Relatório de tolerância a falhas do IMDTravel

Tópicos especiais em engenharia de software IV - Tolerância a falhas em sistemas de software

Docente:

-Gibeon Soares

Discentes:

-Hélio Lima Carvalho

O projeto **IMDTravel** foi desenvolvido como parte da disciplina de Tolerância a falhas.
O sistema simula uma aplicação de compra de passagens áereas que interage com múltiplos micro serviços separados em containers — **IMDTravel**, **AirlinesHub**, **Exchange** e **Fidelity** —, com módulos sujeito a tipos específicos de falhas (omissão, erro, tempo e crash).
O objetivo é aplicar mecanismos de **tolerância a falhas** que permitam a continuidade do funcionamento do sistema mesmo quando alguns desses serviços apresentam problemas.

Estratégias adotadas

Falha de omissão (`/flight`)

Tipo de falha: o serviço pode não responder.
Estratégia: *timeout + retry automático*.

- Se o `axios.get` não responde em até **3 segundos**, o código considera uma **omissão** e tenta novamente a requisição.
- Isso evita que uma requisição travada impeça o fluxo principal de compra.
- Caso todas as tentativas falhem, o sistema lança uma exceção tratável, mantendo a execução controlada.

```
const response = await axios.get("http://airlineshub:3002/search", { timeout: 3000 });
```

👉 **Tolerância implementada através de**: temporização (`timeout`) e tentativa de repetição (`retry`).

Falha por erro (`/convert`)

Tipo de falha: retorna erro.
Estratégia: *uso de cache local e fallback*.

- As últimas **10 taxas bem-sucedidas** são armazenadas em uma lista `ultimasTaxas`.
- Se o serviço `exchange` falhar, o sistema calcula a **média** dessas últimas taxas e a usa como estimativa.
- Caso ainda não existam valores armazenados, é usada uma **taxa padrão (5.5)**.

```
if (ultimasTaxas.length > 0)
    return somaDasTaxas / ultimasTaxas.length;
else
    return 5.5;
```

👉 **Tolerância implementada através de**: *cache e fallback* (continuidade mesmo com falha).

Falha por tempo (`/sell`)

Tipo de falha: o serviço pode responder com atraso ou demorar demais (latência excessiva).
Estratégia: *detecção de latência + limitação de tempo*.

- Timeout do axis para caso não responda a tempo.
- Se a requisição ultrapassar **2 segundos**, mesmo que complete, é tratada como falha de tempo.

```
if (duracao > 2) throw new Error("Latência acima do limite permitido (2s).");
```

👉 **Tolerância implementada através de**: *timeout e falha graciosa*.

Falha de Crash (`/bonus`)

Tipo de falha: o serviço pode ficar indisponível (crash).
Estratégia: *fila de reenvio + verificação periódica*.

- Se o envio do bônus falhar, o sistema salva a operação em uma lista (`listaParaBonificacoes`).
- Um `setInterval` executa a cada **5 segundos** e tenta reenviar os bônus pendentes.
- Cada bonificação tem um **ID único** , permitindo que múltiplas bonificações de um mesmo usuário possam coexistir.

👉 **Tolerância implementada através de**: *reexecução periódica e persistência temporária de estado (fila em memória)*.

Localização e forma de implementação

Os serviços responsáveis pela obtenção de dados e pela aplicação das estratégias de tolerância a falhas foram implementados integralmente no módulo **`RecuperacaoDeFalhas.js`** (localizado na pasta IMDTravel).

Esse módulo é utilizado pelo sistema principal **`server.js`** de **IMDTravel**, que realiza as chamadas a essas funções para obter as informações necessárias e continuar o processo de compra.

Limitações do código

- **Sem persistência real:** a lista de bonificações pendentes é mantida apenas em memória; se o processo reiniciar, as pendências se perdem.

- **O valor do dólar pelo fallback não é confiável:** como é somente uma tentativa de aproximação a partir de coletas passadas, ele não garante que a resposta esteja correta ou sequer próxima. Em uma situação extraordinária que o preço do dólar dispara e não haja compras bem sucedidas nesse meio tempo para balancear o histórico do preço, a empresa de vendas poderia ter prejuízo.

- **Serviço de bonificação sem restart:** apesar do constante retry da bonificação, caso o sistema de Fidelity crash, ele nunca voltará a ativa sem interferência externa, ou seja, vai continuar tentando a bonificação eternamente até que algo/alguém reinicie o Fidelity ou desligue o sistema responsável pelas retrys de bonificação.

- **Retry limitado:** as funções de busca de voo fazem tentativas adicionais, em caso de um Crash, ele não tem como identificar e prosseguir, fazendo-o continuar tentando até que o máximo de tentativas seja atingido.

Outras estratégias possíveis

1. CheckPoint / Persistência de estado

Manter logs de operações ou filas de reenvio em **armazenamento durável**, em caso do sistema cair e vim a perder a memória.
→ Garante recuperação total após reinício do servidor.

2. Circuit Break

Evita sobrecarregar um serviço que está falhando constantemente.
→ Após um número de erros, o circuito "abre" e pausa novas requisições por um tempo.

3. Redundância / Replicação de serviços

Ter mais de uma instância de cada micro serviço e escolher automaticamente a que estiver respondendo melhor.
→ Aumenta a disponibilidade e reduz falhas por crash.

4. WatchDog 🐶

Como dito anteriormente, caso algum sistema crash, destino provável do Fidelity, não há nada que identifique e tome alguma atitude de recuperação, ter um watchdog para monitorar o sistema e acionar um reboot caso necessário é essencial para esse tipo de falha.

→ Identifica falhas por crash.