# Perbandingan Performa Pola Arsitektur *Model-View-Presenter* (MVP) dengan *Model-View-ViewModel* (MVVM)

Allycia Joan Micheline
*Informatika*
*Universitas Pradita*
Tangerang, Indonesia
allycia.joan@student.pradita.ac.id

Bryant Nehemia Natanael
*Informatika*
*Universitas Pradita*
Tangerang, Indonesia
bryant.nehemia@student.pradita.ac.id

Sibgah Rabbani Kusuma
*Informatika*
*Universitas Pradita*
Tangerang, Indonesia
sibgah.rabbani@student.pradita.ac.id

Noven Austin
*Informatika*
*Universitas Pradita*
Tangerang, Indonesia
noven.austin@student.pradita.ac.id

***Abstract*—abstrak**
***Index Terms*—component, formatting, style, styling, insert**

## I. Pendahuluan

Dalam dunia pengembangan perangkat lunak, pemilihan pola arsitektur sangatlah penting untuk memastikan keberhasilan proyek dan kualitas aplikasi yang dihasilkan. Dua pola arsitektur yang sering digunakan pengembangan aplikasi berbasis Android adalah *Model-View-Presenter* (MVP) dan *Model-View-ViewModel* (MVVM).

MVP dan MVVM memisahkan aplikasi menjadi tiga komponen utama, yaitu *Model* (menyimpan data dan logika bisnis), *View* (menampilkan antarmuka pengguna), dan *Presenter* (menghubungkan *Model* dan *View* serta mengatur logika pengontrol) untuk MVP [1], sedangkan MVVM tidak menggunakan *Presenter* namun *ViewModel* yang bertanggung jawab untuk memisahkan logika presentasi dari *Model* dan *View*. MVVM sering digunakan dalam pengembangan aplikasi berbasis *data binding*[1], terutama pada platform seperti Android dan iOS.

Meskipun MVP dan MVVM dapat memisahkan tanggung jawab antara *model, view*, dan logika pengontrol (*viewmodel*), kedua pola ini memiliki perbedaan dalam cara implementasinya. Permasalahan yang sering muncul adalah pemilihan pola yang tepat untuk aplikasi tertentu berdasarkan performa seperti responsifitas, kecepatan, dan penggunaan sumber daya, penggunaan memori, penggunaan CPU, stabilitas, fleksibilitas, kemudahan untuk diuji, dan pemeliharaan dalam jangka panjang.

Tujuan dari penelitian ini adalah untuk membandingkan performa kecepatan dan memori antara pola arsitektur MVP dan MVVM. Melalui penelitian ini, diharapkan dapat memberikan pemahaman yang lebih baik mengenai kelebihan dan kekurangan dari kedua pola arsitektur dan memberikan pemahaman yang lebih mendalam mengenai perbandingan performa antara pola arsitektur MVP dan MVVM.

## II. Kajian Terkait

### A. Pola Arsitektur MVP

MVP adalah pola arsitektur dalam pembuatan software untuk berbagai platform, seperti desktop, website, dan mobile. Bahasa pemrograman seperti Java, C, Swift, dan C++ dapat digunakan dalam membuat pola arsitektur MVP [2]. MVP merupakan evolusi dari pola arsitektur *Model-View-Controller* (MVC), yang lebih dahulu dikenal dan masih banyak digunakan pada saat ini [1], [3].

Pola arsitektur MVP terdiri dari tiga komponen utama, yaitu *Model, View* dan *Presenter*. *Model* menggambarkan komponen yang menangani data dan logika bisnis dalam sebuah aplikasi, serupa dengan fungsi model dalam MVC. Komponen ini bertugas mengolah, menyimpan, dan mengatur data, serta mengimplementasikan segala aturan bisnis yang diperlukan. Dalam struktur ini, model beroperasi secara independen dan tidak berinteraksi langsung dengan *view* atau *presenter*. *View* mewakili antarmuka pengguna dan lapisan presentasi aplikasi. Seperti tampilan di MVC, fungsi utamanya adalah menampilkan data yang diambil dari model. Namun, di MVP, tampilannya lebih pasif dan bergantung pada presenter untuk pembaruan dan penanganan input pengguna. Tampilan hanya berkomunikasi dengan presenter dan bukan dengan model. *Presenter* bertindak sebagai jembatan antara model dan tampilan, mengambil beberapa tanggung jawab pengontrol di MVC. *Presenter* mengambil data dari model dan memperbarui tampilan, memastikan presentasi data yang benar [1].

Salah satu keuntungan utama MVP adalah peningkatan pemisahan kekhawatiran antara tampilan (*View*) dan *Model*. Dengan pemisahan yang jelas antara komponen-komponen ini, pengembang dapat mengelola logika bisnis (*Model*) secara

[1]mengikat data antara *model* dan *view* yang disinkronkan setiap terjadinya perubahan

terpisah dari tampilan pengguna *View*. Hal ini meningkatkan keterbacaan dan pemeliharaan kode [4].

Selain itu, penggunaan *Presenter* dalam pola MVP memfasilitasi kemampuan pengujian dan modularitas yang lebih baik. *Presenter* bertindak sebagai perantara antara *View* dan *Model*, sehingga logika aplikasi dapat diuji secara terpisah dari antarmuka pengguna. Hal ini juga memungkinkan penggantian atau modifikasi komponen-komponen tanpa mempengaruhi bagian lainnya [4].

MVP juga memiliki kemampuan untuk mengatasi aplikasi dengan persyaratan keadaan atau interaksi yang kompleks. Dengan pola MVP, pengembang dapat membagi tugas dan tanggung jawab secara terstruktur antara *View, Model*, dan *Presenter*, sehingga meningkatkan keteraturan dan stabilitas aplikasi [4].

Walaupun pola MVP menawarkan sejumlah keuntungan, pola MVP juga memiliki beberapa kerugian yang perlu ditimbangkan. Salah satu kerugian utama adalah kemungkinan terbentuknya basis kode yang lebih besar dibandingkan pola arsitektur lainnya. Hal ini disebabkan oleh *Presenter* yang bertanggung jawab atas logika aplikasi, yang dapat mengakibatkan kebutuhan akan lebih banyak kode *boilerplate*[2] untuk menghubungkan *Presenter* dengan *View* dan *Model* [4].

Selain itu, MVP juga memiliki potensi *overhead*[3] dalam komunikasi antar komponen. *Presenter* harus mengelola interaksi antara *View* dan *Model*, yang dapat menambah kompleksitas dan mengurangi efisiensi dalam komunikasi antar bagian aplikasi. Hal ini dapat mempengaruhi kinerja aplikasi secara keseluruhan [4].

Secara keseluruhan, pola arsitektur MVP memberikan landasan yang kuat bagi pengembangan aplikasi yang terstruktur, mudah diuji, dan mudah dipelihara. Namun, pola MVP lebih kompleks dibandingkan pola-pola arsitektur lainnya.

### *B. Pola Arsitektur MVVM*

Pola MVVM pemisahan antarmuka pengguna (UI) dan logika bisnis dengan lancar dalam aplikasi modern [3], [5]. Aplikasi lebih mudah untuk diuji, dipelihara, dan dikembangkan ketika logika aplikasi dan antarmuka pengguna dipisahkan dengan jelas. Hal ini membantu memecahkan banyak tantangan pembangunan. Selain itu, hal ini dapat sangat meningkatkan kemungkinan penggunaan kembali kode dan menyediakan komunikasi yang lebih mudah antara perancang dan pengembang UI saat mereka mengerjakan komponen program yang berbeda [6].

MVVM memisahkan perangkat lunak menjadi tiga komponen utama, yaitu *Model, View* dan *ViewModel*. *Model* mewakili struktur data dan logika bisnis dari aplikasi. *Model* berisi data yang digunakan oleh aplikasi bisnis yang diperlukan untuk mengubah data serta data yang digunakan aplikasi. Antarmuka pengguna dan cara data ditampilkan tidak diketahui oleh *model*. *View* adalah bagian antarmuka pengguna yang terlihat oleh pengguna. *View* menampilkan data model dan menerima input pengguna. Di lingkungan web, *view* mungkin berupa halaman HTML atau komponen UI lainnya yang ditampilkan kepada pengguna. *ViewModel* berfungsi sebagai perantara antara *model* dan *view*. Ini mempersiapkan data *model* agar sesuai dengan persyaratan presentasi tampilan. *ViewModel* juga menangani interaksi antara *view* dan *model*. Secara kontekstual, *ViewModel* mengelola logika tampilan, yang mencakup tindakan seperti validasi input, penanganan kejadian UI, dan pengelolaan status [7].

Salah satu keuntungan dari pola MVVM adalah kemudahan dalam pemeliharaan kode. MVVM memungkinkan pengembang untuk merilis versi terbaru aplikasi dengan fitur baru secara berkala tanpa mengganggu keseluruhan kode aplikasi. Hal ini membuat proses pemeliharaan aplikasi menjadi lebih mudah dan terorganisir [7].

Selain itu, MVVM memungkinkan pengembang untuk mengganti atau menambahkan kode baru ke dalam program dengan lebih fleksibel. Penggunaan *ViewModel* sebagai perantara antara *View* dan *Model* memungkinkan pengembang untuk memisahkan logika *Model* dan *View* sehingga memfasilitasi perubahan dan penambahan fungsionalitas dengan lebih efisien [6].

Keuntungan lain dari MVVM adalah kemampuan untuk melakukan pengujian unit secara terpisah untuk *ViewModel* dan *Model*, tanpa perlu menggunakan *View*. Hal ini memungkinkan pengembang untuk menjalankan pengujian fungsionalitas model tampilan dengan cara yang sama seperti *View*, bahkan dapat meningkatkan kualitas dan ketahanan aplikasi [6].

Pola MVVM juga memungkinkan desainer dan pengembang aplikasi untuk bekerja secara independen dan bersamaan pada komponen. Desainer dapat berfokus pada desain tampilan yang estetis, sementara pengembang dapat bekerja pada model tampilan dan komponen model secara terpisah. Hal ini membuat kolaborasi yang lebih baik antara tim, meningkatkan efisiensi, dan kualitas pengembangan [6].

Di sisi lain, pola MVVM tidak terlalu cocok untuk proyek kecil atau aplikasi dengan tampilan yang sederhana. Pola ini dianggap berlebihan untuk aplikasi dengan kompleksitas yang rendah, sehingga penggunaanya tidak efisien dalam konteks tersebut [8].

Pola MVVM juga memiliki kompleksitas dalam *data binding*. Meskipun *data binding* adalah salah satu fitur yang kuat dalam MVVM, kompleksitasnya dapat membuat pengembang kesulitan dalam melakukan *debugging*. Menemukan dan memperbaiki bug atau error pada aplikasi terkait dengan *data binding* menjadi lebih sulit karena keterkaitan yang kompleks antara komponen-komponen MVVM [8].

Secara keseluruhan, MVVM dapat memberikan manfaat signifikan dalam pemeliharaan dan pengembangan aplikasi namun tidak cocok untuk proyek kecil atau aplikasi sederhana.

### *C. Studi Literatur*

Penelitian "*Performance Comparison of Native Android Application on MVP and MVVM*" [9] membandingkan performa antara pola arsitektur MVP dan MVVM dalam aplikasi Android. Performa diukur dari tiga aspek, yaitu penggunaan CPU, penggunaan memori, dan waktu eksekusi. Eksperimen dilakukan dengan menjalankan MVP dan MVVM pada

[2]kode yang digunakan berulang-ulang tanpa perubahan yang signifikan
[3]beban tambahan dalam proses komunikasi atau eksekusi

perangkat Android. Setiap pengujian dimonitor menggunakan "Snapdragon Profiler" dan hasil tersebut akan dieksport menjadi CSV. Pengujian dilakukan berdasarkan dua skenario, yaitu uji kasus dan volume data. Setiap skenario dilakukan sebanyak 5 kali dan hasil dari pengujian tersebut akan dirata-ratakan. Berdasarkan pengujian yang dilakukan, penggunaan CPU pada MVVM lebih rendah dengan perbedaan rata-rata 0,55%, waktu eksekusi MVVM lebih cepat dengan perbedaan rata-rata 126,21 ms, dan penggunaan memori pada MVP lebih rendan dengan perbedaan rata-rata sebesar 0,92 Mb. Dari eksperimen tersebut, dapat disimpulkan bahwa MVVM memiliki performa yang lebih baik dalam penggunaan CPU dan waktu eksekusi, sedangkan MVP memiliki performa yang lebih baik dalam penggunaan memori.

Penelitian "Analisis Perbandingan Implementasi Clean Architecture Menggunakan Design Pattern MVP, MVI, Dan MVVM Pada Pengembangan Aplikasi Android Native" [10] melakukan perbandingan modifiabilitas, testabilitas, dan performa berdasarkan skenario tertentu. Berdasarkan aspek yang telah diuji, MVI unggul dalam hal testabilitas, MVVM dalam hal modifiabilitas, dan MVP dalam hal performa.

Penelitian "*Analysis of Architectural Patterns for Android Development*" [11] membandingkan tentang lima pola arsitektur yaitu MVC, MVP, MVVM, Viper, dan *Clean Architecture*. Penelitian ini membandingkan kemampuan pengujian, pemeliharaan, dan penggunaan kembali. MVP memudahkan pengujian unit dan pemisahan yang jelas antara komponen-komponen, sedangkan MVVM menawarkan manajemen UI yang lebih baik, namun memerlukan banyak kode. Pada sisi lain, semakin banyak fungsionalitas yang ditambahkan, membuat MVC lebih susah dikenali.

Penelitian "*REVIEW OF IOS ARCHITECTURAL PATTERN FOR TESTABILITY, MODIFIABILITY, AND PERFORMANCE QUALITY*" [12] memperkenalkan pentingnya pemilihan pola arsitektur yang tepat untuk pengembangan aplikasi iOS. Pengujian pola arsitektur berdasarkan kualitas uji (*testability*), kemampuan untuk dimodifikasi (*modifiability*), dan performa aplikasi yang optimal. Pengujian kualitas uji berdasarkan tiga parameter yaitu uji global (*globat test effort*), kemampuan pengendalian (*controllability*), dan kemampuan observasi (*observability*). Hasil dari pengujian ini adalah MVVM memiliki baris kode yang lebih sedikit daripada pola arsitektur lainnya karena *data binding* di *ViewModel*. Waktu pengujian paling cepat juga dimiliki oleh MVVM dengan kecepatan 19.6 detik dibanding MVP 20.9 detik. Pada kemampuan untuk dimodifikasi, MVVM memiliki kemampuan kohesi terbaik degan 0,55 *procedural* dibanding MVP yaitu 0,3809 dan merupakan terendah diantara empat pola arsitektur (MVP, MVC, MVVM, dan VIPER). Pada uji performa, dua pengujian dilakukan yaitu penggunaan memori dan CPU. MVVM menggunakan memori paling sedikit yaitu 21,19275 Mb dibanding MVP yaitu 22,9645 Mb. Sedangkan dalam penggunaan CPU, MVVM menggunakan CPU kedua paling banyak yaitu 11,615%. Berdasarkan pengujian yang telah dilakukan, MVVM dan VIPER merupakan dua pola arsitektur terbaik.

## III. Metodologi

## IV. Hasil dan Pembahasan

## V. Kesimpulan

## References


[1] Dan Dan Li and Xiao Yan Liu. Research on mvp design pattern modeling based on mda. *Procedia Computer Science*, 166:51–56, 2020. Proceedings of the 3rd International Conference on Mechatronics and Intelligent Robotics (ICMIR-2019).

[2] Muhammad Fauzi Dwikurnia, Monterico Adrian, and Shinta Yulia Puspitasari. Optimasi kinerja aplikasi mobile berbasis flutter menggunakan pola arsitektur mvp (model-view-presenter) dan state management getx. *Jurnal Tugas AKhir Fakultas Informatika*, Aug 2023.

[3] Sirojiddin Komolov, Gcinizwe Dlamini, Swati Megha, and Manuel Mazzara. Towards predicting architectural design patterns: A machine learning approach. *Computers*, 11(10), 2022.

[4] Bahrur Rizki Putra Surya, Agi Putra Kharisma, and Novanto Yudistira. Perbandingan kinerja pola perancangan mvc, mvp, dan mvvm pada aplikasi berbasis android (studi kasus : Aplikasi laporan hasil belajar siswa sma bss). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 4(11), Nov 2020.

[5] SAMPAYO-RODRÍGUEZ, Carmen Jeannette, GONZÁLEZ-AMBRIZ, Rosalba, GONZÁLEZ-MARTÍNEZ, Blanca Areli, ALDANA-HERRERA, and Jonathan. Processor and memory performance with design patterns in a native android application. *Journal of Applied Computing*, 6(18):53–61, 2022.

[6] michaelstonis. Model-view-viewmodel - .net, Jun 2023.

[7] Matius Martin. Mvc vs mvvm – perbedaan antara keduanya, Feb 2024.

[8] Fikri Maulana, Rita Afyenni, and Aldo Erianda. Aplikasi manajemen laboratorium menggunakan metode mvvm berbasis android. *JITSI: Jurnal Ilmiah Teknologi Sistem Informasi*, 3(3):88–93, Sep 2022.

[9] Bambang Wisnuadhi, Ghifari Munawar, and Ujang Wahyu. Performance comparison of native android application on mvp and mvvm. In *Proceedings of the International Seminar of Science and Applied Technology (ISSAT 2020)*, pages 276–282. Atlantis Press, 2020.

[10] Firmansyah Firdaus Anhar, Made Hanindia Prami Swari, and Firza Prima Aditiawan. Analisis perbandingan implementasi clean architecture menggunakan mvp, mvi, dan mvvm pada pengembangan aplikasi android native. *Jupiter (Publikasi Ilmu Keteknikan Industri, Teknik Elektro dan Informatika)*, 2(2):181–191, 2024.

[11] Nayab Akhtar and Sana Ghafoor. Analysis of architectural patterns for android development. June 2021.

[12] FAUZI SHOLICHIN, MOHD ADHAM BIN ISA, SHAHLIZA ABD HALIM, and MUHAMMAD FIRDAUS BIN HARUN. Review of ios architectural pattern for testability, modifiability, and performance quality. *Journal of Theoretical and Applied Information Technolgy*, Aug 2019.