

Logo da Orange Talents

# Lidando com erros retornados pela API gRPC

## Cenário:

Um novo desenvolvedor(a) junior ficou responsável de implementar um microsserviço REST com Micronaut que será utilizado em um aplicativo mobile de entregas de produtos. Esse microsserviço irá consumir uma API gRPC na qual é responsável por consultar cupons de compra e retornar o valor do cupom. Para isso, o desenvolvedor(a) implementou um controller com Micronaut dessa forma:

@Controller  
class CupomController(  
 @Inject  
 val grpcClient: CupomGrpcServiceGrp.CupomGrpcServiceBlockingStub  
) {  
  
 private val LOGGER = LoggerFactory.getLogger(this::class.java)  
  
 @Get("/api/cupons/{cupom}")  
 fun consultar(@NotBlank @PathVariable cupom: String): HttpResponse<Any> {  
  
 LOGGER.info("Consultando `$cupom` informado no serviço externo")  
 val response = grpcClient.consultar(  
 CupomRequest.newBuilder().setCupom(cupom).build()  
 )  
  
  
 return HttpResponse.ok(CupomResponse(cupom, response.valor))  
 }  
}

Apesar do código funcionar, ele foi recusado durante a revisão de código por um desenvolvedor senior do time pois o código não está fazendo o tratamento de erros. Atualmente, este serviço gRPC de cupons pode reportar os seguintes erros:

* NOT\_FOUND: cupom não encontrado
* INVALID\_ARGUMENT: formato do cupom inválido ou não informado
* FAILED\_PRECONDITION: cupom expirado ou já utilizado
* UNKNOWN: erro interno, inesperado ou desconhecido

Para piorar, o desenvolvedor(a) junior não sabe como tratar erros ao consumir uma API gRPC pois este é seu primeiro contato com a tecnologia gRPC. Seu papel aqui é ajuda-lo(a) a refatorar o código para que ele possa tratar os erros lançados pelo serviço gRPC.

No código acima, como você faria para tratar os possíveis erros reportados pela API gRPC de consulta de cupom?

## O que seria bom ver nessa resposta?

* **Peso 7**: Tratar cada um dos 4 possíveis status de erro gRPC, mapeando-os para seus respectivos status HTTP e retornando-os na API REST. Por ser algo simples, usar try-catch ou algum exception handler do Micronaut é indiferente;
* **Peso 2**: Ter cuidado para não expor dados sensíveis ao receber o erro UNKNOWN ou qualquer outro inesperado do serviço gRPC, pois trata-se de um erro desconhecido na qual sua mensagem pode conter dados sensíveis da aplicação;
* **Peso 1**: Logar a exception como ERROR ou WARN para facilitar o troubleshooting (útil para os erros UNKNOWN ou inesperados);

## Resposta do Especialista:

* Por se tratar de um tratamento simples e pontual, eu adiciono um try-catch na chamada ao serviço gRPC para capturar qualquer exceção do tipo StatusRuntimeException, que é o tipo de exceção padrão lançado em chamadas gRPC;
* Dentro do bloco do catch, eu faço o **mapeamento dos possíveis erros reportados pelo serviço gRPC para seus respectivos status HTTP ou similares**, inclusive eu tomo o devido cuidado para não expor dados sensíveis quando a API me retornar o status UNKNOWN ou qualquer outro inesperado, afinal, trata-se de um erro interno lançado pelo serviço e não faz muito sentido expor diretamente para o usuário final no frontend. No fim, eu teria um código semelhante a este:
* try {  
   // ...  
  } catch (e: StatusRuntimeException) {  
    
   val statusCode = e.status.code  
   val statusDescription = e.status.description ?: ""  
    
   val (httpStatus, message) = when (statusCode) {  
   Status.NOT\_FOUND.code -> Pair(HttpStatus.NOT\_FOUND, statusDescription)  
   Status.INVALID\_ARGUMENT.code -> Pair(HttpStatus.BAD\_REQUEST, statusDescription)  
   Status.FAILED\_PRECONDITION.code -> Pair(HttpStatus.UNPROCESSABLE\_ENTITY, statusDescription)  
   else -> Pair(HttpStatus.INTERNAL\_SERVER\_ERROR, "Não foi possivel completar a requisição")  
   }  
    
   throw HttpStatusException(httpStatus, message)   
  }
* Para melhorar o troubleshooting, eu logo a exceção quando acontecer qualquer erro desconhecido ou inesperado (diferente de NOT\_FOUND, INVALID\_ARGUMENT e FAILED\_PRECONDITION):
* val (httpStatus, message) = when (statusCode) {  
   // ...  
   else -> {  
   LOGGER.error("Erro inesperado ao consultar cupom '$cupom' no serviço externo", e)  
   Pair(HttpStatus.INTERNAL\_SERVER\_ERROR, "Não foi possivel completar a requisição")  
   }  
  }
* Por fim, levanto a aplicação e faço os devidos testes via POSTman ou Insomnia para ter certeza que o tratamento de erro está funcionando como esperado;