PROSIDING Seminar Nasional SISFOTEK ISSN 2597-3584 (Media Online)
Sistem Informasi dan Teknologi
website : www.seminar.iaii.or.id | email : seminar@iaii.or.id

Implementasi MVVM Dan Framework Jetpack Compose Pada Aplikasi Hukum Berbasis Android

Aziz Musthafa[1], Dihin Muriyatmoko[2], Abdullah Hikmatyar Priandika[3*]
[1,2,3]Teknik Informatika, Saints dan Teknologi, Universitas Darussalam Gontor
abdullahpriandika42001@mhs.unida.gontor.ac.id

Abstract

This research discusses the development of Android-based legal applications using Model-View-ViewModel (MVVM) architecture and Jetpack Compose framework. The purpose of this research is to improve the access and understanding of the Indonesian people to legal aid through an efficient and user-friendly mobile application. The research method used is Software Development Life Cycle (SDLC) with a waterfall model that includes the stages of needs analysis, design, implementation, testing, and maintenance. Data was collected through a questionnaire distributed to 107 respondents aged 17-40 years to measure their understanding of the law and access to legal aid. The results of the analysis showed that 67% of the respondents knew how to get legal aid, but did not know the details of how. The developed app has several key features, such as a list of laws and regulations, AI Q&A, and lawyer consultation. The implementation of Jetpack Compose on the login, register, home, chatbot, regulations, and regulation details pages shows efficiency and ease in developing an interactive and responsive user interface. Application testing is carried out through verification by media experts, application users, and black box tests. The test results show that the application functions properly and meets user needs. The conclusion of this research is that the use of MVVM architecture and Jetpack Compose can speed up the development process and facilitate the maintenance of Android-based legal applications. This application is expected to provide practical and effective solutions in obtaining legal assistance and increasing legal understanding among the people of Indonesia.

Keywords: *android, MVVM, jetpack compose, kotlin, law*

Abstrak

Penelitian ini membahas pengembangan aplikasi hukum berbasis Android dengan menggunakan arsitektur Model-View-ViewModel (MVVM) dan framework Jetpack Compose. Tujuan dari penelitian ini adalah untuk meningkatkan akses dan pemahaman masyarakat Indonesia terhadap bantuan hukum melalui aplikasi mobile yang efisien dan user-friendly. Metode penelitian yang digunakan adalah Software Development Life Cycle (SDLC) dengan model waterfall yang mencakup tahap analisis kebutuhan, desain, implementasi, pengujian, dan pemeliharaan. Data dikumpulkan melalui kuesioner yang disebarkan kepada 107 responden berusia 17-40 tahun untuk mengukur pemahaman mereka tentang hukum dan akses terhadap bantuan hukum. Hasil analisis menunjukkan bahwa 67% dari responden mengetahui cara mendapatkan bantuan hukum, tetapi tidak tahu detail caranya. Aplikasi yang dikembangkan memiliki beberapa fitur utama, seperti daftar peraturan undang-undang, tanya jawab AI, dan konsultasi pengacara. Implementasi Jetpack Compose pada halaman login, register, home, chatbot, peraturan, dan detail peraturan menunjukkan efisiensi dan kemudahan dalam pengembangan antarmuka pengguna yang interaktif dan responsif. Uji coba aplikasi dilakukan melalui verifikasi oleh ahli media, pengguna aplikasi, dan uji black box. Hasil pengujian menunjukkan bahwa aplikasi berfungsi dengan baik dan memenuhi kebutuhan pengguna. Kesimpulan dari penelitian ini adalah bahwa penggunaan arsitektur MVVM dan Jetpack Compose dapat mempercepat proses pengembangan dan memudahkan pemeliharaan aplikasi hukum berbasis Android. Aplikasi ini diharapkan dapat memberikan solusi praktis dan efektif dalam mendapatkan bantuan hukum serta meningkatkan pemahaman hukum di kalangan masyarakat Indonesia.

Kata kunci: android, MVVM, jetpack compose, kotlin, hukum

1. Pendahuluan

Negara Indonesia adalah negara hukum, sebagaimana dinyatakan dalam Pasal 1 ayat 3 UUD 1945, yang berarti bahwa supremasi hukum dijunjung tinggi untuk menjamin kebenaran dan keadilan. Dalam prinsip negara hukum, terdapat tiga prinsip dasar yang harus dipatuhi: supremasi hukum, kesetaraan di hadapan hukum, dan penegakan hukum yang tidak bertentangan dengan hukum itu sendiri[1]. Hukum memainkan peran penting dalam mengatur hubungan antara individu dan masyarakat, memastikan keseimbangan kepentingan, dan melindungi masyarakat dari penguasa yang tirani atau totaliter[2]. Namun, berdasarkan data yang diambil dari seluruh pengacara di Indonesia, terdapat banyak sekali perkara hukum, dan sekitar 67,3% dari 107 responden mengetahui cara mendapatkan bantuan hukum tetapi tidak tahu detail caranya. Dan Pada saat

ini penggunaan buku-buku yang berisi istilah dan undang-undang tentang hukum di Indonesia masih banyak digunakan untuk mendapatkan informasi, tetapi dengan demikian menimbulkan kelemahan dari sisi kepraktisan[3].

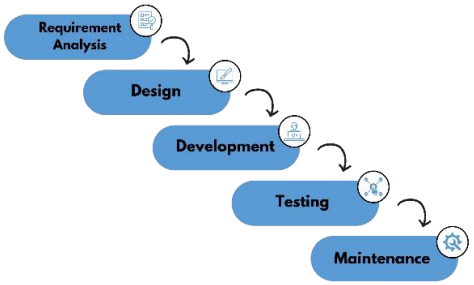
Kemajuan teknologi informasi dan komunikasi telah memungkinkan pengembangan layanan yang lebih efisien melalui aplikasi dan web browser, menggantikan sistem manual yang sebelumnya digunakan. Perkembangan pesat teknologi mobile, khususnya smartphone Android, membuka peluang baru untuk meningkatkan kualitas pelayanan dan bantuan hukum.[4] Arsitektur Model-View-ViewModel (MVVM) telah terbukti memudahkan pengembangan dan perawatan aplikasi dengan memisahkan tampilan, logika bisnis, dan model data. Bedasarkan penelitian Muhammad Syakir Arif, MVVM mengedepankan pemisahan tanggung jawab antara tampilan antarmuka grafis dengan logika bisnis, sehingga pengembangan dan pemeliharaan aplikasi menjadi lebih mudah dan terstruktur[5]. Selain itu, Android Jetpack Compose menawarkan cara yang lebih cepat dan mudah untuk mendesain antarmuka pengguna yang interaktif dan responsif[6].

Penelitian ini diadakan untuk mengatasi permasalahan pengetahuan dan akses terhadap bantuan hukum di kalangan masyarakat Indonesia. Dengan memanfaatkan teknologi modern dan arsitektur aplikasi yang efisien, diharapkan aplikasi ini dapat memberikan solusi yang praktis dan efektif dalam mendapatkan bantuan hukum. Selain itu, aplikasi ini bertujuan untuk meningkatkan pemahaman hukum masyarakat dan memberikan kemudahan dalam mengakses informasi serta layanan hukum.

Dan Penelitian ini bertujuan untuk menjawab pertanyaan - pertanyaan seperti bagaimana mengembangkan aplikasi hukum yang memanfaatkan pola arsitektur Model-View-ViewModel (MVVM), bagaimana penggunaan Android Jetpack Compose dapat mempercepat dan mempermudah proses pengembangan UI aplikasi hukum, dan bagaimana aplikasi ini dapat membantu masyarakat Indonesia dalam memperoleh bantuan hukum yang mereka butuhkan dengan lebih efektif dan fleksibel. Dengan menjawab pertanyaan-pertanyaan tersebut, penelitian ini diharapkan dapat memberikan kontribusi yang signifikan dalam meningkatkan akses dan kualitas bantuan hukum di Indonesia melalui pemanfaatan teknologi modern.

2. Metode Penelitian

Tahapan pada penelitian yang di gunakan untuk mengembangkan aplikasi adalah metode Software Development Life Cycle (SDLC) dengan model waterfall. Metode waterfall adalah metode kerja yang menekankan fase-fase yang berurutan dan sistematis. Dan juga contoh proses yang digerakkan oleh rencana. Pada prinsipnya, peneliti merencanakan dan menjadwalkan semua aktivitas proses sebelum memulai pengembangan perangkat lunak.[7] Disebut waterfall karena proses mengalir satu arah “ke bawah” seperti air terjun. Metode waterfall ini harus dilakukan secara berurutan sesuai dengan tahap yang ada. Berikut adalah Metode *Waterfall* yang tertera pada Gambar 1.



Gambar 1. Tahapan Penelitian dengan metode waterfall

2.1 Requirement Analysis

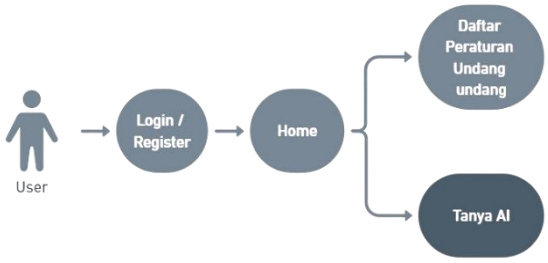
Pada tahap analisis kebutuhan, peneliti memulai studi dengan menyebarkan kuesioner kepada masyarakat Indonesia yang berumur 17 – 40 tahun untuk mengukur pemahaman mereka tentang hukum dan akses terhadap bantuan hukum. Pada tahap ini, peneliti menganalisis keluhan yang disampaikan oleh pengguna dan mengumpulkan data pendukung terkait permasalahan yang dihadapi. Setelah itu, solusi untuk permasalahan tersebut dicari. Hasil analisis menunjukkan bahwa sekitar kurang lebih 67,3% dari 107 responden dari questioner yang di sebar mengetahui cara mendapatkan bantuan hukum dari para ahli akan tetapi tidak tahu detail cara mendapatkannya. Berdasarkan temuan ini, pengguna sangat mendukung pengembangan aplikasi hukum berbasis mobile yang diharapkan dapat membantu dalam memahami hukum dan memperoleh bantuan hukum.

2.2 Design

Pada tahap design, dilakukan setelah analisis masalah dan kebutuhan pengguna selesai. Sebelum memulai proses pengkodean, sangat penting untuk melaksanakan langkah-langkah analisis sistem dan desain aplikasi terlebih dahulu. Dalam tahap perancangan desain, terdapat beberapa elemen yang perlu dimasukkan untuk memandu pengembangan aplikasi secara efektif dan efisien. Elemen-elemen tersebut meliputi pembuatan *Flow Chart* yang akan menggambarkan alur kerja sistem secara keseluruhan, *Use Case* diagram yang akan mengidentifikasi berbagai skenario penggunaan aplikasi oleh pengguna, *Mockup* yang memberikan gambaran visual tentang tampilan antarmuka pengguna, serta pemetaan aplikasi yang akan menjelaskan struktur dan hubungan antar komponen dalam aplikasi. Dengan adanya elemen-elemen ini, proses pengembangan aplikasi akan menjadi lebih terarah dan terstruktur, sehingga menghasilkan aplikasi yang sesuai dengan kebutuhan dan harapan pengguna.

Use case diagram adalah sebuah pemodelan yang menggambarkan hubungan (*relationship*) antara satu atau lebih aktivitas dengan sistem informasi yang dirancang.[8]

Gambar 2 merupakan *usse case diagram* yang menggambarkan sistem pada aplikasi.



Gambar 2. Use Case Diagram

Flowchart merupakan Representasi grafis dari langkah-langkah dan urutan prosedur dalam program.[9]

Alur *flowchart* aplikasi dijelaskan pada Gambar 3.

Gambar 3. Flowchart Aplikasi

Penjelasan alur:

Pertama user akan masuk kedalam halaman login, user melakukan login jika sudah memiliki akun, dan jika blom memiliki akun user diwajibkan untuk melakukan registrasi.

Kedua, Ketika sudah melakukan login, user akan diarahkan ke halaman home yang mana terdapat tiga fitur utama yaitu: fitur daftar peraturan undang – undang, fitur tanya AI, dan fitur konsultasi pengacara.

Pada fitur daftar peraturan undang – undang, didalamnya terdapat beberapa daftar peraturan undang – undang dan Ketika user memilih salah satu peraturan tersebut maka user akan di bawa ke halaman detail peraturan.

Pada fitur tanya AI, user dapat menanyakan permasalahan mengenai hukum dan langsung akan mendapatkan respon dari pertanyaan tersebut. Dan jika pertanyaan di luar permasalahan hukum maka pertanyaan tersebut tidak akan mendapatkan respon.

2.3 Development

Tahap implementasi merupakan langkah berikutnya setelah tahap desain dalam proses pengembangan aplikasi. Pada tahap ini, peneliti akan mengubah desain yang telah dirancang menjadi program fungsional sesuai dengan spesifikasi yang telah dianalisis dan direncanakan sebelumnya. Implementasi adalah fase kritis di mana konsep dan sketsa yang telah dibuat selama fase desain diubah menjadi kode yang berfungsi dan dapat dieksekusi. Dalam penelitian ini, peneliti menggunakan Android Studio sebagai lingkungan pengembangan terintegrasi (IDE) untuk mengembangkan aplikasi. Bahasa pemrograman yang digunakan untuk pengkodean adalah Kotlin. Kotlin dipilih karena memiliki berbagai keunggulan seperti sintaks yang lebih bersih, interoperabilitas dengan Java, dan dukungan yang kuat dari komunitas pengembang Android[10]. dan dengan implementasi pola arsitektur *Model – View – ViewModel* (MVVM) dan *Jetpack Compose* sebagai *UI Toolkit.*

2.4 Testing

Verifikasi merupakan langkah penting setelah tahap pengkodean aplikasi dan sebelum peluncurannya, dilakukan melalui tiga tahap uji: pengguna aplikasi, uji hardware, dan melalui uji black box. Ahli media, seperti dosen Teknik Informatika, mengevaluasi menggunakan kuesioner. Pengguna aplikasi, termasuk mahasiswa dan masyarakat Indonesia berusia 17-40 tahun, mencoba aplikasi dan memberikan evaluasi. Uji black box mengamati hasil eksekusi melalui data uji untuk mengecek fungsionalitas perangkat lunak berdasarkan spesifikasi kebutuhan. Selain itu, uji hardware dilakukan untuk menguji aplikasi pada berbagai perangkat smartphone guna memastikan aplikasi berjalan dengan baik.

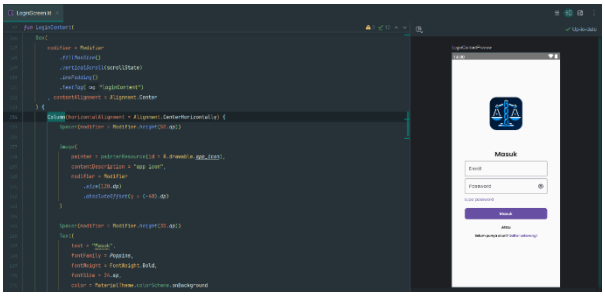
2.5 Maintenance

Pemeliharaan merupakan kegiatan yang melibatkan perawatan, modifikasi, atau pengembangan lebih lanjut dari aplikasi yang telah dibuat jika diperlukan. Tahap pemeliharaan ini adalah tahap akhir dalam siklus pengembangan aplikasi. Pada tahap ini, aplikasi yang telah melalui proses pengujian akan diperbaiki dan disempurnakan berdasarkan evaluasi dan temuan dari tahap sebelumnya. Pemeliharaan mencakup berbagai aktivitas seperti memperbaiki *bug* yang ditemukan setelah peluncuran serta meningkatkan fitur yang ada.

3. Hasil dan Pembahasan

3.1 Implementasi Android Jetpack Compose Pada Halaman Login.

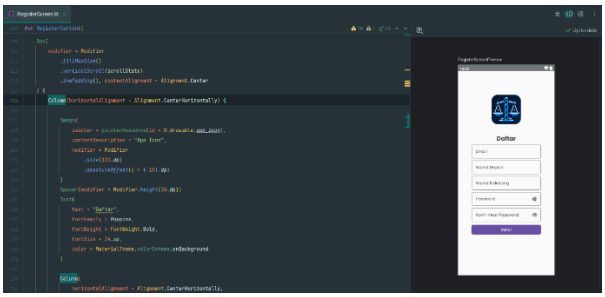
Gambar 4 menggambarkan pengembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman Login. Dimulai dari membuat wadah untuk menampung seluruh komponen dengan menggunakan `Column()` agar komponen di dalamnya tersusun secara vertical. Kemudian komponen pertama yang dibuat yaitu logo aplikasi dengan menggunakan `Image()`, kemudian dilanjutkan dengan dua inputan text field dengan menggunakan sintaks `OutlinedTextField()` untuk memasukkan email dan password, kemudian tombol button dengan text 'masuk' dengan sintaks `Button()`, dan di bawah button terdapat text 'Daftar Sekarang' yang diberi *modifier clickable* yang di isi dengan fungsi `onRegisterClick()`.



Gambar 4. Android Jetpack Compose Pada Halaman Login

3.2 Implementasi Android Jetpack Compose Pada Halaman Register.

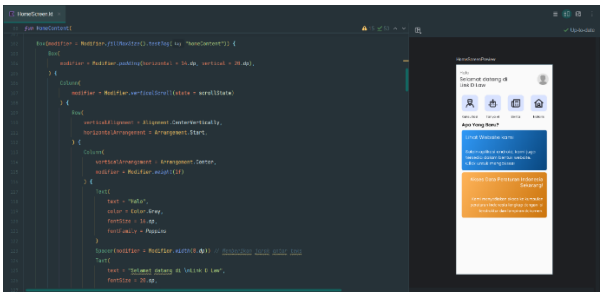
Pada Gambar 5 ketika pengguna diminta untuk melakukan *login* dan tidak memiliki akun. Maka pengguna dapat melakukan pendaftara pada halaman *register*. Pada Gambar ini menggambarkan penggembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman *Register*. Dimulai dari mebuat wadah untuk menampung seluruh komponen menggunakan sintaks `Column()` agar seluruh komponen di dalamnya tersusun secara vertical, kemudian di lanjutkan dengan membuat Logo aplikasi menggunakan sintaks `Image()`, dan dilanjutkan dengan lima text field dengan menggunakan sintaks `OutlinedTextField()`. Pada text field pertama digunakan untuk input email, text field kedua digunakan untuk input nama depan, text field ketiga digunakan untuk input nama belakang, dan text field keempat dan kelima digunakan untuk input password dan konfirmasi password. Dan satu button dengan sintaks `Button()` untuk mendaftarkan data pengguna yang sudah di lengkapi sebelumnya.



Gambar 5. Android Jetpack Compose Pada Halaman Register

3.3 Implementasi Android Jetpack Compose Pada Halaman Home.

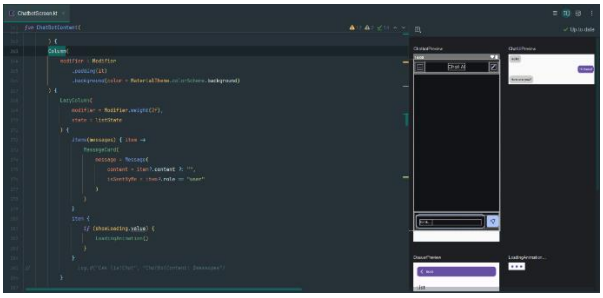
Pada Gambar 6 menggambarkan pengembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman *Home*. Pada halaman home ini, layouting dibagi menjadi tiga bagian. Pada badian pertama terdapat terdapat `Row()` untuk menjadikan komponen di dalamnya tersusun secara horizontal. Dan kemudian di dalamnya terdapat `Column()` untuk menjadikan dua component di dalamnya tersusun secara vertical, dan dilanjutkan dengan sintaks `Image()` untuk profil foto sehingga berada di pojok kanan karena parent wadah sebelumnya yaitu `Row()`. Kemudian pada bagian layout kedua terdapat `LazyRow()` untuk menampilkan list secara horizontal yang di dalamnya terdapat empat komponen *icon* yaitu untuk konsultasi, tanya ai, berita, dan notaris. Dan icon tiap list tersebut terdapat *modifier clickable()* yang di isi dengan fungsi `onClick()`. Kemudian pada layout bagian ketiga terdapat dua kartu dengan menggunakan sintaks `Card()`. Pada card pertama pengguna akan di arahkan ke website aplikasi hukum dan pada card kedua pengguna akan di arahkan ke halaman list peraturang undang – undang.



Gambar 6. Android Jetpack Compose Pada Halaman Home

3.4 Implementasi Android Jetpack Compose Pada Halaman Chatbot.

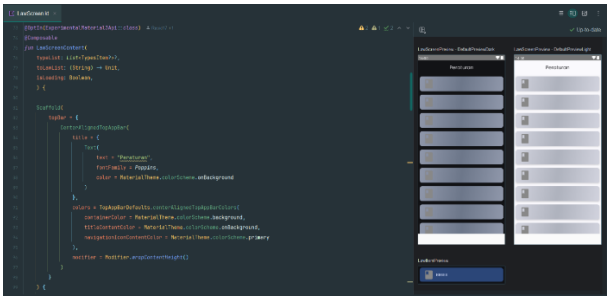
Pada Gambar 7 menggambarkan pengembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman *Chatbot*. Tahap awal membuat desain bubble chat dengan menggunakan sintaks `Card()` untuk tempat text prompt yang di kirimkan. Kemudian membuat text field untuk memasukkan prompt yang ingin di input dengan menggunakan sintaks `OutlinedTextField()` dan button kirim dengan icon pesawat dengan sintaks `IconButton()`.



Gambar 7. Android Jetpack Compose Pada Halaman Chatbot

3.5 Implementasi Android Jetpack Compose Pada Halaman Peraturan.

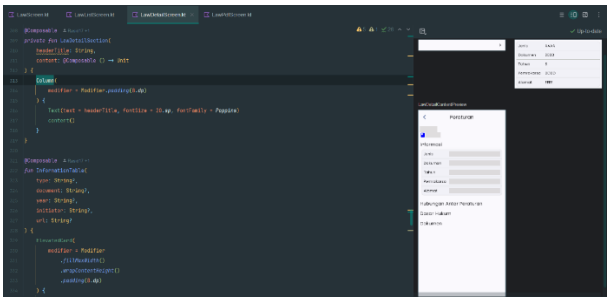
Pada Gambar 8 menggambarkan pengembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman Peraturan. Dimulai dengan mebuat desain layout item yang ditampilkan dengan menggunakan sintaks `ElevatedCard()`, kemudian di panggil di dalam list dengan menggunakan `LazyColumn()` agar list yang ditampilkan tersusun secara vertical. Kemudian pada setiap item di isi dengan fungsi `onClick()` agar Ketika salah satu item di klik maka pengguna akan di arahkan ke halaman detail peraturan.



Gambar 8. Android Jetpack Compose Pada Halaman Peraturan

3.6 Implementasi Android Jetpack Compose Pada Halaman Detail Peraturan.

Gambar 9 menggambarkan pengembangan antarmuka pengguna dengan menggunakan *android jetpack compose* pada halaman Detail Peraturan. Dimulai dari tahap pembuatan table untuk spesifikasi peraturan undang – undang yang di pilih oleh pengguna. Setelah layout dibuat table bisa di panggil di dalam halaman konten detail peraturan. Halaman detail peraturan ini terdapat file pdf yang nantinya dapat ditampilkan untuk pengguna. `VerticalPDFReader()` digunakan untuk menampilkan file pdf pada android studio.



Gambar 9. Android Jetpack Compose Pada Halaman Detail Peraturan

3.7 Implementasi MVVM untuk Menangani Logika Halaman Login.

Pada kelas `LoginViewModel` yang tertera pada Gambar 10, kelas ini mengelola logika login pengguna dengan menggunakan `authRepository`. Kelas ini memiliki dua LiveData untuk mengelola status loading dan status login. Fungsi `login` menggunakan coroutine `viewModelScope` untuk menjalankan operasi login secara asinkron. Saat login dimulai, loading diatur ke true. Jika login berhasil, status login dan informasi sesi pengguna (seperti token akses, token refresh, nama pengguna, dan email) disimpan menggunakan `authRepository`, dan loading diatur ke false. Jika terjadi kesalahan HTTP atau kesalahan lainnya selama login, status login diperbarui dengan pesan kesalahan dan loading diatur ke false. Fungsi `resetLoginState` mengatur ulang status login.

Gambar 10. Implementasi MVVM Pada Halaman Login

3.8 Implementasi MVVM untuk Menangani Logika Halaman Register.

Pada kelas `RegisterViewModel` yang tertera pada Gambar 11, kelas ini menangani logika pendaftaran pengguna dan pengambilan data provinsi serta kabupaten dengan menggunakan `authRepository`. Kelas ini memiliki beberapa LiveData untuk mengelola status loading, status pendaftaran, daftar provinsi, daftar kabupaten, dan status terkait provinsi serta kabupaten. Fungsi `getProvinces` dan `getRegency` menggunakan coroutine `viewModelScope` untuk mengambil data provinsi dan kabupaten secara asinkron dari `authRepository`. Fungsi register juga menggunakan coroutine untuk melakukan operasi pendaftaran pengguna secara asinkron. Selama pendaftaran, status loading diatur ke true, dan jika berhasil, informasi sesi pengguna disimpan, dan status loading diatur ke false. Jika terjadi kesalahan HTTP atau kesalahan lainnya, status pendaftaran diperbarui dengan pesan kesalahan dan status loading diatur ke false. Fungsi `resetRegisterState` mengatur ulang status pendaftaran.

```
class RegisterViewModel(private val authRepository: AuthRepository): ViewModel() {

    private val _loading = MutableLiveData<Boolean>()
    val loading: LiveData<Boolean> = _loading

    private val _registerStatus = MutableLiveData<BasicResponse?>()
    val registerStatus: LiveData<BasicResponse?> = _registerStatus

    private val _provinceList = MutableLiveData<ProvincesData?>()
    val provinceList: LiveData<ProvincesData?> = _provinceList

    private val _regencyList = MutableLiveData<RegenciesData?>()
    val regencyList: LiveData<RegenciesData?> = _regencyList

    private val _provinceState = MutableLiveData<Boolean>()
    val provinceState: LiveData<Boolean> = _provinceState

    private val _regencyState = MutableLiveData<Boolean>()
    val regencyState: LiveData<Boolean> = _regencyState

    fun getProvinces(){
        viewModelScope.launch {
            try {
                val response = authRepository.getProvinces()
                _provinceList.value = response.data
            }catch (e: Exception){
                _provinceState.value = false
            }
        }
    }

    fun getRegency(code: Int){
        viewModelScope.launch {
            try {
                val response = authRepository.getRegencies(code = code)
                _regencyList.value = response.data
            }catch (e: Exception){
                _provinceState.value = false
            }
        }
    }

    fun register(
        registerRequest: RegisterRequest
    ){
        viewModelScope.launch {
            _loading.value = true
            try {
                val response = authRepository.register(registerRequest)
                _registerStatus.value = BasicResponse(
                    ok = response.ok,
                    message = response.message
                )

                val accessToken = response.data?.auth?.accessToken ?: ""
                val isLogin = true
                val refreshToken = response.data?.auth?.refreshToken ?: ""
                val userName = response.data?.user?.name ?: ""
                val userEmail = response.data?.user?.email ?: ""

                authRepository.saveUserSession(
                    UserSession(
                        accessToken = accessToken,
                        refreshToken = refreshToken,
                        isLogin = isLogin,
                        name = userName,
                        email = userEmail
                    )
                )

                _loading.value = false
            }
            catch (e: HttpException){
                _loading.value = false
                _registerStatus.value = BasicResponse(
                    ok = false,
                    message = e.message()
                )
            }
            catch (e: Exception){
                _loading.value = false
                _registerStatus.value = BasicResponse(
                    ok = false,
                    message = e.message
                )
            }
        }
    }

    fun resetRegisterState() {
        _registerStatus.value = null
    }
}
```

Gambar 11. Implementasi MVVM Pada Halaman Register

3.9 Implementasi MVVM untuk Menangani Logika Halaman Chatbot.

Pada kelas `ChatViewModel` yang tertera pada Gambar 12, kelas ini mengelola logika pengambilan data chatbox, chat dalam chatbox, dan pengiriman pesan menggunakan `chatRepository`. Kelas ini memiliki berbagai LiveData dan StateFlow untuk memantau status chatbox, status chat, status loading, status pengiriman pesan, dan UUID saat ini. Fungsi `getChatboxs` mengambil daftar chatbox secara asinkron, mengatur status pengambilan chatbox, dan memperbarui LiveData yang sesuai. Fungsi `getChatInChatbox` mengambil daftar chat dalam chatbox tertentu berdasarkan UUID yang diberikan. Fungsi `sendChat` mengirim pesan ke chatbox tertentu, memperbarui status pengiriman, dan memperbarui daftar chatbox serta chat dalam chatbox setelah pengiriman berhasil. Setiap operasi dilakukan dalam coroutine `viewModelScope` untuk mengelola pekerjaan asinkron dengan baik dan memastikan pembaruan UI yang tepat.

```
class ChatViewModel(private val chatRepository: ChatRepository): ViewModel() {

    private val _chatboxs = MutableLiveData<List<ChatboxsItem?>?>()
    val chatboxs: LiveData<List<ChatboxsItem?>?> = _chatboxs

    private val _loadingState = MutableStateFlow(false)
    val loadingState: StateFlow<Boolean> = _loadingState

    private val _chatInChatbox = MutableLiveData<List<ChatsItem?>?>()
    val chatInChatbox: LiveData<List<ChatsItem?>?> = _chatInChatbox

    private val _sendChat = MutableLiveData<SendChatResponse?>()
    val sendChat: LiveData<SendChatResponse?> = _sendChat

    private val _loadingChat = MutableStateFlow(false)
    val loadingChat: StateFlow<Boolean> = _loadingChat

    private val _sendStatus = MutableStateFlow(false)
    val sendStatus: StateFlow<Boolean> = _sendStatus

    private val _currentUUID = MutableLiveData("")
    val currentUUID: LiveData<String> = _currentUUID


    //new state
    private val _getChatboxStatus = MutableLiveData<Boolean>()
    val getChatboxStatus: LiveData<Boolean> = _getChatboxStatus

    private val _getChatStatus = MutableLiveData<Boolean>()
    val getChatStatus: LiveData<Boolean> = _getChatStatus

    fun getChatboxs(){
        viewModelScope.launch {
            _getChatboxStatus.value = false
            try {
                val response = chatRepository.getChatbox()
                _chatboxs.value = response.data?.chatboxs
                _getChatboxStatus.value = true
            }catch (e: Exception){
                _getChatboxStatus.value = false
            }
        }
    }

    fun getChatInChatbox(uuid: String){
        viewModelScope.launch {
            _getChatStatus.value = false
            try {
                val response = chatRepository.getChatInChatbox(uuid = uuid)
                _chatInChatbox.value = response.data?.chats
                _getChatStatus.value = true
            }catch (e:Exception){
                _getChatStatus.value = false
            }

        }
    }

    fun sendChat(idChatbox: String, message: String){

        viewModelScope.launch {
            _sendStatus.value = true
            try {
                val response = chatRepository.sendChat(idChatbox = idChatbox, message = message)
                _sendChat.value = response
                val newUUID = response.data?.chatbox?.uuid.toString()
                _currentUUID.value = newUUID

                getChatboxs()
                getChatInChatbox(newUUID)

                _sendStatus.value = false
            }catch (e: Exception){
                _sendStatus.value = false
            }
        }
    }
}
```

Gambar 12. Implementasi MVVM Pada Halaman Chatbot

3.10 Implementasi MVVM untuk Menangani Logika Halaman Peraturan.

Pada kelas *LawViewModel* yang tertera pada Gambar 13, kelas ini menangani pengambilan data jenis hukum menggunakan *lawRepository*. Kelas ini memiliki dua LiveData: *lawTypeList* untuk menyimpan daftar jenis hukum dan *loadingState* untuk melacak status loading. Fungsi *getLawTypes* menggunakan coroutine *viewModelScope* untuk menjalankan operasi pengambilan data secara asinkron. Ketika data sedang diambil, *loadingState* diatur ke true. Jika pengambilan data berhasil, *lawTypeList* diperbarui dengan data yang diterima, dan *loadingState* diatur ke false. Jika terjadi kesalahan selama proses pengambilan data, *loadingState* juga diatur ke false untuk mengakhiri status loading.

```
class LawViewModel(private val lawRepository: LawRepository): ViewModel()
{
    private val _lawTypeList = MutableLiveData<LawTypeData?>()
    val lawTypeList: LiveData<LawTypeData?> = _lawTypeList

    private val _loadingState = MutableLiveData<Boolean>()
    val loadingState: LiveData<Boolean> = _loadingState

    fun getLawTypes(){
        viewModelScope.launch {
            _loadingState.value = true
            try {
                val response = lawRepository.getLawTypes()
                _lawTypeList.value = response.data
                _loadingState.value = false
            }catch (e: Exception){
                _loadingState.value = false
            }
        }
    }

}
```

Gambar 13. Implementasi MVVM Pada Halaman Peraturan

3.11 Implementasi MVVM untuk Menangani Logika Halaman Daftar Peraturan.

Pada kelas *LawListViewModel* yang tertera pada Gambar 14, kelas ini mengelola paginasi dan pembersihan data item hukum menggunakan *LawItemDatabase* dan *ApiService*. Kelas ini memiliki fungsi *getItemPagingFlow* yang mengembalikan aliran data paging (PagingData) untuk item hukum berdasarkan kode yang diberikan. Fungsi ini menggunakan Pager dengan konfigurasi paginasi seperti ukuran halaman dan jarak prefetch, serta *LawItemRemoteMediator* untuk sinkronisasi data lokal dan remote. Aliran ini di-cache dalam *viewModelScope* untuk manajemen lifecycle yang lebih efisien. Selain itu, kelas ini menyediakan fungsi *clearLocalData* yang membersihkan semua data lokal dalam database hukum secara asinkron menggunakan coroutine *viewModelScope*.

```
class LawListViewModel(
    private val lawItemDb: LawItemDatabase,
    private val apiService: ApiService
) : ViewModel() {


    @OptIn(ExperimentalPagingApi::class)
    fun getItemPagingFlow(code: String): Flow<PagingData<LawsItemEntity>> {
        return Pager(
            config = PagingConfig(
                pageSize = 20,
                prefetchDistance = 10
            ),
            remoteMediator = LawItemRemoteMediator(lawItemDb, apiService, code),
            pagingSourceFactory = {
                lawItemDb.dao.loadAllLawItemPaged()
            }
        ).flow.cachedIn(viewModelScope)
    }

    fun clearLocalData(){
        viewModelScope.launch {
            lawItemDb.dao.clearAll()
        }
    }

}
```

Gambar 14. Implementasi MVVM Pada Halaman Daftar Peraturan

3.12 Implementasi MVVM untuk Menangani Logika Halaman Detail Peraturan.

Pada kelas *LawDetailViewModel* yang tertera pada Gambar 15, kelas ini mengelola pengambilan detail data hukum menggunakan *lawRepository*. Kelas ini memiliki dua LiveData: *lawDetailData* untuk menyimpan detail data hukum dan *loadingState* untuk melacak status loading. Fungsi *getLawDetail* menggunakan coroutine *viewModelScope* untuk menjalankan operasi pengambilan data secara asinkron berdasarkan ID yang diberikan. Saat proses pengambilan data dimulai, *loadingState* diatur ke true. Jika pengambilan data berhasil, *lawDetailData* diperbarui dengan data yang diterima, dan *loadingState* diatur ke false. Jika terjadi kesalahan selama proses pengambilan data, *loadingState* juga diatur ke false untuk mengakhiri status loading.

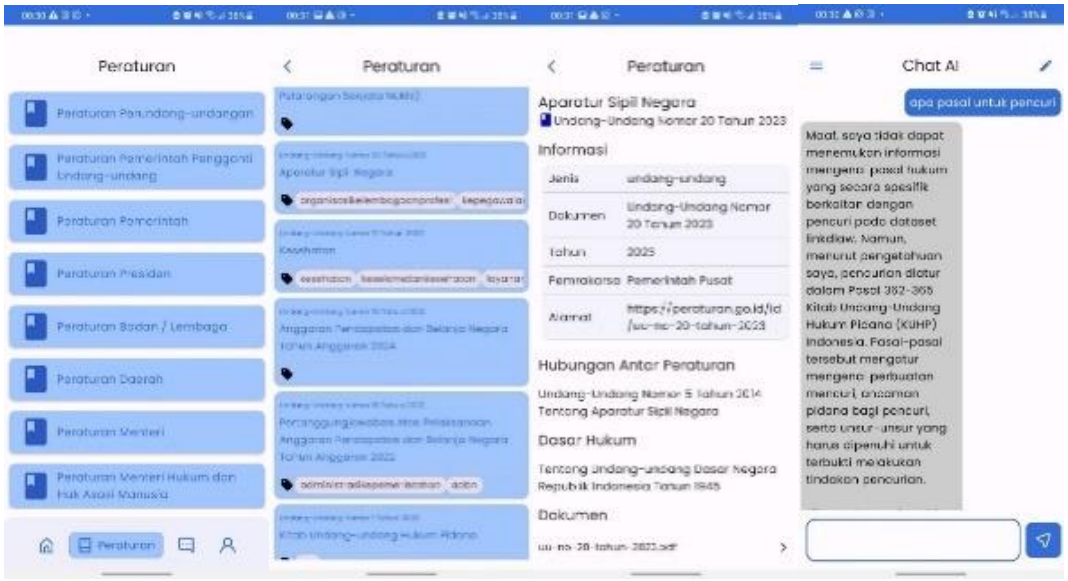
```
class LawDetailViewModel(private val lawRepository: LawRepository): ViewModel() {

    private val _lawDetailData = MutableLiveData<DetailLawData?>()
    val lawDetailData: LiveData<DetailLawData?> = _lawDetailData

    private val _loadingState = MutableLiveData<Boolean>()
    val loadingState: LiveData<Boolean> = _loadingState

    fun getLawDetail(id: Int){
        viewModelScope.launch {
            _loadingState.value = true
            try {
                val response = lawRepository.getDetailLaw(id = id)
                _lawDetailData.value = response.data
                _loadingState.value = false
            }catch (e: Exception){
                _loadingState.value = false
            }
        }
    }

}
```

Gambar 15. Implementasi MVVM Pada Halaman Detail Peraturan

3.13 Hasil Perancangan Aplikasi

Pada tahap ini, aplikasi menampilkan fitur – fitur yang telah dirancang menggunakan jetpack compose sebagai pendekatan antarmuka yang diimplementasikan pada setiap halaman dan mengimplementasikan model –

view – viewmodel untuk memisahkan antara logika UI dan logika bisnis dengan mengelola setiap pengambilan data.



(a) (b) (c) (d)

Gambar 16. Hasil Perancangan Halaman Peraturan (a) Jenis Peraturan (b) List Peraturan dari Jenis yang dipilih (c) Detail Peraturan

Hasil dari implementasi android jetpack compose yang digunakan untuk membuat UI yang deklaratif dan responsive untuk mendesain halaman aplikasi pada Gambar 16 (a), (b), (c) dan (d). selain itu implementasi pola arsitektur model – view – viewmodel yang dimulai dari layer model yang merepresentasikan data yang akan digunakan pada logika bisnis. Kemudian layer view yang berisi UI dari aplikasi untuk mengatur bagaimana informasi akan ditampilkan yang berisi kelas – kelas. Dan layer terakhir viewmodel yang bertugas untuk berinteraksi dengan model di mana data yang ada akan diteruskan ke layer view sehingga data akan tampil pada halaman aplikasi pada Gambar 16 (a), (b), (c) dan (d).

3.14 Pengujian Black Box

Pengujian aplikasi ini mencakup beberapa tahap untuk memaksimalkan performa dan mengurangi kesalahan serta bug. Tahap pertama adalah Black Box Testing, yang berfokus pada pengujian fungsionalitas tanpa memerlukan pemahaman struktur internal atau kode sumber. Penguji memeriksa input dan output untuk memastikan fitur bekerja sesuai spesifikasi. Hasil pengujian ini penting untuk memastikan kualitas aplikasi sebelum dirilis, mencakup berbagai skenario dan kasus uji untuk menutupi kemungkinan penggunaan oleh pengguna akhir. Hasil pengujian Black Box Testing, termasuk detail setiap kasus uji, hasil yang diharapkan, hasil aktual, dan status keberhasilan, dapat dilihat pada Tabel 1

Berdasarkan hasil yang terlihat pada Tabel 1, dapat disimpulkan bahwa semua fungsi yang terdapat pada aplikasi mulai dari Halaman Login sampai Halaman Chatbot dapat berjalan dengan lancar tanpa ada kesalahan. Hasil ini menunjukkan bahwa pengujian Black Box Testing berhasil mendeteksi dan mengatasi potensi masalah dalam berbagai komponen dan fitur aplikasi. Jadi secara keseluruhan, hasil dari pengujian Black Box Testing menunjukkan bahwa aplikasi ini telah dikembangkan dengan baik dan memenuhi standar kualitas yang tinggi. Tidak ada kesalahan atau bug yang ditemukan dalam pengujian ini, menunjukkan bahwa aplikasi siap untuk dirilis dan digunakan oleh publik.

Table 1. Pengujian Black Box

No	Halaman	Proses	Hasil
1	Login	Menampilkan Halaman Login	Berhasil
2	Register	Menampilkan Halaman Register	Berhasil
3	Home	Menampilkan Halaman Home	Berhasil
4	Kategori Peraturan	Menampilkan Halaman Kategori Peraturan	Berhasil
5	Daftar Undang - undang	Menampilkan Halaman Daftar Undang - undang	Berhasil
6	Detail Undang - undang	Menampilkan Halaman Detail Undang - undang	Berhasil
7	Chatbot	Menampilkan Halaman Chatbot	Berhasil

3.15 Pengujian Hardware

Pada tahap ini, peneliti menguji kinerja aplikasi yang dapat dilihat pada Table 2 dengan melakukan pengujian pada beberapa perangkat smartphone berbeda untuk memastikan aplikasi dapat berjalan dengan baik dan optimal pada perangkat dengan ekosistem Android.

Pengujian ini sangat penting karena perangkat Android memiliki berbagai variasi dalam hal perangkat keras, resolusi layar, versi sistem operasi, dan konfigurasi lainnya yang dapat mempengaruhi kinerja aplikasi.

Table 2. Pengujian Hardware

No	Merk	Versi Android	Ukuran Layar	Ram	Hasil
1	Infinix Note 10 Pro	11	6,95 inch	8 GB	Lancar
2	Samsung A50S	11	6,43 inch	4 GB	Lancar
3	Realme 8 Pro	13	6,4 inch	8 GB	Lancar
4	Samsung A50	10	6,43 inci	4GB	Lancar
5	Poco X5 5G	12	6,67 inci	8GB	Lancar

Berdasarkan hasil yang disajikan pada Tabel 2, didapat kesimpulan bahwa aplikasi ini dapat berjalan dengan baik pada beberapa device mulai dari android versi 10 sampai dengan versi 13 dengan ukuran layar terkecil yakni 6,4 inci sampai dengan ukuran layar terbesar dengan ukuran 6,95 inci. Hal ini menunjukkan bahwa aplikasi ini memiliki kompatibilitas yang baik dengan berbagai versi Android, sehingga dapat diakses oleh pengguna dengan perangkat yang lebih baru maupun yang lebih lama. Selain itu, rentang ukuran layar yang didukung oleh aplikasi ini cukup luas, mencakup berbagai jenis ponsel pintar yang umum digunakan saat ini.

3.16 Pengujian Pengguna

Pengujian ini melibatkan pengguna untuk mengidentifikasi dan memperbaiki masalah usability, serta mendapatkan umpan balik langsung mengenai pengalaman pengguna. Proses ini adalah langkah penting dalam pengembangan aplikasi, karena melibatkan pengguna akhir yang sebenarnya dalam evaluasi dan pengujian produk. Dengan melibatkan pengguna, pengembang dapat memperoleh wawasan yang lebih mendalam tentang bagaimana aplikasi digunakan dalam situasi nyata dan dapat mengidentifikasi masalah yang mungkin tidak terlihat selama tahap pengembangan internal.

Tabel 3. Pengujian Pengguna

No	Pertanyaan	Nilai Bagus Sekali	Bagus	Buruk	Sangat Buruk
1	Apakah mudah dalam melakukan Login?	11	6	2	0
2	Apakah mudah dalam melakukan Register?	11	9	0	0
3	Seberapa menarik UI pada halaman Home?	14	6	1	0
4	Seberapa mudah interaksi pada halaman Home?	13	7	2	0
5	Apakah bermanfaat fitur Chatbot pada aplikasi ini?	10	8	2	0
6	Kesesuaian Peraturan Undang undang dengan peraturan.go.id	11	7	1	0
7	Seberapa mudah dalam menggunakan halaman peraturan?	12	6	1	0
8	Seberapa bermanfaatkah aplikasi ini dalam membantu pengguna dalam bantuan hukum?	14	5	1	0
Total		86	54	10	0
Rata - rata		57,33%	36%	6,67%	0%

Hasil pengujian pengguna yang disajikan dalam Tabel 3 mengungkapkan bahwa mayoritas pengguna memberikan penilaian yang sangat positif terhadap aplikasi ini. Sebanyak 57.33% responden menilai aplikasi ini dengan rating "Bagus Sekali". Sebagian besar lainnya, yaitu 36%, memberikan rating "Bagus". Hanya sebagian kecil pengguna, yaitu sekitar 6.67%, yang memberikan penilaian kurang baik, dan tidak ada pengguna yang memberikan penilaian sangat buruk (0%).

4. Kesimpulan

Berdasarkan penelitian yang telah dilakukan yaitu “Implementasi MVVM Dan Framework Jetpack Compose Pada Aplikasi Hukum Berbasis Android” didapatkan kesimpulan sebagai berikut : Penelitian menghasilkan aplikasi bantuan hukum berbasis Android untuk mempermudah Masyarakat Indonesia dalam mendapatkan bantuan hukum. Alpha testing melibatkan 2 penguji ahli yang menyimpulkan bahwa pengembangan aplikasi menggunakan Kotlin, MVVM, dan Jetpack Compose, serta fiturnya berjalan dengan baik untuk dilakukan beta testing. Beta testing melibatkan 20 orang. Berdasarkan pendapat responden yang terlibat menghasilkan persentase sebesar 57,33% menilai ‘Bagus Sekali’, 36%, menilai "Bagus". Hanya sebagian kecil pengguna, yaitu sekitar 6.67%, yang memberikan penilaian ‘Kurang Baik’, dan tidak ada pengguna yang memberikan penilaian sangat ‘Buruk’ (0%) terhadap keseluruhan fitur aplikasi hukum ini. Berdasarkan kesimpulan di atas, peneliti mengajukan beberapa saran sebagai referensi untuk pengembangan penelitian dan aplikasi ini yang akan dilakukan selanjutnya : pada pengembangan selanjutnya, peneliti menyarankan untuk mengembangkan aplikasi ini menjadi multi platform. Pada pengembangan selanjutnya, peneliti menyarankan untuk menambahkan fitur fitur menarik ke aplikasi ini seperti fitur untuk kosultasi pengacara agar dapat menarik banyak pengguna.

Daftar Rujukan

[1] F. A. Candra and F. J. Sinaga, "Peran Penegak Hukum dalam Penegakan Hukum di Indonesia," *Edu Soc. J. Pendidikan, Ilmu Sos. Dan Pengabdi. Kpd. Masy.*, vol. 1, no. 1, pp. 41–50, 2023, doi: 10.56832/edu.v1i1.15.

[2] H. Basri, "Perlindungan Hukum Terhadap Pelaku Tindak Pidana Berdasarkan Sistem Peradilan Pidana Indonesia," *SIGn J. Huk.*, vol. 2, no. 2, pp. 104–121, 2021, doi: 10.37276/sjh.v2i2.90.

[3] G. A. Lambonan, R. Sengkey, X. B. N. Najoan, T. Elektro, U. Sam, and R. Manado, "Rancang Bangun Aplikasi Ensiklopedia Hukum Indonesia Berbasis Android," *J. Tek. Inform. Vol.14*, vol. 14, no. 3, pp. 341–348, 2019.

[4] F. Fadhlillah, A. Kuswandi, and P. Haryono, "Peranan Aplikasi Android Dalam Peningkatan Kualitas Pelayanan ekolah di Pesantren Persis Kota Tasikmalaya," *Kelola J. Manaj. Pendidik.*, vol. 8, no. 1, pp. 22–33, 2021, doi: 10.24246/j.jk.2021.v8.i1.p22-33.

[5] M. S. Arif, A. Musthafa, and D. Muriyatmoko, "Implementation of Model-View-ViewModel (MVVM) Architecture Pattern in the Sistem Informasi Akademik UNIDA Gontor Mobile Application," in *Proceeding International Conference on Science and Engineering*, 2017, pp. 283–289.

[6] Baselam Asefa, *Membangun Library Komponen Android Menggunakan Jetpack Compose*. 2022.

[7] I. Sommerville, *Software Engineering*, 10th ed. England: PEARSON, 2016.

[8] M. Nazir and D. Siahaan, "Structural and Semantic Similarity Measurement of UML Use Case Diagram," *Lontar Komput.*, vol. 11, no. 2, pp. 88–100, 2020.

[9] Z. Tuasamu, A. I. M.Lewaru, and M. Rivaldi Idris, "Analisis Sistem Informasi Akuntansi Siklus Pendapatan Menggunakan DFD Dan Flowchart Pada Bisnis Porobico," *JURBISMAN*, vol. 1, no. 2, pp. 495–510, 2023.

[10] R. Wijaya, N. Ibrahim, J. T. Informatika, and U. K. Maranatha, "Penggunaan Android Kotlin Untuk Pembuatan Aplikasi Lelang," *J. Strateg.*, vol. 1, no. November, pp. 324–335, 2019.