Implementasi Zero Trust Access Control pada Aplikasi Penyimpanan Dokumen

Disusun Oleh :
2201020013 - Luthfi Kurniawan
2201020015 - M. Afief Anugrah
2201020018 - Aditya Firmansyah
2201020092 - Halta Putra Ash Sidiq

Teknik Informatika
Fakultas Teknik dan Teknologi Kemaritiman
Universitas Maritim Raja Ali Haji
2025

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam ekosistem digital saat ini, manajemen dokumen elektronik (*e-Documents*) merupakan aset vital bagi organisasi. Namun, model keamanan tradisional yang mengandalkan *perimeter-based security* (keamanan batas luar) tidak lagi cukup untuk menghadapi ancaman modern. Kelemahan utama model lama adalah pemberian kepercayaan implisit kepada pengguna di dalam jaringan, yang memungkinkan terjadinya pergerakan lateral (*lateral movement*) jika satu akun berhasil dikompromi.

Topik Zero Trust Access Control menjadi sangat penting karena mengadopsi prinsip *"never trust, always verify"*. Implementasi ini memastikan bahwa setiap permintaan akses, baik dari dalam maupun luar jaringan, harus melalui verifikasi ketat. Dengan menerapkan Zero Trust pada aplikasi manajemen dokumen, risiko kebocoran data akibat pencurian kredensial, serangan *insider threat*, maupun serangan *brute-force* dapat diminimalisir secara signifikan melalui perlindungan berlapis pada level aplikasi.

1.2 Tujuan Proyek

Proyek ini bertujuan untuk merancang dan mengimplementasikan sistem keamanan yang tangguh pada aplikasi manajemen dokumen sederhana dengan mencapai sasaran berikut:

1. Verifikasi Eksplisit (*Verify Explicitly*): Menerapkan sistem autentikasi yang kuat menggunakan *JSON Web Token* (JWT) dan enkripsi kata sandi dengan algoritma *Bcrypt* untuk memastikan identitas pengguna divalidasi pada setiap sesi.
2. Akses Hak Istimewa Minimum (*Least Privilege Access*): Mengimplementasikan *Role-Based Access Control* (RBAC) untuk membatasi fungsi kritis (seperti mengunggah, mengedit, dan menghapus dokumen) hanya kepada peran (*role*) yang berwenang (Admin, Editor, Viewer).
3. Penerapan Asumsi Pelanggaran (*Assume Breach*): Membangun mekanisme mitigasi dampak serangan melalui penerapan *Session Timeout* otomatis, pencatatan *Audit Log* yang mendeteksi IP klien, serta sistem *Account Lockout* untuk merespons serangan *brute-force*.

1.3 Ruang Lingkup Implementasi

Agar pengembangan tetap terukur dan sesuai dengan jadwal proyek, ruang lingkup pekerjaan dibatasi sebagai berikut:

Apa yang dilakukan:

- Pengembangan *backend* menggunakan Node.js dengan *framework* Express.js dan database relasional melalui Sequelize ORM.
- Implementasi fitur registrasi pengguna dengan *hashing* kata sandi menggunakan *Bcrypt*.
- Implementasi sistem login dan proteksi *route* menggunakan *middleware* autentikasi (JWT) dan otorisasi (RBAC).
- Penerapan fitur keamanan tambahan: *Session Timeout* (melalui konfigurasi `expiresIn` JWT) dan pelacakan IP address pada sistem *Audit Log*.
- Pengujian keamanan sistem melalui skenario serangan *Brute-Force* dan upaya *Bypass* hak akses menggunakan alat bantu seperti Postman.
- Server hanya bisa di akses menggunakan key dengan login password dimatikan.

Apa yang tidak dilakukan:

- Penerapan keamanan pada lapisan jaringan (*Network Layer*) seperti konfigurasi Firewall fisik atau VPN.
- Penyediaan infrastruktur penyimpanan *cloud* skala besar (proyek fokus pada logika kontrol akses).
- Implementasi Multi-Factor Authentication (MFA) berbasis perangkat keras atau biometrik.
- Pengembangan antarmuka pengguna (*Frontend*) yang kompleks; fokus utama adalah pada keamanan fungsionalitas API dan *backend*.

BAB II
DASAR TEORI DAN PERANCANGAN

2.1 Prinsip Zero Trust

Arsitektur Zero Trust (ZTA) adalah model keamanan siber yang berpusat pada data dan berasumsi bahwa tidak ada kepercayaan implisit yang diberikan kepada aset atau pengguna berdasarkan lokasi fisik atau jaringan mereka. Berdasarkan laporan ini, sistem mengadopsi tiga prinsip utama Zero Trust sebagaimana didefinisikan oleh NIST SP 800-207:

1. Verify Explicitly (Verifikasi Secara Eksplisit): Selalu melakukan autentikasi dan otorisasi berdasarkan semua titik data yang tersedia. Dalam proyek ini, hal ini diimplementasikan melalui penggunaan JSON Web Token (JWT). Setiap permintaan ke *endpoint* yang dilindungi wajib menyertakan token yang valid di dalam header *Authorization* untuk membuktikan identitas pengguna.
2. Least Privilege Access (Akses Hak Istimewa Minimum): Membatasi akses pengguna dengan *Just-In-Time* dan *Just-Enough-Access*. Proyek ini menerapkan Role-Based Access Control (RBAC) dengan tiga peran (Admin, Editor, Viewer). Misalnya, seorang *Viewer* hanya diberikan hak untuk melihat dokumen, sementara hak untuk mengedit atau menghapus hanya diberikan kepada *Editor* atau *Admin*.
3. Assume Breach (Asumsi Pelanggaran): Meminimalkan dampak jika terjadi kebocoran dan melakukan pemantauan berkelanjutan. Implementasi teknisnya mencakup penggunaan Audit Log untuk mencatat setiap aktivitas beserta alamat IP pengguna, Session Timeout (token kedaluwarsa), serta mekanisme Account Lockout jika terjadi percobaan *brute-force* lebih dari batas yang ditentukan.

2.2 Alat yang digunakan

2.2.1 Infrastruktur dan Perangkat Keras (Virtual Machine)

Komponen	**Sistem Operasi**	**Spesifikasi (RAM/Core)**	**Peran dalam Proyek**
Networking/Gateway	OpenWrt	1 GB RAM, 1 Core	Sebagai pengatur lalu lintas jaringan dan firewall antara klien dan server.

Application Server	Ubuntu Server	3 GB RAM, 2 Core	*Host* utama aplikasi backend (Node.js), database, dan logika keamanan.
Client Device	Ubuntu Desktop	5 GB RAM, 2 Core	Perangkat pengguna sah untuk mengakses aplikasi manajemen dokumen.
Attacker Machine	Kali Linux	4 GB RAM, 2 Core	Digunakan untuk simulasi serangan siber (brute-force dan bypass akses).

2.2.2 Perangkat Lunak (Software)

- **Node.js & Express.js:** *Runtime* dan *framework* untuk membangun API backend.
- **Sequelize ORM:** Untuk manajemen database relasional dan pencegahan *SQL Injection*.
- **Bcrypt:** Algoritma *hashing* untuk mengamankan kata sandi pengguna di database.
- **Jsonwebtoken (JWT):** Standar token untuk proses autentikasi dan pertukaran informasi sesi.
- **Postman:** Alat pengujian API untuk validasi fungsionalitas dan keamanan sistem.

2.3 Diagram Arsitektur

Arsitektur sistem ini dirancang menggunakan pendekatan *Layered Security* (keamanan berlapis) yang mengintegrasikan keamanan jaringan dan keamanan level aplikasi. Perancangan ini memastikan bahwa setiap entitas, baik klien sah maupun penyerang, harus melewati kontrol akses yang ketat.

2.3.1 Arsitektur Jaringan (Network Topology)

Secara infrastruktur, sistem ini mengadopsi topologi *Hub-and-Spoke* sederhana di dalam lingkungan virtual. OpenWrt bertindak sebagai *Central Gateway* yang menghubungkan segmen pengguna dengan segmen server.

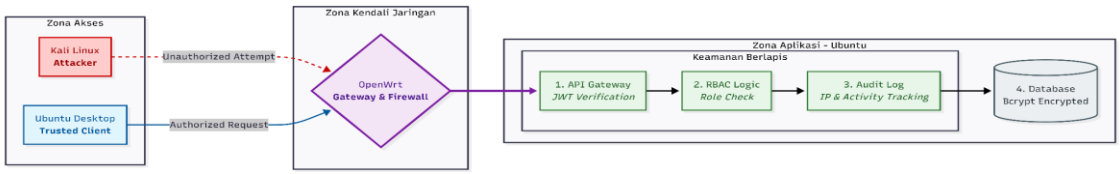
- VLAN/Segment Internal: Menghubungkan Ubuntu Desktop (Client) sebagai node terpercaya dan Kali Linux (Attacker) sebagai node penguji.
- Gateway (OpenWrt): Mengatur lalu lintas data dan berfungsi sebagai garis pertahanan pertama (firewall) sebelum permintaan mencapai server.
- Server Zone: Tempat di mana Ubuntu Server berada, menjalankan layanan API Node.js dan basis data PostgreSQL yang terisolasi dari akses langsung luar tanpa melalui gateway.

2.3.2 Arsitektur Logika Keamanan (Zero Trust Flow)

Diagram logika ini menggambarkan bagaimana prinsip "Never Trust, Always Verify" diterapkan pada setiap lapisan permintaan data:

1. **Identity Layer:** Pengguna melakukan autentikasi di Ubuntu Server. Password diproses menggunakan **Bcrypt** (hashing). Jika valid, server mengirimkan **JWT**.
2. **Authentication Layer:** Setiap permintaan berikutnya wajib menyertakan JWT di header. Middleware `authenticateToken` akan melakukan verifikasi kunci rahasia dan waktu kedaluwarsa (*Session Timeout*).
3. **Authorization Layer (RBAC):** Setelah identitas terverifikasi, middleware `authorizeRole` mengecek apakah peran pengguna (Admin/Editor/Viewer) memiliki izin untuk melakukan aksi tersebut.
4. **Defense Layer (Assume Breach):**
    - **Audit Logging:** Mencatat setiap aktivitas beserta alamat IP yang dideteksi melalui header `x-forwarded-for`.
    - **Account Lockout:** Jika mesin penyerang (Kali Linux) melakukan *brute-force* dan gagal 3 kali, sistem secara otomatis menonaktifkan akun tersebut di tingkat basis data.

2.3.3 Visualisasi Arsitektur (Deskripsi Diagram)



Gambar 1. Visualisasi Arsitektur

BAB III
PERANCANGAN SISTEM

3.1 Topologi

Perancangan topologi sistem ini menggunakan model *Virtual Laboratory* yang diisolasi di dalam satu *Physical Host* menggunakan teknologi virtualisasi **KVM (Kernel-based Virtual Machine)**.

Topologi ini dirancang untuk mensimulasikan lingkungan jaringan nyata yang terdiri dari *Gateway*, *Internal Server*, *Trusted Client*, dan *Untrusted Client (Attacker)*. Berikut adalah detail struktur jaringan yang dibangun:

3.1.1 Arsitektur Jaringan:

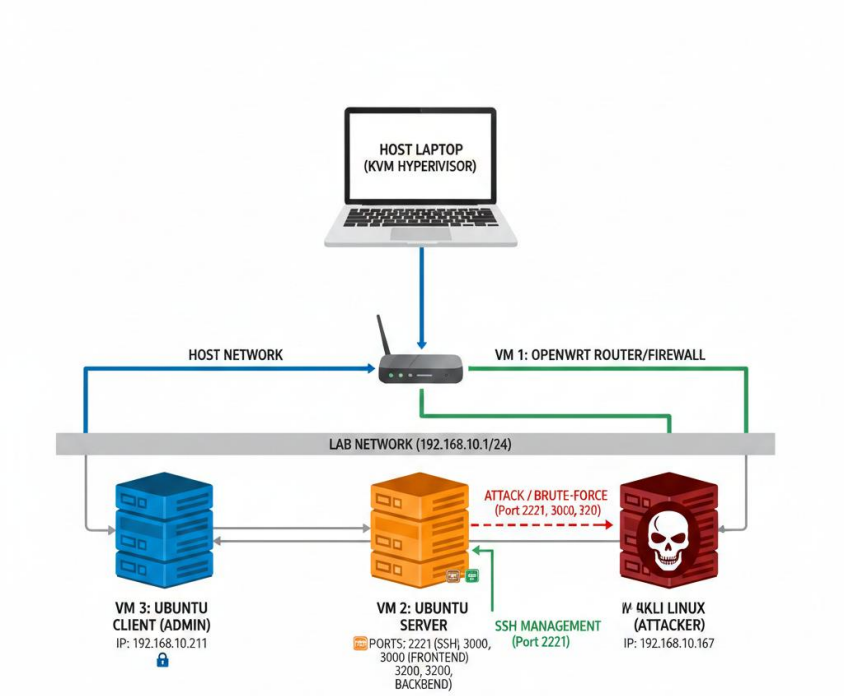
- **Jaringan Host:** Jaringan fisik laptop yang menyediakan akses internet ke seluruh sistem melalui *bridge* atau *NAT*.
- **Jaringan Lab (Internal):** Menggunakan subnet 192.168.10.0/24. Semua VM berada dalam segmen ini agar dapat saling berkomunikasi di bawah pengawasan router.

3.1.2 Detail Entitas Sistem:

- **VM 1 (OpenWRT):** Bertindak sebagai *Firewall* dan *Router* utama. Memiliki dua antarmuka jaringan (WAN dan LAN) untuk mengelola lalu lintas keluar-masuk lab.
- **VM 2 (Ubuntu Server):** Merupakan target sistem yang menjalankan layanan aplikasi pada port non-standar (SSH: 2221, FE: 3000, BE: 3200).
- **VM 3 (Ubuntu Client):** Berperan sebagai administrator legal yang melakukan manajemen server melalui jalur SSH yang aman.
- **VM 4 (Kali Linux):** Berperan sebagai *Attacker* yang melakukan pengujian penetrasi dan simulasi serangan *brute force* terhadap aplikasi di server.

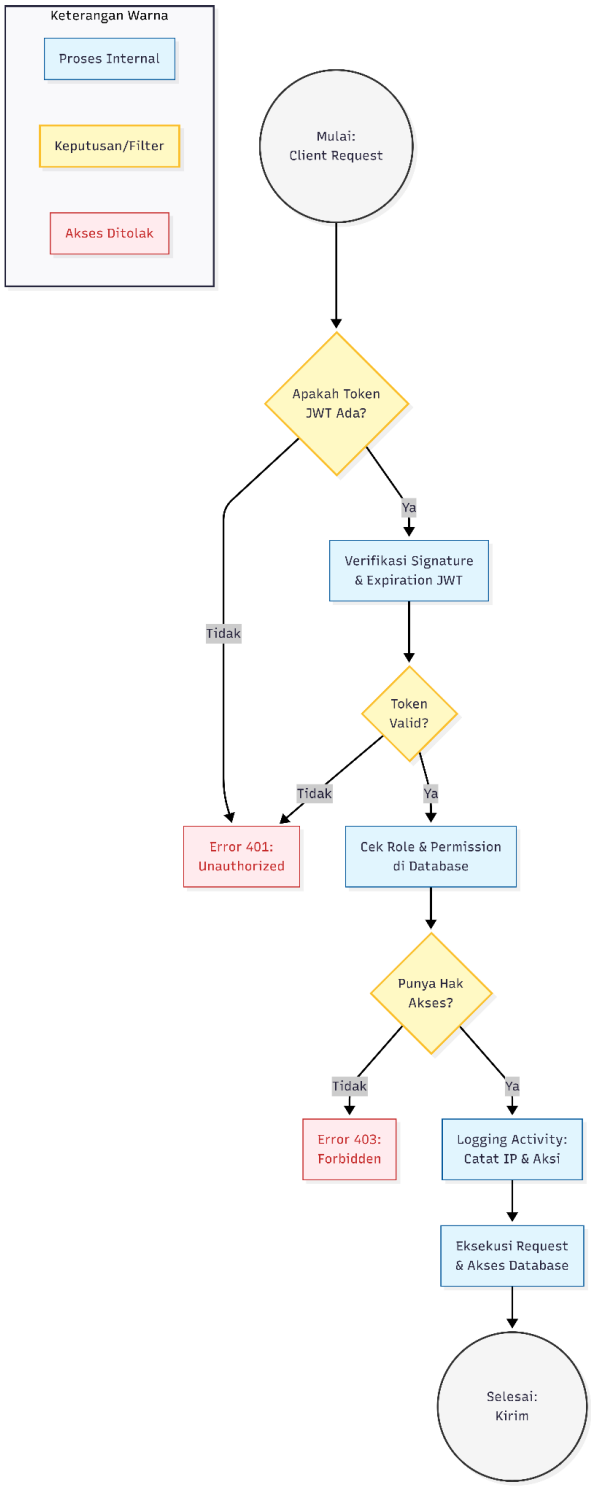
3.1.3 Alur Komunikasi:

Setiap lalu lintas data antar VM dipusatkan melalui OpenWRT, sementara keamanan pada sisi server diperketat dengan penggunaan **UFW (Uncomplicated Firewall)** dan kebijakan limitasi login (3 kali kegagalan) untuk mencegah eksploitasi lebih lanjut.



Gambar 2. Topologi

3.2 Flowchart



Gambar 3. Flowchart

Alur kerja sistem dalam memproses permintaan akses mengikuti logika Zero Trust sebagai berikut:

1. Request: Klien mengirimkan permintaan ke API.
2. Identification: Sistem mengecek keberadaan Token JWT.
3. Authentication (Verify Explicitly): Jika token ada, *middleware* memvalidasi tanda tangan digital dan masa aktifnya.
4. Authorization (Least Privilege): Sistem mengecek peran pengguna (*Admin/Editor/Viewer*) di database untuk menentukan izin akses pada *endpoint* tersebut.
5. Execution & Logging (Assume Breach): Jika diizinkan, aksi dijalankan, IP Address klien dicatat ke Audit Log, dan respons dikirim kembali.

3.3 Pembagian IP dan Resource Server

Pengelolaan sumber daya (*resource*) dilakukan secara terukur menggunakan infrastruktur *Virtual Machine* untuk memastikan performa sistem tetap optimal.

Hostname	Peran	Sistem Operasi	Alamat IP (Static)	CPU	RAM
Gateway-Node	Router/Firewall	Open Wrt	192.168.1.1	1 Core	1 GB
Backend-Node	API & Database	Ubuntu Server	192.168.1.10	2 Core	3 GB
Client-Node	User Interface	Ubuntu Desktop	192.168.1.20	2 Core	5 GB
Attack-Node	Testing/Penetration	Kali Linux	192.168.1.50	2 Core	4 GB

3.4 Tabel Peran, Hak Akses, dan Aturan Keamanan

3.4.1 Matriks Hak Akses (RBAC)

Penetapan hak akses dilakukan secara granular untuk meminimalisir risiko penyalahgunaan wewenang (*privilege escalation*).

Fitur Aplikasi	Endpoint	Method	Viewer	Editor	Admin
Lihat Dokumen	/api/docs	GET	√	√	√
Tambah Dokumen	/api/docs/create	POST	X	√	√
Edit Dokumen	/api/docs/edit	PUT	X	√	√
Hapus Dokumen	/api/docs/delete	DELETE	X	X	√
Audit & User Log	/api/admin/logs	GET	X	X	√

3.4.2 Aturan Keamanan Aplikasi (Security Rules)

Konfigurasi berikut diterapkan pada aplikasi untuk menegakkan prinsip *Assume Breach* dan melindungi sistem dari serangan otomatis:

Parameter Keamanan	Konfigurasi	Tujuan

JWT Timeout	10m (10 Menit)	Membatasi durasi sesi untuk mencegah pembajakan token yang lama.
Account Lockout	3 *Failed Attempts*	Mengunci akun secara otomatis (Status: *Inactive*) jika terdeteksi *brute-force*.
IP Tracking	x-forwarded-for	Merekam alamat IP asli pengguna di setiap entri Audit Log.
Hashing Algorithm	Bcrypt (12 Rounds)	Mengamankan penyimpanan kata sandi agar tidak dapat dibaca secara manual.
Firewall Rule	Port 3000 (HTTP)	Hanya membuka port aplikasi pada jaringan internal yang melalui OpenWrt.

BAB IV
IMPLEMENTASI

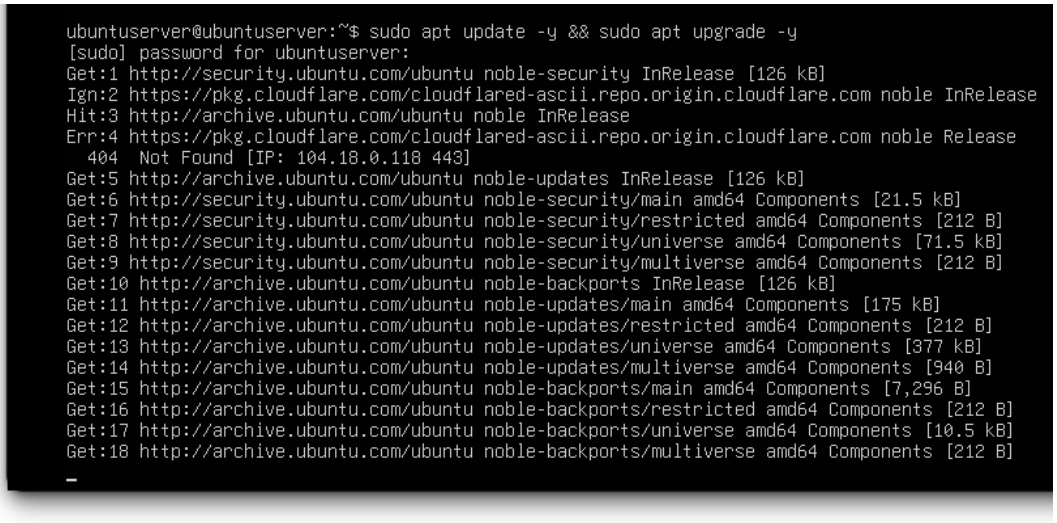
4.1 Konfigurasi Perangkat (Ubuntu Server)

Tahap awal implementasi difokuskan pada penguatan keamanan dasar (*security hardening*) pada level sistem operasi Ubuntu Server sebagai host utama aplikasi.

4.1.1 Update dan Upgrade Repositori

Langkah pertama adalah memastikan seluruh paket keamanan dan repositori sistem berada pada versi terbaru. Hal ini krusial untuk menutup celah kerentanan (*vulnerability*) yang mungkin ada pada paket lama.

- **Perintah:** `sudo apt update && sudo apt upgrade -y`.
- **Proses:** Sistem melakukan pembaruan indeks paket dan mengunduh pembaruan keamanan dari server Ubuntu.



4.1.2 Pengaturan Keamanan Jaringan (UFW Firewall)

Sesuai dengan prinsip *Zero Trust Access Control*, akses ke server harus dibatasi secara ketat hanya pada port yang benar-benar diperlukan oleh layanan aplikasi. Penulis mengaktifkan *Uncomplicated Firewall* (UFW) dengan konfigurasi port sebagai berikut:

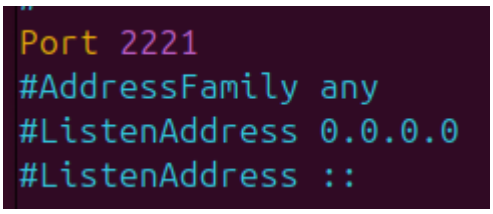
- **Port 2221:** Digunakan untuk akses manajemen administratif melalui SSH.
- **Port 3000:** Digunakan untuk melayani akses antarmuka pengguna (*Frontend*).
- **Port 3200:** Digunakan sebagai jalur komunikasi API backend.
- **Status:** Firewall diaktifkan menggunakan perintah `sudo ufw enable` untuk mulai menegakkan aturan filter jaringan tersebut.

- **Perintah**
    - sudo ufw allow 2221
    - sudo ufw allow 3000
    - sudo ufw allow 3200
    - sudo ufw enable

4.1.3 Penguatan Keamanan Akses (SSH Configuration)

Untuk memitigasi serangan *brute-force* pada tingkat infrastruktur, dilakukan konfigurasi ulang pada layanan SSH melalui file `/etc/ssh/sshd_config`. Langkah-langkah penguatan meliputi:

Perubahan Port Default: Mengubah port standar 22 menjadi **2221** guna mengurangi risiko pemindaian otomatis oleh bot penyerang.



Pembatasan Login Root: Mengatur `PermitRootLogin no` untuk mencegah upaya login langsung menggunakan akun administratif tertinggi secara jarak jauh.

```
#LoginGraceTime 2m
PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```

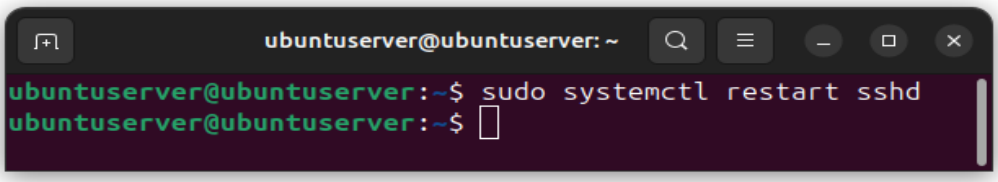
Otentikasi Berbasis Kunci: Mengaktifkan `PubkeyAuthentication yes` agar server mengizinkan login menggunakan kunci kriptografi publik (*Public Key*)

```
PubkeyAuthentication yes
```

Menonaktifkan Kata Sandi: Mengatur `PasswordAuthentication no` untuk menolak semua upaya login yang menggunakan kata sandi biasa. Hal ini memaksa akses hanya dapat dilakukan oleh pemilik kunci privat yang sah, sesuai dengan prinsip *"Verify Explicitly"*.

```
# To disable tunneled clea
PasswordAuthentication no
#PermitEmptyPasswords no
```

Setelah perubahan konfigurasi selesai, layanan SSH dipulihkan kembali menggunakan perintah `sudo systemctl restart sshd` agar seluruh parameter baru dapat diterapkan oleh sistem.



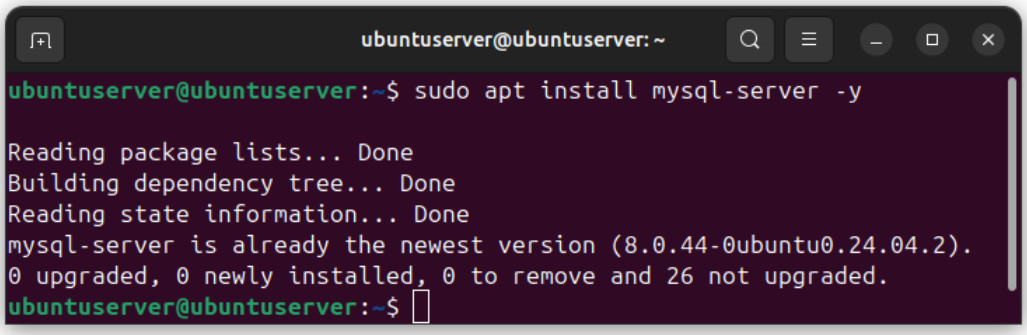
4.2 Instalasi Software

Setelah infrastruktur server dinyatakan siap dan aman, tahap selanjutnya adalah instalasi perangkat lunak pendukung agar aplikasi manajemen dokumen dapat berjalan dengan optimal.

4.2.1 Instalasi dan Konfigurasi Database (MySQL)

MySQL digunakan sebagai sistem manajemen basis data relasional untuk menyimpan data pengguna, dokumen, dan catatan log audit secara terstruktur.

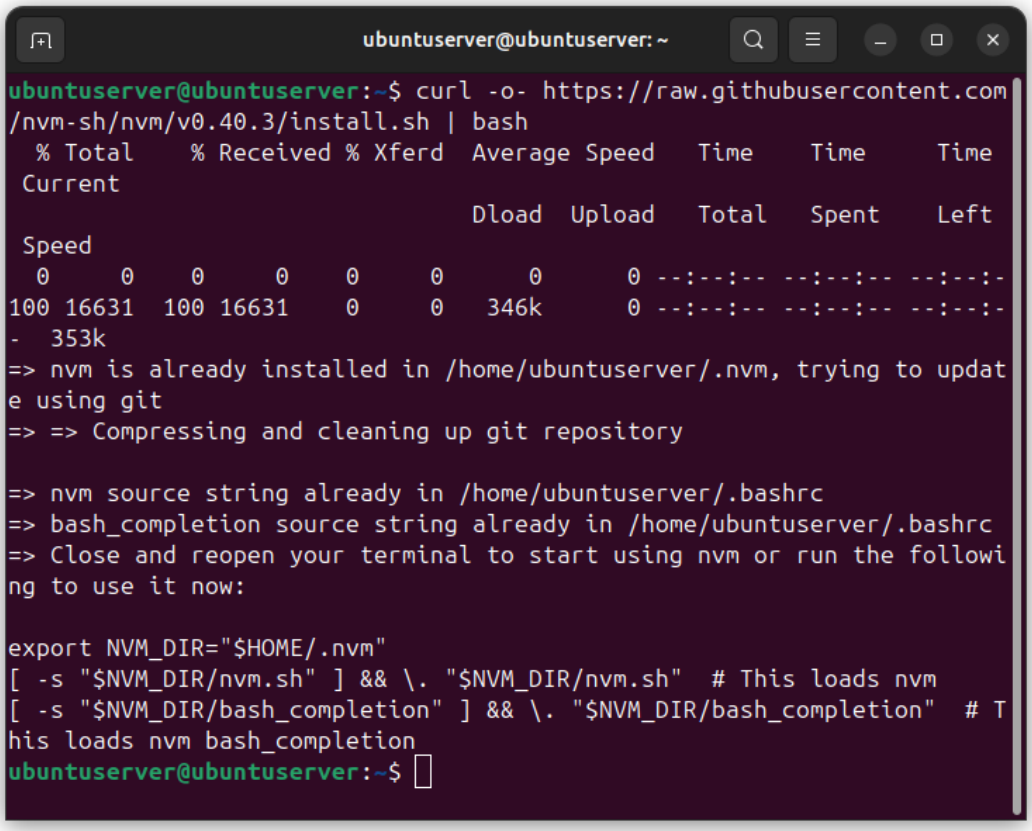
- **Langkah:** Instalasi dilakukan melalui repositori resmi Ubuntu menggunakan perintah `sudo apt install mysql-server -y`.
- **Verifikasi:** Berdasarkan hasil pengujian, sistem menunjukkan bahwa `mysql-server` telah terpasang dengan versi terbaru (8.0.44).
- **Keamanan:** Untuk memitigasi risiko akses ilegal, dilakukan pengamanan tambahan dengan menjalankan skrip `mysql_secure_installation` yang berfungsi menghapus pengguna anonim dan mengamankan akses root database.



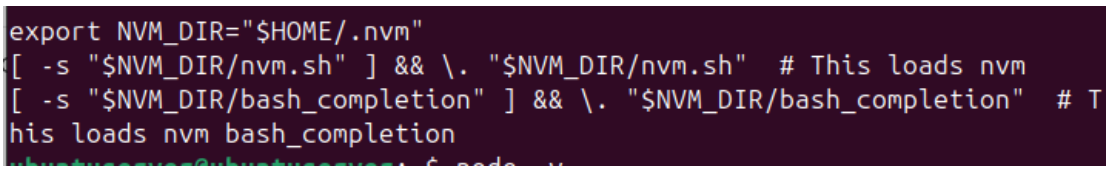
4.2.2 Pengelolaan Node.js menggunakan NVM (Node Version Manager)

NVM diimplementasikan agar penulis memiliki fleksibilitas dalam memilih dan menggunakan versi Node.js tertentu yang stabil. Hal ini sangat penting untuk memastikan kompatibilitas penuh dengan *library* keamanan yang digunakan dalam proyek ini, seperti **Bcrypt** untuk *hashing* dan **JSON Web Token (JWT)** untuk autentikasi.

- **Langkah-langkah:** Proses dimulai dengan mengunduh skrip instalasi NVM langsung dari repositori resminya, dilanjutkan dengan pemasangan Node.js versi terbaru yang stabil.
- **Perintah Instalasi:** curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash

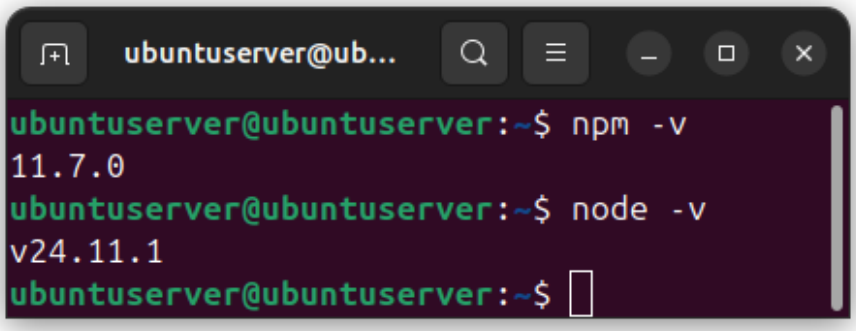


Konfigurasi Environment: Setelah skrip berhasil dijalankan, dilakukan penambahan variabel path pada file profil sistem (`~/.bashrc`) agar perintah `nvm` dapat dikenali secara global oleh terminal.



Verifikasi Instalasi: Untuk memastikan perangkat lunak telah terpasang dengan benar, dilakukan pengecekan versi Node.js dan NPM (Node Package Manager) melalui terminal.

- **Versi Node.js:** v24.11.1
- **Versi NPM:** 11.7.0



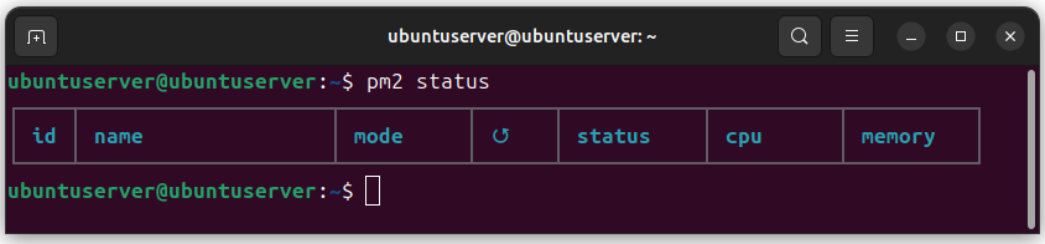
4.2.3 Instalasi PM2 (Process Manager 2)

PM2 diimplementasikan sebagai pengelola proses (*process manager*) untuk menjalankan aplikasi backend di latar belakang secara terus-menerus (*background service*). Penggunaan PM2 sangat krusial dalam arsitektur sistem ini karena mendukung ketersediaan layanan yang berkelanjutan.

- **Relevansi dengan Zero Trust:** Sesuai dengan prinsip *Assume Breach* yang mensyaratkan ketahanan sistem (*system resilience*) , PM2 memastikan bahwa jika terjadi kegagalan aplikasi atau *crash* akibat beban trafik maupun upaya serangan, sistem akan melakukan *restart* secara otomatis untuk menjaga ketersediaan layanan bagi pengguna sah.
- **Langkah Instalasi:** PM2 dipasang secara global di dalam sistem menggunakan Node Package Manager (NPM).

Perintah: Bash npm install pm2 -g

- **Verifikasi Status:** Setelah instalasi selesai, dilakukan pengecekan status untuk memastikan bahwa manajer proses telah siap memantau aplikasi. Berdasarkan pengujian di terminal, perintah `pm2 status` menunjukkan tabel proses yang siap digunakan.



4.3 Script Konfigurasi dan Environment

Tahap ini mencakup pengaturan parameter kritis yang menghubungkan logika aplikasi dengan infrastruktur server, database, serta komunikasi antar-layanan.

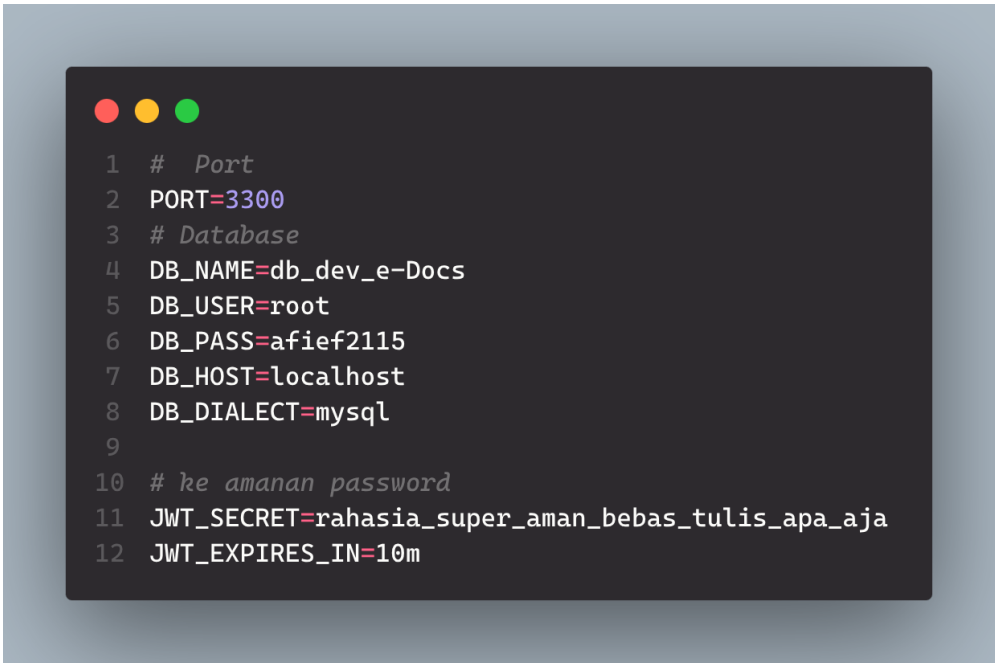
4.3.1 Pengaturan Environment Variables (.env)

Penggunaan file .env merupakan langkah krusial dalam keamanan aplikasi untuk memisahkan data sensitif dari kode sumber. Berdasarkan implementasi pada sistem, berikut adalah konfigurasi pada sisi *Backend* dan *Frontend*:

1. Konfigurasi Backend Server

Pada sisi backend, file .env digunakan untuk mengatur koneksi database dan parameter keamanan utama:

- **Database**: Menghubungkan aplikasi ke MySQL dengan nama database db_e-Docs menggunakan *user* root. Penggunaan dialek mysql memastikan ORM berkomunikasi dengan benar ke server database.
- **Keamanan JWT**: Sesuai dengan prinsip **Verify Explicitly**, parameter JWT_SECRET digunakan sebagai kunci rahasia untuk menandatangani token. Secara *default*, sistem dikonfigurasi dengan JWT_EXPIRES_IN=1d (satu hari), namun sesuai dokumentasi perancangan, nilai ini dapat disesuaikan menjadi 10m (10 menit) untuk membatasi durasi sesi guna mencegah pembajakan token yang lama.



2. Konfigurasi Frontend (Next.js)

Agar *frontend* dapat berinteraksi dengan API, variabel lingkungan dikonfigurasi untuk mengarah ke alamat IP server backend:

- **Base URL**: NEXT_PUBLIC_BASE_URL=http://192.168.10.190:3200/api. Pengaturan ini memastikan setiap permintaan data dari antarmuka pengguna diarahkan ke titik akhir (endpoint) API yang benar pada server produksi.

4.3.2 Konfigurasi Keamanan CORS (Cross-Origin Resource Sharing)

Dalam arsitektur *Zero Trust*, pembatasan akses tidak hanya dilakukan pada level identitas pengguna, tetapi juga pada level sumber permintaan. Konfigurasi CORS diterapkan pada aplikasi untuk memastikan bahwa API hanya menerima permintaan dari sumber yang dipercaya:

- **Origin Whitelist**: Sistem hanya mengizinkan permintaan yang datang dari http://localhost:3000 (pengembangan) dan http://192.168.10.190:3000 (produksi).
- **Methods Control**: Membatasi aksi yang dapat dilakukan melalui HTTP *methods* tertentu seperti GET, POST, PUT, PATCH, dan DELETE sesuai dengan matriks hak akses RBAC yang telah ditentukan.
- **Credentials**: Diatur ke true untuk memungkinkan pertukaran *cookies* atau *authorization headers* yang diperlukan dalam proses validasi JWT pada setiap sesi.

4.3.3 Logika Keamanan Aplikasi (Middleware & Access Control)

Implementasi *Zero Trust* pada level aplikasi dibagi menjadi tiga lapisan utama untuk memastikan setiap permintaan akses terverifikasi secara ketat.

1. Pembuatan Token Otentikasi (JWT Utils)

Sesuai prinsip *Verify Explicitly*, sistem tidak menggunakan sesi berbasis *cookie* tradisional yang rentan, melainkan menggunakan token kriptografi.

- **Fungsi**: Kode pada `generateToken` bertanggung jawab menciptakan identitas digital pengguna yang berisi `id`, `username`, dan `role`.
- **Keamanan**: Menggunakan `JWT_SECRET` yang diambil dari *environment variable* agar kunci enkripsi tidak bocor di kode sumber.

```
import * as jwt from "jsonwebtoken";

interface JWTPayload {
  id: number;
  roleId: number;
  username: string;
}

export function generateToken(payload: JWTPayload): string {
  const SECRET_KEY_STRING = process.env.JWT_SECRET || "rahasia_negara_api";
  const expiresIn = process.env.JWT_EXPIRES_IN || "1d";
  const secretKey = Buffer.from(SECRET_KEY_STRING, "utf8");
  const expiry: string = expiresIn;

  return jwt.sign(payload, secretKey, {
    expiresIn: expiry as jwt.SignOptions["expiresIn"],
  });
}
```

2. Lapisan Verifikasi & Otorisasi (Middleware)

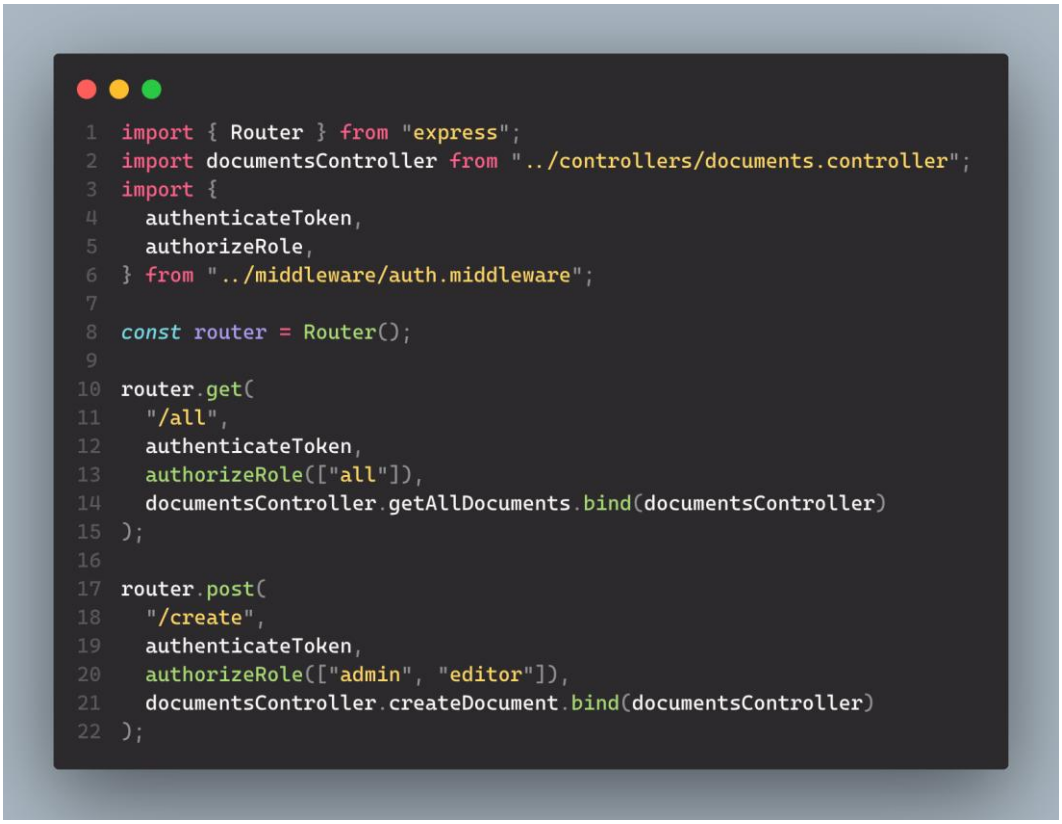
Middleware bertindak sebagai "penjaga gerbang" yang memeriksa setiap *request* sebelum sampai ke database.

- **Authentication Middleware**: Memverifikasi apakah token yang dibawa pengguna sah. Jika token palsu atau kedaluwarsa, akses langsung diputus (Error 401).
- **RBAC Middleware (Role-Based Access Control)**: Menerapkan prinsip *Least Privilege*. Middleware ini mengecek apakah peran (Role) pengguna diizinkan untuk melakukan aksi tersebut (misal: hanya Admin yang bisa menghapus dokumen).

3. Implementasi Kontrol Akses pada Jalur API (Routes)

Penerapan prinsip *Zero Trust* di level kode diakhiri dengan mengintegrasikan middleware keamanan ke dalam skrip *router* aplikasi. Hal ini memastikan bahwa tidak ada satu pun *endpoint* sensitif yang dapat diakses tanpa verifikasi identitas dan pengecekan hak akses secara eksplisit.

- **Verifikasi Berlapis:** Setiap rute dilindungi oleh `authenticateToken` untuk memvalidasi identitas (JWT) dan `authorizeRole` untuk membatasi aksi berdasarkan peran pengguna (Admin/Editor/Viewer).
- **Keamanan Database:** Aplikasi menggunakan variabel lingkungan yang didefinisikan dalam file `.env`, seperti `DB_NAME`, `DB_USER`, dan `DB_PASS`, untuk membangun koneksi database yang aman melalui Sequelize ORM.
- **Audit Logging:** Setiap rute yang berhasil diakses akan memicu pencatatan aktivitas ke dalam *Audit Log*, termasuk perekaman alamat IP asli pengguna untuk keperluan analisis forensik jika terjadi pelanggaran.



```
import { Router } from "express";
import documentsController from "../controllers/documents.controller";
import {
  authenticateToken,
  authorizeRole,
} from "../middleware/auth.middleware";

const router = Router();

router.get(
  "/all",
  authenticateToken,
  authorizeRole(["all"]),
  documentsController.getAllDocuments.bind(documentsController)
);

router.post(
  "/create",
  authenticateToken,
  authorizeRole(["admin", "editor"]),
  documentsController.createDocument.bind(documentsController)
);
```

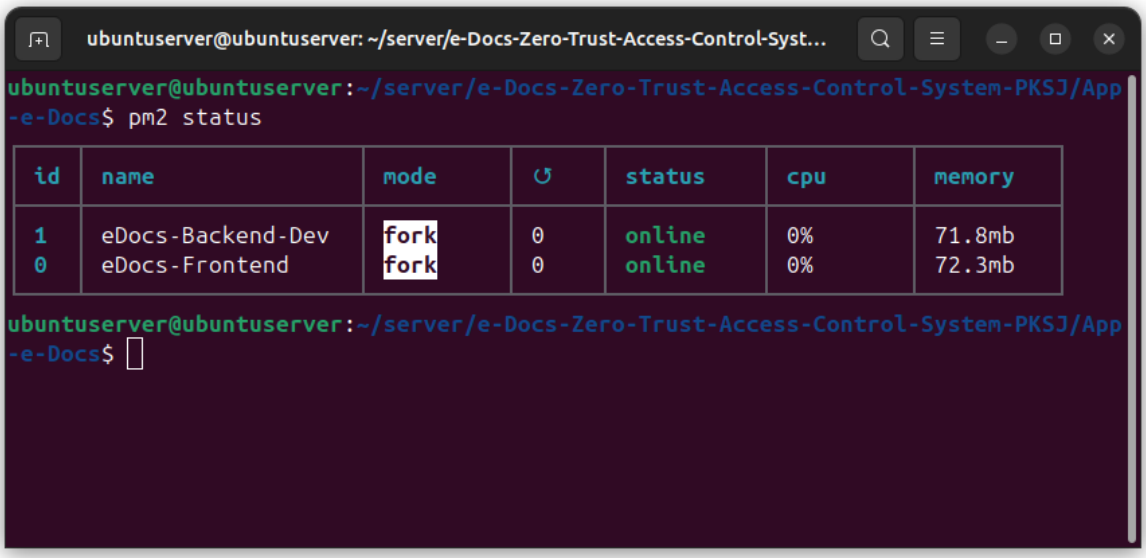
4.4 Deployment dan Manajemen Proses (PM2)

Tahap akhir dari implementasi teknis adalah melakukan deployment agar seluruh layanan dapat berjalan secara permanen di latar belakang Ubuntu Server. Penggunaan

PM2 (Process Manager 2) menjamin resiliensi sistem sesuai prinsip Assume Breach, di mana layanan akan melakukan auto-restart jika terjadi kegagalan proses.
Langkah Eksekusi:

1. **Menjalankan Backend:** Perintah digunakan untuk memulai layanan API dalam mode pengembangan.
    - pm2 start npm --name "eDocs-Backend-Dev" -- run dev
2. **Menjalankan Frontend:** Perintah digunakan untuk memulai antarmuka produksi Next.js.
    - pm2 start npm --name "eDocs-Frontend" -- run start

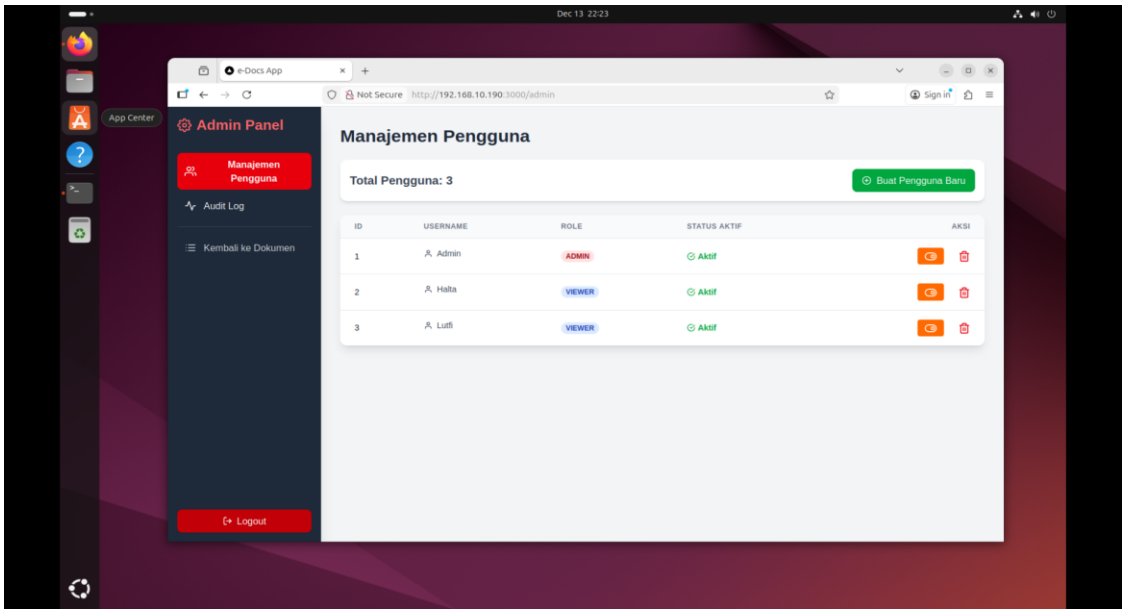
Hasil Akhir (Verifikasi Status): Setelah perintah dijalankan, verifikasi dilakukan melalui perintah `pm2 status`. Hasil menunjukkan bahwa kedua layanan telah aktif secara stabil dengan status **online**, siap melayani permintaan dari pengguna sah (*Trusted Client*) melalui infrastruktur jaringan yang telah dikonfigurasi.



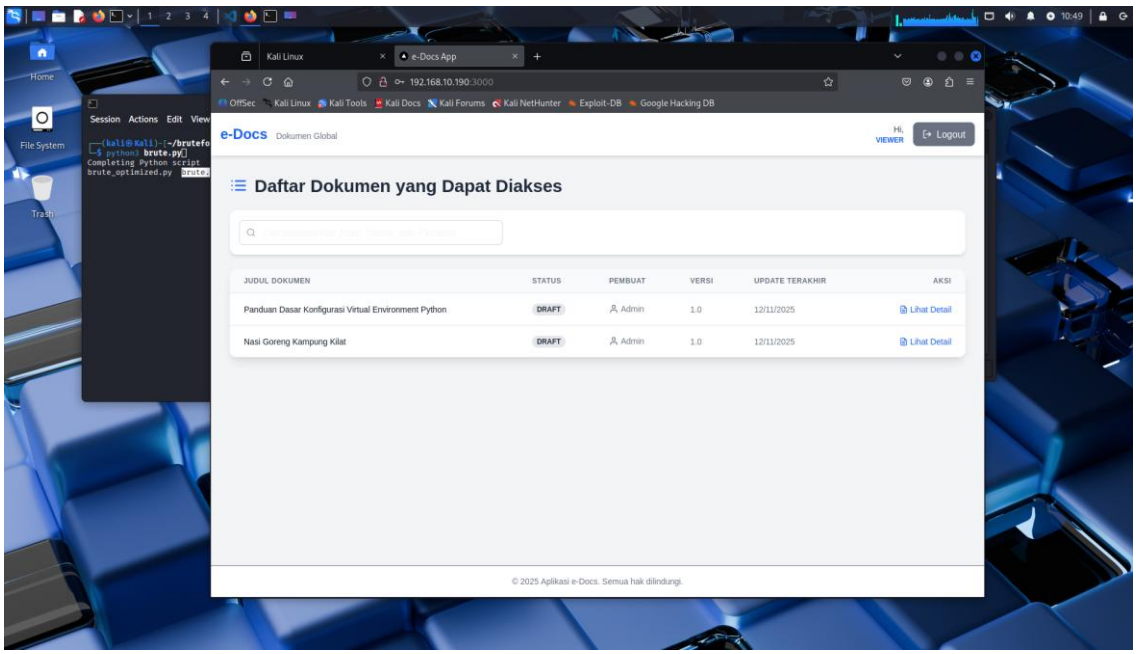
BAB V
PENGUJIAN DAN ANALISIS

5.1 Pengujian Ketahanan Serangan (Brute-Force)

5.1.1 Persiapan Data Serangan dan tampilan login yang akan diserang

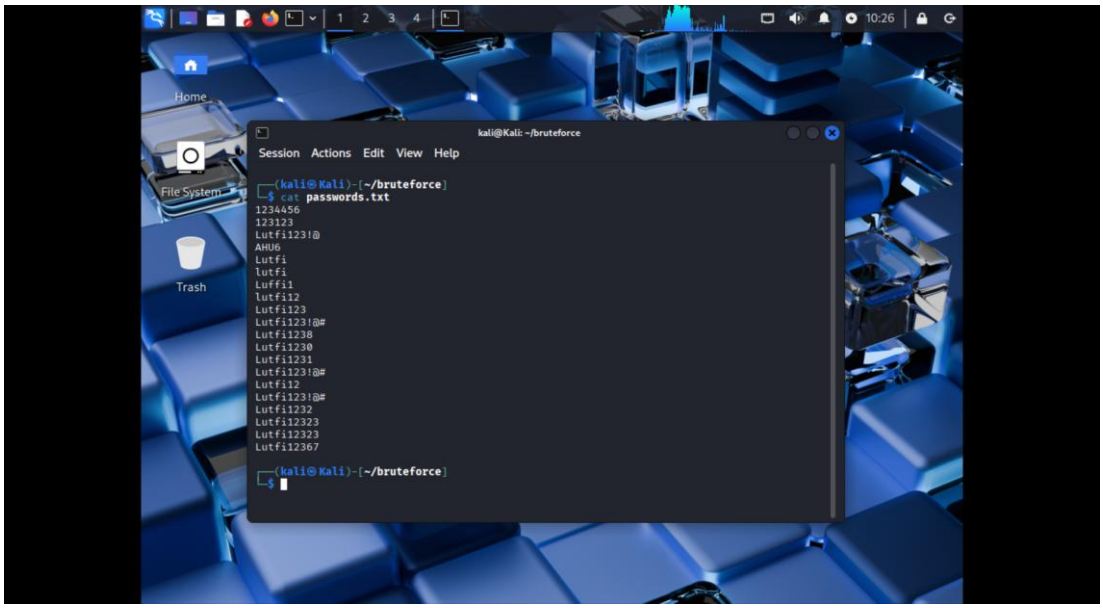


Kondisi awal sistem di mana administrator dapat memantau seluruh pengguna yang terdaftar. Terlihat tiga pengguna dengan status isActive: true, yang menandakan semua akun dalam kondisi aktif dan dapat mengakses sistem sesuai hak aksesnya.



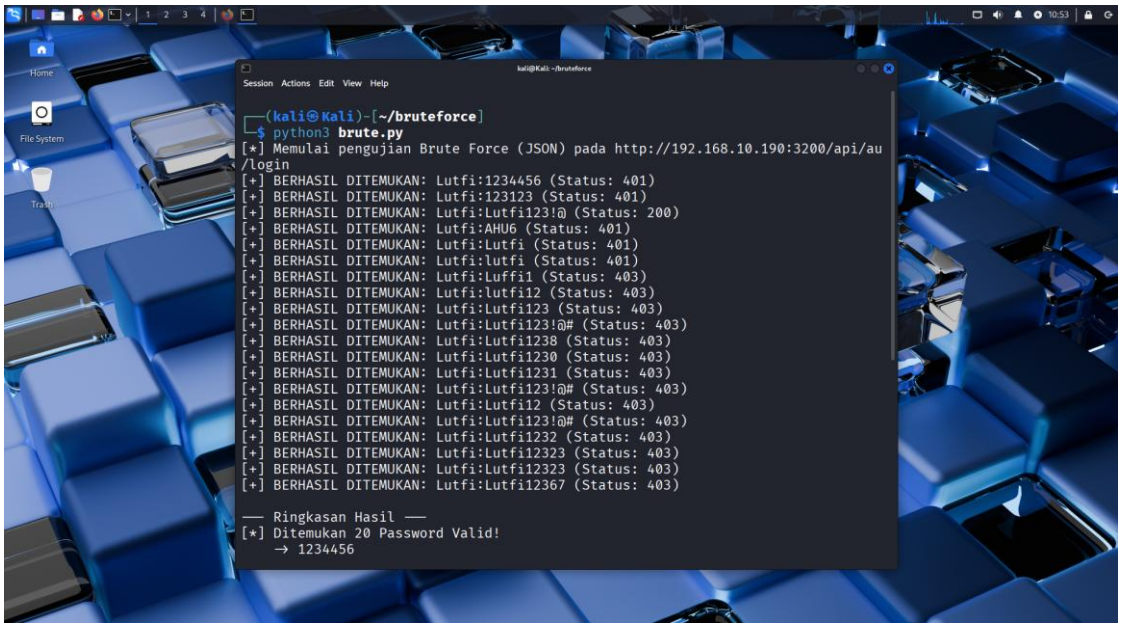
Validasi bahwa sistem berfungsi normal bagi pengguna yang sah. Pengguna "Lutfi" berhasil masuk ke dalam sistem menggunakan kredensial yang benar,

membuktikan bahwa mekanisme *Verify Explicitly* mengizinkan akses bagi identitas yang tervalidasi.



Menampilkan file passwords.txt yang berisi daftar kombinasi kata sandi. Daftar ini digunakan oleh penyerang (dari mesin Kali Linux) untuk melakukan serangan otomatis guna menebak kata sandi pengguna "Lutfi".

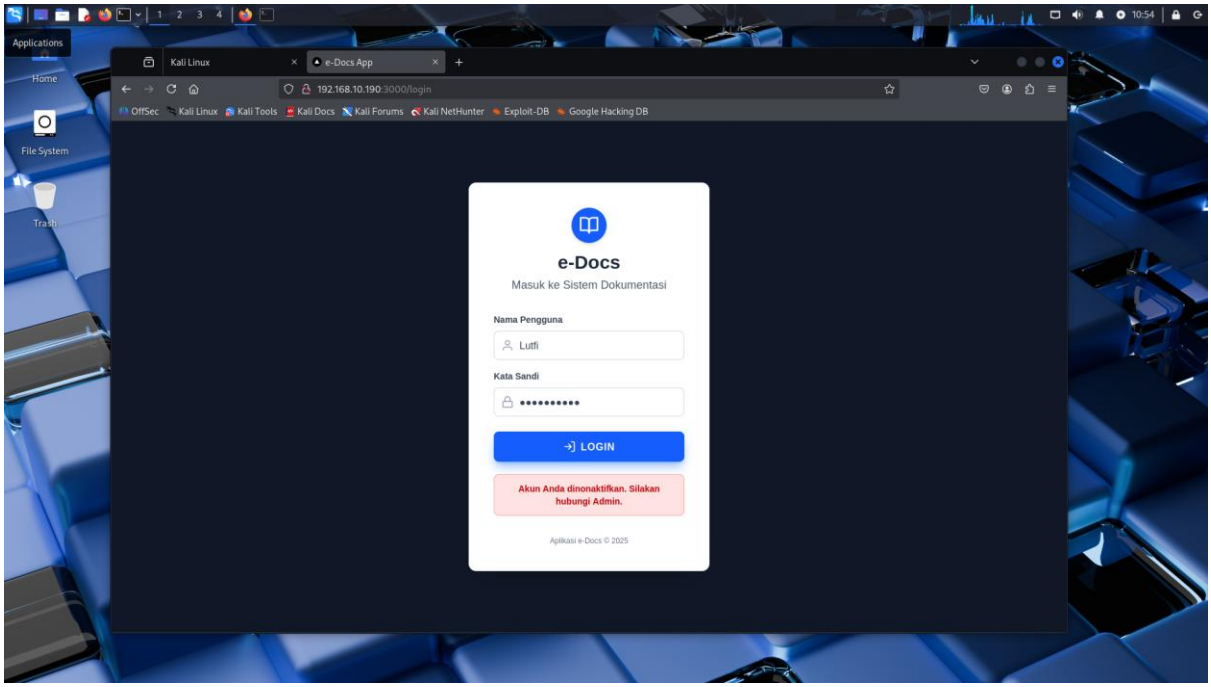
5.1.2 Pelaksanaan dan Hasil Brute-Force



Dibuat Dengan menggunakan python dengan library **RequestS**, Penjelasan Meskipun skrip *brute-force* berhasil menemukan beberapa *password* yang valid (ditunjukkan oleh status **200 OK**), sistem tidak menghentikan serangan secara

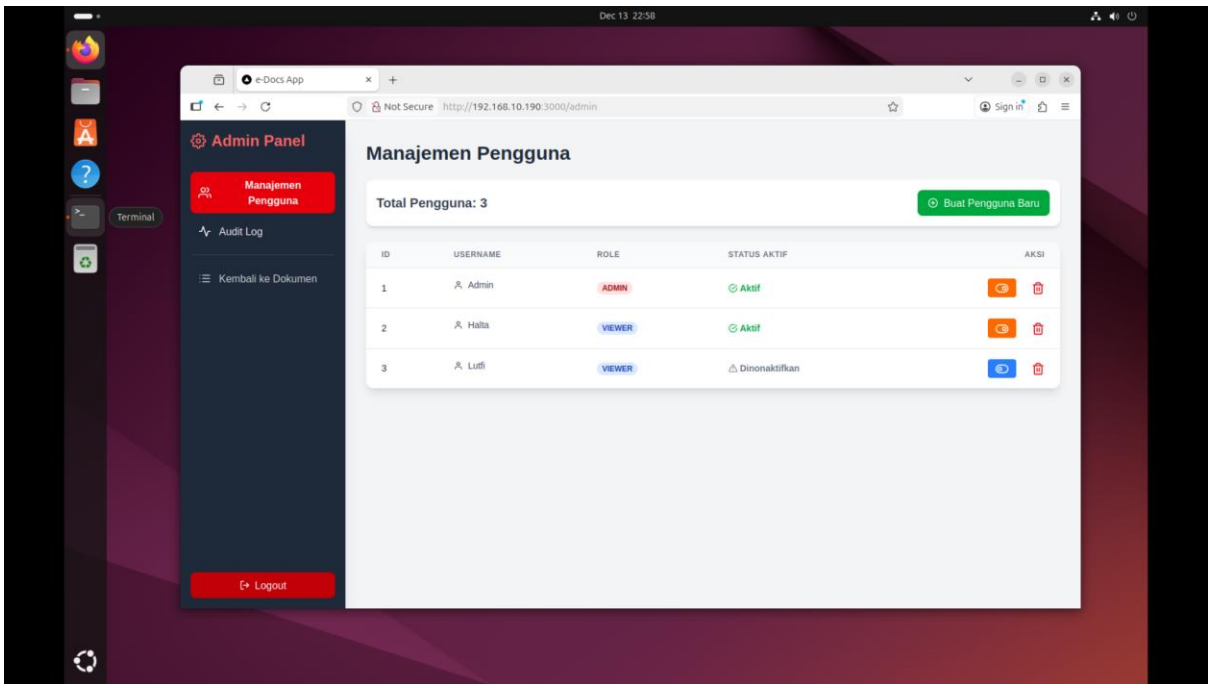
langsung. Karena *lockout* (penguncian akun) diatur setelah **3 kali kegagalan** *login*, dan skrip terus mencoba kombinasi *password* yang salah setelah menemukan yang benar, *limit* kegagalan terlampaui. Akibatnya, akun *user* tersebut dikunci.mnggunkan sistem ketika 3 kali sati salah akan di blok, di porses brute force yang di buat tadi kan gak di stop langsung ketika password nya dapat jadi akan mencobah lagi sampai habis jadi akun ddeng Lutfi akan ke blok

1.3 Verifikasi Account Lockout



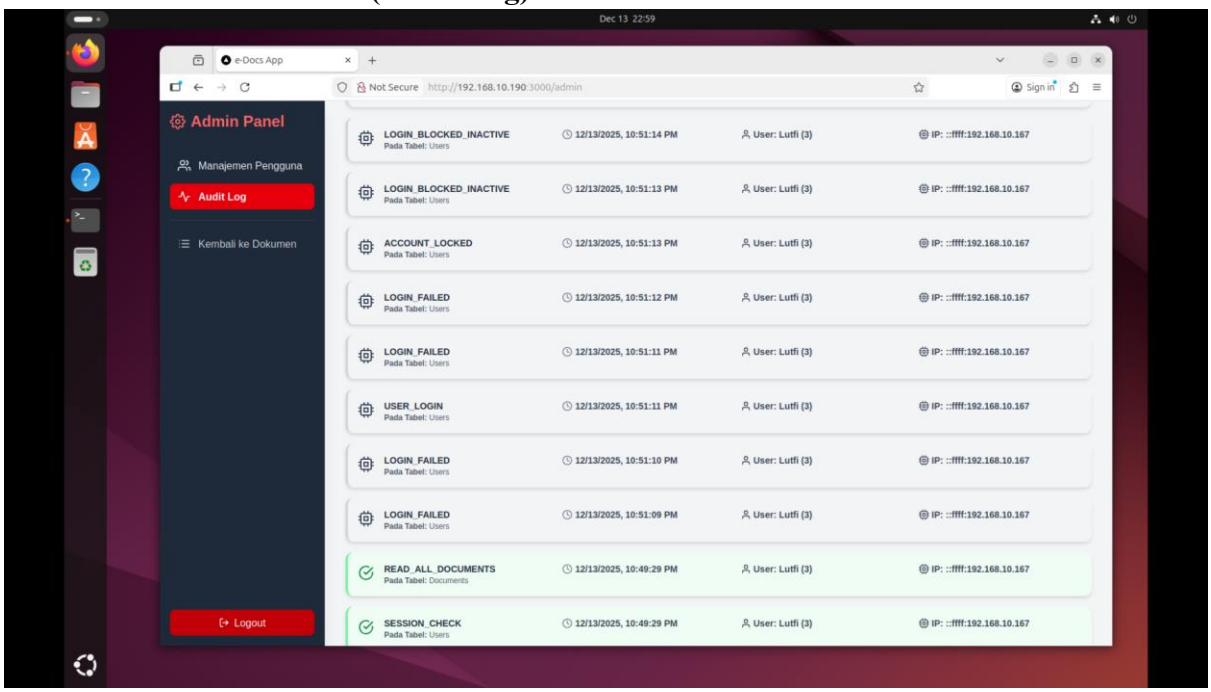
Setelah melampaui ambang batas 3 kali kegagalan, sistem secara otomatis mengaktifkan fitur *Account Lockout*. Gambar ini menunjukkan bahwa meskipun pengguna mencoba login kembali dengan kata sandi yang benar, sistem menampilkan pesan "Akun Anda dikunci", membuktikan pertahanan *Assume Breach* bekerja.

1.4 Konfirmasi Lockout dari Admin Panel



Verifikasi dari sisi administrator. Status pengguna "Lutfi" telah berubah menjadi "Blokir / Non-Aktif" (isActive: false). Hal ini memudahkan Admin dalam mengidentifikasi akun mana yang sedang dalam ancaman atau telah dikompromi.

1.5 Bukti Forensik (Audit Log)



Baris log yang mencatat detil serangan. Informasi yang terekam meliputi *User ID*, *IP Address* penyerang, jenis aktivitas, dan stempel waktu. Log ini adalah bukti krusial untuk analisis forensik pasca-insiden.

5.2 Analisis Hasil Pengujian

Berdasarkan serangkaian uji coba di atas, berikut adalah analisis mendalam mengenai keamanan sistem:

1. Efektivitas Verifikasi (JWT & Bcrypt): Sistem berhasil membedakan antara sesi yang sah dan upaya akses ilegal. Penggunaan *Bcrypt* memastikan bahwa meskipun penyerang mencoba melakukan *brute-force*, mereka membutuhkan waktu dan sumber daya yang besar karena setiap percobaan diverifikasi secara eksplisit oleh server.
2. Ketahanan terhadap Brute-Force (Account Lockout): Mekanisme penguncian akun terbukti menjadi pertahanan paling efektif. Dengan membatasi maksimal 3 kali kegagalan, sistem berhasil menghentikan serangan otomatis sebelum penyerang dapat memanfaatkan kredensial yang mungkin telah mereka temukan di tengah proses.
3. Transparansi dan Audit (Assume Breach): Keberhasilan sistem merekam *IP Address* mesin Kali Linux di dalam *Audit Log* membuktikan bahwa aplikasi siap menghadapi kondisi di mana pertahanan luar telah ditembus. Admin memiliki data yang cukup untuk memblokir IP penyerang pada level *Gateway* (OpenWrt) di masa mendatang.
4. Kesimpulan Keamanan: Secara keseluruhan, sistem telah memenuhi kriteria minimum *Zero Trust Access Control*. Sistem tidak hanya fokus pada "mencegah akses", tetapi juga pada "mendeteksi dan merespon" anomali yang terjadi selama proses autentikasi.

BAB VI
KESIMPULAN

6.1 Kesimpulan

Berdasarkan seluruh tahapan perancangan, implementasi, hingga pengujian penetrasi yang telah dilakukan pada sistem manajemen dokumen elektronik, dapat ditarik kesimpulan sebagai berikut:

- **Keberhasilan Model Zero Trust:** Sistem telah berhasil menggantikan model keamanan tradisional yang berbasis *perimeter* (kepercayaan dalam jaringan) menjadi model keamanan berbasis identitas. Dengan penerapan JWT (JSON Web Token) dan RBAC (Role-Based Access Control), setiap permintaan data diverifikasi secara eksplisit tanpa mempedulikan lokasi jaringan asal pengguna.
- **Efektivitas Pertahanan Aktif:** Mekanisme Account Lockout terbukti menjadi benteng yang efektif dalam menghadapi serangan *brute-force*. Berdasarkan pengujian menggunakan Kali Linux, meskipun penyerang sempat mendapatkan kredensial yang valid di tengah proses otomatisasi, sistem berhasil memutus akses secara otomatis setelah ambang batas 3 kali kegagalan terlampaui.
- **Transparansi Keamanan (Forensik):** Implementasi Audit Log memberikan transparansi penuh terhadap aktivitas sistem. Kemampuan sistem dalam merekam alamat IP penyerang dari mesin Kali Linux menunjukkan bahwa sistem telah memenuhi prinsip *Assume Breach*, di mana setiap aktivitas mencurigakan terdokumentasi untuk kebutuhan analisis forensik.
- **Integrasi Infrastruktur:** Penggunaan OpenWrt sebagai *gateway* telah berhasil memisahkan zona akses pengguna dengan zona server aplikasi, menciptakan lapisan perlindungan jaringan yang stabil untuk mendukung keamanan di level aplikasi.

6.2 Saran Pengembangan

Meskipun sistem telah berhasil mengimplementasikan prinsip dasar *Zero Trust*, terdapat beberapa aspek teknis yang dapat dikembangkan lebih lanjut untuk mencapai standar keamanan tingkat *enterprise*. Berikut adalah rekomendasi peningkatan sistem:

- **Implementasi Protokol HTTPS (SSL/TLS):** Saat ini komunikasi antara klien dan server masih menggunakan protokol HTTP standar. Penggunaan sertifikat SSL/TLS sangat disarankan untuk mengenkripsi data dalam perjalanan (*data-in-transit*). Hal ini krusial dalam arsitektur *Zero Trust* untuk mencegah serangan *Man-in-the-Middle* (MitM) yang dapat mencuri token JWT melalui penyadapan jaringan.
- **Penerapan Multi-Factor Authentication (MFA):** Untuk memperkuat verifikasi identitas (*Verify Explicitly*), sistem perlu menambahkan lapisan autentikasi kedua, seperti kode OTP (One-Time Password) melalui email atau aplikasi autentikator.

MFA memastikan bahwa meskipun kata sandi berhasil ditembus melalui *brute-force*, penyerang tetap tidak dapat mengakses sistem tanpa faktor kepemilikan fisik dari pengguna.

- **Integrasi IPS (Intrusion Prevention System) dan Fail2Ban:** Sistem dapat ditingkatkan dengan menambahkan *Intrusion Prevention System* (IPS) seperti Snort atau Suricata pada *gateway* OpenWrt. Selain itu, integrasi Fail2Ban memungkinkan sistem untuk secara otomatis melakukan *IP Blacklisting* pada level *firewall* jaringan jika terdeteksi aktivitas mencurigakan, sehingga beban kerja mitigasi tidak hanya bertumpu pada lapisan aplikasi.
- **Penggunaan Load Balancer untuk Ketersediaan Tinggi:** Untuk mengantisipasi lonjakan trafik atau serangan *Denial of Service* (DoS), penambahan *Load Balancer* (seperti Nginx atau HAProxy) disarankan untuk mendistribusikan beban kerja ke beberapa instansi server. Hal ini menjamin ketersediaan sistem (*high availability*) sesuai dengan prinsip *Assume Breach* di mana sistem tetap harus resilien meskipun sedang dalam tekanan serangan.

DAFTAR PUSTAKA

ExpressJS. 2024. "Express: Fast, Unopinionated, Minimalist Web Framework for Node.js." Diakses pada 20 Mei 2024. https://expressjs.com/.

Jones, Michael, John Bradley, dan Nat Sakimura. 2015. "JSON Web Token (JWT)." RFC 7519. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7519.

Kindervag, John. 2010. "Build Security Into Your Network's DNA: The Zero Trust Network Architecture." Forrester Research.

Node.js Foundation. 2024. "Node.js Documentation." Diakses pada 20 Mei 2024. https://nodejs.org/en/docs/.

OpenWrt Project. 2024. "OpenWrt Documentation: Firewall and Network Configuration." Diakses pada 20 Mei 2024. https://openwrt.org/docs/.

OWASP Foundation. 2021. "OWASP Top 10:2021 - Broken Access Control & Identification and Authentication Failures." https://owasp.org/www-project-top-ten/.

Provos, Niels, dan David Mazières. 1999. "A Future-Adaptable Password Scheme." Dalam *Proceedings of the USENIX Annual Technical Conference*, 81–92. Berkeley, CA: USENIX Association.

Rose, Scott, Oliver Borchert, Stu Mitchell, dan Sean Connelly. 2020. *Zero Trust Architecture*. NIST Special Publication 800-207. Gaithersburg, MD: National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207.

Sequelize. 2024. "Sequelize: A Modern TypeScript and Node.js ORM." Diakses pada 20 Mei 2024. https://sequelize.org/.

Stallings, William. 2017. *Network Security Essentials: Applications and Standards*. Edisi ke-6. Upper Saddle River, NJ: Pearson.

Walker, Andrew. 2021. *Zero Trust Security: An Enterprise Guide*. Berkeley, CA: Apress.

Wardana. 2022. *Membangun Aplikasi Web Berbasis Node.js dan Database PostgreSQL*. Jakarta: Elex Media Komputindo.

Weidman, Georgia. 2014. *Penetration Testing: A Hands-On Introduction to Hacking*. San Francisco: No Starch Press.