PENGEMBANGAN RESTFUL API UNTUK APLIKASI KLASIFIKASI JENIS TANAH BERBASIS MOBILE PADA GOOGLE CLOUD

(*Restful Api Development For Mobile-Based Soil Type Classification Applications On Google Cloud***)**

Muh. Firdaus*[1], Royana Afwani[1]
[1]Dept Informatics Engineering, Mataram University
Jl. Majapahit 62, Mataram, Lombok NTB, INDONESIA
Email: muhfirdau0805@mhs.unram.ac.id, royana@unram.ac.id

Abstract

Soil plays an important role and is the foundation of life for all living things. Determining soil type is essential in areas such as agriculture, land management, and environmental studies. Various methods, including soil content or color classification methods, as well as conducting laboratory tests, can be used to identify soil types. However, this often requires the involvement of experts and is time-consuming and costly. To overcome these challenges, there is a need for an application that can perform soil type classification in order to minimize the use of resources and time. This research aims to create an Application Programming Interface (API) with REST architecture in order to connect existing machine learning models with mobile-based applications. This application is called Terralysis, which will automatically classify soil types into four types: alluvial soil, red soil, black soil, and clay soil. By utilizing the advanced capabilities of the mobile device's camera, users can take pictures of the soil, upload them to the app, and receive the results. This research resulted in a REST API built using Javascript and Python programming languages, with a total of 11 API endpoints. It is divided into two parts, namely the API for user management and the API for classification models. This API has been successfully run using Google Cloud services and has successfully passed functionality testing for all endpoints.

Keywords: *RESTful API, Google Cloud, Klasifikasi Jenis Tanah*

**Penulis Korespondensi*

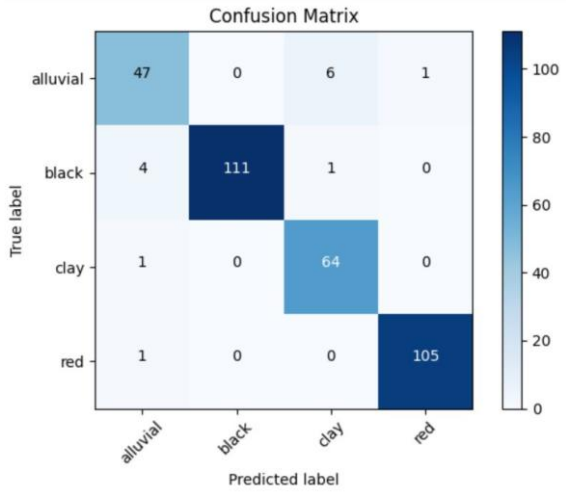
1. PENDAHULUAN

Tanah memiliki peranan penting dan menjadi soko guru kehidupan. Ini karena secara langsung ataupun tidak langsung, seluruh makhluk hidup sangat tergantung pada eksistensi dan manfaat tanah [1]. Tanah memiliki beragam jenis yang dapat dibedakan berdasarkan berbagai faktor seperti tekstur, struktur dan warna [1]. Beberapa jenis tanah yang umum meliputi tanah aluvial, tanah merah, tanah liat (*clay*), dan tanah hitam. Tanah aluvial terbentuk dari material yang diendapkan oleh air sungai atau air banjir, sementara tanah merah kaya akan kandungan zat besi sehingga memberikan warna yang khas. Tanah liat memiliki partikel-partikel halus yang dapat menahan air dengan baik, sementara tanah hitam mengandung humus dalam kuantitas tinggi [1].

Menentukan jenis tanah menjadi penting dalam berbagai bidang seperti pertanian, pertanahan dan lingkungan. Dalam menentukan jenis tanah, terdapat banyak cara yang dapat dilakukan seperti menggunakan metode geolistrik, metode klasifikasi berdasarkan kandungan tanah atau menggunakan metode untuk mengklasifikasi jenis tanah berdasarkan warna tanah yang menjadi salah satu metode yang paling umum digunakan [2]. Untuk proses identifikasi tanah, warna tanah sering dijadikan sebagai salah satu parameter [2]. Dengan warna, karakteristik dan jenis dari tanah dapat diketahui, seperti misalnya tanah dengan warna hitam (gelap) biasanya menunjukkan kandungan bahan organik yang ada pada tanah tersebut cukup tinggi [1]. Selain warna, parameter lain yang dapat digunakan adalah struktur dan tekstur dari tanah itu sendiri [1]. Penentuan jenis tanah yang lebih akurat biasanya memerlukan analisis lebih lanjut, seperti tes laboratorium untuk memeriksa struktur dan tekstur tanah, analisis kimia untuk mengidentifikasi kandungan nutrisi dan mineral, serta penggunaan alat-alat seperti penetrometer untuk mengukur kepadatan tanah. Kombinasi dari berbagai informasi ini akan membantu mengklasifikasikan tanah dengan lebih tepat dan dapat diandalkan. Namun, proses penentuan jenis tanah dengan cara seperti ini membutuhkan waktu, biaya dan keterlibatan ahli di lapangan.

Di era modern ini, teknologi dapat digunakan untuk membantu mengenali jenis tanah. Proses

penentuan jenis tanah dapat dibantu dengan memanfaatkan teknologi komputer. Dalam hal ini, dapat menggunakan model *machine learning* dengan memanfaatkan teknik *computer vision* yang diharapkan mampu menganalisis dan mengenali pola serta fitur kompleks dalam data gambar atau citra tanah sehingga mampu secara otomatis menentukan jenis tanah. Dengan menggunakan teknik *computer vision*, diharapkan dapat melakukan klasifikasi dengan tingkat akurasi yang lebih tinggi dari pada hanya mengandalkan penilaian visual manusia dan mengurangi waktu dan biaya yang diperlukan. Oleh karena itu, maka diperlukan sebuah *model machine learning* yang digunakan untuk melakukan klasifikasi tanah.



	Precision	Recall	f1-Score
Macro Avg	0.94	0.95	0.95
Weighted Avg	0.96	0.96	0.96

Gambar 1. Akurasi Model

Dalam penelitian ini akan menggunakan sebuah model yang sudah dilatih menggunakan *transfer learning* dengan inceptionV3. Berdasarkan **Gambar 1**, model ini dapat melakukan klasifikasi terhadap 4 jenis tanah yaitu *alluvial, red, black, and clay soil*. Tingkat akurasi model dengan nilai *precision*, *recall*, dan *F1-Score* di atas 0.95 serta *weighted average* sebesar 0.96. Hal ini menandakan bahwa model mampu dengan tepat mengidentifikasi sebagian besar prediksi positif yang benar dan mengenali sebagian besar *instance* positif secara keseluruhan, memberikan klasifikasi yang konsisten dan andal terhadap data yang diberikan. Model ini akan diimplementasikan kedalam aplikasi berbasis *mobile* dengan nama Terralysis yang mengandung arti menganalisa tanah. *Terra* yang berarti tanah dan *lysis* yang diambil dari kata *analysis.*

Gambar 2. Logo dari Terralyasis

Melalui aplikasi ini, pengguna dapat mengambil foto tanah dan mengunggahnya untuk mendapatkan hasil klasifikasi jenis tanah. Dalam mengintegrasikan antara aplikasi *mobile* dan model *machine learning* maka diperlukan adanya RESTful API *(Application Programming Interface*).

Penilitian ini merupakan bagian dari penelitian yang terintegrasi yang bertujuan dalam perancangan dan pengembangan RESTful API. Penggunaan RESTful API memungkinkan adanya integrasi yang lebih mudah dan dapat diakses oleh berbagai pengguna. Selain itu, dapat memisahkan model *machine learning* yang berjalan di *server* dengan aplikasi *mobile* sehingga akan memastikan bahwa pembaruan dan peningkatan performa dari model dapat dilakukan secara terpusat pada server tanpa mengganggu penggunaan aplikasi di perangkat *mobile.*

Dengan adanya RESTful API, aplikasi *mobile* dapat secara efisien berkomunikasi dengan model *machine learning* di *server* tanpa harus memiliki pengetahuan teknis yang mendalam tentang algoritma dan logika klasifikasi yang ada di baliknya. Hal ini memungkinkan pengembangan aplikasi yang lebih ringan dan responsif, sementara pemrosesan klasifikasi jenis tanah beratnya dilakukan oleh server yang lebih kuat dan canggih. Selain itu, melalui aplikasi *mobile* yang digunakan untuk klasifikasi tanah berdasarkan gambar, proses ini dapat menjadi lebih efisien, lebih cepat, dan lebih mudah diakses oleh pengguna yang tidak memiliki akses ke laboratorium atau ahli tanah tanpa memakan waktu dan biaya lebih banyak.

2. TINJAUAN PUSTAKA

2.1 Penelitian Terkait

Terdapat beberapa penelitian yang telah dilakukan dalam pendeteksian jenis tanah dengan menggunakan teknologi komputer. Salah satunya adalah penelitian yang berjudul "Penerapan IoT pada Pendeteksi Jenis Tanah Berbasis Web" pada tahun 2022. Penelitian ini menghasilkan data nilai RGB dan kelembaban pada sampel tanah yang digunakan [2]. Penelitian lain yang relevan dengan jenis tanah adalah "Aplikasi Pengenalan Jenis Tanah untuk Lahan Pertanian dengan Menggunakan Metode Euclidean Distance". Penelitian ini menghasilkan aplikasi berbasis desktop yang dapat menganalisis karakteristik tanah dan memberikan rekomendasi penggunaan lahan pertanian [3]. Selain

itu, ada juga penelitian lain yang menggunakan platform mobile dengan judul "Aplikasi Mobile Scotect: Aplikasi Deteksi Warna Tanah dengan Teknologi Citra Digital pada Android" [4]. Penelitian ini lebih fokus pada deteksi warna tanah berdasarkan citra digital. Kemudian, terdapat penelitian lain yang menghasilkan aplikasi berbasis desktop yang disebut "Aplikasi Pengolah Citra untuk Menentukan Jenis Tanah pada Lahan Pertanian" [5]. Penelitian-penelitian sebelumnya menunjukkan bahwa pengembangan aplikasi pendeteksi jenis tanah berbasis *mobile* belum pernah dilakukan, kecuali untuk mendeteksi warna tanah.

Sementara penelitian yang terkait dengan pembuatan RESTful API salah satunya adalah penelitian dengan judul "Pemanfaatan Restful Web Services Pada Perangkat Lunak Penyewaan Lapangan Badminton". Penelitian ini membahas tentang pembuatan RESTful API untuk penyewaan lapangan badminton dengan menggunakan JSON sebagai format penukaran data yang dinilai lebih ringan, mudah dibaca oleh manusia dan mudah diterjemahkan serta dibuat oleh komputer [13]. Penelitian lain yang relevan dengan pembuatan API adalah penelitian dengan judul "Rancang Bangun Web Service API dan Dokumentasi Rest API Web Portal Unit Kegiatan Mahasiswa Di Politeknik Negeri Lampung". Penelitian ini menggunakan arsitektur REST dalam pembuatan *web service* API yang akan digunakan pada *web* portal Unit Kegiatan Mahasiswa (UKM). Penelitian ini juga menyebutkan bahwa penggunaan teknologi RESTful API dapat mempermudah proses pembuatan *web frontend* serta memungkinkan adanya integrasi antar sistem yang berbeda [14]. Penelitian lain yang juga relevan dengan RESTful API adalah penelitian yang berjudul "Perancangan Dan Implementasi Restful API Pada Sistem Informasi Manajemen Dosen Universitas Udayana". Pada penelitian ini, dibahas mengenai pembuatan RESTful API dengan *black box testing* yang akan diimplementasikan pada Sistem Informasi Manajemen Dosen Universitas Udayana. Selain itu, penelitian ini juga membahas tentang keunggulan penggunaan API dan penerapan *microservices* yang dapat memisahkan komponen yang ada pada perangkat lunak berdasarkan kegunaan dan fungsinya, sehingga apabila terjadi *error* pada satu komponen, maka tidak mempengaruhi keseluruhan perangkat lunak [15].

Berdasarkan penelitian-penelitian sebelumnya baik yang terkait dengan identifikasi jenis tanah menggunakan teknologi komputer maupun pembuatan API dengan arsitektur REST, maka pada penelitian ini, penulis bermaksud untuk membuat RESTful API yang akan diimplementasikan pada aplikasi klasifikasi jenis tanah berbasis *mobile.* Penelitian ini berbeda dengan penelitian-penelitian pada penjelasan sebelumnya, dikarenakan objek penelitian ini adalah klasifikasi jenis tanah yang akan diimplementasikan pada aplikasi *mobile*. RESTful API yang dibuat akan terintegrasi dengan model *machine learning* sehingga dapat digunakan atau dipanggil pada *platform* lain yang mana dalam penelitian ini adalah aplikasi berbasis *mobile*. Selain itu, proses komputasi dan *logic* dari aplikasi akan ditangani oleh *server* melalui RESTful API sehingga dapat dipisahkan dengan aplikasi *mobile*. Setelah proses pembuatan, akan dilakukan pengujian fungsionalitas dari RESTful API dengan menggunakan metode *black box testing* untuk memastikan fitur-fitur yang dirancang dapat berjalan sesuai dengan yang diharapkan.

2.2 Teori Penunjang

2.2.1 API

API atau *Application Programming Interface* adalah suatu antarmuka pemrograman aplikasi yang memungkinkan aplikasi untuk saling berinteraksi dan berbagi informasi. API berperan sebagai penghubung antara berbagai sistem dan aplikasi yang memiliki kebutuhan yang sama namun menggunakan mekanisme yang berbeda. API memungkinkan standarisasi dalam pengiriman data, sehingga pengembang dapat menyediakan berbagai jenis data yang dapat dipahami oleh pengembang lain dan mereka dapat menggunakan sistem mereka sendiri [5].

2.2.2 REST

REST atau *Representational State Transfer* adalah filosofi desain yang mendorong kita untuk menggunakan protokol dan fitur *web* yang sudah ada untuk mengaitkan permintaan sumber daya dengan berbagai representasi dan manipulasi data di Internet [7].

2.2.3 HTTP

Hypertext Transfer Protocol (HTTP) adalah protokol yang mengatur prosedur komunikasi antara *client* dan *server* dengan konsep *request-response*. Sebagai protokol, HTTP menetapkan format dan cara komunikasi serta tindakan dan reaksi antara *web server* dan *browser* [8].

2.2.4 JSON

Javascript Object Notation (JSON) adalah format data encoding umum yang populer. Database dan web service banyak menggunakan JSON. Struktur dokumen JSON dapat dibatasi secara opsional oleh skema yang terdiri dari dua komponen: map (pemetaan struktur

nilai berdasarkan klasifikasi jenisnya) dan list (pengelompokan nilai berdasarkan klasifikasi jenisnya) [9].

2.2.5 JWT

JWT atau JSON *Web Token* merupakan representasi dari format data yang penggunaannya ditujukan untuk ditempatkan pada *header* Otorisasi HTTP atau parameter *query* URI. JWT mengkodekan data yang dikirim sebagai objek JSON dengan bantuan *Signature* JSON *Web* (JWS) yang memungkinkan data ditandatangani atau dilindungi secara digital dengan *Message Authentication Code* [10].

2.2.6 Postman

Postman merupakan sebuah *software* yang memuat fungsi lengkap pengembangan sistem dalam mengirimkan dan menerima respon *server*. *Software* ini mendukung pengembangan sistem REST API dengan mengklasifikasi *request* berdasarkan *request method*, URL dan parameter-parameter *request*. Postman dapat digunakan untuk menguji REST API berbasis GUI.

2.2.7 *Black Box Testing*

Black box testing adalah teknik pengujian perangkat lunak yang berfokus pada spesifikasi fungsional tanpa menguji desain atau kode program. Ini hanya membutuhkan batas bawah dan batas atas pada data yang diharapkan. *Black box testing* melibatkan pengujian sistem tanpa pengetahuan sebelumnya tentang bagaimana sistem itu bekerja. Penguji memberikan masukan dan mengamati keluaran sistem. Dengan demikian, memungkinkan untuk mengetahui bagaimana sistem menangani tindakan pengguna yang diharapkan dan tidak diharapkan, waktu *response*, masalah kegunaan, dan masalah keandalan [15].

2.2.8 *Cloud Computing*

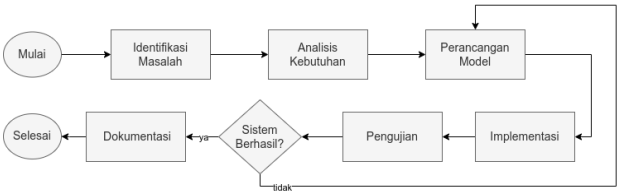
Cloud Computing juga diartikan sebuah model *client-server*, di mana *resources* seperti *server, storage network* dan *software* dapat dipandang sebagai layanan yang dapat diakses oleh pengguna secara *remote* dan setiap saat [11]. *Cloud computing* dapat dinyatakan perangkat keras dan perangkat lunak sumber daya dan jasa yang ditawarkan melalui internet [12].

2.2.9 Google Cloud *Platform*

Google Cloud *Platform* adalah suatu layanan komputasi awan publik dari Google yang terdiri dari berbagai macam layanan. *Platform* ini menyediakan berbagai layanan seperti komputasi, penyimpanan, dan pengembangan aplikasi yang berjalan di infrastruktur *hardware* milik Google. Layanan Google Cloud Platform dapat diakses oleh pengembang perangkat lunak, *cloud administrator*, dan profesional IT lainnya melalui internet publik atau melalui koneksi jaringan yang didedikasikan [11].

3. METODE PENELITIAN

3.1 Alur Pengembangan Sistem



Gambar 3. Alur Pengembangan Sistem

Gambar 3 merupakan alur pengembangan RESTful API untuk aplikasi klasifikasi jenis tanah berbasis *mobile* pada Google Cloud.

Berikut adalah penjelasan tiap proses pada alur pengembangan tersebut.

3.1.1 Identifikasi Masalah

Pada penelitian ini, masalah yang terjadi adalah bagaimana menghubungkan model *machine learning* dengan aplikasi berbasis *mobile* melalui pembuatan RESTfull API yang diimplementasikan ke Google Cloud.

3.1.2 Analisis Kebutuhan

Dalam pengembangan RESTful API yang akan diimplementasikan ke Google Cloud maka dibutuhkan teknologi dan *service* yang akan digunakan seperti pada tabel berikut.

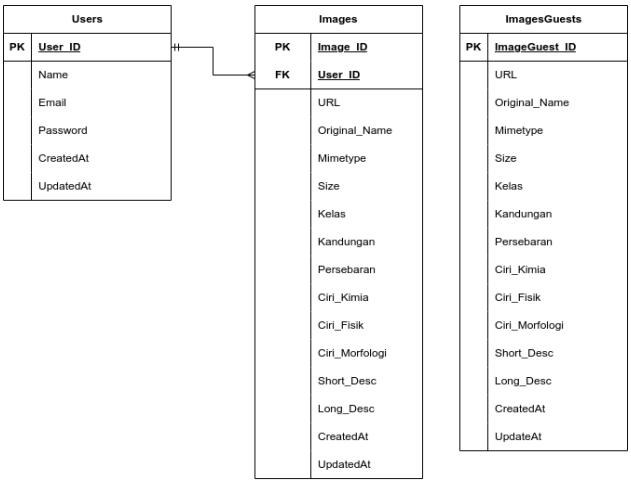
TABEL 1. *TECHNOLOGY/SERVICES*

Technology/Services Fungsi	Fungsi
Framework Express JS - Javascript	*Pembuatan RESTful API user authentication*
Framework FAST-API - Python	*Pembuatan RESTful API model machine learning*
MySQL - Cloud SQL	*Relational databases services*
Artifact Registry	*Container images*
Cloud Storage	*Image files storage*
Cloud Run	*Deploy/run container image*

3.1.3 Perancangan Model

Perancangan model dalam pengembangan RESTful API ini meliputi perancangan basis data, pembuatan *use case diagram*, kandidat API, *activity diagram* dan *project architecture*.

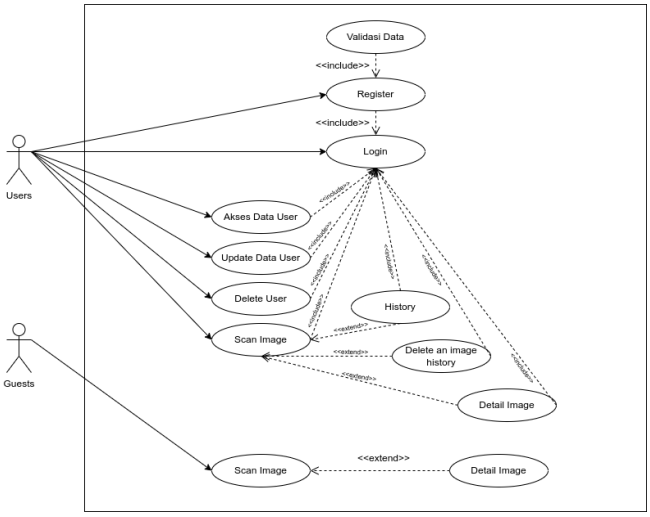
3.1.3.1 Basis Data



Gambar 4. Perancangan Basis Data

Ada tiga tabel utama: *Users*, *Images*, dan *ImagesGuests*. Tabel *Users* digunakan untuk menyimpan data pengguna, tabel Images untuk menyimpan data gambar tanah yang akan diklasifikasikan. Hubungan antara Users dan Images adalah *one to many*. Terdapat juga tabel *ImagesGuests* yang memiliki atribut yang hampir sama dengan tabel *Images*. Tabel ini digunakan untuk menyimpan data gambar pengguna yang belum terdaftar. Tabel *ImagesGuests* dibuat untuk menangani pengguna yang ingin melakukan klasifikasi tanah tanpa registrasi.

3.1.3.2 *Use Case Diagram*



Gambar 5. *Use Case Diagram*

Berdasarkan **Gambar 5**, terdapat dua aktor yaitu “Users” dan “Guest”. "Users" adalah pengguna yang melakukan registrasi dan login sebelum menggunakan fitur lainnya, seperti mengakses, memperbarui, dan menghapus data pengguna, serta melakukan pemindaian gambar. Gambar yang dipindai akan ditampilkan dalam halaman riwayat. "Guests" hanya dapat melakukan pemindaian gambar dan melihat detail gambar hasil pemindaian tanpa perlu login atau registrasi terlebih dahulu.

3.1.3.3 Kandidat API

TABEL 2. KANDIDAT API

Proses	*Method*	*Path*	*Request*	*Response*
Melakukan registrasi	POST	/register	*- name as string* *- email as string; must be unique* *- password as string*	*- status error* *- message* *- data user*
Masuk sebagai *users*	POST	/login	*- email as string* *- password as string*	*- status error* *- message* *- data user* - token
Mengambil data *user* berdasarkan *user id*	GET	/users/{userId}	*- header authorization; bearer token*	*- status error* *- message* *- data user*
Memperbarui data *user* berdasarkan *user id*	PATCH	/users/{userId}	*- name as string (optional)* *- email as string; must be unique (optional)* *- password as string (optional)* *- header authorization; bearer token*	*- status error* *- message* *- data user*
Menghapus *user* berdasarkan *user id*	DELETE	/users/{userId}	*- header authorization; bearer token*	*- status error* *- message* *- data user*
Melakukan *scan image*	POST	/analysis	*- user id as string* *- image as file; key: file* *- header authorization; bearer token*	*- message* *- data image (image id, url, kelas)*
Mendapatkan semua riwayat *image* yang telah di-*scan* (*history)* berdasarkan *user id*	GET	/analysis/{userId}	*- user id as string* *- header authorization; bearer token*	*- status code* *- message* *- array of data image*
Mendapatkan detail sebuah *image* yang telah di-*scan* berdasarkan *user id* dan *image id*	GET	/analysis/{userId}/{imageId}	*- user id as string* *- image id as string* *- header authorization; bearer token*	*- status code* *- message* *- data image*
Menghapus sebuah *image* yang telah di-*scan* berdasarkan *user id* dan *image id*	DELETE	/analysis/{userId}/{imageId}	*- user id as string* *- image id as string* *- header authorization; bearer token*	*- status code* *- message*
Melakukan *scan image*	POST	/analysis-guest	*- image as file;*	*- message*

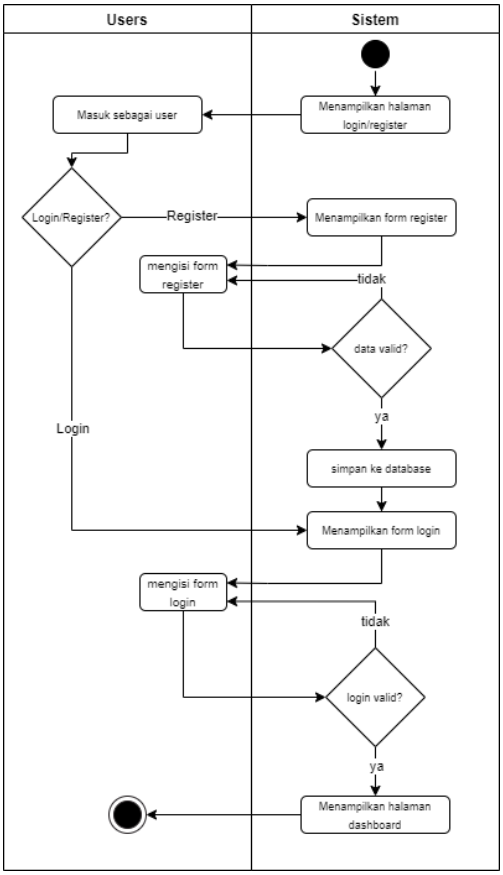
			key: file	- *data image (imagegues ts id, url, kelas)*
Mendapatka n detail *image* yang telah di-*scan* berdasarkan *image guest id*	GET	/analysis-guest/ {image_g uestId}	- *image guest id as string*	- *status code* - *message* - *data image guest*

Berdasarkan **Tabel 2**, diperlukan setidaknya 11 kandidat API yang terbagi menjadi dua bagian berdasarkan entitas yang menggunakannya: entitas pengguna (*users*) dan tamu (*guests*). Selain itu, berdasarkan penggunaannya, API yang dibangun terbagi menjadi dua jenis: API untuk manajemen pengguna dan model klasifikasi. Secara umum, API yang paling penting untuk digunakan dalam melakukan klasifikasi jenis tanah adalah API yang melibatkan proses "melakukan *scan image*" dengan menggunakan *method* POST. API dengan proses ini dibagi menjadi dua *path* berdasarkan entitas yang menggunakannya. *Path* "*/analysis*" akan digunakan oleh entitas *users* yang sudah mendaftar dan masuk ke dalam aplikasi, sementara *path* "*/analysis-guest*" akan digunakan oleh entitas *guests* atau pengguna yang belum mendaftar dan masuk ke dalam aplikasi.

3.1.3.4 *Activity Diagram*

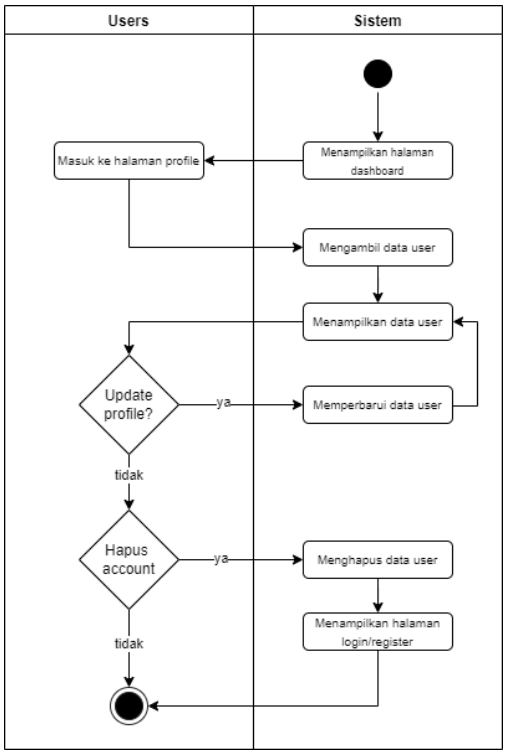
Activity diagram dari pembuatan RESTful API dibagi menjadi empat bagian sesuai dengan kandidat API yang telah ada. Tiga bagian pertama dilakukan oleh entitas *users* dan satu bagian yang lain merupakan aktivitas yang dilakukan oleh entitas *guest*. Entitas *users* dapat melakukan registrasi untuk membuat akun dan melakukan *login* untuk dapat masuk ke dalam sistem. Setelah melakukan pendaftaran dan masuk ke dalam sistem, *users* dapat melakukan pengelolaan terhadap data akun seperti *update,* melihat dan menghapus akun. Selain itu *users* juga dapat melakukan aktivitas untuk mengetahui jenis tanah dengan *scan image* tanah. Hasil dari klasifikasi ini akan tersimpan di halaman *history users*. Sementara untuk entitias *guest* hanya dapat melakukan satu aktivitas saja yaitu melakukan *scan image,* namun memiliki perbedaan dengan *users* yang mana hasil dari *scan image* ini tidak dapat di simpan ke dalam halama *history* karena *guests* tidak melakukan registrasi dan *login* terlebih dahulu sebelum melakukan *scan image*.

a. *Activity* diagram untuk *register* dan *login*



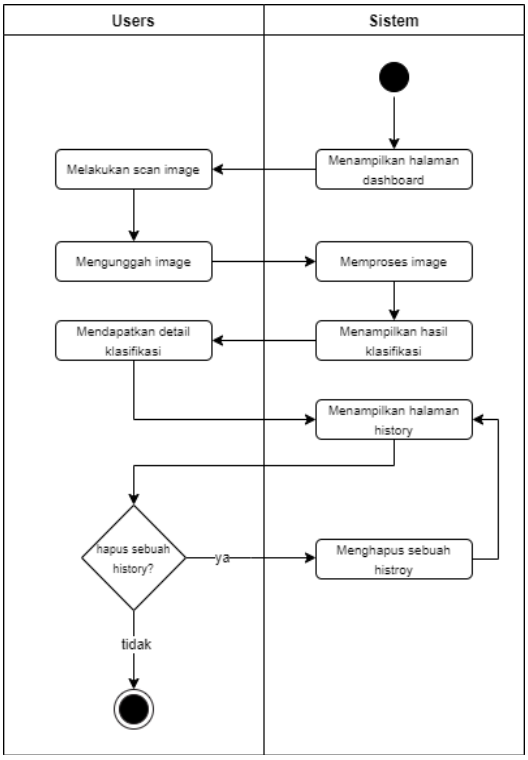
Gambar 6. *Activity Diagram* untuk *Register* dan *Login*

b. *Activity* diagram untuk pengelolaan data *users*



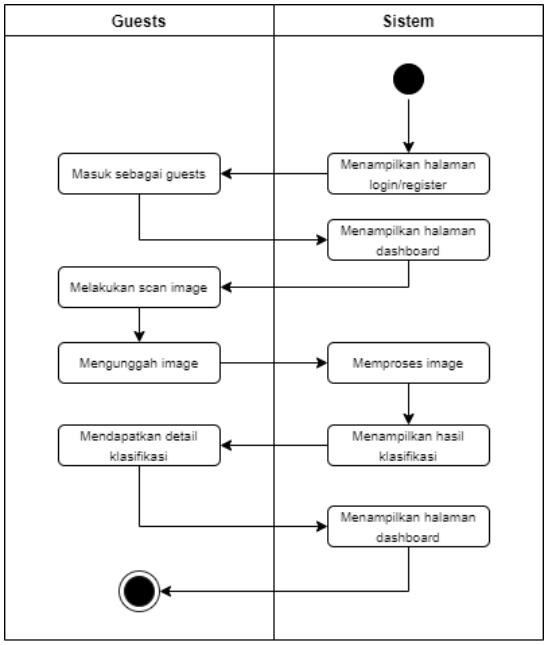
Gambar 7. Activity Diagram untuk Manajemen *Users*

c. *Activity* diagram untuk klasifikasi *image* sebagai *users*



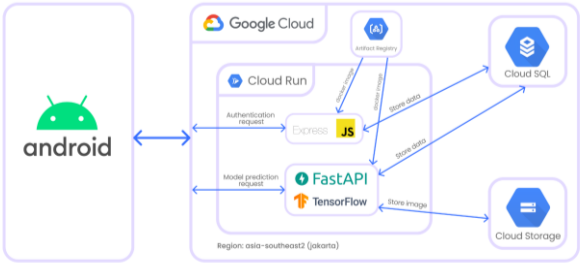
Gambar 8. *Activity Diagram* untuk Klasifikasi *Image* sebagai Users

d. *Activity* diagram untuk klasifikasi *image* sebagai *guest*



Gambar 9. *Activity Diagram* untuk Klasifikasi *Image* sebagai *Guest*

3.1.3.5 *Project Architecture*



Gambar 10. *Project Architecture*

Berdasarkan **Gambar 10**, setidaknya dibutuhkan empat *services* Google Cloud yang akan digunakan, diantaranya Cloud Run, Cloud Storage, Cloud SQL dan Artifact Registry. RESTful API untuk menangani *authentication request* seperti *login*, *register*, *get*, *update* dan *delete* data *user* menggunakan *framework* Express JS dengan bahasa pemrograman Javascript. Sementara RESTful API untuk menangani model *prediction request* menggunakan *framework* FAST-API dengan bahasa pemrograman Python. Model *machine learning* dibuat menggunakan Tensorflow yang akan melakukan klasifikasi sebuah *image*.

Dua RESTful API baik untuk menangani *authentication request* dan model *prediction request* akan dibangun ke dalam masing-masing image container untuk membungkus *source code*, model, dan *dependency*-nya yang kemudian disimpan di Artifact Registry. Setelah itu, akan dijalankan atau di-*deploy* menggunakan Cloud Run dan menghasilkan dua URL dengan protokol jaringan yang sudah *secure* atau dalam bentuk HTTPS. Dua URL ini akan digunakan dalam aplikasi Android dan setiap entitas *users* maupun *guests* yang melakukan *request*, maka data *users* dan data *image* akan di simpan di Cloud SQL. Sementara *file image*-nya akan disimpan ke dalam *bucket* pada Cloud Storage. Seluruh *services* yang digunakan dalam Google Cloud, akan berjalan pada *region asia-southeast2* atau Jakarta untuk meminimalisir *latency*.

3.1.4 Implementasi

Tahap implementasi pada pengembangan RESTful API ini meliputi beberapa proses, diantaranya:

a. Pembuatan *database* baik *local* untuk proses *development* maupun di Google Cloud menggunakan Cloud SQL untuk proses *production*.
b. Pembuatan *bucket* di Cloud Storage untuk menyimpan *file* gambar atau foto.
c. *Dependencies* program untuk RESTful API
d. Pembuatan docker *image* dari kode program

e. Konfigurasi *repository* pada Artifact Registry untuk menyimpan docker *image* di Google Cloud
f. Menjalankan docker *image* menggunakan Cloud Run

3.1.5 Pengujian

Pengujian RESTful API dilakukan baik selama proses *development* maupun setelah *production*. Hal ini, guna memastikan hasil yang didapatkan pada saat *development* tidak jauh berbeda dengan pada saat *production*. Pengujian yang dilakukan menggunakan metode *black box* untuk menguji fungsionalitas tanpa memperhatikan detail *internal*. Dalam konteks pengujian RESTful API, hal ini mencakup pengujian terhadap *input* dan *output* dari API tanpa memperhatikan implementasi *internal*-nya. Tahap pengujian ini menggunakan aplikasi Postman untuk menguji setiap *endpoint* yang menangani *user management* sementara untuk setiap *endpoint* yang menangani klasifikasi model menggunakan Swagger UI hasil dari *generate* otomatis dari *framework* Fast-API. Penggunaan Swagger UI dalam menguji fungsionalitas tidaklah jauh berbeda dengan menggunakan Postman.

3.1.6 Dokumentasi

Dokumentasi menjadi tahapan terakhir dari pengembangan RESTful API. Setelah tahap pengujian, setiap *endpoint* akan ditulis dokumentasinya yang akan di-*publish* ke GitHub. Komponen dari RESTful API yang ditulis meliputi *path*, *method*, *header*, *request*, dan *response*. Dokumentasi ini bertujuan untuk memberikan panduan yang jelas dan komprehensif terhadap penggunaan API.

4. HASIL DAN PEMBAHASAN

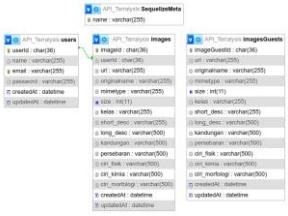
Hasil penerapan RESTful API pada Google Cloud sebelumnya telah melalui proses analisis kebutuhan sesuai dengan **Tabel 1**. Pada penelitian ini, tantangan yang dihadapi adalah biaya penggunaan layanan. Layanan yang dipilih dan digunakan diharapkan dapat beroperasi secara optimal dalam menjalankan RESTful API. Oleh karena itu, perlu dilakukan analisis terlebih dahulu. Dari hasil analisis kebutuhan ini, ditemukan empat layanan Google Cloud yang akan digunakan untuk mengimplementasikan RESTful API. Dalam penggunaannya, keempat layanan Google Cloud ini dipilih karena kesesuaian dengan rancangan dari sistem yang diharapkan mampu menjalankan RESTful API dengan *configuration instances*, CPU dan memori yang digunakan dapat efisien dan *minimum* sehingga dapat menekan biaya yang akan dikeluarkan namun tetap memperhatikan *performance* dan keandalan dari sistem.

Sebagai contoh, penggunaan Cloud Run untuk menjalankan RESTful API dari docker *image* diatur dengan minimal nol *instances* dan maksimal tiga *instances*. Dengan demikian, apabila tidak ada *request* yang masuk, maka *instances* dari Cloud Run dapat di-*scale down* hingga nol sehingga tidak dikenakan biaya. Sementara apabila terdapat banyak *request,* maka *instances* akan secara otomatis *scale* up hingga maksimal tiga *instances*. Dengan mengatur jumlah maksimal *instance,* maka memungkinkan tidak terjadi *over scale up* hingga jumlah *instances* 100 (*maximum*) yang dapat menyebabkan pembengkakan biaya layanan.

Berikut merupakan hasil yang diperoleh setelah melakukan implementasi sesuai rancangan yang dibagi menjadi hasil penerapan, pengujian dan dokumentasi.

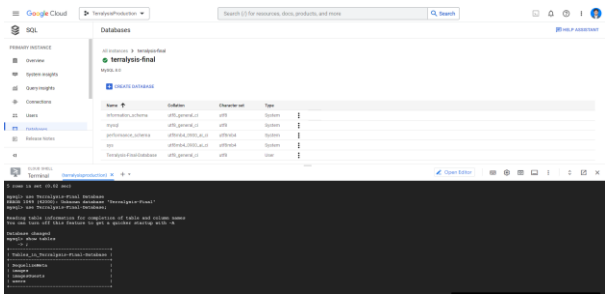
4.1 Penerapan

4.1.1 Basis Data Lokal dan Cloud SQL



Gambar 11. Basis Data Lokal

Gambar 11 menunjukkan basis data yang digunakan untuk pengembangan di mesin lokal. Basis data ini merupakan implementasi rancangan basis data sebelumnya. Terdapat perbedaan kecil yaitu penambahan tabel Sequelize Mata, yang digunakan untuk menyimpan riwayat perubahan *database* secara otomatis oleh paket Sequelize. Sequelize adalah *Object Relational Mapping* (ORM) untuk Node JS yang mendukung berbagai jenis *database*. Dengan menggunakan Sequelize, model *database* dapat dibuat dengan kode dan mempercepat migrasi ke tahap produksi.



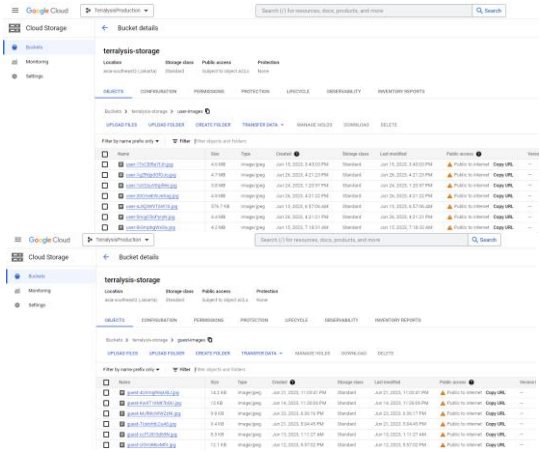
Gambar 12. Cloud SQL

Gambar 12 menunjukan halaman *databases* pada Cloud SQL. Basis data yang ada di lokal dibuat kembali

pada Cloud SQL dengan nama *instance* terralysis-final. Dalam *instance* tersebut, *database* yang digunakan diberi nama Terralysis-Final-Database. Cloud SQL mendukung berbagai macam *database engine* seperti MySQL, PostgreSQL, dan SQL *Server*. Untuk *database engine* yang digunakan pada penerapan ini adalah MySQL dengan versi 8.0. Pada gambar itu juga terdapat *cloud shell* yang digunakan untuk mengakses database dan menjalankan *query*. Cloud SQL dipilih sebagai layanan *database* pada tingkat *production* karena dapat memastikan bahwa *database* yang ada *reliable*, *secure* dan *scalable*.

4.1.2 *Bucket* Cloud Storage

Cloud Storage digunakan untuk menyimpan data yang tidak terstruktur (*unstructured data*) seperti *file* gambar. Kelas Cloud Storage yang digunakan adalah bertipe *standart* karena *file* gambar yang disimpan akan sering kali diakses dalam waktu kurang dari sebulan atau 30 hari. Hal ini mengingat terdapat fitur *history* yang digunakan oleh *user* untuk mengakses gambar yang telah di-*scan*. Dalam penerapannya, hanya membuat satu *bucket* Cloud Storage dengan nama terralysis-storage dan terbagi menjadi dua berdasarkan entitas yang mengunggah gambar.



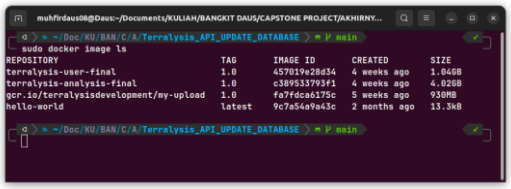
Gambar 13. Cloud Storage

4.1.3 *Dependencies* Program

Gambar 14. *Dependencies Program*

Berdasarkan **Gambar 14**, terdapat dependencies program atau *library* yang digunakan untuk membangun RESTful API. Pada sisi sebelah kiri terdapat *dependencies* untuk *authentication* API dengan menggunakan framework Express JS dan sisi sebelah kanan terdapat *dependencies* untuk prediction API yang menggunakan *framework* FAST-API. *Dependencies* ini yang akan di-*install* bersamaan dengan proses membuat *image container* dengan Docker.

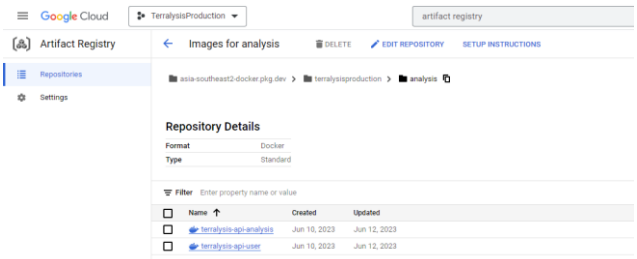
4.1.4 Docker *Image*



Gambar 15. *Docker Image*

Docker *image* yang di-*build* di *local machine* bertujuan untuk mencoba secara langsung RESTful API setelah dijadikan sebagai image *container* sebelum di-*push* ke Artifact Registry. Pada **Gambar 15** ada dua docker *image* yang dibangun untuk pengembangan RESTful API yaitu terralysis-user-final:1.0 dan terralysis-analysis-final:1.0. Dua *image* ini memiliki perbedaan ukuran atau *size* dengan selisih kurang lebih 3 gb. Perbedaan ini dipengaruhi oleh jumlah baris kode besarnya *dependencies* yang di-*install* di dalamnya. Selain itu, pada *image* dengan nama terralysis-analysis-final:1.0 juga terdapat *model machine learning* yang digunakan untuk melakukan klasifikasi gambar.

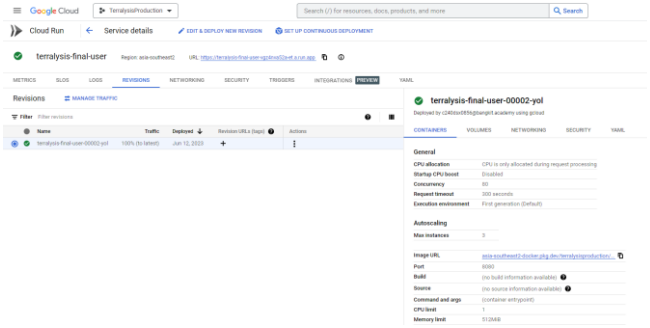
4.1.5 *Repository* Artifact Registry



Gambar 16. Artifact Registry

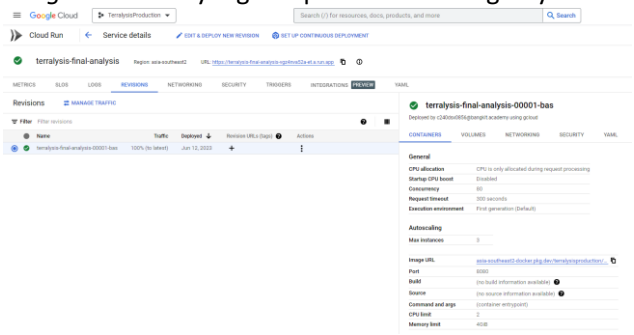
Gambar 16 merupakan tampilan halaman *repository* dari Artifact Registry yang digunakan untuk menyimpan docker *image* pada tingkat *production*. *Repository* yang dibuat bernama *analysis* yang terdapat pada *project terralysisproduction*. Docker *image* yang disimpan juga sama jumlahnya dengan yang ada di lokal namun memiliki nama yang berbeda. Untuk *authentication* API menggunakan nama terralysis-api-user sementara *prediction* API diberi nama terralysis-api-analysis.

4.1.6 Cloud Run *Services*



Gambar 17. *Cloud Run Authentication API*

Gambar 17 menunjukan *service* Cloud Run yang bernama terrlaysis-final-user. Pada gambar tersebut, terdapat keterangan region dan URL yang dapat diakses untuk menggunakan *service* atau menjalankan program yang sudah menggunakan *secure* protokol HTTPS. *Service* pada gambar tersebut dibuat dari *image container* yang ada pada Artifact Registry.



Gambar 18. *Cloud Run Prediction API*

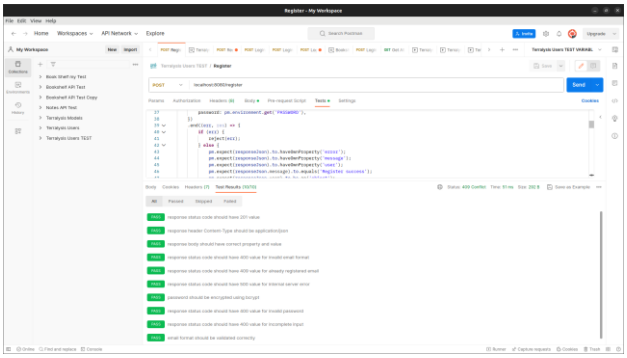
Sama seperti **Gambar 17**, pada **Gambar 18** juga menggunakan *region* yang sama yaitu *asia-southeast2* atau Jakarta dan juga memiliki URL yang dapat diakses. Nama *service* yang dibuat untuk *prediction* API adalah terralysis-final-analysis yang berasal dari *image container* di Artifact Registry.

4.2 Pengujian

Pengujian terhadap RESTful API yang telah dibuat menggunakan Postman dan Swagger UI. Pengujian dilakukan untuk memastikan fungsionalitas dari API telah sesuai dengan yang diharapkan. Berikut merupakan pengujian berdasarkan fungsi dari API.

Pengujian terhadap API dengan menggunakan Postman dapat memastikan bahwa API yang dibangun dapat berfungsi dengan baik dan memberikan respon yang sesuai harapan. Dalam penerapannya, Postman dapat digunakan untuk menguji API baik dengan cara manual maupun otomatis. Berdasarkan **Gambar 19**, dapat dilihat hasil pengujian pada salah satu endpoint API untuk melakukan *register* telah berhasil lulus dari total 10 *test case* yang diberikan.

4.2.1 *Authentication* API



Gambar 19. *Authentication API Testing dengan Postman*

Berikut merupakan contoh tabel dari salah satu pengujian untuk *login user* dengan *path* /login dan *method* POST

TABEL 3. *PENGUJIAN LOGIN USER*

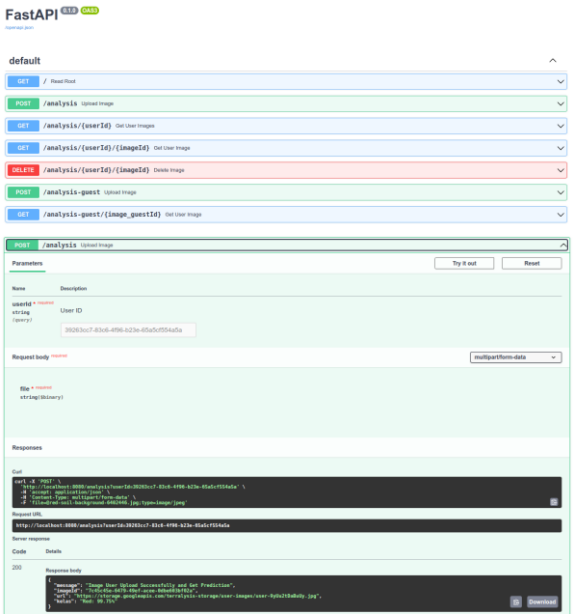
Kasus Uji	**Hasil yang Diharapkan**	**Hasil Uji**
Login success	- Memastikan *response status code* bernilai 200 (Ok) - *Response body error* bernilai *false* - *Response body message* harus sama dengan "login success" - Terdapat *response body loginResult* - *Response body loginResult* merupakan *object* - Memastikan *loginResult* berisi *userId*, *name, email*, dan *token*	*PASS*
Login menggunakan *email* yang belum terdaftar atau *email* invalid	- Memastikan *response status code* bernilai 401 (*Unauthorized*) - *Response body error* bernilai *true* - *Response body message* harus sama dengan "Invalid email or user not found"	*PASS*
Login menggunakan *password* yang salah	- Memastikan *response status code* bernilai 401 (*Unauthorized*) - *Response body error* bernilai *true* - *Response body message* harus sama dengan "Invalid password"	*PASS*
Login dengan tanpa *body request*, tanpa *email* dan/atau tanpa *password*	- Memastikan *response status code* bernilai 404 (*Bad Request*) - *Response body error* bernilai *true* - *Response body message* harus sama dengan "Incomplete request body. Please provide email and/or password."	*PASS*

- Berdasarkan **Tabel 3**, secara umum, skenario pengujian dapat dibagi menjadi dua kategori utama, yaitu skenario positif dan negatif, yang menggambarkan berbagai kemungkinan interaksi pengguna dengan sistem. Baris pertama dari tabel menggambarkan skenario positif, yang mana pengguna berhasil melakukan *login* dengan menggunakan *email* dan *password* yang valid. Dalam skenario ini, harapannya adalah bahwa sistem memberikan respons yang sesuai, seperti *response status code* 200 (Ok), pesan sukses "*login success*", dan

informasi akun pengguna yang sesuai dengan yang diharapkan. Sementara itu, skenario negatif, yang terdiri dari baris kedua hingga keempat, menggambarkan situasi yang mana pengguna mengalami kesulitan saat melakukan login. Ini mencakup kasus di mana pengguna mencoba menggunakan *email* yang belum terdaftar atau tidak valid, memasukkan *password* yang salah, atau bahkan mencoba untuk *login* tanpa mengisi *email* dan/atau *password*. Dalam skenario-skenario ini, harapannya adalah bahwa sistem memberikan respons yang sesuai dengan kegagalan, seperti *response status code* yang menunjukkan kesalahan (misalnya, 401 untuk *Unauthorized*), pesan yang menjelaskan alasan kegagalan, dan tidak ada informasi akun yang disertakan dalam *respons*.

- Hasil dari pengujian ini menunjukkan bahwa *authentication* API berhasil menangani dengan baik semua skenario positif dan negatif yang diuji, sesuai dengan harapan yang telah ditetapkan sebelumnya.

4.2.2 *Prediction* API



Gambar 20. *Prediction API Testing dengan Postman*

Berbeda dengan *authentication* API, *prediction* API selain dapat dilakukan pengujian fungsionalitas menggunakan Postman, juga dapat menggunakan Swagger UI. Hal ini karena *prediction* API dibuat dengan menggunakan *framework* Fast-API sehingga *Graphical User Interface* (GUI) yang berbasis *web* secara otomatis di-*generate* untuk melakukan *request* dan mendapatkan *response* dari setiap *endpoint* yang dibuat. **Gambar 20** menunjukkan setiap *endpoint* bersamaan dengan *method*-nya masing-masing dan terdapat bagian detail untuk *endpoint analysis* yang dalam keadaan *expanded*. Berdasarkan pengujian dengan Swagger UI, fungsionalitas dari setiap *endpoint* pada *prediction* API dapat berjalan sesuai dengan perencanaan.

Berikut merupakan tabel dari salah satu pengujian untuk *scan image by user* dengan *path* /analysis dan *method* POST.

TABEL 4. *PENGUJIAN SCAN IMAGE BY USER*

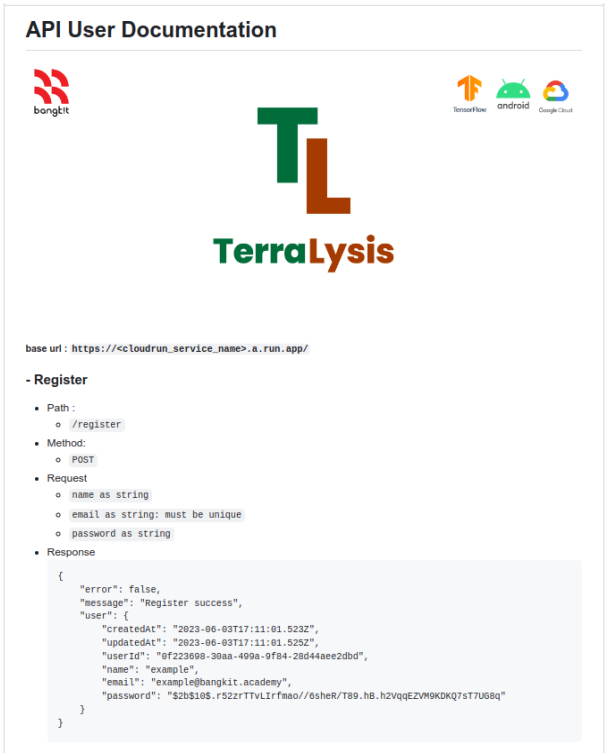
Kasus Uji	**Hasil yang Diharapkan**	**Hasil Uji**
Scan image by user success	- Memastikan *response status code* bernilai 200 (Ok) - *Response body* terdapat *message, imageId, url,* dan kelas - *Response body message* harus sama dengan "Image User Upload Successfully and Get Prediction" - Memastikan url dapat diakses dan menampilkan gambar yang di-*scan* - Hasil dari gambar yang di-*scan* tersimpan di *database*	*PASS*
Scan image tanpa mengunggah gambar dengan key *file*	- Memastikan *response status code* bernilai 422 (*Unprocessable Entity*) - *Response body msg* bernilai "field required" - *Response body type* bernilai "value_error.missing"	*PASS*
Scan image tanpa mencantumkan *user id*	- Memastikan *response status code* bernilai 422 (*Unprocessable Entity*) - *Response body msg* bernilai "field required" - *Response body type* bernilai "value_error.missing"	*PASS*
Scan image format selain JPG/JPEG	- Memastikan *response status code* bernilai 500 (*Internal Server Error*)	*PASS*

Berdasarkan **Tabel 4**, skenario pengujian terbagi menjadi beberapa kasus uji yang mencakup berbagai interaksi pengguna dengan sistem. Baris pertama dari tabel mewakili skenario positif di mana pengguna berhasil melakukan pemindaian gambar. Dalam skenario ini, sistem diharapkan memberikan *respons* yang sesuai, termasuk *response status code* 200 (Ok), serta adanya *message*, *imageId*, *url*, dan kelas dalam *response body*. Pesan yang diharapkan adalah "*Image User Upload Successfully and Get Prediction*". Selain itu, harapannya adalah bahwa URL dapat diakses dan menampilkan gambar yang dipindai, dan hasil dari gambar yang dipindai disimpan dalam *database*. Sementara itu, skenario negatif ditunjukkan oleh baris kedua hingga keempat. Ini melibatkan situasi di mana pengguna mengalami kesulitan saat melakukan pemindaian gambar dengan berbagai kondisi, seperti tidak mengunggah gambar, tidak mencantumkan *user id*, atau mengunggah gambar dalam format selain JPG/JPEG. Dalam skenario-skenario ini, sistem diharapkan memberikan respons yang sesuai dengan kegagalan, termasuk *response status code* yang sesuai

(misalnya, 422 untuk *Unprocessable Entity* atau 500 untuk *Internal Server Error*), serta pesan yang menjelaskan alasan kegagalan dan tipe kesalahan yang terjadi.

Hasil dari pengujian ini menunjukkan bahwa *prediction* API berhasil menangani dengan baik semua skenario yang diuji, baik positif maupun negatif, sesuai dengan harapan yang telah ditetapkan sebelumnya. *Respons* dari sistem konsisten dengan harapan, menegaskan kehandalan fungsionalitas sistem dalam berbagai situasi penggunaan.

4.3 Dokumentasi



Gambar 21. Dokumentasi APi

Dokumentasi API sangatlah penting untuk memudahkan orang lain dalam memahami penggunaan API. Melalui dokumentasi, pengguna dapat menemukan informasi penting seperti daftar *endpoint* yang tersedia, parameter yang dibutuhkan, format data yang diterima, dan contoh penggunaan API. **Gambar 21** menampilkan dokumentasi untuk API *authentication* atau yang berhubungan dengan *users* dengan menggunakan platform GitHub. Dokumentasi ini mencantumkan *base* URL, nama *endpoint*, path, *method*, serta *request* dan *response* yang diharapkan dari API tersebut. Dengan adanya dokumentasi ini, pengguna dapat dengan jelas memahami cara menggunakan API tersebut. Pengguna dapat mempermudah dalam mengintegrasikan API dengan aplikasi berbasis *mobile* karena dapat memahami cara melakukan *request* dan menerima *respons* dari API. Selain itu dapat mengurangi kesalahan dalam penggunaan API dan memfasilitasi dalam melakukan *debungging*. Pengguna dapat menjadikan dokumentasi sebagai referensi dalam melakukan *debungging* dan memeriksa kembali untuk mengetahui penggunaan dari API.

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil pengembangan RESTful API untuk aplikasi klasifikasi berbasis *mobile* pada Google Cloud yang telah dilakukan dapat disimpulkan bahwa:

a. Arsitektur REST dapat diimplementasikan dengan baik dalam pengembangan API dan telah memperoleh keberhasilan pada pengujian fungsionalitas secara menyeluruh melalui berbagai macam skenario pengujian.
b. Penggunaan Google Cloud sebagai penyimpanan data dan komputasi untuk menjalankan RESTful API dapat berjalan dengan baik sesuai dengan hasil rancangan sehingga dapat digunakan untuk diakses oleh aplikasi klasifikasi jenis tanah berbasis *mobile*.

5.2 Saran

Berdasarkan hasil pengembangan RESTful API untuk aplikasi klasifikasi berbasis mobile pada Google Cloud yang telah dilakukan dapat disarankan beberapa hal sebagai berikut:

a. RESTful API yang telah dibuat dapat dikembangkan lagi dengan menambahkan *service* dan *endpoint* agar lebih banyaknya informasi yang dapat diproses dan diterima sehingga aplikasi berbasis *mobile* yang akan dikembangkan menjadi lebih kompleks.
b. Untuk mendapatkan informasi yang lebih banyak lagi mengenai suatu jenis tanah, RESTful API dapat dikembangkan dengan melakukan integrasi terhadap teknologi IoT. Dengan demikian, aplikasi diharapkan dapat memberikan informasi yang lebih akurat dan aktual kepada pengguna, memungkinkan pemantauan kondisi tanah secara langsung dan responsif bahkan dapat digunakan untuk pemantauan kondisi tanah secara *real-time*
c. Selain melakukan pengujian fungsionalitas, RESTful API yang dibuat juga dapat diuji melalui *performance test* dengan berbagai skenario untuk mengetahui penggunaan sumber daya pada layanan Google Cloud. Hal ini dapat membantu dalam mengoptimalkan tipe komputasi yang dipilih.

UCAPAN TERIMA KASIH

Ucapan terima kasih disampaikan kepada tim *mobile development* maupun tim *machine learning* dalam menyelesaikan proyek akhir Bangkit Academy 2023. Ucapan terima kasih juga disampaikan kepada mentor, *advisor* dan *instructor* serta seluruh tim Bangkit yang telah menjalankan program pendidikan yang luar biasa dampaknya.

DAFTAR PUSTAKA

[1] K.S. Abdul, "Ilmu Tanah", Edition 2, Bandar Lampung: Global Madani Press, 2020, pp. 1.

[2] S. Br Kembarem. A. H. Sidiq, dkk, "Penerapan IoT pada Pendeteksi Jenis Tanah Berbasis Web", Jurnal Ilmiah KOMPUTASI, vol. 21, no. 1, hal 71-78, 2022.

[3] Y. Indrawati, A. Tristiyono, "Aplikasi Pengenalan Jenis Tanah untuk Lahan Pertanian dengan Menggunakan Metode Euclidean Distance", MIND Journal, vol. 1, no. 1, hal. 26-36, 2016.

[4] I. H. Robbani, E. Trisnawati, dkk, "Aplikasi Mobile Scotect: Aplikasi Deteksi Warna Tanah dengan Teknologi Citra Digital pada Android", JTIIK, vol. 3, no. 1, hal. 19-26, 2016.

[5] Ilham, Pasnur, "Aplikasi Pengolah Citra untuk Menentukan Jenis Tanah pada Lahan Pertanian", JTIKA, vol. 8, no. 2, hal. 89-96, 2018.

[6] Jiri Hradil, Vilém Sklenak, "Practical Implementation of 10 Rules for Writing REST APIs", JOURNAL OF SYSTEMS INTEGRATION, 45, 2017.

[7] Scribner, Kenn dan Seely Scott. "Effective REST Services via .NET: For .NET Framework 3.5", First Edition, Addison-Wesley Professional, 2009.

[8] Hidayatullah, Priyanto dan Jauhari Khairul Kawistara, "Pemrograman Web", Bandung: Informatika Bandung, 2015.

[9] Kleppmann, Martin dan Alastair R. Beresford, "A Conflict-Free Replicated JSON Datatype", University of Cambridge Computer Laboratory, Cambridge, 2017.

[10] Jones, Michael B. dkk, "JSON Web Token (JWT). Internet Engineering Task Force (IETF)", 2015.

[11] E. Riana, "Implementasi Cloud Computing Technology dan Dampaknya Terhadap Kelangsungan Bisnis Perusahaan Dengan Menggunakan Metode Agile dan Studi Literatur," JURIKOM (Jurnal Ris. Komputer), vol. 7, no. 3, p. 439, 2020, doi: 10.30865/jurikom.v7i3.2192.

[12] T. Hidayat, "Encryption Security Sharing Data Cloud Computing By Using Aes Algorithm: a Systematic Review," Teknokom, vol. 2, no. 2, pp. 11–16, 2019, doi: 10.31943/teknokom.v2i2.41.

[13] D. W. Sari, S. Kosasi, dkk, "Pemanfaatan RESTful Web Services pada Perangkat Lunak Penyewaan Lapangan Badminton", InfoSys Journal, vol. 6, no. 2, hal. 103-114, 2022.

[14] L. Pradana, A. Ambarwari, S. D. Putra, "Rancang Bangun Web Services API dan Dokumentasi REST API Web Portal Unit Kegiatan Mahasiswa di Politeknik Negeri Lampung",Jurnal Sistem dan Teknologi Informasi, vol. 1, no. 1, hal. 9-18, 2023.

[15] I. A. K. P. Paramitha, D. M. Wiharta, I M. A. Suyadnya, "Perancangan dan Implementasi RESTful API pada Sistem Informasi Manajemen Dosen Universitas Udayana", Jurnal SPEKTRUM, vol. 9, no. 3, hal. 15-23, 2022.