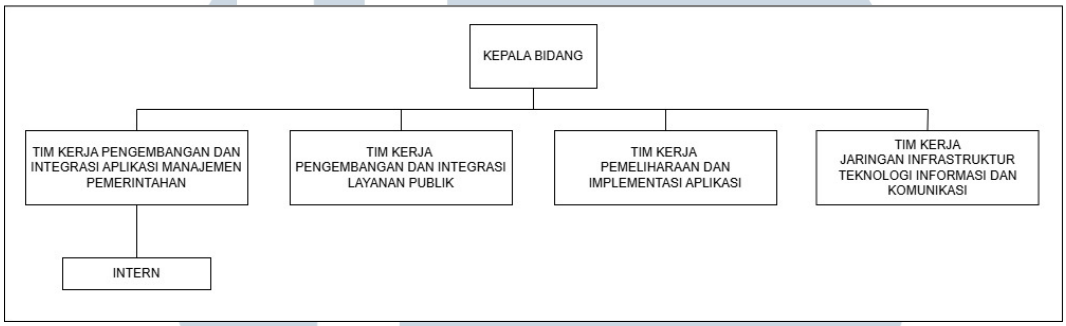
BAB 3
Pelaksanaan Magang

3.1 Kedudukan dan Koordinasi



Gambar 3.1. Struktur Tim Proyek

Selama pelaksanaan program magang di Dinas Komunikasi dan Informatika (Diskominfo) Kabupaten Tangerang, penempatan dilakukan di Bidang Pengelolaan Informatika yang berfokus pada pengembangan aplikasi digital untuk kebutuhan pemerintah daerah. Tugas utama yang diemban meliputi perancangan sistem machine learning berbasis *face recognition* guna mendukung proses verifikasi absensi pegawai.

Dalam kegiatan ini, koordinasi dilakukan secara langsung dengan tim pengembang sistem dan dibimbing oleh pembimbing lapangan, Bapak Budi Kurniawan, yang memberikan arahan serta evaluasi terhadap pekerjaan. Komunikasi dan koordinasi berlangsung melalui pertemuan tatap muka di kantor, serta didukung oleh media seperti WhatsApp dan Zoom Meeting. Pembagian tanggung jawab dan pelaporan perkembangan proyek dilakukan dalam diskusi mingguan.

Peran yang dijalankan mencakup pengembangan awal sistem, dimulai dari pengumpulan data hingga penerapan model *face recognition*, serta kontribusi dalam proses integrasi sistem ke dalam infrastruktur teknologi yang sudah ada di Diskominfo.

3.2 Tugas yang Dilakukan

Selama melaksanakan magang di Diskominfo Kabupaten Tangerang, terdapat satu proyek besar yang telah dikerjakan, yaitu proyek LINTAS (Layanan

Identifikasi Nasional Terintegrasi & Aman Sistem). LINTAS merupakan sistem absensi kehadiran pegawai berupa API dengan mengimplementasikan sistem *face recognition* berbasis *machine learning*. LINTAS hadir sebagai solusi alternatif untuk mempercepat absensi karyawan serta mengatasi masalah mesin fingerprint yang seringkali mengalami *error*. Diharapkan dengan hilangnya ketergantungan terhadap mesin fisik dengan kuantitas besar, maintenance perangkat absensi dapat diminimalisasi dan terfokus. Adapun rincian tugas-tugas yang dilakukan adalah sebagai berikut:

1. Mendesain dan mengimplementasikan sistem backend dengan *framework* `Flask` untuk pengenalan wajah.
2. Mengembangkan sistem login berbasis JSON Web Token (JWT) dan membatasi fitur pendaftaran wajah hanya untuk admin.
3. Menambahkan lapisan keamanan dengan menyimpan data pengguna dalam format `JSON` dan mengenkripsi kata sandi menggunakan `bcrypt`.
4. Mengembangkan API `register` untuk mengunggah data wajah dan penyimpanan *face embedding* menggunakan model FaceNet.
5. Membangun *pipeline* pemrosesan gambar untuk ekstraksi ciri wajah secara otomatis.
6. Mengimplementasikan API `/predict` untuk pengenalan wajah menggunakan perbandingan embedding.
7. Menyusun metode pencocokan identitas berdasarkan nilai kemiripan tertinggi dengan ambang batas (*threshold*) tertentu.
8. Menerapkan validasi dan langkah keamanan tambahan terhadap file gambar yang diunggah.
9. Menyusun struktur penyimpanan data wajah dan embedding berdasarkan NIP pegawai.

3.3 Uraian Pelaksanaan Magang

Berikut adalah uraian kegiatan magang yang telah dilaksanakan di Dinas Komunikasi dan Informatika (Diskominfo) Kabupaten Tangerang, sebagaimana dirangkum dalam Tabel 3.1 dan 3.2.

Tabel 3.1. Pekerjaan yang dilakukan selama pelaksanaan kerja magang dari minggu ke-1 sampai minggu ke-10

Minggu Ke-	Pekerjaan yang Dilakukan
1	Dilakukan riset mengenai teknologi *face recognition*, pengumpulan referensi yang relevan, pemilihan algoritma yang sesuai, penyiapan lingkungan pengembangan, serta implementasi awal algoritma *face recognition* dan *preprocessing dataset*.
2	Pengembangan dan pengujian sistem dilanjutkan dengan menambahkan fungsi evaluasi seperti akurasi, presisi, dan f1-score. Dilakukan juga demo awal *face recognition* menggunakan perangkat kantor.
3	Pengembangan sistem dilanjutkan dengan pembuatan fungsi embedding wajah (menggunakan *precompute* dari FaceNet), pengujian terhadap lebih dari 1000 wajah, serta evaluasi model berdasarkan tingkat kepercayaan, akurasi, dan waktu eksekusi.
4	Ditambahkan fungsi perbandingan wajah menggunakan `compare faces`, pengembangan file `app.py` untuk keperluan *deployment* dengan Flask, serta pengujian *endpoint* menggunakan Postman.
5	Load *testing* API menggunakan Apache JMeter, termasuk pembuatan file `app_jmeter.py` untuk *deployment*, dan pengujian awal dengan 100 serta 1000 *concurrent users*.
6	Pengujian dilanjutkan dengan jumlah pengguna yang sama, dilengkapi pengumpulan data tambahan. *Script* Groovy `JSR223 PreProcessor` dibuat untuk mengatur jalur file saat pengujian.
7	Pengujian dilakukan dengan 100 dan 1000 pengguna yang mengirim file berbeda secara bersamaan ke *server*. *Error* pada *script* Groovy diperbaiki dan hasil pengujian dianalisis dari file CSV.
8	modifikasi `precompute` agar terintegrasi ke dalam proses pendaftaran pengguna. pengambilan sampel 13 wajah untuk pengujian, disertai penyesuaian variabel dalam kode.
9	Struktur sistem *face recognition* diubah dari `return boolean` menjadi `string` dan `float`. Fungsi `compare faces`, API `blueprint`, dan `predict` diperbarui untuk mengembalikan NIP.
10	Sumber input diubah dari nama folder menjadi `string` dari file `.txt`. Fungsi `register` dimodifikasi agar menghasilkan file `.txt` berisi NIP dan nama pengguna.

Tabel 3.2. Pekerjaan yang dilakukan selama pelaksanaan kerja magang dari minggu ke-11 sampai minggu ke-14

Minggu Ke-	Pekerjaan yang Dilakukan
11	Sistem registrasi diperbarui untuk menghasilkan file `.txt`. Implementasi dilakukan di perangkat kantor, struktur API dirapikan, dan pengujian lokal dilaksanakan.
12	Pengujian API dilakukan dengan berbagai perangkat dan jaringan. Sistem diperkuat dengan autentikasi JWT, endpoint `/login`, validasi *input*, dan pembuatan `init_admin.py`.
13	Fungsi `hashing` dan verifikasi kata sandi ditambahkan dengan `bcrypt`. Utilitas pengelolaan data dibuat di `utils.py`, dan pengujian dilakukan agar token otomatis digunakan ke *endpoint* lainnya.
14	Pengecekan isi file dilakukan menggunakan `PIL.Image.verify()`, pengaturan masa berlaku token JWT, *pentesting* dengan Kali Linux, dan perapihan struktur kode dilakukan.

3.4 Perancangan

3.4.1 Uraian Masalah dan Kebutuhan

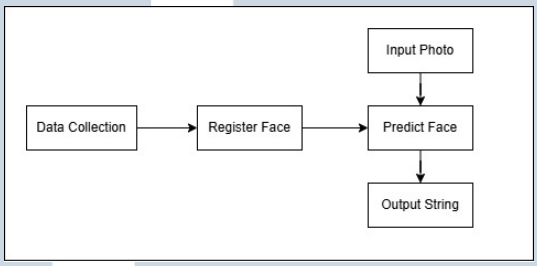
Sebelum sistem absensi LINTAS yang berbasis *face recognition* dikembangkan, Dinas Komunikasi dan Informatika (Diskominfo) Kabupaten Tangerang masih menggunakan *fingerprint* sebagai metode utama pencatatan kehadiran. Namun, mesin *fingerprint* yang digunakan sering mengalami gangguan, sehingga pegawai kesulitan melakukan absen dan harus menunggu proses perbaikan. Kondisi ini menghambat aktivitas perkantoran, sementara mengganti seluruh perangkat *fingerprint* membutuhkan biaya yang besar.

Untuk mengatasi permasalahan tersebut, pihak pengelola memutuskan untuk mengembangkan sistem absensi otomatis yang memanfaatkan teknologi *face recognition*. Sistem ini bekerja dengan mencocokkan foto wajah yang dikirim oleh pengguna dengan data wajah yang telah direkam saat registrasi awal. Dengan cara ini, proses absensi tidak lagi bergantung pada mesin *fingerprint*.

3.4.2 Gambaran Umum Sistem

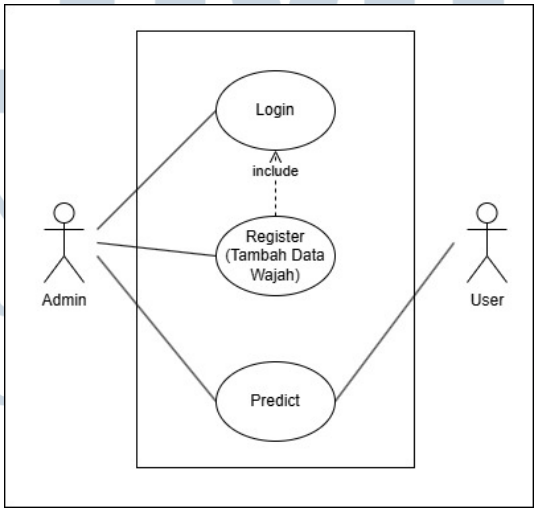
Pada API LINTAS, terdapat dua fungsi utama yaitu `register` dan `predict`. `register` berfungsi untuk mendaftarkan wajah ke database, sedangkan `predict`

untuk mengidentifikasi wajah berdasarkan data yang dimasukkan saat `register`. Gambar wajah yang sudah dikumpulkan diproses dalam fungsi `register` sehingga mengembalikan output berupa file embedding `.pkl` dan `.txt` yang disimpan di server. Kemudian foto baru yang akan dideteksi dikirim ke fungsi `predict` yang akan mendeteksi wajah pertama dari dataset yang dikenali berdasarkan embedding tersebut dan mengembalikan output berupa string yang diambil dari file `.txt` dalam masing-masing folder wajah.



Gambar 3.2. Pipeline Kerja Sistem

Demi menghindari penyalahgunaan fungsi `register`, maka harus dipastikan pula fungsi tersebut hanya dapat diakses oleh pengguna tertentu. Oleh karena itu dibedakanlah pengguna menjadi dua kategori aktor, Admin dan *User* atau pengguna biasa. Admin memiliki kendali penuh atas sistem, termasuk kemampuan untuk masuk (*login*), mendaftarkan data wajah pengguna, dan melakukan prediksi wajah sebagai bagian dari proses verifikasi kehadiran. Di sisi lain, *User* hanya dapat mengunggah gambar untuk keperluan prediksi wajah, tanpa perlu melalui proses *login*.



Gambar 3.3. Use Case Diagram

3.4.3 Framework yang digunakan

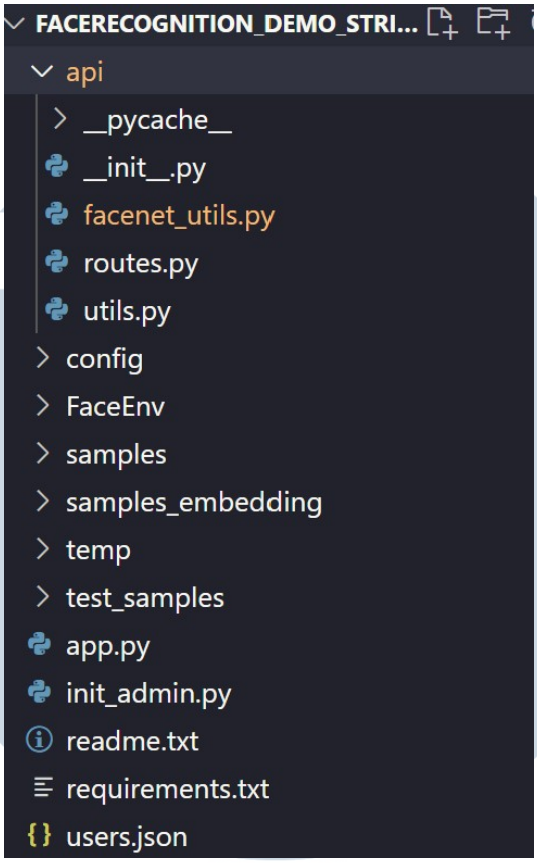
LINTAS dikembangkan menggunakan bahasa pemrograman Python untuk membangun sistem *backend* yang menangani otentikasi pengguna, pendaftaran wajah, dan verifikasi wajah berbasis *machine learning*. Python dipilih karena menyediakan banyak pustaka yang mendukung pengembangan sistem berbasis pembelajaran mesin. Selain cocok dengan kebutuhan teknis pengembang dalam merancang proses pengenalan wajah otomatis, Python juga dikenal karena sintaksisnya yang mudah dipahami, sehingga mempercepat proses pengembangan dan pemeliharaan sistem.

Untuk sisi *backend*, digunakan *framework* Flask— *framework* ringan berbasis Python yang fleksibel dan mudah digunakan. Flask dimanfaatkan untuk membangun REST API yang menghubungkan antarmuka pengguna dengan proses server. Dengan memanfaatkan fitur `blueprint`, pengelolaan *endpoint* seperti `/login`, `/register`, dan `/predict` menjadi lebih terstruktur. Demi menjaga keamanan, digunakan pustaka `flask-jwt-extended` yang memastikan hanya pengguna dengan peran 'admin' yang dapat mengakses *endpoint* `/register` dan menambahkan data wajah, melalui sistem otentikasi token JWT (JSON Web Token).

3.5 Implementasi

3.5.1 Struktur Proyek

LINTAS disusun dari beberapa folder dan file yang secara umum berbentuk struktur umum API yaitu folder API yang berisikan *utility* dan konfigurasi dari masing-masing *endpoint*, `config` yang berisikan konfigurasi dari API itu sendiri, `samples` dan `samples_embedding` yang menyimpan data-data wajah, `app.py`, dan `users.json`.



Gambar 3.4. Struktur Proyek

1. `facenet_utils.py`: Berisi berbagai fungsi untuk pengolahan citra wajah, termasuk pemuatan model `FaceNet`, ekstraksi fitur wajah (*embedding*), dan perhitungan jarak antar fitur wajah.

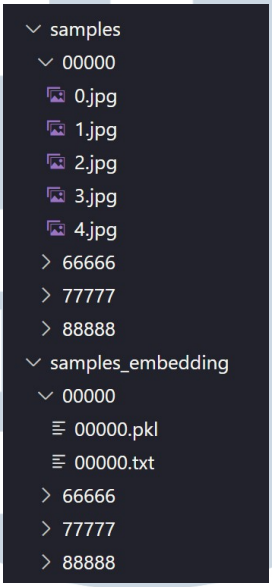
2. `routes.py`: Menyediakan sejumlah *endpoint* REST API seperti `/login`, `/register`, dan `/predict`, yang berperan dalam proses autentikasi dan verifikasi berbasis wajah.

3. `app.py`: Merupakan skrip utama yang mengeksekusi aplikasi berbasis `Flask`. Di dalamnya terdapat konfigurasi seperti pengaturan `JWT`, inisialisasi server, dan pendaftaran `blueprint` API.

3.5.2 Dataset

Dataset yang digunakan dalam proyek LINTAS berbentuk *directory-based dataset* yaitu *dataset* yang berbentuk folder-folder yang terstruktur. Dalam *dataset* ini terdapat dua folder utama yaitu folder `samples` dan `samples_embedding`.

Folder `samples` ditujukan untuk menyimpan gambar asli dari pemilik wajah. Dari folder utama, foto-foto disimpan di dalam *subfolder* yang diberi nama sesuai NIP (Nomor Induk Pegawai) pemilik wajah. Folder `samples_embedding` berisi *subfolder* seperti folder `samples` hanya saja file yang disimpan adalah file *embedding* `.pkl` dan informasi pemilik wajah `.txt`.



Gambar 3.5. Struktur *Database* LINTAS

3.5.3 Implementasi FaceNet

FaceNet merupakan model *deep learning* yang telah dilatih sebelumnya dan banyak dimanfaatkan untuk mengekstraksi ciri khas wajah dalam bentuk representasi angka, yang dikenal sebagai *face embedding*. Karena sudah *pre-trained*, FaceNet bisa langsung digunakan tanpa perlu proses pelatihan dari awal, sehingga mempercepat pengembangan sistem dan secara signifikan mengurangi beban komputasi.

Kemampuannya ini menjadikannya pilihan yang cocok untuk sistem verifikasi wajah di berbagai aplikasi, termasuk dalam proyek ini. *Embedding* yang dihasilkan memiliki keunikan tersendiri untuk setiap individu dan berfungsi sebagai identitas digital yang konsisten dalam proses pencocokan dan verifikasi. Dalam sistem ini, setelah tahap registrasi menyatakan data pengguna dan foto valid, setiap gambar wajah akan diproses menggunakan Kode 3.1 berikut.

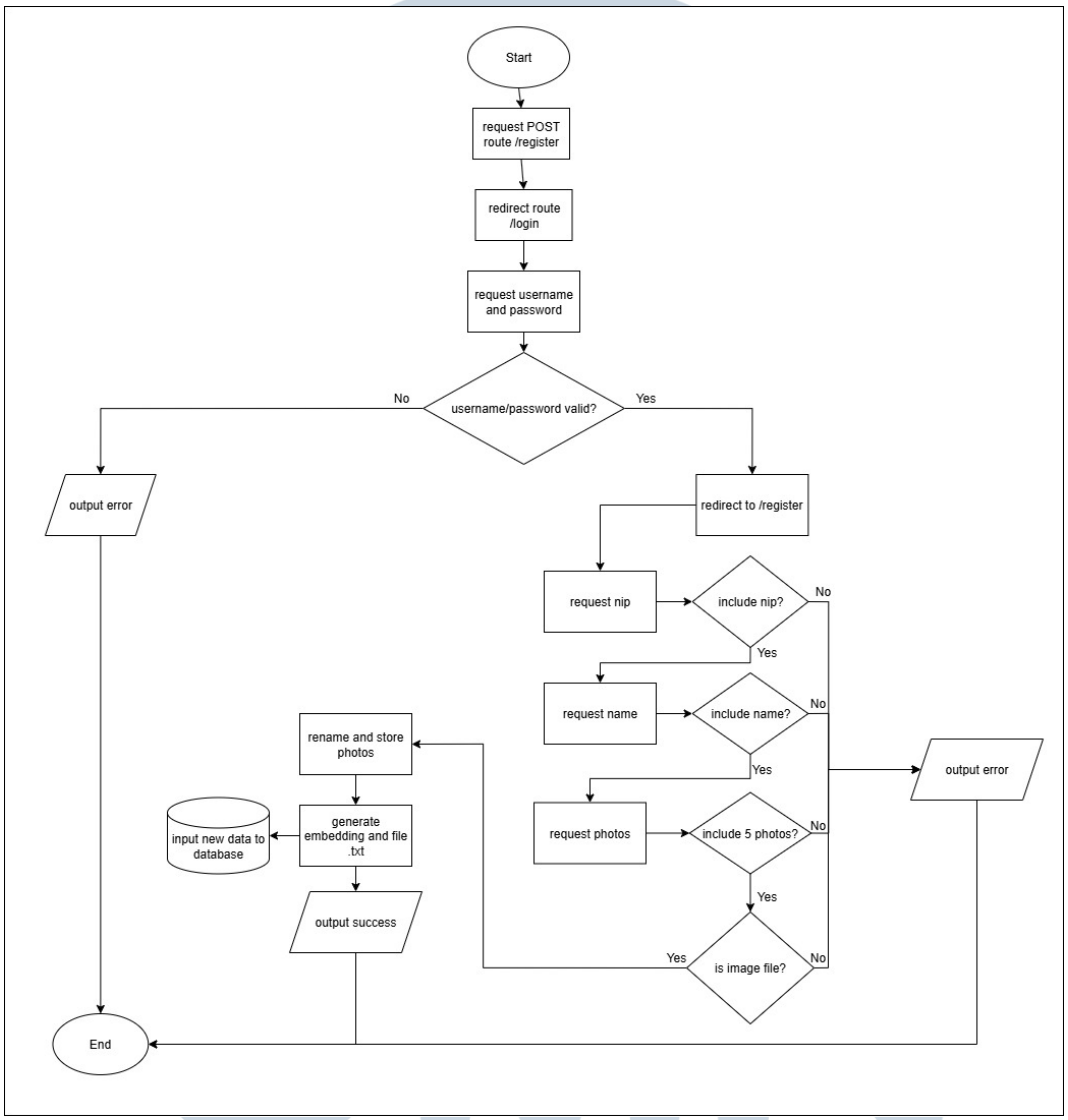
```
facenet = FaceNet()
model = facenet.model
```

```
3
4  def get_embedding(image_path):
5      try:
6          face = preprocess_image(image_path)
7          return model.predict(face)[0]
8      except Exception as e:
9          print(f"Error extracting embedding: {e}")
10         return None
```

Kode 3.1: Kode ekstraksi embedding dengan FaceNet

Fungsi `get_embedding` mengambil `path` gambar yang telah melalui *preprocessing* sebelum mengirim gambar tersebut ke model FaceNet untuk diubah menjadi *embedding*. Hasil *embedding* ini akan disimpan dalam file `pickle` (`.pkl`) yang kemudian digunakan dalam proses pencocokan wajah. Implementasi FaceNet ini memberikan keakuratan tinggi dalam mengenali wajah, bahkan ketika terdapat variasi pencahayaan, pose, atau ekspresi.

3.5.4 Endpoint */Register* (Pendaftaran Data Wajah)



Gambar 3.6. Alur Kerja Endpoint register

Registrasi wajah dimulai dengan melakukan *request* ke *endpoint* `/register`. Sebelum diarahkan ke *endpoint* `/register`, pengguna akan mendapat *redirect* ke *endpoint* `/login` untuk verifikasi akses admin. Jika *username* dan *password* yang diberikan valid, barulah sistem akan mengarahkan pengguna ke *endpoint* `/register`. API akan meminta request data-data yang dibutuhkan yaitu berupa NIP (Nomor Induk Pegawai), nama, dan 5 buah foto yang akan melalui proses *training*. Jika semua data lengkap, maka sistem akan memproses data-data tersebut ke *database*. Jika proses berjalan dengan sukses, maka pengguna akan

mendapatkan *output* yang menyatakan bahwa proses registrasi wajah telah berhasil.

Saat diproses melalui FaceNet, foto-foto wajah yang diproses akan melalui proses *feature extraction* dimana sistem akan mengidentifikasi bentuk fitur wajah seperti mata, hidung, dan mulut yang kemudian disimpan dalam file embedding. Untuk meminimalisir noise dan *overfitting*, maka 5 foto yang digunakan diambil dari berbagai sudut dan terlebih dahulu melalui proses *preprocessing* secara manual dengan *cropping*.

Gambar 3.7. Sampel Foto untuk Proses Register

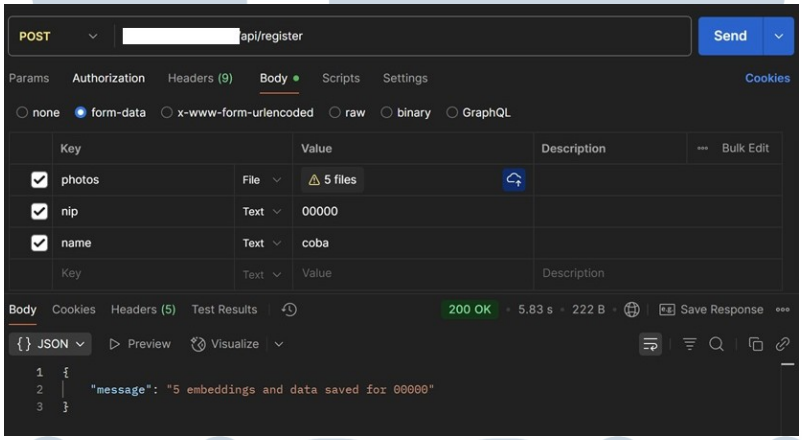
Endpoint ini hanya dapat diakses pengguna dengan peran admin, yang diverifikasi dengan mengimplementasi token JWT. Kode Python untuk *endpoint* `/register`, dapat dilihat pada Kode 3.3.

```
@api_blueprint.route('/register', methods=['POST'])
@jwt_required()
def register():
    claims = get_jwt()
    if claims.get("role") != "admin":
        return jsonify({"error": "Unauthorized. Only admin can register users."}), 403

    if 'nip' not in request.form:
        return jsonify({"error": "Person nip is required"}), 400
    elif 'name' not in request.form:
        return jsonify({"error": "Person name is required"}), 400

    person_nip = request.form['nip']
    person_name = request.form['name']
    files = request.files.getlist('photos')

    if len(files) < 5:
        return jsonify({"error": "Exactly 5 images are required"}), 400

    for file in files:
        if not allowed_file(file.filename):
```

```
22              return jsonify({"error": f"Invalid file type: {file.
    filename}. Only jpg, jpeg, png allowed."}), 400
23
24      result, status = register_user_and_generate_embeddings(
    person_nip, person_name, files)
25      return jsonify(result), status
```

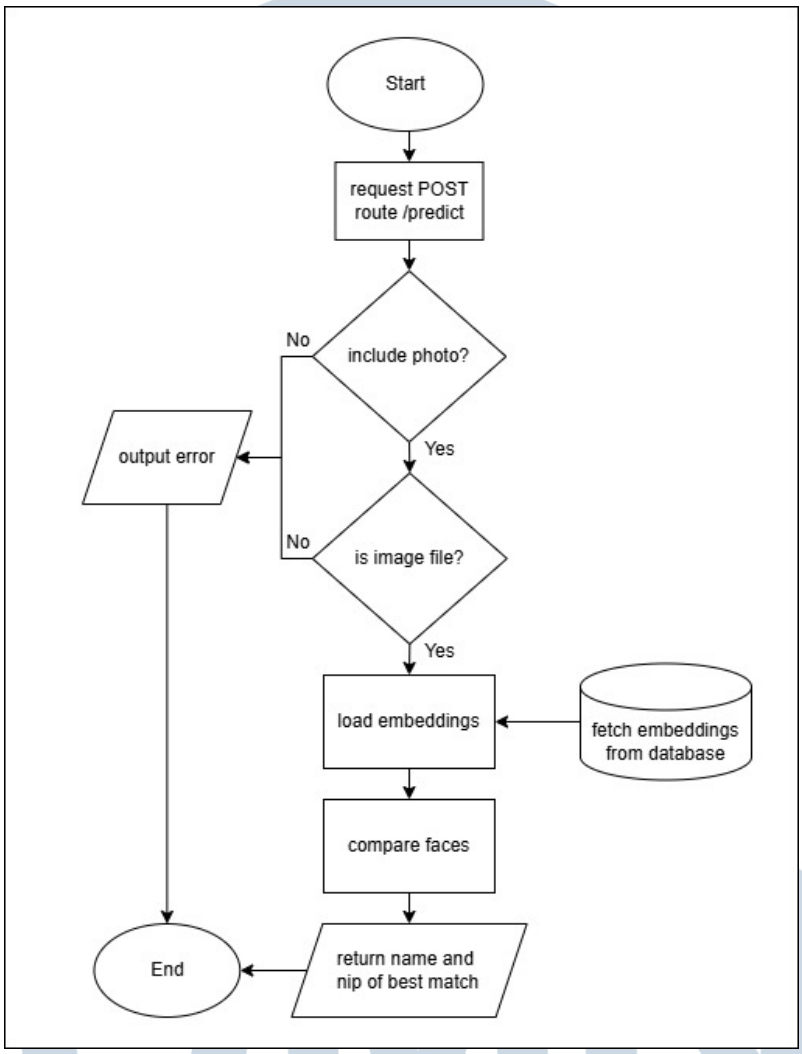
Kode 3.2: Contoh kode endpoint `/register`

Hasil dari ekstraksi gambar berupa embedding dari wajah yang disimpan dalam file `pickle` (`.pkl`). Informasi pengguna seperti nama dan NIP akan disimpan dalam bentuk file teks (`.txt`). Kedua file inilah yang nantinya akan diambil oleh sistem ketika mengidentifikasi wajah.



Gambar 3.8. Respon API `/register`

3.5.5 *Endpoint* `/predict`

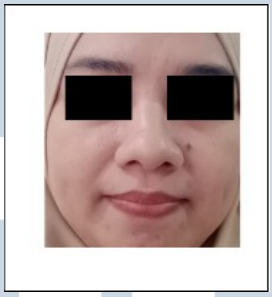


Gambar 3.9. Alur Kerja *Endpoint* `/predict`

Prediksi wajah dimulai dengan melakukan request ke *endpoint* `/predict`. Sistem akan mengecek apakah foto yang dikirim valid yaitu foto merupakan file gambar. Jika sudah maka sistem akan memuat file *embedding* dan melakukan komparasi wajah secara rekursif dari *dataset* yang ada. Jika ditemukan yang sesuai diatas *treshold*, maka sistem akan mengembalikan output berupa string yang berisikan nama dan NIP yang diambil dari file `.txt` yang terdapat di folder di mana file *embedding* sesuai.

Seperti telah dijelaskan pada subbab sebelumnya, data embedding wajah yang tersimpan merupakan foto yang sudah melalui cropping untuk meminimalisir

noise. maka dari itu, foto yang digunakan untuk prediksipun sebelumnya juga harus telah melalui proses cropping.



Gambar 3.10. Sampel Foto untuk Proses *Predict*

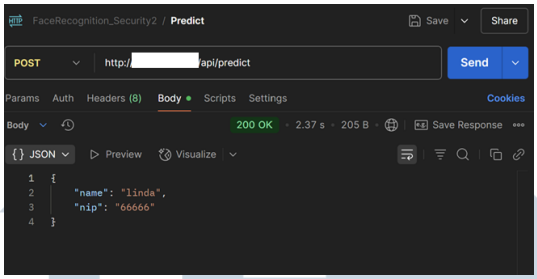
Secara *default*, nama dan NIP diatur sebagai "*unknown*" maka jika tidak ada yang cocok, sistem akan mengembalikan *default value* berupa "*unknown*". Berikut adalah kode Python yang digunakan untuk *endpoint* `/predict`, dapat dilihat pada Kode 3.2.

```
@api_blueprint.route('/predict', methods=['POST'])
def predict():
    if 'photo' not in request.files:
        return jsonify({"error": "Image file is required"}), 400

    photo = request.files['photo']

    if not allowed_file(photo.filename):
        return jsonify({"error": "File type not allowed. Only jpg, jpeg, png accepted."}), 400

    temp_filename = f"{uuid4().hex}.jpg"
    image_path = os.path.join(TEMP_DIR, temp_filename)
    photo.save(image_path)

    test_embedding = get_embedding(image_path)
    os.remove(image_path)

    if test_embedding is None:
        return jsonify({"error": "Failed to extract face embedding"}), 500

    best_nip = "unknown"
    best_name = "unknown"
    best_similarity = 0.0
```

```
24
25      for folder_name in os.listdir(EMBEDDINGS_DIR):
26          pkl_path = os.path.join(EMBEDDINGS_DIR, folder_name, f"{
    folder_name}.pkl")
27          txt_path = os.path.join(EMBEDDINGS_DIR, folder_name, f"{
    folder_name}.txt")
28
29          if not os.path.exists(pkl_path) or not os.path.exists(
    txt_path):
30              continue
31
32          with open(pkl_path, "rb") as f:
33              known_embeddings = pickle.load(f)
34
35          similarity = compare_faces(test_embedding,
    known_embeddings, THRESHOLD)
36
37          if similarity > best_similarity:
38              best_similarity = similarity
39              best_nip, best_name = read_txt(txt_path)
40
41      print(f"Best match: {best_nip} (Confidence: {round(
    best_similarity * 100, 2)}%)")
42      return jsonify({"nip": best_nip, "name": best_name}), 200
```

Kode 3.3: Kode endpoint /predict untuk verifikasi wajah

Pengguna mengirimkan file gambar melalui permintaan `POST`. *Server* akan terlebih dahulu memeriksa validitas file dan formatnya (hanya mendukung `.jpg`, `.jpeg`, atau `.png`). Setelah lolos validasi, gambar disimpan di folder sementara untuk kemudian diproses. Fungsi `get_embedding()` akan digunakan untuk mengekstrak *embedding* wajah dari gambar menggunakan model FaceNet. Jika tidak ada wajah yang terdeteksi atau proses *embedding* gagal, sistem akan memberikan respons berupa *error*. Setelah berhasil, sistem membandingkan *embedding* dari gambar tersebut dengan data *embedding* yang sudah tersimpan sebelumnya. Proses ini dilakukan dengan membuka setiap file `.pkl` yang berisi *embedding* pengguna terdaftar, lalu menggunakan fungsi `compare_faces()` untuk menghitung tingkat kemiripan. Sistem akan memberikan respons seperti ini jika prediksi berhasil.



Gambar 3.11. Respon API `/predict`

Jika nilai *cosine similarity* melebihi ambang batas dan merupakan yang tertinggi sejauh ini, maka sistem akan menyimpan informasi NIP dan nama pengguna dari file `.txt` yang sesuai sebagai kandidat terbaik. Pada akhirnya, sistem akan memberikan respon berupa NIP dan nama pengguna yang paling menyerupai gambar yang diunggah, disertai dengan tingkat kepercayaan yang dicatat di `log` untuk keperluan *debugging*.

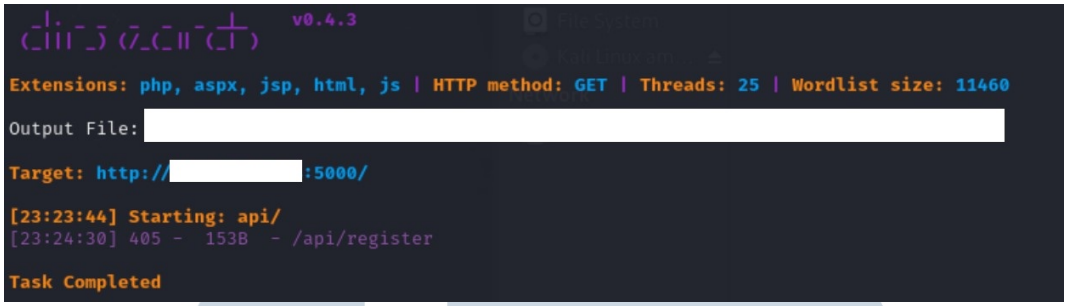
3.5.6 Implementasi Keamanan Sistem

Dalam membangun sistem verifikasi wajah berbasis Flask, aspek keamanan menjadi prioritas utama untuk mencegah penyalahgunaan akses maupun manipulasi data.

3.5.7 Pentesting

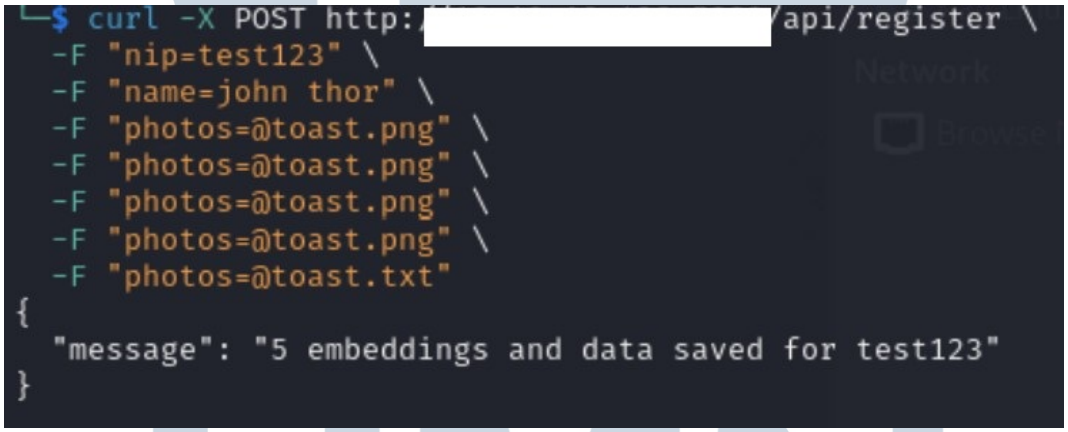
Setelah API berhasil di-*deploy* dan dijalankan di *server*, tahap selanjutnya adalah melakukan pengujian keamanan (*pentesting*). Karena API ini hanya menyediakan endpoint dan tidak menampilkan halaman atau layanan berbasis host secara umum, maka pengujian menggunakan metode *host-based* seperti Nmap atau GVM tidak relevan untuk konteks ini.

Sebagai langkah awal, pengujian dilakukan menggunakan *tool* Dirsearch. Dengan memasukkan nama domain ke dalam Dirsearch, ditemukan bahwa *endpoint* `/api` memiliki *subpath* tambahan yaitu `/register`.



Gambar 3.12. Hasil Uji *Tool* Dirsearch

Endpoint ini kemudian diuji menggunakan perintah `curl` dengan metode `POST`. Dari tanggapan *error* terkait *missing requirement*, dapat dianalisis parameter atau file apa saja yang dibutuhkan untuk mengakses *endpoint* tersebut secara valid.



Gambar 3.13. Uji dengan Curl

Setelah memenuhi parameter yang dibutuhkan, proses registrasi berhasil, dan datanya tercatat ke dalam basis data. Hal ini mengindikasikan bahwa pihak luar, yang bukan pemilik sistem, tetap dapat melakukan manipulasi data melalui *endpoint* publik tersebut.

Lebih lanjut, meskipun sistem secara eksplisit meminta *input* berupa file gambar, nyatanya tidak terdapat validasi ketat terhadap jenis file yang diunggah. File dengan ekstensi `.txt`, misalnya, tetap diterima dan diproses oleh sistem. Celah ini membuka kemungkinan bagi pihak tidak bertanggung jawab untuk mengunggah file berbahaya, seperti *malware*, yang dapat membahayakan infrastruktur server maupun data pengguna lainnya. Oleh karena itu, validasi jenis file dan mekanisme keamanan input perlu ditingkatkan untuk mencegah potensi eksploitasi.

3.5.8 Implementasi JSON Web Token (JWT)

Berdasarkan hasil penetration testing, dua pendekatan utama diterapkan untuk mengatasi permasalahan yang ditemukan: pertama, penggunaan JSON Web Token (JWT) dalam proses *login* untuk mengelola autentikasi dan otorisasi pengguna terhadap fitur-fitur tertentu dalam sistem; kedua, validasi terhadap jenis dan isi berkas gambar yang diunggah untuk memastikan bahwa hanya berkas yang sah dan sesuai standar yang dapat diproses. Kedua pendekatan ini diimplementasikan langsung pada sisi *backend* melalui beberapa *endpoint* yang telah dirancang secara khusus.

JSON Web Token (JWT) merupakan metode autentikasi dan otorisasi berbasis token yang tidak menyimpan status (*stateless*). JWT memastikan bahwa hanya pengguna yang telah tervalidasi yang memiliki izin untuk mengakses fitur tertentu dalam sistem. Token ini terdiri dari tiga komponen utama, yaitu *header*, *payload*, dan *signature*. Informasi penting seperti identitas pengguna dan peran akses (misalnya admin) dimuat dalam *payload*, kemudian diamankan menggunakan *signature* yang dibentuk melalui secret key.

Untuk menerapkan JWT dalam aplikasi Flask, diperlukan pengaturan secret key serta durasi masa berlaku token. Contoh implementasi JWT dalam bentuk skrip Python ditampilkan pada Kode 3.4.

```
app.config["JWT_SECRET_KEY"] = os.getenv("JWT_SECRET_KEY", "dev-
    secret")
app.config["JWT_ACCESS_TOKEN_EXPIRES"] = timedelta(minutes = 5)
jwt = JWTManager(app)
```

Kode 3.4: Konfigurasi JWT pada Flask

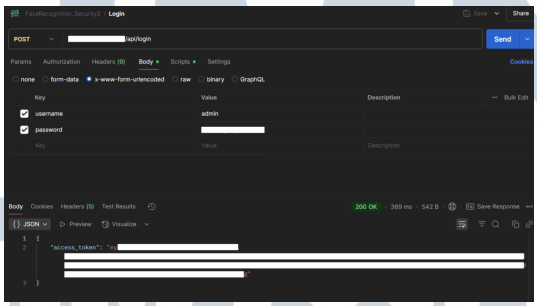
Konfigurasi ini memungkinkan setiap token yang dihasilkan memiliki masa berlaku selama 5 menit, serta menjaga keamanan data melalui *secret key* yang dapat disesuaikan menggunakan *environment variable*. Setelah proses *login* berhasil, token diberikan kepada pengguna dan akan digunakan kembali saat mengakses *endpoint* yang memerlukan autentikasi.

Proses *login* menggunakan JWT diterapkan pada *endpoint* `/login`, seperti yang ditunjukkan pada Kode 3.5. Saat pengguna mengirimkan *username* dan *password* ke `/login`, sistem akan memverifikasi kredensial dengan mencocokkannya terhadap data yang tersimpan dalam berkas `users.json`. Jika valid, sistem akan menghasilkan *access token* melalui fungsi `create_access_token()`.

```python
@api_blueprint.route('/login', methods=['POST'])
def login():
    data = request.form
    username = data.get("username")
    password = data.get("password")

    users = load_users()
    user = users.get(username)

    if user and verify_password(password, user["password"]):
        access_token = create_access_token(
            identity=username,
            additional_claims={"role": user["role"]}
        )
        return jsonify(access_token=access_token), 200

    return jsonify({"error": "Invalid credentials"}), 401
```

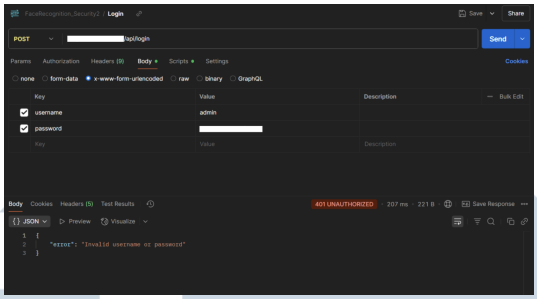
Kode 3.5: Endpoint Login dengan Flask

JWT token yang diperoleh oleh pengguna akan dipakai untuk mengakses *endpoint* lain yang memerlukan autentikasi, seperti proses pendaftaran. Token ini dikirimkan melalui `header` *Authorization* pada setiap `request`. Selama token tersebut valid dan menunjukkan bahwa pengguna memiliki peran yang sesuai, maka akses akan diizinkan.



Gambar 3.14. Respons API Login Success

Token ini memuat identitas pengguna beserta informasi tambahan seperti peran 'admin'. Token tersebut akan disertakan dalam setiap permintaan selanjutnya ke endpoint yang membutuhkan autentikasi. Namun, jika pengguna memasukkan data yang tidak valid (misalnya *username* tidak terdaftar atau *password* salah), maka sistem akan menolak proses *login* dan mengembalikan pesan kesalahan seperti ditampilkan pada Gambar 3.15 berikut.



Gambar 3.15. Respons API Login Failed

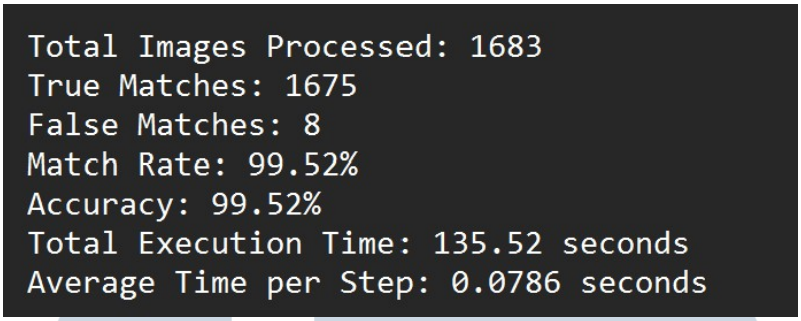
3.5.9 Pengujian

3.5.10 Dataset

Sebelum API disusun, dilakukan terlebih dahulu pengujian dan evaluasi model dengan dataset *Labeled Faces in The Wild* (LFW). Dataset ini dikembangkan oleh para peneliti dari University of Massachusetts, Amherst, dan mencakup 13.233 foto wajah milik 5.749 individu yang diperoleh dari internet. Semua gambar telah diproses dengan algoritma Viola-Jones face detection guna memastikan posisi wajah berada di tengah dan dapat dikenali secara konsisten. Dari keseluruhan individu, terdapat 1.683 orang yang memiliki setidaknya dua foto berbeda, sehingga memungkinkan pengujian sistem dalam mengenali wajah yang sama meskipun dengan gambar yang bervariasi. LFW memang dirancang untuk mendukung penelitian di bidang pengenalan wajah, terutama dalam menghadapi variasi kondisi seperti pencahayaan, ekspresi, sudut pengambilan gambar, dan latar belakang, sehingga sering dijadikan standar referensi dalam evaluasi performa model face recognition karena kualitas dan keragaman datanya yang representatif.

3.5.11 Evaluasi

Pengujian dilakukan untuk menilai kinerja sistem pengenalan wajah dalam proses identifikasi dan verifikasi wajah berdasarkan data yang telah tersedia. Dataset Labeled Faces in the Wild (LFW) dipilih sebagai bahan uji karena memiliki tingkat kompleksitas dan variasi gambar yang tinggi, menjadikannya cocok untuk menguji ketepatan serta ketahanan sistem terhadap berbagai kondisi nyata, seperti pencahayaan, ekspresi wajah, dan sudut pandang. Hasil pengujian menunjukkan bahwa sistem mampu mencocokkan wajah dengan sangat baik.



Gambar 3.16. Hasil Pengujian Model FaceNet

Dari hasil pada Gambar 3.16 tersebut, dapat disimpulkan bahwa sistem memiliki tingkat akurasi sebesar 99% dan waktu eksekusi yang singkat untuk tiap gambar yang diproses. Hal ini menunjukkan bahwa sistem face recognition yang dikembangkan layak digunakan untuk implementasi verifikasi absensi karena mampu mengenali wajah dengan presisi tinggi dalam waktu yang relatif singkat.

3.6 Kendala dan Solusi yang Ditemukan

Selama kegiatan magang berlangsung, terdapat beberapa tantangan dan hambatan yang dihadapi dalam proses pengembangan sistem deteksi wajah. Beberapa permasalahan yang muncul di antaranya:

1. Pada tahap awal, akurasi sistem deteksi wajah masih belum maksimal karena input yang digunakan berupa gambar wajah utuh tanpa melalui proses pemotongan terlebih dahulu. Kondisi ini menyebabkan banyak gangguan (*noise*) pada citra yang masuk, sehingga berdampak negatif terhadap performa model dalam mengenali wajah secara akurat.
2. Sistem awal belum dilengkapi dengan fitur keamanan untuk membatasi akses ke *endpoint* tertentu, sehingga terdapat potensi risiko keamanan. Pengguna yang belum terverifikasi tetap bisa mengakses fitur penting seperti pendaftaran wajah baru, yang menimbulkan kekhawatiran terkait otorisasi dan kontrol akses.

Untuk mengatasi berbagai kendala tersebut, beberapa solusi telah diterapkan guna mendukung kelancaran proses pengembangan sistem, yaitu:

1. Untuk meningkatkan tingkat akurasi, dilakukan proses *preprocessing* secara external berupa pemotongan pada bagian wajah pengguna sebelum gambar

diproses oleh model. Dengan melakukan data yang telah di potong, data input menjadi lebih bersih dan terfokus pada area wajah, yang secara signifikan meningkatkan performa deteksi model.

2. Dalam rangka memperkuat sistem keamanan, diterapkan metode autentikasi dan otorisasi menggunakan *JSON Web Token (JWT)*. Penggunaan JWT memastikan bahwa hanya pengguna yang telah berhasil *login* dan memiliki token yang sah yang dapat mengakses *endpoint* tertentu. Selain itu, informasi peran pengguna (seperti `admin`) juga disimpan di dalam token untuk mengatur dan membatasi akses terhadap fitur tertentu secara lebih ketat.