# 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*—Penelitian ini membandingkan performa kecepatan dan penggunaan memori antara Model-View-Presenter (MVP) dan Model-View-ViewModel (MVVM) dalam pengembangan aplikasi Android. Melalui studi literatur, MVP memiliki penggunaan memori yang lebih efisien, sedangkan MVVM memiliki penggunaan CPU lebih efisien dan waktu eksekusi yang lebih cepat. Setelah melakukan pengujian, MVP unggul dalam penggunaan memori, sementara waktu eksekusi MVVM lebih cepat. Secara keseluruhan, pemilihan antara pola arsitektur MVM dan MVVM harus mempertimbangkan kebutuhan proyek dan karakteristik masing-masing pola arsitektur.**

***Index Terms*—MVP, MVVM, Android**

## 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

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

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 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

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

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 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.

| Aspek Performa | MVP | MVVM | Perbedaan |
|---|---|---|---|
| Penggunaan CPU | Lebih tinggi | Lebih rendah | 0,55% |
| Waktu Eksekusi | Lebih lambat | Lebih cepat | 126,21 ms |
| Penggunaan Memori | Lebih rendah | Lebih tinggi | 0,92 Mb |

TABLE I
PERBANDINGAN PERFORMA MVP DAN MVVM
SUMBER: DIADAPTASI DARI [9]

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.

| Aspek Pengujian | MVP | MVVM | MVC | VIPER |
|---|---|---|---|---|
| Baris Kode | 129 | 113 | 164 | 123 |
| Waktu Pengujian (detik) | 20.9 | 19.6 | 24.1 | 21.4 |
| Kemampuan Kohesi (procedural) | 0,3809 | 0,55 | 0,4 | 0,528 |
| Penggunaan Memori (Mb) | 22,9645 | 21,19275 | 26,12875 | 23,269 |
| Penggunaan CPU (%) | 10,234 | 11,615 | 22,85 | 8,925 |

TABLE II
PERBANDINGAN PERFORMA ANTARA MVP, MVC, MVVM, DAN VIPER
SUMBER: DIADAPTASI DARI [12]

## III. METODOLOGI

### A. Tahap Penelitian

Pada penelitian ini, ada 5 tahapan yang dilakukan yaitu, identifikasi masalah, pengumpulan data, perancangan, implementasi, dan pengujian.

Penelitian dimulai dengan mengidentifikasi masalah yaitu perbandingan performa antara pola arsitektur MVP dan MVVM.

Kemudian, pengumpulan data dilakukan dengan studi pustaka dari penelitian-penelitian sebelumnya. Data yang dikumpulkan berupa informasi berdasarkan referensi, jurnal, dan artikel online.

Tahap selanjutnya yaitu merancang pola arsitektur MVP dan MVVM. Pada tahap ini, pembuatan *Class Diagram* dan *Sequence Diagram* dibuat menggunakan *PlantUML.*

Setelah merancang diagram, implementasi pola MVP dan MVVM dilaksanakan melalui rekaman waktu yang dibutuhkan untuk menyelesaikan operasi pada setiap *instance* MVP dan MVVM.

Pengujian performa pola MVP dan MVVM diukur dari berapa lama waktu yang dibutuhkan untuk menyelesaikan

operasi dan jumlah memori yang digunakan selama operasi berlangsung. Kedua pengukuran ini kemudian dibandingkan berdasarkan jumlah *view* dan *spinner* yang ada di dalam *instance*.

## B. Perancangan

### 1) Pola Arsitektur MVP:

*a) Class Diagram:* Diagram kelas menggambarkan arsitektur Model-View-Presenter (MVP), yang digunakan untuk memisahkan fungsionalitas aplikasi dari antarmuka pengguna (UI). Seperti pada gambar 1, Model memiliki properti data: String untuk menyimpan data, serta metode getData() dan setData(String) untuk mengakses dan memodifikasi data. Antarmuka IView menyediakan kontrak untuk tampilan, yang mencakup metode untuk menampilkan data, mendapatkan input, dan menampilkan pesan kesalahan [13].

Kelas View mengimplementasikan antarmuka IView dan bertanggung jawab mengelola interaksi pengguna melalui metode yang tercantum. Antarmuka IPresenter mendefinisikan kontrak untuk presenter, termasuk metode onDataRequested(), onDataSubmitted(String), dan onError(String) yang dipanggil ketika data diminta, dikirim, atau terjadi kesalahan.

Kelas Presenter mengimplementasikan IPresenter dan berfungsi sebagai penghubung antara Model dan View, mengelola data dan antarmuka pengguna melalui referensi ke Model dan IView. Hubungan kelas menunjukkan bahwa Model memperbarui Presenter, View mengimplementasikan IView, Presenter mengimplementasikan IPresenter, dan View diperbarui oleh Presenter menggunakan IView. Menggunakan pendekatan MVP membuat kode aplikasi lebih modular, dapat dikelola, dan dapat diuji.

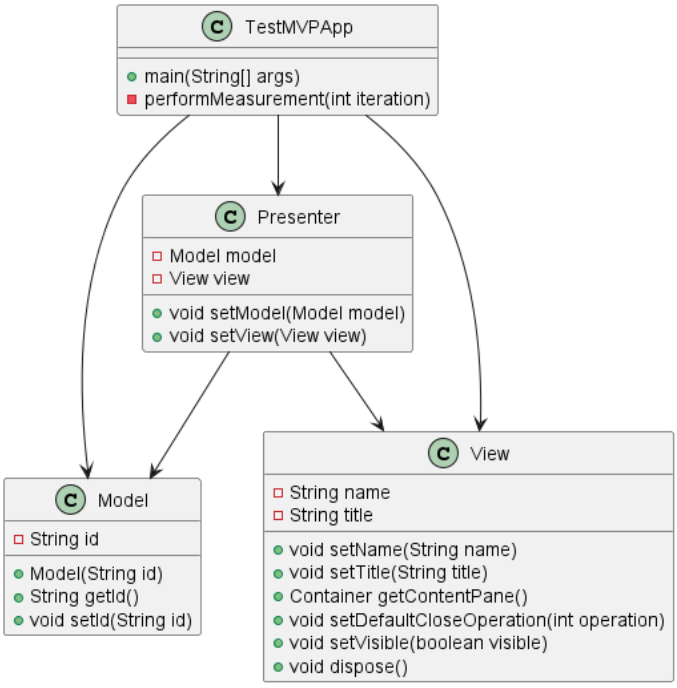


Fig. 1. Class Diagram Arsitektur MVP

*b) Sequence Diagram:* Diagram urutan MVP menunjukkan bahwa proses dimulai ketika pengguna mengubah nilai inputSpinner di View. View mendeteksi perubahan dan mengirimkan acara ke Presenter. Presenter kemudian memperoleh nilai terbaru dari inputSpinner dan mengubah Model, yang kemudian memperbarui data. Presenter memperoleh nilai baru dari Model dan mengubah outputSpinner di View, yang menampilkan nilai baru tersebut. MVP mengelola data dan interaksi pengguna secara langsung antara Model dan View melalui Presenter.

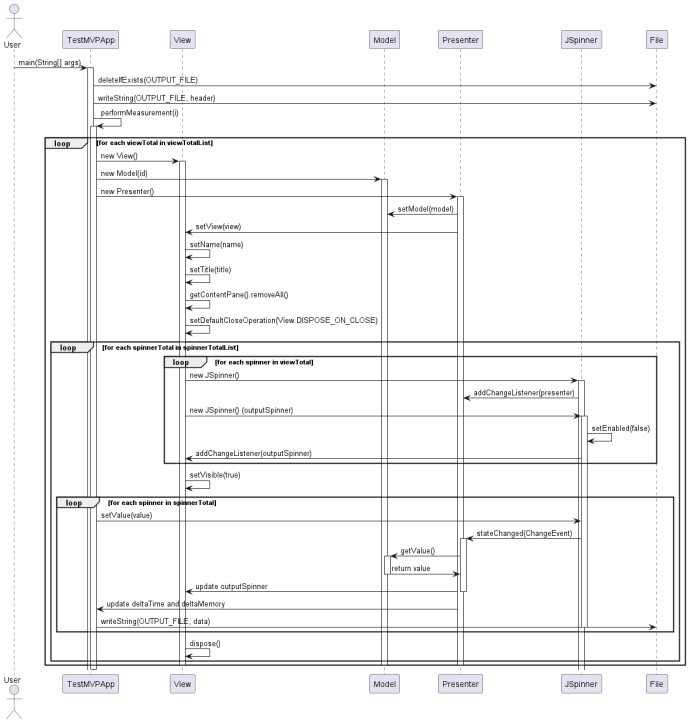


Fig. 2. Sequence Diagram Arsitektur MVP

### 2) Pola Arsitektur MVVM:

*a) Class Diagram:* Diagram kelas Unified Modeling Language (UML) yang disediakan menunjukkan pola arsitektur MVVM (Model-View-ViewModel), yang banyak digunakan dalam pengembangan perangkat lunak, terutama dalam sistem dengan antarmuka pengguna grafis (GUI). Seperti pada gambar 3, diagram ini memiliki tiga komponen utama: Model, View, dan ViewModel [13].

Kelas model mewakili data dan logika bisnis aplikasi. Ini termasuk properti data yang menyimpan data aplikasi, serta metode untuk mengambil dan memodifikasi data tersebut. Model bertanggung jawab untuk memelihara status data dan aturan bisnis aplikasi.

Antarmuka IView mewakili kontrak tampilan aplikasi. Ini mendefinisikan fungsi updateView(data: String), yang harus digunakan oleh tampilan untuk memperbarui tampilan mereka sebagai respons terhadap perubahan dalam data.

Kelas View mengimplementasikan antarmuka IView dan menyediakan antarmuka pengguna (UI) aplikasi. Ini memiliki fungsi updateView(data: String) yang memperbarui tampilan

berdasarkan perubahan dalam data. Ini juga dapat memiliki metode untuk menangani interaksi pengguna, seperti userAction().

Kelas ViewModel berfungsi sebagai perantara antara Model dan View. Ini mencakup referensi ke Model dan View (melalui atribut pribadi -model: Model dan -view: IView). Fungsi utama ViewModel adalah mengirim data dari Model ke View dan mengubah Model sebagai respons terhadap interaksi pengguna di View. Ini menawarkan metode seperti onDataChanged(data: String) yang memberi tahu View tentang perubahan data dan requestData(), yang memulai pengambilan data dari model.

ViewModel ini tergantung pada baik Model maupun View, karena ia memanipulasi Model dan memperbarui View sebagai respons terhadap perubahan Model. View bergantung pada ViewModel untuk menangani perubahan data dan interaksi pengguna. ViewModel berkomunikasi dengan Model untuk mengambil atau memperbarui informasi. Secara keseluruhan, paradigma MVVM membedakan antara manajemen data (Model), antarmuka pengguna (View), dan logika perantara (ViewModel), memungkinkan modularitas, uji coba, dan keberlanjutan dalam program perangkat lunak.

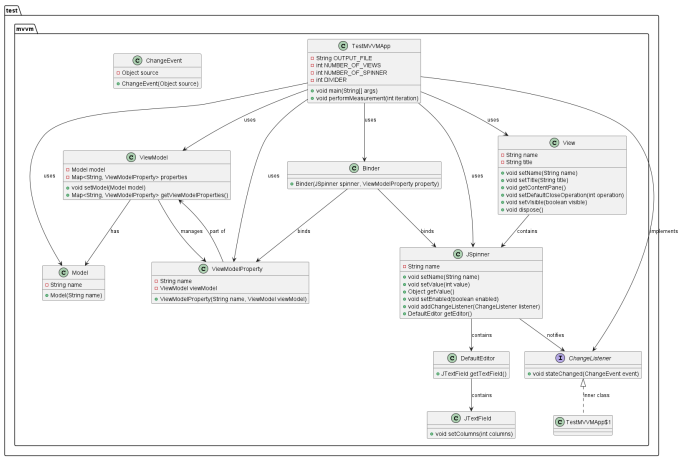


Fig. 3. Class Diagram Arsitektur MVVM

*b) Sequence Diagram:* Diagram urutan MVVM menunjukkan bahwa proses dimulai ketika pengguna mengubah nilai pada inputSpinner di View. Binder mengenali perubahan tersebut dan mengubah ViewModelProperty yang sesuai. ViewModelProperty mengubah data di ViewModel, yang kemudian memperbarui Model dengan nilai baru. Model mengubah datanya, dan ViewModel mengenali perubahan tersebut dan memperbarui ViewModelProperty yang sesuai. Binder mendeteksi perubahan pada ViewModelProperty dan mengubah outputSpinner di View, menampilkan nilai baru tersebut. MVVM mengelola pengikatan data dengan cara yang lebih sistematis, mengisolasi fungsi aplikasi dari komponen UI.

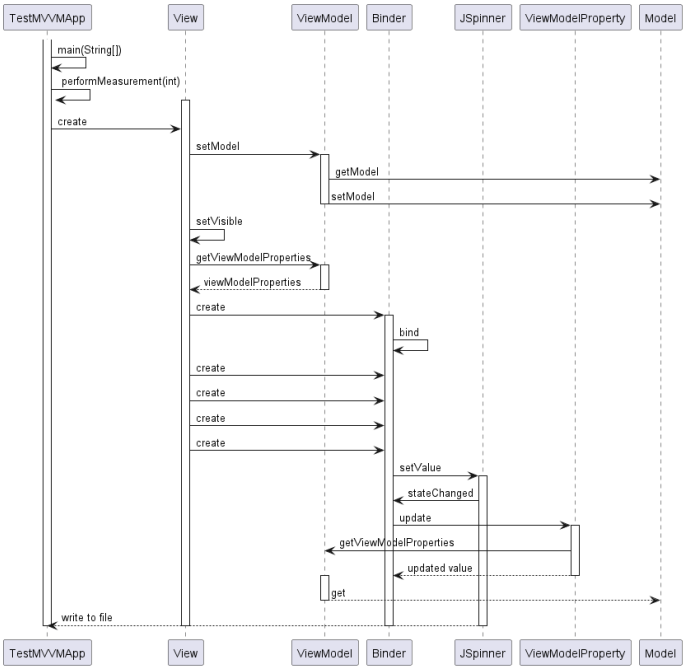


Fig. 4. Sequence Diagram Arsitektur MVVM

## C. Implementasi

*1) Pola Arsitektur MVP:* Dalam arsitektur MVP, *Model*, *View*, dan *Presenter* bekerja bersama untuk menciptakan struktur aplikasi yang terorganisir dengan baik. Kelas *Model* bertanggung jawab dalam mengelola data aplikasi, memiliki atribut nama dan peta pasangan kunci-nilai, serta metode untuk mengatur dan mendapatkan nilai-nilai ini. Kelas *View* bertugas dalam menangani antarmuka pengguna, menggunakan komponen Swing seperti 'JFrame' dan 'JSpinner'. Kelas *View* juga mengatur jendela dan spinner serta memungkinkan *Presenter* untuk mendaftar sebagai pendengar perubahan pada spinner. Kelas *Presenter* berfungsi sebagai perantara antara *Model* dan *View*. Ketika pengguna mengubah nilai spinner, metode *'stateChanged'* dalam *Presenter* dipicu, yang memperbarui *model* dan mencerminkan perubahan tersebut di *View* dengan memperbarui spinner output yang sesuai.

*2) Pola Arsitektur MVVM:* Implementasi dari arsitektur MVVM dilakukan dengan aplikasi Java Swing. Pada pola MVVM, adanya 6 kelas yang dibuat yaitu, *Binder*, *Model*, *View*, *ViewModel*, *ViewModelProperty*, dan *ViewModelPropertyChangeEvent*.

Kelas *Binder* bertanggung jawab untuk mengikat properti-properti antara JSpinner dan *ViewModelProperty*. Ketika nilai JSpinner berubah, *Binder* akan mengatur nilai yang sesuai pada *ViewModelProperty*, dan sebaliknya.

Kelas *Model* merepresentasikan data dan logika bisnis aplikasi. Dalam implementasi, kelas *Model* memiliki sebuah nama dan kumpulan nilai-nilai yang dikaitkan dengan kunci tertentu.

Kelas *View* mewakili tampilan aplikasi. Di dalamnya terdapat JSpinner yang ditampilkan pada pengguna. Kelas

*Binder* digunakan untuk mengaitkan JSpinner dengan properti-properti *ViewModel*.

Kelas *ViewModel* bertanggungjawab untuk mengelola data yang ditampilkan di View. Dalam implementasi ini, *ViewModel* memiliki referensi ke objek *Model* dan peta properti-properti yang terkait dengan JSpinner.

Kelas *ViewModelProperty* merepresentasikan properti di dalam *ViewModel*. Properti ini memiliki nama, nilai, dan *listener* perubahan nilai.

*ViewModelPropertyChangeEvent* merupakan *interface* untuk mendefinisikan sebuah metode yang akan dipanggil ketika nilai properti berubah.

### D. Pengujian

*1) Pola Arsitektur MVP:* Setelah semua setup diimplementasikan dengan benar, maka kelas TestMVPApp.java dipanggil untuk melakukan pengujian dengan membuat beberapa *view* dan mengukur kinerja. Kelas ini mengatur lingkungan uji, menyimpan semua data pengujian data dalam file 'data_mvp.csv', dan melakukan pengukuran kinerja. Kelas ini melakukan iterasi sebanyak 12 kali melalui berbagai konfigurasi *total view* dan spinner (1, 25, 50, 75, 100). Untuk setiap kombinasi, kelas ini menginisialisasi *view* baru dengan *Model* dan *Presenter* yang terkait, mengatur input dan output spinner, dan mencatat waktu mulai saat nilai spinner input diubah. Hasilnya, termasuk nomor iterasi, *total view*, nomor *view*, total spinner, nomor spinner, waktu pemrosesan, dan penggunaan memori, ditulis ke 'data_mvp.csv'. Pendekatan sistematis ini memungkinkan analisis kinerja yang mendetail dari arsitektur MVP dalam menangani beberapa *view* dan interaksi pengguna secara efisien.

*2) Pola Arsitektur MVVM:* TestMVVMApp berfungsi untuk menguji kinerja arsitektur MVVM yang diimplementasikan dalam aplikasi Java Swing. Pengujian dimulai dengan membaut file baru saat pengujian berlangsung. Setiap kali aplikasi dijalankan, header untuk file CSV output juga ditampilkan di konsol untuk memastikan bahwa data yang tercatat sesuai. Kemudian, metode 'performMeasurement' melakukan pengukuran kinerja dengan menghasilkan konfigurasi berbeda untuk jumlah *views* dan spinner berdasarkan nilai *'divider'*. Setelah itu, untuk setiap konfigurasi *view* dan spinner, aplikasi secara dinamis membuat *view* dan mengaitkannya dengan spinner, *model*, dan *viewmodel*. Selanjutnya, *binding* antara spinner input-output dan *viewmodel* dilakukan dengan menggunakan kelas 'Binder'. *Listener* perubahan ditempatkan pada spinner keluaran untuk mengukut metriks kinerja seperti waktu dan penggunaan memori. Setelah semua pengukuran selesai untuk satu konfigurasi, *view* akan dihapus untuk membersihkan sumber daya. Hasil pengukuran, termasuk iterasi, *total views*, nomor *views*, total spinner, nomor spinner, waktu, dan penggunaan memori direkan dalam file 'data_mvvm.csv' untuk analisis lebih lanjut.

## IV. Hasil dan Pembahasan

Setelah pengujian dilakukan, data yang didapatkan, digunakan untuk menghasilkan 4 grafik dengan menggunakan python. Grafik-grafik berupa perbandingan spin atau views terhadap memori atau waktu.

### A. Total Spin to Total Memory

Berdasarkan gambar 5, pola arsitektur MVP memiliki rata-rata penggunaan memori yang meningkat dari 63,348,572 menjadi 188,011,806, seiring spin total meningkat. Rentang antar kuartil juga menunjukkan variasi penggunaan memori seiring dengan peningkatan spin total.

Pola arsitektur MVVM juga memiliki rata-rata penggunaan memori yang meningkat dari 67,522,209 menjadi 188,434,529 dengan meningkatnya spin total. Rentang antar kuartil pada MVVM juga memiliki tren yang serupa dengan MVP.

Secara keseluruhan, kedua arsitektur menunjukkan peningkatan penggunaan memori yang signifikan dengan meningkatnya spin total. Namun, MVVM sedikit lebih efisien dalam penggunaan memori pada setiap tingkat spin total dibandingkan dengan MVP.

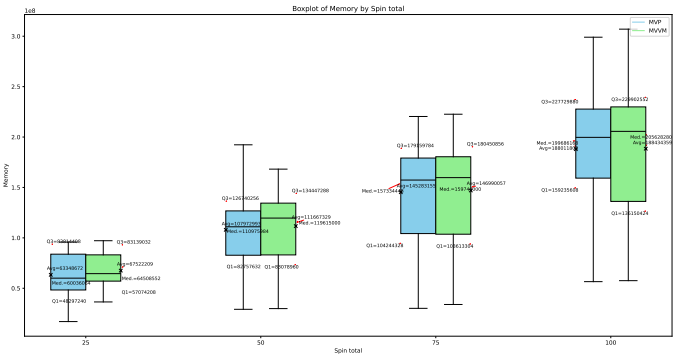


Fig. 5. Perbandingan *Total Spin to Total Memory*

### B. Total Spin to Total Time

Berdasarkan gambar 6, pola arsitektur MVP memiliki rata-rata waktu eksekusi yang meningkat dari 8051 menjadi 9324 dengan meningkatnya spin total dari 25 hingga 100. Rentang antar kuartil juga menunjukkan peningkatan. Hal ini menunjukkan bahwa pola MVP memiliki variasi yang lebih besar dalam waktu eksekusi saat spin total meningkat.

Sedangkan pola arsitektur MVVM memiliki rata-rata waktu eksekusi menurun dari 2355 menjadi 2205 saat spin total meningkat. Rentang antar kuartil juga relatif konstan dibandingkan MVP, menunjukkan konsistensi yang lebih besar dalam waktu eksekusi.

Secara keseluruhan, MVVM cenderung memiliki waktu eksekusi yang lebih cepat dan lebih konsisten dibandingkan arsitektur MVP saat spin total meningkat. Hal ini menunjukkan bahwa MVVM lebih efisien dalam menangani peningkatan beban kerja.

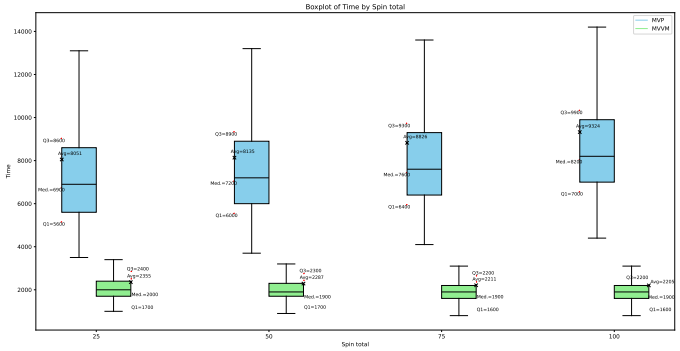


Fig. 6. Perbandingan *Total Spin to Total Time*

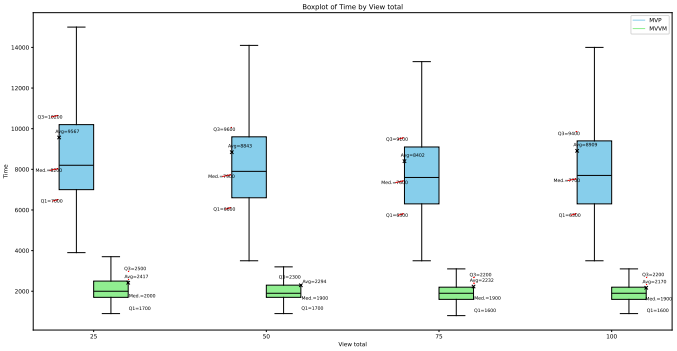


Fig. 8. Perbandingan *Total View to Total Time*

### C. Total View to Total Memory

Berdasarkan gambar 7, pola arsitektur MVP memiliki rata-rata penggunaan memori yang meningkat dari 71,648,523 menjadi 181,236,424, seiring meningkatnya total view.

Pola arsitektur MVVM juga memiliki tren yang serupa dengan MVP dengan rata-rata penggunaan memori meningkat dari 73,473,020 menjadi 185,514,464.

Walaupun kedua pola arsitektur memiliki tren yang meningkat, pola MVP lebih efisien dalam penggunaan memori dibandingkan MVVM. Hal ini berarti MVP menggunakan lebih sedikit memori untuk jumlah view yang sama.

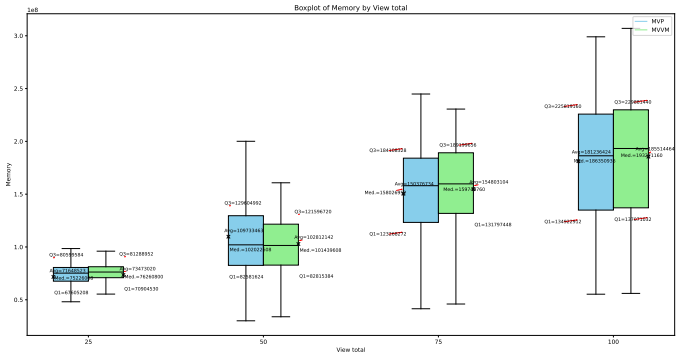


Fig. 7. Perbandingan *Total View to Total Memory*

### D. Total View to Total Time

Berdasarkan gambar 8, pola arsitektur MVP memiliki rata-rata waktu eksekusi yang berpola 'U-shaped'. Rata-rata waktu eksekusi pada view total 25 hingga 75 menurun dari 9567 menjadi 8402, kemudian naik menjadi 8909 pada view total 100.

Sedangkan pola arsitektur MVVM memiliki rata-rata waktu eksekusi menurun, yaitu dari 2417 menjadi 2170.

Secara keseluruhan, pola MVVM menunjukkan waktu eksekusi yang lebih cepat daripada MVP untuk semua view total yang diuji.

## V. Kesimpulan

Pola arsitektur MVP menunjukkan peningkatan penggunaan memori seiring dengan peningkatan jumlah view maupun spinner, namun MVP lebih efisien dalam penggunaan memori dibandingkan dengan MVVM pada setiap jumlah view atau spinner.

Pada waktu eksekusi, MVVM menunjukkan waktu eksekusi yang lebih konsisten dan lebih cepat dibandingkan MVP pada setiap jumalh view atau spinner.

Secara keseluruhan, kedua pola arsitektur memiliki kelebihan dan kekurangan masing-masing. Pemilihan antara MVP dan MVVM harus mempertimbangkan kebutuhan spesifik aplikasi, prioritas dalam efisiensi atau waktu eksekusi, serta kompleksitas pengembangan dan pemeliharaan kode.

## 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.

[13] Irya Muhammad Riyadhi, Intan Purnamasari, and Kamal Prihandani. Penerapan pola arsitektur mvvm pada perancangan aplikasi pengaduan masyarakat berbasis android. *INFOTECH journal*, 9(1):147–158, May 2023.