DOCUMENTACIÓN DEL OBLIGATORIO - DESCRIPCIÓN DE ARQUITECTURA

Plataforma de streaming de eventos de vídeo bajo demanda phiTV

[22/06/2023]

Aramís Cicheski – 228430

Denise Apple - 194854

Marcel Lewi – 239641

Link al repositorio

Índice

1. Introducción

En el siguiente documento presentaremos el diseño y la construcción de la arquitectura de software planteada para la Plataforma de streaming de eventos de vídeo bajo demanda phiTV de la empresa AureaUY

1.1 Propósito

El propósito de este documento es proveer una descripción de la arquitectura de la plataforma phiTV. Se procederá con los Antecedentes del proyecto los cuales contendrán el propósito del sistema así como los requerimientos significativos de la arquitectura (Funcionales y No Funcionales). También tendremos las Vistas las cuales contemplarán su representación primaria, el catálogo de elementos y las decisiones de diseño si corresponden o no.

2. Antecedentes

2.1 Propósito del sistema

El propósito del sistema es brindar una plataforma de streaming de eventos de vídeo bajo demanda denominada phiTV la cual está formada por diversos actores tales como los Proveedores (empresas que utilizan la plataforma para la distribución del contenido que generen mediante suscripciones y a su vez la transmisión del mismo), Clientes (los usuarios que comprarían el contenido ofrecido por el proveedor mediante una suscripción y por acceder a las transmisiones del mismo) y Administradores (técnicos encargados de gestionar el acceso a los proveedores). Específicamente, la misma cuenta con los siguientes objetivos:

- Permitir a las empresas proveedoras de contenido publicar en pocos minutos eventos). Además, se quiere que puedan controlar las suscripciones de los usuarios, los precios que le otorguen a los eventos y a los pagos que realicen los usuarios.
- Permitir soportar las altas demandas provocadas por el masivo ingreso de usuarios que consumen los eventos que proveen las empresas.

- Brindar un marco de interacción confiable entre los interesados mediante el intercambio de información seguro.
- Sea auditable, que permita ver toda la información relacionada a la autenticación de usuarios y realizar diagnósticos rápidos y precisos en caso de errores.
- La plataforma provee mecanismos que le permiten ser flexibles a la hora de utilizar los servicios externos tales como la Pasarela de Pagos, la Unidad Reguladora, e incluso el mecanismo de logger que se adopte.

2.2 Requerimientos significativos de Arquitectura

A continuación, detallamos los Requerimientos significativos de la Arquitectura, en los cuales se incluyen los requerimientos funcionales y los requerimientos de atributos de calidad.

2.2.1 Resumen de Requerimientos funcionales

ID Requerimiento	Descripción	Actor
REQ 1 – Alta de Proveedores	Se debe proveer la capacidad para dar de alta un proveedor para que pueda operar en la plataforma (Nombre, Dirección, Correo Electrónico, Teléfono, Código de moneda, Nombre de moneda, Símbolo de moneda, Precio predeterminado)	Administrador
REQ 2 – Autorizar/Desautorizar un evento	Los eventos publicados por los Proveedores deben ser autorizados previo a su comienzo, manualmente o automática.	Administrador
REQ 3 – Consultar bitácora de la plataforma	Consultar la bitácora de la actividad de la plataforma en un determinado periodo. La actividad de la plataforma que necesitan	Administrador

	consultar es la siguiente: 1. Creación de eventos 2. Modificaciones de eventos 3. Suscripción de clientes a eventos	
REQ 4 – Crear evento	Creación de un evento (Nombre del evento, Descripción, Fecha de inicio, Fecha de fin, Imagen miniatura, Imagen principal, Categoría del evento, Video del evento). Los eventos no deben quedar disponibles automáticamente en la Plataforma para que los usuarios se suscriban. Necesitan ser aprobados por un Administrador. La fecha de inicio debe ser mayor que la fecha actual La fecha de fin debe ser mayor que la fecha de inicio	Proveedor
REQ 5 – Editar evento	Modificar si el evento no está aprobado aún. Mismas validaciones que en el REQ 4 - Crear evento	Proveedor
REQ 6 – Envío de mensajes	Enviar un correo electrónico con información relevante de un evento a cada uno de los Clientes que se suscribieron y solicitaron que se les envíe información.	Proveedor
REQ 7 – Consultar Eventos	Obtener una lista de eventos disponibles que hayan sido aprobados y cuya fecha de fin no sea anterior a la fecha de consulta, con el objetivo de poder elegir uno.	Cliente
REQ 8 – Comprar un evento	Para comprar un evento los usuarios deben ingresar los siguientes datos (Nombre completo, Fecha de	Cliente

	Nacimiento, Dirección de Correo Electrónico, País de Origen, Evento, Proveedor). Dicha compra se valida con la Pasarela de Pagos	
REQ 9 – Acceder a un evento	Permitir acceder a un video del evento en el horario estipulado solamente a los usuarios que pagaron por él.	Cliente
REQ 10 – Consulta de todos los Eventos para Proveedores	Retorna una lista con todos los eventos de un proveedor y por cada uno de ellos, la siguiente información: 1. Cantidad de Clientes suscriptos 2. Tiempo promedio de compra 3. Cantidad máxima de Clientes concurrentes 4. Tiempo promedio que tienen que esperan o tuvieron que esperar los Clientes para que el video se empiece a ejecutar 5. Cantidad de mensajes enviados 6. Cantidad de mensajes fallidos 7. Cantidad de eventos aprobados y desaprobados	Proveedor
REQ 11 – Consulta de un Evento para Proveedores	Para un evento de un proveedor se debe retornar los siguiente: 1. Cantidad de Clientes suscriptos 2. Tiempo promedio de compra 3. Cantidad máxima de Clientes concurrentes 4. Tiempo promedio que tienen que esperan o tuvieron que esperar los Clientes para que el video se	Proveedor

	empiece a ejecutar 5. Cantidad de mensajes enviados 6. Cantidad de mensajes fallidos	
REQ 12 – Consulta de Eventos Activos para Administradores	Para cada uno de los eventos que se están ejecutando actualmente (eventos activos) en la plataforma, se necesita obtener la siguiente información: 1. Cantidad de eventos 2. Cantidad de Clientes suscriptos para cada evento 3. Cantidad de Clientes concurrentes por evento 4. Tiempo promedio que tienen que esperar los Clientes para que el video se empiece a ejecutar	Administrador
REQ 13 – Consulta de todos los Eventos para Administradores	Dado un período, se debe listar la siguiente información: 1. Lista de los eventos aprobados discriminados por autorización manual o automática 2. Lista de los eventos desaprobados 3. Cantidad de eventos registrados 4. Cantidad de eventos aprobados discriminados por autorización manual o automática 5. Cantidad de eventos pendientes de aprobar 6. Cantidad de eventos desaprobados 7. Cantidad de usuarios suscritos	Administrador
REQ 14 – Consulta de todos los Eventos para Usuarios	Dado un usuario, se debe listar la siguiente información tanto para los eventos a los que se suscribió como para los que aún puede	Cliente

	suscribirse: • Nombre del Evento • Descripción • Categoría del evento • Fecha de inicio • Fecha de Fin • Precio • Cantidad de usuarios suscriptos	

2.2.2 Resumen de Requerimientos de Atributos de Calidad

ID Requerimiento	ID Requerimiento de Calidad o restricción	Descripción
REQ 1 – Alta de Proveedores		Se debe autenticar y autorizar solamente a aquellos usuarios que sean Administradores para poder realizar esta acción
REQ 2 – Autorizar/Desautorizar un evento	Modificabilidad Integrabilidad / Interoperabilidad	Debido a que el proveedor tiene que estar autorizado por la Unidad Reguladora, y haber pagado a la empresa validando a través del servicio expuesto por la Pasarela de pagos, nos parece importante que contemos con una buena Interoperabilidad entre el sistema y dichos servicios para asegurar el cumplimiento de las validaciones. A su vez, dichas validaciones deben ser fáciles de agregar, modificar o eliminar, por lo que creemos que con *Pipes and Filters* apuntamos a satisfacer el requerimiento de Modificabilidad (Restringir dependencias, Incrementar

		la coherencia semántica, Encapsular, Diferir enlaces en tiempo de arranque)
REQ 3 – Consultar bitácora de la plataforma	Disponibilidad Seguridad	Se necesita poder consultar la bitácora de la actividad de la plataforma en un determinado periodo. La actividad de la plataforma que necesitan consultar es la siguiente: 1. Creación de eventos 2. Modificaciones de eventos 3. Suscripción de clientes a eventos Para cada uno de los ítems anteriores, se necesita saber quién realizó cuál fue la acción, quién la hizo y cuándo.
REQ 4 – Crear evento		Se debe autenticar y autorizar solamente a aquellos usuarios que sean Proveedores
REQ 5 – Editar evento		Se debe autenticar y autorizar solamente a aquellos usuarios que sean Proveedores
REQ 6 – Envío de mensajes	Performance Interoperabilidad	Se le debe enviar un correo a los Clientes que soliciten y se hayan suscrito a un evento, ya que se podrían tener que enviar muchas solicitudes de correo electrónico simultáneas, y asegurar una buena comunicación con el proveedor de emails.
REQ 7 – Consultar Eventos		Se debe autenticar y autorizar solamente a aquellos usuarios que sean Clientes
REQ 8 – Comprar un evento	Interoperabilidad Performance	Para realizar el pago del evento, se debe utilizar el proveedor que esté utilizando la Pasarela de

		Pagos. Nos debemos asegurar que se tenga un protocolo de intercambio con proveedores de pago que funcione correctamente. Además, se implementó una Queue para procesar el pago, de manera de satisfacer Performance
REQ 9 – Acceder a un evento	Escalabilidad Performance Disponibilidad	Se debe acceder a un video del evento en el horario estipulado solamente a los usuarios que pagaron por él. Es importante realizar un diseño utilizando mecanismos que permitan dotar a la plataforma con la capacidad de que el acceso a los eventos esté disponible para los clientes cuando lo necesiten y que el tiempo de acceso sea el menor posible.
REQ 10 – Consulta de todos los Eventos para Proveedores	Performance	Las consultas deben poder realizarse en un tiempo de menos de 5 segundos, demostrando que, para un elevado número de usuarios concurrentes, los tiempos se deben mantener relativamente constantes en el tiempo. Se utilizó CQRS para manejar la Performance.
REQ 11 – Consulta de un Evento para Proveedores	Performance	Las consultas deben poder realizarse en un tiempo de menos de 5 segundos, demostrando que, para un elevado número de usuarios concurrentes, los tiempos se deben mantener relativamente constantes en el tiempo. Se utilizó CQRS para manejar la Performance.

REQ 12 – Consulta de Eventos Activos para Administradores	Performance	Las consultas deben poder realizarse en un tiempo de menos de 5 segundos, demostrando que, para un elevado número de usuarios concurrentes, los tiempos se deben mantener relativamente constantes en el tiempo. Se utilizó CQRS para manejar la Performance.
REQ 13 – Consulta de todos los Eventos para Administradores	Performance	Las consultas deben poder realizarse en un tiempo de menos de 5 segundos, demostrando que, para un elevado número de usuarios concurrentes, los tiempos se deben mantener relativamente constantes en el tiempo. Se utilizó CQRS para manejar la Performance.
REQ 14 – Consulta de todos los Eventos para Usuarios	Performance	Las consultas deben poder realizarse en un tiempo de menos de 5 segundos, demostrando que, para un elevado número de usuarios concurrentes, los tiempos se deben mantener relativamente constantes en el tiempo. Se utilizó CQRS para manejar la Performance.
REQ 15 - Gestión de errores y fallas	Disponibilidad Modificabilidad Seguridad	Se tiene que tener el detalle de la actividad de los usuarios que requieran autenticación y guardar información acerca de cualquier error, por lo que creemos que mantener un registro que permita cumplicar estos objetivos es clave. A su vez, se espera que la solución contemple la posibilidad de poder cambiar las herramientas o librerías concretas que se utilicen para producir

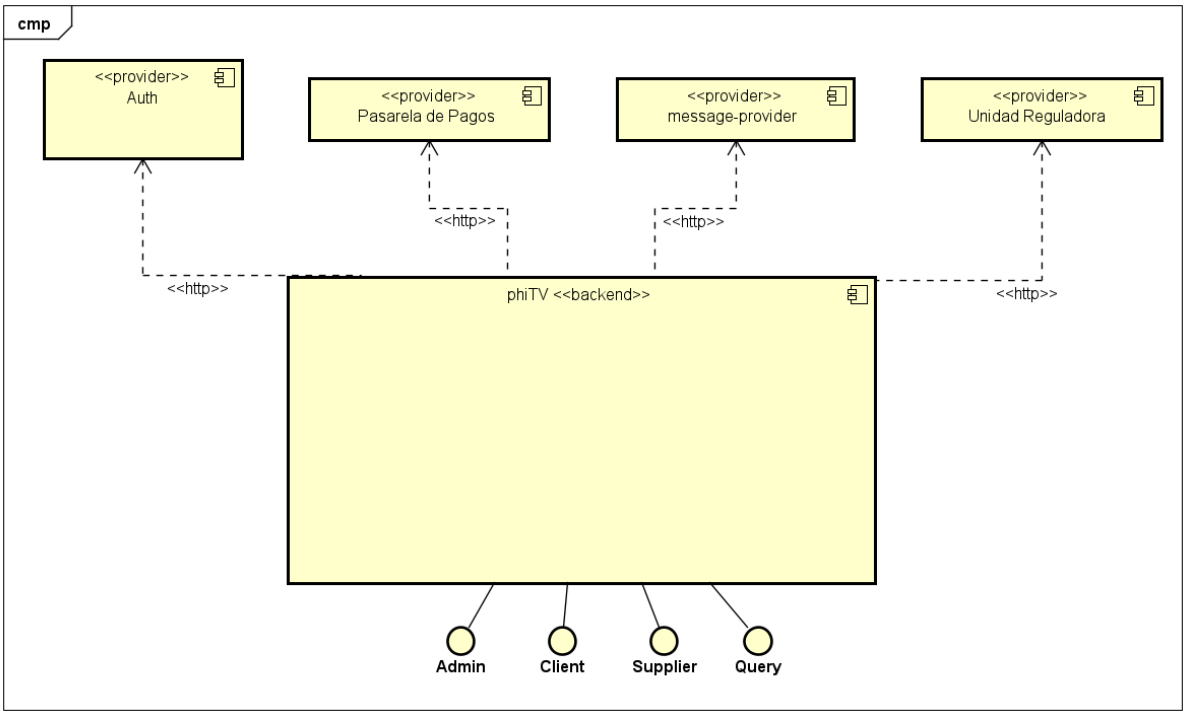
		esta información, así como reutilizar esta solución en otras aplicaciones, con el menor impacto posible en el código.
REQ 16 – Protección de datos y accesos a la plataforma	Seguridad	Se requiere un mecanismo para autenticar y autorizar a los usuarios (Administradores, Proveedores, Clientes) de la plataforma y de esta forma, proteger los endpoints y restringir su acceso según la especificación
REQ 17 - Manejo de carga	Performance Disponibilidad Escalabilidad	Se deberá lograr la mayor capacidad de procesamiento posible de solicitudes de acceso y procesamiento, sin pérdida de datos y logrando la mejor latencia posible. Se deben identificar los principales puntos que pueden llevar a pérdidas de acceso a eventos y procesamiento de información, y que deben estar disponibles en momentos de demanda pico. Se utilizará PM2 como herramienta principal para garantizar el cumplimiento de este requerimiento
REQ 18 - Información de auditoría	Seguridad	Se utiliza la táctica relacionada a Recuperarse de Ataques “Maintain audit trail” que consiste en mantener un registro de quien ejecuta cada acción y los resultados puede ser efectivo para buscar culpables y elaborar nuevas defensas. Tener este registro de información nos permitirá realizar

		auditorías de acceso de forma de identificar los accesos autorizados y no autorizados al sistema

3. Documentación de la Arquitectura

3.1 Diagrama de Contexto

Un diagrama de contexto es una vista de alto nivel de un sistema. Este representa al sistema como un componente y se visualizan todos los elementos con los que se interactúa. Nos permite comprender las interacciones entre el sistema y su entorno, identificar los límites del sistema y sus interfaces externas, así como comprender cómo se conecta con otros sistemas o actores externos.



3.2 Vista de Módulos

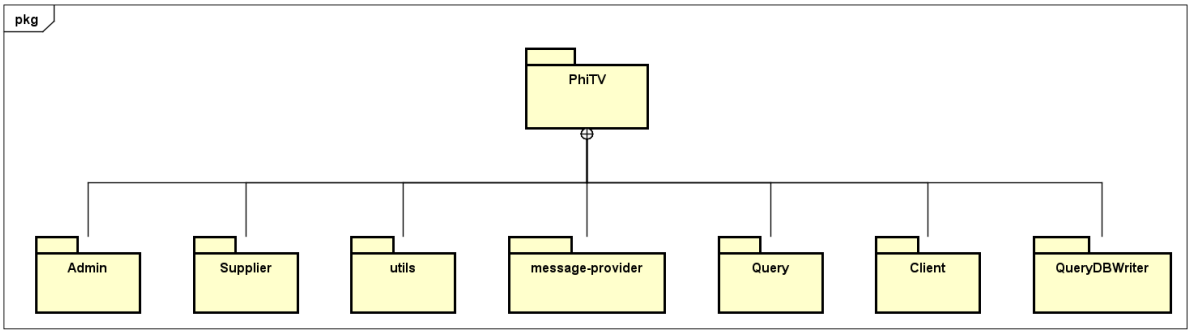
3.2.1 Vista de Descomposición

En esta sección, el objetivo es utilizar diagramas para comprender la estructura interna del sistema y la organización de sus componentes. A continuación, se presenta una lista

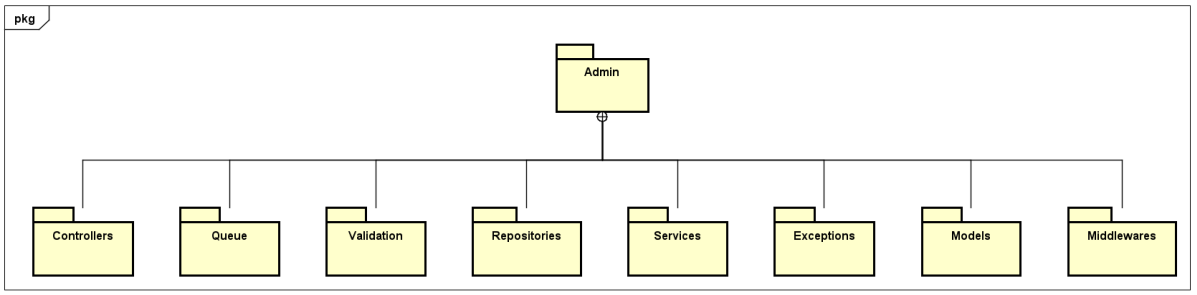
jerárquica de los diferentes componentes del sistema, con el fin de comprender la estructura del proyecto y de cada uno de sus componentes.

3.2.1.1 Representación primaria

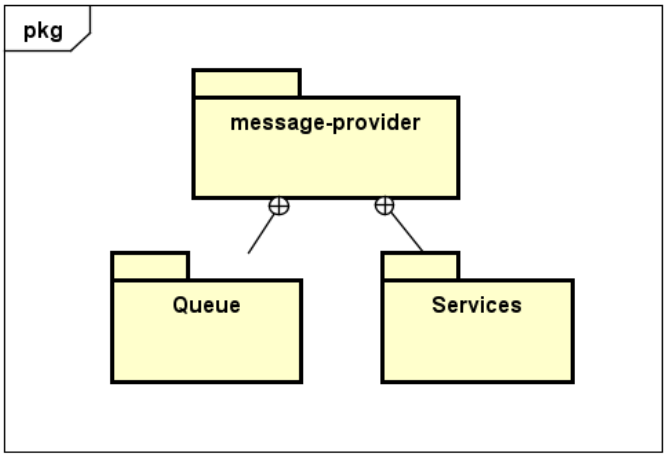
PhiTV:



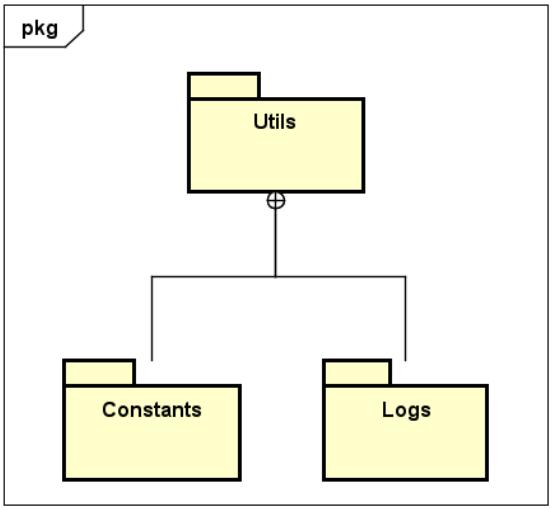
Admin:



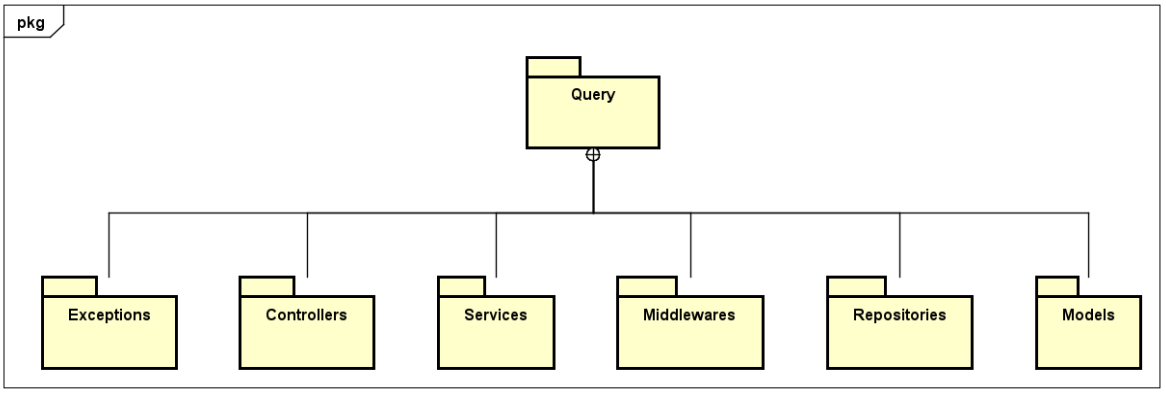
message-provider:



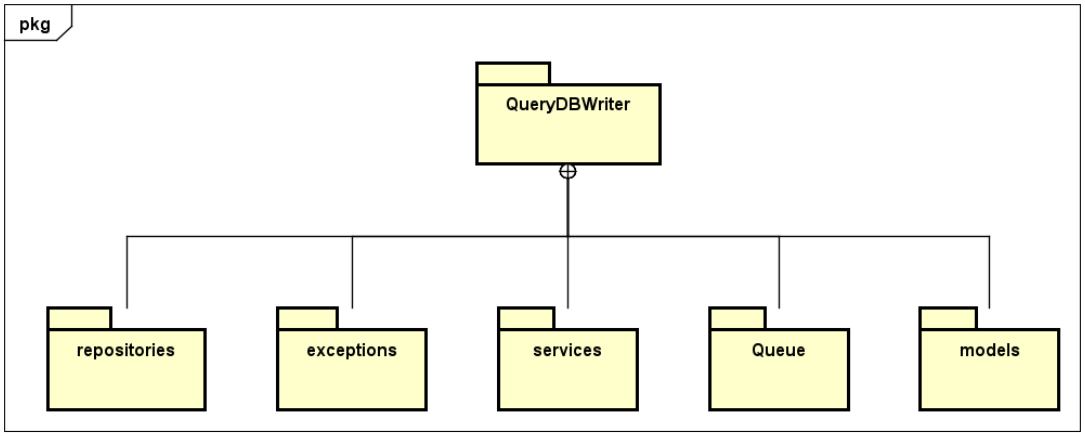
Utils:



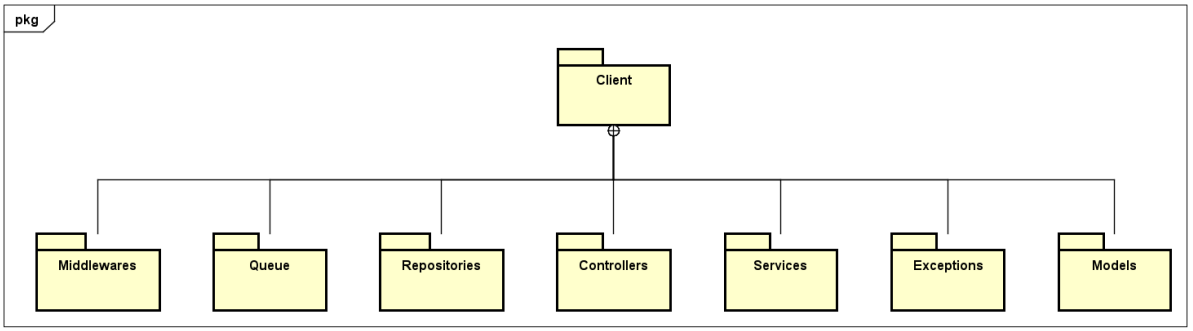
Query:



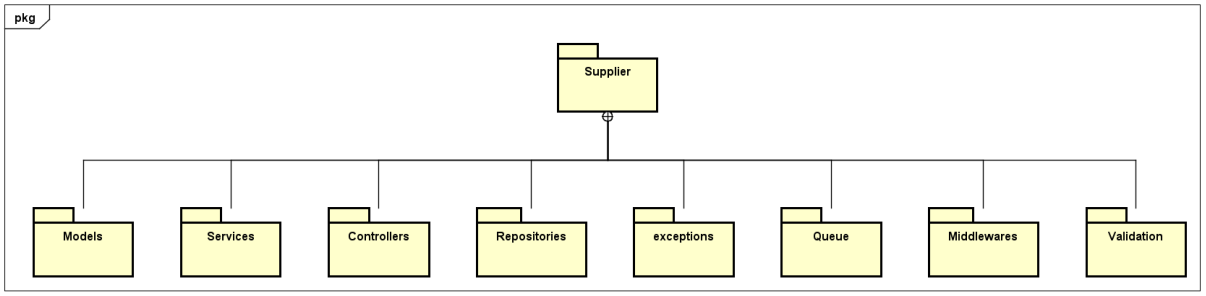
QueryDBWriter:



Client:



Supplier:



3.2.1.2 Catálogo de elementos

A continuación pasaremos a describir los diferentes elementos que forman parte de la plataforma PhiTV.

Elemento	Responsabilidad
PhiTV	Módulo Padre. Sería como la carpeta del repositorio, engloba todos los elementos.
Admin	Módulo con las consultas para los Administradores. Contiene los paquetes Controllers, Middlewares, Models, Queue, Repositories, Services, Validation
message-provider	Módulo encargado de manejar el envío de emails. Se implementa un Adapter a modo de interfaz para estar abierto a cambios.
Utils	Guarda constantes definidas globalmente para el sistema. Incluye también la implementación del logger, acompañado de los archivos de logs.
Query	Módulo que contempla consultas particulares para cada tipo de Actor (Administrador, Proveedor, Cliente) y se estructura de manera similar al Módulo Admin.
QueryDBWriter	Módulo que escribe en la base de datos de lectura específica para realizar las queries. Desencola de una cola y escribe en la base de datos.
Client	Módulo con las consultas para el Cliente. Involucra los paquetes Controllers, Middlewares, Models, Queue, Repositories, Services
Supplier	Módulo que se encarga de las consultas de los Proveedores. Contiene los paquetes Controllers, Middlewares, Models, Queue, Repositories, Services, Validation.

3.2.1.3 Decisiones de diseño

Decisión: Elección de una arquitectura basada en servicios (Service Based Architecture) para la creación de una plataforma PhiTV

Contexto y problema:

PhiTV busca ser una plataforma de streaming de alto rendimiento y escalable, que se adapte rápidamente a la creciente demanda de contenido y a las fluctuaciones en el tráfico. PhiTV debe ser capaz de ofrecer un servicio ininterrumpido con una mínima latencia.

Opciones:

- Arquitectura basada en Servicios
- Arquitectura basada en Capas
- Arquitectura basada en Microservicios

Justificación:

Hemos decidido utilizar una Arquitectura basada en Servicios (SBA). Aunque la arquitectura basada en microservicios también proporciona beneficios en cuanto a escalabilidad y despliegue continuo, no optamos por la misma ya que no nos sentíamos lo suficientemente cómodos como para estar construyendo la plataforma utilizando una arquitectura que apenas conocíamos. Por otro lado, una arquitectura basada en capas fue algo que íbamos a adoptar más a nivel de cada servicio en específico, no en la arquitectura general, por lo que no lo consideramos.

En particular, nuestra manera de elegir los servicios fue según el actor que iba a realizar las funcionalidades. En PhiTV se presentan tres actores, Administrador, Proveedor y Cliente. La excepción a esta regla fue la creación de un servicio Query que contemplaba consultas que iban a ser realizadas por cada uno de los distintos actores mencionados anteriormente.

Aunque en el ámbito real, en esta arquitectura, los servicios se despliegan cada uno con por su nodo separado, al haber sido algo que desarrollamos a nivel académico, solamente se despliega en un solo Nodo que sería las computadoras de los integrantes del equipo. Por lo que en particular, en la arquitectura de este proyecto, no se podría ilustrar de forma correcta que favorecemos la Deployabilidad como tal, pero en otras circunstancias esta arquitectura sería distinta.

La SBA nos permite tener módulos (servicios) que encapsulan una funcionalidad de negocio en específico, lo cual nos ayuda a mantener el código más organizado y facilita el

mantenimiento. Cada uno de estos servicios puede ser escalado de forma independiente, permitiendo un manejo más eficiente de los recursos. Además, en caso de falla de un servicio, el efecto en los demás será mínimo, manteniendo así una alta disponibilidad de la plataforma.

Atributo de calidad y tácticas que cumplen:

Escalabilidad: La Arquitectura basada en Servicios permite que los servicios se escalen independientemente según las demandas de carga, lo que optimiza el uso de recursos.

Modularidad: Con la SBA, cada servicio busca independiente y encapsulamos funcionalidades del negocio específicas por actor, facilitando el mantenimiento y comprensión del código.

Disponibilidad: En caso de una falla en un servicio, la SBA permite que los demás servicios sigan funcionando con poca o ninguna interrupción, garantizando una alta disponibilidad. Por ejemplo, si por alguna razón las consultas que puede realizar un Administrador se ven afectadas, no tiene por qué pasarle a un Cliente y/o Proveedor, y viceversa.

Despliegue: Idealmente cada servicio se desplegará independientemente, pero por cuestiones de tecnología y siendo el objetivo de este obligatorio académico, se despliega toda la aplicación con sus servicios en un solo lugar.

Decisión: Loggers: Registro de auditoría para mantener un historial de eventos y recuperarse de ataques

Contexto y problema:

La seguridad es un atributo de calidad crítico en el sistema, y es importante contar con mecanismos para recuperarse de ataques y comprender las acciones realizadas en el sistema en caso de incidentes o vulnerabilidades. Con el fin de abordar este problema, se consideró la implementación de un registro de auditoría para mantener un historial de eventos y garantizar la capacidad de recuperación.

Opciones:

Utilizar logs para registrar eventos y acciones en el sistema.

Justificación:

Se decidió utilizar logs para registrar eventos y acciones en el sistema, lo que proporciona un mecanismo de auditoría y permite mantener un historial detallado de las operaciones realizadas en el sistema. Esta decisión se basa en los siguientes puntos relacionados con el atributo de calidad de seguridad y la capacidad de recuperación:

Rastreo y detección de actividades autorizadas y desautorizadas: Al registrar eventos y acciones en los logs, se proporciona un medio para rastrear y detectar actividades autorizadas y desautorizadas. Los registros de auditoría pueden ser utilizados para identificar y analizar posibles comportamientos anómalos, lo que facilita la respuesta y la toma de medidas correctivas.

Análisis de incidentes: En caso de un ataque o incidente de seguridad, los logs de auditoría son una valiosa fuente de información para realizar investigaciones y análisis de incidentes. Estos registros permiten reconstruir los eventos y comprender las acciones realizadas durante el incidente, lo que facilita la identificación de la causa raíz y la implementación de medidas correctivas para evitar futuros ataques.

Utilizando logs como registro de auditoría, el sistema puede mantener un historial de eventos y acciones, lo que proporciona una capa adicional de seguridad y la capacidad de recuperarse de ataques. Los registros de auditoría permiten rastrear y detectar actividades sospechosas, realizar investigaciones en caso de incidentes y cumplir con los requisitos de seguridad y regulaciones. Además, en el caso de ocurrir una falla o cualquier tipo de error, el sistema es capaz de proveer toda la información necesaria que permita hacer un diagnóstico rápido y preciso sobre las causas.

Atributo de calidad y tácticas que cumplen:
Seguridad → Recover from attacks → Audit

Decisión: Loggers: utilización de librería y almacenamiento

Contexto y problema:
Los loggers eran una parte fundamental del sistema. No solo por algunos requerimientos específicos, sino que también debido al caso de ocurrir una falla o cualquier tipo de error, en donde es imprescindible que el sistema provea toda la información necesaria que permita a la empresa hacer un diagnóstico rápido y preciso sobre las causas. Es por esta razón que se evaluó cómo crear y administrar estos loggers. La razón por la cual se utilizaron los

loggers como manera para recuperarse de ataques se encuentra en otra decisión diferente. Aquí solo se discute cómo manejarlos

Opciones:
Guardar loggers en la base de datos
Utilizar una librería externa (Winston)

Justificación: Se decidió utilizar la librería externa Winston para administrar y almacenar los loggers en un archivo en lugar de guardarlos directamente en la base de datos. La elección se basó en los siguientes factores:

Evitar impacto en el rendimiento de la base de datos: Al almacenar los loggers en un archivo utilizando Winston, se evita agregar una escritura adicional a la base de datos por cada registro de log. Esto es beneficioso en términos de rendimiento, ya que no se requiere realizar operaciones de escritura en la base de datos, lo que podría afectar posibles lecturas y degradar el rendimiento general del sistema.

Gestión adecuada del archivo de registros: La librería Winston ofrece funciones para gestionar eficientemente el archivo de registros, lo que garantiza que los registros se almacenen correctamente y puedan ser accedidos y analizados de manera conveniente. Hasta el momento, no se han identificado fallas o problemas significativos en la gestión del archivo por parte de la librería.

La elección de utilizar la librería Winston para administrar y almacenar los loggers en un archivo se consideró la opción más adecuada, ya que proporciona la información necesaria en caso de fallas sin afectar negativamente el rendimiento de la base de datos. La gestión adecuada del archivo de registros por parte de la librería brinda confiabilidad en el almacenamiento de los logs.

Atributo de calidad y tácticas que cumplen:
Performance

Decisión: Pipes and Filters como patrón para validación de eventos
Contexto y problema:
Antes de que un evento pueda ser vendido, debe ser aprobado. Hay dos formas de autorización: manual y automática. Sin importar el tipo de autorización, las validaciones que

se deben realizar son las mismas. Los requerimientos especifican que la plataforma debe permitir agregar, modificar y eliminar fácilmente las validaciones de los eventos.

Opciones:
Pipes and Filters

Justificación:
El patrón Pipes and Filters es la elección adecuada para cumplir con la facilidad de agregar, modificar y eliminar las validaciones requeridas. Este patrón utiliza un flujo de procesamiento en el que se aplican funciones de validación paso a paso. En nuestro caso, mantenemos un array llamado "Errores" que registra información sobre los fallos en caso de que las validaciones no pasen. Si el array está vacío, significa que las validaciones han sido exitosas. Agregar y quitar validaciones es sencillo, ya que se puede hacer con una sola línea de código, y la creación de nuevos métodos es independiente de los existentes, lo que permite un sistema flexible y desacoplado.

El uso del patrón Pipes and Filters cumple con el atributo de calidad de modificabilidad, ya que permite adaptar rápidamente las validaciones a medida que cambian los requisitos. Este patrón proporciona una arquitectura desacoplada, lo que facilita el mantenimiento y la evolución del sistema con el tiempo.

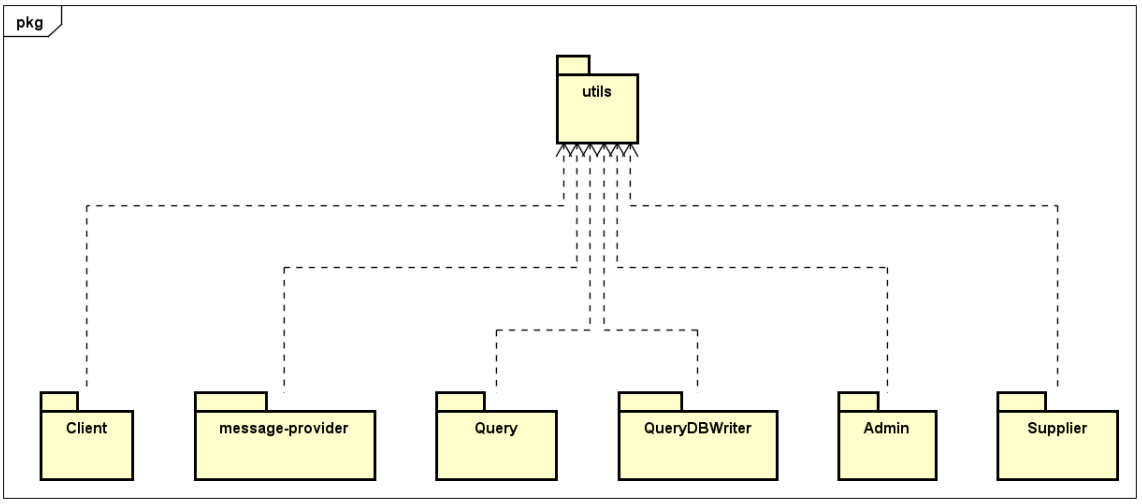
Atributo de calidad y tácticas que cumplen:
Modificabilidad → Pipes & Filters

3.2.2 Vista de Usos

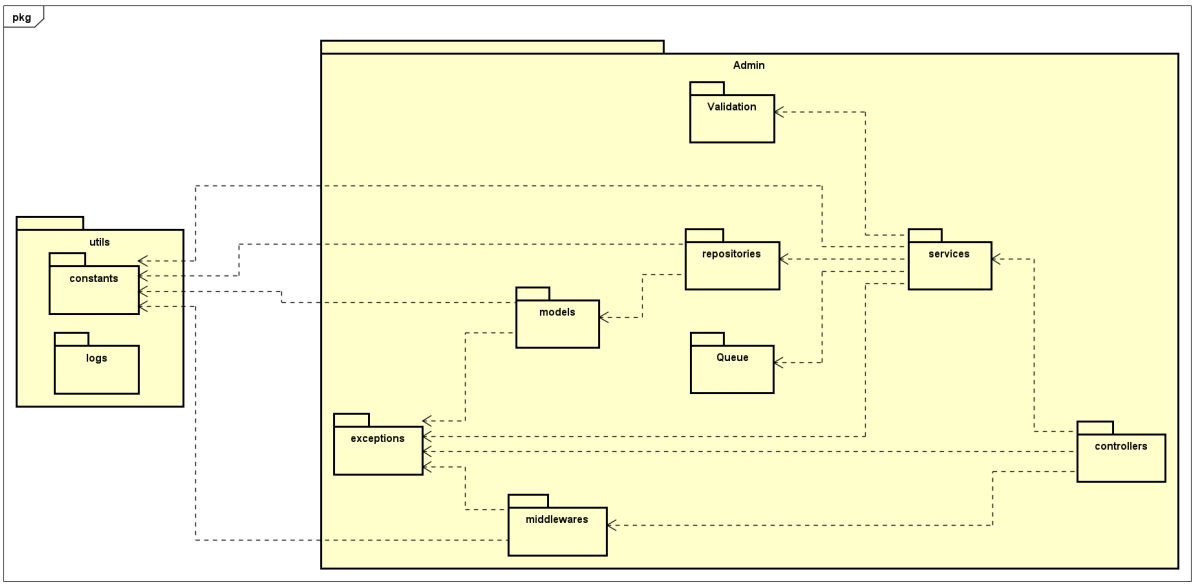
3.2.2.1 Representación primaria

Se incluyen las representaciones a nivel de cada módulo internamente y también una a nivel global de la plataforma

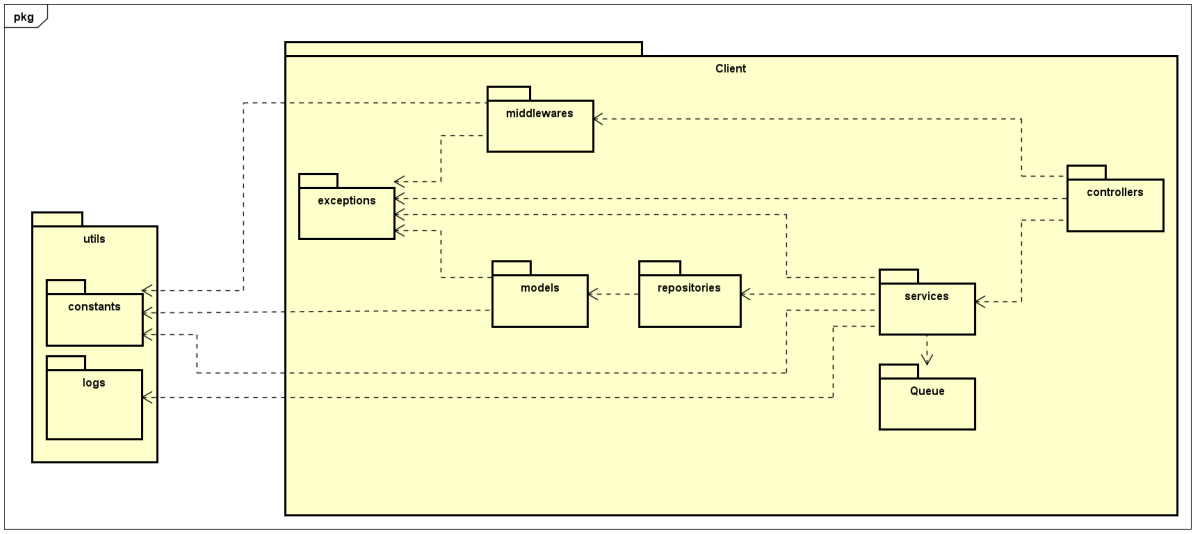
PhiTV:



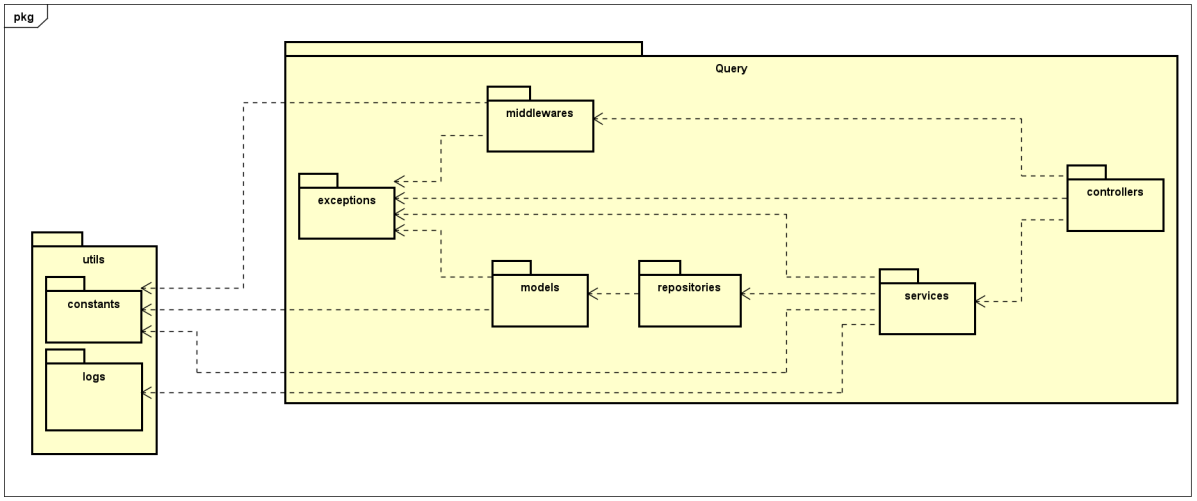
Admin:



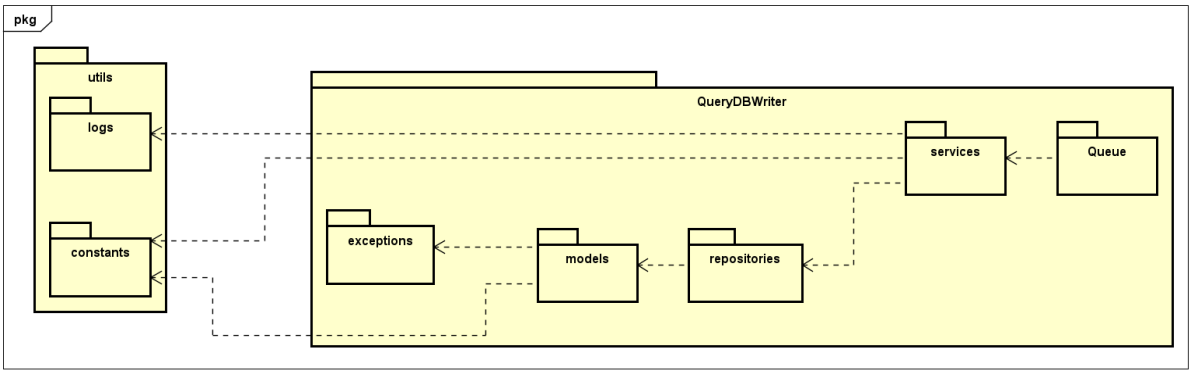
Client:



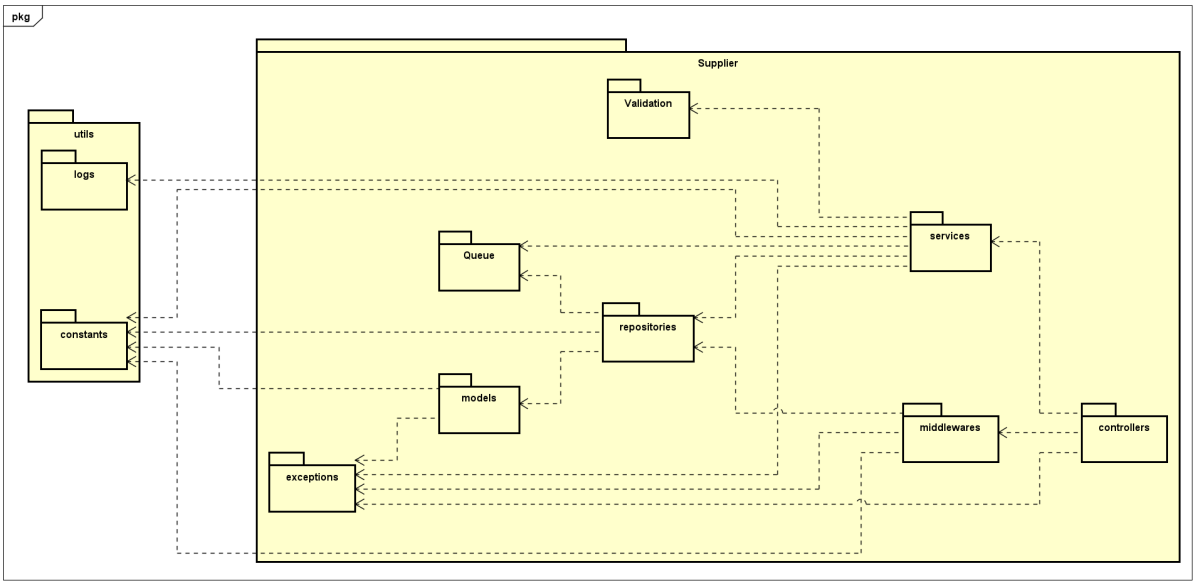
Query:



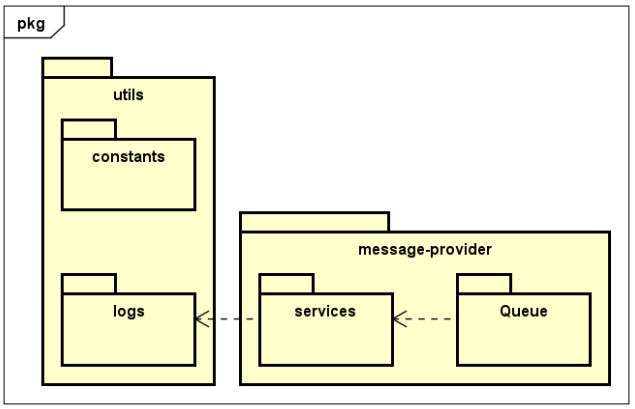
QueryDBWriter:



Supplier:



message-provider:



3.2.2.2 Catálogo de elementos

Elemento	Responsabilidad
controllers	Contiene los controllers y router que manejan los diferentes endpoints a través de los cuales los usuarios pueden llevar a cabo las funcionalidades. Delegan el trabajo a los servicios.
exceptions	Contiene las excepciones utilizadas para el manejo de errores dentro del módulo.
middlewares	Maneja los middlewares que se utilizan

	para la autorización de los usuarios a la hora de acceder a las funcionalidades. Pueden también incluir la subida del video.
models	Son las estructuras que siguen los elementos que guardamos en la base de datos.
Queue	Incluyen las diferentes colas de trabajo utilizadas para el manejo de tareas asíncronas. Se utilizan para el envío de correos, el procesamiento de videos, de consultas, etc.
repositories	Manejan la persistencia de los datos, se encargan de interactuar con la base de datos.
services	Contienen nuestra lógica de negocio central, interactúan con los repositorios para cumplir con las llamadas de los controllers. En ciertos paquetes solamente emulan la ejecución de ciertos servicios como el envío de emails, etc.
Validation	Es el paquete que contiene un Pipes and Filters con diferentes Filters, estos pueden ser desde validaciones a nivel de campos de un objeto, como hasta la interacción con la Pasarela de Pagos y/o la Unidad Reguladora para la autorización de los eventos.
utils	Tanto un módulo del sistema como submódulo de otros. Contiene desde el módulo **constants** (diferentes constantes en el uso de ciertas funcionalidades. Se utiliza para estandarizar valores y evitar confusiones) de uso general en varios módulos, así como el módulo de **logs** que se encarga del manejo del logger para el registro de logs. También puede tener un uso específico como en el caso del módulo Auth que proporciona con uso de otros paquetes una manera para encriptar y comparar contraseñas.

3.2.2.3 Decisiones de diseño

Decisión: Implementación de autenticación y autorización utilizando el patrón Federated Identity y JWT.

Contexto y problema:

La protección de datos y el control de acceso son requisitos fundamentales en términos de seguridad para cualquier sistema. Para abordar este problema, se consideró la necesidad de implementar una solución de autenticación y autorización eficaz y segura.

Opciones:

Utilizar el patrón Federated Identity en combinación con JWT.

Justificación:

La decisión final fue implementar el patrón Federated Identity junto con el uso de JWT para abordar los desafíos de autenticación y autorización en el sistema. Al utilizar el patrón Federated Identity, pudimos aprovechar un proveedor externo confiable para autenticar a los usuarios y proporcionar información de identidad. Una vez que el proveedor externo auténtica con éxito al usuario, se genera un token JWT que contiene la información de identidad necesaria.

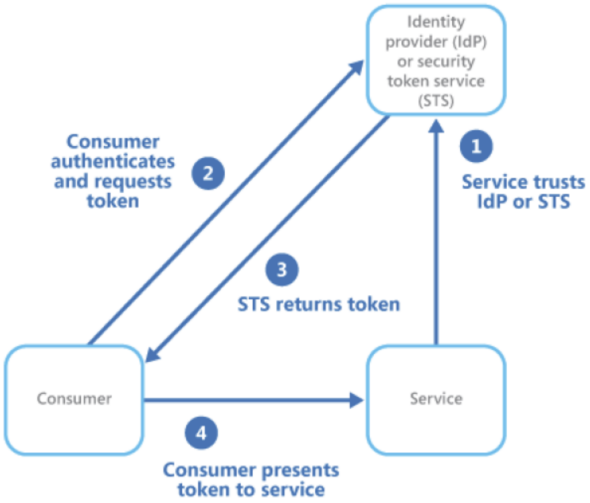
El token JWT se utiliza posteriormente para autenticar y autorizar al usuario dentro del sistema. Al verificar la validez y autenticidad del token JWT, el sistema puede garantizar que el usuario autenticado proviene de un proveedor confiable y no ha sido manipulado. Además, el token JWT puede contener información sobre los roles y permisos del usuario, lo que facilita la implementación de la autorización en el sistema.

Esta solución cumple con el atributo de calidad de seguridad, específicamente la táctica de resistir ataques al autenticar y autorizar actores. Al utilizar el patrón Federated Identity junto con JWT, se logra un sistema seguro y confiable para la autenticación y la autorización.

Además, la utilización de una base de datos independiente para almacenar la información de autenticación y autorización cumple con el requisito de limitar la exposición y dividir y distribuir los recursos críticos dentro del sistema.

Atributo de calidad y tácticas que cumplen:

AC Seguridad → Tactica Resistir Ataques → Authenticate y Authorize Actors
AC Seguridad → Resistir Ataques → Limitar exposición



<u>Decisión: Comunicación con APIs externas al sistema (y envío de mails)</u>

Contexto y problema: Como se indicó en la letra, se debía definir el protocolo de interacción entre la Pasarela de Pagos y la plataforma, así como el protocolo de interacción entre el servicio de la Unidad Reguladora y la plataforma. Además, se agrega también el caso de envío de mails. Aunque el mismo no es externo al sistema por letra, se utiliza una librería externa, en donde se debe de tener la posibilidad de modificar sin mayor problema.

Al tener que interactuar con sistemas externos a nuestro sistema, se busco la manera de adaptarse de la mejor forma a él. Tanto como para que sea fácil de cambiar en el futuro (en caso de que se use algún sistema distinto), como también para que sea fácil de integrar a nuestro proyecto, sin tener problemas de integrabilidad

Opciones:

Utilizar patrones de diseño (Adapter o Proxy)

Justificación: Se decidió utilizar el patrón de diseño Adapter para la comunicación con APIs externas al sistema, así como para el envío de correos electrónicos. Esta elección se basa en los siguientes factores relacionados con los atributos de calidad y las tácticas correspondientes:

Intermediario con APIs externas: El patrón Adapter actúa como un intermediario entre el sistema y las APIs externas, facilitando la comunicación y la integración sin que el sistema

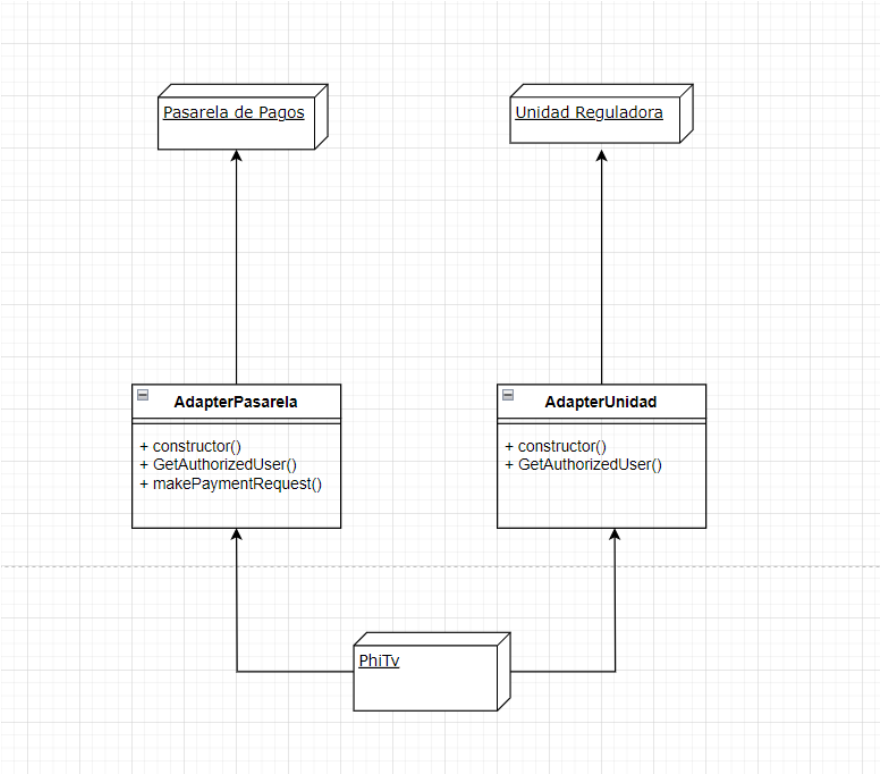
tenga que conocer los detalles específicos de cada API. Esto permite una mayor flexibilidad y adaptabilidad al interactuar con diferentes sistemas externos en el futuro, ya que se pueden crear adaptadores específicos para cada API sin afectar al resto del sistema.

Modificabilidad: Al utilizar el patrón Adapter, se busca reducir el acoplamiento entre el sistema y las APIs externas. Esto significa que los cambios en las APIs externas tendrán un impacto limitado en el sistema, ya que solo será necesario ajustar o crear nuevos adaptadores sin afectar el resto de la lógica del sistema. Esto facilita la modificabilidad del sistema a largo plazo, ya que los cambios en las APIs externas pueden manejarse de manera más eficiente y controlada.

Integrabilidad: El patrón Adapter permite limitar las dependencias directas del sistema en las APIs externas al encapsular la lógica de comunicación dentro de los adaptadores. Esto simplifica la integración de las APIs externas en el proyecto, ya que el sistema solo necesita interactuar con los adaptadores, sin preocuparse por los detalles específicos de implementación de cada API. Además, esta abstracción también facilita la sustitución de las APIs externas por otras en el futuro, ya que solo se necesitará crear nuevos adaptadores sin modificar la lógica principal del sistema.

La elección de utilizar el patrón Adapter cumple con los atributos de calidad de intermediario con APIs externas, así como con las tácticas de reducir acoplamiento y limitar dependencias para mejorar la modificabilidad e integrabilidad del sistema. Al utilizar adaptadores, se logra una mayor flexibilidad, adaptabilidad y mantenibilidad en la comunicación con APIs externas y el envío de correos electrónicos.

El siguiente diagrama ejemplifica el uso de adapters para la comunicación con las apis externas:



Atributo de calidad y tácticas que cumplen:

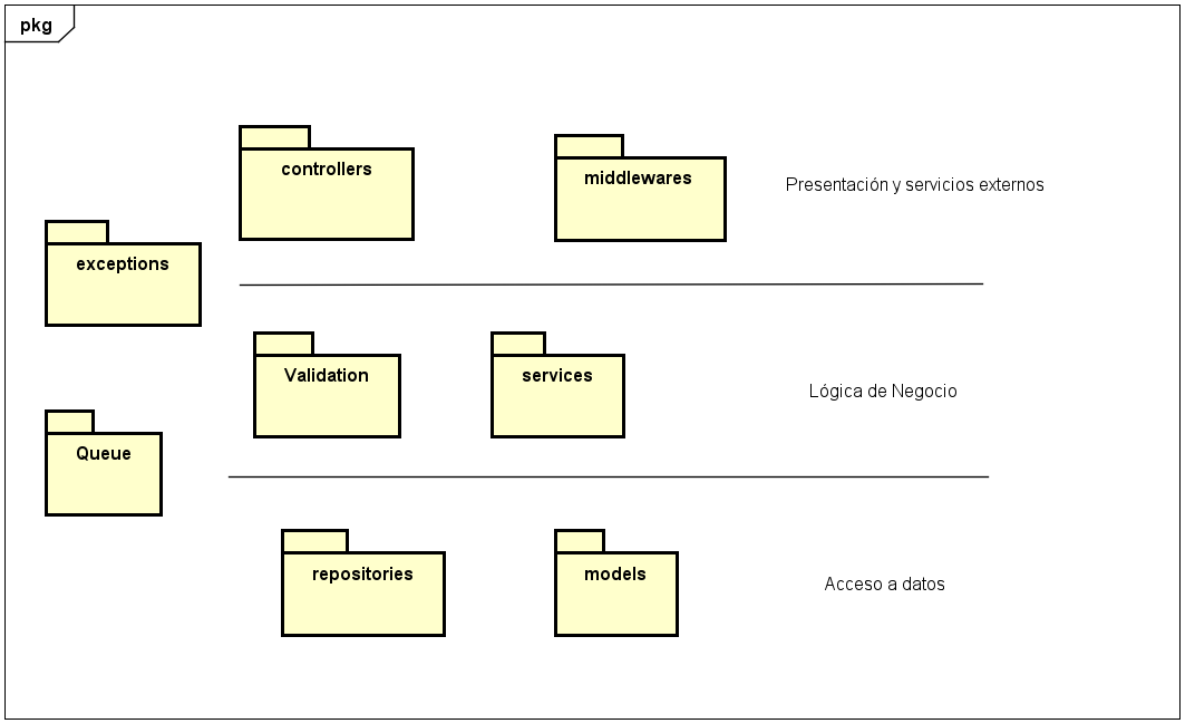
Modificabilidad → Reducir acoplamiento → Intermediario

Integrabilidad → Limitar dependencias → Encapsular

3.2.3 Vista de Layers

3.2.3.1 Representación primaria

Al nivel más alto del proyecto, no existe un patrón de Layers como tal, ya que se dividieron los módulos en base a los actores y/o servicios externos que participarán en el funcionamiento de la plataforma, por lo que en realidad tomaremos el módulo Admin para dar una representación de esta vista ya que es el más explicativo en nuestra opinión.



3.2.3.3 Decisiones de diseño

Decisión: Segmentación por capas

Contexto y problema:
A la hora de comenzar con el desarrollo del sistema nos encontramos con la necesidad de tomar una decisión de cómo organizar los componentes del sistema de forma que beneficiara a la mantenibilidad del sistema

Justificación:
Con el objetivo de mejorar la mantenibilidad, cohesión y reutilización de los componentes del sistema, se ha optado por realizar una segmentación en capas en cada módulo. Esta estrategia permite un desarrollo independiente, con la menor interacción posible entre las diferentes partes del sistema. Para ello, hemos agrupado los componentes en capas: Presentación y servicios externos, Lógica de negocios y Acceso a datos. Consideramos que estas capas representan adecuadamente la funcionalidad de las carpetas que las componen. Los elementos de cada capa se comunican de forma estricta con sus contrapartes y con los elementos de las capas inferiores, lo que nos permite aumentar la cohesión y reducir el acoplamiento. Por ejemplo, los componentes de la capa de acceso a datos no acceden a la capa de Presentación. Sin embargo, existen casos particulares, de módulos y carpetas como “Queue” o “exceptions”, los cuales no se adaptan a las capas

propuestas. En estos casos, estos módulos funcionan más como utilidades que utilizan todas las capas del sistema en lugar de estar estrictamente segmentados en una sola capa.

Especificación de la funcionalidad de cada capa:

- Presentación y servicios externos: Esta capa se encarga de manejar la interfaz de usuario y la comunicación con servicios externos. Gestiona los diferentes eventos del sistema y proporciona respuestas.
- Lógica de negocios: Aquí se encuentra la lógica principal del sistema. Los controladores delegan el trabajo a los servicios correspondientes, que interactúan con los repositorios y realizan las operaciones necesarias para cumplir con las reglas y procesos de negocio.
- Acceso a datos: Esta capa se encarga del acceso y la gestión de los datos. Proporciona los mecanismos necesarios para interactuar con la base de datos u otras fuentes de almacenamiento de datos.

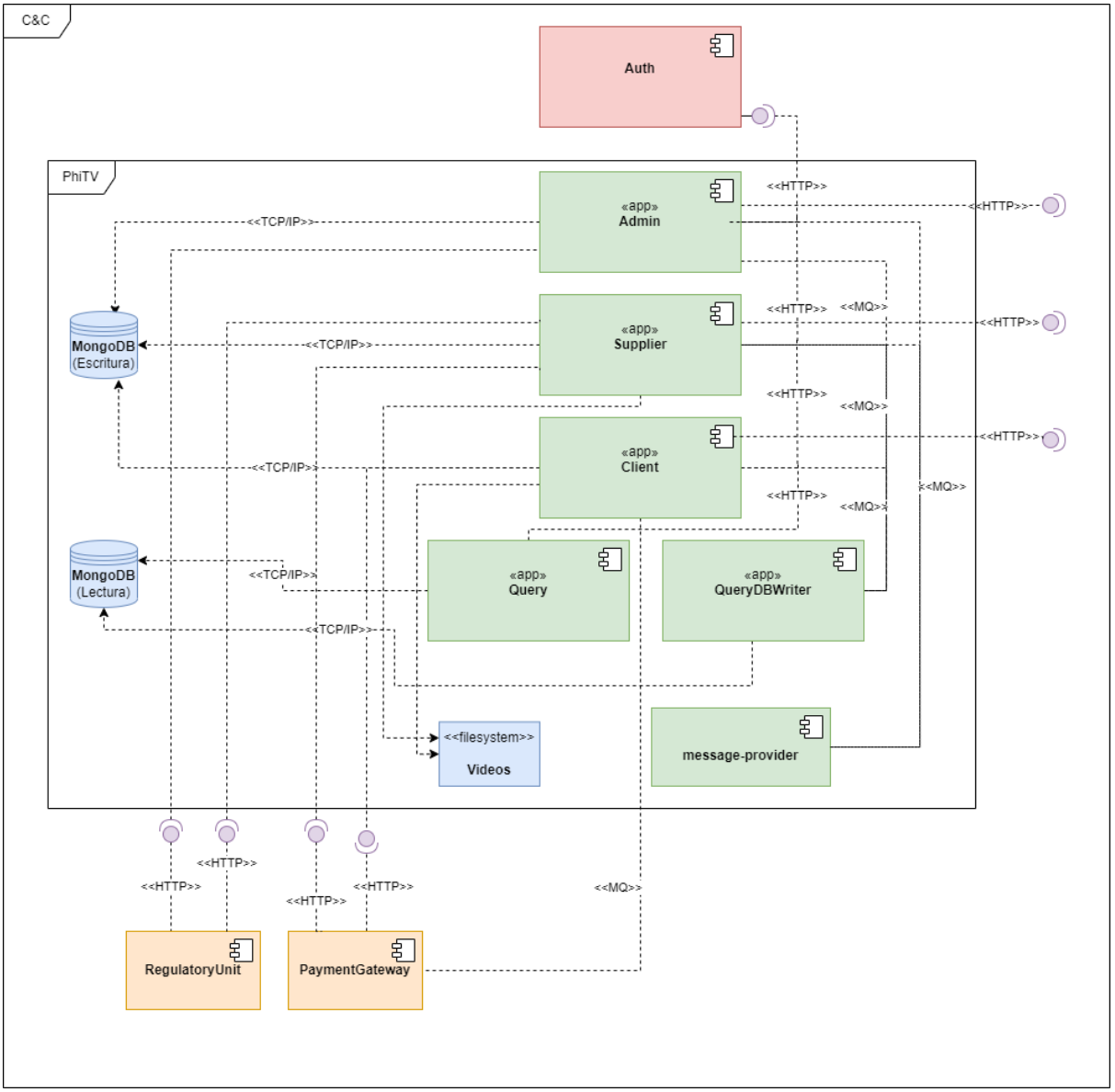
Atributo de calidad y tácticas que cumplen:
Modificabilidad → Reducir el Acoplamiento → Aumentar lo Cohesion → Layers

3.3 Vista de componentes y conectores

Esta sección aborda las vistas de componentes y conectores que se consideraron relevantes para comunicar la visión del sistema durante su tiempo de ejecución. A continuación, nos enfocaremos en resaltar los componentes y conectores más relevantes que reflejan la arquitectura en tiempo de ejecución. Esta representación arquitectónica nos permitirá comprender cómo se comunican y se relacionan los componentes durante la operación del sistema.

Es importante mencionar que todos los módulos siguen una estructura similar a la presentada anteriormente en la vista de descomposición, con componentes como controladores, servicios, repositorios, modelos y entidades. Debido a esta similitud en la estructura, resulta redundante incluir todos los componentes en el siguiente diagrama.

3.3.1 Representación primaria



3.3.2 Catálogo de elementos

Componente/Conector	Tipo	Descripción
Auth	Componente	Encargado de la autenticación/autorización de los diferentes usuarios que interactúan con la plataforma.
Admin	Componente	Concentra las consultas relacionadas al rol de Administrador

Supplier	Componente	Concentra las consultas relacionadas al rol de Proveedor
Client	Componente	Concentra las consultas relacionadas al rol de Cliente
Query	Componente	Concentra consultas por cada tipo de rol
QueryDBWriter	Componente	Se encarga de actualizar la base de datos de lectura con nueva información
message-provider	Componente	Se encarga del envío de emails a ciertos actores dependiendo de la situación
RegulatoryUnit	Componente	Es un proveedor de servicios externo que se encarga de autorizar a los proveedores a transmitir eventos.
PaymentGateway	Componente	Es un proveedor de servicios externo que se encarga de realizar los pagos de los Proveedores a la empresa.
MongoDB (Lectura)	Componente	Base de datos optimizada para las queries. Tiene modelos parecidos a los de la escritura, pero optimizados para hacer las consultas más rápido. Guarda solamente la información mínima e indispensable para responder las queries.
MongoDB (Escritura)	Componente	Base de datos “principal” del sistema. Toda la información se guarda dentro de esta base de datos.
HTTP	Conector	Protocolo utilizado para la comunicación con los componentes del sistema y entre ellos también.
TCP/IP	Conector	Protocolo utilizado para la comunicación con las bases de datos correspondientes.

MQ	Conector	Sistema de mensajería que permite la comunicación asíncrona entre diferentes componentes de un sistema distribuido. Permite el envío y la recepción de mensajes de manera eficiente y escalable, facilitando la integración de aplicaciones y la transmisión de datos entre ellas.

3.3.3 Decisiones de diseño

Decisión: Implementación de una base de datos adicional para consultas (Queries) utilizando el patrón CQRS (Command Query Responsibility Segregation).

Contexto y problema:

Dentro de los requisitos del sistema, se especifica la necesidad de realizar consultas con un tiempo de respuesta inferior a 5 segundos. Es necesario encontrar una táctica de rendimiento que permita agilizar estas consultas y cumplir con este requisito.

Opciones:

Implementar el patrón CQRS.
Utilizar Redis como caché para consultas.

Justificación:

La decisión final fue implementar el patrón CQRS para abordar el problema de las consultas y su rendimiento. Esta elección se basa en varias razones. En primer lugar, el patrón CQRS nos permite tener una base de datos adicional separada para lecturas, lo que significa que podemos tener bases de datos separadas para operaciones de escritura y operaciones de lectura. Esto evita bloqueos de escritura y optimiza la velocidad de las consultas al leer datos de la base de datos de lectura.

La base de datos de lectura está diseñada específicamente para optimizar las consultas, lo que significa que los modelos de datos se estructuran de manera que solo se almacene la información relevante para las consultas. Por ejemplo, en comparación con la base de datos principal que contiene fotos y videos, la base de datos de lectura puede omitir estos datos si no son necesarios en las consultas.

La elección de una base de datos NoSQL como MongoDB se debe a su flexibilidad y velocidad. MongoDB es capaz de manejar consultas rápidamente y se adapta fácilmente a nuevos formatos de consulta en el futuro. Esta flexibilidad es esencial para mantener la escalabilidad y el rendimiento a medida que se agregan nuevas consultas al sistema. Sin embargo, consideramos que, pese a estas características, quizás la mejor opción debería haber sido SQL, debido a su alta performance al momento de realizar Queries justamente. Sin embargo, esto nos creaba un problema de modelos (en donde tenemos una base de escritura en Mongo y otra de Lectura en SQL), lo que nos dificultaba el desarrollo en relación al poco tiempo que teníamos. Una mejora a futuro podría ser cambiar esta base de datos de lectura para que sea SQL y así aumentar aún más la performance, ya que sabemos que la decisión ideal debería haber sido SQL.

La carga de datos en la base de datos de lectura se realiza a través de una cola (Queue). Esta cola se encola cuando se agrega un nuevo dato a la base de datos de escritura, y luego se desencola y se agrega a la base de datos de lectura en un proceso llamado QueryDBWriter. Esta estrategia asincrónica evita retrasos al agregar nueva información, ya que no es necesario esperar a que se escriba en ambas bases de datos de forma sincrónica.

En resumen, la implementación de una base de datos adicional para consultas utilizando el patrón CQRS cumple con el atributo de calidad de rendimiento al gestionar los recursos de manera eficiente y al introducir concurrencia en las operaciones de lectura y escritura. Esta solución garantiza consultas rápidas y optimizadas, mejorando el rendimiento del sistema en general.

Atributo de calidad y tácticas que cumplen:
Performance → Gestionar Recursos → mantener múltiples copias de computo
Performance → Gestionar Recursos → Introducir concurrencia

Decisión: Utilizar una cola (Queue) para procesar los pagos.

Contexto y problema:
Dentro del sistema, se requiere interactuar con una API de pagos externa. Dado que esta API es un sistema externo sobre el cual no tenemos control directo, puede haber situaciones de contención donde la API de pagos demore en responder. Esto puede resultar en bloqueos de miles de transacciones pendientes de pago, lo cual afectaría negativamente el rendimiento y la experiencia del usuario si se maneja de manera síncrona.

Opciones:
Realizar el pago de forma instantánea al hacer una compra de evento.
Agregar una cola (Queue) para el procesamiento de los pagos.

Justificación:
La decisión final fue implementar una cola (Queue) para el procesamiento de los pagos. Con esta solución, una vez que se realiza una compra de evento, se le informa al usuario que la compra está siendo procesada satisfactoriamente. Internamente, los datos del pago se encolan en la cola para su procesamiento posterior.

Este enfoque permite desacoplar el proceso de pago de la interacción directa con la API de pagos externa. Los pagos se encolan y se procesan de manera asíncrona, lo que evita bloqueos y posibles retrasos en la experiencia del usuario. Además, al utilizar una cola, se introduce concurrencia en el sistema, lo que mejora el rendimiento y la eficiencia en el procesamiento de los pagos.

Una vez que los pagos son desencolados y procesados, se notifica al usuario a través de un correo electrónico sobre el resultado de la transacción. En caso de que el pago se haya completado con éxito, se envía una confirmación al usuario. En caso de que haya algún problema durante el procesamiento del pago, también se notifica al usuario para que esté informado.

En resumen, al utilizar una cola para el procesamiento de los pagos, se cumple con el atributo de calidad de Performance al introducir concurrencia y gestionar eficientemente los recursos del sistema. Esto garantiza una mejor experiencia del usuario y evita bloqueos en caso de demoras en la API de pagos externa.

Atributo de calidad y tácticas que cumplen:
Performance → Gestionar Recursos → Introducir concurrencia

Decisión: Separar la escritura y lectura en la base de datos de lectura.

Contexto y problema:

En el contexto del proyecto, es necesario manejar una base de datos de lectura que será utilizada exclusivamente para realizar consultas. Sin embargo, también es necesario realizar operaciones de escritura en dicha base de datos, lo que plantea el problema de cómo gestionar estas operaciones.

Opciones:

Permitir que un mismo proyecto realice tanto la escritura como la lectura en la base de datos de lectura.
Separar la escritura y lectura en la base de datos de lectura.

Justificación:

La decisión final es separar la escritura y lectura en la base de datos de lectura. Esto se debe a las siguientes razones:

Al separar la escritura y lectura en bases de datos distintas, se logra aumentar la cohesión y modularidad del sistema. Esto significa que si se requiere modificar o actualizar las operaciones de escritura, se puede hacer sin afectar las consultas y viceversa. No es necesario detener el sistema de escritura mientras se realizan cambios en las consultas, lo que garantiza la disponibilidad continua del sistema, es decir, no es necesario bajar el sistema completo y frenarlo simplemente para hacer cambios en la lectura.

A partir de esto se logra una mayor modificabilidad y deployabilidad del sistema. Lo que garantiza un rendimiento óptimo en la realización de consultas y una mayor flexibilidad en el desarrollo y mantenimiento del sistema.

Atributo de calidad y tácticas que cumplen:

Modificabilidad → Incrementar cohesión → Dividir módulo
Deployabilidad → Manage Deployment Pipeline → Scale Rollouts

Decisión: Utilizar una cola (Queue) para procesar los emails.

Contexto y problema:

Dentro de los requerimientos del sistema, se especifica la necesidad de enviar emails frente a distintas situaciones. En nuestro caso particular, creamos un módulo llamado “message-provider”, que se encargaría de enviar dichos emails, utilizando la librería nodemailer. Sin embargo, el envío de emails puede llegar a significar un cuello de botella

bastante grande, ya que en una simple request pueden tener que enviar miles de emails simultáneamente.

Opciones:

- Realizar el envío de emails al momento de realizar la request.
- Agregar una cola (Queue) para el procesamiento de los emails.

Justificación:
La decisión final fue implementar una cola (Queue) para el procesamiento de los correos electrónicos. Con esta solución, cuando se realiza una solicitud para enviar correos a todos los compradores de un evento, los datos del envío se encolan para su procesamiento posterior. Dado que la librería utilizada (nodemailer) permite enviar correos de manera individual, es necesario ejecutar este proceso de forma asíncrona. Además, la diferencia de tiempo en la llegada de los correos electrónicos no supone un problema (es decir, si a una persona le llega minutos más tarde, no hay inconveniente).

Este enfoque permite desacoplar el proceso de envío de correos electrónicos. Los correos se enfilan en la cola y se procesan de manera asíncrona, evitando bloqueos y posibles retrasos en la experiencia del usuario. Al utilizar una cola, se introduce concurrencia en el sistema, lo que mejora el rendimiento y la eficiencia en el procesamiento de los correos electrónicos.

Una vez que la información de los correos electrónicos se desencola y se procesa, se envían los correos.

En resumen, al utilizar una cola para el procesamiento de los correos electrónicos, se cumple con el atributo de calidad de rendimiento al introducir concurrencia y gestionar eficientemente los recursos del sistema.

Atributo de calidad y tácticas que cumplen:
Performance → Gestionar Recursos → Introducir concurrencia

Decisión: Procesamiento de archivos de manera asíncrona

Contexto y problema:

Se debe implementar un sistema de procesamiento de archivos que permita un acceso rápido y disponible de los eventos para los clientes. El objetivo es reducir al mínimo el tiempo de acceso a los archivos.

Opciones:

- Delegar el procesamiento de archivos a un proyecto denominado "file-handler" utilizando una cola de mensajes (como Bull).
- Procesar el almacenamiento de archivos de forma asíncrona cuando se realiza la solicitud.

Justificación:

La decisión de cómo manejar el procesamiento de archivos se vio influenciada por consideraciones de tiempo y dificultad de implementación, lo que dificulta la satisfacción óptima de los atributos de calidad esperados. Como resultado, optamos por implementar la opción de procesamiento asíncrono de archivos dentro del mismo componente que maneja la solicitud, utilizando el middleware "upload" que utiliza Multer para almacenar archivos. Sin embargo, reconocemos que esta solución puede mejorarse mediante el desacoplamiento del procesamiento de archivos en un componente separado dentro del sistema. Esto beneficiaría la modificabilidad del sistema al permitir el encapsulamiento de la lógica relacionada con el procesamiento de archivos, eliminando dependencias en la implementación.

Además, consideramos beneficioso introducir colas de mensajes (queues) para encolar el procesamiento de archivos, de manera que los usuarios no tengan que esperar a que los archivos se almacenen por completo para continuar con la solicitud. Esta mejora nos permite lograr un sistema más eficiente en términos de performance mediante la implementación de la táctica de introducción de concurrencia, ya que al encolar las tareas aumentamos la capacidad de procesamiento sin afectar la latencia.

Con el fin de mejorar la disponibilidad de los archivos al solicitarlos, tomamos la decisión de almacenarlos en un filesystem, lo que permite un acceso rápido y confiable, proporcionando una mayor disponibilidad en comparación con otras opciones de almacenamiento.

Atributo de calidad y tácticas que cumplen:

Modificabilidad → Reducir acoplamiento → Encapsular

Performance → Gestionar Recursos → Introducir concurrencia

Disponibilidad

3.3.4 Comportamiento

Diagrama de interacción del proceso de logueo al sistema de un usuario

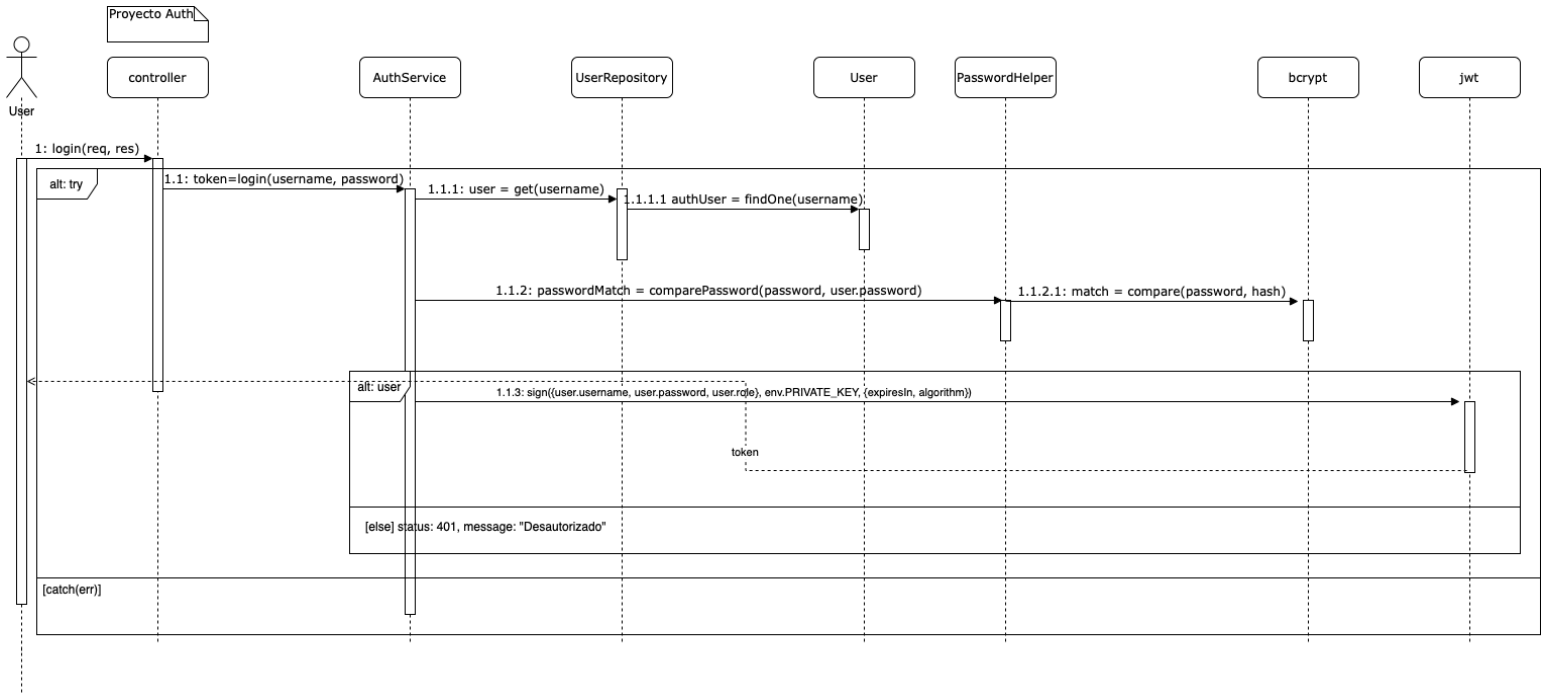
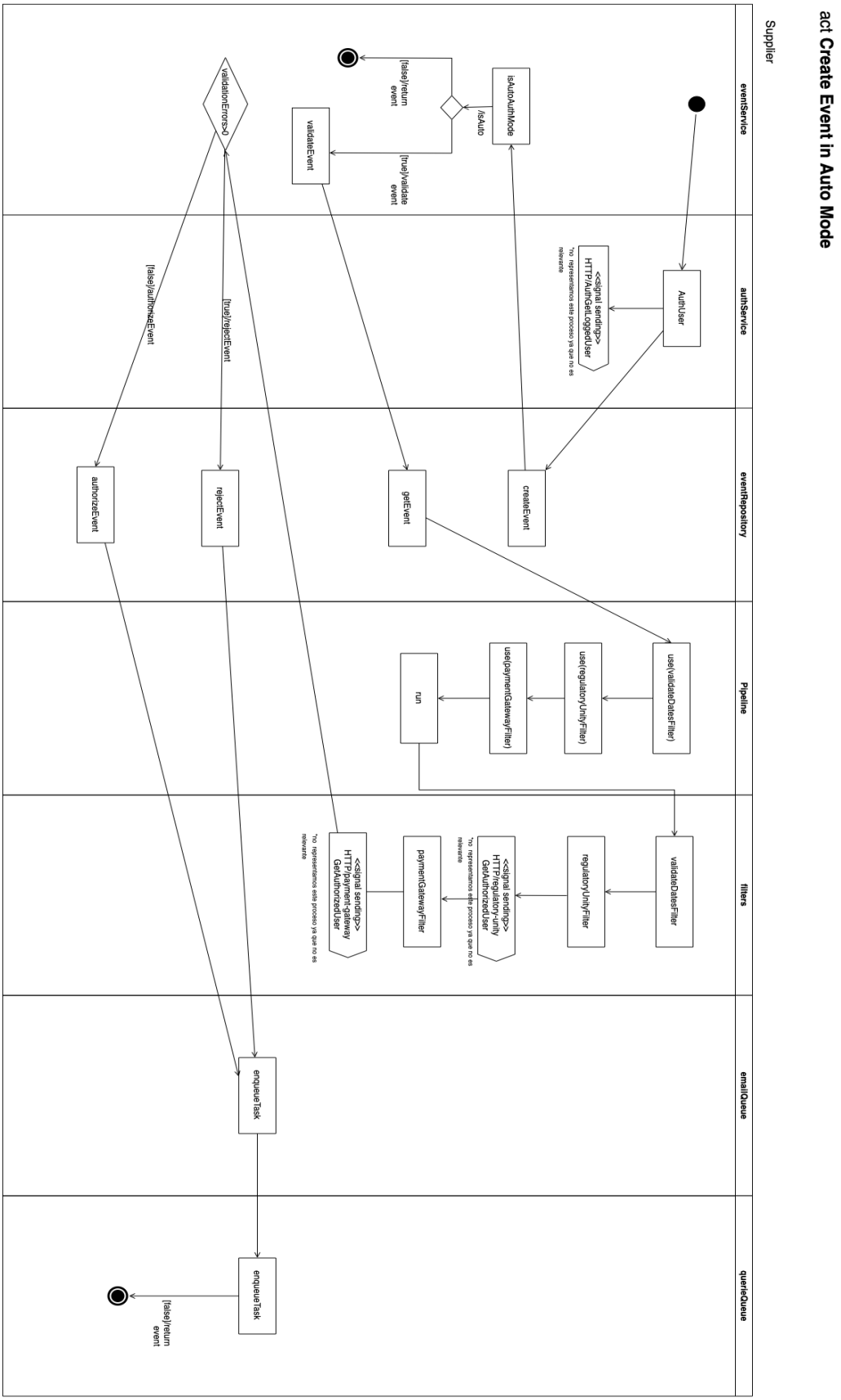


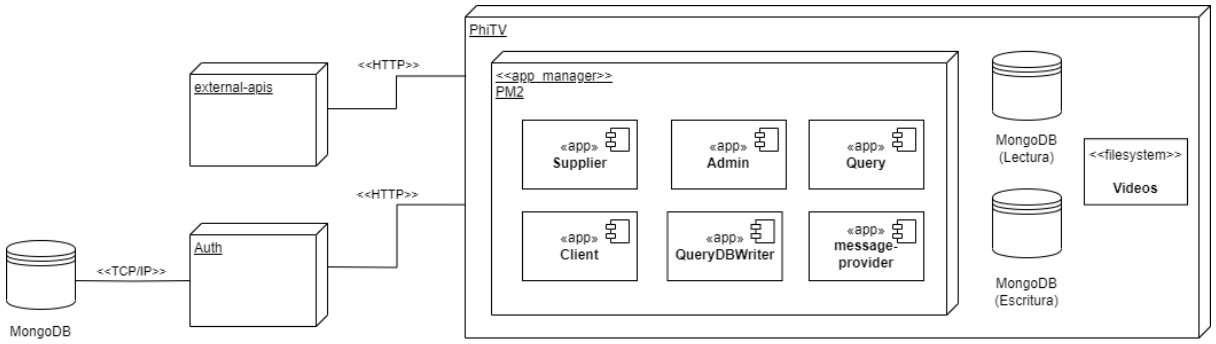
Diagrama de actividad del comportamiento de crear un evento dado un mode de autorización automático:



3.4 Vista de Despliegue

En la siguiente vista se presenta un diagrama de componentes que muestra los nodos, conectores y componentes intervinientes en el proyecto.

3.4.1 Representación primaria



3.4.2 Catálogo de elementos.

Elemento	Responsabilidad
PhiTV	Módulo Padre. Sería como la carpeta root del repositorio, engloba todos los elementos.
PM2	Gestor de procesos que se utiliza para administrar y mantener en funcionamiento aplicaciones en entornos de producción. Proporciona funciones como la supervisión, el reinicio automático y la administración de clústeres para garantizar la disponibilidad y estabilidad de las aplicaciones.
Mongo DB (de Auth)	Dentro de la base de datos de Auth se alojan las credenciales de los usuarios, y se utilizan librerías externas para la encriptación de las contraseñas. Las contraseñas se guardan encriptadas dentro de la base de datos, por lo que nunca se podrá ver la contraseña real, ni siquiera aquellas personas con acceso a la base de datos.
external-apis	Módulo que contiene a la PaymentGateway (Pasarela de Pagos) y RegulatoryUnit (Unidad Reguladora) que funcionan emulando el servicio que proveen, ambas con Adapters que permiten que estén abiertas a cambios.

Auth	Módulo que se encarga de la autenticación y autorización de los usuarios que vayan a acceder a la plataforma. Utiliza JWT como mecanismo para la protección y generación de tokens que permiten el acceso a las queries correspondientes. Maneja una base de datos separada a la principal donde se alojan las credenciales de los usuarios, se utilizan librerías externas para la encriptación de las contraseñas.
Mongo DB (de Lectura)	Base de datos optimizada para las queries. Tiene modelos parecidos a los de la escritura, pero optimizados para hacer las consultas más rápido. Guarda solamente la información mínima e indispensable para responder las queries.
Mongo DB (de Escritura)	Base de datos “principal” del sistema. Toda la información se guarda dentro de esta base de datos.

3.4.3 Decisiones de diseño

Decisión: Utilizar Mongo DB (en todas las bases del proyecto)

Contexto y problema:

Dentro del proyecto, el atributo de calidad más importante es el rendimiento, el cual se relaciona con un corto tiempo de respuesta, alta capacidad de procesamiento, ausencia de pérdida de datos y una latencia óptima. La elección de la base de datos es crucial, ya que es donde se almacena toda la información del sistema y es utilizada por la mayoría de los componentes.

Opciones:

SQL

No SQL

Justificación: Se eligió utilizar MongoDB como la base de datos principal para todas las bases de datos del proyecto, utilizando la librería Mongoose en Node. Esta elección se basa en los siguientes factores:

Performance: MongoDB, como base de datos NoSQL, ofrece consultas más rápidas y eficientes en comparación con las bases de datos SQL tradicionales. Esto cumple con los requisitos de corto tiempo de respuesta y alta capacidad de procesamiento mencionados en los requerimientos. Al utilizar MongoDB, se espera lograr una mejor latencia y un rendimiento óptimo en el sistema.

Disponibilidad: MongoDB es conocido por su tolerancia a fallos y su modelo flexible de datos. Esto favorece la disponibilidad del sistema, ya que permite mantener los datos accesibles incluso en situaciones de fallos. Además, las funciones automáticas de particionamiento y balanceo de carga de MongoDB contribuyen a la escalabilidad y la disponibilidad del sistema.

En resumen, al elegir MongoDB como la base de datos principal, se busca satisfacer los atributos de calidad de disponibilidad y rendimiento. MongoDB ofrece consultas rápidas, tolerancia a fallos y escalabilidad automática, lo cual cumple con los requisitos del proyecto y garantiza un almacenamiento eficiente y confiable de la información del sistema.

Atributo de calidad y tácticas que cumplen:
Disponibilidad
Performance

Decisión: Utilizar PM2

Contexto y problema:
Debido a que la plataforma, una vez puesta en producción, puede verse saturada de solicitudes de acceso o procesamiento en un momento dado del tiempo, se deberá lograr la mayor capacidad de procesamiento posible, sin pérdida de datos y logrando la mejor latencia posible.

Opciones:
Utilizar PM2: PM2 es un gestor de procesos para aplicaciones Node.js que proporciona un conjunto de características avanzadas para escalar y administrar eficientemente las aplicaciones en producción. Permite la ejecución en múltiples instancias, el balanceo de carga y la recuperación automática en caso de fallos.

Justificación: La elección de utilizar PM2 como gestor de procesos para la plataforma se basa en los siguientes puntos:

Disponibilidad: PM2 favorece la disponibilidad del sistema al permitir la ejecución de múltiples instancias de la aplicación. Si una instancia falla o se cae, PM2 puede automáticamente iniciar una nueva instancia para mantener el servicio en funcionamiento y minimizar el tiempo de inactividad.

Capacidad de procesamiento: Al ejecutar múltiples instancias de la aplicación con PM2, se puede aprovechar la capacidad de procesamiento de forma escalable. Esto permite manejar un mayor volumen de solicitudes y evitar la saturación del sistema.

Balanceo de carga: PM2 ofrece funcionalidades de balanceo de carga que distribuyen de manera equilibrada las solicitudes entre las instancias en ejecución. Esto ayuda a evitar la sobrecarga en un solo servidor y mejora el rendimiento general del sistema al distribuir eficientemente la carga de trabajo.

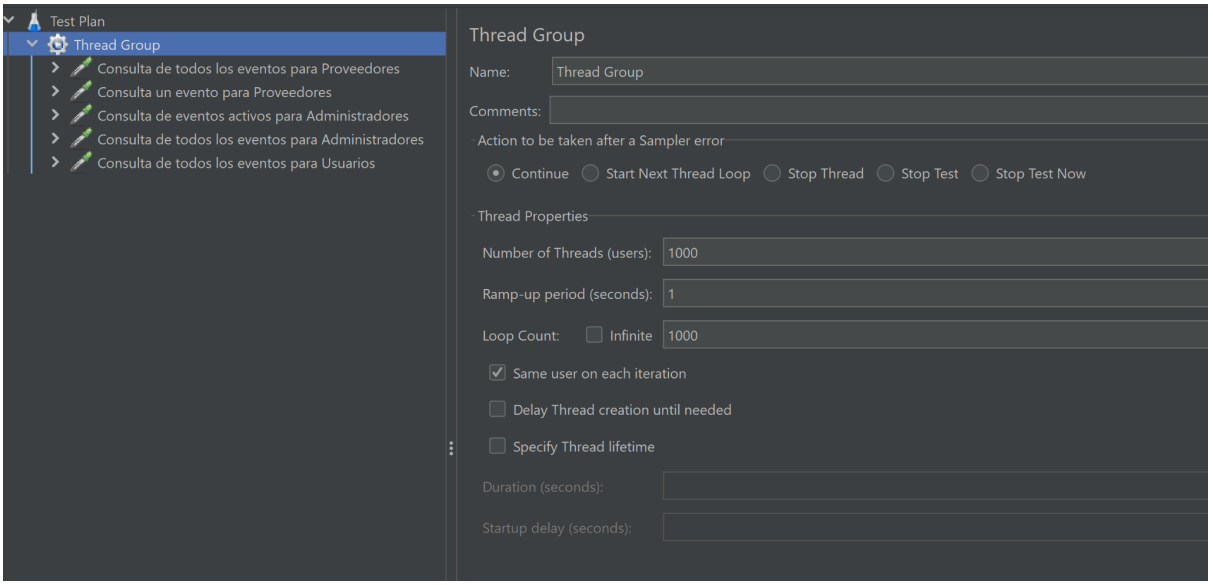
Recuperación automática: PM2 cuenta con mecanismos automáticos de recuperación en caso de fallos. Si una instancia de la aplicación se cae o deja de responder, PM2 puede detectar el fallo y reiniciar automáticamente la instancia afectada. Esto contribuye a mantener la disponibilidad del sistema y garantizar que los datos no se pierdan.

Atributo de calidad y tácticas que cumplen:
Performance → balanceo de carga
Escalabilidad → capacidad de procesamiento
Disponibilidad → Recuperación automática

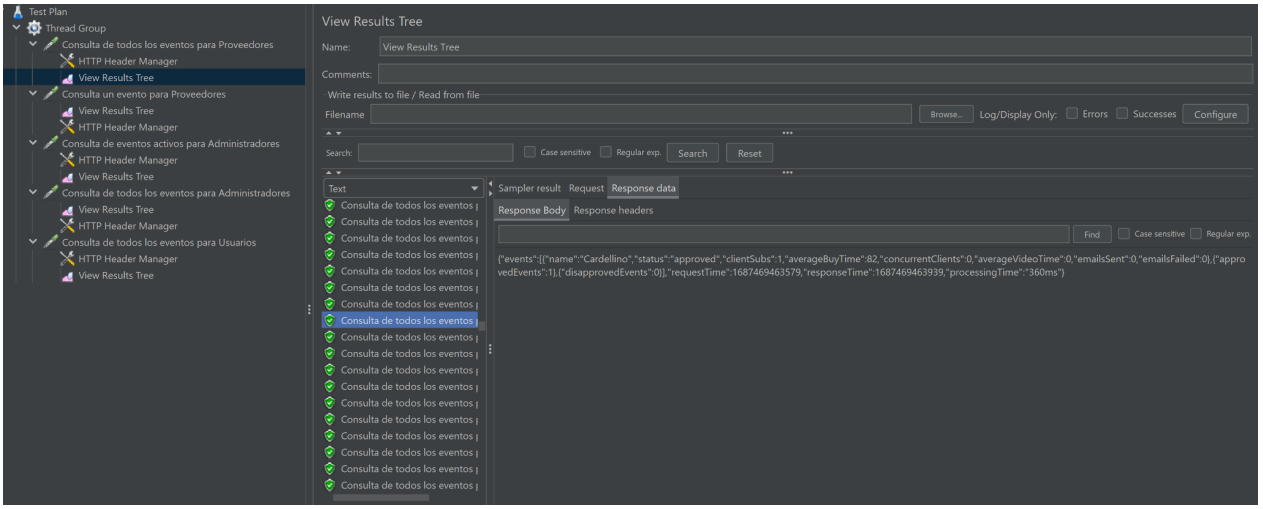
4. Anexo

4.1 Pruebas en JMETER

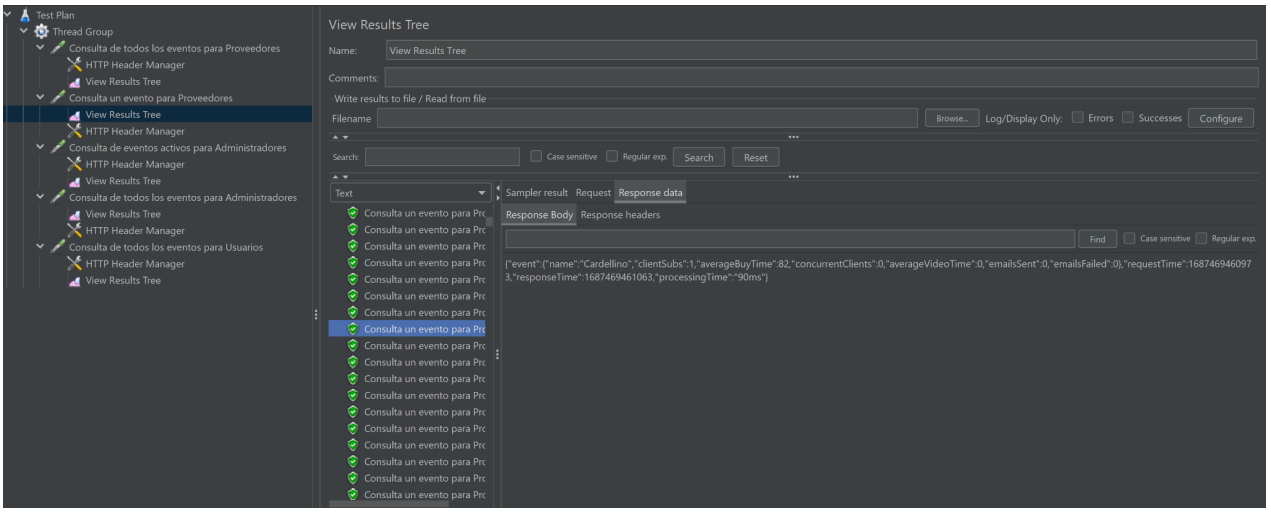
Realizamos pruebas en la aplicación JMeter para poder testear la performance del sistema. Realizamos las pruebas sobre todas las queries para poder ver el rendimiento de cada una de ellas. Además, como los requerimientos indican, las consultas tienen en su response el tiempo que le demoró realizar las consulta. Esto fue de gran ayuda para poder verificar que todo funcione acorde a lo esperado (menos de 5 segundos)
La siguiente imagen es la configuración realizada para las consultas. Como se puede observar, se agregaron 1000 usuarios, en donde cada uno de ellos realizaba 1000 consultas.



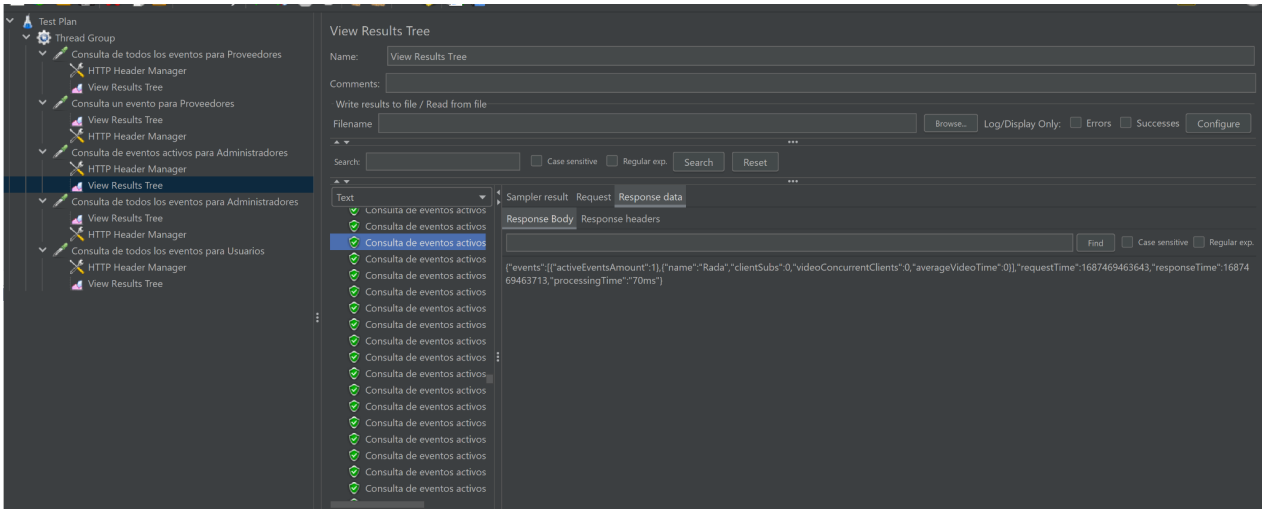
A continuación se muestra el resultado de todas ellas con algún ejemplo en cada una.
Consulta de todos los eventos para Proveedores



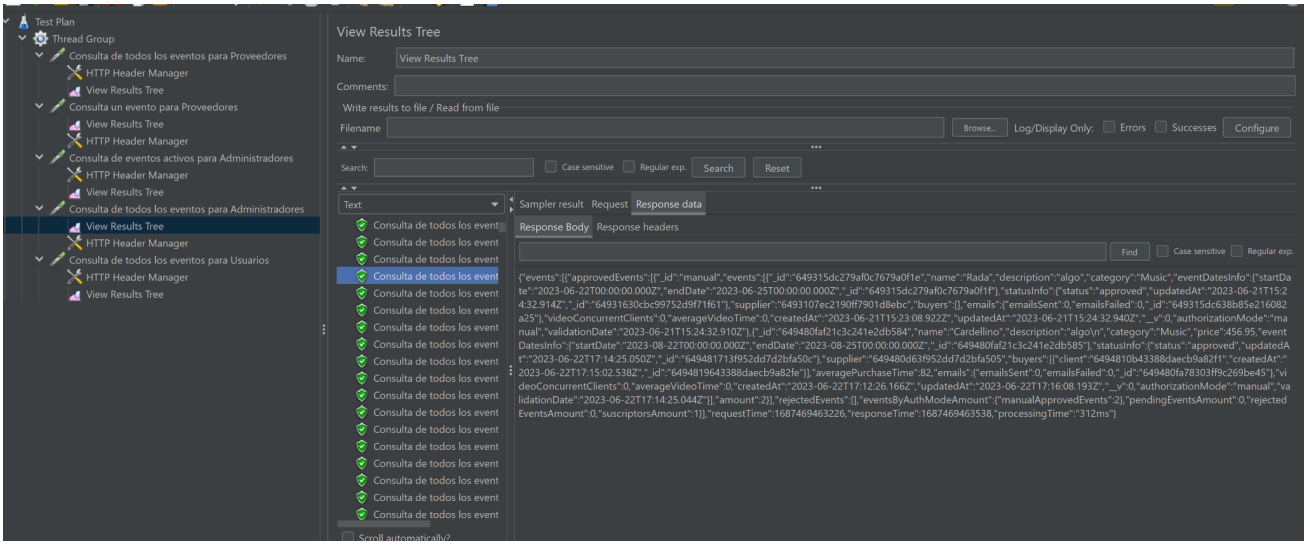
Consulta un evento para Proveedores



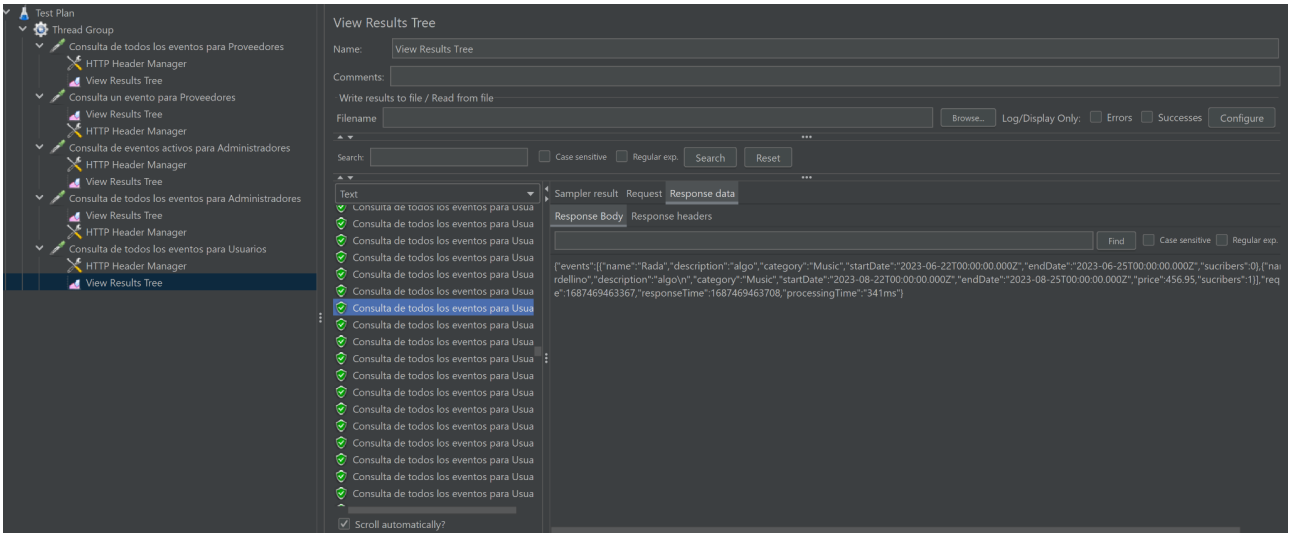
Consulta de eventos activos para Administradores



Consulta de todos los eventos para Administradores



Consulta de todos los eventos para Usuarios



4.2 Guía de instalación

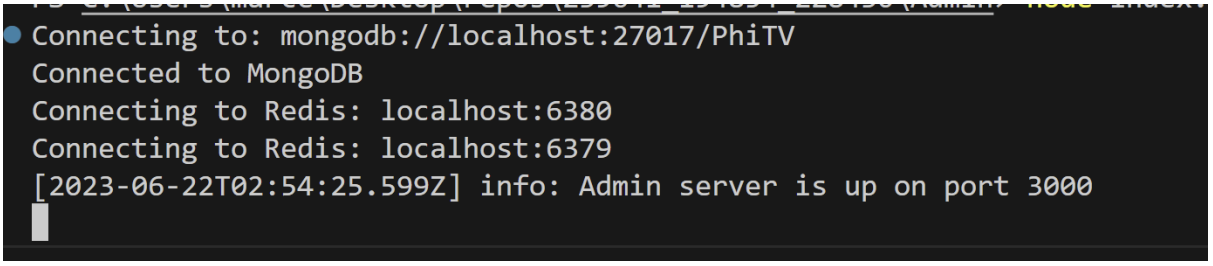
En primer lugar, es fundamental contar con las siguientes tecnologías:

- mongodb
- redis

En nuestro caso particular, corrimos todos estas tecnologías utilizando Docker.

Una vez que se tengan estas tecnologías instaladas, se debe correr el comando *“npm install”* en **todos** los proyectos. De esta forma, se instalarán todas las dependencias de cada proyecto para poder correrlos.

A partir de este punto ya es posible correr todos los proyectos. Para correr *uno por uno* se debe ingresar a cada módulo y correr el comando *“node index.js”*
Una vez corriendo, se debería visualizar en todos los proyectos algo similar a esto:



La otra forma posible es utilizando PM2. Para poder correrlo, se debe de instalar PM2 global utilizando el comando *“npm install pm2 -g”*
El archivo *ecosystem.config.js* ya está configurado. Una vez instalado, se corre el comando *pm2 start* desde el root del proyecto. Para finalizar se corre el comando *pm2 stop all.*

Aspectos importantes:
En varios casos se crean elementos por defecto. Esto se debe a que se necesitan desde un principio.
Por ejemplo, cuando se inicializa el proyecto Admin, se crea un administrador por defecto. Esto es debido a que tomamos la decisión de que solamente los administradores pueden crear más administradores. En caso de que no exista ninguno, nunca se podrían crear admins.
Además, también se crea por defecto el AuthModes en manual. Esto es una clase única, utilizando el patrón singleton, que se utiliza para que el sistema sepa en todo momento cual es el estado de autorización de eventos (es decir, manual o automático)
Por último, se crea por defecto también la instancia de NotificationConfigs, encargada de contener el tiempo necesario para enviar el recordatorio del evento.
Este código se puede ver dentro de Admin/Repositories/repository.js