JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 14 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v14i3.10172

PERANCANGAN *MIDDLEWARE* MODULAR AUTENTIKASI DAN HAK AKSES PADA *RESTFUL API NODE.JS* (STUDI KASUS APLIKASI INSIGHTKU)

Gareth Mu'ammar Dysa[1], Didi Juardi[2], Arip Solehudin[3]

[1,2,3]Universitas Singaperbangsa Karawang Fakultas Ilmu Komputer; Jl. HS. Ronggowaluyo, Telukjambe Timur, Karawang, Jawa Barat 41361; 081218669229

Keywords:
Modular Middleware;
RESTFUL API;
JSON Web Token;
Role-Based Access Control;
Node.js.

Corespondent Email:
garethmuammar@gmail.com

Abstrak. Keamanan sistem *RESTful API* menjadi perhatian penting seiring meningkatnya penggunaan arsitektur layanan web pada aplikasi modern. Penelitian ini merancang dan mengimplementasikan enam *middleware* modular untuk autentikasi dan hak akses pada *RESTful API* berbasis *Node.js* dengan studi kasus aplikasi manajemen keuangan Insightku. Analisis awal menemukan sepuluh permasalahan, termasuk ketiadaan *Role-Based Access Control* (RBAC), duplikasi *middleware authenticateToken* sebanyak 18 kali, tidak adanya proteksi *brute force*, validasi input terpusat, dan *audit trail*. Menggunakan metode *Research and Development* (R&D) dengan model ADDIE serta pengujian *Black Box Testing*, dikembangkan enam *middleware*: *authenticateToken*, *checkRole*, *errorHandler*, *rateLimiter*, *validateRequest*, dan *requestLogger*. Pengujian pada 14 skenario menunjukkan tingkat keberhasilan 100% dengan *overhead response time* 2–22 ms. Duplikasi kode berkurang 94,4% dan solusi yang dikembangkan mampu memitigasi 9 dari 10 ancaman OWASP API Security Top 10. Validasi menghasilkan skor rata-rata 4,67/5,00 (Sangat Baik).

Abstract. *The security of RESTful API systems has become increasingly important with the growing adoption of web service architectures in modern applications. This study designs and implements six modular middlewares for authentication and access control in a Node.js-based RESTful API, using the Insightku personal finance application as a case study. Initial analysis identified ten issues, including the absence of Role-Based Access Control (RBAC), 18 duplicated authenticateToken middleware instances, and the lack of brute force protection, centralized input validation, and audit trails. Using the Research and Development (R&D) methodology with the ADDIE model and Black Box Testing, six middlewares were developed: authenticateToken, checkRole, errorHandler, rateLimiter, validateRequest, and requestLogger. Testing across 14 scenarios achieved a 100% success rate with a response time overhead of 2–22 ms. Code duplication was reduced by 94.4%, and the solution mitigates 9 of the 10 OWASP API Security Top 10 threats. Validation resulted in an average score of 4.67/5.00 (Very Good).*

1. PENDAHULUAN

Dalam era digital saat ini, pertukaran data dan komunikasi antar sistem aplikasi semakin kompleks dan terbuka. Banyak aplikasi modern dibangun menggunakan arsitektur RESTful API sebagai tulang punggung komunikasi antara frontend dan backend [1]. Namun, keterbukaan ini juga membawa risiko keamanan yang signifikan OWASP API Security Top 10 mendokumentasikan sepuluh risiko keamanan API paling kritis yang mengancam sistem berbasis web modern [2].

JSON Web Token (JWT) telah ditetapkan sebagai standar terbuka RFC 7519 dan menjadi mekanisme autentikasi stateless yang paling banyak diadopsi pada sistem RESTful API modern [3]. Namun JWT saja tidak cukup tanpa struktur middleware yang terorganisir, logika autentikasi bercampur dengan logika bisnis sehingga meningkatkan risiko celah keamanan [4].

Permasalahan ini ditemukan secara langsung pada sistem Insightku, yaitu aplikasi manajemen keuangan berbasis mobile yang dikembangkan bersama tim dalam program Bangkit Academy 2024. Analisis kondisi awal mengidentifikasi sepuluh permasalahan struktural yang mencakup: tidak adanya field role di tabel Users, JWT payload tanpa informasi role, middleware authenticateToken yang diimpor dan dipasang secara manual sebanyak 18 instance di enam file route yang berbeda, tidak ada RBAC, tidak ada proteksi brute force, tidak ada validasi input terpusat, dan tidak ada audit trail.

Penelitian ini bertujuan: (1) merancang dan mengimplementasikan middleware modular JWT dan RBAC yang terpisah dan reusable; (2) membuktikan efektivitas middleware dalam menyaring akses tidak sah pada sistem Insightku. Seluruh kode middleware dipublikasikan sebagai repository publik di GitHub dengan lisensi MIT sebagai kontribusi terbuka bagi komunitas pengembang Node.js.

2. TINJAUAN PUSTAKA

2.1. JSON Web Token (JWT)

JSON Web Token (JWT) adalah standar terbuka (RFC 7519) yang mendefinisikan cara yang ringkas dan mandiri untuk mentransmisikan informasi antar pihak sebagai objek JSON [3]. JWT terdiri dari tiga bagian: header, payload, dan signature. Algoritma HMAC SHA-256 digunakan untuk menandatangani token sehingga dapat diverifikasi integritasnya. Sifat stateless JWT menjadikannya pilihan utama untuk autentikasi pada sistem RESTful API [5].

2.2. Role-Based Access Control (RBAC)

RBAC adalah model kontrol akses yang mengatur izin berdasarkan peran pengguna dalam suatu sistem [6]. Sandhu et al. (1996) mendefinisikan RBAC sebagai pendekatan yang efisien untuk mengelola hak akses dalam skala besar. Dalam konteks RESTful API, RBAC diimplementasikan melalui middleware yang memverifikasi peran pengguna sebelum mengizinkan akses ke endpoint tertentu [7].

2.3. Middleware Express.js

Middleware dalam Express.js adalah fungsi yang memiliki akses ke objek request (req), response (res), dan fungsi next() dalam siklus permintaan HTTP [8]. Middleware modular memisahkan setiap tanggung jawab keamanan ke dalam komponen independen yang dapat dipasang secara berurutan (pipeline).

2.4. OWASP API Security Top 10

OWASP API Security Top 10 (2023) mendokumentasikan sepuluh risiko keamanan API paling kritis [2]. Ancaman utama yang relevan dengan penelitian ini meliputi API1 (Broken Object Level Authorization), API2 (Broken Authentication), API4 (Unrestricted Resource Consumption), dan API5 (Broken Function Level Authorization).

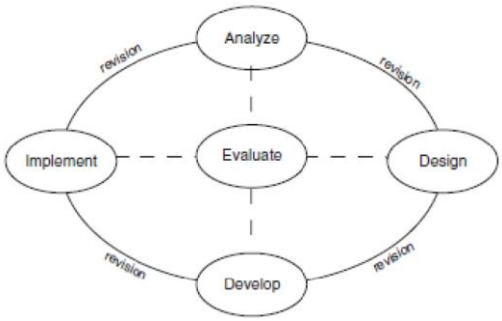
2.5. Penelitian Terdahulu

Beberapa penelitian terdahulu telah mengkaji keamanan RESTful API menggunakan JWT dan RBAC. Cahyono et al. (2022) [1] mengimplementasikan Passport.js untuk autentikasi pada Node.js. Dalimunthe et al. (2023) [5] meneliti penerapan JWT dengan algoritma HMAC-SHA512 untuk keamanan sesi API. Edy et al. (2019) [4] mengkaji pengamanan RESTful API menggunakan JWT pada aplikasi Sales Order. Utama & Indriyanti (2023) [7] menerapkan JWT untuk mengamankan API berbasis mobile di lingkungan akademik. Gunawan & Rahmatulloh (2019) [10] mengimplementasikan JWT untuk autentikasi pada arsitektur RESTful yang interoperabel. Prasetyo & Suharjo (2025) [17] yang diterbitkan di jurnal JITET mengintegrasikan JWT dengan enkripsi AES 256-bit pada backend API berbasis Golang, membuktikan efektivitas JWT sebagai komponen autentikasi dalam sistem keamanan berlapis. Berbeda dengan penelitian-penelitian tersebut, penelitian ini mengintegrasikan enam

middleware sekaligus, diterapkan pada sistem produksi aktif, divalidasi menggunakan standar OWASP, dan dipublikasikan sebagai open source.

3. METODE PENELITIAN

Penelitian ini menggunakan metodologi Research and Development (R&D) yang mengacu pada Sugiyono (2019) [11], dengan model pengembangan ADDIE (Branch, 2009) [12] yang terdiri dari lima fase: Analysis, Design, Development, Implementation, dan Evaluation. Pengujian menggunakan metode Black Box Testing (Myers et al., 2011) [9].



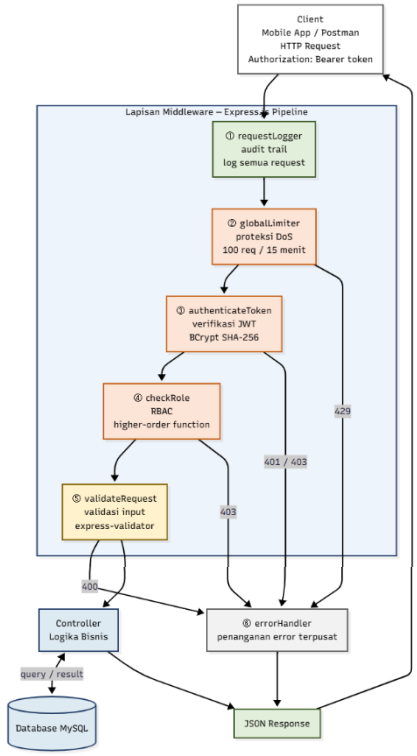
Gambar 1. Tahapan Model *ADDIE*

3.1. Analysis

Fase Analysis mencakup identifikasi masalah pada sistem Insightku dan studi literatur terkait JWT, RBAC, Express.js, dan OWASP API Security Top 10. Hasil analisis mengidentifikasi 10 permasalahan struktural yang dikelompokkan dalam tiga kategori: (1) Model dan Database tidak ada field role dan mekanisme pencatatan peran; (2) Autentikasi JWT payload tanpa role dan duplikasi middleware 18 instance; (3) Otorisasi dan Keamanan tidak ada RBAC, errorHandler, validateRequest, requestLogger, dan proteksi DoS.

3.2. Design

Fase Design mencakup perancangan arsitektur enam middleware modular dengan prinsip fail-fast request yang tidak memenuhi persyaratan ditolak sedini mungkin.Urutan middleware dirancang sebagai: (1) requestLogger → (2) globalLimiter → (3) authenticateToken → (4) checkRole → (5) validateRequest → (6) Controller, dengan errorHandler sebagai penangkap error di seluruh lapisan.



Gambar 2. Arsitektur Berlapis Pada *Middleware* Modular

3.3. Development

Fase Development mencakup implementasi keenam middleware menggunakan Node.js v22, Express.js, MySQL dengan Sequelize, dan library npm: jsonwebtoken (JWT), express-rate-limit (rate limiting), dan express-validator (validasi input). Lingkungan pengembangan: Windows 11, VS Code, localhost:5000.

3.4. Implementation & Evaluation

Fase Implementation mencakup Black Box Testing terhadap 14 skenario menggunakan Postman, mencakup seluruh middleware yang dikembangkan. Fase Evaluation mencakup analisis keakuratan, response time, modularitas, pemetaan OWASP, dan validasi oleh Project Manager Insightku menggunakan kuesioner Likert 15 pernyataan.

4. HASIL DAN PEMBAHASAN

4.1. Implementasi Enam Middleware Modular

Keenam middleware berhasil diimplementasikan dan diintegrasikan pada server.js sistem Insightku. Tabel 1 merangkum

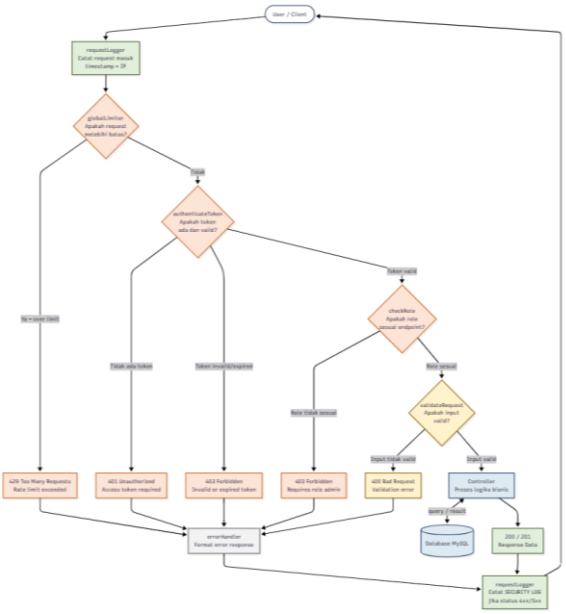
fungsi, library, dan pola implementasi masing-masing middleware.

Tabel 1. Ringkasan Implementasi Enam *Middleware* Modular

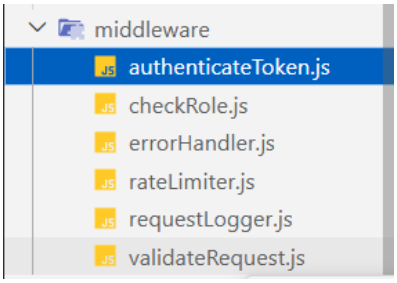
Middleware	Fungsi	Library
Authenticate Token	Verifikasi JWT Bearer Token	Jsonweb token
checkRole	Kontrol akses RBAC per Route	native
errorHandler	Penanganan error terpusat	native
rateLimiter	Proteksi brute force dan DoS	Express-rate-limit
Validate Request	Validasi input request	Express-validator
Request Logger	Audit Trail request dan response	native

authenticateToken menggunakan library jsonwebtoken untuk memverifikasi JWT Bearer token pada setiap request ke endpoint terproteksi. checkRole menggunakan pola higher-order function checkRole('admin') menghasilkan middleware yang hanya mengizinkan role admin, sedangkan checkRole('user','admin') mengizinkan keduanya. errorHandler menangani enam kategori error dengan format respons JSON yang konsisten. rateLimiter menyediakan tiga tingkatan: loginLimiter (5 req/15 menit), authLimiter (10 req/15 menit), dan globalLimiter (100 req/15 menit). validateRequest menggunakan express-validator dengan delapan skema validasi. requestLogger mencatat seluruh traffic dengan label SECURITY LOG untuk respons 4xx/5xx.

Middleware CORS dikonfigurasi untuk membatasi akses hanya dari origin yang telah terdaftar pada environment aplikasi. Sanitasi input diterapkan pada setiap request guna mencegah serangan seperti SQL Injection dan Cross-Site Scripting (XSS). Sistem autentikasi mendukung mekanisme refresh token untuk mempertahankan sesi pengguna tanpa perlu login ulang secara berulang. Seluruh aktivitas autentikasi dan otorisasi dicatat dalam log terpisah untuk keperluan audit serta analisis keamanan. Kombinasi middleware tersebut membentuk lapisan keamanan berlapis yang meningkatkan perlindungan terhadap akses tidak sah dan penyalahgunaan API.



Gambar 3. *Flowchart* Alur Pemrosesan *Request* pada Sistem *Middleware* Modular



Gambar 4. Struktur Folder *Middleware* Modular

4.1.1. Implementasi authenticateToken.js

Middleware authenticateToken merupakan komponen inti yang bertanggung jawab memverifikasi keabsahan JWT pada setiap request ke endpoint yang terproteksi. Middleware ini membaca token dari header Authorization dengan format Bearer [token], memverifikasinya menggunakan secret key yang tersimpan di environment variable JWT_SECRET, dan menyimpan payload yang telah didekode ke dalam objek req.user untuk dapat digunakan oleh middleware selanjutnya dalam rantai pemrosesan.

Gambar 5. Implementasi Kode *authenticateToken.js*

Alur kerja middleware authenticateToken secara detail adalah sebagai berikut:

1. Middleware membaca nilai header Authorization dari objek req.headers pada setiap request yang masuk ke endpoint terproteksi.
2. Token diekstrak dari format string 'Bearer [token]' menggunakan metode split(' ') pada karakter spasi, kemudian diambil elemen indeks ke-1.
3. Jika header Authorization tidak ada atau token tidak dapat diekstrak, middleware langsung mengembalikan respons HTTP 401 Unauthorized dengan pesan 'Access token required' tanpa memanggil next().
4. Token yang berhasil diekstrak diverifikasi menggunakan fungsi jwt.verify() dengan parameter JWT_SECRET dari process.env. Jika token tidak valid (JsonWebTokenError) atau sudah melewati masa berlaku (TokenExpiredError), verifikasi gagal.
5. Jika verifikasi berhasil, payload token yang telah didekode yang memuat userId, username, dan role disimpan ke req.user. Fungsi next() kemudian dipanggil untuk meneruskan request ke middleware berikutnya.
6. Jika terjadi error pada proses verifikasi, error tersebut diteruskan ke middleware errorHandler melalui next(err) untuk ditangani secara terpusat.

Perubahan kritis yang membedakan implementasi ini dengan kondisi awal adalah pada baris JWT payload di controller users.js. Sebelumnya, payload JWT hanya memuat { userId, username }, sehingga informasi role tidak tersedia saat checkRole dieksekusi. Setelah modifikasi, payload menjadi { userId, username, role }, memungkinkan middleware checkRole membaca role langsung dari token tanpa perlu query tambahan ke database pada setiap request.

4.1.2. Implementasi checkRole.js

Middleware checkRole mengimplementasikan mekanisme Role-Based Access Control (RBAC) menggunakan pola higher-order function, sebuah fungsi yang menerima parameter fungsi middleware baru. Pendekatan ini memberikan fleksibilitas tinggi yang diizinkan dapat berbeda-beda untuk setiap endpoint tanpa perlu membuat file middleware yang berbeda.

Gambar 6. Implementasi Kode *checkRole.js*

Cara kerja middleware checkRole dimulai ketika fungsi checkRole(...allowedRoles) dipanggil pada definisi route. Fungsi ini menerima satu atau lebih string nama role yang diizinkan sebagai argumen, kemudian mengembalikan fungsi middleware Express standar yang menerima parameter (req, res, next). Saat request masuk, middleware pertama memeriksa keberadaan req.user, jika tidak ada, berarti authenticateToken belum dijalankan dan middleware mengembalikan 401. Selanjutnya, middleware memeriksa apakah req.user.role termasuk dalam array allowedRoles menggunakan metode includes(). Jika tidak cocok, dikembalikan respons 403 Forbidden disertai pesan yang menyebutkan role yang diizinkan secara eksplisit.

Tabel 2. Konfigurasi *checkRole* pada Setiap *Endpoint*

Endpoint	Konfigurasi checkRole	Keterangan
GET /api/admin/users	checkRole('admin')	Hanya dapat diakses oleh pengguna dengan role admin
GET /api/dashboard	checkRole('user','admin')	Dapat diakses oleh semua pengguna terauthentikasi

4.1.3. Implementasi errorHandler.js

Middleware errorHandler menggantikan penanganan error yang sebelumnya tersebar tidak konsisten di 6 controller menjadi satu titik penanganan terpusat. Posisi middleware ini dalam stack Express.js sangat kritis, ia harus dipasang sebagai middleware terakhir di server.js, setelah seluruh route didefinisikan, karena Express mengenali middleware dengan 4 parameter (err, req, res, next) sebagai error handler.

middleware > errorHandler.js > errorHandler

```
dan mengembalikan response error dengan format yang konsisten.
*/
const errorHandler = (err, req, res, next) => {
    console.error(`[ERROR] ${req.method} ${req.url} →`, err.message);

    // Error dari JWT – token tidak valid
    if (err.name === 'JsonWebTokenError') {
        return res.status(403).json({
            error: 'Invalid token',
            message: err.message
        });
    }

    // Error dari JWT – token sudah kadaluarsa
    if (err.name === 'TokenExpiredError') {
        return res.status(403).json({
            error: 'Token expired',
            message: 'Silakan login kembali'
        });
    }

    // Error validasi Sequelize
    if (err.name === 'SequelizeValidationError') {
        return res.status(400).json({
            error: 'Validation error',
            message: err.errors.map(e => e.message).join(', ')
        });
    }

    // Error unique constraint Sequelize (misal email sudah ada)
    if (err.name === 'SequelizeUniqueConstraintError') {
        return res.status(400).json({
            error: 'Data sudah ada',
            message: err.errors.map(e => e.message).join(', ')
        });
    }
```

Gambar 7. Implementasi Kode *errorHandler.js*

Middleware errorHandler menangani enam kategori error secara spesifik dengan status kode HTTP yang sesuai. Tabel 3 merinci setiap kategori error yang ditangani beserta status kode dan pesan yang dikembalikan.

Tabel 3. Kategori Error yang Ditangani *errorHandler.js*

Jenis Error	Status	Pesan Respons
JsonWebTokenError	403	Invalid token, token tidak valid atau telah dimanipulasi
TokenExpiredError	403	Invalid token, token telah melewati masa berlaku
SequelizeValidationError	400	Validation Error, daftar field yang tidak memenuhi constraint database
SequelizeUniqueConstraintError	400	Data sudah terdaftar, pelanggaran constraint UNIQUE pada field tertentu
Error dengan statusCode custom	4xx/5xx	Pesan sesuai statusCode yang di-set oleh controller
Error umum lainnya	500	Internal Server Error, error tidak terduga dari lapisan manapun

4.1.4. Implementasi rateLimiter.js

Middleware rateLimiter diimplementasikan menggunakan library express-rate- limit yang bekerja dengan cara melacak jumlah request dari setiap IP address dalam jendela waktu tertentu. Jika jumlah request melampaui batas yang dikonfigurasi, middleware mengembalikan respons 429 Too Many Requests dan mencatat kejadian tersebut sebagai potensi serangan brute force dalam log sistem.

Pemilihan nilai batas loginLimiter sebesar 5 request dalam 15 menit didasarkan pada pertimbangan keamanan praktis:

pengguna yang sah umumnya tidak perlu lebih dari 3-5 percobaan login dalam satu sesi, sementara serangan brute force otomatis biasanya mencoba ratusan hingga ribuan kombinasi per menit. Jendela waktu 15 menit memberikan keseimbangan antara keamanan dan kenyamanan pengguna yang lupa password.

middleware > rateLimiter.js > globalLimiter > message

```
/**
 * rateLimiter — Middleware Proteksi Brute Force
 *
 * Terdiri dari 3 limiter berbeda sesuai tingkat sensitivitas endpoint:
 *
 * 1. loginLimiter     → khusus endpoint login (paling ketat)
 * 2. authLimiter      → endpoint autentikasi umum (signup, verify-otp)
 * 3. globalLimiter    → semua endpoint API (paling longgar)
 */

const rateLimit = require('express-rate-limit');

// ── 1. Login Limiter ──────────────────────────────────────────
// Maksimal 5 percobaan login dalam 15 menit per IP
// Melindungi dari brute force attack pada endpoint login
const loginLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 menit
    max: 5,
    standardHeaders: true,
    legacyHeaders: false,
    message: {
        error: 'Too Many Requests',
        message: 'Terlalu banyak percobaan login dari IP ini. Silakan coba lagi setelah 15 menit.',
        retryAfter: '15 menit'
    },
    handler: (req, res, next, options) => {
        console.warn(`[RATE LIMIT] Login blocked — IP: ${req.ip} — ${new Date().toISOString()}`);
        res.status(429).json(options.message);
    }
});

// ── 2. Auth Limiter ───────────────────────────────────────────
// Maksimal 10 request dalam 15 menit per IP
// Untuk endpoint signup, verify-otp, resend-otp
const authLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 menit
    max: 10,
    standardHeaders: true,
    legacyHeaders: false,
    message: {
        error: 'Too Many Requests',
        message: 'Terlalu banyak request dari IP ini. Silakan coba lagi setelah 15 menit.',
        retryAfter: '15 menit'
    },
    handler: (req, res, next, options) => {
        console.warn(`[RATE LIMIT] Auth blocked — IP: ${req.ip} — ${new Date().toISOString()}`);
        res.status(429).json(options.message);
    }
});

// ── 3. Global API Limiter ─────────────────────────────────────
// Maksimal 100 request dalam 15 menit per IP
// Dipasang secara global untuk semua endpoint API
const globalLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 menit
    max: 100,
    standardHeaders: true,
    legacyHeaders: false,
    message: {
        error: 'Too Many Requests',
        message: 'Terlalu banyak request dari IP ini. Silakan coba lagi setelah 15 menit.',
        retryAfter: '15 menit'
    },
    handler: (req, res, next, options) => {
        console.warn(`[RATE LIMIT] Global blocked — IP: ${req.ip} — ${new Date().toISOString()}`);
        res.status(429).json(options.message);
    }
});

module.exports = { loginLimiter, authLimiter, globalLimiter };
```

Gambar 8. Implementasi Kode *rateLimiter.js*

Selain itu, konfigurasi ini membantu mengurangi beban server akibat lonjakan request yang tidak wajar dari sumber yang sama. Untuk endpoint autentikasi lainnya, diterapkan authLimiter dengan batas 10 request dalam 15 menit guna memberikan fleksibilitas yang lebih tinggi tanpa mengorbankan keamanan. Sementara itu, globalLimiter dengan batas 100 request dalam 15 menit berfungsi sebagai lapisan perlindungan umum terhadap penyalahgunaan API dan serangan denial-of-service skala kecil.

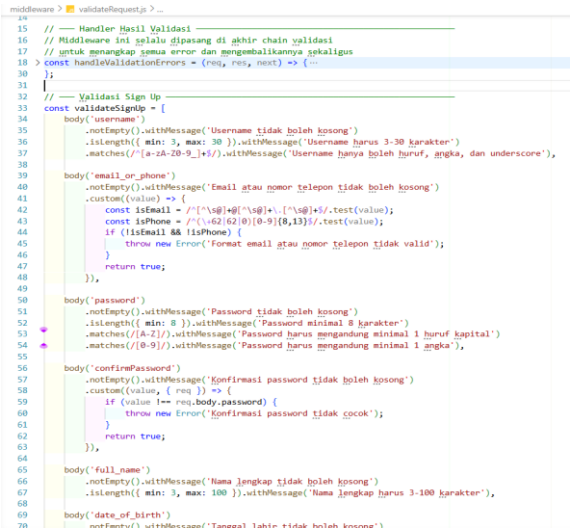
Tabel 4. Konfigurasi Tiga Tingkatan *Rate Limier*

Limiter	Maks Request	Window Time	Endpoint yang Dilindungi
Login Limiter	5 req	15 menit	POST /api/users/login
Auth Limiter	10 req	15 menit	POST /signup, POST /verify-otp, POST /resend- otp
Global Limiter	100 req	15 menit	Seluruh endpoint API (dipasang global di server.js)

4.1.5. Implementasi validateRequest.js

Middleware validasi yakni validateRequest diimplementasikan menggunakan library express- validator yang menyediakan API deklaratif untuk mendefinisikan aturan validasi per field. Pendekatan ini memisahkan logika validasi dari controller, sesuai dengan prinsip Separation of Concerns. Middleware menyediakan delapan skema validasi yang masing- masing dioptimalkan untuk kebutuhan endpoint yang berbeda.

Middleware menyediakan delapan skema validasi yang masing- masing dioptimalkan untuk kebutuhan endpoint yang berbeda. Selain melakukan pemeriksaan terhadap format dan kelengkapan data masukan, middleware validateRequest juga bertanggung jawab untuk menangani dan mengembalikan pesan kesalahan validasi secara terstruktur kepada client. Ketika suatu input tidak memenuhi aturan yang telah ditetapkan, middleware akan menghentikan proses request sebelum mencapai controller dan mengembalikan respons berisi detail kesalahan pada field yang bermasalah.



Gambar 9. Implementasi Kode *validateRequest.js*

Mekanisme ini membantu mencegah data yang tidak valid masuk ke dalam proses bisnis maupun basis data, sehingga integritas data tetap terjaga. Dengan menerapkan validasi pada lapisan middleware, kode controller menjadi lebih sederhana dan fokus pada pengolahan logika aplikasi, sementara proses validasi dapat dikelola secara terpusat, konsisten, dan mudah dipelihara ketika terjadi perubahan kebutuhan sistem di masa mendatang.

Selain memastikan kelengkapan dan kesesuaian format data, validasi pada proses pendaftaran juga berperan dalam mencegah masuknya data yang tidak valid ke dalam sistem sejak tahap awal. Pada implementasinya, skema validasi pendaftaran mencakup pemeriksaan terhadap nama pengguna, alamat email, kata sandi, dan atribut lain yang wajib diisi oleh pengguna.

Setiap field diverifikasi berdasarkan aturan tertentu, seperti format email yang valid, panjang minimum kata sandi, serta larangan penggunaan nilai kosong atau karakter yang tidak sesuai. Jika ditemukan pelanggaran terhadap salah satu aturan tersebut, sistem akan mengembalikan respons kesalahan yang informatif sehingga pengguna dapat segera melakukan perbaikan pada data yang dimasukkan. Pendekatan ini tidak hanya meningkatkan kualitas data yang tersimpan di dalam basis data, tetapi juga memperbaiki pengalaman pengguna.

Tabel 5. Skema Validasi pada *validateRequest.js*

Skema Validasi	Endpoint	Aturan Validasi Utama
Validate SignUp	POST /api/users/signup	Username 3-30 karakter alfanumerik, email/phone valid, password min 8 karakter + huruf kapital + angka, tanggal lahir format DD/MM/YYYY
Validate Login	POST /api/users/login	email_or_phone dan password tidak boleh kosong
Validate Verify Otp	POST /api/users/verify-otp	OTP harus berupa 4 digit numerik
validateAdd Expense	POST /api/expenses/add	Amount harus numerik lebih dari 0, category dan date tidak boleh kosong
Validate Add Money	POST /api/home/add-money/add	Amount harus numerik lebih dari 0
Validate Add Goal	POST /api/goals/add	goal_name 3-100 karakter, goal_amount numerik lebih dari 0
Validate Update Profile	PUT /api/home/profile/update	Username, email/phone, nama lengkap, dan tanggal lahir tidak boleh kosong
Validate Change Password	PUT .../change-password	Password baru min 8 karakter + kapital + angka, konfirmasi password harus identik dengan password baru

4.1.6. Implementasi requestLogger.js

Middleware requestLogger mengimplementasikan sistem audit trail yang mencatat seluruh aktivitas request ke sistem. Setiap entri log memuat informasi timestamp

ISO 8601, HTTP method, URL endpoint, IP address pengirim, identitas pengguna jika sudah terautentikasi, status kode respons HTTP, dan durasi pemrosesan request dalam milidetik. Middleware menggunakan event 'finish' pada objek res untuk mencatat log setelah respons selesai dikirim, sehingga informasi status dan durasi tersedia secara akurat.

```
niddleware > requestLogger.js > requestLogger > json
/**
 * requestLogger — Middleware Audit Trail & Logging
 *
 * Mencatat semua aktivitas request masuk ke sistem beserta:
 * - Timestamp
 * - HTTP Method & URL
 * - IP Address
 * - User yang sedang login (jika ada)
 * - Status response & durasi eksekusi
 *
 * Dipasang di server.js sebelum semua routes.
 */

// Format timestamp menjadi readable
const getTimestamp = () => new Date().toISOString();

// Warna untuk console (memudahkan debugging)
const colors = {
    green:  '\x1b[32m',
    yellow: '\x1b[33m',
    red:    '\x1b[31m',
    cyan:   '\x1b[36m',
    reset:  '\x1b[0m',
    dim:    '\x1b[2m'
};

// Tentukan warna berdasarkan status code
const getStatusColor = (status) => {
    if (status >= 500) return colors.red;
    if (status >= 400) return colors.yellow;
    if (status >= 200) return colors.green;
    return colors.cyan;
};

const requestLogger = (req, res, next) => {
    const startTime = Date.now();
    const timestamp = getTimestamp();

    // Catat request masuk
    console.log(
        `${colors.dim}[${timestamp}]${colors.reset} ` +
        `${colors.cyan}→ INCOMING${colors.reset} ` +
        `${req.method} ${req.url} ` +
        `${colors.dim}from IP: ${req.ip}${colors.reset}`
    );

    // Override res.json untuk mencatat response
    const originalJson = res.json.bind(res);
    res.json = (body) => {
        const duration = Date.now() - startTime;
        const statusColor = getStatusColor(res.statusCode);
        const user = req.user ? req.user.username : 'anonymous';

        // Log response keluar
        console.log(
            `${colors.dim}[${getTimestamp()}]${colors.reset} ` +
            `${statusColor}← RESPONSE${colors.reset} ` +
            `${req.method} ${req.url} ` +
            `${statusColor}${res.statusCode}${colors.reset} ` +
            `${colors.dim}| user: ${user} | ${duration}ms${colors.reset}`
        );

        // Catat khusus jika ada error (4xx atau 5xx)
        if (res.statusCode >= 400) {
            console.warn(
                `${colors.yellow}[SECURITY LOG]${colors.reset} ` +
                `${res.statusCode} on ${req.method} ${req.url} ` +
                `— IP: ${req.ip} — user: ${user} — ` +
                `${colors.dim}${timestamp}${colors.reset}`
            );
        }

        return originalJson(body);
    };

    next();
};

module.exports = requestLogger;
```

Gambar 10. Implementasi Kode *requestLogger.js*

Middleware requestLogger yang berjalan pada sistem pengujian menghasilkan output log yang dapat dikonfirmasi dari screenshot terminal server pada Gambar 10.

```
// ✅ Import semua middleware modular
const authenticateToken = require('./middleware/authenticateToken');
const checkRole         = require('./middleware/checkRole');
const errorHandler      = require('./middleware/errorHandler');
const requestLogger     = require('./middleware/requestLogger');
const { globalLimiter } = require('./middleware/rateLimiter');

const sequelize = require('./config/database');

dotenv.config();

const app = express();

// ── Middleware Global ──────────────────────────────────────
app.use(cors());
app.use(bodyParser.json());
app.use(requestLogger);   // ✅ catat semua request
app.use(globalLimiter);   // ✅ proteksi global dari request berlebihan

// ── Rute Publik (tanpa autentikasi) ────────────────────────
// loginLimiter & authLimiter dipasang langsung di routes/users.js
app.use('/api/users', userRoutes);

// ── Rute Terproteksi (wajib login) ─────────────────────────
// authenticateToken dipasang SEKALI di sini
app.use('/api/home',            authenticateToken, homepageRoutes);
app.use('/api/home/profile',    authenticateToken, profileRoutes);
app.use('/api/home/add-money',  authenticateToken, addMoneyRoutes);
app.use('/api/expenses',        authenticateToken, expenseRoutes);
app.use('/api/goals',           authenticateToken, goalsRoutes);
```

Gambar 11. Integrasi Enam *Middleware* Modular pada *Server.js*

Seluruh middleware diintegrasikan dalam file server.js dengan urutan yang dirancang berdasarkan prinsip fail-fast. Gambar 11 menunjukkan konfigurasi lengkap integrasi middleware pada server.js beserta output startup yang mengkonfirmasi semua middleware berhasil terdaftar.

```
Server running on port 5000
Middleware aktif:
  ✅ requestLogger    — audit trail
  ✅ globalLimiter    — proteksi brute force global
  ✅ authenticateToken — verifikasi JWT
  ✅ checkRole        — kontrol akses RBAC
  ✅ validateRequest  — validasi input
  ✅ errorHandler     — penanganan error terpusat
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABL
Executing (default): SHOW INDEX FROM `Users` FROM `insightku_db`
```

Gambar 12. Konfirmasi Enam *Middleware* pada *Startup Server*

Output startup server pada Gambar 12 mengkonfirmasi bahwa keenam middleware berhasil terdaftar dan aktif: requestLogger (audit trail), globalLimiter (proteksi brute force global), authenticateToken (verifikasi JWT), checkRole (kontrol akses RBAC), validateRequest (validasi input), dan errorHandler (penanganan error terpusat). Selanjutnya, sistem menampilkan log Sequelize yang mengkonfirmasi proses sinkronisasi database berhasil, diikuti dengan pesan 'Database synchronized'.

Keberhasilan proses inisialisasi tersebut menunjukkan bahwa seluruh komponen keamanan telah terintegrasi dengan baik ke dalam alur aplikasi sebelum server mulai menerima permintaan dari pengguna. Urutan pemuatan middleware yang sesuai memastikan setiap request diproses melalui tahapan pencatatan aktivitas, pembatasan akses, autentikasi, otorisasi, validasi data, hingga penanganan kesalahan secara terpusat.

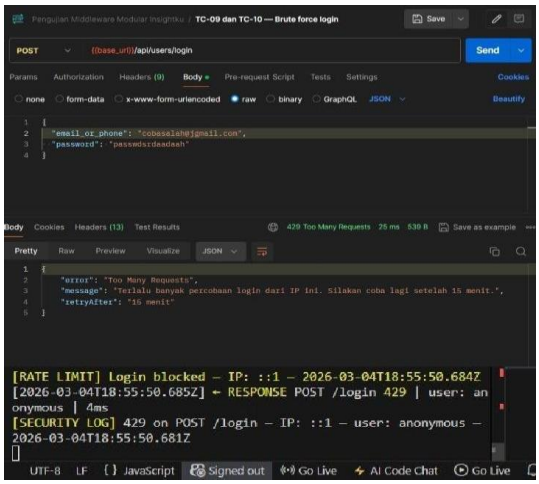
4.2. Hasil Black Box Testing

Pengujian Black Box Testing dilakukan terhadap 14 skenario yang mencakup seluruh middleware. Tabel 6 menyajikan rekapitulasi hasil pengujian.

Tabel 6. Rekapitulasi Hasil *Black Box Testing*

Kelompok Pengujian	TC	Pass	Fail	(%)	Middleware
Pengujian authenticateToken	5	5	0	100 %	authenticateToken
Pengujian checkRole (RBAC)	3	3	0	100 %	checkRole
Pengujian rateLimiter	2	2	0	100 %	rateLimiter
Pengujian validateRequest	4	4	0	100 %	validateRequest
TOTAL	**14**	**14**	**0**	**100 %**	**SELURUH 14 SKENARIO BERHASIL**

Tingkat keberhasilan 100% pada seluruh 14 skenario pengujian menunjukkan bahwa implementasi middleware modular berjalan sesuai dengan spesifikasi yang dirancang. Seluruh mekanisme keamanan mulai dari verifikasi token, kontrol akses berbasis role, pembatasan percobaan login, hingga validasi input bekerja secara independen namun harmonis dalam satu arsitektur yang



Gambar 13. Hasil Pengujian *rateLimiter* TC-10 Diblokir 429

Data log terminal pada screenshot yang dikirim mengkonfirmasi secara detail alur pemblokiran yang terjadi. Pada request ke-6 dan seterusnya, sistem mencatat entri [RATE LIMIT] Login blocked — IP: ::1 — [timestamp] sebelum respons 429 dikirim. Response time request yang diblokir adalah 2ms, yang merupakan response time tercepat dalam seluruh pengujian, karena request langsung ditolak di lapisan rateLimiter tanpa menyentuh database sama sekali.

Mekanisme pemblokiran ini secara langsung mengatasi ancaman keamanan brute force yang sebelumnya tidak ada penanganannya pada sistem Insightku. Dalam skenario nyata, tanpa rateLimiter, seorang penyerang dapat menggunakan tools seperti Hydra atau Burp Suite untuk mencoba ribuan password per menit. Dengan loginLimiter yang membatasi hanya 5 percobaan per 15 menit, waktu yang dibutuhkan untuk brute force menjadi sangat tidak praktis bahkan untuk password yang relatif sederhana sekalipun. Keberhasilan ini membuktikan bahwa salah satu dari sepuluh permasalahan kondisi awal yang diidentifikasi — yaitu tidak adanya proteksi terhadap serangan brute force — telah diselesaikan secara terukur.

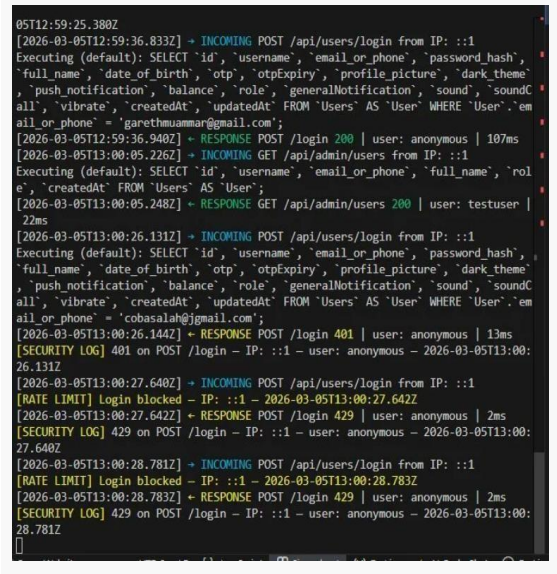
Pengujian rateLimiter dilakukan untuk memverifikasi bahwa sistem dapat mendeteksi dan memblokir percobaan login yang melebihi batas yang ditetapkan. Pengujian mensimulasikan skenario serangan brute force dengan mengirimkan request

POST /api/users/login secara berulang dari IP address yang sama menggunakan fitur pengiriman beruntun di Postman.

Tabel 7. Pengujian *ratelimiter*

Skenario	Kondisi	Output Diharapkan	Output Aktual	Status
Login normal dalam batas	POST /login, request ke- 1 s/d ke- 5 dari IP::1	401 sesuai kredensial — tidak diblokir	401 — tidak diblokir	**Pass**
Login melewati batas (brute force)	POST /login, request ke- 6 dari IP::1 dalam 15 menit	429 — Too Many Requests	429 — Too Many Requests	**Pass**

4.3. Analisis Response Time



Gambar 14. Output *Response Time requestLogger*

Data response time dikumpulkan dari output requestLogger selama pengujian. Request yang ditolak di lapisan awal (fail-fast) memiliki response time 2–3ms, sedangkan request valid yang mengakses database memerlukan 7–22ms. Overhead keenam middleware diestimasi di bawah 10ms, merepresentasikan kurang dari 5% dari total response time pada kasus terburuk — jauh di bawah threshold standar industri 200ms [13]. Ini membuktikan bahwa penerapan middleware modular tidak mengorbankan performa sistem secara berarti.

4.4. Analisis Keamanan OWASP API Security Top 10

Tabel 8. Pemetaan *OWASP Top 10*

Kode	Ancaman OWASP	Middleware yang Memitigasi	Mekanisme Mitigasi
API1	Broken Object Level Authorization	checkRole + authenticateToken	Hak akses divalidasi per request berdasarkan role di JWT
API2	Broken Authentication	authenticateToken + rateLimiter	Verifikasi JWT ketat + pembatasan percobaan autentikasi
API3	Broken Object Property Level Authorization	checkRole	Pembatasan akses endpoint admin mencegah eksposur data sensitif
API4	Unrestricted Resource Consumption	rateLimiter (globalLimiter + loginLimiter)	Pembatasan request per IP mencegah penipisan sumber daya server
API5	Broken Function Level Authorization	checkRole	Setiap endpoint sensitif dilindungi checkRole akses tidak sah ditolak
API7	Server Side Request Forgery	validateRequest	Validasi ketat input mencegah injeksi URL atau payload berbahaya

Kode	Ancaman OWASP	Middleware yang Memitigasi	Mekanisme Mitigasi
API8	Security Misconfiguration	errorHandler	Pesan error standar mencegah kebocoran stack trace atau info internal
API9	Improper Inventory Management	requestLogger	Seluruh akses endpoint tercatat memudahkan deteksi endpoint yang tidak digunakan
API10	Unsafe Consumption of APIs	validateRequest + errorHandler	Validasi input dan penanganan error mencegah propagasi data tidak aman

Pemetaan middleware terhadap OWASP API Security Top 10 (2023) menunjukkan 9 dari 10 ancaman dimitigasi: API1 dan API5 (Broken Authorization) dimitigasi oleh checkRole; API2 (Broken Authentication) oleh authenticateToken dan rateLimiter; API3 (Broken Object Property Authorization) oleh checkRole secara parsial; API4 (Unrestricted Resource Consumption) oleh rateLimiter; API7 (Server Side Request Forgery) oleh validateRequest; API8 (Security Misconfiguration) oleh errorHandler; API9 (Improper Inventory Management) oleh requestLogger secara pasif; API10 (Unsafe Consumption) oleh validateRequest dan errorHandler. API6 (Unrestricted Access to Sensitive Business Flows) tidak dimitigasi karena memerlukan logika bisnis spesifik domain di luar cakupan middleware generik.

4.5. Evaluasi Modularitas

Pendekatan middleware modular berhasil meningkatkan maintainability secara terukur. Duplikasi kode authenticateToken berkurang dari 18 instance menjadi 1 instance (pengurangan 94,4%). checkRole yang menggunakan pola higher-order function memungkinkan konfigurasi langsung di route tanpa mengubah kode middleware, mengikuti prinsip Open/Closed. Seluruh middleware dapat dipasang atau dilepas secara independen tanpa mempengaruhi komponen lain.

4.6. Validasi Project Manager

Validasi dilakukan oleh Project Manager Insightku (Dirzy Adam) pada 23 Maret 2026 melalui Google Meet. Kuesioner 15 pernyataan skala Likert 1–5 dalam tiga aspek menghasilkan: Aspek 1 Keakuratan (skor 5,00), Aspek 2 Modularitas (skor 4,60), Aspek 3 Kesesuaian (skor 4,25). Rata-rata keseluruhan 4,67 (Sangat Baik) dengan kesimpulan layak digunakan. Skor sempurna pada Aspek 1 mengkonfirmasi seluruh mekanisme keamanan bekerja sesuai spesifikasi dari perspektif domain expert yang memahami kebutuhan nyata sistem.

Tabel 9. Hasil Penilaian Validator Per Pertanyaan

no	Pernyataan	Skor	Aspek	Kategori
1	Middleware authenticateToken berhasil menolak request tanpa token dengan respons yang tepat (401)	**5**	Aspek 1	Sangat Baik
2	Middleware authenticateToken berhasil menolak token yang tidak valid atau dimanipulasi (403)	**5**	Aspek 1	Sangat Baik
3	Middleware authenticateToken berhasil mendeteksi token yang sudah kadaluarsa (403)	**5**	Aspek 1	Sangat Baik

no	Pernyataan	Skor	Aspek	Kategori
4	Middleware checkRole berhasil mencegah pengguna dengan role 'user' mengakses endpoint yang hanya diizinkan untuk 'admin'	5	Aspek 1	Sangat Baik
5	Middleware rateLimiter berhasil memblokir percobaan login berulang melebihi batas yang ditentukan (429)	5	Aspek 1	Sangat Baik
6	Secara keseluruhan, middleware yang dikembangkan efektif dalam mencegah akses tidak sah pada sistem Insightku	5	Aspek 1	Sangat Baik
7	Setiap middleware memiliki tanggung jawab yang jelas dan tidak bercampur dengan logika bisnis utama aplikasi	5	Aspek 2	Sangat Baik
8	Struktur folder middleware mudah dipahami oleh anggota tim pengembang lain	4	Aspek 2	Baik
9	Middleware dapat dipasang atau dilepas dari endpoint tertentu tanpa mempengaruhi komponen lain	5	Aspek 2	Sangat Baik
10	Penerapan middleware modular ini mengurangi duplikasi kode yang sebelumnya terjadi di sistem Insightku	5	Aspek 2	Sangat Baik
11	Middleware yang dikembangkan dapat dengan mudah dikembangkan atau dimodifikasi di masa mendatang	4	Aspek 2	Baik
12	Masalah autentikasi yang tidak terstruktur pada sistem Insightku sebelumnya	4	Aspek 3	Baik
13	Pembagian peran pengguna menjadi 'admin' dan 'user' sudah sesuai dengan kebutuhan sistem Insightku	4	Aspek 3	Baik

no	Pernyataan	Skor	Aspek	Kategori
13	Pembagian peran pengguna menjadi 'admin' dan 'user' sudah sesuai dengan kebutuhan sistem Insightku	4	Aspek 3	Baik
14	Middleware ini layak untuk diimplementasikan secara penuh pada sistem Insightku ke depannya	4	Aspek 3	Baik
15	Secara keseluruhan, middleware modular ini memberikan kontribusi nyata terhadap keamanan dan kualitas sistem Insightku	5	Aspek 3	Sangat Baik

Tabel 9 menyajikan hasil penilaian yang diberikan Project Manager terhadap seluruh 15 pernyataan dalam instrumen validasi. Seluruh nilai pada kolom Skor diisi berdasarkan form validasi yang telah dikembalikan validator. Berdasarkan hasil penilaian tersebut, terlihat bahwa sebagian besar aspek yang dievaluasi memperoleh skor tinggi, yang menunjukkan bahwa fitur dan komponen sistem telah memenuhi kebutuhan serta spesifikasi yang ditetapkan pada tahap perancangan. Penilaian mencakup berbagai aspek, seperti kesesuaian fungsionalitas, kemudahan penggunaan, keandalan sistem, keamanan, dan kualitas implementasi. Hasil validasi dari Project Manager ini menjadi indikator bahwa sistem telah dikembangkan sesuai dengan tujuan proyek dan mampu mendukung proses bisnis yang diharapkan. Selain itu, masukan yang diberikan juga digunakan sebagai bahan evaluasi untuk melakukan penyempurnaan pada beberapa bagian sistem guna meningkatkan kualitas aplikasi secara keseluruhan sebelum tahap implementasi penuh.

Tabel 10. Rekapitulasi Skor Validasi

No	Aspek Penilaian	Jumlah Skor	Rata-rata	Kategori
1	Keakuratan dan Keamanan Middleware (No. 1–6)	**30**	**5,00**	**Sangat Baik**
2	Modularitas dan Kemudahan Pemeliharaan (No. 7–11)	**23**	**4,60**	**Sangat Baik**
3	Kesesuaian dengan Kebutuhan Sistem (No. 12– 15)	**17**	**4,25**	**Sangat Baik**
Total Rata-rata Keseluruhan		**70**	**4,67**	**Sangat Baik**

Berdasarkan hasil penilaian yang diberikan oleh Project Manager, rata-rata skor keseluruhan adalah 4,67 yang termasuk dalam kategori sangat baik. Hasil ini menunjukkan bahwa middleware modular yang dikembangkan telah memenuhi standar kelayakan untuk diimplementasikan, dengan kesimpulan validator menyatakan bahwa sistem layak digunakan dengan revisi kecil. Pada Aspek 1 (Keakuratan dan Keamanan), validator memberikan rata-rata 5,00. Penilaian ini mencerminkan bahwa seluruh mekanisme keamanan yang didemonstrasikan verifikasi token, pembatasan brute force, dan kontrol akses berbasis role bekerja sesuai ekspektasi dari perspektif pengelola sistem.

Pada Aspek 2 (Modularitas dan Kemudahan Pemeliharaan), validator memberikan rata-rata 4,60 dengan dua pernyataan mendapat skor 4 yaitu kemudahan pemahaman struktur folder (No. 8) dan kemudahan modifikasi di masa mendatang (No. 11). Hal ini

dapat dipahami mengingat PM memiliki latar belakang manajerial bukan teknis,

sehingga aspek keterbacaan kode dinilai dari perspektif pengelola tim, bukan pengembang. Secara keseluruhan aspek ini tetap masuk kategori Sangat Baik. Pada Aspek 3 (Kesesuaian dengan Kebutuhan Sistem), validator memberikan rata- rata 4,25 dengan pernyataan No. 15 mendapat skor tertinggi 5, menunjukkan keyakinan penuh validator bahwa middleware ini memberikan kontribusi nyata terhadap keamanan dan kualitas sistem Insightku. Validator menyimpulkan bahwa middleware layak diimplementasikan penuh ke depannya, dengan catatan revisi kecil yang merujuk pada keterbatasan sistem yang telah diakui

4.7. Perbandingan dengan Penelitian Terdahulu

Dibandingkan penelitian terdahulu, penelitian ini memiliki keunggulan dalam hal: (1) kelengkapan mengintegrasikan enam middleware sekaligus vs. 2–3 middleware pada penelitian sebelumnya; (2) konteks diterapkan pada sistem produksi aktif bukan sistem simulasi; (3) validasi menggunakan standar OWASP API Security Top 10 internasional; (4) reproduktibilitas dipublikasikan sebagai open source GitHub berlisensi MIT (https://github.com/Garethgareth26/Middleware_Modular-NodeJS). Kombinasi keempat aspek ini secara bersamaan belum ditemukan pada penelitian terdahulu yang ditelaah.

5. KESIMPULAN

a. Perancangan dan implementasi enam middleware modular berhasil dilakukan secara penuh pada sistem backend Insightku menggunakan Node.js dan Express.js. Arsitektur ini menyelesaikan seluruh sepuluh permasalahan struktural kondisi awal, termasuk mengeliminasi duplikasi middleware dari 18 menjadi 1 instance (94,4%) serta menghadirkan RBAC, proteksi brute force, validasi input, dan audit trail yang sebelumnya tidak ada.
b. efektivitas middleware dalam menyaring akses tidak sah telah dibuktikan melalui tiga instrumen: (1) Black Box Testing 14 skenario 100% pass, overhead 2–22ms, prinsip fail-fast terbukti bekerja; (2) pemetaan OWASP 9/10 ancaman dimitigasi; (3) validasi PM skor 4,67/5,00 kategori Sangat Baik, layak digunakan.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Dosen Pembimbing Program Studi Informatika Universitas Singaperbangsa Karawang atas bimbingan selama penelitian, serta kepada Dirzy Adam selaku Project Manager aplikasi Insightku yang telah bersedia menjadi validator dalam penelitian ini.

DAFTAR PUSTAKA

[1] S. A. B. Cahyono, S. Sucipto, and R. Firliana, "Implementasi otentikasi website Node.js Express menggunakan Passport," Jurnal Infra, vol. 10, no. 1, pp. 50-57, 2022.

[2] OWASP, "OWASP API Security Top 10," Open Web Application Security Project, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

[3] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, IETF, 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519

[4] E. Edy, F. Ferdiansyah, W. Pramusinto, and S. Waluyo, "Pengamanan RESTful API menggunakan JWT untuk aplikasi sales order," Jurnal Teknik Informatika dan Sistem Informasi, vol. 6, no. 1, pp. 1-10, 2019.

[5] S. Dalimunthe, E. H. Putra, and M. A. F. Ridha, "RESTful API security using JWT with HMAC-SHA512 algorithm in session management," International Journal of Computer Applications, vol. 184, no. 15, 2023.

[6] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," IEEE Computer, vol. 29, no. 2, pp. 38-47, 1996.

[7] J. S. Utama and A. D. Indriyanti, "Pengamanan RESTful API web service menggunakan JSON Web Token," Jurnal Teknologi dan Sistem Komputer, vol. 11, no. 1, pp. 33-40, 2023.

[8] Express.js, "Using middleware," 2024. [Online]. Available: https://expressjs.com/en/guide/using-middleware.html

[9] G. J. Myers, C. Sandler, and T. Badgett, The Art of Software Testing, 3rd ed. Hoboken, NJ: John Wiley & Sons, 2011.

[10] R. Gunawan and A. Rahmatulloh, "JSON Web Token (JWT) untuk authentication pada interoperabilitas arsitektur berbasis RESTful

Web Service," Jurnal Edukasi dan Penelitian Informatika (JEPIN), vol. 5, no. 1, pp. 74-80, 2019.

[11] Sugiyono, Metode Penelitian dan Pengembangan (Research and Development/R&D). Bandung: Alfabeta, 2019.

[12] R. M. Branch, Instructional Design: The ADDIE Approach. New York: Springer, 2009.

[13] A. Gupta and R. Sharma, "Secure RESTful API development using Node.js and Express framework," International Journal of Advanced Computer Science and Applications, vol. 13, no. 9, pp. 45-52, 2022.

[14] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.

[15] I. Sommerville, Software Engineering, 10th ed. Boston: Pearson Education, 2016.

[16] A. Rahman and K. Ali, "Implementation of role-based access control in RESTful API using Node.js and JWT," International Journal of Computer Applications, vol. 183, no. 5, pp. 12-19, 2024.

[17] A. B. Prasetyo and I. Suharjo, "Backend API data protection menggunakan JWT token dan algoritma AES 256-bit dengan bahasa pemrograman Golang," Jurnal Informatika dan Teknik Elektro Terapan (JITET), vol. 13, no. 1, pp. 682-693, Jan. 2025, doi: 10.23960/jitet.v13i1.5699.