# Analisis Perbandingan Performa Pola Arsitektur MVC dan MVVM dalam Mengeksekusi Operasi


Andrew Kurniawan Gianto
*Informatika*
*Pradita University*
Tangerang, Indonesia
andrew.kurniawan@student.pradita.ac.id

Ryo Ferdinand
*Informatika*
*Pradita University*
Tangerang, Indonesia
ryo.ferdinand@student.pradita.ac.id

Eka Rifail Saipuddin Rachman
*Informatika*
*Pradita University*
Tangerang, Indonesia
eka.rifail@student.pradita.ac.id

Verrel Novendra Sulu
*Informatika*
*Pradita University*
Tangerang, Indonesia
verrel.novendra@student.pradita.ac.id

Hezel Anthonie Norman Piter Papia
*Informatika*
*Pradita University*
Tangerang, Indonesia
hezel.anthonie@student.pradita.ac.id



***Abstract*—Pemilihan model arsitektur yang tepat sangat penting untuk mengembangkan aplikasi. Seiring berkembangnya teknologi, para pengembang harus memahami model arsitektur yang akan digunakan untuk mengembangkan aplikasi. Penelitian ini bertujuan untuk membandingkan dua model arsitektur perangkat lunak, yaitu *Model-View-Controller*(MVC) dan *Model-View-ViewModel*(MVVM).**

**Tahap penelitian ini terbagi menjadi 5 tahap, yaitu Identifikasi Masalah, Pengumpulan Data, Perancangan, Implementasi, dan Pengujian. Pada tahap Identifikasi Masalah, menentukan model arsitektur yang lebih baik dalam beberapa aspek akan dilakukan. Pada tahap Pengumpulan Data, penting mengumpulkan data-data untuk membandingkan kedua model arsitektur. Pada tahap Perancangan, class diagram dan sequence diagram kedua arsitektur dirancang untuk menemukan kedua pola arsitektur. Pada tahap Implementasi, kedua kode arsitektur diterapkan berdasarkan class diagram dan sequence diagram yang telah dirancang. Pada tahap Pengujian, pola arsitektur MVC dan MVVM akan diuji.**

**Testing aplikasi dibutuhkan untuk menghasilkan 4 aspek, yaitu *View to total time, View to total memory, Spin to total time,* dan *Spin to total memory*. Dengan adanya hasil dari 4 aspek tersebut, kita dapat mengetahui arsitektur model manakah yang lebih unggul.**

***Index Terms*—MVC, MVVM, kecepatan, memori**


## I. Pendahuluan

Perangkat lunak telah menjadi dasar bagi banyak aplikasi dan sistem yang digunakan dalam berbagai aspek kehidupan di era teknologi informasi yang terus berkembang. Perangkat lunak telah mendominasi cara kita berinteraksi dengan dunia digital. Dari infrastruktur bisnis yang membantu operasi perusahaan hingga aplikasi yang memenuhi kebutuhan sehari-hari. Tanpa adanya perangkat lunak yang efektif dan terorganisir, akses dalam bertransaksi, berkomunikasi, dan memperoleh informasi akan terbatas. Perangkat lunak telah mengubah cara kita bekerja, bermain, dan juga berinteraksi dengan dunia luar. Oleh karena itu, perangkat lunak tidak hanya harus beroperasi dengan efisien, tetapi juga harus mudah dikembangkan dan diubah sesuai kebutuhan. Pengembang perangkat lunak dituntut untuk menulis kode yang mudah dipahami dan dikelola, terutama untuk sistem yang kompleks dengan masalah yang berulang.

Pola desain antarmuka pengguna dapat membantu mengurangi kekacauan dalam sistem yang kompleks, membuat aplikasi lebih terorganisir dan dapat diskalakan. Dengan mempelajari dan menerapkan pola antarmuka pengguna yang tepat, pengembang dapat membuat aplikasi yang lebih efisien dan terstruktur. Pola arsitektur ini dapat memungkinkan pengembang untuk fokus pada masalah yang spesifik. Saat ini, ada berbagai macam pola arsitektur pengembang yang masing-masing arsitektur memiliki kelebihan dan kekurangan sehingga tidak semua arsitektur cocok dalam segala hal. Karena itu, memilih arsitektur yang tepat sangat penting bagi pengembang untuk memastikan aplikasi yang dikembangkan dapat memenuhi kebutuhan pengguna. Dalam artikel ini, akan dibahas dua arsitektur populer, yaitu *Model-View-Controller* (MVC) dan *Model-View-ViewModel* (MVVM).

Tujuan dari penelitian ini adalah untuk membandingkan kinerja arsitektur MVC dengan MVVM, khususnya dalam hal kecepatan dan penggunaan memori. MVC telah lama menjadi arsitektur yang umum digunakan dalam pengembangan perangkat lunak, terutama dengan pendekatan prototipe. Di sisi lain, MVVM sering digunakan dalam pengembangan aplikasi mobile. Penelitian ini akan menentukan arsitektur mana yang paling efisien dan sesuai dengan berbagai kebutuhan pengembangan perangkat lunak melalui eksperimen yang berfokus pada komponen *Model-View* (MV).

Hasil dari penelitian ini diharapkan dapat membantu

pengembang perangkat lunak memahami kelebihan dan kelemahan masing-masing arsitektur, serta menambah wawasan tentang performa masing-masing arsitektur. Meskipun kedua arsitektur ini telah populer sebagai pengembangan perangkat lunak, pengembang perlu memahami dengan lebih baik mengenai kinerja dan karakteristik masing-masing arsitektur. Dengan demikian, pengembang dapat mengurangi risiko pengembangan dengan membuat keputusan yang lebih baik dalam memilih arsitektur yang tepat untuk proyek mereka, menghasilkan aplikasi yang lebih baik, efisien, dan andal di era teknologi informasi yang terus berkembang.

## II. Kajian Terkait

### A. Model-View-Controller (MVC)

*Model-View-Controller* atau MVC adalah arsitektur perangkat lunak untuk membuat sebuah aplikasi dengan memisahkan data (*Model*) dari tampilan (*View*) dan cara bagaimana memprosesnya (*Controller*) [1] [3] . MVC menekankan tiga elemen penting: perhatian, tanggung jawab, dan logika. Ini mempercepat kinerjanya.

Penjelasan dari masing-masing komponen MVC dapat dilihat pada figure 1, dan mempunyai penjelasan sebagai berikut:

a Bagian *model* biasanya digunakan untuk mengambil data dari database atau menyimpan data ke dalam database.
b Bagian *view* adalah komponen yang menampilkan antarmuka pengguna aplikasi, yang dibangun berdasarkan data model.
c Bagian *control* adalah bagian yang menangani interaksi pengguna, bekerja dengan model, dan menyimpan data ke dalam database.

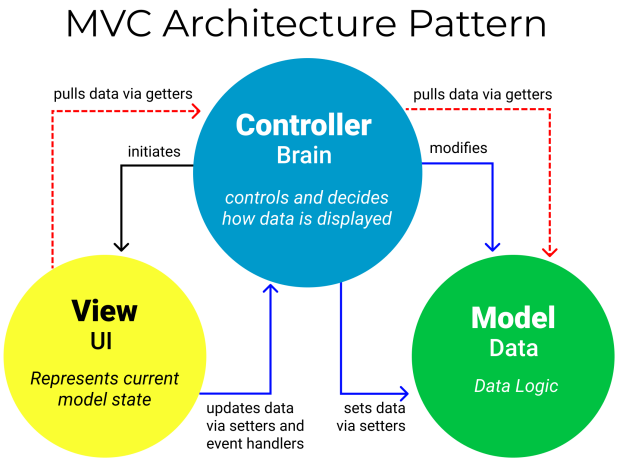


Fig. 1. Arsitektur MVC

Penjelasan mengenai keuntungan dan kerugian dari arsitektur MVC adalah sebagai berikut:

*1) Keuntungan:*

- Pemisahan tanggung jawab antara *Model, View,* dan *Control* membuat aplikasi lebih mudah dipahami, diuji, dan dikembangkan.
- Fleksibilitas dalam mengganti atau memodifikasi salah satu komponen tanpa memengaruhi komponen lainnya.
- Memungkinkan pengembangan paralel antara komponen-komponen aplikasi.

*2) Kerugian:*

- Adanya kompleksitas dalam pengelolaan koneksi antara 3 komponen tersebut.
- Adanya kesulitan dalam pengujian unit pada *View* dan *Controller*.

### B. Model-View-View-Model (MVVM)

*Model View ViewModel* (MVVM) adalah sebuah arsitektur perangkat lunak yang memisahkan antara kode untuk logika bisnis dan tampilan aplikasi [2]. Dalam kata lain, MVVM adalah sebuah pattern desain yang terdiri dari tiga komponen: *Model, View, dan ViewModel* [6].

Penjelasan dari masing-masing komponen MVVM dapat dilihat pada figure 2, dan mempunyai penjelasan sebagai berikut:

a Bagian *model* merupakan representasi dari data yang digunakan dalam logika bisnis.
b Bagian *view* adalah komponen yang terdiri dari layout sumber daya file dan aktivitas/fragmen. Activity/Fragment secara dinamis mengontrol tampilan pada layout sumber daya file.
c Bagian *ViewModel* berinteraksi dengan Model dan menyiapkan variabel yang akan diamati oleh *View*. *ViewModel* bersifat lifecycle-aware, sehingga kelas ini akan hidup ketika sebuah kelas View telah melalui tahapan create dan belum melalui tahapan destroy.

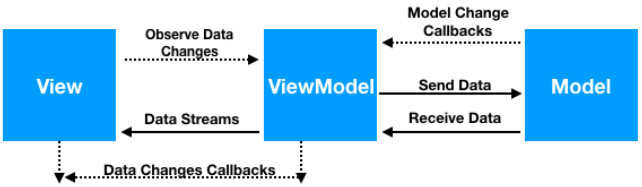


Fig. 2. Arsitektur MVVM

Penjelasan mengenai keuntungan dan kerugian dari arsitektur MVVM adalah sebagai berikut:

*1) Keuntungan:*

- Pemisahan antara *View* dan *ViewModel* sehingga membuat pengujian dan pemeliharaan kode lebih mudah
- Kode MVVM biasanya digerakkan oleh peristiwa, yang berarti kode dapat diuji secara terpisah.

*2) Kerugian:*

- Komponen antarmuka MVVM harus dibuat observable yang berarti dapat memakan waktu dan rumit.
- Banyak kode yang harus ditulis dan dikelola untuk setiap komponen UI sehingga aplikasi dapat menjadi lebih sulit untuk digunakan dan dipahami.

### C. Studi Preseden

Dalam penelitian "Perbandingan Kinerja Pola Perancangan MVC, MVP, dan MVVM Pada Aplikasi Berbasis Android (Studi Kasus: Aplikasi Laporan Hasil Belajar Siswa SMA BSS)" [5], dibahas analisis perbandingan tiga pola perancangan arsitektur perangkat lunak yang populer: Model-View-Controller, Model-View-Presenter, dan Model-View-ViewModel (MVVM). Aplikasi laporan hasil belajar siswa SMA Brawijaya Smart School digunakan sebagai basis untuk penelitian ini. Tujuan dari penelitian ini adalah untuk menemukan pola perancangan yang paling hemat energi dan memori untuk aplikasi Android. Penelitian dimulai dengan rekayasa untuk menentukan persyaratan fungsional dan non-fungsional. Persyaratan ini kemudian digunakan sebagai dasar untuk desain dan pelaksanaan penelitian. Ketiga pola desain ini digunakan untuk memulai implementasi Java. Setelah tahap implementasi, setiap aplikasi yang menggunakan pola ini diuji. Pengujian dilakukan sebanyak lima kali untuk mendapatkan data penggunaan energi dan memori. Hasil pengujian menunjukkan bahwa ketiga pola perancangan memiliki penggunaan energi yang sederhana, dengan penggunaan memori rata-rata 59,7 MB untuk MVC, 59 MB untuk MVP, dan 73,2 MB untuk MVVM. Selain itu, tahap pengujian fungsional dengan metode pengujian blackbox memberikan hasil validitas sebesar 100% untuk semua fungsi.

TABLE I
PERBANDINGAN MEMORI DAN WAKTU EKSEKUSI BERDASARKAN PENELITIAN "PERBANDINGAN KINERJA POLA PERANCAN- GAN MVC, MVP, DAN MVVM PADA APLIKASI BERBASIS ANDROID"

| Arsitektur | Penggunaan Memori (MB) |
|---|---|
| MVC | 59.7 |
| MVP | 59.0 |
| MVVM | 73.2 |

Dalam penelitian "A Comparison of Android Native App Architecture—MVC, MVP, and MVVM" [4], dibahas mengenai masalah efisiensi dan kualitas pengembangan aplikasi Android yang dipengaruhi oleh arsitektur aplikasi. Tujuan dari artikel ini adalah untuk memberikan analisis menyeluruh untuk menentukan apakah arsitektur MVP (Model-View-Presenter) dan MVVM (Model-View-ViewModel) lebih unggul dari arsitektur aplikasi Android native bawaan, yaitu MVC (Model-View-Controller). Dengan menggunakan Metode Analisis Tradeoff Arsitektur untuk menjawab pertanyaan ini dan menetapkan tiga kriteria: ketepatan, modifikasi, dan kinerja. Berdasarkan kriteria ini, mereka mengenali faktor-faktor penting untuk setiap atribut kualitas dan melakukan perbandingan. Hasil analisis dan eksperimen menunjukkan bahwa MVP dan MVVM lebih unggul daripada MVC dalam hal testability, modifiability (tingkat coupling yang rendah), dan performance (mengonsumsi memori yang lebih sedikit).

Dalam penelitian "Performance Comparison of Native Android Application on MVP and MVVM" [**?**], dibahas analisis komparatif terhadap arsitektur MVP (*Model-View-Presenter*) dan MVVM (*Model-View-ViewModel*) dalam konteks pengembangan aplikasi Android asli dilakukan dalam makalah yang ditulis oleh Wisnuadhi, Munawar, dan Wahyu. Tujuan dari penelitian ini adalah untuk mengevaluasi kinerja kedua arsitektur tersebut dengan mempertimbangkan tiga faktor: penggunaan CPU, penggunaan memori, dan waktu eksekusi. Hasil eksperimen menunjukkan bahwa arsitektur MVVM lebih baik dalam penggunaan CPU dan waktu eksekusi, sedangkan MVP unggul dalam penggunaan memori. Aplikasi MVVM memiliki penggunaan CPU yang lebih rendah dengan perbedaan rata-rata 0,55% dan waktu eksekusi yang lebih cepat dengan perbedaan rata-rata 126,21 ms. Aplikasi MVP juga memiliki penggunaan memori yang lebih rendah dengan perbedaan rata-rata 0,92 MB.

TABLE II
PERBANDINGAN PENGGUNAAN MEMORI DAN WAKTU EKSEKUSI BERDASARKAN PENELITIAN "PERFORMANCE COMPARISON OF NATIVE ANDROID APPLICATION ON MVP AND MVVM"

| Arsitektur | Waktu Eksekusi |
|---|---|
| MVP | lebih pelan 126,21 ms |
| MVVM | lebih cepat 126,21 ms |

## III. METODOLOGI

Metodologi penelitian ini terbagi menjadi 5 tahap, yaitu Tahap Identifikasi Masalah, Tahap Pengumpulan Data, Tahap Perancangan, Tahap Implementasi, dan terakhir Tahap Pengujian.

### A. Identifikasi Masalah

Masalah utama penelitian ini adalah untuk menentukan model arsitektur mana yang lebih baik dan cepat dalam empat aspek utama: penggunaan memori total saat melakukan operasi spin, waktu total yang dibutuhkan untuk melakukan operasi spin, penggunaan memori total saat melakukan operasi pandang, dan waktu total yang dibutuhkan untuk melakukan operasi pandang. Pemilihan arsitektur yang tepat sangat penting untuk memastikan bahwa sebuah aplikasi dapat beroperasi dengan efisien daripada sebelumnya. Perbandingan model arsitektur dalam penelitian ini adalah membandingkan kinerja arsitektur *Model-View-Controller* (MVC) dan *Model-View-ViewModel* (MVVM) dalam halnya kecepatan dan penggunaan memori.

### B. Pengumpulan Data

Mengumpulkan data-data tentang arsitektur MVC dan MVVM penting untuk mendapatkan dasar teoritis dan pemahaman yang kuat. Dimulai dari membaca literatur yang ada tentang topik tersebut untuk memahami konsep dasar, manfaat, dan kelemahan dari masing-masing arsitektur. Kemudian mengumpulkan studi kasus yang relevan yang memberikan

penjelasan teoritis dan aplikasi praktis dari arsitektur MVC dan MVVM dalam berbagai konteks. Studi kasus ini juga memberikan wawasan penting tentang bagaimana arsitektur ini digunakan dan diimplementasikan dalam situasi tertentu.

### C. Perancangan

Untuk mendapat gambaran mengenai pola arsitektur untuk setiap *model* (MVC dan MVVM) yang lebih jelas, kami merancang class diagram dan sequence diagram untuk kedua arsitektur *model* tersebut. Untuk memastikan representasi yang jelas dan konsisten dari masing-masing arsitektur, diagram ini dibuat menggunakan kode PlantUML.

*1) Pola Arsitektur MVC:*

*a) Class Diagram:* Class diagram *Model-View-Controller* (MVC), pada figure 3, menggambarkan tiga komponen utama yang berkolaborasi untuk memisahkan logika bisnis dari tampilan antarmuka pengguna. Komponen pertama adalah *Model*, yang bertanggung jawab untuk mengelola data dan logika bisnis. *Model* mengakses basis data atau sumber data lainnya, melakukan operasi seperti perhitungan, dan mengirimkan data ke komponen lain seperti *controller* atau *view*. Komponen kedua adalah *View*, yang bertugas menampilkan data kepada pengguna. *View* mengambil data dari *model* dan menampilkannya dalam format yang sesuai, seperti antarmuka pengguna grafis atau laporan teks. Komponen ketiga adalah *Controller*, yang berfungsi sebagai penghubung antara *model* dan *view*. *Controller* menerima input dari pengguna melalui view, memprosesnya (yang mungkin melibatkan interaksi dengan model), dan menentukan *view* mana yang harus diperbarui dengan data terbaru. Class Diagram MVC menampilkan hubungan ini dengan jelas: *model* dan *view* berinteraksi dengan *controller*, yang mengatur aliran data di antara keduanya.

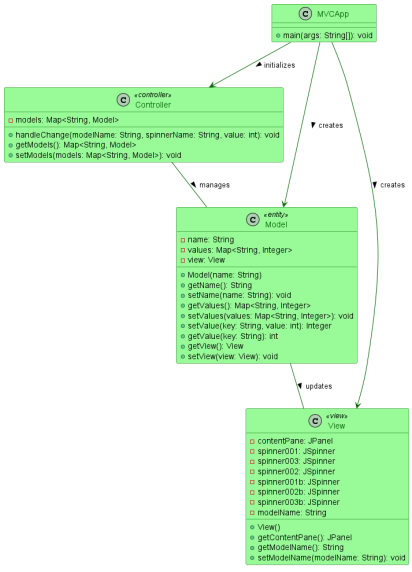


Fig. 3. Class Diagram MVC

*b) Sequence Diagram:* Sequence diagram MVC, seperti yang diilustrasikan figure 4, menggambarkan alur interaksi antara pengguna, *view, controller*, dan *model* dalam skenario operasi umum. Ketika pengguna memasukkan input ke dalam *view, view* meneruskan input tersebut ke *controller. Controller* kemudian memproses input ini dan memperbarui *model* dengan data baru. Setelah *model* diperbarui, *controller* meminta *view* untuk memperbarui tampilannya berdasarkan data terbaru dari *model*. Akhirnya, *view* menampilkan data terbaru kepada pengguna. Proses ini menggambarkan bagaimana MVC memisahkan tanggung jawab di antara komponen-komponennya untuk menjaga modularitas dan skalabilitas aplikasi.

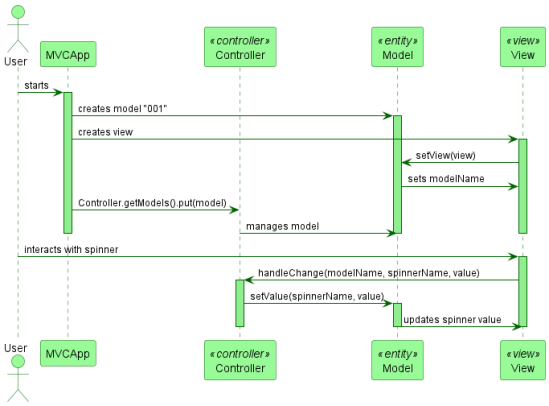


Fig. 4. Sequence Diagram MVC

*2) Pola Arsitektur MVVM:*

*a) Class Diagram:* Dalam Class Diagram *Model-View-ViewModel* (MVVM) yang ada pada figure 5, terdapat hubungan antara tiga komponen utama yang dirancang untuk mendukung data binding dan memisahkan logika UI dari logika bisnis. *Model* dalam MVVM bertanggung jawab untuk data dan logika bisnis, sama seperti dalam MVC. *View* bertanggung jawab untuk menampilkan data kepada pengguna, namun dalam MVVM, *view* lebih terikat erat dengan *ViewModel* melalui mekanisme data *binding*. *ViewModel* adalah komponen yang berfungsi sebagai penghubung antara *view* dan *model*. *ViewModel* mengambil data dari *model*, mengolahnya jika perlu, dan menyediakan data yang diformat untuk *view*. *ViewModel* juga menangani logika UI yang diperlukan oleh *view*, memungkinkan *view* untuk tetap sederhana dan fokus pada rendering antarmuka.

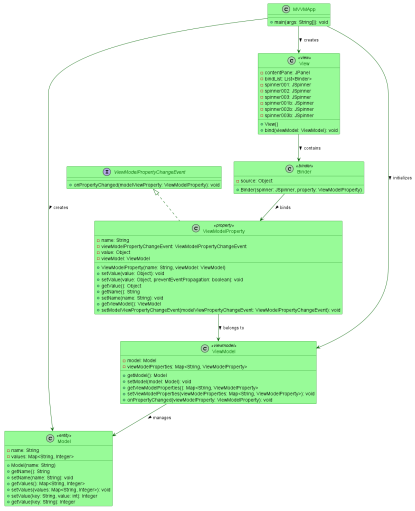

Fig. 5. Class Diagram MVVM

*b) Sequence Diagram:* Sequence diagram MVVM, seperti yang diilustrasikan pada figure 6, menggambarkan alur interaksi antara pengguna, *view, ViewModel*, dan *model* dalam skenario operasi umum. Ketika pengguna memasukkan input ke dalam *view*, input tersebut dikirim ke *ViewModel*. *ViewModel* memproses input ini dan memperbarui *model* sesuai dengan data yang baru. *Model* kemudian memperbarui datanya. Setelah *model* diperbarui, *ViewModel* mengambil data terbaru dari *model* dan memperbarui data yang terikat pada *view*. *View* kemudian menampilkan data terbaru kepada pengguna. Proses ini menunjukkan bagaimana MVVM memanfaatkan data *binding* untuk mengelola aliran data antara *model* dan *view* melalui *ViewModel*, memudahkan pengembangan dan pemeliharaan antarmuka pengguna yang dinamis dan responsif.

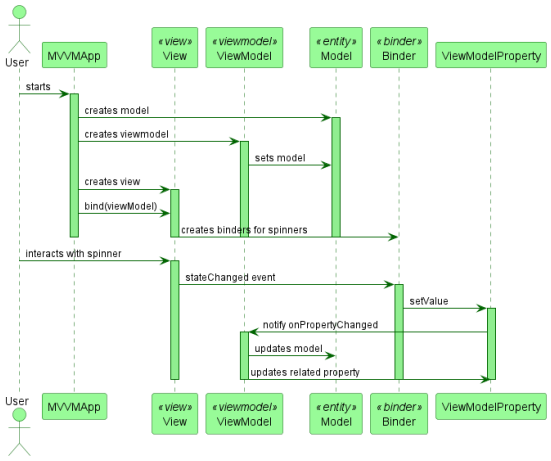


Fig. 6. Sequence Diagram MVVM

## D. Implementasi

Dalam tahap implementasi ini, ada penerapan kode untuk kedua arsitektur (MVC dan MVVM) berdasarkan class diagram dan sequence diagram yang sudah dirancang. Implementasi kode MVC melibatkan pembuatan kelas *model, view*, dan *controller* sesuai dengan Class Diagram yang telah dirancang. Pada implementasi kode MVVM, kami membuat kelas *model, view*, dan *viewmodel* serta *binder* untuk menghubungkan *view* dengan *viewmodel properties*. Kode yang diimplementasikan lalu akan diuji dalam TestApp masing-masing *model* arsitektur. Source code tersebut terlihat pada figure 7.

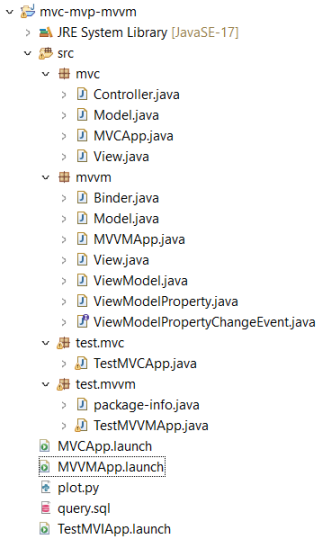


Fig. 7. Source Code mvc-mvvm

*1) Pola Arsitektur MVC: Model-View-Controller* (MVC) terdiri dari: *Model*, *View*, *Controller*, dan *MVCApp*. Masing-masing file memiliki tugas khusus yang berkontribusi terhadap keseluruhan arsitektur MVC.

File *Model* berfungsi untuk menyimpan data aplikasi, memanipulasi data, dan mengubah tampilan komponen di *view* berdasarkan perubahan data.

File *View* berfungsi untuk menampilkan data, menerima input dari pengguna dan mengirimkan perubahan ke *controller*, dan menghubungkan *view* dengan *model* untuk memperbarui tampilan berdasarkan perubahan data.

File *Controller* berfungsi untuk menerima input dari *view* dan memprosesnya, memperbarui data di *model* berdasarkan input yang diterima, menyimpan dan mengelola beberapa *model* dalam aplikasi.

File MVCApp berfungsi untuk menyiapkan *model, view*, dan *controller*, menghubungkan *model* dengan *view* dan *controller*, dan menjalankan aplikasi dengan menampilkan *view* kepada pengguna.

*2) Pola Arsitektur MVVM:* Model-View-ViewModel (MVVM) memiliki tujuh file: *Binder, Model, View, ViewModel, ViewModelProperty, ViewModelPropertyChangeEvent,* dan *MVVMApp*.

File *Binder* berfungsi untuk menghubungkan JSpinner dengan *ViewModelProperty*, *Binder* memungkinkan komunikasi dua arah antara komponen GUI dengan logika aplikasi dalam arsitektur MVVM.

File *Model* bertanggung jawab untuk menyimpan dan mengelola data yang dibutuhkan oleh aplikasi.

File *View* berfungsi untuk menampilkan antarmuka kepada pengguna serta memfasilitasi interaksi antara pengguna dan aplikasi.

File *ViewModel* berfungsi untuk mengelola data yang diperlukan oleh *View* dan melakukan komunikasi dengan *View* dan *Model*, file *ViewModel* juga bertanggung jawab untuk menjaga konsistensi data antara *Model* dan *View*.

File *ViewModelProperty* berfungsi untuk merepresentasikan data atau properti dalam *ViewModel*.

File *ViewModelPropertyChangeEvent* berfungsi untuk menangani apabila terjadi adanya perubahan pada nilai dengan cara memperbarui tampilan yang terkait dengan perubahan nilai tersebut.

File MVVMApp berfungsi untuk menjalankan aplikasi dan mengatur pembuatan dan konfigurasi komponen MVVM yaitu *Model, ViewModel* dan *View*. Aplikasi ini memungkinkan sinkronisasi antara data dan antarmuka pengguna sehingga memudahkan pengelolaan.

### *E. Pengujian*

Dalam tahap pengujian ini, kami menguji kedua pola arsitektur (MVC dan MVVM) pada TestApp masing-masing. Setiap model akan mempunyai 10 instance dan setiap instance ini memiliki *view* yang berisi jumlah JSpinner yang berbeda. Instance pertama (*View* 1) hanya berisi 1 JSpinner, instance kedua (*View* 2) berisi 2 JSpinner, dan seterusnya hingga instance ke-100 (*View* 100) yang berisi 100 JSpinner. Dalam setiap eksperimen, semua nilai JSpinner dari *View* 1 hingga *View* 100 akan diubah, dan waktu yang dibutuhkan untuk menyelesaikan operasi ini akan direkam. Eksperimen ini akan diulang sebanyak 12 kali untuk memastikan konsistensi dan reliabilitas hasil. Testing yang dijalankan akan mengeluarkan output *iteration, view total, view num, spin total, spin num, time, memory*. Begitu juga ada konfigurasi untuk total view dan spinner (1, 25, 50, 75, 100). Output tersebut lalu akan disimpan pada file data.csv masing-masing model. Hasil dari pengujian lalu akan digunakan untuk membandingkan penggunaan memori dan waktu eksekusi antara MVC dan MVVM.

*1) Testing Pola Arsitektur MVC:* Cara kerja TESTMVCApp adalah *View* akan mendeteksi data yang dimasukkan pada input spinner lalu mengirimkan data tersebut ke *controller*. *Controller* akan menerima data tersebut dan akan mencari *model* yang sesuai dengan *model* yang diterima dari *view*. Setelah menemukan *model* yang sesuai, *controller* akan mengubah data tersebut. Lalu *model* akan mengubah tampilan pada *view* sehingga pengguna dapat melihat data yang baru. Program akan mengulang proses pengukurannya sebanyak 12 kali. Setiap hasil pengukuran yang telah dilakukan akan disimpan di dalam file MVCdata.csv yang baru dibuat program.

*2) Testing Pola Arsitektur MVVM:* Cara kerja TESTMVVMApp adalah *model* menyimpan data, *ViewModel* bertindak sebagai mediator yang menghubungkan *Model* dengan *View*, dan *View* menampilkan data serta memperbarui dirinya berdasarkan perubahan yang terjadi di *ViewModel*, dengan *binding* ini memungkinkan untuk *View* memperbarui secara otomatis ketika data pada *ViewModel* berubah. Input *spinner* di *View* ditambahkan yang akan mengubah data di *ViewModel* dan kemudian diperbarui secara otomatis di *View* melalui *binding*. Program lalu akan menjalankan looping proses measurementnya sebanyak 12 kali. Pada setiap iterasi, program membuat dan mengatur hubungan antara *Model*, *ViewModel*, dan *View*. Setiap *View* dibuat dengan sejumlah *spinner* input dan output yang sesuai dengan jumlah yang ditentukan. *Spinner* input terhubung dengan listener yang memperbarui data di *ViewModel*, yang kemudian menggandakan nilai input dan memperbarui *spinner* output. Setelah semua *View* dibuat dan *spinner* diatur, program mencatat data performa, seperti jumlah *view*, jumlah *spinner*, waktu eksekusi, dan penggunaan memori ke dalam file ”MVVMdata.csv”.

## IV. Hasil dan Pembahasan

### *A. Pola Arsitektur MVC*

Figure 8 menunjukkan setiap baris yang ada di dalam file CSV mewakili satu pengukuran, termasuk kolom untuk nomor iterasi (iter), jumlah total *view* yang dibuat (view total), nomor *view* (view num), jumlah total spinner yang dibuat (spin total), indeks spinner dalam total (spin num), waktu sistem dalam menit saat pengukuran diambil (time), dan memori yang digunakan oleh Java Virtual Machine (JVM) pada saat pengukuran (memory).

```
1  iter,view_total,view_num,spin_total,spin_num,time,memory
2  1,25,25,25,0,1716721377045,21078792
3  1,25,25,25,1,1716721377045,21078792
4  1,25,25,25,2,1716721377045,21078792
5  1,25,25,25,3,1716721377045,21078792
6  1,25,25,25,4,1716721377045,21277104
7  1,25,25,25,5,1716721377045,21277104
8  1,25,25,25,6,1716721377045,21277104
9  1,25,25,25,7,1716721377045,21277104
10 1,25,25,25,8,1716721377045,21277104
```

Fig. 8. Output awal MVC

Data ini menunjukkan bahwa program secara iteratif membuat dan membuang sejumlah tampilan dan spinner yang ditentukan, mencatat penggunaan memori dan waktu untuk setiap konfigurasi. Seperti yang diilustrasikan figure 9, dalam iterasi 1 dengan 25 tampilan dan 25 spinner, penggunaan memori konsisten pada 24.753.584 byte dan total time sebanyak 1.716.721.377.045 millisec, sedangkan pada iterasi 12 dengan 100 tampilan dan 100 spinner, penggunaan memori stabil sekitar 215.465.472 byte dan total time sebanyak 1.716.722.517.068 millisec. Total waktu untuk testing app tersebut adalah 19.09346 menit.

Problems Javadoc Declaration Console

<terminated> TestMVCApp (1) [Java Application] C:\Program Files\J

```
12,100,100,100,93,1716722517068,375744224
12,100,100,100,94,1716722517068,375744224
12,100,100,100,95,1716722517068,375744224
12,100,100,100,96,1716722517068,375744224
12,100,100,100,97,1716722517068,375744224
12,100,100,100,98,1716722517068,375744224
12,100,100,100,99,1716722517068,375744224
Total Time: 19.093466666666664 minutes
```

Fig. 9. Output akhir MVC

## B. Pola Arsitektur MVVM

Pada figure 10, diilustrasikan hasil output untuk setiap baris dalam file CSV mewakili satu pengukuran, termasuk kolom untuk nomor iterasi (iter), jumlah total tampilan yang dibuat (*view total*), nomor tampilan (*view num*), jumlah total spinner yang dibuat (*spin total*), indeks spinner dalam total (*spin num*), waktu sistem dalam milidetik saat pengukuran diambil (time), dan memori yang digunakan oleh Java Virtual Machine (JVM) pada saat pengukuran (memory).

```
1  iter,view_total,view_num,spin_total,spin_num,time,memory
2  1,25,25,25,0,1716722568655,31948536
3  1,25,25,25,1,1716722568655,31948536
4  1,25,25,25,2,1716722568655,31948536
5  1,25,25,25,3,1716722568655,31948536
6  1,25,25,25,4,1716722568656,31948536
7  1,25,25,25,5,1716722568656,31948536
8  1,25,25,25,6,1716722568656,31948536
9  1,25,25,25,7,1716722568656,31948536
10 1,25,25,25,8,1716722568656,31948536
```

Fig. 10. Output awal MVVM

Mirip dengan penjelasan MVC, program MVVM ini secara iteratif membuat dan membuang sejumlah tampilan dan spinner yang ditentukan, mencatat penggunaan memori dan waktu untuk setiap konfigurasi. Pada figure 11, kita bisa lihat dalam iterasi 1 dengan 25 *view* dan 25 *spinner*, penggunaan memori konsisten pada 20.872.144 byte dan total time sebanyak 1.716.722.568.656 millisec sedangkan pada iterasi 12 dengan 100 *view* dan 100 *spinner*, penggunaan memori konsisten pada 379.372.808 dan total time sebanyak 1.716.723.718.948 millisec. Total waktu untuk testing app tersebut adalah 19.26506 menit.

Problems Javadoc Declaration Console

<terminated> TestMVVMApp (1) [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe (May 26, 2024, 6:22:42 PM – 6:42:00 PM) [pid: 12904]

```
12,100,100,100,93,1716723718948,242957048
12,100,100,100,94,1716723718948,242957048
12,100,100,100,95,1716723718948,242957048
12,100,100,100,96,1716723718948,242957048
12,100,100,100,97,1716723718948,242957048
12,100,100,100,98,1716723718948,242957048
12,100,100,100,99,1716723718948,242957048
Total Time: 19.265066666666666 minutes
```

Fig. 11. Output akhir MVVM

## C. Perbandingan MVC dan MVVM

Perbandingan hasil testing MVC dan MVVM terbagi menjadi 4 perbandingan yaitu: *View to total time, View to total memory, Spin to total time,* dan *Spin to total memory*.

*1) Perbandingan View to total time:* Hasil perbandingan *View to Total Time* adalah MVVM lebih unggul dan cepat untuk menangani peningkatan jumlah *view* dibandingkan dengan MVC. Hal ini terlihat pada average time taken untuk setiap *view* dalam MVVM (1716723180500 millisec) lebih rendah daripada MVC (1716721982868 millisec) yang dapat dilihat pada figure 12.

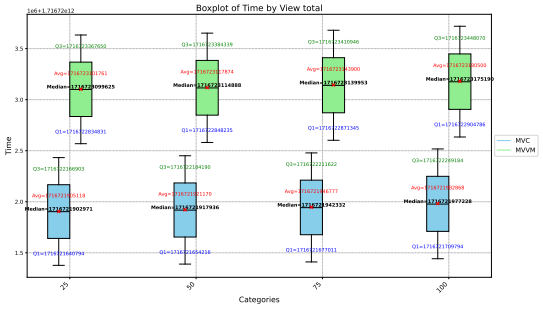


Fig. 12. Perbandingan *Total View to Total Time*

*2) Perbandingan View to total memory:* Hasil perbandingan *View to Total Memory* adalah MVVM lebih unggul dalam penggunaan memori berdasarkan jumlah *view* dibandingkan dengan MVC. Hal ini terlihat pada average memori untuk setiap *view* dalam MVVM (182523212 byte) lebih rendah daripada MVC (269571898 byte) serta diilustrasikan pada figure 13.

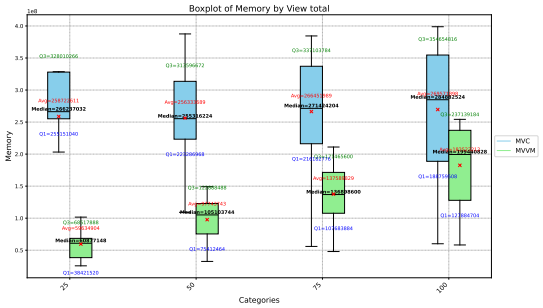


Fig. 13. Perbandingan *Total View to Total Memory*

*3) Perbandingan Spin to total time:* Hasil perbandingan *Spin to Total Time* adalah MVC lebih unggul dan cepat untuk menangani peningkatan jumlah *spin* dibandingkan dengan MVVM. Hal ini terlihat pada average time taken untuk setiap *spin* dalam MVC (1716721944593 millisec) lebih rendah daripada MVVM (1716723141670 millisec) yang dapat dilihat pada figure 14.

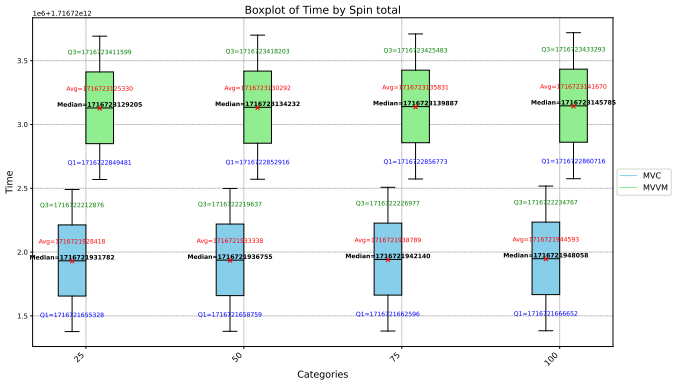


Fig. 14. Perbandingan *Total Spin to Total Time*

*4) Perbandingan Spin to total memory:* Hasil perbandingan *Spin to Total Memory* adalah MVVM lebih unggul dalam penggunaan memori berdasarkan jumlah *spin* dibandingkan dengan MVC. Hal ini terlihat pada average memori untuk setiap *spin* dalam MVVM (155467091 byte) lebih rendah daripada MVC (315097569 byte) serta diilustrasikan pada figure 15.

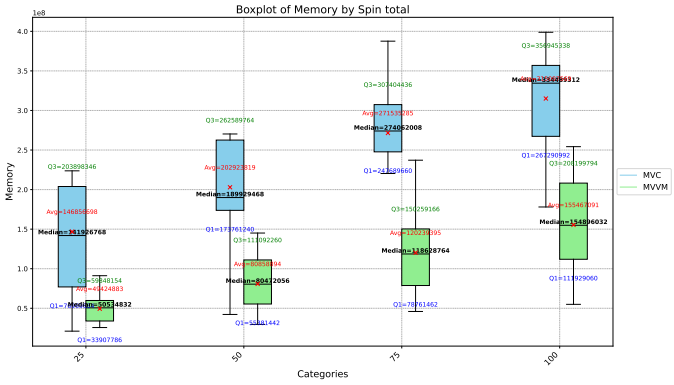


Fig. 15. Perbandingan *Total Spin to Total Memory*

Hasil testing ini adalah MVVM lebih efisien dalam hal waktu dan penggunaan memori saat menangani peningkatan jumlah *view*. Namun, MVC lebih unggul dalam menangani peningkatan jumlah *spin*. Hal yang menarik dari hasil testing ini adalah bahwa alasan mengapa MVC menyelesaikan test dengan total time yang lebih cepat daripada MVVM mungkin disebabkan oleh halnya MVC lebih unggul dalam menangani peningkatan jumlah *spin*.

## V. Kesimpulan

Penelitian ini menyimpulkan bahwa dalam hal seperti kecepatan eksekusi, arsitektur Model-View-ViewModel (MVVM) cenderung lebih cepat dibandingkan dengan Model-View-Controller (MVC). Hal ini terlihat dari waktu eksekusi yang lebih konsisten dan efisien dalam menangani operasi yang melibatkan perubahan data pada banyak tampilan (view). Namun, penggunaan memori oleh MVVM lebih tinggi dibandingkan dengan MVC, kemungkinan disebabkan oleh mekanisme data binding yang memerlukan lebih banyak sumber daya untuk sinkronisasi antara ViewModel dan View. Maka dari hasil itu dapat dikatakan bahwa MVC cocok untuk aplikasi yang membutuhkan arsitektur yang lebih sederhana dan mudah dipahami, sementara MVVM lebih cocok untuk aplikasi kompleks dengan banyak interaksi UI dan yang memerlukan efisiensi dalam pemeliharaan dan pengujian.

Penelitian ini memberikan wawasan berharga tentang kelebihan dan kelemahan masing-masing arsitektur, membantu pengembang untuk membuat keputusan yang lebih baik dalam memilih arsitektur yang tepat untuk proyek mereka, sehingga dapat menghasilkan aplikasi yang lebih efisien dan andal.

## References


[1] Gloria Arcos-Medina, Jorge Menéndez, and Javier Vallejo. Comparative study of performance and productivity of mvc and mvvm design patterns. *KnE Engineering*, pages 241–252, 1 2018.

[2] Arief Rahman Fajri and Septia Rani. Penerapan design pattern mvvm dan clean architecture pada pengembangan aplikasi android (studi kasus: Aplikasi agree partner). *Automata: Disemasi Tugas Akhir Mahasiswa*, (16):8, 2022.

[3] Laberto Kelen. Implementasi model-view-controller (mvc) pada ujian online melalui penerapan framework codeigniter. *Jurnal Pendidikan Teknologi Informasi (JUKANTI)*, 1(1):10–16, Mar. 2018.

[4] Tian Lou. A comparison of android native app architecture – mvc, mvp and mvvm. 2016.

[5] 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):4089–4095, Nov 2020.

[6] Irya Muhammad Riyadhi, Intan Purnamasari, and Kamal Prihandani. Penerapan pola arsitektur mvvm pada perancangan aplikasi pengaduan masyarakat berbasis android. *INFOTECH journal*, 2023.

[7] Bambang Wisnuadhi, Ghifari Munawar, and Ujang Wahyu. Performance comparison of native android application on mvp and mvvm. pages 276–282, 2020.