

Logo da Orange Talents

# Fluxo completo para salvar um objeto no banco de dados

## Cenário:

Com o conhecimento de Micronaut adquirido até este momento, imagine que precisamos criar uma API REST para cadastrar um veículo (mais especificamente um carro) no sistema. Para isso, um carro deve possuir as seguintes informações:

* um carro deve possuir placa e modelo;
* o modelo deve possuir máximo de 60 caracteres;
* a placa deve ser validada via regex [A-Z]{3}[0-9][0-9A-Z][0-9]{2};
* a placa do carro deve ser única, portanto o sistema não deve permitir carros duplicados;

Os dados precisam ser validados e em seguida gravados no banco de dados. Em caso de sucesso a API REST deve retornar o status HTTP 201, caso contrário deve retornar o status HTTP 400 para erros de validação ou status HTTP 500 para erros inesperados no sistema.

Como você implementaria esta API REST com Micronaut?

## O que seria bom ver nessa resposta?

* **Peso 5**: Criar um controller com um método do tipo POST recebendo os dados do carro via DTO, validando os dados via Bean Validation, convertendo o DTO para entidade da JPA e persistindo a entidade via repository;
* **Peso 3**: Adicionar validação de unicidade através de constraint no banco de dados, por exemplo via @Column(unique=true), @Table ou mesmo DDL;
* **Peso 2**: Criar método existsByPlaca() no repository e usá-lo de alguma forma para verificar se determinada carro já existe no sistema;

## Resposta do Especialista:

* Crio um controller do Micronaut com um método do tipo POST para receber os dados do carro. O motivo de usar um verbo HTTP POST em vez de outro se deve ao fato de ser uma operação de criação de recurso, no caso o carro; além disso, crio um DTO especifico para receber os dados (placa e modelo) no método, dessa forma isolo o modelo de API REST do modelo de domínio da aplicação;
* Ainda no controller, anoto a classe com @Validated e seu método com @Valid para que o Micronaut consiga disparar as validações da Bean Validation antes de executar o método de fato, desse modo eu tenho certeza que a execução do método ocorre com dados minimamente válidos. Enquanto no DTO, eu anoto seus atributos com as anotações da Bean Validation:
  + anoto o atributo **nome** com @NotBlank e Size(max=60) por causa da obrigatoriedade e restrição de tamanho;
  + anoto o atributo **placa** com @NotBlank por causa da obrigatoriedade e com @Pattern passando a regex sugerida para garantir o formato válido da placa;
* Crio uma classe Carro como entidade da JPA para representar um carro dentro sistema, adiciono os atributos **placa** e **modelo** como String, **criadoEm** como LocalDateTime para representar a data de criação, e por fim um **id** como UUID (aqui usei UUID para facilitar uso como ID opaco numa API REST). Em seguida anoto a classe com @Entity para que a JPA consiga mapeá-la para uma tabela no banco de dados e anoto o **id** com @Id e @GeneratedValue para representar uma PK auto-gerada pelo Hibernate. Para os demais atributos, eu reforço a integridade dos dados no banco através de constraints pois o banco é nossa última barreira de defesa:
  + anoto os atributos **nome** e **placa** com @Column(nullable=false, length=xxx) para garantir obrigatoriendade e tamanho da coluna no banco;
  + anoto também o atributo **placa** com @Column(..., unique=true) para garantir que a tabela não permita linhas duplicadas mesmo que a validação da aplicação falhe (sem essa constraint há grandes riscos de inserir linhas duplicadas em cenários de concorrência);
  + anoto o atributo **criadoEm** com @Column(nullable=false, updatable=false) para garantir obrigatoriedade e não permitir que a aplicação altere o valor da coluna em operações de atualização;
* Crio um repository para entidade Carro, que basicamente é significa criar uma interface CarroRepository que estende da JpaRepository do Micronaut e a anotar com @Repository, desse modo posso executar operações de CRUD em cima da entidade;
* No controller, injeto o repository via construtor e o utilizo dentro do método do controller para gravar a entidade no banco de dados. Mas antes converto o DTO da API REST para entidade Carro através de um método toModel() no próprio DTO;
* No repository, eu crio o método boolean existsByPlaca(string) para verificar se existe algum carro com determinada placa, dessa forma posso utilizá-lo no controller sem a necessidade de escrever uma query JPQL ou mesmo SQL;
* Agora no método do controller, verifico se a placa informada já existe no banco através do método existsByPlaca() do repository. Caso ela exista, eu lanço uma exception HttpStatusException com o status HTTP 400 juntamente com a mensagem de erro "carro já existente no sistema". A idéia aqui é fazer essa verificação a nível de aplicação para que possamos falhar o mais cedo possível (fail-fast), além de ter um controle fino do erro e de poupar o uso de recursos no fluxo de execução;
* Por fim, em caso de sucesso eu retorno do método do controller um status HTTP 201 passando o cabeçalho HTTP location indicando a URI para acessar o novo recurso criado para o carro (aqui uso o **id** da entidade gerado pelo Hibernate);