

Hola. Vamos a avanzar de una forma más conceptual. Como te comentaba el objetivo es montar una solución de diario electrónico que registre toda la actividad de un cliente/usuario con nuestros sistemas. La aproximación que quiero seguir es mediante el patron de event sourcing. Para ello el primer paso es conseguir una manera eficiente de generar estos eventos en nuestra capa de APIs (BFF y Domain), y esto es lo que hemos hecho hasta ahora. Estos eventos se enviaran de forma asíncrona a un topic Kafka y en Kafka tendremos un primer nivel de persistencia, ya que debemos garantizar que estos eventos se procesan, aunque no es necesario un tiempo real (no necesitamos condicionar la respuesta de la API a que el evento de auditoria se haya procesado pero si que se ha registrado en el topic Kafka aunque aún nadie lo haya consumido). El siguiente paso ya es procesar los eventos para ir construyendo la información de diario electrónico. Aquí es donde me pierdo un poco, porque no conozco a fondo todo el concepto de Event Sourcing. Entiendo más o menos los conceptos de agregación, proyección , y que de todo esto procesamiento igual deberíamos acabar generando vistas materializadas para diferentes casos de uso, por ejemplo, el de diario electrónico cuyo modelo de uso normalmente es que alguien de auditoria requiere conocer la actividad de un cliente determinado en un espacio de tiempo concreto. Con la información que tenemos podemos identificar usuarios en un periodo tiempo con el evento de login y con la correlación de la sesión podríamos saber la actividad que un usuario hizo despues de ese login. Te paso a continuación los eventos que estoy generando en mi prueba de concepto con las APIs mockeadas que simula la interacción de un usuario con la aplicación movil de banco haciendo el login, que le carga la home page consultando sus cuentas, la selección de una cuenta para hacer una transferencia, notificando al destinatario y después una consulta de movimientos para asegurarse que la transferencia se realizo:

<EVENTS>(@ASYNC\_EVENT):

```
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:57.486243",  
"sessionId":"session-9c9ef85f-af30-4743-8dd9-  
e7708416bf8b","method":"login","arguments":  
{"username":"matt","password":"****s123"},"result":  
{"sessionId":"session-9c9ef85f-af30-4743-8dd9-  
e7708416bf8b","loginDate":"2025-07-18T08:47:57.486190"}}
```

(@ASYNC\_EVENT):

```
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:57.808895",  
"sessionId":"session-9c9ef85f-af30-4743-8dd9-  
e7708416bf8b","method":"loadLandingPage","arguments":  
{"userId":"125689"},"result":{"element[0]":  
{"accountId":"acc-001","accountType":"savings","balance":10250.75},"element  
[1]":{"accountId":"acc-002","accountType":"checking","balance":2450.0}}}
```

(@ASYNC\_EVENT):

```
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:58.146430",  
"sessionId":"session-9c9ef85f-af30-4743-8dd9-  
e7708416bf8b","method":"executeTransfer","arguments":{"request":  
{"fromAccountId":"acc-001","toAccountId":"acc-002","amount":150.75,"notifica
```

```

tionMethod":"EMAIL","recipientContact":"pepegrillo@mymail.com"}}, "result":
{"transactionId":"277b6c7a-1a90-4171-9e71-9f3f4ec9519c","status":"success
"}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:58.478748",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"getAccountMovements","arguments":
{"accountId":"acc-001","from":"2025-05-12","to":"2025-07-12"},"result":{}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:57.484547",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"login","arguments":
{"username":"matt","password":"****s123"},"result":
{"sessionId":"session-9c9ef85f-af30-4743-8dd9-e7708416bf8b"}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:57.807486",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"getAccountsByUser","arguments":
{"userId":"125689"},"result":{"element[0]":
{"accountId":"acc-001","accountType":"savings","balance":10250.75},"element
[1]":{"accountId":"acc-002","accountType":"checking","balance":2450.0}}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:58.142052",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"makeTransfer","arguments":{"request":
{"fromAccountId":"acc-001","toAccountId":"acc-002","amount":150.75}}, "result
":
{"transactionId":"277b6c7a-1a90-4171-9e71-9f3f4ec9519c","status":"success
"}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:58.145839",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"sendNotification","arguments":{"request":
{"recipientContact":"pepegrillo@mymail.com","method":"EMAIL","message":"Tr
ansfer for rental car survey"}}}
(@ASYNCEVENT):
{"eventType":"MethodExecuted","timestamp":"2025-07-18T08:47:58.477620",
"sessionId":"session-9c9ef85f-af30-4743-8dd9-
e7708416bf8b","method":"getMovements","arguments":
{"accountId":"acc-001","from":"2025-05-12","to":"2025-07-12"},"result":{}}</
EVENTS>.

```

La idea es que estos eventos los consuma un listener que se integre con Axon Server como framework de Event Sourcing, y este procese eventos para ir generando la información válida para el caso de uso de diario electrónico, y lo persista en algún data store (que debe guardar esta información durante años por temas de auditoría), probablemente una base de datos. Me gustaría saber tu opinión sobre el planteamiento, y en caso de que así sea, que me detalles el papel en concreto de Axon Server, como debería

procesar estos eventos para .... agruparlos? como los persiste en el data store, si hemos de crear alguna vista materializada para este caso de uso. En general, como aplico los conceptos de Event Sourcing a todo esto y que debe hacer esta capa de Event Sourcing (Axon Server) con los eventos que le van llegando y que vamos a acabar generando. Explicamelo todo, como aplicamos Event Sourcing, que conceptos tienen sentido, cual es el papel de Axon Server y como avanzamos en el desarrollo de esta solución considerando que el tipo de eventos que te he adjuntado estan en Kafka dispuestos a ser consumidos por un listener que empiece a procesar y aplicar Event Sourcing. Puedes contestarme en castellano si lo deseas



¡Hola! Excelente pregunta. Estás en el camino correcto al aplicar Event Sourcing para construir tu solución de diario electrónico. Vamos a desglosar cómo Axon Server, junto con los conceptos de Event Sourcing, puede ayudarte a procesar esos eventos que ya tienes en Kafka.

#### **Resumen del Planteamiento:**

Tu planteamiento general es sólido:

1. **Generación de eventos:** Ya tienes los eventos generándose en la capa de APIs.
2. **Kafka como buffer:** Usas Kafka para asegurar la persistencia y desacoplamiento de la generación y el procesamiento de eventos.
3. **Axon Server:** Planeas usar Axon Server para aplicar patrones de Event Sourcing y construir tu diario electrónico.
4. **Data Store:** Persistir la información procesada en una base de datos para el caso de uso específico de auditoría.

#### **El Papel de Axon Server y los Conceptos de Event Sourcing:**

Axon Server es una plataforma que facilita la implementación de patrones de Event Sourcing, Command Query Responsibility Segregation (CQRS) y Domain-Driven Design (DDD). En tu caso, Axon Server te ayudará a:

- **Almacenar los eventos:** Axon Server proporciona un Event Store, que es un almacén de eventos inmutable. Almacenarás tus eventos de Kafka en Axon Server.
- **Construir el estado de la aplicación:** A partir de los eventos almacenados, podrás construir el estado de tu diario electrónico.
- **Crear vistas materializadas:** Axon Server te permite crear vistas materializadas para optimizar las consultas y facilitar el acceso a la información.

#### **Cómo Aplicar los Conceptos de Event Sourcing:**

1. **Agregados (Aggregates):**

- **Definición:** Un agregado es un grupo de entidades que se tratan como una sola unidad. En tu caso, un agregado natural podría ser la "Sesión de Usuario".
- **Eventos de Dominio:** Los eventos que generas (login, transferencia, consulta de movimientos) representan cambios de estado dentro de ese agregado "Sesión de Usuario".
- **Beneficios:** Los agregados encapsulan la lógica de negocio y garantizan la consistencia de los datos.

## 2. Eventos de Dominio (Domain Events):

- **Tus Eventos:** Ya tienes eventos como MethodExecuted con información sobre la ejecución de métodos.
- **Convertir a Eventos de Dominio:** Es importante que estos eventos genéricos se transformen en eventos de dominio más específicos y semánticos. Por ejemplo, en lugar de MethodExecuted, podrías tener eventos como UserLoggedIn, FundsTransferred, AccountMovementsRetrieved. Esto hace que tu modelo sea más claro y expresivo.

## 3. Proyecciones (Projections):

- **Definición:** Las proyecciones son procesos que consumen los eventos de dominio y construyen vistas de la información que son optimizadas para casos de uso específicos.
- **Tu Diario Electrónico:** En tu caso, la proyección clave es la que construye el "Diario Electrónico" del usuario.

## 4. Vistas Materializadas (Materialized Views):

- **Definición:** Son representaciones pre-calculadas de los datos que se almacenan en una base de datos para optimizar las consultas.
- **Tu Vista:** Tu vista materializada podría ser una tabla en una base de datos que almacena la actividad del usuario en un período de tiempo específico, ordenada por fecha y hora. Esta tabla estaría optimizada para consultas rápidas por parte del equipo de auditoría.

## Pasos Concretos con Axon Server:

### 1. Configurar el Listener de Kafka:

- Crea un listener de Kafka que consuma los eventos del topic.
- Este listener será el punto de entrada a tu aplicación de Event Sourcing con Axon.

### 2. Transformar los Eventos de Kafka a Eventos de Dominio de Axon:

- Dentro del listener, transforma tus eventos MethodExecuted a eventos de dominio más específicos.
- Usa la información del campo "method" para determinar qué tipo de evento de dominio crear.
- Por ejemplo:
  - ◆ Si method es "login", crea un evento UserLoggedInEvent.
  - ◆ Si method es "executeTransfer", crea un

evento FundsTransferredEvent.

- ◆ Si method es "getAccountMovements", crea un evento AccountMovementsRetrievedEvent.
- Incluye el sessionId y otra información relevante en los eventos de dominio.

### 3. Definir el Agregado "Sesión de Usuario":

- Crea una clase que represente el agregado "Sesión de Usuario".
- Esta clase contendrá el sessionId, el userId, la fecha de inicio de la sesión, y cualquier otra información relevante.
- Esta clase no contendrá el estado actual de las cuentas ni las transferencias. El objetivo es agrupar la actividad del usuario.

### 4. Crear un Handler de Eventos (Event Handler):

- Crea un handler de eventos que reciba los eventos de dominio y actualice el estado del agregado "Sesión de Usuario".
- Por ejemplo, cuando reciba un evento UserLoggedInEvent, actualizará la fecha de inicio de la sesión.
- Axon Framework facilita la creación de estos handlers.

### 5. Crear la Proyección del "Diario Electrónico":

- Crea una proyección que consuma los eventos de dominio y construya la información del diario electrónico.
- Esta proyección podría crear una tabla en una base de datos con las siguientes columnas:
  - ◆ sessionId (para agrupar la actividad)
  - ◆ userId
  - ◆ timestamp
  - ◆ eventType (UserLoggedIn, FundsTransferred, AccountMovementsRetrieved, etc.)
  - ◆ details (un campo JSON con información específica del evento, como el monto de la transferencia, los IDs de las cuentas, etc.)

### 6. Optimizar con Vistas Materializadas:

- Si las consultas del equipo de auditoría son lentas, crea una vista materializada que pre-calcule la información que necesitan.
- Por ejemplo, podrías crear una vista que agrupe la actividad del usuario por día, o que calcule el saldo de la cuenta al final de cada día.

### Ejemplo Simplificado de Código (Ilustrativo - Adaptar a Axon Framework):

java  
Copy

```
// Evento de Dominio
public class FundsTransferredEvent {
    private final String sessionId;
    private final String fromAccountId;
    private final String toAccountId;
    private final double amount;
```

```

    public FundsTransferredEvent(String sessionId, String fromAccountId, String
toAccountId, double amount) {
        this.sessionId = sessionId;
        this.fromAccountId = fromAccountId;
        this.toAccountId = toAccountId;
        this.amount = amount;
    }

    // Getters
}

// Agregado "Sesión de Usuario" (Simplificado)
public class UserSessionAggregate {
    private String sessionId;
    private String userId;
    private LocalDateTime startTime;

    // Métodos para aplicar eventos y actualizar el estado
}

// Proyección para el "Diario Electrónico"
@Component
public class ElectronicJournalProjection {

    @EventHandler
    public void on(FundsTransferredEvent event) {
        // Insertar información en la tabla del diario electrónico
        // ...
    }

    @EventHandler
    public void on(UserLoggedInEvent event) {
        // Insertar información en la tabla del diario electrónico
        // ...
    }
}

```

#### **Puntos Clave:**

- **Eventos Semánticos:** La clave es transformar tus eventos genéricos a eventos de dominio que tengan un significado claro en tu modelo de negocio.
- **Axon Framework:** Axon Framework simplifica enormemente la implementación de patrones de Event Sourcing. Te recomiendo estudiarlo a fondo.
- **Vistas Materializadas:** No tengas miedo de crear vistas materializadas para optimizar las consultas.
- **Iteración:** Empieza con una implementación sencilla y ve iterando y

mejorando tu modelo a medida que aprendas más.  
Espero que esta explicación te haya sido útil. ¡Avísame si tienes alguna pregunta más!

Gracias por la explicación. Si, el problema es como transformar estos eventos genericos de MethodExecuted que genera de forma transparente el aspecto a un evento de negocio, solo tenemos el nombre del método. Como lo harías, alguna configuración de mapeo entre nombre de método y evento? Para aquellos métodos que en un momento dado no tengan un evento asociado se podrían asociar a un evento generico "ToBeReviewed" que permita un ajuste manual y la creación de nuevos eventos de dominio que tengan sentido. Defino un agregado "sesion de usuario", pero al procesar acabaré teniendo diferentes instancias de agregados de sesion, una por usuario-sesion, no? Es así? La persistencia en Axon Server es suficiente para mantener toda esta información durante años o debería persistir en otro tipo de data store? Las vistas materializadas se crean en Axon Server o son, por ejemplo tablas de una base datos? nsi es así, tiene sentido que lo que persista durante años sean las vistas materializadas de actividad de usuario? Lo que necesito persistir durante años son los eventos de dominio transformados para poder recrear cuando sea necesario las proyecciones y las vistas materializadas? Las proyecciones y las vistas materializadas se crean on the fly y se mantienen un tiempo para sus consultas o se crean y siempre existen (pueden ocupar muchos recursos al tener que persistir esto durante años). Entiendo que lo que si o si se debe persistir son los eventos de dominio transformados, a partir de los cuales se puede recrear cualquier agregado o vista materializada, es así? ya ves que tengo mucha dudas



¡De nada! Es normal tener dudas al principio. Vamos a clarificar esos puntos:

### **1. Transformación de Eventos Genéricos a Eventos de Dominio:**

Tienes razón, depender solo del nombre del método para crear los eventos de dominio no es ideal. Aquí hay algunas opciones:

- **a) Configuración de Mapeo (Recomendada):**

- Crea un archivo de configuración (YAML, JSON, propiedades) que mapee los nombres de los métodos a los tipos de eventos de dominio.
- Este archivo podría tener algo como:  
yaml  
Copy

methodMappings:

- login: UserLoggedInEvent
- executeTransfer: FundsTransferredEvent
- getAccountMovements: AccountMovementsRetrievedEvent
- 
- En el listener de Kafka, lee este archivo de configuración y usa la información para crear los eventos de dominio.
- **b) Enumeración (Si Tienes un Conjunto Fijo de Métodos):**
  - Crea un enum que liste todos los nombres de métodos posibles y sus eventos de dominio correspondientes.

● java  
Copy

```

public enum MethodEventMapping {
    LOGIN("login", UserLoggedInEvent.class),
    EXECUTE_TRANSFER("executeTransfer",
        FundsTransferredEvent.class),
    GET_ACCOUNT_MOVEMENTS("getAccountMovements",
        AccountMovementsRetrievedEvent.class);

    private final String methodName;
    private final Class<? extends Event> eventClass;

    MethodEventMapping(String methodName, Class<? extends Event>
        eventClass) {
        this.methodName = methodName;
        this.eventClass = eventClass;
    }

    public String getMethodName() {
        return methodName;
    }

    public Class<? extends Event> getEventClass() {
        return eventClass;
    }

    public static Class<? extends Event>
        getEventClassByMethodName(String methodName) {
        return Arrays.stream(values())
            .filter(mapping ->
                mapping.getMethodName().equals(methodName))
            .map(MethodEventMapping::getEventClass)
            .findFirst()
    }

```



- `.orElse(ToBeReviewedEvent.class); // Default event`
- `}`
- `}`
- 

- **c) Evento Genérico "ToBeReviewed":**

- Si el nombre del método no se encuentra en la configuración o enum, crea un evento genérico `ToBeReviewedEvent`.
- Este evento indicaría que un humano debe revisar el evento y determinar qué tipo de evento de dominio crear.

## 2. Múltiples Instancias del Agregado "Sesión de Usuario":

- Sí, al procesar los eventos, tendrás diferentes instancias del agregado "Sesión de Usuario", una por cada combinación única de usuario y sesión (`sessionId`).
- Cada instancia representará una sesión específica del usuario.
- Axon Framework te ayudará a gestionar estas instancias y garantizar la consistencia de los datos.

## 3. Persistencia en Axon Server vs. Otro Data Store:

- **Axon Server (Event Store):**

- Está diseñado para almacenar los eventos de dominio *inmutables*.
- Es *ideal* para almacenar los eventos que necesitas para recrear el estado de tu aplicación.
- No está optimizado para consultas complejas o agregaciones.

- **Otro Data Store (Base de Datos, etc.):**

- Es donde almacenarás las *vistas materializadas* que son optimizadas para tus casos de uso específicos (como el diario electrónico para auditoría).
- Puedes usar una base de datos relacional, NoSQL, o incluso un data lake, dependiendo de tus necesidades.

**En resumen:** Axon Server es para los *eventos de dominio*, mientras que tu otro data store es para las *vistas materializadas*.

## 4. Creación de Vistas Materializadas:

- **Son Tablas en la Base de Datos:** Las vistas materializadas son, por ejemplo, tablas en una base de datos (u otras estructuras optimizadas para tu data store).
- **Axon Framework y Proyecciones:** Axon Framework te ayuda a crear proyecciones que consumen los eventos de dominio y actualizan estas tablas.

## 5. Persistencia a Largo Plazo:

- **Eventos de Dominio:** Lo que *necesitas* persistir a largo plazo son los *eventos de dominio transformados* en Axon Server. Estos eventos son la fuente de la verdad y te permiten recrear cualquier agregado o vista materializada en el futuro.
- **Vistas Materializadas:** Puedes optar por persistir *también* las vistas materializadas, pero esto es opcional. Si tienes los eventos de

dominio, siempre puedes recrear las vistas materializadas.

#### **Beneficios de Persistir las Vistas Materializadas:**

- **Mayor rendimiento en las consultas:** Evitas tener que recalcular las vistas materializadas cada vez que necesitas consultarlas.
- **Disponibilidad de datos históricos:** Puedes consultar las vistas materializadas incluso si Axon Server no está disponible.

#### **Desventajas de Persistir las Vistas Materializadas:**

- **Mayor espacio de almacenamiento:** Necesitas almacenar tanto los eventos de dominio como las vistas materializadas.
- **Mayor complejidad:** Necesitas garantizar que las vistas materializadas estén sincronizadas con los eventos de dominio.

#### **6. Creación "On-the-Fly" vs. Persistencia Continua de Proyecciones y Vistas Materializadas:**

- **Proyecciones:**
  - Las proyecciones están *siempre* consumiendo eventos y actualizando las vistas materializadas.
  - No se crean "on-the-fly". Son procesos continuos.
- **Vistas Materializadas:**
  - Las vistas materializadas se *crean* una vez y se *actualizan continuamente* por las proyecciones.
  - No se crean "on-the-fly".

#### **En resumen:**

- **Persistir SIEMPRE:** Los eventos de dominio transformados en Axon Server (Event Store).
- **Persistir OPCIONALMENTE:** Las vistas materializadas en otro data store (si necesitas mayor rendimiento en las consultas o disponibilidad de datos históricos).

#### **Consejos Adicionales:**

- **Snapshots:** Considera usar "snapshots" en Axon Framework para reducir el tiempo necesario para recrear el estado de los agregados. Un snapshot es una copia del estado del agregado en un momento dado.
- **Escalabilidad:** Diseña tu solución para que sea escalable. Puedes usar Axon Server en un cluster para garantizar la alta disponibilidad y el rendimiento.
- **Monitorización:** Monitoriza tu solución para identificar cuellos de botella y optimizar el rendimiento.

Espero que esto aclare tus dudas. ¡No dudes en preguntar si necesitas más ayuda!

ok, me queda claro. Cual crees que seria la mejor propuesta de vista materializada? Actividad d usuario diaria, mensual, anual? Estas vistas materializadas, si uso una base de datos relacional, que sería, una tabla indexada por usuario? usuario y sesion? una tabla por usuario, sesion y franja temporal? como lo montarias?



Perfecto. Vamos a diseñar la estructura de la vista materializada, considerando el caso de uso de auditoría y la necesidad de consultar la actividad de un usuario en un período de tiempo específico.

#### **Análisis de Requisitos:**

- **Auditoría:** El equipo de auditoría necesita poder revisar la actividad de un usuario en un rango de fechas dado.
- **Granularidad:** Necesitamos determinar la granularidad de las vistas materializadas (diaria, mensual, anual) para equilibrar el espacio de almacenamiento y el rendimiento de las consultas.
- **Indexación:** La indexación es clave para optimizar las consultas.

#### **Propuesta de Vista Materializada:**

Creo que una combinación de granularidad diaria y mensual es la mejor opción:

##### **1. Tabla Principal: Actividad Diaria del Usuario**

- Nombre: user\_activity\_daily
- Columnas:
  - ◆ user\_id (VARCHAR, clave foránea a la tabla de usuarios)
  - ◆ session\_id (VARCHAR)
  - ◆ activity\_date (DATE, representa el día de la actividad)
  - ◆ event\_timestamp (TIMESTAMP)
  - ◆ event\_type (VARCHAR, por ejemplo, UserLoggedIn, FundsTransferred, AccountMovementsRetrieved)
  - ◆ event\_details (JSON, contiene información específica del evento)
- Índices:
  - ◆ user\_id (Índice estándar)
  - ◆ activity\_date (Índice estándar)
  - ◆ user\_id, activity\_date (Índice compuesto, muy importante para consultas por usuario y rango de fechas)
  - ◆ session\_id (Índice para buscar la actividad por sesión)
- Comentarios:
  - ◆ Cada fila en esta tabla representa un solo evento de actividad de un usuario en un día específico.
  - ◆ El campo event\_details almacena la información específica del evento en formato JSON.
  - ◆ El índice compuesto user\_id, activity\_date es fundamental para optimizar las consultas del equipo de auditoría.

## 2. Tabla de Agregación: Actividad Mensual del Usuario (Opcional)

- Nombre: user\_activity\_monthly
- Columnas:
  - ◆ user\_id (VARCHAR, clave foránea a la tabla de usuarios)
  - ◆ activity\_month (DATE, representa el primer día del mes)
  - ◆ login\_count (INT, número de inicios de sesión)
  - ◆ transfer\_count (INT, número de transferencias realizadas)
  - ◆ total\_transfer\_amount (DECIMAL, monto total transferido)
  - ◆ movement\_count (INT, número de consultas de movimientos)
  - ◆ ... (otros campos agregados relevantes)
- Índices:
  - ◆ user\_id (Índice estándar)
  - ◆ activity\_month (Índice estándar)
  - ◆ user\_id, activity\_month (Índice compuesto)
- Comentarios:
  - ◆ Cada fila en esta tabla representa un resumen de la actividad de un usuario en un mes específico.
  - ◆ Esta tabla se actualiza *a partir* de la tabla user\_activity\_daily.
  - ◆ Esta tabla puede acelerar algunas consultas, pero no es estrictamente necesaria.

### Justificación:

- **Granularidad Diaria:** La tabla user\_activity\_daily proporciona la granularidad necesaria para auditorías detalladas.
- **Rendimiento de las Consultas:** El índice compuesto user\_id, activity\_date permite al equipo de auditoría consultar la actividad de un usuario en un rango de fechas de forma eficiente.
- **Agregación Mensual (Opcional):** La tabla user\_activity\_monthly puede acelerar las consultas que necesitan un resumen mensual de la actividad.

### Implementación:

1. **Proyección:** En tu proyección de Axon, consume los eventos de dominio y actualiza las tablas user\_activity\_daily y user\_activity\_monthly.
2. **Transacciones:** Asegúrate de que las actualizaciones a ambas tablas se realicen dentro de una transacción para garantizar la consistencia de los datos.
3. **Agregación Mensual:** Puedes actualizar la tabla user\_activity\_monthly de forma asíncrona, por ejemplo, al final de cada día, o puedes actualizarla en tiempo real cada vez que recibes un evento.

### Ejemplo de Consulta:

Para obtener la actividad de un usuario en un rango de fechas, el equipo de auditoría podría ejecutar la siguiente consulta:

sql

Copy

```
SELECT
    event_timestamp,
    event_type,
    event_details
FROM
    user_activity_daily
WHERE
    user_id = 'usuario123'
    AND activity_date BETWEEN '2025-06-01' AND '2025-06-30'
ORDER BY
    event_timestamp;
```

#### Otras Consideraciones:

- **Partitioning:** Si tienes un gran número de usuarios, considera usar partitioning en la tabla user\_activity\_daily para mejorar el rendimiento de las consultas.
- **Compresión:** Considera usar compresión en las tablas para reducir el espacio de almacenamiento.
- **Archiving:** Después de un cierto período de tiempo, puedes archivar los datos de las tablas user\_activity\_daily y user\_activity\_monthly a un almacenamiento más barato.

Espero que esta propuesta te sea útil. ¡Avísame si tienes alguna pregunta más! Esta tabla de actividad diaria puede tener millines de registros, y tenemos que mantener la historia de años. Igual para Axon Server, que debe persistir los eventos durante años? Que estrategia podemos seguir? Una única tabla con la información de los 15 años? más de una tabla? una por año? Como lo ves? Que tipo de data store seria el más apropiado consierando la cantidad de información y tiempo a persistirla?



Tienes razón, la tabla de actividad diaria puede crecer enormemente, y mantener la historia durante años presenta desafíos significativos tanto para la base de datos de la vista materializada como para el Event Store de Axon Server. Vamos a explorar estrategias para abordar este problema:

#### Estrategias para la Tabla de Actividad Diaria (Vista Materializada):

Dado el volumen y la necesidad de mantener la historia a largo plazo, es crucial adoptar una estrategia de particionamiento o archiving. Aquí tienes varias opciones:

##### 1. Partitioning por Año (Recomendado):

- Crea una tabla principal (user\_activity\_daily) y luego crea particiones (sub-tablas) por año.

- Ejemplo: user\_activity\_daily\_2025, user\_activity\_daily\_2026, user\_activity\_daily\_2027, etc.
- En la mayoría de las bases de datos relacionales (PostgreSQL, MySQL, etc.), puedes usar partitioning nativo para lograr esto.
- **Ventajas:**
  - ◆ **Mejor rendimiento de las consultas:** Puedes consultar solo las particiones relevantes para un rango de fechas específico.
  - ◆ **Gestión más sencilla:** Puedes archivar o eliminar particiones antiguas de forma independiente.
  - ◆ **Escalabilidad:** Facilita la escalabilidad horizontal de la base de datos.
- **Desventajas:**
  - ◆ **Mayor complejidad:** Requiere configurar y mantener el partitioning.
- **Ejemplo (PostgreSQL):**  

```
sql
Copy
```

```
-- Tabla principal
CREATE TABLE user_activity_daily (
  user_id VARCHAR,
  session_id VARCHAR,
  activity_date DATE,
  event_timestamp TIMESTAMP,
  event_type VARCHAR,
  event_details JSON
) PARTITION BY RANGE (activity_date);

-- Partición para 2025
CREATE TABLE user_activity_daily_2025 PARTITION OF
user_activity_daily
FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');

-- Partición para 2026
CREATE TABLE user_activity_daily_2026 PARTITION OF
user_activity_daily
FOR VALUES FROM ('2026-01-01') TO ('2027-01-01');

-- (Crear particiones para cada año)
```

## 2. Archiving a un Almacenamiento Más Barato:

- Después de un cierto período de tiempo (por ejemplo, 5 años), puedes mover los datos a un almacenamiento más barato (por

ejemplo, un data lake, un sistema de archivos, etc.).

- Puedes usar una herramienta de ETL (Extract, Transform, Load) para automatizar este proceso.
- **Ventajas:**
  - ◆ **Reduce los costos de almacenamiento.**
  - ◆ **Mejora el rendimiento de las consultas en la base de datos principal.**
- **Desventajas:**
  - ◆ **Mayor complejidad:** Requiere configurar y mantener el proceso de archiving.
  - ◆ **Consultas más lentas a los datos archivados.**

### 3. Usar una Base de Datos Columnar:

- Considera usar una base de datos columnar (por ejemplo, ClickHouse, Apache Druid) en lugar de una base de datos relacional tradicional.
- Las bases de datos columnares están optimizadas para consultas analíticas y pueden manejar grandes cantidades de datos de forma eficiente.
- **Ventajas:**
  - ◆ **Mejor rendimiento de las consultas analíticas.**
  - ◆ **Mejor compresión de datos.**
- **Desventajas:**
  - ◆ **Puede ser más costoso que una base de datos relacional tradicional.**
  - ◆ **Puede ser más difícil de configurar y mantener.**

### 4. Data Lake:

- Almacena los eventos en un data lake (por ejemplo, Apache Hadoop, Amazon S3, Azure Data Lake Storage).
- Utiliza herramientas de procesamiento de datos (por ejemplo, Apache Spark, Apache Flink) para analizar los datos en el data lake.
- **Ventajas:**
  - ◆ **Escalabilidad masiva.**
  - ◆ **Flexibilidad:** Puedes almacenar datos en cualquier formato.
- **Desventajas:**
  - ◆ **Mayor complejidad:** Requiere configurar y mantener el data lake y las herramientas de procesamiento de datos.
  - ◆ **Consultas más lentas (en general):**

### Estrategias para el Event Store de Axon Server:

#### 1. Snapshots (Esencial):

- Utiliza snapshots para reducir el tiempo necesario para recrear el estado de los agregados.
- Configura Axon Framework para crear snapshots periódicamente (por ejemplo, cada 100 eventos).

#### 2. Archiving (Opcional):

- Puedes archivar los eventos antiguos del Event Store a un almacenamiento más barato.
- Sin embargo, esto significa que no podrás recrear el estado de tu aplicación a partir de los eventos si los necesitas.

### 3. AxonDB Clustering (Escalabilidad):

- Ejecuta Axon Server en un cluster para garantizar la alta disponibilidad y el rendimiento.
- AxonDB está diseñado para ser escalable horizontalmente.

#### Tipo de Data Store:

La elección del tipo de data store depende de tus requisitos específicos:

- **Base de Datos Relacional (PostgreSQL, MySQL):**

- Buena opción si necesitas ACID

Explícame mejor como va el tema del snapshot en Axon Server?



¡Claro! Vamos a profundizar en cómo funcionan los snapshots en Axon Framework y Axon Server, y cómo te ayudan a mejorar el rendimiento y la escalabilidad de tu aplicación de Event Sourcing.

#### ¿Qué es un Snapshot?

En esencia, un snapshot es una instantánea del estado de un agregado (Aggregate) en un momento específico en el tiempo. En lugar de tener que reproducir todos los eventos desde el principio para reconstruir el estado actual de un agregado, puedes cargar el snapshot más reciente y luego reproducir solo los eventos que ocurrieron *después* de ese snapshot.

#### ¿Por qué Usar Snapshots?

- **Reducción del Tiempo de Reconstrucción:** Cuando necesitas cargar un agregado (por ejemplo, para procesar un comando), Axon Framework necesita reconstruir el estado del agregado a partir de su historial de eventos. Si el agregado tiene un gran número de eventos, esto puede llevar mucho tiempo. Los snapshots reducen significativamente este tiempo, especialmente para agregados con un historial extenso.
- **Mejor Rendimiento:** Al reducir el tiempo de reconstrucción, los snapshots mejoran el rendimiento general de tu aplicación, especialmente en escenarios donde necesitas cargar agregados con frecuencia.
- **Escalabilidad:** Al reducir la carga en el Event Store, los snapshots contribuyen a la escalabilidad de tu aplicación.

#### ¿Cómo Funcionan los Snapshots en Axon Framework?

##### 1. Configuración:



- Primero, necesitas habilitar y configurar los snapshots en tu aplicación Axon. Esto generalmente se hace a través de la configuración de Spring Boot.
- Debes especificar la frecuencia con la que quieres crear snapshots (por ejemplo, cada 100 eventos).

## 2. Creación de Snapshots:

- Después de que un agregado maneja un cierto número de comandos (configurable), Axon Framework crea un snapshot del estado actual del agregado.
- Este snapshot se almacena en un SnapshotStore.

## 3. Almacenamiento de Snapshots:

- Axon Framework proporciona varias implementaciones de SnapshotStore, incluyendo:
  - ◆ EventStoreSnapshotStore: Almacena los snapshots en el mismo Event Store que los eventos. (Es el más común)
  - ◆ JpaSnapshotStore: Almacena los snapshots en una base de datos relacional usando JPA.
  - ◆ FileSystemSnapshotStore: Almacena los snapshots en el sistema de archivos.

## 4. Carga de Agregados:

- Cuando Axon Framework necesita cargar un agregado, primero busca un snapshot en el SnapshotStore.
- Si encuentra un snapshot, lo carga y luego reproduce solo los eventos que ocurrieron *después* de la versión del snapshot.
- Si no encuentra un snapshot, reproduce *todos* los eventos desde el principio.

## Ejemplo de Configuración (Ilustrativo):

java  
Copy

```
@Configuration
public class AxonConfig {

    @Bean
    public Snapshotter snapshotter(EventStore eventStore) {
        AggregateSnapshotter snapshotter = new
        AggregateSnapshotter(eventStore);
        return snapshotter;
    }

    @Bean
    public SnapshotTriggerDefinition snapshotTriggerDefinition(Snapshotter
    snapshotter) {
        // Create a snapshot every 100 events
        return new EventCountSnapshotTriggerDefinition(snapshotter, 100);
    }
}
```

}

### Consideraciones Importantes:

- **Snapshot Frequency:**
  - **Muy Frecuente:** Aumenta la carga de escritura en el SnapshotStore.
  - **Poco Frecuente:** Aumenta el tiempo de reconstrucción de los agregados.
  - **Encuentra un Equilibrio:** Experimenta para encontrar la frecuencia óptima para tu aplicación.
- **Snapshot Store:**
  - **Elige la Implementación Correcta:** Elige la implementación de SnapshotStore que mejor se adapte a tus necesidades. EventStoreSnapshotStore es generalmente una buena opción por simplicidad.
- **Tamaño de los Snapshots:**
  - **Snapshots Grandes:** Pueden ocupar mucho espacio de almacenamiento.
  - **Snapshots Pequeños:** Pueden no contener suficiente información para reducir significativamente el tiempo de reconstrucción.
  - **Optimiza el Tamaño:** Intenta optimizar el tamaño de tus snapshots eliminando datos innecesarios.
- **Serialización:**
  - **Usa una Serialización Eficiente:** Utiliza un formato de serialización eficiente (como Avro o Protocol Buffers) para reducir el tamaño de los snapshots.
- **Inmutabilidad:**
  - **Garantiza la Inmutabilidad:** Asegúrate de que tus agregados sean inmutables. Esto evitará problemas de concurrencia y corrupción de datos.

### En Resumen:

Los snapshots son una técnica valiosa para mejorar el rendimiento y la escalabilidad de tus aplicaciones de Event Sourcing. Al almacenar instantáneas del estado de los agregados, puedes reducir significativamente el tiempo necesario para reconstruirlos, lo que a su vez mejora el rendimiento general de tu aplicación. Recuerda configurar los snapshots correctamente y elegir una implementación de SnapshotStore que se adapte a tus necesidades.

ok, gracias, entiendo el concepto de snapshot, pero en nuestro caso el agregado es por sesion de usuario, contiene el sessionId, userId y la fecha de inicio de la sesion, no tenemos mas información calculao tras procesado de los eventos de dominio asociados. Luego tengo los eventos de dominio vinculados a esta instancia de agregado (los eventos que son de un usuario y sesión concreta). Que me aporta en esta caso un snapshot?



En tu caso específico, donde el agregado "Sesión de Usuario" contiene solo información básica (sessionId, userId, fecha de inicio), es cierto que el beneficio directo de los snapshots es *mínimo* para ese agregado en particular.

#### **Por qué el Beneficio es Mínimo:**

- **Estado Pequeño:** El estado del agregado "Sesión de Usuario" es muy pequeño. Reconstruir este estado a partir de los eventos de dominio es muy rápido, incluso sin snapshots.
- **Pocos Eventos:** Es probable que haya relativamente pocos eventos *directamente* asociados con el agregado "Sesión de Usuario". Los eventos más numerosos (como FundsTransferredEvent o AccountMovementsRetrievedEvent) están más relacionados con las cuentas o las transferencias, no con la sesión en sí misma.

#### **Entonces, ¿Snapshots Inútiles? No del Todo:**

Aunque el beneficio directo para el agregado "Sesión de Usuario" es pequeño, los snapshots *pueden* seguir siendo útiles en el contexto general de tu aplicación, pero no de la forma tradicional.

#### **Posibles Beneficios Indirectos:**

##### **1. Consistencia de Datos:**

- Aunque el agregado "Sesión de Usuario" en sí mismo no se beneficia mucho, *otros* agregados en tu sistema podrían beneficiarse de los snapshots.
- Al tener una estrategia de snapshots consistente en toda tu aplicación, reduces el riesgo de inconsistencias entre diferentes agregados.
- Podrías considerar una estrategia de snapshots que aplique a todos los agregados para mantener la consistencia.

##### **2. Preparación para el Futuro:**

- Aunque el agregado "Sesión de Usuario" sea simple ahora, podría evolucionar en el futuro y almacenar más información.
- Habilitar los snapshots ahora te prepara para ese futuro sin tener que hacer cambios importantes en tu código.

##### **3. Simplificación de la Infraestructura:**

4. Si utilizas EventStoreSnapshotStore, el uso de snapshot simplificará la configuración y administración de snapshots.

#### **Alternativas a los Snapshots Tradicionales:**

Dado que el beneficio directo de los snapshots tradicionales es limitado, considera estas alternativas:

### 1. Enfoque en Otros Agregados:

- Si tienes otros agregados en tu sistema que tienen un historial de eventos extenso y un estado complejo, enfoca tus esfuerzos de optimización en esos agregados.
- Aplica snapshots a esos agregados para reducir el tiempo de reconstrucción y mejorar el rendimiento.

### 2. Caché (Cache):

- Considera usar una caché para almacenar las instancias del agregado "Sesión de Usuario".
- Como el estado es pequeño, puedes cargar el agregado una vez y luego almacenarlo en la caché para un acceso rápido.

### 3. java

Copy

```
import org.springframework.cache.annotation.Cacheable;
4. import org.springframework.stereotype.Service;
5.
6. @Service
7. public class UserSessionService {
8.
9.     @Cacheable("userSessions")
10.    public UserSessionAggregate getSession(String sessionId) {
11.        // Load the aggregate from the Event Store
12.        // ...
13.    }
14. }
15.
```

- **Consideraciones:**

- ◆ Usa un mecanismo de invalidación de caché para garantizar que los datos en la caché estén actualizados.
- ◆ Elige una implementación de caché que se adapte a tus necesidades (por ejemplo, Caffeine, Redis).

### 16. Sin Snapshots (Si el Rendimiento es Aceptable):

- Si el rendimiento de tu aplicación es aceptable sin snapshots, simplemente puedes omitirlos.
- En este caso, Axon Framework siempre reproducirá todos los eventos para reconstruir el estado del agregado.

### En Resumen:

Para el agregado "Sesión de Usuario", los snapshots tradicionales probablemente no te proporcionen muchos beneficios. Considera las alternativas (caché, enfoque en otros agregados) o simplemente omite los snapshots si el rendimiento es aceptable. La clave es analizar tus requisitos específicos y elegir la estrategia que mejor se adapte a tus necesidades.

Para mi caso de uso de diario electronico con el objetivo de ver la actividad de un usuario en un periodo concreto, a parte del agregado sesion de usuario, ves algun otro agregado válido?



En el contexto específico de tu caso de uso de "diario electrónico" y con el objetivo de rastrear la actividad del usuario para auditoría, además del agregado "Sesión de Usuario", hay otros agregados que podrían ser relevantes, dependiendo de la complejidad de tu dominio y de los tipos de interacciones que quieras modelar.

Aquí hay algunas opciones a considerar:

### 1. Agregado "Cuenta" (Account):

- **Información Clave:** Podría contener información sobre la cuenta del usuario, como el tipo de cuenta (ahorros, corriente), el saldo, la fecha de creación, etc.
- **Eventos Relevantes:**
  - ◆ AccountCreatedEvent: Cuando se crea una nueva cuenta.
  - ◆ AccountUpdatedEvent: Cuando se actualiza la información de la cuenta.
  - ◆ FundsDepositedEvent: Cuando se depositan fondos en la cuenta.
  - ◆ FundsWithdrawnEvent: Cuando se retiran fondos de la cuenta.
- **Justificación:** Si quieres rastrear el estado y las modificaciones de las cuentas a lo largo del tiempo, este agregado es útil.

### 2. Agregado "Transferencia" (Transfer):

- **Información Clave:** Podría contener información sobre una transferencia de fondos, como el ID de la transferencia, la cuenta de origen, la cuenta de destino, el monto, la fecha y hora de la transferencia, el estado de la transferencia, etc.
- **Eventos Relevantes:**
  - ◆ TransferInitiatedEvent: Cuando se inicia una transferencia.
  - ◆ TransferApprovedEvent: Cuando se aprueba una transferencia (si hay un proceso de aprobación).
  - ◆ TransferExecutedEvent: Cuando se ejecuta la transferencia.
  - ◆ TransferFailedEvent: Cuando falla una transferencia.
- **Justificación:** Si quieres rastrear el ciclo de vida de las transferencias, desde su inicio hasta su finalización, este agregado es valioso.

### 3. Agregado "Notificación" (Notification):

- **Información Clave:** Podría contener información sobre una

notificación enviada al usuario, como el ID de la notificación, el tipo de notificación (email, SMS), el destinatario, el contenido de la notificación, la fecha y hora de envío, el estado de la entrega, etc.

- **Eventos Relevantes:**

- ◆ NotificationCreatedEvent: Cuando se crea una nueva notificación.
- ◆ NotificationSentEvent: Cuando se envía la notificación.
- ◆ NotificationDeliveredEvent: Cuando se entrega la notificación.
- ◆ NotificationFailedEvent: Cuando falla el envío de la notificación.

- **Justificación:** Si quieres rastrear el envío de notificaciones a los usuarios, este agregado es útil.

#### 4. Agregado "Intento de Login" (LoginAttempt):

- **Información Clave:** Permite hacer un seguimiento de los intentos de inicio de sesión
- **Eventos Relevantes:**
  - ◆ LoginAttemptedEvent: Se intenta un inicio de sesión.
  - ◆ LoginSucceededEvent: Inicio de sesión correcto
  - ◆ LoginFailedEvent: Fallo en el inicio de sesión.

#### Cómo Elegir los Agregados Correctos:

- **Considera tus Requisitos:** ¿Qué tipo de información necesitas rastrear para tu caso de uso de auditoría? ¿Qué tipo de consultas necesita realizar el equipo de auditoría?
- **Empieza de Forma Sencilla:** No intentes modelar todo de una vez. Empieza con los agregados más importantes y luego añade más agregados a medida que sea necesario.
- **Prueba y Refina:** A medida que implementes tu solución, prueba y refina tu modelo de agregados. Es posible que descubras que necesitas añadir, eliminar o modificar agregados a medida que aprendas más sobre tu dominio.
- **Modelo de Lectura vs Modelo de Escritura:** En el concepto de CQRS, te puedes plantear los modelos de lectura y escritura de forma separada. Quizá tu modelo de escritura esté más centrado en las operaciones, y tu modelo de lectura sea más sencillo de forma que las proyecciones que construyas para tu modelo de lectura se basen en combinar eventos de diferentes fuentes.

#### Justificación de la Necesidad de Agregados Adicionales:

Con solo el agregado "Sesión de Usuario", puedes rastrear la actividad del usuario por sesión, pero no tienes información sobre el *qué* de esa actividad. Por ejemplo, puedes saber que el usuario realizó una transferencia, pero no sabes los detalles de la transferencia (cuentas, monto, etc.).

Al añadir agregados como "Cuenta" y "Transferencia", puedes rastrear el *qué* de la actividad del usuario y proporcionar al equipo de auditoría una visión más completa de la actividad del usuario.

#### En tu Caso Específico:

Dado que tu objetivo es construir un diario electrónico para auditoría, te

recomendaría empezar con:

1. **Agregado "Sesión de Usuario" (Ya lo tienes):** Para rastrear la actividad por sesión.
2. **Agregado "Transferencia":** Para rastrear los detalles de las transferencias.
3. **Agregado "Intento de Login":** Para registrar los intentos de inicios de sesión

A medida que evolucione tu solución, considera añadir otros agregados según sea necesario. Recuerda que el modelo de agregados debe reflejar tus requisitos de negocio y facilitar la implementación de tu caso de uso de auditoría.

¿Tienes alguna otra pregunta?

Claro, per aquí te he puesto un caso de uso muy concreto de interacción de usuario, que básicamente el usuario hace una transferencia, pero operativa que un usuario puede hacer es muy grande (contratar productos, fondos, amortizar hipoteca, consultar documentación, consultar tarjetas, aumentar limites tarjeta, cancelar tarjeta, esto puede ser un numero de eventos de dominio enorme y el nivel de agregacion muy grande. Igual tendría mas sentido agregar por operativa lanzada desde alguna opción de menu de la aplicación, que actualmente contiene Posicion Global, Inversion, Portfolios, Transferencias, Tarjetas, Pagos,



¡Exacto! Estás dando en el clavo. Si tienes una gran variedad de operaciones que los usuarios pueden realizar, y el objetivo es construir un diario electrónico completo, entonces agregar por "operativa" o "funcionalidad" tiene mucho sentido.

### **Reconsiderando los Agregados: Enfoque en Funcionalidades**

En lugar de centrarnos en agregados que representan entidades (como "Cuenta" o "Transferencia"), vamos a enfocarnos en agregados que representan *acciones* o *procesos de negocio* iniciados por el usuario.

#### **Propuesta de Agregados Basados en Funcionalidades:**

1. **Agregado "Sesión de Usuario" (Sigue Siendo Relevante):**
  - Como discutimos antes, contiene información básica de la sesión.
  - Los otros agregados pueden referenciar el sessionId de este agregado para vincular la actividad a una sesión específica.
2. **Agregados por Funcionalidad/Operativa:**
  - **Agregado "Transferencia" (Transfer):** (Ya lo habíamos mencionado, pero ahora lo vemos como un agregado de "funcionalidad").

- ◆ Representa el proceso completo de realizar una transferencia, desde el inicio hasta la confirmación.
- ◆ Eventos: TransferInitiatedEvent, TransferDetailsEnteredEvent, TransferConfirmedEvent, TransferExecutedEvent, TransferFailedEvent, etc.
- **Agregado "Contratación de Producto" (ProductEnrolment):**
  - ◆ Representa el proceso de contratar un nuevo producto financiero.
  - ◆ Eventos: ProductEnrolmentStartedEvent, ProductDetailsEnteredEvent, ProductTermsAcceptedEvent, ProductEnrolledEvent, ProductEnrolmentFailedEvent, etc.
- **Agregado "Amortización de Hipoteca" (MortgageAmortization):**
  - ◆ Representa el proceso de realizar una amortización anticipada de la hipoteca.
  - ◆ Eventos: MortgageAmortizationStartedEvent, AmortizationDetailsEnteredEvent, AmortizationConfirmedEvent, AmortizationExecutedEvent, AmortizationFailedEvent, etc.
- **Agregado "Cancelación de Tarjeta" (CardCancellation):**
  - ◆ Representa el proceso de cancelar una tarjeta.
  - ◆ Eventos: CardCancellationStartedEvent, CancellationReasonEnteredEvent, CancellationConfirmedEvent, CardCancelledEvent, CardCancellationFailedEvent, etc.
- **Agregado "Aumento de Límite de Tarjeta" (CardLimitIncrease):**
  - ◆ Representa el proceso de solicitar un aumento del límite de la tarjeta.
  - ◆ Eventos: CardLimitIncreaseStartedEvent, LimitIncreaseDetailsEnteredEvent, LimitIncreaseApprovedEvent, CardLimitIncreasedEvent, LimitIncreaseRejectedEvent, etc.
- ... (Y así sucesivamente para cada funcionalidad importante)
- 3. **Agregado "Consulta de Documentación" (DocumentConsultation):**
  - Representa el proceso de consultar un documento (por ejemplo, un extracto bancario).
  - Eventos: DocumentConsultationStartedEvent, DocumentSelectedEvent, DocumentDownloadedEvent, etc.

#### **Justificación del Cambio de Enfoque:**

- **Mayor Claridad:** Este enfoque se alinea mejor con las acciones que los usuarios realizan en la aplicación.
- **Mejor Agrupación:** Permite agrupar todos los eventos relacionados con una funcionalidad específica bajo un solo agregado.
- **Escalabilidad:** Facilita la adición de nuevas funcionalidades en el futuro.



- **Caso de Uso de Auditoría:** Proporciona al equipo de auditoría una visión clara de las acciones que los usuarios están realizando.

#### **Mapeo con las Opciones del Menú:**

Como mencionaste, puedes mapear estos agregados a las opciones del menú de la aplicación:

- **Posición Global:** No parece ser una acción en sí misma, sino más bien una vista de la información. No necesita un agregado.
- **Inversión:** Podría tener un agregado "Inversión" con eventos como `InvestmentStartedEvent`, `InvestmentDetailsEnteredEvent`, `InvestmentConfirmedEvent`, `InvestmentExecutedEvent`, etc.
- **Portfolios:** Podría tener un agregado "Portfolio" con eventos como `PortfolioCreatedEvent`, `PortfolioUpdatedEvent`, `PortfolioDeletedEvent`, etc.
- **Transferencias:** Agregado "Transferencia" (ya definido).
- **Tarjetas:** Podría tener un agregado "Tarjeta" con eventos como `CardCreatedEvent`, `CardActivatedEvent`, `CardDeactivatedEvent`, `CardCancelledEvent`, etc.
- **Pagos:** Podría tener un agregado "Pago" con eventos como `PaymentScheduledEvent`, `PaymentExecutedEvent`, `PaymentFailedEvent`, etc.

#### **Ventajas Adicionales:**

- **Control de Acceso:** Puedes usar estos agregados para implementar el control de acceso. Por ejemplo, puedes verificar si un usuario tiene permiso para realizar una transferencia antes de permitirle iniciar el proceso.
- **Validación:** Puedes usar los agregados para validar los datos que los usuarios introducen. Por ejemplo, puedes verificar si el monto de una transferencia es válido antes de permitir que se ejecute.

#### **Cómo Implementar:**

1. **Crear los Agregados:** Crea las clases Java que representan los agregados.
2. **Definir los Eventos:** Define los eventos que pueden ocurrir en cada agregado.
3. **Crear los Command Handlers:** Crea los command handlers que manejan los comandos que inician los procesos de negocio.
4. **Crear los Event Handlers:** Crea los event handlers que actualizan el estado de los agregados cuando ocurren los eventos.
5. **Actualizar las Proyecciones:** Actualiza tus proyecciones para consumir los nuevos eventos y construir las vistas materializadas que necesitas.

Espero que esta nueva perspectiva te sea útil. ¡Avísame si tienes alguna pregunta más!

con el agregado de sesión de usuario veía claro tener una vista materializada de actividad de usuario y sus particiones anuales. Entiendo que las proyecciones consumen eventos de dominio y actualizan vistas, luego la

dependencia entre proyección y vista es clara. Pero que relación existe entre agregado y vista? o entre agregado y proyección?



Excelente pregunta. La relación entre Agregado, Proyección y Vista Materializada es fundamental en Event Sourcing y CQRS. Vamos a clarificarlo:

### **Relación Agregado - Proyección - Vista Materializada:**

#### **1. Agregado:**

- **Representa:** Un concepto del dominio de negocio (en tu caso, podría ser "Transferencia", "Contratación de Producto", "Sesión de Usuario", etc.).
- **Contiene:** El estado y la lógica de negocio relacionados con ese concepto.
- **Genera:** Eventos de dominio que representan cambios de estado.
- **Responsabilidad:** Garantizar la consistencia del estado *dentro* del agregado.
- **Persistencia:** El agregado en sí *no se persiste directamente*. Lo que se persiste son los eventos que genera.

#### **2. Proyección:**

- **Representa:** Un proceso que *consume* eventos de dominio.
- **Función:** Transforma los eventos en un formato que es más adecuado para un caso de uso específico.
- **Crea/Actualiza:** Vistas materializadas.
- **Dependencia:** Depende de los eventos de dominio. Si cambias los eventos de dominio, es posible que tengas que actualizar la proyección.
- **Responsabilidad:** Construir y mantener las vistas materializadas.

#### **3. Vista Materializada:**

- **Representa:** Una estructura de datos optimizada para un caso de uso específico (por ejemplo, una tabla en una base de datos que almacena la actividad de un usuario en un período de tiempo).
- **Construida por:** Una proyección.
- **Consumida por:** Consultas para mostrar la información en la interfaz de usuario o para generar informes.
- **Dependencia:** Depende de la proyección. Si cambias la proyección, tendrás que regenerar la vista materializada.

### **En Resumen:**

text

Copy

Agregado --> Genera Eventos de Dominio

Eventos de Dominio --> Son Consumidos por Proyecciones

Proyecciones --> Crean/Actualizan Vistas Materializadas

Vistas Materializadas --> Son Consultadas para Mostrar Datos

#### **Relación Agregado - Proyección:**

- **Indirecta:** La relación entre el agregado y la proyección es *indirecta*. La proyección no interactúa directamente con el agregado.
- **Mediación por Eventos:** La proyección interactúa con el agregado a través de los eventos de dominio que el agregado genera.
- **Desacoplamiento:** Este desacoplamiento es un principio clave de Event Sourcing y CQRS. Permite que los agregados evolucionen independientemente de las proyecciones y viceversa.

#### **Relación Agregado - Vista Materializada:**

- **Indirecta:** La relación entre el agregado y la vista materializada es también *indirecta*. La vista materializada no conoce la existencia del agregado.
- **Mediación por Eventos y Proyecciones:** La vista materializada se construye a partir de los eventos que genera el agregado y que son procesados por la proyección.
- **Aislamiento:** Este aislamiento permite cambiar la estructura de la vista materializada sin afectar al agregado, y viceversa.

#### **Ejemplo Concreto (Agregado Transferencia - Proyección Diario Electrónico**

##### **- Vista Materializada Actividad Diaria):**

##### **1. Agregado "Transferencia":**

- Genera un evento TransferExecutedEvent cuando una transferencia se completa.

##### **2. Proyección "Diario Electrónico":**

- Consume el evento TransferExecutedEvent.
- Extrae la información relevante (ID de usuario, fecha y hora, cuentas involucradas, monto, etc.).
- Actualiza la vista materializada "Actividad Diaria" con esta información.

##### **3. Vista Materializada "Actividad Diaria":**

- Almacena la información de la transferencia en una tabla con las columnas user\_id, activity\_date, event\_timestamp, event\_type, event\_details.
- Esta tabla es consultada por el equipo de auditoría para ver la actividad del usuario.

#### **Importancia del Desacoplamiento:**

Este desacoplamiento entre agregados, proyecciones y vistas materializadas es fundamental para la escalabilidad y la mantenibilidad de tu sistema. Permite:

- **Evolución Independiente:** Puedes cambiar la lógica de negocio de un agregado sin afectar a las proyecciones o vistas materializadas, siempre y cuando sigas generando los mismos eventos de dominio.

- **Optimización de Lectura:** Puedes optimizar las vistas materializadas para casos de uso específicos sin afectar a la lógica de negocio de los agregados.
- **Adición de Nuevos Casos de Uso:** Puedes añadir nuevos casos de uso creando nuevas proyecciones que consuman los eventos existentes y construyan nuevas vistas materializadas.

Espero que esto aclare la relación entre agregados, proyecciones y vistas materializadas. ¡No dudes en preguntar si tienes alguna pregunta más! como lo que recibo en Kafka son eventos raw que deben transformarse a eventos de dominio tendré tantos listeners consumiendo estos eventos de Kafka como agregaciones tenga, no? Estos listeners de agregación transforman estos eventos raw en eventos de dominio que consumen las proyecciones de es agregación que transforman y alimentan las vistas materializadas, es así?



No necesariamente. Si bien tu planteamiento de tener un listener por agregado es *posible*, no es la única opción y podría no ser la más eficiente. Vamos a analizar las opciones y cuál podría ser la mejor para tu caso:

#### Opciones para el Listener de Kafka:

##### 1. Un Listener por Agregado (Un Listener por Tipo de Evento Raw):

- Cada listener se enfoca en un tipo específico de evento raw y lo transforma en un evento de dominio específico para un agregado en particular.
- Ejemplo:
  - ◆ Listener 1: Consume eventos MethodExecuted con method = "executeTransfer" y los transforma en eventos FundsTransferredEvent para el agregado "Transferencia".
  - ◆ Listener 2: Consume eventos MethodExecuted con method = "productEnrolment" y los transforma en eventos ProductEnrolledEvent para el agregado "Contratación de Producto".
- **Ventajas:**
  - ◆ **Simple:** Fácil de entender y configurar.
  - ◆ **Desacoplado:** Cada listener es independiente de los demás.
- **Desventajas:**
  - ◆ **Duplicación de Código:** Es probable que tengas duplicación de código para la conexión a Kafka, la deserialización de eventos, etc.
  - ◆ **Gestión:** Necesitas gestionar múltiples listeners.

- ♦ **Potencial Ineficiencia:** Si tienes muchos tipos de eventos raw, tendrás muchos listeners, lo que puede ser ineficiente.

## 2. Un Listener Centralizado (Recomendado):

- Un único listener consume todos los eventos raw de Kafka.
- Este listener centralizado tiene la responsabilidad de:
  - ♦ Deserializar los eventos raw.
  - ♦ Determinar el tipo de evento de dominio que se va a crear (usando tu configuración de mapeo).
  - ♦ Crear el evento de dominio correspondiente.
  - ♦ Enviar el evento de dominio a Axon Server.
- **Ventajas:**
  - ♦ **Centralizado:** Facilita la gestión y el mantenimiento del código.
  - ♦ **Reutilización de Código:** Reduce la duplicación de código.
  - ♦ **Eficiencia:** Un solo listener es más eficiente que muchos listeners.
- **Desventajas:**
  - ♦ **Mayor Complejidad:** El listener centralizado es más complejo que un listener individual.
  - ♦ **Único Punto de Fallo:** Si el listener centralizado falla, ningún evento se procesará. (Puedes mitigar esto con estrategias de resiliencia).

## Arquitectura Propuesta (Listener Centralizado):

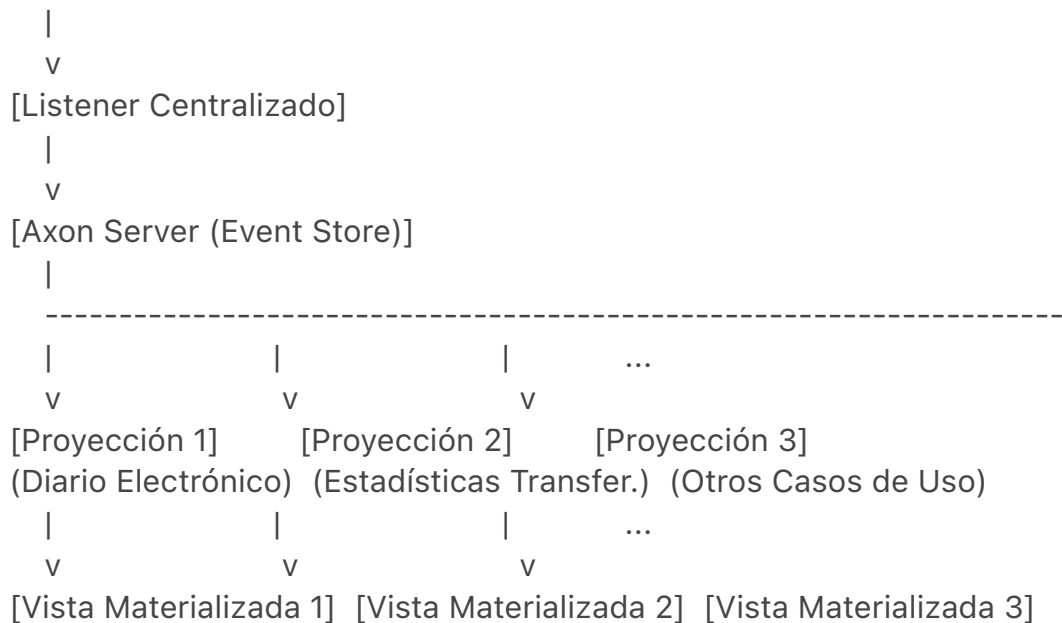
1. **Kafka Topic:** Recibe los eventos raw (MethodExecuted).
2. **Listener Centralizado:**
  - Consume todos los eventos del Kafka Topic.
  - Usa la configuración de mapeo (método -> evento de dominio) para determinar el tipo de evento de dominio que se va a crear.
  - Crea el evento de dominio.
  - Envía el evento de dominio a Axon Server.
3. **Axon Server:** Almacena los eventos de dominio en el Event Store.
4. **Proyecciones:**
  - Cada proyección se suscribe a los eventos de dominio que le interesan.
  - Por ejemplo, la proyección del "Diario Electrónico" se suscribe a todos los eventos.
  - La proyección de "Estadísticas de Transferencias" se suscribe solo a los eventos relacionados con transferencias.
5. **Vistas Materializadas:**
  - Las proyecciones actualizan las vistas materializadas correspondientes.

## Diagrama:

text

Copy

[Kafka Topic (Eventos Raw)]



### Cómo Implementar el Listener Centralizado:

#### 1. Crear una Clase para el Listener:

java  
Copy

```

import org.springframework.kafka.annotation.KafkaListener;
2. import org.springframework.stereotype.Component;
3. import com.fasterxml.jackson.databind.ObjectMapper;
4. import com.jcm.prospective.events.DomainEvent; // Interface para
   eventos de dominio
5.
6. @Component
7. public class CentralizedKafkaListener {
8.
9.     private final ObjectMapper objectMapper; // Inyecta un
       ObjectMapper
10.     private final EventMappingConfiguration
        eventMappingConfiguration; // Tu configuración de mapeo
11.     private final AxonGateway axonGateway; // Para enviar comandos a
        Axon
12.
13.     public CentralizedKafkaListener(ObjectMapper objectMapper,
        EventMappingConfiguration eventMappingConfiguration,
        AxonGateway axonGateway) {
14.         this.objectMapper = objectMapper;
15.         this.eventMappingConfiguration = eventMappingConfiguration;
16.         this.axonGateway = axonGateway;
17.     }
18.
19.     @KafkaListener(topics = "tu-topic-de-eventos-raw")
  
```

```

20. public void listen(String message) {
21.     try {
22.         RawEvent rawEvent = objectMapper.readValue(message,
            RawEvent.class);
23.
24.         // Mapea el nombre del método al tipo de evento de dominio
25.         Class<? extends DomainEvent> eventClass =
            eventMappingConfiguration.getEventClass(rawEvent.getMethod());
26.
27.         // Crea el evento de dominio (puedes usar reflection o un
patrón factory)
28.         DomainEvent domainEvent = createDomainEvent(eventClass,
            rawEvent);
29.
30.         // Publica el evento en Axon
31.         axonGateway.publish(domainEvent);
32.
33.     } catch (Exception e) {
34.         // Maneja el error (log, retry, dead-letter queue)
35.         e.printStackTrace();
36.     }
37. }
38.
39. private DomainEvent createDomainEvent(Class<? extends
    DomainEvent> eventClass, RawEvent rawEvent) {
40.     // Lógica para crear una instancia del evento de dominio
41.     // (Puedes usar reflection, un patrón factory, o un ObjectMapper)
42.     // ...
43. }
44. }
45.

```

#### 46. Crear la Configuración de Mapeo:

- Implementa EventMappingConfiguration para leer la configuración de mapeo (YAML, JSON, propiedades) que mapea los nombres de los métodos a los tipos de eventos de dominio.

#### 47. Crear una Interfaz para los Eventos de Dominio:

```

java
Copy

```

```

    public interface DomainEvent {
48.     // Métodos comunes a todos los eventos de dominio
49. }

```

50.

### 51. Implementar los Eventos de Dominio:

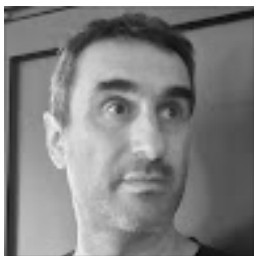
- Crea clases que implementen la interfaz DomainEvent para cada tipo de evento de dominio (por ejemplo, UserLoggedInEvent, FundsTransferredEvent).

#### Justificación:

- **Centralizado:** Facilita la gestión y el mantenimiento del código.
- **Reutilización de Código:** Reduce la duplicación de código.
- **Escalabilidad:** Un solo listener es más eficiente que muchos listeners.

Esta arquitectura proporciona una forma flexible y escalable de procesar tus eventos raw y transformarlos en eventos de dominio que pueden ser consumidos por tus proyecciones.

me pierdo un poco. Veo perfecto tener un único listener que transforma los eventos raw de Kafka en klos eventos de dominio y las proyecciones que consumen los eventos de dominio que requieran. Por otro lado dices que las agregaciones generan los eventos de dominio, pero el evento de dominio lo está generando el listener de Kafka que procesa el evento raw y lo transforma. Donde situamos aquí el agregado sesion de usuario y el agregado de tarjetas, por ejemplo?



¡Tienes razón! Me expliqué mal y causé confusión. Vamos a clarificar el papel de los agregados y cómo encajan en el flujo de procesamiento de eventos.

#### Corrección del Flujo de Eventos:

1. **Eventos Raw (Kafka):** Los eventos raw (como MethodExecuted) llegan a tu topic de Kafka.
2. **Listener Centralizado (Transformador):** El listener centralizado *no* genera directamente los eventos de dominio, sino que:
  - Lee el evento raw.
  - Determina *qué comando debe enviarse* basándose en el evento raw (usando la configuración de mapeo).
  - Crea y envía un *comando* a Axon Framework.
3. **Comandos (Commands):**
  - Los comandos representan la *intención* de realizar una acción (por ejemplo, TransferFundsCommand, EnrollInProductCommand).
  - Los comandos son enviados a los agregados.
4. **Agregados (Ahora Sí):**
  - Los agregados *reciben* los comandos.
  - Los agregados validan si la acción es permitida (basándose en su



estado actual y en las reglas de negocio).

- Si la acción es válida, el agregado *genera* uno o más *eventos de dominio* que representan los cambios de estado que resultan de la acción.
- Ejemplo: El agregado "Transferencia" recibe un comando TransferFundsCommand, valida que la cuenta tenga fondos suficientes y genera un evento FundsTransferredEvent.

## 5. Eventos de Dominio (Axon Server):

- Axon Server almacena los eventos de dominio.
- Los eventos de dominio se utilizan para actualizar el estado del agregado y para construir las vistas materializadas.

## 6. Proyecciones (Construcción de Vistas Materializadas):

- Las proyecciones escuchan los eventos de dominio en Axon Server.
- Cuando reciben un evento de dominio, actualizan las vistas materializadas correspondientes.

### Esquema Actualizado:

text

Copy

[Kafka Topic (Eventos Raw)]

|  
v

[Listener Centralizado] -- Lee el evento raw, crea y envía un COMANDO -->

[Axon Command Bus]

^

|

[Agregado (Ej: Transferencia)] -- Recibe el COMANDO, genera EVENTOS DE DOMINIO --> [Axon Event Bus]

|  
v

[Axon Server (Event Store)]

|

-----  
|                      |                      |                      ...  
v                      v                      v

[Proyección 1]           [Proyección 2]           [Proyección 3]

(Diario Electrónico) (Estadísticas Transfer.) (Otros Casos de Uso)

|                      |                      |                      ...  
v                      v                      v

[Vista Materializada 1] [Vista Materializada 2] [Vista Materializada 3]

### Ejemplo Concreto (Transferencia):

1. **Evento Raw:** El servicio de transferencia genera un evento MethodExecuted en Kafka cuando se llama al método executeTransfer.
2. **Listener Centralizado:** El listener lee el evento MethodExecuted y determina que corresponde a una transferencia. Crea un comando

TransferFundsCommand con la información del evento raw (cuenta de origen, cuenta de destino, monto).

3. **Agregado "Transferencia":** El comando TransferFundsCommand se envía al agregado "Transferencia". El agregado valida que la cuenta de origen tenga fondos suficientes y que la cuenta de destino exista. Si todo es válido, el agregado genera un evento FundsTransferredEvent con la información de la transferencia.
4. **Axon Server:** Axon Server almacena el evento FundsTransferredEvent.
5. **Proyección "Diario Electrónico":** La proyección del "Diario Electrónico" recibe el evento FundsTransferredEvent y actualiza la vista materializada con la información de la transferencia.

#### En Resumen:

- **Listener de Kafka:** Transforma eventos raw en *comandos*.
- **Comandos:** Representan *intenciones*.
- **Agregados:** Reciben comandos, validan la acción y generan *eventos de dominio*.
- **Eventos de Dominio:** Representan *hechos* que han ocurrido.
- **Proyecciones:** Construyen vistas materializadas a partir de los eventos de dominio.

¡Perdón por la confusión anterior! Espero que esta explicación sea más clara. Este modelo basado en comandos y eventos de dominio es más robusto y escalable que simplemente generar eventos de dominio directamente en el listener de Kafka.

estamos generando un sistema de event sourcing para auditoria, los eventos que llegan corresponden a acciones ya hechas. No tiene sentido validar si una cuenta tiene saldo para la transferencia. Los eventos que voy a recibir son de transferencia hecha o no ejecutada



¡Ah, entiendo! Dado que tu objetivo es la *auditoría* y no la aplicación de la lógica de negocio, tienes toda la razón. En este caso, el modelado es diferente. La validación ya ha ocurrido, y los eventos que recibes son *registros* de lo que sucedió.

#### Adaptación del Flujo de Eventos para Auditoría:

Dado que los eventos que recibes representan acciones ya realizadas (o intentadas), el flujo de eventos se simplifica. No necesitas enviar comandos a los agregados. En cambio, puedes generar los eventos de dominio directamente en el listener de Kafka, como habías pensado originalmente.

#### Flujo de Eventos Simplificado:

1. **Eventos Raw (Kafka):** Los eventos raw (como MethodExecuted)

llegan a tu topic de Kafka.

## 2. **Listener Centralizado (Transformador):** El listener centralizado:

- Lee el evento raw.
- Determina el tipo de evento de dominio que se va a crear (usando la configuración de mapeo).
- Crea el evento de dominio correspondiente.
- *Publica el evento de dominio directamente en el Event Bus de Axon (sin pasar por los agregados).*

## 3. **Axon Server:**

- Axon Server almacena los eventos de dominio en el Event Store.
- Los eventos de dominio se utilizan para construir las vistas materializadas.

## 4. **Proyecciones (Construcción de Vistas Materializadas):**

- Las proyecciones escuchan los eventos de dominio en Axon Server.
- Cuando reciben un evento de dominio, actualizan las vistas materializadas correspondientes.

### **Esquema Simplificado:**

text

Copy

[Kafka Topic (Eventos Raw)]

|  
v

[Listener Centralizado] -- Lee el evento raw, crea y publica EVENTOS DE DOMINIO --> [Axon Event Bus]

|  
v

[Axon Server (Event Store)]

|

-----

|                      |                      |                      ...  
v                      v                      v

[Proyección 1]           [Proyección 2]           [Proyección 3]  
(Diario Electrónico) (Estadísticas Transfer.) (Otros Casos de Uso)

|                      |                      |                      ...  
v                      v                      v

[Vista Materializada 1] [Vista Materializada 2] [Vista Materializada 3]

### **Agregados: Ahora con un Papel Diferente (Opcional):**

#### • **Agregados como "Contenedores de Sesión":**

- Podrías seguir teniendo un agregado "Sesión de Usuario" que contenga el sessionId, el userId y la fecha de inicio de la sesión.
- Sin embargo, este agregado *no* validaría nada ni generaría eventos. Simplemente actuaría como un contenedor para la información de la sesión.
- Este agregado se crearía cuando recibes el

evento UserLoggedInEvent.

- **Agregados como "Vistas de Lectura":** Podrías crear agregados más complejos de lectura que consuman eventos y generen vistas de información. Por ejemplo "Usuario", "Actividad".

#### **Justificación:**

- **Simplicidad:** Este modelo es más simple que el modelo basado en comandos y agregados que validan la acción.
- **Eficiencia:** Evita la necesidad de enviar comandos y esperar a que los agregados los procesen.
- **Auditoría:** Se centra en el objetivo de registrar lo que ha sucedido, no en aplicar la lógica de negocio.

#### **El Listener Centralizado (Revisado):**

El listener centralizado (en este caso) sí que genera los eventos de dominio directamente tras la transformación del evento raw.

- **Eventos:** En Axon, los eventos siempre se publican a través de un EventBus. Un componente, como un Agregado, o un EventHandler, publican el evento. Axon proporciona una interfaz llamada EventMessage, pero no es obligatorio usarla.
- **Handler:** Un evento siempre tiene un handler, un método que se llama cuando se publica el evento. Por ejemplo, este handler podría ser un método en una clase con la anotación @EventHandler.

#### **Eventos: Inmutables**

En Axon los eventos deben ser inmutables ya que tienen el historial de la aplicación, un registro que tiene que prevalecer en el tiempo.

Con este enfoque simplificado, puedes construir un sistema de auditoría eficiente y fácil de mantener que registre todas las acciones que los usuarios realizan en tu aplicación.

Que me aporta este agregado de sesión que no tiene estado?



Tienes toda la razón en cuestionar la utilidad del agregado "Sesión de Usuario" si no tiene estado ni lógica de negocio. En un sistema de auditoría pura, donde solo estás registrando lo que ya ha sucedido, un agregado sin estado realmente no aporta ningún beneficio.

#### **Eliminando el Agregado "Sesión de Usuario" (Simplicidad es Clave):**

En este escenario, lo más simple y eficiente es *eliminar* el agregado "Sesión de Usuario" por completo.

#### **Flujo de Eventos Aún Más Simplificado:**

1. **Eventos Raw (Kafka):** Los eventos raw (MethodExecuted) llegan a tu topic de Kafka.

## 2. Listener Centralizado (Transformador): El listener centralizado:

- Lee el evento raw.
- Determina el tipo de evento de dominio que se va a crear (usando la configuración de mapeo).
- Crea el evento de dominio correspondiente.
- *Incluye el sessionId y el userId directamente en el evento de dominio.*
- Publica el evento de dominio directamente en el Event Bus de Axon.

## 3. Axon Server:

- Axon Server almacena los eventos de dominio en el Event Store.
- Los eventos de dominio se utilizan para construir las vistas materializadas.

## 4. Proyecciones (Construcción de Vistas Materializadas):

- Las proyecciones escuchan los eventos de dominio en Axon Server.
- Cuando reciben un evento de dominio, actualizan las vistas materializadas correspondientes.

### Esquema Simplificado (Sin Agregado):

text

Copy

[Kafka Topic (Eventos Raw)]

|  
v

[Listener Centralizado] -- Lee el evento raw, crea y publica EVENTOS DE DOMINIO (con sessionId y userId) --> [Axon Event Bus]

|  
v

[Axon Server (Event Store)]

|

-----

|                      |                      |                      ...  
v                      v                      v

[Proyección 1]           [Proyección 2]           [Proyección 3]  
(Diario Electrónico) (Estadísticas Transfer.) (Otros Casos de Uso)

|                      |                      |                      ...  
v                      v                      v

[Vista Materializada 1] [Vista Materializada 2] [Vista Materializada 3]

### Justificación:

- **Simplicidad:** Este modelo es aún más simple que el anterior.
- **Eficiencia:** Elimina la necesidad de crear y gestionar un agregado sin estado.
- **Flexibilidad:** Puedes seguir utilizando el sessionId y el userId para agrupar los eventos por sesión y por usuario.

### Cómo Implementar:

1. **Eliminar el Agregado:** Elimina la clase que representa el agregado "Sesión de Usuario".
2. **Modificar los Eventos de Dominio:** Añade los campos `sessionId` y `userId` a todos los eventos de dominio que necesiten esta información.
3. **Actualizar el Listener Centralizado:** Asegúrate de que el listener centralizado extraiga el `sessionId` y el `userId` del evento raw y los incluya en los eventos de dominio.
4. **Actualizar las Proyecciones:** Actualiza tus proyecciones para leer el `sessionId` y el `userId` directamente de los eventos de dominio.

**Ejemplo:**

java

Copy

```
public class FundsTransferredEvent implements DomainEvent {
    private final String sessionId;
    private final String userId;
    private final String fromAccountId;
    private final String toAccountId;
    private final double amount;

    public FundsTransferredEvent(String sessionId, String userId, String
fromAccountId, String toAccountId, double amount) {
        this.sessionId = sessionId;
        this.userId = userId;
        this.fromAccountId = fromAccountId;
        this.toAccountId = toAccountId;
        this.amount = amount;
    }

    // Getters
}
```

**En Resumen:**

En un sistema de auditoría pura, un agregado sin estado no aporta valor. Elimina el agregado y simplifica tu arquitectura. Al incluir el `sessionId` y el `userId` directamente en los eventos de dominio, puedes seguir agrupando y filtrando los eventos por sesión y por usuario.

Este enfoque más simple es más eficiente y fácil de mantener, y te permitirá construir un sistema de auditoría robusto y escalable.