Alumno: Kevin Alirio Pérez Rengifo

Enlace git:
https://github.com/KevinAPerez/Ptrones_Diseno_Software/tree/main/Final%20Arquitectura%20SISVENT

Contexto:

Se ha construido un sistema de ventas en línea similar a eBay o Facebook Marketplace que permite la compra y venta de productos. Este sistema se construyó en un estilo arquitectónico monolítico. El producto ha tenido un éxito inesperado dejando el esquema actual de arquitectura, limitado ya que no permite la escalabilidad, por lo cual se requiere hacer reingeniería del sistema.

Los elementos claves del sistema actual son los siguientes:

1. Las fotografías y videos de los usuarios se almacenan de manera persistente en el servidor del backend.
2. La información de los usuarios, los productos, compras y ventas se almacenan en una base de datos relacional.

Se desea que el sistema, en esta reingeniería cuente con la posibilidad:

1. Las fotos y videos puedan encolarse para ser redimensionado y procesados de manera asincrónica.
2. El sistema debe contar con un módulo para un sistema de pago tolerante a fallos.
3. El sistema debe permitir la subscripción a “Tiendas” que permitan enviar a un usuario información de su interés.
4. El sistema debe contar con un mecanismo de autenticación y autorización de solicitudes.
5. El sistema debe contar con un módulo de publicidad, que se encargará de enviar información personalizada al usuario.

Solución:

· ¿Qué estilos arquitectónicos escogería? Y ¿Cuál es la razón para usar estos estilos arquitectónicos?

Arquitectura	¿Por qué?	Retos
1. Arquitectura de Microservicios	1. Responsabilidad por dominios respectando cada funcionalidad 2. Independencia de cada micro para facilitar el mantenimiento y evolución 3. Escalabilidad individual	1. Descomposición y definición de dominios 2. Comunicación entre servicios 3. Costos 4- Consistencia y sincronización de datos
2. Arquitectura Orientada en Eventos	1. Desacoplamiento de servicios 2. Manejo de tareas asíncronas para el	1. Consistencia y sincronización de eventos generados 2. Manejo de errores y

	procesamiento de multimedia 3. Escalabilidad independiente	reintentos 3. Evolución de los contratos de eventos
3. Arquitectura en Capas	1. Flexibilidad en la aplicación de políticas por capas 2. Facilidad en la implementación de accesos y sus controles	1. Latencia entre las capas definidas 2. Coherencia y escalabilidad en las políticas configuradas para cada capa

- ¿Cómo se asegura la extensibilidad, escalabilidad y mantenibilidad al hacer uso de estos patrones?

Extensibilidad	Escalabilidad	Mantenibilidad
Adición de nuevos servicios sin modificar los existentes. Suscripción de nuevos consumidores a los eventos existentes. Manejo de versionamiento y publicación mediante un Api Gateway para asegurar la compatibilidad.	Escalabilidad individual basada en la demanda recibida con el uso de orquestadores y la contenerización. Procesamiento desacoplado y asincrónico. Cacheo de información e implementación de colas..	Separación de responsabilidades por dominios. Compilación y despliegues independientes. Versionamiento de Apis y eventos. Definición de métricas y trazabilidad por dominios.

- ¿Qué patrones arquitectónicos emplearía y cuál sería la razón?

Patrón de capas: permite estructurar o descomponer el programa en grupos de subtareas, cada una de las cuales con un nivel particular de abstracción. Capas: Capa de presentación UI (punto de entrada de las solicitudes), Capa de aplicación (servicios independientes) y Capa de acceso a datos (almacenamiento y BDs particulares para cada servicio).

Patrón de eventos: Los servicios no dependerán directamente uno del otro, danto respuestas inmediatas a cada solicitud, en cuenta al procesamiento multimedia facilita el redimensionamiento de las imágenes y controla el envió de notificaciones.

- ***¿Qué patrón de diseño emplearía y cuál sería la razón para su uso?***

Patrón de cola de eventos: Este patrón permitirá desacoplar los componentes del sistema y manejar tareas de forma asincrónica, lo cual es esencial para asegurar la escalabilidad, resiliencia y alta disponibilidad. Con este patrón se podrá escalar horizontalmente los workers según su demanda, encolar las solicitudes si un worker falla, adición de funciones sin afectar en sistema principal, etc.

- ***¿Como gestionar el almacenamiento que asegure disponibilidad entre los diferentes módulos?***

1. Almacenamiento centralizado y escalable, para la accesibilidad desde múltiples servicios, control de acceso granular y almacenamiento de multimedia.
2. Uso de servicios de archivos distribuidos, para facilitar el acceso de los diferentes servicios a los mismos archivos.
3. Almacenamiento de metadata en BD para el acceso rápido a los archivos.
4. Definición de accesos por roles.

- ***¿Como garantizar que exista una comunicación tolerante a fallos en el procesamiento de los pagos?***

Con el uso de cola de Eventos se pueden realizar el procesamiento asíncrono de modo que se encolen las peticiones e ir actualizando el estado le las mismas, manteniendo al sistema aislado de picos de carga y reintentando de forma automática.

- ***¿Como permitir el encolado de fotos y videos para su procesamiento?***

Se debe contar con los siguientes elementos:

UI --> Servicio de carga de archivo --> Bus --> Workers --> Añmacenamiento de archivos --> almacenamiento de Metadata.

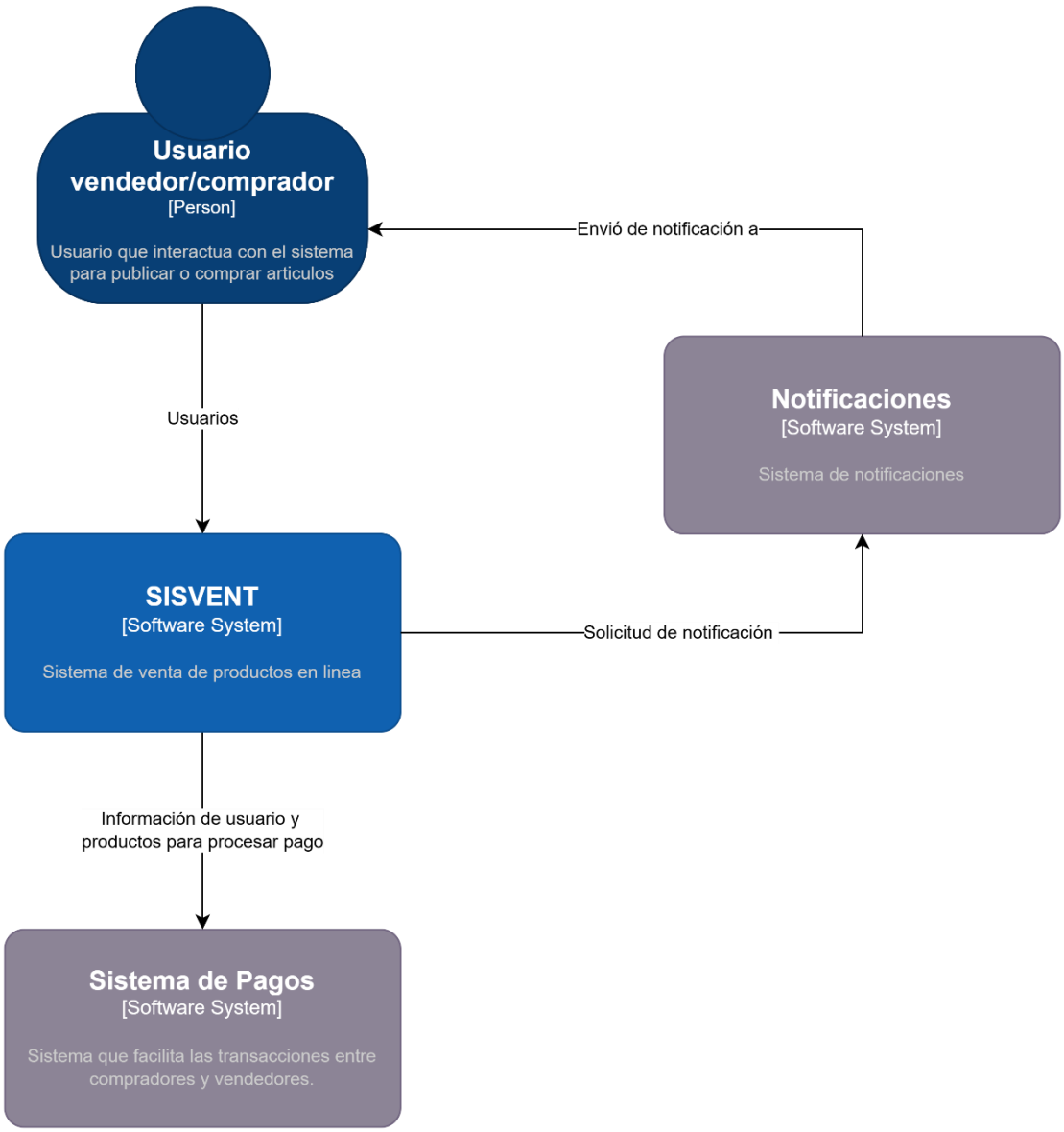
- ***¿Como garantizar que la aplicación sea escalable horizontalmente y cuente con alta disponibilidad?***

Escalabilidad: Creación de servicios independientes (separación de dominios), utilizar Kubernetes para contenerizar y orquestar los servicios, implementar balanceos de carga e independizar el almacenamiento propio de cada servicio y almacenamiento compartido para los archivos.

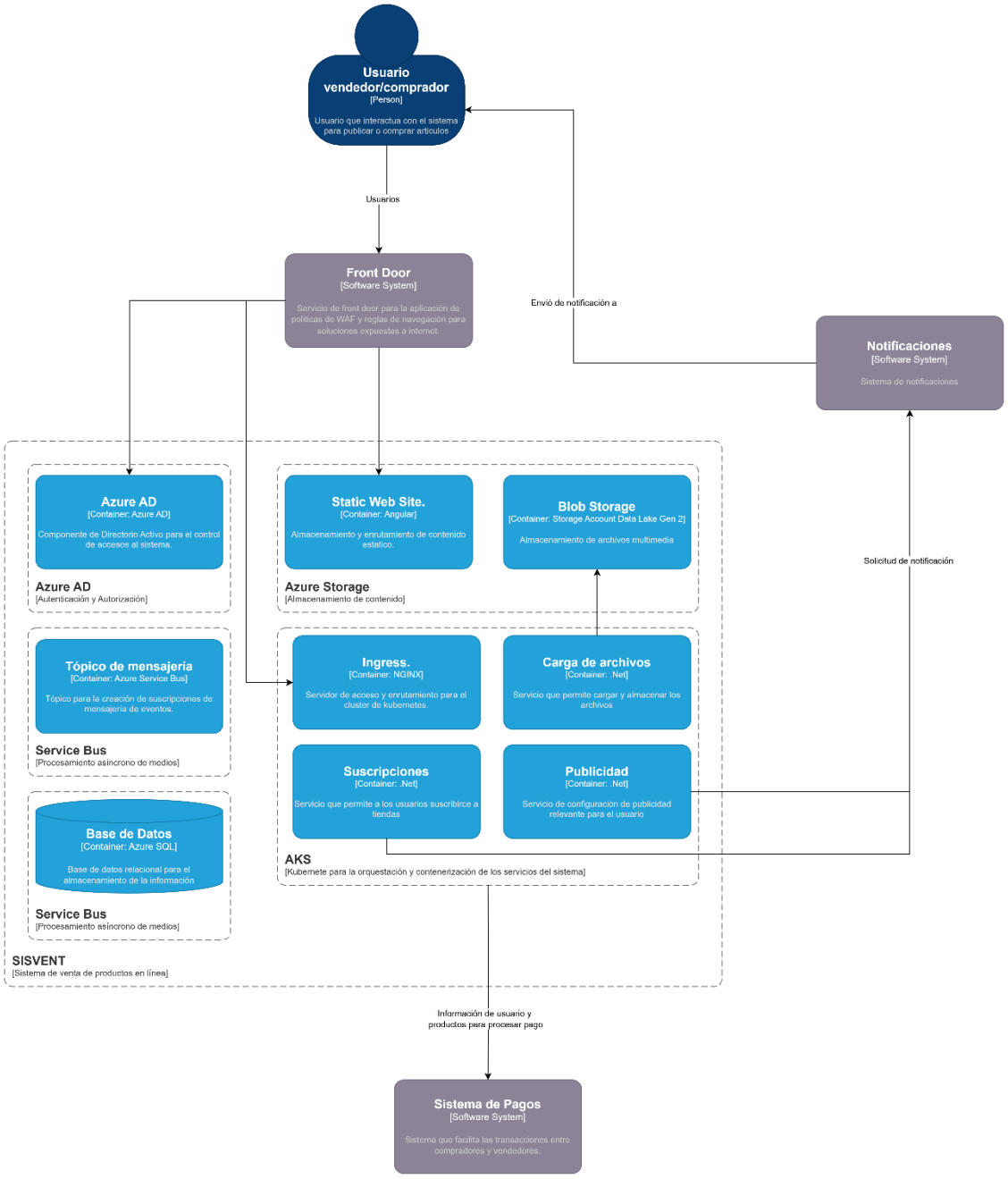
Alta disponibilidad: Despliegue multi zonas, infraestructura activa/activa, configuraciones de monitoreo.

Diagramas:

C4 – Nivel 1



C4 – Nivel 2


Usuario vendedor/comprador
[Person]
Usuario que interactua con el sistema para publicar o comprar articulos
Usuarios
Front Door
[Software System]
Servicio de front door para la aplicación de políticas de WAF y reglas de navegación para soluciones expuestas a internet.
Envió de notificación a
Notificaciones
[Software System]
Sistema de notificaciones
Azure AD
[Container: Azure AD]
Componente de Directorio Activo para el control de accesos al sistema.
Azure AD
[Autenticación y Autorización]
Static Web Site.
[Container: Angular]
Almacenamiento y enrutamiento de contenido estatico.
Blob Storage
[Container: Storage Account Data Lake Gen 2]
Almacenamiento de archivos multimedia
Azure Storage
[Almacenamiento de contenido]
Solicitud de notificación
Tópico de mensajería
[Container: Azure Service Bus]
Tópico para la creación de suscripciones de mensajería de eventos.
Service Bus
[Procesamiento asincrono de medios]
Ingress.
[Container: NGINX]
Servidor de acceso y enrutamiento para el cluster de kubernetes.
Carga de archivos
[Container: .Net]
Servicio que permite cargar y almacenar los archivos
Suscripciones
[Container: .Net]
Servicio que permite a los usuarios suscribirce a tiendas
Publicidad
[Container: .Net]
Servicio de configuración de publicidad relevante para el usuario
AKS
[Kubernete para la orquestación y contenerización de los servicios del sistema]
Base de Datos
[Container: Azure SQL]
Base de datos relacional para el almacenamiento de la información
Service Bus
[Procesamiento asincrono de medios]
SISVENT
[Sistema de venta de productos en línea]
Información de usuario y productos para procesar pago
Sistema de Pagos
[Software System]
Sistema que facilita las transacciones entre compradores y vendedores.

Arquitectura Nube Azure