

wImplementasi 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 2 : Implementasi login + hashing (bcrypt)

1. Kode Implementasi: Fungsi Registrasi

Tampilkan kode fungsi `postRegisterUsers` di sini:



```
1  public async postRegisterUsers(
2    req: Request,
3    res: Response
4  ): Promise<Response> {
5    try {
6      const { username, password, roleId, isActive } =
7        req.body as createUserBody;
8
9      if (!username || !password || !roleId) {
10        return sendError(res, "Mohon isi username, password, dan role.", 400);
11      }
12
13      const cekUsername = await Users.findOne({ where: { username } });
14
15      if (cekUsername) {
16        return sendError(
17          res,
18          "Username sudah digunakan. Silakan pilih yang lain.",
19          409
20        );
21      }
22
23      const usernameCheck = validateUsername(username);
24      if (!usernameCheck.isValid) {
25        return sendError(res, usernameCheck.message!, 400);
26      }
27
28      const passwordCheck = validatePassword(password);
29      if (!passwordCheck.isValid) {
30        return sendError(res, passwordCheck.message || "Password lemah", 400);
31      }
32
33      const passwordHash = await bcrypt.hash(password, 12);
34
35      const newUser = await Users.create({
36        username,
37        password: passwordHash,
38        roleId,
39        isActive,
40      });
41
42      const responseData = {
43        id: newUser.id,
44        username: newUser.username,
45        roleId: newUser.roleId,
46        isActive: newUser.isActive,
47        createdAt: newUser.createdAt,
48      };
49
50      return sendSuccess(res, responseData, "Pengguna berhasil dibuat.", 201);
51    } catch (error) {
52      return sendError(res, "Gagal membuat pengguna", 500, error);
53    }
54  }
```

Penjelasan Singkat Kode (Fokus Utama: Bcrypt)

Fungsi ini menangani pendaftaran pengguna baru dengan prioritas utama pada keamanan kata sandi.

A. Validasi dan Pencegahan Konflik

- **Pengecekan Required Fields:** Memastikan input dasar (`username`, `password`, `roleId`) telah terisi.
- **Pengecekan Keunikan Username:** Mencari `username` di `database`. Jika ditemukan, mengembalikan status **409 Conflict** (sudah digunakan).
- **Validasi Kualitas Kata Sandi:** Menggunakan `validatePassword(password)` untuk memastikan kata sandi pengguna memenuhi standar keamanan minimum (misalnya, panjang dan kompleksitas).

B. Inti Keamanan: Hashing Bcrypt

Ini adalah langkah paling penting untuk melindungi data sensitif pengguna.

- **Baris Kunci:** `const passwordHash = await bcrypt.hash(password, 12);`
- **Mekanisme:**
 - Fungsi `bcrypt.hash()` mengambil kata sandi mentah (`password`) dan mengubahnya menjadi hash yang aman dan tidak dapat di-dekripsi (diubah kembali ke teks asli).
 - Parameter `12` adalah Cost Factor (jumlah *salt rounds*), yang menentukan tingkat kesulitan proses *hashing*. Nilai 12 dipilih untuk memberikan perlindungan kuat terhadap serangan *brute force* karena membutuhkan waktu komputasi yang signifikan.
 - Hasil `passwordHash` (yang mencakup *salt* unik) adalah satu-satunya data yang disimpan di `database` (`newUser.create`). Ini memastikan bahwa kata sandi asli tidak pernah tersimpan, melindungi pengguna dari kebocoran data.

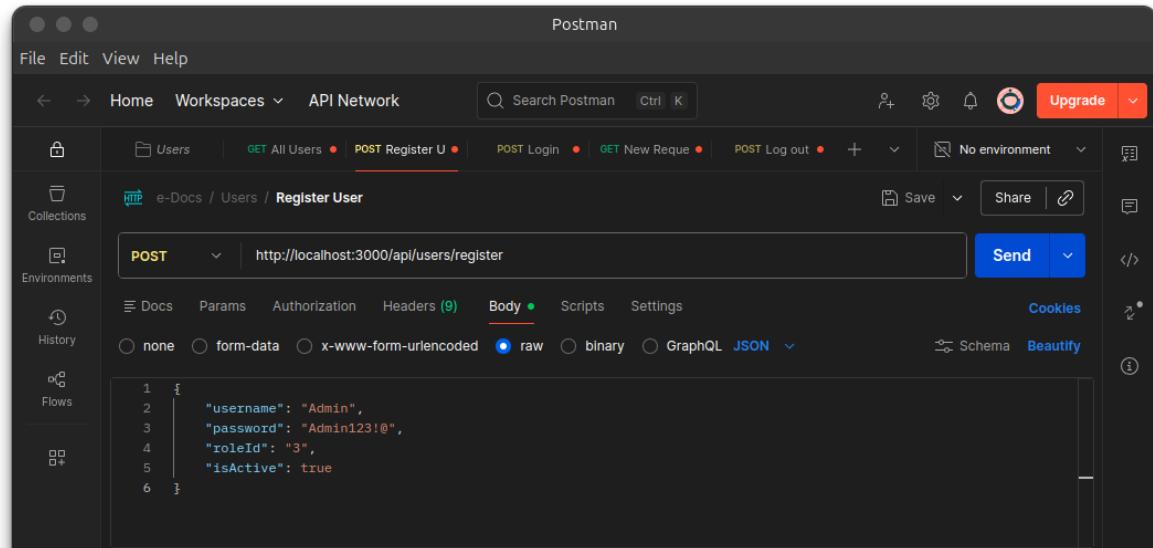
C. Respons

- Setelah berhasil, sistem mengembalikan status **201 Created** dan hanya menampilkan data aman (seperti ID dan `username`), tidak pernah menyertakan kata sandi atau `hash` ke klien.

Contoh Penggunaan di Postman dan Respons

Tujuan: Kode ini memproses permintaan POST ke *endpoint* registrasi (misalnya, /api/users/register).

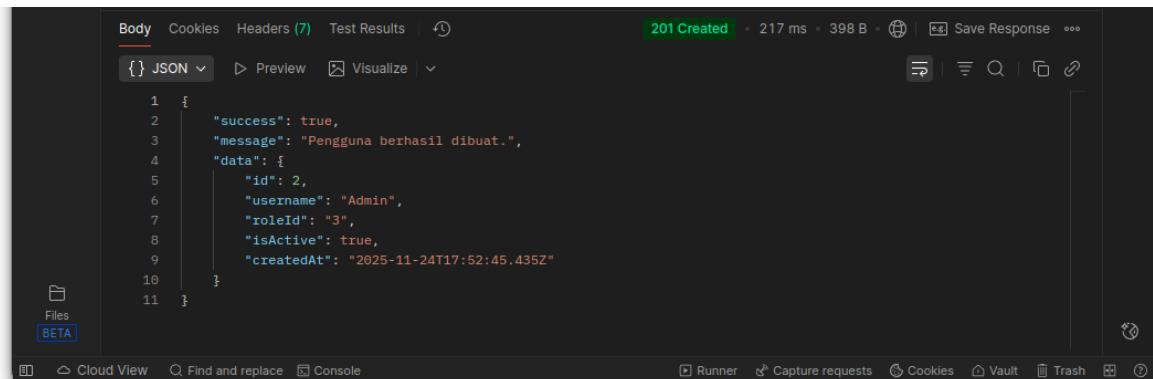
Body Request (JSON): Anda akan mengirim data *username*, *password*, dan *roleId*.



The screenshot shows the Postman application interface. In the center, there is a request card for a POST operation to 'Register User' at 'http://localhost:3000/api/users/register'. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {
2   "username": "Admin",
3   "password": "Admin123!@",
4   "roleId": "3",
5   "isActive": true
6 }
```

Contoh Respons Sukses (Status 201 Created):



The screenshot shows the Postman interface displaying the response to the previous POST request. The status bar indicates '201 Created' with a response time of 217 ms and a size of 398 B. The response body is shown in JSON format:

```
1 {
2   "success": true,
3   "message": "Pengguna berhasil dibuat.",
4   "data": {
5     "id": 2,
6     "username": "Admin",
7     "roleId": "3",
8     "isActive": true,
9     "createdAt": "2025-11-24T17:52:45.435Z"
10   }
11 }
```

2. Kode Implementasi: Login (`postLogin`)

Ini adalah fungsi di *Controller* yang menangani proses otentikasi.

```
● ● ●
1  public async postLogin(req: Request, res: Response): Promise<Response> {
2      try {
3          const { username, password } = req.body as LoginBody;
4
5          const user = await Users.findOne({ where: { username } });
6          if (!user) {
7              return sendError(res, "Username atau Password salah", 401);
8          }
9
10         if (!user.isActive) {
11             return sendError(
12                 res,
13                 "Akun Anda dinonaktifkan. Silakan hubungi Admin.",
14                 403
15             );
16         }
17
18         const isMatch = await bcrypt.compare(password, user.password);
19         if (!isMatch) {
20             return sendError(res, "Username atau Password salah", 401);
21         }
22
23         const payload = {
24             id: user.id,
25             roleId: user.roleId,
26             username: user.username,
27         };
28
29         const token = generateToken(payload);
30
31         return sendSuccess(
32             res,
33             {
34                 token: token,
35                 user: {
36                     id: user.id,
37                     username: user.username,
38                     roleId: user.roleId,
39                 },
40             },
41             "Login Berhasil"
42         );
43     } catch (error) {
44         return sendError(res, "Gagal Login", 500, error);
45     }
46 }
```

Penjelasan Singkat:

Fungsi ini melakukan tiga langkah keamanan utama:

1. **Verifikasi Pengguna:** Mencari *username* di database dan memastikan akun **aktif** (*isActive*).
2. **Verifikasi Kata Sandi (Bcrypt):** Baris `await bcrypt.compare(password, user.password)` membandingkan kata sandi yang dimasukkan dengan *hash* yang tersimpan secara aman. **Ini adalah inti keamanan otentikasi.**
3. **Pemberian Akses (JWT):** Jika verifikasi berhasil, fungsi membuat *payload* dan memanggil `generateToken(payload)` untuk menghasilkan **Token Akses** yang dikirimkan kembali ke pengguna.

Utility Pembuatan JWT (`generateToken`)

Ini adalah fungsi yang membuat token sesi setelah *login* berhasil.

```
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 }
```

Penjelasan Singkat:

Fungsi ini bertanggung jawab untuk mengubah data pengguna menjadi kunci sesi yang aman.

1. **Kunci Rahasia:** Mengambil `SECRET_KEY` dari lingkungan (`process.env`). Kunci ini sangat penting untuk menandatangani (`sign`) token, menjamin keasliannya.
2. **Masa Berlaku:** Mengatur `expiresIn` (masa berlaku token), mencegah token digunakan selamanya jika dicuri.
3. **Penandatanganan:** Menggunakan `jwt.sign()` untuk menggabungkan data pengguna (`payload`) dengan kunci rahasia. Hasilnya adalah JSON Web Token (JWT) yang akan digunakan pengguna untuk mengakses *endpoint* terproteksi.

Contoh Penggunaan di Postman dan Respons

The screenshot shows the Postman application interface. In the top navigation bar, the title is "Login - My Workspace". Below it, the "File", "Edit", "View", and "Help" menus are visible. The "Workspaces" dropdown is set to "e-Docs / auth / Login". The search bar contains "Search Postman". On the right side, there are buttons for "Save", "Share", and "Upgrade". The main workspace shows a POST request to "http://localhost:3000/api/auth/login". The request body is set to "raw" and contains the following JSON:

```
1 {
2   "username": "Admin",
3   "password": "Admin123!@"
4 }
```

Below the request, the "Body" tab of the response panel is selected, showing the JSON response from the server:

```
200 OK 219 ms 523 B Save Response
{} JSON > Preview Visualize
1 {
2   "success": true,
3   "message": "Login Berhasil",
4   "data": {
5     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwigcm9sZUlkIjozLC1c2VybmtzSI6IkFkbWluIiwiaWF0IjoxNzY0MDA4NzI5LCJleHAiOjE3NjQwMDkzMj19.zEDXnI0FsvjoyMwPH4Z4jL9QvOy71sg_0mARS6_tvk",
6     "user": {
7       "id": 2,
8       "username": "Admin",
9       "roleId": 3
10    }
11  }
12 }
```

At the bottom of the interface, there are buttons for "Cloud View", "Find and replace", "Console", "Runner", "Capture requests", "Cookies", "Vault", "Trash", and "Help".

Detail Pengujian:

- **Endpoint:** `POST /api/auth/login` (atau sesuai konfigurasi Anda)
- **Request Body:** Berisi `username` dan `password` yang dikirimkan dalam format JSON.
- **Status Respons:** `200 OK` (Login Berhasil).

- **Data Respons:** Sistem berhasil mengembalikan **token JWT** yang merupakan kunci akses untuk sesi pengguna, beserta data dasar pengguna (**id**, **username**, **roleId**).

3. Implementasi login

A. Fungsi Middleware (`authenticateToken`)

Fungsi ini dijalankan **sebelum controller** diakses, bertugas memverifikasi keaslian Token JWT yang dikirim oleh pengguna.

```

● ● ●

1 import { Request, Response, NextFunction } from "express";
2 import * as jwt from "jsonwebtoken";
3 import { sendError } from "../utils/response.utils";
4
5 export const authenticateToken = (
6   req: Request,
7   res: Response,
8   next: NextFunction
9 ) => {
10   const authHeader = req.headers["authorization"];
11
12   const token = authHeader && authHeader.split(" ")[1];
13
14   if (!token) {
15     return sendError(res, "Akses ditolak. Token tidak ditemukan.", 401);
16   }
17
18   const SECRET_KEY = process.env.JWT_SECRET || "rahasia_negara_api";
19
20   jwt.verify(token, SECRET_KEY, (err: any, user: any) => {
21     if (err) {
22
23       if (err.name === "TokenExpiredError") {
24         return sendError(
25           res,
26           "Sesi Anda telah berakhir. Silakan login ulang.",
27           401
28         );
29       }
30
31       if (err.name === "JsonWebTokenError") {
32         return sendError(res, "Token tidak valid. Silakan login ulang.", 401);
33       }
34
35       return sendError(res, "Autentikasi gagal.", 401);
36     }
37
38     (req as any).user = user;
39     next();
40   });
41 };
42

```

Penjelasan Singkat:

- **Pengecekan Token:** Kode mengambil token dari *header Authorization* (format: `Bearer [token]`). Jika token tidak ada, akses ditolak (`401`).
- **Verifikasi JWT:** Fungsi `jwt.verify()` mencoba memecahkan token menggunakan **Kunci Rahasia Server**.
 - Jika verifikasi gagal (misalnya, token kedaluwarsa atau dimanipulasi), akses ditolak (`401`).
 - Jika berhasil, *payload* token (informasi pengguna: `id`, `roleId`, dll.) dilampirkan ke objek *request* (`(req as any).user = user;`) dan fungsi memanggil `next()`, mengizinkan akses ke *controller*.

B. Router ([users.route.ts](#))

Kode ini mendefinisikan *endpoint* dan menerapkan *middleware* proteksi.

```
● ● ●  
1 import { Router } from "express";  
2 import UserController from "../controllers/users.controller";  
3 import { authenticateToken } from "../middleware/auth.middleware";  
4  
5 const router = Router();  
6  
7 router.get(  
8   "/",
9   authenticateToken,
10  UserController.getAllUsers.bind(UserController)
11 );
12  
13 export default router;
```

Penjelasan Singkat:

- **Proteksi Rute:** Perhatikan *route* `router.get("/")`. *Middleware* `authenticateToken` diletakkan di antara *path* dan *controller*.
- **Fungsi:** Ini memastikan bahwa siapa pun yang mencoba mengakses `/api/users` (untuk memanggil `UserController.getAllUsers`) WAJIB memiliki Token JWT yang valid dan belum kedaluwarsa. Tanpa token yang valid, permintaan akan dihentikan oleh *middleware* pada Kode 1.

C. Controller (getAllUsers)

Fungsi ini dijalankan **HANYA JIKA** *middleware* pada Kode 1 berhasil diverifikasi.

```
● ● ●
1  public async getAllUsers(req: Request, res: Response): Promise<Response> {
2    try {
3      const users = await Users.findAll({
4        include: [
5          {
6            model: Roles,
7            as: "role",
8            attributes: ["name"],
9          },
10         ],
11        attributes: { exclude: ["password", "updatedAt"] },
12      });
13
14      return sendSuccess(
15        res,
16        users,
17        "Berhasil mengambil data pengunjung",
18        200,
19        {
20          total: users.length,
21        }
22      );
23    } catch (error) {
24      return sendError(res, "Gagal Mengambil data pengguna", 500, error);
25    }
26  }
```

Penjelasan Singkat:

- **Fungsi:** Bertugas mengambil semua data pengguna (`Users.findAll`) dan menyertakan data *role* terkait.
- **Keamanan Data:** Penggunaan `attributes: { exclude: ["password", "updatedAt"] }` memastikan bahwa kata sandi pengguna tidak pernah dikirim dalam respons API, bahkan kepada pengguna yang sudah terotentikasi.
- **Akses Terjamin:** Karena *route* ini dilindungi oleh `authenticateToken`, kita yakin bahwa *request* yang mencapai fungsi ini adalah permintaan yang sah.

4. Contoh Penggunaan di Postman: Akses dengan Token JWT

The screenshot shows the Postman application interface with the following details:

- Workspace:** All Users - My Workspace
- Request Type:** GET
- URL:** http://localhost:3000/api/users
- Authorization:** Bearer Token (Token field contains a redacted token)
- Response Status:** 200 OK
- Response Body (JSON):**

```
1 {
2   "success": true,
3   "message": "Berhasil mengambil data pengunjung",
4   "data": [
5     {
6       "id": 2,
7       "username": "Admin",
8       "roleId": 3,
9       "isActive": true,
10      "createdAt": "2025-11-24T17:52:45.000Z",
11      "role": {
12        "name": "admin"
13      }
14    ],
15    "metadata": {
16      "total": 1
17    }
18  }
19 }
```

Detail Pengujian:

- **Endpoint:** GET /api/users
- **Header Authorization:** Token JWT yang diperoleh setelah *login* berhasil disisipkan di sini dengan format Bearer [token_jwt_anda].
- **Status Respons:** 200 OK.
- **Data Respons:** Berhasil mengambil daftar pengguna, sesuai dengan fungsionalitas getAllUsers.

5. Contoh Penggunaan di Postman: Akses Tanpa Token JWT

The screenshot shows the Postman application window titled "All Users - My Workspace". In the center, there is a request card for a "GET All Users" endpoint. The URL is set to "http://localhost:3000/api/users". The "Authorization" tab is selected, showing "Bearer Token" as the auth type and a field labeled "Token" which is empty. Below the request card, the response pane displays a 401 Unauthorized status with the message: "The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization." The response body is a JSON object with "success": false and "message": "Akses ditolak. Token tidak ditemukan." (Access denied. Token not found.).

Detail Pengujian:

- **Endpoint:** GET /api/users
- **Header Authorization:** Tidak disertakan atau disertakan dengan token yang tidak valid/kedaluwarsa.
- **Status Respons:** 401 Unauthorized.
- **Data Respons:** Sistem mengembalikan pesan error dari *middleware authenticateToken* (misalnya, "Akses ditolak. Token tidak ditemukan." atau "Token tidak valid.").

6. Mekanisme Proteksi Token JWT

Mekanisme ini bekerja dengan memanfaatkan tiga komponen utama yang telah kita bahas sebelumnya:

- **Token JWT dari Login:** Setelah pengguna berhasil *login*, mereka menerima sebuah **Token JWT**. Ini adalah bukti identitas mereka.
- **authenticateToken Middleware:** Ini adalah penjaga gerbang. Sebelum setiap permintaan ke *endpoint* yang dilindungi (seperti GET /api/users), *middleware* ini akan:
 1. Mengekstrak Token JWT dari *header Authorization* (*request*).
 2. Memverifikasi Token JWT tersebut menggunakan kunci rahasia server.

3. Jika token valid dan belum kedaluwarsa, permintaan diizinkan untuk melanjutkan ke *controller*.
 4. Jika token tidak ada, tidak valid, atau kedaluwarsa, *middleware* akan segera menghentikan permintaan dan mengirimkan respons error ([401 Unauthorized](#) atau [403 Forbidden](#)).
- **Controller yang Aman (`getAllUsers`):** Fungsi *controller* `getAllUsers` hanya akan dieksekusi jika token sudah berhasil diverifikasi oleh *middleware*. Ini menjamin bahwa data pengguna hanya dapat diakses oleh pengguna yang sudah terotentikasi.