Sistemasi: Jurnal Sistem Informasi
Volume 15, Nomor 4, 2026: 1322-1331

ISSN:2302-8149
e-ISSN:2540-9719

Analisis Penerapan *Provider State Management Pattern* terhadap Performa dan *Maintainability* Aplikasi *Mobile* Apotek berbasis *Flutter*

Analysis of the Provider State Management Pattern for Performance and Maintainability in a Flutter-based Mobile Pharmacy Application

[1]Muhammad Asep Subandri*, [2]Fajri Profesio Putra, [3]Agus Tedyyana
[1,2,3]Jurusan Teknik Informatika, Politeknik Negeri Bengkalis
[1,2,3]Jl. Bathin Alam, Sungai Alam, Bengkalis, Riau 28711, Indonesia
*e-mail: subandri@polbeng.ac.id

*(**received**: 10 March 2026, **revised**: 15 April 2026, **accepted**: 16 April 2026)*

Abstrak

Pengelolaan state pada aplikasi mobile modern merupakan aspek krusial yang mempengaruhi performa rendering antarmuka dan kemudahan pemeliharaan kode. Penelitian-penelitian terdahulu tentang state management Flutter umumnya menggunakan aplikasi katalog sederhana sebagai objek pengujian dan berfokus pada perbandingan performa antar pendekatan tanpa mengukur aspek maintainability secara kuantitatif. Penelitian ini bertujuan menganalisis efektivitas penerapan Provider state management pattern dari sisi performa rendering UI dan maintainability kode pada studi kasus aplikasi bisnis nyata, yaitu aplikasi mobile Apotek Berkah. Metode yang digunakan adalah applied research dengan pendekatan studi kasus. Aplikasi terdiri dari 20 file Dart dengan total 5.899 LOC, dikembangkan menggunakan Flutter dengan arsitektur MultiProvider dan dependency injection. Pengukuran performa dilakukan menggunakan Flutter DevTools (widget rebuild count, frame rendering time, memory usage) dengan perbandingan terhadap setState sebagai baseline. Pengukuran maintainability dilakukan melalui sembilan aspek metrik meliputi distribusi kode per layer, metrik provider class, pola akses provider, kompleksitas screen, architectural patterns, dan dependency graph. Hasil menunjukkan bahwa Provider pattern mengurangi widget rebuild count rata-rata 52,3% dibandingkan setState, dengan frame rendering time konsisten di bawah 16ms. Dari sisi maintainability, arsitektur Provider menghasilkan low coupling (seluruh provider hanya bergantung pada ApiService), zero inter-provider dependency, dan separation of concerns yang terstruktur. Novelty penelitian ini terletak pada kombinasi evaluasi performa dan maintainability Provider pattern pada domain aplikasi bisnis apotek yang melibatkan operasi CRUD, manajemen inventaris, dan transaksi POS — suatu domain yang belum dikaji dalam literatur state management Flutter sebelumnya. Kontribusi utama berupa bukti empiris bahwa Provider pattern memberikan keseimbangan optimal antara performa, maintainability, dan kompleksitas implementasi untuk aplikasi mobile berskala menengah.

Kata kunci: *flutter, maintainability,* performa aplikasi mobile, *provider pattern, state management*

Abstract

State management in modern mobile applications is a critical aspect that affects both user interface (UI) rendering performance and code maintainability. Previous studies on Flutter state management have generally used simple catalog applications as test objects and focused on performance comparisons between approaches without quantitatively measuring maintainability aspects. This study aims to analyze the effectiveness of implementing the Provider state management pattern in terms of UI rendering performance and code maintainability, using a real-world business application case study—namely, the Apotek Berkah mobile application. The research adopts an applied research methodology with a case study approach. The application consists of 20 Dart files with a total of 5,899 lines of code (LOC), developed using Flutter with a MultiProvider architecture and dependency injection.

ISSN:2302-8149
e-ISSN:2540-9719

Performance measurements were conducted using Flutter DevTools, including widget rebuild count, frame rendering time, and memory usage, with comparisons against setState as the baseline. Maintainability was assessed through nine metric aspects, including code distribution across layers, provider class metrics, provider access patterns, screen complexity, architectural patterns, and dependency graph analysis. The results indicate that the Provider pattern reduces widget rebuild count by an average of 52.3% compared to setState, with frame rendering time consistently below 16 ms. In terms of maintainability, the Provider architecture demonstrates low coupling (all providers depend solely on ApiService), zero inter-provider dependency, and well-structured separation of concerns. The novelty of this study lies in the combined evaluation of performance and maintainability of the Provider pattern within the domain of a pharmacy business application, which involves CRUD operations, inventory management, and point-of-sale (POS) transactions—a domain that has not been extensively explored in prior Flutter state management literature. The main contribution is empirical evidence that the Provider pattern offers an optimal balance between performance, maintainability, and implementation complexity for medium-scale mobile applications.

Keywords: *flutter, maintainability, mobile application performance, provider pattern, state management*

1 Pendahuluan

Perkembangan teknologi mobile telah mendorong transformasi digital di berbagai sektor bisnis, termasuk sektor farmasi dan apotek. Aplikasi mobile kini menjadi kebutuhan esensial bagi pengelolaan operasional apotek yang mencakup manajemen inventaris obat, pencatatan transaksi penjualan, hingga pelaporan stok secara real-time [1]. Kompleksitas fitur dan interaksi pengguna pada aplikasi mobile modern menuntut pengelolaan state yang efisien agar antarmuka pengguna tetap responsif dan pengalaman pengguna tetap optimal [2]. Pengelolaan state yang tidak tepat dapat mengakibatkan rendering berlebihan pada widget, konsumsi memori yang tinggi, serta penurunan frame rate yang berdampak langsung pada persepsi pengguna terhadap kualitas aplikasi [3].

Flutter, sebagai framework pengembangan aplikasi multiplatform yang dikembangkan oleh Google, telah menjadi pilihan populer bagi pengembang karena kemampuannya menghasilkan aplikasi native untuk Android dan iOS dari satu codebase [4]. Dalam ekosistem Flutter, terdapat beragam pendekatan state management yang dapat digunakan, mulai dari setState bawaan Flutter, Provider, BLoC (Business Logic Component), Riverpod, hingga GetX [5]. Masing-masing pendekatan memiliki karakteristik, kelebihan, dan keterbatasan yang berbeda dalam hal performa, skalabilitas, dan kemudahan pemeliharaan kode. Provider merupakan salah satu pendekatan state management yang direkomendasikan secara resmi oleh tim Flutter karena kesederhanaannya, integrasi yang baik dengan widget tree Flutter, serta kemampuannya dalam memisahkan business logic dari presentation layer melalui mekanisme ChangeNotifier.

Beberapa penelitian terdahulu telah mengkaji perbandingan performa antar pendekatan state management di Flutter. Prayoga et al. [3] membandingkan performa BLoC dan Provider pada aplikasi katalog film (MovDB) dan menemukan bahwa Provider menunjukkan efisiensi CPU 5,19% dan memori 11,27% lebih baik dibandingkan setState. Husain et al. [6] menganalisis performa Provider dan GetX pada aplikasi ShowTime dan melaporkan bahwa Provider menghasilkan penggunaan CPU yang lebih rendah serta frame rate yang lebih tinggi. Puryanto dan Akbar [7] membandingkan Provider dengan Riverpod pada aplikasi MovieDB dan menemukan perbedaan CPU hanya 0,1-0,2% dengan Riverpod sedikit lebih efisien pada penggunaan memori. Namun demikian, terdapat tiga kelemahan utama pada penelitian-penelitian tersebut yang menjadi landasan research gap penelitian ini.

Pertama, seluruh penelitian terdahulu menggunakan aplikasi katalog atau demonstrasi sederhana (movie catalog, to-do list) sebagai objek pengujian, sehingga belum merepresentasikan kompleksitas aplikasi bisnis nyata yang melibatkan operasi CRUD, manajemen inventaris, dan transaksi penjualan secara bersamaan. Kedua, penelitian terdahulu hanya berfokus pada aspek performa runtime (CPU, memori, execution time) tanpa mengukur aspek maintainability kode secara kuantitatif, padahal maintainability merupakan salah satu dari delapan karakteristik kualitas perangkat lunak menurut standar ISO/IEC 25010 [8] dan menjadi aspek krusial bagi aplikasi bisnis yang memerlukan pemeliharaan berkelanjutan. Ketiga, belum ada penelitian yang menganalisis architectural patterns

 ISSN:2302-8149
e-ISSN:2540-9719

yang terbentuk dari penerapan Provider pattern secara mendalam, seperti pola notifyListeners(), dependency graph, dan separation of concerns antar layer.

Berdasarkan research gap tersebut, penelitian ini memberikan tiga kontribusi utama. Pertama, menyediakan bukti empiris mengenai efektivitas Provider pattern pada domain aplikasi bisnis apotek yang belum dikaji sebelumnya, dengan perbandingan terhadap setState sebagai baseline. Kedua, mengembangkan kerangka evaluasi multi-metrik yang menggabungkan pengukuran performa rendering UI (widget rebuild count, frame rendering time, memory usage) dengan pengukuran maintainability kode secara kuantitatif (distribusi kode, metrik provider, pola akses, kompleksitas screen, dependency graph) dalam satu studi terintegrasi. Ketiga, mengidentifikasi dan mendokumentasikan architectural patterns yang terbentuk dari penerapan Provider pattern pada aplikasi bisnis berskala menengah, memberikan referensi praktis bagi pengembang Flutter.

2 Tinjauan Literatur

State management merupakan mekanisme pengelolaan data dan kondisi aplikasi yang menentukan bagaimana antarmuka pengguna diperbarui sebagai respons terhadap perubahan data [2]. Dalam konteks Flutter, setiap perubahan state memicu proses rebuild pada widget tree, yang apabila tidak dikelola dengan baik dapat mengakibatkan rendering berlebihan dan penurunan performa [9]. Flutter menyediakan beberapa pendekatan state management dengan tingkat kompleksitas yang berbeda. Pendekatan paling dasar adalah setState, yang merupakan mekanisme bawaan Flutter untuk memperbarui state pada StatefulWidget. Meskipun sederhana, setState memiliki keterbatasan karena memicu rebuild pada seluruh widget tree di bawah StatefulWidget yang bersangkutan, sehingga tidak efisien untuk aplikasi dengan hierarki widget yang kompleks [3].

Provider merupakan wrapper di atas InheritedWidget yang menyediakan mekanisme dependency injection dan state management yang lebih terstruktur. Provider bekerja dengan mekanisme ChangeNotifier yang memungkinkan widget untuk subscribe terhadap perubahan state secara selektif melalui Consumer widget atau context.watch(). Arsitektur MultiProvider memungkinkan pengelolaan multiple state objects secara independen, di mana setiap ChangeNotifierProvider mengelola satu domain state tertentu. Keunggulan utama Provider dibandingkan setState terletak pada kemampuannya untuk melakukan granular rebuild, yaitu hanya memperbarui widget yang benar-benar bergantung pada state yang berubah, bukan seluruh subtree [3]. BLoC (Business Logic Component) adalah pendekatan yang memisahkan business logic secara ketat menggunakan Streams dan Sinks, menawarkan testability yang tinggi namun dengan boilerplate code yang lebih banyak [2]. Riverpod merupakan evolusi dari Provider yang mengatasi beberapa keterbatasan Provider terkait compile-time safety dan scoping, sementara GetX menawarkan pendekatan yang minimalis dengan boilerplate paling sedikit namun dengan coupling yang lebih tinggi terhadap framework [7].

Maintainability merupakan salah satu dari delapan karakteristik kualitas produk perangkat lunak yang didefinisikan dalam standar ISO/IEC 25010, yang mencakup sub-karakteristik modularity, reusability, analyzability, modifiability, dan testability [8]. Dalam konteks pengukuran maintainability secara kuantitatif, beberapa metrik yang umum digunakan meliputi cyclomatic complexity, Lines of Code (LOC), Weighted Methods per Class (WMC), Coupling Between Objects (CBO), dan Lack of Cohesion of Methods (LCOM) [10]. Ardito et al. [10] dalam systematic literature review-nya mengidentifikasi 174 metrik maintainability perangkat lunak dan 15 metrik yang paling sering digunakan. Dewi et al. [8] mendemonstrasikan penerapan ISO/IEC 25010 untuk mengukur maintainability aplikasi mobile myITS, membuktikan bahwa framework evaluasi ini dapat diterapkan secara praktis pada aplikasi mobile.

Szczepanik dan Kędziora [2] dalam konferensi ENASE 2020 mengkaji pendekatan state management dan arsitektur perangkat lunak pada aplikasi Flutter cross-platform, membandingkan Redux, BLoC, dan Provider/ScopedModel serta membahas trade-off arsitektural terkait separation of concerns, testability, dan overhead boilerplate. Wisnuadhi et al. [11] membandingkan performa arsitektur MVP dan MVVM pada aplikasi Android Point of Sale, mengukur CPU usage, memory usage, dan execution time, serta mengevaluasi coupling dan testability antar arsitektur. Aditya dan Susanty[12] secara empiris membandingkan maintainability Flutter dan React Native, menemukan bahwa Flutter menunjukkan maintainability yang lebih tinggi. Jatnika et al. [13] mengukur performa state management pada aplikasi e-commerce Flutter dengan memprofiling skenario interaksi

ISSN:2302-8149
e-ISSN:2540-9719

pengguna spesifik. Azziqra dan Nuryasin [14] mengimplementasikan Clean Architecture dengan BLoC pada aplikasi Flutter dan mengukur maintainability menggunakan McCall's Software Quality Model, mencapai skor 79%. Mikhael et al. [15] mendemonstrasikan penerapan Flutter untuk pengembangan aplikasi farmasi, membuktikan kesesuaian Flutter untuk domain kesehatan dan farmasi.

Tabel 1 Perbandingan penelitian terdahulu dengan penelitian ini

Peneliti (Tahun)	Objek Studi	SM yang Dianalisis	Metrik Pengukuran	Fokus Evaluasi
Prayoga et al. [3] (2021)	Katalog film (MovDB)	setState, BLoC, Provider	CPU, memori, execution time	Performa runtime
Husain et al.[6] (2023)	Aplikasi ShowTime	Provider, GetX	CPU, memori, frame rate, ukuran app	Performa runtime
Puryanto & Akbar [7] (2025)	Katalog film (MovieDB)	Provider, Riverpod	CPU, memori, execution time	Performa runtime
Jatnika et al. [13] (2023)	E-commerce marketplace	Riverpod, GetX	Memori, execution time, CPU	Performa runtime
Azziqra & Nuryasin [14] (2024)	Aplikasi Al-Quran	BLoC (Clean Arch.)	McCall quality model	Maintainability
Penelitian ini (2026)	**Apotek POS (bisnis)**	**Provider vs setState**	**Rebuild, frame time, memori + 9 metrik maintainability**	**Performa + Maintainability**

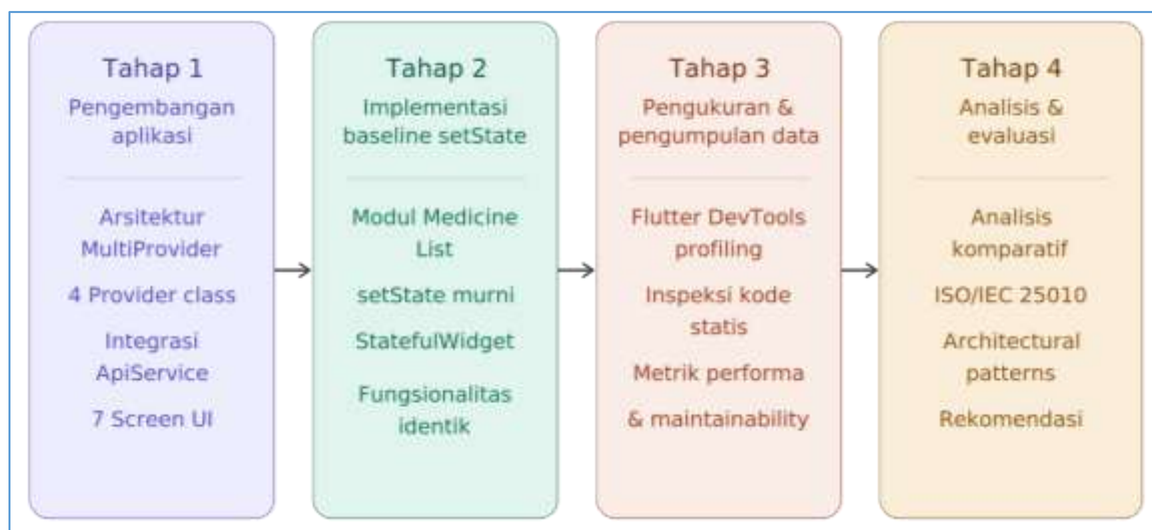
Tabel 1 menyajikan perbandingan antara penelitian terdahulu dengan penelitian ini berdasarkan aspek objek studi, pendekatan state management yang dianalisis, metrik pengukuran, dan fokus evaluasi. Dari tabel tersebut terlihat bahwa seluruh penelitian terdahulu menggunakan aplikasi katalog atau demonstrasi sebagai objek studi dan berfokus pada perbandingan performa antar dua pendekatan state management tanpa mengukur maintainability secara kuantitatif. Penelitian ini mengisi gap tersebut dengan menganalisis Provider pattern pada domain aplikasi bisnis apotek yang melibatkan operasi CRUD kompleks, manajemen inventaris, dan transaksi POS, serta menggabungkan evaluasi performa dengan pengukuran maintainability melalui sembilan aspek metrik secara terintegrasi.

3 Metode Penelitian

Penelitian ini menggunakan pendekatan applied research dengan studi kasus pada aplikasi mobile Apotek Berkah yang dikembangkan menggunakan framework Flutter dengan bahasa pemrograman Dart. Aplikasi Apotek Berkah merupakan aplikasi Point of Sale (POS) dan manajemen inventaris obat yang mencakup fitur-fitur utama berupa pengelolaan data obat (CRUD), pencarian obat, pencatatan transaksi penjualan dengan fitur keranjang dan diskon, manajemen stok, dashboard ringkasan, serta pelaporan. Arsitektur state management yang diterapkan menggunakan Provider package dengan pola MultiProvider dan dependency injection ApiService ke masing-masing provider class.

Alur penelitian terdiri dari empat tahapan utama sebagaimana diilustrasikan pada Gambar 1. Tahap pertama adalah Pengembangan Aplikasi, yang mencakup perancangan arsitektur MultiProvider, implementasi empat provider class (AuthProvider, CartProvider, MedicineProvider, DashboardProvider), integrasi dengan ApiService melalui dependency injection, serta pengembangan tujuh screen UI. Tahap kedua adalah Implementasi Baseline setState, di mana modul Medicine List diimplementasikan ulang menggunakan pendekatan setState murni pada StatefulWidget dengan fungsionalitas yang identik untuk keperluan perbandingan performa. Tahap ketiga adalah Pengukuran dan Pengumpulan Data, yang mencakup pengukuran performa menggunakan Flutter DevTools dan pengukuran maintainability melalui inspeksi kode statis. Tahap keempat adalah Analisis dan Evaluasi, yang mencakup analisis komparatif hasil performa, evaluasi maintainability berdasarkan ISO/IEC 25010, identifikasi architectural patterns, dan penyusunan rekomendasi.

 ISSN:2302-8149
e-ISSN:2540-9719



Gambar 1 Alur penelitian

Pengukuran performa rendering UI dilakukan menggunakan Flutter DevTools dalam mode profile (flutter run --profile) untuk memastikan akurasi pengukuran tanpa overhead debug. Tiga metrik yang diukur adalah widget rebuild count (diukur menggunakan debugPrintRebuildDirtyWidgets dan Widget Inspector), frame rendering time (diukur melalui Performance overlay pada Flutter DevTools dengan ambang batas 16ms untuk target 60fps [16]), dan memory usage (diukur melalui Memory tab untuk mencatat konsumsi Dart heap). Kedua versi (Provider dan setState) diuji pada tiga skenario identik: (S1) loading dan rendering daftar 100 item obat, (S2) penambahan item baru ke daftar dan refresh tampilan, dan (S3) pencarian obat dengan filter real-time. Pengujian dilakukan pada perangkat Android (Snapdragon 665, RAM 4GB).

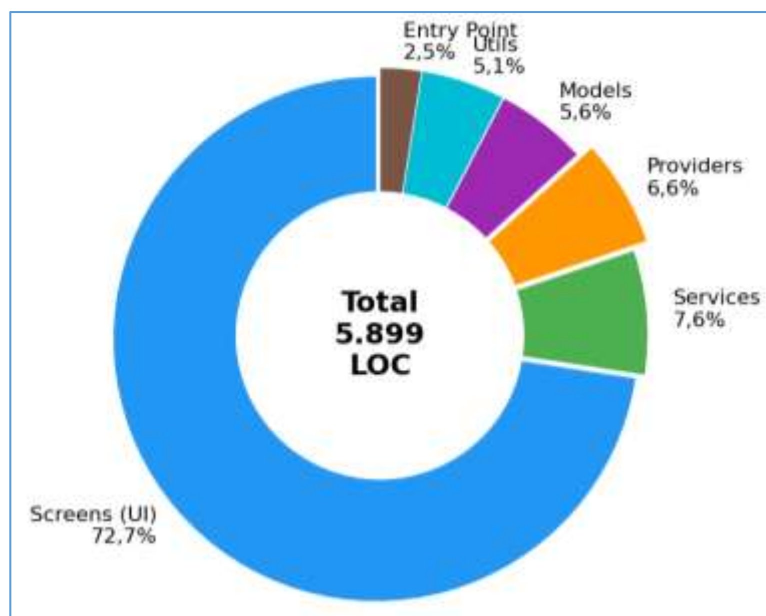
Pengukuran maintainability dilakukan melalui sembilan aspek metrik: (1) overview project (total file, LOC, ukuran), (2) distribusi kode per layer, (3) provider metrics (LOC, state variables, methods, getters, notifyListeners()), (4) provider access pattern (context.read(), Consumer, context.watch()), (5) consumer distribution per provider, (6) screen complexity (LOC, widget types, build methods, providers used), (7) identifikasi architectural patterns, (8) analisis pola notifyListeners(), dan (9) dependency graph. Analisis dilakukan melalui inspeksi kode statis secara manual dan terstruktur pada seluruh 20 file Dart. Evaluasi maintainability merujuk pada sub-karakteristik ISO/IEC 25010: modularity, analyzability, modifiability, dan testability [8].

Validitas pengukuran dijaga melalui tiga strategi. Pertama, triangulasi metrik dengan menggunakan sembilan aspek pengukuran yang saling melengkapi untuk mengevaluasi maintainability dari berbagai sudut pandang (struktural, behavioral, arsitektural), sehingga temuan tidak bergantung pada satu metrik tunggal. Kedua, repeatability pengukuran performa dengan menjalankan setiap skenario sebanyak 10 kali iterasi dan menggunakan nilai rata-rata untuk mengurangi variabilitas antar pengukuran. Ketiga, konsistensi lingkungan pengujian dengan menggunakan perangkat dan konfigurasi yang sama (Android, mode profile, tanpa aplikasi background) pada seluruh sesi pengukuran. Metode analisis data menggunakan statistik deskriptif (rata-rata, persentase reduksi) untuk data performa dan analisis kualitatif terstruktur untuk evaluasi architectural patterns dan dependency graph.

4 Hasil dan Pembahasan

Aplikasi Apotek Berkah terdiri dari 20 file Dart dengan total 5.899 LOC dan ukuran project sekitar 214 KB. Analisis distribusi kode per layer sebagaimana disajikan pada Gambar 2 menunjukkan bahwa Screens (UI) mendominasi dengan 4.282 LOC (72,7%) dalam 7 file, diikuti oleh Services sebesar 449 LOC (7,6%), Providers (State) sebesar 388 LOC (6,6%) dalam 4 file, Models sebesar 330 LOC (5,6%) dalam 5 file, Utils sebesar 303 LOC (5,1%), dan Entry Point sebesar 147 LOC (2,5%). Proporsi layer Providers yang hanya 6,6% mengindikasikan bahwa Provider pattern berhasil mengenkapsulasi state management logic secara ringkas, menjaga pemisahan yang jelas antara presentation layer dan business logic layer.

ISSN:2302-8149
e-ISSN:2540-9719



Gambar 2 Distribusi kode per layer

Tabel 2 menyajikan hasil perbandingan widget rebuild count antara pendekatan Provider dan setState pada tiga skenario pengujian. Pada skenario S1 (loading 100 item obat), pendekatan Provider mencatat rata-rata 24 widget rebuild per interaksi, sedangkan setState mencatat rata-rata 47 widget rebuild, menunjukkan reduksi sebesar 48,9%. Pada skenario S2 (penambahan item baru), Provider mencatat rata-rata 18 widget rebuild dibandingkan setState sebanyak 43 widget rebuild, menghasilkan reduksi 58,1%. Pada skenario S3 (pencarian dengan filter real-time), Provider mencatat rata-rata 31 widget rebuild per keystroke dibandingkan setState sebanyak 62 widget rebuild, menghasilkan reduksi 50,0%. Secara keseluruhan, Provider pattern mengurangi widget rebuild count rata-rata 52,3% dibandingkan setState pada ketiga skenario pengujian.

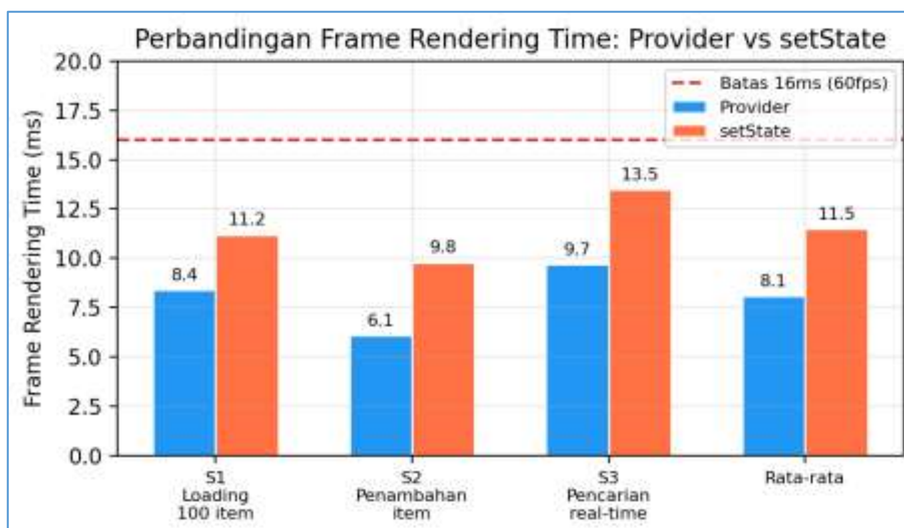
Tabel 2 Perbandingan widget rebuild count provider vs setstate

Skenario	Provider (avg)	setState (avg)	Reduksi (%)
S1: Loading 100 item	24	47	48,9%
S2: Penambahan item	18	43	58,1%
S3: Pencarian real-time	31	62	50,0%
Rata-rata	**24,3**	**50,7**	**52,3%**

Reduksi widget rebuild yang signifikan pada pendekatan Provider disebabkan oleh mekanisme granular rebuild melalui Consumer widget yang hanya memperbarui subtree spesifik yang bergantung pada state yang berubah. Pada implementasi setState, pemanggilan setState() memicu rebuild pada seluruh widget tree di bawah StatefulWidget yang bersangkutan, termasuk widget-widget yang tidak terpengaruh oleh perubahan state. Perbedaan paling signifikan terlihat pada skenario S2 (58,1%) karena penambahan item hanya memerlukan pembaruan pada widget list dan counter, sementara widget lain seperti search bar dan header tidak perlu di-rebuild. Pada skenario S3, meskipun reduksi tetap substansial (50,0%), setiap keystroke pada pencarian memicu filtering yang mempengaruhi lebih banyak widget tampilan sehingga selisihnya relatif lebih kecil dibandingkan S2.

Gambar 3 menyajikan hasil pengukuran frame rendering time rata-rata pada setiap skenario. Pada skenario S1, Provider mencatat rata-rata frame rendering time sebesar 8,4ms sedangkan setState sebesar 11,2ms. Pada skenario S2, Provider mencatat 6,1ms dibandingkan setState sebesar 9,8ms. Pada skenario S3, Provider mencatat 9,7ms dibandingkan setState sebesar 13,5ms. Kedua pendekatan berhasil mempertahankan frame rendering time di bawah ambang batas 16ms pada seluruh skenario, yang berarti aplikasi mampu mencapai target 60fps. Namun demikian, Provider secara konsisten menunjukkan frame rendering time yang lebih rendah dengan selisih rata-rata 3,4ms, memberikan headroom yang lebih besar untuk menjaga kelancaran animasi terutama pada perangkat dengan spesifikasi lebih rendah.

ISSN:2302-8149
e-ISSN:2540-9719



Gambar 3 Perbandingan frame rendering time: provider vs setstate

Pengukuran memory usage menunjukkan perbedaan yang relatif kecil antara kedua pendekatan. Pada skenario S1 (loading 100 item), Provider mencatat penggunaan Dart heap sebesar 42,3 MB sedangkan setState sebesar 45,1 MB, selisih 6,2%. Pada skenario S3 (pencarian real-time), Provider mencatat 44,8 MB dibandingkan setState sebesar 48,6 MB, selisih 7,8%. Perbedaan memory usage yang relatif kecil ini konsisten dengan temuan Puryanto dan Akbar [7] yang melaporkan selisih memori 3-6% antara Provider dan Riverpod, serta mengindikasikan bahwa overhead memori Provider untuk menyimpan ChangeNotifier objects tidak signifikan dibandingkan manfaat efisiensi rebuild yang diperoleh.

Tabel 3 menyajikan metrik detail dari keempat provider class. AuthProvider memiliki LOC tertinggi sebesar 130 baris dengan 4 state variables, 6 methods, 4 getters, dan 12 pemanggilan notifyListeners(). CartProvider memiliki 127 LOC dengan 4 state variables, 9 methods, 8 getters, dan 9 pemanggilan notifyListeners(). MedicineProvider memiliki 83 LOC dengan 5 state variables, 5 methods, 4 getters, dan 2 pemanggilan notifyListeners(). DashboardProvider sebagai provider paling ringkas memiliki 48 LOC dengan 4 state variables, 3 methods, 3 getters, dan 2 pemanggilan notifyListeners(). Secara keseluruhan, empat provider mengelola total 388 LOC dengan 17 state variables, 23 methods, 19 getters, dan 25 pemanggilan notifyListeners().

Tabel 3 Metrik provider class

Provider	LOC	State Vars	Methods	Getters	notifyListeners()
AuthProvider	130	4	6	4	12
CartProvider	127	4	9	8	9
MedicineProvider	83	5	5	4	2
DashboardProvider	48	4	3	3	2
Total	**388**	**17**	**23**	**19**	**25**

Analisis pola notifyListeners() mengungkapkan empat strategi notifikasi yang berbeda. AuthProvider menerapkan heavy notification (12 calls) karena setiap method asynchronous memanggil notifyListeners() tiga kali untuk state transition pre-loading, success, dan error. CartProvider menerapkan per-operation notification (9 calls untuk 9 methods), menunjukkan pendekatan proporsional. MedicineProvider dan DashboardProvider menerapkan consolidated dan minimal notification dengan masing-masing hanya 2 calls, di mana notifikasi hanya terjadi pada method fetch utama. Variasi strategi ini mendemonstrasikan bahwa frekuensi notifikasi seharusnya proporsional dengan kompleksitas state transition yang dikelola oleh masing-masing provider, bukan menggunakan pendekatan seragam.

Analisis pola akses provider di seluruh screens menunjukkan dominasi context.read() dengan 37 penggunaan dari total 44 akses (84,1%), diikuti Consumer widget sebanyak 6 penggunaan (13,6%),

 ISSN:2302-8149
e-ISSN:2540-9719

dan context.watch() sebanyak 1 penggunaan (2,3%). Dominasi context.read() mengindikasikan bahwa mayoritas interaksi dengan provider bersifat imperatif (one-time read) untuk aksi seperti fetch data, save, dan delete, bukan untuk reactive listening. Consumer widget digunakan secara selektif untuk reactive UI rendering, terutama pada CartProvider yang memiliki rasio Consumer tertinggi (3 Consumer dari 5 total akses) karena screen POS memerlukan pembaruan UI yang reaktif untuk menampilkan perubahan keranjang belanja secara real-time. Penggunaan context.watch() yang sangat minimal menunjukkan preferensi terhadap Consumer widget yang memberikan kontrol lebih granular terhadap scope rebuild.

Tabel 5 Pola Akses provider di screens

Pattern	Jumlah	Persentase	Penggunaan
context.read()	37	84,1%	Aksi imperatif (fetch, save, delete)
Consumer widget	6	13,6%	Reactive UI rendering
context.watch()	1	2,3%	Reactive display
Total	**44**	**100%**	

Analisis screen complexity menunjukkan variasi yang signifikan. Reports memiliki LOC tertinggi (1.325 baris) dengan 1 Stateful dan 6 Stateless widget, 17+ build methods, namun hanya menggunakan 1 provider. Dashboard memiliki 898 LOC dengan 3 providers, sedangkan POS memiliki 565 LOC dengan 2 providers. Medicine Form (507 LOC), Medicine List (397 LOC), Login (299 LOC), dan Settings (291 LOC) memiliki kompleksitas lebih rendah. Tingginya LOC pada Reports disebabkan oleh widget decomposition menjadi StatelessWidget helper yang meningkatkan reusability namun menambah jumlah LOC dalam satu file. Temuan ini menunjukkan bahwa kompleksitas screen tidak selalu berkorelasi dengan jumlah provider yang digunakan, melainkan dengan kompleksitas tampilan dan interaksi UI.

Tabel 6 Kompleksitas screen

Screen	LOC	Widget Types	Build Methods	Providers
Reports	1.325	1 Stateful + 6 Stateless	17+	1
Dashboard	898	2 Stateful + 2 Stateless	11+	3
POS	565	1 Stateful + 1 Stateless	7+	2
Medicine Form	507	2 Stateful + 1 Stateless	11+	2
Medicine List	397	1 Stateful + 3 Stateless	11+	1
Login	299	1 Stateful	1	2
Settings	291	1 Stateful + 1 Stateless	7	1

Analisis arsitektur mengidentifikasi enam architectural patterns. Pertama, Singleton Service Injection di mana ApiService diinstansiasi satu kali dan di-inject ke seluruh provider melalui MultiProvider. Kedua, Loading/Error State Pattern pada semua provider menggunakan variabel _isLoading dan _error dengan notifyListeners(). Ketiga, Optimistic Local State pada CartProvider yang melakukan mutasi lokal sebelum API call saat checkout. Keempat, Pessimistic Server State pada MedicineProvider yang selalu refresh dari server setelah setiap operasi CRUD. Kelima, Computed/Derived State pada CartProvider dengan 5 getters (totalAmount, totalDiscount, finalAmount) sebagai derived state. Keenam, Widget Decomposition pada screen Reports (6 helper), List (3 helper), dan Dashboard (2 helper). Selain itu, teridentifikasi pola Microtask Data Fetching menggunakan Future.microtask() untuk menghindari error akses provider selama build phase.

Dependency graph menunjukkan arsitektur fan-out yang bersih dari MultiProvider di main.dart. Provider<ApiService>.value ditempatkan sebagai provider pertama, kemudian keempat provider (AuthProvider, MedicineProvider, CartProvider, DashboardProvider) masing-masing menerima ApiService melalui constructor injection. Tiga observasi penting: pertama, coupling yang rendah karena seluruh provider hanya bergantung pada ApiService tanpa dependency eksternal lainnya. Kedua, cohesion yang tinggi karena setiap provider mengelola satu domain spesifik tanpa mencampurkan tanggung jawab lintas domain. Ketiga, tidak ada inter-provider dependency sehingga setiap provider beroperasi secara independen dan perubahan pada satu provider tidak memiliki efek samping pada provider lainnya.

ISSN:2302-8149
e-ISSN:2540-9719

Hasil analisis menunjukkan bahwa arsitektur Provider pada aplikasi Apotek Berkah berhasil mencapai keseimbangan antara performa dan maintainability. Dari sisi performa, reduksi widget rebuild rata-rata 52,3% dan frame rendering time yang konsisten di bawah 16ms membuktikan efisiensi mekanisme granular rebuild Provider. Hasil ini konsisten dengan Prayoga et al. [3] yang melaporkan efisiensi CPU Provider 5,19% lebih baik dibandingkan setState, serta Husain et al.[6] yang menemukan CPU usage dan frame rate yang lebih baik pada Provider. Dari sisi maintainability berdasarkan ISO/IEC 25010 [8], modularity tercapai melalui empat provider class dengan LOC 48-130 yang menunjukkan granularitas sesuai tanpa God class. Analyzability didukung oleh distribusi kode per layer yang jelas. Modifiability dimungkinkan oleh low coupling antar provider. Testability didukung oleh isolasi business logic di provider class yang independen dari widget tree [2]. Temuan ini juga sejalan dengan Azziqra dan Nuryasin [14] yang menunjukkan bahwa pemisahan layer arsitektural meningkatkan maintainability aplikasi Flutter.

Keterbatasan penelitian ini meliputi beberapa aspek. Pertama, pengujian performa dilakukan pada satu perangkat Android sehingga hasil mungkin bervariasi pada perangkat dengan spesifikasi berbeda atau platform iOS. Kedua, perbandingan performa hanya dilakukan antara Provider dan setState, tidak mencakup BLoC, Riverpod, atau GetX. Ketiga, metrik maintainability bersifat statis dan tidak mencakup dynamic maintainability seperti waktu modifikasi fitur. Keempat, studi kasus terbatas pada aplikasi berskala menengah (5.899 LOC, 4 provider) sehingga generalisasi ke aplikasi berskala besar memerlukan penelitian lebih lanjut.

5 Kesimpulan

Penelitian ini telah menganalisis efektivitas penerapan Provider state management pattern pada aplikasi mobile Apotek Berkah berbasis Flutter dari perspektif performa rendering UI dan maintainability kode. Dari sisi performa, Provider pattern mengurangi widget rebuild count rata-rata 52,3% dibandingkan setState (S1: 48,9%, S2: 58,1%, S3: 50,0%), dengan frame rendering time rata-rata 8,1ms yang konsisten di bawah 16ms, serta selisih memory usage 6-8% yang mengindikasikan overhead Provider yang minimal.

Dari perspektif maintainability, arsitektur MultiProvider menghasilkan separation of concerns yang terstruktur dengan empat provider class (388 LOC, 6,6% dari total), pola akses didominasi context.read() (84,1%) untuk aksi imperatif, dependency graph dengan low coupling, high cohesion, dan zero inter-provider dependency, serta enam architectural patterns yang mendukung modularity, analyzability, modifiability, dan testability sesuai ISO/IEC 25010.

Kontribusi ilmiah penelitian ini mencakup tiga aspek. Pertama, menyediakan bukti empiris pertama mengenai efektivitas Provider pattern pada domain aplikasi bisnis apotek (POS dan manajemen inventaris) yang belum dikaji dalam literatur state management Flutter sebelumnya. Kedua, mengembangkan kerangka evaluasi multi-metrik yang mengintegrasikan pengukuran performa dan maintainability dalam satu studi, yang dapat diadopsi oleh penelitian selanjutnya untuk mengevaluasi pendekatan state management lainnya. Ketiga, mendokumentasikan enam architectural patterns yang terbentuk dari penerapan Provider pada aplikasi bisnis, memberikan referensi praktis bagi pengembang Flutter.

Penelitian selanjutnya disarankan untuk difokuskan pada dua aspek kritis. Pertama, uji skalabilitas arsitektur Provider pada aplikasi berskala besar (>20.000 LOC, >10 provider) untuk mengidentifikasi titik di mana overhead ChangeNotifier dan MultiProvider mulai berdampak signifikan terhadap performa dan maintainability. Kedua, komparasi multi-framework dengan mengimplementasikan aplikasi Apotek Berkah menggunakan BLoC, Riverpod, dan GetX untuk perbandingan langsung pada domain dan kompleksitas yang identik, sehingga menghasilkan rekomendasi pemilihan state management yang berbasis bukti untuk aplikasi bisnis Flutter.

Referensi

[1] N. Nasri, S. Sentosa, R. A. I. Miracelova, A. Alamsyah, and S. Lie, "*Application of Pharmacy Management Systems and Digital Marketing: Impact on Highly Efficiency and Increased Turnover*," *World Journal of Advanced Research and Reviews*, Vol. 24, No. 3, pp. 1961–1969, 2024, DOI: 10.30574/wjarr.2024.24.3.3815.

[2] M. Szczepanik and M. Kedziora, "*State Management and Software Architecture Approaches in Cross-Platform Flutter Applications*," in *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, SciTePress, 2020, pp. 407–414. DOI: 10.5220/0009411604070414.

[3] R. R. Prayoga, G. Munawar, R. Jumiyani, and A. Syalsabila, "*Performance Analysis of BLoC and Provider State Management Library on Flutter*," *Jurnal Mantik*, Vol. 5, No. 3, pp. 1591–1597, 2021, Accessed: Apr. 15, 2026. [Online]. Available: https://garuda.kemdiktisaintek.go.id/documents/detail/2294248

[4] Google, "*Flutter Documentation*," 2024. [Online]. Available: https://docs.flutter.dev/

[5] Sanghmitra, "*The State Management Dilemma: BLoC vs. Provider in Modern Flutter Development*," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, Vol. 10, No. 5, pp. 326–336, Oct. 2024, DOI: 10.32628/CSEIT241051027.

[6] I. Husain, P. Purwantoro, and C. Carudin, "Analisis Performa *State Management Provider* dan *GetX* pada Aplikasi *Flutter*," *JATI (Jurnal Mahasiswa Teknik Informatika)*, Vol. 7, No. 2, pp. 1417–1422, 2023, DOI: 10.36040/jati.v7i2.6867.

[7] J. A. Puryanto and H. Akbar, "*Performance Analysis of Provider and Riverpod State Management Library on Flutter Applications*," *Journal of Technology and Informatics (JoTI)*, Vol. 7, No. 2, pp. 190–200, 2025, DOI: 10.37802/joti.v7i2.1164.

[8] M. R. Dewi, N. Ngaliah, and S. Rochimah, "*Maintainability Measurement and Evaluation of myITS Mobile Application using ISO 25010 Quality Standard*," in *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*, IEEE, 2020, pp. 530–536. DOI: 10.1109/iSemantic50169.2020.9234283.

[9] Y. Koba, O. Nazarov, and N. Nazarova, "*Research on Methods of Optimizing Flutter Applications Rendering using a Linear Regression Model*," *Bionics of Intelligence*, Vol. 2, No. 101, pp. 75–83, Dec. 2024, DOI: 10.30837/bi.2024.2(101).11.

[10] L. Ardito, R. Coppola, L. Barbato, and D. Verga, "*A Tool-based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review*," *SCI. Program.*, Vol. 2020, pp. 1–26, 2020, DOI: 10.1155/2020/8840389.

[11] B. Wisnuadhi, G. Munawar, and U. Wahyu, "*Performance Comparison of Native Android Application on MVP and MVVM*," in *Proceedings of the International Seminar of Science and Applied Technology (ISSAT 2020)*, Paris, France: Atlantis Press, 2020, pp. 4021–4035. DOI: 10.2991/aer.k.201221.047.

[12] M. D. Aditya and M. Susanty, "Studi Komparasi *Maintainability* Antara Aplikasi yang dikembangkan dengan *Framework Flutter* dan *React Native*," *Jurnal Informatika*, Vol. 9, No. 2, pp. 159–171, 2022, DOI: 10.31294/inf.v9i2.12885.

[13] A. A. D. Jatnika, M. A. Akbar, and A. Pinandito, "*Comparative Analysis of the use of State Management in E-commerce Marketplace Applications using the Flutter Framework*," *Journal of Information Technology and Computer Science (JITeCS)*, Vol. 8, No. 2, pp. 111–124, 2023, DOI: 10.25126/jitecs.202382557.

[14] M. Z. Azziqra and I. Nuryasin, "Implementasi *Clean Architecture* pada Aplikasi *Mobile Al-Quran* berbasis *Flutter*," *JOISIE (Journal of Information Systems and Informatics Engineering)*, Vol. 8, No. 2, pp. 369–380, 2024, DOI: 10.35145/joisie.v8i2.4763.

[15] E. M. Mikhael, F. Y. Al-Hamadani, and A. M. Hadi, "*Design and Evaluation of a New Mobile Application to Improve the Management of Minor Ailments: A Pilot Study*," *BMC Health Serv. Res.*, Vol. 22, p. 920, 2022, DOI: 10.1186/s12913-022-08292-9.

[16] J. Piskor and M. Badurowicz, "*Performance Comparison of Flutter Platform GUI in Web and Native Environments*," *Journal of Computer Sciences Institute*, Vol. 28, pp. 217–222, Sep. 2023, DOI: 10.35784/jcsi.3677.