



# Kresil - Kotlin Multi-Platform Library for Fault-Tolerance

Francisco Engenheiro, n.º 49428, e-mail: a49428@alunos.isel.pt, tel.: 928051992

Orientadores: Pedro Félix, e-mail: pedro.felix@isel.pt

Março de 2024

## 1 Contexto

### 1.1 Necessidade de Desenho de Software Resiliente

Os sistemas distribuídos representam um conjunto de computadores independentes e interligados em rede, que se apresentam aos utilizadores como um sistema único e coerente [1].

Dado a constante necessidade destes sistemas estarem disponíveis, aliados à sua complexidade de funcionamento, é natural que estejam suscetíveis a falhas de comunicação, de hardware, de software, entre outras. Por esse motivo, existe a necessidade de garantir que os serviços que disponibilizam sejam resilientes, e mais concretamente, tolerantes a falhas.

Um serviço tolerante a falhas, é um serviço que é capaz de manter a sua funcionalidade total ou parcial, ou apresentar uma alternativa, quando um ou mais componentes que lhes estão associados falham. De forma a alcançar este objetivo, foram desenhados mecanismos de resiliência. Alguns exemplos:

- **Retry:** Tenta novamente uma operação que falhou, aumentando a sua probabilidade de sucesso;
- **Rate Limiter:** Limita a taxa de requisições que um determinado serviço pode receber;
- **Circuit Breaker:** Interrompe, temporariamente, a comunicação com um serviço que está a falhar, de forma a evitar que o mesmo sobrecarregue o sistema. Semelhante a um disjuntor elétrico;
- **Fallback:** Fornece um valor ou executa uma ação alternativa caso uma operação falhe.

## 1.2 Mecanismos de Resiliência

Existem bibliotecas que fornecem mecanismos de resiliência (Tabela 1). Estes atuam em tempo de execução e implementam uma determinada estratégia. A configuração de um mecanismo de resiliência é feita através de uma política que define o seu comportamento.

Tabela 1: Exemplos de bibliotecas que fornecem mecanismos de resiliência.

Biblioteca	Linguagem	Plataforma
Netflix's Hystrix [2]	Java	JVM
Resilience4j [3]	Java/Kotlin	JVM
Polly [4]	C#	.NET

A biblioteca *Polly* [4] divide os mecanismos de resiliência em duas categorias:

- **Resiliência Reativa:** Reage a falhas e mitiga o seu impacto (e.g., *Retry*, *Circuit Breaker*);
- **Resiliência Proativa:** Previne que as falhas aconteçam (e.g., *Rate Limiter*, *Timeout*).

## 1.3 Kotlin Multiplatform

A tecnologia *Kotlin MultiPlatform* [5] (*KMP*) possibilita a partilha de código entre várias plataformas. A sua arquitetura (Figura 1) é composta por três categorias de código principais:

- **Comum:** Código partilhado entre todas as plataformas (i.e., *CommonMain*, *CommonTest*);
- **Intermediário:** Código que pode ser partilhado num subconjunto de plataformas (i.e., *AppleMain*, *AppleTest*);
- **Específico:** Código específico de uma plataforma-alvo (i.e., *<Platform>Main*, *<Platform>Test*).



Figura 1: Exemplo de uma arquitetura *KMP*.

O objetivo principal é a maximização da reutilização de código, ou seja, agregar o máximo de código possível nas categorias hierarquicamente superiores. No entanto, por vezes é necessário criar código específico para uma plataforma-alvo, regularmente denominada como *target*, nas seguintes situações:

- Uma determinada funcionalidade não consegue ser implementada de forma comum porque:
  - é necessário acesso a API's específicas do *target*;

- as bibliotecas disponíveis para código comum (i.e., *Standard Kotlin Library*, *Kotlinx*) não cobrem as funcionalidades pretendidas;
- Um determinado *target* não suporta diretamente o *KMP* (e.g., *Node.js*), e por isso é necessário criar um *adapter*. Este permite a comunicação com o código comum, em *Kotlin*, a partir do código nativo do *target*, e que pode estar definido na categoria *Intermediário* ou *Específico*.

Para criar código específico para um *target* é utilizado o mecanismo *expect/actual* [6], que permite a definição do código a ser implementado e a sua implementação, respetivamente.

## 1.4 Ktor

*Ktor* [7] é uma framework *KMP* modular para desenvolver sistemas (i.e., aplicações web, bibliotecas, microserviços) assíncronos de servidor e cliente. Desenvolvida pela *JetBrains*, foi construída com *Kotlin* puro (i.e., sem dependências de outras bibliotecas) e está integrada com o sistema de *Coroutines*. Sistema esse que permite a definição de código assíncrono de forma sequencial e a sua execução sem bloqueio de threads, tirando maior proveito do sistema computacional disponível.

## 2 Problema

A análise das bibliotecas mais usadas que fornecem mecanismos de resiliência permitiu concluir que não existem bibliotecas que suportem *KMP*. A título de exemplo, a biblioteca *Resilience4j* [3] que foi desenhada para *Java*, já providencia um módulo de interoperabilidade com *Kotlin*, mas apenas exclusivamente para a *JVM*. Por esse motivo, aplicações *KMP* que necessitem de mecanismos de resiliência têm de escolher entre:

- recorrer a bibliotecas que fornecem mecanismos de resiliência e que são específicas para cada *target*, o que aumenta a complexidade e a redundância do código;
- implementar a sua própria solução, o que aumenta, principalmente, o tempo de desenvolvimento.

## 3 Solução

Dado o contexto e o problema apresentado, a solução proposta passa por:

1. Construir uma biblioteca *open-source* que forneça mecanismos de resiliência multiplataforma em *Kotlin*, utilizando a tecnologia *KMP*.
2. Simplificar a utilização de mecanismos de resiliência em clientes e serviços que utilizam a framework *Ktor*.

## 4 Desafios e Potenciais Riscos

### 4.1 Desafios

- Primeiro projeto em *KMP* do arguente;
- Precisar de funcionalidades que não estão na biblioteca standard do *Kotlin* ou noutras, mas que são necessárias para a implementação da biblioteca;
- Testar a biblioteca em diferentes *targets*. De referir que o *target iOS* será excluído da lista de *targets* suportados, visto que o arguente não possui um dispositivo *iOS* ou outro meio de testar a biblioteca nesse *target*.
- Criar *adapters* para *targets* que não suportam diretamente o *KMP*.

- Integrar a biblioteca com a framework *Ktor*.

## 4.2 Riscos

- Defeitos do *KMP*, visto que é uma tecnologia recente e em constante evolução.

## 5 Planeamento

Tabela 2: Cronograma do Projeto

Data	Tarefas
18/03/2024	<b>Entrega da proposta de projeto</b>
25/03/2024	Finalizar o estudo e aprendizagem da tecnologia <i>KMP</i> e da framework <i>Ktor</i> .
01/04/2024	Desenvolvimento da arquitetura da biblioteca baseada na <i>Resilience4j</i>
15/04/2024	Apresentação de pelo menos uma estratégia de resiliência implementada e com testes em várias plataformas
22/04/2024	<b>Apresentação de progresso</b>
29/04/2024	Continuação do desenvolvimento da biblioteca
20/05/2024	Integração com a framework <i>Ktor</i>
03/06/2024	<b>Entrega da versão beta</b>
01/07/2024	Lançamento oficial da biblioteca com documentação completa
13/07/2024	<b>Entrega da versão final</b>

## Referências

- [1] FreeCodeCamp contributors. A thorough introduction to distributed systems. <https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c>, 2024. [Online; accessed 5-March-2024].
- [2] Netflix contributors. Hystrix: Latency and fault tolerance for distributed systems. <https://github.com/Netflix/Hystrix>, 2024. [Online; accessed 6-March-2024].
- [3] resilience4j contributors. Resilience4j: User guide. <https://resilience4j.readme.io/docs/getting-started>, 2024. [Online; accessed 6-March-2024].
- [4] App-vNext contributors. Polly: Resilience strategies. <https://github.com/App-vNext/Polly#resilience-strategies>, 2024. [Online; accessed 6-March-2024].
- [5] JetBrains contributors. Kotlin multiplatform. <https://kotlinlang.org/docs/multiplatform.html>, 2024. [Online; accessed 7-March-2024].
- [6] JetBrains contributors. Kotlin multiplatform: Expect/actual. <https://kotlinlang.org/docs/multiplatform-expect-actual.html>, 2024. [Online; accessed 12-March-2024].
- [7] JetBrains contributors. Ktor: Web applications. <https://ktor.io>, 2024. [Online; accessed 7-March-2024].