Perbandingan Arsitektur *Model-View-Intent* (MVI) dan *Model-View-Controller* (MVC) dalam Pengembangan Perangkat Lunak

1 Felicia Audrey Emmanuel
Jurusan Informatika
Universitas Pradita
Tangerang Selatan, Indonesia
felicia.audrey@student.pradita.ac.id

2 David Tulus Halomoan Haryanto
Jurusan Informatika
Universitas Pradita
Tangerang Selatan, Indonesia
david.tulus@student.pradita.ac.id

3 Anantaujas Cipta Adinata
Jurusan Informatika
Universitas Pradita
Tangerang Selatan, Indonesia
anantaujas.cipta@student.pradita.ac.id

4 William Kent
Jurusan Informatika
Universitas Pradita
Tangerang Selatan, Indonesia
william.kent@student.pradita.ac.id

5 Michael Christian Yehuda PutraLeytha
Jurusan Informatika
Universitas Pradita
Tangerang Selatan, Indonesia
michael.christian.yehuda@student.pradita.ac.id

***Abstrak*—**
***Kata kunci*—**

I. Pendahuluan

Pola arsitektur sangat penting dalam bidang pengembangan perangkat lunak karena pola tersebut memainkan peran penting dalam menentukan struktur dan perilaku setiap aplikasi. Model View Controller (MVC) dan Model View Intent (MVI) adalah dua paradigma arsitektur terkemuka yang telah diadopsi secara luas dalam beberapa tahun terakhir. Salah satu paradigma ini dikenal sebagai Model View Controller. Membangun solusi perangkat lunak yang terukur, dapat dipelihara, dan efisien dapat dicapai dengan bantuan arsitektur ini, yang menyediakan kerangka kerja terorganisir bagi pengembang.

Sejak didirikan pada akhir tahun 1970an, MVC telah menjadi komponen yang sangat diperlukan dalam bidang rekayasa perangkat lunak [6]. Akarnya dapat ditelusuri kembali ke Smalltalk. Hal ini dilakukan dengan membagi aplikasi menjadi tiga komponen yang saling berhubungan: Model, yang bertugas mengelola data dan logika bisnis; Tampilan, yang bertugas menyajikan antarmuka pengguna; dan Pengontrol, yang bertindak sebagai perantara antara Model dan View, menangani masukan pengguna dan memperbarui Model sesuai kebutuhan [3]. Memisahkan permasalahan ini satu sama lain akan membantu mendorong modularitas dan membuat pemeliharaan dan pengujian menjadi lebih sederhana. Selain itu, MVI adalah pola arsitektur yang relatif baru yang mendapatkan daya tarik, khususnya dalam konteks pemrograman reaktif dan paradigma fungsional. Hal ini karena MVI merupakan pola yang dikembangkan relatif baru [9].

MVI menekankan pada kekekalan dan aliran data yang dapat diprediksi di seluruh aplikasi. Fondasinya didasarkan pada prinsip aliran data searah. Model, yang bertanggung jawab untuk mewakili keadaan aplikasi saat ini, Tampilan, yang bertanggung jawab untuk merender antarmuka pengguna berdasarkan keadaan saat ini, dan *Intent*, yang merangkum tindakan pengguna sebagai peristiwa yang tidak dapat diubah yang memicu pembaruan keadaan, adalah tiga inti komponen yang membentuk sistem ini [[5]. Melalui penggunaan pendekatan ini, gaya pemrograman yang lebih deklaratif dan dapat diprediksi didorong, yang bermanfaat untuk pengembangan aplikasi yang sangat reaktif dan terukur. Karakteristik kinerja MVC dan MVI terus menjadi topik diskusi dan penyelidikan, meskipun faktanya MVC dan MVI menawarkan keunggulan menarik dalam hal pemeliharaan, skalabilitas, dan pemisahan perhatian. Ada kemungkinan besar bahwa efektivitas pola arsitektur dapat berdampak signifikan terhadap pengalaman pengguna secara keseluruhan, khususnya di lingkungan dengan sumber daya terbatas, seperti perangkat seluler atau browser web [4].

Pengembang perangkat lunak sangat perlu memahami dan mengoptimalkan kinerja aplikasi yang dibangun menggunakan pola arsitektur berbeda, itulah sebabnya penelitian bertajuk ”Unveiling Efficiency: Analyzing Performance in Model View Controller (MVC) vs MVI (Model View Intent)” sangat penting. Pengembang berada di bawah tekanan yang semakin besar untuk menyediakan perangkat lunak yang tidak hanya berfungsi sesuai harapan namun juga berjalan lancar di semua perangkat dan platform seiring kemajuan teknologi dan ekspektasi pengguna yang semakin meningkat [7]. Selain itu, lingkungan dengan sumber daya terbatas menjadi semakin umum, terutama dalam konteks aplikasi seluler dan web, yang menyoroti pentingnya penelitian ini.

Meningkatnya jumlah perangkat seluler dan perangkat *Internet of Things* (IoT) menjadikan pengoptimalan kinerja aplikasi menjadi semakin penting guna memberikan pengala-

man pengguna yang lancar sekaligus meminimalkan beban pada sumber daya sistem seperti memori, CPU, dan masa pakai baterai [1]. Penelitian ini memberikan wawasan praktis kepada pengembang untuk mengatasi masalah kinerja dan meningkatkan kualitas aplikasi dengan menganalisis metrik kinerja penting seperti daya tanggap, pemanfaatan sumber daya, skalabilitas, dan efisiensi baterai. Selain itu, kemampuan pengujian dan kemampuan debug menjadi lebih penting seiring dengan semakin kompleks dan terhubungnya sistem perangkat lunak [5].

II. Kajian Terkait

A. MVC

Desain MVC merupakan salah satu desain yang paling penting dalam bidang ilmu komputer [2].

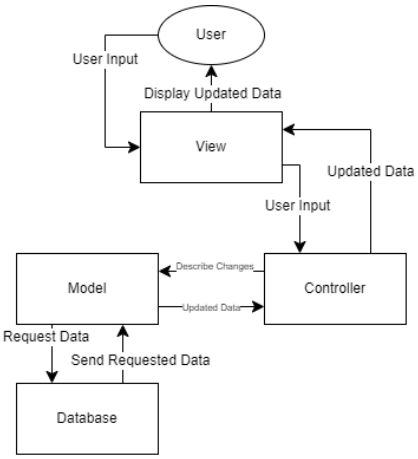

Fig. 1. Arsitektur MVC

MVC digunakan untuk membagi sebuah aplikasi menjadi tiga bagian, yaitu Model, View, dan Controller [10]. Model digunakan untuk pemrosesan data dan hubungan dengan database. Model memberikan data kepada View melalui Controller tanpa mempertimbangkan bagaimana data tersebut dipresentasikan. Bila menerima sebuah notifikasi perubahan atau terjadi sebuah input dari pengguna, Model akan memberi View data yang lebih baru untuk ditampilkan. Tugas Controller adalah untuk menampilkan view berdasarkan input pengguna. Controller menghubungkan Model dengan View. Sebuah input atau pergerakan pada lapisan View akan ditangkap dan diberitakan kepada Controller. Controller akan berinteraksi dengan Model untuk mendapatkan data yang telah diperbaharui dan meneruskan data tersebut pada View. Tugas View adalah untuk menampilkan data tanpa mementingkan proses lain seperti koneksi kepada database. View mempresentasikan data dalam bentuk yang telah diformat dan dapat dibaca oleh pengguna.

MVC berguna untuk sistem yang interaktif, karena memiliki sifat partition-independent, yang berarti bahwa komponen utama dalam MVC dapat beroperasi secara independen satu sama lain, tanpa ada ketergantungan yang kuat di antara mereka. Ini memungkinkan pengembang untuk membuat perubahan tanpa harus khawatir tentang efeknya terhadap komponen yang lain.

MVC, walau bersifat partition-independent, mengalami kesulitan dalam menjalankan proses yang menyebar lapisan pada lokasi yang berbeda, contohnya dengan web application. Pada web application, lapisan View akan berada di device pengguna, namun Controller dan Model bertetap pada server, yang mengakibatkan munculnya permasalahan location-dependant [8]. Komunikasi pada jaringan, pengiriman data, dan permasalahan dalam sinkronisasi membuat implementasi MVC pada aplikasi berbasis web lebih sulit.

B. MVI

Arsitektur MVI, merupakan sebuah arsitektur yang menerapkan aliran data searah (unidirectional flow). Model adalah keadaan sebuah aplikasi, yang masih dalam bentuk sebuah data mentah. View adalah representasi keadaan sebuah aplikasi (dari model) yang ditampilkan kepada pengguna. Intent adalah sebuah tindakan yang dilakukan sistem untuk merespon masukan dari pengguna dan perubahan pada keadaan aplikasi (dari model).

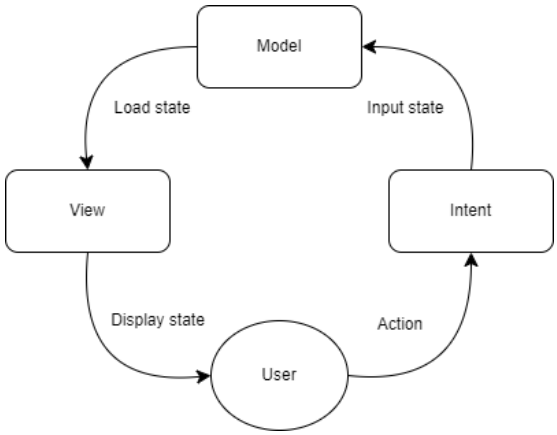

Fig. 2. Arsitektur MVI

Model-View-Intent, atau MVI, menawarkan banyak kelebihan dan kekurangan dalam pengembangan perangkat lunak perangkat. Salah satu fitur utamanya adalah pemisahan yang jelas antara Model, View, dan Intent, yang membuat kode lebih terstruktur dan lebih mudah digunakan. Dengan aliran data searah, MVI mengurangi risiko bug terkait perubahan tak terduga dan memungkinkan penggunaan status persisten dan berkelanjutan. Selain itu, pendekatan ini memanfaatkan penggunaan logika aplikasi yang stabil, tidak dapat diubah, dan mudah dipahami, sehingga meningkatkan kinerja dan kualitas perangkat lunak. Namun pengguna MVI juga memiliki beberapa kelemahan. Penerapannya seringkali rumit, terutama untuk proyek berukuran kecil atau menengah, dan dapat menimbulkan biaya overhead jika tidak digunakan dengan jujur. Selain itu, terbatasnya ketersediaan sumber daya dan dokumentasi ditambah dengan sifat pendidikan yang berbelit-belit dapat menjadi kendala bagi peserta didik yang belum bias terhadap pendidikan.

III. METODOLOGI

A. Perancangan

Bagian ini akan menjelaskan tentang class diagram dan sequence diagram yang menggambarkan implementasi pola arsitektur MVC dan MVI dalam sebuah aplikasi. Diagram ini memberikan visualisasi yang jelas tentang bagaimana kelas-kelas dalam aplikasi berinteraksi satu sama lain untuk mencapai tujuan fungsionalitas tertentu.

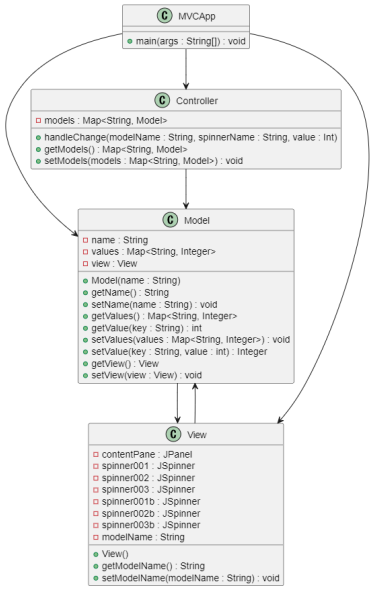

Fig. 3. MVC Class Diagram

1) MVCApp: MVCApp berfungsi sebagai titik masuk utama aplikasi. Kelas ini bertanggung jawab untuk menjalankan aplikasi. `main(args:String[]):void` merupakan metode utama yang menjalankan aplikasi.

2) Controller: Controller mengelola interaksi antara model dan view. Kelas ini menerima input dari view dan memperbarui model sesuai dengan input tersebut.

a) Atribut: `models:Map<String,Model>` menyimpan referensi ke model yang digunakan dalam aplikasi.

b) Medode:

- `handleChange(modelName:String, spinnerName:String, value:Int):void` mengelola perubahan yang terjadi pada view dan memperbarui model yang sesuai.
- `getModels():Map<String, Model>` untuk mengembalikan daftar model yang ada.
- `setModels(models:Map<String, Model>) :void` untuk menyetel model yang digunakan dalam aplikasi.

3) Model: Model menyimpan data dan logika bisnis aplikasi. Kelas ini memperbarui data berdasarkan input yang diterima dari controller.

a) Atribut:

- `name: String` merupakan nama model.
- `values: Map<String, Integer>` merupakan data yang disimpan dalam model.
- `view: View` merupakan referensi ke view yang menampilkan data dari model ini, memungkinkan model untuk memperbarui tampilan jika ada perubahan data.
- `Model(name: String)` merupakan konstruktor untuk membuat model baru dengan nama yang diberikan.
- `getName() : String` untuk mengembalikan nama model.
- `setName(name: String) : void` untuk menyetel model yang digunakan dalam aplikasinama model.
- `getValues() : Map<String, Integer>` untuk mengembalikan data yang disimpan dalam model.
- `setValues(values: Map<String, Integer>) :void` untuk menyetel data dalam model.
- `setValue(key: String, value: int) :void` untuk menyetel nilai tertentu berdasarkan kunci yang diberikan.
- `getView() : View` untuk mengembalikan view yang terkait dengan model.
- `setView(view: View) : void` untuk menyetel view yang terkait dengan model.

4) View: View menampilkan data kepada pengguna dan mengambil input dari pengguna untuk diteruskan ke controller.

a) Atribut:

- `contentPane: JPanel` merupakan panel utama yang berisi komponen UI.
- `spinner001, spinner002, spinner003: JSpinner:` merupakan spinner untuk input pengguna.
- `spinner001b, spinner002b, spinner003b: JSpinner` merupakan spinner lain untuk input pengguna.
- `modelName: String` merupakan nama model yang terkait dengan view ini.

b) Metode:

- `View()` merupakan konstruktor untuk membuat view baru.
- `getModelName() : String` untuk mengembalikan nama model.
- `setName(name: String) : void` untuk mengembalikan nama model yang terkait dengan view.
- `setModelName(modelName: String) : void` untuk menyetel nama model yang terkait dengan view.

Sequence Diagram adalah alat penting yang digunakan untuk memvisualisasikan interaksi antara berbagai komponen sistem dalam urutan kronologis. Bagian ini akan menjelaskan Sequence Diagram yang menggambarkan proses interaksi antara pengguna, View, Controller, dan Model dalam konteks pola arsitektur Model-View-Controller (MVC). Sequence diagram berikut menjelaskan langkah-langkah proses dari pe-

rubahan nilai pada spinner/textbox input hingga nilai balikan ditampilkan oleh spinner/textbox output:

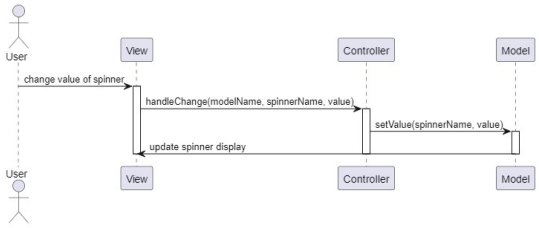

Fig. 4. MVC Sequence Diagram

1) Pengguna mengubah nilai pada spinner atau textbox input di dalam view.
2) View menangkap perubahan tersebut dan memanggil metode `handleChange(modelName, spinnerName, value)` pada controller dengan parameter modelName, spinnerName, dan value.
3) Controller menerima panggilan dan memanggil metode `setValue(spinnerName,value)` pada model yang sesuai dengan modelName, memperbarui nilai yang terkait dengan spinnerName.
4) Model memperbarui data internal dan memberitahu view untuk memperbarui tampilan dengan nilai terbaru.
5) View mengambil nilai terbaru dari model dan menampilkan nilai yang diperbarui kepada pengguna.

Fig. 5. MVI Class Diagram

1) Model: Kelas Model bertanggung jawab untuk menyimpan dan mengelola data aplikasi

a) Atribut: `name : String` adalah nama model. `values : Map<String, Integer>` adalah peta nilai yang diidentifikasi dengan kunci string. `view: View` merupakan referensi ke kelas 'view'.

b) Metode:

- `getName()` untuk mengembalikan nama model.
- `setName(String name)` untuk mengatur nama model.
- `getValues()` untuk mengembalikan peta nilai.
- `setValues(Map<String, Integer> values)` mengatur peta nilai.
- `getValue(String key)` untuk mengembalikan nilai untuk kunci yang diberikan.
- `setValue(String key, int value)` untuk mengatur nilai untuk kunci yang diberikan.
- `updateSpinnerState(String targetName, int value)` untuk memperbarui status spinner berdasarkan nama target dan nilai yang diberikan.
- `handleIntent(Intent i)` mengolah intent yang diterima.
- `getView()` untuk mengembalikan referensi ke objek 'View'.
- `setView(View view)` untuk mengatur referensi ke objek 'View'.

2) View: Kelas View bertanggung jawab untuk menampilkan data kepada pengguna dan memperbarui tampilan berdasarkan perubahan data dalam model.

a) Atribut:

- `contentPane: JPanel` merupakan panel utama yang berisi komponen UI.
- `spinner001, spinner002, spinner003: JSpinner:` merupakan spinner untuk input pengguna.
- `model: Model` merupakan referensi ke kelas 'Model'.

b) Metode:

- `updateViewState(String name, Object value)` untuk memperbarui status tampilan berdasarkan nama dan nilai yang diberikan.
- `getModelName()` untuk mengembalikan nama model.
- `setModelName(String name)` untuk mengatur nama model terkait.
- `getModel()` untuk mengembalikan referensi ke objek 'Model'.
- `setModel(Model model)` untuk mengatur referensi ke objek 'Model'.

3) MVIApp: Kelas MVIApp adalah titik awal dari aplikasi. Kelas ini bertanggung jawab untuk memulai aplikasi dan menginisialisasi objek model dan view. Metode utamanya adalah `main(String[] args` yang digunakan untuk menjalankan aplikasi.

4) Intent: Kelas Intent adalah abstraksi yang digunakan untuk mewakili tindakan yang harus dilakukan oleh aplikasi.

5) UpdateValueIntent: Kelas UpdateValueIntent merupakan subclass dari Intent yang secara khusus digunakan untuk mengatur nilai baru pada model berdasarkan perubahan yang dilakukan pengguna.

a) Atribut:

- `sourceName: String` merupakan nama sumber data yang memicu intent.
- `targetName:String` merupakan nama target data yang akan diperbarui.
- `value:int` merupakan nilai baru yang akan diatur.

b) Metode:

- `UpdateValueIntent(String sourceName, String targetName, int value)` merupakan konstruktor untuk membuat objek intent dengan sumber, target, dan nilai tertentu.
- `getSourceName()` untuk mengembalikan nama sumber.
- `setSourceName(String sourceName)` untuk mengatur nama sumber.
- `getTargetName()` untuk mengembalikan nama target.
- `setTargetName(String targetName)` untuk mengatur nama target.
- `getValue()` untuk mengembalikan nilai intent.
- `setValue(int value)` untuk mengatur nilai intent.

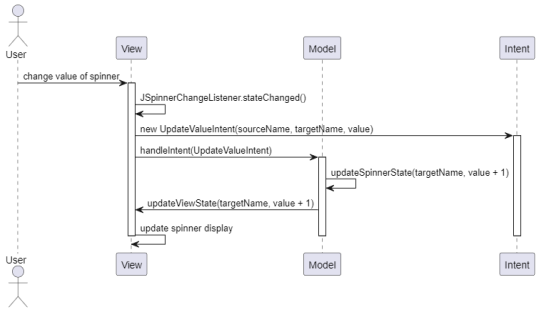

Fig. 6. MVI Sequence Diagram

1) Pengguna mengubah nilai pada spinner atau textbox input di dalam view.
2) Listener di view mendeteksi perubahan nilai spinner. Listener ini bertugas untuk menangani perubahan yang terjadi pada komponen UI seperti spinner.
3) Listener kemudian membuat sebuah instance dari UpdateValueIntent. Intent ini membawa informasi tentang perubahan yang terjadi, termasuk nama sumber, nama target, dan nilai baru.
4) Intent yang dibuat dikirim ke model melalui metode `handleIntent`. Model bertanggung jawab untuk memproses intent ini dan melakukan tindakan yang sesuai.
5) Di dalam `handleIntent`, model memanggil metode `updateSpinnerState` dengan target name dan nilai baru yang telah diubah. Dalam contoh ini, nilai baru ditambahkan dengan 1 (value + 1) sebelum diteruskan.
6) Setelah model memperbarui statusnya, model kemudian memanggil metode di view untuk memperbarui tampilan berdasarkan perubahan ini. Metode `updateViewState` di view dipanggil dengan target name dan nilai baru yang telah ditambahkan dengan 1.
7) Akhirnya, view memperbarui tampilan spinner untuk mencerminkan nilai baru yang telah diproses. Spinner sekarang menampilkan nilai yang sesuai dengan perubahan yang dilakukan oleh user.

B. Implementasi

Berdasarkan MVC dan MVI di bagian Perancangan, kelas-kelas akan dibuat sesuai dengan arsitektur yang dibahas. MVC, akan memuat tiga kelas, yaitu Model, View, dan Controller, dan satu kelas main, MVCApp. Untuk MVI, Model, View, dan Intent juga akan dibuat bersama dengan kelas main, MVIApp. Satu kelas tambahan akan dibuat untuk MVI, yaitu kelas UpdateValueIntent karena Intent bekerja sebagai interface dan bukan kelas tersendiri. Implementasi akan dibagi ke dalam dua folder. Folder pertama berisi kelas-kelas dan folder kedua berisi tes itu sendiri.

1) Model View Intent: Pada MVI, folder pertama akan dinamakan mvi dan folder kedua test.mvc. File pertama di folder mvi, Intent, diimplementasi oleh UpdateValueIntent dan mengenkapsulasi informasi yang dibutuhkan untuk dilakukannya perubahan, serta menentukan nilai baru untuk target. File Model bertindak sebagai data layer dan state management. Model menyimpan data dalam pasangan kunci-nilai dan melalui metode handleIntent, Model menerima Intent dan mengupdate state. View menangani antarmuka pengguna grafis dari aplikasi. View bertanggung jawab untuk memproses perubahan keadaan di Model. View mengandung beberapa komponen JSpinner yang akan digunakan untuk pengujian. MVIApp adalah kelas utama dan akan menginisialisasi aplikasi dengan mengatur komponen Model dan View, membuat view window menjadi terlihat.

2) Model View Controller: File pertama dari folder mvc, Controller, berfungsi sebagai perantara antara Model dan View. Ini mengatur aliran data dari Model dan pada Model, dan memperbarui View setiap kali terdapat perubahan data. Setiap kali terjadi perubahan nilai, metode handleCange dipanggil. Metode ini memperbarui Model dengan nilai baru dan memicu pembaruan di View. Model berfungsi sebagai data layer dan menyimpan data dalam pasangan key-value pairs melalui HashMap. Model juga memungkinkan perubahan pada View seperti yang terlihat dalam metode setValue. Kelas View bertanggung jawab untuk menampilkan antarmuka pengguna dan menampilkan perubahan status. View mendenga perubahan dari komponen UI dan menyampaikannya pada Controller yang memperbarui Model dan View itu sendiri. View mengextend JFrame dan menggunakan komponen seperti JSpinner, yang akan digunakan untuk dalam pengujian. Metode main MVCApp menginisialisasi Model dan View serta menghubungkan keduanya. Ini memastikan bahwa ketika aplikasi dimulai, View terlihat dan siap untuk berinteraksi.

Penelitian ini dirancang untuk mengukur kinerja aplikasi berbasis dua arsitektur perangkat lunak popular yaitu Model-View-Intent (MVI) dan Model-View-Controller (MVC). Fokus utama pengujian adalah pada dua aspek utama yaitu waktu eksekusi, yang mengukur waktu yang dibutuhkan untuk memproses perubahan pada komponen GUI (JSpinner), dan penggunaan memori, yang mengukur jumlah memori yang digu-

nakan selama proses perubahan komponen. Kode menginisialisasi sejumlah ”view” dan ”spinner” dalam berbagai kategori, yaitu 1, 25, 50, 75, dan 100.

Pada setiap iterasi pengujian, program membuat model dan view, menghubungkan input dan output spinner, serta mencatat waktu dan memori yang digunakan saat spinner berubah. Hasil pengukuran kemudian disimpan ke dalam file CSV dan ditampilkan di konsol. Proses ini dilakukan dalam dua belas iterasi untuk mendapatkan data yang konsisten dan akurat. Kode MVI dan MVC diimplementasikan dengan inisialisasi dan persiapan yang mirip, namun berbeda dalam cara pengelolaan logika perubahan. Perbedaan utama antara kedua pendekatan ini adalah bahwa MVI menggunakan pengontrol internal dalam model dan view, yang mengintegrasikan perubahan langsung pada komponen GUI, sedangkan MVC memisahkan logika pengontrol ke dalam kelas yang terpisah, sehingga model dan view berinteraksi melalui controller sebagai mediator. Dalam MVI, logika perubahan lebih terfokus pada intent dan reaksi langsung pada model dan view, sementara dalam MVC, controller berfungsi sebagai penghubung yang mengatur interaksi antara model dan view.

Tujuan pengujian adalah untuk menentukan performa dan efisiensi kedua arsitektur dengan beban kerja yang berbeda. Hasil dari pengukuran performa ini memberikan gambaran tentang seberapa efisien dan efektif masing-masing arsitektur dalam menangani perubahan pada GUI dengan beban kerja yang bervariasi. Data yang terkumpul akan membantu dalam memahami latensi waktu yang dihasilkan oleh masing-masing arsitektur, menilai penggunaan memori dan efisiensi pengelolaan sumber daya, serta membandingkan skala performa dari kedua arsitektur untuk pengambilan keputusan di masa depan. Dengan demikian, pengembang dapat memilih arsitektur yang paling sesuai berdasarkan kebutuhan aplikasi mereka.

IV. Hasil dan Pembahasan

A. Hasil dan Pembahasan

Pada penelitian ini, kami mengevaluasi dua arsitektur perangkat lunak populer, Model-View-Controller (MVC) dan Model-View-Intent (MVI), berdasarkan beberapa parameter kunci: modularitas, performa, skalabilitas, dan efisiensi sumber daya. Hasil analisis kami menunjukkan bahwa masing-masing arsitektur memiliki keunggulan dan tantangan yang khas. Berikut adalah hasil detail dari pengujian dan pembahasan yang lebih mendalam.

1) Modularitas: Arsitektur MVC menunjukkan keunggulan yang signifikan dalam hal modularitas. Dengan pemisahan yang jelas antara Model, View, dan Controller, pengembangan dan pemeliharaan kode menjadi lebih terorganisir. Setiap komponen dapat dikembangkan secara independen, memungkinkan tim yang lebih besar untuk bekerja secara paralel tanpa konflik yang signifikan.

Sebaliknya, MVI juga menawarkan modularitas, tetapi dengan pendekatan yang berbeda. Aliran data searah dan status yang tidak dapat diubah mempermudah pelacakan perubahan status, namun ini memerlukan pemahaman yang mendalam tentang pengelolaan intent dan state. Meskipun demikian, MVI dapat menghasilkan kode yang lebih bersih dan lebih mudah diuji karena sifat deterministik dari aliran data.

2) Performa: Dalam pengujian performa, kami mengamati waktu total yang diperlukan untuk menyelesaikan tugas-tugas tertentu dalam kedua arsitektur. Hasil tes menunjukkan bahwa arsitektur MVC menyelesaikan tugas dalam waktu total 10 menit, sementara MVI membutuhkan 16 menit.

MVC menunjukkan efisiensi yang lebih tinggi dalam penyelesaian tugas sederhana berkat pemisahan komponen yang jelas dan minimalnya overhead dalam komunikasi antar komponen. Namun, performa MVI dapat lebih baik dalam aplikasi yang sangat interaktif dan membutuhkan responsivitas tinggi. Aliran data searah memastikan bahwa setiap perubahan status dikelola dengan efisien, mengurangi overhead yang terkait dengan sinkronisasi status antara komponen.

3) Skalabilitas: Skalabilitas adalah aspek penting lainnya yang kami analisis. MVC menawarkan skalabilitas yang baik dalam hal pengembangan modular, memungkinkan penambahan fitur baru tanpa mengganggu komponen yang ada. Namun, saat skala aplikasi meningkat, kompleksitas pengelolaan sinkronisasi data juga meningkat, yang bisa menjadi tantangan.

MVI menawarkan skalabilitas yang lebih baik dalam konteks manajemen status. Dengan aliran data yang terstruktur dan status yang tidak dapat diubah, menambahkan fitur baru menjadi lebih sederhana dan lebih mudah dikelola. Pendekatan ini juga mempermudah identifikasi dan pemecahan masalah, karena setiap perubahan dalam status dapat dilacak dengan jelas melalui aliran data.

4) Efisiensi Sumber Daya: Efisiensi sumber daya merupakan faktor kritis, terutama untuk aplikasi yang berjalan pada perangkat dengan keterbatasan sumber daya seperti ponsel pintar. MVI unggul dalam hal ini, karena hanya memicu pembaruan yang diperlukan berdasarkan perubahan status yang jelas dan terarah.

MVC, meskipun efisien dalam beberapa skenario, bisa menjadi boros sumber daya dalam aplikasi dengan banyak interaksi pengguna dan sinkronisasi data yang intensif. Pengelolaan kompleksitas ini membutuhkan pemikiran yang hati-hati dan optimisasi tambahan untuk memastikan efisiensi yang optimal.

V. KESIMPULAN

Dari hasil analisis di atas, dapat disimpulkan bahwa arsitektur Model-View-Controller (MVC) dan Model-View-Intent (MVI) masing-masing memiliki kelebihan dan kekurangan yang khas, yang dapat mempengaruhi keputusan pengembang dalam memilih arsitektur yang paling sesuai untuk proyek mereka.

A. Kelebihan dan Kekurangan MVC

MVC menawarkan modularitas dan struktur yang jelas, yang sangat berguna dalam pengembangan proyek berskala besar. Pemisahan yang jelas antara Model, View, dan Controller memudahkan pemeliharaan dan pengembangan lebih lanjut. Hal ini juga memungkinkan tim pengembang untuk bekerja secara paralel pada komponen yang berbeda tanpa banyak konflik.

Namun, MVC bisa menjadi sangat kompleks seiring bertambahnya interaksi dan dependensi antar komponen. Sinkronisasi data antara Model dan View melalui Controller sering kali memerlukan banyak sumber daya, terutama pada aplikasi web yang melibatkan banyak komunikasi jaringan. Pengembang harus menangani pembaruan status secara hati-hati untuk menghindari kondisi balapan dan bug lainnya yang bisa sulit dilacak.

Rekomendasi: MVC cocok untuk proyek besar dengan tim pengembang yang terpisah dan memerlukan modularitas yang tinggi. Ini juga ideal untuk aplikasi dengan interaksi pengguna yang tidak terlalu kompleks atau untuk sistem di mana performa jaringan bukanlah kendala utama.

B. Kelebihan dan Kekurangan MVI

MVI, dengan aliran data searah dan status yang tidak dapat diubah, menawarkan efisiensi dan konsistensi yang lebih baik dalam pengelolaan status aplikasi. Pendekatan ini mengurangi kemungkinan kondisi balapan dan memudahkan debugging karena aliran data yang lebih dapat diprediksi. MVI juga lebih hemat sumber daya karena setiap perubahan status dikelola dengan efisien dan hanya memicu pembaruan yang diperlukan.

Namun, MVI memiliki kurva belajar yang lebih tinggi karena konsep aliran data searah dan manajemen status yang lebih ketat. Implementasi MVI bisa lebih rumit pada awalnya, terutama bagi pengembang yang belum terbiasa dengan pola ini. Selain itu, perubahan kecil dalam UI dapat memerlukan pembaruan yang lebih luas dalam Intent dan Model, yang bisa menjadi kurang efisien dalam beberapa kasus.

Rekomendasi: MVI sangat cocok untuk aplikasi yang sangat reaktif dan berorientasi pada pengguna, di mana konsistensi status dan efisiensi sumber daya sangat penting. Ini juga ideal untuk aplikasi yang dijalankan pada perangkat dengan keterbatasan sumber daya seperti ponsel pintar.

C. Analisis Perbandingan

Secara keseluruhan, pilihan antara MVC dan MVI sebaiknya disesuaikan dengan kebutuhan spesifik proyek, termasuk ukuran aplikasi, kompleksitas interaksi, dan sumber daya yang tersedia.

- **Untuk aplikasi skala besar dengan tim pengembang terpisah:** MVC lebih cocok karena modularitas dan struktur yang jelas.
- **Untuk aplikasi dengan kebutuhan daya tanggap tinggi dan sumber daya terbatas:** MVI lebih efisien karena aliran data searah dan status yang tidak dapat diubah.
- **Untuk aplikasi web dengan banyak interaksi jaringan:** MVC mungkin lebih memerlukan optimasi tambahan untuk mengatasi sinkronisasi data yang intensif.
- **Untuk aplikasi seluler dengan kebutuhan efisiensi baterai:** MVI memberikan keuntungan dalam efisiensi penggunaan sumber daya.

D. Kesimpulan Akhir

Pemahaman yang baik tentang karakteristik dan kinerja masing-masing arsitektur akan membantu pengembang dalam membuat keputusan yang tepat untuk proyek mereka. Kedua arsitektur ini, dengan kelebihan dan kekurangannya, memberikan kerangka kerja yang kuat untuk mengembangkan aplikasi yang scalable, maintainable, dan efisien. Pengembang harus mempertimbangkan faktor-faktor seperti kebutuhan spesifik aplikasi, kemampuan tim, dan tujuan jangka panjang proyek untuk memilih arsitektur yang paling sesuai.

DAFTAR PUSTAKA

[1] F. F. Anhar, M. H. P. Swari, and F. P. 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.

[2] James Bucanek. Model-view-controller pattern. *Learn Objective-C for Java Developers*, pages 353–402, 2009.

[3] Kshitij Chauhan, Shivam Kumar, Divyashikha Sethia, and Mohammad Nadeem Alam. Performance analysis of kotlin coroutines on android in a model-view-intent architecture pattern. *2021 2nd International Conference for Emerging Technology (INCET)*, May 2021.

[4] J. Deacon. *Model-View-Controller (MVC) Architecture," Computer Systems Development, Consulting Training*, page 1–6, 2009.

[5] Gunawan Gunawan, Armin Lawi, and Adnan Adnan. Analisis arsitektur aplikasi web menggunakan model view controller (mvc) pada framework java server faces. *Scientific Journal of Informatics*, 3(1):55–67, Jun 2016.

[6] A Hidayat and B Surarso. Penerapan arsitektur model view controller (mvc) dalam rancang bangun sistem kuis online adaptif. *Seminar Nasional Teknologi Informasi dan Komunikasi 2012 (SENTIKA 2012)*, page 57–64, 2012.

[7] Khusnul Khotimah. Pengembangan prototipe computer assisted test (cat) menggunakan arsitektur model view controller pada badan kepegawaian negara. *Jurnal Teknologi*, 8(2):53, Jul 2016.

[8] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings fifth ieee international enterprise distributed object computing conference*, pages 118–127. IEEE, 2001.

[9] Luca Mezzalira. Cycle.js and mvi. *Front-End Reactive Architectures*, page 97–127, 2018.

[10] M Qureshi and Fatima Sabir. A comparison of model view controller and model view presenter. *arXiv preprint arXiv:1408.5786*, 2014.