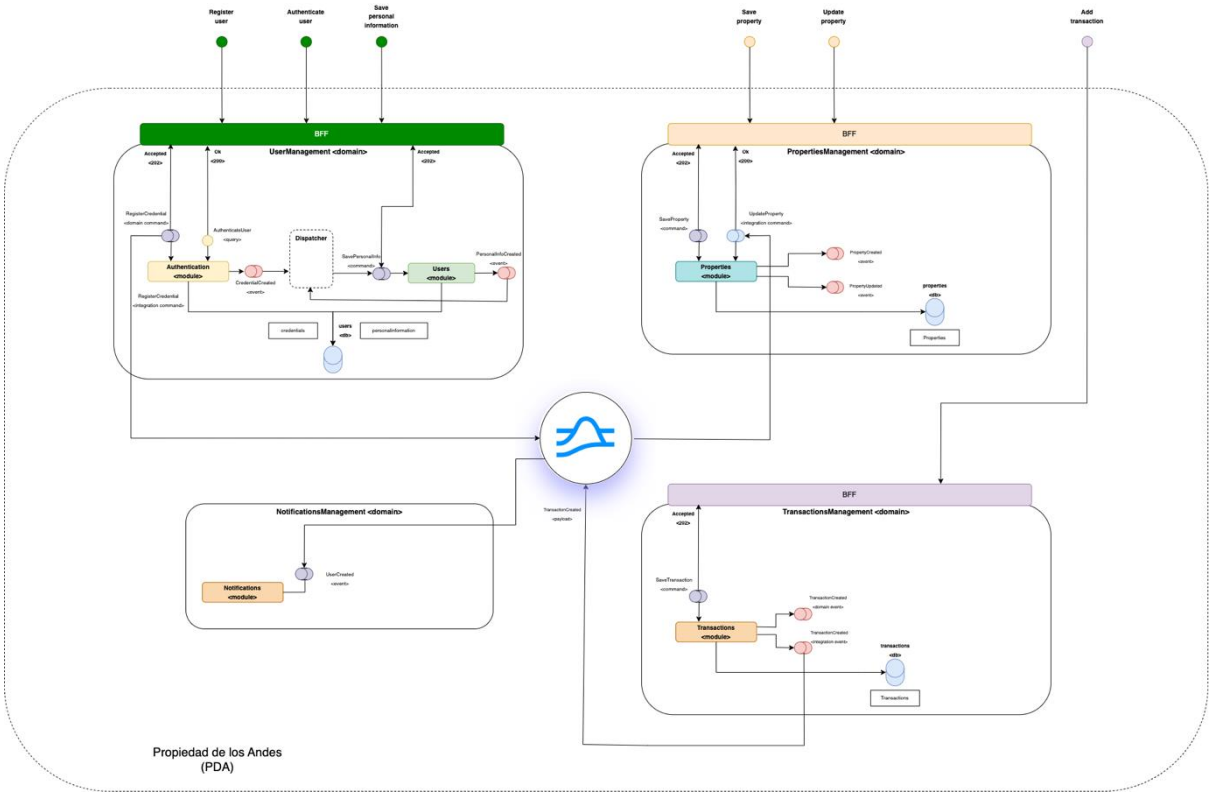
ARQUITECTURA BAJO PRUEBA

1. Diagrama de componentes



Descripción:

El anterior diagrama es una primera aproximación de la integración de los diferentes contextos acotados propuestos en documentos anteriores para la desmantelación del monolito de Propiedad de los Andes.

Hemos añadido 3 microservicios más, *propiedades, transacciones y notificaciones.* Estos interactúan mediante eventos y comandos de integración usando Apache Pulsar.

Este diagrama pretende simular el comportamiento de venta o pago de arrendamiento de una propiedad. Dado que el microservicio *transacciones* también soporta registro de ventas, se ve la necesidad de ofrecer una orquestación donde también se cambien el dueño de la propiedad en la DB administrada por *propiedades.* El objetivo es permitir que un usuario de la aplicación puede cambiar el propietario de un bien raíz, es decir permitir desde algún frontend llamar esta funcionalidad.

Hemos añadido el microservicio de notificaciones que se encargará de enviar mensajes vía correo electrónico cuando suceden eventos del módulo de transacciones. Este código base se usa de referencia para uno de los experimentos de la entrega final (atributo Mantenibilidad).

Consideraciones y patrones:

- La arquitectura está diseñada para separar distintas áreas de responsabilidad en módulos independientes.
- Utiliza eventos para la comunicación asincrónica, lo que ayuda a desacoplar los servicios y facilita la escalabilidad.
- Cada dominio idealmente tiene un BFF asociado lo que puede mejorar la experiencia del usuario al proporcionar funcionalidades backend optimizadas para diferentes necesidades de cliente. De momento, sólo se armará un BFF dedicado para agentes que registren información recolectada (y es el que se está desarrollando esta semana).

Consideraciones de desarrollo y tácticas:

- Se eligió modelo CRUD tradicional debido a su facilidad de implementación y familiaridad del equipo con la misma. En cambio, el equipo al no tener práctica con arquitecturas basadas en Event Sourcing, estima que implementar y verificar su aplicación correcta al modelo base de arquitectura originalmente propuesto excedería el presupuesto de tiempo que se tiene. Se necesitaría al menos el doble de tiempo (se esperan máximo 5 horas de implementación a la semana por integrante), y tener un dominio más completo de Pulsar para poder diagnosticar correctamente problemas posibles, como aquellos que deriven de persistir eventos de forma incorrecta, o con atributos no apropiados, o una mala configuración de log append-only, entre otros.
- Se escoge Avro Schema para mostrar de forma más explícita el seguimiento de un estándar público para formato de mensajes publicado en Broker de eventos. Además, el modelo Python de programación elegido permite generar el schema Avro al conectarse por primera vez a Pulsar localmente. Luego se extrae el schema resultante por REST API de Apache Pulsar, ver https://pulsar.apache.org/admin-rest-api/?version=3.2.0#operation/SchemasResource_getSchema
- Se escoge topología de administración de datos Descentralizada. En parte, porque permite a los desarrolladores trabajar en grupos donde no necesitan coordinarse fuertemente con otros para avanzar sus diseños, facilitar el escalamiento de bases de datos llegado el momento, y para evitar todas las desventajas derivadas de una topología Centralizada. No se ve ganancia para la arquitectura elegida el empleo de una topología Híbrida, asi que se descarta por extensión.

2. Escenarios de calidad elegidos

El equipo ha elegido los siguientes escenarios de calidad.

- Disponibilidad 1: Durante periodos de temporada alta, vacaciones, festividades, eventos locales, el número de usuarios activos aumenta repentinamente, el sistema sigue siendo capaz de seguir funcionando ante todos los usuarios
- Mantenibilidad 3: Cuando un usuario realice el pago de la renta de una propiedad por medio de un canal digital externo, el sistema debe confirmar si la transacción fue aprobada o rechazada a través de una notificación.
- Escalabilidad 3: Un usuario en potencia del aplicativo web de PDA ingresa al portal web de PDA para conocer la solución y comenzar a explorar las funcionalidades que este ofrece.

Para más detalles de los escenarios, ver Entrega 3.

En particular, esto implica ejecutar al menos 2 experimentos. El Primero, durante el sistema bajo alta carga (aprox. 1000 llamados por segundo al BFF invocador de funcionalidad backend, o al iniciador de Saga) verificar que, por los menos 900 de 1000 respuestas sean exitosas (idealmente 999/1000 viendo la medida de respuesta elegida), que no sean de error tipo HTTP 500 o similares, incluyendo timeouts (sin respuesta). El Segundo, viendo que el sistema ya experimentado (con o sin éxito) el poder montar un nuevo componente y ver que las respuestas se mantengan exitosas, a la vez que se persiste nueva data por el componente de Recepción de Pagos. El equipo no considera posible validar todos los escenarios de calidad elegidos con un experimento singular, y además ha escogido aquellos que se pueden demostrar (o no) de forma clara, dejando de lado los escenarios triviales o que requieren mayor infraestructura. Un ejemplo de ello son Escalabilidad 2 y 3, al requerir DBs particionadas y que además sincronicen su data de forma consistente en una infraestructura nube.