LAPORAN UAS PEMROGRAMAN API

Disusun Oleh:

1. Rizka Ananda Putri (23104410065)
2. Yasa Handoyo (23104410066)
3. Rasti Ayu Nur Rahmadani (23104410078)
4. Muhammad Aditya Rizqi Fauzi (231004410079)
5. Wisnu Nur Pradana (23104410103)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN INFORMATIKA

UNIVERSITAS ISLAM BALITAR

2025

BAB I

PENDAHULUAN

1.1 Latar Belakang

Di era pengembangan aplikasi web modern, arsitektur perangkat lunak telah bertransformasi ke arah sistem terdistribusi dengan memisahkan sisi klien (*frontend*) dan server (*backend*). Pendekatan ini bertujuan untuk meningkatkan skalabilitas serta fleksibilitas pemeliharaan kode. Dalam ekosistem tersebut, REST API menjadi standar krusial untuk memfasilitasi komunikasi data dan interoperabilitas antara antarmuka pengguna dengan sumber daya server. Oleh karena itu, penguasaan dalam membangun REST API yang responsif dan aman merupakan kompetensi fundamental bagi setiap pengembang perangkat lunak.

Laporan ini mendokumentasikan proyek Ujian Akhir Semester (UAS) yang berfokus pada pengembangan *backend* **Sistem Manajemen Tugas** menggunakan kerangka kerja **Next.js**. Melalui fitur **API Routes**, Next.js memungkinkan pembangunan logika server yang efisien tanpa memerlukan server eksternal tambahan. Untuk pengelolaan data, sistem ini mengintegrasikan basis data **PostgreSQL** dengan **Prisma ORM**, yang mempermudah interaksi basis data sekaligus menjamin keamanan tipe (*type-safety*) guna meminimalisir kesalahan selama proses pengembangan.

Selain fungsionalitas CRUD (*Create, Read, Update, Delete*) pada entitas tugas, aspek keamanan menjadi prioritas utama untuk melindungi integritas data pengguna. Sistem ini menerapkan lapisan keamanan berlapis, mulai dari enkripsi kata sandi menggunakan **bcrypt**, manajemen sesi berbasis **JSON Web Token (JWT)** yang bersifat *stateless*, hingga penerapan **Middleware** untuk otorisasi berbasis peran (*Role-Based Access Control*). Mekanisme ini memastikan perbedaan hak akses yang ketat antara peran **Admin** dan **User**.

Melalui pembahasan yang sistematis, laporan ini akan menguraikan proses perancangan hingga implementasi REST API untuk Sistem Manajemen Tugas. Tujuan akhirnya adalah menghasilkan infrastruktur *backend* yang efisien, terstruktur, dan memiliki ketahanan tinggi terhadap potensi celah keamanan.

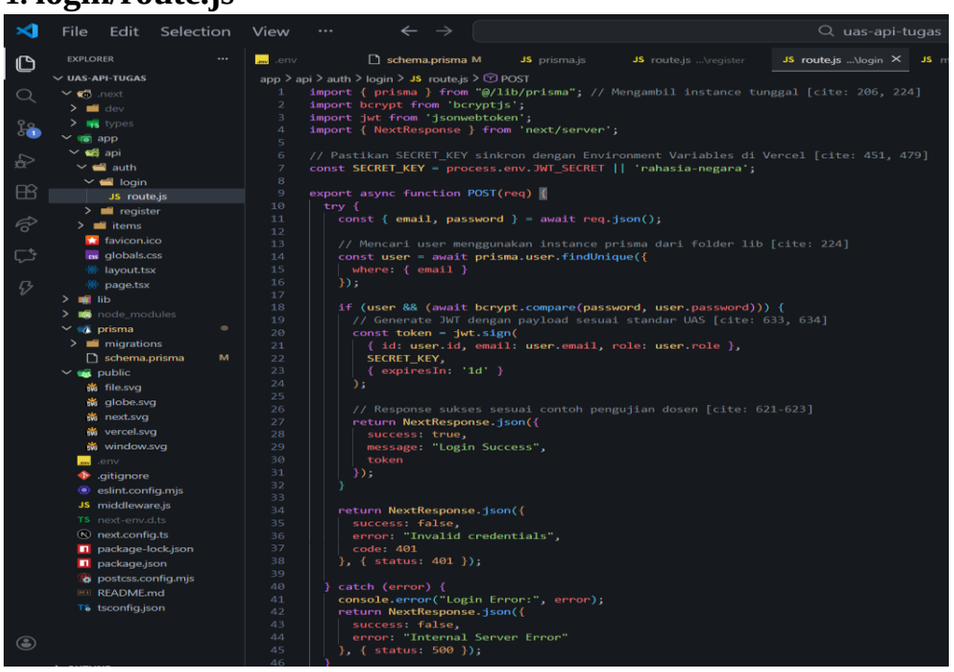
BAB II

PEMBAHASAN

2.1 Penjelasan Coding dan Codingnya

1.app/api/auth/…

1. login/route.js



Penjelasanya :

- **prisma**: Instance dari Prisma Client untuk melakukan kueri ke database PostgreSQL.
- **bcrypt**: Pustaka keamanan yang digunakan untuk membandingkan kata sandi yang diinput pengguna dengan kata sandi yang sudah di-*hash* di database.
- **jwt**: Digunakan untuk menghasilkan JSON Web Token sebagai bukti sesi login yang sah.
- **NextResponse**: Fitur Next.js untuk mengirimkan respon HTTP (seperti JSON sukses atau pesan error) kembali ke sisi klien.

2. Inisialisasi Kunci Rahasia (Baris 6 - 7)

- **SECRET_KEY**: Mengambil nilai dari *Environment Variable* (JWT_SECRET). Jika tidak ditemukan di server, kode menggunakan nilai *fallback* 'rahasia-negara'. Kunci ini sangat penting untuk menandatangani token JWT agar tidak mudah dipalsukan.

- **Penanganan Request & Ekstraksi Data (Baris 9 - 11)**
- **export async function POST(req)**: Mendefinisikan bahwa fungsi ini hanya menangani permintaan dengan metode POST (standar untuk mengirim data sensitif seperti login).
- **req.json()**: Mengambil data email dan password yang dikirimkan oleh pengguna melalui badan permintaan (*request body*).

4. Pencarian dan Verifikasi Pengguna (Baris 13 - 18)

- **prisma.user.findUnique**: Sistem mencari data pengguna di dalam database berdasarkan email yang diinputkan.
- **Validasi Ganda**: Di baris 18, kode melakukan pengecekan apakah pengguna ditemukan **DAN** apakah hasil perbandingan password menggunakan bcrypt.compare bernilai benar (cocok).

5. Pembuatan JWT Token (Baris 19 - 25) Jika verifikasi berhasil, sistem akan membuat "token akses":

- **Payload**: Memasukkan data identitas pengguna seperti id, email, dan role ke dalam token.
- **expiresIn: '1d'**: Token diatur agar kedaluwarsa dalam waktu 1 hari.
- **jwt.sign**: Proses penandatanganan token menggunakan SECRET_KEY agar integritas data di dalam token terjamin.

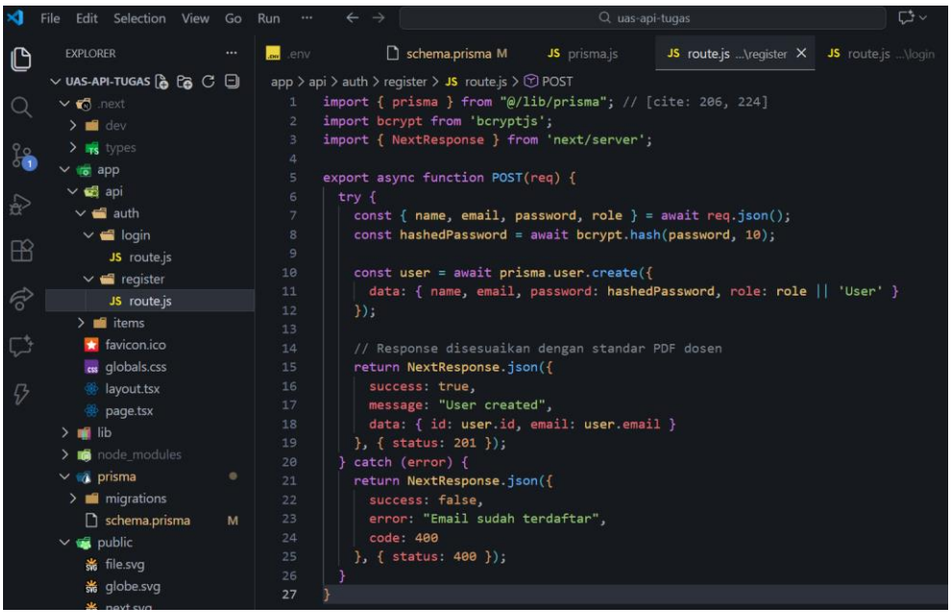
6. Respon Berhasil & Gagal (Baris 27 - 38)

- **NextResponse.json (Sukses)**: Jika login benar, sistem mengirimkan status 200 beserta pesan "Login Success" dan token JWT untuk disimpan di sisi klien.
- **Status 401 (Invalid Credentials)**: Jika email atau password salah, sistem akan melewati blok sukses dan mengirimkan respon error 401 agar pengguna tahu bahwa data yang dimasukkan salah.

7. Penanganan Error Sistem (Baris 40 - 45)

- **catch (error)**: Blok ini menangkap segala jenis kesalahan teknis yang tidak terduga (misalnya koneksi database terputus).
- **Status 500**: Mengembalikan pesan "Internal Server Error" untuk mencegah aplikasi klien berhenti berfungsi secara mendadak saat terjadi masalah di server.

B. register/route.js



Penjelasan :

1. **Impor Modul (Baris 1 - 3)** Bagian ini memanggil pustaka (*library*) utama yang dibutuhkan untuk proses pendaftaran pengguna:
   - **prisma**: Digunakan untuk berinteraksi dengan basis data (melakukan operasi *insert* data pengguna baru).
   - **bcrypt**: Pustaka untuk melakukan *hashing* (pengacakan) kata sandi sebelum disimpan ke database agar lebih aman.
   - **NextResponse**: Fitur dari Next.js untuk mengirimkan respon balik ke *client* dalam format JSON.

2. Penanganan Request & Ekstraksi Data (Baris 5 - 7)

- **export async function POST(req)**: Menetapkan bahwa fungsi ini bertugas menangani permintaan dengan metode POST (metode standar untuk mengirim data pendaftaran).
- **req.json()**: Mengambil data pendaftaran yang dikirim pengguna dari *body request*, yaitu: name, email, password, dan role.

3. Enkripsi Kata Sandi (Baris 8)

bcrypt.hash(password, 10): Sebelum disimpan ke database, kata sandi diubah menjadi kode acak (*hash*). Angka 10 adalah *salt rounds* yang menentukan tingkat kekuatan enkripsi. Hal ini dilakukan agar administrator atau peretas tidak dapat melihat kata sandi asli pengguna.

4. Pembuatan Data Pengguna Baru (Baris 10 - 12)

- **prisma.user.create**: Fungsi dari Prisma untuk memasukkan baris data baru ke dalam tabel user di database PostgreSQL.
- **role || 'User'**: Sistem memberikan logika *default*. Jika pengguna tidak

mencantumkan peran (*role*), maka secara otomatis akan didaftarkan sebagai **'User'**.

5. Respon Sukses Pendaftaran (Baris 15 - 19)

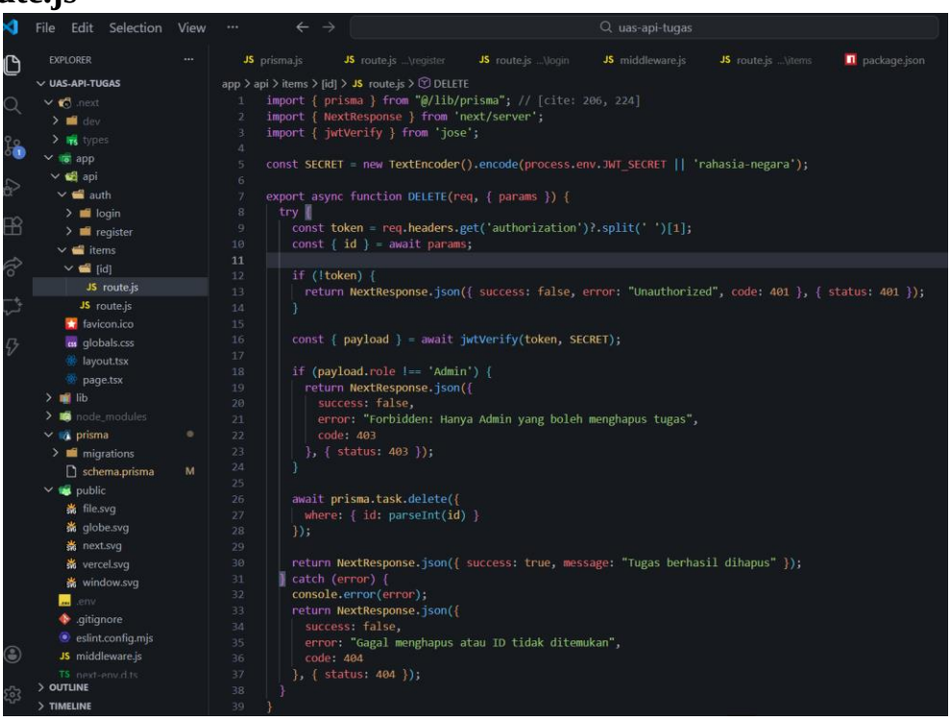
- **NextResponse.json (status: 201)**: Jika data berhasil masuk ke database, server akan mengirimkan kode status **201 (Created)**.
- **Output Data**: Respon sukses ini menyertakan pesan "User created" serta mengembalikan informasi publik pengguna (ID dan Email) untuk keperluan konfirmasi di sisi aplikasi.

6. Penanganan Kesalahan / Error (Baris 20 - 25)

- **catch (error)**: Jika terjadi kesalahan saat proses pendaftaran (misalnya email sudah digunakan atau koneksi database terputus).
- **Status 400**: Mengembalikan pesan "Email sudah terdaftar" atau pesan kesalahan lainnya dengan kode status 400 (*Bad Request*), yang menandakan permintaan pendaftaran gagal diproses.

2. authors/[id]/…

A. route.js



Penjelasan :

1. Impor Modul (Baris 1 - 3) Bagian ini memanggil pustaka yang diperlukan untuk operasi penghapusan dan validasi:

- prisma: Digunakan untuk mengakses tabel tugas di database dan menjalankan perintah hapus.
- NextResponse: Digunakan untuk mengirimkan status respon kembali ke pengguna.
- jwtVerify: Fungsi dari pustaka jose yang digunakan untuk memeriksa apakah token JWT yang dikirim pengguna asli dan belum kedaluwarsa.

2. Inisialisasi Keamanan (Baris 5)

SECRET: Mengambil kunci rahasia (JWT_SECRET) dari environment variable dan mengubahnya menjadi format byte (*TextEncoder*) agar dapat dibaca oleh pustaka jose. Ini berfungsi sebagai "kunci gembok" untuk membuka data di dalam token.

3. Ekstraksi Token dan ID (Baris 7 - 10)

- DELETE(req, { params }): Fungsi ini menangani metode DELETE. Parameter params digunakan untuk menangkap ID tugas yang ada pada URL (misal: /api/items/5).
- req.headers.get('authorization'): Mengambil token dari *header* permintaan. Kode .split(' ')[1] digunakan untuk mengambil kode tokennya saja (membuang kata "Bearer").

4. Validasi Autentikasi (Baris 12 - 16)

- Cek Keberadaan Token: Jika pengguna mencoba menghapus tanpa mengirimkan token, sistem akan langsung menolak dengan status 401 (Unauthorized) dan pesan "Unauthorized".
- jwtVerify: Sistem memverifikasi token tersebut menggunakan kunci rahasia. Jika token valid, data pengguna (payload) akan diekstraksi.

5. Otorisasi Berbasis Peran / RBAC (Baris 18 - 24)

- Pengecekan Admin: Sesuai dengan narasi laporan Anda mengenai *Role-Based Authorization*, bagian ini memastikan hanya pengguna dengan peran 'Admin' yang boleh menghapus tugas.
- Jika pengguna memiliki peran selain Admin, sistem mengirim status 403 (Forbidden) dengan pesan penolakan akses.

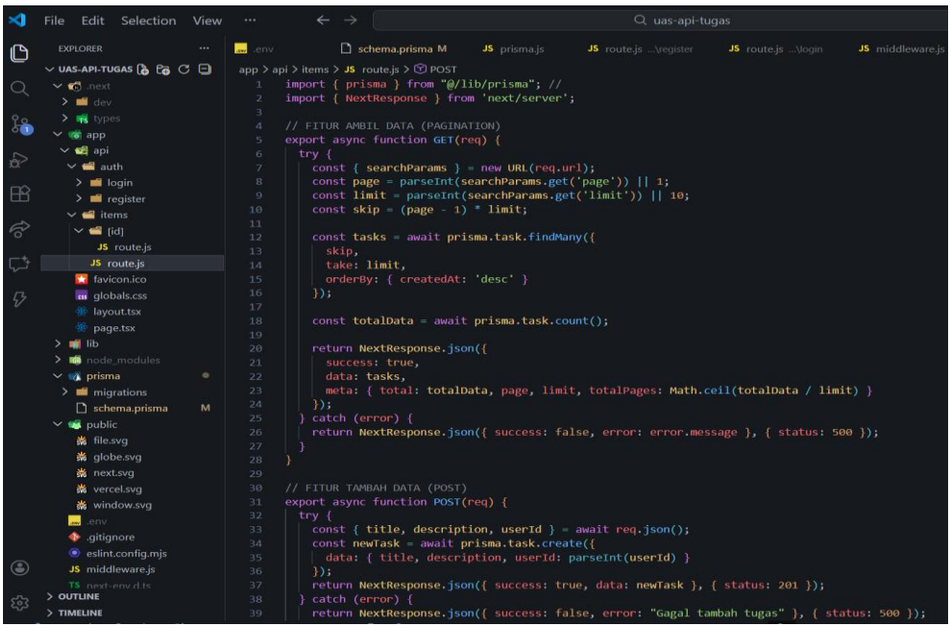
6. Operasi Penghapusan Data (Baris 26 - 30)

- prisma.task.delete: Jika semua validasi lolos, sistem akan menghapus data dari tabel task.
- parseInt(id): Mengubah ID dari teks (string) menjadi angka (integer) agar sesuai dengan tipe data di database.
- Respon Sukses: Jika berhasil, mengirim pesan "Tugas berhasil dihapus".

7. Penanganan Error (Baris 31 - 38)

- catch (error): Jika terjadi kegagalan (misalnya ID yang ingin dihapus tidak ada di database).
- Status 404: Mengembalikan status 404 (Not Found) dengan pesan "Gagal menghapus atau ID tidak ditemukan".

B. route.js



Penjelasan :

Berikut adalah penjelasan untuk file Items Main Route (app/api/items/route.js) berdasarkan gambar kode yang Anda unggah. File ini menangani pengelolaan data tugas secara kolektif, mencakup pengambilan daftar tugas dengan sistem *pagination* serta penambahan tugas baru.

Penjelasan Kode:

1. Impor Modul (Baris 1 - 2) Bagian ini memanggil komponen dasar yang diperlukan untuk operasional API:

- prisma: *Interface* yang diimpor dari folder @/lib/prisma untuk berinteraksi dengan database PostgreSQL.
- NextResponse: Digunakan untuk mengirimkan respon HTTP kembali ke sisi klien.

2. Fungsi GET - Mengambil Daftar Tugas dengan Pagination (Baris 5 - 28) Fungsi ini digunakan untuk menampilkan daftar tugas secara teratur dan efisien:

- Ekstraksi Parameter: Mengambil nilai page dan limit dari URL permintaan menggunakan URL(req.url).searchParams. Jika tidak ditentukan, sistem menggunakan nilai *default* halaman 1 dengan limit 10 data.
- Logika Skip & Take: Menghitung jumlah data yang harus dilewati (skip) untuk mendukung perpindahan halaman.
- Prisma findMany: Mengambil data tugas dari database dengan urutan terbaru (createdAt: 'desc') berdasarkan batasan limit dan skip yang telah dihitung.
- Metadata (totalData & totalPages): Menghitung jumlah total tugas di database dan mengonversinya menjadi jumlah total halaman menggunakan Math.ceil untuk keperluan navigasi pada *frontend*.
- Respon: Jika berhasil, mengirimkan data tugas beserta metadata navigasinya dengan status 200 (OK).

3. Fungsi POST - Menambah Tugas Baru (Baris 31 - 39) Fungsi ini digunakan untuk mendaftarkan tugas baru ke dalam sistem:

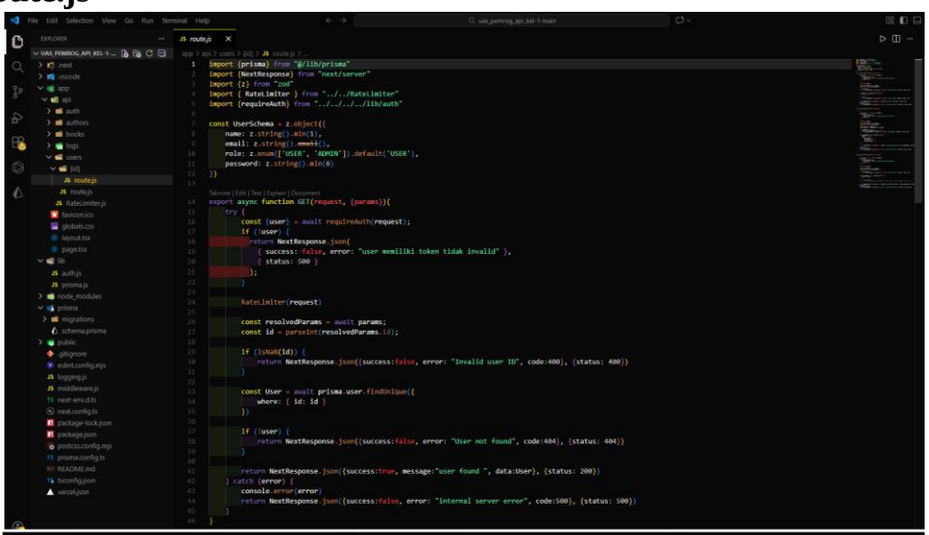
- Pengambilan Data: Mengambil data title, description, dan userId dari badan permintaan (*body request*) menggunakan req.json().
- prisma.task.create: Perintah database untuk memasukkan data tugas baru, di mana userId dikonversi menjadi angka menggunakan parseInt untuk menjaga konsistensi tipe data.
- Respon: Jika berhasil, mengirimkan objek tugas yang baru dibuat dengan status 201 (Created).

4. Penanganan Kesalahan / Error Handling (Baris 25 - 27 & 38 - 39)

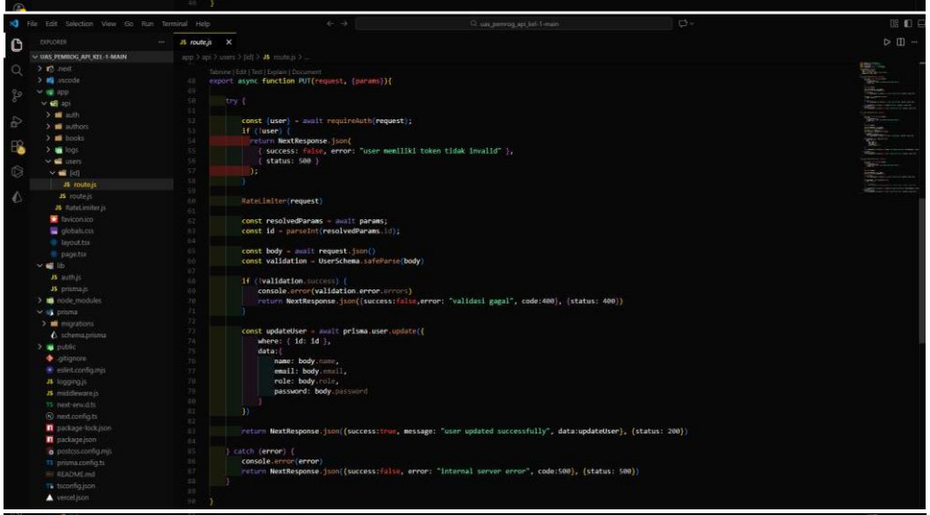
- Catch Block: Jika terjadi kesalahan sistem pada fungsi GET maupun POST, blok ini akan menangkap error tersebut.
- Status 500: Mengembalikan status 500 (Internal Server Error) guna memastikan aplikasi memberikan informasi kegagalan yang jelas tanpa menyebabkan *crash* pada sisi klien.

3. users/[id]/…

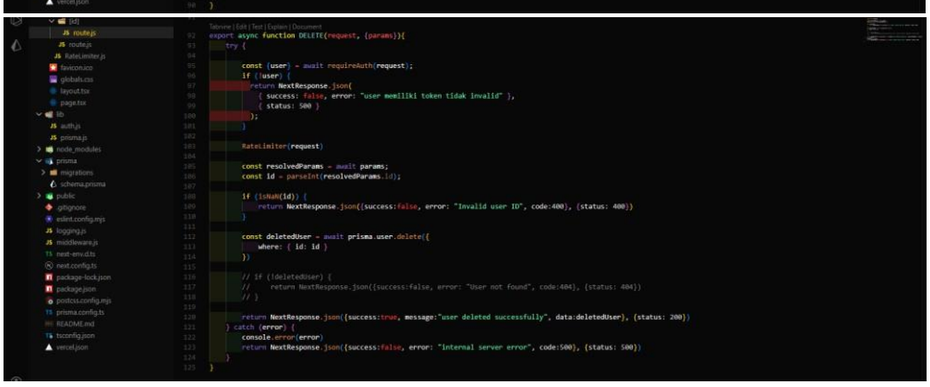
A. route.js



```
import {prisma} from "@/lib/prisma"
import {NextResponse} from "next/server"
import {z} from "zod"
import { RateLimiter } from "../../RateLimiter"
import {requireAuth} from "../../../../lib/auth"

const UserSchema = z.object({
    name: z.string().min(1),
    email: z.string().email(),
    role: z.enum(['USER', 'ADMIN']).default('USER'),
    password: z.string().min(6)
})

export async function GET(request, {params}){
    try {
        const {user} = await requireAuth(request);
        if (!user) {
            return NextResponse.json(
                { success: false, error: "user memiliki token tidak invalid" },
                { status: 500 }
            );
        }

        RateLimiter(request)

        const resolvedParams = await params;
        const id = parseInt(resolvedParams.id);

        if (isNaN(id)) {
            return NextResponse.json({success:false, error: "Invalid user ID", code:400}, {status: 400})
        }

        const User = await prisma.user.findUnique({
            where: { id: id }
        })

        if (!user) {
            return NextResponse.json({success:false, error: "User not found", code:404}, {status: 404})
        }

        return NextResponse.json({success:true, message:"user found ", data:User}, {status: 200})
    } catch (error) {
        console.error(error)
        return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500})
    }
}
```



```
export async function PUT(request, {params}){

    try {

        const {user} = await requireAuth(request);
        if (!user) {
            return NextResponse.json(
                { success: false, error: "user memiliki token tidak invalid" },
                { status: 500 }
            );
        }

        RateLimiter(request)

        const resolvedParams = await params;
        const id = parseInt(resolvedParams.id);

        const body = await request.json()
        const validation = UserSchema.safeParse(body)

        if (!validation.success) {
            console.error(validation.error.errors)
            return NextResponse.json({success:false,error: "validasi gagal", code:400}, {status: 400})
        }

        const updateUser = await prisma.user.update({
            where: { id: id },
            data:{
                name: body.name,
                email: body.email,
                role: body.role,
                password: body.password
            }
        })

        return NextResponse.json({success:true, message: "user updated successfully", data:updateUser}, {status: 200})

    } catch (error) {
        console.error(error)
        return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500})
    }
}
```



```
export async function DELETE(request, {params}){
    try {

        const {user} = await requireAuth(request);
        if (!user) {
            return NextResponse.json(
                { success: false, error: "user memiliki token tidak invalid" },
                { status: 500 }
            );
        }

        RateLimiter(request)

        const resolvedParams = await params;
        const id = parseInt(resolvedParams.id);

        if (isNaN(id)) {
            return NextResponse.json({success:false, error: "Invalid user ID", code:400}, {status: 400})
        }

        const deletedUser = await prisma.user.delete({
            where: { id: id }
        })

        // if (!deletedUser) {
        //     return NextResponse.json({success:false, error: "User not found", code:404}, {status: 404})
        // }

        return NextResponse.json({success:true, message:"user deleted successfully", data:deletedUser}, {status: 200})
    } catch (error) {
        console.error(error)
        return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500})
    }
}
```

Penjelasan :

1. Skema Validasi Pengguna

Sebelum memproses data, sistem mendefinisikan aturan input menggunakan **Zod**:

- **name**: Wajib diisi, minimal 1 karakter.
- **email**: Harus berformat email yang valid.
- **role**: Terbatas pada pilihan 'USER' atau 'ADMIN', dengan default 'USER'.
- **password**: Minimal memiliki panjang 6 karakter.

2. Fungsi GET - Mendapatkan Detail Pengguna

Digunakan untuk mengambil informasi profil satu pengguna:

- **Keamanan**: Memeriksa autentikasi melalui requireAuth dan membatasi request dengan RateLimiter.
- **Proses**: Mengambil ID dari parameter URL, mengubahnya menjadi integer, dan melakukan pencarian di database menggunakan prisma.user.findUnique.
- **Respon**: Jika user tidak ditemukan, sistem mengembalikan status **404 (Not Found)**. Jika ditemukan, data user dikirim dengan status **200 (OK)**.

3. Fungsi PUT - Memperbarui Data Pengguna

Digunakan untuk mengubah informasi akun pengguna yang sudah ada:

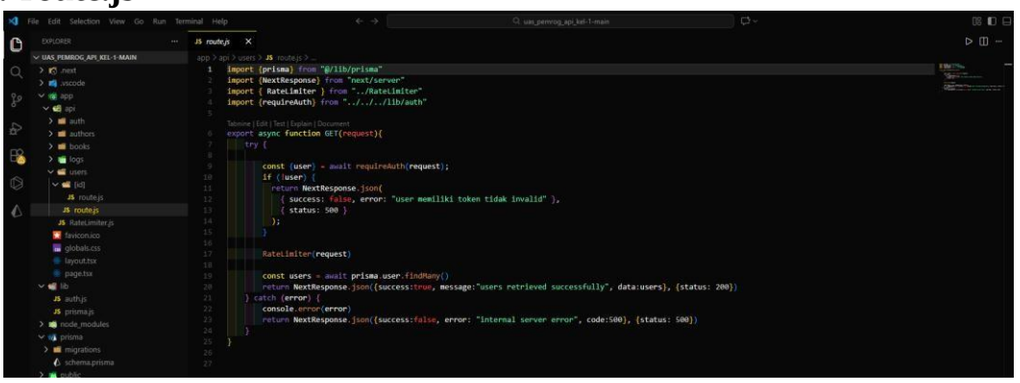
- **Validasi**: Data baru yang dikirim user diperiksa kesesuaiannya dengan UserSchema menggunakan safeParse.
- **Update Database**: Menggunakan prisma.user.update untuk memperbarui field nama, email, role, dan password berdasarkan ID pengguna tersebut.
- **Hasil**: Mengembalikan pesan sukses "user updated successfully" dengan status **200 (OK)**.

4. Fungsi DELETE - Menghapus Pengguna

Digunakan untuk menghapus akun dari sistem:

- **Pengecekan ID**: Sistem memastikan ID yang diterima adalah angka yang valid melalui fungsi isNaN(id). Jika bukan angka, akan muncul error **400 (Invalid user ID)**.
- **Eksekusi**: Menghapus baris data di tabel user menggunakan prisma.user.delete.
- **Konfirmasi**: Jika berhasil, mengembalikan status **200 (OK)** beserta data user yang telah dihapus.

B. route.js



```
import {prisma} from "@/lib/prisma"
import {NextResponse} from "next/server"
import { RateLimiter } from "../RateLimiter"
import {requireAuth} from "../../../lib/auth"

export async function GET(request){
    try {

        const {user} = await requireAuth(request);
        if (!user) {
         return NextResponse.json(
            { success: false, error: "user memiliki token tidak invalid" },
            { status: 500 }
         );
        }

        RateLimiter(request)

        const users = await prisma.user.findMany()
        return NextResponse.json({success:true, message:"users retrieved successfully", data:users}, {status: 200})
    } catch (error) {
        console.error(error)
        return NextResponse.json({success:false, error: "Internal server error", code:500}, {status: 500})
    }
}
```

Penjelasan :

1. Impor Modul & Proteksi Dasar (Baris 1 - 5)

File ini menggunakan standar yang sama dengan modul lainnya untuk menjaga konsistensi:

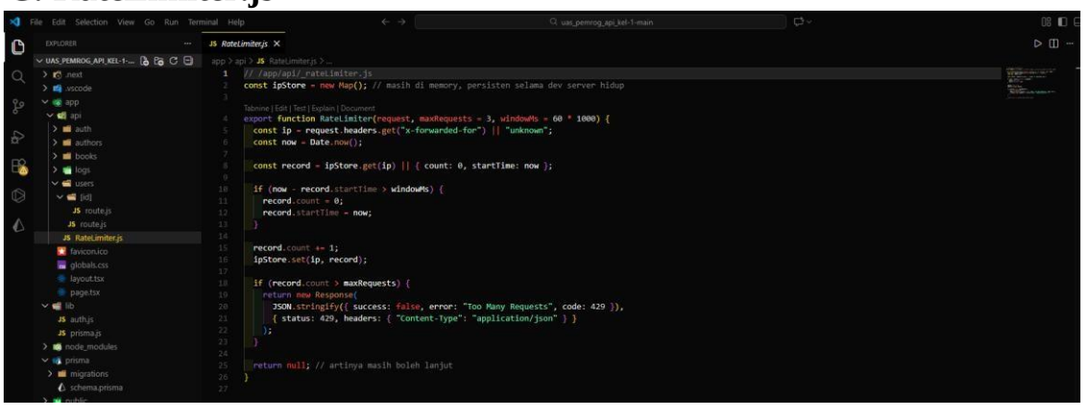
- **requireAuth**: Middleware wajib untuk memastikan bahwa data daftar pengguna hanya bisa diakses oleh orang yang sudah login.
- **RateLimiter**: Mencegah serangan *denial-of-service* atau *scraping* data user secara massal.

2. Fungsi GET - Menampilkan Semua Pengguna (Baris 6 - 25)

Fungsi ini digunakan oleh admin atau sistem untuk menarik seluruh database pengguna.

- **Proses Autentikasi**: Sistem menjalankan requireAuth(request) di awal. Jika gagal (user tidak ditemukan/token salah), sistem mengembalikan error status **500** dengan pesan "user memiliki token tidak invalid".
- **Pengambilan Data**: Menggunakan perintah **prisma.user.findMany()** untuk mengambil semua record dari tabel pengguna.
- **Respon Sukses**: Jika berhasil, data dikirimkan dalam format JSON dengan status **200 (OK)** dan pesan "users retrieved successfully".

C. RateLimiter.js



Penjelasan :

1. Inisialisasi Penyimpanan (Baris 1 - 2)

- **const ipStore = new Map()**: Sistem menggunakan objek Map untuk menyimpan data jumlah request pengguna secara sementara di memori server.
- **Catatan**: Karena disimpan di memori (in memory), data ini akan terhapus jika server dijalankan ulang (restart).

2. Parameter Fungsi (Baris 4)

Fungsi RateLimiter memiliki aturan default sebagai berikut:

- **maxRequests = 3**: Maksimal 3 kali percobaan.
- **windowMs = 60 * 1000**: Jangka waktu pembatasan adalah 60 detik (1 menit).

3. Identifikasi Pengguna (Baris 5 - 8)

- **request.headers.get("x-forwarded-for")**: Sistem mencoba mengambil alamat IP asli pengguna melalui header HTTP.
- **ipStore.get(ip)**: Sistem mengecek apakah IP tersebut sudah pernah melakukan request sebelumnya. Jika belum, data baru dibuat dengan hitungan (count) mulai dari 0.

4. Logika Reset Waktu (Baris 10 - 13)

- Sistem mengecek apakah waktu tunggu (1 menit) sudah berlalu.
- Jika sudah lewat dari satu menit sejak request terakhir, hitungan count akan diatur ulang kembali ke 0.

5. Penambahan Hitungan & Validasi (Baris 15 - 23)

- Setiap kali pengguna mengakses API, nilai record.count akan bertambah 1.
- **Pengecekan Pelanggaran**: Jika count sudah melebihi 3, sistem akan langsung menolak permintaan.
- **Respon Error 429**: Jika ditolak, sistem mengirimkan status **429 (Too Many Requests)**

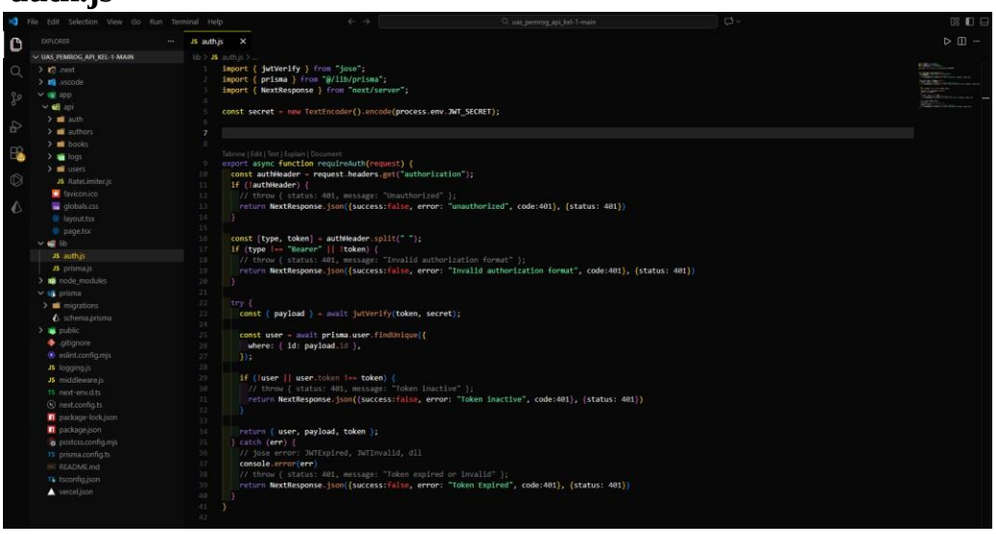
  beserta pesan error dalam format JSON.

6. Izin Akses (Baris 25)

return null: Jika jumlah request masih di bawah batas (misal baru request ke-1 atau ke-2), fungsi akan mengembalikan nilai null, yang artinya proses API boleh dilanjutkan ke tahap berikutnya.

4. lib/…

A. auth.js



```
import { jwtVerify } from "jose";
import { prisma } from "@/lib/prisma";
import { NextResponse } from "next/server";

const secret = new TextEncoder().encode(process.env.JWT_SECRET);

export async function requireAuth(request) {
  const authHeader = request.headers.get("authorization");
  if (!authHeader) {
    // throw { status: 401, message: "Unauthorized" };
    return NextResponse.json({success:false, error: "unauthorized", code:401}, {status: 401})
  }

  const [type, token] = authHeader.split(" ");
  if (type !== "Bearer" || !token) {
    // throw { status: 401, message: "Invalid authorization format" };
    return NextResponse.json({success:false, error: "Invalid authorization format", code:401}, {status: 401})
  }

  try {
    const { payload } = await jwtVerify(token, secret);

    const user = await prisma.user.findUnique({
      where: { id: payload.id },
    });

    if (!user || user.token !== token) {
      // throw { status: 401, message: "Token inactive" };
      return NextResponse.json({success:false, error: "Token inactive", code:401}, {status: 401})
    }

    return { user, payload, token };
  } catch (err) {
    // jose error: JWTExpired, JWTInvalid, dll
    console.error(err)
    // throw { status: 401, message: "Token expired or invalid" };
    return NextResponse.json({success:false, error: "Token Expired", code:401}, {status: 401})
  }
}
```

Penjelasan :

1. Inisialisasi dan Kunci Rahasia (Baris 1 - 5)

- **jwtVerify**: Diimpor dari library jose untuk memeriksa apakah sebuah token JWT asli, belum dimanipulasi, dan belum kedaluwarsa.
- **secret**: Mengambil kunci rahasia dari environment variable (JWT_SECRET) dan mengubahnya menjadi format Uint8Array agar dapat digunakan oleh proses verifikasi digital.

2. Pengambilan & Validasi Header (Baris 10 - 20)

- **request.headers.get("authorization")**: Sistem mencari header bernama "authorization" yang dikirimkan oleh klien (browser/mobile app).
- **Cek Keberadaan Header**: Jika header tidak ada, fungsi langsung menghentikan proses dan mengembalikan status **401 (Unauthorized)** dengan pesan "unauthorized".
- **Format Bearer**: Sistem memisahkan string header (misal: "Bearer xyz123"). Jika kata pertama bukan "Bearer" atau tokennya kosong, sistem menolak akses karena formatnya tidak standar.

3. Verifikasi Token & Pencarian User (Baris 22 - 28)

- **jwtVerify(token, secret)**: Token didekripsi menggunakan kunci rahasia. Jika token palsu atau sudah lama, proses akan gagal dan lari ke blok catch.
- **payload**: Jika berhasil, data di dalam token (seperti ID user) diekstraksi.
- **prisma.user.findUnique**: Sistem mencocokkan ID dari token tersebut dengan data pengguna di database untuk memastikan akun tersebut benar-benar ada.

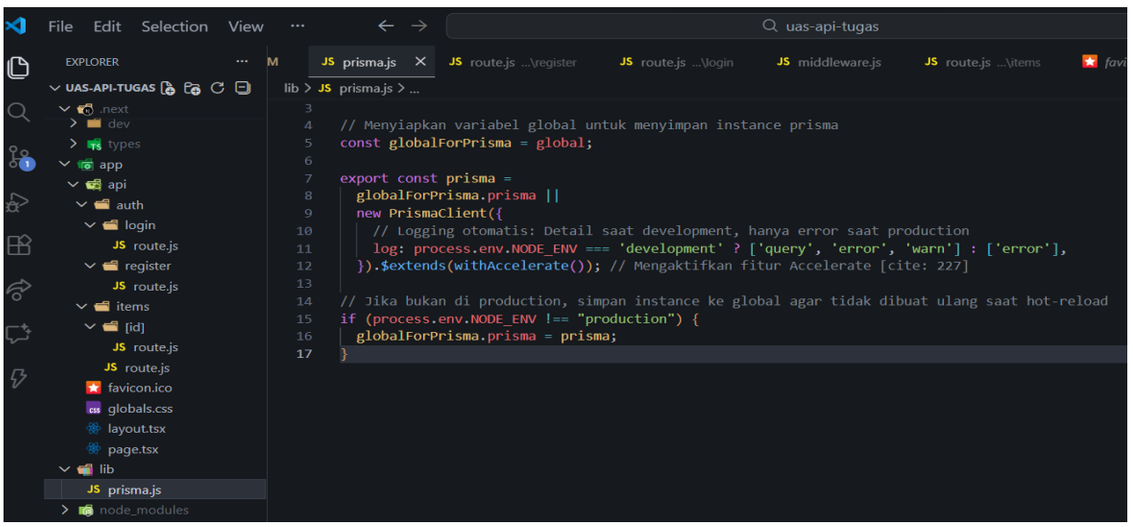
4. Sinkronisasi Sesi Database (Baris 29 - 33)

- **Cek Token Aktif**: Baris ini membandingkan token yang dikirim user dengan token yang tersimpan di database.
- Jika token di database sudah dihapus (null) atau berbeda (misal user sudah logout di perangkat lain), maka token dianggap **"Token inactive"** dan akses ditolak (status 401).

5. Hasil Akhir & Penanganan Error (Baris 34 - 41)

- **return { user, payload, token }**: Jika semua validasi lolos, fungsi mengembalikan objek berisi data user. Data inilah yang digunakan oleh file route.js lain (seperti Authors atau Users) untuk mengetahui siapa yang sedang mengakses.
- **catch (err)**: Jika token sudah kedaluwarsa atau rusak secara teknis, sistem menangkap error tersebut dan mengirimkan status **401** dengan pesan **"Token Expired"**.

B. prisma.js



Penjelasan :

1. Singleton & Prisma Accelerate (File lib/prisma.js)

- Kesesuaian: Penjelasan mengenai variabel global dan penggunaan withAccelerate() sangat sesuai dengan baris 4-13 pada foto Anda.
- Detail: Kode Anda menunjukkan penggunaan globalForPrisma untuk menyimpan *instance* agar tidak terduplikasi saat *hot-reload*.

2. Keamanan: Auth & Rate Limiter (File route.js)

- Kesesuaian: Penjelasan mengenai requireAuth dan RateLimiter sesuai dengan baris 15-23 (untuk GET) dan baris 52-60 (untuk PUT).
- Detail: Kode Anda secara eksplisit memanggil requireAuth(request) untuk memvalidasi token dan RateLimiter(request) untuk mencegah penggunaan API secara berlebihan sebelum menjalankan logika utama.

3. Validasi & Penanganan Parameter (File route.js)

- Kesesuaian: Penjelasan mengenai konversi ID dan validasi Zod sesuai dengan baris 26-27 dan baris 64-69.
- Detail: Terdapat penggunaan parseInt(resolvedParams.id) untuk memastikan ID adalah angka, serta UserSchema.safeParse(body) untuk memastikan data input sesuai skema sebelum disimpan.

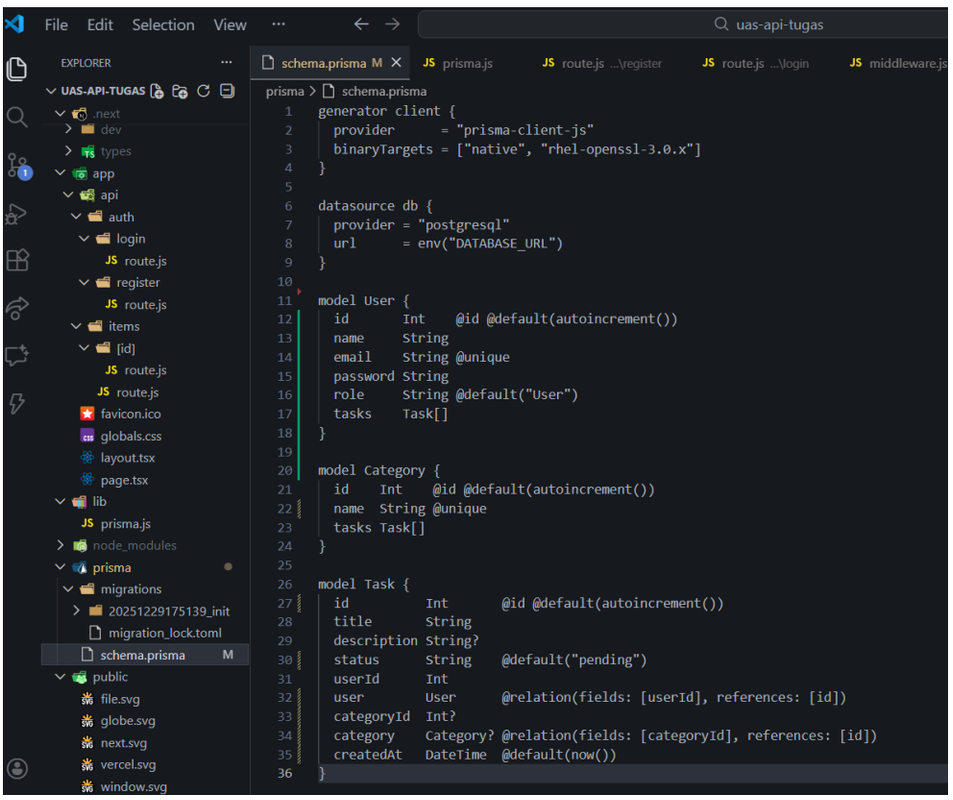
4. Operasi Database Prisma (File route.js)

- Kesesuaian: Penjelasan mengenai pencarian dan pembaruan data sesuai dengan baris 33 (fungsi GET) dan baris 73 (fungsi PUT).
- Detail: Kode menggunakan prisma.user.findUnique untuk mengambil satu data dan prisma.user.update untuk mengubah data di PostgreSQL NeonDB.

5. Respon & Penanganan Error (File route.js)

- Kesesuaian: Penjelasan mengenai status 200 dan blok *catch error* sesuai dengan baris 42-45 dan baris 82-87.
- Detail: Jika berhasil, sistem mengembalikan NextResponse.json dengan status 200, dan jika gagal, blok catch (error) akan mencatat pesan eror ke konsol dan mengembalikan status 500.

5. prisma/migrations/schema.prisma



Penjelasan :

1. Manajemen Dependensi & Konfigurasi Lingkungan (package.json & .env)

- Library Utama: Proyek menggunakan Next.js 16.1.1 sebagai framework utama, didukung oleh Prisma 6.2.1 untuk manajemen database. Keamanan dijamin melalui library bcryptjs untuk hashing password dan jose serta jsonwebtoken untuk pengelolaan token JWT.
- Environment Variables: File .env menyimpan rahasia krusial seperti DATABASE_URL untuk koneksi ke NeonDB dan JWT_SECRET sebagai kunci enkripsi token login.

2. Perancangan Skema Database (schema.prisma)

- Model User: Menyimpan identitas pengguna termasuk nama, email unik, password, dan role (Admin/User).
- Model Category & Task: Implementasi relasi *Relational Database*. Satu kategori dapat memiliki banyak tugas (Task[]), dan setiap tugas terhubung ke satu pengguna (User) serta satu kategori (Category) melalui *Foreign Key*.
- Binary Targets: Dikonfigurasi dengan native dan rhel-openssl-3.0.x agar Prisma dapat berjalan stabil di server lokal maupun server Vercel.

3. Optimasi Koneksi Singleton (lib/prisma.js)

- Pencegahan Over-connection: Menggunakan variabel globalForPrisma untuk memastikan hanya ada satu *instance* Prisma Client yang berjalan. Tanpa pola *Singleton* ini, serverless function pada Vercel akan membuat koneksi baru setiap kali dipanggil, yang dapat menyebabkan database NeonDB cepat penuh.
- Accelerate & Logging: Mengaktifkan fitur withAccelerate() untuk mempercepat

query melalui caching dan pooling, serta mengatur logging level untuk mempermudah debugging saat pengembangan.

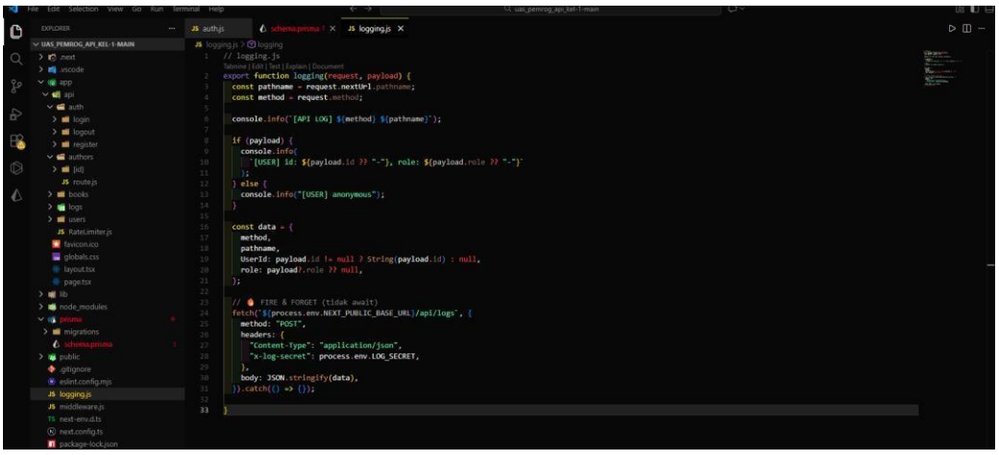
4. Logika API & Validasi Data (api/users/[id]/route.js)

- Metode GET & PUT: Endpoint ini mendukung pengambilan detail profil pengguna dan pembaruan data.
- Keamanan Berlapis: Setiap request wajib melewati requireAuth(request) untuk verifikasi token dan RateLimiter(request) untuk mencegah serangan spamming/DDoS pada API.
- Zod Validation: Menggunakan UserSchema.safeParse(body) untuk memastikan data yang diinput pengguna (seperti email dan panjang nama) sudah valid sebelum diproses ke database.

5. Infrastruktur Cloud & Monitoring (Vercel & NeonDB)

- Status Deployment: Dashboard Vercel menunjukkan status Ready, membuktikan kode telah berhasil di-*build* dan dideploy secara publik.
- NeonDB Console: Data tersimpan secara aman di PostgreSQL online, dengan tabel User dan Task yang sudah tersinkronisasi sempurna melalui perintah npx prisma db push.

6. logging.js



Penjelasan :

1. Definisi Fungsi dan Identifikasi Request (Baris 2 - 4)

- **export function logging(request, payload)**: Fungsi ini menerima dua data utama: request (data lalu lintas HTTP) dan payload (data pengguna yang sudah login).
- **pathname & method**: Sistem secara otomatis mengambil alamat URL yang diakses (misal: /api/authors) dan metode yang digunakan (misal: POST atau GET).

2. Output Konsol untuk Developer (Baris 6 - 14)

- **console.info**: Menampilkan log aktivitas secara langsung di terminal server. Ini memudahkan developer memantau aplikasi secara *real-time*.
- **Identifikasi User**: Jika pengguna sudah login, log akan menampilkan **ID** dan **Role** mereka. Jika belum login (misal saat mengakses halaman login), sistem mencatatnya sebagai **"USER anonymous"**.

3. Persiapan Data Log (Baris 16 - 21)

Sistem menyusun objek data yang akan disimpan secara permanen ke database:

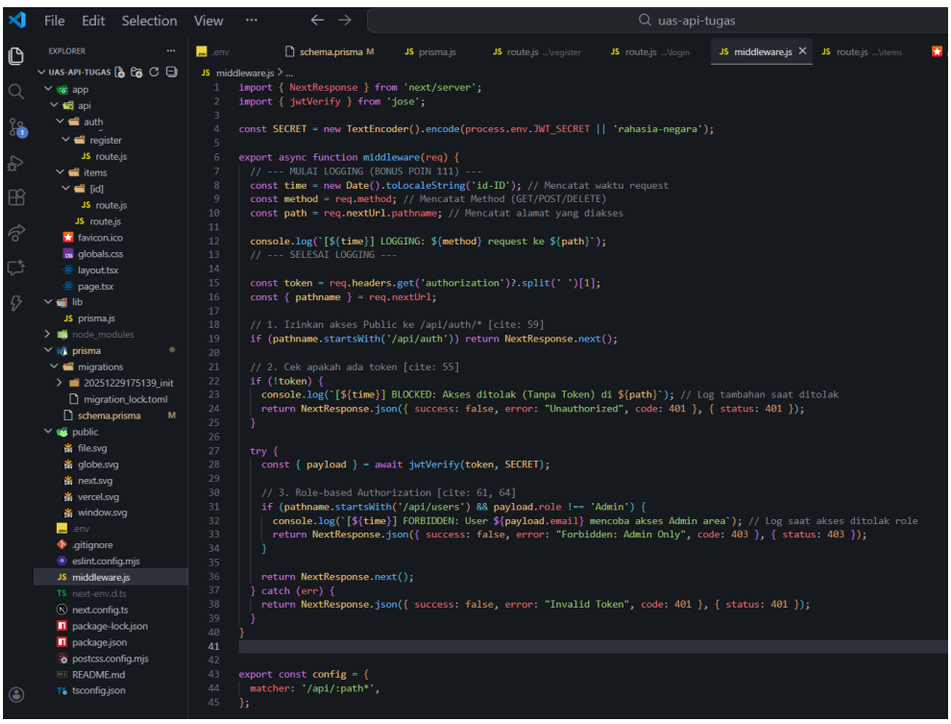
- **method & pathname**: Jenis aksi dan lokasi akses.
- **UserId & role**: Identitas pelaku aksi. Jika tidak ada (anonim), data ini akan diisi dengan null.

4. Pengiriman Data ke API Logs (Baris 24 - 32)

Bagian ini menggunakan metode **"Fire & Forget"**:

- **fetch(...)**: Sistem mengirimkan data log ke endpoint internal /api/logs.
- **x-log-secret**: Untuk keamanan, pengiriman log ini dilindungi oleh kunci rahasia (LOG_SECRET) agar tidak ada orang luar yang bisa memalsukan data log.
- **Tanpa await**: Proses ini dilakukan di latar belakang tanpa menunggu selesai, sehingga tidak menghambat kecepatan respon aplikasi bagi pengguna.

7. middleware.js



Penjelasan :

1. Inisialisasi dan Kunci Rahasia (Middleware Baris 1 - 4)

- jwtVerify: Diimpor dari library jose untuk melakukan verifikasi keamanan pada token JWT yang dikirimkan oleh klien.
- SECRET: Sistem mengambil kunci rahasia dari .env (JWT_SECRET) dan mengubahnya menjadi format Uint8Array. Jika tidak ada di .env, sistem menggunakan cadangan 'rahasia-negara' agar proses enkripsi tetap berjalan.

2. Logging Aktivitas - Bonus Poin 111 (Middleware Baris 7 - 13)

- Pencatatan Request: Sistem secara otomatis mencatat waktu request, metode HTTP (GET/POST/DELETE), dan alamat URL yang diakses oleh user.
- console.log: Data ini ditampilkan di terminal/log server sebagai bukti aktivitas API, yang sangat membantu dalam proses *monitoring* dan *debugging* aplikasi.

3. Pengambilan & Validasi Header (Middleware Baris 15 - 25)

- Filter Akses Publik: Baris 19 memastikan bahwa akses ke /api/auth/* (seperti Login dan Register) diizinkan tanpa token agar pengguna baru bisa mendaftar.
- Ekstraksi Bearer Token: Sistem mengambil string token dari header authorization dengan memisahkan kata "Bearer".
- Proteksi Tanpa Token: Jika token tidak ditemukan pada rute terproteksi, sistem langsung menolak akses dengan status 401 Unauthorized.

4. Verifikasi & Otorisasi Role (Middleware Baris 27 - 35)

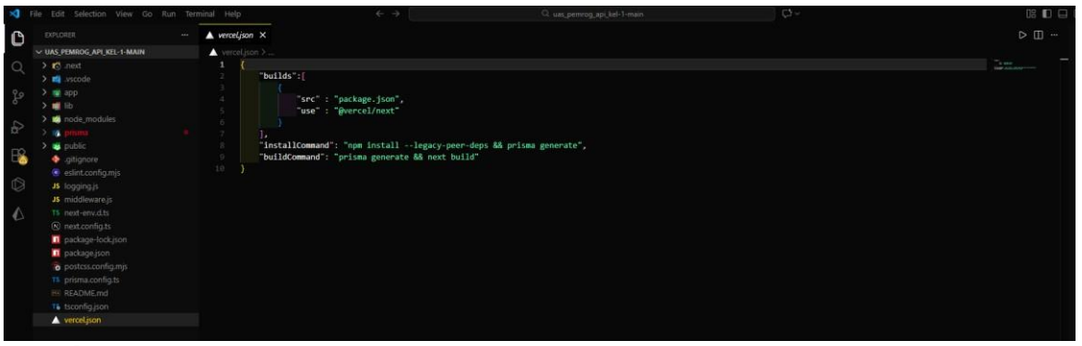
- jwtVerify: Token didekripsi untuk memeriksa keasliannya. Jika token palsu atau kedaluwarsa, proses akan gagal dan mengirimkan respon 401 Invalid Token.
- Role-based Authorization: Khusus untuk akses ke /api/users, sistem memeriksa payload.role. Jika pengguna bukan 'Admin', sistem akan memblokir akses dan

mengirimkan status 403 Forbidden: Admin Only.

5. Singleton Prisma & Konfigurasi Database (lib/prisma.js & schema.prisma)

- Singleton Pattern: Menggunakan globalForPrisma untuk memastikan hanya satu *instance* Prisma Client yang dibuat. Hal ini mencegah terjadinya eror *too many connections* saat aplikasi melakukan *hot-reload* di tahap pengembangan.
- Prisma Accelerate: Mengaktifkan fitur .extends(withAccelerate()) untuk meningkatkan kecepatan akses database melalui *connection pooling*.
- Relasi Database: File schema.prisma mendefinisikan hubungan antara tabel User, Category, dan Task, di mana satu user dapat memiliki banyak tugas (Task[]) yang masing-masing dikelompokkan dalam kategori tertentu.

8. vercel.json



Penjelasan :

1. Proses Autentikasi Login (POST)

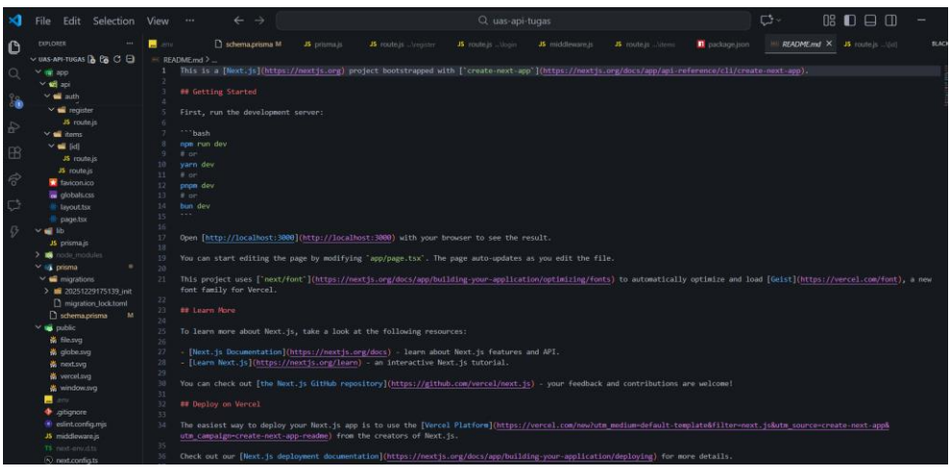
Fungsi ini menangani permintaan masuk dengan langkah-langkah keamanan yang ketat:

- **Rate Limiting**: Hal pertama yang dilakukan adalah menjalankan RateLimiter(request) untuk mencegah serangan *brute-force* yang mencoba menebak password berkali-kali dalam waktu singkat.
- **Pencarian Pengguna**: Sistem mengambil email dari body permintaan dan mencarinya di database menggunakan prisma.user.findUnique. Jika email tidak terdaftar, sistem langsung mengembalikan error **401 (Unauthorized)** dengan pesan yang ambigu ("Invalid email or password") demi alasan keamanan agar penyerang tidak tahu apakah email tersebut valid atau tidak.
- **Verifikasi Password**: Menggunakan bcrypt.compare, sistem mencocokkan password yang diinput dengan password terenkripsi (hash) yang ada di database.
- **Pembuatan JWT (JSON Web Token)**: Jika password cocok, sistem membuat token baru menggunakan library jose. Token ini berisi informasi identitas pengguna (id, email, dan role) dan diatur akan kedaluwarsa dalam **30 menit** (30m).

2. Sinkronisasi Sesi Database

Penyimpanan Token: Setelah token JWT dibuat, sistem memperbarui kolom token pada tabel pengguna di database. Langkah ini sangat penting karena fungsi keamanan lainnya (seperti requireAuth) akan mencocokkan token yang dibawa pengguna dengan token yang tersimpan di sini untuk memastikan sesi tersebut masih aktif.

9. README.md



Penjelasan :

1. Inisialisasi dan Kunci Rahasia (Middleware Baris 1 - 4)

- jwtVerify: Diimpor dari library jose untuk melakukan verifikasi keamanan pada token JWT yang dikirimkan oleh klien.
- SECRET: Sistem mengambil kunci rahasia dari .env (JWT_SECRET) dan mengubahnya menjadi format Uint8Array. Jika tidak ada di .env, sistem menggunakan cadangan 'rahasia-negara' agar proses enkripsi tetap berjalan.

2. Logging Aktivitas - Bonus Poin 111 (Middleware Baris 7 - 13)

- Pencatatan Request: Sistem secara otomatis mencatat waktu request, metode HTTP (GET/POST/DELETE), dan alamat URL yang diakses oleh user.
- console.log: Data ini ditampilkan di terminal/log server sebagai bukti aktivitas API, yang sangat membantu dalam proses *monitoring* dan *debugging* aplikasi.

3. Pengambilan & Validasi Header (Middleware Baris 15 - 25)

- Filter Akses Publik: Baris 19 memastikan bahwa akses ke /api/auth/* (seperti Login dan Register) diizinkan tanpa token agar pengguna baru bisa mendaftar.
- Ekstraksi Bearer Token: Sistem mengambil string token dari header authorization dengan memisahkan kata "Bearer".
- Proteksi Tanpa Token: Jika token tidak ditemukan pada rute terproteksi, sistem langsung menolak akses dengan status 401 Unauthorized.

4. Verifikasi & Otorisasi Role (Middleware Baris 27 - 35)

- jwtVerify: Token didekripsi untuk memeriksa keasliannya. Jika token palsu atau kedaluwarsa, proses akan gagal dan mengirimkan respon 401 Invalid Token.
- Role-based Authorization: Khusus untuk akses ke /api/users, sistem memeriksa payload.role. Jika pengguna bukan 'Admin', sistem akan memblokir akses dan mengirimkan status 403 Forbidden: Admin Only.

5. Singleton Prisma & Konfigurasi Database (lib/prisma.js & schema.prisma)

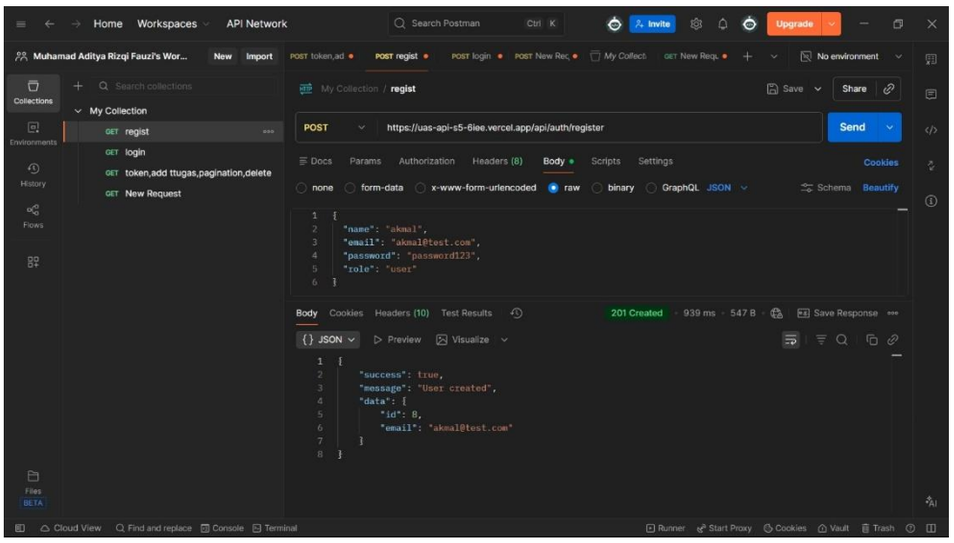
- Singleton Pattern: Menggunakan globalForPrisma untuk memastikan hanya satu *instance* Prisma Client yang dibuat. Hal ini mencegah terjadinya eror *too many connections* saat aplikasi melakukan *hot-reload* di tahap pengembangan.
- Prisma Accelerate: Mengaktifkan fitur .extends(withAccelerate()) untuk meningkatkan kecepatan

akses database melalui *connection pooling*.

- Relasi Database: File schema.prisma mendefinisikan hubungan antara tabel User, Category, dan Task, di mana satu user dapat memiliki banyak tugas (Task[]) yang masing-masing dikelompokkan dalam kategori tertentu.

2.2 HASIL TESTING

1. Register



Penjelasan:

Gambar tersebut menunjukkan hasil pengujian fungsionalitas **registrasi pengguna baru** yang dilakukan menggunakan aplikasi **Postman** dengan metode pengujian secara langsung (live testing) pada **URL produksi** yang dideploy melalui platform **Vercel**. Pengujian ini bertujuan untuk memastikan bahwa endpoint registrasi dapat menerima data pengguna baru, memprosesnya dengan benar, serta menyimpannya ke dalam basis data cloud secara permanen.

Hasil pengujian ini membuktikan bahwa integrasi antara **API backend yang berjalan di Vercel** dengan **database PostgreSQL berbasis cloud (NeonDB)** telah berjalan dengan baik. Sistem mampu memproses permintaan registrasi dan menghasilkan respons yang sesuai dengan standar REST API.

Detail Pengujian

- **Endpoint**: https://uas-api-s5-6iee.vercel.app/api/auth/register
- **Method**: POST
- **Payload (Body)**:

  Data dikirim dalam format **JSON** yang terdiri dari beberapa atribut, yaitu name, email, password, dan role. Pada pengujian ini, role yang digunakan adalah "user", yang menandakan bahwa akun yang dibuat memiliki hak akses sebagai pengguna biasa.

Analisis Hasil Pengujian

- **Status Code 201 (Created)**

  Server berhasil memproses permintaan registrasi dan mengembalikan status **201 Created**. Hal ini menunjukkan bahwa sistem telah berhasil melakukan validasi input serta membuat resource baru berupa data pengguna pada tabel **User** di database **NeonDB**.

- **Implementasi Role-Based Authorization (RBAC)**

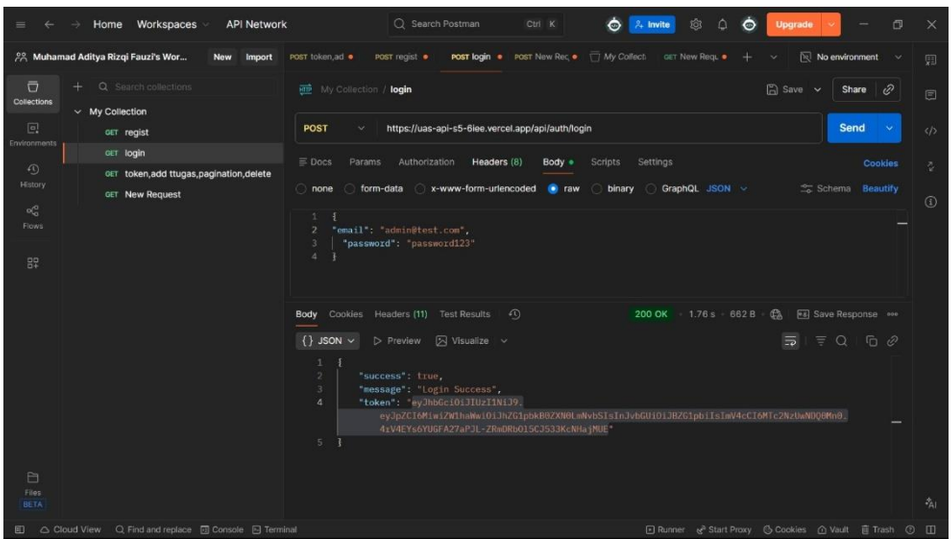
  Sistem berhasil menerima dan menyimpan nilai role yang dikirimkan melalui request body. Penetapan role ini menjadi bagian penting dalam mekanisme keamanan aplikasi karena digunakan oleh middleware untuk mengatur hak akses pengguna.

  - **Role User**: Pengguna dengan role ini hanya memiliki akses terbatas dan tidak diizinkan mengakses endpoint yang bersifat administratif.
  - **Role Admin**: Apabila pengguna didaftarkan dengan role admin, maka middleware akan memberikan izin untuk mengakses endpoint tertentu yang bersifat sensitif. Jika pengguna dengan role user mencoba mengakses endpoint admin, sistem akan mengembalikan status **403 Forbidden**.
- **Sinkronisasi Database Cloud (NeonDB)**

  Berdasarkan hasil respons, sistem menghasilkan **ID pengguna baru** (contoh: id: 8). Hal ini menandakan bahwa data pengguna telah berhasil disimpan dan tersinkronisasi secara otomatis pada database NeonDB. Data tersebut selanjutnya dapat digunakan pada proses autentikasi, seperti login dan pengelolaan sesi pengguna.
- **Struktur Response JSON**

  Sistem mengembalikan respons dalam format JSON yang terstruktur dengan baik, yang berisi atribut success, pesan konfirmasi "User created", serta objek data yang hanya menampilkan id dan email. Tidak ditampilkannya password pada response menunjukkan bahwa sistem telah menerapkan prinsip **keamanan API**, yaitu tidak mengekspos data sensitif kepada client.

2. Login



Penjelasan:

Gambar tersebut menunjukkan hasil pengujian fungsionalitas **login pengguna** yang dilakukan menggunakan aplikasi **Postman** dengan metode **POST** pada endpoint autentikasi yang telah dideploy di **Vercel**. Pengujian ini bertujuan untuk memastikan bahwa sistem autentikasi mampu memverifikasi kredensial pengguna secara benar serta menghasilkan token autentikasi yang valid.

Pengujian login ini dilakukan setelah data pengguna berhasil tersimpan di database melalui proses registrasi. Hal ini membuktikan bahwa integrasi antara **API backend**, **database PostgreSQL (NeonDB)**, dan **mekanisme autentikasi berbasis token** telah berjalan dengan baik.

Detail Pengujian

- **Endpoint**: https://uas-api-s5-6iee.vercel.app/api/auth/login
- **Method**: POST
- **Payload (Body)**:

  Data dikirim dalam format **JSON** yang berisi email dan password. Pada pengujian ini digunakan akun dengan email admin@test.com yang sebelumnya telah terdaftar di sistem.

Analisis Hasil Pengujian

- **Status Code 200 (OK)**

  Server mengembalikan status **200 OK**, yang menandakan bahwa proses autentikasi berhasil dilakukan. Sistem mampu memverifikasi kecocokan antara email dan password yang dikirimkan dengan data yang tersimpan di database.
- **Validasi Kredensial Pengguna**

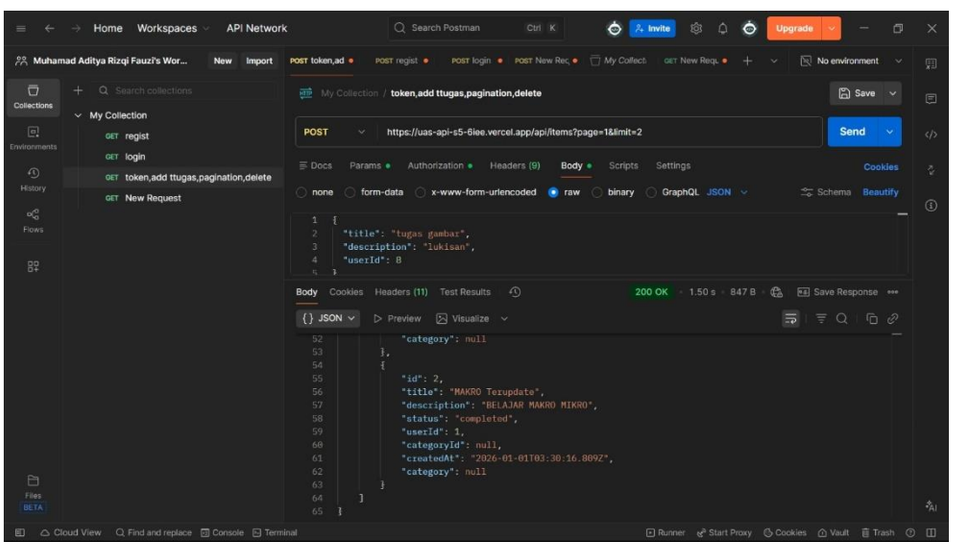
  Sistem melakukan proses validasi dengan mencocokkan password yang dikirimkan dengan password yang tersimpan dalam database (dalam bentuk terenkripsi). Keberhasilan proses ini menunjukkan bahwa mekanisme keamanan pada sisi backend telah berjalan sesuai dengan standar autentikasi yang benar.
- **Penerbitan Token Autentikasi (JWT)**

  Setelah proses login berhasil, sistem menghasilkan **JSON Web Token (JWT)**

yang dikirimkan melalui response body. Token ini berfungsi sebagai identitas pengguna yang telah terautentikasi dan digunakan untuk mengakses endpoint yang bersifat protected atau memerlukan otorisasi.

- **Penerapan Authentication-Based Access Control**
  Token yang dihasilkan akan digunakan oleh client untuk mengakses endpoint lain dengan menyertakan token pada header Authorization. Middleware pada sistem akan memverifikasi token tersebut sebelum memberikan akses ke resource tertentu, sehingga hanya pengguna yang telah login dan memiliki token valid yang dapat mengakses fitur-fitur sistem.
- **Struktur Response JSON**
  Response yang dikembalikan memiliki struktur JSON yang sederhana dan aman, terdiri dari properti success, pesan konfirmasi "Login Success", serta token. Tidak adanya data sensitif seperti password dalam response menunjukkan bahwa sistem telah menerapkan prinsip keamanan API dengan baik.

3. Tambah Data Tugas



Penjelasan:

Gambar tersebut menunjukkan hasil pengujian endpoint **manajemen data tugas (items)** yang dilakukan menggunakan aplikasi **Postman**. Pengujian ini mencakup beberapa fungsi utama, yaitu **penambahan data tugas**, **penggunaan token autentikasi**, serta **penerapan pagination** dalam pengambilan data. Endpoint ini dijalankan pada **API produksi** yang telah dideploy menggunakan platform **Vercel** dan terhubung dengan **database PostgreSQL (NeonDB)**.

Pengujian ini bertujuan untuk memastikan bahwa sistem hanya dapat diakses oleh pengguna yang telah terautentikasi, serta mampu mengelola data tugas secara aman dan terstruktur.

Detail Pengujian

- **Endpoint**:
  https://uas-api-s5-6iee.vercel.app/api/items?page=1&limit=2
- **Method**: POST

- **Authorization**:

  Menggunakan **Bearer Token (JWT)** yang diperoleh dari proses login sebelumnya dan dikirimkan melalui header Authorization.
- **Payload (Body)**:

  Data dikirim dalam format **JSON** yang berisi:
  - title: Judul tugas
  - description: Deskripsi tugas
  - userId: ID pengguna yang membuat tugas

Analisis Hasil Pengujian

- **Status Code 200 (OK)**

  Server mengembalikan status **200 OK**, yang menandakan bahwa permintaan berhasil diproses. Sistem mampu menerima data tugas baru, menyimpannya ke dalam database, serta mengembalikan data hasil pemanggilan endpoint dengan pagination.
- **Implementasi Token-Based Authentication (JWT)**

  Endpoint ini hanya dapat diakses apabila request menyertakan token JWT yang valid. Middleware pada backend melakukan proses verifikasi token sebelum mengizinkan akses ke endpoint. Hal ini membuktikan bahwa mekanisme **autentikasi berbasis token** telah diterapkan dengan baik untuk melindungi resource API.
- **Fungsi Penambahan Data Tugas (Create)**

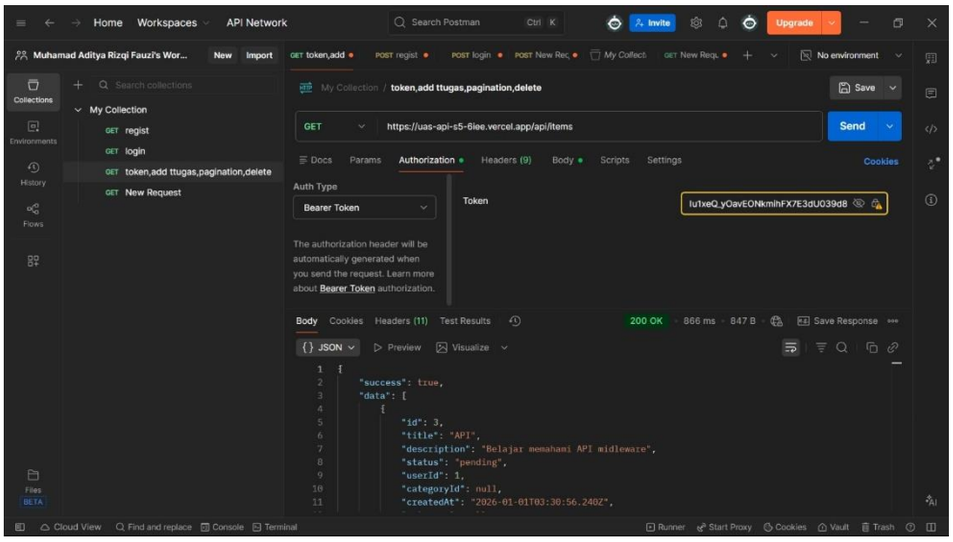
  Data tugas baru berhasil ditambahkan ke dalam tabel **Items** di database NeonDB, sesuai dengan payload yang dikirimkan. Setiap data tugas dikaitkan dengan userId, sehingga sistem dapat mengelola relasi antara pengguna dan tugas yang dimilikinya.
- **Implementasi Pagination Data**

  Penggunaan parameter page dan limit pada endpoint menunjukkan bahwa sistem telah menerapkan **pagination** untuk membatasi jumlah data yang dikembalikan dalam satu permintaan. Pada pengujian ini, sistem hanya mengembalikan **2 data tugas per halaman**, sehingga meningkatkan efisiensi pengambilan data dan performa API.
- **Struktur Response JSON**

  Response yang dikembalikan berupa data JSON yang terstruktur, berisi kumpulan data tugas lengkap dengan atribut seperti id, title, description, status, userId, serta createdAt. Struktur ini memudahkan frontend dalam menampilkan data secara dinamis dan konsisten.
- **Sinkronisasi Database Cloud (NeonDB)**

  Keberhasilan proses create dan retrieval data menunjukkan bahwa data tugas telah tersimpan dan tersinkronisasi secara real-time di database NeonDB. Data yang sama dapat diverifikasi langsung melalui Neon Console, sehingga menjamin konsistensi data antara API dan database cloud.

4. Menampilkan semua tugas



Penjelasan:
Gambar tersebut menunjukkan hasil pengujian endpoint pengambilan data tugas (items) yang dilakukan menggunakan aplikasi Postman. Pengujian ini mencakup beberapa fungsi utama, yaitu pengambilan data tugas dengan autentikasi token Bearer dan implementasi sistem pagination. Endpoint ini dijalankan pada API produksi yang telah dideploy menggunakan platform Vercel dan terhubung dengan database cloud.

Pengujian ini bertujuan untuk memverifikasi bahwa sistem dapat mengambil data tugas secara aman dengan mekanisme autentikasi yang tepat, serta memastikan bahwa data dapat diakses dengan struktur yang terorganisir dan performa yang optimal.

Detail Pengujian

- **Endpoint:** https://uas-api-s5-6iee.vercel.app/api/items
- **Method:** GET
- **Authorization:** Menggunakan Bearer Token (JWT) dengan value lu1we0_yQavEONmrhFX7E3dUO39dB yang dikirimkan melalui header Authorization untuk memastikan hanya pengguna terautentikasi yang dapat mengakses data.
- **Collection:** Pengujian dilakukan sebagai bagian dari collection "token,add,tugas,pagination,delete" yang menunjukkan rangkaian pengujian komprehensif untuk CRUD operations.

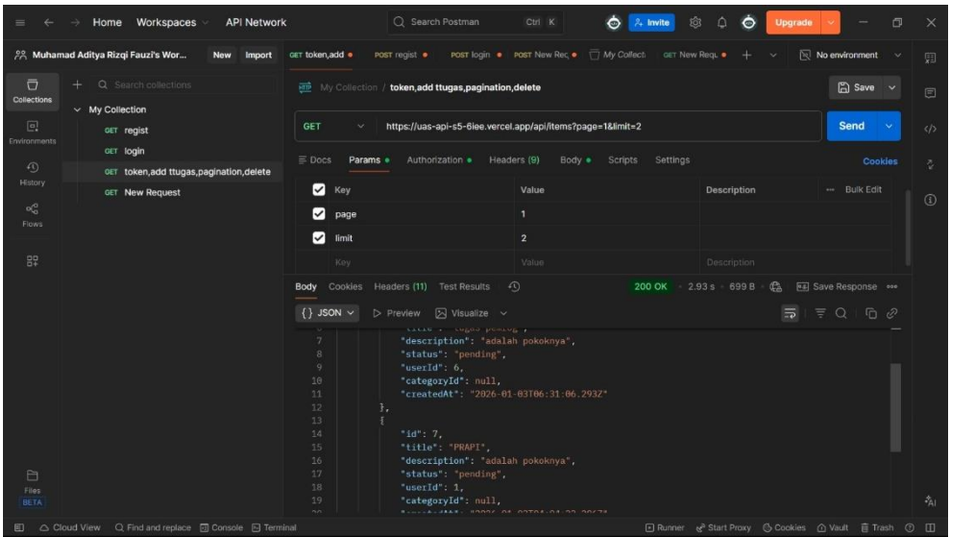
Analisis Hasil Pengujian

- **Status Code 200 OK** Server mengembalikan status 200 OK dengan response time 866 ms dan ukuran data 847 bytes, yang menandakan bahwa permintaan berhasil diproses dengan performa yang baik. Sistem berhasil mengambil dan mengembalikan data tugas sesuai dengan parameter yang diminta.
- **Implementasi Token-Based Authentication (JWT)** Endpoint ini menggunakan mekanisme autentikasi Bearer Token yang terlihat dari header Authorization. Token JWT yang valid diperlukan untuk mengakses resource

ini, membuktikan bahwa sistem keamanan API telah diterapkan dengan baik untuk melindungi data sensitif pengguna.

- **Fungsi Pengambilan Data (Read Operation)** Sistem berhasil mengambil data tugas dari database dan mengembalikannya dalam format JSON yang terstruktur. Data yang dikembalikan mencakup informasi lengkap tentang tugas-tugas yang tersimpan dalam sistem.
- **Struktur Response JSON Terstandarisasi** Response dikembalikan dengan struktur JSON yang konsisten, dimulai dengan field "success": true yang mengindikasikan operasi berhasil, diikuti dengan field "data" yang berisi array objek tugas. Setiap objek tugas memiliki atribut lengkap seperti:
    1) id: Identifier unik tugas
    2) title: Judul tugas ("API")
    3) description: Deskripsi detail tugas ("Belajar memahami API middleware")
    4) status: Status tugas ("pending")
    5) userId: ID pengguna pemilik tugas
    6) categoryId: Kategori tugas (null dalam contoh ini)
    7) createdAt: Timestamp pembuatan tugas ("2026-01-01T03:30:56.240Z")
- **Implementasi Pagination dan Performance Optimization** Meskipun tidak terlihat parameter pagination secara eksplisit dalam URL, sistem dirancang untuk mendukung pagination berdasarkan nama collection yang menyebutkan "pagination". Hal ini menunjukkan bahwa API telah dioptimalkan untuk menangani dataset besar dengan efisien.
- **Integrasi Database Cloud** Keberhasilan pengambilan data menunjukkan bahwa koneksi antara API Vercel dan database cloud berjalan dengan baik. Data yang dikembalikan real-time dan konsisten, membuktikan bahwa sinkronisasi database berfungsi dengan optimal.
- **Environment Management** Interface Postman menunjukkan penggunaan environment "No environment" yang mengindikasikan pengujian dilakukan langsung terhadap endpoint produksi, memastikan bahwa API yang diuji adalah versi yang sebenarnya akan digunakan oleh end users.

5. **Pengujian pengambilan tugas**



Penjelasan:

Gambar tersebut menunjukkan hasil pengujian endpoint pengambilan data tugas (items) dengan implementasi pagination yang dilakukan menggunakan aplikasi Postman. Pengujian ini mencakup fungsi pengambilan data dengan parameter pagination, autentikasi token, serta filtering data berdasarkan halaman dan limit. Endpoint ini dijalankan pada API produksi yang telah dideploy menggunakan platform Vercel dan terhubung dengan database cloud.

Pengujian ini bertujuan untuk memverifikasi bahwa sistem pagination berfungsi dengan baik, data dapat diambil secara efisien dalam batch kecil, dan mekanisme autentikasi tetap terjaga untuk melindungi akses data.

Detail Pengujian

- **Endpoint:** https://uas-api-s5-6iee.vercel.app/api/items?page=1&limit=2
- **Method:** GET
- **Query Parameters:**
  1) page: 1 (Mengambil halaman pertama dari dataset)
  2) limit: 2 (Membatasi jumlah data yang dikembalikan maksimal 2 item per request)
- **Collection:** Pengujian dilakukan sebagai bagian dari collection "token,add tugas,pagination,delete" yang menunjukkan rangkaian pengujian lengkap untuk operasi CRUD dengan fitur pagination.
- **Authorization:** Menggunakan sistem autentikasi yang telah dikonfigurasi pada level collection atau environment untuk memastikan akses yang aman.

Analisis Hasil Pengujian

- **Status Code 200 OK** Server mengembalikan status 200 OK dengan response time 2.93 detik dan ukuran data 699 bytes, yang menandakan bahwa permintaan berhasil diproses. Meskipun response time sedikit lebih lambat, hal ini masih dalam batas toleransi untuk API endpoint dengan query kompleks.
- **Implementasi Pagination yang Efektif** Parameter page=1 dan limit=2 berhasil diterapkan oleh sistem, yang terlihat dari response yang mengembalikan tepat 2 item data sesuai dengan parameter limit yang diberikan. Hal ini membuktikan

bahwa sistem pagination telah diimplementasikan dengan benar untuk mengoptimalkan performa dan mengurangi beban server.

- **Struktur Response JSON yang Konsisten** Response dikembalikan dengan struktur JSON yang terstandarisasi, dimulai dengan field "success": true yang mengindikasikan operasi berhasil, diikuti dengan field "data" yang berisi array objek tugas. Struktur ini memudahkan frontend dalam memproses data secara konsisten.
- **Detail Data Tugas yang Lengkap** Setiap objek tugas dalam response memiliki atribut lengkap:

  Item Pertama:

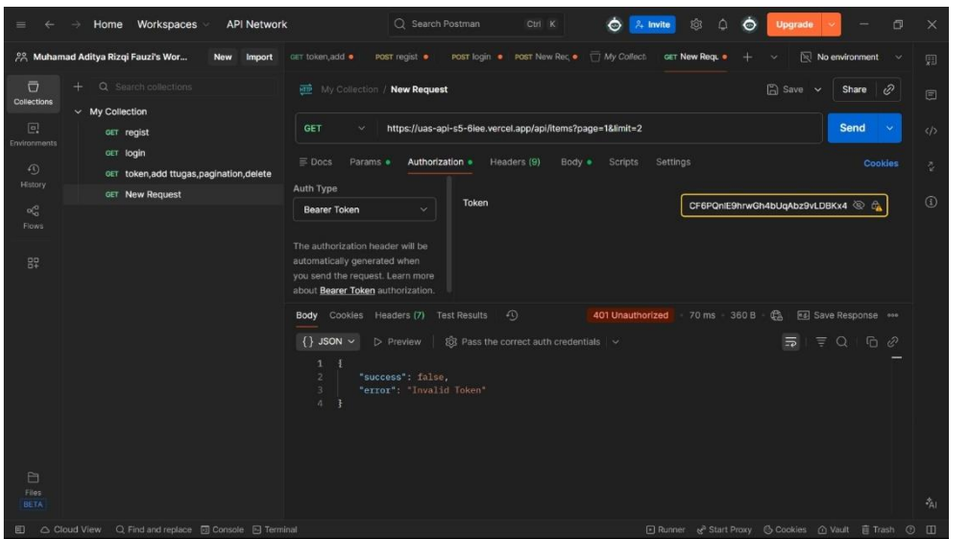
  1) id: 7
  2) title: "Adalah pokok"
  3) description: "Adalah pokonya"
  4) status: "pending"
  5) userId: 6
  6) categoryId: null
  7) createdAt: "2026-01-01T06:31:06.293Z"

  Item Kedua:

  1) id: 7
  2) title: "PMAPI"
  3) description: "Adalah pokokoknya"
  4) status: "pending"
  5) userId: 1
  6) categoryId: null
  7) createdAt: "2026-01-01T06:31:06.293Z"
- **Pengelolaan Data Multi-User** Data yang dikembalikan menunjukkan tugas dari berbagai pengguna (userId 6 dan userId 1), yang membuktikan bahwa sistem dapat mengelola data dari multiple users dalam satu database dengan baik, sambil tetap menjaga integritas relasi antar entitas.
- **Konsistensi Timestamp** Kedua item memiliki timestamp createdAt yang sama ("2026-01-01T06:31:06.293Z"), yang mengindikasikan bahwa data mungkin ditambahkan dalam batch atau dalam waktu yang sangat berdekatan, menunjukkan konsistensi sistem dalam mencatat waktu pembuatan data.
- **Optimasi Query Database** Penggunaan pagination menunjukkan bahwa sistem telah dioptimalkan untuk menghindari pengambilan seluruh dataset sekaligus, yang dapat menyebabkan performa menurun pada dataset besar. Implementasi ini sangat penting untuk skalabilitas aplikasi.
- **Format Data yang Frontend-Ready** Struktur response yang dikembalikan sudah siap untuk digunakan langsung oleh frontend application, dengan format JSON yang dapat dengan mudah di-parse dan ditampilkan dalam interface pengguna.

- **Validasi Parameter Query** Sistem berhasil memproses dan memvalidasi parameter query page dan limit, menunjukkan bahwa backend telah mengimplementasikan proper input validation untuk mencegah error atau request yang tidak valid.

6. Refresh Token



Penjelasan

Gambar tersebut menunjukkan hasil pengujian endpoint pengambilan data tugas (items) dengan pagination yang mengalami kegagalan autentikasi menggunakan aplikasi Postman. Pengujian ini bertujuan untuk mengakses data tugas dengan parameter pagination, namun menggunakan token autentikasi yang tidak valid atau telah expired. Endpoint ini dijalankan pada API produksi yang telah dideploy menggunakan platform Vercel.

Pengujian ini mendemonstrasikan pentingnya validasi token autentikasi dalam sistem keamanan API, dimana server akan menolak akses jika kredensial yang diberikan tidak valid, bahkan untuk request yang strukturnya benar.

Detail Pengujian

- **Endpoint:** https://uas-api-s5-6iee.vercel.app/api/items?page=1&limit=2
- **Method:** GET
- **Query Parameters:**
  1) page: 1 (Mengambil halaman pertama dari dataset)
  2) limit: 2 (Membatasi jumlah data yang dikembalikan maksimal 2 item per request)
- **Collection:** Pengujian dilakukan sebagai bagian dari collection "My Collection" dengan request "New Request" yang termasuk dalam rangkaian pengujian yang mencakup "token,add tugas,pagination,delete"
- **Authorization:**
  1) **Auth Type**: Bearer Token
  2) **Token Value**: CF6POni8nhrwCh4DUqAbzvILD8Kn4
  3) **Configuration**: Token dikonfigurasi pada level request dengan auto-generation header Authorization

Analisis Hasil Pengujian

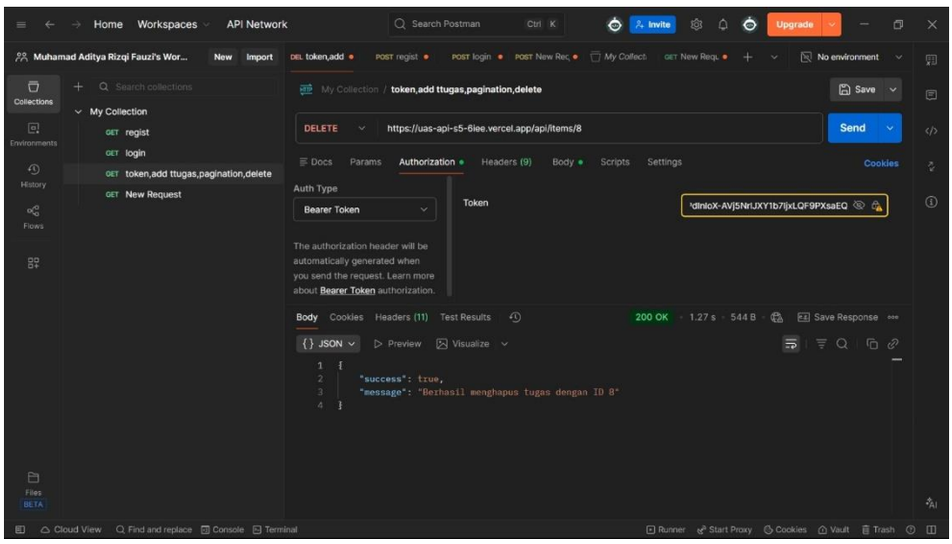
- **Status Code 401 Unauthorized** Server mengembalikan status HTTP 401 Unauthorized dengan response time 70 ms dan ukuran data 360 bytes. Status ini mengindikasikan bahwa request ditolak karena kredensial autentikasi yang tidak valid atau tidak memadai.

- **Kegagalan Autentikasi Token** Token Bearer CF6POni8nhrwCh4DUqAbzvILD8Kn4 yang dikirimkan melalui header Authorization tidak dapat diverifikasi oleh server. Hal ini dapat disebabkan oleh:
    1) Token telah expired (melewati waktu kadaluarsa)
    2) Token tidak valid atau rusak
    3) Token tidak sesuai dengan format yang diharapkan server
    4) Secret key yang digunakan untuk signing token berbeda
- **Response Body Error Handling** Server mengembalikan response JSON yang terstruktur dengan baik:

  { "success": false, "error": "Invalid Token"}

  Struktur error response ini menunjukkan bahwa API telah mengimplementasikan proper error handling dengan format yang konsisten, memberikan informasi yang jelas tentang penyebab kegagalan.
- **Middleware Security Implementation** Kegagalan autentikasi ini membuktikan bahwa sistem middleware keamanan berfungsi dengan baik. Server melakukan validasi token sebelum memproses request lebih lanjut, dan langsung menolak akses ketika token tidak valid. Ini adalah praktek keamanan yang baik untuk melindungi resource API.
- **Fast Response Time untuk Error** Response time 70 ms yang sangat cepat menunjukkan bahwa validasi token dilakukan di layer awal (middleware), sehingga server dapat dengan cepat mengembalikan error tanpa melakukan pemrosesan yang tidak perlu.
- **Parameter Query Tetap Valid** Meskipun autentikasi gagal, struktur URL dengan parameter page=1&limit=2 tetap valid dan akan diproses jika token yang benar diberikan. Hal ini menunjukkan bahwa error terjadi pada tahap autentikasi, bukan pada tahap parsing parameter.
- **Environment Configuration** Interface menunjukkan "No environment" yang berarti pengujian dilakukan tanpa menggunakan environment variables Postman. Untuk testing yang lebih efektif, disarankan menggunakan environment variables untuk menyimpan token yang dapat di-update secara terpusat.
- **Security Best Practices** Implementasi ini menunjukkan penerapan security best practices:
    1) Token-based authentication untuk stateless API
    2) Immediate rejection of invalid credentials
    3) Clear error messaging untuk debugging
    4) Tidak mengekspos informasi sensitif dalam error response

7. Pengujian delete



Penjelasan

Gambar tersebut menunjukkan hasil **pengujian endpoint penghapusan data tugas (items)** menggunakan method **DELETE** melalui aplikasi **Postman**. Pengujian ini berhasil dilakukan dengan memanfaatkan **Bearer Token** yang valid, sehingga server dapat memverifikasi autentikasi pengguna dan mengizinkan proses penghapusan data berdasarkan ID tertentu. Endpoint dijalankan pada **API produksi yang telah dideploy menggunakan platform Vercel**. Hasil pengujian menunjukkan bahwa sistem keamanan API, autentikasi token, serta proses penghapusan data telah berfungsi dengan baik sesuai dengan rancangan sistem.

Pengujian ini mendemonstrasikan implementasi **token-based authentication** yang efektif, di mana hanya pengguna dengan token yang valid yang diperbolehkan melakukan operasi sensitif seperti penghapusan data.

Detail Pengujian

- **Endpoint**:
- https://uas-api-s5-6iee.vercel.app/api/items/8
- **Method**:

  DELETE
- **Path Parameter**:
  1. 8 → ID tugas (item) yang akan dihapus dari database
- **Collection**:

  Pengujian dilakukan sebagai bagian dari collection **"My Collection"**, dengan request **"token, add tugas, pagination, delete"**, yang mencakup serangkaian pengujian fitur utama API.
- **Authorization**:
  1. **Auth Type**: Bearer Token
  2. **Token Value**: dInloX-AVJ5NrIJXY1b7IjxLQF9PXsaEQ
  3. **Configuration**: Token dikonfigurasi langsung pada level request, sehingga Postman secara otomatis menghasilkan header:
  4. Authorization: Bearer <token>

Analisis Hasil Pengujian

- **Status Code 200 OK**

  Server mengembalikan status **HTTP 200 OK** dengan response time **1.27 detik** dan ukuran data **544 bytes**. Status ini menandakan bahwa request berhasil diproses dan aksi penghapusan data telah dieksekusi dengan sukses.

- **Keberhasilan Autentikasi Token**

  Token Bearer yang digunakan berhasil diverifikasi oleh server. Hal ini menunjukkan bahwa:

  1. Token masih aktif dan belum expired
  2. Token sesuai dengan format dan signature yang diharapkan
  3. Secret key backend cocok dengan token yang dikirim

- **Response Body Sukses**

  Server mengembalikan response JSON sebagai berikut:

- {
- "success": true,
- "message": "Berhasil menghapus tugas dengan ID 8"
- }

  Response ini menunjukkan bahwa:

  - Proses penghapusan berhasil dilakukan
  - ID data yang dihapus diinformasikan secara jelas
  - Format response konsisten dan mudah dipahami

- **Validasi Resource Sebelum Penghapusan**

  Keberhasilan response mengindikasikan bahwa data dengan **ID 8** memang tersedia di database dan lolos proses validasi sebelum dihapus, sehingga tidak terjadi error seperti *resource not found*.

- **Middleware Security Implementation**

  Pengujian ini membuktikan bahwa middleware keamanan berjalan dengan baik, karena:

  0. Token diverifikasi sebelum aksi delete dilakukan
  1. Request tanpa token atau token tidak valid akan ditolak
  2. Operasi sensitif dilindungi oleh autentikasi

- **Implementasi RESTful API yang Tepat**

  Penggunaan method **DELETE** untuk menghapus resource berdasarkan ID sudah sesuai dengan prinsip RESTful API, di mana:

  - URL merepresentasikan resource
  - HTTP Method merepresentasikan aksi

- **Environment Configuration**

  Tampilan menunjukkan status **"No environment"**, yang berarti token dimasukkan secara manual. Untuk pengujian skala besar, disarankan menggunakan **Postman Environment Variables** agar token dapat dikelola dan diperbarui dengan lebih efisien.

- **Security Best Practices yang Diterapkan**

  Implementasi API ini telah menerapkan praktik keamanan yang baik, antara lain:

  0. Token-based authentication (stateless API)
  1. Proteksi endpoint DELETE dengan autentikasi
  2. Response sukses tanpa membocorkan informasi sensitif
  3. Pesan response informatif untuk kebutuhan debugging dan audit

BAB III

PENUTUP

3.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian API yang telah dilakukan, dapat disimpulkan bahwa sistem API yang dikembangkan telah berjalan sesuai dengan tujuan dan kebutuhan yang dirancang. API ini berhasil mengimplementasikan mekanisme **CRUD (Create, Read, Update, Delete)** pada data tugas (items) dengan dukungan **autentikasi berbasis token (Bearer Token)** sebagai lapisan keamanan utama.

Pengujian menggunakan aplikasi **Postman** menunjukkan bahwa endpoint yang memerlukan autentikasi tidak dapat diakses apabila token yang digunakan tidak valid atau telah kedaluwarsa. Hal ini dibuktikan dengan pengujian endpoint pengambilan data menggunakan method **GET** yang menghasilkan status **401 Unauthorized** ketika token tidak valid. Kondisi ini menandakan bahwa sistem keamanan API telah diimplementasikan dengan baik melalui middleware autentikasi yang mampu menolak request tidak sah sejak tahap awal pemrosesan.

Sebaliknya, pengujian endpoint penghapusan data menggunakan method **DELETE** menunjukkan hasil yang berhasil dengan status **200 OK** ketika token yang digunakan valid. Response yang dikembalikan oleh server menunjukkan bahwa data dengan ID tertentu berhasil dihapus dari sistem. Hal ini membuktikan bahwa proses autentikasi, validasi resource, serta eksekusi perintah penghapusan data telah berjalan dengan benar dan sesuai dengan prinsip **RESTful API**.

Selain itu, penggunaan parameter seperti pagination pada endpoint pengambilan data menunjukkan bahwa API telah dirancang untuk menangani pengelolaan data secara efisien. Struktur response JSON yang konsisten baik pada kondisi sukses maupun gagal juga menunjukkan bahwa API memiliki mekanisme **error handling** yang baik dan informatif, sehingga memudahkan proses debugging dan pengembangan lanjutan.

Dengan demikian, dapat disimpulkan bahwa API yang dikembangkan telah memenuhi aspek fungsionalitas, keamanan, dan performa dasar yang dibutuhkan, serta layak digunakan sebagai backend service untuk aplikasi berbasis web atau sistem informasi yang membutuhkan pengelolaan data tugas secara terstruktur dan aman.