PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Gerenciamento de Estado em Flutter: Estratégias, Desempenho e Práticas*

State Management in Flutter: Strategies, Performance and Practices

Gabriel Hirano Gomes

Resumo

Este artigo aborda a complexidade do dilema na escolha do gerenciador de estado ideal para aplicativos Flutter, considerando critérios como complexidade, escalabilidade, testabilidade, suporte, latência, consumo de memória, quantidade de código boilerplate, código gerado e persistência de estado. O cerne do problema reside na diversidade de estratégias disponíveis, gerando incertezas para os desenvolvedores. O estudo visa oferecer orientações práticas para auxiliar os desenvolvedores na tomada de decisão sobre o gerenciamento de estado, explorando estratégias, comparando desempenho e eficiência em diferentes cenários, e apresentando desafios comuns e melhores práticas para garantir a manutenibilidade e escalabilidade dos sistemas.

Palavras-chave: Flutter, gerenciamento de estado, escalabilidade, desempenho, práticas recomendadas.

1 INTRODUÇÃO

O desenvolvimento de aplicativos móveis se tornou uma parte essencial da vida cotidiana, impulsionando inovações em diversas áreas, desde entretenimento até produtividade. À medida que a demanda por aplicativos móveis continua a crescer, a necessidade de criar aplicativos de alta qualidade e eficientes torna-se cada vez mais crucial. Uma das áreas críticas no desenvolvimento de aplicativos móveis é o gerenciamento de estado, que desempenha um papel fundamental na criação de aplicativos responsivos e eficientes.

O gerenciamento de estado refere-se à maneira como os aplicativos rastreiam, atualizam e compartilham informações em tempo real entre os diferentes componentes da interface do usuário. Uma implementação eficaz do gerenciamento de estado é essencial para garantir que os aplicativos sejam capazes de responder de forma rápida e precisa às interações do usuário e aos eventos do sistema.

No ecossistema do Flutter, uma estrutura de desenvolvimento de código aberto amplamente adotada para aplicativos móveis multiplataforma, os desenvolvedores se deparam com uma ampla variedade de opções para o gerenciamento de estado. Cada opção possui suas características singulares, tornando a escolha do gerenciador de estado adequado uma tarefa desafiadora e crucial. A decisão de qual abordagem de gerenciamento de estado utilizar impacta diretamente na manutenibilidade do sistema e no desempenho à medida que o aplicativo cresce em complexidade.

O cerne do dilema reside na diversidade de estratégias disponíveis e na falta de uma resposta única e definitiva para a escolha do gerenciador de estado ideal, a escolha do gerenciador de estado certo depende de uma série de fatores, incluindo a natureza do aplicativo, a complexidade dos fluxos de dados e os objetivos de escalabilidade. Essa diversidade, oferece flexibilidade, mas também gera incertezas para os desenvolvedores.

A presente pesquisa visa abordar a questão complexa e crítica mencionada, viabilizando orientações práticas para ajudar os desenvolvedores na tomada de decisão sobre o gerenciamento de estado. Explora-se estratégias de gerenciamento de estado, comparando seu desempenho e eficiência em diferentes cenários, além disso, argumenta-se os desafios comuns enfrentados pelos desenvolvedores e as melhores práticas para garantir a manutenibilidade e escalabilidade dos sistemas.

1.1 Problema

No contexto do Flutter, uma estrutura de desenvolvimento de código aberto amplamente adotada para aplicativos móveis multiplataforma, os desenvolvedores enfrentam um desafio considerável: a seleção do gerenciador de estado mais apropriado para seus projetos. O problema central se encontra na diversidade de opções disponíveis, cada uma apresentando suas próprias características e vantagens únicas. Os desenvolvedores que ingressam no ecossistema do Flutter frequentemente se debatem com a seguinte questão complexa: "Como escolher o gerenciador de estado que melhor se adapte às necessidades

específicas de um projeto Flutter?" Essa indagação se torna mais complexa devido à ausência de uma resposta única e definitiva. A escolha do gerenciador de estado ideal é uma decisão intrinsecamente influenciada por uma série de fatores, incluindo a natureza do aplicativo, a complexidade dos fluxos de dados, a dinâmica da equipe de desenvolvimento e considerações de escalabilidade.

1.2 Objetivos

O presente artigo tem como objetivo auxiliar desenvolvedores na tomada de decisão de qual gerenciador de estado usar, considerando diferentes contextos de aplicativos.

1.2.1 Objetivo Específico

Explorar diferentes estratégias de gerenciamento de estado, comparar seu desempenho e proporcionar orientações práticas.

1.3 Justificativa

A realização desta pesquisa reside no contexto de crescimento da adoção do Flutter. À medida que os aplicativos desenvolvidos nesta plataforma se tornam mais complexos e interativos, a seleção de um gerenciador de estado apropriado assume uma importância ainda maior. Os desenvolvedores estão agora encarando o desafio de lidar com fluxos de dados intrincados e interfaces de usuário altamente reativas.

Este estudo visa preencher uma lacuna crítica de conhecimento, fornecendo diretrizes práticas e informações aprofundadas que podem beneficiar significativamente a comunidade de desenvolvedores Flutter. Tais orientações permitirão que estes tomem decisões embasadas na seleção de uma abordagem de gerenciamento de estado que se alinhe de maneira otimizada aos requisitos de seus projetos específicos. A disseminação de informações e recomendações práticas é fundamental para auxiliar os desenvolvedores na prevenção de desafios comuns, na melhoria da qualidade de seus aplicativos e na aceleração do processo de desenvolvimento, garantindo, assim, que os aplicativos Flutter atendam às expectativas dos usuários e sejam sustentáveis a longo prazo.

2 REFERENCIAL TEÓRICO

A finalidade desta seção consiste em introduzir o leitor no universo da construção de aplicativos por meio do Flutter, elucidando a intrincada compilação Dart, elemento que confere ao framework sua versatilidade multiplataforma e intensifica a agilidade no processo de desenvolvimento. Além disso, busca-se exemplificar conceitos fundamentais do framework, completando na exposição do cerne deste trabalho, que concentra-se nos gerenciadores de estado.

2.1 Dart

É uma linguagem de programação apresentada na conferência GOTO em 2011, lançada em 2013 e mantida pela Google, foi inicialmente proposta com o intuito de substituir o Javascript para a web.

Surgiu com a premissa de utilizar uma Virtual Machine (VM) no Google chrome, contudo, essa integração ocasionou um déficit de otimização e um retardo na depuração. Posteriormente houve um levantamento de comentários dos usuários, com isso, notou-se um descontentamento relativo à integração da VM na web, então na versão 1.9 o foco foi direcionado para a compilação do Dart para Javascript, dessa forma, foi decidido não mais integrar a VM (DART NEWS E UPDATES, 2015).

Em 2018 foi lançado o Dart 2.0, um reboot da linguagem, otimizado para o desenvolvimento client-side para Web e dispositivos móveis, conforme o anúncio feito pela equipe de desenvolvimento do Dart, publicado na revista Medium (Thorhauge, 2018), limparam a sintaxe e reconstruiram grande parte da cadeia de ferramentas do desenvolvedor do zero.

É uma linguagem orientada a objeto, multiparadigma (DART, 2023), o código fonte do dart demonstra que o uso do mesmo é principalmente com o Flutter e não sozinho, porém, também pode ser usado em outros frameworks como, Angular Dart e Vue Dart.

Tem como suas principais características:

- **Declaração explícita:** apesar de ser fortemente tipada a linguagem é capaz de inferir um tipo a uma variável;
- **Segurança nula:** possibilita controlar se o tipo da variável permite ser anulada, caso a mesma possa ser atribuída null, essa característica impõe a segurança nula de som, na qual altera possíveis erros de tempo de execução em erros de análise de tempo de edição.

2.1.1 Compilação

Atualmente o Dart possui diferentes modos de compilação a depender da plataforma, estes sendo:

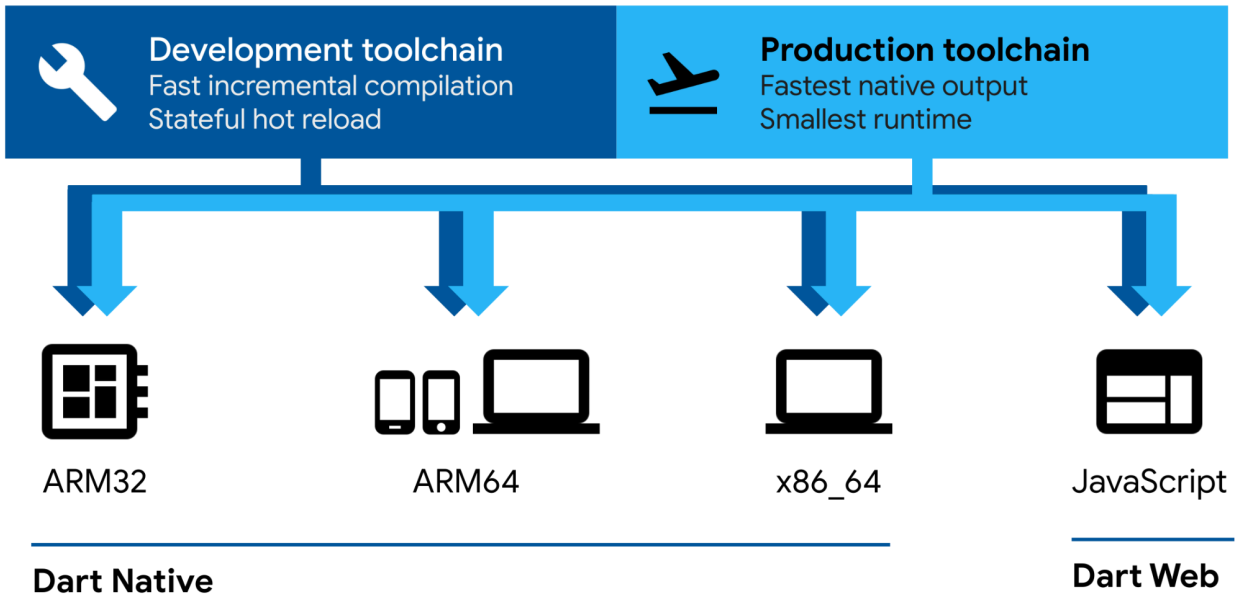
- **Plataforma nativa:** para aplicativos direcionado ao desktop e dispositivos móveis, nesse possui uma VM Dart com compilação just-in-time (JIT), e um compilador ahead-of-time (AOT);
- **Plataforma web:** para aplicações web, o compilador traduz Dart para Javascript, sendo assim, facilmente interpretado pelos navegadores.

Durante o desenvolvimento o compilador usado é o JIT, em que usa a VM Dart, com recompilação incremental, coleções de métricas em tempo real e suporte avançado à depuração (DART, 2023). A recompilação incremental habilita o hot reload, que é capacidade da VM se comunicar com o código e após alguma alteração no mesmo ele irá refletir essa mudança no

aplicativo sem precisar recompilar a aplicação, diferente do swift, no qual, caso precise alterar algo na aplicação para conseguir ver a mudança é necessário recompilar a aplicação.

Quando o aplicativo é publicado em produção o compilador ahead-of-time (AOT) é utilizado, no qual, permite ser compilado para código de máquina x64 ou ARM nativo, o aplicativo compilado pelo AOT é livre do uso da VM Dart, sendo assim, é iniciado com consistência e curto tempo de inicialização. A Figura 1 ilustra sobre o conjunto de ferramentas para quando o aplicativo estiver em desenvolvimento ou produção

Figura 1 - Plataformas Dart



Fonte: Documentação Dart

2.2 Flutter

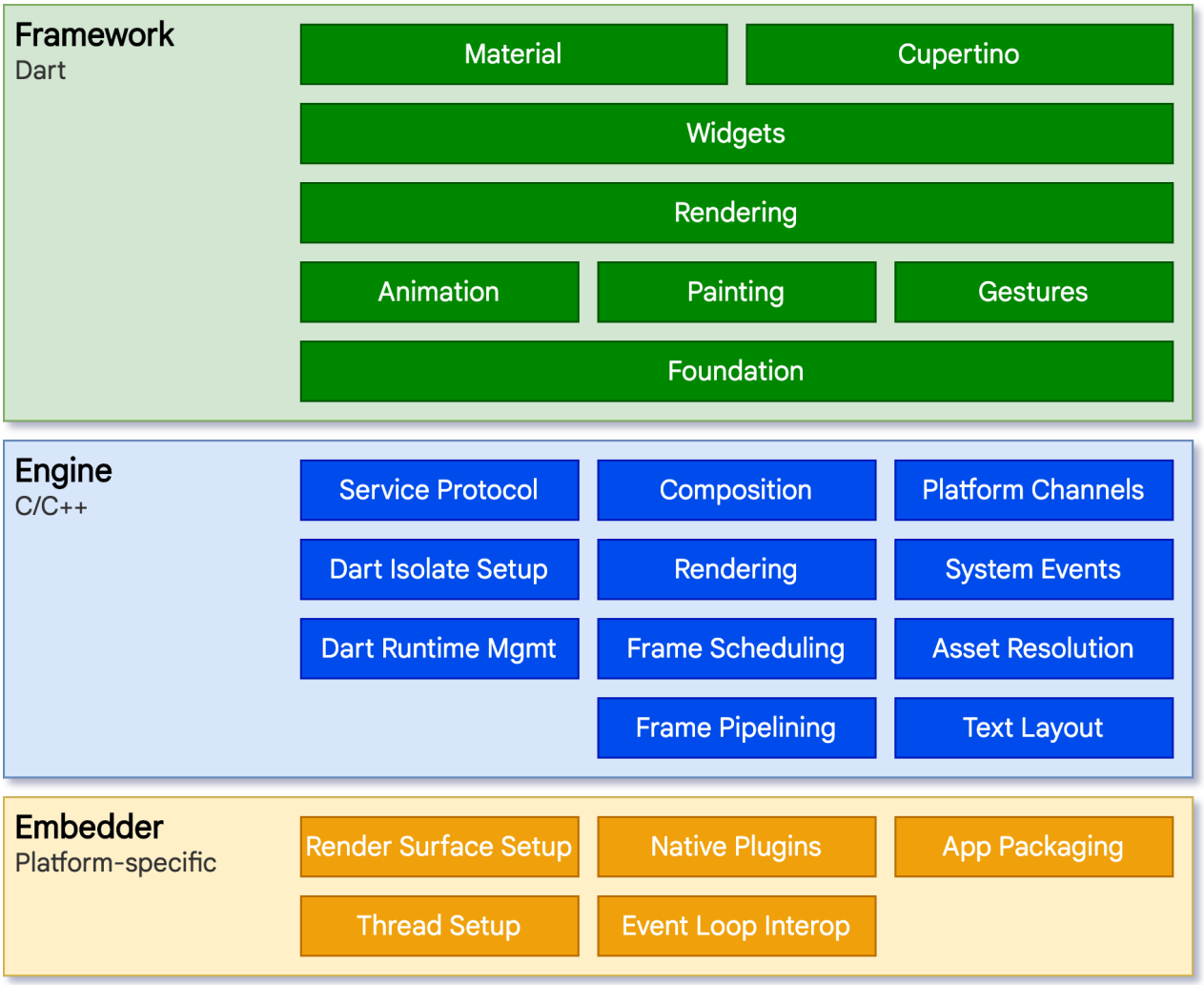
O Flutter constitui-se como um software development kit (SDK) desenvolvido e mantido pela Google, sendo implementado em linguagens de programação C e C + + (FLUTTER, 2023). Nota-se que a Google, já detentora de um SDK denominado Android para dispositivos móveis, direcionou seu enfoque para o desenvolvimento multiplataforma, conforme destacado por Rony Amadeo na revista Ars Technica em 2018, após o anúncio do Beta do Flutter.

Conforme a descrição de tradução direta retirada da documentação oficial do Flutter, o framework é um kit de ferramentas de interface do usuário de plataforma cruzada projetado para permitir a reutilização de código em sistemas operacionais como iOS e Android, ao mesmo tempo em que permite aplicativos para interface direta com os serviços da plataforma subjacente. O objetivo é permitir que os desenvolvedores entreguem aplicativos de alto desempenho que pareçam naturais em diferentes plataformas, abraçando diferenças onde elas existem enquanto compartilham como muito código possível (FLUTTER, 2023).

O design do Flutter, de acordo com sua documentação oficial, é caracterizado por uma abordagem em camadas que permite a extensibilidade do sistema, ele se manifesta como uma série de bibliotecas independentes, cada uma dependendo da camada subjacente. Essa arquitetura é

projetada com a premissa de que nenhuma camada deve ter acesso privilegiado à camada abaixo, e cada componente do nível estrutural é concebido como opcional e substituível, essa estrutura modular proporciona flexibilidade e adaptabilidade ao ambiente de desenvolvimento (FLUTTER, 2023). A Figura 2 exemplifica essa abordagem de camadas.

Figura 2 - Arquitetura do Flutter



Fonte: Documentação Flutter

O cerne do Flutter reside em seu motor, primariamente implementado em C + + e oferece suporte aos primitivos necessários para o funcionamento de todas as aplicações Flutter. O motor é responsável por rasterizar cenas, convertendo elementos gráficos em pixels na tela, também engloba a implementação de baixo nível da API principal do Flutter, incluindo gráficos, layout de texto, operações de entrada/saída de arquivos e rede, suporte à acessibilidade, arquitetura de plug-ins (FLUTTER, 2023).

2.2.1 Widgets

No Flutter, um widget é um conceito fundamental que representa um elemento da interface do usuário (UI) (FLUTTER, 2023). Widgets são os blocos de construção básicos para criar interfaces de usuário no Flutter, e tudo no Flutter é um widget, desde o layout até os elementos

visuais, como botões e imagens.

No artigo "Entendendo os tipos de Widgets do Flutter", Fabiano Santana (2019) discute os diferentes tipos de widgets no framework Flutter, exemplificando a descrição do que é um widget: "Widgets nada mais são do que componentes que podem ou não possuir um estado e são usados para moldar nossa interface do usuário (UI)" (SANTANA, 2019)

2.2.2 Gerenciador de estado

Ao migrar de estruturas imperativas, como Android SDK ou iOS UIKit, para o ambiente do Flutter, é essencial adotar uma nova perspectiva no desenvolvimento de aplicativos. Operando sob uma abordagem declarativa, o Flutter introduz nuances únicas que demandam uma reavaliação das práticas convencionais.

Diferentemente de estruturas tradicionais, no Flutter, reconstruir partes específicas da interface do usuário do zero é incentivado pela sua agilidade, mesmo a cada quadro, se necessário. Essa flexibilidade marca uma distinção crucial em relação a outras estruturas e desafia preconcepções comuns (FLUTTER, 2023). A Figura 3 ilustra essa abordagem.

Figura 3 - Pensar declarativamente

Fonte: Documentação Flutter

Adotando uma abordagem declarativa, o Flutter constrói a interface de usuário para refletir o estado atual do aplicativo, oferecendo uma visão eficiente e intuitiva do desenvolvimento de interfaces, priorizando a clareza na expressão do estado (FLUTTER, 2023). Neste contexto, a transição para essa abordagem requer não apenas familiaridade com a sintaxe e as ferramentas, mas também uma mudança fundamental na mentalidade do desenvolvedor para abraçar o paradigma declarativo e suas implicações no processo de construção de interfaces reativas e eficazes.

2.3 Abordagens comuns para gerenciamento de estado

O ecossistema oferece uma variedade de abordagens para o gerenciamento de estado, cada uma com suas características distintas. Antes de nos aprofundarmos em cada uma delas, é necessário ter uma visão geral para compreender o cenário de escolhas disponíveis. Abaixo estão algumas das abordagens comuns de gerenciadores de estado, conforme apresentados na documentação oficial do Flutter:

- **Provider:** É uma biblioteca que simplifica o gerenciamento de estado, fornecendo

uma forma fácil de compartilhar e acessar dados em toda a árvore de widgets;

- **Riverpod:**É uma extensão do Provider, o Riverpod visa aprimorar a legibilidade e a escalabilidade do código, oferecendo uma solução mais robusta e declarativa para o gerenciamento de estado;
- **setState:** É um método padrão do Flutter, o setState é utilizado para notificar o framework sobre mudanças no estado de um widget, desencadeando uma reconstrução da interface de usuário;
- **InheritedWidget & InheritedModel:** Essas classes fornecem uma maneira eficiente de propagar informações para baixo na árvore de widgets, sendo úteis para compartilhar dados que não mudam frequentemente;
- **Redux:** Inspirado pela arquitetura Flux, o Redux é uma abordagem de gerenciamento de estado que centraliza o estado da aplicação em um único local, facilitando a previsibilidade e testabilidade;
- **BLoC / Rx:** BLoC (Business Logic Component) é uma abordagem que utiliza Streams do Dart para gerenciar o estado da aplicação, oferecendo uma separação clara entre a lógica de negócios e a interface de usuário;
- **MobX:** Baseado no paradigma de programação reativa, o MobX simplifica a gestão do estado ao permitir que as mudanças no estado automaticamente acionem atualizações na interface de usuário;
- **GetX:** É uma biblioteca que oferece um ecossistema completo para o Flutter, incluindo gerenciamento de estado, navegação, roteamento e até mesmo injeção de dependência.

3 PESQUISAS ANTERIORES E CONTRIBUIÇÕES RELEVANTES

A dissertação de Dmitrii Slepnev, intitulada "State Management Approaches in Flutter", apresenta uma análise abrangente das abordagens de gerenciamento de estado em Flutter, fornecendo um conjunto de critérios de tomada de decisão para selecionar a abordagem mais adequada para diferentes casos de uso. Slepnev (2020) utilizou métodos de pesquisa quantitativa e qualitativa para analisar a popularidade das abordagens de gerenciamento de estado em Flutter, bem como para categorizá-las e compará-las com base em critérios predefinidos. Essa metodologia pode ser útil para o presente trabalho, pois permite uma análise abrangente das abordagens de gerenciamento de estado em Flutter, fornecendo uma base sólida para a seleção da abordagem mais adequada para diferentes casos de uso.

O trabalho de Slepnev resultou na compilação de uma lista das abordagens mais populares de gerenciamento de estado em Flutter, categorizadas e comparadas com base em critérios predefinidos. Além disso, o autor apresentou uma tabela de comparação das abordagens estudadas, fornecendo um conjunto de critérios de tomada de decisão para selecionar a abordagem de gerenciamento de estado mais adequada para diferentes casos de uso. Esses resultados podem ser úteis para o presente trabalho, pois fornecem uma base sólida para a seleção da abordagem de gerenciamento de estado mais adequada para diferentes casos de uso em Flutter.

De acordo com Slepnev (2020, p. 9), a metodologia utilizada em seu trabalho envolveu a análise de materiais científicos, como livros, artigos, relatórios e conferências, além de pesquisas quantitativas e qualitativas. O autor também realizou uma pesquisa sobre a popularidade das abordagens de gerenciamento de estado em Flutter, como mencionado por Slepnev (2020, p. 92), e apresentou uma lista das abordagens mais populares de gerenciamento de estado em Flutter.

4 METODOLOGIA

Esta seção delineia a estratégia metodológica empregada para conduzir a pesquisa e alcançar os objetivos propostos. O desenvolvimento desta pesquisa seguirá uma abordagem exploratória descritiva, fundamentada em análises documentais e práticas.

4.1 Seleção de Gerenciadores de Estado

A revisão da literatura sobre abordagens de gerenciamento de estado no ecossistema Flutter foi conduzida mediante análise direta das propostas disponibilizadas no código fonte da framework, mais especificamente na seção de Lista de abordagens de gerenciamento de estado. Dentre as alternativas apresentadas, foram identificadas as seguintes: Provider, Riverpod, setState, InheritedWidget & InheritedModel, Redux, Fish-Redux, BLoC / Rx, MobX, Flutter Commands Binder, GetX, states_rebuilder, Triple Pattern (Segmented State Pattern), solidart, flutter_reactive_value.

Para refinar a seleção e proporcionar um embasamento mais robusto, foram consideradas as informações apresentadas no artigo "Bibliotecas de gerenciamento de estado Flutter mais populares em 2023", escrito por Martin Jablečník na revista Dev To. Este autor destacou nove abordagens, classificando a popularidade com base no número de estrelas nos repositórios oficiais no GitHub e na quantidade de curtidas nas páginas das bibliotecas no Pub.dev. As abordagens mencionadas são: BLoC, GetX, Provider, Riverpod, MobX, Redux, states_rebuilder, Creator e Triple.

Cruzando as informações, identificou-se um conjunto de abordagens de gerenciamento de estado que despertam considerável engajamento na comunidade. Das bibliotecas analisadas, e com o intuito de abranger a diversidade de soluções disponíveis, optou-se por selecionar as seguintes: BLoC, GetX, Provider, Riverpod, MobX, Redux, InheritedWidget & InheritedModel. Adicionalmente, para incorporar uma solução nativa, foi decidido incluir o ValueNotifier. Esta seleção foi feita com base não apenas nos critérios de popularidade, mas também na representatividade e na diversidade que essas abordagens oferecem para uma análise abrangente do cenário de gerenciamento de estado no ecossistema Flutter.

4.2 Critérios de Avaliação

Para atingir o propósito de comparar o desempenho dos gerenciadores de estado e oferecer orientações práticas, é necessário estabelecer critérios de avaliação que permitam uma análise aprofundada, possibilitando a tomada de decisões informadas em diferentes contextos de

aplicativos. Os critérios delineados para esta avaliação são os seguintes:

- **Complexidade:** Este critério visa avaliar a complexidade intrínseca dos gerenciadores de estado, considerando a simplicidade ou complexidade de sua implementação. Será definido como baixa, médio ou alta;
- **Escalabilidade:** A escalabilidade é examinada no sentido de verificar a capacidade do gerenciador de estado lidar eficientemente com o crescimento e a complexidade do aplicativo. Será definida como ruim, boa ou ótima;
- **Testabilidade:** Este critério foca na facilidade de realização de testes automatizados no código que incorpora o gerenciador de estado. Definida como fácil, difícil;
- **Suporte:** A análise do suporte engloba a verificação da existência de recursos de apoio e da participação ativa da comunidade de desenvolvedores. Definida como bom ou ruim;
- **Latência:** Avaliado considerando a latência, que representa o tempo de resposta do gerenciador em operações específicas. Definido como baixa ou alta;
- **Consumo de memória:** Avaliado indicando a eficiência do gerenciador em ambientes com recursos limitados. Definido baixo ou alto;
- **Quantidade de código boilerplate:** Refere-se à avaliação da quantidade de código repetitivo e não essencial (boilerplate) necessária para a integração do gerenciador de estado. Definido como baixa ou alta;
- **Código gerado:** Este critério examina a qualidade e eficácia do código produzido pelo gerenciador de estado. Definido como sim ou não;
- **Persistência de estado:** Avalia a capacidade do gerenciador de estado em manter e recuperar o estado da aplicação, mesmo em cenários de reinicialização ou mudança de contexto. Definido como permite ou não permite.

4.3 Implementação e Comparação

4.4 Contextualização dos Resultados

4.5 Orientações Práticas

5 CONCLUSÃO

CRONOGRAMA

Atividade	Janeiro	Fevereiro	Março	Abril	Maio	Junho

Implementação e Comparação	x	x	x			
Contextualização dos Resultados				x		
Orientações Práticas					x	x
Revisão da escrita	x	x	x	x	x	x

REFERÊNCIAS

AMADEO, Ron. Google starts a push for cross-platform app development with Flutter SDK. 2018. Disponível em: https://arstechnica.com/gadgets/2018/02/google-starts-a-push-for-cross-platform-app-development-with-flutter-sdk

D'ANTONIO, Olavo. Gerenciamento de Estado em Flutter. 2023. Disponivel em: https://www.dio.me/articles/gerenciamento-de-estado-em-flutter

GOOGLE LCC. Flutter documentation. Disponível em: https://docs.flutter.dev/data-and-backend/state-mgmt/options#flutter_reactive_value

HOANG, Ly. State Management Analyses of the Flutter Application. 2019. Disponível em: https://core.ac.uk/reader/286245850

JABLEČNÍK, Martin. Most popular Flutter state management libraries in 2023. 2023. Disponivel em: https://dev.to/mjablecnik/most-popular-flutter-state-management-libraries-in-2023-amc

PRAYOGA, R. R. et al. Performance Analysis of BLoC and Provider State Management Library on Flutter. 2021. Disponível em: https://www.researchgate.net/publication/355476480_Performance_Analysis_of_BLoC_and_Provider_State_Management_Library_on_Flutter

SLEPNEV, Dmitrii. State management approaches in Flutter. 2020. Disponível em: https://www.theseus.fi/bitstream/handle/10024/355086/Dmitrii_Slepnev.pdf

SZCZEPANIK, Michal; KEDZIORA, Michal. State Management and Software Architecture Approaches in Cross-platform Flutter Applications. 2020. Disponível em: https://pdfs.semanticscholar.org/9825/9e8cc97c2e073409d920dfd08be34e90463b.pdf?_gl=1*11djqfq*_ga*MTcyODA5MTU1OS4xNzAyODU5NzM1*_ga_H7P4ZT52H5*MTcwMjg1OTczNS4xLjAuMTcwMjg1OTczNi41OS4wLjA