Penerapan Arsitektur MVVM pada Pengembangan Aplikasi Android AI.YAM untuk Deteksi Ayam Mati

Oktariyan Nur Saputra[1], I Made Suartana[2]

[1,2]Program Studi Teknik Informatika/Teknik Informatika, Universitas Negeri Surabaya
[1]oktariyan.22066@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

Abstrak*—Penelitian ini dilatarbelakangi oleh tantangan peternak dalam mencegah penyebaran penyakit akibat keterlambatan identifikasi kematian ayam. Kehadiran teknologi kecerdasan buatan Artificial Intelligence(AI) pada aplikasi Android berpotensi meningkatkan produktivitas peternak melalui pemantauan data secara langsung (real-time). Penelitian ini bertujuan mengembangkan aplikasi Android bernama AI.YAM yang terintegrasi dengan model AI yang terhubung layanan cloud untuk mendeteksi ayam mati, serta menguji fungsionalitas dan performanya. Aplikasi ini dikembangkan menggunakan arsitektur Model-View-ViewModel(MVVM) untuk memisahkan logika bisnis dan antarmuka pengguna demi kemudahan perawatan. Pengujian fungsionalitas dilakukan menggunakan pendekatan black-box testing. Sementara itu, pengujian performa memanfaatkan Android Studio Profiler untuk mengukur penggunaan CPU dan memori, serta Logcat untuk mengukur waktu respons klasifikasi gambar. Hasil penelitian menunjukkan penerapan arsitektur MVVM berhasil menghubungkan AI.YAM dengan layanan cloud untuk sinkronisasi penyimpanan data sesuai dengan rencana pengujian fungsionalitas. Pada pengujian performa di Android versi 16, Logcat mencatat total waktu respons 2260 ms (4 ms unggah data dan 2256 ms inferensi). Pengujian Profiler menunjukkan penggunaan CPU sebesar 1% dan konsumsi memori 285,6 MB. Kesimpulannya, arsitektur MVVM berhasil membuat aplikasi AI.YAM berjalan dengan stabil dan ringan.

***Kata Kunci*—AI.YAM, Model-View-ViewModel(MVVM), Android, Artificial Intelligence(AI), Google Cloud Platform, Logcat, Profiler.**

I. PENDAHULUAN

Ayam merupakan salah satu sumber makanan terbaik, untuk memenuhi kebutuhan protein hewani di Indonesia. Data dari Badan Pusat Statistik menunjukan bahwa tingkat konsumsi daging ayam meningkat setiap tahun, menunjukan permintaan yang tinggi untuk produk tersebut [1]. Namun, produksi yang lebih tinggi juga terdapat berbagai masalah, terutama dalam menjaga kesehatan dan kelangsungan hidup ayam. Keterlambatan dalam mengidentifikasi ayam mati merupakan masalah utama, yang dapat mengakibatkan penyebaran penyakit dan kerugian ekonomi bagi peternak [2].

Untuk mengatasi masalah tersebut, penerapan teknologi kecerdasan buatan Artificial Intelligence(AI) dapat meningkatkan produktivitas dalam mengelola peternakan unggas, terutama dalam memantau peternakan ayam secara otomatis [3]. Integrasi AI dengan sistem Internet of Things(IoT) berbasis android membuat peternak dapat memantau data ayam secara langsung melalui perangkat seluler [4]. Dengan adanya aplikasi yang ringan dan mudah digunakan, peternak dapat melakukan pengawasan kapan saja dan dimana saja [5]. Agar aplikasi tidak terlalu membebani kinerja perangkat android, penerapan arsitektur Model View ViewModel(MVVM) membuat kinerja aplikasi lebih stabil karena memisahkan logika bisnis dan user interface [6].

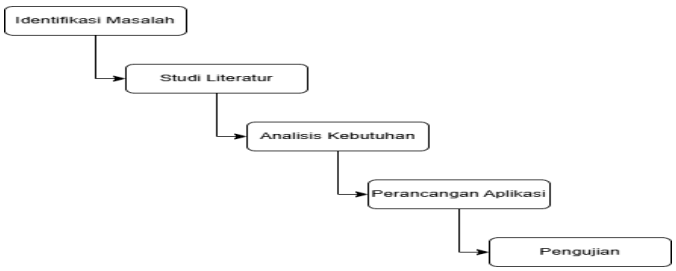
Dalam pengembangannya, Aplikasi AI.YAM menggunakan arsitektur MVVM. Arsitektur ini dipilih karena memberikan tanggung jawab yang jelas antara logika bisnis dan User Interface. Selain itu menerapkan arsitektur ini dapat memudahkan dalam pengembangan dan perawatan aplikasi. Penerapan arsitektur MVVM juga dapat mempermudah melakukan pemeliharaan code [7]. Penelitian yang dilakukan oleh Arfianto (2024) di Universitas Negeri Surabaya yang melakukan perbandingan performa pada arsitektur MVVM dan Model View Presenter(MVP). Hasilnya menunjukan bahwa arsitektur MVVM menggunakan CPU dan RAM lebih baik dari pada arsitektur MVP [8]. Selain itu penelitian Zakaria(2023) menunjukan bahwa arsitektur MVVM lebih unggul dalam mengonsumsi memori dan CPU yang lebih rendah sehingga perangkat lunak lebih ringan saat digunakan dibandingkan perangkat lunak yang dikembangkan menggunakan arsitektur MVP [9]. Selain itu, Achdan Epiloksa(2022) menemukan bahwa menerapkan arsitektur MVVM dapat meningkatkan kecepatan respon dan mengurangi konsumsi memori hingga 20% dibandingkan dengan arsitektur Model View Controller (MVC) [10]. Keunggulan ini sangat relevan untuk aplikasi AI.YAM, yang membutuhkan sinkronisasi secara langsung antara hasil klasifikasi AI dengan User Interface pada aplikasi. Dengan Data Binding, setiap hasil deteksi yang diterima dari API dapat langsung diperbarui pada User Interface aplikasi tanpa perlu melakukannya secara manual.

Berdasarkan tinjauan literatur, sebagian besar penelitian sebelumnya berfokus pada penerapan IoT dan sensor fisik dalam peternakan, sementara penelitian mengenai rekayasa perangkat lunak berbasis AI yang terintegrasi dengan cloud masih terbatas. Penelitian ini berfokus pada penerapan arsitektur MVVM dalam aplikasi android berbasis AI yang terhubung dengan layanan cloud. Tujuan dari penelitian ini adalah menerapkan arsitektur MVVM, memastikan fungsionalitas pada aplikasi dan menguji performa pada aplikasi. Pengujian performa adalah bagian penting dari penelitian ini. Perbedaan versi pada sistem operasi android dapat mempengaruhi kecepatan respon dan stabilitas aplikasi [11]. Karena itu, penelitian ini mengujikan aplikasi AI.YAM pada versi android 16, untuk menilai kestabilan, waktu respon dan kinerja pada perangkat. Penelitian ini berfokus pada rekayasa perangkat lunak, penulis ingin menerapkan arsitektur MVVM pada aplikasi android berbasis AI.

Tujuan penelitian adalah menerapkan arsitektur MVVM pada pengembangan aplikasi AI.YAM yang dapat mendeteksi ayam mati melalui gambar dari kamera atau galeri. Penelitian ini diharapkan dapat memberikan kontribusi nyata dalam pengembagan sistem yang mendukung produktifitas peternakan unggas di Indonesia dengan melakukan pengujian fungsionalitas dan analisis performa.

II. METODOLOGI PENELITIAN

Langkah-Langkah metode penelitian ini antara lain Identifikasi Masalah, Studi Literatur, Analisis Kebutuhan, Perancangan Aplikasi, dan Pengujian. Seperti pada Gbr.1 berikut :



Gbr.1 Alur Tahapan Penelitian

A. Identifikasi Masalah

Penelitian ini berfokus pada penerapan arsitektur MVVM untuk aplikasi AI.YAM dimana aplikasi tersebut dapat mendeteksi ayam mati melalui gambar ayam yang diambil dari kamera atau galeri. Dalam penelitian ini, masalah yang ingin dipecahkan adalah untuk mengetahui performa pada sistem operasi android versi 16 dalam menilai waktu respon, CPU dan memori yang digunakan saat proses klasifikasi gambar ayam.

B. Studi Literatur

Untuk menjadikan penelitian ini sebagai referensi yang dapat dipercaya, maka dilakukan pengumpulan data dan informasi dari berbagai sumber yang dapat dipercaya. Oleh karena itu, jurnal, artikel dan dokumen resmi dari keunggulan arsitektur MVVM akan dijadikan sebagai sumber acuan dalam penelitian ini. Hal ini untuk memastikan keakuratan dan validitas data yang digunakan dalam penelitian dan untuk menjadikan hasil penelitian sebagai referensi yang dapat dipercaya oleh pembaca.

C. Analisis Kebutuhan

Analisis kebutuhan adalah analisis yang dibutuhkan untuk mengidentifikasi bagian teknis dan fungsional yang diperlukan pada aplikasi AI.YAM, yang dikembangkan dengan arsitektur MVVM dan terintegrasi dengan model AI melalui API yang dapat dijalankan pada versi android 16. Analisis kebutuhan perangkat keras, perangkat lunak dan sistem yang mendukung pengujian performa, seperti waktu respon, penggunaan CPU dan memori, sehingga pengujian dapat dilakukan secara terkontrol dan menggambarkan kondisi pengguna. Berikut adalah beberapa kebutuhan yang dibagi menjadi 4 (empat) bagian :

(1) Kebutuhan Perangkat Keras (*Hardware*)

Perangkat keras yang diperlukan dalam penelitian ini yaitu laptop dan telepon pintar (smartphone). Laptop yang digunakan untuk pengembangan dan pengujian dengan spesifikasi sebagai berikut :

Nama Perangkat : ASUS TUF Gaming F15
Prosesor : Intel(R) Core(TM) i5-10300H
RAM : 16 GB
Storage : 477 GB
Sistem Operasi : Windows 11 Home 64-bit

Sedangkan, untuk telepon pintar (smartphone) yang digunakan untuk pengujian dengan spesifikasi sebagai berikut :

Nama Perangkat : ROG Phone 8 Pro
Prosesor : Snapdragon 8 Gen 3
RAM : 16 GB
Storage : 512 GB
Sistem Operasi : Android 16

(2) Kebutuhan Perangkat Lunak (*Software*)

Pada tabel 1 terdapat perangkat lunak dan teknologi yang diperlukan dalam penelitian ini adalah :

TABEL I
KEBUTUHAN PERANGKAT LUNAK

Perangkat Lunak	Keterangan
Android Studio	Perangkat lunak yang digunakan untuk pembuatan aplikasi
Kotlin	Bahasa yang digunakan dalam pembuatan program aplikasi
Dagger Hilt	Library yang digunakan untuk mengatur dan membagikan kebutuhan atau alat ke setiap bagian aplikasi secara otomatis
Firebase	Platform untuk *authentication* pengguna
Glide	Library yang digunakan untuk menangani proses pemuatan dan menampilkan gambar di aplikasi
Retrofit	Library HTTP *client* yang digunakan untuk berkomunikasi dengan API
Camera X	Komponen android jetpact yang digunakan untuk memanfaatkan fitur kamera
Media Store	API android untuk mengakses, menyimpan dan mengelola file media di perangkat
Android Jetpack	Kumpulan library android
Material Design	Digunakan untuk mendesain UI atau UX pada aplikasi
MPAndroidChart	Library untuk visualisasi data grafik berupa chart pada aplikasi
Android Profiler	Alat yang digunakan untuk menilai penggunaan CPU dan memori
Android Logcat	Alat yang untuk menilai waktu respon
Visual Studio Code	Perangkat lunak yang digunakan untuk membuat backend aplikasi
python	Bahasa yang digunakan dalam program *backend* aplikasi
Node.js	Digunakan untuk membuat API dan *backend* aplikasi yang membutuhkan pemrosesan cepat
Flask	Digunakan untuk membuat REST API sederhana untuk memanggil model AI

(3) Kebutuhan Fungsional

Tabel II berikut menunjukkan kebutuhan fungsional dalam penelitian ini :

TABEL II
KEBUTUHAN FUNGSIONAL

ID	Kebutuhan Fungsional	Keterangan	Prioritas
FR1	Mengambil gambar melalui kamera	Aplikasi dapat mengambil gambar ayam secara langsung menggunakan kamera perangkat	Harus
FR2	Memilih gambar dari galeri	Pengguna dapat memilih gambar ayam dari galeri untuk deteksi	Harus
FR3	Mengirim gambar ke *server*	Aplikasi dapat mengirim gambar ke *server* untuk memperoleh hasil klasifikasi	Harus
FR4	Menerima Hasil Prediksi	Aplikasi menerima hasil prediksi dari server dan menampilkan hasilnya	Harus
FR5	Menyimpan hasil prediksi	Aplikasi harus menyimpan hasil prediksi secara otomatis ke database *cloud* melalui API kemudian dapat menampilkannya di histori	Harus
FR6	Visualisasi data	Aplikasi dapat menampilkan grafik berbentuk chart	Harus

(4) Kebutuhan Non Fungsional

Tabel III berikut menunjukkan kebutuhan non fungsional dalam penelitian ini :

TABEL III
KEBUTUHAN NON FUNGSIONAL

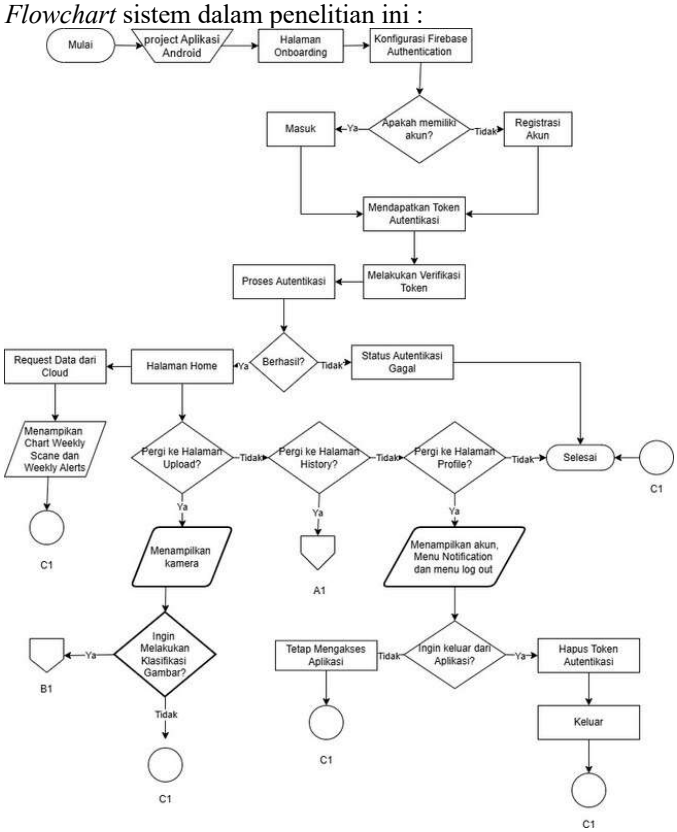
ID	Kebutuhan Non Fungsional	Keterangan	Prioritas
NFR1	Kemudahan penggunaan	Antarmuka pengguna (*user interface*) aplikasi harus sederhana dan mudah dipahami oleh peternak	Harus
NFR2	Aplikasi dapat dijalankan pada perangkat berbasis android	Aplikasi dapat digunakan di sistem operasi android versi 16	Harus
NFR3	Performa	Aplikasi dapat menampilkan hasil prediksi dalam waktu maksimal 3 detik setelah gambar dikirim ke *server*	Harus
NFR4	Penggunaan sumber daya	Aplikasi harus ringan, tidak menggunakan CPU dan memori terlalu banyak untuk mencegah perangkat menjadi lambat	Harus
NFR5	Stabilitas sistem	Aplikasi harus tetap stabil dan tidak mengalami *crash* selama proses pengujian yang berulang	Harus
NFR6	Monitoring performa	Aplikasi harus terhubung dengan alat monitoring seperti *android profiler* dan *android logcat* untuk mencatat hasil pengujian	Harus

D. Perancangan Aplikasi

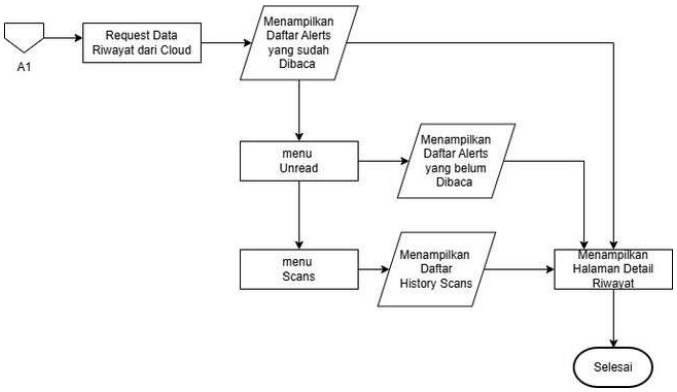
Perancangan bertujuan untuk memberikan gambaran desain sistem pada implementasi aplikasi AI.YAM. Berikut adalah desain sistem pada penelitian ini :

(1) Flowchart

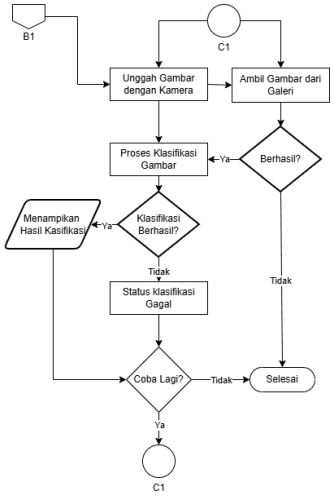
Flowchart memberikan gambaran desain untuk perancangan sistem aplikasi yang menggunakan arsitektur MVVM. Pada Gbr.2 Gbr.3 dan Gbr.4 berikut adalah *Flowchart* sistem dalam penelitian ini :



Gbr.2 Flowchart Utama Aplikasi



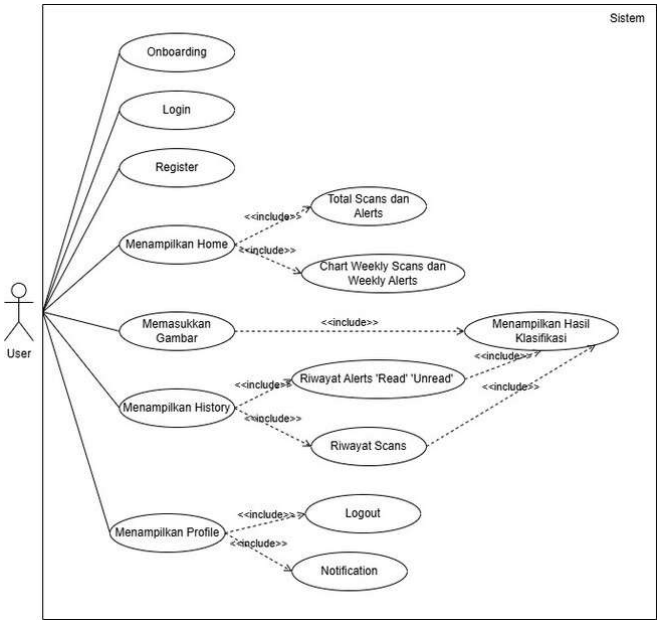
Gbr.3 Flowchart A1 History



Gbr.4 Flowchart B1 Klasifikasi Gambar

(2) Usecase Diagram

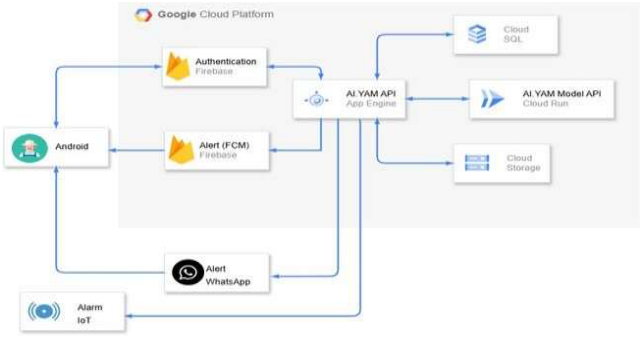
Usecase diagram dapat digunakan untuk memberikan gambaran sistem. Gbr.5 menunjukkan *usecase diagram* pada penelitian ini



Gbr.5 UseCase Diagram Aplikasi

(3) Arsitektur Aplikasi

Arsitektur aplikasi adalah desain sistem arsitektur pada aplikasi AI.YAM yang digunakan sebagai dasar dalam pembuatan aplikasi. Gbr.6 berikut menunjukkan arsitektur aplikasi AI.YAM :



Gbr.6 Arsitektur AI.YAM

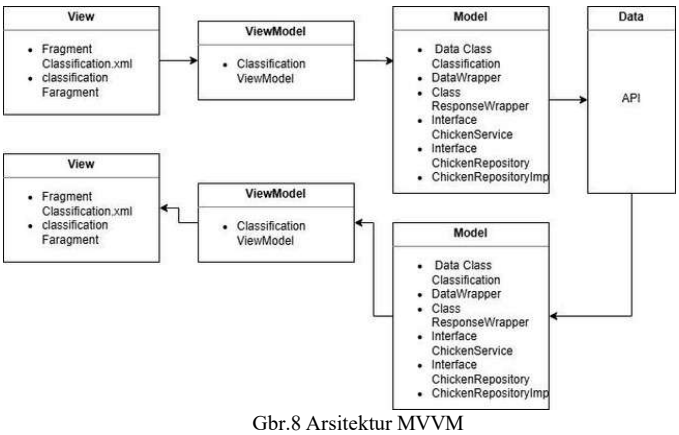
(4) Wireframe

Wireframe adalah representasi visual dari *user interface* sebuah aplikasi yang menampilkan tata letak, struktur, dan fungsionalitas dari setiap layar secara mendetail. Dalam penelitian ini terdapat *wireframe onboarding*, *login* dan lain sebaginya. Gbr.7 berikut adalah *wireframe* yang di uji dalam penelitian ini :

Gbr.7 Wireframe Aplikasi

(5) Arsitektur MVVM

Dalam pembuatan aplikasi menggunakan arsitektur MVVM. MVVM terdiri dari 3 komponen yaitu Model, View, dan ViewModel. Setiap komponen dalam arsitektur memiliki fungsi yang berbeda. Pada Gbr.8 berikut adalah diagram yang memvisualisasikan komponen dan teknologi yang digunakan serta interaksi antara komponen dan teknologi tersebut :



Gbr.8 Arsitektur MVVM

E. Pengujian

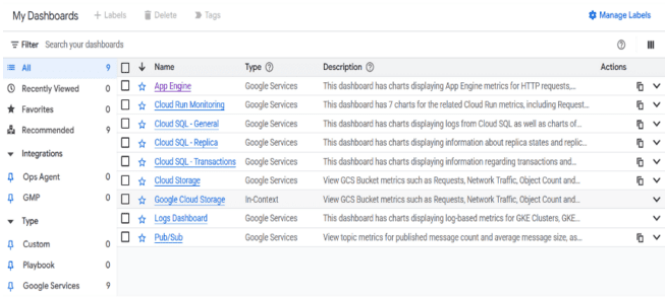
Pengujian yang dilakukan dalam penelitian ini adalah uji fungsionalitas dan uji performa. Uji fungsionalitas akan dilakukan perencanaan dengan pendekatan black box testing dan pada uji performa akan dilakukan dengan android versi 16, versi 15, versi 14, versi 11 dan versi 9. Sebelum pengujian, perangkat android harus memenuhi kondisi tertentu, tidak ada aplikasi latar belakang yang berjalan, notifikasi dimatikan dan perangkat smartphone harus terhubung dengan laptop melalui kabel USB dengan mode pengembang diaktifkan. Uji coba telah dilakukan untuk evaluasi tingkat akurasi, uji coba dilakukan menggunakan 24 gambar ayam, yang terdiri dari 13 ayam hidup dan 11 ayam mati. Dari hasil pengujian diperoleh satu kesalahan klasifikasi pada gambar ayam hidup yang diambil dari sudut belakang. Untuk memperoleh hasil yang lebih akurat akan dilakukan uji coba lanjutan dengan menggunakan variasi sudut pengambilan gambar dari beberapa posisi yang berbeda dengan jumlah ayam 1 sampai 5 ekor dalam satu pengambilan gambar. Gambar yang akan digunakan dalam uji coba lanjutan menggunakan gambar dari dataset dan gambar yang diambil langsung dari peternak ayam.

III. HASIL DAN PEMBAHASAN

Penelitian ini akan melakukan pengujian fungsionalitas dan pengujian performa pada aplikasi AI.YAM yang dikembangkan menggunakan arsitektur MVVM(*Model View ViewModel*). Tujuan pengujian ini dilakukan adalah untuk mengetahui fungsionalitas dan performa pada aplikasi AI.YAM yang dikembangkan menggunakan arsitektur MVVM dengan menilai waktu respon klasifikasi gambar, CPU dan memori.

A. *Implementasi Backend*

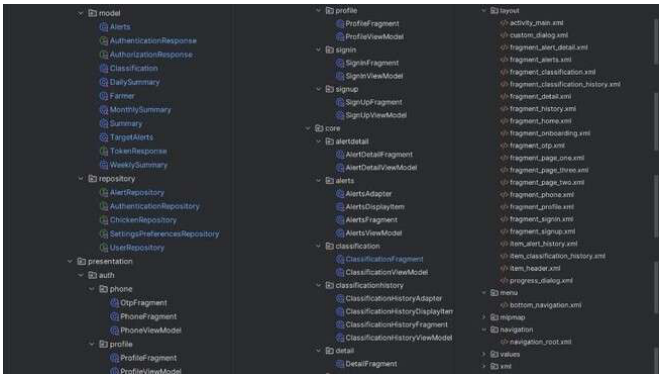
Pada pengembangan aplikasi *google cloud platform* berperan penting dalam sistem pengelolaan aplikasi. Gbr.9 berikut adalah layanan yang digunakan dalam pengembangan aplikasi :



Gbr.9 Layanan Google Cloud Platform

B. Penerapan Arsitektur MVVM

Arsitektur MVVM terdiri dari model, view, dan viewmodel. Berikut adalah class-class dalam penerapan arsitektur pada aplikasi AI.YAM :



Gbr.10 Arsitektur MVVM pada AI.YAM

Pada Gbr.10 tersebut terdapat class class yang mengelola aplikasi. Berikut adalah penjelasan MVVM bagian klasifikasi pada aplikasi :

1) Model

Pada bagian model ini terdapat enam komponen utama yang digunakan untuk klasifikasi aplikasi AI.YAM yang bertanggung jawab untuk mereprentasikan data, logika bisnis dan komunikasi dengan *server*.

```
@Parcelize
data class Classification(
    @SerializedName("id")
    val id: Int?,
    @SerializedName("media_url") //url gambar
    val mediaUrl: String,
    @SerializedName("dead_chicken") //hasil true,false
    val deadChicken: Boolean,
    @SerializedName("created_at") //waktu klasifikasi
    val createdAt: String,
    @SerializedName("is_alert") //status alert
    val isAlert: Boolean?
): Parcelabl
```

Gbr.11 Data Class Classification

Pada Gbr.11 *class* ini dianotasikan dengan *@Parcalize* dan mengimplementasikan *parcelable* agar objeknya dapat dikirim antar *fragment* melalui *save args navigation*. *@SerializedName* digunakan untuk mencocokan nama dengan *key Json* dari *respons server*.

```
class DataWrapper<T> (
    //respon api dalam bentuk json
    @SerializedName("data")
    val data: T,
)
```

Gbr.12 DataWrapper

Pada Gbr.12 *Class datawerapper* ini yang merepresentasikan struktur *Json* dari *server*. *Server* selalu membungkus data di dalam *kay* "data". *Datawrapper* diperlukan untuk memetakan struktur tersebut sehingga *Repository* dapat mengakses data langsung melalui *response.data.*

```
sealed class ResponseWrapper<out R> private constructor() {
//untuk menghandle respon api di ui
data class Success<out T>(val data: T) : ResponseWrapper<T>()
data class Error(val error: String) : ResponseWrapper<Nothing>()
data object Loading : ResponseWrapper<Nothing>()
}
```

Gbr.13 Class ResponseWrapper

Gbr.13 adalah *Sealed class responsewarper* ini berfungsi untuk semua kemungkinan kondisi respons dari *server* menjadi satu tipe data yang aman.

```
interface ChickenService {
//untuk upload gambar dan klasifikasi ayam
    @Multipart
    @POST("processes")
    suspend fun postChicken(
        @Header("Authorization") token: String,
        @Part file: MultipartBody.Part
    ): DataWrapper<Classification>
}
```

Gbr.14 Interface ChickenService

Pada Gbr.14 *Chickenservice* merupakan *interface retrofit* yang mendefinisikan seluruh *endpoint API* yang digunakan pada fitur klasifikasi. *postChicken* mengirim file gambar ke *server* menggunakan format *multipart/form-data* yang ditandai dengan *anotasi @Multipart*, kemudian *server* memproses gambar dan mengembalikan hasil klasifikasi berupa *DataWrapper.*

```
interface ChickenRepository {
    fun classifyChicken(file: File, mediaType: String): Flow<ResponseWrapper<Classification>>
}
```

Gbr.15 Interface ChickenRepository

Pada Gbr.15 menunjukkan *Interface chickenrepository* ini mendefinisikan operasi yang tersedia tanpa mengekspos detail implementasi, implementasinya dilakukan pada *chickenrepositoryimpl.*

```
override fun classifyChicken(file: File, mediaType: String): Flow<ResponseWrapper<Classification>> = flow
{
emit(ResponseWrapper.Loading)//(1) emit loading
    val requestBody = file.asRequestBody(mediaType.toMediaTypeOrNull())
    val multipartBody =  MultipartBody.Part.createFormData("file", file.name, requestBody)//(2)bungkus
file

    repeat(3) { attempt -> //(3)  retry hingga 3 kali
        try {
     val chicken = withToken(user, userRepository::getFirebaseToken) {
 chickenService.postChicken(it, multipartBody)}//(4)  autentikasi token
emit(ResponseWrapper.Success(chicken.data))
 //(5)  emit success
            return@flow
        } catch (e: Exception) {
            if (attempt == 2) {          emit(ResponseWrapper.Error(e.message.toString()))
//(6)  emit error
                return@flow
            } else {
                delay(500L) //(7)tunggu sebelum retry
            }
        }
    }
}
```

Gbr.16 Implementasi ChickenRepositoryImpl

Pada Gbr.16 *ChickenRepositoryImpl* implementasi dari *Chicken Repository* yang berisi seluruh logika bisnis pengiriman data ke *server.*

2) View

Terdapat lima komponen utama *view* untuk klasifikasi pada aplikasi AI.YAM terdiri dari dua *fragment*, dua dialog dan satu *layout* yang memiliki tanggung jawab yang berbeda beda.

```
tools:context=".presentation.core.classification.ClassificationFragment">
//tampilan kamera live
<androidx.camera.view.PreviewView
    android:id="@+id/previewView"
    android:layout_width="0dp"
    android:layout_height="0dp" app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"   app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
//tombol galeri
<androidx.cardview.widget.CardView
    android:id="@+id/galleryIB"
    style="@style/CircularIconButton"
    android:clickable="true"
    android:focusable="true"
app:layout_constraintBottom_toBottomOf="@+id/ captureIB"
app:layout_constraintEnd_toStartOf="@+id/captureIB"
app:layout_constraintStart_toStartOf="parent">
//tombol ambil foto
<androidx.cardview.widget.CardView
    android:id="@+id/captureIB"  style="@style/CircularIconButtonLarge"
    android:layout_marginBottom="106dp"
    android:clickable="true"
    android:focusable="true"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent">
//tombol flip kamera
<androidx.cardview.widget.CardView
    android:id="@+id/flipCameraIB"
    style="@style/CircularIconButton"
    android:clickable="true"
    android:focusable="true"
app:layout_constraintBottom_toBottomOf="@+id/ captureIB"
app:layout_constraintEnd_toEndOf="parent" app:layout_constraintStart_toEndOf="@+id/captureIB">
<Chronometer
    android:id="@+id/recodingTimerC"
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
    android:layout_marginEnd="24dp"
    android:layout_marginTop="12dp"
    android:text="@string/_10_20"
    android:textColor="@color/white"
    android:textSize="24sp"
    android:visibility="gone"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>
```

Gbr.17 Layout fragment_classification.xml

Gbr.17 tersebut merupakan *user interface* aplikasi murni tanpa logika apapun.

```
private fun startCamera() {
val cameraProviderFuture = ProcessCameraProvider.getInstance(requireContext())
    cameraProviderFuture.addListener({
        cameraProvider = cameraProviderFuture.get()
        bindCameraUseCases()
// inisialisasi selesai hubungkan kamera
    }, ContextCompat.getMainExecutor(requireContext()))
}
private fun bindCameraUseCases() {
    val preview = Preview.Builder() .setTargetAspectRatio(aspectRatio)
.setTargetRotation(binding.previewView.display .rotation)
        .build().also {
            it.surfaceProvider = binding.previewView. surfaceProvider // hubungkan ke PreviewView
        }
imageCapture = ImageCapture.Builder()
.setTargetAspectRatio(aspectRatio)
.setCaptureMode(ImageCapture.CAPTURE_MODE_ MINIMIZE_LATENCY) // kecepatan
.setTargetRotation(binding.previewView.display.rotation)
    .build()
val cameraSelector = CameraSelector.Builder()
        .requireLensFacing(lensFacing)
// kamera depan atau belakang
        .build()
    cameraProvider.unbindAll()
// lepas semua binding sebelumnya
    try {  cameraProvider.bindToLifecycle(viewLifecycleOwner, cameraSelector, preview, imageCapture)
    } catch (e: Exception) {
        e.printStackTrace()
        showToast("Failed: ${e.message}")
    }
}
private fun takePhoto() {
    val tempFile = File.createTempFile("photo_", ".jpg", requireContext().cacheDir)
    val outputOptions = ImageCapture.OutputFileOptions. Builder(tempFile).build()
    imageCapture.takePicture(
        outputOptions,
ContextCompat.getMainExecutor(requireContext()),
object : ImageCapture.OnImageSavedCallback {
override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
alertDialog(tempFile, "image/*").show()
//lanjut ke konfirmasi
        }
override fun onError(exception: ImageCapture Exception) {
showToast("Failed: ${exception.message}")
        }
    }
)
}
```

Gbr.18 Camera

Pada Gbr.18 menunjukkan *StartCamera()* menginisialisasi ProcessCamera *Provider* secara *asynchronous*. Setelah siap, *bindCameraUseCases()* menghubungkan *Preview* untuk menampilkan *live* kamera ke *PreviewView* dan *ImageCapture*

untuk mengambil foto. *cameraProvider.unbindAll()* dipanggil lebih dulu agar tidak terjadi konflik saat kamera *diflip.*

```
private val launcherGallery = registerForActivityResult(
ActivityResultContracts.PickVisualMedia()
) { uri ->
    isGalleryLaunched = false
binding?.galleryIB?.isEnabled = true
// aktifkan kembali tombol galeri
    if (uri != null) {
        onMediaSelected(uri)
    }
}
private fun startGallery() {
    if (isGalleryLaunched) return
// mencegah peluncuran ganda
    isGalleryLaunched = true
    binding.galleryIB.isEnabled = false
// nonaktifkan tombol selama galeri terbuka
    launcherGallery.launch(    PickVisualMediaRequest(ActivityResultContracts. PickVisualMedia.ImageOnly)
    )
}
private fun onMediaSelected(uri: Uri) {
    val file = uriToFile(uri)
    val mediaType = getMediaType(uri) Handler(Looper.getMainLooper()).postDelayed({
// pastikan Fragment masih aktif sebelum tampilkan dialog
if (_binding != null && isAdded && activity?.isFinishing == false) {
alertDialog(file, mediaType).show()
        }
    }, 300) // delay 300ms menghindari crash karena galeri belum sepenuhnya tertutup
}
private fun getMediaType(uri: Uri): String {
    val contentResolver = requireContext().contentResolver
    return contentResolver.getType(uri) ?: "*/*"
}
private fun uriToFile(uri: Uri): File {
    val contentResolver = requireContext().contentResolver
    val mimeType = contentResolver.getType(uri) ?: "*/*"
    val fileExtension = MimeTypeMap.getSingleton()     .getExtensionFromMimeType(mimeType) ?: "tmp"
    val storageDir = requireContext().getExternalFilesDir(Environment.DIRECTORY_PICTURES)
    val file = File.createTempFile("temp_", ".$fileExtension", storageDir)
    try {
        val inputStream = contentResolver.openInputStream(uri)
        val outputStream = FileOutputStream(file)  inputStream?.copyTo(outputStream)
// salin konten URI ke File
        inputStream?.close()
        outputStream.close()
    } catch (e: IOException) {
        showToast(e.message.toString())
    }
    return file
}
```

Gbr.19 Galeri

Pada Gbr.19 tersebut, URI dari galeri tidak dapat langsung dikirim ke *server*. *uriToFile*() membaca konten URI melalui *ContentResolver* dan menyalinnya ke objek File sementara di *DIRECTORY_PICTURES*. *isGalleryLaunched* mencegah galeri terbuka dua kali apabila tombol dipencet lebih dari sekali. *Handler.postDelayed*(300ms) pada *onMediaSelected* memastikan dialog konfirmasi tidak muncul sebelum galeri benar benar tertutup.

```
@SuppressLint("SetTextI18n")
private fun alertDialog(file: File, mediaType: String): AlertDialog {
    val dialog = CustomAlertDialog(
        context = requireActivity(),
        title = "Confirmation",
        message = "Proceed with classification?",
        negativeButtonClick = {}
// batal → tidak ada aksi
    ) { classify(file, mediaType) }
// konfirmasi → jalankan klasifikasi
    return dialog.alert()
}
```

Gbr.20 AlertDialog

Pada Gbr.20, sebelum klasifikasi dijalankan, pengguna diminta konfirmasi melalui *CustomAlertDialog*. Apabila pengguna menyetujuinya, *Fragment* memanggil fungsi *classify.*

```
private fun classify(file: File, mediaType: String) {
    classifyJob?.cancel()
// batalkan job sebelumnya jika masih ada yang berjalan
    classifyJob = lifecycleScope.launch {
        viewModel.classify(file, mediaType).collect {
            handleOnClassify(it)
        }
    }
}
private fun handleOnClassify(result: ResponseWrapper<Classification>) {
    when (result) {
    is ResponseWrapper.Error -> {    progressDialogFragment?.dismiss()
progressDialogFragment = null showToast(result.error)
        }
    is ResponseWrapper.Loading -> {
showToast("File has been uploaded")
  if (progressDialogFragment == null) {
progressDialogFragment = LoadingDialog()
          }  progressDialogFragment?.show(parentFragment Manager, "progressDialog")
        }
    is ResponseWrapper.Success -> {  progressDialogFragment?.dismiss()
progressDialogFragment = null
val action = ClassificationFragmentDirections           .actionClassificationFragmentToDetailFragment
(result.data)           findNavController().navigate(action)
// navigasi ke DetailFragment
        }
    }
}
override fun onDestroyView() {
    classifyJob?.cancel()
// pastikan coroutine berhenti saat Fragment mati
    super.onDestroyView()
    _binding = null
}
```

Gbr.21 Mengobservasi *Flow* dari *ViewModel*

Pada Gbr.21 tersebut *progressDialogFragment* dicek *null* sebelum dibuat, agar *LoadingDialog* tidak muncul duplikat. Setelah *Success* atau *Error*, *progressDialogFragment* diset *null* kembali untuk mereset *state dialog. classifyJob?.cancel()* pada *onDestroyView()* memastikan tidak ada *coroutine* yang berjalan di latar belakang setelah *Fragment* dihancurkan.

```
class CustomAlertDialog(
    private val context: Context,
    private val title: String,
    private val message: String,
    private val negativeButtonClick: () -> Unit,
    private val positiveButtonClick: () -> Unit) {
    fun alert(): AlertDialog {
        val dialogView = LayoutInflater.from(context).inflate(R.layout.custom_dialog, null)
dialogView.findViewById<TextView> (R.id.dialog_title).text = title
dialogView.findViewById<TextView> (R.id.dialog_message).text = message
        val dialog = AlertDialog.Builder(context)
            .setView(dialogView)
            .create()
        val positiveButton = dialogView.findViewById<Button>(R.id.dialog_positive_button)
        val negativeButton = dialogView.findViewById<Button>(R.id.dialog_negative_button)
positiveButton.setOnClickListener {   positiveButtonClick.invoke()// jalankan klasifikasi
            dialog.dismiss()
        } negativeButton.setOnClickListener {   negativeButtonClick.invoke() //batalkan
            dialog.dismiss()}
        return dialog
    }
}
```

Gbr.22 CustomAlertDialog

Pada Gbr.22 *class customalertdialog* ini komponen dialog dapat digunakan ulang di seluruh aplikasi. Dialog ini menerima judul, pesan, serta dua fungsi sebagai *callback* untuk tombol positif dan negatif. Penggunaan ( ) -> *Unit* membuat tombol dapat didefinisikan oleh pemanggil, tidak oleh dialog itu sendiri. Ini membuat *CustomAlertDialog* dapat digunakan pada berbagai skenario konfirmasi di seluruh aplikasi.

```
class LoadingDialog : DialogFragment() {
override fun onCreateDialog(savedInstanceState: Bundle?):
  Dialog {_binding = ProgressDialogBinding.inflate (layoutInflater)
    val dialog = Dialog(requireContext())
  dialog.requestWindowFeature(Window.FEATURE_ NO_TITLE)
  dialog.setContentView(binding.root)
  dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))
        return dialog
    }
```

Gbr.23 LoadingDialog

Gbr.23 menunjukkan *Class loadingdialog* ini menggunakan *DialogFragment* yang menampilkan indikator loading selama

proses klasifikasi berlangsung di *server*. Backgroundnya diset transparan dengan *ColorDrawable(Color. \TRANSPARE NT)* agar hanya animasi *loading* yang terlihat tanpa kotak dialog bawaan *Android*. Dialog ini ditampilkan saat *Response Wrapper .Loading* diterima dan berhenti saat *Success* atau *Error* diterima.

```
@SuppressLint("SetTextI18n")
private fun bindViews(classification: Classification) { binding.apply {
    if (classification.deadChicken) {
        chickenValue.text = "Dead Chicken Detected"
        chickenValueDesc.text = "Immediate action required to maintain flock health."
icon.setImageResource(R.drawable.ded)
    } else {
        chickenValue.text = "Chickens Alive and Well!"
        chickenValueDesc.text = "No immediate action necessary—flock in good health."
 icon.setImageResource(R.drawable.chiav)
    }
Glide.with(requireContext()).load(classification.mediaUrl).into(headerImage)
    signInButton.setOnClickListener {  findNavController().popBackStack()
    }
}}
```

Gbr.24 DetailFragment

Gbr.24 tersebut menampilkan fungsi hasil klasifikasi, Fungsi *bindViews* menampilkan berdasarkan nilai *deadChicken* dari objek *Classification*. jika bernilai *true*, ditampilkan teks peringatan beserta ikon ayam mati. jika bernilai *false*, ditampilkan teks kondisi normal beserta ikon ayam hidup. Gambar hasil klasifikasi dimuat dari URL server secara langsung menggunakan *Glide* ke komponen *headerImage*. Tombol *signInButton* berfungsi sebagai tombol kembali yang memanggil *popBackStack()* untuk kembali ke *Classification Fragment.*

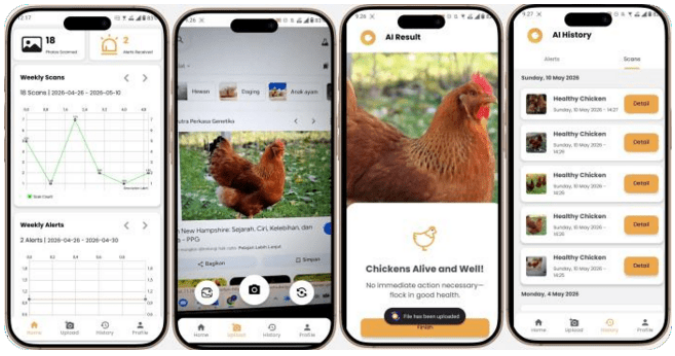
3) ViewModel

ViewModel berperan sebagai jembatan antara *Model* dan *View*. Pada aplikasi AI.YAM *ViewModel* tidak mengetahui detail dari *Model* atau detail *View ViewModel* hanya meneruskan pamanggilan ke *repository* dan langsung mengembalikan hasilnya ke *View* tanpa logika tambahan di tunjukkan pada Gbr.25 berikut ini :

```
@HiltViewModel
class ClassificationViewModel @Inject constructor(
    private val chickenRepository: ChickenRepository
) : ViewModel() {
    fun classify(file: File, mediaType: String) = chickenRepository.classifyChicken(file, mediaType)
}
```

Gbr.25 ViewModel

C. Aplikasi AI.YAM



Gbr.26 Aplikasi AI.YAM

Dari Gbr.26 menunjukkan bahwa terdapat empat tangkapan layar yang mencakup halaman home, halaman upload, hasil klasifikasi dan halaman histori.

D. Uji Fungsionalitas

Berikut merupakan hasil dari pengujian fungsionalitas yang telah dilakukan dengan pendekatan *black box testing* di tunjukkan pada tabel IV berikut :

TABEL IV
HASIL UJI FUNGSIONALITAS

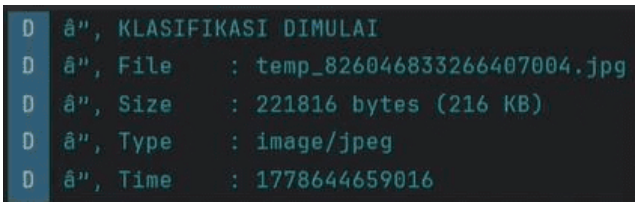
ID	Skenario	Deskripsi	Result
TC1	Tidak mengisi e-mail dan password lalu menekan tombol *sign in*	*View* mengirim data kosong ke *ViewModel*. *ViewModel* melakukan validasi dan mengembalikan pesan kesalahan ke *View*	[✓] berhasil [ ] tidak
TC2	Memasukkan nama, e-mail dan password dengan salah lalu menekan tombol *register*	*ViewModel* meneruskan data ke *Model* untuk validasi. *Model* mengembalikan status gagal. *ViewModel* menampilkan pesan kesalahan pada *View*	[✓] berhasil [ ] tidak
TC3	Melakukan *register* dengan benar	*ViewModel* mengirim data ke *Model* untuk disimpan. Jika berhasil, *Model* mengembalikan status sukses dan *ViewModel* mengarahkan pengguna ke halaman *Home*	[✓] berhasil [ ] tidak
TC4	Mengisi e-mail dan password dengan benar	*ViewModel* mengirim data ke *Model* untuk autentikasi. Jika *valid*, *Model* mengembalikan status sukses dan *ViewModel* mengarahkan ke halaman *Home*	[✓] berhasil [ ] tidak
TC5	Membuka halaman *home*	*ViewModel* mengambil data total *scans*, *alerts*, dan grafik dari *Model*, lalu mengirimkan hasil ke *View* untuk ditampilkan	[✓] berhasil [ ] tidak
TC6	Membuka halaman *upload*	*ViewModel* menginisialisasi fungsi kamera dan memerintahkan *View* menampilkan *preview* kamera	[✓] berhasil [ ] tidak
TC7	Melakukan klasifikasi gambar ayam melalui kamera depan atau belakang	*View* menangkap gambar, *ViewModel* mengirim gambar ke *Model*. *Model* memproses gambar dan mengirim hasil kembali ke *ViewModel*, lalu *View* menampilkan hasilnya	[✓] berhasil [ ] tidak

TC8	Terjadi masalah koneksi saat proses klasifikasi berjalan	*Model* gagal mengirim data ke server. *ViewModel* menangkap error dan menampilkan pesan peringatan pada *View*	[✓] berhasil [ ] tidak
TC9	Melakukan klasifikasi gambar ayam melalui galeri, gambar dengan resolusi dari 640 x 480 sampai 3840 x 2160	*View* memilih gambar dari galeri, *ViewModel* mengirim ke *Model*, lalu hasil klasifikasi dikembalikan dan ditampilkan pada *View*	[✓] berhasil [ ] tidak
TC10	Membuka halaman history	*ViewModel* mengambil daftar hasil klasifikasi yang sudah dibaca (*read*) dari *Model*, kemudian menampilkan daftar *read history* di *View*	[✓] berhasil [ ] tidak
TC11	Membuka fitur *unread*	*ViewModel* mengambil daftar hasil klasifikasi yang belum dibaca (*unread*) dari *Model*, lalu menampilkannya di *View*	[✓] berhasil [ ] tidak
TC12	Membuka fitur *scans*	*ViewModel* meminta data hasil *scans* terbaru dari Model, kemudian *View* menampilkan daftar *scans*	[✓] berhasil [ ] tidak
TC13	Membuka halaman *profile*	*ViewModel* memuat data akun pengguna dari *Model*, lalu *View* menampilkan fitur *logout*, fitur *notifikasi*, dan fitur *whatsapp notifikasi*	[✓] berhasil [ ] tidak
TC14	Membuka fitur whatsapp	*View* menampilkan *form input* nomor w*hatsapp*. *ViewModel* memanggil *Model* yang berkomunikasi dengan API untuk memverifikasi atau menyimpan nomor pengguna. Setelah *respons* diterima, *View* menampilkan pesan statusnya	[✓] berhasil [ ] tidak
TC15	Mengisi nomor whatsapp dengan salah	*View* menerima *input* nomor *whatsapp* dari pengguna dan mengirimkannya ke *ViewModel*. *ViewModel* memvalidasi format nomor, lalu meneruskan ke *Model*. Jika format tidak *valid*, *ViewModel* mengirimkan pesan peringatan ke *View* tanpa memanggil *Model*	[✓] berhasil [ ] tidak
TC16	*Logout* dari aplikasi dan masuk kembali	*View* mengirim perintah *logout* ke *ViewModel*. *ViewModel* menghapus token pengguna yang tersimpan dan mengarahkan kembali ke halaman *Sign In*, saat pengguna masuk kembali. *ViewModel* memverifikasi data *login* dan memanggil *Model*	[✓] berhasil [ ] tidak

E. Uji Performa

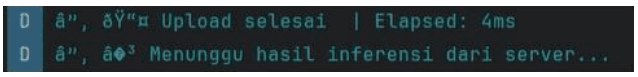
Pengujian performa pada aplikasi AI.YAM menggunakan fitur *logcat* dan *profiler* pada *android studio. logcat* digunakan untuk menilai waktu selama proses pada klasifikasi gambar dan profiler digunakan untuk menilai CPU dan memori saat proses klasifikasi gambar sedang berjalan. Berikut merupakan hasil dari pengujian pada versi 16 yang telah dilakukan:

1. Logcat



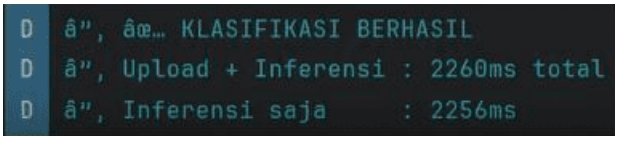
Gbr.27 Logcat Mulai

Hasil *logcat* pada Gbr.27 tersebut menunjukkan bahwa proses klasifikasi telah dimulai dengan jenis *file, size, type* dan *time* yang digunakan untuk klasifikasi.



Gbr.28 Logcat Upload

Hasil *logcat* pada Gbr.28 menunjukkan bahwa gambar telah dikirim ke *server* dalam waktu 4ms dan menunggu hasil *inferensi* dari *server.*

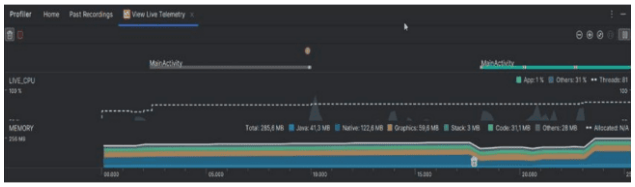


Gbr.29 logcat Hasil

Hasil *logcat* pada Gbr.29 ini menunjukkan hasil *inferensi* berhasil didapatkan dari *server* dengan total waktu 2260ms dan waktu *inferensi* 2256ms. Dari hasil logcat tersebut dapat disimpulkan bahwa proses klasifkasi gambar ayam pada AI.YAM memerlukan waktu tidak lebih dari 3 detik.

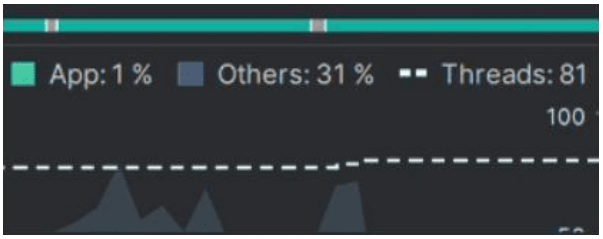
2. Profiler

Berikut adalah hasil dari pengujian performa yang telah dilakukan:



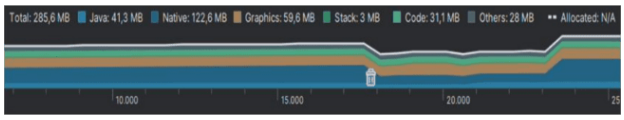
Gbr.30 Hasil Profiler

Gbr.30 tersebut menunjukkan hasil performa aplikasi AI.YAM saat proses klasifikasi gambar. Berikut penjelasan dari CPU dan memori yang didapatkan *profiler* pada aplikasi:



Gbr.31 CPU

Pada Gbr.31 tersebut menunjukkan penggunaan CPU pada aplikasi AI.YAM memakai 1% selama proses klasifikasi berlangsung. Android *profiler* juga memberikan nilai *others* 31% dan *threads* 81. 31% tersebut merupakan CPU yang sedang berjalan di latar belakang pada perangkat termasuk sistem operasi android itu sendiri. *Threads* 81 merupakan komponen komponen yang sedang berjalan secara bersamaan pada aplikasi AI.YAM seperti *thread CameraX* untuk pemrosesan camera, *thread* jaringan untuk komunikasi dengan *server* dan komponen *thread* lainnya.



Gbr.32 Memori

Pada Gbr.32 Tersebut menjelaskan penggunaan memori yang digunakan pada aplikasi AI.YAM selama proses klasifikasi berlangsung. Total memori yang digunakan mencapai 285,6MB yang terdiri dari memori *Java* sebesar 42,3MB untuk objek *Kotlin/Java* seperti *ViewModel* dan *Fragment*, *memori Native* sebesar 122,6MB yang digunakan oleh *library CameraX*, memori *Graphics* sebesar 59,6MB untuk *rendering* camera *preview*, memori *stack* sebesar 3MB digunakan untuk menyimpan informasi eksekusi dari 81 *threads* yang aktif. Memori Code sebesar 31,1MB untuk *bytecode* aplikasi, dan memori *others* sebesar 28MB untuk sistem lainnya. Hasil ini menunjukkan bahwa aplikasi berjalan dengan efisien dan tidak menyebabkan tekanan memori pada perangkat.

F. Table Hasil Pengujian Pada Versi Android

Table hasil dari pengujian pada android versi 16, versi 15, versi 14, versi 11 dan versi 9 yang telah dilakukan pada tabel V:

TABEL V
PENGUJIAN VERSI ANDROID

Versi Android	Logcat	CPU	Memori
Android 16	2260ms	1%	285,6MB
Android 15	2025ms	1%	277,7MB
Android 14	2119ms	1%	249,4MB
Android 11	3719ms	1%	259MB
Android 9	-	-	-

Pada android versi 9 atau dibawahnya aplikasi AI.YAM tidak dapat *terinstall* dan *androidnya crash.*

IV. KESIMPULAN

Bardasarkan seluruh rangkaian proses penelitian, mulai dari bagian analisi kebutuhan, perancangan aplikasi, implementasi dan pengujian yang telah dilakukan pada aplikasi AI.YAM, maka dapat ditarik beberapa kesimpulan utama sebagai berikut:

Aplikasi AI.YAM telah berhasil dibuat dengan penerapan arsitektur MVVM. Penerapan arsitektur ini memisahkan logika bisnis (*ViewModel*) dari *user interface(View)*, sehingga memudahkan dalam pengembangan aplikasi yang dapat mengonsumsi *API endpoint* yang terdapat *model AI.* Selain itu, aplikasi juga dapat terhubung dengan layanan *cloud* untuk melakukan *sinkronisasi* dalam menyimpan data.

Pengujian fungsionalitas menunjukkan bahwa seluruh fitur utama pada aplikasi AI.YAM dapat berjalan dan sesuai dengan 16 rancangan yang diharapkan. Sementara itu, berdasarkan pengujian performa yang telah dilakukan dengan perangkat sistem operasi android 16, aplikasi menunjukkan waktu respon tidak lebih dari 3 detik dengan pemakaian CPU mencapai 1% dan penggunaan memori mencapai 285,6MB selama proses klasifikasi sedang berlangsung sampai mendapatkan hasilnya.

Beberapa saran untuk pengembangan aplikasi AI.YAM selanjutnya:

Pengembangan selanjutnya diharapkan tidak hanya terbatas pada pengambilan gambar untuk klasifikasi tetapi menggunakan video atau cctv untuk memantau ayam secara langsung. Menambahkan penyimpanan mode offline saat aplikasi tidak terhubung ke jaringan, sehingga user dapat melihat data dan histori terakhir klasifikasi yang terhubung pada jaringan.

REFERENSI

[1] *Produksi Daging Ayam Ras Pedaging menurut Provinsi*. (2025, April 28). 2024. https://www.bps.go.id/id/statistics-table/2/NDg4IzI=/produksi-daging-ayam-ras-pedaging-menurut-provinsi--ton-.html

[2] Himel, G. M. S., Islam, M. M., Kader, M. N., & Rahman, M. (2025). Artificial intelligence-based smart galliformes farm management system. *Journal of Umm Al-Qura University for Engineering and Architecture*, *16*(1), 64–77. https://doi.org/10.1007/s43995-024-00089-7

[3] Aini, N. Q., & Septaningsih, A. C. (2025). Artificial Intelligence untuk Peningkatan Produktivitas Ternak: Pendekatan Inovatif dalam Peternakan. *Jurnal Ilmiah Peternakan Halu Oleo*, 7(1), 9–17. https://doi.org/10.56625/jipho.v7i1.234

[4] Nalendra, A. K., & Waspada, H. P. (2025). Smart Poultry Farming: A Mobile-Based IoT System for Real-Time Broiler Monitoring and Management. *International Journal of Electronics and*

Communications Systems, *5*(1), 81–91. https://doi.org/10.24042/ijecs.v5i1.27622

[5] Qonita, V. A., Muchlisinia, N., Sugiana, L. R., Stefanny, A., & Ridha, A. (2025). Analisis Perbandingan Fitur Pada Aplikasi My Cattle Manager dan CowMaster Untuk Managemen Peternakan Sapi. *SIMKOM*, *10*(2), 242–253. https://doi.org/10.51717/simkom.v10i2.895

[6] Asimiyu, Z. (2024). *Evaluating Android Architectural Patterns: A Deep Dive into MVP, MVVM, and MVI.* https://www.researchgate.net/publication/385492745_Evaluating_Android_Architectural_Patterns_A_Deep_Dive_into_MVP_MVVM_and_MVI

[7] Riyadhi, I. M., Intan Purnamasari, & Kamal Prihandani. (2023). PENERAPAN POLA ARSITEKTUR MVVM PADA PERANCANGAN APLIKASI PENGADUAN MASYARAKAT BERBASIS ANDROID. *INFOTECH Journal*, *9*(1), 147–158. https://doi.org/10.31949/infotech.v9i1.5246

[8] Arfianto, B., & Prapanca, A. (2024). Analisis Perbandingan Performa Pola Arsitektur Model-View-ViewModel (MVVM) dan Model-View-Presenter (MVP) pada Pengembangan Aplikasi Desa Wisata Berbasis Android. *Journal of Informatics and Computer Science*, *06*. https://doi.org/10.26740/jinacs.v6n02.p465-478

[9] Zakaria, A. H., & Nuryana, I. K. D. (2023). *MVVM Perangkat Lunak Android dan Analisis Arsitektur MVP dengan Studi Kasus Aplikasi iTourism*. https://ejournal.unesa.ac.id/index.php/jinacs/article/download/51585/42072

[10] Hammamul Achdan Epiloksa*, D. S. K. M. A. (2022). Effect Of MVVM Architecture Pattern on Android Based Application Performance. *JURNAL MEDIA INFORMATIKA BUDIDARMA* , 1949–1955. https://www.academia.edu/download/101069301/2977.pdf

[11] Firdaus, L. (2024). Analisis Performa Operating System Android Pada Implementasi Aplikasi Media Belajar Bahasa Isyarat IConnect. *Jurnal PETISI*, *5*(2). https://e-journal.unimudasorong.ac.id/index.php/jurnalpetisi/article/view/769