OPEN ACCESS

Volume 9 (3), July-September 2025, 944-955

E-ISSN:2580-1643

Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)

DOI: https://doi.org/10.35870/jtik.v9i3.3524

Pembuatan dan Deploy API di Google Cloud Platform Menggunakan Cloud Run Services

Oswaldo Da Conceicao [1*], Yasinta O. L. Rema [2], Budiman Baso [3], Guido Adolfus Suni [4]

[1,2,3,4] Program Studi Teknologi Informasi, Fakultas Pertanian, Sains dan Kesehatan, Universitas Timor, Kabupaten Timor Tengah Utara, Provinsi Nusa Tenggara Timur, Indonesia.*

a r t i c l e i n f o

Article history:
Received 20 December 2024
Received in revised form
10 January 2025
Accepted 15 February 2025
Available online July 2025.

Keywords:
Cloud Computing; API; Cloud Run.

Kata Kunci:
Cloud Computing; API; Cloud Run.

a b s t r a c t

This research discusses the development, testing, and deployment of a backend API using Google Cloud Platform (GCP), with Firestore as the database and Cloud Storage for storing user profile photos. The API was developed using Express.js, integrated with Firestore and Cloud Storage, and tested for functionality using Postman. Deployment is done to Cloud Run via creating a Dockerfile and building a Docker image. Test results showed that the API was able to handle 7,875 requests in five minutes, with an average of 25.64 requests per second and an average response time of 97 ms. All endpoints are stable, responsive, and error-free. This research focuses on the application of cloud technology to the development of sign language applications, and shows that the use of GCP services can produce efficient and scalable API solutions. These results are significant in supporting the development of cloud-based applications to meet user needs effectively, especially in mobile applications that require reliable backend services

a b s t r a k

Penelitian ini membahas pengembangan, pengujian, dan deployment API backend menggunakan Google Cloud Platform (GCP), dengan Firestore sebagai database dan Cloud Storage untuk penyimpanan foto profil pengguna. API dikembangkan menggunakan Express.js, diintegrasikan dengan Firestore dan Cloud Storage, serta diuji fungsionalitasnya menggunakan Postman. Deployment dilakukan ke Cloud Run melalui pembuatan Dockerfile dan pembangunan Docker image. Hasil pengujian menunjukkan bahwa API mampu menangani 7.875 permintaan dalam lima menit, dengan rata-rata 25,64 permintaan per detik dan waktu respons rata-rata 97 ms. Semua endpoint stabil, responsif, dan bebas error. Penelitian ini berfokus pada penerapan teknologi cloud untuk pengembangan aplikasi bahasa isyarat, dan menunjukkan bahwa penggunaan layanan GCP dapat menghasilkan solusi API yang efisien dan skalabel. Hasil ini signifikan dalam mendukung pengembangan aplikasi berbasis cloud untuk memenuhi kebutuhan pengguna secara efektif, khususnya dalam aplikasi mobile yang membutuhkan layanan backend handal.

Corresponding Author. Email: oswaldodc28@gmail.com [1].

ACM Computing Classification System (CCS)
EBSCOhost
Communication and Mass Media Complete (CMMC)

1. Pendahuluan

Bahasa isyarat adalah bahasa yang menggunakan gerakan tubuh sebagai media untuk berkomunikasi, dalam kelompok tuli, umumnya menggerakkan kedua tangan saat berkomunikasi (Muhtadi Ambarak dan Zakki Falani, 2023). Penggunaan bahasa isyarat memungkinkan mereka untuk berinteraksi dengan lingkungan sosialnya, memahami informasi, dan mengekspresikan diri dengan cara yang lebih efektif. Badan Pusat Statistik menyatakan, lapangan pekerjaan bagi disabilitas di rentang waktu antara 2016-2019 tidak mengalami pertumbuhan lebih dari 49%. Hal itu disebabkan oleh adanya perbedaan bahasa dan gaya komunikasi sehingga kebanyakan penyandang Tuna Rungu atau Tuli tidak mampu mandiri sejahtera (Azizah, Resmi, dan Alam, 2023). Pengembangan aplikasi yang menerjemahkan bahasa isyarat ke bahasa lisan dapat menjembatani kesenjangan komunikasi. Sehingga tidak hanya mempermudah interaksi antara individu tunarungu dan mereka yang tidak berbahasa isyarat, tetapi juga meningkatkan kesadaran dan pemahaman masyarakat akan pentingnya inklusi bagi komunitas tunarungu.

Penelitian sebelumnya yang dilakukan oleh Sholawati (2022), menerapkan teknologi *Convolutional Neural Network* (CNN) untuk mengembangkan aplikasi pengenalan bahasa isyarat secara real-time, yang bertujuan membantu siswa tunarungu dalam belajar bahasa isyarat. Namun, dalam penelitian ini, implementasi masih terbatas pada penggunaan perangkat keras lokal tanpa memanfaatkan teknologi *cloud computing*. Akibatnya, aplikasi tersebut memiliki keterbatasan dalam hal skalabilitas dan fleksibilitas. Selain itu, data yang dihasilkan dan digunakan oleh aplikasi hanya tersimpan secara lokal, yang berpotensi menimbulkan risiko kehilangan data dan mempersulit aksesibilitas dari berbagai perangkat. Teknologi *cloud* memiliki potensi untuk mengatasi kekurangan tersebut dengan layanan yang tersedia sehingga akses terhadap aplikasi menjadi mudah dan waktu pemrosesan data serta implementasi *machine learning* dapat dilakukan dengan lebih cepat (Fujiyanti, Suranegara, dan Ichsan, 2024). *Cloud computing* menawarkan model layanan yang fleksibel dan terukur, memungkinkan akses mudah ke sumber daya komputasi tanpa memerlukan kepemilikan infrastruktur fisik secara langsung (Muhamad Dyo Arganata, 2024). Layanan *Cloud Platform* memberikan berbagai layanan termasuk *storage*, *upload*, dan *download* (Gupta, Mittal, dan Mufti, 2021). Dengan *cloud computing*, program perangkat lunak yang digunakan tidak berada pada komputer kita, melainkan tersimpan pada server-server yang diakses melalui internet sehingga seluruh *cloud services* dan *storage* dapat diakses dari mana saja dan kapan saja selama terdapat koneksi internet (Ginanjar dan Setiyadi, 2020). Google *Cloud Platform* (GCP) merupakan salah satu penyedia layanan *cloud* yang banyak digunakan karena menyediakan berbagai layanan *cloud computing* (Irfansyah *et al.*, 2024). Salah satu layanan yang terintegrasi dalam GCP adalah *Cloud Run*, yang memungkinkan pengembang dapat mengunggah dan menjalankan aplikasi dengan *container* tanpa harus memikirkan infrastruktur yang mendukungnya (Kejora dan Susetyo, 2024). *Cloud Run* secara otomatis menambah dan menghapus instance *container* untuk menangani semua permintaan yang masuk, sebuah fitur yang dikenal sebagai *penskalaan otomatis*. *Cloud Run* juga memiliki kemampuan *penskalaan hingga nol*, yang berarti jika tidak ada permintaan yang masuk, semua instance *container*, termasuk instance terakhir, akan dihapus.

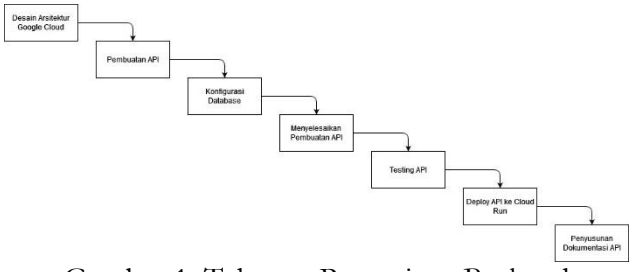
Perilaku ini adalah pengaturan default, dengan jumlah instance minimum adalah nol. Kemampuan ini tidak hanya membantu meningkatkan keandalan dan kinerja aplikasi, tetapi juga memungkinkan penghematan biaya dengan menghindari alokasi sumber daya yang tidak perlu (Abraham dan Yang, 2023). Penerapan teknologi yang tepat dapat membantu masyarakat berkomunikasi dengan penderita tunarungu dan tunawicara. Melalui program Bangkit Academy, penulis diajarkan untuk memberikan solusi di bidang teknologi dengan mengembangkan aplikasi pembelajaran bahasa isyarat sebagai tema capstone project tim penulis. Bangkit merupakan program pembelajaran yang dipimpin oleh Google dengan dukungan GoTo, Traveloka, dan Deeptech Foundation. Dengan dukungan Kampus Merdeka, Bangkit akan menawarkan tempat belajar untuk mahasiswa Indonesia untuk memastikan mereka relevan dengan kecakapan yang dibutuhkan oleh industri (Fajrul Falah, 2023). Dalam proyek ini, penulis bertugas di bagian *cloud computing*, berkontribusi untuk membuat backend API dan

melakukan deployment API tersebut bersama API model *machine learning* yang dikembangkan oleh tim *machine learning* ke *Cloud Run Services*. API tersebut nantinya akan diintegrasikan oleh tim mobile developer untuk membangun aplikasi yang memudahkan pengguna mempelajari bahasa isyarat secara efektif. Aplikasi ini diharapkan untuk menjadi solusi untuk masyarakat luas dalam mempelajari bahasa isyarat guna mempermudah komunikasi dengan penyandang tunarungu maupun tunawicara.

2. Metodologi Penelitian

Tahapan Penelitian

Pada penelitian ini, penulis menggunakan metode *waterfall* untuk pengembangan sistem karena urutan proses pengerjaan menggunakan metode *waterfall* ini lebih teratur dari satu tahap ke tahap yang selanjutnya (Badrul, 2021). Metode ini menggunakan pendekatan sistematis dan urut dimulai dari level kebutuhan sistem lalu menuju ke tahapan analisis, desain, *coding*, *testing/verification*, dan *maintenance* (Wahyu Rafsan Zani, Kartini, dan Mustika Rizki, 2024). Terdapat beberapa tahapan dalam pengembangan backend, yaitu desain arsitektur *cloud*, pembuatan API, konfigurasi database, menyelesaikan pembuatan API, *testing* API, *deploy* API ke *cloud run*, dan yang terakhir penyusunan dokumentasi API. Tahapan pengerjaan dapat dilihat pada gambar berikut ini.



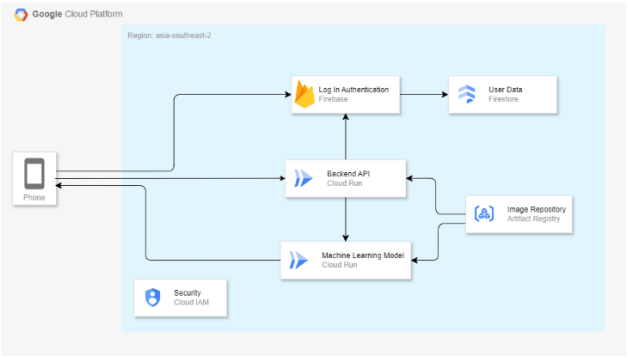
Gambar 1. Tahapan Pengerjaan Backend

3. Hasil dan Pembahasan

Hasil

Desain Arsitektur Google Cloud

Desain arsitektur memberikan gambaran bagaimana komponen aplikasi berinteraksi satu sama lain dalam lingkungan *Google Cloud Platform* (GCP) untuk mendukung fungsionalitas aplikasi secara keseluruhan. Proses pembuatan desain arsitektur dilakukan pada website *draw.io* dan dapat dilihat seperti pada gambar berikut:



Gambar 2. Desain Arsistektur Google Cloud

Alur interaksi dalam arsitektur ini dimulai saat pengguna membuka aplikasi di perangkat mereka dan melakukan login melalui *Firebase Authentication*. Setelah autentikasi berhasil, aplikasi mengakses data pengguna yang tersimpan di *Firestore* dan menampilkannya kepada pengguna. Pengguna kemudian dapat mengunggah atau mengubah foto profil mereka, yang disimpan dengan aman di bucket *Cloud Storage* melalui interaksi dengan Backend API yang berjalan di *Cloud Run*. Selain itu, pengguna juga dapat memanfaatkan fitur berbasis *machine learning*, di mana aplikasi backend mengirim permintaan ke model *machine learning* yang juga di-deploy di *Cloud Run*. Model ini memproses permintaan dan mengembalikan hasil analisis atau prediksi ke aplikasi. Setiap langkah dalam proses ini dilindungi oleh *Google Cloud Identity and Access Management* (IAM), yang memastikan bahwa hanya pengguna dan layanan yang memiliki izin yang sesuai yang dapat mengakses dan memodifikasi data atau sumber daya. Sementara itu, *Docker images* untuk aplikasi backend dan model *machine learning* disimpan di *Image Repository* (*Artifact Registry*) sebelum di-deploy ke *Cloud Run*, memastikan proses pengembangan dan *deployment* yang terorganisir dan efisien.

Pembuatan API

Application Programming Interface (API) merupakan antarmuka yang dibangun oleh pengembang sistem supaya sebagian atau keseluruhan fungsi sistem dapat diakses secara programatis (Kurniawan, Humaira, dan Rozi, 2020). Pada API berbasis website dibagi menjadi dua, yaitu *REST API* dan *SOAP API*. *REST API*

adalah API berbasis website yang menggunakan teknologi *REST* dan menggunakan format *JSON* (*JavaScript Object Notation*) (Wardhana, Arwani, dan Rahayudi, 2020). Formatnya adalah teks murni dan mudah diidentifikasi serta dapat diproses oleh mesin di seluruh jaringan dan platform (Ehsan *et al.*, 2022). Proses pembuatan API dimulai dengan menyiapkan repositori GitHub untuk mengelola kode sumber proyek secara terstruktur dan kolaboratif. Pertama, repositori baru dibuat di GitHub dengan nama *Project-Capstone*, memungkinkan tim untuk melakukan version control dan berkolaborasi dalam pengembangan kode. Setelah repositori dibuat, proyek diinisialisasi secara lokal dengan mengkloning repositori tersebut ke dalam lingkungan pengembangan di *Visual Studio Code*. Selanjutnya, dependensi yang diperlukan diinstal menggunakan *npm* (*Node Package Manager*). Ini mencakup instalasi *Express.js*, yang merupakan framework utama untuk membangun API, serta berbagai package lain yang dibutuhkan, seperti *body-parser* untuk memproses request body, *dotenv* untuk mengelola variabel lingkungan, dan *nodemon* untuk membantu proses *developing* sehingga tidak perlu menjalankan ulang server setiap ada perubahan pada baris kode. *ExpressJS* juga dikenal sebagai framework yang ringan karena tidak memerlukan banyak dependensi tambahan sehingga ideal untuk pengembangan aplikasi web dan API (Bachtiar *et al.*, 2024). Dependensi ini didefinisikan di dalam file *package.json*, yang memudahkan pengelolaan dan instalasi ulang jika diperlukan.

```
"dependencies": {
  "@google-cloud/firestore": "^7.8.0",
  "body-parser": "^1.20.2",
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "firebase-admin": "^12.1.1",
  "multer": "^1.4.5-lts.1",
  "nanoid": "^3.3.0",
  "nodemon": "^3.1.3"
}
```

Gambar 3. Dependencies yang digunakan

Setelah semua dependensi terinstal, tahap berikutnya adalah memastikan bahwa *server API* dapat berjalan di lingkungan lokal. Sebuah *file server.js* dibuat sebagai *entry point* aplikasi. Di dalam *file* ini, *server Express* diinisialisasi dengan mendefinisikan beberapa *middleware* dasar seperti *body-parser*, dan *endpoint* pertama juga dibuat, untuk menguji *respons server*. Akhirnya, *server* dijalankan menggunakan perintah *npm run start*, dan penulis memastikan bahwa *server* berfungsi dengan baik dengan mengakses *http://localhost:3000*.

```
PS D:\Project\Capstone\Cloud Computing> npm run start

> cloud-computing@1.0.0 start
> node server.js

Server is running on port 3000
```

Gambar 4. *Server* berjalan di *port* 3000

Jika server berjalan tanpa masalah, ini berarti bahwa infrastruktur dasar API telah siap untuk dikembangkan lebih lanjut, dan kode dapat didorong (*push*) kembali ke GitHub untuk disimpan dan dibagikan dengan tim lainnya.

Konfigurasi Database

Konfigurasi database untuk aplikasi dimulai dengan pembuatan *Service Account Key* di *Google Cloud Platform* (GCP). *Service Account Key* ini penting karena memungkinkan aplikasi backend mengakses layanan Google seperti *Firestore* dan *Cloud Storage* secara aman. Proses ini dimulai dengan membuat *Service Account* baru di *Google Cloud Console*, di mana peran yang relevan seperti "Storage Object Admin" diberikan. Setelah *Service Account* dibuat, kunci otentikasi diunduh dalam format *JSON*. File ini berisi kredensial yang dibutuhkan aplikasi untuk berkomunikasi dengan *Firestore* dan *Cloud Storage*. Langkah berikutnya adalah menambahkan *Service Account Key* ke proyek backend. File *JSON* tersebut disimpan di direktori proyek dan diimpor melalui variabel lingkungan menggunakan package *dotenv*. Ini memastikan bahwa saat aplikasi backend dijalankan, kredensial tersebut sudah tersedia untuk digunakan.

Pengaturan ini sangat penting karena memungkinkan aplikasi untuk mengakses database dan *storage* tanpa harus bergantung pada login pengguna atau interaksi manual lainnya. Setelah kredensial terkonfigurasi, aplikasi dihubungkan ke *Firestore*. Hal ini dilakukan dengan menginisialisasi *Firebase Admin SDK* di dalam kode backend pada file *db.js*. Dengan inisialisasi ini, aplikasi siap untuk melakukan operasi database seperti pembuatan, pembacaan, pembaruan, dan penghapusan (CRUD) data di *Firestore*. Pada saat yang sama, aplikasi juga dikonfigurasi untuk terhubung ke

Cloud Storage, yang digunakan untuk menyimpan dan mengelola file gambar profil pengguna. *Google Cloud Storage* menyediakan objek penyimpanan yang cocok untuk berbagai jenis data, dengan opsi untuk redundansi regional atau multi-regional untuk memastikan ketahanan data (Borra, 2024). Kode konfigurasi dengan database dapat dilihat pada Gambar 5.

```
const admin = require('firebase-admin');
const serviceAccount = require('./ServiceAccountKeyGencaraApp.json');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  storageBucket: "gencarabucket"
});

const db = admin.firestore();
const bucket = admin.storage().bucket();

module.exports = { db, bucket };
```

Gambar 5. Kode Konfigurasi *Database*

Menyelesaikan Pembuatan API

Pada tahap penyelesaian pembuatan API, fokus utama adalah menyempurnakan seluruh endpoint yang diperlukan untuk aplikasi. Setelah konfigurasi dengan *Firestore* dan *Cloud Storage* telah selesai, semua endpoint API diimplementasikan untuk menangani operasi CRUD (*Create, Read, Update, Delete*) pada data pengguna. Setiap endpoint dirancang untuk mengelola interaksi dengan database dan *storage* secara efisien, memastikan bahwa data pengguna dapat diakses dan dimodifikasi sesuai dengan kebutuhan.

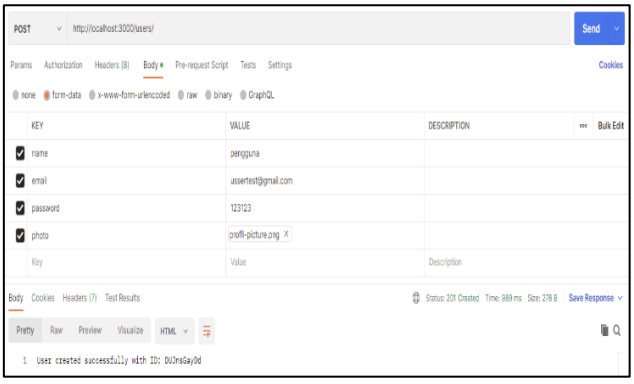
```
const express = require('express');
const { createUser, getUser, putUser, deleteUser } = require('./handler');
const router = express.Router();

router.post('/', createUser);
router.get('/:uid', getUser);
router.put('/:uid', putUser);
router.delete('/:uid', deleteUser);

module.exports = router;
```

Gambar 6. *Endpoint* dan *Request Method*

Dari gambar di atas dapat dilihat bahwa endpoint yang dibutuhkan untuk aplikasi telah diimplementasikan. Misalnya, endpoint untuk membuat pengguna baru, mengambil detail pengguna, memperbarui informasi pengguna, dan menghapus akun pengguna. Selain itu, integrasi dengan *Firestore* memastikan bahwa data disimpan dan diambil dengan cara yang terstruktur dan optimal. Setiap operasi CRUD juga diuji secara menyeluruh untuk memastikan bahwa data dapat diakses dan dimodifikasi tanpa masalah.

Testing API

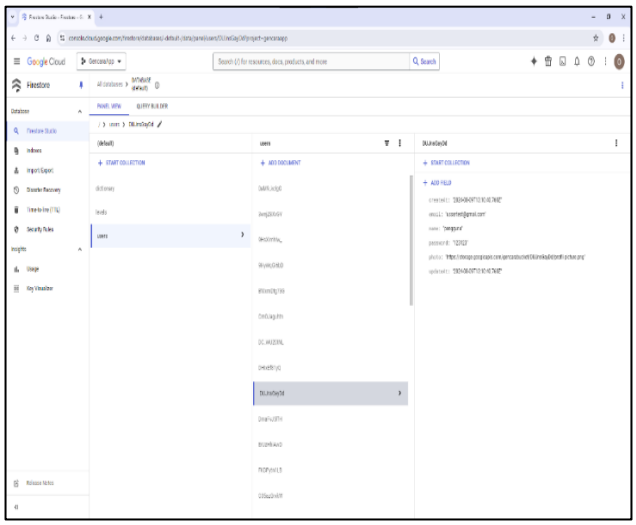
Tahap pengujian API bertujuan memastikan bahwa API yang telah selesai dibuat berfungsi dengan baik dan siap digunakan dalam lingkungan produksi. Pada tahap ini, API dijalankan secara lokal di lingkungan pengembangan, dan pengujian dilakukan menggunakan aplikasi *Postman*. *Postman* adalah sebuah software yang memuat fungsi lengkap pengembangan sistem dalam mengirimkan dan menerima respons 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 (Firdaus dan Afwani, 2024). Proses pengujian dilakukan dengan cara pengujian fungsional. Jenis data yang digunakan dalam pengujian fungsional adalah data dummy yang dirancang untuk mencerminkan skenario nyata, seperti data pengguna dengan berbagai atribut (nama, email, password, dan foto profil). Proses pengujian dimulai dengan memuat endpoint API di *Postman*. Setiap endpoint diuji untuk memastikan bahwa fungsionalitasnya sesuai dengan yang diharapkan. Penulis mengirimkan berbagai jenis permintaan HTTP (seperti *GET*, *POST*, *PUT*, *DELETE*) ke API, sesuai dengan operasi CRUD yang telah diimplementasikan. Sebagai contoh, untuk endpoint yang berfungsi membuat pengguna baru, pengembang akan mengirimkan permintaan *POST* dengan data pengguna yang sesuai dan memeriksa apakah API menanggapi dengan benar dengan membuat entri baru di *Firestore*.



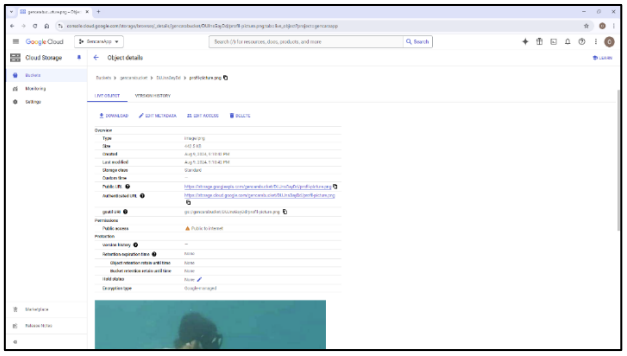
Gambar 7. Pengujian API dengan Method POST

Gambar di atas menunjukkan proses pengujian API dengan menggunakan metode *POST* di *Postman*. Berikut penjelasannya:

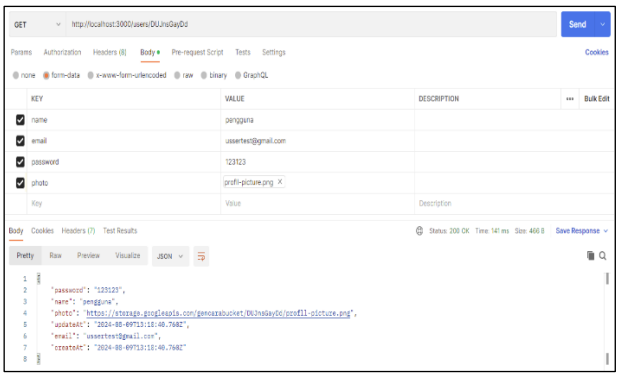
1) Endpoint:
   URL yang digunakan adalah http://localhost:3000/users/, yang menunjukkan bahwa penulis sedang melakukan permintaan *POST* ke endpoint /users pada server lokal yang berjalan di port 3000.
2) Body Request
   Penulis mengirimkan data dengan menggunakan format *form-data*, yang biasanya digunakan untuk mengirim data multipart, termasuk file. Ada empat kunci yang dikirimkan dalam permintaan ini:
   a) *name*: diisi dengan nilai "pengguna".
   b) *email*: diisi dengan nilai "ussertest@gmail.com".
   c) *password*: diisi dengan nilai "123123".
   d) *photo*: diisi dengan file bernama "profil-picture.png".
3) Response
   Status respons adalah '201 Created' yang menunjukkan bahwa permintaan *POST* telah berhasil dibuat di server. *Response body* menunjukkan bahwa pengguna telah berhasil dibuat dengan ID 'DUJnsGayDd', yang dapat digunakan untuk mengidentifikasi pengguna tersebut di dalam sistem. Dari *response* yang diterima dari server dapat diambil kesimpulan bahwa API sudah berjalan dengan normal dan tanpa kendala. Jika diperiksa, data pengguna juga sudah terdapat dalam *Firestore* dan *Bucket Cloud Storage*, seperti pada Gambar 7.



Gambar 8. Data *User* pada *Firestore*



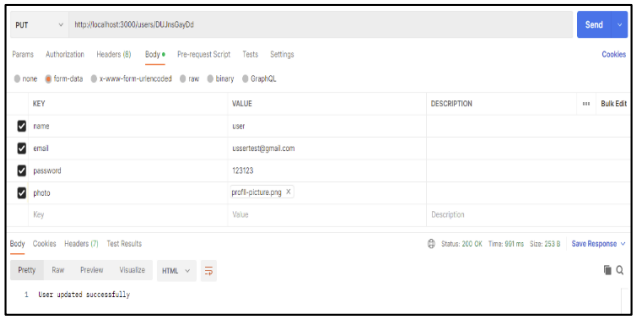
Gambar 9. Data *User* pada *Bucket Cloud Storage*



Gambar 10. Pengujian *API* dengan *Method GET*

Gambar di atas menunjukkan proses pengujian API dengan menggunakan metode *GET* di *Postman*. Berikut penjelasannya:

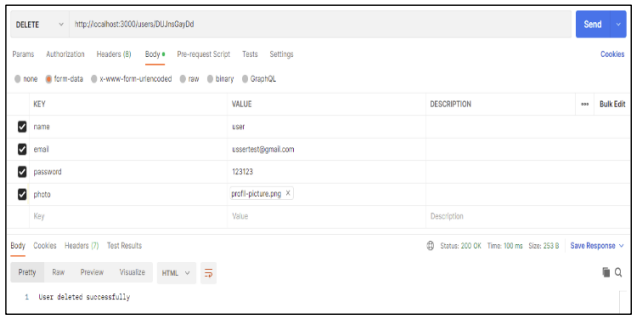
1) Endpoint
   Penulis mengakses endpoint yang sama dengan yang digunakan saat *POST*, yaitu http://localhost:3000/users/. Namun, untuk mendapat data spesifik dari pengguna yang baru dibuat, penulis menambahkan ID pengguna yang diterima dalam respons *POST* sebelumnya sebagai parameter URL sehingga menjadi http://localhost:3000/users/DUJnsGayDd.
2) Body Request
   Tidak ada body yang perlu dikirimkan dengan metode *GET*. Cukup panggil endpoint tersebut.
3) Response
   Status respons adalah '200 OK' yang menunjukkan bahwa permintaan *GET* telah berhasil di server. Server akan mengirimkan data pengguna dalam format *JSON*, yang berisi informasi mengenai data pengguna.



Gambar 11. Pengujian API dengan Method PUT

Gambar di atas menunjukkan proses pengujian API dengan menggunakan metode *PUT* di *Postman*. Berikut penjelasannya:

1) Endpoint
   Penulis akan menggunakan endpoint yang sama seperti saat mengakses data pengguna dengan metode *GET*, yaitu http://localhost:3000/users/DUJnsGayDd. Di sini, DUJnsGayDd adalah ID pengguna yang ingin penulis perbarui.
2) Body Request
   Dalam metode *PUT*, penulis akan mengirimkan *body* yang berisi data yang ingin diperbarui. Penulis ingin mengganti *name* dari "pengguna" menjadi "user".
3) Response
   Status respons adalah '200 OK' dengan pesan *response* "user updated successfully" yang menunjukkan bahwa data pengguna telah berhasil diperbarui di server.



Gambar 12. Pengujian *API* dengan *Method DELETE*

Gambar di atas menunjukkan proses pengujian API dengan menggunakan metode *DELETE* di *Postman*. Berikut penjelasannya:

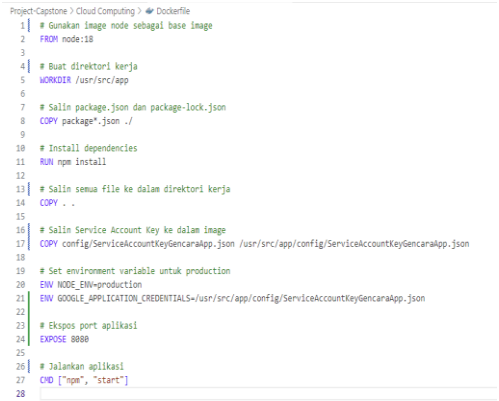
1) Endpoint
   Penulis akan menggunakan endpoint yang sama seperti saat melakukan pengujian *GET* dan *PUT*, yaitu http://localhost:3000/users/DUJnsGayDd. Di sini, "DUJnsGayDd" adalah ID pengguna yang ingin penulis hapus.
2) Body Request
   Sama seperti metode *GET*, tidak perlu mengirimkan *body request* saat menggunakan metode *DELETE*. Cukup kirim permintaan ke endpoint dengan ID yang ingin dihapus.
3) Response
   Status respons adalah '200 OK' dengan pesan *response* "user deleted successfully" yang menunjukkan bahwa data pengguna telah berhasil dihapus.

Deploy API ke Cloud Run

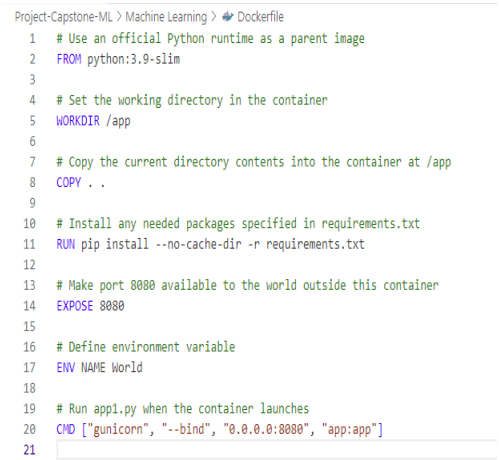
Proses *deploy* aplikasi ke *Google Cloud Run* adalah langkah penting yang membawa API dari lingkungan pengembangan lokal ke lingkungan produksi yang aman dan dapat diakses oleh pengguna. *Cloud Run* merupakan sebuah teknologi berbasis *container* yang memungkinkan penggunanya untuk memilih apakah aplikasi *container* akan dijalankan dengan menggunakan *Cloud Run* yang dikelola oleh Google atau dijalankan di atas *Google Kubernetes Engine* (GKE) dengan menggunakan *Cloud Run on GKE* (Hafizin, 2024). Berikut ini adalah tahapan rinci dari proses tersebut.

1) *Cloning Repository* di *Cloud Shell*
   Langkah pertama adalah meng-kloning repositori GitHub yang berisi kode sumber API ke *Cloud Shell*, sebuah terminal berbasis web yang disediakan oleh *Google Cloud*. Ini memungkinkan akses langsung ke *resource* GCP.
2) Membuat *Image Repository* di *Artifact Registry*
   Artifact Registry adalah tempat penyimpanan *Docker images* sebelum di-deploy ke *Cloud Run*. Untuk membuat *Image Repository* di *Artifact Registry*, navigasi ke *Google Cloud Console* dan buat repositori baru. Pilih "Docker" dan penulis memberi nama "gencara-app" untuk backend dan "ml-gencara" untuk *machine learning* API. Langkah ini menyiapkan lokasi di mana *Docker images* akan disimpan.
3) Membuat *Dockerfile*
   Di dalam direktori proyek yang sudah dikloning, sebuah *Dockerfile* dibuat. *Dockerfile* ini adalah skrip yang menjelaskan bagaimana aplikasi akan dibangun ke dalam *image Docker*. File ini biasanya berisi instruksi untuk menarik *image* dasar

(misalnya, *Node.js* untuk aplikasi *Express.js*), menginstal dependensi, dan mengatur *environment* yang dibutuhkan untuk menjalankan aplikasi. Isi *Dockerfile* untuk kedua API dapat dilihat pada Gambar 13 dan Gambar 14.



Gambar 13. *Dockerfile untuk Backend*



Gambar 14. *Dockerfile* untuk *Machine Learning*

Build Docker Image

Setelah *Dockerfile* dibuat, langkah berikutnya adalah membangun *Docker image* dari aplikasi menggunakan perintah berikut:

```
(gencaraapp)$ docker build -t gcr.io/gencaraapp/gencara-app:1.0 .
```

Gambar 15. *Command Build Docker Image Backend*

Untuk *API machine learning,* menggunakan perintah berikut:

```
(gencaraapp)$ docker build -t gcr.io/gencaraapp/ml-gencara:1.0 .
```

Gambar 16. *Command Build Docker Image Machine Learning*

Perintah ini membuat *Docker image* lokal dari aplikasi dan menandainya (*tag*) dengan nama yang sesuai (*gencara-app:1.0* dan *ml-gencara:1.0*).

Push Docker Image ke Artifact Registry

Setelah *Docker image* dibangun, langkah selanjutnya adalah mendorong (*push*) *image* tersebut ke *Artifact Registry* dengan menggunakan perintah:

```
(gencaraapp)$ docker push gcr.io/gencaraapp/gencara-app:1.0
```

Gambar 17. *Command Push Docker Image Backend*

Untuk *API machine learning* menggunakan perintah berikut:

```
(gencaraapp)$ docker push gcr.io/gencaraapp/ml-gencara:1.0
```

Gambar 18. *Command Push Docker Image Machine Learning*

Dengan perintah ini, *Docker image* akan diunggah ke *Artifact Registry* yang telah dibuat sebelumnya, sehingga siap untuk di-*deploy ke Cloud Run.*

Deploy ke Cloud Run

Langkah terakhir adalah men-*depoloy* aplikasi ke *Google Cloud Run*. Ini dilakukan dengan perintah "*gcloud*" berikut:

```
c472d4ky1158@cloudshell:~ (gencaraapp)$ gcloud run deploy gencara-backend --image gcr.io/
gencaraapp/gencara-app:1.0 --region asia-southeast2 --project gencaraapp
```

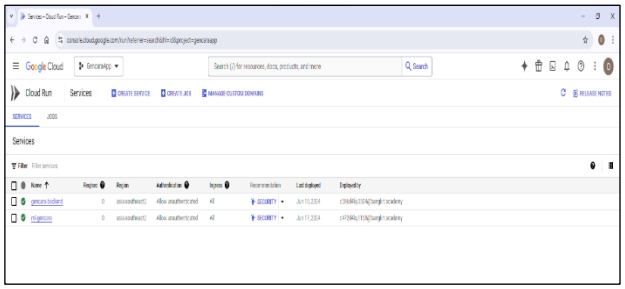
Gambar 19. *Command Deploy Backend*

Untuk *API machine learning* menggunkana perintah berikut:

```
c472d4ky1158@cloudshell:~ (gencaraapp)$ gcloud run deploy gencara-ml --image gcr.io/genca
raapp/gencara-ml:1.0 --region asia-southeast2 --project gencaraapp --allow-unauthenticate
d
```

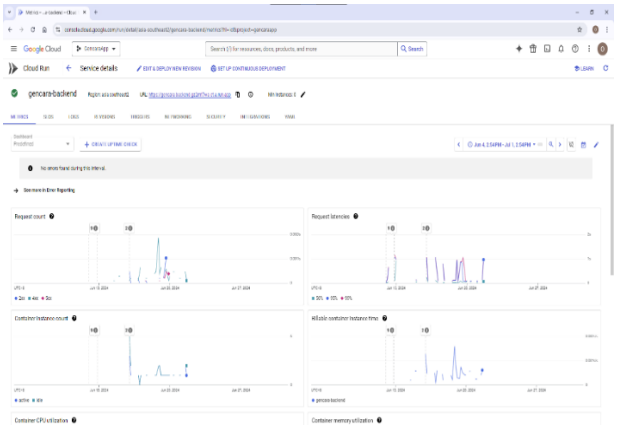
Gambar 20. *Command Deploy Backend*

Perintah ini melakukan beberapa hal: pertama, ia memberitahu *Cloud Run* untuk membuat layanan baru bernama gencara-app dan ml-gencara, yang akan menggunakan *Docker image* yang telah penulis *push* ke *Artifact Registry*. Perintah --allow-unauthenticated pada *deploy machine learning API* memungkinkan akses publik ke layanan ini.

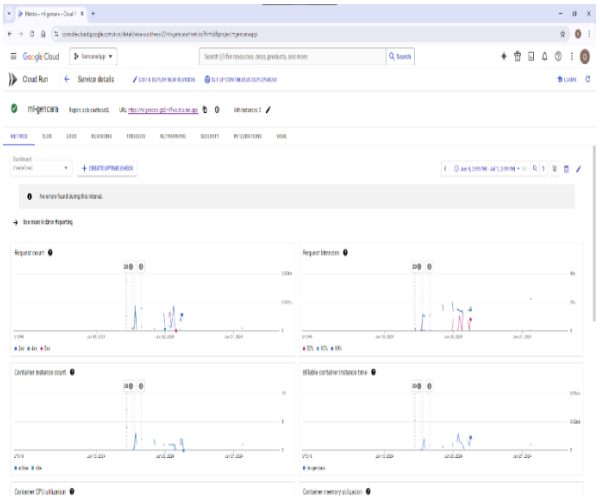


Gambar 21. *Cloud Run Services*

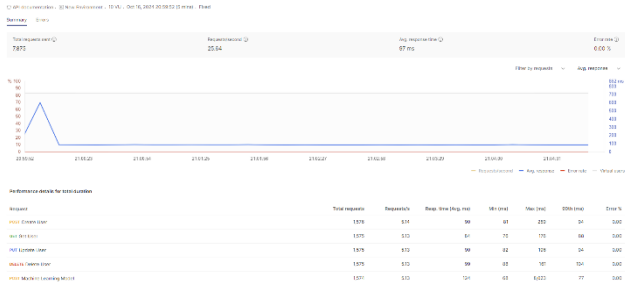
Setelah perintah ini dieksekusi, *Cloud Run* akan mengatur infrastruktur yang dibutuhkan, seperti *load balancing* dan *autoscaling*, sehingga aplikasi dapat berjalan di *cloud* seperti pada Gambar 21 dengan performa optimal dan dapat diakses secara publik melalui *URL* yang dihasilkan oleh *Cloud Run*. Layanan *Cloud Run* yang sudah berjalan dapat dilihat pada Gambar 22 dan Gambar 23.



Gambar 22. *Cloud Run Backend*



Gambar 23. *Cloud Run Machine Learning*



Gambar 24. *Load Test* Hasil *Deploy API*

Hasil *load test* menunjukkan performa API yang cukup baik selama pengujian lima menit dengan total 7.875 *request* atau sekitar 25,64 *request* per detik. Rata-rata waktu respons API adalah 97 ms, yang menandakan bahwa API merespons dengan cepat, terutama karena *error rate* tercatat 0,00%, artinya tidak ada *request* yang gagal. Grafik menunjukkan peningkatan *request* di awal pengujian, tetapi segera stabil setelah itu. Setiap endpoint yang diuji, termasuk *POST*, *GET*, *PUT*, dan *DELETE*, memiliki waktu respons rata-rata di bawah 100 ms, dengan variasi kecil antara 84 ms hingga 99 ms. Namun, endpoint *POST Machine Learning Model* menunjukkan waktu respons yang sedikit lebih tinggi, yaitu rata-rata 124 ms, dan terdapat *outlier* dengan waktu respons maksimum hingga 8.023 ms, dikarenakan proses ini lebih berat. Meski demikian, tidak ada *error* pada endpoint tersebut. Secara keseluruhan, hasil *load test* ini menunjukkan bahwa API mampu menangani beban dengan stabil dan responsif. Hasil pengetesan API yang sudah dideploy di *Cloud Run* dapat dilihat pada Gambar 24.

Penyusunan Dokumentasi API

Dalam pembuatan dokumentasi API backend yang sudah selesai dikembangkan dan dideploy, penulis menggunakan *Postman* sebagai alat utama. *Postman* tidak hanya memfasilitasi pengujian API, tetapi juga sangat efektif dalam menghasilkan dokumentasi yang nantinya akan digunakan oleh tim *mobile development* dalam pengembangan aplikasi. Proses dimulai dengan mengorganisasi semua endpoint API ke dalam satu *Collection* di *Postman*. Penulis mengelompokkan endpoint berdasarkan fungsionalitasnya, seperti endpoint untuk operasi pengguna (registrasi, login, dan pengelolaan profil) dan endpoint untuk *machine learning*. Setiap endpoint di *Collection* ini disertai dengan deskripsi rinci yang menjelaskan tujuan, metode *HTTP* yang digunakan, serta parameter yang diperlukan dalam *request*.

Pembahasan

Berdasarkan hasil pengujian *load test*, API yang dikembangkan menunjukkan performa yang baik dalam menangani beban. Dalam lima menit pengujian, API mampu menangani total 7.875 *request*, dengan rata-rata sekitar 25,64 *request* per detik. Rata-rata waktu respons sebesar 97 ms menunjukkan bahwa API dapat merespons permintaan dengan cepat, yang sangat penting dalam aplikasi mobile yang memerlukan kecepatan dan responsifitas. Hasil ini sesuai dengan temuan Fujiyanti, Suranegara, dan Ichsan (2024), yang mencatatkan bahwa *cloud-based* services seperti *Cloud Run* dapat memberikan performa yang optimal dalam menangani *request* dalam jumlah besar, dengan waktu respons yang sangat cepat. Kelebihan lain dari pengujian ini adalah tercatatnya *error rate* 0,00%, yang menandakan bahwa tidak ada permintaan yang gagal selama pengujian. Hal ini mencerminkan stabilitas dan keandalan API dalam menghadapi *traffic* yang tinggi, sebagaimana disarankan oleh Kejora dan Susetyo (2024) dalam studi mereka tentang penerapan *Cloud Run* untuk pengembangan aplikasi web. Hasil ini juga mengindikasikan bahwa API mampu mengelola *traffic* yang lebih besar secara efisien, yang penting untuk memastikan pengalaman pengguna yang baik dan aplikasi yang dapat diandalkan dalam skala besar.

Namun, hasil pengujian juga menunjukkan adanya sedikit lonjakan pada waktu respons endpoint *POST Machine Learning Model*, dengan waktu respons rata-rata 124 ms, yang lebih tinggi dibandingkan dengan endpoint lainnya. Ini menunjukkan bahwa proses *machine learning* yang dijalankan di backend memerlukan waktu komputasi lebih lama, yang juga sejalan dengan temuan oleh Abraham dan Yang (2023), yang menunjukkan bahwa model berbasis *machine learning* dapat mempengaruhi performa API, terutama ketika membutuhkan pemrosesan data yang lebih berat. Walaupun demikian, tidak ada *error* pada endpoint tersebut, menunjukkan bahwa API dapat tetap stabil meskipun dengan peningkatan kompleksitas proses. Proses pengujian yang dilakukan menggunakan *Postman* untuk setiap metode HTTP (*POST*, *GET*, *PUT*, *DELETE*) sangat penting dalam memastikan bahwa semua endpoint berfungsi sesuai dengan yang diharapkan. Menurut Ehsan *et al.* (2022), pengujian API menggunakan metodologi yang tepat sangat penting untuk memastikan bahwa fungsionalitas API berjalan dengan baik di berbagai skenario. Pengujian fungsional yang dilakukan dalam penelitian ini menggunakan data dummy yang mencakup nama, email, password, dan foto profil, memberikan gambaran yang jelas tentang cara kerja API dalam kondisi yang lebih realistis. Selain itu, dokumentasi API yang disusun menggunakan *Postman* memberikan keuntungan besar dalam pengembangan aplikasi, karena mendokumentasikan setiap endpoint secara terperinci, termasuk tujuan, metode HTTP yang digunakan, serta parameter yang diperlukan dalam setiap *request*. Hal ini sejalan dengan penelitian oleh Kurniawan *et al.* (2020), yang menekankan pentingnya dokumentasi API yang baik untuk memastikan bahwa pengembang dapat dengan mudah memahami dan mengintegrasikan API dalam aplikasi mereka. Dengan dokumentasi yang terorganisir, tim pengembang mobile dapat dengan mudah mengakses informasi yang diperlukan untuk mengintegrasikan API dalam aplikasi mereka tanpa kebingungannya. Hasil dari *load test* dan dokumentasi API yang dikembangkan menunjukkan bahwa API yang dibangun mampu menangani beban dengan baik dan responsif, serta dapat diintegrasikan dengan mudah ke dalam aplikasi mobile untuk mempermudah interaksi dengan pengguna.

4. Kesimpulan

API berhasil dirancang, dikembangkan, dan di-*deploy* ke *Cloud Run*, dimulai dari perancangan arsitektur *backend* hingga integrasi dengan *Firestore* dan *Google Cloud Storage*. Proses pengujian memastikan *API* stabil, dengan waktu respons rata-rata 97 ms dan kemampuan menangani hingga 7.875 permintaan dalam lima menit. Penelitian ini menunjukkan bahwa layanan *Google Cloud Platform* dapat mendukung pengembangan *API* yang efisien, skalabel, dan mudah diimplementasikan, sehingga menjadi solusi bagi pengembangan aplikasi berbasis *cloud*. Kontribusi utama penelitian ini adalah menunjukkan keberhasilan penerapan teknologi *cloud* dalam pengembangan *API*, yang dapat menjadi acuan untuk proyek serupa. Penelitian lanjutan disarankan untuk mengintegrasikan *CI/CD pipelines* untuk meningkatkan efisiensi dan mengurangi risiko human *error* dalam proses pengembangan dan *deployment*.

5. Daftar Pustaka

Abraham, A., & Yang, J. (2023). Analyzing the system features, usability, and performance of a containerized application on serverless cloud computing systems. *Research Square*. https://doi.org/10.21203/rs.3.rs-3167840/v1.

Azizah, E. N., Resmi, M. G., & Alam, S. (2023). Penerapan Metode Design Thinking Pada Perancangan User Interface Aplikasi Mobile Pengenalan Bahasa Isyarat Indonesia (Bisindo). *Jurnal Mnemonic*, *6*(1), 71-76. https://doi.org/10.36040/mnemonic.v6i1.5711.

Bachtiar, D. H., et al. (2024). Perancangan back-end API pada aplikasi mobile Fruityfit menggunakan framework Express JS. *Mars: Jurnal Teknik Mesin, Industri, Elektro Dan Ilmu Komputer*, 2(3), 107–117. https://doi.org/10.61132/mars.v2i3.138.

Badrul, M. (2021). Penerapan metode waterfall Untuk Perancangan sistem informasi inventory Pada Toko Keramik bintang terang. *PROSISKO J. Pengemb. Ris. dan Obs. Sist. Komput*, *8*(2), 57-52.

Borra, P. (2024). A survey of Google Cloud Platform (GCP): Features, services, and applications. *International Journal of Advanced Research in Science, Communication and Technology*, 191–199. https://doi.org/10.48175/ijarsct-18922.

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences (Switzerland)*, 12(9), 1–22. https://doi.org/10.3390/app12094369.

Falah, R. F., & Komarudin, M. (2023). Perancangan Microservice Berbasis REST API pada Google Cloud Platform Menggunakan Nodejs dan Python. *Jurnal Informatika dan Teknik Elektro Terapan*, *11*(3s1).

Falani, A. Z. (2023). Pengembangan Aplikasi Bahasa Isyarat Indonesia Berbasis Realtime Video Menggunakan Model Machine Learning. *JIKA (Jurnal Informatika)*, *7*(1), 89-96. http://dx.doi.org/10.31000/jika.v7i1.7277

Firdaus, M., & Afwani, R. (2024). Pengembangan Restful API untuk Aplikasi Klasifikasi Jenis Tanah Berbasis Mobile Pada Google Cloud. *Jurnal Teknologi Informasi, Komputer, dan Aplikasinya (JTIKA)*, *6*(1), 275-287. https://doi.org/10.29303/jtika.v6i1.335.

Fujiyanti, V., Suranegara, G. M., & Ichsan, I. N. (2024). Comparative analysis of server-based and serverless service performance on Google Cloud Platform (GCP) (Case study: Machine learning model deployment). *Journal of Information Systems and Informatics*, 6(2), 1172–1194. https://doi.org/10.51519/journalisi.v6i2.773.

Ginanjar, H. P., & Setiyadi, A. (2020). Penerapan Teknologi Cloud Computing Pada Katalog Produk Di Balatkop Jawa Barat. *Komputa: Jurnal Ilmiah Komputer Dan Informatika*, *9*(1), 25-33. https://doi.org/10.34010/komputa.v9i1.3722.

Gupta, B., Mittal, P., & Mufti, T. (2021). A review on Amazon Web Service (AWS), Microsoft Azure & Google Cloud Platform (GCP) services. https://doi.org/10.4108/eai.27-2-2020.2303255

Hafizin, M. (2024). Perancangan dan implementasi API pada aplikasi deteksi mata katarak menggunakan Google Cloud Run. *Mars: Jurnal Teknik Mesin, Industri, Elektro Dan Ilmu Komputer*, 2(4), 172–180.

Irfansyah, M. B., Arief, S. N., & Nugraha, B. S. D. (2024). DESAIN DAN ARSITEKTUR SERVERLESS CLOUD COMPUTING PADA APLIKASI PENGHITUNG KALORI MAKANAN BERBASIS MOBILE MENGGUNAKAN LAYANAN GOOGLE CLOUD PLATFORM. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *8*(4), 6090-6097. https://doi.org/10.36040/jati.v8i4.10180.

Kejora, C. B., & Susetyo, Y. A. (2024). Analisis Perbandingan Compute Engine dan Cloud Run sebagai lingkungan Pengembangan Aplikasi Web di Google Cloud Platform. *Jurasik (Jurnal Riset Sistem Informasi dan Teknik Informatika)*, *9*(1), 491-503. http://dx.doi.org/10.30645/jurasik.v9i1.756.

Kurniawan, I., Humaira, & Rozi, F. (2020). REST API menggunakan NodeJS pada aplikasi transaksi jasa elektronik berbasis Android. *JITSI: Jurnal Ilmiah Teknologi Sistem Informasi*, 1(4), 127–132. https://doi.org/10.30630/jitsi.1.4.18.

Prasetya, A., Arganata, M. D., & Sutabri, T. (2024). Analisis Perbandingan Antara Teknologi Cloud Computing Dan Infrastruktur Komputer Tradisional Dalam Konteks Bisnis. *Scientifica: Jurnal Ilmiah Sains Dan Teknologi*, *2*(7), 143-147.

Sholawati, M., Auliasari, K., & Ariwibisono, F. X. (2022). Pengembangan aplikasi pengenalan bahasa isyarat abjad Sibi menggunakan metode Convolutional Neural Network (CNN). *JATI (Jurnal Mahasiswa Teknik Informatika)*, 6(1), 134–144. https://doi.org/10.36040/jati.v6i1.4507.

Wardhana, W. G., Arwani, I., & Rahayudi, B. (2020). Implementasi Teknologi Restful Web Service Dalam Pengembangan Sistem Informasi Perekaman Prestasi Mahasiswa Berbasis Website (Studi Kasus: Fakultas Teknologi Pertanian Universitas Brawijaya). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *4*(2), 680-689.

Zani, A. W. R., Kartini, K., & Rizki, A. M. (2024). Rancang bangun aplikasi mobile bank sampah menggunakan framework React Native dan REST API. *Uranus: Jurnal Ilmiah Teknik Elektro, Sains dan Informatika*, 2(3), 112–124. https://doi.org/10.61132/uranus.v2i3.254.