UNICESUMAR – UNIVERSIDADE CESUMAR
CENTRO DE CIÊNCIAS EXATAS TECNOLÓGICAS E AGRÁRIAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

JOÃO VITOR MARTINS

Bloc x Provider: Uma análise comparativa em aplicações de pequeno porte

MARINGÁ/2024

JOÃO VITOR MARTINS

Bloc x Provider: Uma análise comparativa em aplicações de pequeno porte

Trabalho de Conclusão de Curso apresentado para obtenção do grau de bacharel em Engenharia de Software, do Centro Universitário Cesumar — Unicesumar

Orientador: Prof. Dr. João Choma

MARINGÁ/2024

JOÃO VITOR MARTINS

Bloc x Provider: Uma análise comparativa em aplicações de pequeno porte

Trabalho de Conclusão de Curso apresentado para obtenção do grau de bacharel em Engenharia de Software, do Centro Universitário Cesumar — Unicesumar

Orientador: Prof. Dr. João Choma

BANCA EXAMINADORA

_____

Prof. (Nome do orientador)

Afiliações

_____

Prof. (Nome do professor avaliador)

Afiliações

_____

Prof. (Nome do professor avaliador)

Afiliações

AGRADECIMENTOS

Dedico essa seção para agradecer primeiramente meu pai, William Michael Martins, e minha mãe, Rosemeyre de Oliveira Moraes Martins, por terem sempre me apoiado nessa minha jornada que desde o inicio passou por altos e baixos e agora está chegando ao fim. É um ciclo que se encerra, mas uma nova caminhada que se inicia. Sem eles nunca teria sequer pensado no caminho em que futuramente trilharia.

Agradeço também, meu irmão, Miguel Henrique Martins, que mesmo morando longe a alguns anos, e com uma vida atípica como tenente da polícia militar do estado de São Paulo, sempre tirava do seu tempo para me incentivar a continuar a seguir meus sonhos, ele sempre estava lá, me dizendo como ele estava feliz por eu estar pensando grande. Todo dia sinto sua falta, mas é o acaso da vida, caminhos foram e continuam sendo trilhados.

Sou grato pelo meu Professor Orientador, João Choma, que desde o inicio me apoiou e me ajudou a sustentar a minha ideia do TCC, sem ele, não saberia nem como e por onde começar.

Sou grato a todos por terem participado de processo tão importe que é a reta final/conclusão de uma graduação.

RESUMO

Nos últimos anos, o desenvolvimento de aplicativos móveis evoluiu com a introdução de frameworks que facilitar a criação de interfaces de usuário mais interativas e responsivas. O Flutter permite a criação de aplicativos nativos, com sua base de código sendo a mesma para aplicativos Android e iOS, o que é o recurso mais vantajoso. O gerenciamento de estado, no entanto, continua sendo um problema persistente neste cenário. Provider e BloC são os dois padrões de gerenciamento de estado do Flutter, onde seus objetivos diferem em como manter a IU(Interface do Usuário) atualizada e eficiente na mudança de estado. O Provider é a ferramenta oficial de gerenciamento de estado do framework e a mais comum devido à sua facilidade de uso. Ele torna mais fácil conectar diferentes componentes de um aplicativo usando o padrão de design InheritedWidget, portanto, adequado para aplicativos de pequena porte. O BloC, por outro lado, é a implementação completa que separa a regra de negócios da implementação da IU. Por esse motivo, o BloC se adequará melhor a aplicativos grandes e complexos, mas também a implementação levará mais tempo e curva de aprendizado. Este estudo comparou dois métodos de gerenciamento de estado, destacando descobertas relacionadas ao desempenho em uso de memória e CPU. Para isso, foram construídas duas aplicações idênticas, cada uma implementando um dos gerenciadores de estado. Os resultados obtidos foram analisados, e a análise final compilou os pontos fortes e fracos de cada técnica. Essa comparação oferece uma base para desenvolvedores interessados em construir aplicativos móveis com foco em desempenho, indicando como cada abordagem se diferencia em relação aos recursos consumidos.

ABSTRACT

In recent years, mobile app development has evolved with the introduction of frameworks that facilitate the creation of more interactive and responsive user interfaces. Flutter enables the creation of native applications with a single codebase for both Android and iOS apps, which is its most advantageous feature. However, state management remains a persistent challenge in this context. Provider and BloC are two state management patterns in Flutter, with distinct approaches for keeping the UI (User Interface) updated and efficient during state changes. Provider is the official state management tool of the framework and is the most commonly used due to its ease of use. It simplifies connecting different components of an application by using the InheritedWidget design pattern, making it well-suited for smaller applications. BloC, on the other hand, is a more comprehensive implementation that separates business logic from UI implementation. For this reason, BloC is better suited for large and complex applications, though it also requires more time to implement and has a steeper learning curve. This study compared the two state management methods, highlighting findings related to memory and CPU usage. To do so, two identical applications were built, each implementing one of the state managers. The results were analyzed, and the final analysis compiled the strengths and weaknesses of each technique. This comparison provides a foundation for developers interested in building mobile applications with a focus on performance, illustrating how each approach differs in terms of resource consumption.

LISTA DE FIGURAS

LISTA DE SIGLAS

UI - Interface do Usuário

CPU - Unidade Central de Processamento (Central Processing Unit)

BLoC - Business Logic Component

MVVM - Model-View-ViewModel

CRUD - Criar, Ler, Atualizar e Excluir (Create, Read, Update, Delete)

SDK - Kit de Desenvolvimento de Software (Software Development Kit)

NDK - Kit de Desenvolvimento Nativo (Native Development Kit)

LLVM - Máquina Virtual de Baixo Nível (Low-Level Virtual Machine)

RAM - Memória de Acesso Aleatório (Random Access Memory)

RES - Memória Residente (Resident Memory)

POO - Programação Orientada a Objetos

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, o desenvolvimento de aplicativos móveis tem evoluído significativamente, impulsionado por frameworks que facilitam a criação de interfaces de usuário interativas e responsivas. Entre esses frameworks, o Flutter, desenvolvido pelo Google, tem ganhado destaque pela sua capacidade de construir aplicativos nativos com uma única base de código, oferecendo uma experiência de usuário consistente e de alta performance tanto em Android quanto em iOS (docs.flutter.dev).

Uma das questões cruciais no desenvolvimento com Flutter é o gerenciamento de estado, que se refere à maneira como os dados são mantidos e compartilhados entre diferentes partes do aplicativo. O gerenciamento eficaz do estado é essencial para garantir que a interface de usuário reaja corretamente às mudanças nos dados, mantendo a consistência e a performance do aplicativo. Diversas abordagens e bibliotecas foram desenvolvidas para abordar essa necessidade, destacando-se entre elas o Provider e o BloC (Business Logic Component) (docs.flutter.dev e bloclibrary.dev).

O **Provider** é uma solução oficial do Flutter para o gerenciamento de estado, sendo amplamente utilizado pela sua simplicidade e integração com a arquitetura do Flutter. Essa solução facilita a propagação de mudanças no estado de maneira eficiente, usando o poder do InheritedWidget (docs.flutter.dev). Por outro lado, o **BloC** promove uma separação clara entre a lógica de negócios e a interface de usuário, utilizando o conceito de *streams* para gerenciar o estado de forma reativa, o que pode ser vantajoso para aplicativos complexos e com muitas interações de usuário (bloclibrary.dev).

O Provider, por sua simplicidade e integração direta com o Flutter, é eficiente para aplicações menores e menos complexas, mas pode sofrer com problemas de escalabilidade em aplicações maiores e com maior número de interações. Quando múltiplos widgets precisam ser reconstruídos frequentemente, o tempo de resposta pode aumentar, impactando a fluidez da aplicação (*Flutter Docs: Provider*). Por outro lado, o BLoC, embora mais complexo de implementar, oferece uma maneira mais robusta de gerenciar o estado em aplicativos maiores, especialmente em cenários onde há uma grande quantidade de interações simultâneas ou mudanças de estado. O BlocC usa streams, o que permite um gerenciamento de dados mais eficiente,

evitando que a UI seja reconstruída desnecessariamente. No entanto, essa abordagem pode ter uma curva de aprendizado mais acentuada e, se mal implementada, pode também introduzir problemas de desempenho, como atrasos na atualização da UI ou complexidade excessiva na gestão dos eventos (*Bloc Library*).

No desenvolvimento de aplicativos móveis, especialmente aqueles que requerem grande interação com o usuário, o desempenho é um fator crítico. O tempo de resposta e o consumo de recursos, como memória e processamento, impactam diretamente a experiência do usuário, considerando a grande diversidade de hardware disponível. O aplicativo deve ser capaz de funcionar de maneira eficiente tanto em dispositivos de alta performance quanto em equipamentos mais limitados, garantindo uma experiência consistente independentemente da capacidade do hardware utilizado. Por isso, a escolha de uma abordagem adequada para o gerenciamento de estado é fundamental.

Este trabalho tem como objetivo comparar os gerenciadores de estado Provider e BloC, analisando suas principais características, vantagens, desvantagens e desempenho. Para isso, será executado um estudo experimental que avaliará a implementação de aplicativos acadêmicos utilizando ambas as abordagens, com o intuito de observar e documentar as diferenças práticas no desenvolvimento e na performance dos aplicativos. A escolha desse tema se justifica pela relevância e atualidade no contexto do desenvolvimento móvel, especialmente para desenvolvedores que buscam construir aplicativos escaláveis e de alta qualidade. Além disso, a comparação entre Provider e BloC pode fornecer *insights* valiosos para a tomada de decisões sobre qual abordagem adotar em diferentes cenários de desenvolvimento.

A estrutura do trabalho é organizada da seguinte maneira: na seção 3 a fundamentação teórica, onde serão apresentados os conceitos básicos e as principais características dos gerenciadores de estado em Flutter. Já na seção 04, é apresentado os trabalhos relacionados, na qual fornecem uma visão aproximada do tema na qual é abordado neste trabalho. Na seção 5 é apresentada a metodologia, na qual descreve o método de pesquisa adotado, incluindo os critérios de comparação e as ferramentas utilizadas. Seção 06, Desenvolvimento, onde é exemplificado e elucidado a forma na qual o fluxo de desenvolvimento que as aplicações seguiram,

mantendo uma parcimônia de como os dados são transicionados e construídos em tela. A Seção 07 apresenta os resultados encontrados no estudo das implementações práticas e a análise comparativa dos resultados, respectivamente. Finalmente, a Seção 08 indica as principais descobertas e considerações finais sobre o estudo realizado.

2. OBJETIVO

O objetivo geral deste trabalho é realizar uma análise comparativa entre os gerenciadores de estado Provider e BloC no contexto do desenvolvimento de aplicativos móveis utilizando o framework Flutter. Para alcançar esse objetivo, o trabalho se propõe a:

1. Explorar as Características Técnicas: Investigar as principais características técnicas de cada gerenciador de estado, incluindo arquitetura, princípios de funcionamento, e integração com o Flutter.
2. Comparar a Performance: Realizar testes de performance para comparar a eficiência de cada gerenciador em consumo de memória, consumo de processador e capacidade de lidar com as requisições de dados.
3. Identificar Vantagens e Desvantagens: Identificar e documentar as vantagens e desvantagens de cada gerenciador, considerando diferentes cenários de desenvolvimento, como aplicativos simples, aplicativos com lógica de negócios complexa e aplicativos com alta demanda de interatividade.
4. Contribuir para a Comunidade de Desenvolvedores: Compartilhar os resultados e insights obtidos com a comunidade de desenvolvedores, contribuindo para o avanço do conhecimento e melhores práticas no desenvolvimento de aplicativos móveis com Flutter.

Através dessa análise detalhada, espera-se fornecer uma visão clara e fundamentada sobre as diferenças e semelhanças entre Provider e BloC, permitindo que desenvolvedores façam escolhas informadas e adequadas aos requisitos de seus projetos.

3. FUNDAMENTAÇÃO TEÓRICA

3.1 Gerenciamento de estado

Segundo Arshad, W. (2021), o gerenciamento de estado pode ser definido como uma ou várias técnicas, que é utilizada para cuidar de mudanças que pode ocorrer a partir das interações dentro do código, sendo usada conforme expansão da aplicação e a necessidade de responder ao usuário de acordo com todas as mudanças realizadas dentro do software. O gerenciamento de estado destaca as características apresentadas:

1. Escalabilidade: De acordo com a evolução de um sistema, o controle e gerenciamento dos estados de um atributo, se tornam cada vez mais crucial para a usabilidade do produto. Sendo assim, é possível manter e controlar todas a comunicações entre as telas, tornando-as mais consistentes e responsivas.
2. Responsividade ao Usuário: Com o sucesso da implementação de um bom gerenciador de estado, é possível que o usuário tenha uma boa experiencia a partir de suas respectivas interações, já que, o sistema estará respondendo aos inputs do usuário de forma satisfatória as expectativas do usuário.

3.2 Flutter/Dart

Segundo a documentação oficial do Flutter (Google, 2024), Flutter/Dart é uma ferramenta de desenvolvimento de UI (Interface do Usuário) responsivas, criada pela empresa Google, destinada ao desenvolvimento de aplicações nativas, tanto Mobile (Android e IOS), quanto para Web e Desktop, oferecendo um framework que unifica a construção de interfaces, com uma única base de código. É possível citar suas principais características:

- Multiplataforma: Facilita o desenvolvimento de aplicações para multiplataformas (vários sistemas operacionais diferentes), tendo apenas uma única base de código.
- POO (Programação Orientada a Objetos): Exige conhecimentos básicos em POO, ou seja, familiaridade com o desenvolvimento de classes, métodos, funções, atributos, polimorfismo, herança e entre outros.

- Desempenho: Por se tratar de uma tecnologia que foi construída em C e C++ para a execução do código, incluindo o Native Development Kit (NDK) no Android e o Low-Level Virtual Machine (LLVM) no IOS, ajudando em um desenvolvimento mais eficiente.
- Arquitetura: O Flutter se baseia na arquitetura em Widgets e no Hot Reload, onde nos Widgets, são blocos de construção fundamentais para a interação e visualização da interface de usuário. Já no Hot Reload, que permite o recarregamento das alterações realizadas dentro do código em tempo real, sem precisar recompilar o código, acelerando o desenvolvimento.

3.3 Provider

O Provider está inserido ao Flutter e é uma ferramenta utilizada para gerenciar o estado de um aplicativo, permitindo acompanhar, receber e notificar mudanças de maneira eficiente. Ele trabalha com componentes como Notifier, Provider e Consumer para garantir que a interface do usuário responda corretamente às atualizações no estado do aplicativo (docs.flutter.dev).

Os Notifiers funcionam como observadores, detectando quando algo muda e alertando o sistema para que ele reaja adequadamente. Isso é feito com o uso de classes como *ChangeNotifier*. Já o *Provider* atua como um intermediário, distribuindo informações por toda a árvore de widgets, o que facilita a comunicação entre diferentes partes da aplicação (docs.flutter).

Segundo a documentação oficial da dependência do Provider (pub.dev) existem diferentes tipos de Providers que você pode usar, como o MultiProvider, que permite trabalhar com vários providers ao mesmo tempo, e o ChangeNotifierProvider, que é útil para gerenciar estados dinâmicos, como quando um usuário adiciona ou remove itens de um carrinho de compras. Além disso, há o FutureProvider e o StreamProvider, que são especializados para trabalhar com dados assíncronos, embora não reajam a mudanças de estado por conta própria sem a ajuda do ChangeNotifierProvider.

Os Consumers são a parte da aplicação que utiliza os dados providos pelo *Provider*. Eles podem ser configurados de várias formas, como com o Provider.of, Consumer, e Selector, cada um oferecendo diferentes níveis de controle sobre quando

e como a interface deve ser atualizada com base nas informações recebidas (SAMBO, 2020).

3.4 BloC (streams)

De acordo com os pesquisadores Szczepanik e Kędziora (2020), Business Logic Component ou como é mais conhecido BloC, é um sistema de gerenciamento de estado centralizado, ou seja, é possível acessar e gerenciar os dados a partir de um local central e único, isso tudo dentro do projeto. O BloC padrão para Flutter ajuda a gerenciar o estado da aplicação usando streams, sem precisar de uma biblioteca externa, sendo parecido com o padrão MVVM (Model-View-ViewModel), tendo a diferença da comunicação, que é as streams.

Segundo a documentação do BloC (pub.dev 2024), esta ferramenta acaba por ser um padrão de design que tenta ajudar a separar a apresentação da lógica de negócios, onde seguindo está arquitetura, facilita na manutenibilidade e na reutilização de código, abstraindo aspectos reativos do padrão, permitindo que os desenvolvedores consigam focar mais nas regras de negócios. O Bloc é uma ferramenta sofisticada no padrão BloC para flutter que ajuda a gerenciar o estado de uma aplicação de forma mais organizada, onde ao invés de mudar o estado diretamente com uma função, o Bloc acaba por trabalhar, escutar e processar eventos, onde após o processamento ele atualiza o estado do aplicativo (pub.dev 2024).

3.5 Comparação

Segundo estudos e comparações realizados como o Prayoga et al. (2021), foi concluído que o gerenciamento de estado proporcionado pelo BloC, há um consumo menor de CPU (Core process unit), uso de memória RAM e também um tempo de execução menor em comparação com o gerenciamento de estado padrão do Flutter (setState). Já sobre a utilização do Provider, como o gerenciamento de estado da aplicação, houve baixo consumo de CPU, memória RAM e tempo de execução em comparação com o setState. Porém, ao analisar o setState, ele acaba por ser mais eficiente em termos de resposta em um nível alto da hierarquia de widgets

Esses dados sugerem que tanto o uso do Provider quanto do BloC é eficiente em termos de recursos computacionais, como CPU e RAM, especialmente em widgets profundamente aninhados na hierarquia de componentes. No entanto, essa eficiência tende a diminuir em níveis mais elevados da hierarquia, indicando que a escolha entre as duas abordagens deve ser orientada pelas necessidades específicas do projeto e os objetivos a serem alcançados (Prayoga et al., 2021).

4. TRABALHOS RELACIONADOS

Prayoga (2021) apresentou um modelo de análise comparativa entre dois gerenciadores de estado, o Provider e o BloC. Nesse estudo, os autores examinaram essas dependências utilizando métricas de desempenho como tempo de resposta e consumo de recursos, aplicados a diferentes cenários de desenvolvimento. A análise resultou em insights sobre as situações em que cada gerenciador é mais adequado, levando em conta a complexidade das aplicações e a eficiência de cada abordagem.

É importante frisar, que no estudo realizado por Prayoga (2021), foi realizado a comparação entre Bloc e Provider, porém, não uma comparação direta entre ambos. No estudo apresentado, foi feito um processo comparativo, entre Bloc e Provider, olhando como base, o gerenciador de estado básico do flutter, o setState, que informa o framework quando um estado interno de uma variável sofre algum tipo alteração, interferindo na e impactando na visualização da interface que está disponível para o usuário, agendado uma compilação que altera apenas de forma visual, o que é apresentado na tela. Sendo assim, quando o estado é atualizado utilizando o setState, ele informa a tela de forma instantânea que algo foi modificação, mostrando assim, o novo dado. (api.flutter.dev).

5. METODOLOGIA

Nesta seção, serão apresentadas a classificação de pesquisa em relação a abordagem utilizada para comparação entre os gerenciadores, os objetivos, fontes de informações e procedimento técnico adotado. A classificação da pesquisa, de forma resumida, pode ser visualizada no quadro a seguir:

Pesquisa	Classificação
Natureza	Aplicada
Objetivos	Comparativa
Abordagem	Método misto e explanatório
Fonte de Informação	Campo e literatura
Procedimentos técnicos	Estudo bibliográfico Implementação de exemplos práticos (Provider e BloC) Testes Análise comparativa dos resultados

Tabela 1 Ilustração metodologia.

A pesquisa busca aprofundar o conhecimento sobre gerenciadores de estado com foco em sua aplicação prática, especificamente no desenvolvimento de sistemas e nas técnicas de gerenciamento em aplicações Flutter. Este estudo visa produzir e disseminar insights que contribuam para o desenvolvimento mais eficiente de projetos futuros. Através de uma abordagem comparativa, o objetivo é analisar os resultados de testes para identificar diferenças e semelhanças entre Provider e BloC. Espera-se oferecer uma análise fundamentada que auxilie desenvolvedores a fazer escolhas informadas sobre qual gerenciador de estado adotar em seus projetos.

Esta pesquisa adota uma metodologia de métodos mistos, combinando dados qualitativos e quantitativos para proporcionar uma análise mais completa. A escolha dessa abordagem se justifica pelo uso da análise explanatória, direcionada a esclarecer as diferenças de desempenho entre Provider e BloC, examinando suas influências no desenvolvimento de aplicações Flutter.

5.4 Fonte de Informação

Os dados coletados para esta pesquisa originam-se de duas fontes principais: análise experimental e revisão de literatura. Desenvolvimento de duas aplicações práticas semelhantes, nas quais serão analisadas métricas específicas para avaliar o desempenho dos gerenciadores de estado. Consulta a artigos e trabalhos acadêmicos para aquisição de conhecimento teórico prévio. Essa etapa incluiu a análise de

documentações oficiais do Provider, BloC e Flutter/Dart, além de pesquisas no Google Acadêmico para identificar estudos relacionados relevantes.

6. ESTUDO EXPERIMENTAL

Para o desenvolvimento deste projeto, foram seguidos processos fundamentais típicos de criação de software, incluindo:

- **Elicitação de Requisitos:** Responsável pela definição das funcionalidades do sistema, esta etapa estabeleceu as bases para o que cada aplicação deveria oferecer.
- **Desenvolvimento da Arquitetura:** Focada na estruturação e no fluxo de dados do aplicativo, garantiu que as soluções fossem adequadamente planejadas para o uso de Provider e BloC.
- **Desenvolvimento e Aplicação de Testes:** Foram definidos e implementados testes específicos para a coleta de métricas necessárias à comparação entre os gerenciadores de estado.

Após esses processos, realizou-se o experimento, onde o aplicativo foi testado em condições próximas ao uso real, simulando a experiência de um usuário comum.

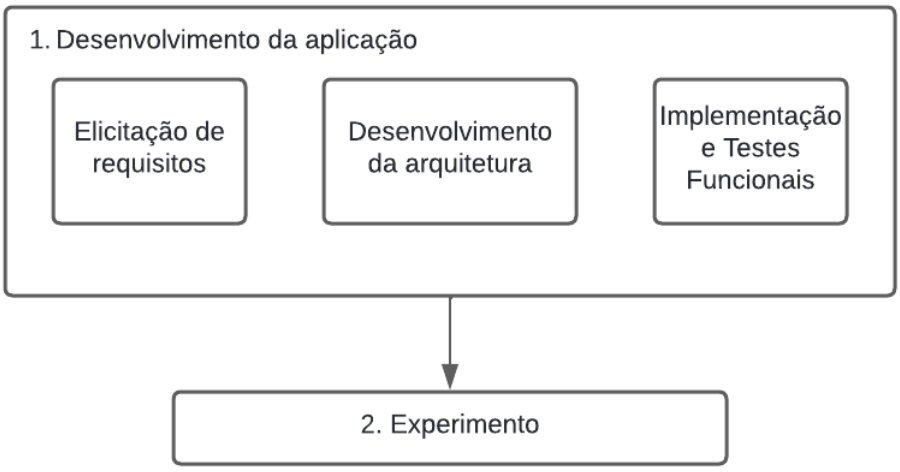

Figura 1. Ilustração do escopo de desenvolvimento.

Durante o processo de desenvolvimento, foi idealizada uma aplicação de lista de tarefas (*To-do List*), abrangendo todas as operações de CRUD (Criar, Ler, Atualizar e Excluir) das tarefas. Entre as funcionalidades, destaca-se a possibilidade de alterar o status das tarefas entre “aberta”, “em execução” e “fechada”, além de definir prioridades (baixa, média e alta) que organizam as tarefas em páginas distintas.

Para essa aplicação, foi escolhida uma arquitetura modular. Esse modelo permite uma separação clara entre a regra de negócio e as demais camadas do projeto, promovendo uma estrutura organizada e flexível. A arquitetura modular facilita, ainda, a criação de rotas independentes, a injeção de dependências moduladas e a padronização da construção do projeto, promovendo uma navegação intuitiva e uma manutenção simplificada, apresentado na Figura 1, onde mostra a estrutura da pasta de um projeto Flutter. Tudo está diretamente abaixo do diretório principal lib, que está sendo dividido em subpastas, e dentro de cada subpasta existem arquivos que estão relacionados aos recursos específicos da aplicação. Esta é a aparência da estrutura.

- **Model (Modelo):** Alocação dos arquivos referentes a modelagem dos dados.
- **Module (Módulo):** Alocação dos arquivos referente as telas de visualização e suas respectivas states, modules (definição de rotas) e controllers (intermédio da chamada da regra de negócio). É importante ressaltar, que a state é apenas utilizada para aplicação que utiliza o Bloc, sendo descartada no Provider.
- **Service (Serviço):** Alocação da assinatura do método, representada pela “service” e respectivamente sua implementação da regra de negócio.

Três dos arquivos mais importantes dentro do diretório raiz do projeto são “app_module.dart”, “app_widget.dart”, e “main.dart”.

- **App module:** Definição de rotas e o instanciamento das services globais.
- **App widget:** Chamada da tela inicial e a inicialização das rotas globais.
- **App main:** Inicialização do aplicativ0

6.3 Análise experimental

Os testes de comparação foram estruturados para incluir uma série de 10 requisições para cada operação do CRUD em ambas as implementações. Para o BloC, foram realizados 10 procedimentos de criação, 10 de alteração, 10 de exclusão e 10 de listagem, repetidos sequencialmente por 5 ciclos. Os mesmos testes foram aplicados à versão da aplicação desenvolvida com o Provider. É importante destacar que ambas as aplicações foram desenvolvidas utilizando a mesma arquitetura modular e conectadas ao mesmo servidor RESTful para as requisições, diferenciando-se apenas no método de gerenciamento de estado adotado. Além disso, a renderização da interface do usuário foi padronizada, fazendo com que as aplicações sejam visualmente idênticas, como pode ser visualizado na Figura 2, onde as aplicações seguem os mesmos padrões de UI/Design.

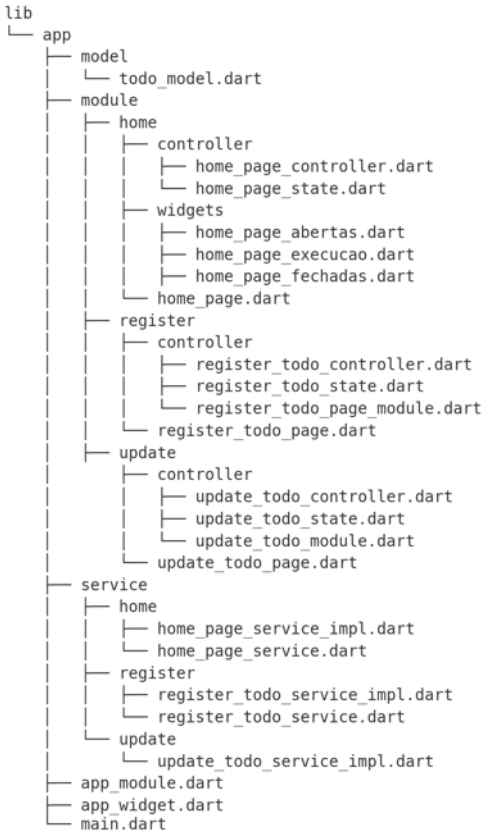

Figura 2. Ilustração Arquitetura de pastas.

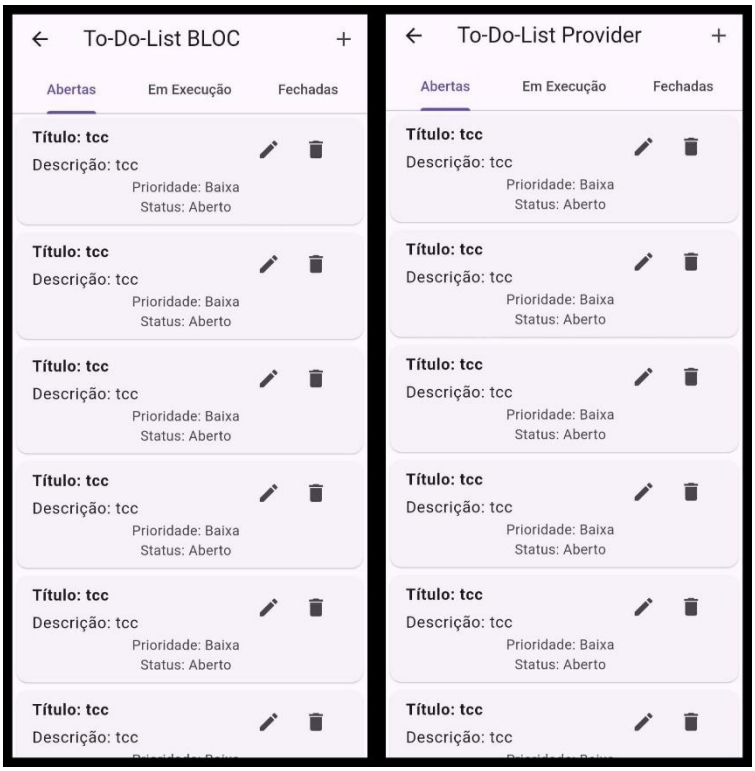

Figura 3. Ilustração UI/Design

7. RESULTADOS

Após uma série de testes, foram obtidos dados relevantes sobre o consumo percentual de memória RAM, uso de CPU e utilização total de memória, fatores cruciais para entender a eficiência no gerenciamento de recursos. Esses dados oferecem uma base valiosa para orientar desenvolvedores na identificação de possíveis gargalos de desempenho em suas aplicações, possibilitando ajustes específicos que minimizem disfunções de performance.

Os testes foram executados em um dispositivo físico móvel equipado com 8 GB de RAM e processador Exynos 2100 – arquitetura arm64, garantindo uma avaliação precisa e prática do consumo de recursos em um ambiente real.

Na figura 4, é possível verificar com mais alguns detalhes tanto a arquitetura do processador utilizado (Arquitetura arm64 de 64 bit), com as versões 3.4.3 do sdk do dart, juntamente com a versão stable channel (versão mais estável do framework flutter) 3.22.2.

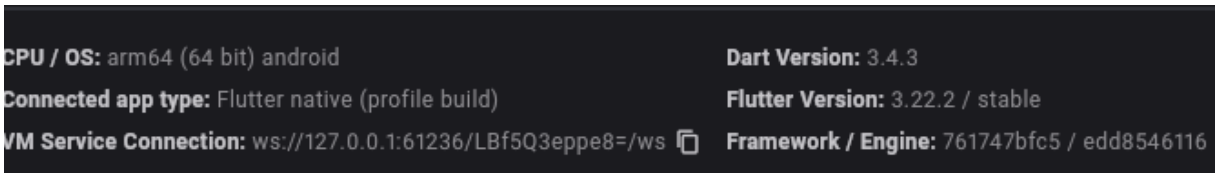

Figura 4. Ilustração arquitetura móvel.

7.2 Uso CPU e Memória - BLOC

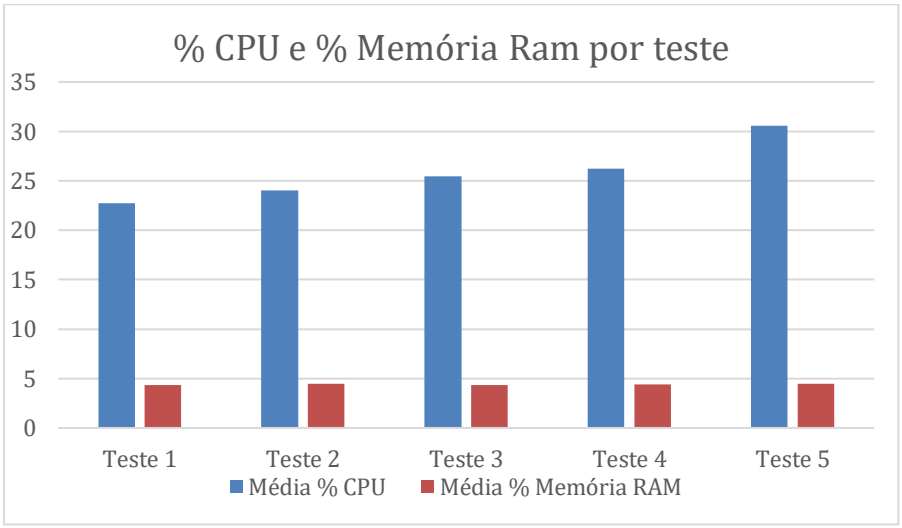

Figura 5. Ilustração % cpu e % memória utilizada

A Figura 5 apresenta os resultados do uso de CPU, revelando uma tendência crescente no consumo ao longo dos testes. No Teste 1, o consumo inicial é de 22,74%, enquanto no Teste 5 alcança 30,56%, evidenciando um crescimento contínuo. Esse comportamento sugere que o BloC, ao lidar com eventos e transições de estado mais complexas, demanda maior capacidade de processamento. Esse aumento gradual requer análise adicional para identificar possíveis trechos de código que intensifiquem a demanda por CPU. No entanto, ao observar o consumo de memória, o BloC se mostra eficiente, mantendo-se abaixo de 5% de utilização total, o que indica um bom gerenciamento de recursos nesse aspecto.

7.3 Uso Memória (RES) – BLOC

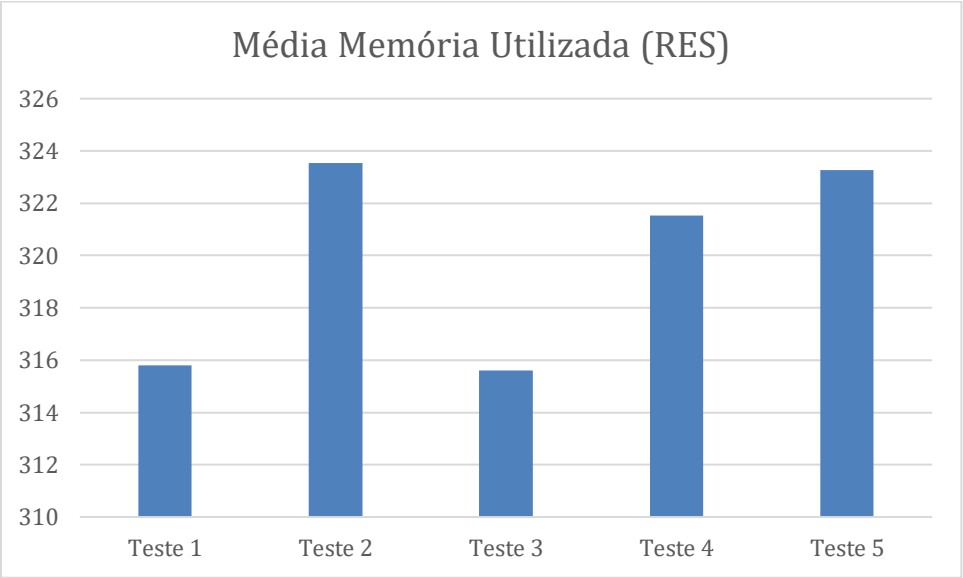

Figura 6. Ilustração memória res utilizada.

A Figura 6 ilustra os resultados do uso de memória, mostrando que a "Média de Memória Utilizada (RES)" apresenta uma variação entre os testes, com uma tendência de aumento gradual. No Teste 1, a memória RES média inicia em 315,8 MB, enquanto nos testes subsequentes esse valor cresce, atingindo seu pico em 323,533 MB no Teste 2 e se mantendo próximo, com 323,267 MB, no Teste 5. Esse aumento indica um leve crescimento no consumo de memória ao longo do tempo, sugerindo a necessidade de monitoramento contínuo para avaliar potenciais otimizações.

7.5 Uso CPU e Memória – Provider

A Figura 7 apresenta uma comparação do uso de CPU e memória entre o Provider e o BloC. Observando os gráficos referentes ao Provider, percebe-se uma tendência de consumo mais elevado de CPU ao longo dos testes. No Teste 1, o consumo inicial de CPU do Provider é de 35,74%, diminuindo para 29,24% no Teste 5, o que, apesar de representar uma leve redução, indica que o Provider exige maior capacidade de processamento em comparação ao BloC, mesmo utilizando uma estrutura menos complexa.

No entanto, ao analisar o uso de memória, o Provider mostra-se eficiente, mantendo o consumo abaixo de 5% da memória total, semelhante ao BloC. Esse dado

sugere um gerenciamento de memória adequado, com estabilidade no consumo, o que é um aspecto positivo em relação à eficiência dos recursos.

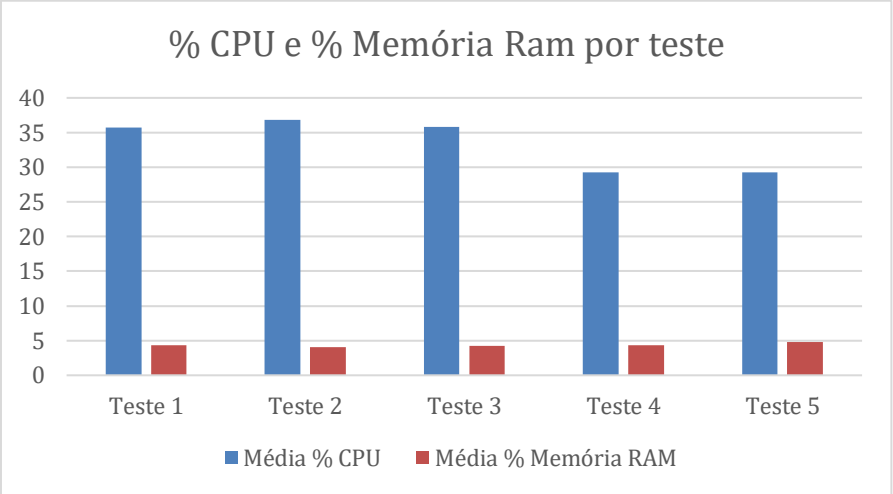

Figura 7. Ilustração % cpu e % memória utilizada (Provider).

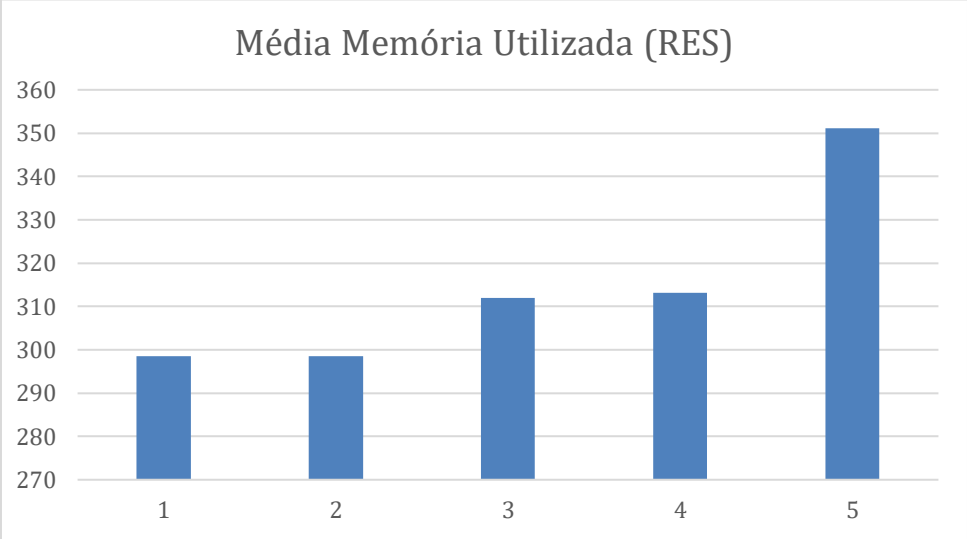

Figura 8. Ilustração memória res utilizada.

A Figura 8 apresenta a "Média de Memória Utilizada (RES)", que exibe uma variação ao longo dos testes, com uma tendência de aumento gradual. No Teste 1, a média inicial de memória RES é de 298,53 MB. Nos testes seguintes, esse valor cresce progressivamente, atingindo seu pico em 351,13 MB no último teste. Esse aumento contínuo indica um crescimento no consumo de memória ao longo do tempo,

sugerindo a necessidade de atenção para identificar eventuais otimizações que possam estabilizar esse uso.

7.6 Provider x Bloc

A Figura 9 evidencia uma diferença significativa no uso médio de CPU entre os gerenciadores de estado BloC e Provider. Conforme mostram as estatísticas, o BloC registra um consumo médio de CPU de aproximadamente 25,8%, enquanto o Provider apresenta uma média de 33,39%. Essa diferença de 7,5% sugere que o BloC é ligeiramente mais eficiente em termos de processamento, demandando menos tempo de CPU. Em relação ao uso de memória, ambos os gerenciadores demonstraram desempenho bastante semelhante. O BloC registra um consumo médio de memória de 4,4%, comparado a 4,36% do Provider. Essa diferença mínima indica que, no quesito uso de RAM, ambos operam de maneira quase idêntica, sem grande impacto no desempenho geral do aplicativo.

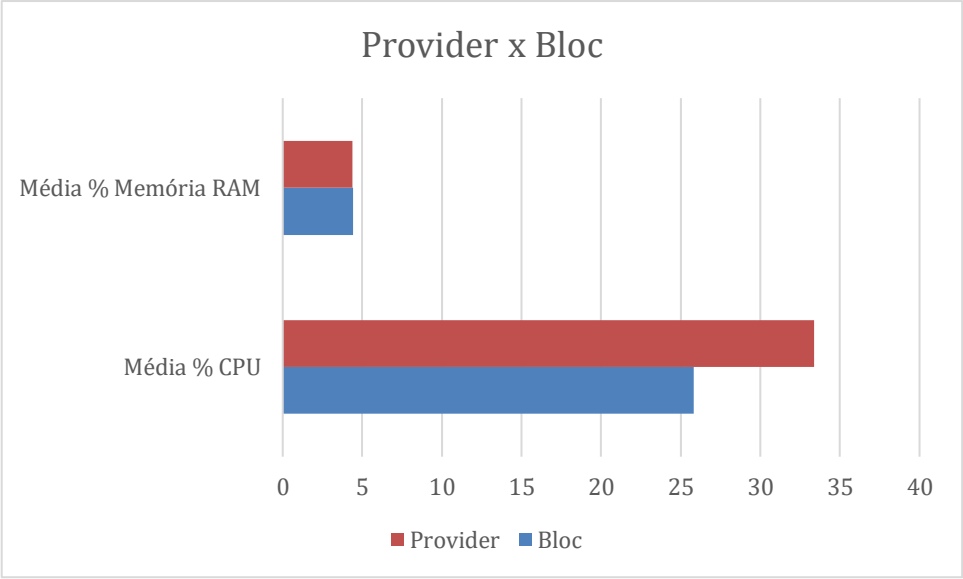

Figura 9. Comparação % cpu e % memória (Bloc x Provider).

Por fim, a Figura 10 apresenta uma análise da memória real utilizada (RES), indicando a quantidade exata de memória ocupada pelos processos de cada gerenciador. Os resultados mostram valores muito próximos, com o BloC utilizando em média 319,94 MB e o Provider 318,19 MB. Essa proximidade sugere que ambos

os gerenciadores mantêm um consumo de memória similar, sem diferenças significativas que possam impactar o desempenho da aplicação.

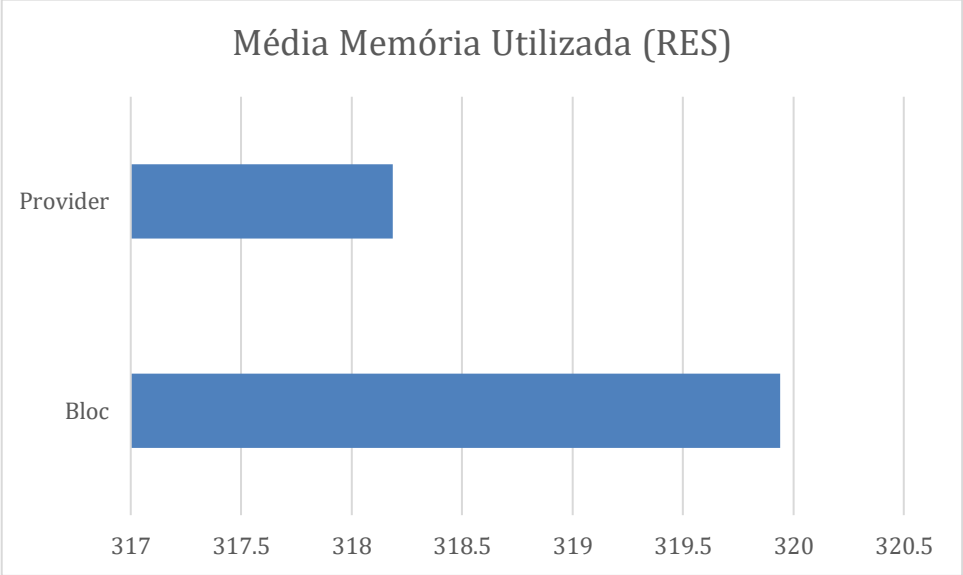

Figura 10. Ilustração memória res utilizada (Bloc x Provider).

A comparação abrangeu entre os gerenciadores de estados Bloc e Provider, abrangeu tantos parâmetros técnicos, quanto parâmetros de desempenho, dando uma visão base assim de seus prós e contras. As métricas de desempenho medidas incluem uso médio de CPU, uso médio de RAM e uso médio de memória residente (RES), e os seguintes resultados obtidos foram:

BloC:

- Uso médio de CPU: 25,8%
- Uso médio de memória: 4,4%
- Memória residente usada (RES) média: 319,94 MB

Provider:

- Uso médio de CPU: 33,38%
- Uso médio de memória: 4,36%
- Memória residente usada (RES) média: 318,19 MB

Com base nos dados, o BloC consumiu menos CPU em comparação com o Provider, mostrando melhor eficiência no processamento de dados na CPU. A arquitetura reativa do BloC, que utiliza streams no gerenciamento de estado, traz mais eficiência, evitando reconstruções desnecessárias da interface do usuário. Quanto ao uso de RAM e memória residente, os dois gerenciadores apresentaram resultados muito próximos, sem grandes diferenças. Isso significa que, em relação à memória ocupada, é praticamente o mesmo para ambos: Provider e BloC. Nesse sentido, não há um impacto significativo no desempenho do aplicativo.

Ao escolher entre Provider e Bloc, as necessidades específicas do projeto devem ser o principal fator de decisão. O Provider é simples de implementar e eficiente para aplicativos mais simples ou que não exigem muito desempenho. Para aplicativos complexos com muitas interações e que precisam ser otimizados em termos de desempenho, o Bloc seria mais apropriado, considerando que consome menos CPU e segue uma arquitetura que mantém o código mais organizado e escalável.

8. CONCLUSÕES

O presente trabalho é uma análise comparativa dos gerenciadores de estado Provider e BloC no processo de desenvolvimento de aplicativos móveis com o framework Flutter. O estudo apresentou características técnicas, vantagens e desvantagens, e o desempenho de cada um, com o objetivo de fornecer insights que possam ser ajudar os desenvolvedores na decisão da ferramenta que mais se encaixa com base na necessidade do programador.

Os conceitos básicos de gerenciamento de estado foram considerados dentro de um escopo teórico, onde as peculiaridades de um ambiente de desenvolvimento Flutter/Dart e peculiaridades de ferramentas de gerenciamento de estado como Provider e BloC podem impactar na sua eficiencia. Além disso, os trabalhos relacionados foram analisados quanto à contribuição para os fundamentos e direções de pesquisa.

Esta análise experimental criou duas aplicações idênticas, tanto em funcionalidades quanto em arquitetura, sendo a única diferença o gerenciador de estado. Ambas as aplicações tinham uma Lista de Tarefas completa, totalmente capaz de operações CRUD para gerenciar o estado de cada tarefa, juntamente com

funcionalidades para gerenciar a prioridade e o status. Usando o método de implementação, testes experimentais foram realizados para medir a CPU, RAM e memória residente (RES) consumidas pelas duas implementações usando métricas precisas e replicáveis.

O resultado mostrou que o BloC realmente consumiu uma quantidade menor de CPU em comparação com o Provider e, portanto, é muito mais eficaz no processamento de dados. Isso pode ser atribuído à arquitetura reativa do BloC, onde fluxos são capturados e tratados para otimizar o gerenciamento do estado sem construções desnecessárias de interfaces de usuário. Em termos de memória residente e memória RAM, ambos os gerenciadores apresentaram desempenhos muito similares, sem grandes diferenças que pudessem influenciar o desempenho geral das aplicações.

Com base nesses resultados, podemos concluir que o BloC é a alternativa mais eficiente em termos de uso de CPU, especialmente para aplicações exigentes que necessitam de um gerenciamento de estado resiliente e escalável. Já o Provider, apesar de ser um dos maiores consumidores de CPU, destaca-se pela integração estreita com o Flutter e pela facilidade de uso, sendo uma escolha viável para projetos menores ou equipes com menos experiência.

Esses fatores devem ser considerados ao escolher entre Provider e BloC, levando em conta as características específicas do projeto. Isso inclui a complexidade da lógica de negócios, as necessidades de desempenho e escalabilidade, a manutenibilidade da arquitetura e, acima de tudo, a curva de aprendizado necessária para a equipe de desenvolvimento.

Um dos principais focos deste estudo está nas métricas de desempenho relacionadas ao consumo de recursos, excluindo outros atributos de qualidade como facilidade de uso, legibilidade do código-fonte ou experiência do desenvolvedor. Além disso, para trabalhos futuros, a relização de pesquisas tendo em vista outros gerenciadores de estado disponíveis dentro do ecossistema Flutter, como por exemplo os gerenciadores, MobX, Redux, Riverpod.

Pode-se dizer ainda que os aspectos qualitativos podem incluir a produtividade da equipe, a manutenibilidade do código e como cada gerenciador de estado pode se

integrar facilmente com outras ferramentas e padrões arquiteturais, de modo a tentar dar uma visão panorâmica das implicações do gerenciador de estado. Em conclusão, este artigo é uma contribuição importante para a clarificação das diferenças entre as técnicas de gerenciamento de estado Provider e BloC em aplicações Flutter, de modo a fornecer dados de maneira empírica que possam guiar o desenvolvedor em sua tomada de decisão. Optar pelo gerenciamento de estado correto é uma garantia chave para a criação de aplicações eficientes, escaláveis e qualitativas, o que afeta a experiência final dos usuários e a sustentabilidade do projeto a longo prazo. Assim, espera-se que, com os insights que esta pesquisa irá proporcionar, profissionais e acadêmicos utilizem esta referência para buscar soluções otimizadas no desenvolvimento móvel com Flutter.

9. REFERÊNCIAS

ARSHAD, Waleed. Managing State in Flutter Pragmatically: Discover how to adopt the best state management approach for scaling your Flutter app. Packt Publishing Ltd, 2021.

Flutter Documentation. *Flutter FAQ: What is Flutter?*. Disponível em: https://docs.flutter.dev/resources/faq#:~:text=Flutter%20is%20Google's%20portable%20UI,is%20free%20and%20open%20source. Acesso em: 14 ago. 2024.

PRAYOGA, R. R.; ALIFIA SYALSABILA; GHIFARI MUNAWAR; RAHIL JUMIYANI. Performance Analysis of BLoC and Provider State Management Library on Flutter. Jurnal Mantik, *[S. l.]*, v. 5, n. 3, p. 1591-1597, 2021. Disponível em: https://www.ejournal.iocscience.org/index.php/mantik/article/view/1539. Acesso em: 14 aug. 2024.

SZCZEPANIK, M.; KĘDZIORA, M. **State Management and Software Architecture Approaches in Cross-platform Flutter Applications**. In: *Proceedings of the 15th*

International Conference on Software Technologies (ICSOFT 2020). SciTePress, 2020. p. 221-228. Disponível em: https://www.scitepress.org/Papers/2020/94116/94116.pdf. Acesso em: 14 ago. 2024.

SAMBO, Igor. *State Management: Provider*. Medium, 26 mai. 2020. Disponível em: https://medium.com/android-dev-moz/state-management-provider-c63dd9202a53. Acesso em: 14 ago. 2024.

BLOC LIBRARY. *BLoC Concepts*. Disponível em: https://bloclibrary.dev/bloc-concepts/. Acesso em: 14 ago. 2024.

BLOC LIBRARY. *BLoC Package Documentation*. Disponível em: https://pub.dev/documentation/bloc/latest/index.html. Acesso em: 14 ago. 2024.

GOOGLE LLC. *Flutter Performance - DevTools*. Disponível em: https://docs.flutter.dev/tools/devtools/performance. Acesso em: 20 out. 2024.