Engenharia de Softwares Escaláveis

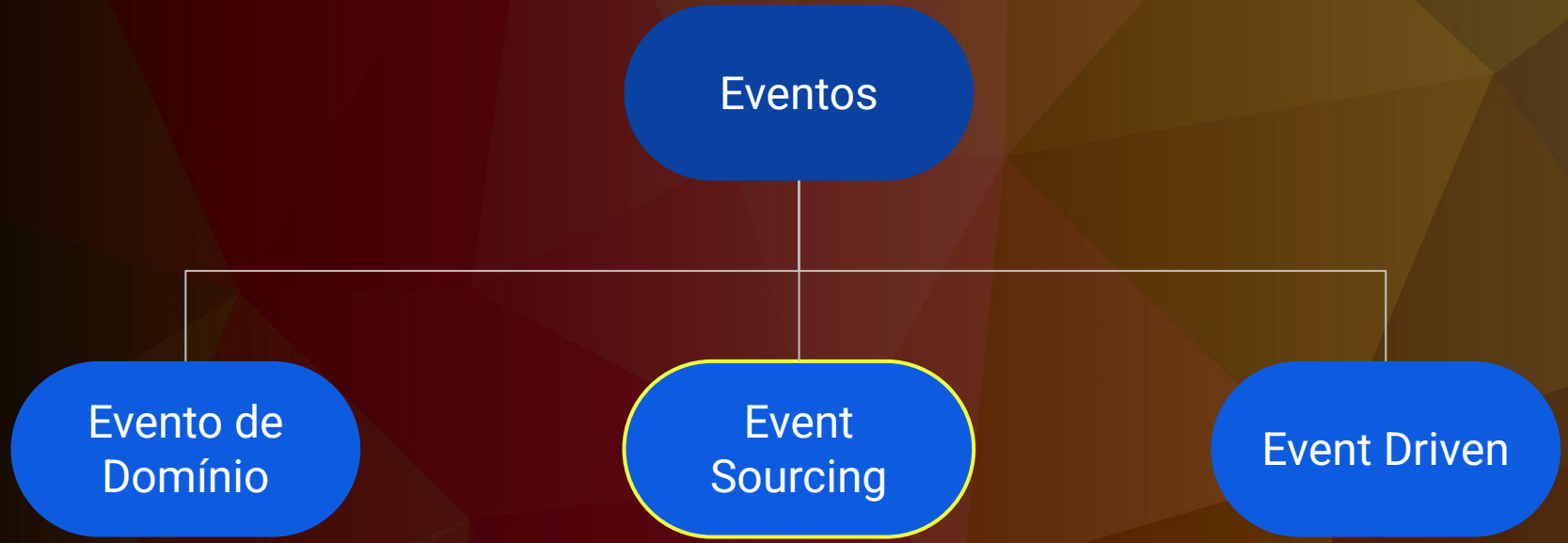
Domain-Driven Design (DDD) e Arquitetura de Softwares Escaláveis com Java

Agenda

Etapa 5: Armazenamento, Publicação e Persistência de Domain Events em Java utilizando Event Sourcing.

- Event Sourcing.
- CQRS.
- CQRS X Event Sourcing.
- Axon Framework.

Event Sourcing


Eventos
Evento de Domínio
Event Sourcing
Event Driven
É um padrão / elemento de modelagem de domínio.
Torna "as coisas que acontecem" em objetos em seu modelo de domínio.
É um padrão de modelagem de dados.
Temos uma entidade de domínio com histórico de alterações.
É um padrão de comunicação.
O sistema A publica um evento e o sistema B reage. O sistema A e o sistema B nem precisam saber um do outro.

Evento de Domínio

Representa algo que aconteceu no passado e que é significativo para o domínio do negócio.

É um fato imutável que ocorreu em um ponto específico no tempo.

Descrito em termos do domínio do negócio, utilizando a linguagem comum aos especialistas do domínio.

Representa um fato que já ocorreu e não pode ser alterado.

Exemplo: Um cliente fez um pedido, um produto foi enviado, uma fatura foi paga.

Event Sourcing

É um padrão de arquitetura onde o estado de um sistema é representado por uma sequência de eventos.

Em vez de armazenar apenas o estado atual, é armazenado o histórico completo de eventos que levaram a esse estado.

Os eventos são a única fonte de verdade sobre o estado do sistema.

Novos eventos são sempre adicionados à sequência, nunca modificados ou removidos.

Exemplo: Um banco de dados que armazena todas as transações desde a criação da conta, permitindo reconstruir o saldo atual a qualquer momento.

	Evento de Domínio	Event Sourcing
Natureza	Fato isolado e significativo	Mecanismo de persistência
Propósito	Modelar o que aconteceu no domínio	Armazenar o histórico de mudanças
Uso	Base para a modelagem de agregados e bounded contexts	Forma de implementar a persistência de agregados

Eventos de Domínio são a base do **Event Sourcing**: os eventos que são armazenados em um sistema que utiliza event sourcing são, na maioria das vezes, eventos de domínio.

Event Sourcing permite rastreabilidade e auditabilidade: ao armazenar o histórico completo de eventos, é possível rastrear as mudanças no estado do sistema e auditar as ações realizadas.

Event Sourcing facilita a implementação de CQRS - Command Query Responsibility Segregation: event sourcing e CQRS são frequentemente usados em conjunto, pois event sourcing fornece um mecanismo para armazenar o histórico de eventos, enquanto CQRS separa as responsabilidades de leitura e escrita.

Event Sourcing não é para todos os projetos:

- Evite para aplicações CRUD simples (como um blog ou um cadastro de clientes). **A complexidade não compensa**.
- Considere fortemente para **sistemas que são o "core"** do negócio, especialmente aqueles onde a auditoria, a lógica de negócio complexa e o histórico de mudanças são críticos (ex: sistemas financeiros, logística, e-commerce, saúde).

CQRS

CQRS - Command Query Responsibility Segregation é um padrão arquitetural que significa "Segregação de Responsabilidade de Comando e Consulta".

A ideia central é simples: em vez de usar o mesmo modelo de dados para escrever (alterar dados) e ler (consultar dados), separamos em dois modelos distintos.

- **Comandos:** São operações que mudam o estado do sistema. Eles representam a intenção de mudar algo. (Ex: CriarPedidoCommand, AtualizarEnderecoCommand). Eles não retornam dados, apenas acusam sucesso ou falha.
- **Consultas:** São operações que apenas leem o estado do sistema e retornam dados. Elas nunca modificam o estado. (Ex: BuscarDetalhesDoPedidoQuery, ListarTodosOsClientesQuery).

A motivação é que na maioria das aplicações tradicionais, usamos o mesmo objeto tanto para mudar o nome (usuario.setNome()) quanto para exibir o nome (usuario.getNome()).

O problema é que os requisitos para escrita e leitura são fundamentalmente diferentes:

- **Escrita (Commands):** Precisa de validação complexa, regras de negócio, consistência transacional e segurança. O foco é garantir que o sistema nunca entre em um estado inválido.
- **Leitura (Queries):** Precisa ser rápida, eficiente e formatada para a UI (Interface do Usuário). Muitas vezes, precisamos de dados de várias tabelas (JOINS), o que pode ser lento.

CQRS X Event Sourcing

1. **Comando (CQRS):** um usuário tenta fazer algo, disparando um comando `AdicionarItemAoCarrinhoCommand`.
2. **Processamento (Write Model):** o sistema direciona esse comando para um "Agregado", que contém a lógica de negócio "O estoque está disponível?".
3. **Persistência (Event Sourcing):** se a lógica de negócio for válida, o Agregado não salva seu novo estado `carrinho.setTotalItens(5)`. Em vez disso, ele publica um Evento `ItemAdicionadoAoCarrinhoEvent`.
4. **Event Store**: o sistema captura esse evento e o salva permanentemente no Event Store. Esta é a "fonte da verdade". O lado de escrita terminou seu trabalho aqui.

5. **Publicação (ES -> CQRS):** Event Store agora publica esse evento para o resto da aplicação "Ei, pessoal, um item foi adicionado ao carrinho!".
6. **Projeção (CQRS Read Model):** um componente separado, chamado "Projector" ou "Event Listener", ouve esse evento. A única tarefa dele é atualizar o lado de leitura.
7. **Read Model (CQRS):** Projector pega o `ItemAdicionadoAoCarrinhoEvent` e atualiza uma tabela simples no banco de dados `UPDATE view_carrinhos SET total_itens = 5 WHERE id = ...` .
8. **Consulta (CQRS):** quando a UI precisa exibir o carrinho, ela faz uma consulta simples e rápida diretamente contra essa tabela view_carrinhos.

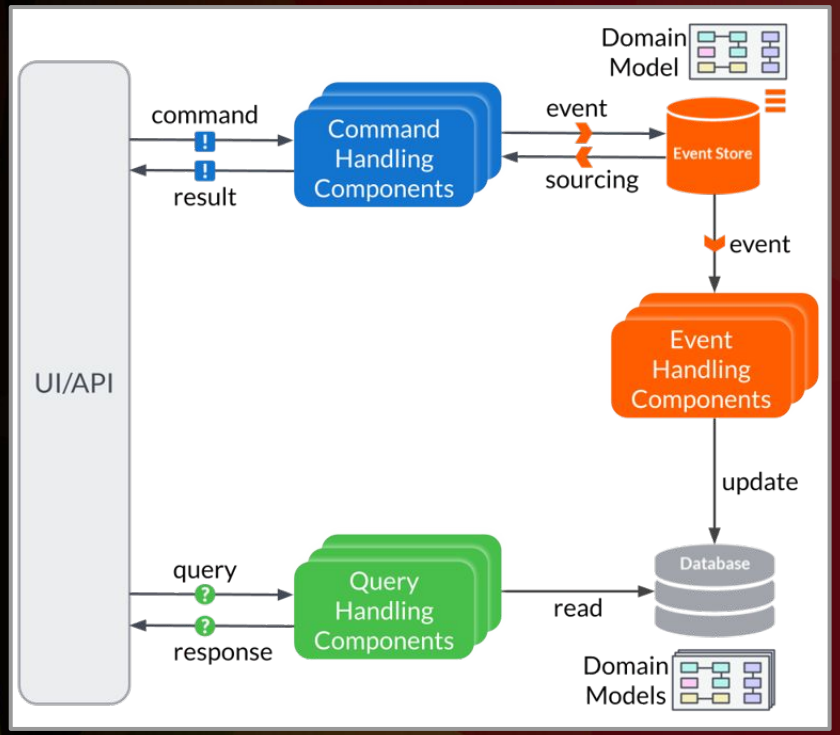
Axon Framework

A **Axon** fornece o **Axon Framework** e o **Axon Server** para ajudar a criar aplicativos centrados em três conceitos principais:

- CQRS - Command Query Responsibility Segregation.
- Event Sourcing.
- DDD - Domain Driven Design.

Embora muitos tipos de aplicativos possam ser criados usando o **Axon**, ele provou ser muito popular para arquiteturas de microsserviços.

O **Axon** fornece uma maneira inovadora e poderosa de evoluir aplicações orientados a eventos dentro de uma arquitetura de microsserviços.



Comandos: expressam a intenção de alterar o estado do aplicativo. Os comandos são roteados para um único destino e podem fornecer uma resposta.

Queries: expressam o desejo por informação. Dependendo da estratégia de despacho, Queries podem ser roteadas para um ou mais destinos simultaneamente.

Eventos: representam uma notificação de que algo relevante aconteceu. Eventos são publicados para qualquer componente interessado e não fornecem nenhuma forma de valor de retorno.