Aula 3 – Modelagem de Domínio e Identificação de Microserviços

Objetivos da Aula

- Compreender como transformar requisitos em um modelo conceitual claro.
- Aprender a modelar entidades, atributos e relacionamentos de um sistema corporativo.
- Aplicar o conceito de domínio e *bounded context* para dividir o sistema em microserviços.
- Elaborar o diagrama de domínio do sistema de eventos corporativos.
- Definir as responsabilidades de cada microserviço que comporá a solução.

---

1. Por que modelar o domínio antes de programar?

Ao finalizar o levantamento de requisitos, sabemos **o que** o sistema precisa fazer. O próximo passo é decidir **como** vamos organizar essas funcionalidades, evitando improviso na hora de codificar. Modelar o domínio é criar um mapa claro do sistema – quais entidades existem, como se relacionam, que atributos possuem e quais regras de negócio são essenciais.

> **Exemplo prático**: Imagine um sistema de eventos onde os requisitos apontam que usuários podem se inscrever em eventos, eventos podem ser pagos ou gratuitos, e após o evento é possível coletar feedback. Essas ações e dados são traduzidos em entidades e relações que precisam ser modeladas.

Vantagens de uma boa modelagem

- Evita retrabalho ao identificar problemas cedo.
- Ajuda a dividir as tarefas entre as equipes.
- Serve de guia para o design dos microserviços.
- Facilita o entendimento entre todos os envolvidos no projeto.

---

2. Conceitos-Chave: Entidades, Atributos e Relacionamentos

Entidade

Representa um objeto ou conceito importante para o sistema. Geralmente, vira uma tabela no banco de dados ou uma classe no código.

- **Exemplo**: Evento, Usuário, Inscrição, Pagamento, Feedback, Notificação.

Atributos

São as informações que caracterizam cada entidade.

- **Exemplo (Evento)**: id, nome, data, local, capacidade, valor, status (ativo/encerrado).

Relacionamentos

Indicam como as entidades se conectam. Podem ser:

- **Um para muitos**: Um evento tem várias inscrições.
- **Muitos para um**: Uma inscrição pertence a um único evento.
- **Muitos para muitos**: Um participante pode se inscrever em vários eventos (dependendo da modelagem).

Dica: Use notação UML simplificada ou um quadro com linhas conectando entidades para visualizar.

---

3. Construindo o Diagrama de Domínio

Passo a Passo

1. **Liste todas as entidades identificadas nos requisitos.**
2. **Defina os atributos principais de cada entidade.**
3. **Desenhe os relacionamentos entre as entidades.**
4. **Indique as cardinalidades (um para muitos, muitos para muitos, etc).**

Exemplo de Diagrama Inicial (texto)

- **Usuário**
- id, nome, email, senhaHash, tipo (participante/organizador)
- **Evento**
- id, nome, data, local, capacidade, valor, status
- **Inscrição**
- id, usuarioId, eventoId, status, dataInscricao
- **Pagamento**
- id, inscricaoId, valor, status, dataPagamento
- **Feedback**
- id, usuarioId, eventoId, nota, comentario
- **Notificação**
- id, usuarioId, eventoId, tipo, dataEnvio

Relacionamentos:

- Um Evento tem muitas Inscrições
- Uma Inscrição pertence a um Usuário e a um Evento
- Uma Inscrição pode gerar um Pagamento
- Um Usuário pode enviar Feedback para um Evento
- Notificações são disparadas para Usuários sobre Eventos

Você pode desenhar este diagrama no papel, usar o draw.io, Lucidchart, ou até um quadro físico em sala. O importante é que fique visível e compreensível!

---

4. Da Modelagem ao Microserviço: O que é *Bounded Context*?

Definição

Um *bounded context* (contexto delimitado) é um recorte do domínio onde um conjunto de regras, dados e operações faz sentido de maneira isolada. Cada microserviço deve focar em um contexto específico, garantindo baixo acoplamento e alta coesão.

- **Exemplo**: O serviço de Inscrições não deve ter lógica de pagamento interna – ele apenas registra a inscrição, enquanto o serviço de Pagamentos se responsabiliza por processar pagamentos e atualizar o status.

Heurísticas para Dividir Microserviços

- Cada microserviço deve ser responsável por um conjunto de funcionalidades de negócio coeso.
- Evite criar microserviços “por tabela”. O foco não é fragmentar demais!
- Considere separar quando regras de negócio mudam de forma independente (ex: cadastro de usuários vs. processamento de pagamentos).
- Avalie se o serviço pode ser implementado, testado e implantado separadamente.

Tabela de Microserviços e Responsabilidades

Microserviço	Entidades Principais	Responsabilidade
Usuários/Auth	Usuário	Cadastro, login, autenticação, gerenciamento de perfis
Eventos	Evento	Criação, edição, listagem e gerenciamento de eventos
Inscrições	Inscrição	Registro de participação dos usuários em eventos
Pagamentos	Pagamento	Processamento de pagamentos de inscrições
Notificações	Notificação	Envio de e-mails, confirmações e alertas
Feedback	Feedback	Coleta e análise de opiniões dos participantes

Observação: Para MVP, algumas funcionalidades podem ser agrupadas e depois separadas conforme o projeto evoluir.

---

5. Prática em Equipe: Construindo o Modelo Conceitual

Atividade Orientada

1. Em equipe, elabore o diagrama de domínio do sistema de eventos corporativos (pode ser no papel ou digital).
2. Preencha uma tabela relacionando cada entidade ao microserviço responsável.
3. Descreva, em poucas linhas, a função de cada microserviço do sistema.
4. Revise se cada requisito funcional identificado está coberto por pelo menos um microserviço.

Exemplo de perguntas a serem respondidas:

- Todo requisito da semana anterior está coberto?

- Se um requisito depende de mais de um serviço (ex: pagamento e inscrição), como os serviços vão se comunicar?
- Alguma entidade ficou “órfã”, sem serviço responsável?

---

6. Validação com o Método CERTO

Reforce a importância de aplicar o método CERTO na validação do modelo:

- **Contexto:** “Temos entidades Evento, Usuário, Inscrição, etc, para um sistema de eventos.”
- **Exigências:** “O modelo deve estar consistente e normalizado.”
- **Referências:** “Descrevemos relações, como um usuário pode ter várias inscrições.”
- **Tarefa:** “Verificar se essa modelagem faz sentido e sugerir ajustes.”
- **Observações:** “Explicar brevemente os motivos das sugestões.”

> **Dica de prompt ao ChatGPT:** “Contexto: Estamos modelando o domínio de um sistema de eventos corporativos com entidades Evento, Usuário, Inscrição, Pagamento, Feedback e Notificação. Exigências: o modelo deve ser consistente, normalizado e cobrir todos os requisitos levantados. Referências: relacionamentos já descritos acima. Tarefa: revise o modelo e sugira ajustes ou identifique potenciais problemas. Observações: resposta detalhada em português.”

---

7. Entrega e Discussão

Cada equipe deve entregar:

- Diagrama de domínio com entidades, atributos e relacionamentos principais.
- Tabela final de microserviços e suas responsabilidades.
- Pequena justificativa para as principais escolhas de divisão.

Na aula seguinte, todos poderão discutir as decisões, comparar soluções e ajustar o modelo antes de partir para o design arquitetural.

---

8. Dicas Finais

- Priorize clareza e lógica na modelagem – não tente detalhar tudo demais neste momento, foque no essencial.
- Documente todas as decisões, inclusive as dúvidas e justificativas.
- Não tenha medo de revisar o modelo se perceber incoerências ou oportunidades de melhoria – o objetivo é construir uma base sólida para o sistema!
- Lembre-se: modelar bem agora é ganhar tempo e qualidade depois, especialmente em projetos de microserviços!

> **Resumo:**

A modelagem de domínio e a identificação dos microserviços são etapas fundamentais para garantir que o sistema de eventos corporativos seja organizado, escalável e fácil de evoluir. Com um bom diagrama de domínio, a equipe tem clareza sobre o que cada parte do sistema fará, reduzindo retrabalho e aumentando as chances de sucesso do projeto.

---

9. Ferramenta de Apoio: Mermaid para Diagramas de Domínio

Uma forma simples e moderna de criar diagramas de domínio (ou diagramas de classes) é usando o Mermaid. O Mermaid permite criar diagramas a partir de um código textual fácil de ler, que pode ser renderizado em plataformas como o GitHub, VS Code (com extensões), HackMD, Obsidian, entre outras.

Exemplo de código Mermaid para o nosso domínio:

```
classDiagram
    class Usuario {
        +id
        +nome
        +email
        +senhaHash
        +tipo
    }
    class Evento {
        +id
        +nome
        +data
        +local
        +capacidade
        +valor
        +status
    }
    class Inscricao {
        +id
        +usuarioId
        +eventoId
        +status
        +dataInscricao
    }
    class Pagamento {
        +id
        +inscricaoId
        +valor
        +status
        +dataPagamento
    }
    class Feedback {
        +id
        +usuarioId
        +eventoId
        +nota
```

```
        +comentario
    }
    class Notificacao {
        +id
        +usuarioId
        +eventoId
        +tipo
        +dataEnvio
    }

    Evento "1" o-- "*" Inscricao : "possui"
    Usuario "1" o-- "*" Inscricao : "realiza"
    Inscricao "1" o-- "0..1" Pagamento : "gera"
    Usuario "1" o-- "*" Feedback : "envia"
    Evento "1" o-- "*" Feedback : "recebe"
    Usuario "1" o-- "*" Notificacao : "recebe"
    Evento "1" o-- "*" Notificacao : "dispara"
```

Como gerar o diagrama?

- Copie e cole o código acima em um editor online como o Mermaid Live Editor.
- Visualize o diagrama gerado automaticamente e faça ajustes se quiser.
- Você pode exportar o diagrama como imagem para usar em slides, PDFs, etc.

Exemplo da Imagem Gerada:

> **Dica:** Incentive as equipes a experimentar ferramentas como o Mermaid para documentar e compartilhar rapidamente diagramas em ambientes colaborativos e repositórios de código.