controladorApp	Contiene la lógica referente a la inicialización de la app, levantar servidor,

	establecer conexión a la BD y autenticar la app solicitando el token.
Configuración	Contiene variables de entorno.
Configuración.app	Contiene variables referentes a la aplicación.
Configuración.autenticacion	Contiene variables referentes a la autenticación.
Configuración.db	Contiene variables referentes a la BD
Configuración.evento	Contiene variables referentes al manejo de eventos, tanto errores como log

Tabla 3.2 – Catálogo de elementos

3.1.2.3 Decisiones de diseño

En la ilustración 3.2 se muestra un diagrama "tipo" de unos los aplicativos de la solución. Todos respetan mayormente la misma organización en cuanto a módulos, por tanto se detallan aspectos de diseño referente a este aplicativo pero también se mencionan las decisiones tomadas en los otros.

En la ilustración se pueden visualizar a simple vista las relaciones entre los distintos módulos, entre las cuales se encuentra la dependencia de "uso". Estas hacen que sea posible comprender las dependencias entre los módulos, ayudando a evaluar la calidad del diseño de la solución en base a los atributos de calidad de software especificados en la Tabla 2.2.

Se buscó que las dependencias o relaciones de uso en los módulos vayan hacia los componentes que menos cambian, por ejemplo es más probable que cambie la lógica de negocio en un sistema que la capa de persistencia o los modelos de datos en los que estos se basan. Adicionalmente se realizó el diseño tratando de eliminar y/o reducir los ciclos.

Tener claridad en las dependencias existentes entre los distintos módulos, facilita el entendimiento del impacto de cualquier cambio, pudiendo identificar qué partes del software pueden verse afectadas por el mismo.

Como se puede ver en la ilustración, se buscó agrupar las responsabilidades similares, encapsulándolas en módulos, para esto nos apoyamos en la táctica de Incrementar la Coherencia Semántica. De tal modo, frente a la existencia de un cambio, evitar que este se propague a módulos que no tienen relación con el cambio.

Por otro lado, se buscó que las funciones implementadas, hagan una única “cosa”, tengan un único motivo por el cual cambiar frente a un cambio, por tanto se utilizó la táctica de Dividir Modulos respetando además, el principio de Responsabilidad Única.

Se utilizó el patrón modelo-vista-controlador en un contexto donde fue necesario separar la lógica del negocio y de los datos. Por temas de alcance de proyecto, no se implementó para este release una interfaz de usuario que represente la vista, pero se utilizó el aplicativo postman para visualizar los datos. Este tipo de arquitectura favoreció la reutilización de código y la separación de responsabilidades, lo que facilitó el desarrollo del sistema y su futuro mantenimiento.

Para el almacenamiento de datos, se utilizó una base no relacional MongoDB, una herramienta opensource orientada a documentos JSON, lo cual facilitó el manejo de los mismos. Para acceder a la base, se utilizó la librería Mongoose de JavaScript, donde por medio de una API intuitiva, se logró manejar los datos fácilmente.

El uso de la herramienta permitió hacer frente a los cambios en los requerimientos, permitiendo agregar y eliminar entradas de la base de forma sencilla, algo que en una base de datos relacional como lo es SQL, quizás hubiera demandado más tiempo.

Otros aspectos importantes a destacar de la misma, son la validación de datos que trae y la extracción de la mayor parte del código de cada aplicativo.

Además, es importante mencionar que utilizando una base de datos no relacional se optimizan aspectos de performance, evitando los joins necesarios para cruzar tablas en una base de datos relacional normalizada. Esta decisión de implementación fue fundamental para mejorar los tiempos de respuesta y así promover los requerimientos REQNF02, REQNF04 y REQNF05.

Respecto al REQNF05, se podrían haber mejorado aún más los tiempos de respuesta utilizando el concepto de views de MongoDB. Estas solo admiten operaciones de lectura y sus tiempos de respuesta son más performante ya que no se consultan los documentos; pero por temas de alcance y proyecto, no se implementan para este reléase.

Para asegurar los dos primeros requerimientos REQNF02 y REQNF04, se utilizó la táctica de mantener múltiples copias de datos, replicando los datos en repositorios distintos con velocidades de acceso distintas: cache y base de datos. De esta forma se optimizaron los tiempos de respuesta, ya que se accede a memoria cache en lugar de utilizar memoria secundaria (Ver. Anexo: 4.1, anexo: 4.2).

Para crear el servidor web de cada aplicativo, se utilizó el framework Express lo que facilitó su creación y configuración, así como también el manejo de los diferentes verbos http; algo que resultó sumamente importante en un contexto de necesidad de desarrollo-rápido. Además, su popularidad facilitó la fase inicial de investigación y capacitación de la herramienta.

Siguiendo con el objetivo de promover la mantenibilidad del sistema y promover los atributos de calidad asociados a los requerimientos no funcionales REQNF01, REQNF03 y REQNF11, se utilizó la táctica de Diferir el Tiempo de Enlazado.

Por un lado, en reglas generales de la solución, se aplicó la táctica por medio de la parametrización de varios aspectos de configuración (DB, aplicación, autenticación, etc.), para esto se creó un paquete con módulos referentes a la configuración tal cual se expresa en la ilustración 3.2

Por otro lado, en lo que refiere al REQNF01, se aplicó la táctica mencionada por medio de MongoDB, para esto el comercio cuenta con una tabla que tiene la relación gateway - categoría, por tanto si se requiere modificar la categoría asociada a un gateway, simplemente se modifica el registro de la tabla en la base de datos del comercio. De esta forma, se puede modificar fácilmente la selección del gateway sin tocar código existente, y por otro lado permitir su configuración en tiempo de ejecución. Este mismo esquema de datos fue utilizado para la identificación de la red en el gateway y la identificación del emisor en la red.

Para cumplir con el REQNF03, se creó una tabla referente a configuraciones en la base de datos del emisor, por tanto razonando análogamente al punto anterior, se provee flexibilidad para la modificación del límite de compras y su configuración puede ser en tiempo de ejecución.

Finalmente, para promover el REQNF07 y la modificabilidad, se creó un documento en la base de datos de tePagoYa para manejar el almacenamiento de los endpoints de los distintos actores, tal como se puede ver en la ilustración 3.2 bajo el nombre: URLAplicativoEsquema. Por temas de alcance, no se define un endpoint en tepagoYa para dar de alta los actores.

Además, se decide no centrar toda la información en tePagoYa, ya que de ser así, cada actor necesitaría - previo a la resolución de una transacción - refrescar su información respecto a los actores disponibles. A modo de ejemplo, cuando un comercio desea identificar el gateway, este debiera pedirle a tePagoYa la lista de gateways, eso agregaría una request más en las comunicaciones del sistema, algo que a nivel de performance sería un aspecto negativo. Por otro lado, para el contexto dado, las altas y bajas de actores no es algo que suceda periódicamente como para necesitar continuamente refrescar la información.

Por tanto para un siguiente release, se debiera generar un endpoint en tePagoYa, comercio, gateway y red, de forma que al dar de alta un nuevo actor, se le comunica al resto para que cada uno dé de alta el nuevo actor en su base.

3.2. Vistas de Componentes y conectores

3.2.1 Vistas de Componentes y conectores tePagoYa

3.2.1.1 Representación primaria

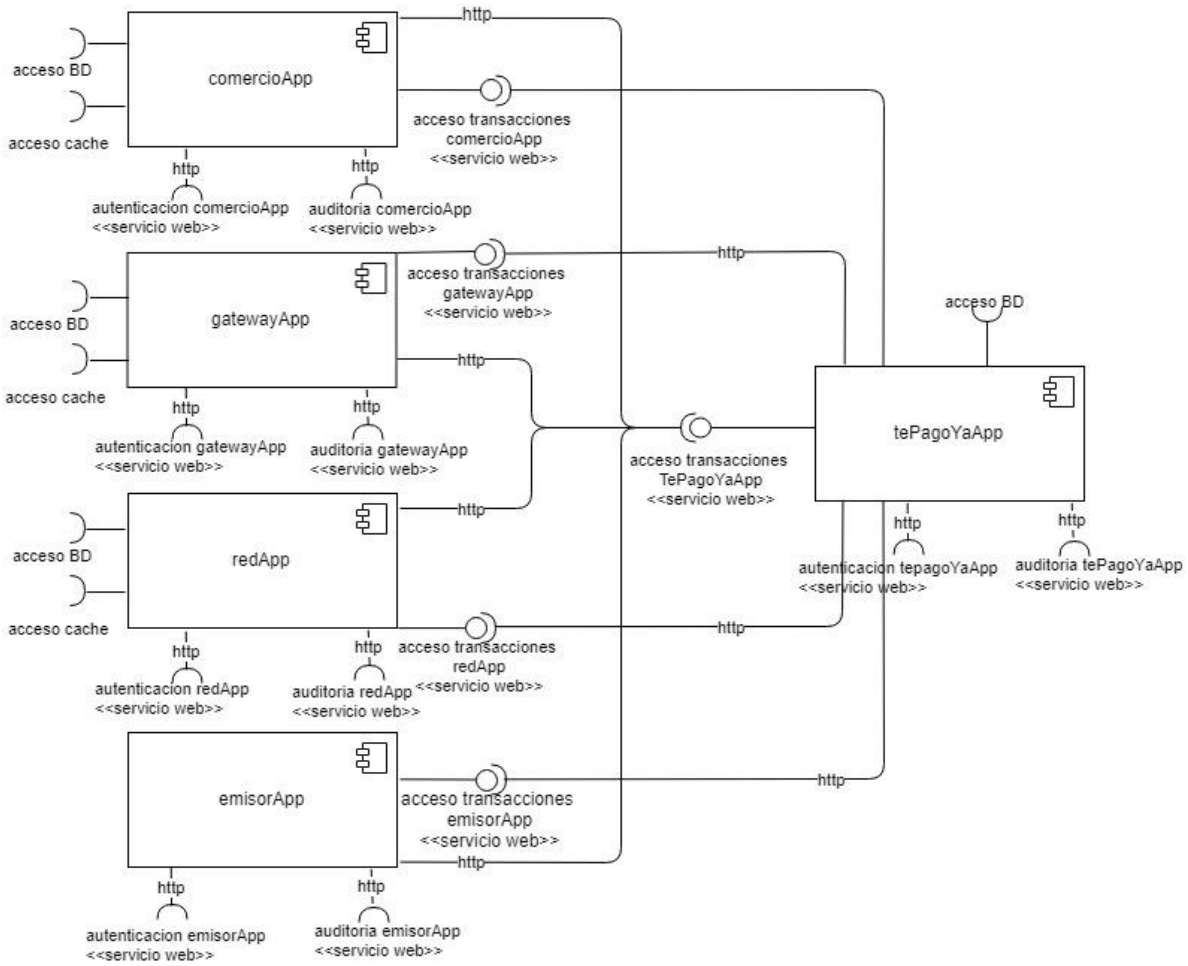

Ilustración 3.5 – Diagrama de componentes y conectores

3.2.1.2 Catálogo de elementos

Componente/Conector	Descripcion
autenticacionApp	Servicio web que expone métodos para obtener un token y autenticar usuarios.
erroresApp	Servicio web que expone métodos para registrar eventos tanto referente a acciones como a errores/fallas del sistema.
comercioApp	Servicio web que expone métodos para registrar transacciones, devoluciones, charbacks.
gatewayApp	Servicio web que expone métodos para registrar transacciones, devoluciones, chargebacks y realizar cierre de lotes.
redApp	Servicio web que expone métodos para registrar transacciones, devoluciones, chargebacks.

emisorApp	Servicio web que expone métodos para registrar transacciones, devoluciones, chargebacks.
tePagoYa	Servicio web que expone métodos para registrar transacciones, devoluciones, chargebacks y solicitar cierre de lotes a un gateway.

Tabla 3.3 – Catálogo de elementos

3.2.1.3 Decisiones de diseño

Como se puede observar en la Ilustración 3.3, se buscó desacoplar los componentes utilizando interfaces API REST por medio de comunicaciones http, de esta forma se minimiza las dependencias en la comunicación de los aplicativos, independizándose así de las implementaciones concretas de cada parte del sistema. Esto permite que varias implementaciones de los aplicativos puedan coexistir, permitiendo intercambiarlos. A modo de ejemplo: podría tener dos implementaciones de una API en Comercio y evaluar que implementación es más performante.

Es importante destacar que esta arquitectura permite el desarrollo en base a abstracciones, cumpliendo así con las buenas prácticas de la industria, y apoyando el desarrollo de sistemas de calidad. Además, en base a interfaces definidas que funcionan como contratos se permite el desarrollo en paralelo.

3.3. Vista de Asignación

Con el objetivo de representar la asignación de elementos de software a elementos de hardware, se muestra a continuación una Vista de Despliegue, aprovechando las ventajas de la arquitectura utilizada.

3.3.1 Vista de Despliegue

3.3.1.1 Representación Primaria

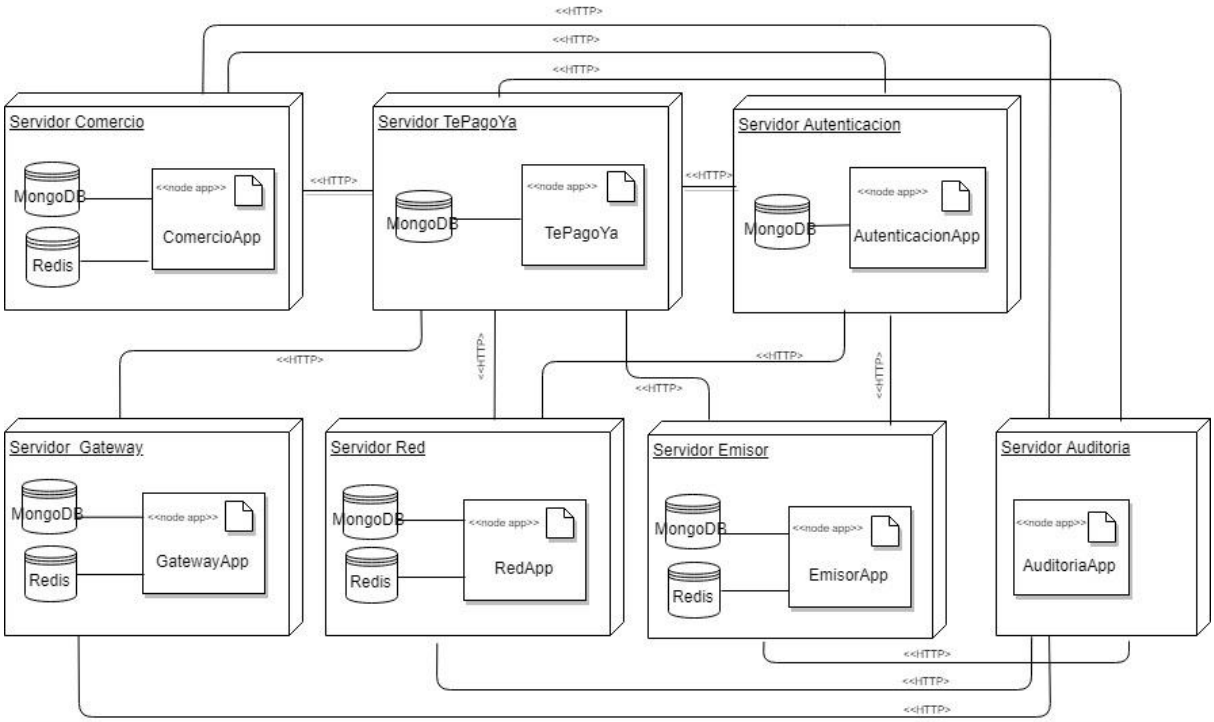

Ilustración 3.6 – Diagrama de Despliegue

4. Anexo

4.1. Pruebas Redis

Se recomienda su despliegue en siete servidores físicos distintos, permitiendo una mejor capacidad de mantenimiento, re-despliegue y escalabilidad.

Como se puede observar las pruebas realizadas donde se visualiza cómo se pasa de tiempos de respuesta de 59ms a 4ms.

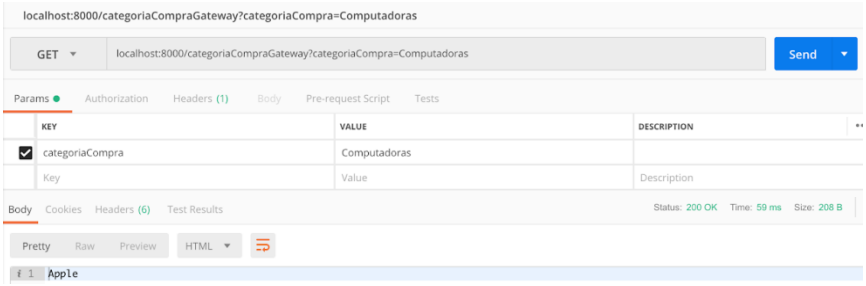

Ilustración 3.3 – Prueba consulta Gateway en base de datos

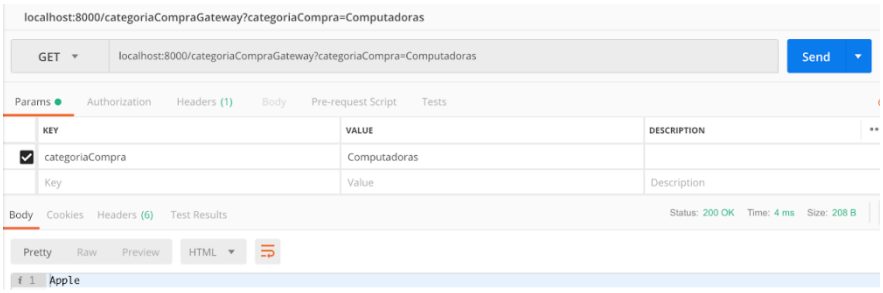

Ilustración 3.4 – Prueba consulta Gateway en cache

4.2. Pruebas de Carga

Se llevaron a cabo pruebas de Carga en distintas modalidades:

- **Sistema completo:** comenzando la transacción en el comercio, pasando por todos los nodo y persistiendo. Resultados:
    - 1.200 peticiones enviadas.
    - 50% aprox. peticiones procesadas exitosamente.
    - Tiempo promedio por petición: 30.000 ms.
- **Gateway con autenticación**: se prueba la resolución de la red autenticando contra autenticacionApp. Resultados:
    - 1.200 peticiones enviadas.
    - 100% peticiones procesadas exitosamente.
    - Tiempo promedio por petición: 9 ms.
- **Gateway sin autenticación**: se prueba la resolución de la red sin autenticar contra el aplicativo. Resultados:
    - 1.200 peticiones enviadas.
    - 100% peticiones procesadas exitosamente.
    - Tiempo promedio por petición: 4 ms.

Las pruebas fueron realizadas en un PC con las siguientes características: procesador i3 1.9GHz, 4 procesadores lógicos, 6GB RAM.

De acuerdo a las pruebas, se puede verificar el cumplimiento de los requerimientos REQNF02 y REQNF04.

Ilustración 4.1 – Prueba carga al Sistema Completo

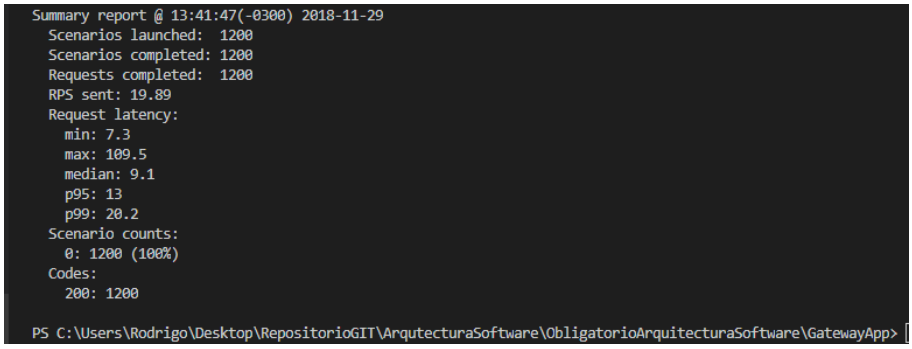

Ilustración 4.2 – Prueba carga gateway con autenticación.

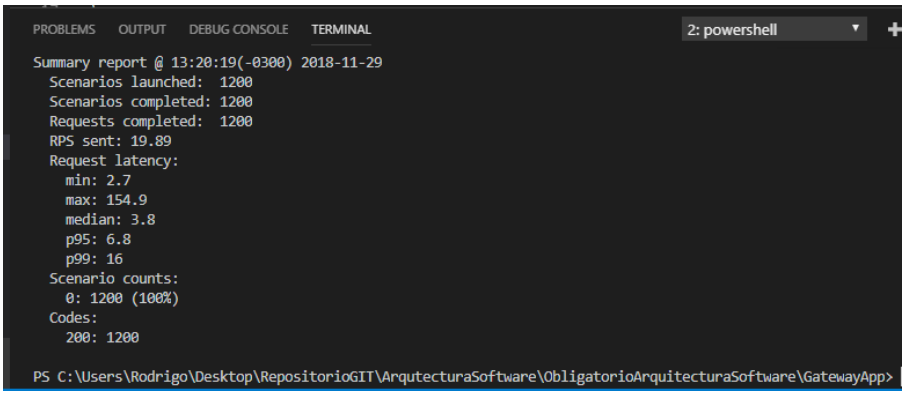

Ilustración 4.3 – Prueba carga gateway sin autenticación.