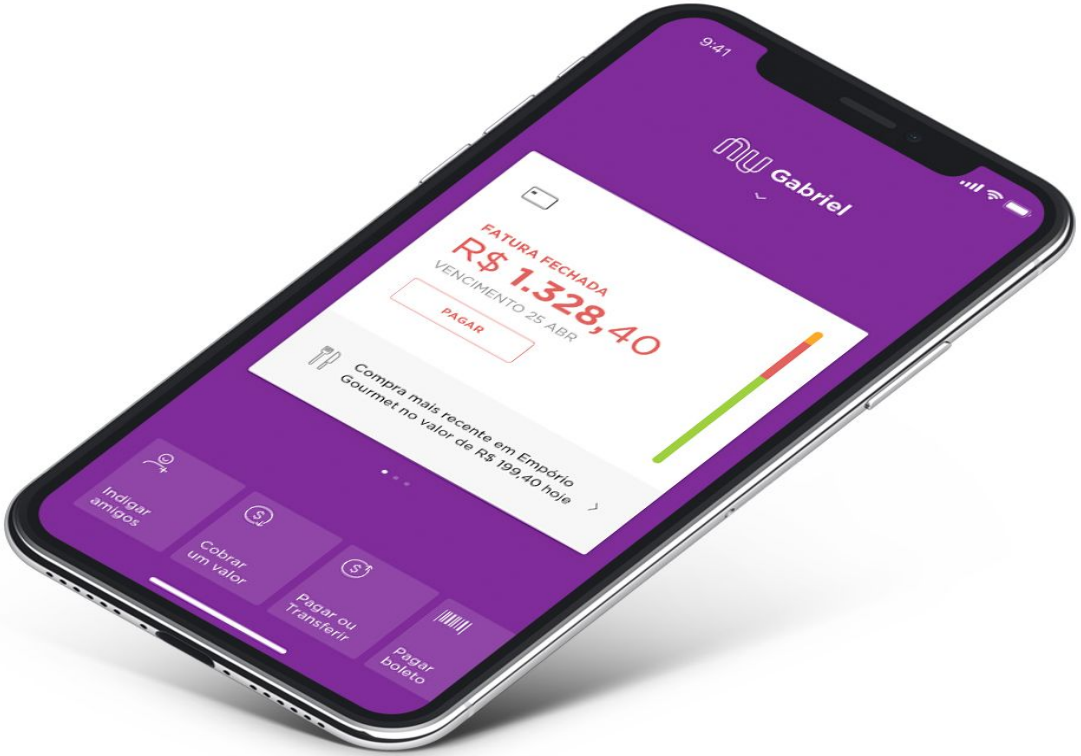
Application Services
x
Domain Services

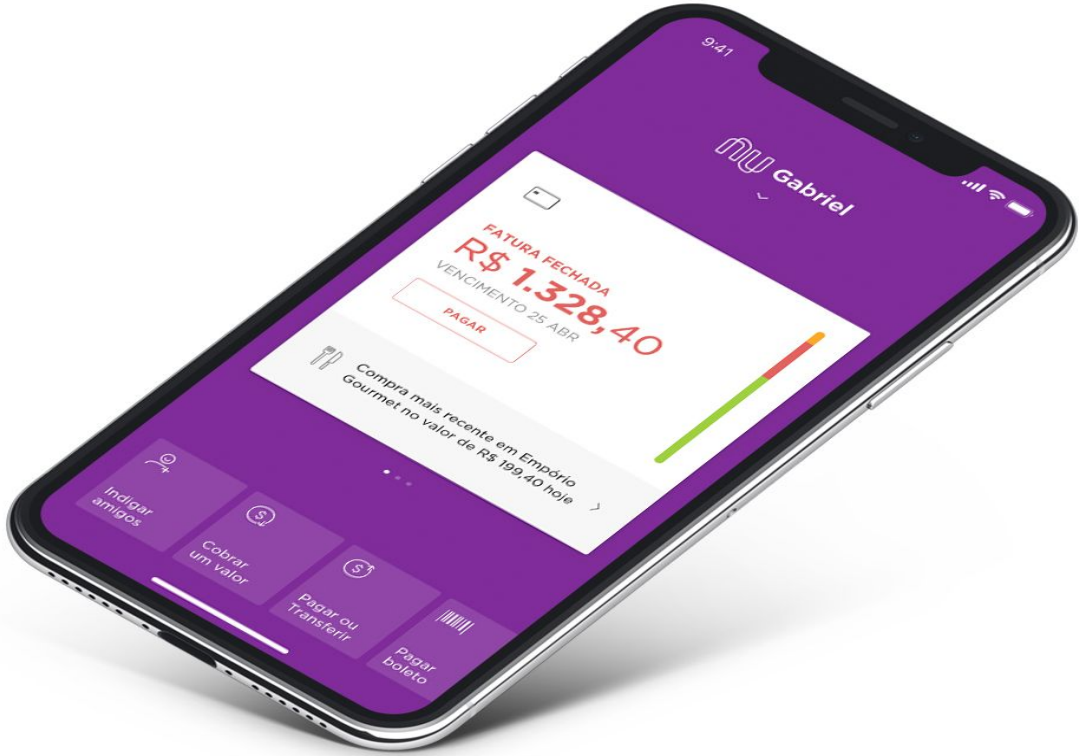
Porque

Código orientado a objeto sofre quando não bem encapsulado e quando não mostra suas intenções



Porque

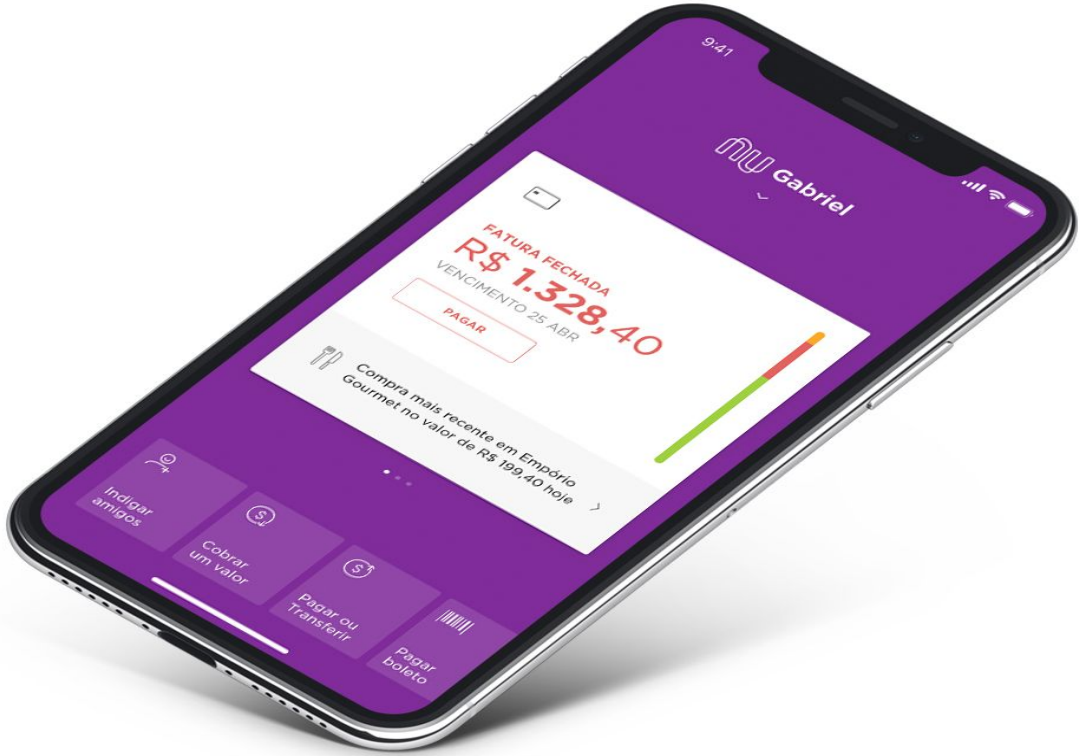
Praticantes de DDD ou não, sempre usaram Application Services mas a relação com Domain Service nem sempre é óbvia



Para quem

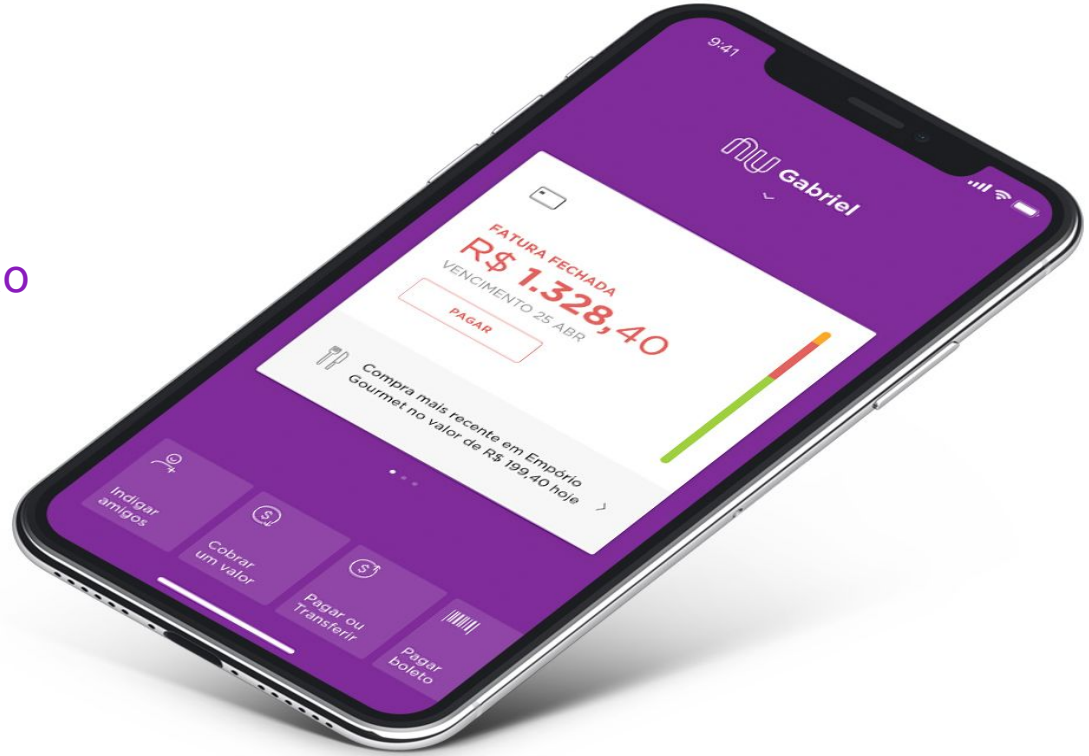
Iniciantes em DDD

Engenheiros que sofrem diariamente com falta de delimitações e encapsulamento



Onde queremos chegar

- Código revelando intenção
- Encapsulamento
- Delimitações



Application Services

Application Service

Orquestra um caso de uso do inicio ao fim
em uma única operação

OrderService
.AddItem()
Adicionar mercadoria
a encomenda

VehicleService.
Rent()
Alugar veiculo

AccountService.
Transfer()
Transferir quantia

DiscountService.
Apply()
Aplicar desconto

ProposalService
.Accept()
Aceitar proposta

LendingService.
Lend()
Emprestar quantia

Operações comuns

O que um um Application Service faz numa orquestração de caso de uso

- **Calcular juros**
- **Chamada HTTP para um serviço de terceiro**
- **Salvar uma entidade em um repositório**
- **Escrever um log**
- **Verificar autorização**
- **Buscar uma entidade de um repositório**
- **Chamar o método de uma entidade**
- **Enviar um email**

Controller grande

->

Extract method

Application Services

The DDD way...

Essa camada deve ser fina. Ela não contêm regras de negócio ou sabedoria, mas apenas coordena tarefas e delega o trabalho para os objetos de domínio.


Software de
Vagas de
Emprego
mastercard
GABRIELA LIMA

Caso de Uso

Aceitar proposta.

Como candidato, quero aceitar uma proposta de emprego de uma empresa, para que eu possa começar a trabalhar nela.

Aceitar proposta

Critério 1

Proposta não deve ter sido enviada há mais de 15 dias

Critério 2

Aceite deve ir para o log de atividades

Critério 3

Outros sistemas devem ser notificados

```
public class ProposalService
{
    public void AcceptProposal(Guid proposalId)
    {
        //retrieve proposal aggregate from repository
        var proposal = _repository.Find(proposalId);

        //if proposal is expired, fail fast
        if(DateTime.Now.Substract(proprosal.SentAt).TotalDays > 15)
                throw new DomainException("This proprosal has expired");

        //else, change its state to accepted
        proposal.Status = ProposalStatus.Accepted;
        proposal.AcceptedAt = DateTime.Now;
        proposal.EventsRaised.Add(new ProposalAcceptedEvent(proposal.Id, proposal.AcceptedAt))

        //save in through repository
        _repository.Save(proposal);
        //publish domain events
        _eventBus.Publish(proposal.EventsRaised);
        //logs
        _logger.Info($"Proposal {proposal.id} was accepted");
    }
}
```

Encapsulamento?

```
public class Proposal
{
    public DateTime SentAt { get; set; }
    public DateTime AcceptedAt { get; set; }
    public ProposalStatus Status { get; set; }
    public Guid CandidateId { get; set; }
}
```

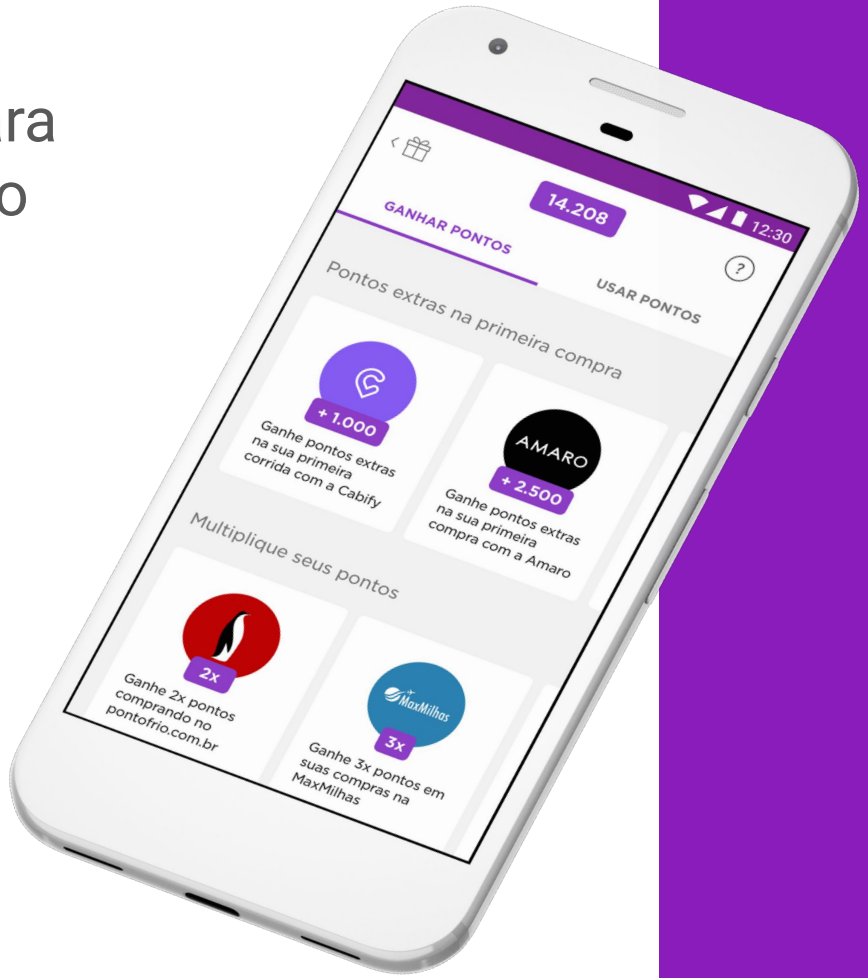
DOMÍNIO ANÊMICO

Dados em entidade e comportamentos em serviços é um anti padrão chamado de domínio anêmico

INFORMATION HIDING

Um objeto deveria tanto guardar dados (propriedades) quanto processá-los (métodos)

Refatorando para um domínio rico



```
public class Proposal : Entity
{
    public Guid CandidateId { get; }
    public Guid JobListingId { get; }
    public ProposaStatus Status { get; private set; }
    public DateTime? ClosedAt { get; private set; }

    public Proposal(Guid candidateId, Guid jobListingId)
    {
        CandidateId = candidateId;
        JobListingId = jobListingId;
        Status = ProprosalStatus.Open;
    }
```

```
public void Accept()
{
    ChangeStatus(ProposalStatus.Accepted);
    base.EventsRaised.Add(new ProposalAcceptedEvent(proposal.Id, proposal.AcceptedAt))
}

public void Deny()
{
    ChangeStatus(ProposalStatus.Denied);
    base.EventsRaised.Add(new ProposalDeniedEvent(proposal.Id, proposal.AcceptedAt))
}

private void ChangeStatus(ProposalStatus newStatus)
{
    if(DateTime.Now.Substract(proprosal.SentAt).TotalDays > 15)
        throw new DomainException("This proprosal has expired");

    if(Status != ProposalStatus.Pending)
        throw new DomainException("Cannot accept or deny this proposal");

    Status = newStatus;
    ClosedAt = DateTime.Now;
}
```

Código demonstrando intenção

```
public class ProposalService
{
    public void AcceptProposal(Guid proposalId)
    {
        var proposal = _repository.Find(proposalId);
        //calling Accept method instead of changing its properties
        proposal.Accept();
        _repository.Save(proposal);
        _logger.Info($"Proposal {proposal.id} was accepted");
    }
}
```

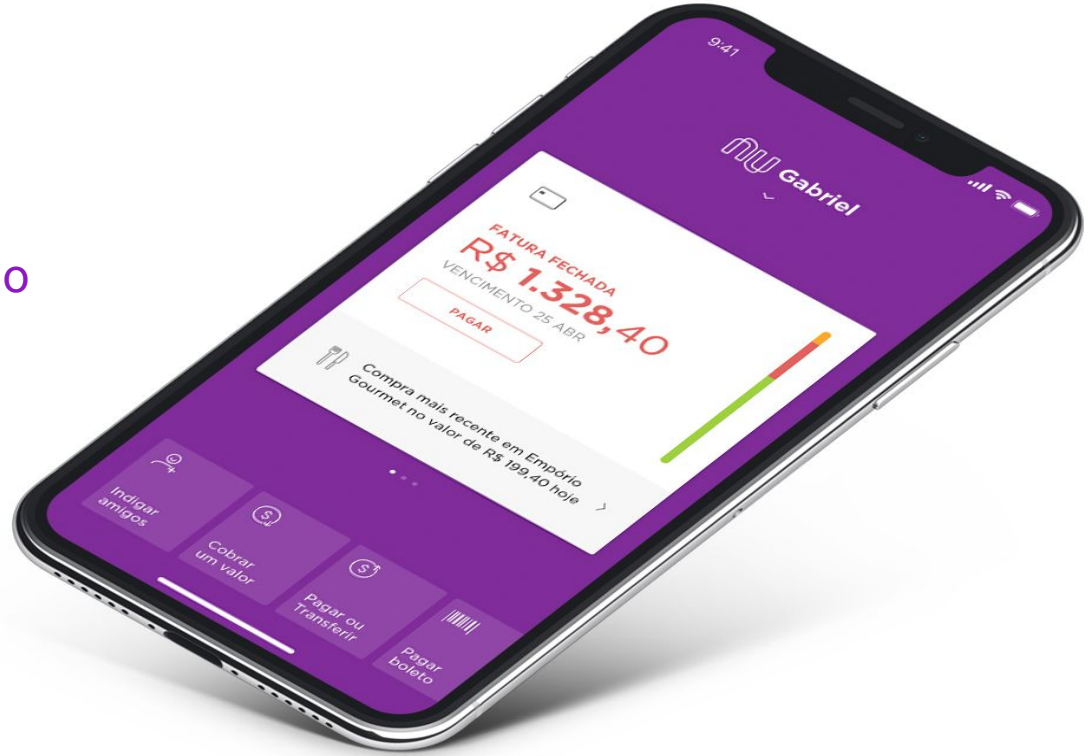
APPLICATION SERVICES

Job Done!

Essa camada deve ser fina. Ela não contêm regras de negócio ou sabedoria, mas apenas coordena tarefas e delega o trabalho para os objetos de domínio.

Onde queremos chegar

- Código revelando intenção
- Encapsulamento
- Delimitações



Easy Gains

Ao extrair o application service do seu controller

“

Evita vendor lock-in

“

Casos de uso acessíveis por outras portas

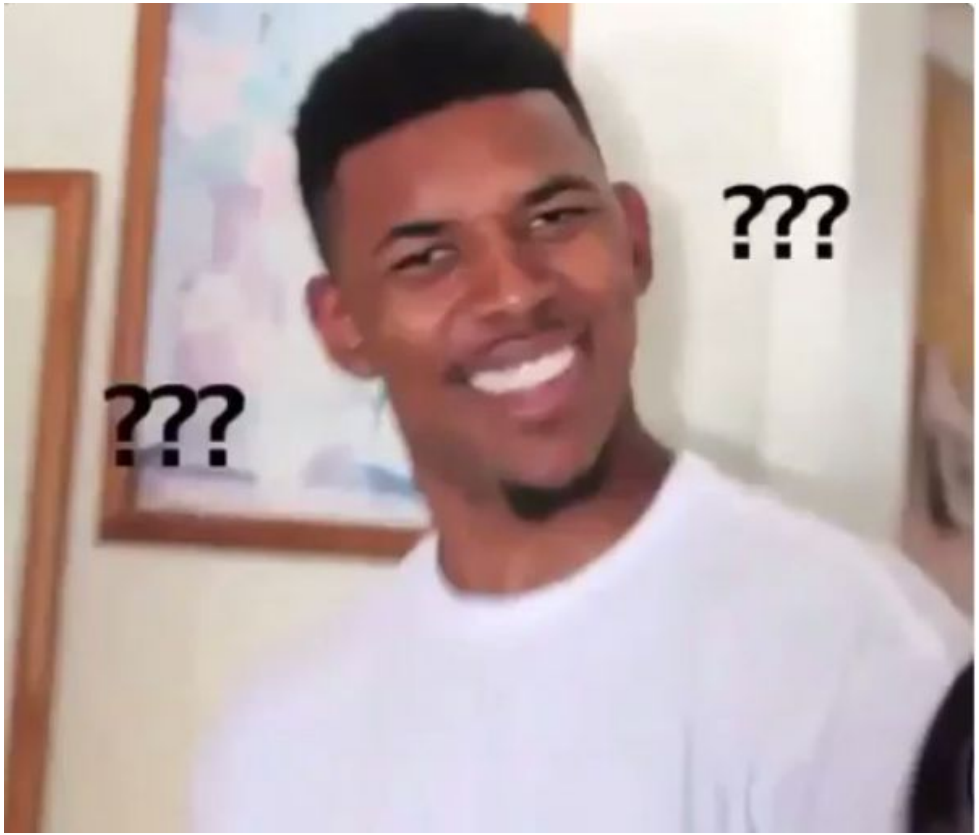
“

Testabilidade

Domain Services

Domain Service

Utilizamos Domain Services para processar lógica de domínio que não diz respeito a um único **agregado** ou uma entidade de domínio



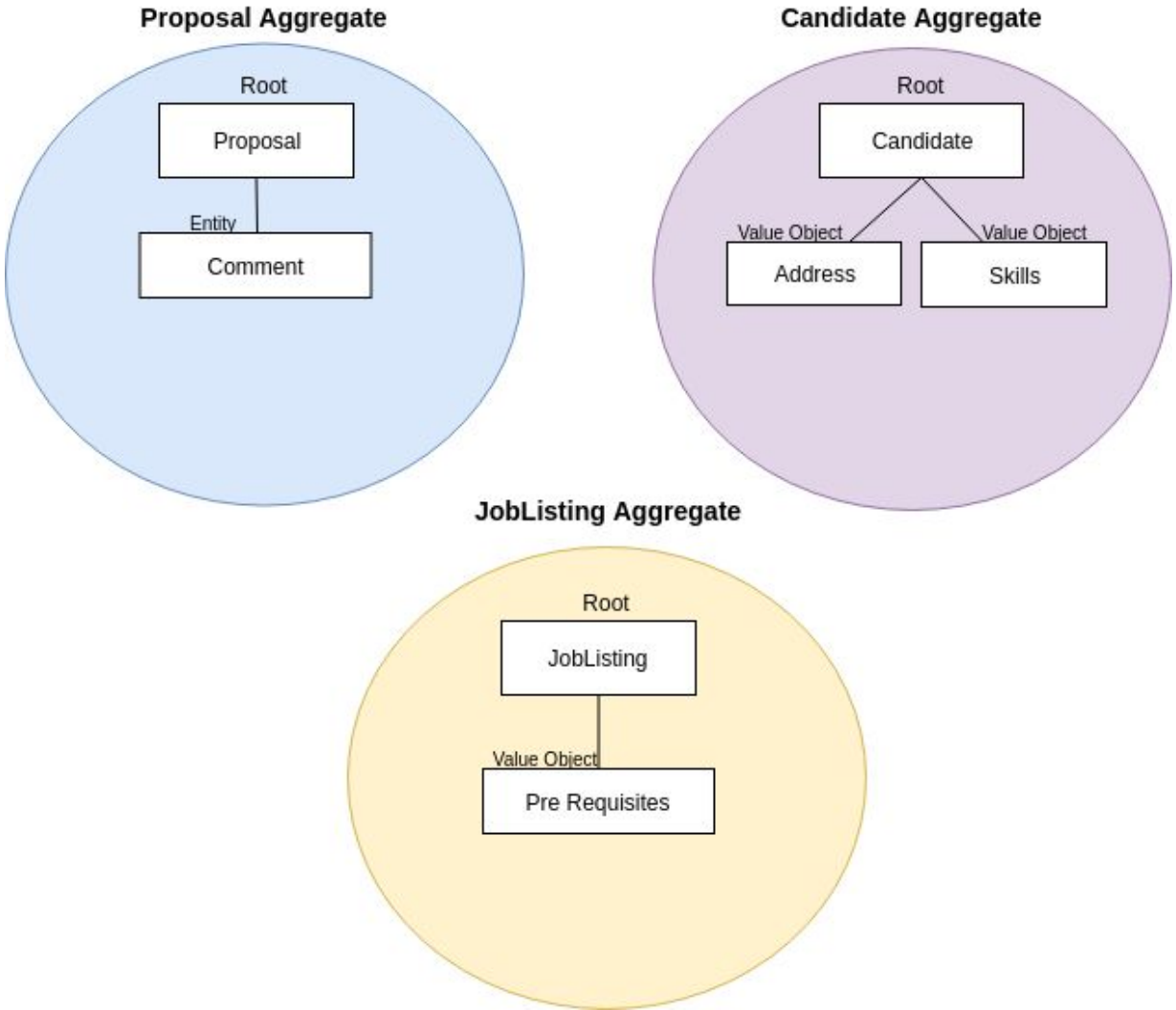
Agregado

Rationale

Objetos devem manter seu estado interno consistente, mas seu estado pode ficar inconsistente através de mudanças em outros objetos que são conceitualmente partes constituintes.

Agregado

Um conjunto de entidades agrupados por uma entidade raíz. A raíz do agregado é responsável por manter a árvore de entidades em um estado válido, protegendo suas regras de negócio.


Proposal Aggregate
Root
Proposal
Entity
Comment
Candidate Aggregate
Root
Candidate
Value Object
Address
Value Object
Skills
JobListing Aggregate
Root
JobListing
Value Object
Pre Requisites

Feature

Discutir proposta

Como candidato, quero discutir a proposta com o recrutador, para esclarecer detalhes da mesma e tomar uma decisão melhor.

Discutir proposta

Critério 1

Uma proposta está aberta discussão se ela não foi aceita ou recusada ainda

Critério 2

Um comentário não pode estar vazio

Critério 3

Um comentário não pode trocar de dono

```
public class Comment
{
    public Guid ProposalId { get; }
    public Guid UserId { get; }
    public string Content { get; private set; }

    public Comment(Guid proposalId, Guid userId, string content)
    {
        if(String.IsNullOrWhiteSpace(content))
            throw new DomainException("Empty comments not allowed");

        ProposalId = proposalId;
        UserId = userId;
        Content = content;
    }

    public void Edit(string newContent)
    => Content = newContent;
}
```

Critério 1

Uma proposta está aberta discussão se ela não foi aceita ou recusada ainda

```
public class Proposal
{
    private List<Comments> _comments;
    public IReadOnlyList<Comments> Comments => _comments.AsReadOnly();
    private bool IsOpenForDiscussion => Status != ProposalStatus.Pending;

    public void Discuss(Guid userId, string content)
    {
        if(!IsOpenForDiscussion)
            throw new DomainException("Can't discuss about an closed proposal");

        _comments.Add(new Comment(userId, Id, content));
    }
}
```

Comentário inconsistente

```
var inconsistentComment = new Comment(closedProposalId, anyCandidateIdIWant, content);
_commentsRepository.Save(comment);
```

Repositório

Agregados são as únicas entidades que podem ser buscadas e persistidas através de um repositório

```
var proposal = _proprosalRepository.Find(proposalId);
proposal.Discuss(candidateId, "I would like to know more about benefits");
_proposalRepository.Save(proposal);
```

Agregado de Proposta

```
public class Proposal
{
    private List<Comments> _comments;
    public IReadOnlyList<Comments> Comments => _comments.AsReadOnly();
    private bool IsOpenForComments => Status == ProposalStatus.Pending;
    public Guid CandidateId { get; }
    public Guid JobListingId { get; }
    public ProposaStatus Status { get; private set; }
    public DateTime ClosedAt { get; private set; }

    public Proposal(Guid candidateId, Guid jobListingId)
    {
        CandidateId = candidateId;
        JobListingId = jobListingId;
        Status = ProposalStatus.Pending;
    }
```

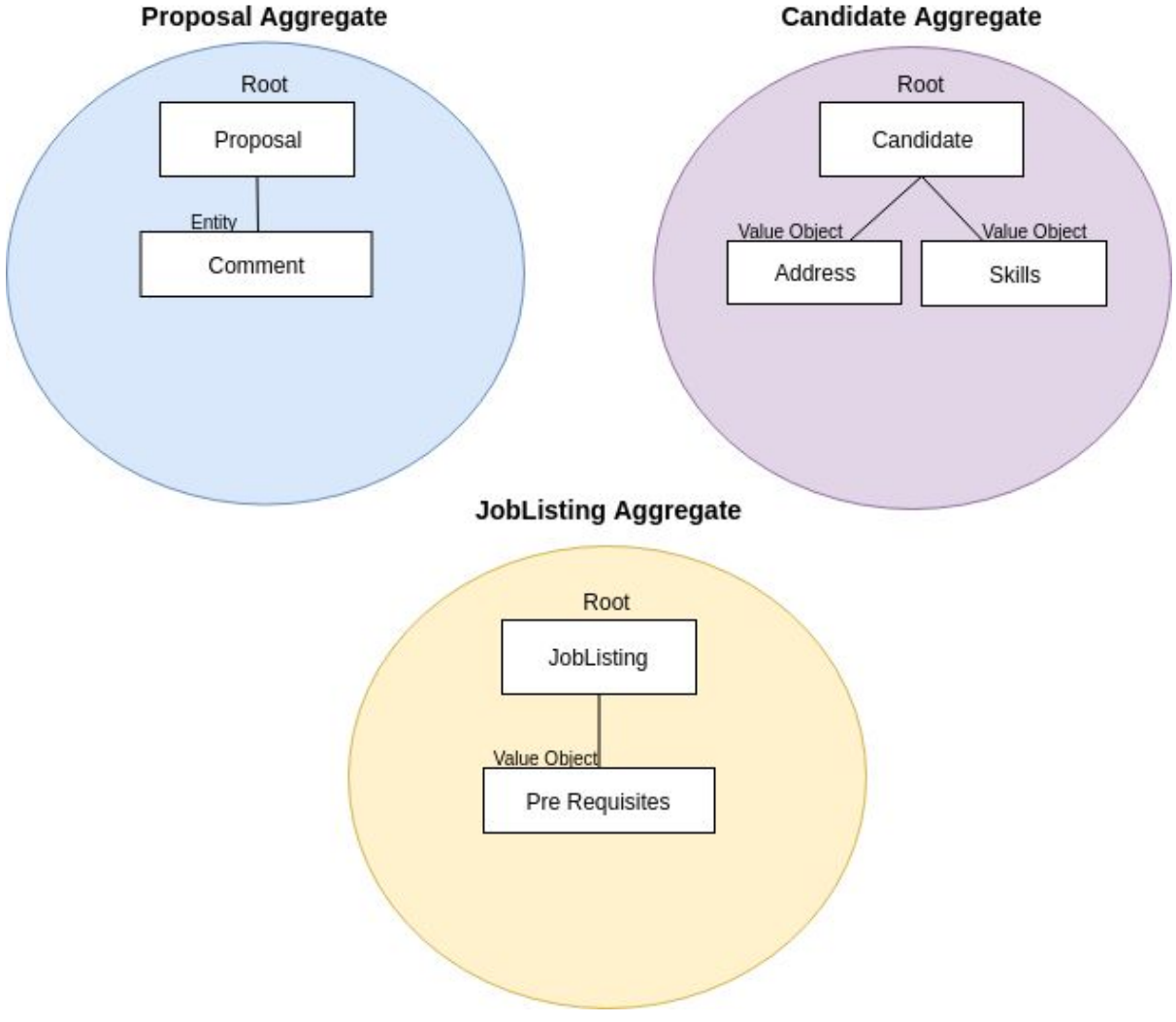
```
public void Accept()
{
    ChangeStatus(ProposalStatus.Accepted);
    proposal.EventsRaised.Add(new ProposalAcceptedEvent(proposal.Id, proposal.AcceptedAt))
}

public void Deny()
{
    ChangeStatus(ProposalStatus.Denied);
    proposal.EventsRaised.Add(new ProposalDeniedEvent(proposal.Id, proposal.AcceptedAt))
}

private void ChangeStatus(ProposalStatus newStatus)
{
    if(DateTime.Now.Substract(proprosal.SentAt).TotalDays > 15)
        throw new DomainException("This proprosal has expired");

    if(Status != ProposalStatus.Pending)
        throw new DomainException("Cannot accept or deny this proposal");

    Status = newStatus;
    ClosedAt = DateTime.Now;
}
```

```
public void Discuss(Guid userId, string content)
{
    if(!IsOpenForComments)
        throw new DomainException("Can't discuss about an closed proposal");

    _comments.Add(new Comment(userId, Id, content));
}
```

Relação com outros agregados

```
public class Proposal : Entity
{
    public Guid CandidateId { get; }
    public Guid JobListingId { get; }
}
```

```
public class Proposal : Entity
{
    public Candidate CandidateId { get; }
    public JobListing JobListingId { get; }
}
```

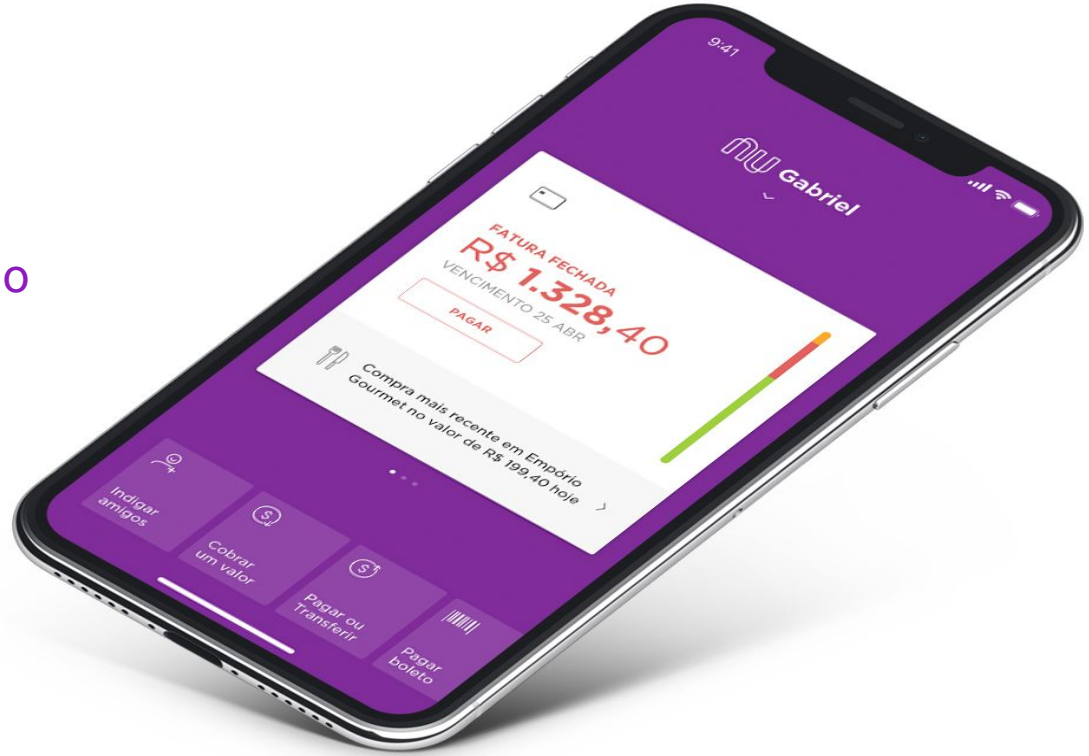

Proposal Aggregate
Root
Proposal
Entity
Comment
Candidate Aggregate
Root
Candidate
Value Object
Address
Value Object
Skills
JobListing Aggregate
Root
JobListing
Value Object
Pre Requisites

```
var proposal = _proposalRepository.Find(id);

proposal.Candidate.Ban();

proposal.JobListing.AddPreRequisite(PreRequisite.Clojure);

proposalRepository.Save(proposal);
```

Relacionar agregados por referencia leva a falta de delimitações

Onde queremos chegar

- Código revelando intenção
- Encapsulamento
- Delimitações



Bug fix com Domain Service

Bug

Ao aceitar a proposta, propostas de outras empresas para este candidato devem ser recusadas.

Sem relação com outras propostas

```
public class Proposal
{
    private List<Comments> _comments;
    public IReadOnlyList<Comments> Comments => _comments.AsReadOnly();
    private bool IsOpenForComments => Status == ProposalStatus.Pending;
    public Guid CandidateId { get; }
    public Guid JobListingId { get; }
    public ProposaStatus Status { get; private set; }
    public DateTime ClosedAt { get; private set; }
```

```
public class ProposalService
{
    public void AcceptProposal(Guid proposalId)
    {
        var proposal = _repository.Find(proposalId);
        //calling Accept method instead of changing its properties
        proposal.Accept();
        _repository.Save(proposal);

        var proposalsToDeny = _repository.FindOpenForCandidate(proposal.CandidateId);
        foreach(var proposal in proposalsToDeny)
          proposal.Deny();

        _repository.Save(proposalsToDeny);
        _logger.Info($"Proposal {proposal.id} was accepted");
    }
}
```

[...] apenas coordena tarefas e delega o trabalho para os objetos de domínio.

Domain Service

Alguns conceitos do domínio não se encaixam naturalmente no seu modelo. Movê-los para o modelo ou o distorce ou tira seu significado. Quando um processo ou transformação no domínio não é uma responsabilidade natural de uma entidade ou objeto de valor, adicione a operação como uma interface declarada como um serviço.

```
public class DomainService
{
    public class DenyOtherCompaniesProposals(Proposal proposal)
    {
        var proposalsToDeny = _repository.FindOpenForCandidate(proposal.CandidateId);
        foreach(var proposal in proposalsToDeny)
          proposal.Deny();

        _repository.Save(proposalsToDeny);
    }
}
```

Camada fina

```
public class ProposalService
{
    public void AcceptProposal(Guid proposalId)
    {
        var proposal = _repository.Find(proposalId);

        proposal.Accept();
        _repository.Save(proposal);

        _domainService.DenyOtherCompaniesProposals(proposal);

        _logger.Info($"Proposal {proposal.id} was accepted");
    }
}
```

Double dispatch (ish)

```
public class Proposal : Aggregate
{
    public void Accept(IDomainService domainService)
    {
        ChangeStatus(ProposalStatus.Accepted);
        base.EventsRaised.Add(new ProposalAcceptedEvent(proposal.Id, proposal.AcceptedAt))
        domainService.DenyOtherCompaniesProposals(this);
    }
}
```

```
public class ProposalService
{
    public void AcceptProposal(Guid proposalId)
    {
        var proposal = _repository.Find(proposalId);

        proposal.Accept(domainService);

        _repository.Save(proposal);

        _logger.Info($"Proposal {proposal.id} was accepted");
    }
}
```

```
public void AcceptProposal(Guid proposalId)
{
    var proposal = _repository.Find(proposalId);
    //calling Accept method instead of changing its properties

    if(DateTime.Now.Substract(proprosal.SentAt).TotalDays > 15)
        throw new DomainException("This proprosal has expired");

    if(Status != ProposalStatus.Pending)
        throw new DomainException("Cannot accept or deny this proposal");

    proposal.Status = newStatus;
    proposal.ClosedAt = DateTime.Now;

    _repository.Save(proposal);

    var proposalsToDeny = _repository.FindOpenForCandidate(proposal.CandidateId);
    foreach(var p in proposalsToDeny)
    {
        p.Status = ProposalStatus.Denied;
        p.ClosedAt = DateTime.Now;
        _repository.Save(p);
    }

    _logger.Info($"Proposal {proposal.id} was accepted");
}
```

Marco Nicolodi

Interested in joining our revolution?

nubank.com.br/en/careers/