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 3 : Implementasi role & protected route

1. Implementasi Role

A. Middleware Autentikasi: authenticateToken

Fungsi ini adalah lapisan pertahanan pertama yang mengatur apakah seorang pengguna login atau tidak, terlepas dari peran mereka.

```
export const authenticateToken = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const authHeader = req.headers["authorization"];

  const token = authHeader && authHeader.split(" ")[1];

  if (!token) {
    return sendError(res, "Akses ditolak. Token tidak ditemukan.", 401);
  }

  const SECRET_KEY = process.env.JWT_SECRET || "rahasia_negara_api";

  jwt.verify(token, SECRET_KEY, (err: any, user: any) => {
    if (err) {
      if (err.name === "TokenExpiredError") {
        return sendError(
          res,
          "Sesi Anda telah berakhir. Silakan login ulang.",
          401
        );
      }

      if (err.name === "JsonWebTokenError") {
        return sendError(res, "Token tidak valid. Silakan login ulang.", 401);
      }

      return sendError(res, "Autentikasi gagal.", 401);
    }

    (req as any).user = user;
    next();
  });
};
```

Komponen	Penjelasan Singkat	Fungsi Utama
authenticateToken	Middleware Autentikasi. Berjalan untuk semua *route* terproteksi.	Memverifikasi token JWT yang dikirim oleh pengguna.
Aksi Utama	Mengekstrak token dari *header* Authorization dan memverifikasinya menggunakan kunci rahasia (JWT_SECRET).	"Jika token valid, *payload* data pengguna (id, roleName, dll.) dilampirkan ke objek req.user."
Gagal	"Jika token hilang, palsu, atau kedaluwarsa."	Mengembalikan **401 Unauthorized**.
Berhasil	Token valid.	Memanggil next() untuk melanjutkan ke *middleware* berikutnya (misalnya authorizeRole).

B. Middleware Otorisasi: authorizeRole

Fungsi ini adalah mekanisme **Role-Based Access Control (RBAC)** yang membatasi akses berdasarkan peran yang diizinkan.

```
export const authorizeRole = (allowedRoles: string[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const userRole = req.user?.roleName;

    if (!userRole) {
      return sendError(
        res,
        "Akses Ditolak: Informasi peran tidak ditemukan.",
        403
      );
    }

    if (allowedRoles.includes("all")) {
      return next();
    }

    if (allowedRoles.includes(userRole)) {
      next();
    } else {
      return sendError(
        res,
        "Akses Ditolak: Peran Anda tidak memiliki izin untuk aksi ini.",
        403
      );
    }
  };
};
```

Komponen	Penjelasan Singkat	Fungsi Utama
authorizeRole(allowedRoles)	"Middleware Otorisasi. Menerima *array* peran yang diizinkan (['admin', 'editor'])."	Membandingkan peran pengguna (req.user.roleName) dengan peran yang

		diizinkan untuk *route* tersebut.
Aksi Utama	Mengambil peran (userRole) dari req.user (yang sudah disediakan oleh authenticateToken).	"Jika userRole ditemukan di allowedRoles, akses diizinkan."
Kasus ['all']	"Mengizinkan semua peran yang terautentikasi (Admin, Editor, Viewer) untuk melanjutkan."	Digunakan pada *route* umum (misalnya GET /documents).
Gagal	"Peran pengguna (viewer) tidak termasuk dalam daftar yang diizinkan (['admin']). "	"Mengembalikan **403 Forbidden** ("Akses Ditolak")."
Berhasil	Peran diizinkan.	Memanggil next() untuk melanjutkan ke *controller* (misalnya createDocument).

2. Contoh Penggunaan (Di Route)

Kode ini mendefinisikan *endpoint* untuk manajemen dokumen dan mengatur dua lapisan keamanan di *middleware*: Autentikasi (authenticateToken) dan Otorisasi (authorizeRole).

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

const router = Router();

router.get(
  "/",
  authenticateToken,
  documentsController.getAllDocuments.bind(documentsController)
);

router.get(
  "/:id",
  authenticateToken,
  documentsController.getDocumentById.bind(documentsController)
);

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

router.put(
  "/:id",
  authorizeRole(["admin", "editor"]),
  documentsController.updateDocument.bind(documentsController)
);

export default router;

```

Endpoint	Metode	Middleware yang Diterapkan	Peran yang Diizinkan	Penjelasan RBAC
GET /	GET	authenticateToken	Semua Role	Mengambil daftar dokumen. Hanya memerlukan *login* saja. (Akses dibatasi ke data yang *Approved* di dalam *controller*).
GET /:id	GET	authenticateToken	Semua Role	Mengambil detail dokumen tunggal. Hanya memerlukan *login* saja.
POST /create	POST	authorizeRole(['admin', 'editor'])	Admin, Editor	Membuat dokumen baru. *Middleware* ini memastikan hanya peran kontributor yang dapat mengakses.

PUT /:id	PUT	authorizeRole(['admin', 'editor'])	Admin, Editor	Memperbarui dokumen. *Middleware* memastikan hanya peran yang memiliki hak *edit* yang dapat memanggil *route* ini.

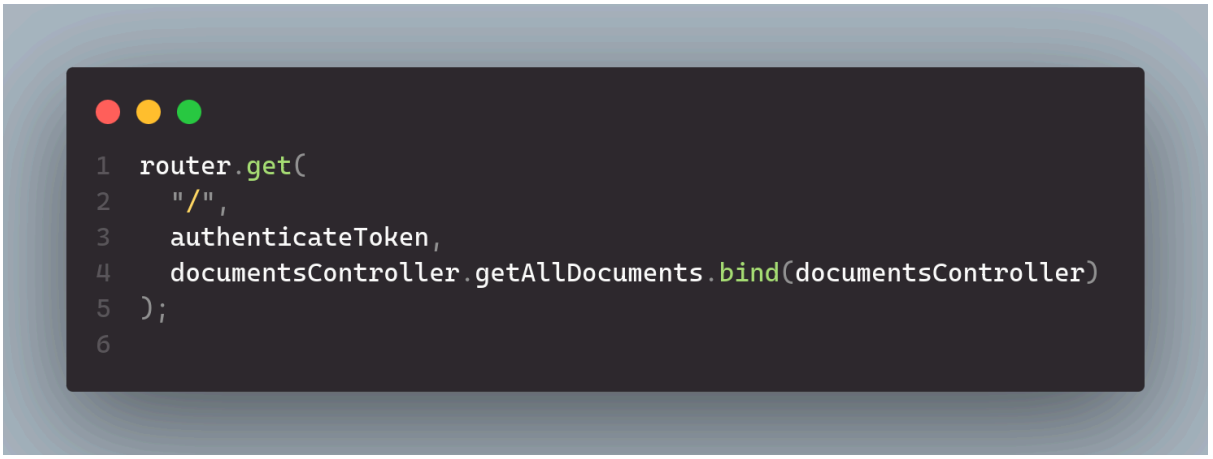
3. Verifikasi Implementasi

A. Pengujian Role-Based Access Control (RBAC) di Postman

Pengujian ini bertujuan untuk memverifikasi fungsionalitas dari kedua lapisan keamanan (*middleware* authenticateToken dan authorizeRole) yang telah diimplementasikan.

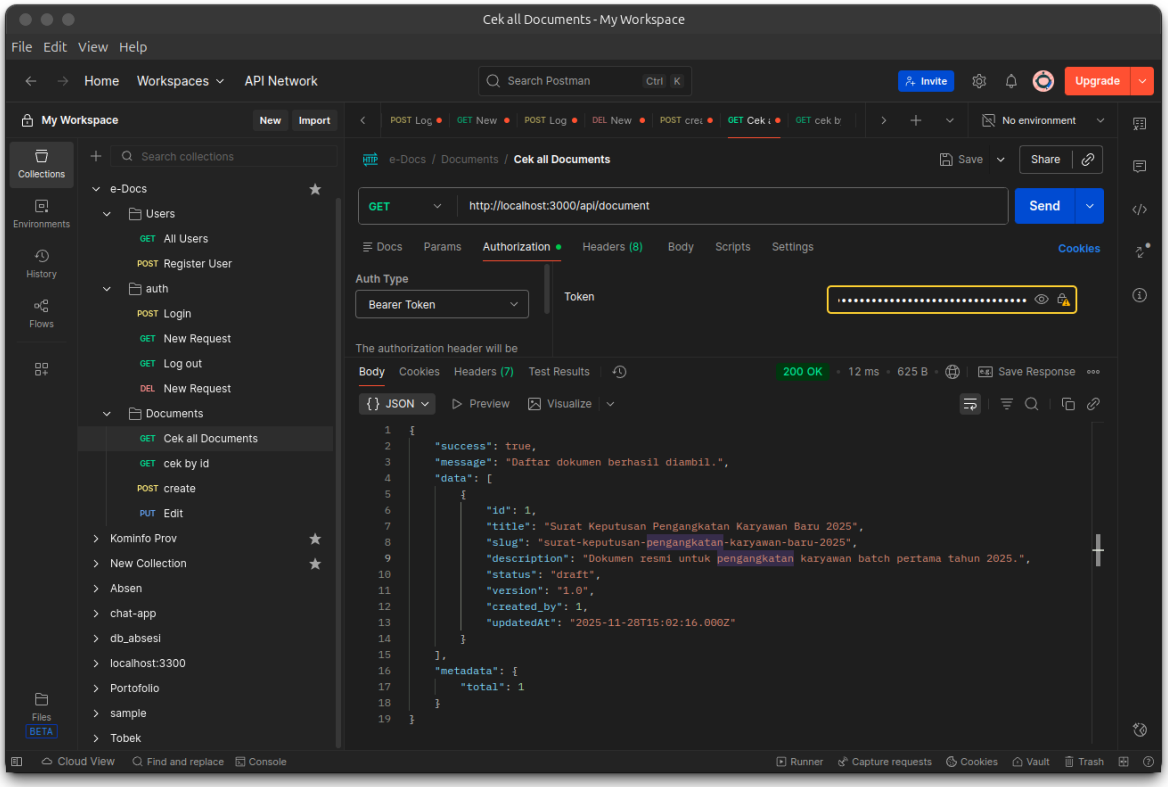
1. Proses cek semua dokumen

Sesuai konfigurasi *router*, *route* GET / hanya dilindungi oleh *middleware* **authenticateToken**. Logika ini hanya memeriksa apakah Token JWT valid dan belum kedaluwarsa. Setelah token valid, permintaan dilanjutkan ke *controller*.



a. Sebagai viewer

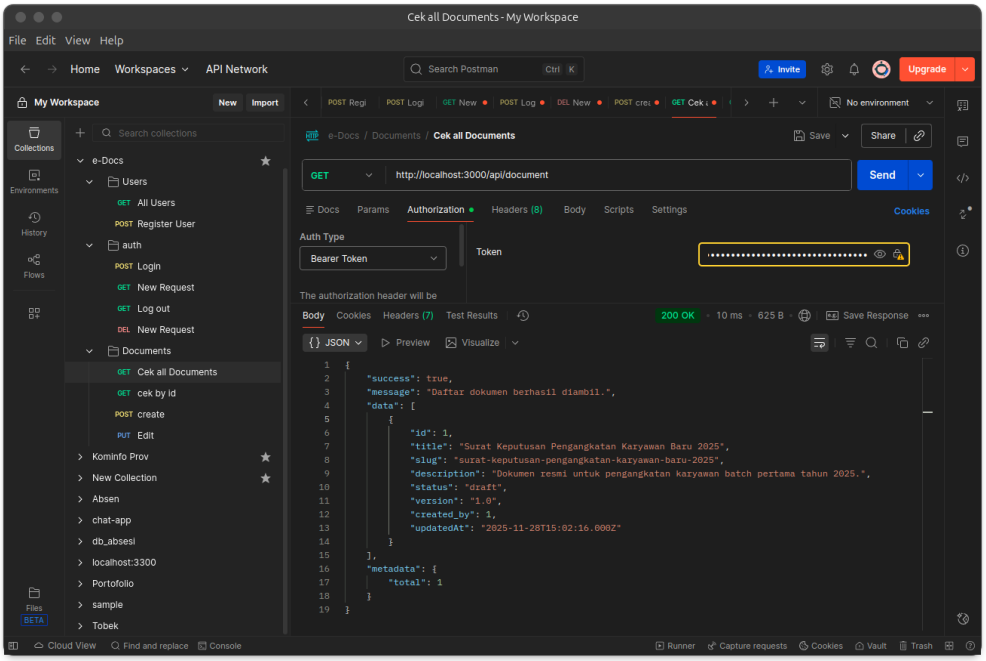
Hasil Pengujian : Status **200 OK**. Berhasil melihat data dokumen.



Akses berhasil karena hanya diperlukan **Autentikasi** sesi aktif. Viewer memiliki hak akses dasar untuk melihat data.

b. Sebagai Editor

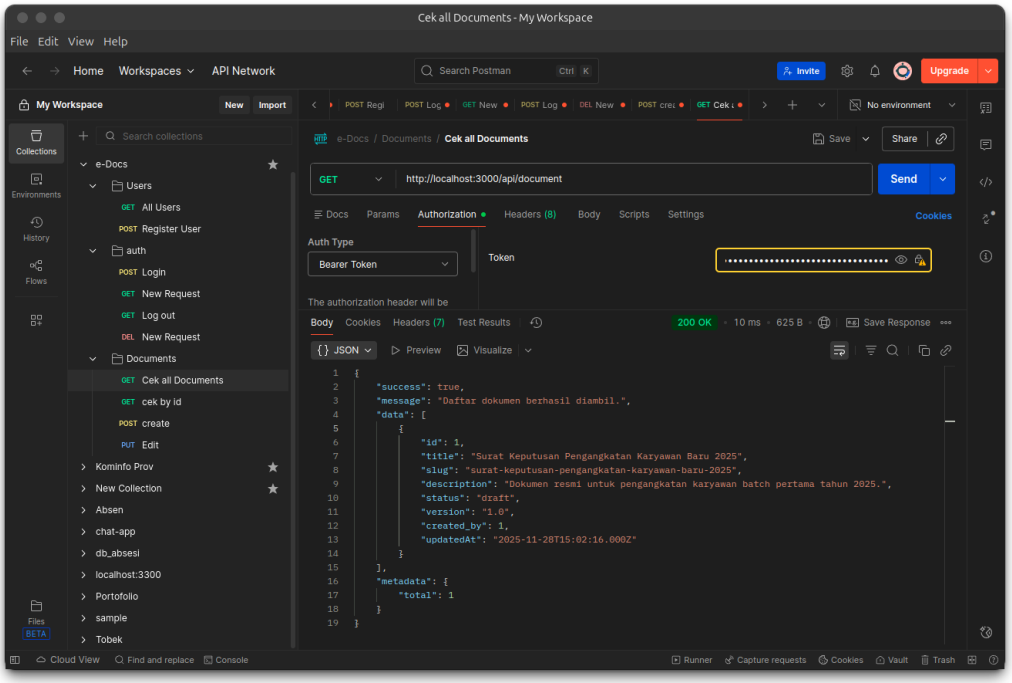
Hasil Pengujian : Status **200 OK** Berhasil melihat data dokumen.



Akses berhasil. Semua peran yang terotentikasi diizinkan melihat daftar dokumen.

c. Sebagai Admin

Hasil Pengujian : Status **200 OK** Berhasil melihat data dokumen.



Akses berhasil. Admin memiliki hak paling tinggi, sehingga lolos tahap autentikasi.

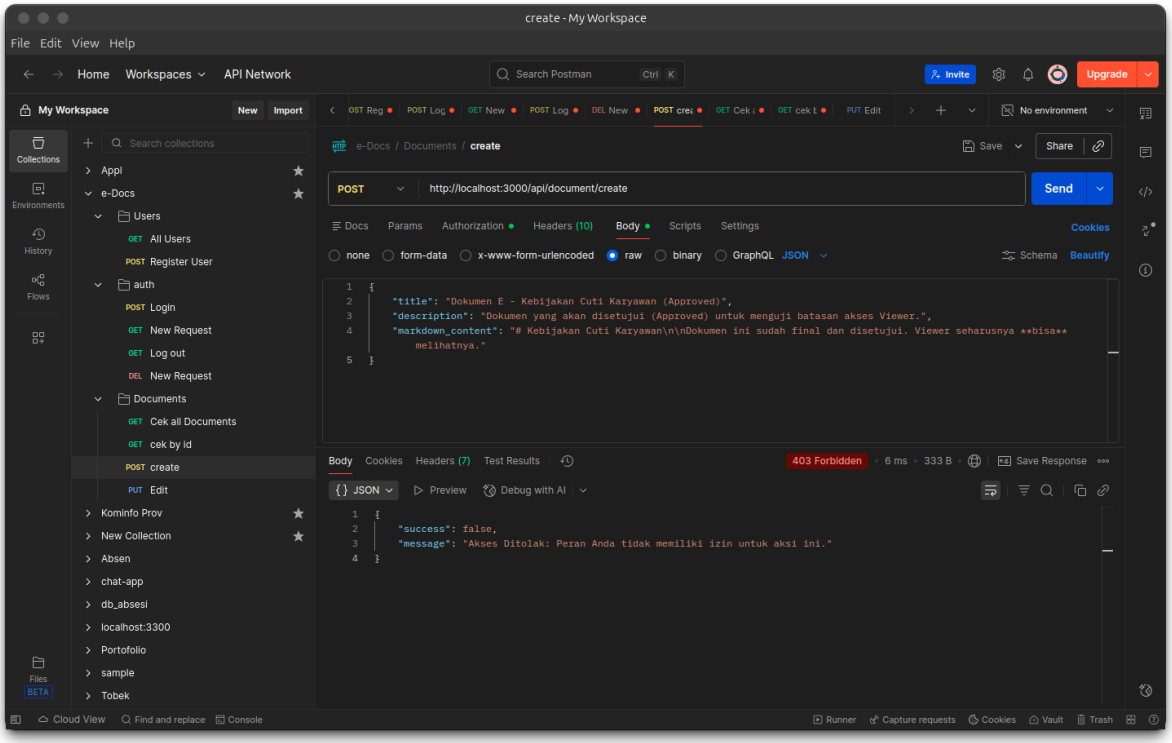
2. Pengujian untuk create documents

Route POST /create dilindungi oleh **dua** *middleware*: authenticateToken dan **authorizeRole(['admin', 'editor'])**. Setelah token diverifikasi, authorizeRole akan memeriksa apakah peran pengguna *secara eksplisit* termasuk dalam daftar ['admin', 'editor'].

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

a. Sebagai Viewer

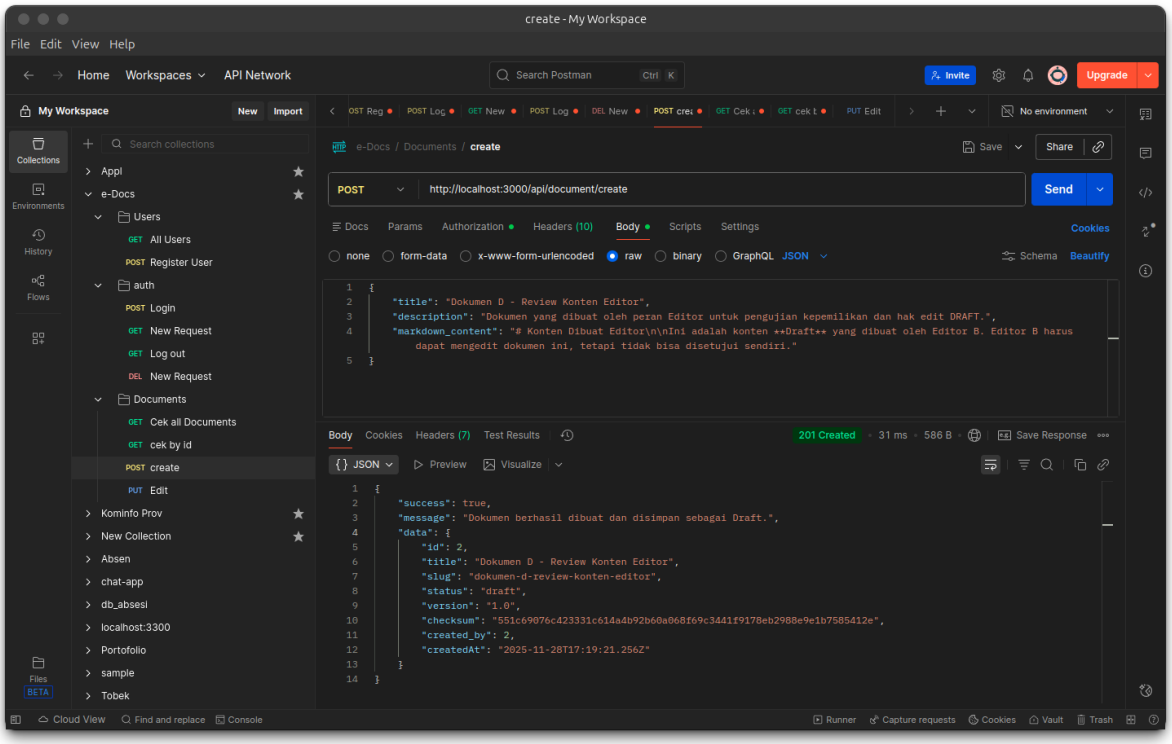
Hasil Pengujian : Status 403 Forbidden. Respons: "Akses Ditolak: Peran Anda tidak memiliki izin untuk aksi ini."



Akses Ditolak. Ini adalah bukti kunci penerapan Zero Trust. Meskipun *Viewer* berhasil Terautentikasi (memiliki Token JWT valid), ia gagal di lapisan Otorisasi (RBAC). Sistem menolak akses karena *Viewer* tidak diberikan hak istimewa (privilege) untuk membuat dokumen.

b. Sebagai Editor

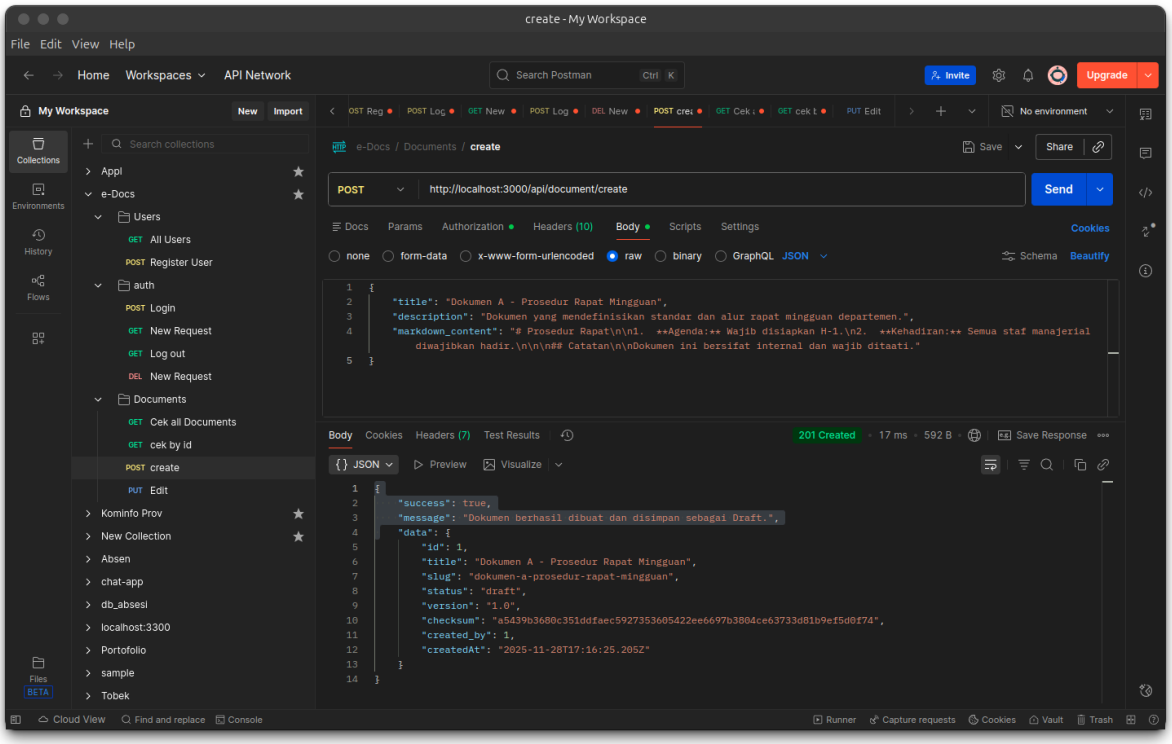
Hasil Pengujian : Status 201 Created.



Akses Diizinkan. Editor adalah peran kontributor, membuktikan otorisasi berhasil.

c. Sebagai Admin

Hasil Pengujian : Status 201 Created.



Akses Diizinkan. Admin memiliki izin kontributor. Ini adalah kasus sukses yang memverifikasi bahwa peran yang berhak dapat mengakses fungsi kritis.