

Logo da Orange Talents

# Retornando códigos de erro em uma API gRPC

## Antes de começar

Para complementar o curso, recomendamos estudar a documentação oficial da Google Cloud APIs sobre [gRPC: Error Model](https://cloud.google.com/apis/design/errors) caso você não tenho estudado ainda. Nela é possível ter uma idéia mais clara do poder e das possibilidades oferecidas pela tecnologia na hora de tratar e manipular erros em uma API gRPC.

## Cenário:

Um novo desenvolvedor(a) que acabou de entrar no seu time da Zup implementou uma API gRPC com Micronaut para cadastrar carros no sistema. Após finalizar a implementação, ele(a) submeteu o código para que alguém do time pudesse fazer o Code Review.

Por coincidência você é o único membro do time disponível no momento para fazer essa revisão de código. A Pull Request (PR) com o código submetido é este abaixo:

@Singleton  
class NovoCarroEndpoint(  
 @Inject val repository: CarroRepository   
) : CarrosGrpcServiceGrpc.CarrosGrpcServiceImplBase() {  
  
 override fun adicionar(request: NovoCarroRequest, responseObserver: StreamObserver<NovoCarroResponse>) {  
  
 if (repository.existsByPlaca(request.placa)) {  
 // tratamento de erro #1  
 throw CarroExistenteException("carro com placa existente")  
 }  
  
 try {  
  
 val carro = Carro(  
 placa = request.placa,   
 modelo = request.modelo  
 )  
  
 repository.save(carro)  
  
 } catch (e: ConstraintViolationException) {  
 // tratamento de erro #2  
 responseObserver.onError(e)  
 return;  
 } catch (e: Exception) {  
 // tratamento de erro #3  
 throw StatusRuntimeException(Status.INTERNAL  
 .withDescription("erro interno inesperado")  
 .withCause(e))  
 }  
   
 responseObserver.onNext(NovoCarroResponse.newBuilder().setId(carro.id).build())  
 responseObserver.onCompleted()  
 }  
}  
  
@Repository  
interface CarroRepository : JpaRepository<Carro, Long> {  
  
 fun existsByPlaca(placa: String): Boolean  
}  
  
@Entity  
class Carro(  
 @NotBlank @Placa val placa: String,   
 @NotBlank @Size(max=42) val modelo: String  
) {  
 @Id  
 @GeneratedValue  
 val id: Long? = null  
}

Com base no seu conhecimento sobre tratamento de erros com gRPC, e olhando para o código submetido acima, mais especificamente os 3 blocos de comentários indicados no código com "**tratamento de erro #n**", precisamos que você responda as seguintes perguntas:

1. Nos 3 blocos **tratamento de erro**: quais os status de erro retornados pela API gRPC? Explique sua resposta;
2. Sobre o bloco **tratamento de erro #1**: qual status você acha apropriado para retornar nesse tipo de erro? Explique sua resposta;
3. Sobre o bloco **tratamento de erro #2**: qual status você acha apropriado para retornar nesse tipo de erro? Explique sua resposta;
4. Sobre o bloco **tratamento de erro #1**: você acha que o código está correto? Caso não esteja, o que você mudaria para ele funcionar?
5. Sobre o bloco **tratamento de erro #2**: você acha que o código está correto? Caso não esteja, o que você mudaria para ele funcionar?
6. Sobre o bloco **tratamento de erro #3**: você acha que o código está correto? Caso não esteja, o que você mudaria para ele funcionar?

## O que seria bom ver nessa resposta?

* **Peso 3**: Demonstrar domínio em tratamento de erros com gRPC identificando que todos os 3 blocos de código estão incorretos e reportam o status UNKNOWN por padrão. Também ter idéia dos possíveis status que cada bloco de código deveria (ou poderia) retornar;
* **Peso 4**: Propor uma solução para cada bloco de código para retornar um status apropriado (aqui a solução mais simples do onError() passando um status e uma descrição para o usuário é suficiente);
* **Peso 2**: Sugerir mudança de código do bloco **tratamento de erro #2** para retornar via API com.google.rpc.Status as mensagens de erro da Bean Validation como detalhe da resposta, geralmente criando um novo tipo de mensagem no .proto com um atributo para representar a lista de erros (aqui não precisa lembrar de todo o código pois é muito decoreba);
* **Peso 1**: Sugerir mudança de código do bloco **tratamento de erro #2** para retornar via API com.google.rpc.Status as mensagens de erro da Bean Validation onde o payload de detalhes **aproveita** a API BadRequest do próprio gRPC (aqui não precisa lembrar de todo o código pois é muito decoreba);

## Resposta do Especialista:

1. Todos os 3 blocos retornam o status UNKNOWN pois eles estão reportando erros de forma incorreta com gRPC. Quando isso acontece o framework retorna o status UNKNOWN sem detalhes para que não haja riscos de vazar informações sensíveis na API. Portanto, a maneira correta de reportar erros via gRPC é através do método StreamObserver.onError() passando como parâmetro uma instância de StatusRuntimeException;
2. Eu retornaria o status ALREADY\_EXISTS pois representa bem a lógica de negócio. Contudo, os status INVALID\_ARGUMENT ou FAILED\_PRECONDITION também são aceitáveis, especialmente quando por questões de segurança não temos a intenção de detalhar os erros na API para o usuário ou outro sistema;
3. Eu retornaria o status INVALID\_ARGUMENT pois trata-se de erros de validação da entrada de dados submetidos para API gRPC;
4. Não, o código está incorreto. Para corrigi-lo eu assumo que o status de erro será ALREADY\_EXISTS com a descrição existente no código. Invoco o método responseObserver.onError() passando a exception criada e adiciono um return para finalizar o fluxo de execução. O código ficaria semelhante a este abaixo:

* responseObserver.onError(Status.ALREADY\_EXISTS  
   .withDescription("carro com placa existente")  
   .asRuntimeException())  
  return

1. Não, o código está incorreto. Para corrigi-lo eu assumo que o status de erro será INVALID\_ARGUMENT com uma descrição genérica "dados de entrada inválidos". Agora, eu uso a API com.google.rpc.Status para criar uma nova mensagem pois ela suporta adicionar detalhamentos na resposta, assim posso devolver uma lista de erros. Para o detalhamento, eu converto a lista de violações da Bean Validation contidas na exception ConstraintViolationException para uma instância de BadRequest, que é um tipo existente na API do gRPC; em seguida eu empacoto esse detalhamento na mensagem de erro (details da classe Status). Por fim, eu gero uma StatusRuntimeException via método StatusProto.toStatusRuntimeException() passando a instância da mensagem (Status), e passo exception criada para o método responseObserver.onError(); termino o código com um return para parar o fluxo. O código ficaria semelhante a este:

* val badRequest = BadRequest.newBuilder()  
   .addAllFieldViolations(e.constraintViolations.map {  
   BadRequest.FieldViolation.newBuilder()  
   .setField(it.propertyPath.last().name)  
   .setDescription(it.message)  
   .build()  
   }).build()  
    
  val statusProto = Status.newBuilder() // com.google.rpc.Status  
   .setCode(Code.INVALID\_ARGUMENT\_VALUE)  
   .setMessage("dados de entrada inválidos")  
   .addDetails(Any.pack(badRequest)) // com.google.protobuf.Any  
   .build()  
    
  val exception = StatusProto.toStatusRuntimeException(statusProto)  
  responseObserver.onError(exception)  
  return

1. Não, o código está incorreto. Para corrigi-lo basta repassar a instância de StatusRuntimeException para o método responseObserver.onError() e adicionar um return para encerrar o fluxo:

* responseObserver.onError(Status.INTERNAL  
   .withDescription("erro interno inesperado")  
   .withCause(e)  
   .asRuntimeException())  
  return