

Logo da Orange Talents

# Expondo dados via GRPC

## Cenário:

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

* um carro deve possuir placa (ex: HPX-1234, OIP-9876 etc);
* um carro deve possuir modelo (ex: Gol, Palio etc);

Nesse momento, não se preocupe com a validação dos dados de entrada. Os dados precisam apenas ser gravados em um banco de dados. Em caso de sucesso a API gRPC deve retornar o ID interno gerado pelo sistema.

Como você implementaria esta API gRPC com Micronaut?

## O que seria bom ver nessa resposta?

* **Peso 5**: Desenhar a API gRPC via Protobuf e implementar a classe de endpoint como serviço do Micronaut, criar entidade da JPA e fazer a persistência via repository do Micronaut;
* **Peso 2**: Fazer testes manuais com a ferramenta BloomRPC ou Insomnia para verificar funcionamento da API;
* **Peso 2**: Usar extension functions do Kotlin para fazer a conversão da request para entidade;
* **Peso 1**: Habilitar logs SQL da JPA/Hibernate;

## Resposta do Especialista:

* Primeiramente desenho a API em um arquivo .proto seguindo os requisitos da atividade. Nesse arquivo teria algo como:
* service CarrosGrpcService {  
   rpc adicionar(NovoCarroRequest) returns (NovoCarroResponse) {}  
  }  
    
  message NovoCarroRequest {  
   string placa = 1;  
   string modelo = 2;  
  }  
    
  message NovoCarroRequest {  
   int32 id = 1;  
  }
* Depois, uso a IDE e Gradle para gerar os stubs a partir desse .proto. Dessa forma, eu posso criar uma classe para representar o endpoint da API gRPC (chamo ela de NovoCarroEndpoint). Essa classe estenderia a classe "ImplBase" gerada pelo compilador do Protobuf, e também faria a sobrescrita do método adicionar() da classe mãe;
* Anoto a classe NovoCarroEndpoint com a anotação @Singleton do Micronaut para que ela possa se gerenciada e registrada no container do framework;
* Faço a implementação mais simples desse método retornando uma response com um id qualquer. Em seguida, levanto a aplicação na IDE e uso a ferramenta BloomRPC para fazer o consumo da API gRPC passando os dados necessários. A idéia aqui é ter certeza que a configuração básica para rodar a API gRPC com Micronaut está ok;
* Agora, crio a entidade da JPA Carro com os atributos placa e modelo como String, além de um atributo id do tipo Long. Anoto a classe com @Entity e o atributo id com @Id. Em seguida crio um repository CarroRepository que estende de JpaRepository e a anoto com @Repository para poder fazer a persistência da entidade;
* Por se tratar de ambiente de desenvolvimento, habilito os logs SQL da JPA/Hibernate para poder visualizar o SQL gerado (show\_sql=true e format\_sql=true);
* Na classe de endpoint, injeto o repository via construtor. E implemento a lógica para fazer a persistência de um novo carro no banco de dados. Para tranformar o DTO de request em entidade eu crio uma extension function do Kotlin para facilitar o trabalho e simplificar o design:
* fun NovoCarroRequest.toModel(): Carro {  
   return Carro(  
   placa = this.placa,   
   modelo = this.modelo  
   )  
  }
* Após gravar a entidade no banco via repository, eu retorno a response com o id da entidade gerado pelo Hibernate. Aqui basicamente invoco os método onNext() passando a response e onComplete(), ambos da StreamObserver do gRPC;
* Por fim, levanto a aplicação pela IDE e testo novamente a API via BloomRPC. Em caso de erro, analiso o erro retornado pelo gRPC e também verifico os detahes no console da IDE para fazer as correções;