# 4 PROPOSTA DE SOLUÇÃO

A proposta deste trabalho consiste na implementação de um módulo desacoplado para o gerenciamento de falhas em aplicativos Flutter, voltado para o gerenciamento de qualquer aplicações. O objetivo é avaliar se o processo de gerenciamento cobre diversas falhas e erros diferentes, mantem a aplicação funcionando com um comportamento normal, além de capturar os erros e eventos gerados registrando nos reporters cadastrados. Além disso, testar como diferentes erros e diferentes estruturas arquiteturais influenciam na capacidade de identificar e corrigir erros durante a execução.

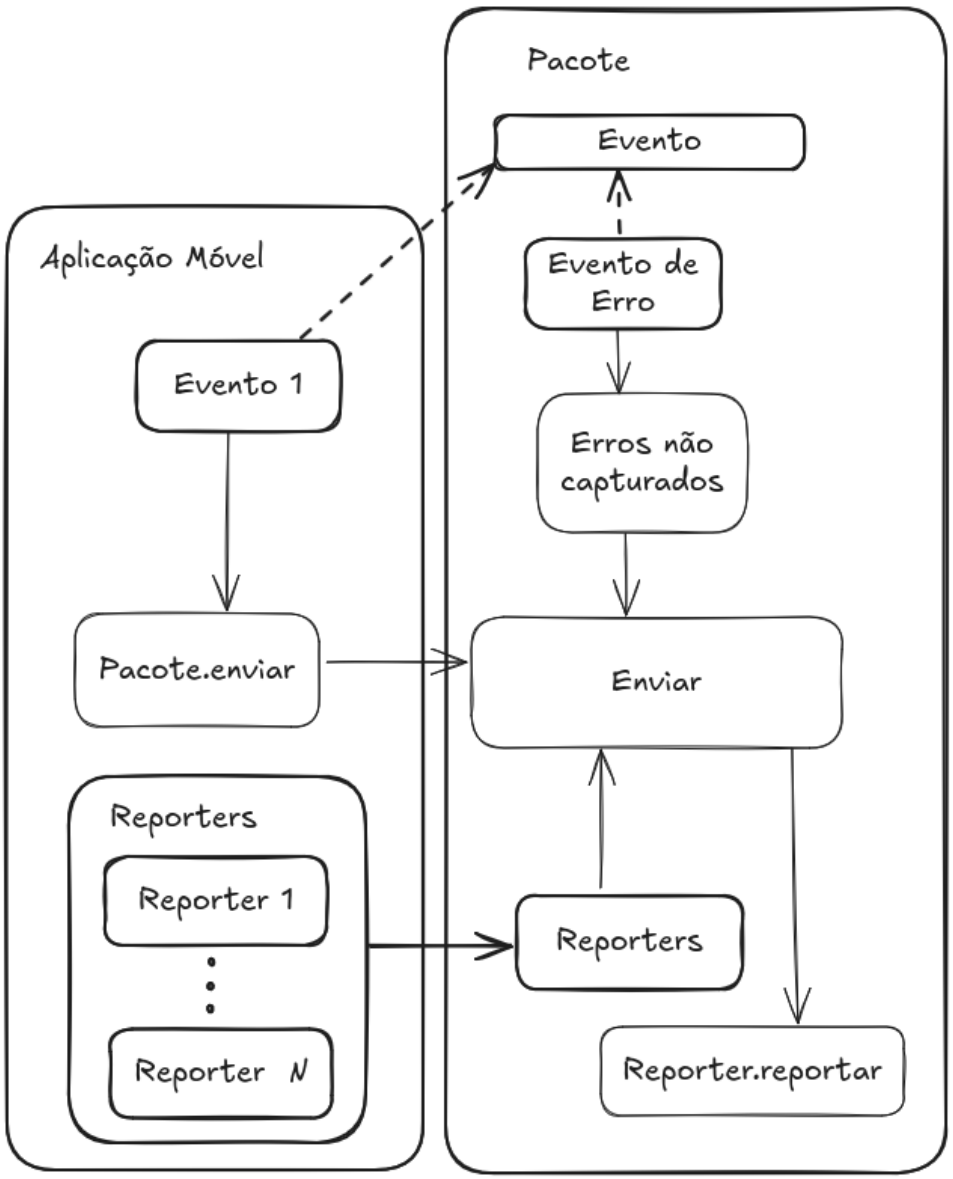
## 4.1 MÓDULO DE GERENCIAMENTO

O módulo de gerenciamento será responsável pela captura de enventos de erros, tais como exceções e de erros de execução, e eventos definidos pelo desenvolvedor. Ao efetuar uma captura o módulo de gerenciamento envia o evento capturado para todos os reporters préviamente inicializados no módulo de gerenciamento, salvando as informações disponibilizadas pela aplicação e programadas pelo desenvolvedor, tais como métricas, contexto adicional e dados úteis para diagnóstico.

Apesar de ser testado apenas em alguns casos de integrações e tipos de eventos, o módulo deve conseguir ser extendido de forma a facilitar a criação personalizada de outros tipos de eventos, e outros reporters para efetuar o registro dos eventos.

Apesar de cada protótipo ter sua própria estrutura e organização, o objetivo é fazer um módulo de gerenciamento que tenha o mesmo comportamento, ao menos para uso básico, entre as diferentes aplicações de teste, como representado na figura a seguir.

Figura 21 – Idealização do pacote




Fonte: Figura elaborada pelo autor

## 4.2 GERENCIAMENTO DA APLICAÇÃO

Nesta seção são apresentados exemplos mínimos práticos que ilustram o comportamento de uma aplicação móvel com e sem um mecanismo de gerenciamento de erros. Para isso, são demonstrados trechos de código, diagramas de sequência e telas resultantes da execução, permitindo comparar de forma clara os impactos da ausência e da presença de gerenciamento no funcionamento da aplicação.

Os exemplos a seguir apresentam um aplicativo sem qualquer mecanismo de gerenciamento de erros e outro utilizando uma camada adicional de gerenciamento.

No primeiro exemplo, a inicialização tradicional de um aplicativo Flutter, realizada por

meio da chamada direta ao método runApp.

Algoritmo 1 – Exemplo de inicialização normal

```
1 void main() => runApp(const MyApp());
```

No segundo exemplo, é apresentado um fluxo de inicialização que utiliza o pacote Acta para incorporar o gerenciamento de erros. Nesse caso, o aplicativo é configurado para registrar eventos, coletar informações relevantes do dispositivo e exibir um alerta visual sempre que um erro for capturado.

O processo ocorre da seguinte forma: um reporter é inicializado, informações adicionais sobre o dispositivo são coletadas, uma função de callback (onCaptured) é definida para apresentação de um popup, e, por fim, a função appRunner executa a inicialização normal do aplicativo.

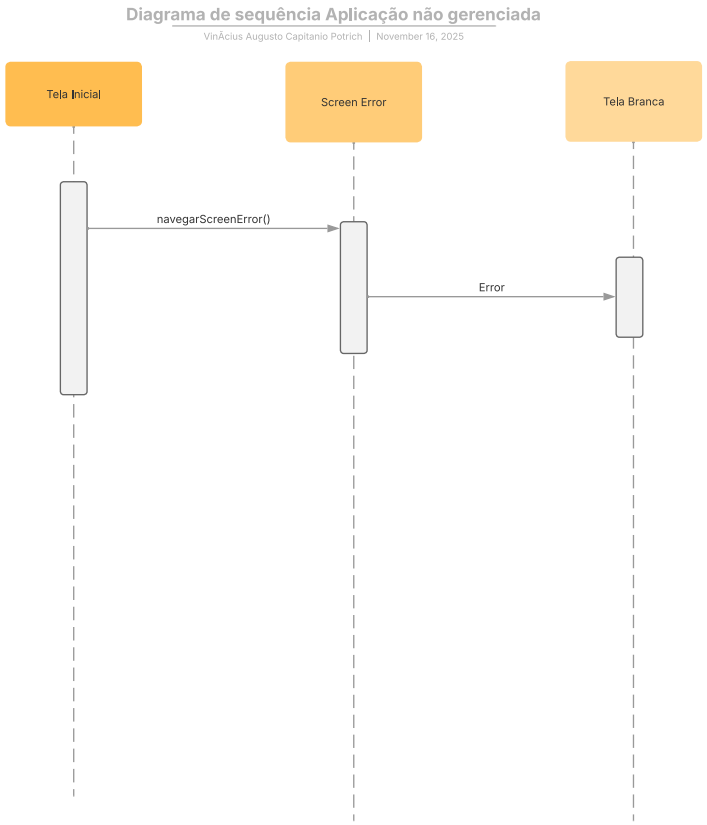
Algoritmo 2 – Exemplo de inicialização usando gerenciamento

```
1 void main() async {
2   ActaJournal.initialize(
3     reporters: [ConsoleReporter()],
4     initialContext: await InfoService.collectAsJson(), //Helper info
          device
5     onCaptured: showPopup, //Handler do pos evento
6     appRunner: () async => runnable(), //chama main pode ser async
7   );
8 }
9 void runnable() => runApp(const MyApp());
```

A seguir, são apresentados dois diagramas de sequência que ilustram o comportamento da versão sem gerenciamento de erros e a versão com gerenciamento implementado.

O primeiro diagrama representa o fluxo da aplicação não gerenciada. Nele, após o usuário navegar da tela inicial para a tela Screen Error, o sistema tenta processar um componente com falha de layout. Como não há mecanismos de captura ou tratamento do erro, a aplicação entra em estado inconsistente, resultando na exibição de uma tela completamente branca. Esse comportamento impede que o usuário retorne ao fluxo normal, tornando a aplicação inutilizável após a ocorrência do erro.

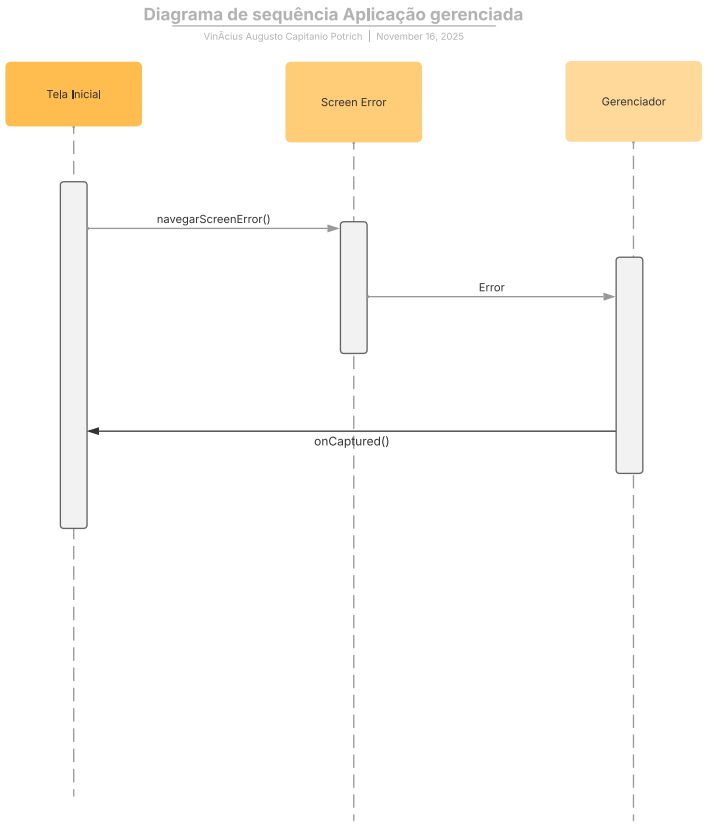
Figura 22 – Diagrama de sequência da aplicação não gerenciada




Fonte: Figura elaborada pelo autor

O segundo diagrama apresenta a sequência de execução da aplicação com gerenciamento de erros. Nesse caso, mesmo após a navegação para a tela Screen Error e a consequente ocorrência da falha de layout, o erro é interceptado pelo mecanismo de gerenciamento, que redireciona a aplicação de volta para a tela inicial. Dessa forma, a aplicação permanece funcional, garantindo continuidade de uso mesmo após a ocorrência do erro.

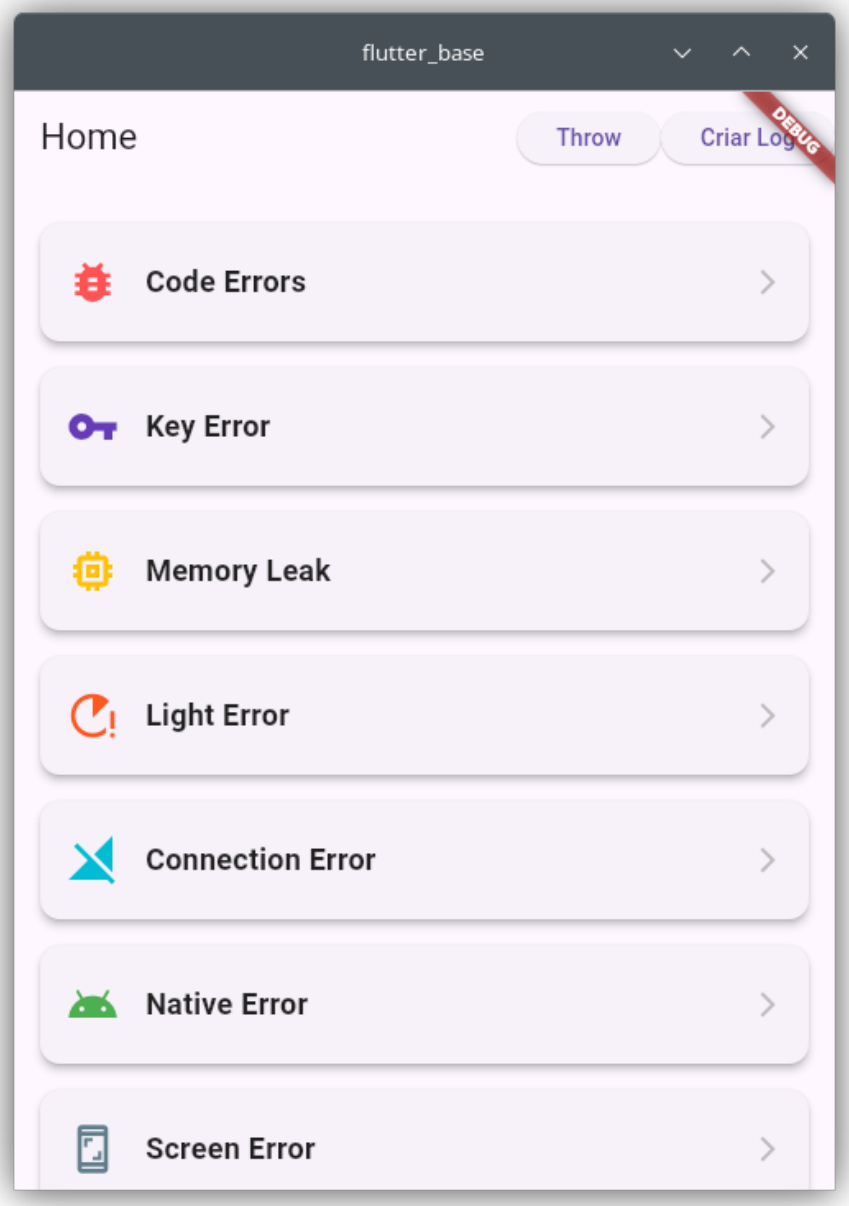
Figura 23 – Diagrama de sequência da aplicação gerenciada




Fonte: Figura elaborada pelo autor

A seguir, são apresentadas as telas que representam os dois cenários de execução. A primeira imagem mostra a tela inicial da aplicação de testes. Até esse ponto, o comportamento é idêntico nos dois casos, uma vez que nenhum erro foi acionado.

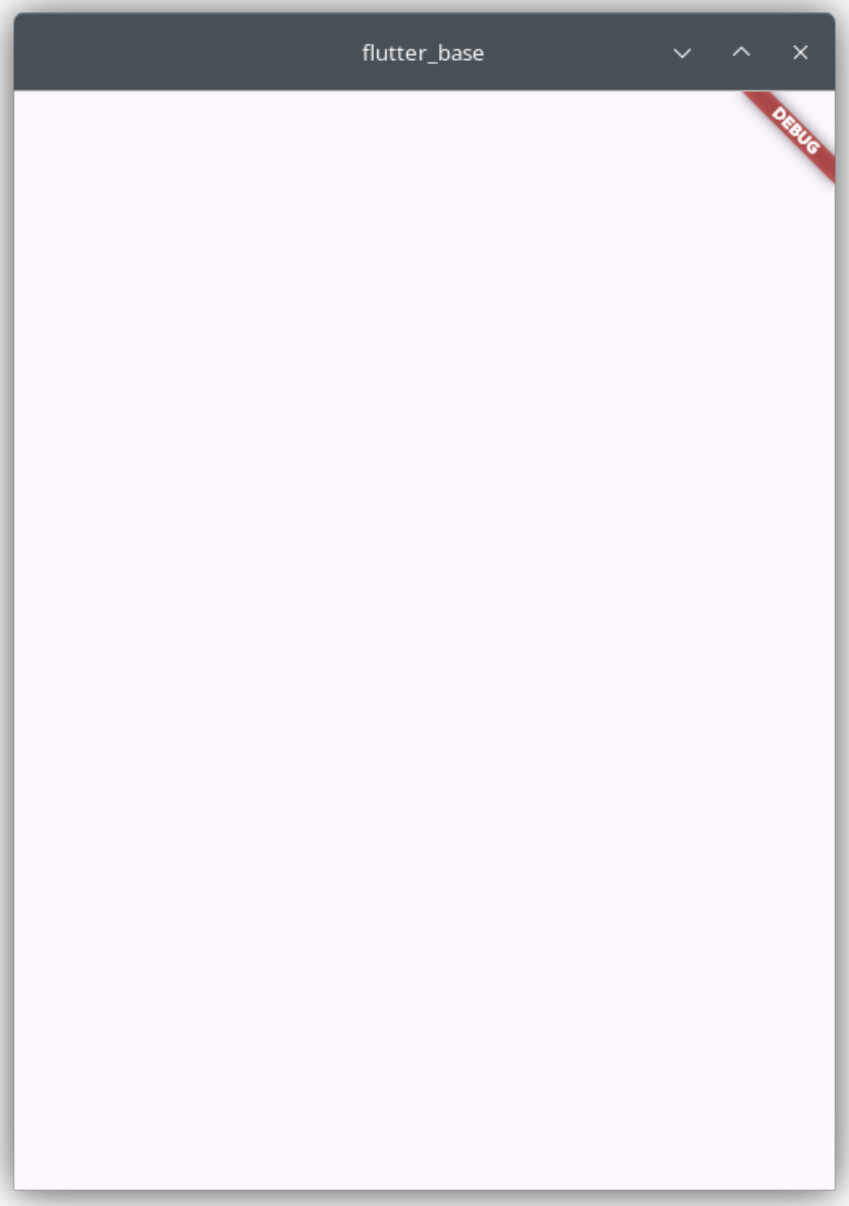
Figura 24 – Tela inicial




Fonte: Figura elaborada pelo autor

A segunda imagem representa a execução da aplicação sem gerenciamento de erros. Ao selecionar a opção Screen Error, a aplicação tenta renderizar um componente com falha de layout. Como o erro não é tratado, toda a interface é substituída por uma tela completamente branca, impossibilitando a continuidade do uso.

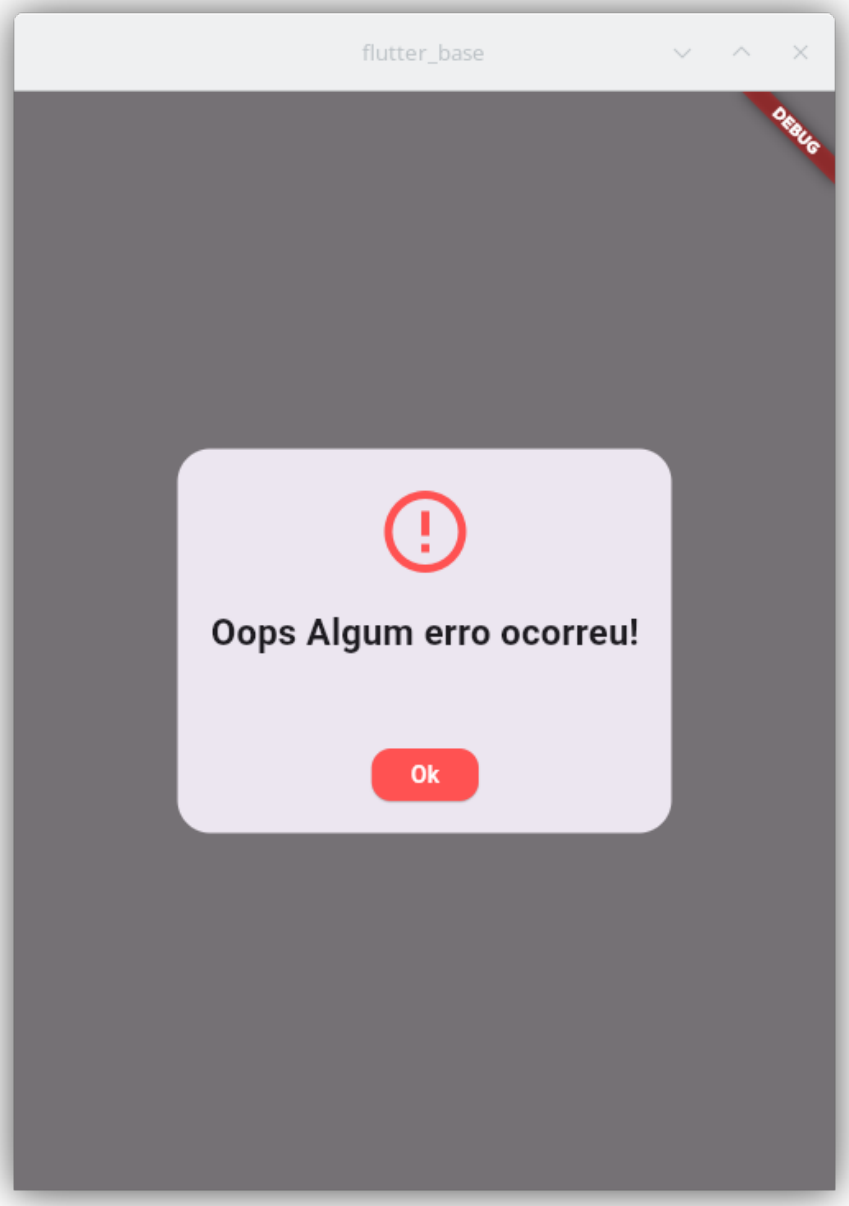
Figura 25 – Tela de erro




Fonte: Figura elaborada pelo autor

A terceira imagem apresenta o comportamento da aplicação com gerenciamento de erros. Da mesma forma que no exemplo anterior, ao selecionar Screen Error ocorre uma falha de layout. No entanto, o sistema de gerenciamento captura o erro e exibe um popup informando o problema ocorrido. Ao confirmar a mensagem, a aplicação retorna automaticamente para a tela inicial, permitindo que o usuário continue a usando o aplicativo.

Figura 26 – Popup notificando o error




Fonte: Figura elaborada pelo autor

## 4.3 AMBIENTE DE TESTES

Para avaliar o comportamento do módulo de gerenciamento proposto, foi definido um ambiente de testes controlado, composto pelos quatro protótipos de aplicações apresentados anteriormente. A solução será testada baseado na criação de quatro testes de protótipo de aplicativos em Flutter, desenvolvidos a partir de modelos comumente utilizados pela comunidade, os modelos consistem Flutter básico, consiste em um projeto sem definições estruturais, modelos que estipulam algumas camadas préviamente tais como GetX e Very Good Ventures, e um

modelo baseado em Clean Architecture.

Os erros selecionados representam falhas comuns em aplicações Flutter, frequentemente responsáveis por travamentos, telas brancas ou interrupção completa da execução. Cada teste foi configurado manualmente dentro dos protótipos e executado tanto com quanto sem gerenciamento. Os principais tipos de erros avaliados foram:

Erros de Layout: Ocorrências durante a renderização de widgets, como limites estourados, tamanhos inválidos ou falhas internas na árvore de renderização. Exceções Não Tratadas: Exceções lançadas durante a execução que não estão envolvidas em blocos try/catch, atingindo os handlers globais do Flutter. Erros de Canal de Método (MethodChannel): Falhas ao tentar comunicar com o código nativo ou ao invocar métodos inexistentes. Erros de Chaves (Key Errors): Inconsistências relacionadas ao gerenciamento de chaves na árvore de widgets, como duplicidade ou perda de estado. Erros de Conexão Externa: Falhas provocadas por tentativas de acessar serviços inexistentes, desconectados ou com URL inválida.

Esses testes permitirão visualizar o comportamento da aplicação ao sofrer um erro, e a eficiência do módulo de gerenciamento para preservar a execução da aplicação mesmo após ocorrer falhas e erros. Podendo observar cara da erro se o módulo capturou o evento, se o comportamento da aplicação permaneceu estável após a falha, e se os dados relevantes foram encaminhados ao reporter configurado.

## 4.4 VISUALIZAÇÃO

Para facilitar a análise dos eventos capturados pelo módulo de gerenciamento, foi adotada uma solução de visualização baseada no ElasticSearch e no Kibana. O objetivo dessa etapa é permitir que os erros registrados pelos reporters possam ser consultados de forma organizada, estruturada e acessível durante os testes.

O ElasticSearch é um mecanismo de busca e indexação de dados orientado a documentos, amplamente utilizado para armazenar e consultar grandes volumes de informações em tempo real. Por utilizar documentos JSON e oferecer consultas flexíveis, ele se torna adequado para registrar eventos de erro capturados pelo módulo, incluindo mensagens, contextos adicionais e metadados do dispositivo.

O Kibana, por sua vez, é a interface de visualização que opera diretamente sobre os dados armazenados no ElasticSearch. Por meio dele, é possível criar dashboards locais com gráficos, tabelas e filtros dinâmicos que facilitam a inspeção dos eventos registrados.

A comunicação entre as ferramentas ocorre de forma direta o módulo de gerenciamento envia cada evento capturado para um reporter responsável por registrar os dados no ElasticSearch, o ElasticSearch armazena e indexa esses eventos, o Kibana acessa essas informações e as apresenta visualmente no dashboard configurado.

Com essa abordagem, torna-se possível não apenas verificar se os erros foram devidamente capturados, mas também analisar os padrões e frequências das falhas durante os testes. Embora ElasticSearch e Kibana sejam ferramentas robustas e amplamente utilizadas em ambientes de observabilidade de larga escala, neste trabalho elas serão utilizadas de forma simplificada, servindo essencialmente como suporte à visualização dos erros coletados.

## 4.5 DISPONIBILIDADE

Caso o módulo de gerenciamento seja visto como útil no auxílio ao gerenciamento de aplicações móveis, poderá ser disponibilizado, de forma oficial, para a comunidade como um facilitador para o processo de fetuar o gerenciamento de falha de aplicativos móveis.