JTIM : Jurnal Teknologi Informasi dan Multimedia

p-ISSN : 2715-2529
e-ISSN : 2684-9151
https://journal.sekawan-org.id/index.php/jtim

Pengembangan *Back-end* pada Aplikasi *Smart Nutrition* Berbasis *Node.js* dan *Hapi* dengan Integrasi *Google Cloud Platform*

Lalu Kurnia M. Ridho [1], Jarir [2], dan Akbar Juliansyah [3]*

[1] Program Studi Pendidikan Teknologi Informasi, Universitas Pendidikan Mandalika;, Infonesia.

* Korespondensi: akbarjuliansyah@undikma.ac.id

Sitasi: Ridho, L. K. M.; Jarir, J.; Juliansyah, A. (**2025**). Judul. JTIM: Pengembangan Back-end pada Aplikasi Smart Nutrition Berbasis Node.js dan Hapi dengan Integrasi Google Cloud Platform. Teknologi Informasi Dan Multimedia, 7(4), 754-764
https://doi.org/10.35746/jtim.v7i4.812

Diterima: 01-07-2025
Direvisi: 20-08-2025
Disetujui: 10-09-2025

Abstract: Advances in digital technology drive the need for smart and integrated nutrition monitoring systems, but developers often focus only on features without considering architectural design. This research aims to develop and implement a RESTful API on the Google Cloud Platform (GCP) backend for the Smart Nutrition App, which has the ability to support daily fruit consumption tracking powered by machine learning. The methodology used is based on the Software Development Life Cycle (SDLC) model, including requirements analysis, cloud-native system design, modular API development using Node.js and Hapi.js, functional testing in Postman, and stress testing in K6 to 4000 virtual users. The results show that the RESTful API can sustain a load of up to 1000 virtual users with 0% error rate, but performance degrades very sharply above this level, to the point where the error rate is 100% at 4000 users. These findings indicate the need for infrastructure optimization to support the demands of real applications. The result of this research is that the system meets the functional requirements and performs well at small scale but requires infrastructure improvements such as load balancing and auto-scaling for scaled environments. The main contribution of this research is to present a scalable and modular backend framework for Smart Nutrition App as a future reference when developing similar systems.

Keywords: RESTful API, Google Cloud Platform, Backend, Smart Nutrition App, Cloud Computing.

Abstrak: Kemajuan teknologi digital mendorong kebutuhan akan sistem pemantauan nutrisi yang cerdas dan terintegrasi, namun sering kali pengembang hanya berfokus pada fitur tanpa mempertimbangkan desain arsitektur. Penelitian ini bertujuan untuk mengembangkan dan mengimplementasikan *RESTful API* pada *backend Google Cloud Platform (GCP)* untuk Smart Nutrition App, yang memiliki kemampuan untuk mendukung pelacakan konsumsi harian buah-buahan yang didukung oleh *machine learning*. Metodologi yang digunakan berdasarkan model *Software Development Life Cycle (SDLC),* termasuk analisis kebutuhan, desain sistem *cloud-native,* pengembangan *API* modular menggunakan *Node.js* dan *Hapi.js*, pengujian fungsional di *Postman*, dan *stress testing* di *K6* kepada 4000 pengguna *virtual*. Hasilnya menunjukkan bahwa *RESTful API* dapat mempertahankan beban hingga 1000 pengguna *virtual* dengan tingkat kesalahan 0%, tetapi kinerja menurun sangat tajam di atas tingkat ini, hingga mencapai titik di mana tingkat kesalahan 100% pada 4000 pengguna. Temuan ini menunjukkan perlunya optimasi infrastruktur untuk mendukung tuntutan aplikasi yang sesungguhnya. Hasil dari penelitian ini adalah bahwa sistem memenuhi persyaratan fungsional dan berkinerja baik dalam skala kecil tetapi membutuhkan pexiganan infrastruktur seperti *load balancing* dan penskalaan otomatis untuk lingkungan yang ditingkatkan. Kontribusi utama dari penelitian ini adalah menyajikan kerangka kerja *backend* yang

dapat diskalakan dan modular untuk Smart Nutrition App sebagai referensi di masa depan ketika mengembangkan sistem serupa.

Kata kunci: *RESTful API, Google Cloud Platform, Backend, Smart Nutrition App*, Komputasi Awan

1. Pendahuluan

Perkembangan teknologi digital di bidang kesehatan telah menghasilkan berbagai solusi baru, termasuk pelacakan asupan nutrisi secara otomatis dan personal. Salah satu solusi terbaru yang dikembangkan adalah penggunaan teknologi *machine learning (ML)* dan sensor multimodal untuk mendeteksi dan memperkirakan asupan makanan pengguna secara *real time*. Aplikasi pengenalan gambar yang disempurnakan dengan *AI* ditemukan dapat menghitung kandungan *makronutrien* sama akuratnya dengan pencatatan manual, dan oleh karena itu merupakan alternatif masa depan yang menarik untuk pencatatan nutrisi sehari-hari [1]. Sistem pemantauan kesehatan berbasis *wearable* dan pintar telah berkembang pesat dalam permintaan karena multifungsi, fleksibilitas, dan penerapannya pada pemantauan kesehatan *real-time* [2]. Perangkat sensor yang dapat dikenakan untuk membantu memantau detak jantung dan pernapasan sedang dikembangkan di seluruh dunia.

Inovasi lain datang dari penerapan sistem pengenalan makanan berdasarkan gambar untuk meningkatkan penilaian nutrisi secara otomatis. Dalam sebuah tinjauan sistematis, telah ditunjukkan bahwa teknologi pengenalan gambar makanan mampu mengidentifikasi makanan secara otomatis, memperkirakan volume, serta menentukan kalori dan nutrisi, sehingga dapat menggantikan metode laporan manual konvensional yang selama ini rentan terhadap bias dan ketidakakuratan [4]**.** Penerapan teknologi ini memiliki potensi besar untuk memungkinkan pelacakan nutrisi harian berbasis gambar, terutama dalam upaya untuk memungkinkan personalisasi kebutuhan nutrisi di antara penduduk kota.

Selain itu, sistem pengenalan makanan yang diaktifkan dengan *deep learning* berkembang pesat dan tidak hanya dirancang untuk estimasi nutrisi, tetapi juga digunakan untuk memantau penyakit kronis seperti diabetes dan obesitas [5]. Manajemen data yang kompleks dan *real-time*, terutama dalam klasifikasi makanan dari foto digital, diperlukan untuk implementasi sistem semacam itu. Penelitian ini juga menunjukkan perlunya kerangka kerja *backend* yang kuat, modular, dan terukur dalam mendukung proses klasifikasi yang efektif, mengelola data nutrisi, dan komunikasi antara *server* dan antarmuka pengguna.

Namun, ada sejumlah masalah yang masih terbuka dalam penerapan sistem pemantauan nutrisi yang didukung teknologi. Sebagian besar penelitian yang dilakukan sejauh ini berfokus pada pembangunan antarmuka pengguna *(frontend)* atau pada mekanisme prediksi berbasis pembelajaran mesin tanpa memperhatikan bagaimana sistem *backend* dirancang untuk menangani kebutuhan dunia nyata seperti interaksi intensif data, skalabilitas, dan keandalan sistem. Fokus utama penelitian ini bukanlah untuk menciptakan aplikasi nutrisi yang baru secara keseluruhan, melainkan untuk merancang, mengimplementasikan, dan menguji sebuah arsitektur *backend cloud-native* pada Aplikasi *Smart Nutrition*. Arsitektur ini dibangun dengan memanfaatkan layanan-layanan khusus dari *Google Cloud Platform*, seperti *Compute Engine* untuk server virtual dan *Cloud Storage* untuk penyimpanan data, sehingga lebih adaptif dibanding sekadar penggunaan server tunggal konvensional. Dengan pendekatan ini, Aplikasi *Smart Nutrition* tidak hanya memperoleh pondasi *backend* yang lebih tangguh, tetapi hasil rancangan arsitektur ini juga diharapkan dapat menjadi cetak biru *(blueprint)* yang relevan bagi pengembangan aplikasi sejenis di masa depan. Faktanya, dalam

pemantauan nutrisi secara *real-time,* pipeline data dengan kemampuan untuk mengonsumsi aliran data yang terus menerus, dengan jaminan kualitas data saat dikonsumsi, dan dengan pemantauan yang responsif. Kerangka kerja untuk pemantauan kualitas data secara *real-time* telah diusulkan untuk membantu sistem streaming dengan berbagai aplikasi pemantauan berkelanjutan [6].

Beberapa penelitian telah dilakukan untuk menyarankan integrasi teknologi pintar dalam sistem pelacakan nutrisi, namun belum ada yang mengintegrasikan semua elemen yang diperlukan secara sistematis, seperti desain *RESTful API* modular, arsitektur skalabilitas *cloud-native*, otentikasi berbasis *token (JWT)*, dan uji coba kinerja sistem. Membangun sebuah *RESTful API* sebagai jembatan antara aplikasi pengguna dan *server*. *API* inilah yang menangani semua permintaan, seperti registrasi pengguna, prediksi gambar buah, dan pengambilan riwayat nutrisi. *JWT* seperti sebuah tiket masuk konser atau kartu akses sementara. Setelah Anda berhasil *login* (menunjukkan KTP sekali), sistem akan memberikan Anda sebuah "tiket" digital *(JWT)* yang aman. Semua fitur ini harus diintegrasikan agar sistem pelacakan nutrisi dapat diandalkan, terutama jika akan digunakan oleh ribuan pengguna sekaligus. Penelitian lain juga menyoroti pentingnya arsitektur desain berbasis *cloud* yang dapat diskalakan untuk memfasilitasi pemrosesan data nutrisi secara *real-time* [7] .Artinya, strategi *end-to-end back-end* dengan ilmu pembelajaran mesin adalah area yang belum banyak diteliti dalam literatur.

Pada penelitian ini tujuannya adalah untuk membuat dan menerapkan *Google Cloud Platform (GCP) RESTful API* untuk mengaktifkan *Smart Nutrition App,* sebuah sistem pelacakan nutrisi harian berdasarkan klasifikasi buah dan penyesuaian konsumsi. Penggunaan API memungkinkan aplikasi untuk terus mengembangkan diri di masa depan dengan kemampuan menambah atau menghapus layanan tergantung pada permintaan [8] . Sistem ini telah disiapkan untuk menyediakan transfer data yang toleran terhadap kesalahan, memungkinkan integrasi pembelajaran mesin, dan mendukung skalabilitas dengan penggunaan infrastruktur *cloud*. Proses pengembangan dilakukan dengan mengadopsi pendekatan *Software Development Life Cycle (SDLC)* yang mencakup pengumpulan persyaratan, desain sistem, pengembangan *API*, pengujian fungsional dan pengujian *stres*, dan penyebaran dalam lingkungan *cloud.*

Penelitian ini benar-benar melakukan upaya pengembangan sistem *backend* pelacakan nutrisi dengan pendekatan *holistik* dan *pragmatis*. Pertama, penelitian ini menghasilkan *API RESTful* modular dari *Node.js* dan kerangka kerja *Hapi.js* yang dikembangkan berdasarkan prinsip pemisahan urusan, sehingga memudahkan untuk dipelihara dan dikembangkan dalam jangka panjang. *API* ini memiliki fitur-fitur penting seperti otentikasi *JWT,* manajemen pengguna, klasifikasi buah dengan berbasis model pembelajaran mesin, pelaporan konsumsi, dan manajemen data nutrisi, yang semuanya terbagi dalam *endpoint* yang terorganisir. Kedua, penelitian ini mengembangkan arsitektur *cloud-native* dengan menggunakan layanan *Google Cloud Platform (GCP),* termasuk *Compute Engine dan Cloud Storage*. Konsep ini menekankan manfaat skalabilitas otomatis dan efisiensi manajemen sumber daya dalam pengembangan aplikasi berbasis *cloud* [9].

Ketiga, dan yang paling penting, penelitian ini melakukan uji kinerja sistem yang komprehensif melalui simulasi beban dengan ribuan *virtual users (VU)* melalui alat *K6*. Hasil pengujian ini memberikan bukti empiris tentang toleransi sistem dan menegaskan persyaratan penskalaan otomatis dan penyeimbangan beban, yang sering kali diabaikan. Selain itu, pendekatan pengujian yang membandingkan fungsionalitas dan kinerja menambah metodologi penelitian teknologi *backend* berbasis *cloud.* Oleh karena itu, penelitian ini tidak hanya menghasilkan prototipe teknis yang dapat segera digunakan, tetapi juga menambah badan teknis praktik terbaik dalam menciptakan *backend* yang dapat diskalakan, modular, dan aman untuk aplikasi kesehatan.

2. Metode Penelitian

Penelitian ini merupakan penelitian terapan yang bertujuan untuk merancang dan mengimplementasikan *RESTful API* pada *Google Cloud Platform (GCP)* sebagai *backend* untuk aplikasi *Smart Nutrition App*. Metodologi penelitian yang digunakan berdasarkan kerangka kerja *Software Development Life Cycle (SDLC),* yang dimulai dengan mengidentifikasi kebutuhan fungsional dan non-fungsional yang signifikan dari tinjauan literatur, misalnya kebutuhan otentikasi yang aman dan skalabilitas sistem. Untuk menjawab kebutuhan ini, arsitektur layanan mikro berbasis *cloud-native* berbasis *Node.js* dan *Hapi* dibuat dan diterapkan. Terakhir, serangkaian uji coba fungsional dan *stress test* dilakukan untuk memvalidasi keandalan dan kinerja sistem di bawah beban tinggi

2.1. Pengumpulan Kebutuhan

Tahap awal dimulai dengan mengidentifikasi persyaratan fungsional dan non-fungsional. Persyaratan fungsional diperoleh melalui tinjauan literatur serta diskusi dengan calon pengguna Aplikasi *Smart Nutrition*, yang memberikan masukan terkait fitur pemantauan nutrisi yang dibutuhkan. Sedangkan persyaratan non-fungsional ditetapkan melalui koordinasi dengan anggota tim pengembang bagian *frontend*, agar arsitektur *backend* yang dirancang selaras dengan kebutuhan integrasi antarmuka pengguna.

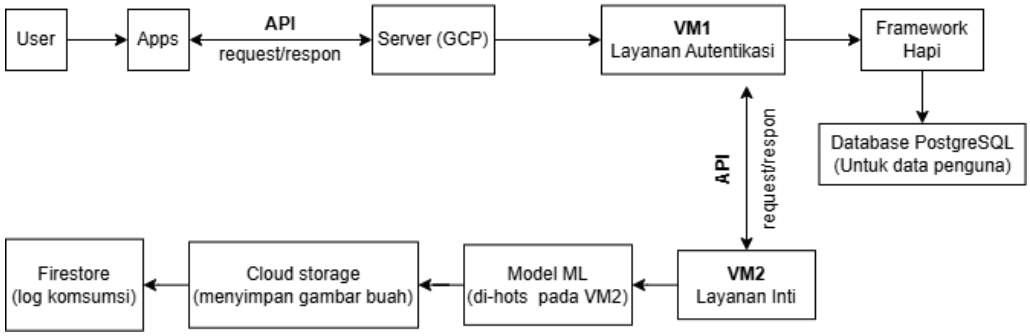
- Kebutuhan Fungsional (KF):
  - (KF1) Sistem harus mengautentikasi pengguna dengan aman *(registrasi, login).*
  - (KF2) Sistem harus menyediakan endpoint untuk menerima gambar dan mendeteksi jenis buah melalui model *ML.*
  - (KF3) Sistem harus dapat menyimpan dan menampilkan riwayat konsumsi gizi
- Kebutuhan Non-Fungsional (KNF):
  - (KNF1) Sistem harus dapat menangani setidaknya 1000 pengguna secara bersamaan dengan tingkat kesalahan yang sangat rendah (target: 0%).
  - (KNF2) Arsitektur harus modular dan dapat diskalakan di masa depan.

2.2. Perancangan Sistem dan Infrastruktur

Desain sistem menggunakan pendekatan layanan mikro *(microservices)* untuk meningkatkan skalabilitas dan modularitas. Gambar 1 mengilustrasikan bahwa sistem dibagi menjadi dua layanan utama yang berjalan pada mesin virtual *(VM)* terpisah di Google Compute Engine. Untuk mengatur lalu lintas permintaan dari klien, sistem menggunakan *API Gateway* yang berfungsi sebagai pintu masuk tunggal sebelum permintaan diteruskan ke masing-masing layanan. Layanan Otentikasi *(VM1)* bertanggung jawab dalam proses autentikasi dan otorisasi pengguna, dengan mengelola informasi kredensial di basis data relasional *PostgreSQL*. Proses autentikasi dilakukan langsung di dalam layanan ini (bukan melalui pihak ketiga), sehingga kontrol terhadap data pengguna tetap berada pada sistem. Sementara itu, Layanan Inti *(VM2)* menangani tugas-tugas logika bisnis yang penting, seperti prediksi konsumsi buah dan pelacakan nutrisi. *VM2* memanfaatkan *Cloud Storage* untuk menyimpan sumber daya gambar dan *Cloud Firestore* sebagai basis data *NoSQL* yang fleksibel untuk mencatat statistik konsumsi setiap hari.

Arsitektur sistem telah dirancang dengan keputusan fundamental untuk membagi layanan menjadi dua *Virtual Machine (VM)* dan mengimplementasikan pendekatan *database hybrid* untuk memaksimalkan efisiensi dan skalabilitas. Pemisahan *VM* dilakukan dengan mengalokasikan *VM1* untuk menangani layanan otentikasi seperti manajemen data pengguna, sedangkan *VM2* hanya melakukan operasi *machine learning* yang intensif secara komputasi. Metodologi ini mencegah beban kerja proses prediksi gambar agar tidak menghambat operasi layanan autentikasi aplikasi. Sistem ini juga menggunakan

pendekatan database *hybrid* untuk mengelola berbagai jenis data secara efisien. *PostgreSQL* digunakan untuk menyimpan data relasional dan terstruktur seperti profil pengguna dan sesi di mana integritas data dan dukungan untuk kueri yang kompleks dianggap paling penting. Sebaliknya, *Firestore* sebagai basis data *NoSQL* digunakan untuk menyimpan data prediksi pemindaian gambar yang fleksibel dan tidak terstruktur sehingga data dokumen dapat disimpan dan diambil secara efisien tanpa hubungan yang rumit antar data.



Gambar 1. Rancangan arsitektur

2.3. Perancangan dan Implementasi Model Machine Learning

Untuk mendukung fungsionalitas utama aplikasi terkait klasifikasi buah, penelitian ini membangun dan merancang model *machine learning* yang dikustomisasi sesuai kebutuhan. Model ini dikembangkan berdasarkan arsitektur *Convolutional Neural Network (CNN)*, yang telah terbukti kuat untuk klasifikasi gambar. Proses kustomisasi dilakukan pada tahap penyusunan *dataset* dan pelatihan model. Dataset disusun secara manual melalui pengambilan gambar buah secara langsung dengan memfoto setiap jenis buah dari berbagai sudut *(angle)* sehingga diperoleh variasi data yang lebih representatif dan juga beberapa gambar buah yang diambil dari internet. *Dataset* ini terdiri dari 1.244 gambar yang mewakili 10 kelas (jenis buah) dan 1 kelas (non-buah) dan merupakan hasil dokumentasi pribadi, bukan dataset publik. Seluruh proses pengembangan, mulai dari perancangan arsitektur hingga pelatihan model, dilakukan menggunakan kerangka kerja *TensorFlow.* Dalam arsitektur sistem, model yang telah dilatih di-hosting pada mesin virtual kedua *(VM2),* yang terintegrasi dengan layanan inti aplikasi untuk memastikan latensi rendah saat proses prediksi gambar dijalankan.

2.4. *Implementasi RESTful API*

Pengembangan *RESTful API* pada penelitian ini menggunakan bahasa pemrograman *JavaScript* dengan standar *ECMAScript 6 (ES6). Framewor*k yang digunakan adalah *Hapi.js* versi 21.x, karena fleksibilitasnya dalam membangun layanan *web* dan kemampuannya untuk mendukung desain modular yang terorganisir dengan baik, digunakan sebagai *framework*. Lingkungan untuk pengembangan dijalankan pada lingkungan *runtime Node.js* versi 20.x, yang menawarkan kinerja berkualitas dan dukungan ekosistem yang kaya. *API* dirancang mengikuti prinsip-prinsip *RESTful*, di mana format pertukaran data adalah *JSON* untuk memiliki interoperabilitas antar sistem. Otentikasi berbasis *token* digunakan untuk tujuan otentikasi, di mana *JSON Web Token (JWT)* digunakan untuk memastikan akses yang aman ke titik akhir yang dapat diakses.

2.5. Mekanisme Prediksi dan Pengambilan Data Nutrisi

Proses untuk mengenali nutrisi dari gambar buah yang dikirim oleh pengguna tidak terjadi dalam satu langkah, melainkan melalui alur kerja dua tahap yang mengintegrasikan layanan *machine learning* dan basis data. Alur ini dirancang untuk memastikan akurasi dan efisiensi, yang melibatkan langkah-langkah berikut saat pengguna mengakses *endpoint /predict:*

Tahap 1: Klasifikasi Jenis Buah oleh Model *ML*. Ketika pengguna mengirimkan sebuah gambar, permintaan tersebut diterima oleh Layanan Inti *(VM2)*. Gambar kemudian diteruskan ke model *machine learning CNN* yang telah di-hosting di *VM* yang sama. Tugas utama model ini adalah menganalisis gambar dan mengembalikan hasil prediksi berupa nama atau kelas dari buah tersebut (contoh: "Apel", "Pisang")

Tahap 2: Pencarian Data Nutrisi dalam Kode Aplikasi. Setelah nama buah berhasil diidentifikasi, sistem tidak melakukan kueri ke basis data eksternal. Sebaliknya, nama buah tersebut (misalnya, "Apel") digunakan untuk melakukan pencarian langsung di dalam sebuah struktur data, seperti array objek, yang telah didefinisikan di dalam kode program Layanan Inti *(VM2*). Proses ini secara internal akan mencari objek yang cocok dengan nama buah hasil prediksi untuk mendapatkan detail nutrisinya.

2.6. Menguji Sistem

Pengujian dilakukan dalam dua fase utama, yaitu pengujian fungsional dan pengujian *stress.* Fase pengujian fungsional dilakukan dengan menggunakan *Postman* dengan tujuan untuk memastikan bahwa semua titik akhir *API* berfungsi sesuai dengan spesifikasi sistem. Pengujian ini mencakup berbagai skenario, baik *input* yang valid maupun tidak valid, dan memastikan hal-hal seperti validasi data, manajemen kesalahan, dan protokol otentikasi pengguna. Selain itu, untuk memastikan kinerja sistem di bawah beban tinggi, pengujian *stress* dilakukan dengan menggunakan *K6*, yaitu sebuah *open-source load testing tool* berbasis JavaScript yang dirancang untuk menguji kinerja *API*, *microservices*, dan situs web modern. K6 memungkinkan pengembang untuk mensimulasikan ribuan pengguna virtual secara bersamaan dengan menulis skrip uji beban dalam JavaScript. Hal ini dilakukan secara bertahap dengan 1000 hingga 4000 pengguna virtual untuk mensimulasikan penggunaan secara bersamaan. Beberapa parameter yang diperiksa dalam pengujian ini adalah waktu respons, jumlah permintaan per detik yang dapat ditangani *(throughput),* dan tingkat kesalahan, untuk menilai seberapa besar *API* dapat mempertahankan kinerja dan kestabilannya di bawah beban berat.

2.7. Deployment

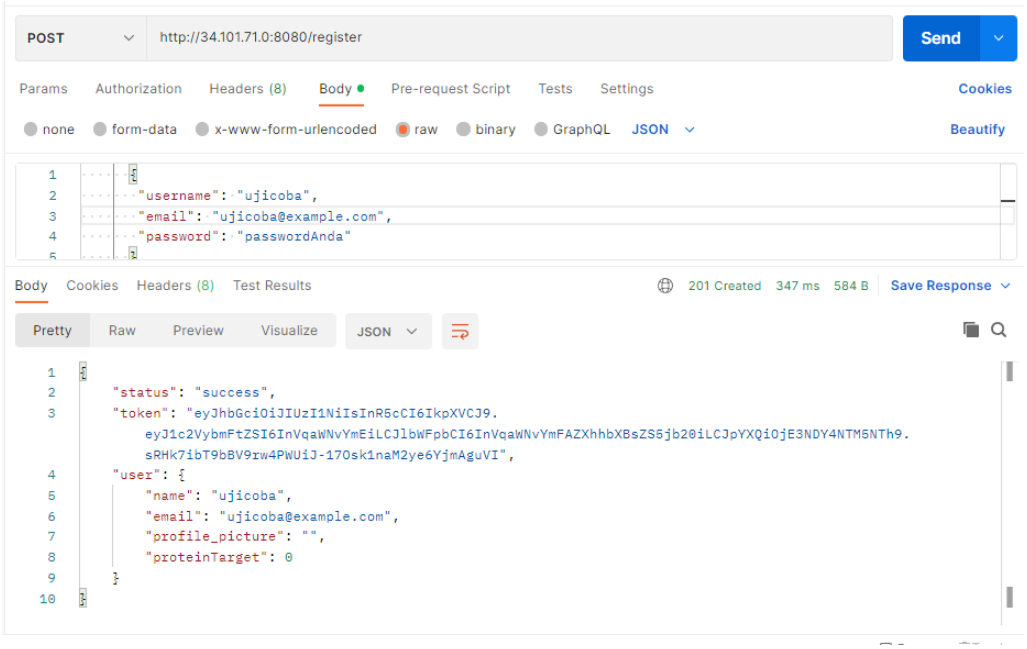
Penerapan *API* dilakukan dengan menggunakan dua mesin *virtual* dari layanan *Google Compute Engine.* Satu mesin *virtual* digunakan untuk menangani *API* utama, misalnya sistem autentikasi dan pengguna. Mesin virtual lainnya digunakan untuk menjalankan *API* yang melibatkan integrasi *machine learning,* dalam hal ini, penanganan pemrosesan data buah. Opsi keamanan juga menjadi prioritas utama dalam penerapan ini, di mana pengaturan *firewall* dan pengaturan hak akses diberikan kepada setiap layanan melalui *fitur Identity and Access Management (IAM role)* yang terdapat pada *GCP.*

3. Hasil dan Pembahasan

3.1. Implementasi RESTful API Smart Nutrition App

Keberhasilan implementasi ini secara langsung memvalidasi keputusan-keputusan kunci dalam desain arsitektur. Fungsionalitas otentikasi pengguna yang berjalan lancar *(registrasi, login, logout),* membuktikan keandalan Layanan Otentikasi pada *VM1* yang didukung oleh *database PostgreSQL* untuk menjamin konsistensi data kredensial. Sementara itu, keberhasilan fungsi prediksi gambar. menegaskan efektivitas penempatan model *ML* pada *VM2*, yang sengaja dirancang terpisah untuk menangani beban kerja komputasi tinggi tanpa mengganggu layanan lainnya. Keberhasilan implementasi menunjukkan bahwa desain arsitektur sistem dapat dicapai secara fungsional. Temuan ini konsisten dengan studi sebelumnya yang menemukan bahwa implementasi *REST API* berbasis layanan mikro untuk aplikasi manajemen nutrisi di *cloud* telah divalidasi menggunakan otomatisasi *(Postman, JMeter)* untuk fungsionalitas dan kinerja *backend* [10]. Selain itu, sebuah tinjauan sistematis terhadap teknik pengujian *RESTful API* menyoroti

peran yang dimainkan oleh pendekatan berbasis spesifikasi (misalnya, *OpenAPI*) dan otentikasi *token*/*JWT* dalam memastikan keamanan dan keandalan *API* [11] .



Gambar 2. Respon Endpoint Register

Secara khusus, untuk menentukan keefektifan fungsional dari fitur prediksi *machine learning*, kinerja model klasifikasi buah dievaluasi. Model yang telah diimplementasikan diuji untuk mengukur seberapa akurat model tersebut dapat membedakan kelas buah yang ditentukan. Hasil evaluasi menunjukkan bahwa model tersebut memiliki tingkat akurasi yang diinginkan untuk digunakan dalam aplikasi. Model metrik kinerja, seperti akurasi.

3.2. *Hasil Pengujian Fungsionalitas RESTful API*

Pengujian telah dilakukan untuk setiap *API endpoint* untuk memastikan validitas dan stabilitas fungsionalitas sistem. Tabel 1 menunjukkan bahwa setiap *endpoint API* mampu merespon sesuai kondisi *input*, baik dalam skenario normal maupun kesalahan, yang menegaskan kestabilan dan keandalanya.

Tabel 1. Hasil Pengujian Fungsionalitas RESTful API

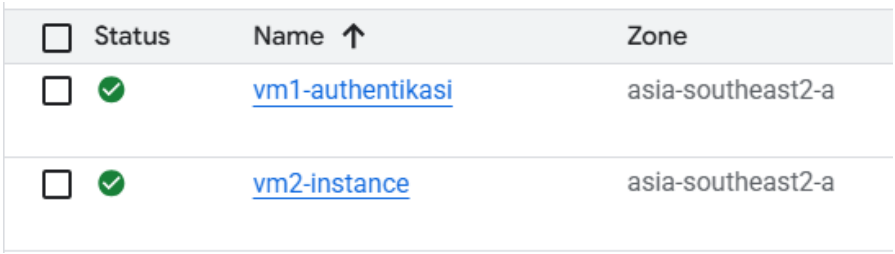
Endpoint	***Method***	**Skenario Pengujian**	**Status Code**	**Hasil Aktual**	Status
/register	*POST*	Registrasi pengguna baru dengan data yang valid.	201	Respon success dengan token JWT.	Berhasil
		Registrasi dengan email yang sudah terdaftar.	409	Respon fail dengan pesan error duplikasi.	Berhasil
		Registrasi dengan data tidak lengkap	400	Respon fail dengan pesan error validasi.	Berhasil
/login	*POST*	Login dengan kredensial yang valid.	200	Respon success dengan token JWT.	Berhasil
		Login dengan password atau email yang salah.	401	Respon success dengan token JWT.	Berhasil

Endpoint	***Method***	**Skenario Pengujian**	**Status Code**	**Hasil Aktual**	Status
/predict	*POST*	Mengirim gambar buah yang valid untuk diprediksi	200	Buah berhasil diprediksi	Berhasil
		Mengirim file dengan format yang tidak didukung (bukan gambar).	400	Respon fail dengan pesan error format file.	Berhasil
/fruit	*GET*	Meminta info nutrisi buah	200	Respon success dengan detail informasi nutrisi buah.	Berhasil
/consumption	*POST*	Menambah data konsumsi buah	201	Respon success dengan konfirmasi data telah disimpan.	Berhasil
/nutrisi-harian	*GET*	Meminta rekapitulasi data nutrisi harian pengguna	200	Respon success dengan data agregat nutrisi harian.	Berhasil
/nutrisi-bulanan	*GET*	Meminta rekapitulasi data nutrisi bulanan pengguna	200	Respon success dengan data agregat nutrisi bulanan.	Berhasil

Hasil pada Tabel 1 tidak hanya menunjukkan fungsionalitas, tetapi juga memvalidasi desain arsitektur *microservice* yang diterapkan. Keberhasilan pengujian pada *endpoint /register dan /login* memverifikasi kinerja Layanan Otentikasi pada *VM1*. Di sisi lain, respons sukses dari *endpoint /predict* dan */fruit* membuktikan bahwa Layanan Inti pada *VM2* mampu menjalankan logika bisnis dan pemrosesan *ML* sesuai rancangan.

3.3. Hasil Deployment pada Infrastruktur Cloud

RESTful API berhasil diterapkan pada dua mesin *virtual* di *Google Cloud Platform* yang memiliki arsitektur berbeda untuk layanan otentikasi *(VM1)* dan layanan inti *(VM2). VM1* menggunakan *PostgreSQL 16* untuk menangani data pengguna, sedangkan *VM2* menggunakan *Cloud Firestore* dan *Cloud Storage* untuk menangani konsumsi *log* dan penyimpanan gambar. Pola penyebaran ini bukan hanya sebuah implementasi, melainkan, validasi praktis dari desain pemisahan layanan mikro yang diusulkan. Dengan menempatkan layanan otentikasi *(VM1)* dan layanan inti *(VM2)* pada infrastruktur yang terpisah, penelitian ini secara nyata mendemonstrasikan bagaimana konsep *cloud-native* dapat diterapkan untuk mencapai modularitas dan efisiensi. Hal ini didukung oleh penelitian sebelumnya yang menguraikan bagaimana arsitektur layanan mikro meningkatkan modularitas, komunikasi antar layanan, dan sebagai konsekuensinya adalah skalabilitas *cloud-native* [12]. Selain itu, penelitian sebelumnya juga menentukan bahwa arsitektur layanan mikro secara langsung berdampak pada kinerja sistem melalui manajemen data dan komunikasi antar komponen [13].



Gambar 4. VM1 dan VM2 yang berjalan di asia-southeast2-a

3.4. Pengujian Kinerja: Stress Test

Pengujian kinerja dilakukan dengan pendekatan *stress test* menggunakan *K6*. Tujuan pengujian ini adalah untuk mengetahui sejauh mana sistem mampu menangani lonjakan beban permintaan (*request*) secara bersamaan dari sejumlah *Virtual Users (VU)*. Pada pengujian ini, *Virtual Users* dijalankan dalam kelipatan 500 mulai dari 500 VU hingga 4000 VU. Selama uji coba, *K6* mengirimkan sejumlah permintaan HTTP ke server sesuai dengan jumlah VU yang ditentukan. Tabel 2 menyajikan hasil pengujian berupa *error rate* yang dihitung menggunakan rumus berikut:

$$Error\ Rate\ = \frac{Jumlah\ Request\ Gagal}{Total\ Request} \times 100\%$$

Keterangan: Request gagal mencakup time*out*, kegagalan autentikasi, maupun *HTTP error* (kode 4xx dan 5xx).

Tabel 2. Hasil Stress Test

Beban (VU)	Error Rate	Status
100–1000	0%	Optimal
1500	54%	Down
2000–4000	57–100%	Down

Berdasarkan Tabel 2, terlihat bahwa pada beban 500 hingga 1000 VU sistem masih dapat beroperasi dengan baik tanpa menghasilkan kesalahan. Namun mulai dari 1500 VU, *error rate* meningkat signifikan hingga mencapai 54%, menunjukkan bahwa lebih dari setengah permintaan gagal diproses. Hal ini disebabkan oleh keterbatasan sumber daya CPU pada VM1 yang menjalankan layanan autentikasi sekaligus menangani sebagian besar permintaan. Pada rentang 2000–4000 VU, *error rate* terus meningkat hingga mencapai 100%. Kondisi ini menunjukkan bahwa server telah mencapai ambang batas dan tidak lagi mampu melayani permintaan tambahan, sehingga hampir seluruh permintaan berakhir dengan kegagalan. Hasil pengujian ini memberikan gambaran nyata mengenai kapasitas maksimum sistem saat ini, sekaligus menjadi dasar rekomendasi untuk pengembangan lebih lanjut, misalnya dengan menambahkan *load balancer* atau melakukan skala horizontal untuk meningkatkan ketersediaan layanan. penskalaan otomatis untuk mencegah kemacetan selama beban puncak [14]. Pernyataan ini juga didukung oleh analisis dari penelitian lain yang menemukan bahwa penskalaan otomatis, prediktif dan reaktif merupakan strategi utama untuk kinerja, efektivitas biaya, dan ketersediaan *API* berbasis *cloud* [15].

Gambar 5. Utilisasi CPU pada VM2 Selama Stress Test Proses Authentikasi Login

3.5. Keterkaitan dengan Penelitian

Tujuan pada penelitian ini untuk membuat dan menerapkan *RESTful API* berbasis *cloud* dengan dukungan untuk aplikasi seluler pelacak nutrisi berbasis klasifikasi buah. Berdasarkan hasil pengujian fungsionalitas, *deployment*, dan *stress testing*, dapat dinyatakan bahwa tujuan tersebut telah tercapai, meskipun sistem masih memiliki keterbatasan pada skalabilitas yang perlu diupayakan.

3.6. Keterbatasan dan Implikasi

Beberapa keterbatasan yang disadari dalam penelitian ini adalah:

1. Ketergantungan pada pengaturan *VM* secara manual, yang dapat menyulitkan penskalaan otomatis.
2. Tidak ada penyeimbangan beban atau penskalaan otomatis yang telah diatur.
3. Tidak ada ukuran kinerja dari model klasifikasi buah dalam kondisi beban sistem.

Di masa depan, kita dapat fokus pada pembangunan otomatisasi infrastruktur (misalnya menggunakan *Kubernetes* atau *Cloud Run*), mengintegrasikan model *ML* yang lebih optimal, dan pengujian kegunaan di sisi pengguna. **Data ini dapat menjadi referensi penting untuk pengembangan penelitian selanjutnya dalam bidang ini**

4. Kesimpulan

Penelitian ini berhasil mengembangkan dan menerapkan *RESTful API* untuk *Smart Nutrition App* pada infrastruktur *Google Cloud Platform (GCP)* dengan menggunakan *Node.js* dan *framework Hapi.js.* Seluruh fitur utama, termasuk otentikasi pengguna, prediksi jenis buah, pelacakan konsumsi buah, serta penyajian laporan nutrisi harian dan bulanan telah diimplementasikan dan berfungsi sesuai dengan skenario pengujian fungsional. Pengujian sistem menunjukkan bahwa *RESTful API* dapat menangani beban hingga 1000 *virtual users (VU)* dengan *error rate* 0%, namun mengalami penurunan performa secara signifikan pada beban di atas 1000 *VU,* dengan *error rate* mencapai 100% pada 4000 *VU*. Hal ini menunjukkan bahwa sistem memerlukan penguatan infrastruktur seperti *load balancing* dan mekanisme penskalaan otomatis agar dapat mendukung skenario penggunaan dengan jumlah pengguna tinggi secara lebih andal. ***Load balancing*** **Ini seperti seorang manajer di pintu tol yang mengarahkan mobil ke gerbang tol yang paling sepi. Dalam komputasi, penyeimbang beban adalah alat yang mendistribusikan lalu lintas jaringan yang masuk ke beberapa *server*.** Secara keseluruhan, solusi yang dikembangkan telah menjawab tujuan penelitian, yakni menyediakan layanan *backend* yang andal dan terintegrasi untuk aplikasi gizi berbasis *cloud*. Penelitian ini diharapkan dapat memberikan landasan bagi pengembanga lebih lanjut terkait konteks aplikasi berbasis *machine learning* yang skalabel pada platform *cloud computing*. Penelitian di masa depan dapat difokuskan pada pengujian performa dengan pendekatan *auto-scaling*

dinamis, integrasi dengan teknologi *container* seperti *Docker* dan *Kubernetes*, serta perluasan fitur untuk mendukung personalisasi gizi berbasis preferensi dan kondisi kesehatan pengguna.

Referensi

[1] X. Li, A. Yin, H. Y. Choi, V. Chan, M. Allman-Farinelli, and J. Chen, "Evaluating the Quality and Comparative Validity of Manual Food Logging and Artificial Intelligence-Enabled Food Image Recognition in Apps for Nutrition Care," Nutrients, vol. 16, no. 15, p. 2573, 2024. https://doi.org/10.3390/nu16152573

[2] M. Stoppa and A. Chiolerio, "Wearable Electronics and Smart Textiles: A Critical Review," Sensors, vol. 14, no. 7, pp. 11957–11992, 2014. https://doi.org/10.3390/s140711957

[3] Y. Koyama, M. Nishiyama, and K. Watanabe, "Smart Textile Using Hetero-Core Optical Fiber for Heartbeat and Respiration Monitoring," *IEEE Sensors Journal*, vol. 18, no. 15, pp. 6175–6180, 2018. https://doi.org/10.1109/JSEN.2018.2847333

[4] K. v Dalakleidi, M. Papadelli, I. Kapolos, and K. Papadimitriou, "Applying image-based food-recognition systems on dietary assessment: a systematic review," *Advances in Nutrition*, vol. 13, no. 6, pp. 2590–2619, 2022. https://doi.org/10.1093/advances/nmac078

[5] M. Mansouri, S. B. Chaouni, S. J. Andaloussi, and O. Ouchetto, "Deep learning for food image recognition and nutrition analysis towards chronic diseases monitoring: A systematic review," SN Computer Science, vol. 4, no. 5, p. 513, 2023. https://doi.org/10.1007/s42979-023-01972-1

[6] E. Costa e Silva, O. Oliveira, and B. Oliveira, "Enhancing Real-Time Analytics: Streaming Data Quality Metrics for Continuous Monitoring," in Proceedings of the 7th International Conference on Mathematics and Statistics (ICoMS), 2024. https://doi.org/10.1145/3686592.3686609

[7] S. Lee, J. Kim, and H. Park, "Cloud-based architecture for real-time food intake monitoring and nutritional analysis," *IEEE Access*, vol. 11, pp. 10123–10135, 2023. https://ieeexplore.ieee.org/document/10078027/

[8] Hasanuddin, H. Asgar, and B. Hartono, "Rancang Bangun REST API Aplikasi Weshare sebagai Upaya Mempermudah Pelayanan Donasi Kemanusiaan," JINTEKS (Jurnal Inform. Teknol. dan Sains), vol. 4, no. 1, pp. 8–14, 2022. https://doi.org/10.51401/jinteks.v4i1.1474

[9] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, "Survey on Serverless Computing," Journal of Cloud Computing: Advances, Systems and Applications, vol. 10, no. 1, p. 39, 2021. https://doi.org/10.1186/s13677-021-00253-7

[10] K. Pandit and K. Wanjale, "Implementation of Nutrition based REST APIs for Health Management Applications and Testing with Automation," International Journal of Scientific Research in Science, Engineering and Technology, vol. 7, no. 3, pp. 62–66, 2020. https://doi.org/10.32628/ijsrset207310

[11] A. Ehsan and et al., "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," Applied Sciences, vol. 12, no. 9, p. 4369, 2022. https://doi.org/10.3390/app12094369

[12] C. O. Oyekunle and et al., "A comprehensive review of leveraging cloud-native technologies for scalability and resilience in software development," International Journal of Science and Research Archive, vol. 11, no. 2, pp. 330–337, 2024. https://doi.org/10.30574/ijsra.2024.11.2.0432

[13] V. B. Ramu, "Performance impact of microservices architecture," The Review of Contemporary Scientific and Academic Studies, vol. 3, no. 6, 2023 https://doi.org/10.55454/rcsas.3.06.2023.010

[14] S. Raj and A. K. Raghav, "Elasticity in the Cloud Related to Database Autonomies and Scalability," i-manager's Journal on Cloud Computing, vol. 9, no. 1, pp. 26–31, 2022. [Online]. Available: https://www.researchgate.net/publication/363689885_Elasticity_in_the_cloud_related_to_database_autonomies_and_scalability

[15] P. Vemasani, S. M. Vuppalapati, S. Modi, and S. Ponnusamy, "Achieving Agility through Auto-Scaling: Strategies for Dynamic Resource Allocation in Cloud Computing," International Journal for Research in Applied Science and Engineering Technology, vol. 12, no. 4, pp. 3169–3177, 2024. https://doi.org/10.22214/ijraset.2024.60566