VLADMIR CRUZ

PDS TECH

SOFTWARE ARCHITECTURE

DOMAIN-DRIVE DESIGN

AULA 05

SUMÁRIO

O QUE VEM POR AÍ?

Até aqui falamos de conceitos, de “o que” e “por que” fazer um design de software. De agora em diante, falaremos em “como”, e isso se dará por meio de padrões que refletem o que vimos até agora, ou seja, vamos materializar tudo que vimos aqui, no sentido de arquitetar o sistema que queremos criar, montando estruturas e entendendo como estas interagem entre si e como isso vai nos guiar na criação da solução.

HANDS ON

Na aula 5, é a vez do design tático. Abordaremos os principais conceitos de como implementar DDD, falando de Arquitetura e dos Blocos de Construção do DDD, discutindo as bases do nosso sistema e definindo as tecnologias, ou seja, dando uma cara tech ao nosso sistema.

Além disso, vamos abordar os modelos de domínio, trabalhando com o ciclo de atividades que nosso cenário de entrega de atividade se encaixa, seus objetos de valor, entidades e agregados. Bora assistir a videoaula?

SAIBA MAIS

O Design Tático

Em nosso estudo de DDD, começamos falando do Design Estratégico, do “por quê” e “o que” fazer. Agora falaremos do Design Tático, momento em que analisaremos o “como”. Abordaremos os principais conceitos de como implementar DDD, falando de Arquitetura e dos Blocos de Construção do DDD. É neste momento que vamos discutir as bases do nosso sistema, definindo quais tecnologias utilizar (como por exemplo: bancos relacionais ou NoSQL? Microsserviços ou Barramento de Serviços?), e como elas interagem entre si.

Arquitetura

Chegou a hora de montar nossa solução! Já mapeamos nosso domínio e subdomínios, definimos nossa linguagem ubíqua e nossos contextos delimitados, definindo como estes se comportam e comunicam.

Precisamos agora, criar o desenho de nosso sistema, como cada subdomínio será implementado, onde ficarão as lógicas e suas devidas partes.

O DDD organiza sua arquitetura em quatro (4) partes:

Camada de Interface de Usuário

Essa é a camada que contém a Interface do usuário (GUI), interfaces de linha de comando (CLI) e APIs para integração com outros sistemas.

Camada de Aplicação

Nesta camada temos interfaces que fazem a mediação entre a camada de interface do usuário e a camada de domínio. Por exemplo, essa camada não contém lógica de negócios, não altera o estado de objetos, mas monitora e reporta essas mudanças a camada superior. Também é a camada que organiza as tarefas que o

sistema tem que realizar. Importante ressaltar que, em algumas arquiteturas, essa camada não existe, ela é integrada à camada de interface de usuário.

Nesta camada podemos definir as rotinas que são os “gatilhos” para atualizações em massa do sistema. Na nossa escola, exemplificando, podemos ter uma camada que faz a rotina de contabilização das faltas dos alunos por dia, e atualiza os registros centrais da escola; é uma rotina que pode ser executada em tempo real, ou em lote, ela não faz nada além de disparar uma lógica de negócio (atualizar presença), pegar o resultado, e enviar para a camada de interface do usuário.

Camada de Domínio

“O coração do Software”, como denominado por Eric Evans (2003), é a camada que contém os conceitos de negócios, onde estão todas as regras de negócio. É nessa camada que a lógica de negócio é executada, as mudanças de estado acontecem e os registros e afins são criados. Essa camada não armazena os dados, mas provê as devidas informações para serem registradas à Camada de Infraestrutura.

Nesta camada temos as funções que contém “o segredo” do nosso sistema, aquilo que é essencial para seu funcionamento e diferenciação. É nessa camada por exemplo, que teríamos toda a lógica de notas dos alunos, considerando que os mesmos entregaram na data ou não, e assim calcular o desconto na nota, ou até mesmo a lógica por trás dos planos de aula e como são aplicados e avaliados.

Camada de Infraestrutura

Essa é a camada que possui as capacidades técnicas de suporte às camadas superiores. Aqui existem os meios de mensageria, persistência de dados e é utilizada como o padrão de interações entre as camadas (caso não haja integração direta entre elas).

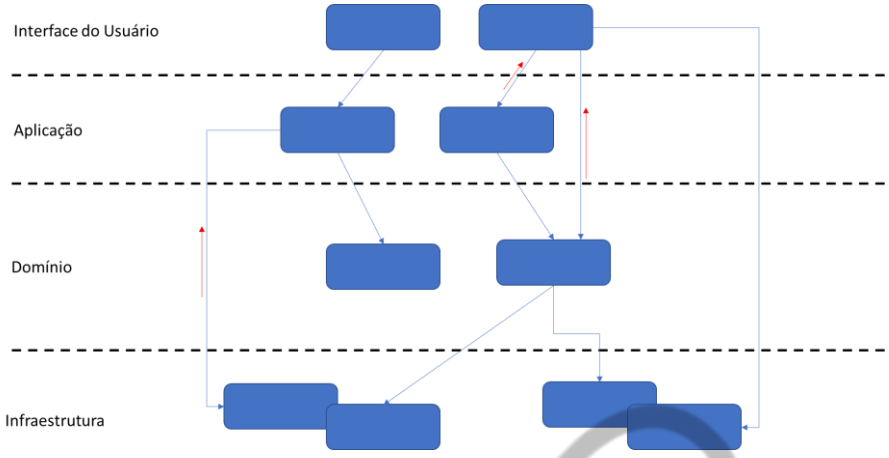

Figura 1 – Esquema de Camadas do DDD
Fonte: Adaptado por FIAP (2023), de Evans (2003), p. 62

Olhando para nosso Contexto Delimitado, podemos montar uma arquitetura da forma demonstrada na figura 2 – “Arquitetura de solução mostrando as camadas DDD”:

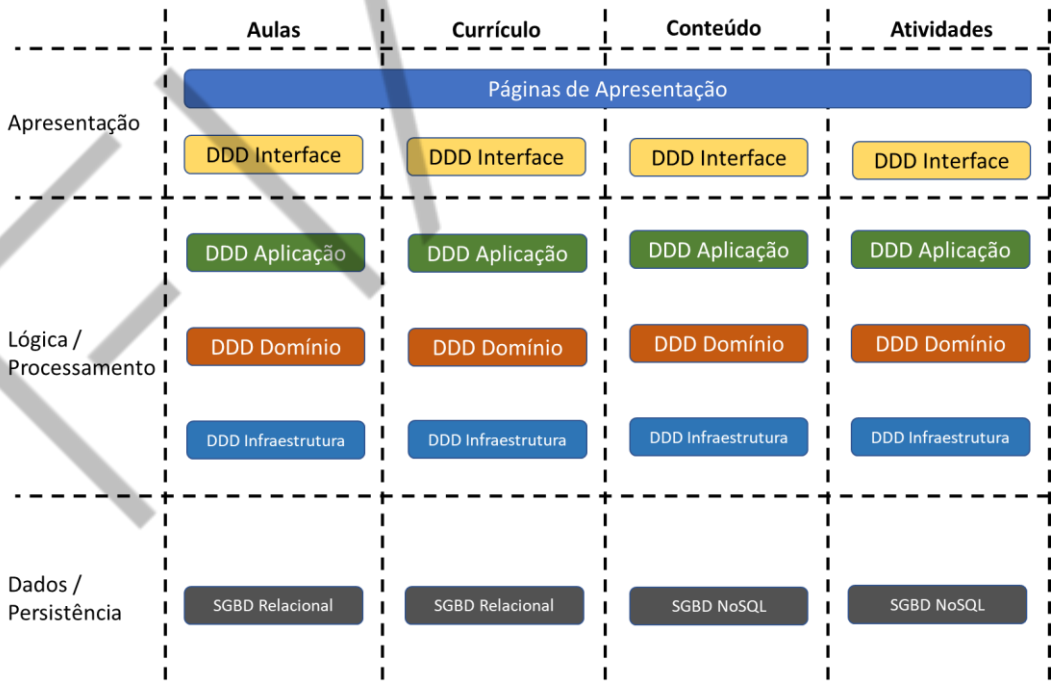

Figura 2 – Arquitetura de Solução mostrando as camadas DDD
Fonte: Adaptado por FIAP (2023), de Evans (2003), p. 62

Agora que já temos a figura da nossa arquitetura, podemos começar a construir, e é aqui que o DDD se diferencia de outros modelos.

Objetos de valor (Value objects)

Objetos de valor são reconhecidos por não possuírem identificadores, sendo assim, usamos seus valores para os distinguir um do outro. Cada um é único e imutável (o valor é criado como um todo e não muda depois de sua criação).

Vaughn Vernon (2013), tem uma descrição muito legal de objetos de valor:

- Eles mensuram, quantificam, ou descrevem algo no domínio.
- Eles podem ser mantidos como imutáveis.
- Criam um modelo conceitual de integridade se compondo de todos os atributos como uma unidade.
- São completamente substituíveis quando uma medida ou descrição muda.
- Podem ser comparados uns aos outros por igualdade de valores.

Por ser imutável, um objeto de valor não pode ser alterado depois de criado, tal como citamos anteriormente. E então, o que se passa é que, se algum atributo muda, criamos um valor completamente novo. Usando um exemplo bem simples, se temos x = 3, e depois mudamos para 4, criamos um x com valor 4. O mesmo acontece com objetos de valor: não alteramos o objeto já criado, substituímos por um totalmente novo.

Se um novo objeto for inserido nessa tabela, antes temos que verificar se seus valores existem. Por exemplo, se tentarmos inserir “Nome 3”, “03/01/200”, “Rua 3”, este não será inserido, pois já existe. Mas se alterarmos um item “Nome 3”, “03/01/200”, “Rua 5”, então um novo item entra em nossa lista.

Assim mantemos a unicidade da lista de objetos de valor, o que nos obriga a fazer uma checagem do objeto antes de criá-lo.

Entidades (Entities)

Entidades, ao contrário do que encontramos com Objetos de Valor, possuem identificadores, e são mutáveis, ou seja, cada um tem um ID único que nunca mais

será utilizado por outros, dessa forma, a imutabilidade é impossível, pois a Entidade pode e deve ser alterada depois de criada.

Se um novo Objeto for inserido nessa tabela, diferente dos objetos de valor, não temos que verificar se seus valores existem, por exemplo, se tentarmos inserir (“Nome 3”, “03/01/200”, “Rua 3”), este será inserido, independente de existir, pois teremos um índice novo.

Em caso de alteramos um valor, utilizamos o seu índice para encontrá-lo na lista, e então podemos alterar o que for necessário.

Agregados (Aggregate)

Um agregado é um conjunto de entidades e objetos de valor, que mantém relacionamentos entre si e possui um limite que o circunda e define.

Uma das premissas básicas de agregados é a consistência forçada, que garante a integridade de dados. Ou seja, somente a lógica do agregado pode alterar o seu estado, garantindo assim que a lógica de negócio que o define seja garantida sempre.

Nenhum objeto fora do agregado pode alterar o seu estado. Esse pode ler seu estado, mas não pode alterar, isso é feito somente pelo próprio agregado, porém, entidades externas podem solicitar que o agregado execute ações que alterem seu estado.

Atenção: as entidades externas não podem alterar, mas podem solicitar que seja modificado. Isto é exposto no agregado por meio de interfaces externas, que habilitam o que chamamos de comandos. Sendo assim, um objeto externo “comanda” que algo ocorra.

Serviços de domínio (Domain services)

Serviços de domínio são objetos tratados separadamente, e que trabalham com diversas entidades e agregados, sempre que são necessários cálculos, execuções de rotinas e muito mais.

O QUE VOCÊ VIU NESTA AULA?

Vimos o que é o design tático, ou seja, como faremos o nosso desenvolvimento, e começamos a montar nossa arquitetura, desenhando cada uma das camadas do DDD, a de Interface do Usuário, a de Aplicação, a de Domínio e a de Infraestrutura. Além disso, falamos de alguns elementos chave do DDD, os objetos de valor, as entidades, os agregados e os serviços de domínio, e como se organizam em nossa arquitetura.

REFERÊNCIAS

COCKBURN, A. **Writing Effective Use Cases**. [s.l.]. Addison Wesley, 2001.

EVANS, E. **Domain-Driven Design, Tackling Complexity in the heart of Software**. Boston: Pearson Education, 2003.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [s.l.]. Addison-Wesley, 2002.

KHONONOV, V. **Learning Domain-Driven Design**. Sabastopol: O'Reilly Media, 2021.

MILLET, S. TUNE, N. **Patterns**, **Principles and Practicves of Domain-Driven Design**. Indianapolis: Wrox, 2015.

VERNON, V. **Implementing Domain-Driven Design**. Boston: Pearson Education, 2013.

PALAVRAS-CHAVES

Design tático. Modelo de domínio. DDD.