Buku Panduan Interoperabilitas dengan Node.js

MODUL 5: Pengamanan API - Autentikasi dan Autorisasi dengan JWT

Teknologi Rekayasa Perangkat Lunak

Oktober 2025

Daftar Isi

Daftar Gambar

Bab 1

Landasan Teori

1.1 Kata Pengantar

Selamat datang di Modul 5. Setelah berhasil membangun API yang fungsional dan terhubung ke basis data SQLite, langkah esensial berikutnya adalah mengamankannya. API yang kita miliki saat ini, meskipun datanya persisten, masih bersifat terbuka. Siapa saja dapat mengakses dan memanipulasi data tanpa batasan, sebuah kondisi yang tidak dapat diterima dalam aplikasi dunia nyata karena menimbulkan risiko keamanan yang signifikan.

Modul ini akan membawa Anda menyelami dunia keamanan API, sebuah aspek krusial dalam rekayasa perangkat lunak. Kita akan membedah dua konsep fundamental: **Autentikasi**, proses verifikasi identitas pengguna, dan **Autorisasi**, proses penentuan hak akses pengguna. Anda akan belajar mengimplementasikan sistem registrasi dan login pengguna yang aman dari awal, termasuk praktik penting penyimpanan kata sandi menggunakan *hashing* yang akan melindungi kredensial pengguna bahkan jika basis data terkompromi.

Teknologi inti yang akan kita gunakan adalah **JSON Web Tokens (JWT)**, sebuah standar industri modern untuk mengelola sesi pengguna secara *stateless* dalam API. Kita akan mempelajari strukturnya, alur kerjanya, dan bagaimana menggunakannya untuk melindungi *endpoint-endpoint* penting pada API Manajemen Film kita yang berbasis SQLite. Setelah menyelesaikan modul ini, API Anda akan memiliki lapisan keamanan dasar, memastikan bahwa hanya pengguna yang terverifikasi yang dapat melakukan operasi sensitif terhadap data film dan sutradara.

1.2 Prasyarat Pengetahuan

- Telah berhasil menyelesaikan dan memahami seluruh materi dan tugas pada **Modul 3** (Persistensi Data dengan SQLite).

- Memiliki proyek API Manajemen Film yang terhubung dengan basis data SQLite dan berfungsi penuh.

1.3 Tujuan Pembelajaran

Setelah menyelesaikan modul ini, peserta pembelajaran diharapkan mampu:

- Membedakan konsep **Autentikasi** dan **Autorisasi**.
- Menjelaskan cara kerja **JSON Web Tokens (JWT)** sebagai standar untuk autentikasi API.
- Mengimplementasikan *password hashing* menggunakan `bcrypt` untuk menyimpan kata sandi secara aman di basis data SQLite.
- Membangun *endpoint* untuk registrasi (`/auth/register`) dan login (`/auth/login`) pengguna yang berinteraksi dengan SQLite.
- Membuat *middleware* untuk memverifikasi token JWT dan melindungi *endpoint*.
- Menerapkan perlindungan berbasis autentikasi pada rute-rute API yang memerlukan akses terbatas.

1.4 Urgensi Keamanan API

Dalam ekosistem digital saat ini, API seringkali menjadi ”pintu depan” menuju data dan fungsionalitas inti sebuah sistem. API yang tidak diamankan ibarat rumah tanpa kunci; ia rentan terhadap akses tidak sah, pencurian data, manipulasi informasi, hingga penolakan layanan (*Denial of Service*). Mengamankan API bukan lagi pilihan, melainkan sebuah keharusan fundamental untuk melindungi data pengguna, menjaga integritas sistem, dan mencegah penyalahgunaan sumber daya. Tanpa keamanan yang memadai, API dapat menjadi vektor serangan utama terhadap sebuah aplikasi.

1.5 Autentikasi vs. Autorisasi: Memverifikasi Identitas dan Hak Akses

Dua pilar utama keamanan API adalah autentikasi dan autorisasi. Memahami perbedaan keduanya sangatlah penting.

Autentikasi (Authentication - ”Siapa Anda?”) adalah proses verifikasi identitas. Tujuannya adalah untuk memastikan bahwa entitas (pengguna atau sistem lain) yang mencoba mengakses API adalah benar-benar siapa yang mereka klaim. Ini adalah langkah pertama dalam proses

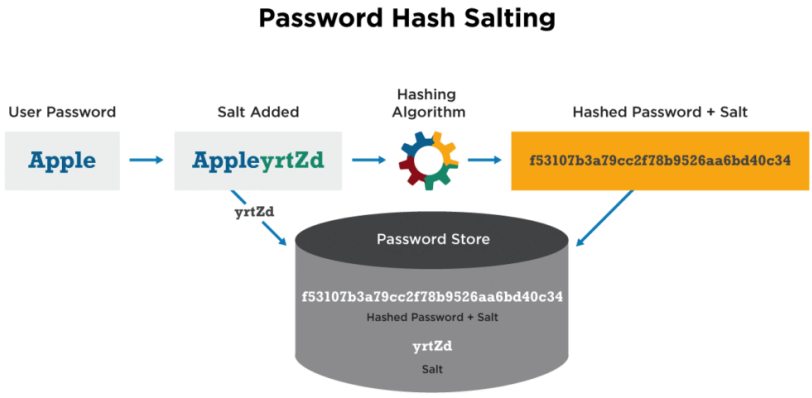
keamanan. Hasil dari autentikasi yang berhasil biasanya adalah sebuah bukti identitas digital yang dapat diverifikasi, seperti token sesi atau JWT.

Autorisasi (Authorization - "Apa yang Boleh Anda Lakukan?") adalah proses penentuan hak akses yang terjadi *setelah* identitas berhasil diverifikasi melalui autentikasi. Tujuannya adalah untuk menentukan apakah pengguna yang telah terautentikasi memiliki izin untuk mengakses sumber daya tertentu atau melakukan tindakan spesifik. Pada modul ini, fokus utama kita adalah pada implementasi **autentikasi**.

1.6 Keamanan Kata Sandi: Hashing dan Salting dengan `bcrypt`

Menyimpan kata sandi pengguna sebagai teks biasa adalah praktik keamanan yang sangat buruk. Jika basis data bocor, semua kata sandi akan terekspos. Teknik standar untuk mengamankan kata sandi adalah **hashing**. *Hashing* adalah proses transformasi data (kata sandi) menjadi sebuah string dengan panjang tetap (disebut *hash*) menggunakan algoritma kriptografi satu arah. Artinya, *hash* tidak dapat dengan mudah dikembalikan (*decrypted*) menjadi kata sandi aslinya.

Untuk meningkatkan keamanan lebih lanjut, digunakan teknik **salting**. *Salt* adalah data acak unik yang ditambahkan ke setiap kata sandi *sebelum* proses *hashing*. Ini memastikan bahwa bahkan jika dua pengguna memiliki kata sandi yang sama, *hash* yang disimpan di basis data akan berbeda.



Gambar 1.1: Ilustrasi proses hashing kata sandi dengan salt.

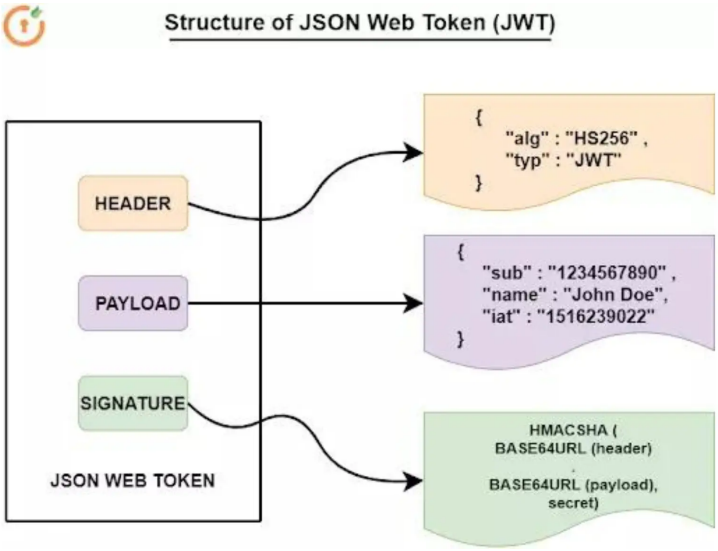
Pustaka **`bcryptjs`** di Node.js adalah implementasi populer dari algoritma *hashing* bcrypt yang secara otomatis menangani pembuatan *salt* dan proses *hashing*. Fungsi utamanya adalah:

- `bcrypt.hash(password, saltRounds)`: Menghasilkan *hash* dari kata sandi.
- `bcrypt.compare(submittedPassword, storedHash)`: Membandingkan kata sandi yang dimasukkan pengguna saat login dengan *hash* yang tersimpan.

1.7 JSON Web Tokens (JWT): Tiket Masuk Digital Anda

Setelah pengguna berhasil diautentikasi, server perlu memberikan ”bukti” autentikasi yang dapat digunakan klien pada permintaan berikutnya. **JWT (JSON Web Token)** adalah standar terbuka (RFC 7519) untuk membuat token akses semacam ini yang bersifat *self-contained* dan ideal untuk API *stateless*.

Struktur JWT: Sebuah JWT terdiri dari tiga bagian: `Header.Payload.Signature`.

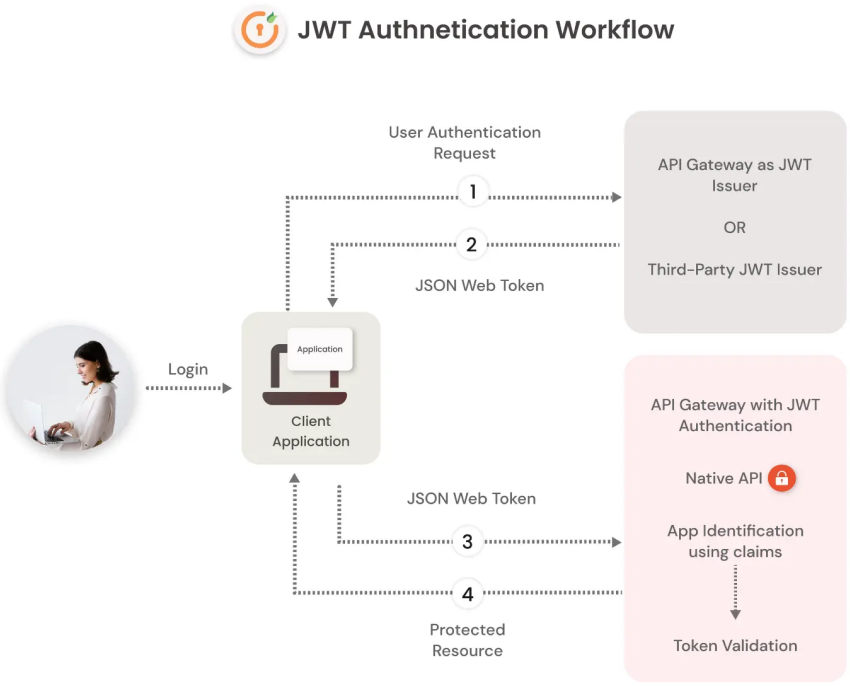


Gambar 1.2: Struktur JSON Web Token (JWT).

1. **Header**: Metadata tentang token (tipe: JWT, algoritma: misal, HS256).
2. **Payload**: Berisi *claims* atau pernyataan tentang pengguna (misal: `userId`, `username`) dan waktu kedaluwarsa (`exp`).
3. **Signature**: Bagian krusial untuk keamanan. Dibuat dengan mengenkripsi *header*, *payload*, dan sebuah **kunci rahasia (*secret key*)** yang hanya diketahui oleh server.

Alur Kerja JWT:

1. **Login**: Klien mengirim kredensial ke server.
2. **Verifikasi & Pembuatan Token**: Server memvalidasi kredensial. Jika cocok, server membuat JWT yang ditandatangani dengan kunci rahasia.
3. **Pengiriman Token**: Server mengirim JWT kembali ke klien.
4. **Penyimpanan Token**: Klien menyimpan JWT.
5. **Permintaan Berikutnya**: Klien menyertakan JWT dalam *header* `Authorization: Bearer <token>`.



Gambar 1.3: Diagram alur kerja autentikasi menggunakan JWT.

6. **Verifikasi di Server**: Server mengekstrak token, memverifikasi *signature*. Jika valid, permintaan diproses.

Bab 2

Sesi Praktikum

2.1 Persiapan: Instalasi Dependensi dan Modifikasi Basis Data

1. Buka proyek `api-manajemen-film` Anda (yang sudah menggunakan SQLite dan `.env`).
2. Instal pustaka yang dibutuhkan:

Listing 2.1: Instalasi dependensi keamanan.

```
npm install jsonwebtoken bcryptjs
```

3. Definisikan Kunci Rahasia JWT: Buka file `.env` dan tambahkan variabel baru.

```
PORT=3300
DB_SOURCE="movies.db"
JWT_SECRET="GANTI_DENGAN_KUNCI_RAHASIA_YANG_SANGAT_KUAT_DAN_UNIK"
```

Listing 2.2: Menambahkan JWT$_{S}ECRETke.env$.

4. Modifikasi Modul Basis Data: Buka file `database.js` dan tambahkan kode untuk membuat tabel `users` jika belum ada.

```
// ... di dalam callback sqlite3.Database, setelah membuat tabel
↪ movies
db.run(`CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
)`, (err) => {
    if (err) {
```

```
        console.error("Gagal membuat tabel users:", err.message);
    }
});
```

Listing 2.3: Menambahkan pembuatan tabel users di database.js.

2.2 Implementasi Endpoint Registrasi (/auth/register)

Endpoint ini akan menangani pembuatan akun pengguna baru dengan kata sandi yang di-*hash*.

1. Buka server.js. Impor bcryptjs dan muat $\texttt{JWT}_{S}ECRET$.

```
require('dotenv').config();
// ... impor express, cors, db
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET;
```

Listing 2.4: Impor dependensi keamanan.

2. Tambahkan rute baru untuk registrasi.

```
// === AUTH ROUTES ===

app.post('/auth/register', (req, res) => {
    const { username, password } = req.body;
    if (!username || !password || password.length < 6) {
        return res.status(400).json({ error: 'Username dan password
        ↪ (min 6 char) harus diisi' });
    }

    bcrypt.hash(password, 10, (err, hashedPassword) => {
        if (err) {
            console.error("Error hashing:", err);
            return res.status(500).json({ error: 'Gagal memproses
            ↪ pendaftaran' });
        }

        const sql = 'INSERT INTO users (username, password) VALUES
        ↪ (?, ?)';
        const params = [username.toLowerCase(), hashedPassword];

```

```
        db.run(sql, params, function(err) {
            if (err) {
                if (err.message.includes('UNIQUE constraint')) {
                    return res.status(409).json({ error: 'Username
                    ↪ sudah digunakan' });
                }
                console.error("Error inserting user:", err);
                return res.status(500).json({ error: 'Gagal
                ↪ menyimpan pengguna' });
            }
            res.status(201).json({ message: 'Registrasi berhasil',
            ↪ userId: this.lastID });
        });
    });
});
```

Listing 2.5: Implementasi endpoint POST /auth/register.

2.3 Implementasi Endpoint Login (/auth/login)

Endpoint ini akan memvalidasi kredensial dan mengembalikan JWT.

1. Tambahkan rute `POST /auth/login` setelah registrasi:

```
app.post('/auth/login', (req, res) => {
    const { username, password } = req.body;
    if (!username || !password) {
        return res.status(400).json({ error: 'Username dan password
        ↪ harus diisi' });
    }

    const sql = "SELECT * FROM users WHERE username = ?";
    db.get(sql, [username.toLowerCase()], (err, user) => {
        if (err || !user) {
            return res.status(401).json({ error: 'Kredensial tidak
            ↪ valid' });
        }

        bcrypt.compare(password, user.password, (err, isMatch) => {
            if (err || !isMatch) {
                return res.status(401).json({ error: 'Kredensial
                ↪ tidak valid' });
```

```
            }

            const payload = { user: { id: user.id, username:
            ↪ user.username } };

            jwt.sign(payload, JWT_SECRET, { expiresIn: '1h' }, (err,
            ↪ token) => {
                if (err) {
                    console.error("Error signing token:", err);
                    return res.status(500).json({ error: 'Gagal
                    ↪ membuat token' });
                }
                res.json({ message: 'Login berhasil', token: token
                ↪ });
            });
        });
    });
});
```

Listing 2.6: Implementasi endpoint POST /auth/login.

2.4 Membuat Middleware Autentikasi

Middleware ini akan memeriksa token di setiap *request* ke *endpoint* yang dilindungi.

1. Buat folder baru bernama `middleware`.
2. Di dalam `middleware`, buat file baru bernama `authMiddleware.js`.

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET;

function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (token == null) {
        return res.status(401).json({ error: 'Akses ditolak, token
        ↪ tidak ditemukan' });
    }

    jwt.verify(token, JWT_SECRET, (err, decodedPayload) => {
        if (err) {
```

```
            console.error("JWT Verify Error:", err.message);
            return res.status(403).json({ error: 'Token tidak valid
            ↪ atau kedaluwarsa' });
        }
        req.user = decodedPayload.user;
        next();
    });
}

module.exports = authenticateToken;
```

Listing 2.7: Kode untuk authMiddleware.js.

2.5 Menerapkan Perlindungan pada Rute Film

Gunakan *middleware* `authenticateToken` untuk melindungi operasi tulis pada `/movies`.

1. Impor *middleware* di `server.js`.

```
// ... impor lainnya
const authenticateToken = require('./middleware/authMiddleware');
```

Listing 2.8: Impor middleware autentikasi.

2. Tambahkan `authenticateToken` sebagai argumen kedua pada rute `POST /movies`, `PUT /movies/:id`, dan `DELETE /movies/:id`.

```
// Contoh untuk POST. Terapkan pola yang sama untuk PUT dan DELETE.
app.post('/movies', authenticateToken, (req, res) => {
  console.log('Request POST /movies oleh user:', req.user.username);
  // ...logika endpoint POST dengan db.run ...
});

app.put('/movies/:id', authenticateToken, (req, res) => {
    // ...logika endpoint PUT dengan db.run ...
});

app.delete('/movies/:id', authenticateToken, (req, res) => {
    // ...logika endpoint DELETE dengan db.run ...
});
```

Listing 2.9: Menerapkan middleware pada rute.

3. **Pengujian Alur Lengkap**:
    - **Registrasi** & **Login**: Buat akun, login, dan simpan token.
    - **Akses Tanpa Token**: Coba `POST /movies` tanpa token (harus 401).
    - **Akses Dengan Token**: Tambahkan token ke *header* `Authorization` (`Bearer <token>`) untuk `POST /movies` (harus 201).
    - Ulangi pengujian akses untuk `PUT` dan `DELETE`.

Bab 3

Tugas Praktikum 5

3.1 Tujuan Tugas

- Mengaplikasikan *middleware* autentikasi pada serangkaian *endpoint* baru.
- Memperkuat pemahaman tentang alur kerja autentikasi berbasis token.

3.2 Skenario

Amankan semua *endpoint* yang bersifat tulis (memodifikasi data) untuk sumber daya `/directors` yang telah Anda buat, sehingga hanya pengguna yang sudah login yang dapat menggunakannya.

3.3 Kriteria Fungsionalitas

1. Terapkan *middleware* `authenticateToken` ke *endpoint* berikut:
    - `POST /directors`
    - `PUT /directors/:id`
    - `DELETE /directors/:id`
2. Biarkan *endpoint* `GET /directors` dan `GET /directors/:id` tetap publik.

3.4 Kriteria Pengumpulan

1. **Kode Sumber (.zip)**: Seluruh folder proyek Node.js Anda yang telah diperbarui.

2. **Koleksi Postman (.json)**: File ekspor dari *Collection* Postman Anda, menunjukkan cara penggunaan *endpoint* yang dilindungi (dengan dan tanpa token).

Lampiran A

Glosarium Istilah Kunci

Autentikasi Proses verifikasi identitas pengguna.

Autorisasi Proses penentuan hak akses pengguna yang telah terautentikasi.

Hashing Proses transformasi data menjadi string acak satu arah.

Salting Penambahan data acak pada kata sandi sebelum hashing.

bcrypt Algoritma hashing kata sandi yang populer dan aman.

JWT (JSON Web Token) Standar terbuka untuk membuat token akses yang *self-contained*.

Stateless Sifat API yang tidak menyimpan status sesi klien di server.