

## **Implementasi Zero Trust Access Control pada Aplikasi Sederhana**

### **kelompok 4**

Luthfi Kurniawan (2201020013)

M. Afief Anugrah (2201020015)

Aditya Firmansyah (2201020018)

Halta Putra Ash Sidiq (2201020092)

### **Minggu 4 : Tambah audit & session timeout**

## 1. Tambah audit

### A. Mendapatkan IP Klien

```
1 private getAddress(req: Request): string {  
2     return (  
3         (req.headers["x-forwarded-for"] as string)?.split(",").shift() ||  
4         req.socket.remoteAddress ||  
5         ""  
6     );  
7 }
```

## Fungsi getAddress: Mendapatkan Alamat IP Klien

Fungsi ini dirancang untuk mendapatkan **alamat IP klien** yang membuat permintaan HTTP dengan **akurasi tertinggi**, terutama dalam lingkungan yang menggunakan **reverse proxy** atau **load balancer**.

### 1. Prioritas Utama: Header **x-forwarded-for**

Fungsi ini pertama-tama memeriksa header **x-forwarded-for**.

- **Tujuan:** Header ini digunakan ketika aplikasi Anda berjalan di belakang **load balancer** (seperti NGINX) atau layanan **cloud**.
- **Isi:** Header ini berisi rantai alamat IP, di mana IP pertama adalah IP publik pengguna yang sebenarnya.
- **Cara Kerja:** Menggunakan `.split(", ").shift()` untuk mengambil IP yang paling awal, yaitu **alamat IP klien asli**.

### 2. Prioritas Kedua (Fallback): **req.socket.remoteAddress**

Jika pengecekan header **x-forwarded-for** **gagal** (misalnya, jika tidak ada proxy, atau jika nilai **x-forwarded-for** dianggap tidak akurat/tidak ada), fungsi akan beralih ke properti ini.

- **Tujuan:** Mendapatkan alamat IP dari koneksi yang terhubung **langsung** ke server Express Anda.
- **Kondisi Penggunaan:** Ini adalah **fallback** yang andal saat tidak ada reverse proxy yang terlibat, atau ketika header yang lebih spesifik tidak tersedia.
-

## B. Implementasi Audit pada Fitur Login

Code Pengambilan Alamat IP Klien

```
1  const ipAddress = this.getAddress(req);
```

Jika proses verifikasi **username** atau **password** gagal:

- **Aksi Audit:** Log audit akan dibuat dengan **actionType**: "LOGIN\_FAILED". Log ini mencatat upaya login yang gagal beserta alamat IP klien.
- **Pesan Respons Klien:** Meskipun log audit dibuat, pesan yang dikirimkan kembali kepada pengguna tetap **generik** untuk alasan keamanan: "Username atau Password salah"

```
1  const isMatch = await bcrypt.compare(password, user.password);
2  if (!isMatch) {
3    await AuditLog.create({
4      userId: user.id,
5      actionType: "LOGIN_FAILED",
6      tableName: "Users",
7      recordId: user.id,
8      ipAddress: ipAddress,
9      details: { reason: "Incorrect Password Attempt" },
10   });
11
12   return sendError(res, "Username atau Password salah", 401);
13 }
```

Jika proses verifikasi **username dan password berhasil** dan pengguna diizinkan masuk:

- **Aksi Audit:** Log audit baru akan dibuat di **auditLog** dengan **actionType: "USER\_LOGIN"**. Log ini mencatat keberhasilan login dan alamat IP klien yang bersangkutan.

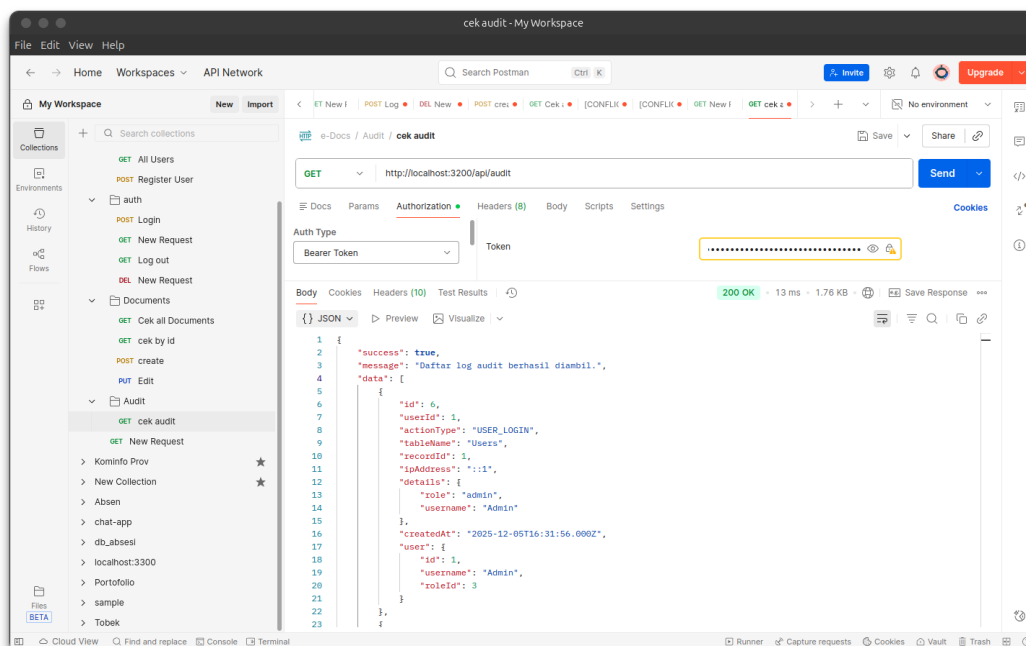
```
1  const role = await Roles.findByPk(user.roleId);
2  const roleName = role ? (role.name as string).toLowerCase() : "viewer";
3
4  const payload = {
5    id: user.id,
6    roleId: user.roleId,
7    roleName: roleName,
8    username: user.username,
9  };
10
11  const token = generateToken(payload);
12
13  await AuditLog.create({
14    userId: user.id,
15    actionType: "USER_LOGIN",
16    tableName: "Users",
17    recordId: user.id,
18    ipAddress: ipAddress,
19    details: { username: user.username, role: roleName },
20  });
21
22  return sendSuccess(
23    res,
24    {
25      token: token,
26      user: {
27        id: user.id,
28        username: user.username,
29        roleId: user.roleId,
30      },
31    },
32    "Login Berhasil"
33  );
```

## C. Implementasi Audit di Pengecekan Sesi

Setiap kali sesi divalidasi melalui endpoint `/api/auth/me`, log audit dengan `actionType: "SESSION_CHECK"` akan dibuat, lengkap dengan ID pengguna dan alamat IP.

```
1 public async getMe(req: Request, res: Response): Promise<Response> {
2   try {
3     const user = (req as any).user;
4     const ipAddress = this.getIpAddress(req);
5
6     await AuditLog.create({
7       userId: user.id,
8       actionType: "SESSION_CHECK",
9       tableName: "Users",
10      recordId: user.id,
11      ipAddress: ipAddress,
12      details: { endpoint: "/api/auth/me" },
13    });
14
15    return sendSuccess(res, user, "Token Valid. User sedang login.");
16  } catch (error) {
17    return sendError(res, "Gagal memuat data user", 500, error);
18  }
19 }
```

## D. Contoh Admin Cek auditLog di Postman



## 2. session timeout

### 1. Fungsi generateToken

Fungsi ini bertanggung jawab untuk membuat JWT yang ditandatangani, di mana masa aktif (*expiration*) token diatur melalui opsi `expiresIn`.

```
1 import * as jwt from "jsonwebtoken";
2
3 interface JWTPayload {
4   id: number;
5   roleId: number;
6   username: string;
7 }
8
9 export function generateToken(payload: JWTPayload): string {
10   const SECRET_KEY_STRING = process.env.JWT_SECRET || "rahasia_negara_api";
11   const expiresIn = process.env.JWT_EXPIRES_IN || "1d";
12
13   const secretKey = Buffer.from(SECRET_KEY_STRING, "utf8");
14
15   const expiry: string = expiresIn;
16
17   return jwt.sign(payload, secretKey, {
18     expiresIn: expiry as jwt.SignOptions["expiresIn"],
19   });
20 }
```

### 2. Penerapan Session Timeout

Meskipun nilai *default* pada kode adalah "1d" (1 hari), Anda telah **menimpunya** (override) menggunakan *file* `.env`.

```
1 JWT_EXPIRES_IN=10m
```

### 3. Tampilan Postman Saat Token Kedaluwarsa

