

Logo da Orange Talents

# Gravando os dados com Micronaut Data

## Cenário:

Imagine que acabamos de entrar em um novo time (squad) na Zup que cuida de um microsserviço responsável pelo gerenciamento de tickets de um dos nossos clientes (algo no estilo JIRA ou GitHub Issues). Nosso tech lead, como primeira tarefa, nos pede para modelar um conceito importante de negócio dentro do sistema: conceito de ticket. Para essa primeira etapa, ele nos passou os requisitos levantados com nosso cliente, portanto um ticket para o cliente possui os seguintes atributos com suas respectivas restrições:

* um ticket deve ter um título, uma descrição, data de criação e status;
* o título deve ser um texto com tamanho máximo de 60 caractéres;
* a descrição deve ser um texto com tamanho máximo de 4000 caractéres;
* a data de criação deve armazenar data e hora e deve ser informada pelo próprio sistema;
* o status pode estar nos seguintes estados: ABERTO, FECHADO e EM\_ANALISE;
* um ticket deve sempre iniciar com status ABERTO;

Esse ticket poderá ser inserido, atualizado, deletado e pesquisado dentro do sistema, portanto nosso código deve permitir estas operações básicas. Além disso, não podemos inserir dados inválidos na tabela do banco de dados.

Como você modelaria um ticket com JPA e permitiria fazer sua persistência em um projeto com Micronaut?

## O que seria bom ver nessa resposta?

* **Peso 4**: Criar uma entidade da JPA para representar um ticket com os atributos indicados e anotados com as anotações básicas da JPA, além de criar um repository para entidade com Micronaut;
* **Peso 2**: Adicionar uma ou mais validações a nível de banco de dados, como obrigatoriedade e tamanho máximo (geralmente via anotação @Column ou DDL);
* **Peso 2**: Adicionar uma ou mais validações a nível de aplicação via as anotações da Bean Validation;
* **Peso 1**: Modelar e mapear o atributo status como enum em vez de outros tipos mais abertos como Integer ou String por exemplo;
* **Peso 1**: Inicializar os atributos status e data de criação ao instanciar a entidade (por exemplo, via construtor ou diretamente na declaração do atributo);

## Resposta do Especialista:

* Crio uma classe Ticket para representar o conceito de ticket no modelo de domínio da nossa aplicação. Essa classe possui os seguintes atributos:
  + atributo **titulo** como String;
  + atributo **descricao** como String;
  + atributo **criadoEm** como LocalDateTime;
  + atributo **status** como enum do tipo StatusDoTicket possuindo 3 constantes: ABERTO, FECHADO e EM\_ANALISE. Aqui uso uma enum pois ela representa de forma segura e tipada os possíveis estados da nosso ticket;
* Ainda na classe Ticket, crio um construtor para receber os atributos obrigatórios: **titulo** e **descricao**. Recebo apenas estes 2 atributos no construtor pois o **status** já inicia como ABERTO enquanto o **criadoEm** inicializa com a data atual;
* Se não houver as dependências da JPA/Hibernate configuaradas no Maven ou Gradle do projeto, eu as adiciono. Nesse caso, eu sigo as orientações da documentação oficial do Micronaut pois não tenho como decorar;
* Para mapear a classe para uma entidade do banco de dados, uso as anotações da JPA. Nesse caso, anoto a classe com @Entity para que ela seja mapeada para uma tabela no banco, e crio um novo atributo **id** do tipo Long anotado com @Id para representar a chave primária (PK) da tabela. Por fim, anoto o **id** com @GeneratedValue para que a JPA delegue a geração da PK para o banco de dados;
* Para reforçar a integridade dos dados do schema no banco de dados, eu adiciono constraints na colunas via anotações, afinal de contas a aplicação pode conter bugs e o banco é nossa última barreira de defesa:
  + anoto o atributo **titulo** com @Column(nullable=false, length=60) para garantir obrigatoriedade e tamanho máximo da coluna;
  + anoto o atributo **descricao** com @Column(nullable=false, length=4000) para garantir obrigatoriedade e tamanho máximo da coluna;
  + anoto o atributo **status** com @Column(nullable=false) para obrigatoriedade e também com @Enumerated(STRING) para indicar a JPA para armazenar a representação da enum no banco como texto (usando as constantes da enum em vez da posição);
  + anoto o atributo **criadoEm** como @Column(nullable=false, updatable=false) para obrigatoriedade e também para impossibilitar que a aplicação altere sem querer a coluna em operações de atualização;
* Para validar os dados da entidade, eu anoto os atributos com as anotações da Bean Validation, como @NotBlank, @Size, @NotNull e @PastOrPresent. A idéia aqui é ter validação a nível de aplicação disparada pela JPA sempre que tentarmos inserir ou atualizar um ticket no sistema, ou seja, nem seria necessário bater no banco de dados para validar os dados;
* Agora, crio o repository TicketRepository para fazer as operações de CRUD. Para isso, a crio como uma interface que estende de JpaRepository do Micronaut e a anoto com @Repository, dessa forma eu não preciso manipular uma EntityManager em operações básicas;
* Por fim, se já não estiver configurado, eu configuro o datasource e a JPA/Hibernate no application.yml para acessar o banco de dados e também para que a JPA crie a nova tabela e constraints a partir da entidade. Levanto a aplicação e verifico se o schema foi gerado como esperado;