

Pengembangan Aplikasi Mobile Eco-system Livestock untuk Pencatatan Ternak Menggunakan Framework Flutter



Disusun Oleh:

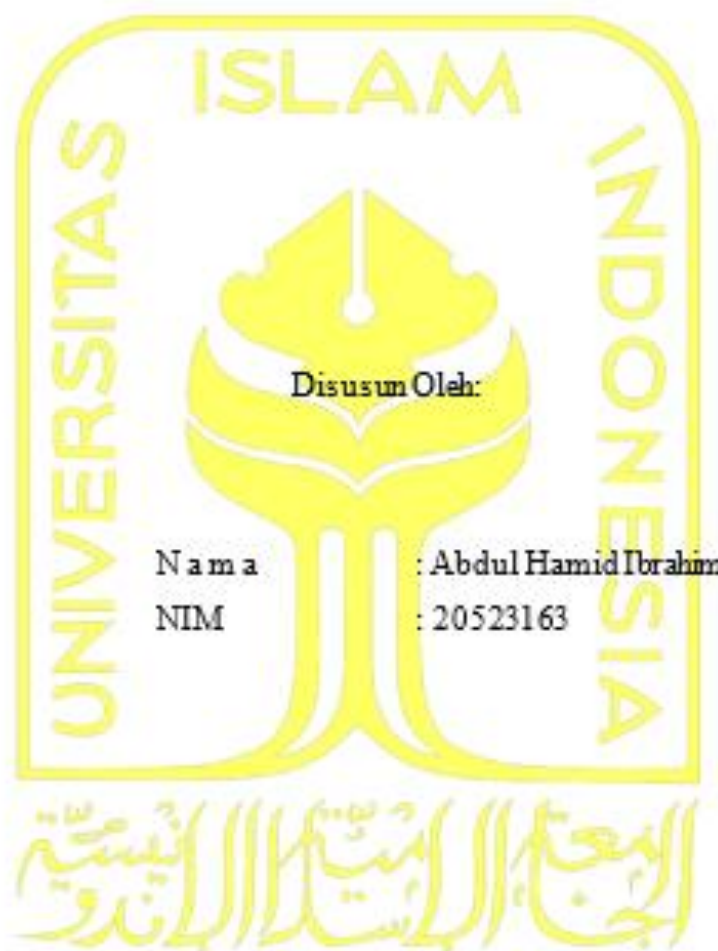
N a m a : Abdul Hamid Ibrahim
NIM : 20523163

**PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2026**

HALAMAN PENGESAHAN DOSEN PEMBIMBING

**Pengembangan Aplikasi Mobile Eco-system Livestock untuk
Pencatatan Ternak Menggunakan Framework Flutter**

TUGAS AKHIR



Yogyakarta, 27 April 2026

Pembimbing,

(Hari Setiaji, S.Kom., M.Eng.)

HALAMAN PENGESAHAN DOSEN PENGUJI

Pengembangan Aplikasi Mobile Eco-system Livestock untuk Pencatatan Ternak Menggunakan Framework Flutter

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 27 April 2026

Tim Penguji

Hari Setiaji, S.Kom., M.Eng.

Anggota 1

Rahadian Kurniawan, S.Kom., M.Kom

Anggota 2

Dr. Novi Setiani, S.T., M.T.

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia



(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Abdul Hamid Ibrahim

NIM : 20523163

Tugas akhir dengan judul:

Pengembangan Aplikasi Mobile Eco-system Livestock untuk Pencatatan Ternak Menggunakan Framework Flutter

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 27 April 2026



(Abdul Hamid Ibrahim)

HALAMAN PERSEMBAHAN

Alhamdulillahirabbil'aalamin, puji syukur saya haturkan kepada Allah SWT yang telah memberikan kesehatan, rahmat dan hidayah, sehingga saya masih diberikan kesempatan untuk menyelesaikan tugas akhir ini. Shalawat serta salam saya curahkan kepada Nabi Muhammad SAW, beserta keluarga, sahabat dan pengikutnya yang setia.

Laporan tugas akhir ini saya persembahkan kepada kedua orang tua saya, Ibrahim Noer dan Aznel Nofia, yang telah memberikan dukungan baik secara finansial, doa, dan dukungan lain yang tidak ada hentinya. Semoga laporan tugas akhir ini dapat membalas kebaikan dan memberikan kebahagiaan bagi kalian. Terima kasih juga untuk saudara-saudara saya, yang telah memberikan masukan, semangat, dan pengertiannya selama ini.

Ucapan terima kasih yang tak terhingga untuk dosen pembimbing penelitian tugas akhir saya, Bapak Hari Setiaji, S.Kom., M.Eng., atas bimbingan yang telah bapak berikan kepada saya selama ini. Semoga ilmu, waktu, dan kesabaran yang telah bapak berikan kepada saya dapat bermanfaat bagi saya dan mendapatkan balasan yang setimpal dari Allah SWT.

Terima kasih juga kepada tim peneliti kolaborasi dari Universitas Sriwijaya dan tim IT eksternal yang telah memberikan kesempatan dan ruang berkembang dalam pengembangan platform Eco-system Livestock.

HALAMAN MOTO

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya.”

(QS. Al-Baqarah: 286)

KATA PENGANTAR

Segala puji syukur kehadiran Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga saya dapat menyelesaikan penyusunan laporan tugas akhir ini dengan baik. Sholawat serta salam semoga senantiasa tercurah kepada Nabi Muhammad SAW, yang merupakan teladan bagi seluruh umat manusia.

Laporan tugas akhir ini disusun sebagai salah satu syarat dalam menyelesaikan studi saya di Program Studi Informatika, Universitas Islam Indonesia. Penelitian disusun dengan judul "PENGEMBANGAN APLIKASI MOBILE ECO-SYSTEM LIVESTOCK UNTUK PENCATATAN TERNAK MENGGUNAKAN FRAMEWORK FLUTTER". Selama menyusun tugas akhir ini, saya merasakan berbagai tantangan, baik keterbatasan waktu, pengumpulan data, dan kesulitan teknis lainnya. Perasaan senang dan sedih telah saya lewati dalam penulisan tugas akhir ini, berkat dukungan dari berbagai pihak, saya dapat menyelesaikan penelitian ini dengan sebaik-baiknya. Penulis menyadari dalam laporan tugas akhir ini terdapat banyak kekurangan. Oleh sebab itu diperlukan kritik dan saran yang membangun.

Dengan penuh rasa hormat, saya ingin menyampaikan ucapan terima kasih yang tulus, kepada:

1. Allah SWT yang telah memberikan kesehatan, kekuatan, dan kelancaran selama penelitian dan penyusunan laporan Tugas Akhir ini,
2. Orang tua saya, Ibrahim Noer dan Aznel Nofia yang selalu memberikan saya dukungan serta doa yang tidak pernah putus, dan memberikan kepercayaan kepada saya untuk menuntaskan kewajiban kuliah ini,
3. Saudara saya, Maulana Ihsan Ibrahim dan Mardhiyah Fadila Ibrahim, terima kasih telah menjadi saudara sekaligus teman yang baik,
4. Bapak Prof. Fathul Wahid, S.T., M.Sc., Ph.D., selaku Rektor Universitas Islam Indonesia,
5. Bapak Prof. Dr. Ir. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia,
6. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Jurusan Informatika Universitas Islam Indonesia,
7. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D., selaku Ketua Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia,

8. Bapak Dr. Syarif Hidayat, S.Kom., M.I.T., selaku Dosen Pembimbing Akademik yang selalu memberikan arahan selama masa perkuliahan,
9. Bapak Hari Setiaji, S.Kom., M.Eng., selaku Dosen Pembimbing Tugas Akhir yang telah meluangkan waktunya untuk memberikan bimbingan kepada saya selama pelaksanaan penelitian hingga penyusunan laporan Tugas Akhir,
10. Tim peneliti dari Universitas Sriwijaya dan tim IT eksternal yang telah berkolaborasi dalam pengembangan platform Eco-system Livestock dan menyediakan sumber daya teknis bagi penelitian ini,
11. Dosen Informatika yang telah memberikan ilmu yang bermanfaat semasa saya kuliah,
12. Teman-teman saya yang telah menemani dan menyemangati perjalanan kuliah saya dari awal perkuliahan hingga saat ini, serta semua pihak yang telah berjasa dalam penelitian ini yang tidak bisa saya sebutkan satu-persatu.

Akhir kata, penulis berharap agar segala yang dituangkan dalam penelitian ini dapat bermanfaat bagi pembacanya.

Yogyakarta, 27 April 2026

(Abdul Hamid Ibrahim)

SARI

Sektor peternakan di Indonesia masih menghadapi kendala pencatatan ternak yang sebagian besar dilakukan secara manual menggunakan kertas dan *ear tag* visual. Pendekatan tersebut rentan terhadap kesalahan, sulit direkapitulasi, dan tidak terhubung satu sama lain. Universitas Islam Indonesia, Universitas Sriwijaya, dan tim IT eksternal mengembangkan platform Eco-system Livestock untuk mengatasi masalah ini, namun belum tersedia klien *mobile* yang dapat digunakan langsung oleh penyuluh dan peternak di lapangan. Penelitian ini bertujuan mengembangkan aplikasi *mobile Android* sebagai klien dari platform tersebut, dengan dukungan pemindai *QR code* untuk identifikasi ternak.

Aplikasi dikembangkan menggunakan *framework Flutter* dengan pola arsitektur *Model-View-ViewModel* (MVVM) dan *state management Provider*. Aplikasi ini melayani dua peran pengguna dengan hak akses yang berbeda, yaitu penyuluh produksi sebagai supervisor untuk memantau peternak binaan pada lingkup desa, dan peternak yang melalui aplikasi hanya menampilkan data hewan miliknya sendiri. Aplikasi memanggil *REST API* yang sudah disediakan tim *backend* untuk operasi pembacaan dan penulisan data. Pengujian dilakukan dengan tiga metode, yaitu *black-box testing* untuk memvalidasi kebutuhan fungsional, *API integration testing* untuk memastikan komunikasi yang stabil dengan platform utama, dan pengujian *usability* menggunakan *System Usability Scale* (SUS).

Hasil penelitian menunjukkan aplikasi *mobile Eco-system Livestock* berhasil dikembangkan dan memenuhi seluruh kebutuhan fungsional yang telah dirumuskan untuk kedua peran pengguna. Pengujian *black-box* terhadap seluruh skenario fungsional menghasilkan status diterima, dan pengujian *API integration* memverifikasi kesesuaian komunikasi aplikasi dengan *REST API*. Pengujian *usability* dengan *System Usability Scale* (SUS) memperoleh skor rata-rata 71,1 yang tergolong kategori *Good* dan dapat diterima. Implementasi pemindai *QR code* berbasis kamera perangkat juga terbukti mempercepat proses identifikasi ternak dibanding *input* manual.

Kata kunci: aplikasi mobile, Eco-system Livestock, Flutter, MVVM, pencatatan ternak, QR code, *REST API*.

GLOSARIUM

<i>API</i>	Application Programming Interface, antarmuka yang memungkinkan komunikasi antara dua aplikasi atau sistem perangkat lunak yang berbeda.
<i>Backend</i>	Sisi server dari sebuah aplikasi yang bertanggung jawab atas pemrosesan data, logika bisnis, dan komunikasi dengan basis data.
<i>Black-box testing</i>	Metode pengujian perangkat lunak yang berfokus pada pengamatan input dan output tanpa melihat struktur kode internal.
<i>Dart</i>	Bahasa pemrograman yang dirancang oleh Google dan digunakan sebagai bahasa utama dalam pengembangan aplikasi berbasis Flutter.
<i>Ear tag</i>	Penanda berbentuk anting yang dipasang di telinga ternak sebagai identitas individual hewan.
<i>Flutter</i>	Kerangka kerja pengembangan aplikasi cross-platform yang dikembangkan oleh Google untuk membangun aplikasi mobile, web, dan desktop dari satu basis kode.
<i>Frontend</i>	Sisi aplikasi yang berinteraksi langsung dengan pengguna, mencakup tampilan dan elemen antarmuka.
<i>MVVM</i>	Model-View-ViewModel, pola arsitektur perangkat lunak yang memisahkan logika tampilan dari logika bisnis dan data.
<i>NFC</i>	Near Field Communication, teknologi komunikasi nirkabel jarak dekat yang memungkinkan pertukaran data antar perangkat.
<i>Provider</i>	Pustaka manajemen state pada Flutter yang berfungsi menyebarkan data dari satu titik sumber ke berbagai widget.
<i>QR code</i>	Quick Response code, jenis kode dua dimensi yang dapat dipindai untuk menyimpan dan membaca informasi.
<i>REST API</i>	Representational State Transfer Application Programming Interface, gaya arsitektur API yang menggunakan protokol HTTP untuk pertukaran data.

<i>RFID</i>	Radio Frequency Identification, teknologi identifikasi berbasis frekuensi radio yang dapat membaca data tag dari jarak tertentu tanpa kontak fisik.
<i>Smartphone</i>	Telepon genggam pintar dengan kemampuan komputasi, konektivitas internet, dan pemasangan aplikasi tambahan.
<i>State management</i>	Mekanisme pengelolaan data dan kondisi aplikasi agar tampilan tetap konsisten dengan data terbaru.
<i>ViewModel</i>	Komponen dalam pola MVVM yang berperan sebagai jembatan antara Model dan View, mengelola data tampilan dan logika presentasi.
<i>Widget</i>	Komponen dasar pembentuk antarmuka pengguna pada aplikasi Flutter, yang dapat berupa elemen tampilan maupun elemen logika tata letak.

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGESAHAN DOSEN PEMBIMBING	ii
HALAMAN PENGESAHAN DOSEN PENGUJI	iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR.....	iv
HALAMAN PERSEMBAHAN	v
HALAMAN MOTO	vi
KATA PENGANTAR.....	vii
SARI	ix
GLOSARIUM	x
DAFTAR ISI	xii
DAFTAR TABEL	xiv
DAFTAR GAMBAR.....	xv
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	4
1.3 Batasan Masalah	4
1.4 Tujuan Penelitian	5
1.5 Manfaat Penelitian	5
1.6 Sistematika Penulisan	5
BAB II KAJIAN PUSTAKA DAN LANDASAN TEORI.....	7
2.1 Penelitian Terdahulu	7
2.2 Landasan Teori.....	10
2.2.1 Inisiasi Pencatatan Ternak	10
2.2.2 Flutter	11
2.2.3 Dart.....	12
2.2.4 Android Studio	12
2.2.5 Arsitektur MVVM.....	13
2.2.6 <i>Provider</i>	13
2.2.7 <i>REST API</i>	14
2.2.8 QR Code.....	15
2.2.9 Pengujian Perangkat Lunak	15
BAB III METODOLOGI PENELITIAN	17

3.1	Metodologi dan Alur Penelitian	17
3.2	Ruang Lingkup Kontribusi pada Proyek Kolaborasi	19
3.3	Analisis Kebutuhan	21
3.3.1	Kebutuhan Fungsional	22
3.3.2	Kebutuhan Non-Fungsional	24
3.3.3	Kebutuhan Perangkat Keras dan Lunak	26
3.4	Perancangan Sistem	26
3.4.1	Arsitektur Aplikasi	26
3.4.2	Use Case Diagram	28
3.4.3	Activity Diagram	30
3.4.4	Pemetaan Kelas Model Dart dari Response API	34
3.5	Metode Pengujian	34
3.5.1	Black-box Testing	35
3.5.2	API Integration Testing	36
3.5.3	API Integration Testing	37
BAB IV IMPLEMENTASI DAN PENGUJIAN		38
4.1	Implementasi Aplikasi	38
4.1.1	Implementasi Arsitektur MVVM	38
4.1.2	Implementasi Autentikasi dan Role-Based Access	41
4.1.3	Implementasi Menu Data Peternak	47
4.1.4	Implementasi Menu Data Ternak	57
4.1.5	Implementasi Fitur Pemutakhiran Riwayat Ternak	60
4.1.6	Implementasi Pemindai QR Code	62
4.1.7	Implementasi Akses <i>REST API</i>	66
4.2	Hasil Pengujian	67
4.2.1	Hasil Black-box Testing	67
4.2.2	Hasil API Integration Testing	72
4.2.3	Pengujian Usability dengan System Usability Scale (SUS)	77
4.3	Pembahasan	79
BAB V KESIMPULAN DAN SARAN		82
5.1	Kesimpulan	82
5.2	Saran	83
DAFTAR PUSTAKA		85
LAMPIRAN		88

DAFTAR TABEL

Tabel 2.1 Perbandingan Penelitian Terdahulu	7
Tabel 3.1 Peta Kontribusi pada Proyek Kolaborasi Eco-system Livestock.....	20
Tabel 3.2 Kebutuhan Fungsional Aplikasi.....	22
Tabel 3.3 Format <i>Test Case Black-box Testing</i>	35
Tabel 3.4 Format <i>Test Case API Integration Testing</i>	36
Tabel 4.1 Hasil <i>Black-box Testing</i> Aplikasi Eco-system Livestock	68
Tabel 4.3 Hasil Pengujian <i>Response Time</i> API	76
Tabel 4.4 Rekapitulasi Rerata Jawaban Responden per Pernyataan SUS	77
Tabel 4.5 Perhitungan Skor SUS per Responden	78

DAFTAR GAMBAR

Gambar 3.1 Diagram Alur Penelitian	18
Gambar 3.2 Peta Kontribusi Penulis pada Proyek Kolaborasi Eco-system Livestock	20
Gambar 3.3 Diagram Arsitektur Aplikasi	27
Gambar 3.4 Struktur Folder Proyek Flutter	28
Gambar 3.5 Use Case Diagram Aplikasi Eco-system Livestock	29
Gambar 3.6 Activity Diagram Autentikasi (Login)	31
Gambar 3.7 Activity Diagram Pemindaian QR Code	32
Gambar 3.8 Activity Diagram Penambahan Data Ternak	33
Gambar 3.9 Activity Diagram Pemutakhiran Riwayat Ternak	33
Gambar 4.1 Penggalan kode pemetaan field opsional pembeda operasi tambah dan pemutakhiran ternak	40
Gambar 4.2 Tampilan halaman LoginPage	42
Gambar 4.3 Tampilan halaman HomePage berdasarkan role pengguna	44
Gambar 4.4 Tampilan halaman ProfilePage dengan tombol Logout	45
Gambar 4.5 Penggalan kode persistensi sesi login menggunakan FutureBuilder	46
Gambar 4.6 Alur navigasi menu Data Peternak	48
Gambar 4.7 Tampilan header profil peternak dengan tombol LOKASI, KONTAK, dan PROFILE	49
Gambar 4.8 Penggalan kode pemanggilan paralel endpoint master peternak	51
Gambar 4.9 Tampilan hasil pencarian pada search bar menu Data Peternak	52
Gambar 4.10 Penggalan kode pencarian lokal data peternak	53
Gambar 4.11 Tampilan halaman TambahTernakPage	54
Gambar 4.12 Tampilan cascading dropdown jenis hewan dan kategori ras	55
Gambar 4.13 Perbandingan tampilan form tambah ternak berdasarkan jenis kelamin: (a) Jantan tanpa section reproduksi, (b) Betina dengan section reproduksi	56
Gambar 4.14 Alur navigasi menu Data Ternak	57
Gambar 4.15 Tampilan tab PROFILE pada halaman RiwayatTernakPage	58
Gambar 4.16 Tampilan tab RIWAYAT pada halaman RiwayatTernakPage: (a) seluruh entri terkuncup, (b) salah satu entri mekar dengan snapshot lengkap	59
Gambar 4.17 Tampilan tab PERBARUI pada halaman RiwayatTernakPage	61
Gambar 4.18 Alur penggunaan fitur pemindai QR code	63
Gambar 4.19 Penggalan kode widget MobileScanner pada halaman pemindai QR code	65

BAB I PENDAHULUAN

1.1 Latar Belakang

Sektor peternakan menjadi salah satu pilar ketahanan pangan nasional. Skala kegiatannya tidak kecil. Badan Pusat Statistik (2026) mencatat populasi sapi potong Indonesia pada tahun 2025 sebanyak 12,12 juta ekor, dengan produksi daging sapi dan kerbau 515,60 ribu ton atau naik 5,61% dibanding tahun sebelumnya. Angka ini sekaligus menandai pemulihan, setelah pada tahun 2023 populasi sapi potong sempat anjlok ke 10,83 juta ekor akibat wabah Penyakit Mulut dan Kuku. Pemulihan populasi maupun upaya mencegah wabah serupa sama-sama membutuhkan pencatatan data operasional yang rinci dan terstruktur. Data yang diperlukan mencakup identitas ternak, riwayat kesehatan, manajemen pakan, sampai catatan performa berkala. Bagi peternak, data tersebut menjadi dasar pengambilan keputusan harian. Bagi penyuluh, data tersebut menjadi bahan *monitoring* dan evaluasi (Neethirajan, 2023).

Pada tingkat operasional, persoalan mendasar justru muncul dari hal yang paling sederhana, yaitu bagaimana membedakan satu ekor ternak dari yang lain dan ke mana data hewan itu dikumpulkan. Sampai saat ini, identifikasi ternak masih banyak mengandalkan *ear tag* visual di telinga hewan dan catatan kertas yang dipegang penyuluh saat berkunjung ke lokasi (Fariani dkk., 2025). Cara seperti ini cukup ketika populasi masih kecil, tetapi mulai goyah ketika jumlah ternak binaan bertambah dan rotasi hewan cukup tinggi. Akar masalahnya bukan pada pengamatan visualnya, melainkan pada absennya tempat penyimpanan data yang terstruktur dan dapat dibuka kembali. Akibatnya, pemantauan riwayat pertumbuhan, catatan kesehatan, sampai histori reproduksi sulit dilakukan secara konsisten karena setiap kunjungan melahirkan catatan terpisah yang tidak terhubung dengan catatan sebelumnya.

Pencatatan manual di atas kertas juga rentan terhadap kesalahan manusia. Data yang dihasilkan seringkali tidak lengkap atau tidak konsisten, sehingga rekapitulasi ke level desa, kecamatan, atau binaan penyuluh menjadi pekerjaan yang menyulitkan (Resti dkk., 2024). Dokumen kertas juga rentan hilang, rusak, atau tertinggal di lokasi yang berbeda dari tempat data baru perlu dicatat. Ketika data lama dibutuhkan untuk keperluan analisis, misalnya untuk mengevaluasi performa reproduksi atau memantau pola kesehatan kawanan, proses rekonstruksinya memakan waktu cukup lama dan tetap berisiko menghasilkan angka yang kurang akurat.

Kondisi tersebut menjadi titik tolak kolaborasi lintas institusi antara Universitas Islam Indonesia, Universitas Sriwijaya, dan tim IT eksternal yang mengembangkan platform pencatatan ternak bernama Eco-system Livestock. Pembagian peran dalam kolaborasi ini cukup jelas. Pakar peternakan dari Universitas Sriwijaya menetapkan variabel domain dan kebutuhan data seperti siklus birahi, bobot, dan indikator kesehatan, sekaligus menyediakan rancangan antarmuka visual aplikasi. Tim IT eksternal mengerjakan sisi *backend* beserta *REST API* yang akan diakses seluruh klien. Salah satu hasil awal kolaborasi ini adalah prototipe sistem *ear tag* multi-teknologi yang mengintegrasikan RFID, NFC, dan QR *code* untuk identifikasi ternak, yang menjadi konteks domain bagi penelitian ini (Fariani dkk., 2025). Dari ruang lingkup yang luas tersebut, porsi yang dikerjakan pada penelitian ini adalah pengembangan aplikasi *mobile* Android sebagai klien dari platform Eco-system Livestock, salah satu komponen dalam ekosistem yang sedang dibangun secara kolaboratif.

Pemilihan platform *mobile* didasarkan pada karakter pekerjaan penyuluh dan peternak yang bersifat lapangan. Aktivitas utama mereka berlangsung di kandang, lahan pakan, atau Rumah Potong Hewan, bukan di depan komputer. Memindahkan proses pencatatan dari kertas ke *browser desktop* belum tentu menyelesaikan akar persoalan, sebab kebutuhan *input* data justru terjadi di titik aktivitas, bukan di meja kerja. Perangkat *mobile* lebih menyatu dengan aktivitas profesional petugas lapangan sehari-hari dibanding komputer atau laptop (Masyhur, 2016). Thannithi dkk. (2024) menguatkan dari sisi yang berbeda, melaporkan bahwa aplikasi *mobile* pencatatan ternak dapat meningkatkan efisiensi operasional sekaligus memberikan pengalaman pengguna yang cukup baik bagi peternak skala kecil.

Selain sifatnya yang portabel, perangkat *mobile* juga membawa satu keunggulan yang jarang dimanfaatkan sistem *web*, yaitu kamera bawaan. Proses identifikasi ternak secara manual, misalnya dengan membaca nomor *ear tag* lalu mengetiknya ulang, cenderung memakan waktu dan rentan salah ketik. Pemindai optik berbasis QR *code* melalui kamera perangkat sudah banyak diadopsi pada sistem manajemen ternak untuk menghilangkan proses pengetikan ulang tersebut (Fabiyyi dkk., 2022). Pendekatan serupa pernah diterapkan di Indonesia untuk pencatatan ternak berbasis QR *code*, walaupun masih dibangun di atas platform *web* (Fuady dkk., 2023). Dibanding proses *input* manual, pemindai optik mempercepat identifikasi ternak sekaligus menekan potensi *human error* pada tahap paling awal dari alur pencatatan.

Dari sudut pandang rekayasa perangkat lunak, pilihan aplikasi *mobile native* juga lebih unggul dibanding pendekatan berbasis *web* yang dibungkus menjadi aplikasi (*web wrapper*

atau *progressive web app*). Aplikasi Flutter dikompilasi menjadi kode *native* ARM (*Advanced RISC Machine*) melalui *ahead-of-time* (AOT) *compiler* Dart, sehingga berjalan langsung di atas mesin perangkat tanpa perantara mesin peramban (*browser engine*) seperti pada aplikasi *web*. Hal ini memberikan waktu render antarmuka yang lebih ringkas dan konsumsi sumber daya yang lebih terkendali, terutama pada perangkat Android kelas menengah ke bawah yang umum dipakai di lapangan. Selain itu, aplikasi *native* memperoleh akses langsung dan stabil ke perangkat keras melalui *platform channel*, antara lain kamera untuk pemindaian QR *code* secara *real-time* dan keterpaduan dengan aplikasi lain melalui *intent* sistem operasi seperti membuka peta atau *WhatsApp*. Akses semacam ini pada aplikasi *web* bergantung pada izin dan kapabilitas peramban yang tidak selalu konsisten antarperangkat. Dengan demikian, pemilihan platform *mobile native* bukan sekadar memindahkan antarmuka *web* ke layar ponsel, melainkan keputusan teknis untuk memperoleh performa, keandalan akses sensor, dan keterpaduan dengan sistem operasi yang lebih baik bagi konteks penggunaan di lapangan.

Aplikasi Eco-system Livestock pada penelitian ini diposisikan sebagai klien *mobile* dari platform pencatatan ternak yang sudah dibangun bersama dalam kolaborasi. Aplikasi ini tidak membangun ulang basis data atau *backend*-nya sendiri, melainkan memanggil *REST API* yang sudah disediakan tim *backend*. Data yang di-*input* dari sisi *mobile* langsung tersimpan di pusat dan dapat dibaca dari antarmuka *web* yang sudah ada, sehingga data peternak dan ternak terkumpul di satu tempat. Aplikasi ini melayani dua peran pengguna dengan hak akses yang berbeda, yaitu penyuluh produksi sebagai supervisor yang memantau peternak binaan pada lingkup desa, dan peternak yang melalui aplikasi hanya menampilkan data hewan miliknya sendiri. Validasi hak akses ditegakkan di sisi API, sehingga aplikasi *mobile* berperan sebagai klien yang menghormati aturan otorisasi *server*.

Penelitian ini tidak mengambil bentuk studi kasus pada peternakan tertentu. Fokus penelitiannya berada pada platform Eco-system Livestock itu sendiri, yaitu hasil kolaborasi UII, UNSRI, dan tim IT eksternal yang sudah menjalankan uji prototipe *ear tag* di lahan UNSRI pada 50 sapi Bali (Fariani dkk., 2025). Prototipe tersebut sudah berhasil pada sisi perangkat keras dan platform penyimpanan data, tetapi belum dilengkapi klien *mobile* yang dapat dipakai langsung oleh penyuluh dan peternak di lokasi kerja. Skripsi ini mengisi celah itu, dengan cara mengembangkan aplikasi *mobile* Android sebagai klien dari *REST API* yang sudah disediakan tim *backend*.

Pengembangan aplikasi difokuskan pada sisi *mobile client*. Ruang lingkupnya mencakup perancangan arsitektur aplikasi berbasis Flutter dengan pola *Model-View-ViewModel*

(MVVM) dan *state management Provider*, implementasi integrasi *REST API*, pemanfaatan kamera sebagai pemindai optik *QR code*, serta pengujian aplikasi melalui *black-box testing* dan *API integration testing*. Perancangan basis data, pengembangan *backend*, penetapan variabel domain peternakan, dan perancangan antarmuka visual tidak masuk ke dalam ruang lingkup karena menjadi tanggung jawab tim lain dalam proyek kolaborasi ini.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, rumusan masalah pada penelitian ini adalah sebagai berikut:

- a. Bagaimana mengembangkan aplikasi *mobile* Android Eco-system Livestock untuk manajemen data ternak yang memanfaatkan *REST API* platform sebagai sumber datanya?
- b. Bagaimana memverifikasi bahwa aplikasi yang dikembangkan telah memenuhi seluruh kebutuhan fungsional manajemen data ternak bagi peran penyuluh produksi dan peternak?

1.3 Batasan Masalah

Pencatatan ternak merupakan persoalan yang luas. Agar penelitian tetap terarah, tidak seluruh aspek pencatatan ternak dikaji dalam penelitian ini. Batasan masalah pada penelitian ini adalah sebagai berikut.

- a. Masalah yang ditangani dibatasi pada pendataan ternak ruminansia, yaitu sapi, kerbau, kambing, dan domba. Jenis ternak lain seperti unggas dan ternak non-ruminansia tidak termasuk.
- b. Aspek data yang dikelola dibatasi pada pencatatan identitas ternak, data produktivitas, kondisi kesehatan, dan riwayat reproduksi sebagai catatan berkala. Pencatatan keuangan dan biaya usaha ternak, manajemen stok serta penjadwalan pakan, dan penjadwalan kegiatan seperti vaksinasi maupun perkawinan tidak termasuk dalam masalah yang digarap.
- c. Pemanfaatan data dibatasi pada pengumpulan dan penyimpanan secara terpusat agar catatan ternak tersimpan rapi dan dapat dibuka kembali. Pengolahan data lanjutan, seperti analisis populasi, peringatan dini kesehatan, prediksi performa, dan penyusunan laporan ringkasan untuk pengambilan keputusan, tidak termasuk dalam masalah yang digarap.
- d. Pihak yang dilayani dibatasi pada dua peran di lapangan, yaitu penyuluh produksi yang mendata ternak milik peternak binaannya, dan peternak yang melihat data ternak

miliknya sendiri. Kebutuhan pihak lain seperti dinas peternakan, pengelola rumah potong hewan, dan pelaku rantai pasok tidak termasuk.

1.4 Tujuan Penelitian

Tujuan yang hendak dicapai melalui penelitian ini adalah sebagai berikut:

- a. Mengembangkan aplikasi *mobile* Android Eco-system Livestock untuk manajemen data ternak yang memanfaatkan *REST API* platform sebagai sumber datanya.
- b. Memverifikasi bahwa aplikasi yang dikembangkan telah memenuhi seluruh kebutuhan fungsional manajemen data ternak bagi peran penyuluh produksi dan peternak melalui *black-box testing* dan *API integration testing*.

1.5 Manfaat Penelitian

Penelitian ini diharapkan memberikan manfaat bagi beberapa pihak yang terkait dengan pengembangan aplikasi Eco-system Livestock. Manfaat tersebut ditinjau dari empat sudut, yaitu bidang akademik, pengguna aplikasi, proyek kolaborasi lintas institusi, dan penulis.

Bagi pengguna aplikasi, yaitu penyuluh produksi dan peternak yang menjadi sasaran sistem Eco-system Livestock, aplikasi ini menyediakan sarana akses berbasis *mobile* untuk mengelola dan mengakses data ternak. Petugas dapat langsung menggunakan aplikasi di lokasi kerja tanpa harus kembali ke komputer, dan pemindai berbasis *QR code* mempercepat identifikasi hewan.

Bagi proyek kolaborasi lintas institusi antara Universitas Islam Indonesia, Universitas Sriwijaya, dan tim IT eksternal, penelitian ini menjadi komponen klien *mobile* dari platform pencatatan ternak yang dikembangkan secara kolaboratif, sehingga titik akses pengguna menjadi lebih luas dibandingkan ketika hanya tersedia antarmuka *web*.

Bagi penulis, penelitian ini menjadi sarana penerapan ilmu rekayasa perangkat lunak pada kasus nyata pengembangan aplikasi *mobile* dengan integrasi *REST API*, yang mencakup perancangan arsitektur aplikasi, pemanfaatan sensor perangkat keras, serta pengujian perangkat lunak secara sistematis.

1.6 Sistematika Penulisan

Laporan penelitian ini disusun dalam lima bab. Bab I Pendahuluan memuat latar belakang penelitian, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian,

dan sistematika penulisan laporan. Bab II Kajian Pustaka dan Landasan Teori berisi telaah terhadap penelitian terdahulu yang relevan dan landasan teori yang mendasari penelitian. Cakupan landasan teori meliputi inisiasi pencatatan ternak dalam kerangka proyek kolaborasi, metodologi *Waterfall*, pengembangan aplikasi *mobile* Android dengan *framework* Flutter dan bahasa pemrograman Dart, IDE *Android Studio*, arsitektur MVVM dengan *state management Provider*, komunikasi *client-server* melalui *REST API*, pemindai optik *QR code* pada perangkat *mobile*, serta metode *black-box testing* dan *API integration testing*. Bab III Metodologi Penelitian menjelaskan metodologi pengembangan perangkat lunak yang digunakan, alur penelitian, analisis kebutuhan fungsional dan non-fungsional, perancangan sistem pada sisi *mobile client*, serta rancangan metode pengujian. Bab IV Implementasi dan Pengujian berisi implementasi aplikasi beserta tangkapan layar antarmuka, implementasi fitur pemindai *QR code*, implementasi integrasi *REST API*, hasil *black-box testing*, *API integration testing*, dan pengujian *usability* (SUS), serta pembahasan atas hasil yang diperoleh. Bab V Kesimpulan dan Saran berisi kesimpulan atas hasil penelitian dan saran untuk pengembangan selanjutnya, termasuk kemungkinan implementasi pemindai berbasis NFC dan RFID untuk melengkapi ekosistem identifikasi ternak multi-teknologi.

BAB II

KAJIAN PUSTAKA DAN LANDASAN TEORI

2.1 Penelitian Terdahulu

Sub-bab ini menelaah enam penelitian terdahulu yang relevan dengan pengembangan aplikasi *mobile* manajemen ternak dan yang menopang pemilihan teknologi pada penelitian ini. Telaah dilakukan dari dua sudut pembandingan: sudut domain berupa aplikasi pencatatan ternak, dan sudut teknologi berupa arsitektur *mobile* dan pemindai optik. Ringkasan keenam penelitian disajikan pada Tabel 2.1, kemudian disintesis pada tiga paragraf setelahnya untuk memperjelas kluster pendekatan dan posisi penelitian ini dalam peta literatur yang ada.

Tabel 2.1 Perbandingan Penelitian Terdahulu

Peneliti (Tahun)	Fokus Penelitian	Teknologi	Platform	Temuan/Metode Kunci	Posisi terhadap Penelitian Ini
Aisuwarya dkk. (2020)	Implementasi teknologi NFC untuk perekaman data sapi perah Kelompok Tani Lembu Alam Serambi, Padang Panjang.	NFC <i>tag</i> pasif + Firebase Realtime Database.	Aplikasi Android (MIT App Inventor) + <i>cloud database</i> .	Dua aplikasi dibangun: perekaman data dan tap-tag. Data ternak dapat dibaca ulang dengan menempelkan ponsel ke <i>ear tag</i> NFC di telinga hewan.	Pembandingan teknologi pemindai. NFC memiliki jarak baca sangat pendek (<5 cm), sedangkan penelitian ini memakai QR <i>code</i> melalui kamera.
Prayoga dkk. (2021)	Analisis performa tiga pendekatan <i>state management</i> pada Flutter: <i>setState</i> ,	Flutter, Dart, <i>library provider</i> , <i>library flutter_bloc</i> , Android	Aplikasi <i>benchmark</i> Flutter pada Android.	<i>Provider</i> : CPU 0,54%, memori 32,01 MB, waktu eksekusi 4,10 detik. Kedua <i>library</i> lebih efisien dibanding	Pembandingan arsitektur. Mendukung pemilihan <i>Provider</i> sebagai lapisan <i>ViewModel</i> dalam pola MVVM pada

Peneliti (Tahun)	Fokus Penelitian	Teknologi	Platform	Temuan/Metode Kunci	Posisi terhadap Penelitian Ini
	BLoC, dan <i>Provider</i> .	DevTools Profiler.		<i>setState</i> mentah pada <i>leaf widget</i> .	penelitian ini dengan dukungan data empiris.
Fabiyi dkk. (2022)	Pengembangan <i>mobile platform</i> untuk pemantauan dan inspeksi ternak melalui paspor digital.	QR code + RFID + Xamarin + <i>web service backend</i> .	Aplikasi <i>mobile cross-platform</i> (Xamarin).	Platform mendukung pencatatan riwayat hewan secara portabel dan terintegrasi dengan <i>backend</i> terpusat untuk inspeksi lapangan.	Pembandingan pola terdekat (aplikasi <i>mobile</i> + pemindai + <i>REST API</i>). Perbedaan pada <i>framework</i> (Xamarin vs. Flutter) dan tidak ada diferensiasi peran pengguna pada Fabiyi dkk.
Asmara dkk. (2022)	Pengenalan QR code berbasis <i>framework</i> Flutter untuk sistem keamanan pintu masuk di Politeknik Negeri Malang.	Flutter + kamera <i>smartphone</i> + Google Cloud Platform + NoSQL.	Aplikasi <i>mobile</i> Flutter pada Android.	Kecerahan layar QR optimal pada rentang 25–50% dan jarak pemindaian optimal 10–30 cm. Tujuh dari sepuluh responden menilai sistem mengurangi kecurangan kehadiran.	Pembandingan langsung fitur pemindai. Sama-sama Flutter dan kamera <i>smartphone</i> , domain berbeda (kehadiran, bukan identifikasi ternak).
Fuady dkk. (2023)	Perancangan sistem informasi pencatatan dan	QR code + aplikasi <i>web</i> + basis data <i>server</i> .	Aplikasi <i>web</i> (bukan <i>mobile</i>).	Sistem menyediakan pencatatan ternak berbasis QR yang	Pembandingan platform. Penelitian ini melanjutkan

Peneliti (Tahun)	Fokus Penelitian	Teknologi	Platform	Temuan/Metode Kunci	Posisi terhadap Penelitian Ini
	pengawasan hewan ternak berbasis QR code dengan metode <i>Agile</i> .			diakses lintas perangkat melalui <i>browser</i> .	pendekatan QR ke <i>platform mobile native</i> untuk mengatasi keterbatasan mobilitas di kandang atau lahan pakan.
Thannithi dkk. (2024)	Perancangan dan evaluasi aplikasi <i>mobile</i> manajemen peternakan sapi potong penggemukan di Thailand.	Aplikasi <i>mobile native</i> + <i>input</i> manual (tanpa pemindai).	Aplikasi <i>mobile</i> + basis data peternakan.	Skor SUS 75,17 (kategori <i>Good</i>) dan kepuasan pengguna 4,26/5. Aplikasi <i>mobile</i> meningkatkan efisiensi operasional peternak skala kecil.	Pembandingan efisiensi <i>mobile</i> tanpa pemindai. Penelitian ini menambahkan pemindai optikal dan diferensiasi dua peran dalam satu aplikasi.

Kluster pertama terlihat dari sudut domain pencatatan ternak. Fabiyi dkk. (2022), Fuady dkk. (2023), dan Thannithi dkk. (2024) sama-sama berangkat dari kebutuhan pencatatan riwayat ternak yang sulit dikelola secara manual, tetapi menempuh jalur teknologis yang berbeda. Fabiyi dkk. memilih aplikasi *mobile cross-platform* berbasis Xamarin dengan kombinasi pemindai QR dan RFID, Fuady dkk. memilih pendekatan *web* dengan pemindai QR, sementara Thannithi dkk. membangun aplikasi *mobile* tanpa pemindai dan melaporkan skor *System Usability Scale* (SUS) sebesar 75,17 sebagai bukti bahwa platform *mobile* sudah dapat diterima pengguna peternak. Aisuwarya dkk. (2020) memperkaya peta ini dengan alternatif teknologi pemindai berupa NFC untuk sapi perah di Padang Panjang. Gabungan keempat penelitian menunjukkan bahwa kombinasi antara aplikasi *mobile* dan pemindai identifikasi otomatis sudah dieksplorasi di berbagai konteks, namun belum ada yang secara bersamaan

mengonsolidasikan dua peran pengguna, yaitu supervisor lapangan dan peternak, dalam satu aplikasi klien yang terhubung ke platform terpusat.

Kluster kedua terlihat dari sudut arsitektur dan teknologi pemindai. Prayoga dkk. (2021) menunjukkan secara kuantitatif bahwa *Provider* merupakan *library state management* yang efisien pada Flutter dengan penggunaan CPU dan memori yang terjaga. Temuan ini menjadi dasar empiris pemilihan *Provider* sebagai lapisan *ViewModel* dalam pola MVVM pada penelitian ini. Asmara dkk. (2022) pada sisi lain mengonfirmasi bahwa kamera *smartphone* dapat berfungsi sebagai pemindai QR yang andal ketika aplikasi dibangun dengan Flutter, lengkap dengan rentang jarak dan kecerahan optimal yang dapat dijadikan *baseline* pengujian. Kedua penelitian ini tidak berhubungan langsung dengan domain ternak, tetapi keduanya mengukuhkan kelayakan *stack* teknis yang dipakai oleh penelitian ini, yaitu Flutter dengan *Provider* dan pemindai optik berbasis kamera.

Dari telaah di atas, penelitian ini menempatkan diri pada titik temu kedua kluster tersebut. Secara domain, penelitian ini melanjutkan arah Fabiyi dkk. (2022) dan Thannithi dkk. (2024) yang memilih platform *mobile*, tetapi menambahkan dua pembeda. Pertama, diferensiasi hak akses untuk peran penyuluh produksi dan peternak yang validasinya ditegakkan di sisi *REST API*. Kedua, integrasi ke platform pencatatan ternak yang sudah dibangun secara kolaboratif sehingga aplikasi *mobile* tidak membangun ulang basis data sendiri. Secara teknologi, penelitian ini mengadopsi kombinasi Flutter dan *Provider* (Prayoga dkk., 2021) dengan pemindai QR *code* berbasis kamera (Asmara dkk., 2022; Fabiyi dkk., 2022). Kombinasi spesifik ini, yaitu *mobile Android native client* berbasis Flutter dengan MVVM dan *Provider*, pemindai optik berbasis QR *code*, serta otorisasi dua peran yang ditegakkan *server*, belum ditemukan pada keenam penelitian yang ditelaah sehingga menjadi celah yang dijawab oleh penelitian ini.

2.2 Landasan Teori

2.2.1 Inisiasi Pencatatan Ternak

Penelitian ini berangkat dari kolaborasi lintas institusi antara Universitas Islam Indonesia, Universitas Sriwijaya, dan tim IT eksternal yang mengembangkan platform pencatatan ternak bernama Eco-system Livestock. Kolaborasi tersebut menghasilkan laporan penelitian berupa pengembangan dan pengujian prototipe sistem *ear tag* untuk pemantauan ternak di Indonesia (Fariani dkk., 2025). Laporan ini menjadi acuan utama pada sub-bab ini

karena menjadi titik awal munculnya kebutuhan aplikasi *mobile* pada penelitian ini. Pada laporan tersebut dipaparkan bahwa pencatatan ternak di lapangan masih banyak mengandalkan *ear tag* visual dan catatan manual yang rentan hilang serta sulit diintegrasikan dengan sistem pengelolaan data secara digital.

Pihak kolaborator merancang prototipe *ear tag* dengan beberapa teknologi pemindai identifikasi digital, lalu mengujinya pada 50 ekor sapi Bali di lahan penelitian Universitas Sriwijaya. Pengujian lapangan dilakukan dengan tiga indikator utama, yaitu akurasi identifikasi individu ternak, efisiensi waktu pemindaian, dan keberhasilan sinkronisasi data ke sistem penyimpanan terpusat. Hasilnya menunjukkan akurasi identifikasi di atas 95 persen dan tingkat keberhasilan sinkronisasi data lebih dari 96 persen (Fariani dkk., 2025). Dari sisi pengguna, sekitar 85 persen peternak yang mengikuti pelatihan singkat mampu mengoperasikan sistem dengan benar, yang menunjukkan bahwa pendekatan berbasis perangkat digital cukup bisa diterima di tingkat peternak skala kecil.

Meski hasil uji tersebut positif, laporan Fariani dkk. (2025) mencatat sejumlah keterbatasan yang perlu ditindaklanjuti. Salah satunya adalah belum tersedianya aplikasi *mobile* yang dapat dipakai langsung oleh peternak dan penyuluh di lokasi. Prototipe yang diuji pada laporan tersebut baru mencakup sisi *ear tag* fisik dan platform penyimpanan data, sedangkan antarmuka untuk pengguna akhir masih berbasis *web* dan dinilai kurang praktis ketika digunakan langsung di kandang atau lahan pakan. Berdasarkan rekomendasi laporan tersebut, pengembangan klien *mobile* menjadi salah satu arah tindak lanjut yang penting. Penelitian ini merupakan bagian dari tindak lanjut tersebut, dengan fokus pada pengembangan aplikasi *mobile* Android yang memanfaatkan *REST API* dari platform Eco-system Livestock yang sudah dibangun sebelumnya.

2.2.2 Flutter

Flutter adalah *framework* pengembangan aplikasi *cross-platform* yang dikembangkan oleh Google (Flutter, n.d.). Dengan Flutter, pengembang dapat membangun aplikasi untuk Android, iOS, *web*, dan *desktop* dari satu basis kode yang sama. Sebelumnya, pengembangan aplikasi *native* untuk Android menggunakan Kotlin atau Java, sedangkan untuk iOS menggunakan Swift, sehingga pengembang perlu menulis kode secara terpisah untuk masing-masing platform.

Flutter menggunakan pendekatan berbasis *widget*, di mana seluruh elemen antarmuka aplikasi disusun dari kombinasi *widget*. Setiap komponen seperti tombol, teks, dan tata letak

merupakan *widget* yang dapat digabungkan menjadi struktur antarmuka yang lebih kompleks. Flutter juga menyediakan fitur *hot reload* yang memungkinkan pengembang melihat hasil perubahan kode secara langsung tanpa perlu melakukan *restart* aplikasi (Flutter, n.d.).

Pada penelitian ini, Flutter dipilih sebagai *framework* pengembangan karena dokumentasinya yang lengkap dan kemampuannya menghasilkan *build* Android yang sesuai dengan platform target. Flutter juga memiliki dukungan *package* yang luas melalui *pub.dev*, termasuk *package* untuk pemindai kamera dan komunikasi HTTP yang digunakan pada penelitian ini.

2.2.3 Dart

Dart adalah bahasa pemrograman yang digunakan pada *framework* Flutter. Dart dirancang sebagai bahasa yang dioptimalkan untuk pengembangan aplikasi *client* dengan tujuan menyediakan bahasa pemrograman yang produktif untuk pengembangan multi-platform, dipadukan dengan *runtime* yang fleksibel bagi *framework* aplikasi (Dart, n.d.). Dart juga menjadi fondasi dari Flutter, yaitu menyediakan bahasa dan *runtime* yang menggerakkan aplikasi Flutter. Dari sisi kompilasi, Dart menyediakan Dart VM dengan *just-in-time* (JIT) *compiler* untuk pengembangan dan *ahead-of-time* (AOT) *compiler* untuk produksi, sehingga aplikasi Flutter yang dibangun dengan Dart dapat dikompilasi menjadi kode *native* untuk platform Android dan iOS.

2.2.4 Android Studio

Android Studio adalah *Integrated Development Environment* (IDE) resmi untuk pengembangan aplikasi Android yang berbasis pada *IntelliJ IDEA* (Android Developers, n.d.). *Android Studio* menyediakan berbagai fitur yang membantu proses pengembangan aplikasi, antara lain *code editor*, sistem *build* berbasis *Gradle*, *emulator* untuk menjalankan aplikasi tanpa perangkat fisik, dan *debugger* untuk menelusuri kesalahan pada aplikasi. Meskipun *Android Studio* secara bawaan ditujukan untuk pengembangan aplikasi Android menggunakan Kotlin atau Java, *Android Studio* juga dapat digunakan untuk pengembangan aplikasi Flutter dengan memasang *plugin* Flutter dan Dart (Flutter, n.d.). Pada penelitian ini, *Android Studio* digunakan sebagai IDE utama untuk menulis kode, menjalankan aplikasi pada *emulator*, dan membangun berkas APK untuk distribusi aplikasi Android.

2.2.5 Arsitektur MVVM

Model-View-ViewModel (MVVM) adalah pola arsitektur yang memisahkan fitur aplikasi menjadi tiga bagian, yaitu *Model*, *View*, dan *ViewModel*. Pada pola ini, *View* dan *ViewModel* membentuk *UI layer* dari aplikasi, sedangkan *Repository* dan *service* merepresentasikan data aplikasi atau *model layer* dari MVVM (Flutter, n.d.). Pemisahan ini membuat kode lebih terstruktur karena setiap komponen memiliki tanggung jawab yang jelas.

Pada dokumentasi panduan arsitektur Flutter, pemisahan fungsi (*separation of concerns*) disebut sebagai prinsip paling penting yang harus diikuti saat mendesain aplikasi. *View* bertugas menampilkan antarmuka ke pengguna dan meneruskan interaksi pengguna ke *ViewModel*, sedangkan *ViewModel* berisi logika yang mengubah data aplikasi menjadi *UI state* yang siap ditampilkan oleh *View*. *View* tidak langsung berkomunikasi dengan sumber data, melainkan melalui *ViewModel* yang kemudian mengambil data dari *repository* dan *service* pada lapisan data.

Flutter secara resmi merekomendasikan MVVM sebagai pola arsitektur pada dokumentasi panduannya (Flutter, n.d.). Pada penelitian ini, pola MVVM diterapkan dengan menempatkan *widget* dan halaman sebagai *View*, kelas *provider* sebagai *ViewModel*, dan kelas *service* sebagai lapisan akses data ke *REST API*.

2.2.6 Provider

Provider adalah salah satu *library state management* untuk Flutter. Pada halaman resminya di *pub.dev*, *Provider* dideskripsikan sebagai pembungkus (*wrapper*) untuk *InheritedWidget* yang membuatnya lebih mudah digunakan dan lebih dapat dipakai ulang (*Provider*, n.d.). *Provider* menyediakan sejumlah kemudahan, antara lain penyederhanaan alokasi dan pelepasan sumber daya, dukungan *lazy-loading*, serta pengurangan *boilerplate* dibanding membuat kelas *InheritedWidget* secara manual.

Dalam praktiknya, *Provider* menyediakan beberapa varian kelas untuk kebutuhan yang berbeda. Salah satu yang umum digunakan adalah *ChangeNotifierProvider*, yaitu *provider* khusus untuk objek *ChangeNotifier* yang secara otomatis memanggil metode *dispose* ketika tidak lagi digunakan (*Provider*, n.d.). *ChangeNotifier* kemudian memanggil *notifyListeners* untuk memberi tahu *widget* bahwa *state* telah berubah dan perlu disusun ulang. Pada sisi *widget*, *Provider* menyediakan mekanisme pembacaan nilai yang dapat dipilih sesuai kebutuhan, baik untuk sekadar mengambil nilai sekali maupun untuk terus mendengarkan perubahan *state*.

Provider dipilih pada penelitian ini karena dua alasan. Pertama, *Provider* merupakan *library* yang dokumentasinya tersedia secara resmi dan ditandai sebagai *Flutter Favorite* di *pub.dev* (*Provider*, n.d.). Kedua, dari sisi performa, Prayoga dkk. (2021) melaporkan bahwa *Provider* mencatatkan penggunaan CPU sebesar 0,54% dan memori 32,01 MB pada pengujian *state management* di Flutter, sehingga tergolong efisien dibanding alternatif seperti *setState*. Pada penelitian ini, *Provider* berperan sebagai lapisan *ViewModel* dalam pola MVVM yang telah dijelaskan pada sub-bab sebelumnya.

2.2.7 REST API

Representational State Transfer (REST) adalah gaya arsitektur perangkat lunak yang digunakan untuk membangun layanan terdistribusi berbasis *web*. Pada dokumentasi MDN *Web Docs*, REST dijelaskan sebagai sekumpulan batasan desain arsitektur yang mendukung sistem terdistribusi yang efisien, andal, dan dapat diskalakan (Mozilla, n.d.). Ide dasar REST adalah sumber daya, misalnya sebuah dokumen atau data, dipindahkan antara klien dan *server* melalui interaksi yang terstandarisasi. *Web service* yang mengikuti batasan-batasan REST dan berjalan di atas protokol HTTP umumnya disebut sebagai *REST API*.

Komunikasi pada *REST API* dilakukan melalui *method* HTTP yang merepresentasikan aksi yang ingin dilakukan terhadap sumber daya. Beberapa *method* yang umum digunakan adalah GET untuk mengambil data, POST untuk membuat data baru, PUT untuk memperbarui data, dan DELETE untuk menghapus data (Mozilla, n.d.). Pertukaran data antara klien dan *server* biasanya menggunakan format *JavaScript Object Notation* (JSON), yaitu format teks yang ringan dan mudah dibaca mesin maupun manusia. Pada setiap permintaan, klien mengirim *request* ke *server*, kemudian *server* mengembalikan *response* berisi data dan kode status HTTP yang menunjukkan hasil pemrosesan.

Pada *REST API*, akses ke *endpoint* umumnya dilindungi oleh mekanisme autentikasi seperti *bearer token*, yaitu *token* yang disertakan pada *header* setiap permintaan untuk membuktikan identitas klien. Selain autentikasi, *REST API* juga dapat menerapkan otorisasi berbasis peran, yaitu pembatasan akses terhadap *endpoint* tertentu berdasarkan peran pengguna yang terautentikasi. Pada penelitian ini, *REST API* dari platform Eco-system Livestock berperan sebagai sumber data bagi aplikasi *mobile* yang dikembangkan, termasuk pengelolaan autentikasi dan otorisasi dua peran pengguna yaitu penyuluh produksi dan peternak.

2.2.8 QR Code

Quick Response Code (QR code) adalah kode dua dimensi yang menyimpan informasi dalam bentuk pola kotak hitam-putih. Informasi yang disimpan dapat berupa teks, tautan, atau data identifikasi lainnya yang dapat dibaca oleh pemindai optik. Pembacaan QR code dilakukan dengan mengarahkan kamera ke permukaan yang memuat kode tersebut, kemudian perangkat lunak pemindai akan menerjemahkan pola yang ditangkap menjadi data teks. Pemanfaatan QR code semakin luas pada berbagai aplikasi *mobile* karena sebagian besar perangkat *smartphone* saat ini sudah dilengkapi kamera dengan resolusi yang memadai untuk pemindaian (Asmara dkk., 2022).

Pada pengembangan aplikasi *mobile* berbasis Flutter, proses pemindaian QR code dapat dilakukan menggunakan *package mobile_scanner* yang tersedia di *pub.dev*. *Package* ini dideskripsikan sebagai *plugin* Flutter yang ringan dan cepat untuk memindai QR code melalui kamera perangkat, dengan dukungan pendeteksian secara *real-time* (Steenbakker, n.d.). *Package mobile_scanner* mendukung platform Android dan iOS serta menyediakan parameter konfigurasi untuk menyesuaikan perilaku pemindai. Pada penelitian ini, *package mobile_scanner* digunakan untuk memindai QR code yang tertera pada *ear tag* ternak, kemudian hasil pemindaian berupa ID ternak dikirim ke *REST API* untuk menampilkan detail ternak yang bersangkutan.

2.2.9 Pengujian Perangkat Lunak

Black-box testing adalah metode pengujian perangkat lunak yang memeriksa fungsionalitas aplikasi tanpa melihat struktur atau kode program di dalamnya. Pada pengujian ini, penguji memperlakukan aplikasi seperti kotak hitam yang hanya diamati dari sisi masukan dan keluaran, sehingga fokus pengujian berada pada kesesuaian perilaku aplikasi dengan kebutuhan fungsional yang telah ditetapkan (BrowserStack, n.d.). Pada penelitian ini, *black-box testing* dilakukan dengan menyusun *test case* berdasarkan skenario penggunaan fitur aplikasi, yang meliputi pengujian dengan masukan valid maupun tidak valid untuk masing-masing fitur yang dikembangkan.

API Integration Testing adalah pengujian yang memastikan bahwa aplikasi klien dan *REST API* yang menjadi sumber datanya dapat berkomunikasi dengan benar. Pengujian ini berfokus pada kesesuaian *request* yang dikirim aplikasi dengan kontrak *endpoint*, serta kemampuan aplikasi dalam memproses *response* yang dikembalikan *server*, termasuk ketika *response* yang diterima berupa pesan kesalahan (Postman, n.d.). Salah satu alat bantu yang

umum digunakan untuk pengujian ini adalah Postman, yaitu aplikasi yang memungkinkan pengembang mengirim *request* ke *endpoint* dan memverifikasi *response* yang diterima. Postman juga mendukung penyusunan *test case* dalam bentuk *collection* sehingga sejumlah skenario pengujian dapat dijalankan secara berurutan.

Selain pengujian fungsional, kualitas perangkat lunak juga dinilai dari sisi keterterimaan pengguna melalui pengujian *usability*. Salah satu instrumen baku yang umum digunakan adalah *System Usability Scale* (SUS), yaitu kuesioner ringkas yang terdiri atas sepuluh pernyataan dengan nada positif dan negatif berselang-seling, masing-masing dinilai memakai skala Likert lima tingkat (Brooke, 2013). SUS dapat diterapkan pada beragam jenis produk tanpa bergantung pada teknologi tertentu, sehingga sesuai untuk menilai aplikasi *mobile*. Skor SUS dihitung dengan mengonversi jawaban tiap pernyataan menjadi kontribusi bernilai nol sampai empat, menjumlahkannya, lalu mengalikannya dengan 2,5 sehingga diperoleh skor akhir pada rentang 0 sampai 100. Untuk menafsirkan skor tersebut, Bangor dkk. (2009) menyediakan kerangka interpretasi berupa rentang keterterimaan (*acceptability*) dan label kata sifat, dengan skor di atas 68 tergolong di atas rata-rata dan dapat diterima.

Pada penelitian ini, ketiga metode pengujian di atas dijalankan dengan peran yang saling melengkapi. *Black-box testing* digunakan untuk memverifikasi perilaku aplikasi dari sisi antarmuka pengguna, misalnya alur navigasi antar halaman, validasi *form*, dan respons aplikasi terhadap interaksi pengguna. *API integration testing* digunakan untuk memverifikasi komunikasi aplikasi dengan *REST API*, termasuk skenario pengujian *endpoint* dengan kredensial yang valid dan tidak valid serta pengujian otorisasi berbasis peran pengguna. Adapun pengujian *usability* dengan SUS digunakan untuk mengukur tingkat keterterimaan aplikasi dari sudut pandang pengguna. Dengan demikian, *black-box testing* dan *API integration testing* menjangkau aspek fungsional pada dua sisi yang berbeda, sedangkan SUS melengkapinya dari sudut pandang pengalaman pengguna.

BAB III

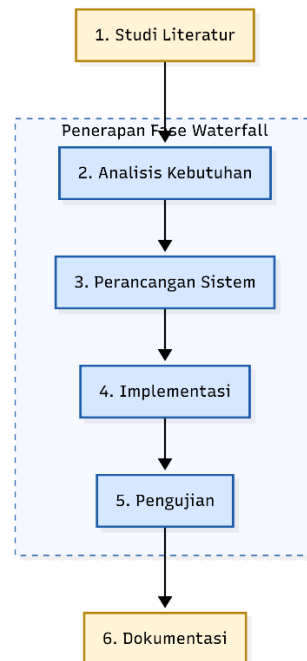
METODOLOGI PENELITIAN

Bab ini menjelaskan metodologi yang digunakan pada pengembangan aplikasi *mobile* Android Eco-system Livestock beserta alur kerja penelitian yang ditempuh. Pembahasan dimulai dari metodologi dan alur penelitian, kemudian dilanjutkan dengan analisis kebutuhan aplikasi, perancangan sistem pada sisi klien *mobile*, dan diakhiri dengan metode pengujian yang digunakan.

3.1 Metodologi dan Alur Penelitian

Pengembangan aplikasi Eco-system Livestock pada penelitian ini menggunakan metodologi Waterfall sebagai model pengembangan perangkat lunak. Waterfall adalah pendekatan linear yang membagi proses pengembangan perangkat lunak menjadi fase-fase berurutan. Tiap fase diselesaikan sebelum berlanjut ke fase berikutnya (Kavlakoglu, n.d.). Model ini terdiri dari lima fase, yaitu analisis kebutuhan, desain, implementasi, pengujian, dan pemeliharaan. Waterfall dipilih pada penelitian ini karena seluruh kebutuhan aplikasi sudah tersedia di awal pengerjaan. Kontrak *REST API* sudah disediakan tim *backend*, *mockup* antarmuka sudah dirancang tim kolaborator dari Universitas Sriwijaya, dan variabel data ternak sudah didefinisikan oleh pakar peternakan. Kondisi tersebut membuat alur pengerjaan dapat berjalan secara linear tanpa perlu iterasi berulang seperti pada metodologi Agile.

Alur penelitian yang ditempuh terdiri dari enam tahap, mulai dari studi literatur hingga dokumentasi. Empat tahap di tengahnya, yaitu analisis kebutuhan, perancangan sistem, implementasi, dan pengujian, merupakan penerapan fase-fase Waterfall pada pengembangan aplikasi. Dua tahap lainnya, yaitu studi literatur di awal dan dokumentasi di akhir, merupakan aktivitas akademik yang khas pada penelitian skripsi. Diagram alur penelitian tersebut ditampilkan pada Gambar 3.1.



Gambar 3.1 Diagram Alur Penelitian

Penjabaran tiap tahap pada alur penelitian dijelaskan sebagai berikut.

Studi Literatur. Tahap ini dilakukan untuk memperoleh landasan teori dan referensi penelitian terdahulu yang relevan dengan topik penelitian. Kegiatan yang dilakukan meliputi penelusuran jurnal ilmiah, dokumentasi resmi framework dan library yang digunakan, serta laporan penelitian terkait pengembangan sistem pencatatan ternak. Hasil dari tahap ini menjadi dasar penyusunan Bab II Kajian Pustaka dan Landasan Teori.

Analisis Kebutuhan. Tahap ini merupakan fase pertama Waterfall dan bertujuan untuk mengidentifikasi kebutuhan aplikasi yang akan dikembangkan. Kegiatan yang dilakukan meliputi diskusi dengan tim kolaborator dari Universitas Sriwijaya terkait variabel data ternak, penelaahan kontrak *REST API* yang disediakan tim backend, dan penelaahan mockup antarmuka yang disediakan tim perancang. Hasil tahap ini berupa daftar kebutuhan fungsional dan non-fungsional aplikasi yang dijabarkan pada Sub-bab 3.2.

Perancangan Sistem. Tahap ini merupakan fase desain pada Waterfall. Perancangan pada penelitian ini difokuskan pada arsitektur sisi klien mobile, meliputi penerapan pola Model-View-ViewModel (MVVM) dengan state management *Provider*, penyusunan struktur folder proyek Flutter, dan pemetaan kelas model Dart dari response *REST API*. Perancangan juga mencakup penyusunan diagram UML berupa use case diagram dan activity diagram untuk memodelkan perilaku aplikasi. Perancangan antarmuka visual tidak termasuk pada tahap ini

karena mockup disediakan oleh tim kolaborator dari Universitas Sriwijaya sebagaimana sudah dijelaskan pada Bab I batasan masalah. Hasil tahap ini dijabarkan pada Sub-bab 3.3.

Implementasi. Tahap ini merupakan fase ketiga Waterfall, yaitu penulisan kode aplikasi berdasarkan hasil perancangan. Aplikasi dibangun menggunakan framework Flutter dengan bahasa pemrograman Dart pada IDE Android Studio. Implementasi mencakup pembuatan kelas model, pengembangan *provider* sebagai ViewModel, penyusunan service untuk akses *REST API*, dan pembangunan halaman beserta widget antarmuka sesuai mockup. Hasil tahap implementasi beserta tangkapan layar aplikasi dibahas pada Bab IV.

Pengujian. Tahap ini merupakan fase keempat Waterfall. Pengujian dilakukan dengan tiga metode yang saling melengkapi, yaitu *black-box testing* untuk memverifikasi perilaku aplikasi dari sisi antarmuka pengguna, *API integration testing* untuk memverifikasi komunikasi aplikasi dengan *REST API*, dan pengujian *usability* dengan SUS untuk mengukur keterterimaan aplikasi oleh pengguna. Rancangan metode pengujian dijabarkan pada Sub-bab 3.4, sedangkan hasil pengujian dibahas pada Bab IV.

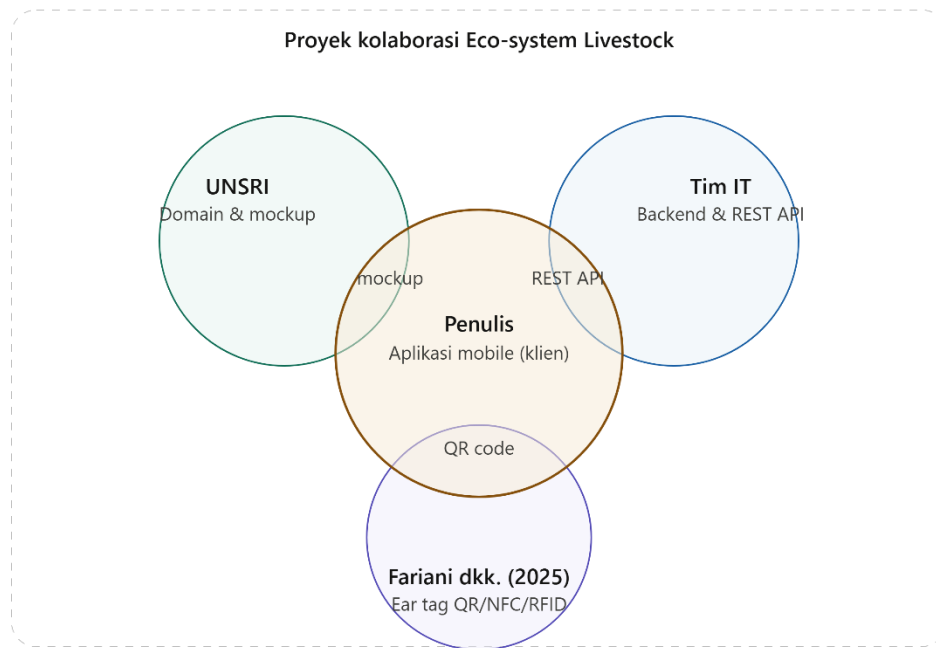
Dokumentasi. Tahap ini merupakan tahap terakhir pada alur penelitian dan mencakup penyusunan naskah skripsi secara keseluruhan. Dokumentasi disusun berdasarkan hasil yang diperoleh dari tahap-tahap sebelumnya, mulai dari studi literatur hingga pengujian. Penyusunan naskah dilakukan secara retrospektif setelah implementasi dan pengujian aplikasi selesai.

Fase kelima Waterfall, yaitu pemeliharaan, tidak termasuk ruang lingkup penelitian ini. Pemeliharaan pada umumnya mencakup perbaikan bug dan penambahan fitur setelah aplikasi dirilis kepada pengguna akhir. Pada penelitian ini, aplikasi Eco-system Livestock belum mencapai tahap deployment secara luas kepada peternak, sehingga aktivitas pasca-deployment tidak dilakukan. Arah pengembangan aplikasi setelah tahap *deployment* dibahas pada Bab V sebagai saran untuk penelitian selanjutnya.

3.2 Ruang Lingkup Kontribusi pada Proyek Kolaborasi

Aplikasi Eco-system Livestock dikembangkan dalam kerangka proyek kolaborasi lintas institusi antara Universitas Islam Indonesia, Universitas Sriwijaya, dan tim IT eksternal. Karena platform ini dikerjakan oleh banyak pihak, sub-bab ini menegaskan batas pekerjaan yang menjadi kontribusi orisinal penulis dan memisahkannya dari pekerjaan kolaborator lain. Sistem informasi peternakan mencakup ranah yang luas, mulai dari beragam platform hingga beragam fungsi pengelolaan; kontribusi penulis menempati bagian yang spesifik dan terbatas di dalamnya, yaitu aplikasi *mobile* sisi klien untuk pencatatan data ternak ruminansia secara

daring. Posisi kontribusi tersebut beserta titik singgungnya dengan para kolaborator diilustrasikan pada Gambar 3.2.



Gambar 3.2 Peta Kontribusi Penulis pada Proyek Kolaborasi Eco-system Livestock

Gambar 3.2 menunjukkan bahwa penulis bersinggungan dengan kolaborator hanya pada titik temu antarbagian, bukan pada kepemilikan komponennya. Penulis menggunakan kontrak *REST API* yang disediakan tim IT eksternal (mengonsumsi *endpoint*, bukan membangun *backend*), mengimplementasikan rancangan antarmuka (*mockup*) yang dibuat tim Universitas Sriwijaya tanpa ikut merancangnya, serta memanfaatkan jalur identifikasi *QR code* yang bersumber dari konteks penelitian *ear tag* Fariani dkk. (2025). Dengan demikian, peran penulis bersifat implementatif, yaitu membangun klien yang memadukan komponen-komponen tersebut, bukan menghasilkan komponennya.

Tabel 3.1 Peta Kontribusi pada Proyek Kolaborasi Eco-system Livestock

Komponen / Aktivitas	Penulis	UNSRI	Tim IT eksternal	Fariani dkk. (2025)
Penetapan variabel domain ternak (siklus birahi, bobot, indikator kesehatan)		✓		

Perancangan mockup antarmuka (Figma)		✓		
Pengembangan backend dan <i>REST API</i>			✓	
Perancangan basis data			✓	
Prototipe ear tag multi-teknologi (RFID/NFC/QR) dan uji 50 sapi Bali		✓		✓
Pengembangan aplikasi mobile Android (klien)	✓			
Implementasi arsitektur MVVM dengan <i>Provider</i>	✓			
Integrasi <i>REST API</i> dari sisi klien	✓			
Implementasi pemindai QR code	✓			
Pengujian black-box, API integration, dan SUS	✓			

Tabel 3.1 merinci pembagian kerja tersebut per komponen. Penetapan variabel domain ternak dan perancangan *mockup* antarmuka menjadi tanggung jawab tim Universitas Sriwijaya, sedangkan pengembangan *backend*, *REST API*, dan basis data menjadi tanggung jawab tim IT eksternal. Prototipe *ear tag* multi-teknologi beserta pengujiannya pada 50 sapi Bali merupakan hasil yang dilaporkan Fariani dkk. (2025) dan menjadi konteks domain bagi penelitian ini. Adapun kontribusi orisinal penulis pada skripsi ini adalah pengembangan aplikasi *mobile* Android sisi klien, yang mencakup penerapan arsitektur MVVM dengan *Provider*, integrasi *REST API* dari sisi klien, implementasi pemindai *QR code*, serta seluruh pelaksanaan pengujian *black-box*, *API integration*, dan *usability* (SUS).

3.3 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengidentifikasi kebutuhan aplikasi yang akan dikembangkan. Analisis kebutuhan pada penelitian ini dibagi menjadi tiga bagian, yaitu kebutuhan fungsional yang menjabarkan fitur-fitur yang harus disediakan aplikasi, kebutuhan non-fungsional yang menjabarkan karakteristik operasional aplikasi, dan kebutuhan perangkat keras dan lunak yang menjabarkan lingkungan pengembangan aplikasi.

3.3.1 Kebutuhan Fungsional

Kebutuhan fungsional adalah daftar fitur yang harus disediakan oleh aplikasi. Pada penelitian ini, kebutuhan fungsional dijabarkan berdasarkan dua *role* pengguna yang didukung oleh aplikasi, yaitu penyuluh produksi dan peternak. Kedua *role* tersebut memiliki hak akses yang berbeda terhadap data ternak, dan validasinya dilakukan di sisi *REST API*. Selain fitur yang spesifik untuk masing-masing *role*, terdapat juga fitur yang dapat diakses oleh kedua *role*, yaitu fitur autentikasi dan pemindai QR code.

Daftar kebutuhan fungsional aplikasi disajikan pada Tabel 3.2. Setiap kebutuhan fungsional dicantumkan beserta *role* pengguna yang dapat mengaksesnya dan *endpoint REST API* yang dipanggil aplikasi untuk memenuhi kebutuhan tersebut.

Tabel 3.2 Kebutuhan Fungsional Aplikasi

Kode	Nama Fitur	Role Pengguna	Endpoint <i>REST API</i>	Method HTTP
KF-01	Login	Kedua role	/login	POST
KF-02	Lihat profil pengguna	Kedua role	/profile	GET
KF-03	Logout	Kedua role	—	—
KF-04	Scan QR code dan tampilkan detail ternak	Kedua role	/livestock/scan/:id	GET
KF-05	Lihat daftar desa binaan	Penyuluh produksi	/fostered-breeders	GET
KF-06	Lihat daftar peternak per desa	Penyuluh produksi	/fostered-breeders/each-village?uuid={uuid_desa}	GET

KF-07	Lihat detail peternak	Penyuluh produksi	/breeder/:uuid	GET
KF-08	Lihat daftar ternak milik peternak	Penyuluh produksi	/livestock?uuid_peternak={uuid_peternak}	GET
KF-09	Lihat detail ternak	Kedua role	/livestock/:id	GET
KF-10	Lihat riwayat ternak	Kedua role	/livestock-history?id={id_ternak}	GET
KF-11	Tambah ternak baru	Penyuluh produksi	/livestock	POST
KF-12	Perbarui data dan riwayat ternak	Penyuluh produksi	/livestock	PUT
KF-13	Lihat daftar ternak miliknya	Peternak	/livestock?uuid_peternak={uuid_peternak}	GET
KF-14	Ambil master data untuk dropdown form	Penyuluh produksi	/type-of-animal /type-of-production /type-of-livestock /type-of-feed /supplement /health-status /reproduction-status	GET

Berdasarkan Tabel 3.2, fitur-fitur aplikasi dikelompokkan ke dalam tiga kategori. Kategori pertama adalah fitur universal yang dapat diakses oleh kedua role, meliputi login,

lihat profil, logout, scan QR code, lihat detail ternak, dan lihat riwayat ternak. Kategori kedua adalah fitur khusus role penyuluh produksi yang meliputi manajemen data peternak binaan, manajemen data ternak, dan pengambilan master data untuk mendukung form tambah dan perbarui ternak. Kategori ketiga adalah fitur khusus role peternak yang terbatas pada melihat daftar ternak miliknya sendiri.

Pada Tabel 3.2 terlihat bahwa KF-08 dan KF-13 menggunakan endpoint yang sama, yaitu */livestock* dengan parameter *uuid_peternak*. Perbedaan keduanya terletak pada nilai *uuid* yang dikirim. Pada KF-08, penyuluh produksi mengirim *uuid* peternak yang sedang dilihat detailnya, sedangkan pada KF-13, peternak mengirim *uuid*-nya sendiri yang diperoleh dari data profil. *REST API* memvalidasi otorisasi berdasarkan *role* pengguna yang terautentikasi melalui *bearer token*, khususnya untuk operasi tulis pada KF-11 dan KF-12. Pada sisi aplikasi, peternak hanya mengirim *uuid*-nya sendiri yang diperoleh dari data profil, sehingga melalui penggunaan normal aplikasi peternak hanya mengakses daftar ternak miliknya sendiri. Hasil pengujian *API integration* terhadap pembatasan berbasis kepemilikan pada *endpoint* pembacaan diuraikan lebih lanjut pada Bab IV. Pembatasan akses terhadap fitur yang sensitif seperti KF-11 tambah ternak dan KF-12 perbarui ternak juga dilakukan oleh *REST API* berdasarkan *role* pengguna yang terautentikasi melalui *bearer token*. Pengiriman *request* pada KF-11 dan KF-12 menggunakan format *multipart form-data* karena menyertakan berkas gambar ternak yang disisipkan sebagai *field* bertipe *base64 string*.

3.3.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional adalah karakteristik operasional aplikasi yang tidak berkaitan langsung dengan fitur, tetapi menentukan kualitas aplikasi secara keseluruhan. Pada penelitian ini, kebutuhan non-fungsional mencakup lima aspek sebagai berikut.

- a. **Performa.** Aplikasi diharapkan mampu merespons interaksi pengguna dalam waktu yang wajar. Waktu respon ini mencakup waktu yang dibutuhkan aplikasi untuk mengirim *request* ke *REST API*, menerima *response*, dan menampilkannya pada antarmuka. Karena aplikasi menggunakan arsitektur *real-time sync* dengan *REST API*, performa aplikasi juga dipengaruhi oleh kondisi jaringan pengguna. Pemenuhan aspek performa ini diverifikasi melalui pengukuran *response time endpoint* dari sisi klien yang hasilnya disajikan pada Bab IV Tabel 4.3.
- b. **Usabilitas.** Aplikasi diharapkan dapat digunakan dengan mudah oleh dua kelompok pengguna yang berbeda, yaitu penyuluh produksi dan peternak. Antarmuka aplikasi

harus mengikuti *mockup* yang sudah dirancang oleh tim kolaborator dari Universitas Sriwijaya, sehingga konsistensi tampilan dengan aplikasi *web* yang sudah ada dapat terjaga. Fitur pemindai *QR code* juga dirancang untuk mengurangi kebutuhan pengetikan manual ID ternak, sehingga mempercepat identifikasi individu ternak di lapangan. Tingkat pemenuhan aspek usabilitas ini diukur melalui pengujian *System Usability Scale* (SUS) yang hasilnya diuraikan pada Bab IV sub-bab 4.2.3.

- c. **Kompatibilitas Android.** Aplikasi dikembangkan menggunakan *framework* Flutter dan ditargetkan untuk berjalan pada perangkat *smartphone* Android. Aplikasi harus dapat dijalankan pada perangkat Android yang umum digunakan oleh penyuluh produksi dan peternak tanpa memerlukan spesifikasi perangkat keras khusus. Pemenuhan aspek ini dibuktikan melalui pelaksanaan *black-box testing* secara langsung pada perangkat Android sebagaimana disajikan pada Bab IV Tabel 4.1, yang menunjukkan seluruh fitur berjalan sebagaimana mestinya pada perangkat target.
- d. **Ketergantungan Konektivitas.** Aplikasi dirancang dengan pendekatan *full online*, artinya seluruh operasi aplikasi membutuhkan koneksi internet untuk berkomunikasi dengan *REST API*. Aplikasi tidak menyimpan data ternak atau data peternak secara lokal di perangkat, melainkan mengambilnya langsung dari *REST API* setiap kali dibutuhkan. Pengelolaan mode *offline* dan penyimpanan basis data lokal tidak termasuk ruang lingkup penelitian ini sebagaimana sudah dijelaskan pada Bab I batasan masalah. Perilaku aplikasi ketika koneksi tidak tersedia diverifikasi pada skenario *black-box* TC-03, yaitu aplikasi menampilkan pesan kesalahan jaringan secara aman tanpa mengalami *crash*.
- e. **Keamanan.** Aplikasi menerapkan dua lapis pengamanan untuk akses data. Lapis pertama adalah autentikasi menggunakan mekanisme *bearer token* yang diperoleh saat *login* dan disertakan pada *header* setiap permintaan ke *REST API*. Lapis kedua adalah otorisasi berbasis *role* yang ditegakkan oleh *REST API*, sehingga permintaan terhadap *endpoint* tertentu akan ditolak apabila *role* pengguna tidak memiliki hak akses. Token autentikasi disimpan secara lokal di perangkat menggunakan *SharedPreferences* (Shared Preferences, n.d.) dan dihapus ketika pengguna melakukan *logout*. Pemenuhan kedua lapis pengamanan ini diverifikasi melalui pengujian *API integration*, yaitu penolakan permintaan tanpa token atau dengan token tidak valid (AT-04 dan AT-12) untuk lapis autentikasi, serta penolakan akses oleh *role* yang tidak berwenang (AT-08 dan AT-16) untuk lapis otorisasi, sebagaimana diuraikan pada Bab IV Tabel 4.2.

3.3.3 Kebutuhan Perangkat Keras dan Lunak

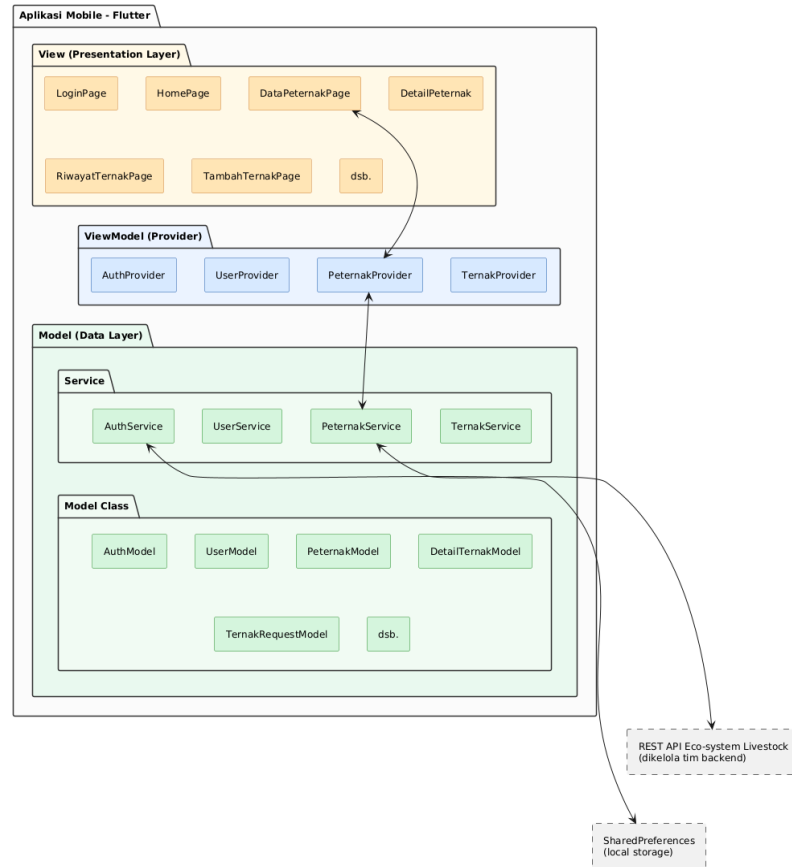
Analisis kebutuhan perangkat keras dan lunak dilakukan untuk mengidentifikasi perangkat yang digunakan dalam proses pembuatan aplikasi. Perangkat lunak yang digunakan meliputi *framework* Flutter versi 3.41.4 dengan bahasa pemrograman Dart versi 3.11.1, serta *Integrated Development Environment* (IDE) *Android Studio* Panda 2 versi 2025.3.2. Perangkat keras yang digunakan berupa laptop dengan sistem operasi Windows sebagai lingkungan pengembangan, dan *smartphone* Android sebagai *device* target untuk pengujian aplikasi.

3.4 Perancangan Sistem

Perancangan sistem pada penelitian ini dilakukan untuk memodelkan aplikasi *mobile* Eco-system Livestock dari sisi klien sebelum tahap implementasi. Perancangan difokuskan pada arsitektur aplikasi, pemodelan interaksi pengguna melalui diagram UML, dan pemetaan kelas model Dart yang merepresentasikan data dari *REST API*. Perancangan antarmuka visual tidak termasuk pada sub-bab ini karena *mockup* antarmuka disediakan oleh tim kolaborator dari Universitas Sriwijaya sebagaimana sudah dijelaskan pada Bab I batasan masalah..

3.4.1 Arsitektur Aplikasi

Aplikasi Eco-system Livestock menerapkan pola arsitektur *Model-View-ViewModel* (MVVM) yang sudah dijelaskan pada Bab II sub-bab 2.2.5. Penerapan pola MVVM pada aplikasi dilakukan dengan membagi kode ke dalam tiga lapisan utama, yaitu *View* sebagai antarmuka pengguna, *ViewModel* sebagai penghubung antara *View* dan sumber data, dan *Model* sebagai lapisan data yang terdiri dari *service* dan kelas model. Diagram arsitektur aplikasi disajikan pada Gambar 3.3

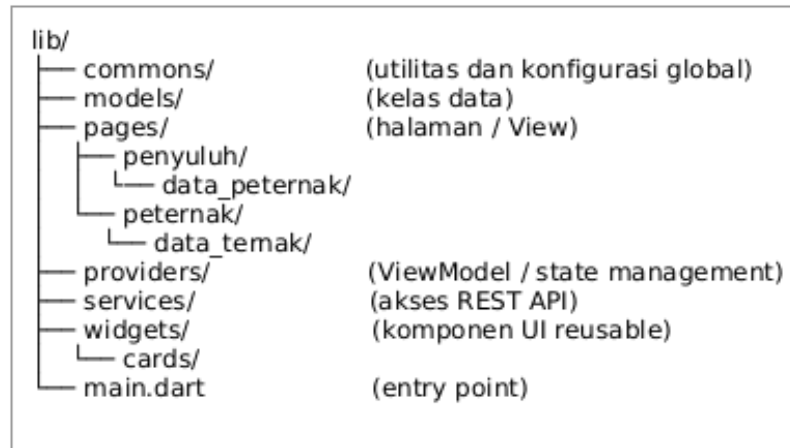


Gambar 3.3 Diagram Arsitektur Aplikasi

Berdasarkan Gambar 3.3, alur data pada aplikasi mengikuti pola berjenjang dari lapisan paling atas ke lapisan paling bawah. Saat pengguna berinteraksi dengan antarmuka, misalnya menekan tombol atau mengisi *form*, lapisan *View* memanggil *method* pada lapisan *ViewModel* yang sesuai. *ViewModel* kemudian memanggil *method* pada lapisan *Service* untuk mengambil atau mengirim data dari *REST API*. Setelah *Service* menerima *response* dari *REST API*, data dipetakan ke kelas model, lalu dikembalikan ke *ViewModel*. Perubahan *state* pada *ViewModel* diteruskan ke *View* melalui mekanisme *notifyListeners* dari *Provider*, sehingga *View* melakukan *rebuild* dengan data terbaru.

Pada aplikasi ini, lapisan *View* diimplementasikan sebagai *widget* dan halaman Flutter yang terletak pada folder *pages/* dan *widgets/*. Lapisan *ViewModel* diimplementasikan sebagai kelas *provider* yang meng-*extends* *ChangeNotifier* dan terletak pada folder *providers/*. Lapisan *Service* terletak pada folder *services/* dan bertanggung jawab melakukan pemanggilan HTTP menggunakan *package http*, sedangkan kelas model terletak pada folder *models/*. Pemisahan folder ini mencerminkan penerapan prinsip *separation of concerns* pada pola MVVM, yaitu

setiap lapisan hanya bertanggung jawab pada satu fungsi tertentu. Struktur folder aplikasi disajikan pada Gambar 3.4.

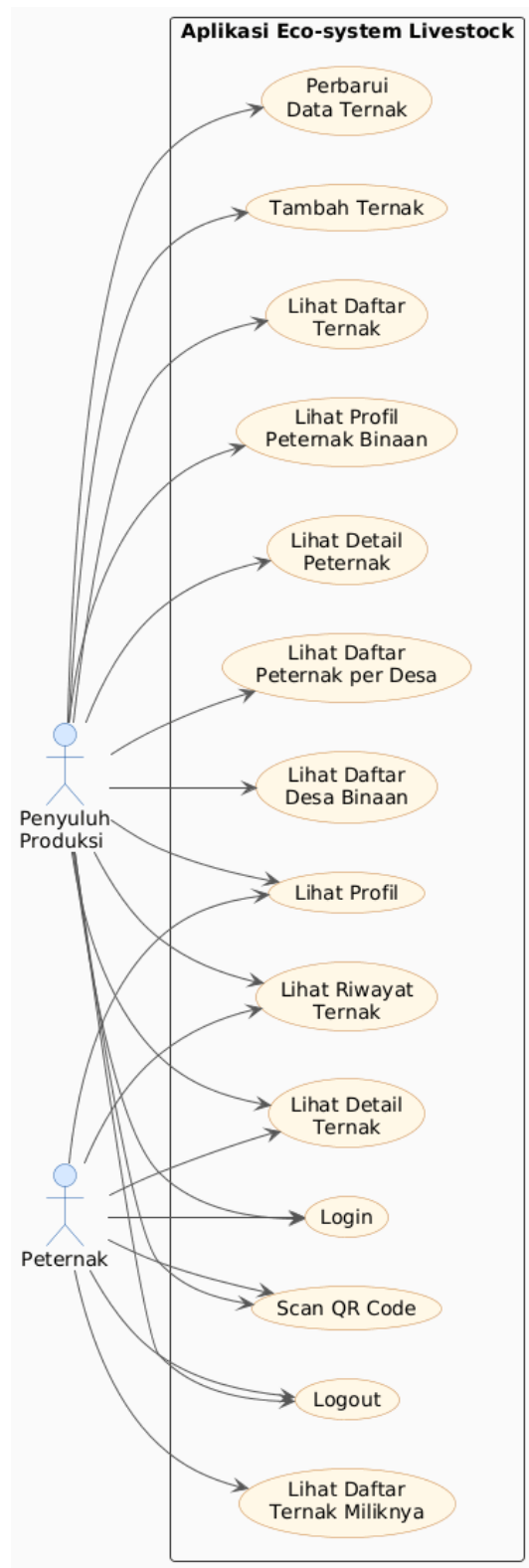


Gambar 3.4 Struktur Folder Proyek Flutter

Pada *main.dart* sebagai *entry point* aplikasi, keempat *provider* didaftarkan melalui *MultiProvider* sehingga dapat diakses dari seluruh lapisan *View* yang berada di bawahnya pada *widget tree*. Token autentikasi yang diperoleh setelah *login* disimpan secara lokal melalui *SharedPreferences* agar dapat digunakan pada pembukaan aplikasi berikutnya tanpa harus *login* ulang, selama *token* masih berlaku.

3.4.2 Use Case Diagram

Use case diagram digunakan untuk memodelkan interaksi antara aktor dengan sistem. Pada aplikasi Eco-system Livestock terdapat dua aktor, yaitu penyuluh produksi sebagai supervisor yang memantau peternak binaan, dan peternak sebagai pengguna yang hanya melihat data ternak miliknya sendiri. Kedua aktor terautentikasi melalui mekanisme yang sama, namun hak akses terhadap fitur dibedakan oleh *REST API* berdasarkan *role* pengguna. *Use case diagram* aplikasi disajikan pada Gambar 3.5.



Gambar 3.5 Use Case Diagram Aplikasi Eco-system Livestock

Berdasarkan Gambar 3.5, aktor penyuluh produksi dapat mengakses 13 *use case* sementara peternak hanya mengakses 7 *use case*. Keenam *use case* yang dapat diakses kedua

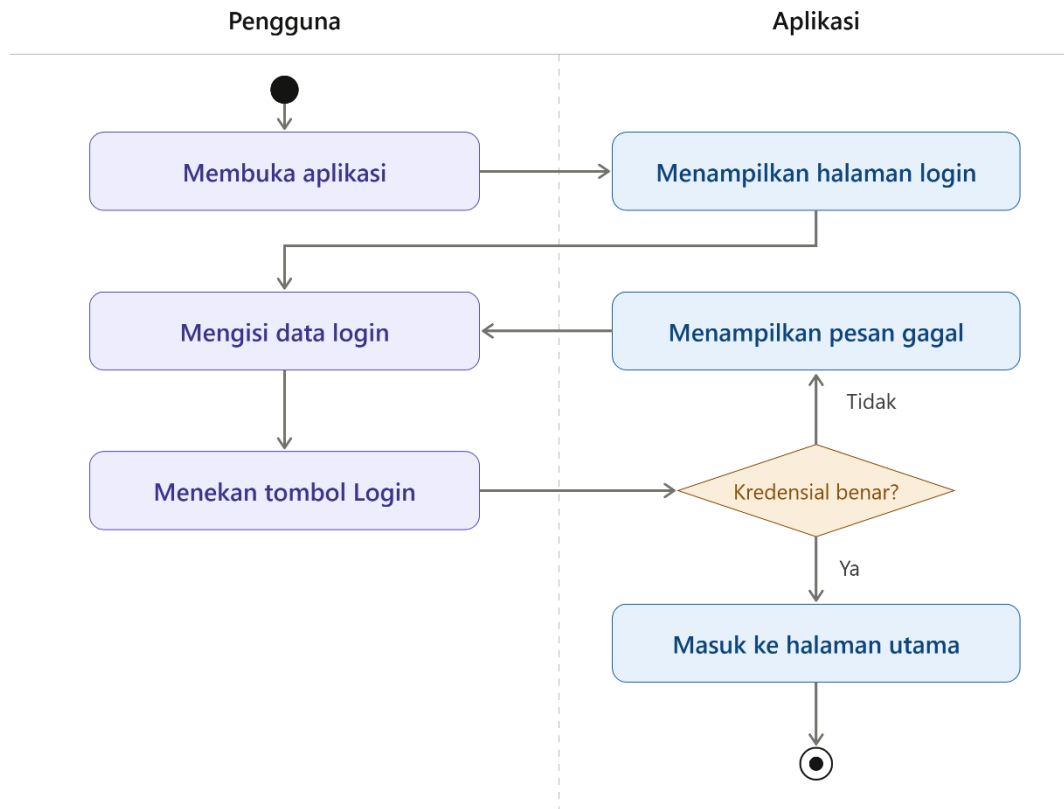
aktor adalah *use case* universal yang terdiri dari *login*, *logout*, lihat profil diri, *scan QR code*, lihat detail ternak, dan lihat riwayat ternak. *Use case* khusus penyuluh produksi adalah *use case* yang berkaitan dengan manajemen data peternak binaan (lihat daftar desa, lihat daftar peternak per desa, lihat detail peternak, lihat profil peternak binaan) dan manajemen data ternak pada lingkup peternak binaan (lihat daftar ternak milik peternak tertentu, tambah ternak, perbarui data ternak). *Use case* khusus peternak adalah melihat daftar ternak miliknya sendiri.

Pembedaan jumlah *use case* antar aktor tersebut mencerminkan perbedaan tanggung jawab keduanya. Penyuluh produksi bertindak sebagai supervisor yang mengelola data ternak untuk sejumlah peternak binaan, sehingga memerlukan akses terhadap *use case* tambah dan perbarui data. Peternak bertindak sebagai pengguna akhir yang hanya memantau ternak miliknya sendiri, sehingga akses *use case*-nya terbatas pada membaca data. Pembatasan akses tersebut tidak dilakukan pada sisi aplikasi, melainkan pada sisi *REST API* melalui validasi *bearer token* dan *role* pengguna pada setiap permintaan. Rincian *endpoint REST API* yang dipanggil untuk mendukung setiap *use case* sudah dijabarkan pada Tabel 3.2 sub-bab sebelumnya.

3.4.3 Activity Diagram

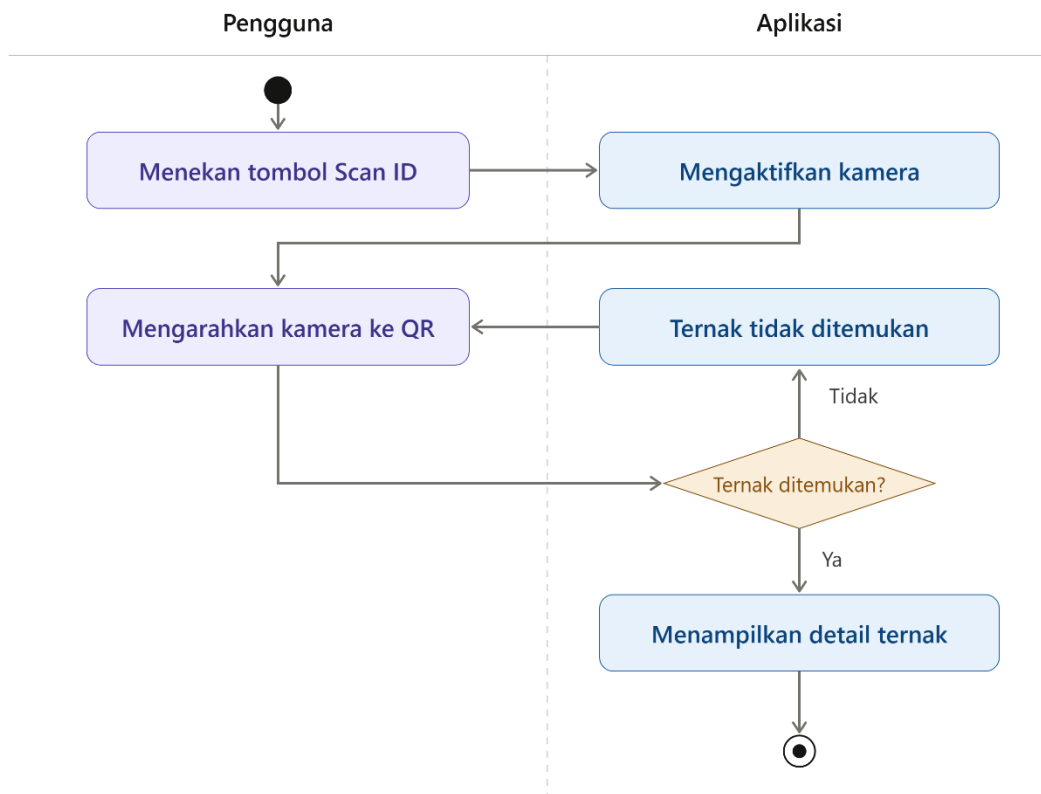
Activity diagram digunakan untuk memodelkan alur aktivitas pengguna saat berinteraksi dengan aplikasi, termasuk percabangan keputusan (*decision*) yang terjadi di dalamnya. Pada penelitian ini, *activity diagram* dibatasi pada empat alur yang memiliki percabangan signifikan, yaitu alur autentikasi, alur *scan QR code*, alur tambah ternak, dan alur pemutakhiran riwayat ternak. Alur-alur tanpa percabangan seperti melihat daftar atau *logout* tidak ditampilkan sebagai *activity diagram* terpisah karena sudah tercakup pada *use case* dan sifatnya linear.

Alur autentikasi merupakan titik masuk utama aplikasi. Pengguna membuka aplikasi, memasukkan *username* dan *password*, kemudian menekan tombol *Login*. Aplikasi memeriksa koneksi internet terlebih dahulu, lalu mengirim *request* ke *REST API*. Jika kredensial valid, aplikasi menerima *access token* dan *role* pengguna, menyimpannya pada *SharedPreferences*, dan meneruskan pengguna ke halaman utama. Jika kredensial tidak valid atau koneksi terputus, aplikasi menampilkan pesan kesalahan dan pengguna tetap berada pada halaman *login*. *Activity diagram* untuk alur autentikasi disajikan pada Gambar 3.6.



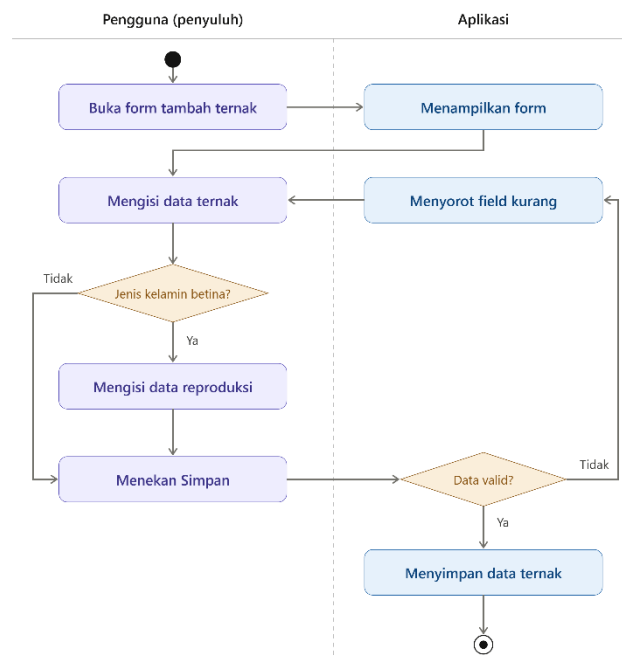
Gambar 3.6 Activity Diagram Autentikasi (Login)

Alur *scan QR code* memungkinkan pengguna mengidentifikasi ternak secara cepat tanpa harus mengetik ID secara manual. Pengguna menekan tombol *Floating Action Button* (FAB) bertanda "Scan ID" pada bagian tengah bawah aplikasi, kemudian kamera perangkat diaktifkan. Setelah QR terbaca, aplikasi mengekstrak ID ternak dan mengirim *request* ke *endpoint /livestock/scan/:id*. Jika ID ditemukan di basis data, aplikasi menampilkan halaman detail ternak. Jika tidak ditemukan, aplikasi menampilkan pesan kesalahan. *Activity diagram* untuk alur *scan QR code* disajikan pada Gambar 3.7.



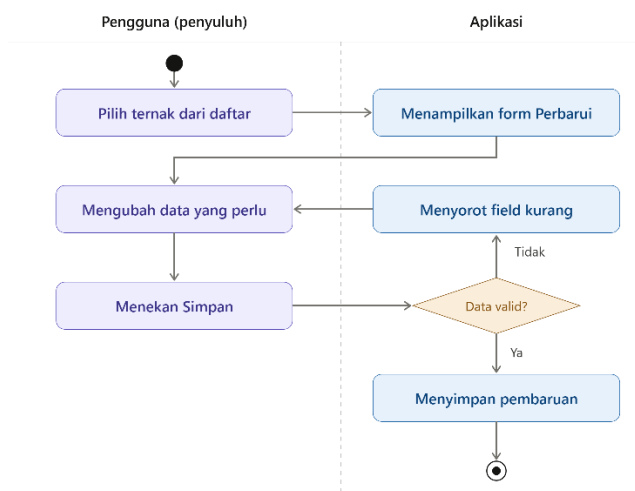
Gambar 3.7 Activity Diagram Pemindaian QR Code

Alur tambah ternak hanya dapat diakses oleh penyuluh produksi. Penyuluh membuka halaman detail peternak, menekan tombol "Tambah Ternak", lalu mengisi *form* yang terbagi dalam empat *section*, yaitu identitas utama, produktivitas, kesehatan, dan reproduksi. *Section* reproduksi hanya tampil apabila jenis kelamin ternak dipilih "Betina". Setelah *form* diisi, penyuluh menekan tombol simpan dan aplikasi melakukan validasi *form*. Jika validasi lolos, aplikasi mengirim *request* POST ke *endpoint* `/livestock`. Jika tidak lolos, aplikasi menggulir otomatis ke *field* pertama yang belum terisi. *Activity diagram* untuk alur tambah ternak disajikan pada Gambar 3.8.



Gambar 3.8 Activity Diagram Penambahan Data Ternak

Alur pemutakhiran riwayat ternak juga hanya dapat diakses oleh penyuluh produksi. Penyuluh memilih satu ternak dari daftar, lalu membuka tab "Perbarui" pada halaman riwayat ternak. Tab ini menampilkan *form* serupa dengan *form* tambah ternak, namun nilai *field* sudah terisi secara otomatis dengan data riwayat terbaru. Penyuluh dapat memperbarui nilai yang dibutuhkan, kemudian menekan tombol simpan. Aplikasi mengirim *request* PUT ke *endpoint* /*livestock*. *Activity diagram* untuk alur ini disajikan pada Gambar 3.9.



Gambar 3.9 Activity Diagram Pemutakhiran Riwayat Ternak

3.4.4 Pemetaan Kelas Model Dart dari Response API

REST API Eco-system Livestock mengembalikan data dalam format JSON untuk setiap *endpoint* yang diakses. Pada sisi aplikasi *mobile*, struktur JSON tersebut perlu dipetakan ke dalam kelas Dart agar data dapat digunakan secara aman dan terstruktur. Pemetaan ini dilakukan melalui pola *factory constructor fromJson*. Pola tersebut membaca *Map<String, dynamic>* dari *response*, lalu mengisi *field* kelas model dengan nilai yang sesuai. Struktur data pada *REST API* dirancang oleh tim *backend*, sehingga tugas pada sisi klien *mobile* adalah memetakan struktur tersebut tanpa mengubah skema aslinya.

Pola pemetaan yang diterapkan pada aplikasi dibedakan menjadi dua arah, yaitu pemetaan dari *response* API ke kelas model untuk ditampilkan di *View*, dan pemetaan dari *form input* ke kelas model untuk dikirim sebagai *request* ke API. Pemetaan arah pertama dijumpai pada sebagian besar kelas model, misalnya *PeternakModel* yang memetakan data peternak dengan lima *field* sederhana, serta *DetailTernakModel* yang memetakan data ternak secara *nested* ke dalam empat kelas anak yaitu *IdentitasTernak*, *ProduktivitasTernak*, *KesehatanTernak*, dan *ReproduksiTernak*. Pola *nested class* ini dipilih karena *response* JSON dari API juga terstruktur dalam empat grup serupa, sehingga pemisahan kelas pada sisi aplikasi dapat memudahkan akses data melalui *dot notation* yang rapi.

Pemetaan arah kedua diterapkan pada kelas *TernakRequestModel* yang digunakan untuk dua *endpoint*, yaitu tambah ternak (KF-11) dan pemutakhiran ternak (KF-12). Kelas tersebut memiliki *method toJson()* yang mengubah seluruh *field* menjadi *Map<String, String>* yang kemudian dikirim sebagai *multipart body*. Perbedaan antara *request* tambah dan pemutakhiran ditentukan oleh satu *field* opsional bernama *uuidTernak*. Jika *field* ini bernilai *null*, *request* yang dikirim merupakan permintaan tambah ternak. Jika berisi nilai, *field uuid_ternak* akan disertakan pada *payload*, menandakan permintaan pemutakhiran data ternak yang sudah ada. Pendekatan ini memungkinkan satu kelas *request* dipakai untuk dua tujuan berbeda tanpa menduplikasi definisi *field*. Implementasi kode dari kelas-kelas model yang sudah dijelaskan pada sub-bab ini akan dijabarkan pada Bab IV sub-bab 4.1.1.

3.5 Metode Pengujian

Pengujian aplikasi Eco-system Livestock dilakukan untuk memastikan fitur-fitur yang sudah dikembangkan berjalan sesuai dengan kebutuhan fungsional yang ditetapkan pada Sub-bab 3.2.1. Pada penelitian ini, pengujian dilakukan melalui tiga metode yang saling melengkapi, yaitu *black-box testing* untuk memverifikasi perilaku aplikasi dari sisi antarmuka

pengguna, *API integration testing* untuk memverifikasi komunikasi aplikasi dengan *REST API*, dan pengujian *usability* dengan *System Usability Scale* (SUS) untuk mengukur tingkat keterterimaan aplikasi oleh pengguna. Landasan teori ketiga metode pengujian sudah dijabarkan pada sub-bab 2.2.9.

3.5.1 Black-box Testing

Black-box testing dilakukan untuk memeriksa perilaku aplikasi dari sisi pengguna tanpa melihat struktur kode internal. Pengujian ini berfokus pada kesesuaian perilaku aplikasi dengan kebutuhan fungsional yang sudah didefinisikan. Penyusunan *test case* dilakukan berdasarkan kode kebutuhan fungsional (KF-01 sampai KF-14) pada Tabel 3.2, dengan skenario yang mencakup masukan valid (*positive test case*) maupun masukan tidak valid (*negative test case*) untuk setiap fitur. Skenario valid memeriksa perilaku aplikasi ketika masukan yang diberikan sesuai dengan aturan aplikasi, sedangkan skenario tidak valid memeriksa bagaimana aplikasi menangani masukan yang tidak sesuai atau kondisi yang tidak diharapkan seperti *field* wajib yang dikosongkan.

Setiap *test case* diberi kode TC-01 dan seterusnya sebagai pengenal. Format *test case* yang digunakan untuk pengujian *black-box* pada penelitian ini disajikan pada Tabel 3.3.

Tabel 3.3 Format *Test Case Black-box Testing*

Kode TC	KF	Nama Skenario	Langkah Pengujian	Hasil yang Diharapkan	Status
TC-01	KF-01	Login dengan kredensial valid	1. Buka aplikasi. 2. Masukkan username dan password yang benar. 3. Tekan tombol Masuk.	Aplikasi berpindah ke HomePage sesuai <i>role</i> pengguna.	Valid

Setiap *test case* akan dieksekusi pada aplikasi yang sudah diimplementasikan, kemudian hasilnya dicatat pada kolom Status dengan keterangan berhasil atau gagal. Daftar lengkap *test case* beserta hasil eksekusinya akan dijabarkan pada Bab IV sub-bab 4.2.1.

3.5.2 API Integration Testing

API integration testing dilakukan untuk memastikan komunikasi antara aplikasi dan *REST API* berjalan sesuai dengan kontrak *endpoint* yang disediakan tim *backend*. Pengujian ini berfokus pada kesesuaian *request* yang dikirim aplikasi dengan spesifikasi *endpoint*, serta kemampuan aplikasi memproses *response* yang dikembalikan *server*, baik pada kondisi sukses maupun pada kondisi kesalahan. Alat bantu yang digunakan untuk pengujian ini adalah Postman, sebagaimana sudah dijelaskan pada Bab II sub-bab 2.2.9.

Pengujian dilakukan dengan mengirim *request* ke setiap *endpoint* yang tercantum pada Tabel 3.2 menggunakan konfigurasi yang sama dengan yang dipakai aplikasi. Konfigurasi tersebut mencakup *method* HTTP, URL lengkap dengan *query parameter*, *header* (*Authorization: Bearer {token}* dan *Content-Type*), serta *body* sesuai format *endpoint*. Pengujian pada setiap *endpoint* dirancang dengan empat kategori skenario: (1) sukses dengan kredensial dan parameter valid; (2) gagal autentikasi ketika *token* tidak disertakan atau tidak berlaku; (3) gagal otorisasi ketika *role* pengguna tidak memiliki hak akses terhadap *endpoint* yang diminta; (4) gagal validasi ketika *body request* tidak sesuai dengan spesifikasi *endpoint*. Keempat kategori tersebut penting untuk membuktikan bahwa mekanisme keamanan berbasis *bearer token* dan pembatasan *role-based access* pada *REST API* berjalan sesuai rancangan sebagaimana disebutkan pada kebutuhan non-fungsional keamanan di Sub-bab 3.2.2.

Setiap *test case API integration* diberi kode AT-01 dan seterusnya sebagai pengenalan. Format *test case* yang digunakan untuk pengujian *API integration* pada penelitian ini disajikan pada Tabel 3.4.

Tabel 3.4 Format *Test Case API Integration Testing*

Kode AT	Endpoint	Method	Skenario	Request	Hasil yang Diharapkan	Status
AT-01	/login	POST	Sukses login kredensial valid	body: username dan password valid	200 OK, body berisi access_token, role, uuid	Valid

Setiap *test case* akan dieksekusi menggunakan Postman, kemudian hasilnya dicatat pada kolom Status. Daftar lengkap *test case API integration* beserta hasil eksekusinya akan dijabarkan pada Bab IV sub-bab 4.2.2.

Selain memverifikasi kesesuaian kontrak, pengujian pada *endpoint REST API* juga mencakup pengukuran *response time* sebagai indikator pemenuhan kebutuhan non-fungsional performa pada sub-bab 3.2.2. Pengukuran dilakukan menggunakan Postman terhadap *endpoint* yang sama. Setiap *endpoint* dipanggil beberapa kali; *request* pertama diabaikan untuk menghilangkan *overhead* penyiapan koneksi yang hanya terjadi sekali, lalu *response time* rata-rata dihitung dari sejumlah *request* berikutnya yang merepresentasikan kondisi *steady-state*. Pengukuran ini bersifat indikatif dari sisi klien dan bukan merupakan *load testing* maupun pengujian performa *server*. Hasil pengukuran *response time* dijabarkan pada Bab IV sub-bab 4.2

3.5.3 API Integration Testing

Pengujian usability dirancang untuk mengukur tingkat keterterimaan aplikasi Ecosystem Livestock dari sudut pandang pengguna, sebagai pelengkap pengujian fungsional pada sub-bab 3.4.1 dan 3.4.2. Pengujian ini menggunakan instrumen System Usability Scale (SUS) yang landasan teorinya sudah dijabarkan pada sub-bab 2.2.9.

Instrumen SUS terdiri atas sepuluh pernyataan baku dengan nada positif dan negatif berselang-seling, yang masing-masing dinilai menggunakan skala Likert lima tingkat dari Sangat Tidak Setuju (1) sampai Sangat Setuju (5). Kuesioner disusun dalam bahasa Indonesia dan disebar secara daring melalui Google Form. Responden diminta menggunakan aplikasi terlebih dahulu sebelum mengisi kuesioner, agar penilaian didasarkan pada pengalaman pemakaian langsung.

Skor SUS dihitung dengan ketentuan baku, yaitu kontribusi pernyataan bernomor ganjil dihitung dari nilai jawaban dikurangi satu, kontribusi pernyataan bernomor genap dihitung dari lima dikurangi nilai jawaban, lalu seluruh kontribusi dijumlahkan dan dikalikan 2,5 sehingga menghasilkan skor akhir pada rentang 0 sampai 100 untuk tiap responden. Skor akhir aplikasi diperoleh dari rata-rata skor seluruh responden, kemudian ditafsirkan menggunakan kerangka keterterimaan dan label kata sifat dari Bangor dkk. (2009). Pelaksanaan beserta hasil pengujian usability dijabarkan pada Bab IV sub-bab 4.2.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

Bab ini memaparkan hasil implementasi aplikasi *mobile* Eco-system Livestock yang dibangun berdasarkan perancangan pada Bab III, beserta hasil pengujian yang dilakukan untuk memastikan fungsionalitas aplikasi sudah sesuai kebutuhan. Pembahasan dimulai dari implementasi arsitektur aplikasi, dilanjutkan dengan implementasi fitur per menu, hasil pengujian *black-box* dan *API integration*, dan diakhiri dengan pembahasan atas hasil yang diperoleh.

4.1 Implementasi Aplikasi

Implementasi aplikasi dilakukan mengikuti rancangan yang telah disusun pada Bab III. Sub-bab ini menjabarkan hasil implementasi dari sisi arsitektur aplikasi, fitur autentikasi, fitur-fitur utama yang dapat diakses oleh masing-masing *role* pengguna, pemindai QR *code* sebagai fitur universal, dan pola akses ke *REST API*. Cuplikan kode dan tangkapan layar antarmuka disajikan pada setiap sub-bab untuk menunjukkan bukti realisasi fitur.

4.1.1 Implementasi Arsitektur MVVM

Pola MVVM diwujudkan di kode aplikasi melalui pemisahan empat folder utama: *pages/* dan *widgets/* untuk lapisan *View*, *providers/* untuk *ViewModel*, *services/* untuk akses *REST API*, dan *models/* untuk kelas data. Pemisahan fisik ini membuat setiap perubahan terkurung pada satu lapisan saja. Misalnya, ketika tampilan halaman *login* diubah, kode pemanggilan *endpoint /login* di *services/* tidak ikut tersentuh. Sebaliknya, ketika kontrak *REST API* berubah dari sisi *backend*, perubahan cukup dilakukan di kelas *service* dan *model* tanpa harus menyusuri seluruh *widget*.

Sub-bab ini fokus pada hasil implementasi lapisan *Model*, yaitu pemetaan *response REST API* ke kelas Dart agar data dapat digunakan secara aman dan terstruktur oleh lapisan di atasnya. Implementasi lapisan *View*, *ViewModel*, dan *Service* dibahas pada sub-bab selanjutnya seiring fitur yang memanfaatkan lapisan tersebut. Tiga kelas dipilih sebagai representasi karena masing-masing menunjukkan pola implementasi yang berbeda: *PeternakModel* untuk pemetaan sederhana dengan sumber data tambahan, *DetailTernakModel* untuk pemetaan *nested* dengan struktur kelas berjenjang, dan *TernakRequestModel* untuk pemetaan dua arah dengan *field* opsional sebagai pembeda operasi.

Kelas *PeternakModel* merupakan contoh pemetaan sederhana dari *response* API ke objek Dart. Kelas ini terdiri dari lima *field* inti, yaitu *uuid*, *id*, *nama*, *nik*, dan *desa*. *Factory fromJson* pada kelas ini menerapkan pola *fallback* melalui operator *null-aware* (??) untuk menjaga aplikasi tetap aman apabila *response* API mengembalikan *field* yang bernilai *null*. *Factory* tersebut menerima dua parameter, yaitu *json* sebagai *Map<String, dynamic>* hasil *parsing response* API dan *namaDesa* sebagai *String* yang bersumber dari *response* wilayah. Hal ini dilakukan karena struktur *response* API pada *endpoint* peternak tidak menyertakan nama desa secara langsung pada objek peternak, melainkan hanya menyertakan identifier wilayah, sehingga nama desa perlu dilengkapi dari sumber lain pada sisi aplikasi. Pola ini menunjukkan bahwa kelas model tidak selalu berupa pemetaan satu lawan satu dari *response* API, tetapi dapat pula menggabungkan data dari beberapa sumber.

Kelas *DetailTernakModel* merupakan contoh pemetaan *response* API yang lebih kompleks dengan struktur *nested*. *Response endpoint /livestock/:id* mengembalikan data ternak yang sudah dikelompokkan ke dalam empat objek anak, yaitu *identitas_ternak*, *produktivitas_ternak*, *kesehatan_ternak*, dan *reproduksi_ternak*. Empat objek anak ini dipetakan ke empat kelas Dart terpisah, yaitu *IdentitasTernak*, *ProduktivitasTernak*, *KesehatanTernak*, dan *ReproduksiTernak*. Kelas induk *DetailTernakModel* bertanggung jawab menampung keempat kelas anak tersebut sebagai *field*-nya. Pemisahan ini menjaga tipe data pada sisi *View* tetap kuat secara statis, sehingga pengaksesan *field* oleh *widget* tidak perlu melalui *Map<String, dynamic>* yang bertipe *dynamic*. Tiga kelas anak yaitu *ProduktivitasTernak*, *KesehatanTernak*, dan *ReproduksiTernak* mengikuti pola implementasi yang sama dengan *IdentitasTernak* sehingga tidak diuraikan secara terpisah. Pemanggilan *Formatters.capitalizeEachWord* dan *Formatters.formatTanggal* pada *factory fromJson* merupakan contoh penerapan utilitas global dari *folder commons/* yang bertugas menyeragamkan format data sebelum ditampilkan ke antarmuka pengguna.

Kelas *TernakRequestModel* merupakan kebalikan arah dari dua kelas sebelumnya. Jika *PeternakModel* dan *DetailTernakModel* memetakan *response* API ke objek Dart untuk *View*, maka *TernakRequestModel* memetakan isian *form* dari *View* ke format yang dapat diterima API sebagai *request body*. Kelas ini memegang seluruh *field* isian *form* tambah dan pemutakhiran ternak, yaitu 23 *field* wajib yang mencakup data identitas, produktivitas, kesehatan, dan reproduksi, beserta satu *field* opsional bernama *uuidTernak*. *Field* opsional inilah yang menjadi penanda pembeda antara operasi tambah ternak dan operasi pemutakhiran ternak. Apabila *uuidTernak* bernilai *null*, *method toJson* tidak menyertakan *field uuid_ternak*

pada *map* hasil, sehingga *request* yang dihasilkan cocok untuk operasi POST tambah ternak. Apabila *uuidTernak* berisi nilai, *field uuid_ternak* disertakan pada *map* hasil, sehingga *request* yang dihasilkan cocok untuk operasi PUT pemutakhiran ternak. Penggalan kode bagian relevan dari *TernakRequestModel* disajikan pada Gambar 4.1.

```
class TernakRequestModel {
    // Variabel pembeda Tambah (POST) atau Pemutakhiran (PUT)
    // Kalau null = Tambah. Kalau ada isinya = Pemutakhiran.

    final String? uuidTernak;
    // 23 field wajib lainnya: id, nfc, rfid, tanggalLahir,
    // jenisKelamin, uuidPeternak, uuidJenisTernak, bobotLahir,
    // bobot, pbb, jenisPakan, ... (dst)
    // ...
    TernakRequestModel({
        this.uuidTernak,
        required this.id,
        required this.nfc,
        // ... (field lainnya required)
    });
    Map<String, String> toJson() {
        Map<String, String> data = {
            'id': id,
            'uuid_peternak': uuidPeternak,
            'pertambahan_bobot_badan': pbb,
            'uuid_pakan': jenisPakan,
            // ... (mapping camelCase ke snake_case lainnya)
        };
        // Inti logika: uuid_ternak HANYA disertakan saat Pemutakhiran
        if (uuidTernak != null) {
            data['uuid_ternak'] = uuidTernak!;
        }
        return data;
    }
}
```

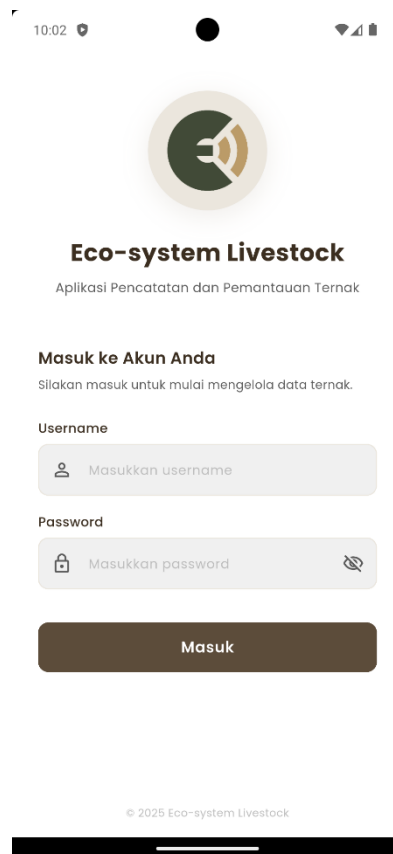
Gambar 4.1 Penggalan kode pemetaan field opsional pembeda operasi tambah dan pemutakhiran ternak

Terlihat pada Gambar 4.1 bahwa *method toJson* mengembalikan *Map<String, String>*, bukan *Map<String, dynamic>*. Tipe kembalian ini dipilih karena *request body* dikirim ke API menggunakan format *multipart form-data*, yang seluruh nilainya perlu dikirim sebagai *string*. Konversi nilai menjadi *string* dilakukan pada tahap pembuatan objek *TernakRequestModel*, yaitu ketika isian *form* disusun menjadi objek *request*, sehingga lapisan *Service* tidak perlu melakukan konversi lagi sebelum mengirim *request*. Pola penamaan *field* pada *map* hasil juga tidak selalu identik dengan nama *field* pada kelas Dart. Sebagai contoh, *field pbb* pada kelas dipetakan ke *key pertambahan_bobot_badan* pada *map*, dan *field jenisPakan* dipetakan ke *key uuid_pakan*. Hal ini terjadi karena nama *field* Dart disusun mengikuti konvensi *camelCase* yang lazim pada bahasa Dart, sementara nama *key* pada *request* API disusun mengikuti konvensi *snake_case* yang didefinisikan oleh tim *backend*. Pemetaan nama antar dua konvensi tersebut menjadi salah satu tanggung jawab lapisan *Model* pada aplikasi.

Tiga kelas yang telah dibahas menunjukkan tiga pola implementasi yang berbeda pada lapisan *Model*, yaitu pemetaan sederhana dengan sumber data tambahan dari sisi aplikasi pada *PeternakModel*, pemetaan *nested* dengan struktur kelas berjenjang pada *DetailTernakModel*, dan pemetaan dua arah dengan *field* opsional sebagai pembeda operasi pada *TernakRequestModel*. Lapisan *ViewModel* mengakses kelas-kelas model ini melalui lapisan *Service* yang akan dibahas pada sub-bab 4.1.7. Implementasi lapisan *View* pada fitur-fitur spesifik yang memanfaatkan kelas-kelas model ini dijabarkan mulai sub-bab 4.1.2 sampai sub-bab 4.1.6.

4.1.2 Implementasi Autentikasi dan Role-Based Access

Autentikasi merupakan fitur pertama yang diakses pengguna saat membuka aplikasi. Implementasi fitur ini melibatkan tiga lapisan pada arsitektur MVVM, yaitu halaman *LoginPage* pada lapisan *View*, kelas *AuthProvider* pada lapisan *ViewModel*, dan kelas *AuthService* pada lapisan akses *REST API*. Antarmuka halaman *login* disajikan pada Gambar 4.2.



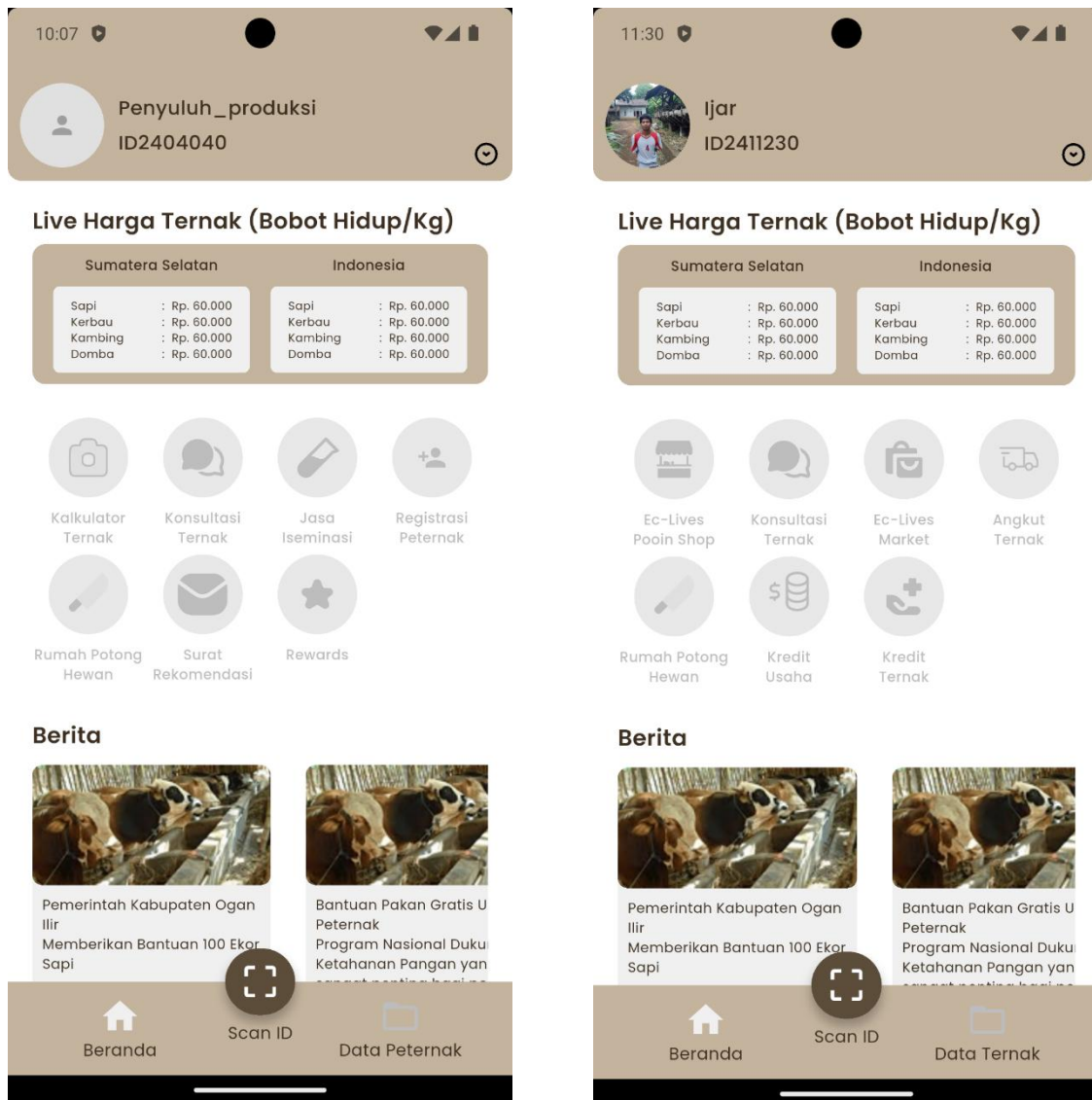
Gambar 4.2 Tampilan halaman LoginPage

Lapisan akses *REST API* pada autentikasi diwujudkan oleh kelas *AuthService* yang memanggil *endpoint /login* dengan *method* POST. *Request* dikirim dengan *header Content-Type: application/x-www-form-urlencoded* dan *body* berisi *username* dan *password* yang diisi pengguna pada halaman *login*. Apabila *response* dikembalikan dengan *status code* 200, *response body* di-parse menjadi objek *AuthModel* melalui *factory fromJson*. Tiga nilai yang diambil adalah *access token*, *role*, dan *uuid* pengguna. Pada tahap yang sama, *token* dan *role* juga disimpan secara lokal di perangkat menggunakan *package shared_preferences* agar dapat digunakan pada sesi berikutnya tanpa harus *login* ulang. *Package* ini menyediakan antarmuka *key-value* sederhana yang dibungkus ke *NSUserDefaults* pada iOS dan *SharedPreferences* pada Android (Shared Preferences, n.d.).

Lapisan *ViewModel* pada autentikasi diwujudkan oleh kelas *AuthProvider* yang meng-*extends* *ChangeNotifier*. Kelas ini memegang *state isLoading* yang dipakai oleh halaman *login* untuk menampilkan indikator pemuatan saat *request login* sedang diproses, serta menyimpan hasil *login* berupa *token*, *role*, dan *uuid* pada variabel privat. *Method login* pada *AuthProvider* memanggil *AuthService.login* di dalam blok *try-catch* untuk menjaga aplikasi tetap stabil

apabila terjadi gangguan jaringan atau galat dari sisi *server*. Pemanggilan *notifyListeners* dilakukan setiap kali *state isLoading* berubah, sehingga indikator pemuatan pada halaman *login* aktif saat *request* dikirim dan mati setelah *response* diterima.

Role pengguna yang diperoleh dari *response login* menentukan menu mana yang dapat diakses setelah aplikasi berpindah ke halaman utama. Nilai *role* disimpan di *SharedPreferences* bersamaan dengan *token* pada saat *login* berhasil, sehingga tetap tersedia selama sesi *login* aktif. Halaman *HomePage* membaca nilai *role* tersebut untuk menampilkan menu sesuai peran pengguna, yaitu menu Data Peternak untuk *role penyuluh_produksi* atau menu Data Ternak untuk *role peternak*. Tangkapan layar halaman utama untuk kedua *role* disajikan pada Gambar 4.3. Perlu dicatat bahwa pembatasan akses pada sisi aplikasi hanya berupa pengaturan tampilan menu, sedangkan validasi hak akses terhadap *endpoint REST API* tetap dilakukan di sisi *server* sebagaimana telah dijelaskan pada Bab III sub-bab 3.2.2. Aplikasi yang memanggil *endpoint* yang tidak sesuai dengan *role* akan menerima *response* berisi pesan penolakan dari *REST API*.



Gambar 4.3 Tampilan halaman HomePage berdasarkan role pengguna

Pada Gambar 4.3 terlihat bahwa *HomePage* juga menampilkan beberapa *section* tambahan di bawah area menu utama, yaitu *section* berita peternakan, materi edukasi, dan informasi harga ternak per wilayah. Ketiga *section* tersebut saat ini masih berisi konten contoh (*placeholder*) yang dimuat secara *hard-coded* dari *folder assets* aplikasi, karena *endpoint* untuk konten dinamis tersebut belum tersedia pada *REST API*. Konten *placeholder* ditampilkan untuk mempertahankan tata letak antarmuka sesuai *mockup* yang dirancang tim kolaborator dari Universitas Sriwijaya. Pengembangan konten dinamis untuk *section* tersebut menjadi salah satu saran pengembangan yang dijabarkan pada Bab V sub-bab 5.2.

Halaman *ProfilePage* merupakan halaman yang dapat diakses pengguna dari ikon profil pada *header HomePage*. Halaman ini menampilkan informasi identitas pengguna aktif yang

diambil dari *UserProvider*, mencakup nama, NIK, *role*, dan informasi pendukung lainnya yang tersedia pada *response endpoint /profile*. Pada bagian bawah halaman tersedia tombol *logout* yang memicu pemanggilan *method logout* untuk mengakhiri sesi pengguna. *Method* tersebut menghapus seluruh data sesi yang tersimpan di *SharedPreferences*, kemudian mengarahkan navigasi kembali ke *LoginPage*. Tampilan halaman *ProfilePage* beserta tombol *logout* disajikan pada Gambar 4.4



Gambar 4.4 Tampilan halaman ProfilePage dengan tombol Logout

Persistensi sesi *login* diimplementasikan pada *file main.dart* yang merupakan *entry point* aplikasi. Pada *method build* dari kelas *MyApp*, halaman *home* ditentukan melalui *FutureBuilder* yang membaca nilai *token* dari *SharedPreferences* di awal pemuatan aplikasi. Apabila *token* tersedia dan tidak kosong, aplikasi langsung membuka *MainWrapper*. Apabila tidak ditemukan, pengguna diarahkan ke *LoginPage*. Mekanisme ini menghilangkan kebutuhan *login* ulang selama *token* masih tersimpan di perangkat. Pada saat pengguna melakukan *logout* dari halaman *ProfilePage*, *method logout* memanggil *prefs.clear* sehingga seluruh nilai yang

tersimpan di *SharedPreferences*, termasuk *token*, *role*, dan *uuid*, terhapus sekaligus dan pengguna diarahkan kembali ke halaman *login*. Penggalan konfigurasi *FutureBuilder* pada *main.dart* disajikan pada Gambar 4.5.

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Eco-system Livestock',
      home: FutureBuilder<String?>(
        future: getUserData(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Scaffold(
              body: Center(child: CircularProgressIndicator()),
            );
          }

          String? token = snapshot.data;

          return (token != null && token.isNotEmpty)
            ? MainWrapper()
            : LoginPage();
        },
      ),
    );
  }

  Future<String?> getUserData() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs.getString('token');
  }
}
```

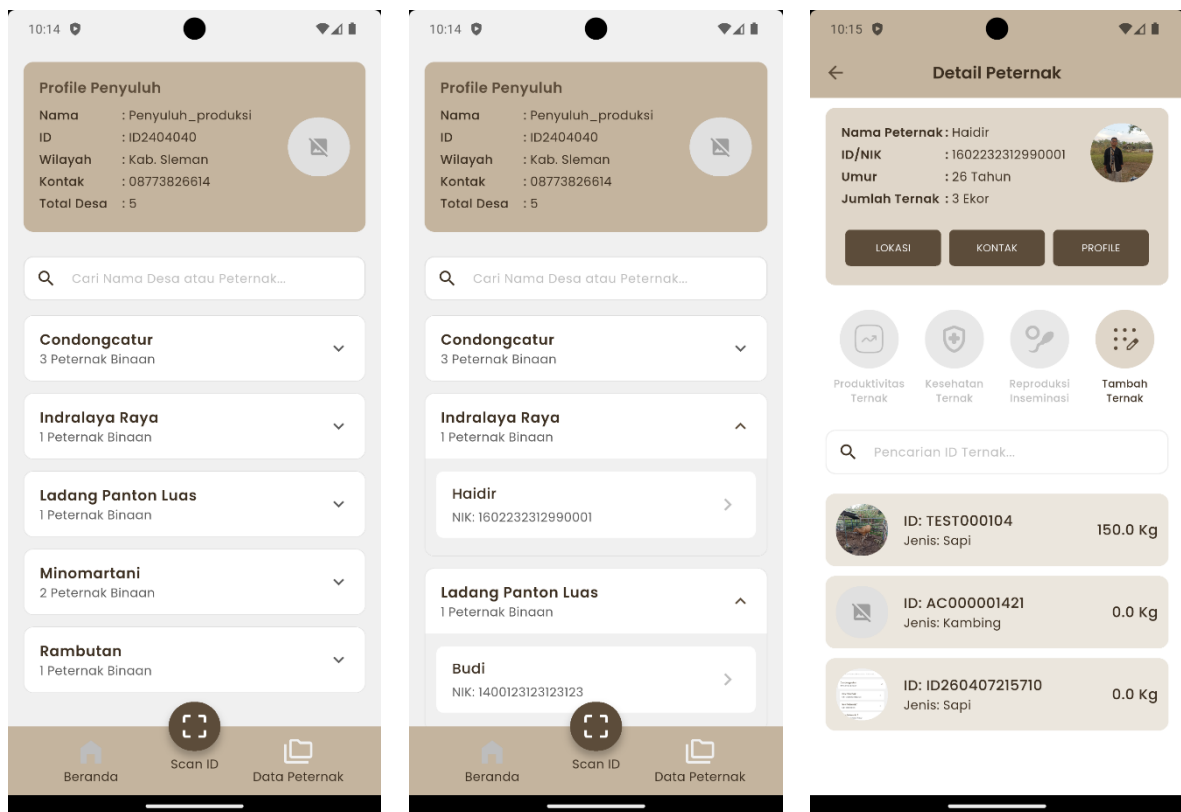
Gambar 4.5 Penggalan kode persistensi sesi login menggunakan FutureBuilder

Implementasi autentikasi menunjukkan bagaimana tiga lapisan pola MVVM bekerja sama: memproses *login*, menyimpan *token*, dan mempertahankan sesi pengguna. *View* menyediakan *form* pengisian kredensial, *ViewModel* mengelola *state* pemuatan dan pemanggilan *Service*, sedangkan *Service* bertanggung jawab atas komunikasi dengan *REST API* dan penyimpanan *token* ke *SharedPreferences*. Mekanisme *role-based access* diterapkan dalam dua lapis, yaitu pengaturan tampilan menu pada sisi aplikasi dan validasi hak akses pada sisi *REST API*. Fitur-fitur yang dapat diakses pengguna setelah *login* berhasil dijabarkan pada sub-bab selanjutnya, dimulai dari menu Data Peternak yang dikhususkan untuk *role penyuluh_produksi*.

Aplikasi tidak menerapkan mekanisme *refresh token* otomatis. Token autentikasi yang diperoleh saat *login* disimpan secara lokal pada *SharedPreferences* sehingga sesi pengguna tetap bertahan selama token masih berlaku, termasuk ketika aplikasi ditutup lalu dibuka kembali. Ketika token tidak lagi berlaku, permintaan berikutnya ke *REST API* menerima *response 401 Unauthorized* dan aplikasi secara otomatis mengakhiri sesi serta mengarahkan pengguna kembali ke halaman *login* untuk masuk ulang. Aplikasi sengaja tidak menyimpan kredensial pengguna (nama pengguna dan kata sandi) pada perangkat demi keamanan; hanya token sesi yang disimpan dan token tersebut dihapus saat pengguna *logout*. Penambahan mekanisme *refresh token* untuk memperpanjang sesi tanpa *login* ulang memerlukan dukungan dari sisi *REST API* dan dijadikan saran pengembangan pada Bab V.

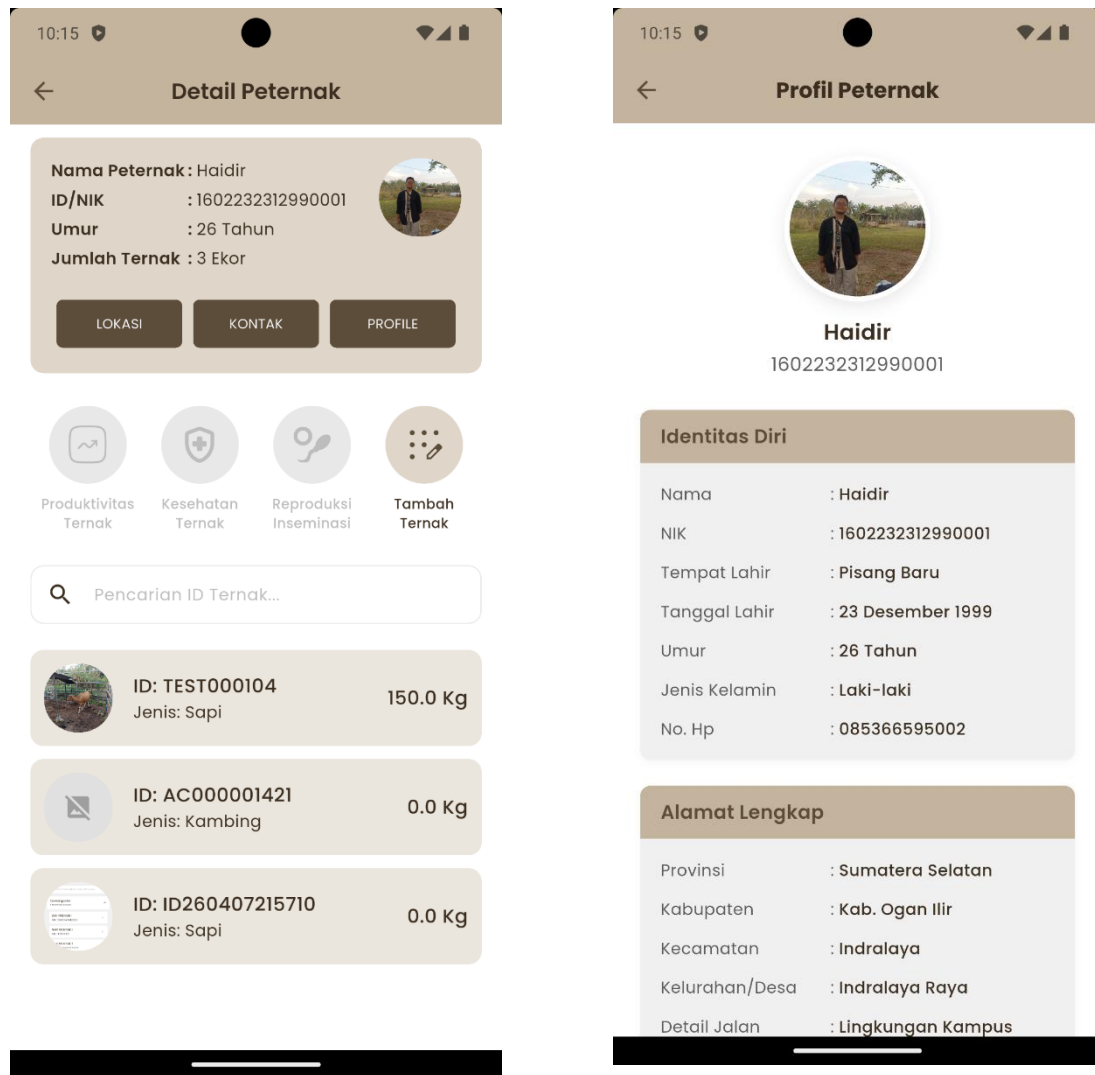
4.1.3 Implementasi Menu Data Peternak

Menu Data Peternak merupakan fitur utama yang diakses oleh pengguna dengan *role penyuluh_produksi*. Menu ini mengikuti alur berjenjang: dari daftar desa binaan ke daftar peternak pada desa tertentu, kemudian ke detail peternak yang berisi daftar ternak miliknya, dilengkapi *form* tambah ternak. Alur navigasi tersebut disajikan pada Gambar 4.6.



Gambar 4.6 Alur navigasi menu Data Peternak

Pada halaman detail peternak, bagian *header* menampilkan informasi profil peternak dilengkapi dengan tiga tombol aksi cepat untuk mengakses informasi peternak yang lebih detail, yaitu tombol LOKASI, KONTAK, dan PROFILE. Tombol LOKASI memanfaatkan *package url_launcher* untuk membuka aplikasi peta dengan koordinat lokasi peternak yang tersimpan pada basis data, sehingga penyuluh dapat langsung memperoleh petunjuk arah ke lokasi peternak. Tombol KONTAK juga memanfaatkan *url_launcher* untuk membuka aplikasi *WhatsApp* dengan nomor telepon peternak yang sudah diformat, sehingga penyuluh dapat langsung memulai percakapan tanpa harus menyalin nomor secara manual. Tombol PROFILE membuka halaman *ProfilePeternakPage* yang menampilkan informasi peternak secara lebih lengkap dibanding ringkasan pada halaman detail. Tampilan *header* dengan tiga tombol tersebut beserta halaman *ProfilePeternakPage* yang dibuka melalui tombol PROFILE disajikan pada Gambar 4.7.



Gambar 4.7 Tampilan header profil peternak dengan tombol LOKASI, KONTAK, dan PROFILE

Pengambilan data desa dan peternak pada menu ini diimplementasikan dengan pola pemanggilan *endpoint* paralel untuk efisiensi waktu muat. Lapisan *PeternakService* memanggil *endpoint /fostered-breeders* untuk memperoleh daftar desa, lalu melakukan pemanggilan ke *endpoint /fostered-breeders/each-village* untuk setiap desa secara bersamaan menggunakan *Future.wait*. Hasil pemanggilan tersebut digabungkan menjadi sebuah *Map* yang memetakan objek *WilayahModel* ke daftar *PeternakModel* yang termasuk di dalamnya. Pola ini menghindari pemanggilan berurutan yang memperlama waktu pemuatan ketika jumlah desa cukup banyak. Penggalan *method getMasterDataPeternak* pada *PeternakService* disajikan pada Gambar 4.8.

```

Future<Map<WilayahModel, List<PeternakModel>>> getMasterDataPeternak(
    String token) async {
  try {
    Map<WilayahModel, List<PeternakModel>> masterData = {};

    // Hit API 1: ambil daftar wilayah/desa
    final responseWilayah = await http.get(
      Uri.parse('$baseUrl/fostered-breeders?page=1&limit=-1'),
      headers: {
        'Authorization': 'Bearer $token',
        'Accept': 'application/json',
      },
    );

    if (responseWilayah.statusCode == 200) {
      final body = jsonDecode(responseWilayah.body);
      final List<dynamic> dataWilayah = body['data'] ?? [];

      List<Future<void>> futures = [];

      // Hit API 2: ambil peternak per desa SECARA PARALEL
      for (var item in dataWilayah) {
        WilayahModel wilayah = WilayahModel.fromJson(item);
        masterData[wilayah] = [];

        futures.add(
          http.get(
            Uri.parse('$baseUrl/fostered-breeders/each-village'
              '?uuid=${wilayah.uuid}&page=1&limit=-1'),
            headers: {'Authorization': 'Bearer $token'},
          ).then((responsePeternak) {
            if (responsePeternak.statusCode == 200) {
              final bodyP = jsonDecode(responsePeternak.body);
              final List<dynamic> listJson =
                bodyP['data']?['peternak'] ?? [];

              masterData[wilayah] = listJson
                .map((j) => PeternakModel.fromJson(

```

```

        j, wilayah.namaWilayah))
        .toList();

    }

    }),

    );

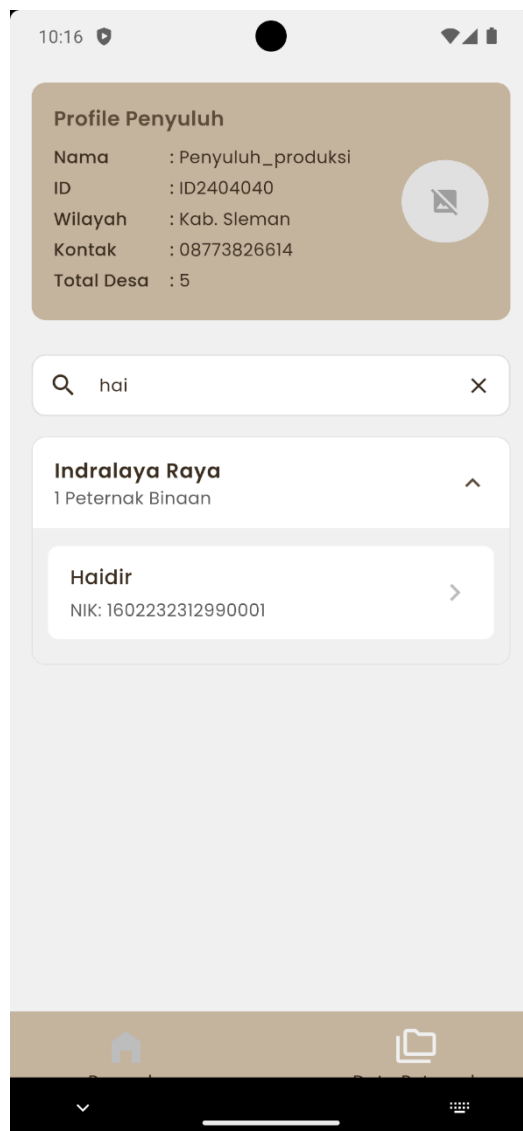
}

// Tunggu seluruh task paralel selesai
await Future.wait(futures);
return masterData;
} else {
    throw Exception('Gagal memuat data wilayah');
}
} catch (e) {
    throw Exception('Terjadi kesalahan Service: $e');
}
}

```

Gambar 4.8 Penggalan kode pemanggilan paralel endpoint master peternak

Lapisan *ViewModel* pada menu ini diwujudkan oleh *PeternakProvider* yang memegang dua *state* berbeda, yaitu *_masterData* sebagai salinan utuh hasil pemanggilan *Service* dan *_displayData* yang digunakan halaman untuk menampilkan isi daftar. Pemisahan ini memungkinkan fitur pencarian lokal diimplementasikan tanpa pemanggilan ulang ke *REST API*. Ketika pengguna mengetik kata kunci pada kolom pencarian, *method search* pada *PeternakProvider* menyaring isi *_masterData* secara lokal berdasarkan kecocokan nama desa atau nama peternak, kemudian menugaskan hasilnya ke *_displayData* dan memanggil *notifyListeners*. Apabila kata kunci dikosongkan kembali, *_displayData* diisi ulang dengan salinan penuh dari *_masterData*. Tampilan hasil pencarian pada *search bar* disajikan pada Gambar 4.9, dan penggalan *method search* pada *PeternakProvider* disajikan pada Gambar 4.10.



Gambar 4.9 Tampilan hasil pencarian pada search bar menu Data Peternak

```
void search(String query) {
    if (query.isEmpty()) {
        _displayData = Map.from(_masterData);
    } else {
        Map<WilayahModel, List<PeternakModel>> filteredData = {};
        String lowerQuery = query.toLowerCase();

        _masterData.forEach((wilayah, listPeternak) {
            bool isDesaMatch =
                wilayah.namaWilayah.toLowerCase().contains(lowerQuery);
```

```

List<PeternakModel> filteredPeternak =
    listPeternak.where((peternak) {
        return peternak.nama.toLowerCase().contains(lowerQuery);
    }).toList();

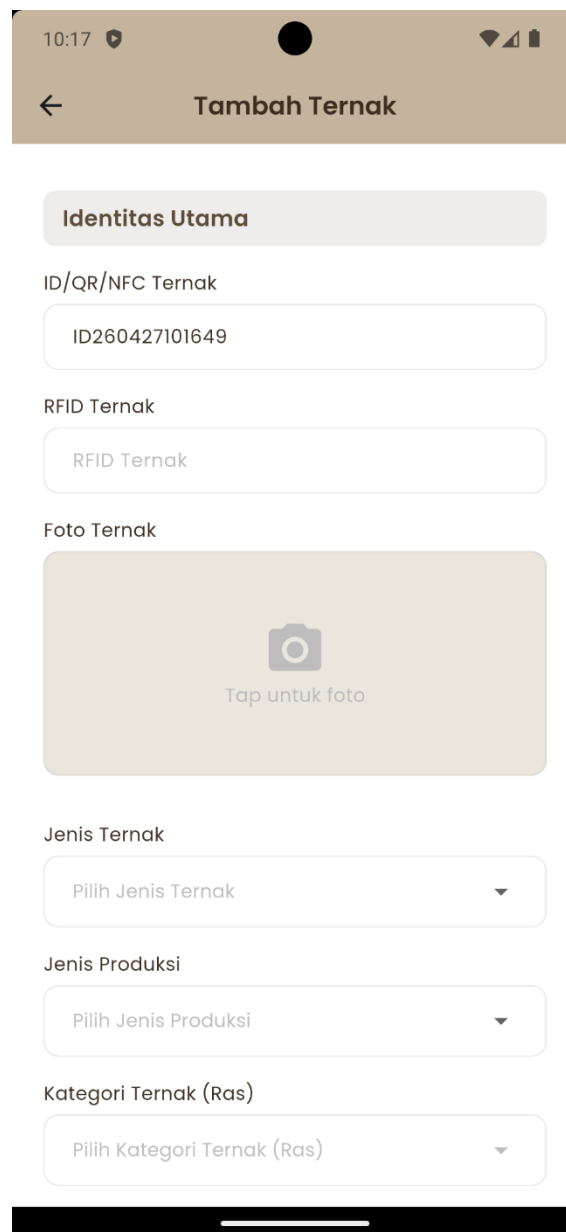
if (isDesaMatch || filteredPeternak.isNotEmpty) {
    filteredData[wilayah] =
        isDesaMatch ? listPeternak : filteredPeternak;
}
});

_displayData = filteredData;
}
notifyListeners();
}

```

Gambar 4.10 Penggalan kode pencarian lokal data peternak

Fitur tambah ternak baru diakses dari halaman detail peternak melalui tombol tambah yang membuka halaman *TambahTernakPage*. Halaman tersebut memuat *form* dengan empat kelompok isian yang mencerminkan struktur *TernakRequestModel*, yaitu identitas, produktivitas, kesehatan, dan reproduksi. Beberapa isian *form* berupa *dropdown* yang opsinya diambil dari *endpoint* master data melalui *method loadMasterDropdown* pada *TernakProvider*. *Method* tersebut memanggil enam *endpoint* master data secara paralel dengan pola *Future.wait* yang sama seperti pada *PeternakService*. Tampilan *form* tambah ternak secara keseluruhan disajikan pada Gambar 4.11.



Gambar 4.11 Tampilan halaman TambahTernakPage

Pada *form* tambah ternak diterapkan pola *cascading dropdown* untuk *field* yang saling bergantung secara hierarkis. Khusus pada *dropdown* kategori ternak, opsi hanya dimuat setelah pengguna memilih jenis hewan dan jenis produksi, karena *endpoint* */type-of-livestock* membutuhkan dua parameter tersebut. Pola *cascading dropdown* ini diterapkan di *method* *setJenisHewan* dan *setJenisProduksi* pada *TernakProvider* yang memicu pemuatan ulang *dropdown* kategori setiap kali jenis hewan atau jenis produksi diubah. Tampilan *cascading dropdown* jenis hewan dan kategori ras disajikan pada Gambar 4.12.

10:18

← Tambah Ternak

Jenis Ternak

Sapi

Jenis Produksi

Potong

Kategori Ternak (Ras)

Pilih Kategori Ternak (Ras)

Simental

Peranakan Ongole (PO)

Brahman

Lokal

Bali

Limousin

Produktivitas

Bobot Ternak (kg)

Bobot Ternak (kg)

Gambar 4.12 Tampilan cascading dropdown jenis hewan dan kategori ras

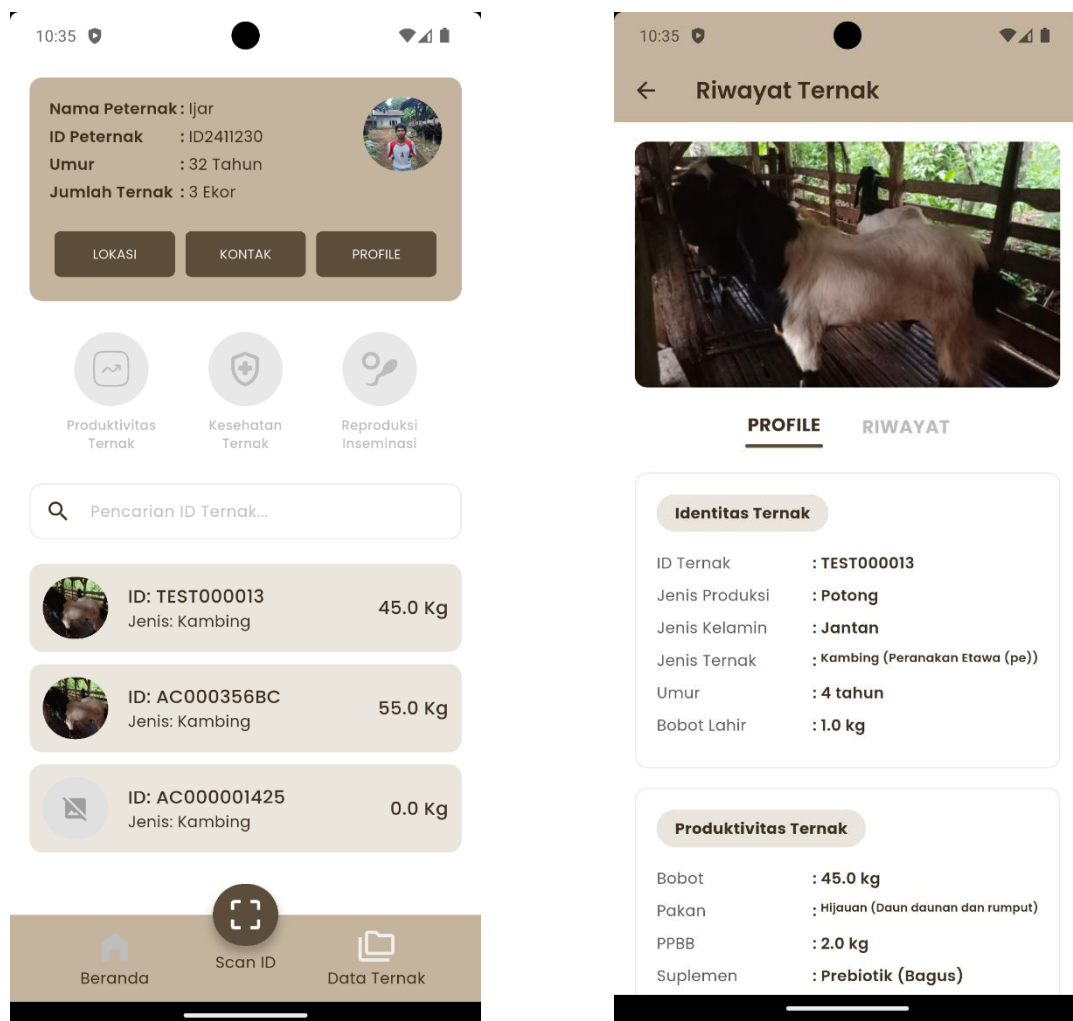
Selain *cascading dropdown*, *form* tambah ternak juga menerapkan logika kondisional pada *section* reproduksi berdasarkan jenis kelamin yang dipilih. Ketika pengguna memilih jenis kelamin Jantan pada *section* identitas, *method setJenisKelamin* pada *TernakProvider* secara otomatis menyembunyikan *section* reproduksi karena *field* status reproduksi, jumlah kelahiran, dan siklus birahi tidak relevan untuk ternak jantan. Sebaliknya, ketika jenis kelamin Betina dipilih, *section* reproduksi ditampilkan dan dapat diisi pengguna. Logika ini menjaga konsistensi data agar *field* yang tidak relevan tidak terisi nilai yang menyesatkan. Perilaku ini sesuai dengan skenario TC-20 pada Tabel 4.1. Perbandingan tampilan *form* tambah ternak berdasarkan jenis kelamin disajikan pada Gambar 4.13.

Gambar 4.13 Perbandingan tampilan form tambah ternak berdasarkan jenis kelamin: (a) Jantan tanpa section reproduksi, (b) Betina dengan section reproduksi

Pengisian *form* tambah ternak diakhiri dengan pemanggilan *method* `submitTambahTernak` pada `TernakProvider`. *Method* tersebut membungkus seluruh isian ke dalam objek `TernakRequestModel` tanpa `uuidTernak`, kemudian meneruskannya ke `TernakService.tambahTernak` untuk dikirim sebagai *request* POST ke *endpoint* `/livestock`. Detail format *request multipart form-data* dan pola error handling dijabarkan pada sub-bab 4.1.7. Setelah pengiriman berhasil, *Provider* memanggil `loadHewanTernakData` untuk menyegarkan daftar ternak pada halaman detail peternak, sehingga ternak baru langsung muncul.

4.1.4 Implementasi Menu Data Ternak

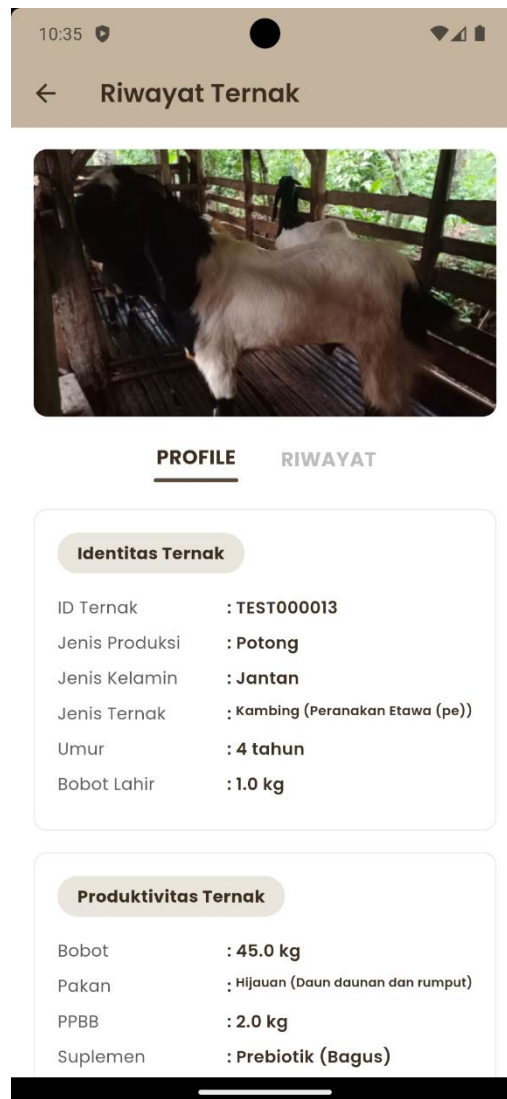
Menu Data Ternak merupakan fitur yang diakses oleh pengguna dengan *role peternak*. Berbeda dengan menu Data Peternak yang memiliki alur berjenjang dari desa ke peternak, menu ini menampilkan langsung daftar ternak milik pengguna yang sedang *login*, kemudian dapat dilanjutkan ke halaman riwayat ternak untuk melihat detail dan catatan riwayat ternak tertentu. Alur navigasi menu ini disajikan pada Gambar 4.14.



Gambar 4.14 Alur navigasi menu Data Ternak

Setelah pengguna memilih salah satu kartu ternak, aplikasi membuka halaman *RiwayatTernakPage* yang berisi tiga tab navigasi, yaitu PROFILE, RIWAYAT, dan PERBARUI. Tab pertama yang aktif secara *default* adalah tab PROFILE yang menampilkan informasi identitas ternak terkini. Informasi yang ditampilkan mencakup ID ternak, jenis hewan, jenis kelamin, tanggal lahir, bobot lahir, foto ternak, beserta data identifikasi tambahan

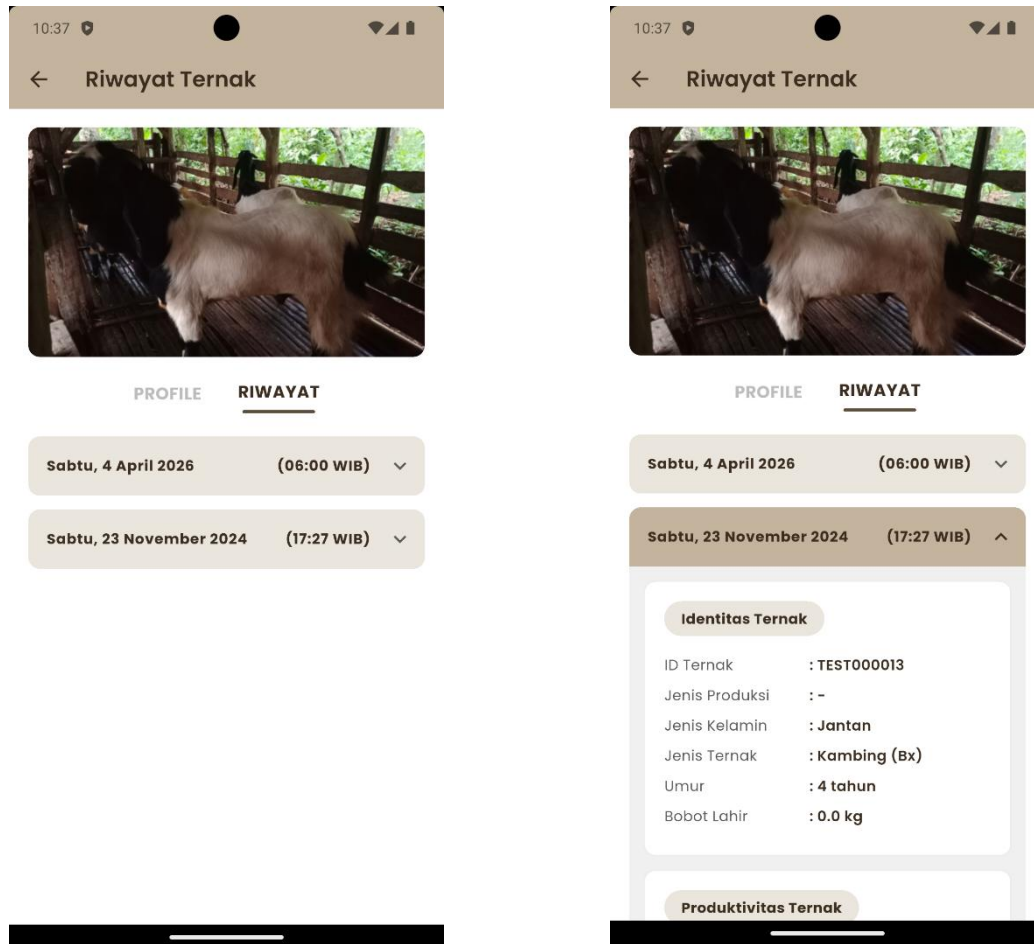
seperti NFC dan RFID apabila tersedia. Data pada tab ini diperoleh dari *response endpoint /livestock/:id* yang dipanggil melalui *method loadDetailTernak* pada *TernakProvider*. Tampilan tab PROFILE disajikan pada Gambar 4.15.



Gambar 4.15 Tampilan tab PROFILE pada halaman RiwayatTernakPage

Tab RIWAYAT menampilkan daftar entri pemutakhiran ternak yang tersusun secara kronologis dari yang terbaru ke yang terlama. Setiap entri secara *default* tampil dalam kondisi terkuncup yang hanya menampilkan tanggal dan waktu pemutakhiran, dengan kemampuan *expand* untuk menampilkan *snapshot* lengkap data ternak pada waktu itu. Pendekatan ini memungkinkan pengguna menelusuri perubahan bobot, status kesehatan, status reproduksi, dan data lainnya dari waktu ke waktu tanpa kehilangan jejak nilai sebelumnya. Data riwayat

diperoleh dari *response endpoint /livestock-history* yang dipanggil melalui *method loadRiwayat* pada *TernakProvider*. Tampilan tab RIWAYAT pada kedua kondisi disajikan pada Gambar 4.16.



Gambar 4.16 Tampilan tab RIWAYAT pada halaman RiwayatTernakPage: (a) seluruh entri terkuncup, (b) salah satu entri mekar dengan snapshot lengkap

Secara implementasi pada sisi *mobile client*, menu Data Ternak memanggil *endpoint* yang sama dengan yang digunakan pada menu Data Peternak untuk daftar ternak, yaitu */livestock* dengan parameter *uuid_peternak*. Perbedaannya terletak pada nilai *uuid_peternak* yang dikirim. Pada menu Data Peternak, nilai tersebut adalah *uuid* peternak yang sedang dilihat detailnya oleh penyuluh. Pada menu Data Ternak, nilai tersebut adalah *uuid* pengguna yang sedang *login* yang diperoleh dari *UserProvider*. Pengambilan *uuid_peternak* pengguna aktif dilakukan di *method _loadAllData* pada *MainWrapper* saat aplikasi selesai memuat halaman utama, sebagaimana telah dijelaskan pada sub-bab 4.1.2.

Pada sisi aplikasi, nilai *uuid_peternak* yang dikirim selalu merupakan *uuid* pengguna yang sedang *login* yang diperoleh dari *UserProvider*, sehingga melalui aplikasi peternak hanya mengakses ternak miliknya. Namun, pengujian *API integration* (Tabel 4.2, AT-23) menemukan bahwa *endpoint* pembacaan */livestock* tidak membatasi nilai *uuid_peternak* terhadap pemilik *token*, sehingga permintaan dengan *uuid_peternak* pengguna lain tetap dilayani. Penegakan isolasi kepemilikan pada *endpoint* pembacaan berada di sisi *REST API* yang dikembangkan tim eksternal dan menjadi saran pengembangan pada Bab V. Sub-bab 3.2.1 Bab III sudah menjelaskan perbedaan KF-08 dan KF-13 dalam konteks ini. Setelah pengguna memilih salah satu ternak pada daftar, aplikasi melakukan dua pemanggilan secara berurutan ke *endpoint /livestock/:id* untuk detail dan */livestock-history* untuk daftar riwayat. Kedua pemanggilan tersebut diwakili oleh *method loadDetailTernak* dan *loadRiwayat* pada *TernakProvider*. Hasilnya kemudian ditampilkan pada tab PROFILE dan tab RIWAYAT masing-masing.

Menu Data Ternak memanfaatkan sebagian besar lapisan yang sama dengan menu Data Peternak, khususnya *TernakProvider* dan *TernakService*. Perbedaan fungsional antara kedua menu ditentukan oleh *role* pengguna yang sedang *login* dan oleh penegakan hak akses pada sisi *REST API*, bukan oleh duplikasi kode pada sisi aplikasi.

4.1.5 Implementasi Fitur Pemutakhiran Riwayat Ternak

Fitur pemutakhiran riwayat ternak merupakan fitur yang disediakan khusus untuk *role penyuluh_produksi* agar dapat mencatat perubahan data ternak binaan dari waktu ke waktu. Fitur ini diakses melalui tab PERBARUI pada halaman *RiwayatTernakPage* yang telah disajikan pada Gambar 4.14(b). Tab PERBARUI berisi *form* pemutakhiran yang strukturnya identik dengan *form* tambah ternak, yaitu mencakup empat kelompok isian berupa identitas, produktivitas, kesehatan, dan reproduksi ternak. Tampilan tab PERBARUI disajikan pada Gambar 4.17.

Gambar 4.17 Tampilan tab PERBARUI pada halaman RiwayatTernakPage

Pengisian awal *form* pemutakhiran dilakukan secara otomatis oleh aplikasi melalui *method* *siapkanDataUpdate* pada *TernakProvider*. *Method* ini mengambil entri riwayat terbaru dari *_listRiwayat* yang sudah dimuat sebelumnya pada tab RIWAYAT, kemudian mengisi *TextEditingController* masing-masing *field form* dengan nilai-nilai dari entri tersebut. Pada *field* yang berbentuk *dropdown*, nilai yang diisi adalah *uuid* opsi yang cocok dengan nama pada riwayat, bukan nama opsinya. Pemetaan nama ke *uuid* dilakukan melalui fungsi pembantu yang mencari *uuid* pada daftar opsi *dropdown* master data yang sudah dimuat. Pola pengisian awal ini mempercepat proses pemutakhiran karena penyuluh hanya perlu mengubah *field* yang memang berubah tanpa harus mengisi ulang seluruh *form*.

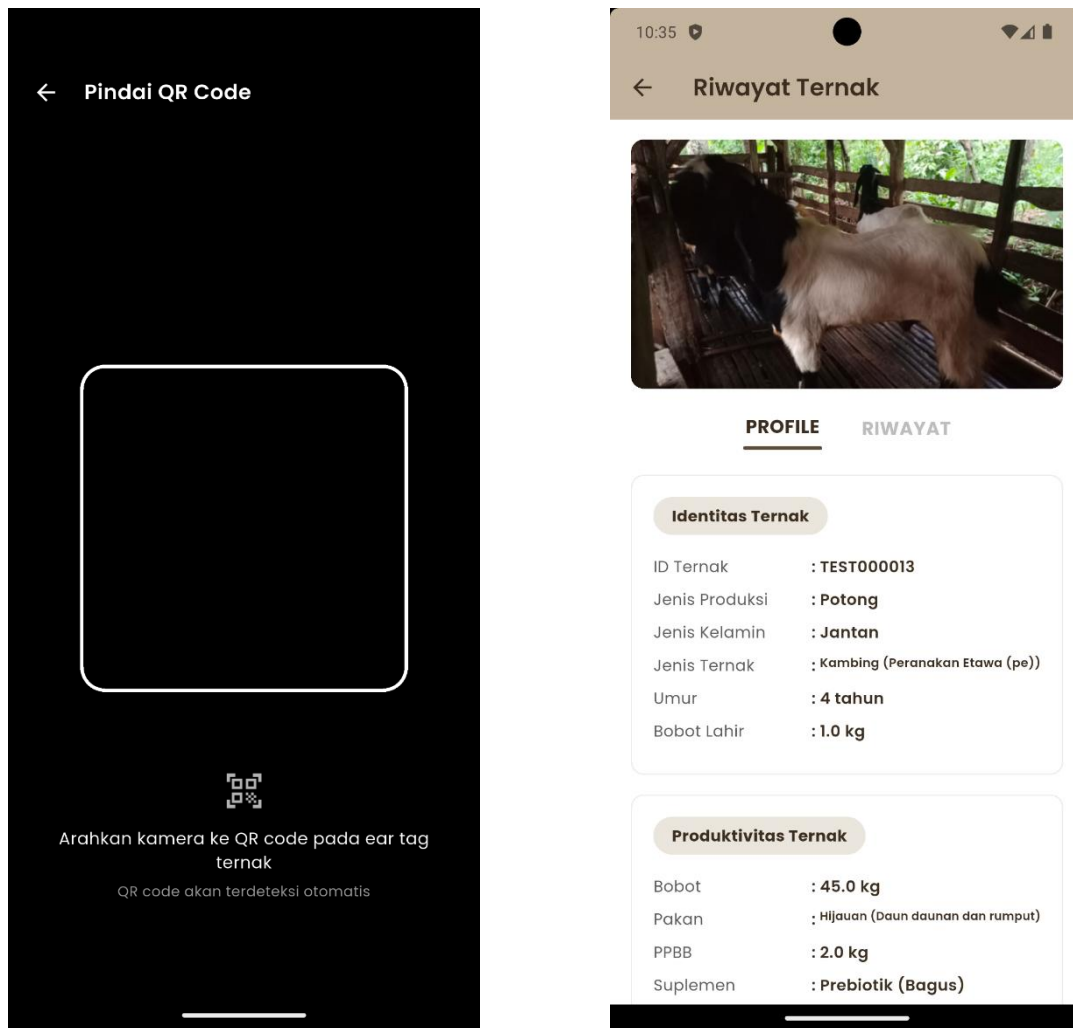
Setelah penyuluh menyelesaikan pengisian *form*, proses pengiriman dilakukan melalui *method* *submitUpdateTernak* pada *TernakProvider*. *Method* ini membungkus isian ke dalam objek *TernakRequestModel* beserta *uuidTernak* yang sedang diperbarui, lalu meneruskannya

ke *TernakService.perbaruiTernak*. Kehadiran *uuidTernak* inilah yang membedakan operasi pemutakhiran dari operasi tambah ternak pada *method toJson*, sesuai dengan perancangan kelas model yang dijelaskan pada Bab III sub-bab 3.3.4 dan diimplementasikan pada sub-bab 4.1.1. *TernakService.perbaruiTernak* kemudian mengirim *request* dengan *method* PUT ke *endpoint /livestock* dalam format *multipart form-data*, dengan *uuid_ternak* disertakan pada *body* sebagai penanda operasi pemutakhiran.

Apabila pengiriman *request* pemutakhiran berhasil dengan *status code* 200 atau 201, *TernakProvider* memanggil ulang *loadDetailTernak* dan *loadRiwayat* sehingga tampilan pada tab PROFILE dan tab RIWAYAT ikut tersegarkan dengan data pemutakhiran terbaru. Dengan demikian, penyuluh dapat langsung melihat hasil pemutakhiran tanpa perlu keluar dan masuk kembali ke halaman tersebut. Penyegaran otomatis ini terjadi karena tab PROFILE dan tab RIWAYAT mendengarkan *state* yang sama dari *TernakProvider*, sehingga pembaruan *state* setelah *request* PUT berhasil langsung memicu *rebuild* pada kedua tab.

4.1.6 Implementasi Pemindai QR Code

Pemindai QR *code* merupakan fitur universal yang dapat diakses oleh kedua *role* pengguna melalui *Floating Action Button* (FAB) berlabel 'Scan ID' di tengah *bottom navigation bar*. Fitur ini memanfaatkan kamera perangkat *mobile* untuk membaca QR *code* pada *ear tag* ternak, mengekstrak ID ternak dari QR *code* tersebut, kemudian menampilkan detail ternak yang bersangkutan. Fitur ini diimplementasikan menggunakan *package mobile_scanner* yang telah dibahas pada Bab II sub-bab 2.2.8 (Steenbakker, n.d.). Alur penggunaan fitur ini disajikan pada Gambar 4.18.



Gambar 4.18 Alur penggunaan fitur pemindai QR code

Pemicu fitur ini berada di *widget MainWrapper* yang juga bertanggung jawab atas *Scaffold* utama aplikasi. FAB di tengah *bottom navigation* dikonfigurasi dengan ikon berbentuk bingkai pemindai dan label 'Scan ID'. Ketika pengguna menekan FAB tersebut, aplikasi berpindah ke halaman pemindai yang memuat *widget MobileScanner* dari *package mobile_scanner*. *Widget MobileScanner* mengelola siklus kamera perangkat secara otomatis, mencakup pengajuan izin kamera, pengaktifan *preview* kamera, dan proses deteksi *QR code* secara berkelanjutan. Ketika *QR code* terdeteksi dan ID berhasil diekstrak, *handler onDetect* pada *widget* dipicu untuk melanjutkan alur pemanggilan ke *REST API*.

Pemanggilan *REST API* pasca pemindaian menggunakan *endpoint /livestock/scan/:id* dengan *method* GET. Nilai *:id* pada *endpoint* tersebut adalah ID ternak 14 karakter yang dipindai dari *QR code*. *Endpoint* ini mengembalikan detail ternak yang sama dengan yang dikembalikan */livestock/:id* pada menu Data Peternak, sehingga *response*-nya dapat dipetakan

ke *DetailTernakModel* yang sudah ada. Setelah *response* diterima dan di-parse, aplikasi menutup halaman pemindai dan membuka *RiwayatTernakPage* dengan *uuid* ternak hasil pindai sebagai parameter. Apabila ID yang dipindai tidak ditemukan di basis data, *REST API* mengembalikan response gagal dan aplikasi menampilkan pesan pemberitahuan pada halaman pemindai. Sesuai temuan pada Tabel 4.2, pemindaian tidak membatasi akses berdasarkan kepemilikan ternak; setiap pengguna terautentikasi dapat memindai dan melihat data ternak selama ID-nya terdaftar. Penggalan *widget MobileScanner* beserta *handler onDetect*-nya disajikan pada Gambar 4.19.

```
import 'package:flutter/material.dart';
import 'package:mobile_scanner/mobile_scanner.dart';
import 'package:provider/provider.dart';

class ScanQRPage extends StatefulWidget {
  const ScanQRPage({super.key});

  @override
  State<ScanQRPage> createState() => _ScanQRPageState();
}

class _ScanQRPageState extends State<ScanQRPage> {
  bool _isProcessing = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Pindai QR Code')),
      body: MobileScanner(
        controller: MobileScannerController(
          formats: const [BarcodeFormat.qrCode],
        ),
        onDetect: (BarcodeCapture capture) async {
          if (_isProcessing) return;

          final String? idTernak =
            capture.barcodes.first.rawValue;
          if (idTernak == null) return;
```

```

        setState(() => _isProcessing = true);

        final token = context.read<AuthProvider>().token;
        final detail = await context
            .read<TernakProvider>()
            .loadDetailTernakByScan(idTernak, token);

        if (!mounted) return;

        if (detail != null) {
            Navigator.pushReplacement(
                context,
                MaterialPageRoute(
                    builder: (_) => RiwayatTernakPage(
                        ternakId: detail.uuid,
                    ),
                ),
            );
        } else {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                    content: Text('Ternak tidak ditemukan'),
                ),
            );
            setState(() => _isProcessing = false);
        }
    },
),
);
}
}

```

Gambar 4.19 Penggalan kode widget MobileScanner pada halaman pemindai QR code

Fitur pemindai QR *code* menghubungkan tiga komponen: *widget MobileScanner* di sisi *View*, *method* pemanggilan *endpoint /livestock/scan/:id* di lapisan *Service*, dan *TernakProvider* sebagai penghubung keduanya. Pemanfaatan kamera perangkat menggantikan pengetikan manual ID ternak sepanjang 14 karakter, sehingga waktu identifikasi di lapangan lebih singkat

dan potensi salah ketik berkurang. Manfaat ini sejalan dengan motivasi pengembangan aplikasi yang dipaparkan pada Bab 1 latar belakang.

4.1.7 Implementasi Akses *REST API*

Sub-bab ini merangkum pola implementasi akses *REST API* yang diterapkan secara konsisten di seluruh kelas *Service*. Komunikasi HTTP dilakukan melalui *package http* dari *pub.dev* yang menyediakan *method* GET, POST, PUT, dan DELETE, beserta *MultipartRequest* untuk pengiriman berkas (HTTP, n.d.).

Pola dasar pemanggilan *endpoint* pada lapisan *Service* mengikuti urutan yang seragam. *Method Service* menerima *token* dan parameter lain yang dibutuhkan, menyusun URL lengkap termasuk *query parameter* bila ada, memanggil *http.get* atau *http.post* dengan *header* yang sesuai, lalu memeriksa *status code response*. Apabila *status code* sama dengan 200 untuk GET atau 201 untuk POST, *response body* di-parse dengan *jsonDecode*, *key data* diambil dari *Map* hasil *parsing*, lalu dipetakan ke kelas model yang sesuai melalui *factory fromJson*. Apabila *status code* tidak sesuai harapan, *method* melempar *Exception* dengan pesan yang informatif atau mengembalikan nilai *null* agar lapisan *Provider* dapat memperlakukan kasus gagal tanpa aplikasi mengalami *crash*.

Pemanggilan *endpoint* yang memerlukan pengiriman berkas gambar, yaitu tambah ternak dan pemutakhiran ternak, menggunakan *MultipartRequest* dari *package http*. Pola yang diterapkan aplikasi adalah pengiriman seluruh *field* teks melalui *request.fields*, sedangkan berkas gambar dibaca dari *path* perangkat menggunakan *File.readAsBytes*, dikonversi ke *string* dengan *base64Encode*, lalu dikirim sebagai *field* teks bernama *foto* dengan awalan *data:image/jpeg;base64,*. Pengiriman gambar sebagai *base64 string* pada *field* teks dipilih sesuai kontrak *backend* yang menerima *payload* dalam bentuk demikian. *Method send* pada *MultipartRequest* mengembalikan *StreamedResponse* yang kemudian dikonversi menjadi *Response* biasa melalui *http.Response.fromStream*, sehingga *status code* dan *body* dapat diperiksa dengan pola yang sama seperti pada pemanggilan GET sederhana.

Penanganan galat pada aplikasi dilakukan dalam dua lapis. Lapis pertama berada pada lapisan *Service* yang membungkus pemanggilan *http.get* atau *http.post* ke dalam blok *try-catch*, melemparkan *Exception* dengan pesan yang deskriptif apabila *status code* tidak sesuai atau terjadi kegagalan koneksi. Lapis kedua berada pada lapisan *Provider* yang memanggil *Service* di dalam blok *try-catch-finally* miliknya sendiri, mengisi *state _errorMessage* dengan pesan kesalahan apabila *Exception* dilempar, lalu memanggil *notifyListeners* agar halaman dapat

menampilkan *Snackbar* atau indikator kegagalan kepada pengguna. Blok *finally* pada lapisan *Provider* selalu mengembalikan *state _isLoading* ke *false* apapun hasil pemanggilan, sehingga indikator pemuatan tidak tertinggal aktif. Pola dua lapis ini memungkinkan pesan kesalahan tetap sampai ke pengguna dalam bentuk yang bersahabat, sekaligus menjaga *state* aplikasi tetap konsisten meski *request* ke *REST API* gagal.

Header yang disertakan pada setiap *request* mengikuti dua kebutuhan utama, yaitu autentikasi dan penentuan format *payload*. *Header Authorization* dengan nilai *Bearer* diikuti *token* selalu disertakan pada seluruh *request* kecuali *endpoint /login* karena pada tahap *login* aplikasi belum memiliki *token*. *Header Content-Type* menyesuaikan dengan jenis *payload* yang dikirim, yaitu *application/x-www-form-urlencoded* pada *request login*, *application/json* pada *request GET* yang mengharapkan *response* dalam format JSON, dan *multipart/form-data* secara otomatis diatur oleh *MultipartRequest* pada *request* tambah serta pemutakhiran ternak. Konsistensi konfigurasi *header* ini penting karena menjadi bagian dari spesifikasi *request* yang divalidasi oleh *REST API*.

Akses *REST API* pada aplikasi Eco-system Livestock mengikuti pola yang seragam di seluruh kelas *Service*: konstruksi URL, konfigurasi *header*, penanganan *response*, dan error handling dua lapis. Keseragaman pola ini menyederhanakan proses pengembangan, mempermudah proses pemeliharaan kode, dan menjadi dasar bagi pengujian *API integration* yang hasilnya akan dibahas pada sub-bab 4.2.2.

4.2 Hasil Pengujian

Sub-bab ini memaparkan hasil pengujian yang dilakukan terhadap aplikasi dan *REST API* yang menjadi pendukungnya. Tiga metode pengujian yang diterapkan adalah *black-box testing* untuk memverifikasi perilaku fungsional aplikasi dari sisi pengguna, *API integration testing* untuk memverifikasi integrasi aplikasi dengan *REST API* melalui pengujian langsung pada *endpoint*, dan pengujian *usability* dengan SUS untuk mengukur keterterimaan aplikasi. Ketiga metode tersebut telah dirancang pada Bab III sub-bab 3.4, dengan daftar skenario yang diturunkan dari kebutuhan fungsional yang tercantum pada Tabel 3.2.

4.2.1 Hasil Black-box Testing

Pengujian *black-box* dilakukan pada aplikasi yang telah terpasang pada perangkat Android untuk memverifikasi bahwa seluruh fitur berjalan sesuai perilaku yang diharapkan. Pengujian dirancang dengan menurunkan skenario dari setiap kebutuhan fungsional pada Tabel

3.2, mencakup skenario sukses dan skenario gagal untuk masing-masing fitur. Total terdapat 31 skenario pengujian yang disusun dengan kode TC-01 sampai TC-31. Rincian hasil pengujian *black-box* disajikan pada Tabel 4.1.

Tabel 4.1 Hasil *Black-box Testing* Aplikasi Eco-system Livestock

Kode TC	KF	Nama Skenario	Langkah Pengujian	Hasil yang Diharapkan	Status
TC-01	KF-01	Login dengan kredensial valid	1. Buka aplikasi. 2. Masukkan username dan password yang benar. 3. Tekan tombol Masuk.	Aplikasi berpindah ke HomePage sesuai role pengguna.	Valid
TC-02	KF-01	Login dengan password salah	1. Buka aplikasi. 2. Masukkan username benar dan password salah. 3. Tekan tombol Masuk.	Aplikasi tetap pada LoginPage dan menampilkan pesan kesalahan kredensial.	Valid
TC-03	KF-01	Login tanpa koneksi internet	1. Matikan koneksi internet. 2. Buka aplikasi. 3. Masukkan kredensial. 4. Tekan tombol Masuk.	Aplikasi menampilkan pesan kesalahan jaringan dan tetap pada LoginPage.	Valid
TC-04	KF-02	Lihat profil pengguna aktif	1. Login sebagai pengguna. 2. Buka HomePage. 3. Tekan ikon profil.	Halaman profil menampilkan nama, NIK, dan data identitas pengguna aktif.	Valid
TC-05	KF-03	Logout dari aplikasi	1. Login sebagai pengguna. 2. Tekan tombol logout. 3. Konfirmasi keluar.	Aplikasi kembali ke LoginPage dan token tersimpan dihapus dari perangkat.	Valid
TC-06	KF-04	Pindai QR code valid	1. Tekan FAB Scan ID. 2. Arahkan kamera ke QR code ear tag ternak yang terdaftar.	Aplikasi membuka halaman RiwayatTernakPage dengan detail ternak yang sesuai.	Valid
TC-07	KF-04	Pindai QR code tidak terdaftar	1. Tekan FAB Scan ID. 2. Arahkan kamera ke QR code	Aplikasi menampilkan pesan bahwa ID ternak tidak ditemukan.	Valid

			dengan ID yang tidak terdaftar.		
TC-08	KF-05	Lihat daftar desa binaan	1. Login sebagai penyuluh_produksi. 2. Buka menu Data Peternak.	Aplikasi menampilkan daftar seluruh desa binaan penyuluh.	Valid
TC-09	KF-05	Cari desa berdasarkan nama	1. Buka menu Data Peternak. 2. Ketik sebagian nama desa pada kolom pencarian.	Daftar desa tersaring sesuai kata kunci pencarian.	Valid
TC-10	KF-06	Lihat daftar peternak per desa	1. Buka menu Data Peternak. 2. Pilih sebuah desa pada daftar.	Aplikasi menampilkan daftar peternak yang tergabung pada desa tersebut.	Valid
TC-11	KF-06	Cari peternak berdasarkan nama	1. Buka menu Data Peternak. 2. Ketik sebagian nama peternak pada kolom pencarian.	Daftar peternak tersaring sesuai kata kunci pencarian.	Valid
TC-12	KF-07	Lihat detail peternak	1. Buka daftar peternak pada sebuah desa. 2. Pilih sebuah kartu peternak.	Aplikasi membuka halaman detail peternak berisi identitas dan daftar ternak miliknya.	Valid
TC-13	KF-08	Lihat daftar ternak milik peternak	1. Buka halaman detail peternak. 2. Tinjau bagian daftar ternak.	Aplikasi menampilkan seluruh ternak yang dimiliki peternak tersebut.	Valid
TC-14	KF-09	Lihat detail ternak dari menu Data Peternak	1. Buka halaman detail peternak. 2. Pilih sebuah kartu ternak. 3. Tinjau tab PROFILE.	Aplikasi menampilkan detail identitas, produktivitas, kesehatan, dan reproduksi ternak.	Valid
TC-15	KF-09	Lihat detail ternak dari menu Data Ternak	1. Login sebagai peternak. 2. Buka menu Data Ternak. 3. Pilih sebuah kartu ternak. 4. Tinjau tab PROFILE.	Aplikasi menampilkan detail ternak sesuai dengan ID yang dipilih.	Valid
TC-16	KF-10	Lihat riwayat ternak	1. Buka halaman RiwayatTernakPage. 2. Pilih tab RIWAYAT.	Aplikasi menampilkan daftar riwayat pemutakhiran ternak secara kronologis.	Valid

TC-17	KF-11	Tambah ternak baru dengan data lengkap	1. Buka halaman detail peternak. 2. Tekan tombol Tambah Ternak. 3. Isi seluruh field form. 4. Tekan tombol Simpan.	Aplikasi menampilkan pesan sukses dan daftar ternak bertambah.	Valid
TC-18	KF-11	Tambah ternak tanpa mengisi field wajib	1. Buka halaman Tambah Ternak. 2. Kosongkan salah satu field wajib. 3. Tekan tombol Simpan.	Aplikasi menolak pengiriman dan menampilkan pesan validasi pada field kosong.	Valid
TC-19	KF-11	Tambah ternak dengan foto	1. Buka halaman Tambah Ternak. 2. Ambil atau pilih foto dari galeri. 3. Isi seluruh field. 4. Tekan tombol Simpan.	Foto terkirim bersama data dan tersimpan di server.	Valid
TC-20	KF-11	Logika khusus jenis kelamin jantan	1. Buka halaman Tambah Ternak. 2. Pilih jenis kelamin Jantan. 3. Tinjau field reproduksi.	Field status reproduksi, jumlah kelahiran, dan siklus birahi dinonaktifkan dan dikosongkan.	Valid
TC-21	KF-11	Cascading dropdown jenis hewan dan produksi	1. Buka halaman Tambah Ternak. 2. Pilih jenis hewan. 3. Pilih jenis produksi. 4. Tinjau dropdown kategori ras.	Dropdown kategori ras terisi opsi sesuai kombinasi jenis hewan dan jenis produksi.	Valid
TC-22	KF-12	Perbarui data ternak	1. Buka halaman RiwayatTernakPage. 2. Pilih tab PERBARUI. 3. Ubah salah satu field. 4. Tekan tombol Simpan.	Aplikasi menampilkan pesan sukses dan tab RIWAYAT bertambah satu entri baru.	Valid
TC-23	KF-12	Form pemutakhiran terisi otomatis dari riwayat terbaru	1. Login sebagai penyuluh. 2. Buka halaman RiwayatTernakPage seekor ternak. 3. Pilih tab PERBARUI.	Seluruh field form terisi otomatis dengan nilai dari riwayat terbaru ternak.	Valid

TC-24	KF-13	Peternak melihat daftar ternak sendiri	1. Login sebagai peternak. 2. Buka menu Data Ternak.	Aplikasi menampilkan daftar ternak milik peternak yang sedang login.	Valid
TC-25	KF-14	Dropdown master data terisi pada form tambah ternak	1. Buka halaman Tambah Ternak. 2. Tinjau opsi dropdown jenis hewan, jenis produksi, jenis pakan, suplemen, status kesehatan, dan status reproduksi.	Seluruh dropdown menampilkan opsi yang valid dari master data <i>REST API</i> .	Valid
TC-26	KF-02	Persistensi sesi login setelah aplikasi ditutup	1. Login sebagai pengguna. 2. Tutup aplikasi. 3. Buka aplikasi kembali.	Aplikasi langsung membuka MainWrapper tanpa meminta login ulang.	Valid
TC-27	KF-04	Tombol kembali pada halaman pemindai	1. Tekan FAB Scan ID. 2. Tekan tombol kembali.	Aplikasi kembali ke halaman sebelumnya dan kamera dinonaktifkan.	Valid
TC-28	KF-11	Batalkan tambah ternak sebelum disimpan	1. Buka halaman Tambah Ternak. 2. Isi sebagian field. 3. Tekan tombol kembali. 4. Buka halaman Tambah Ternak lagi.	Form dikembalikan ke kondisi kosong tanpa menyimpan isian sebelumnya.	Valid
TC-29	KF-11	Pilih tanggal lahir setelah hari ini	1. Buka halaman Tambah/Perbarui Ternak. 2. Buka pemilih tanggal lahir. 3. Coba pilih tanggal setelah hari ini.	Tanggal setelah hari ini tidak dapat dipilih; pemilih tanggal terkunci maksimal pada hari ini.	Valid
TC-30	KF-11	Unggah foto ternak berukuran besar	1. Buka halaman Tambah Ternak. 2. Pilih foto beresolusi besar dari galeri. 3. Tekan tombol Simpan.	Foto dikompres terlebih dahulu di sisi aplikasi sebelum dikirim ke <i>REST API</i> .	Valid
TC-31	KF-11	Tekan tombol Simpan berulang kali	1. Buka halaman Tambah Ternak. 2. Isi seluruh field. 3. Tekan tombol Simpan berkali-kali secara cepat.	Tombol dinonaktifkan dan menampilkan indikator loading sejak	Valid

				penekanan pertama; data tersimpan satu kali.	
--	--	--	--	--	--

Berdasarkan Tabel 4.1, seluruh 31 skenario pengujian *black-box* menghasilkan status Valid, yang berarti aplikasi menampilkan perilaku sesuai dengan hasil yang diharapkan pada setiap skenario. Skenario sukses meliputi pemanggilan fitur dengan masukan dan kondisi yang valid, sementara skenario gagal meliputi masukan tidak valid, kondisi jaringan terputus, dan pembatalan operasi. Cakupan pengujian meliputi seluruh 14 kebutuhan fungsional pada Tabel 3.2. Beberapa kebutuhan fungsional mendapat lebih dari satu skenario agar perilaku pada kondisi sukses dan gagal dapat diverifikasi secara terpisah. Khusus pada KF-11 tambah ternak, skenario pengujian mencakup lima turunan yaitu pengisian data lengkap, pengisian tanpa *field* wajib, pengisian disertai foto, logika khusus jenis kelamin jantan, dan perilaku *cascading dropdown*, karena fitur tambah ternak merupakan fitur paling kompleks dengan paling banyak *field* isian.

4.2.2 Hasil API Integration Testing

Pengujian *API integration* dilakukan langsung pada seluruh *endpoint REST API* yang terdaftar pada Tabel 3.2 dengan menggunakan *tool* Postman. Pengujian ini bertujuan memverifikasi bahwa kontrak antara aplikasi *mobile* dan *REST API* berfungsi sesuai spesifikasi, khususnya pada aspek autentikasi berbasis *bearer token*, pembatasan hak akses berdasarkan *role*, dan validasi *request body*. Sesuai rancangan pada Bab III sub-bab 3.4.2, setiap *endpoint* diuji dengan empat kategori skenario yaitu sukses dengan kredensial dan parameter valid, gagal autentikasi tanpa *token* atau dengan *token* tidak berlaku, gagal otorisasi ketika *role* tidak sesuai, dan gagal validasi ketika *body request* tidak lengkap. Total terdapat 24 skenario pengujian yang disusun dengan kode AT-01 sampai AT-24. Rincian hasil pengujian *API integration* disajikan pada Tabel 4..

Tabel 4. Hasil *API Integration Testing* Aplikasi Eco-system Livestock

Kode AT	Endpoint	Method	Skenario	Request	Hasil yang Diharapkan	Status
AT-01	/login	POST	Sukses login kredensial valid	body: username dan password valid	200 OK, body berisi	Valid

					access_token, role, uuid	
AT-02	/login	POST	Gagal validasi password salah	body: username valid dengan password salah	401 Unauthorized, body berisi pesan kesalahan	Valid
AT-03	/profile	GET	Sukses ambil profil dengan token valid	header: Authorization Bearer token valid	200 OK, body berisi data profil pengguna	Valid
AT-04	/profile	GET	Gagal autentikasi tanpa token	tanpa header Authorization	401 Unauthorized	Valid
AT-05	/livestock/scan/:id	GET	Sukses pindai ID ternak valid	header: token valid, path: ID ternak terdaftar	200 OK, body berisi detail ternak	Valid
AT-06	/livestock/scan/:id	GET	Gagal ID ternak tidak terdaftar	header: token valid, path: ID acak	404 Not Found	Valid
AT-07	/fostered-breeders	GET	Sukses ambil daftar desa role penyuluh	header: token role penyuluh_produksi	200 OK, body berisi daftar wilayah	Valid
AT-08	/fostered-breeders	GET	Gagal otorisasi role peternak	header: token role peternak	403 Forbidden	Valid
AT-09	/fostered-breeders/each-village	GET	Sukses ambil peternak per desa	header: token penyuluh, query: uuid desa valid	200 OK, body berisi daftar peternak	Valid
AT-10	/breeder/:uuid	GET	Sukses ambil detail peternak	header: token penyuluh, path: uuid peternak valid	200 OK, body berisi detail peternak	Valid
AT-11	/livestock	GET	Sukses ambil daftar ternak per peternak	header: token valid, query: uuid_peternak valid	200 OK, body berisi daftar ternak	Valid
AT-12	/livestock	GET	Gagal autentikasi token tidak valid	header: Authorization Bearer token acak	401 Unauthorized	Valid

AT-13	/livestock/:id	GET	Sukses ambil detail ternak	header: token valid, path: id ternak valid	200 OK, body berisi detail ternak	Valid
AT-14	/livestock-history	GET	Sukses ambil riwayat ternak	header: token valid, query: id ternak valid	200 OK, body berisi daftar riwayat	Valid
AT-15	/livestock	POST	Sukses tambah ternak data lengkap	header: token penyuluh, body: multipart seluruh field	201 Created, body berisi konfirmasi sukses	Valid
AT-16	/livestock	POST	Gagal otorisasi tambah ternak oleh peternak	header: token role peternak, body: valid	403 Forbidden	Valid
AT-17	/livestock	POST	Gagal validasi body tidak lengkap	header: token penyuluh, body: field wajib dikosongkan	400 Bad Request, body berisi pesan field wajib	Valid
AT-18	/livestock	PUT	Sukses perbarui data ternak	header: token penyuluh, body: multipart dengan uuid_ternak	200 OK, body berisi konfirmasi sukses	Valid
AT-19	/livestock	PUT	Gagal validasi uuid_ternak tidak ada	header: token penyuluh, body: tanpa uuid_ternak	400 Bad Request	Valid
AT-20	/type-of-animal	GET	Sukses ambil master jenis hewan	header: token valid	200 OK, body berisi daftar jenis hewan	Valid
AT-21	/type-of-livestock	GET	Sukses ambil master ras dengan parameter valid	header: token valid, query: uuid_jenis_hewan dan uuid_jenis_produksi valid	200 OK, body berisi daftar ras	Valid
AT-22	/supplement	GET	Sukses ambil master suplemen	header: token valid	200 OK, body berisi daftar suplemen	Valid

AT-23	/livestock	GET	Akses daftar ternak peternak lain	header: token peternak A; query: uuid_peternak milik peternak B	403 Forbidden (permintaan ditolak berdasarkan kepemilikan)	Tidak Sesuai
AT-24	/livestock	POST	Tambah ternak dengan id_ternak duplikat	header: token penyuluh; body: id_ternak yang sudah terdaftar	Response gagal; data dengan id_ternak duplikat ditolak	Valid

Berdasarkan Tabel 4.2, dari 24 skenario pengujian *API integration* sebanyak 23 skenario berstatus Valid dan satu skenario, yaitu AT-23, berstatus Tidak Sesuai. Skenario yang berstatus Valid menunjukkan bahwa *REST API* merespons *request* sesuai kontrak yang diharapkan aplikasi. Pada kategori skenario sukses, aplikasi memperoleh data yang dibutuhkan pada kondisi normal. Pada kategori gagal autentikasi, *endpoint* yang membutuhkan *token* menolak *request* tanpa *token* atau dengan *token* tidak valid, sesuai mekanisme keamanan pada kebutuhan non-fungsional di Bab III sub-bab 3.2.2. Pada kategori gagal otorisasi, pembatasan akses berdasarkan *role* ditegakkan di sisi *REST API*, sehingga *role peternak* tidak dapat memanggil *endpoint* yang dikhususkan untuk *role penyuluh produksi*. Pada kategori gagal validasi, *REST API* menolak *request* dengan *body* yang tidak lengkap, sehingga aplikasi dapat menampilkan pesan kesalahan kepada pengguna ketika *form* diisi tidak sempurna.

Berdasarkan keseluruhan pengujian, model kontrol akses aplikasi diterapkan dalam dua lapis. Pada sisi aplikasi, tampilan menu dan ketersediaan aksi dibedakan menurut *role* pengguna, yaitu menu Data Peternak untuk penyuluh produksi dan menu Data Ternak untuk peternak. Pada sisi *server*, penegakan otorisasi dilakukan oleh *REST API*, sebagaimana terbukti pada AT-08 dan AT-16 ketika *token role* peternak ditolak saat mengakses fitur penyuluh maupun saat mencoba menambah ternak. Perlu ditegaskan bahwa pembatasan pada sisi aplikasi berfungsi sebagai pelengkap pengalaman pengguna, sedangkan keputusan keamanan tetap berada di sisi *server*, sesuai prinsip bahwa sisi klien tidak dapat dijadikan satu-satunya lapis pengamanan.

Adapun satu skenario yang berstatus Tidak Sesuai, yaitu AT-23, menguji akses daftar ternak milik peternak lain. Ketika permintaan *GET /livestock* dikirim dengan *token* seorang peternak namun memuat *uuid_peternak* milik peternak lain, *REST API* tetap mengembalikan *response* 200 beserta data ternak peternak lain tersebut, bukan menolaknya. Hal ini

menunjukkan bahwa isolasi data berbasis kepemilikan belum ditegakkan pada *endpoint* pembacaan. Pada sisi klien, aplikasi telah berperilaku benar karena hanya mengirim *uuid* pengguna yang sedang *login*; penegakan isolasi tersebut berada pada ranah *REST API* yang dikembangkan tim eksternal dan di luar ruang lingkup pengembangan klien *mobile* pada penelitian ini. Penguatan validasi kepemilikan pada sisi *REST API* direkomendasikan sebagai pengembangan lanjutan pada Bab V.

Sebagai pelengkap pengujian fungsional, dilakukan pengukuran *response time* pada *endpoint REST API* untuk menilai pemenuhan kebutuhan non-fungsional performa pada sub-bab 3.2.2. Pengukuran dilakukan menggunakan Postman terhadap sembilan *endpoint* utama yang paling sering dipanggil aplikasi. Setiap *endpoint* dipanggil sebanyak enam kali; *request* pertama diabaikan untuk menghilangkan *overhead* penyiapan koneksi (DNS, TCP, dan TLS *handshake*) yang hanya terjadi sekali, sehingga nilai yang dilaporkan adalah rata-rata dari lima *request* berikutnya yang merepresentasikan kondisi *steady-state*. Pengukuran ini bersifat indikatif dari sisi klien, bukan *load testing*. Hasil pengukuran disajikan pada Tabel 4.3.

Tabel 4.2 Hasil Pengujian *Response Time* API

No	Endpoint	Method	Rerata (ms)	Min–Maks (ms)	Ukuran Response
1	/login	POST	472	405–518	734 B
2	/profile	GET	114	95–138	974 B
3	/fostered-breeders	GET	132	115–184	554 B
4	/fostered-breeders/each-village	GET	141	124–175	2,24 KB
5	/breeder/:uuid	GET	102	95–111	1,49 KB
6	/livestock	GET	99	88–133	2,84 KB
7	/livestock/:id	GET	118	86–183	1,94 KB
8	/livestock-history	GET	94	88–98	2,03 KB
9	/livestock/scan/:id	GET	95	90–98	591 B

Berdasarkan Tabel 4.3, seluruh *endpoint* mengembalikan status 200 OK dengan *response time* rata-rata di bawah satu detik. Delapan dari sembilan *endpoint* merespons di bawah 150 ms, sedangkan *endpoint* /login memiliki rata-rata tertinggi (472 ms) karena melibatkan proses autentikasi dan verifikasi kredensial yang lebih berat dibanding *endpoint* pembacaan data. Ukuran *response* yang relatif kecil, seluruhnya di bawah 3 KB, turut menjaga waktu transfer tetap singkat. Pengukuran ini mencerminkan waktu respon *REST API*, yaitu komponen utama dari waktu respon yang dirasakan pengguna sebagaimana dimaksud pada kebutuhan non-

fungsiional performa di sub-bab 3.2.2. Dengan demikian, dari sisi *response time*, *REST API* tergolong responsif dan mendukung pemenuhan kebutuhan performa tersebut.

4.2.3 Pengujian Usability dengan System Usability Scale (SUS)

Selain pengujian fungsiional melalui *black-box testing* dan *API integration testing*, dilakukan pengujian *usability* untuk mengukur tingkat keterterimaan aplikasi dari sudut pandang pengguna. Pengujian ini memakai instrumen *System Usability Scale* (SUS), yaitu kuesioner baku yang umum dipakai untuk menilai *usability* perangkat lunak karena ringkas dan tidak bergantung pada jenis teknologi tertentu (Brooke, 2013).

Instrumen dan Metode Pengujian

Instrumen SUS terdiri atas sepuluh pernyataan yang berselang-seling antara nada positif (nomor ganjil) dan negatif (nomor genap). Setiap pernyataan dinilai dengan skala Likert lima tingkat, yaitu Sangat Tidak Setuju (1), Tidak Setuju (2), Netral (3), Setuju (4), dan Sangat Setuju (5). Kuesioner disusun dalam bahasa Indonesia dan disebar melalui *Google Form* kepada sembilan responden dari kalangan akademik dan praktisi bidang peternakan yang difasilitasi oleh kolaborator dari Universitas Sriwijaya. Seluruh responden menggunakan aplikasi terlebih dahulu sebelum mengisi kuesioner, sehingga penilaian didasarkan pada pengalaman pemakaian langsung.

Skor SUS dihitung dengan ketentuan baku. Untuk pernyataan nomor ganjil, kontribusi skor adalah nilai jawaban dikurangi satu. Untuk pernyataan nomor genap, kontribusi skor adalah lima dikurangi nilai jawaban. Seluruh kontribusi dijumlahkan lalu dikalikan 2,5 sehingga menghasilkan skor akhir 0 sampai 100 untuk tiap responden. Skor akhir aplikasi adalah rata-rata skor seluruh responden.

Rekapitulasi Jawaban Responden

Rerata jawaban untuk tiap pernyataan ditunjukkan pada Tabel 4.4. Pernyataan positif memperoleh rerata tinggi dan pernyataan negatif memperoleh rerata rendah, yang sama-sama menunjukkan penilaian positif terhadap aplikasi.

Tabel 4.3 Rekapitulasi Rerata Jawaban Responden per Pernyataan SUS

No	Pernyataan	Rerata
1	Saya pikir saya akan sering menggunakan aplikasi ini	4,67

2	Saya merasa aplikasi ini terlalu kompleks (ribet)	2,33
3	Saya pikir aplikasi ini mudah digunakan	4,11
4	Saya butuh bantuan orang teknis untuk menggunakan aplikasi ini	3,33
5	Saya merasa berbagai fungsi dalam aplikasi ini terintegrasi dengan baik	4,33
6	Saya merasa terlalu banyak ketidakkonsistenan dalam aplikasi ini	2,56
7	Saya bayangkan kebanyakan orang akan belajar menggunakan aplikasi ini dengan cepat	4,56
8	Saya merasa aplikasi ini sangat rumit untuk digunakan	2,00
9	Saya merasa sangat percaya diri menggunakan aplikasi ini	4,56
10	Saya perlu belajar banyak hal sebelum bisa menggunakan aplikasi ini	3,56

Perhitungan dan Hasil Skor

Kontribusi skor tiap pernyataan (P1 sampai P10), jumlah kontribusi (Σ), dan skor akhir tiap responden ditunjukkan pada Tabel 4.5.

Tabel 4.4 Perhitungan Skor SUS per Responden

R	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Σ	Skor
R1	4	1	3	1	4	1	3	3	4	1	25	62,5
R2	3	3	3	2	3	3	3	3	3	1	27	67,5
R3	4	2	3	0	2	2	4	3	3	1	24	60,0
R4	4	3	3	2	3	3	3	3	3	2	29	72,5
R5	3	2	4	2	3	1	4	3	4	2	28	70,0
R6	3	3	3	3	3	3	3	3	3	2	29	72,5
R7	4	2	2	0	4	4	4	2	4	0	26	65,0
R8	4	4	4	1	4	3	4	3	4	2	33	82,5
R9	4	4	3	4	4	2	4	4	4	2	35	87,5
Rata-rata	71,1											

Dari sembilan responden diperoleh skor terendah 60,0 dan tertinggi 87,5, dengan rata-rata skor akhir 71,1.

Interpretasi Hasil

Skor rata-rata 71,1 diinterpretasikan dengan kerangka Bangor dkk. (2009). Skor di atas 68 berada di atas rata-rata dan termasuk kategori dapat diterima (*acceptable*), sedangkan dari

sisi label kata sifat skor pada kisaran ini tergolong *Good*. Dengan demikian, aplikasi Eco-system Livestock dinilai memiliki tingkat *usability* yang baik dan dapat diterima pengguna. Hasil ini sejalan dengan kecenderungan jawaban pada Tabel 4.4, yaitu pernyataan tentang kemudahan penggunaan dan rasa percaya diri memperoleh rerata tinggi, sementara pernyataan tentang kerumitan memperoleh rerata rendah.

4.3 Pembahasan

Sub-bab ini membahas hasil implementasi dan pengujian, serta mengaitkannya dengan rumusan masalah pada Sub-bab 1.2. Dua rumusan masalah yang diangkat pada penelitian ini adalah bagaimana mengembangkan aplikasi *mobile* pencatatan ternak yang terintegrasi dengan *REST API* Eco-system Livestock, serta bagaimana memverifikasi bahwa fitur aplikasi dan integrasinya dengan *REST API* berjalan sesuai kebutuhan fungsional yang ditetapkan.

Rumusan masalah pertama terjawab melalui implementasi pada sub-bab 4.1. Aplikasi *mobile* Eco-system Livestock dibangun menggunakan *framework* Flutter dengan pola MVVM yang memisahkan empat lapisan: *View*, *ViewModel* berbasis *Provider*, *Service* untuk akses *REST API*, dan *Model* untuk pemetaan data. Pemisahan ini diterapkan konsisten di seluruh fitur — autentikasi, menu Data Peternak untuk *role penyuluh_produksi*, menu Data Ternak untuk *role peternak*, pemutakhiran riwayat, dan pemindai QR code. Integrasi dengan *REST API* menggunakan *package http* di seluruh kelas *Service*, dengan pola pemanggilan seragam: konstruksi URL, *header* autentikasi *Bearer token*, *parsing response* lewat *factory fromJson*, dan *error handling* dua lapis yang menjaga aplikasi tetap stabil saat jaringan atau *server* gagal. Seluruh 14 kebutuhan fungsional pada Tabel 3.2 berhasil diwujudkan, sebagaimana ditunjukkan cuplikan kode dan tangkapan layar di sub-bab 4.1.

Rumusan masalah kedua terjawab melalui hasil pengujian pada sub-bab 4.2. Pengujian *black-box* dengan 31 skenario memverifikasi perilaku fungsional aplikasi dari sisi pengguna, dan seluruhnya berstatus Valid. Pengujian *API integration* dengan 24 skenario memverifikasi kontrak aplikasi dengan *REST API*; 23 skenario berstatus Valid dan satu skenario (AT-23) berstatus Tidak Sesuai karena menemukan bahwa isolasi data berbasis kepemilikan belum ditegakkan pada sisi *REST API*, sebagaimana dibahas pada Sub-bab 4.2.2. Kombinasi kedua metode pengujian membuktikan aplikasi berjalan sesuai kebutuhan fungsional pada dua sisi sekaligus: antarmuka pengguna dan integrasi data. Empat kategori skenario pada pengujian *API integration* secara khusus menunjukkan bahwa autentikasi *bearer token* dan pembatasan

role-based access pada *REST API* berjalan sesuai rancangan, sejalan dengan kebutuhan non-fungsional keamanan pada sub-bab 3.2.2.

Selain itu, pengujian *usability* dengan SUS memperoleh skor rata-rata 71,1 yang termasuk kategori *Good* dan dapat diterima (*acceptable*) menurut kerangka Bangor dkk. (2009). Hasil ini memperkuat jawaban atas rumusan masalah kedua dengan menunjukkan bahwa selain memenuhi kebutuhan fungsional, aplikasi juga dinilai memiliki tingkat keterterimaan yang baik dari sudut pandang pengguna.

Di luar fungsionalitas dasar, penelitian ini menerapkan tiga pola implementasi yang berdampak pada kinerja aplikasi. Pertama, pemanggilan *endpoint* secara paralel menggunakan *Future.wait* pada pengambilan daftar desa beserta peternak per desa di *PeternakService*, dan pada pengambilan enam *master data dropdown* di *TernakProvider*. Pola ini memperpendek waktu pemuatan halaman yang membutuhkan data dari beberapa *endpoint* sekaligus. Kedua, pemisahan *state* pada *PeternakProvider* menjadi *masterData* dan *displayData*, sehingga fitur pencarian lokal dapat berjalan tanpa memanggil *REST API* kembali. Ketiga, penggunaan satu kelas *TernakRequestModel* untuk operasi tambah dan pemutakhiran, dibedakan oleh kehadiran *field* opsional *uuidTernak* pada *method toJson*. Ketiga pola ini menjadi kontribusi penelitian pada implementasi praktis arsitektur MVVM untuk aplikasi *mobile* yang terhubung ke *REST API*. Pengukuran *response time* pada Tabel 4.3 menunjukkan seluruh *endpoint* utama merespons di bawah satu detik, sehingga pola pemanggilan paralel dengan *Future.wait* dapat memuat data dari beberapa *endpoint* sekaligus tanpa menambah waktu tunggu pengguna secara berarti.

Dua keputusan implementasi kecil ikut menyumbang konsistensi data yang tersimpan di *REST API*. Pertama, validasi *form* di sisi *mobile* yang menolak pengiriman *request* ketika *field* wajib belum terisi, sebagaimana ditunjukkan skenario TC-18 pada Tabel 4.1. Validasi ini bukan pengganti validasi sisi *server*, melainkan saringan pertama yang mencegah *request* tidak lengkap meninggalkan perangkat. Kedua, tab Riwayat pada *RiwayatTernakPage* yang menyimpan setiap pemutakhiran sebagai entri tanggal terpisah. Setiap entri dapat diperluas untuk menampilkan *snapshot* data ternak pada saat itu, sehingga perubahan bobot, status kesehatan, dan status reproduksi tetap dapat ditelusuri dari waktu ke waktu. Pencatatan baru tidak menimpa pencatatan sebelumnya, dan kekeliruan pengisian masih dapat dibandingkan dengan riwayat sebelumnya. Aplikasi ini tidak mencegah pengguna lupa mencatat, tetapi ketika pencatatan dilakukan, struktur data yang disimpan menjaga jejaknya tetap utuh.

Pemindai QR *code* menjadi elemen pembeda aplikasi ini dibandingkan aplikasi sejenis pada penelitian terdahulu di sub-bab 2.1. Pemindaian melalui kamera perangkat menggantikan pengetikan manual ID ternak sepanjang 14 karakter, sehingga waktu identifikasi di lapangan lebih singkat dan potensi salah ketik berkurang. Pembatasan fitur pada QR *code* saja — tanpa NFC dan RFID — menjaga aplikasi tetap dapat berjalan pada perangkat Android kelas menengah yang umum digunakan peternak dan penyuluh.

Hasil pengujian yang sebagian besar berstatus Valid tidak berarti aplikasi bebas keterbatasan. Tiga keterbatasan teridentifikasi: dukungan *platform* yang terbatas pada Android, kebutuhan koneksi internet aktif untuk mayoritas fitur, dan ukuran *payload* foto ternak dalam format *base64 string* yang membesar pada foto beresolusi tinggi. Keterbatasan ini menjadi bahan saran untuk pengembangan berikutnya pada sub-bab 5.2.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian pengembangan aplikasi *mobile* Eco-system Livestock ini menghasilkan dua luaran utama yang menjawab dua rumusan masalah pada Sub-bab 1.2. Kesimpulan diuraikan sebagai berikut.

Pertama, aplikasi *mobile* Eco-system Livestock berhasil dikembangkan menggunakan *framework* Flutter dengan pola *Model-View-ViewModel* (MVVM). Arsitektur aplikasi memisahkan tanggung jawab ke dalam empat lapisan: *View* berupa halaman dan *widget* Flutter, *ViewModel* berupa kelas *Provider* yang meng-*extends* *ChangeNotifier*, *Service* yang berkomunikasi dengan *REST API*, dan *Model* yang memetakan data antara *response* API dan objek Dart. Aplikasi memuat seluruh 14 kebutuhan fungsional pada Tabel 3.2. Cakupannya meliputi autentikasi dengan pembatasan *role-based access*, menu Data Peternak untuk *role penyuluh_produksi*, menu Data Ternak untuk *role peternak*, pemutakhiran riwayat ternak berbasis empat aspek data, pemindai *QR code* sebagai fitur universal lintas *role*, serta akses ke seluruh *endpoint REST API* dengan pola *service layer* yang seragam. Integrasi dengan *REST API* didukung oleh *package http* untuk pemanggilan *endpoint* dan *package shared_preferences* untuk persistensi sesi *login*.

Kedua, hasil pengujian menunjukkan bahwa seluruh kebutuhan fungsional yang ditetapkan berjalan sesuai perilaku yang diharapkan. Pengujian *black-box* dengan 31 skenario memverifikasi perilaku aplikasi dari sisi pengguna dan seluruhnya berstatus Valid. Pengujian *API integration* dengan 24 skenario memverifikasi kontrak antara aplikasi *mobile* dan *REST API*, dengan 23 skenario Valid dan satu temuan (AT-23) terkait isolasi kepemilikan pada sisi *REST API*. Pengujian *usability* dengan SUS memperoleh skor 71,1 (kategori *Good*), yang menunjukkan aplikasi dapat diterima pengguna. Pengukuran *response time* terhadap sembilan *endpoint* utama menunjukkan seluruhnya merespons di bawah satu detik, sehingga *REST API* tergolong responsif dan mendukung pemenuhan kebutuhan non-fungsional performa. Empat kategori skenario pada pengujian *API integration* — sukses kredensial valid, gagal autentikasi, gagal otorisasi, dan gagal validasi — membuktikan bahwa autentikasi *bearer token* dan pembatasan akses berdasarkan *role* pada *REST API* berjalan sesuai rancangan. Kombinasi kedua metode pengujian membuktikan aplikasi berfungsi pada dua aspek: antarmuka pengguna dan integrasi data.

5.2 Saran

Aplikasi *mobile* Eco-system Livestock yang dihasilkan pada penelitian ini masih memiliki sejumlah keterbatasan yang dapat menjadi bahan pengembangan pada penelitian berikutnya. Saran yang dapat dipertimbangkan adalah sebagai berikut.

Pertama, dukungan *platform* aplikasi perlu diperluas ke iOS agar dapat digunakan oleh pengguna perangkat Apple. Pengembangan menggunakan Flutter memungkinkan perluasan *platform* dari satu basis kode, sehingga sebagian besar logika aplikasi yang sudah ada dapat digunakan kembali. Penyesuaian yang diperlukan meliputi pengaturan *build* iOS, penanganan izin kamera dan penyimpanan sesuai ketentuan iOS, serta pengujian antarmuka pada berbagai ukuran perangkat iOS.

Kedua, aplikasi saat ini membutuhkan koneksi internet aktif untuk hampir seluruh fitur karena setiap pemanggilan fitur utama memerlukan *request* ke *REST API*. Pada kondisi jaringan yang tidak stabil, pengalaman pengguna dapat terganggu. Pengembangan berikutnya dapat menambahkan kapabilitas *offline-first* dengan menyimpan sebagian data pada basis data lokal, misalnya menggunakan *package sqflite* atau *Hive*, lalu melakukan sinkronisasi ketika koneksi internet tersedia kembali. Fitur ini akan sangat membantu ketika aplikasi digunakan di lokasi lapangan dengan sinyal terbatas.

Ketiga, format pengiriman foto ternak yang saat ini menggunakan *base64 string* pada *field* teks perlu dikaji ulang. Pendekatan ini dipilih sesuai kontrak *backend* yang sudah ada, tetapi ukuran *payload* menjadi sekitar sepertiga lebih besar dibanding format biner karena sifat pengkodean *base64*. Pada foto dengan resolusi tinggi, hal ini dapat memperlambat proses pengiriman pada jaringan yang lambat. Pengembangan berikutnya dapat mempertimbangkan pengiriman foto dalam bentuk *binary* melalui *multipart form-data* klasik, atau menambahkan kompresi gambar pada sisi aplikasi sebelum pengiriman.

Keempat, penelitian ini belum mengintegrasikan teknologi identifikasi NFC dan RFID yang juga tersedia pada prototipe *ear tag* hasil penelitian Fariani dkk. (2025). Penelitian berikutnya dapat memperluas fitur pemindai dengan membaca tag NFC dan RFID melalui perangkat *smartphone* yang mendukung, sehingga aplikasi dapat menangani seluruh pilihan metode identifikasi yang tersedia pada *ear tag* dan tidak terbatas pada *QR code* saja.

Kelima, antarmuka pengguna dapat ditingkatkan dengan menambahkan fitur visualisasi data riwayat ternak dalam bentuk grafik sederhana, misalnya grafik tren bobot atau tren produksi susu dari waktu ke waktu. Saat ini data riwayat ditampilkan sebagai daftar kronologis saja, sehingga pengguna perlu membandingkan nilai antar entri secara manual. Grafik dapat

membantu pengguna melihat tren yang bermakna secara langsung, khususnya untuk penyuluh yang melakukan pemantauan berkala terhadap perkembangan ternak binaan.

Keenam, halaman utama (*HomePage*) saat ini menampilkan beberapa *section* tambahan berupa berita peternakan, materi edukasi, dan informasi harga ternak yang masih berisi konten contoh (*placeholder*) hard-coded dari *folder assets* aplikasi. Pengembangan berikutnya dapat melengkapi *section* tersebut dengan integrasi konten dinamis dari *REST API*, sehingga pengguna memperoleh informasi terkini yang sesuai dengan wilayah dan kategori ternak masing-masing. Penambahan *endpoint* khusus untuk konten dinamis ini akan menjadikan *HomePage* tidak hanya sebagai pintu masuk ke menu utama, tetapi juga sebagai dasbor informasi yang aktif dan berguna bagi penyuluh maupun peternak dalam pengambilan keputusan harian.

Ketujuh, isolasi data berbasis kepemilikan pada *endpoint* pembacaan data ternak di sisi *REST API* perlu diperkuat. Pengujian *API integration* (AT-23) menemukan bahwa permintaan pembacaan data ternak masih dilayani meskipun *uuid_peternak* yang dikirim bukan milik pengguna yang sedang *login*, sehingga seorang peternak berpotensi membaca data ternak milik peternak lain apabila mengetahui *uuid*-nya. Karena penegakan otorisasi berada di sisi *REST API*, pengembangan berikutnya perlu menambahkan validasi yang memastikan setiap peternak hanya dapat membaca data ternak miliknya sendiri, sejalan dengan otorisasi berbasis *role* yang sudah ditegakkan untuk operasi tulis.

Kedelapan, mekanisme *refresh token* dapat ditambahkan pada *REST API* agar sesi pengguna dapat diperpanjang secara aman tanpa mengharuskan *login* ulang ketika *token* kedaluwarsa. Saat ini aplikasi menyimpan *token* secara lokal dan mengarahkan pengguna kembali ke halaman *login* ketika *token* tidak lagi berlaku. Dengan *refresh token*, aplikasi dapat memperoleh *token* baru secara otomatis selama sesi masih sah, sehingga kenyamanan penggunaan meningkat tanpa mengorbankan keamanan.

DAFTAR PUSTAKA

- Aisuwarya, R., Hersyah, M. H., Putri, R. E., Ferdian, R., & Fatimah, F. (2020). Implementasi teknologi NFC untuk *recording* data sapi perah Kelompok Tani Lembu Alam Serambi Kota Padang Panjang. *Journal of Appropriate Technology for Community Services (JATTEC)*, 1(2), 74–81. <https://doi.org/10.20885/jattec.vol1.iss2.art3>
- Android Developers. (n.d.). *Meet Android Studio*. Android Developers. <https://developer.android.com/studio/intro>
- Asmara, R. A., Budiman, R. P. P., Astiningrum, M., Sayudha, B., Handayani, A. N., & Rahmad, C. (2022). QRCode recognition on Flutter framework mobile application implemented on entrance security system. In *Proceedings of the 2022 Annual Technology, Applied Science and Engineering Conference (ATASEC 2022)* (pp. 113–124). Atlantis Press. https://doi.org/10.2991/978-94-6463-106-7_11
- Badan Pusat Statistik. (2026). *Peternakan dalam angka 2025*. Jakarta: Badan Pusat Statistik. <https://www.bps.go.id/id/publication/2026/01/09/50771bc6f5761886458558ba/peternakan-dalam-angka-2025.html>
- Bangor, A., Kortum, P. T., & Miller, J. T. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4(3), 114–123. <https://uxpajournal.org/determining-what-individual-sus-scores-mean-adding-an-adjective-rating-scale/>
- Brooke, J. (2013). SUS: A retrospective. *Journal of Usability Studies*, 8(2), 29–40. <https://uxpajournal.org/sus-a-retrospective/>
- BrowserStack. (n.d.). *What is black box testing: Types, tools & examples*. BrowserStack. <https://www.browserstack.com/guide/black-box-testing>
- Dart. (n.d.). *Dart overview*. Dart. <https://dart.dev/overview>
- Fabiyi, S. D., Ren, J., Han, Y., Zhu, Q., & Barclay, D. (2022). Mobile platform for livestock monitoring and inspection. In *2022 3rd International Informatics and Software Engineering Conference (IISEC)* (pp. 1–6). IEEE. <https://doi.org/10.1109/IISEC56263.2022.9998297>
- Fariani, A., Warly, L., Pratama, A. N. T., Setiaji, H., & Paputungan, I. (2025). *Development and prototype testing of a multi-functional ear tag system integrating RFID, NFC, and*

- QR for livestock monitoring in Indonesia* [Manuskrip tidak dipublikasikan]. Universitas Sriwijaya & Universitas Islam Indonesia.
- Flutter. (n.d.). *Flutter — Build apps for any screen*. Flutter. <https://flutter.dev/>
- Fuady, T., Amirudin, D., Surahmat, A., & Rifai, A. (2023). Perancangan sistem informasi catatan dan pengawasan hewan ternak menggunakan QR code berbasis web dengan metode Agile. *Jurnal Ilmiah Sains dan Teknologi*, 7(1), 33–42. <https://doi.org/10.47080/saintek.v7i1.2540>
- HTTP. (n.d.). *http: A composable, multi-platform, Future-based API for HTTP requests*. pub.dev. <https://pub.dev/packages/http>
- Kavlakoglu, E. (n.d.). *Agile vs. Waterfall: Which project management methodology is best for you?* IBM. <https://www.ibm.com/think/topics/agile-vs-waterfall>
- Masyhur, F. (2016). Model pengembangan literasi TIK masyarakat tani dan nelayan. *Jurnal Pekommas*, 1(1), 101–110. <https://jkd.komdigi.go.id/index.php/pekommas/article/view/353>
- Mozilla. (n.d.). *REST*. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Glossary/REST>
- Neethirajan, S. (2023). The significance and ethics of digital livestock farming. *AgriEngineering*, 5(1), 488–505. <https://doi.org/10.3390/agriengineering5010032>
- Postman. (n.d.). *Test API integrations and data flow in Postman*. Postman Docs. <https://learning.postman.com/docs/tests-and-scripts/test-apis/integration-testing>
- Prayoga, R. R., Munawar, G., Jumiyan, R., & Syalsabila, A. (2021). Performance analysis of BLoC and *Provider* state management library on Flutter. *Jurnal Mantik*, 5(3), 1591–1597. <https://iocscience.org/ejournal/index.php/mantik/article/view/1539>
- Provider*. (n.d.). *provider: A wrapper around InheritedWidget to make them easier to use and more reusable*. pub.dev. <https://pub.dev/packages/provider>
- Resti, Y., Gutierrez Reynoso, G., Probst, L., Indriasari, S., Mindara, G. P., Hakim, A., & Wurzinger, M. (2024). A review of on-farm recording tools for smallholder dairy farming in developing countries. *Tropical Animal Health and Production*, 56(5), Article 168. <https://doi.org/10.1007/s11250-024-04024-9>
- Shared Preferences. (n.d.). *shared_preferences: Flutter plugin for reading and writing simple key-value pairs*. pub.dev. https://pub.dev/packages/shared_preferences
- Steenbakker, J. (n.d.). *mobile_scanner: A universal Flutter barcode and QR code scanner*. pub.dev. https://pub.dev/packages/mobile_scanner

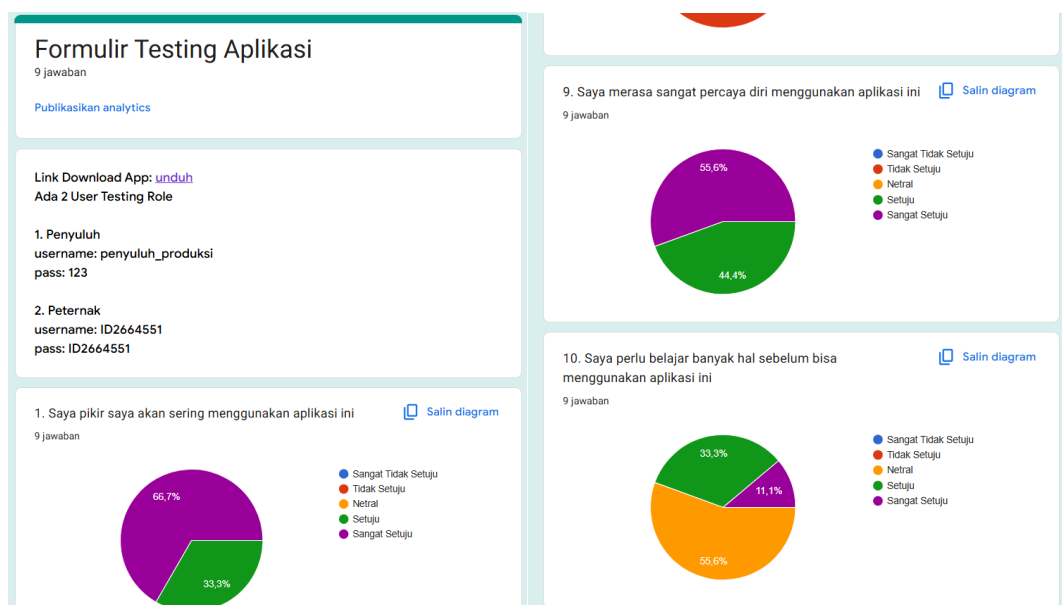
- Thannithi, W., Intawicha, P., Phuwisaranakom, P., & Saengwong, S. (2024). Design and evaluation of a mobile application for enhancing farm management and performance assessment in fattening beef cattle. *Journal of Advanced Veterinary and Animal Research*, 11(1), 203–211. <https://doi.org/10.5455/javar.2024.k766>

LAMPIRAN

Lampiran 1 — Dokumentasi Koordinasi Daring (Zoom) bersama Tim Kolaborator UII, UNSRI, dan Tim IT Eksternal



Lampiran 2 — Kuesioner *System Usability Scale (SUS)* dan Ringkasan Jawaban Responden melalui *Google Form*



Lampiran 3 — Rekapitulasi Jawaban Mentah Responden SUS pada *Google Form* (tabel respons lengkap 9 responden beserta *timestamp*)

[illegible]