BAB III
ANALISIS DAN PERANCANGAN

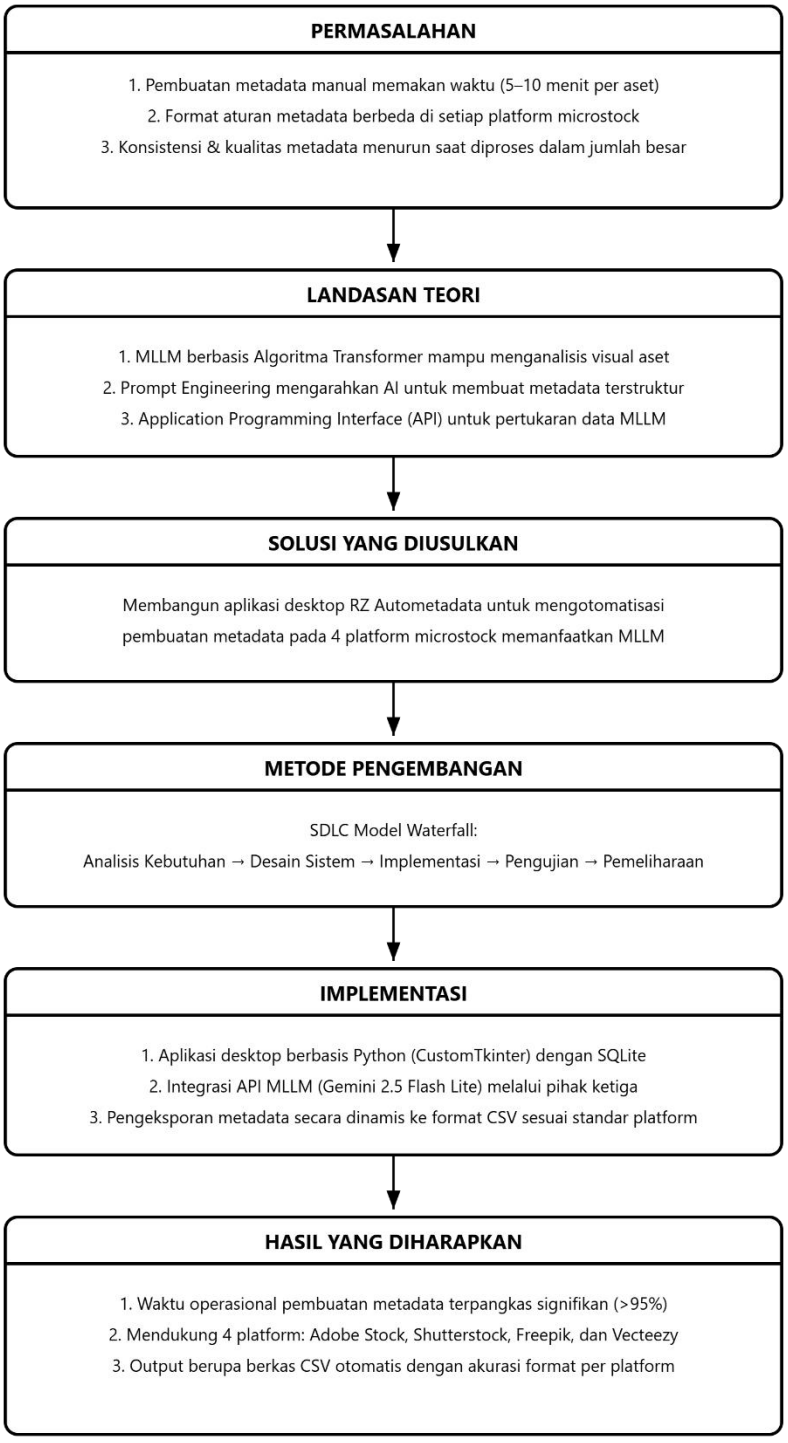
3.1 Kerangka Berpikir

Penelitian ini dilakukan karena adanya permasalahan yang dihadapi oleh kontributor microstock dalam mengelola metadata aset digital. Setiap aset yang diunggah ke *platform* microstock wajib dilengkapi metadata berupa judul, kata kunci, dan kategori agar dapat ditemukan oleh calon pembeli melalui fitur pencarian. Namun, setiap *platform* memiliki format dan ketentuan metadata yang berbeda. Adobe Stock menggunakan kategori dalam bentuk angka, Shutterstock menggunakan kategori berbasis teks, Freepik menambahkan kolom khusus untuk menandai aset buatan kecerdasan buatan, sedangkan Vecteezy mewajibkan adanya kolom lisensi tersendiri. Perbedaan tersebut mengharuskan kontributor yang aktif di lebih dari satu *platform* untuk menyusun metadata secara terpisah bagi setiap *platform*, sehingga proses ini membutuhkan waktu yang cukup lama dan berulang. Studio RZ Digital ID sebelumnya sudah memiliki aplikasi desktop yang dapat membuat metadata secara otomatis, tetapi aplikasi tersebut baru mendukung satu *platform* saja, yaitu Adobe Stock. Untuk *platform* lain, kontributor masih harus mengerjakan metadata secara manual. Keterbatasan ini menjadi kendala utama ketika jumlah aset yang harus diunggah cukup banyak dan perlu didistribusikan ke beberapa *platform* sekaligus. Oleh karena itu, diperlukan pengembangan lebih lanjut agar aplikasi tersebut mampu menangani kebutuhan metadata untuk empat *platform* microstock dalam satu sistem.

Di sisi lain, perkembangan teknologi kecerdasan buatan saat ini sudah sampai pada tahap di mana model *Multimodal Large Language Model* (MLLM) mampu memahami isi sebuah gambar atau video, lalu mendeskripsikannya dalam bentuk teks secara akurat. Model seperti ini dibangun dengan arsitektur algoritma *Transformer* yang bekerja melalui mekanisme *self-attention*, sehingga dapat menangkap makna dan hubungan antar elemen dalam data masukan secara menyeluruh. Kemampuan inilah yang menjadi dasar untuk mengotomatisasi pembuatan metadata dari konten visual aset digital.

Berdasarkan permasalahan dan peluang tersebut, penelitian ini bertujuan mengembangkan aplikasi RZ Autometadata agar tidak lagi terbatas pada satu *platform*, melainkan mampu menghasilkan metadata untuk empat *platform* microstock sekaligus, yaitu Adobe Stock, Shutterstock, Freepik, dan Vecteezy. Pengembangan dilakukan mengikuti tahapan metode Waterfall mulai dari analisis kebutuhan, perancangan, implementasi, hingga pengujian menggunakan metode *Black Box* Testing. Hasil akhir yang diharapkan adalah berkas CSV untuk setiap *platform* yang siap diunggah langsung oleh kontributor tanpa perlu penyesuaian manual.

Kerangka berpikir dari alur penelitian ini digambarkan secara visual pada Gambar berikut:



Gambar 3. 1 Kerangka Berpikir Penelitian

Gambar 3.1 menjelaskan alur kerangka berpikir penelitian ini secara keseluruhan. Tahap pertama dimulai dari identifikasi masalah, yaitu kontributor

microstock mengalami kesulitan dalam menyusun metadata secara manual untuk empat *platform* yang memiliki format berbeda-beda. Permasalahan ini kemudian dianalisis lebih lanjut melalui studi literatur dan observasi langsung terhadap kebutuhan Studio RZ Digital ID.

Berdasarkan hasil analisis tersebut, dirumuskan solusi berupa pengembangan aplikasi desktop RZ Autometadata yang memanfaatkan teknologi *Multimodal Large Language Model* (MLLM) dengan arsitektur algoritma *Transformer* untuk menghasilkan metadata secara otomatis dari konten visual aset digital. Aplikasi ini dirancang agar mampu menangani empat *platform* microstock sekaligus, yaitu Adobe Stock, Shutterstock, Freepik, dan Vecteezy, dalam satu kali proses.

Proses pengembangan dilakukan mengikuti tahapan dengan metode Waterfall yang mencakup analisis kebutuhan, perancangan sistem, implementasi (*coding*), serta pengujian menggunakan metode *Black Box Testing*. Keluaran akhir dari sistem ini adalah berkas CSV per *platform* yang siap diunggah langsung oleh kontributor tanpa memerlukan penyesuaian manual, sehingga dapat menghemat waktu dan meningkatkan produktivitas dalam proses distribusi aset digital ke berbagai *platform* microstock.

3.2 Metode Penelitian

Metode pengembangan sistem yang digunakan dalam penelitian ini adalah Metode Waterfall. Waterfall dipilih karena pendekatan ini bersifat sistematis, linear, dan sekuensial, sehingga sesuai untuk pengembangan aplikasi RZ Autometadata yang kebutuhan fungsional, spesifikasi, dan alur kerjanya telah terdefinisi secara jelas sejak awal penelitian.

Pendekatan kualitatif deskriptif tetap digunakan pada tahap awal untuk memahami permasalahan secara mendalam melalui observasi langsung terhadap alur kerja kontributor di Studio RZ Digital ID, studi literatur, serta dokumentasi terhadap standar metadata dari empat *platform* microstock (Adobe Stock, Shutterstock, Freepik, dan Vecteezy).

3.2.1 Lokasi dan Waktu Penelitian

Penelitian ini dilaksanakan di Studio RZ Digital ID yang berlokasi di Kampung Rancajigang RT 03 RW 15, Desa Padamulya, Kecamatan Majalaya, Kabupaten Bandung. Waktu pelaksanaan penelitian berlangsung selama enam bulan, terhitung sejak Februari 2026 sampai Juli 2026.

3.2.2 Objek dan Subjek Penelitian

Objek dalam penelitian ini adalah proses otomatisasi pembuatan metadata aset digital yang meliputi judul, kata kunci, dan kategori pada *platform* microstock. Fokus utama penelitian terletak pada penerapan algoritma *Transformer* melalui *Multimodal Large Language Model* (MLLM) ke dalam aplikasi desktop untuk menghasilkan metadata otomatis yang sesuai dengan standar masing-masing *platform*.

Subjek dalam penelitian ini adalah kontributor aktif pada *platform* Adobe Stock, Shutterstock, Freepik, dan Vecteezy. Dalam penelitian ini, peneliti juga berperan sebagai pengguna utama yang memberikan masukan berupa data visual sekaligus melakukan evaluasi langsung terhadap fungsionalitas aplikasi yang dikembangkan.

3.2.3 Alat dan Bahan Penelitian

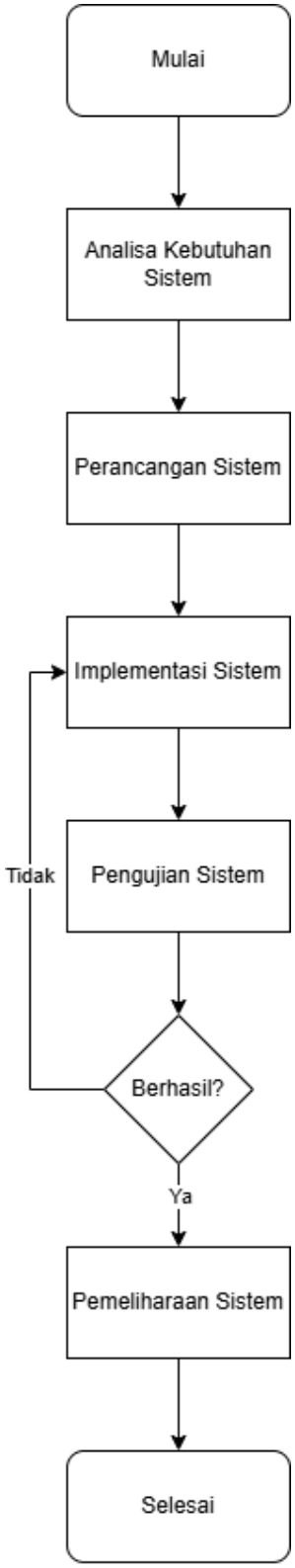
Perangkat keras yang digunakan dalam penelitian ini berupa satu unit laptop Lenovo dengan spesifikasi prosesor AMD Ryzen 3 5300U, memori akses acak (Random Access Memory/RAM) sebesar 20 GB, serta media penyimpanan sebesar 512 GB.

Perangkat lunak yang digunakan meliputi sistem operasi Windows 11, Visual Studio Code sebagai editor kode, bahasa pemrograman Python versi 3.10, pustaka *CustomTkinter* untuk pengembangan antarmuka pengguna grafis, SQLite sebagai basis data lokal, serta GitHub sebagai pendukung proses *Continuous Integration/Continuous Deployment* (CI/CD) dan sistem pembaruan otomatis aplikasi.

3.2.4 Prosedur Pengembangan Sistem

Proses pengembangan sistem dalam penelitian ini menggunakan tahapan Waterfall dengan pendekatan sekuensial. Menurut (Nagara et al., 2023) model pengembangan perangkat lunak yang mengharuskan setiap tahapan diselesaikan secara tuntas sebelum melanjutkan ke tahap berikutnya. Pendekatan ini dipilih karena kebutuhan fungsional aplikasi, termasuk format metadata untuk setiap *platform* dan spesifikasi keluaran dalam bentuk berkas CSV, telah dapat didefinisikan secara jelas sejak awal sehingga tidak memerlukan perubahan kebutuhan secara berulang selama proses pengembangan.

Tahapan pengembangan sistem yang dilakukan meliputi:



Gambar 3. 2 Waterfall

1. Analisis Kebutuhan Sistem

   Pada tahap ini dilakukan identifikasi kebutuhan sistem yang mencakup kebutuhan fungsional dan nonfungsional. Kebutuhan fungsional meliputi proses unggah aset, pemilihan *platform* tujuan, generasi metadata oleh MLLM, serta ekspor data ke dalam format CSV. Sementara itu, kebutuhan nonfungsional meliputi kecepatan respons sistem, kemudahan penggunaan antarmuka, serta kompatibilitas aplikasi.

2. Perancangan Sistem

   Tahap ini mencakup perancangan arsitektur sistem secara menyeluruh, meliputi desain antarmuka pengguna, alur kerja aplikasi, mekanisme komunikasi dengan API layanan MLLM, serta struktur basis data SQLite.

3. Implementasi Sistem

   Pada tahap implementasi, aplikasi dikembangkan menggunakan Python dengan pustaka CustomTkinter. Sistem juga diintegrasikan dengan API dari penyedia MLLM berbasis algoritma *Transformer* untuk menghasilkan metadata secara otomatis, serta dilengkapi dengan fitur ekspor CSV sesuai format yang ditentukan oleh masing-masing *platform*.

4. Pengujian Sistem

   Tahap pengujian dilakukan menggunakan metode black-box testing untuk memastikan seluruh fungsi sistem berjalan sesuai dengan kebutuhan yang telah ditetapkan. Pengujian meliputi fungsi unggah aset, generasi metadata, ekspor CSV, dan pembaruan otomatis aplikasi.

5. Pemeliharaan Sistem

   Tahap pemeliharaan dilakukan untuk memperbaiki kesalahan yang ditemukan setelah sistem digunakan serta melakukan pengembangan fitur tambahan apabila diperlukan. Pemeliharaan juga mencakup mekanisme pembaruan otomatis melalui GitHub Releases agar pengguna dapat memperoleh versi terbaru aplikasi secara langsung.

3.3 Teknik Pengumpulan Data

Teknik pengumpulan data dalam penelitian ini dilakukan dengan pendekatan kualitatif untuk memperoleh informasi yang mendalam terkait proses pengembangan sistem. Adapun teknik yang digunakan meliputi:

1. Observasi Partisipatif (*Participatory Observation*): Observasi dilakukan dengan cara mengamati dan terlibat secara langsung dalam kegiatan operasional pembuatan metadata aset digital di Studio RZ Digital ID. Mengingat peneliti juga merupakan bagian dari tim internal studio tersebut, metode ini memungkinkan peneliti untuk merasakan langsung dinamika kerja kontributor yang dituntut mendistribusikan karya mereka tidak hanya ke Adobe Stock, melainkan juga ke berbagai *platform* lain secara bersamaan demi memaksimalkan pendapatan (seperti Shutterstock, Freepik, dan Vecteezy). Pada tahap ini, peneliti melakukan pencatatan waktu (*time tracking*) terhadap proses penyusunan serta penyesuaian format judul, kata kunci, dan kategori secara manual untuk keempat *platform* yang berbeda. Dari observasi tersebut, didapatkan data faktual bahwa untuk menyelesaikan metadata secara manual bagi 100 aset digital membutuhkan waktu produktif sekitar 17 hingga 33 jam kerja, yang seringkali memicu kelelahan berpikir (burnout) dan penurunan kualitas kata kunci. Fakta faktual inilah yang mendasari urgensi pengembangan sistem otomatisasi multi-*platform* untuk memangkas inefisiensi waktu operasional tersebut.
2. Studi Literatur: Studi literatur dilakukan dengan mengumpulkan referensi dan landasan teori pendukung dari jurnal ilmiah nasional maupun internasional terindeks, serta dokumentasi API teknis. Topik literatur difokuskan pada pengimplementasian metode waterfall, pemanfaatan kecerdasan buatan khususnya *Multimodal Large Language Model* (MLLM) dengan arsitektur algoritma *Transformer*, serta perancangan aplikasi desktop menggunakan pustaka *CustomTkinter* pada Python.
3. Dokumentasi**:** Tahap dokumentasi dilakukan dengan mengumpulkan berbagai data sekunder dan empiris yang digunakan sebagai acuan

pengembangan maupun bahan pengujian (test case) aplikasi. Kegiatan dokumentasi yang dilakukan meliputi:

a. Mengumpulkan 100 sampel aset digital nyata dari Studio RZ Digital ID yang mencakup multiformat, yaitu format gambar (JPG), vektor (EPS), dan video (MOV) untuk digunakan dalam pengujian fungsionalitas sistem.
b. Mengunduh dan mendokumentasikan pedoman resmi dan template berkas CSV dari empat *platform* microstock (Adobe Stock, Shutterstock, Freepik, dan Vecteezy) untuk memastikan format output aplikasi sesuai dengan standar masing-masing *platform*.

3.4 Analisis Sistem

Tahap analisis sistem dilakukan untuk mengevaluasi kinerja sistem lama serta merumuskan perbaikan yang diperlukan. Analisis ini dibagi menjadi tinjauan terhadap sistem yang sedang berjalan dan identifikasi kebutuhan baru pada perangkat lunak.

3.4.1 Analisis Sistem Berjalan

Sebelum penelitian ini dilakukan, Studio RZ Digital ID sudah memiliki aplikasi desktop untuk membuat metadata aset digital secara otomatis. Namun, aplikasi tersebut masih memiliki beberapa keterbatasan yang menjadi dasar dilakukannya pengembangan. Keterbatasan sistem yang sedang berjalan dapat diuraikan sebagai berikut:

1. Aplikasi hanya mendukung satu *platform* microstock, yaitu Adobe Stock. Kontributor yang ingin mengunggah ke Shutterstock, Freepik, atau Vecteezy tetap harus membuat metadata secara manual.
2. Format keluaran CSV hanya tersedia dalam satu format, yaitu format Adobe Stock, sehingga tidak dapat langsung digunakan untuk *platform* lain yang memiliki ketentuan kolom dan aturan penulisan metadata yang berbeda.

3. Aplikasi hanya menerima satu *API Key* untuk setiap penyedia layanan MLLM, sehingga proses pembuatan metadata dilakukan satu per satu secara berurutan. Hal ini menyebabkan waktu pemrosesan menjadi lebih lama ketika jumlah aset yang harus diproses cukup banyak.
4. Belum tersedia fitur penamaan ulang berkas secara massal (*bulk rename*), sehingga kontributor harus mengganti nama berkas satu per satu sebelum mengunggahnya ke *platform*.
5. Belum tersedia sistem pembaruan otomatis, sehingga pengguna harus mengunduh ulang aplikasi secara manual dari situs web setiap kali tersedia versi terbaru.

Berdasarkan keterbatasan tersebut, diperlukan pengembangan sistem agar aplikasi dapat mendukung empat *platform* microstock sekaligus, memproses metadata secara lebih cepat, serta menyediakan fitur tambahan yang dapat menunjang produktivitas kontributor.

3.4.2 Analisis Kebutuhan Fungsional

Kebutuhan fungsional merupakan fungsi atau fitur yang harus tersedia di dalam sistem agar aplikasi dapat berjalan sesuai tujuan penelitian. Berikut adalah daftar kebutuhan fungsional yang telah diidentifikasi:

Tabel 3. 1 Kebutuhan Fungsional

Kode	Kebutuhan	Keterangan
F-01	Memilih *platform* microstock	Pengguna dapat memilih satu atau lebih *platform* tujuan, yaitu Adobe Stock, Shutterstock, Freepik, dan Vecteezy
F-02	Mengunggah aset digital	Pengguna dapat menambahkan berkas gambar (JPG, PNG, PSD), vektor (EPS, SVG), atau video (MP4, MOV) ke dalam aplikasi melalui tombol unggah atau seret dan lepas (drag and drop)
F-03	Mengatur API Key	Pengguna dapat menyimpan satu atau lebih API Key untuk setiap penyedia layanan MLLM melalui menu pengaturan
F-04	Memilih penyedia dan model AI	Pengguna dapat memilih penyedia layanan MLLM (seperti Google, OpenAI,) beserta model lainnya

Kode	Kebutuhan	Keterangan
F-05	Menulis perintah tambahan	Pengguna dapat menambahkan kata kunci atau instruksi khusus yang harus disertakan dalam hasil metadata
F-06	Membuat metadata secara otomatis	Sistem mengirimkan gambar ke API MLLM dan menerima hasil metadata berupa judul, kata kunci, dan kategori sesuai format *platform* yang dipilih
F-07	Memproses metadata secara paralel	Apabila pengguna memasukkan lebih dari satu API Key, sistem secara otomatis membagi pekerjaan ke beberapa worker yang bekerja bersamaan sehingga proses menjadi lebih cepat
F-08	Mengekspor berkas CSV	Pengguna dapat menyimpan hasil metadata ke dalam berkas CSV sesuai format masing-masing *platform* yang siap diunggah langsung
F-09	Mengubah nama berkas secara massal	Pengguna dapat mengganti nama banyak berkas aset sekaligus melalui halaman bulk rename
F-10	Memeriksa pembaruan aplikasi	Sistem secara otomatis memeriksa ketersediaan versi terbaru melalui GitHub Releases dan memungkinkan pengguna memperbarui aplikasi tanpa mengunduh ulang secara manual
F-11	Melihat pratinjau & sunting manual	Pengguna dapat melakukan klik ganda pada aset untuk melihat pratinjau ukuran besar dan menyunting judul/kata kunci secara manual, dilengkapi dengan penghitung karakter interaktif.
F-12	Mengatur batasan karakter	Pengguna dapat mengatur batas minimum dan maksimum panjang karakter judul serta jumlah batas kata kunci melalui menu pengaturan.
F-13	Memantau penggunaan token	Pengguna dapat memantau estimasi penggunaan token (token usage) dari layanan API kecerdasan buatan secara real-time pada antarmuka aplikasi

3.4.3 Analisis Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan aspek di luar fitur utama yang turut menentukan kualitas dan kenyamanan penggunaan sistem. Berikut adalah kebutuhan non-fungsional yang telah diidentifikasi

Tabel 3. 2 Kebutuhan Non Fungsional

Kode	Kebutuhan	Keterangan
NF-01	Koneksi internet	Sistem memerlukan koneksi internet yang stabil untuk berkomunikasi dengan layanan API MLLM
NF-02	Waktu respons	Proses pembuatan metadata untuk satu aset tidak lebih dari 5-10 detik dalam kondisi jaringan normal

Kode	Kebutuhan	Keterangan
NF-03	Kompatibilitas	Aplikasi dapat berjalan pada sistem operasi Windows 10 dan Windows 11
NF-04	Keamanan Transmisi dan Penyimpanan	API Key pengguna dan seluruh riwayat metadata disimpan secara eksklusif pada pangkalan data SQLite lokal di perangkat pengguna (tanpa database terpusat). Selain itu, seluruh proses pertukaran data (termasuk pengiriman *prompt* dan gambar) antara aplikasi dengan server API MLLM dilakukan menggunakan protokol terenkripsi (HTTPS/TLS) untuk mencegah penyadapan data (Man-in-the-Middle Attack).
NF-05	Privasi Aset Digital Pengguna	Sistem menjamin privasi karya pengguna dengan cara tidak pernah mengunggah aset digital (foto/video) ke server mana pun milik pengembang aplikasi. Aset hanya diubah menjadi format kode Base64 secara lokal, lalu dikirimkan secara langsung (sekali jalan) ke penyedia layanan API MLLM resmi tanpa ada penyimpanan bayangan (shadow server).
NF-06	Integritas dan Perlindungan Kode Sistem	Aplikasi didistribusikan dalam bentuk file executable (.exe) yang telah dikompilasi (Compiled Application). Pendekatan ini tidak hanya meniadakan kebutuhan pengguna untuk menginstal environment Python secara mandiri, tetapi juga menjaga source code asli. Hal ini mencegah pihak yang tidak bertanggung jawab untuk membaca, memodifikasi (tampering), atau menyisipkan baris kode berbahaya ke dalam logika utama aplikasi.
NF-07	Kemudahan penggunaan	Antarmuka dirancang sederhana dan mudah dipahami sehingga pengguna tidak memerlukan pelatihan khusus untuk mengoperasikan aplikasi

3.5 Perancangan Sistem

Tahap perancangan sistem memvisualisasikan alur, struktur, dan antarmuka aplikasi yang akan dibangun. Perancangan ini dimodelkan menggunakan pendekatan *Unified Modeling Language* (UML) serta perancangan tata letak visual (mockup) antarmuka pengguna.

3.5.1 Identifikasi Aktor

Langkah awal dalam perancangan sistem adalah menentukan siapa saja pihak yang akan berinteraksi langsung dengan aplikasi RZ Autometadata.

Pihak-pihak tersebut dalam pemodelan *UML* disebut sebagai aktor. Setelah dilakukan analisis terhadap kebutuhan sistem pada sub-bab sebelumnya, diperoleh dua aktor utama yang terlibat dalam penggunaan dan pengelolaan sistem ini.

Tabel 3. 3 Jumlah Aktor

No	Aktor	Hak dan Akses Sistem
1	User (Kontributor)	Mengunggah aset digital, memilih *platform* microstock, mengatur API Key, memilih provider dan model AI, menulis custom prompt, menjalankan generate metadata, mengekspor berkas CSV, menggunakan fitur bulk rename, serta menerima notifikasi pembaruan aplikasi
2	Admin	Memelihara sistem dan merilis pembaruan (update) aplikasi ke server apabila terdapat perbaikan, penyesuaian, atau penambahan fitur

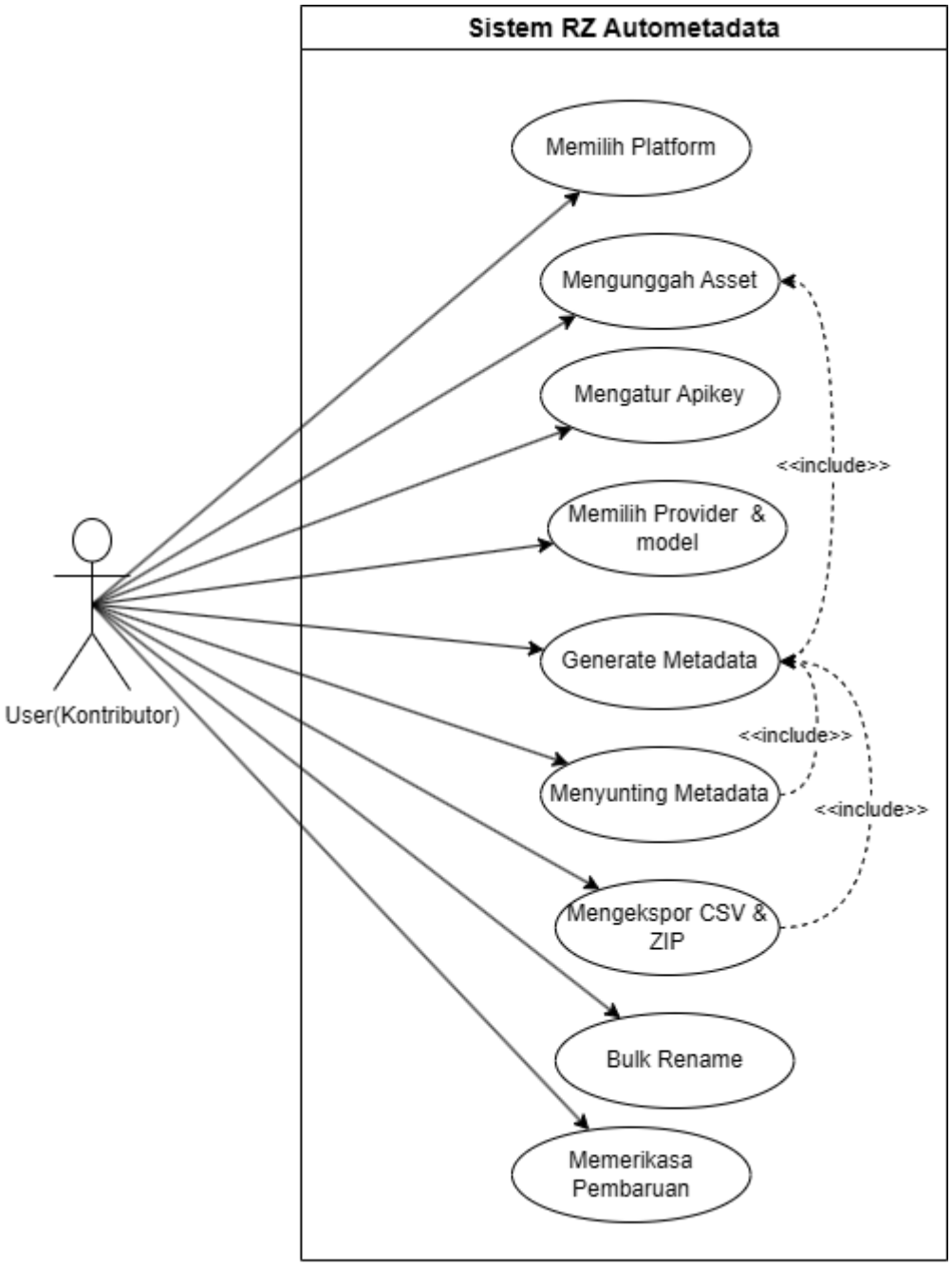
Aktor pertama adalah *User*, yaitu kontributor microstock yang menjadi pengguna utama aplikasi. *User* merupakan pihak yang secara langsung mengoperasikan seluruh fitur inti aplikasi, mulai dari mengunggah aset digital, memilih *platform* tujuan, mengatur konfigurasi *API Key* dan model *AI*, menjalankan proses pembuatan metadata otomatis, hingga mengekspor hasil metadata ke dalam berkas *CSV*. Selain itu, *user* juga dapat memanfaatkan fitur tambahan seperti penulisan *custom prompt* untuk menambahkan konteks tertentu pada metadata yang dihasilkan, serta fitur *bulk rename* untuk mengganti nama berkas secara massal.

Aktor kedua adalah *Admin*, yaitu pihak yang bertanggung jawab atas pemeliharaan dan keberlangsungan sistem. Peran *admin* tidak berkaitan dengan proses pembuatan metadata, melainkan berfokus pada pengelolaan pembaruan aplikasi. Ketika ditemukan kesalahan (*bug*), diperlukan penyesuaian terhadap perubahan aturan *platform* microstock, atau terdapat penambahan fitur baru, *admin* bertugas untuk menyiapkan dan merilis versi terbaru aplikasi ke *server* pembaruan. Aplikasi yang terpasang di sisi *user* kemudian akan mendeteksi ketersediaan versi baru tersebut secara otomatis saat dibuka.

3.5.2 *Use Case* Diagram

Use Case Diagram digunakan untuk menggambarkan hubungan antara pengguna (aktor) dengan fitur-fitur yang tersedia di dalam sistem. Dalam aplikasi RZ Autometadata, terdapat satu aktor utama yaitu Kontributor yang merupakan pengguna aplikasi. Kontributor dapat mengakses seluruh fitur yang disediakan oleh sistem. Berikut adalah *Use Case* Diagram dari sistem yang dirancang:

1. *Use Case* Diagram User (Kontributor)

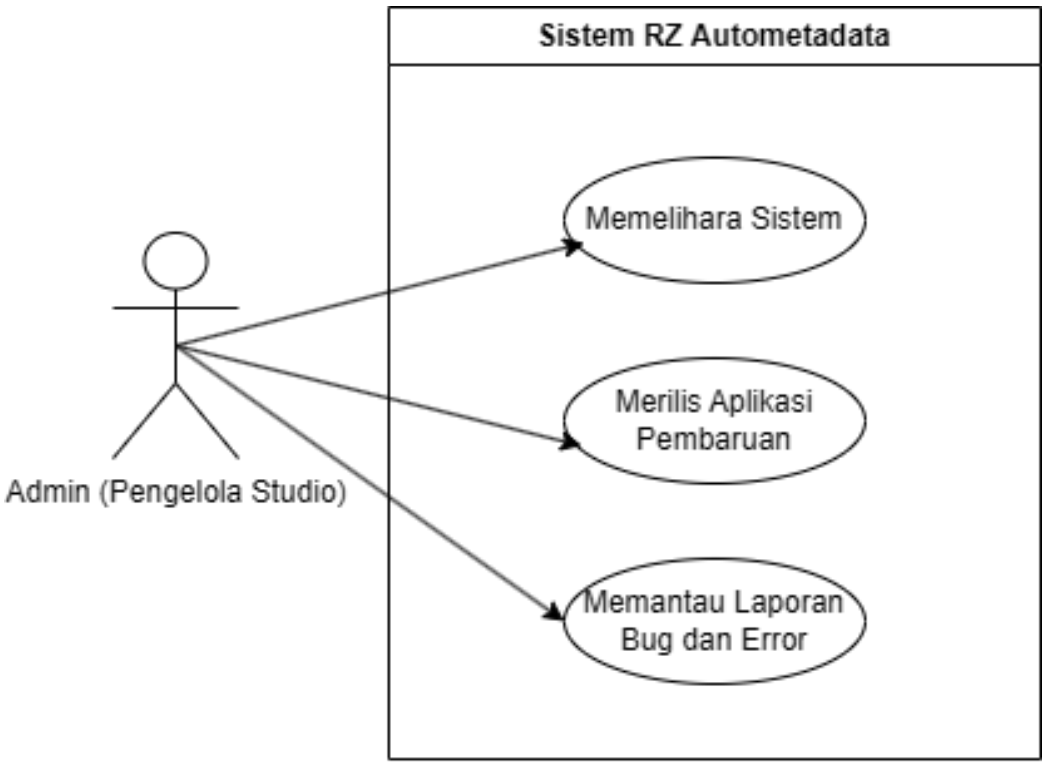


Gambar 3. 3 *Use Case* Diagram User (Kontributor)

Berdasarkan Diagram tersebut, Kontributor dapat melakukan sembilan aktivitas utama di dalam sistem. Terdapat dua hubungan <<include>> yang menunjukkan ketergantungan antar fitur, yaitu fitur Generate Metadata yang memerlukan aset untuk diunggah terlebih

dahulu, serta fitur Mengekspor CSV yang memerlukan metadata untuk dibuat terlebih dahulu sebelum dapat diekspor.

2. *Use Case* Diagram Admin (Pengelola Studio)



Gambar 3. 4 *Use Case* Diagram Admin (Pengelola Studio)

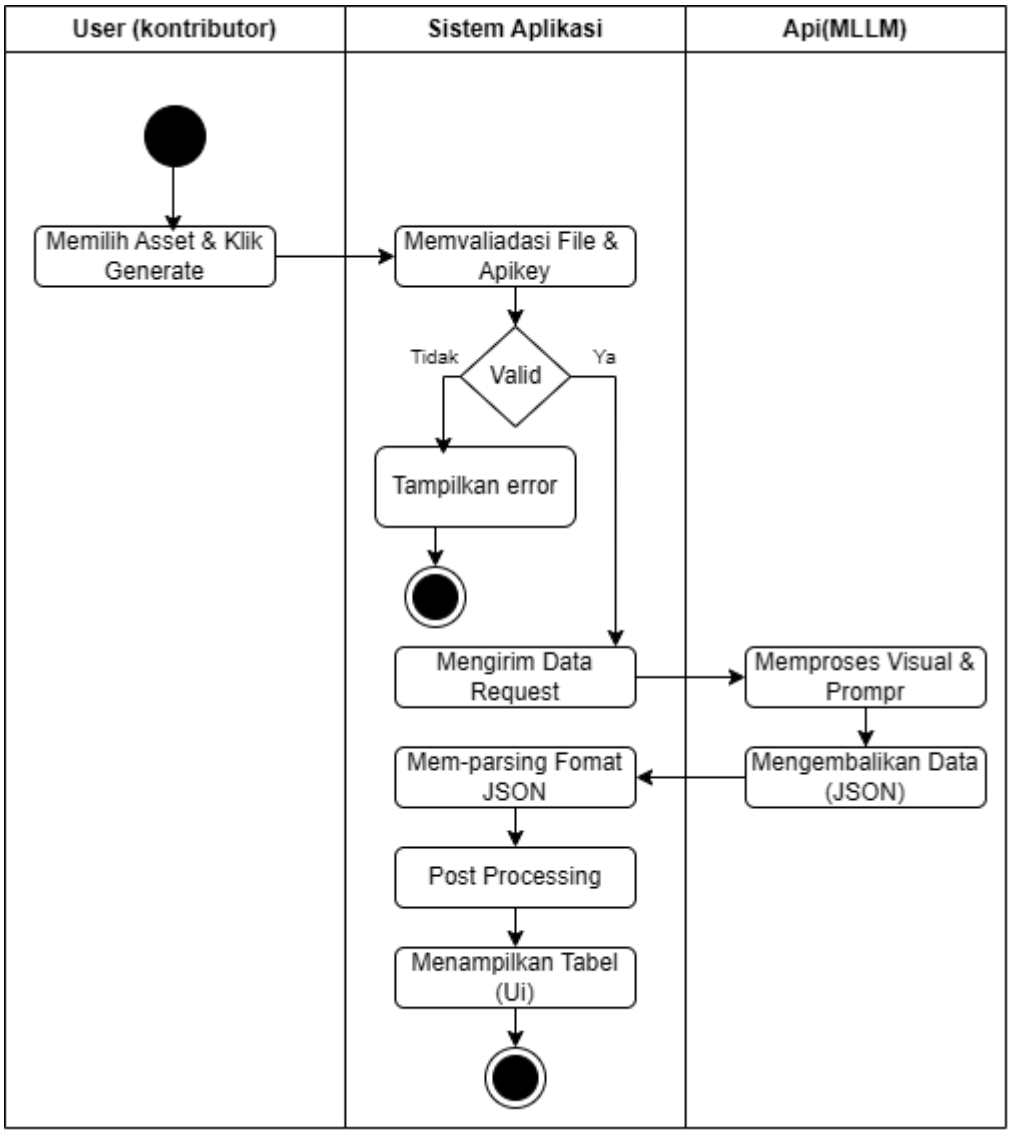
Berbeda dengan User, aktor Admin (Pengelola Studio) memiliki peran yang berfokus pada sisi dukungan (support) dan pemeliharaan aplikasi. Berdasarkan Diagram di atas, Admin memiliki tiga aktivitas utama yang bersifat independen (tanpa ada hubungan <<include>>), yaitu memelihara sistem aplikasi secara berkala, merilis pembaruan (update) aplikasi versi terbaru ke server repositori, serta memantau jika ada laporan bug maupun error dari pengguna. Seluruh aktivitas ini dilakukan di luar antarmuka operasional kontributor agar aplikasi RZ Autometadata dapat terus digunakan dengan stabil.

3.5.3 *Activity* Diagram

Activity Diagram digunakan untuk menggambarkan alur aktivitas interaksi yang dilakukan oleh pengguna (aktor) saat menggunakan aplikasi RZ Autometadata dari awal hingga selesai. Mengingat kompleksitas proses di dalam aplikasi, *Activity* Diagram pada sistem ini dipecah menjadi tujuh bagian spesifik berdasarkan fungsionalitas utamanya.

Pemodelan ini dirancang menggunakan format *Swimlanes* untuk memberikan batas yang tegas (*boundary*) mengenai tanggung jawab dari masing-masing pihak, yaitu interaksi Aktor, Sistem Aplikasi Desktop, Database Lokal (SQLite), dan Server Eksternal (API MLLM atau GitHub). Berikut adalah tujuh *Activity* Diagram dari sistem yang dirancang:

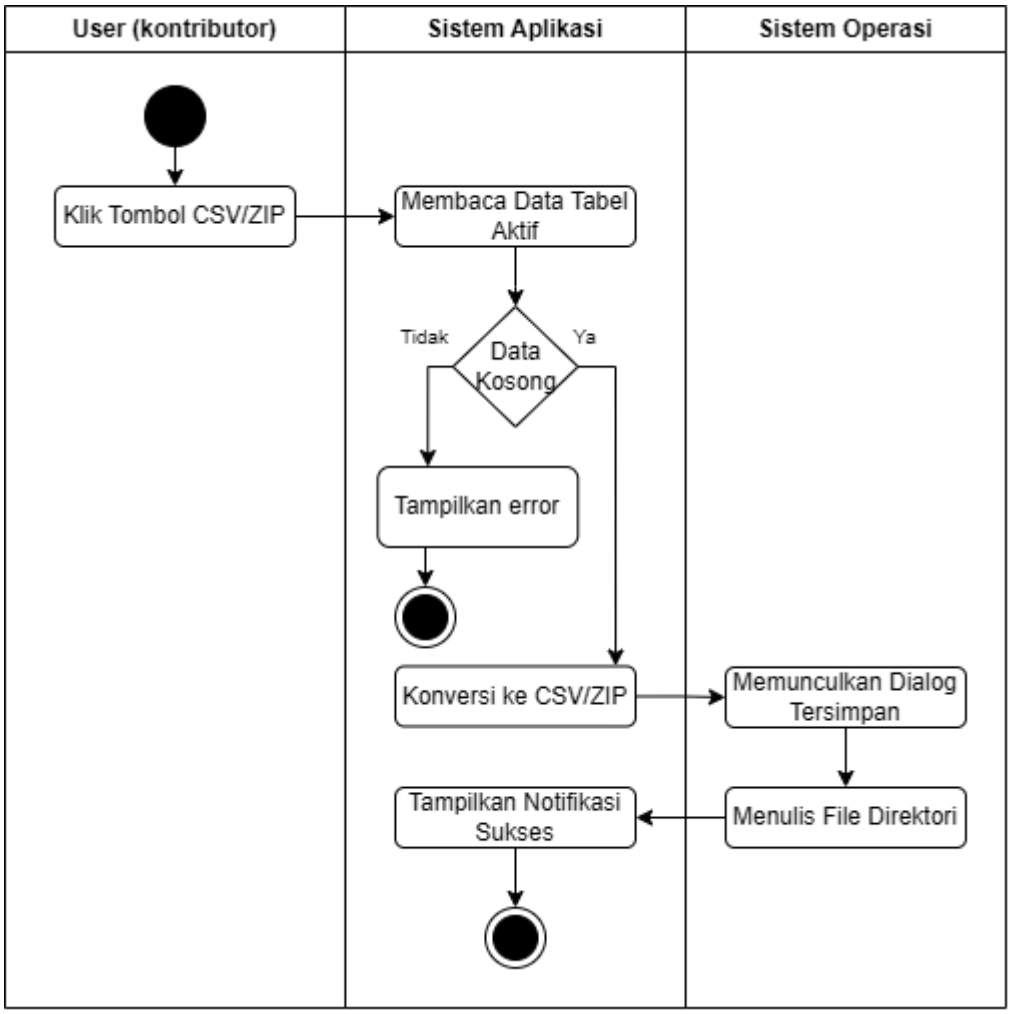
1. *Activity* Diagram Generate Metadata



Gambar 3. 5 *Activity* Diagram Generate Metadata

Gambar 3.5 di atas menggambarkan alur inti aplikasi di mana *User* mengunggah aset digital, lalu sistem mengubahnya menjadi format *Base64* dan mengirimkannya ke server MLLM (Google/*OpenAI*/*Groq*) untuk dianalisis. Sistem kemudian mem-*parsing* respons *JSON* dari MLLM menjadi metadata, menyimpannya ke database, dan menampilkannya pada tabel *UI*.

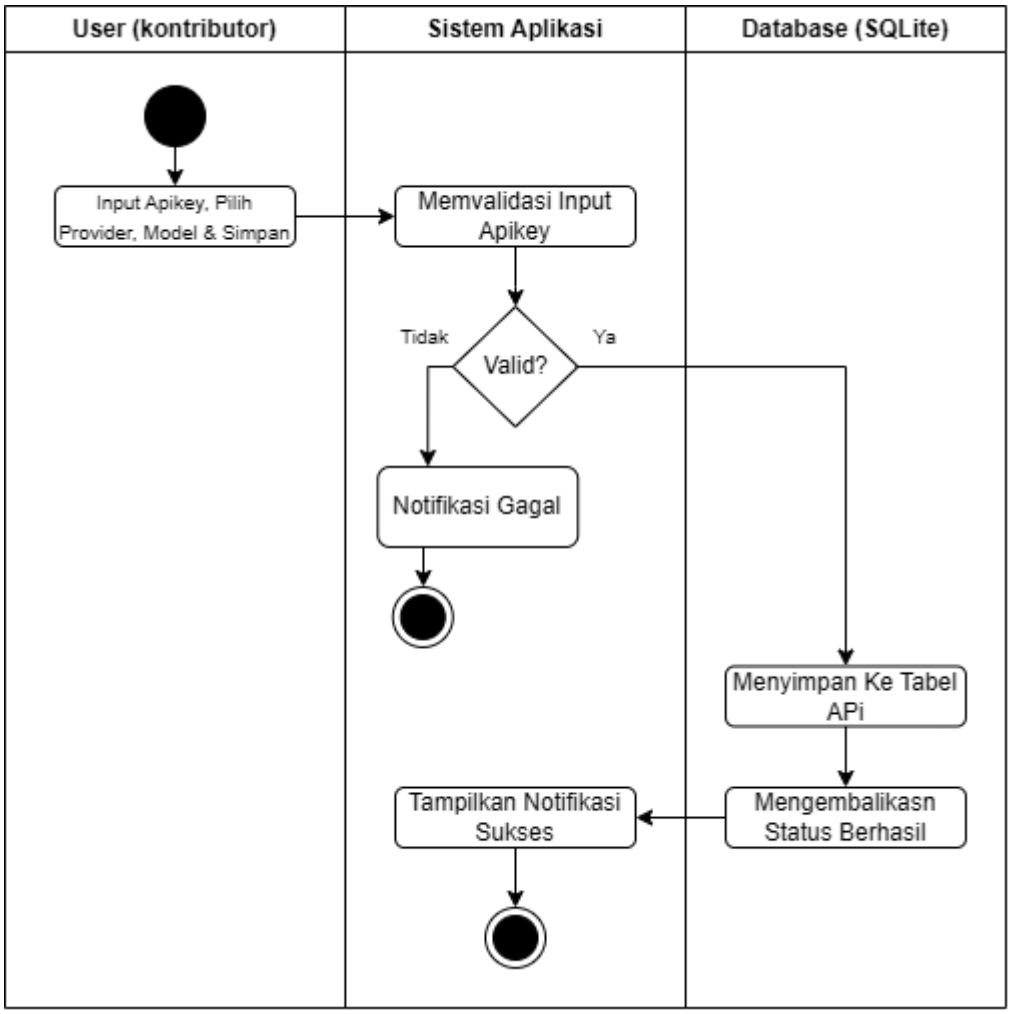
2. *Activity* Diagram Mengekspor CSV



Gambar 3. 6 *Activity* Diagram Mengekspor CSV

Gambar 3.6 di atas menunjukkan alur ketika *User* menekan tombol ekspor. Sistem akan mengambil seluruh data metadata yang berstatus selesai dari database, kemudian memformat ulang (*re-formatting*) struktur kolomnya secara dinamis mengikuti standar empat *platform* microstock (Adobe Stock, Shutterstock, Freepik, Vecteezy), lalu menyimpannya dalam bentuk berkas .CSV ke perangkat lokal *User*.

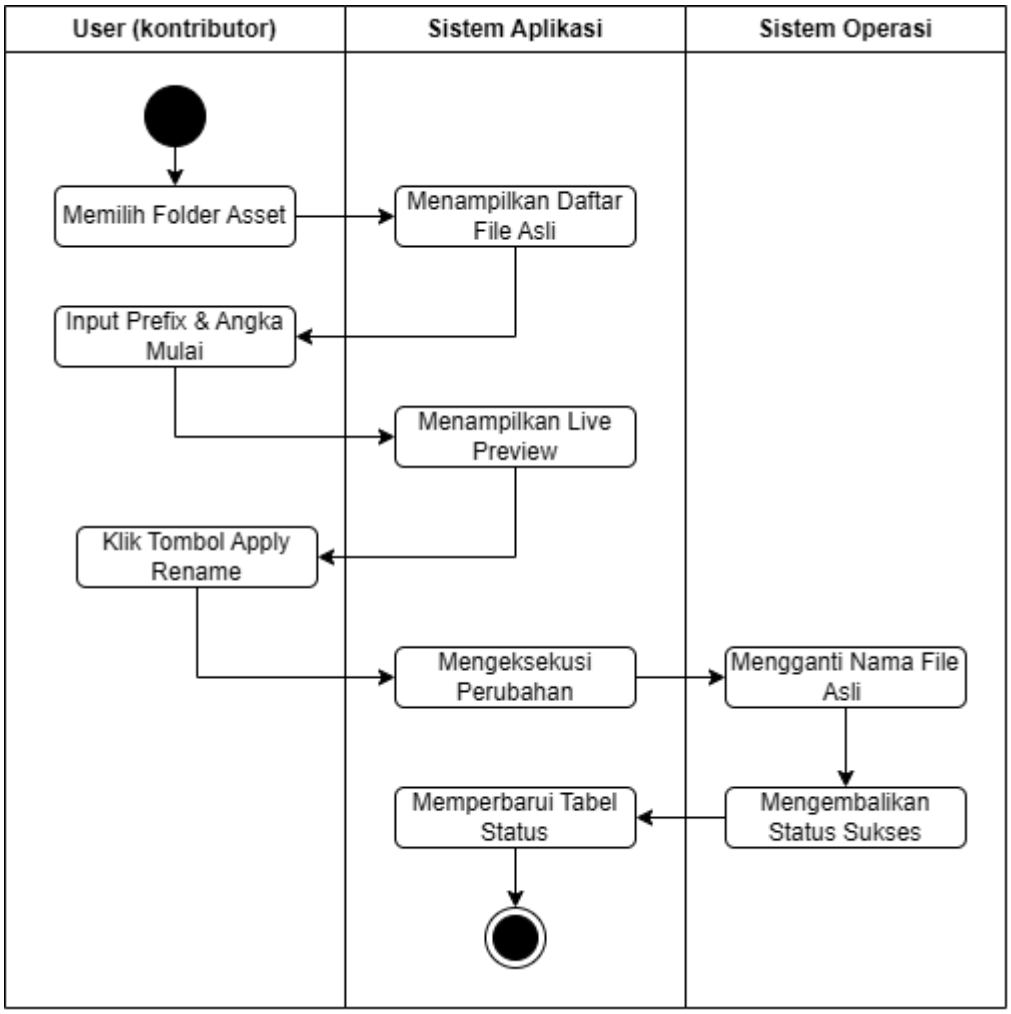
3. *Activity* Diagram Mengatur API Key & Provider



Gambar 3. 7 *Activity* Diagram Mengatur API Key & Provider

Gambar 3.7 di atas memodelkan proses ketika *User* melakukan input *API Key*, memilih *provider*, dan menentukan model MLLM yang ingin digunakan. Sistem akan memvalidasi input tersebut dan menyimpannya secara persisten ke dalam database agar pengaturan tidak hilang saat aplikasi ditutup.

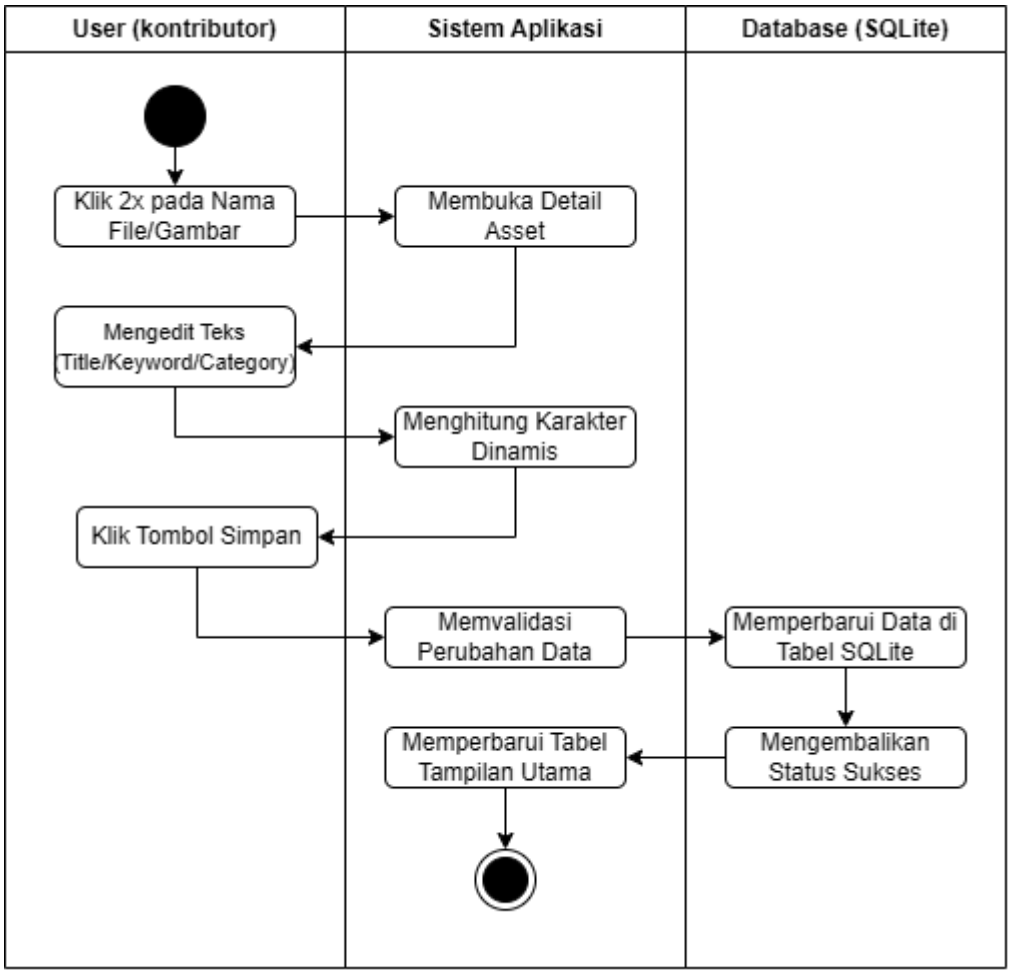
4. *Activity* Diagram Bulk Rename



Gambar 3. 8 *Activity* Diagram Bulk Rename

Gambar 3.8 di atas menjelaskan alur fitur penggantian nama berkas secara massal. Saat *User* mengeksekusi fitur ini, sistem akan membaca judul dari metadata di database, kemudian mengeksekusi perubahan nama berkas fisik (*file renaming*) di dalam sistem operasi lokal, serta mensinkronkan ulang jalur berkas (*file path*) terbaru ke database.

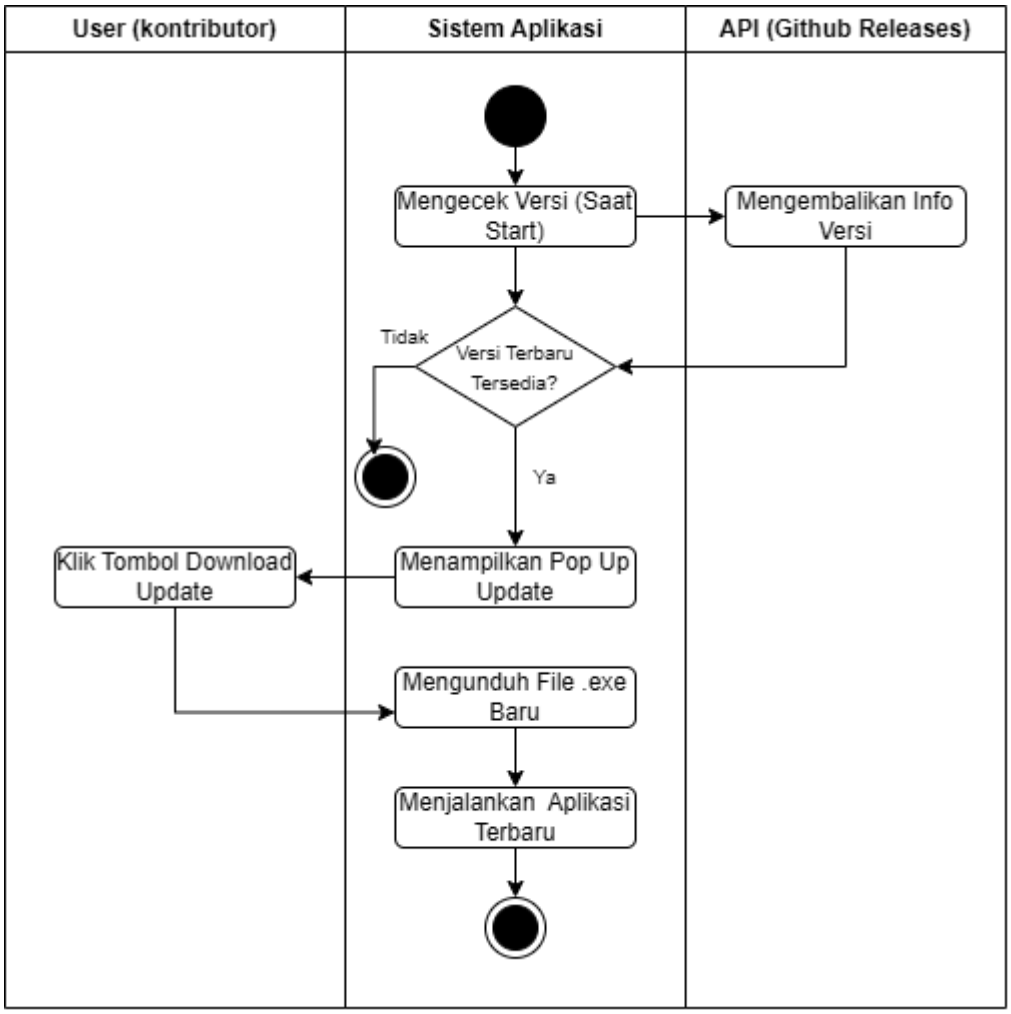
5. *Activity* Diagram Menyunting Metadata



Gambar 3. 9 *Activity* Diagram Menyunting Metadata

Gambar 3.9 di atas menggambarkan proses penyuntingan secara manual. Jika *User* merasa hasil deskripsi dari *AI* kurang tepat, *User* dapat mengeklik ganda pada baris tabel untuk mengubahnya. Selama proses pengetikan, antarmuka sistem akan menghitung pembatasan jumlah karakter secara *real-time*, sebelum akhirnya divalidasi dan disimpan ke database.

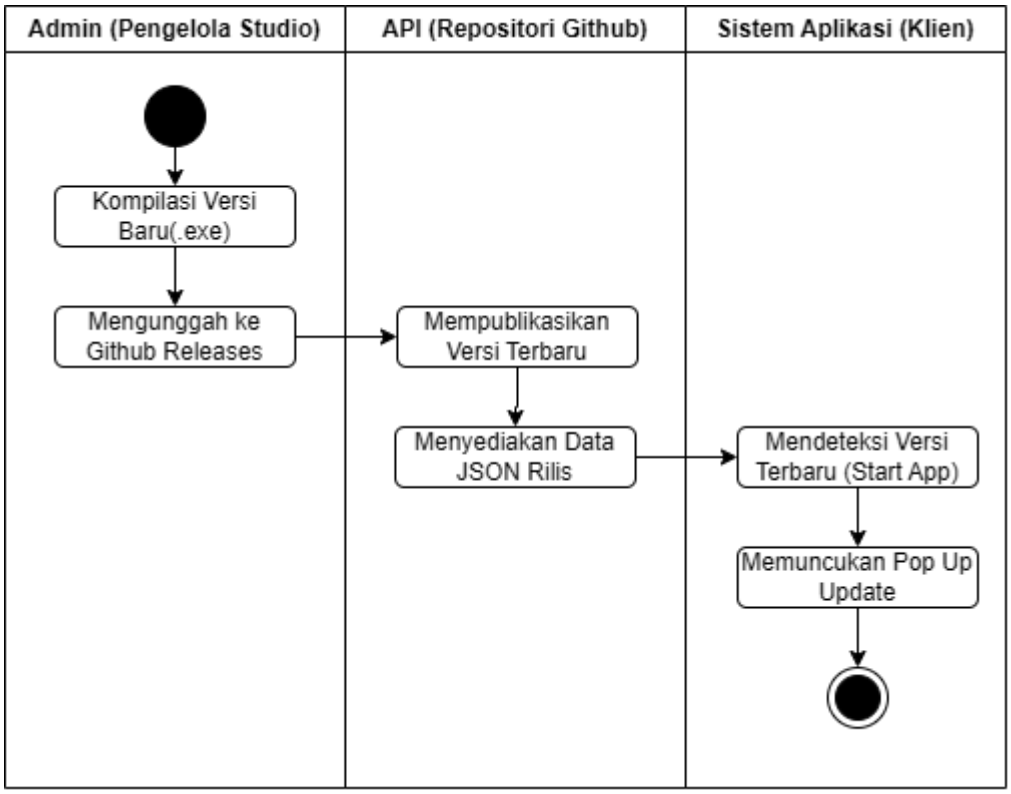
6. *Activity* Diagram Memeriksa Pembaruan (Auto-Update)



Gambar 3. 10 *Activity* Diagram Memeriksa Pembaruan (Auto-Update)

Gambar 3.10 di atas memodelkan pengecekan pembaruan otomatis. Saat *User* membuka aplikasi, *thread* latar belakang pada sistem akan melakukan permintaan (*HTTP GET*) secara *asinkronus* ke server *GitHub Releases*. Jika sistem mendeteksi adanya versi baru (*version comparison*), sistem akan memunculkan dialog notifikasi kepada *User*.

7. *Activity* Diagram Merilis Pembaruan (Khusus Admin)



Gambar 3. 11 *Activity* Diagram Merilis Pembaruan (Khusus Admin)

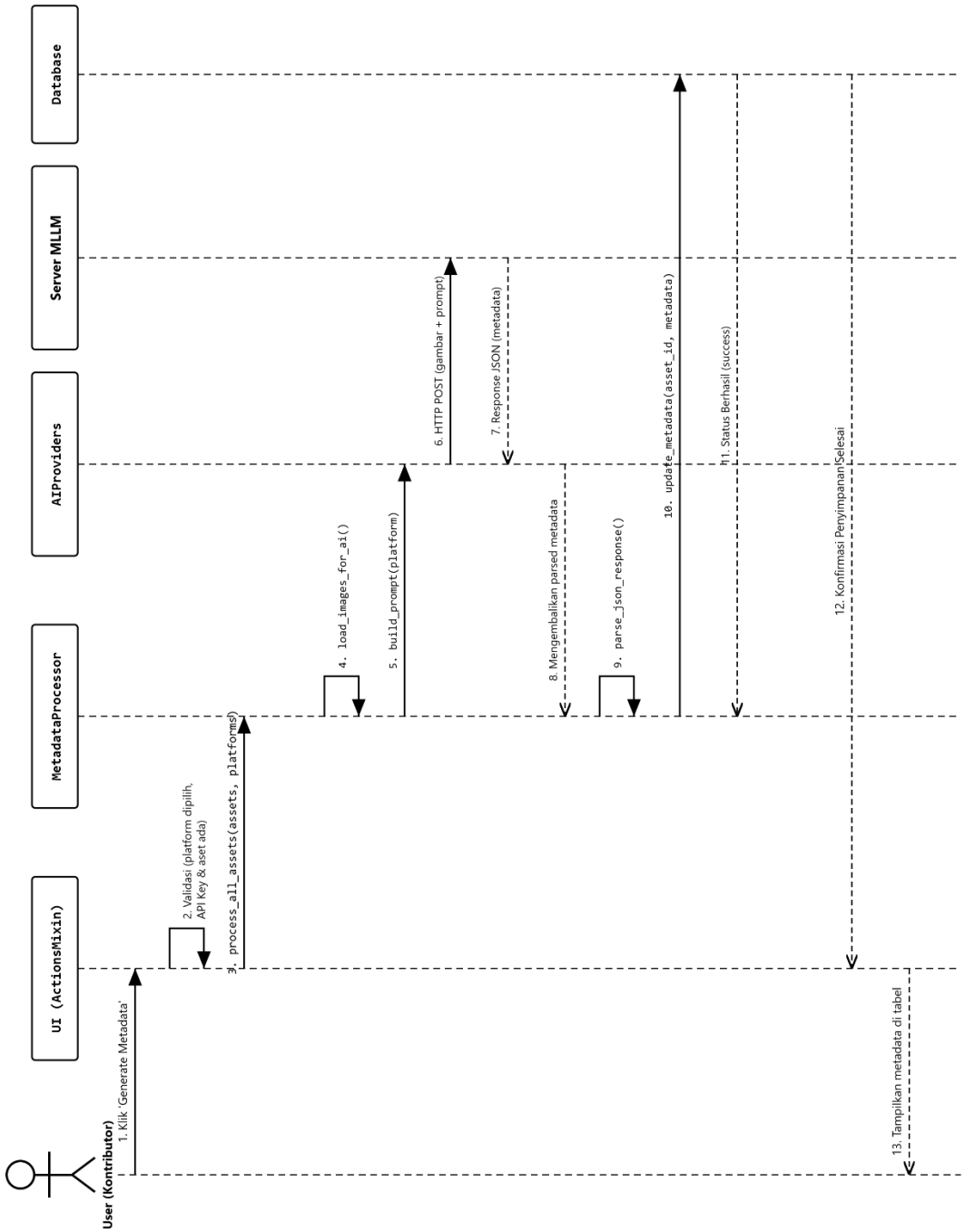
Gambar 3.11 di atas merupakan aktivitas yang didedikasikan khusus untuk aktor Admin. Diagram ini memodelkan alur kerja Admin di luar antarmuka sistem utama, di mana Admin melakukan proses kompilasi versi terbaru dari aplikasi dan merilis berkas instalasinya ke *repository GitHub Releases* agar dapat dijangkau oleh deteksi sistem pada perangkat *User*.

3.5.4 *Sequence* Diagram

Setelah mengetahui alur aktivitas pengguna, langkah selanjutnya adalah memodelkan bagaimana objek-objek sistem saling berkomunikasi ketika suatu proses dijalankan. Pemodelan ini menggunakan *Sequence* Diagram, yaitu Diagram UML yang menunjukkan urutan pesan (message) yang dikirim antar objek secara kronologis. Pada sistem RZ Autometadata, terdapat dua

proses utama yang melibatkan interaksi antarkomponen secara kompleks, sehingga dimodelkan dalam dua *Sequence* Diagram tersendiri.

1. *Sequence* Diagram Proses Generate Metadata

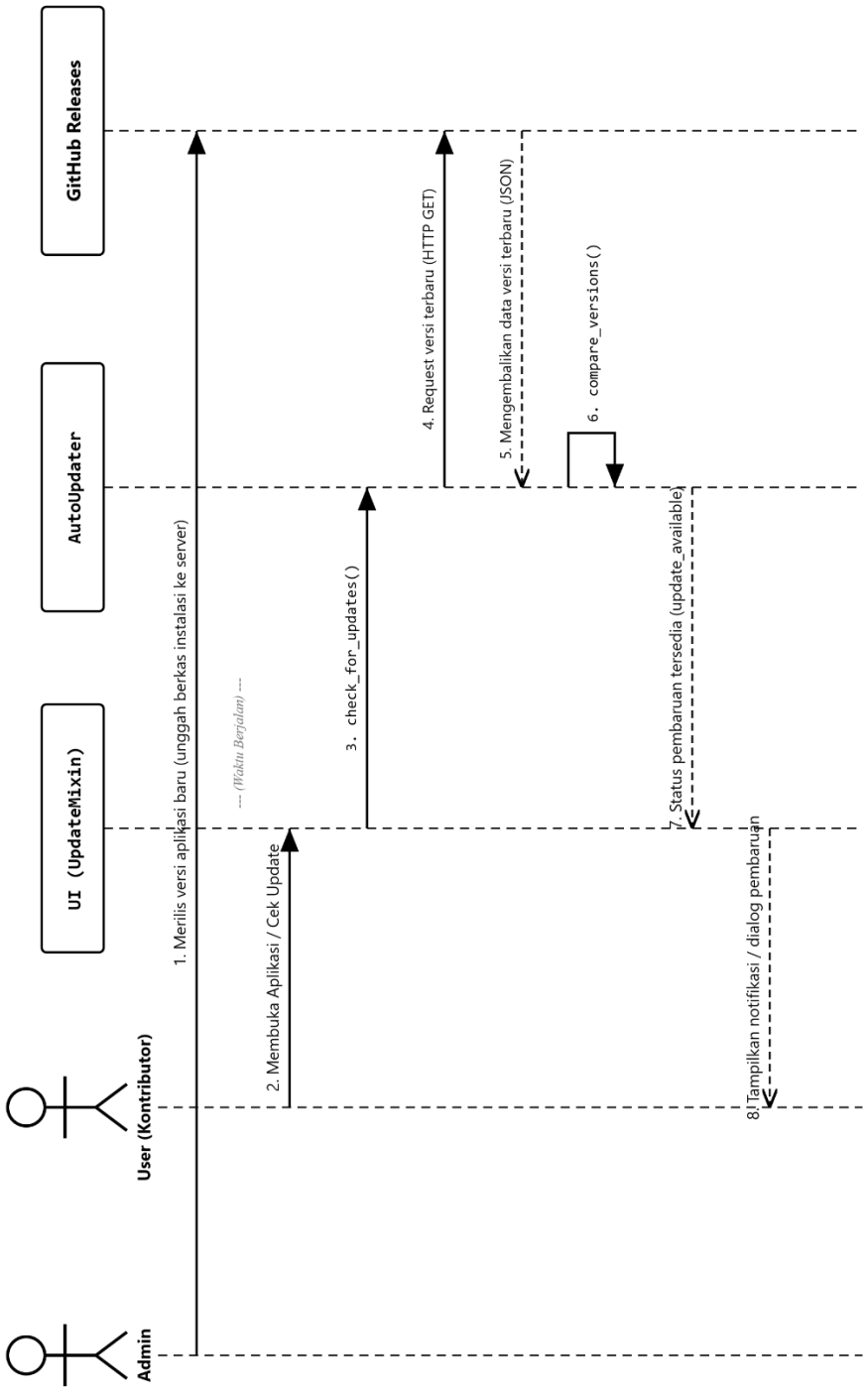


Gambar 3. 12 *Sequence* Diagram Proses Generate Metadata

Gambar 3.12 di atas mengilustrasikan proses *generate* metadata yang merupakan fungsi inti dari sistem. Ketika *User* mengeklik tombol, sistem akan memvalidasi kesiapan *input*. Jika validasi berhasil, antarmuka (*UI*) akan memanggil fungsi *process_all_assets()* pada modul *MetadataProcessor*.

Di dalam tahapan tersebut, sistem akan menjalankan proses perulangan (*loop*) untuk setiap kombinasi aset dan *platform* yang dipilih. *MetadataProcessor* akan memuat gambar melalui *load_images_for_ai()*, lalu modul *AIProviders* menyusun teks perintah (*prompt*) menggunakan *build_prompt()*. Keduanya kemudian dikirimkan ke server *MLLM* melalui protokol *HTTP POST*. Server *MLLM* mengembalikan respons *JSON* berisi judul, kata kunci, dan kategori yang kemudian di-*parse* oleh *AIProviders*. Pada tahap akhir, hasil akhir ini disimpan ke dalam database lokal menggunakan fungsi *update_metadata()* dan ditampilkan pada antarmuka tabel.

2. *Sequence* Diagram Proses Pembaruan Aplikasi



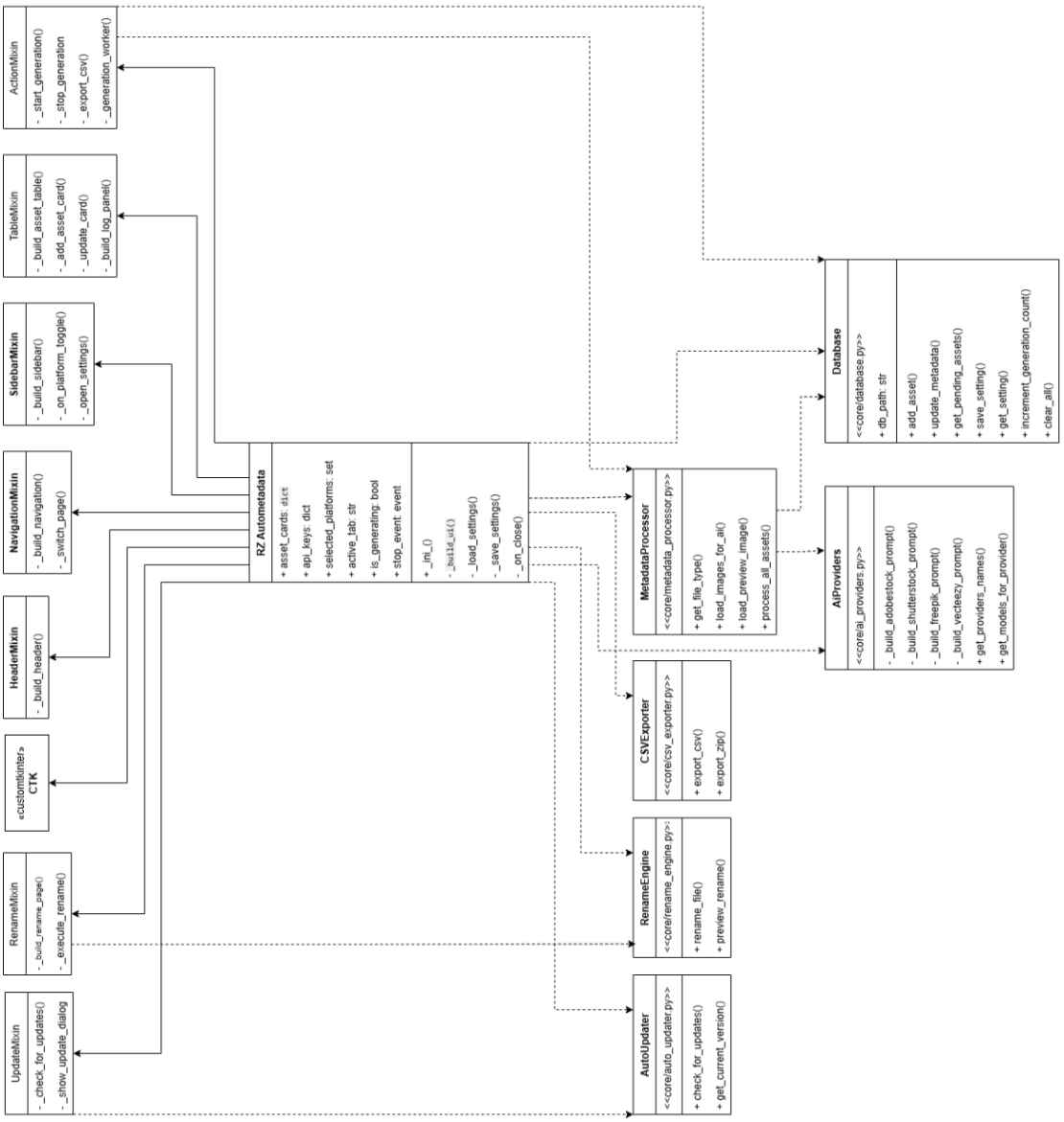
Gambar 3. 13 *Sequence* Diagram Proses Pembaruan Aplikasi

Gambar 3.13 di atas mengilustrasikan proses pembaruan aplikasi yang melibatkan dua aktor, yaitu Admin dan *User*. Proses asinkronus ini diawali oleh Admin yang merilis versi terbaru aplikasi dengan mengunggah berkas instalasinya ke server *GitHub Releases*.

Di sisi lain, pada saat *User* membuka aplikasi, antarmuka *UI* akan memanggil fungsi *check_for_updates()* pada modul *AutoUpdater*. Modul ini bertugas mengirimkan *HTTP GET* ke *API GitHub* untuk mengambil metadata rilis terbaru. Setelah menerima data *JSON* dari server, *AutoUpdater* mengeksekusi komparasi versi. Jika sistem mendeteksi bahwa versi dari server lebih tinggi dibandingkan versi lokal yang sedang berjalan, sistem akan mengirimkan instruksi ke *UI* untuk memunculkan dialog notifikasi yang menawarkan *User* untuk mengunduh versi terbaru tersebut.

3.5.5 *Class* Diagram

Class Diagram digunakan untuk menggambarkan struktur internal aplikasi RZ Autometadata secara teknis. Diagram ini memperlihatkan kelas-kelas yang menyusun sistem, atribut dan *method* yang dimiliki, serta hubungan antarkelas tersebut. Aplikasi ini dibangun menggunakan bahasa pemrograman *Python* dengan pendekatan arsitektur modular.



Gambar 3. 14 *Class* Diagram

Berdasarkan Gambar 3.14 diatas, kelas utama bernama RZ Autometadata bertindak sebagai *entry point* sekaligus pengendali utama aplikasi. Kelas ini mewarisi (*inheritance*) dari kelas *CTk* milik *library CustomTkinter* sebagai fondasi antarmuka grafis. Struktur kelas pada sistem ini dibagi menjadi dua lapisan utama, yaitu:

1. Lapisan Antarmuka (UI Layer)

   Untuk mencegah penumpukan kode pada satu berkas, antarmuka aplikasi dipecah menjadi tujuh kelas mixin yang mewariskan fungsinya ke kelas utama RZAutometadata:

   a. *HeaderMixin*: Menangani pembuatan bagian kepala aplikasi yang memuat judul dan logo.
   b. *NavigationMixin*: Mengelola navigasi perpindahan antarhalaman di dalam aplikasi.
   c. *SidebarMixin*: Membangun panel samping yang berisi *checkbox* pemilihan *platform* dan akses pengaturan.
   d. *TableMixin*: Menangani tampilan tabel metadata dan panel log secara *real-time*.
   e. *ActionsMixin*: Mengatur logika tombol-tombol aksi utama (*Generate*, *Stop*, *Export CSV*) dan menjalankan proses *thread* terpisah agar tampilan antarmuka tidak membeku (*freeze*).
   f. *RenameMixin*: Membangun halaman *Bulk Rename* beserta antarmuka penamaan ulang berkas secara massal.

      UpdateMixin: Menangani dialog notifikasi pengecekan pembaruan aplikasi saat pertama kali dibuka.

2. Lapisan Inti (Backend Layer)

   Di luar antarmuka, terdapat enam kelas pada lapisan inti yang menangani logika bisnis dan pemrosesan data. Kelas-kelas ini tidak diwarisi, melainkan digunakan (dependency) oleh UI Layer:

   a. *Database* (*core/database.py*): Mengelola penyimpanan data aset dan pengaturan menggunakan *SQLite* lokal. *Method* utamanya meliputi *add_asset()*, *update_metadata()*, dan *save_setting()*.
   b. *MetadataProcessor* (*core/metadata_processor.py*):

      Bertanggung jawab memuat berkas aset (gambar, vektor,

video) dan mengoordinasikan seluruh alur pemrosesan *AI* melalui fungsi *process_all_assets()*.

c. *AIProviders* (*core/ai_providers.py*): Menyusun *prompt* sesuai *platform*, mengirimkan permintaan ke *API*, dan mem-*parse* respons *JSON*. Kelas ini mendukung *provider MLLM* seperti Google (Gemini), *OpenAI*, dan *Groq*.

d. *CSVExporter* (*core/csv_exporter.py*): Mengekspor metadata dari database ke dalam berkas *CSV* sesuai standar *platform* microstock.

e. *AutoUpdater* (*core/auto_updater.py*): Berkomunikasi dengan *API GitHub Releases* untuk membandingkan dan memeriksa ketersediaan versi terbaru.

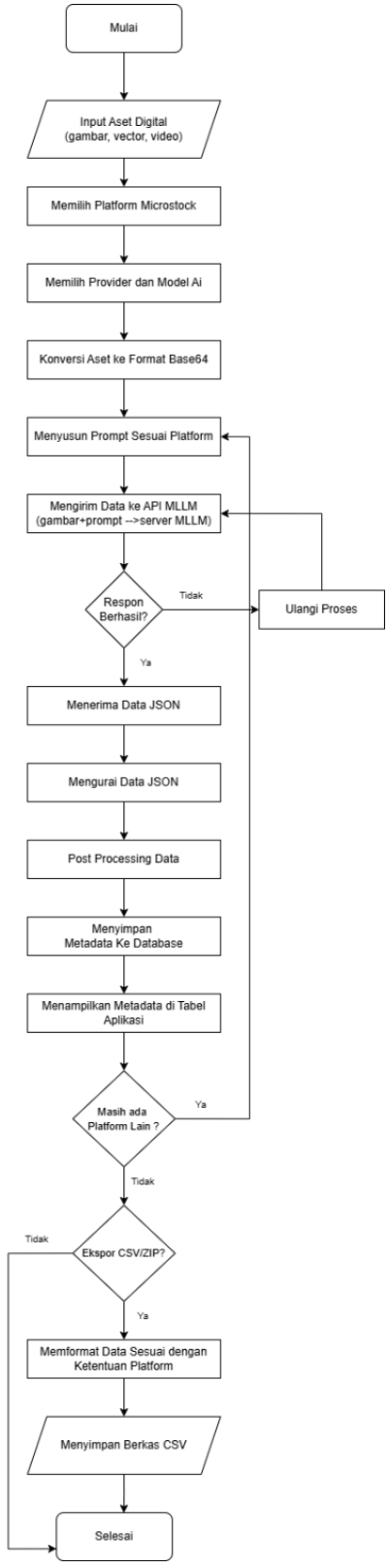
f. *RenameEngine* (*core/rename_engine.py*): Menjalankan logika penamaan ulang berkas fisik secara massal.

3. Relasi Antar Kelas

Hubungan antarkelas pada Diagram ini terbagi menjadi dua jenis. Pertama, relasi *Inheritance* (garis berpanah segitiga kosong) yang menunjukkan bahwa *RZAutometadata* mewarisi seluruh *method* dari ketujuh *mixin* dan *CTk*. Kedua, relasi *Dependency* (garis putus-putus berpanah terbuka) yang menunjukkan ketergantungan antarkelas dalam menjalankan tugasnya. Contohnya, *ActionsMixin* sangat bergantung pada *MetadataProcessor* dan *Database* saat proses *generate* berjalan.

3.5.6 *Flowchart* Sistem

Flowchart sistem digunakan untuk menggambarkan alur proses teknis yang terjadi di dalam aplikasi RZ Autometadata mulai dari aset digital dimasukkan hingga berkas CSV dihasilkan. Berikut adalah *Flowchart* sistem yang dirancang:

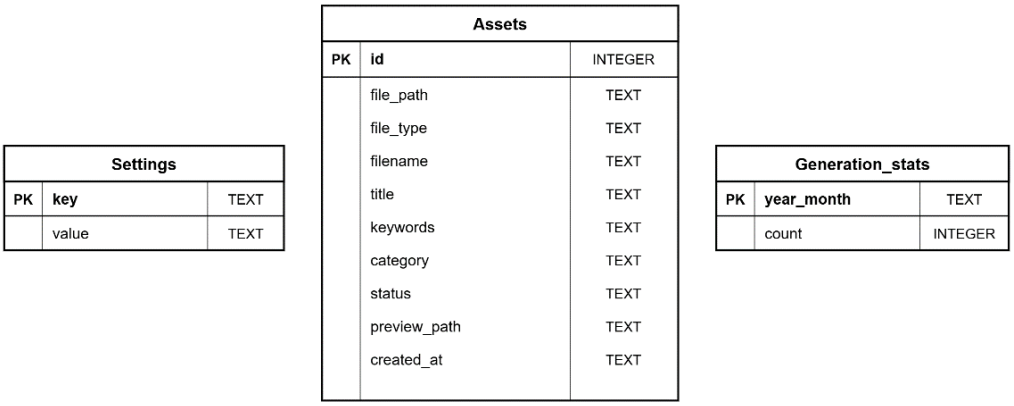


Gambar 3. 15 *Flowchart* Sistem

Berdasarkan Diagram tersebut, proses dimulai ketika pengguna memasukkan aset digital berupa gambar, vector, atau video ke dalam aplikasi. Selanjutnya pengguna memilih *platform* microstock yang dituju serta menentukan penyedia dan model AI yang akan digunakan. Sistem kemudian mengonversi aset ke dalam format Base64 dan menyusun *prompt* yang sesuai dengan ketentuan *platform* yang dipilih. Data tersebut dikirimkan ke server MLLM melalui API. Apabila server memberikan respons yang tidak berhasil, sistem akan mengulangi proses pengiriman. Apabila respons berhasil diterima, sistem mengurai data JSON yang dikembalikan oleh MLLM, lalu melakukan tahap post-processing berupa penghapusan kata kunci ganda dan penyesuaian format metadata. Hasil metadata disimpan ke dalam database SQLite dan ditampilkan di tabel aplikasi. Jika pengguna memilih lebih dari satu *platform*, sistem akan mengulangi proses penyusunan *prompt* dan pengiriman ke API untuk *platform* berikutnya hingga seluruh *platform* selesai diproses. Pada tahap akhir, pengguna dapat memilih untuk mengekspor metadata ke dalam berkas CSV yang formatnya disesuaikan dengan ketentuan masing-masing *platform*.

3.5.7 Struktur Pangkalan Data Lokal

Struktur Pangkalan Data (Database Schema) digunakan untuk menggambarkan arsitektur penyimpanan data yang digunakan oleh aplikasi RZ Autometadata. Pangkalan data yang digunakan adalah SQLite yang tersimpan secara lokal di perangkat pengguna. Berikut adalah gambar struktur tabel dari sistem yang dirancang:



Gambar 3. 16 Struktur Pangkalan Data Lokal

Tabel pertama adalah *assets* yang berfungsi menyimpan riwayat data aset digital beserta hasil metadata yang telah dibuat. Tabel ini memiliki sepuluh atribut, yaitu *id* sebagai kunci utama (*Primary Key*) yang terisi secara otomatis, *file_path* untuk menyimpan lokasi berkas aset, *file_type* untuk mencatat jenis berkas (gambar, vektor, atau video), *preview_path* untuk menyimpan lokasi gambar pratinjau, *filename* untuk mencatat nama berkas, *title* untuk menyimpan judul metadata, *keywords* untuk menyimpan kata kunci, *category* untuk menyimpan kategori, *status* untuk menandai apakah aset sudah diproses atau belum, serta *created_at* untuk mencatat waktu aset ditambahkan.

Tabel kedua adalah *settings* yang berfungsi menyimpan seluruh konfigurasi aplikasi menggunakan pendekatan pasangan kunci dan nilai (*key-value*). Tabel ini digunakan untuk menyimpan *API Key* dari setiap penyedia layanan *MLLM*, batas minimal dan maksimal jumlah karakter judul, batas jumlah kata kunci, serta pengaturan lain yang dapat disesuaikan oleh pengguna secara dinamis.

Tabel ketiga adalah *generation_stats* yang berfungsi mencatat jumlah metadata yang berhasil dibuat setiap bulannya. Tabel ini memiliki dua atribut, yaitu *year_month* sebagai kunci utama yang berisi periode dalam format

tahun dan bulan, serta *count* yang berisi jumlah kumulatif metadata yang telah dibuat pada periode tersebut.

Ketiga tabel tersebut dirancang secara independen (*loosely coupled*) dan secara sengaja tidak memiliki relasi langsung (seperti *Foreign Key*) satu sama lain. Keputusan arsitektural ini diambil karena masing-masing entitas melayani domain fungsional yang sepenuhnya terisolasi: tabel *assets* menangani riwayat operasional file, tabel *settings* beroperasi layaknya berkas konfigurasi mandiri (*key-value store*), dan tabel *generation_stats* berfungsi murni sebagai pencatat analitik (*counter*). Ketiadaan relasi struktural antar tabel ini merupakan bentuk optimasi pada aplikasi berbasis desktop *utility*. Dengan tidak adanya ketergantungan relasional, *SQLite* mampu meminimalkan *overhead* kueri, menghindari risiko penguncian basis data (*database lock*) saat proses *multi-threading* berjalan secara bersamaan, serta memaksimalkan kecepatan operasi baca dan tulis.

Spesifikasi basis data menjabarkan rincian tipe data dan struktur dari setiap tabel yang telah dimodelkan di atas:

1. Spesifikasi Tabel Basis Data

Tabel 3. 4 Tabel *Assets*

No	Akronim (Nama Field)	Type	Keterangan
1	id	INT	Primary Key, Auto Increment
2	file_path	TEXT	Lokasi direktori berkas asli
3	file_type	TEXT	Jenis berkas (image, vector, video)
4	preview_path	TEXT	Lokasi direktori gambar pratinjau
5	filename	TEXT	Nama berkas beserta ekstensinya
6	title	TEXT	Judul metadata hasil generate
7	keywords	TEXT	Kata kunci metadata hasil generate
8	category	TEXT	Kategori / ID Kategori (jika ada)
9	status	TEXT	Status pemrosesan (pending, success)
10	created_at	TEXT	Waktu data ditambahkan (Timestamp)

Tabel 3. 5 *Settings*

No	Akronim (Nama Field)	Type	Keterangan
1	key	TEXT	Primary Key, Nama kunci pengaturan
2	value	TEXT	Nilai dari pengaturan terkait

Tabel 3. 6 *Generation Stats*

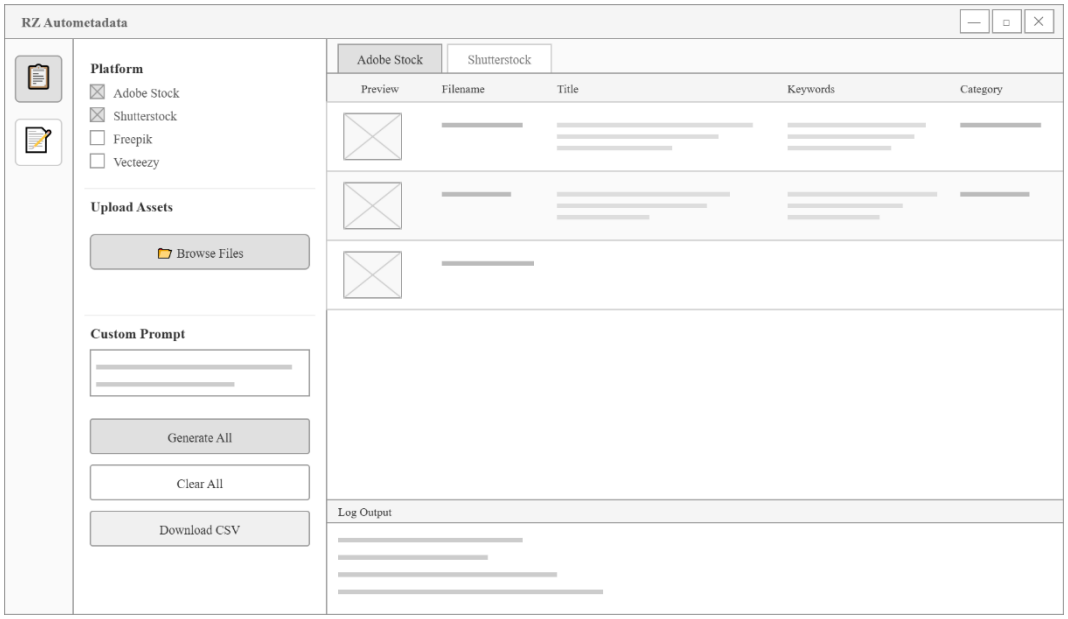
No	Akronim (Nama Field)	Type	Keterangan
1	year_month	TEXT	Primary Key, Periode (Contoh: 2026-06)
2	value	INT	Jumlah kumulatif metadata yang dibuat

Penggunaan tipe data *TEXT* secara dominan pada spesifikasi tabel di atas bukan merupakan keputusan desain tanpa dasar yang jelas, melainkan didasarkan pada karakteristik bawaan arsitektur *SQLite* itu sendiri. Menurut dokumentasi resmi *SQLite* (Hipp, 2024), *SQLite* menerapkan sistem *Dynamic Type Affinity* yang hanya mengenal lima kelas penyimpanan utama (*Storage Class*), yaitu *NULL*, *INTEGER*, *REAL*, *TEXT*, dan *BLOB*. Berbeda dengan *MySQL* yang membedakan tipe teks ke dalam beberapa varian ketat seperti *CHAR*, *VARCHAR*, dan *LONGTEXT* dengan batasan panjang karakter, *SQLite* secara *native* menyatukan seluruh data berbasis karakter ke dalam satu kelas penyimpanan *TEXT* dan mengalokasikan memori secara dinamis sesuai jumlah karakter yang dimasukkan.

3.5.8 Rancangan Antarmuka

Rancangan antarmuka menggambarkan tata letak visual yang akan diterapkan pada aplikasi RZ Autometadata. Aplikasi ini memiliki dua halaman utama serta satu jendela pop-up pengaturan.

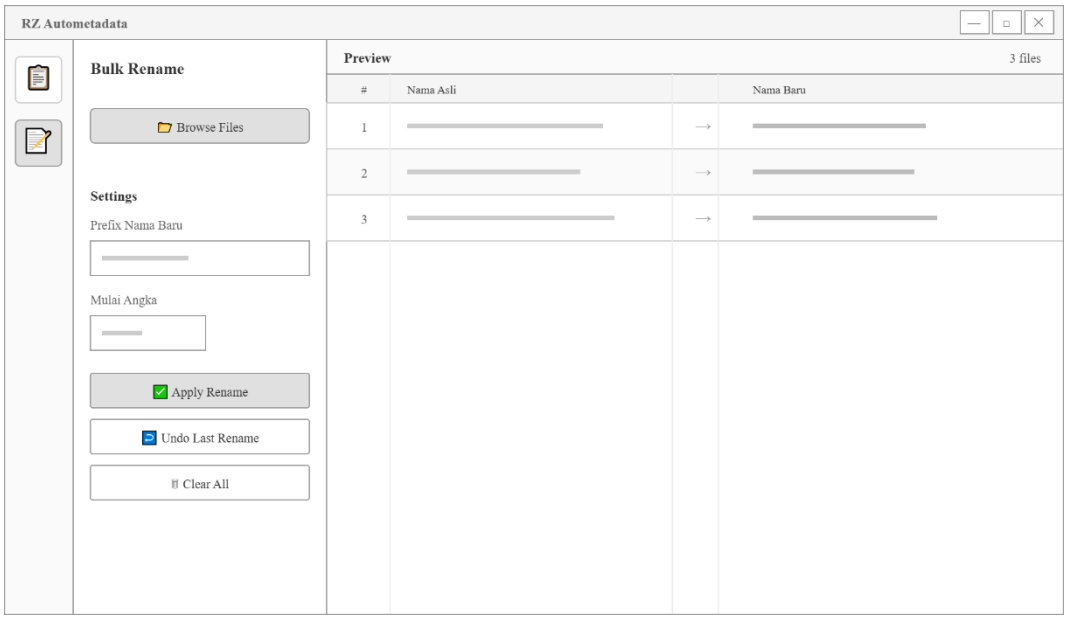
1. Halaman Metadata Generator



Gambar 3. 17 Halaman Metadata

Halaman ini merupakan halaman utama yang ditampilkan saat aplikasi pertama kali dibuka. Pada sisi paling kiri terdapat bilah navigasi yang berisi ikon untuk berpindah antar halaman. Di sebelahnya terdapat panel samping (sidebar) yang memuat beberapa elemen, yaitu kotak centang untuk memilih *platform* microstock yang dituju, area unggah untuk memasukkan aset digital baik dengan cara menyeret berkas maupun memilih melalui dialog, kolom custom *prompt* untuk menambahkan kata kunci tambahan, dropdown untuk memilih penyedia dan model AI, tombol Generate Metadata untuk memulai proses pembuatan metadata, tombol Export CSV untuk mengunduh hasil dalam format CSV, serta tombol Settings untuk membuka jendela pengaturan. Pada bagian utama di sisi kanan terdapat tabel yang menampilkan daftar aset beserta hasil metadata yang meliputi pratinjau gambar, nama berkas, judul, kata kunci, dan status pemrosesan. Di bagian bawah tabel terdapat panel log yang menampilkan informasi proses secara langsung.

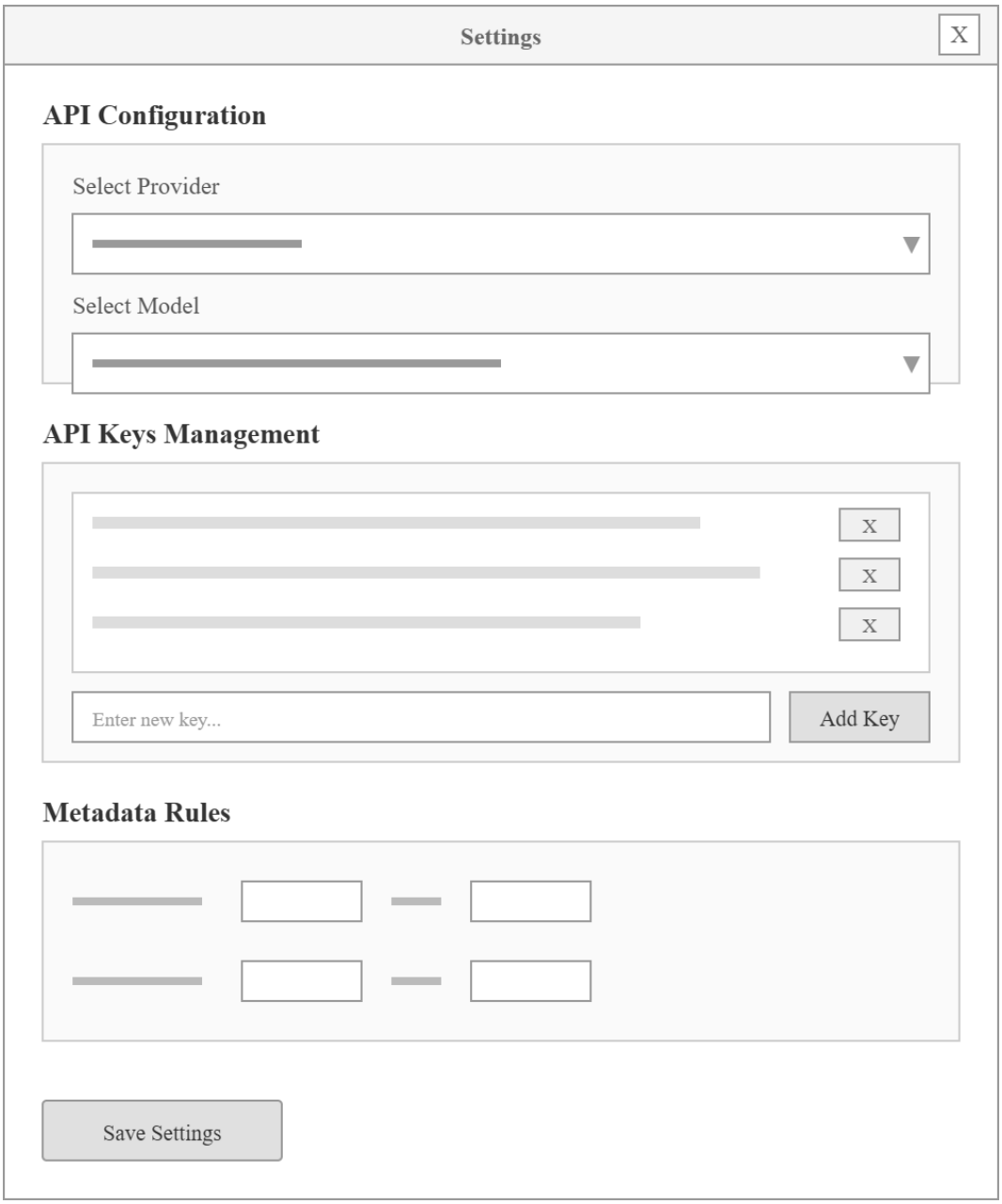
2. Halaman *Bulk Rename*



Gambar 3. 18 *Bulk Rename*

Halaman ini berfungsi untuk mengganti nama berkas secara massal. Pada bagian atas terdapat kolom untuk mengatur awalan (prefix) dan nomor awal penomoran. Di sebelahnya terdapat tombol untuk memilih folder yang berisi berkas serta tombol untuk menjalankan proses penggantian nama. Tabel di bawahnya menampilkan daftar nama berkas lama dan nama berkas baru yang akan diterapkan, sehingga pengguna dapat memeriksa hasil penggantian nama sebelum mengeksekusi proses

3. *Pop-up* Pengaturan

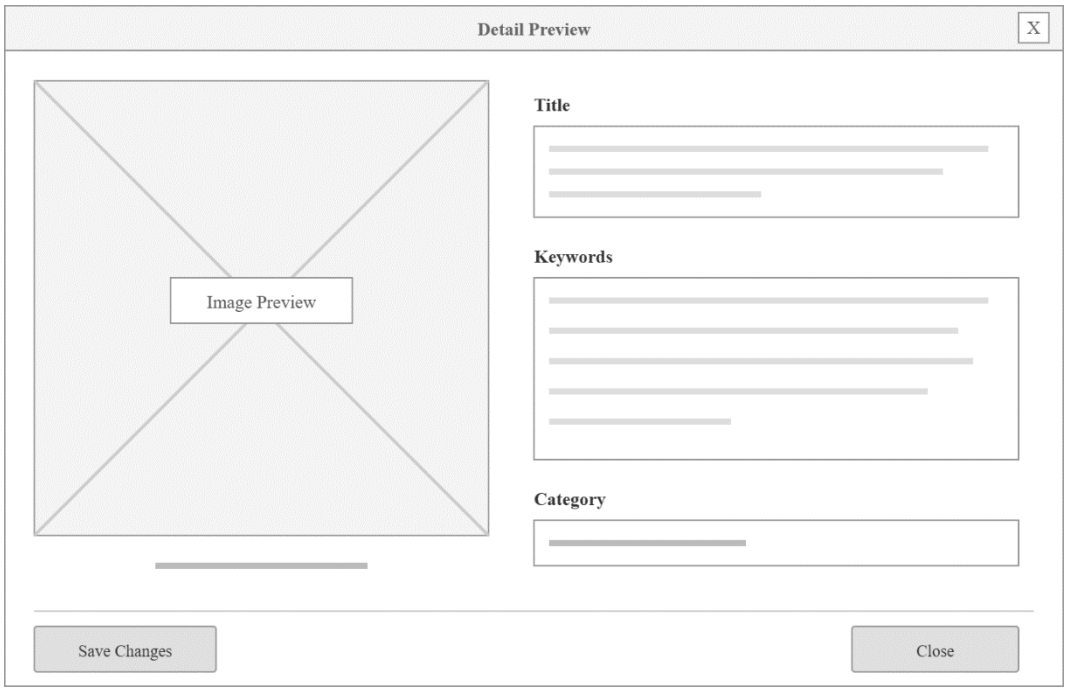


Gambar 3. 19 *Pop Up* Pengaturan

Jendela pop-up ini muncul ketika pengguna menekan tombol Settings di panel samping. Pengaturan dibagi menjadi tiga bagian. Bagian pertama berisi dropdown untuk memilih penyedia AI dan model yang akan digunakan. Bagian kedua berisi pengelolaan API Key yang memungkinkan pengguna menambahkan lebih dari satu kunci API untuk setiap penyedia. Jumlah kunci

API yang tersimpan menentukan jumlah worker yang bekerja secara paralel saat proses pembuatan metadata. Bagian ketiga berisi pengaturan rentang metadata yang mencakup jumlah karakter minimal dan maksimal untuk judul serta jumlah minimal dan maksimal untuk kata kunci. Pengaturan rentang ini disesuaikan secara otomatis berdasarkan *platform* yang sedang aktif.

4. Rancangan Antarmuka Halaman *Detail Preview* (Menyunting Metadata)



Gambar 3. 20 Rancangan Antarmuka Halaman *Detail Preview*

Secara tata letak (*layout*), halaman ini dibagi menjadi dua bagian utama:

a. **Sisi Kiri (Pratinjau Visual):** Menampilkan *thumbnail* atau gambar pratinjau (*preview*) dari aset digital yang dipilih, beserta nama *file* di bawahnya. Hal ini bertujuan agar pengguna dapat dengan mudah mengkorelasikan kesesuaian metadata dengan visual aset yang sebenarnya.

b. **Sisi Kanan (Formulir Penyuntingan):** Menyediakan kotak isian (*input text*) yang terdiri dari atribut *Title* (Judul), *Keywords* (Kata Kunci), dan Kategori. Pada masing-masing kotak isian, sistem dilengkapi dengan fitur penghitung karakter dan jumlah kata (*character & word counter*) yang akan memberikan indikator warna

secara *real-time* (hijau jika memenuhi standar batas minimum, dan merah jika melanggar ketentuan batas *platform* microstock).

c. **Tombol Aksi:** Pada bagian bawah jendela, terdapat tombol "Simpan" untuk menerapkan perubahan metadata ke dalam sistem (memperbarui database sementara), dan tombol "Batal" untuk menutup jendela tanpa menyimpan perubahan.

Rancangan antarmuka ini memastikan bahwa meskipun proses *generate* dilakukan oleh *Artificial Intelligence (AI)*, Kontributor tetap memiliki kontrol penuh (kendali manusia / *human-in-the-loop*) untuk melakukan kurasi akhir dan modifikasi metadata sebelum berkas diekspor menjadi format *CSV*.