

Logo da Orange Talents

# Demarcação de transação

## Antes de começar

Para complementar o curso, recomendamos estudar a documentação oficial do Micronaut sobre [Transactions](https://micronaut-projects.github.io/micronaut-data/latest/guide/#transactions) caso você não tenho estudado ainda. Nela é possível ter uma idéia mais clara do poder e das possibilidades oferecidas pelo framework.

Aqui é um momento importante de conectar vários conteúdos que estudamos e treinamos durante nossa jornada.

O que estou querendo dizer é que alguns conhecimentos sobre persistência de dados, controle transacional com Spring e bancos de dados relacionais que adquirimos através de alguns cursos da ALURA podem ser bastante úteis para este formulário, por exemplo:

* curso de [Spring Data JPA](https://www.alura.com.br/curso-online-spring-data-jpa), mais especificamente o tópico **Operações CRUD**;
* curso de [Spring Boot API REST: Construa uma API](https://www.alura.com.br/curso-online-spring-boot-api-rest), mais especificamente o tópico **Usando Spring Data**;
* curso de [Formação SQL com MySQL Server da Oracle](https://www.alura.com.br/formacao-oracle-mysql), mais especificamente o módulo **Avançando em manipulação de dados**;
* curso de [Persistência com JPA](https://www.alura.com.br/curso-online-persistencia-jpa-introducao-hibernate), mais especificamente o tópico **Ciclo de vida de uma entidade**;

Além disso, por o Micronaut ser inspirado no Spring Boot, seu controle transacional via @Transactional funciona da mesma maneira, portanto a documentação oficial do Spring sobre [Propagação de Transações](https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#tx-propagation) possui conteúdo útil que pode ser facilmente transferido para o ecossistema do Micronaut.

## Cenário:

Temos um microsserviço responsável por adicionar comentários aos artigos publicados por nossos usuários. Para isso, o time de backend implementou a seguinte API REST com Micronaut, JPA/Hibernate e um banco de dados relacional:

@Validated  
@Controller  
class ArtigoController(@Inject val service: ArtigoService) {  
  
 @Transactional  
 @Post("/api/artigos/{id}/comentarios")  
 fun novoComentario(@PathVariable id: UUID, @Body @valid request: ComentarioRequest): HttpResponse<Any> {  
  
 val comentario = request.toModel()  
 if (isDuplicado(comentario)) {  
 return HttpResponse.unprocessableEntity() // status 422  
 }  
  
 service.comenta(id = id, comentario = comentario)  
 return HttpResponse.ok()  
 }  
  
 private fun isDuplicado(request: ComentarioRequest): Boolean {  
 // implementação não importante  
 }  
}  
  
@Singleton  
class ArtigoService(@Inject val repository: ArtigoRepository) {  
  
  
 @Transactional  
 fun comenta(id: UUID, comentario: Comentario) {  
  
 val artigo = repository.findById(id)  
 if (artigo == null) {  
 throw ArtigoNaoEncontradoException("Artigo não encontrado")  
 }  
  
 artigo.comenta(comentario)  
  
 // atualiza entidade no banco  
 repository.update(artigo)  
 }  
}  
  
@Repository  
interface ArtigoRepository : JpaRepository<Artigo, UUID> {  
  
}  
  
@Entity  
class Artigo(  
 @Id @GeneratedValue   
 val id: UUID,   
 @OneToMany(mappedBy = "artigo", cascade = CascadeType.ALL, ...)  
 val comentarios: MutableList<Comentario>,  
 // outros atributos  
) {  
  
 /\*\*  
 \* Adiciona novo comentário ao artigo  
 \*/  
 fun comenta(comentario: Comentario) {  
 comentario.artigo = this  
 this.comentarios.add(comentario)  
 }  
  
}  
  
@Entity  
class Comentario(  
 @Id @GeneratedValue   
 val id: Long,   
 @ManyToOne  
 var artigo: Artigo,  
 // outros atributos  
)  
  
class ArtigoNaoEncontradoException(message: String?) : RuntimeException(message) {  
   
}

Um novo desenvolvedor(a) junior entrou no time e ficou responsável por fazer alterações no código acima, mas ele(a) resolveu tirar algumas dúvidas com você por ser iniciante com Micronaut e JPA. Por esse motivo, com base no código acima, como você responderia as perguntas abaixo feitas pelo desenvolvedor(a) junior:

1. Quantas transações são abertas ao executar a API REST para adicionar um novo comentário no artigo? Explique sua resposta;
2. De acordo com a resposta anterior, quais classes e métodos são responsáveis por iniciar e comitar as transações? Explique sua resposta;
3. O que acontece com a transação quando o controller retorna o status de erro HTTP 422 (Unprocessable Entity)?
4. O que acontece com a transação quando uma exceção ArtigoNaoEncontradoException é lançada da classe service? Explique sua resposta;
5. Imagine que a anotação @Transactional seja removida do controller, a lógica para inserir um novo comentário continuaria funcionando ou algo quebraria? Explique sua resposta;
6. Imagine que o trecho de código repository.update(artigo) seja removido da classe service, o comentário ainda seria inserido corretamente no banco de dados? Explique sua resposta;

## O que seria bom ver nessa resposta?

* **Peso 4**: Demonstrar domínio sobre como o controle transacional funciona ao anotar um método ou classe com a anotação @Transacional do Micronaut;
* **Peso 4**: Demonstrar entendimento sobre como e quando o controle transacional do Micronaut faz o rollback de uma transação em caso de erros;
* **Peso 2**: Ter conhecimento sobre como o contexto de persistência da JPA/Hibernate funciona (tem a ver com os estados de uma entidade e o mecanismo dirty checking);

## Resposta do Especialista:

1. Uma única transação. Um método anotado com @Transactional é o responsável por iniciar uma transação ou participar de uma transação caso exista uma aberta, como um controller geralmente é o ponto de partida de um fluxo de negócio podemos assumir que a transação inicia e termina nele;
2. Somente o método novoComentario() do controller inicia e comita a transação. Por ele ser o ponto de entrada do fluxo e estar anotado com @Transactional ele é o responsável por iniciar e comitar a transação, enquanto o método comenta() da classe service apenas participa da transação corrente que foi aberta pelo controller;
3. Por ser um retorno válido do método, a transação é comitada normalmente. Geralmente um rollback acontece quando uma exceção lançada interrompe a chamada do método anotado com @Transactional, mas não foi o caso;
4. A transação é desfeita (rollback) pois a exceção ArtigoNaoEncontradoException sobe acima do método do controller (ela não é tratada dentro do controller), que foi o método responsável por iniciar a transaçao corrente. Para o controle transacional do Micronaut uma unchecked-exception lançada significa que deve ser feito o rollback;
5. Continuaria funcinonando pois o método da classe service também está anotado com @Transactional. Desse modo, esse método seria o responsável por demarcar onde uma transação inicia e termina. Mesmo se a exceção ArtigoNaoEncontradoException fosse lançada ainda teríamos o mesmo comportamento: rollback. No final das contas, para esse caso em particular ter somente o service anotado seria suficiente para fazer o controle transacional apropriado;
6. Sim, o comentário seria inserido normalmente. Isso acontece porque o contexto de persistência da JPA se mantém aberto enquanto a transação corrente estiver aberta. O contexto de persistência na JPA nada mais é do que a instância da EntityManager utilizada internamente pelo repository. Quando o Micronaut inicia a transação ele faz a ligação (binding) da transação corrente com a EntityManager utilizada pelo repository. Portanto, quando a transação é comitada a JPA faz a verificação das entidades gerenciadas que foram alteradas para aplicar as mudanças ainda não sincronizadas com o banco de dados (esse mecanismo é chamado de dirty checking), se houver mudanças os comandos SQL são gerados e enviados ao banco de dados;