LAPORAN UAS PEMROGRAMAN API

Dosen Pengampu : Saiful Nur Budiman S.Kom.,M.Kom

Disusun oleh:

1. Dalilul Falihin (23104410087)
2. Adrian Nugraha Ardi (23104410097)
3. Dini Ika Kurnia (23104410064)
4. Helda imilda nugroho (23104410109)
5. M Rizky Galuh P (23104410107)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN INFORMATIKA

UNIVERSITAS ISLAM BALITAR

2026

BAB 1

PENDAHULUAN

Dalam ekosistem pengembangan perangkat lunak modern, pemisahan antara sisi klien (*frontend*) dan sisi server (*backend*) telah menjadi arsitektur standar. Penghubung utama dari kedua sisi tersebut adalah **Application Programming Interface (API)**. Di antara berbagai jenis API, **REST (Representational State Transfer)** merupakan gaya arsitektur yang paling banyak diadopsi karena kesederhanaannya, skalabilitasnya, dan kemampuannya untuk beroperasi di atas protokol HTTP yang bersifat *stateless*.

Seiring dengan meningkatnya volume pertukaran data sensitif di internet, keamanan API menjadi isu yang sangat krusial. Sebuah API yang buruk tidak hanya menyebabkan kebocoran data, tetapi juga rentan terhadap serangan seperti *SQL Injection* atau akses ilegal. Oleh karena itu, pengembangan API masa kini menuntut integrasi teknologi yang mampu menjamin integritas dan kerahasiaan data.

Penggunaan **Next.js** dalam proyek ini dipilih karena kemampuannya menyediakan *API Routes* yang efisien di lingkungan Node.js. Untuk mendukung manajemen data yang kuat, **PostgreSQL** digunakan sebagai sistem manajemen basis data relasional, yang dikelola melalui **Prisma ORM**. Prisma memberikan keuntungan berupa *type-safety* dan abstraksi yang memudahkan pengembang dalam berinteraksi dengan database tanpa harus menulis query SQL manual secara mentah.

Banyak aplikasi web yang masih memiliki kelemahan pada sisi kontrol akses. Melalui proyek ini, masalah-masalah tersebut diatasi dengan cara:

- Mencegah akses ilegal pada endpoint sensitif menggunakan *Middleware* validasi token.
- Mengatur hak akses CRUD agar lebih tertata, di mana tidak semua pengguna memiliki hak yang sama (Leveling User vs Admin).
- Menjamin penyimpanan data kredensial yang aman melalui teknik *hashing* satu arah.

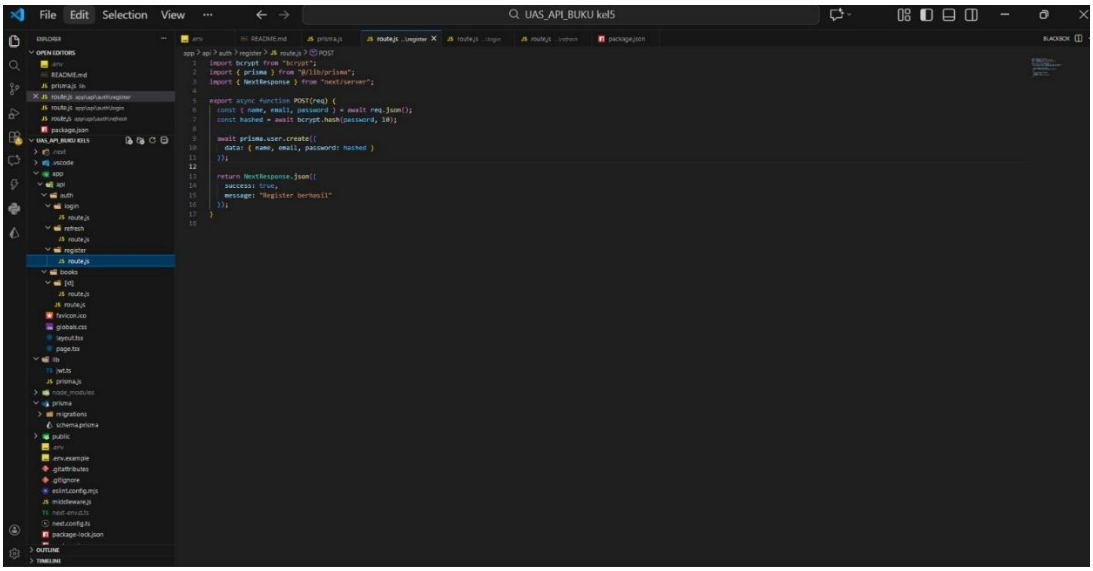
Pengerjaan ini digunakan untuk pemenuhan tugas UAS yang diberikan dan memberikan manfaat praktis bagi pengembang dalam memahami alur kerja *full-stack pembangunan API*, mulai dari perancangan skema database, pembuatan endpoint, hingga pengujian keamanan menggunakan perangkat lunak seperti Postman. Selain itu, proyek ini melatih kemampuan dalam menyusun kode yang bersih (*clean code*) dan terstruktur sesuai dengan standar industri teknologi informasi saat ini

BAB 2

PEMBAHASAN

1. Authentication dan Authorization

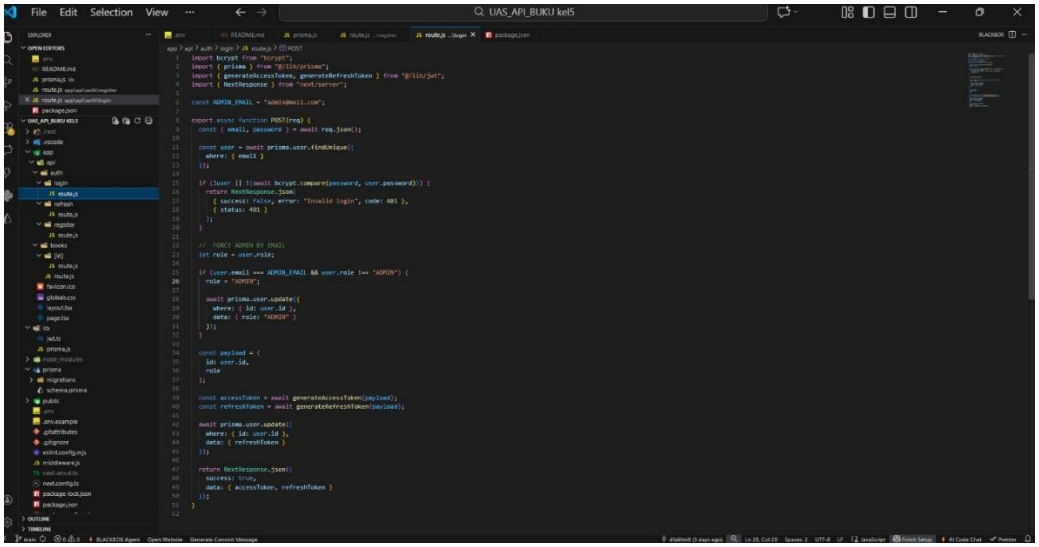
a. Endpoint REGISTER



- **import bcrypt**: Memanggil library untuk melakukan *hashing* password. Ini mengubah password asli menjadi deretan karakter acak yang tidak bisa dibaca manusia.
- **import { prisma }**: Memanggil koneksi database Prisma yang biasanya sudah kamu konfigurasi di folder lib. Ini adalah alat untuk berinteraksi (baca/tulis) dengan database.
- **import { NextResponse }**: Fitur bawaan Next.js untuk mengirim balik respon (seperti pesan sukses atau error) ke sisi client/browser.
- **export async function POST**: Menentukan bahwa file ini hanya menerima request dengan metode **POST** (biasanya digunakan untuk mengirim data sensitif atau membuat data baru). async digunakan karena operasi database butuh waktu (asinkron).
- **(req)**: Singkatan dari *request*, yaitu objek yang berisi semua data yang dikirimkan oleh pengguna (nama, email, password).
- Pada angka **10** di password disebut *salt rounds*. Semakin tinggi angkanya, semakin sulit password itu diretas, tapi semakin lama juga proses pembuatannya. Angka 10 adalah standar yang seimbang
- **prisma.user.create**: Memerintahkan Prisma untuk memasukkan baris data baru ke tabel user.
- **data: { ... }**: Berisi objek data yang akan disimpan. Perhatikan bahwa yang disimpan ke kolom password adalah variabel **hashed**, bukan password asli dari user.

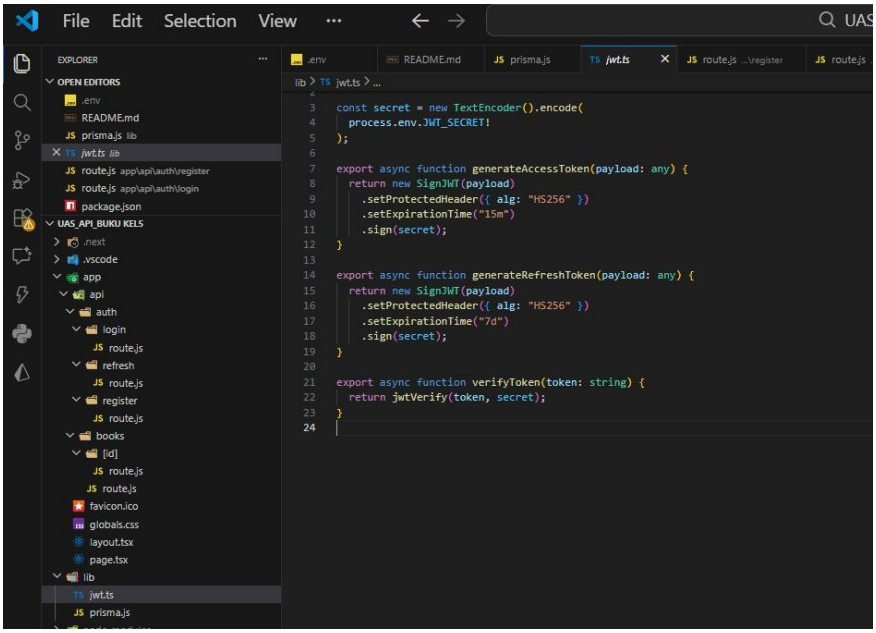
.

b. Endpoint LOGIN



- **generateAccessToken & generateRefreshToken**: Fungsi untuk membuat token JWT. Access token biasanya berumur pendek (misal 15 menit), sedangkan refresh token berumur panjang (misal 7 hari).
- **ADMIN_EMAIL**: Sebuah variabel konstanta untuk menentukan email mana yang secara otomatis akan dianggap sebagai Admin.
- Program mengambil input email dan password dari user.
- **prisma.user.findUnique**: Mencari data pengguna di database berdasarkan email. Jika email tidak ditemukan, variabel user akan bernilai null.
- **!user**: Mengecek jika user tidak ditemukan.
- **bcrypt.compare**: Membandingkan password teks biasa dari input dengan password ter-hash yang ada di database.
- Jika salah satu gagal, kirim respon **401 (Unauthorized)**.
- **payload**: Data ringkas pengguna yang akan dimasukkan ke dalam token.

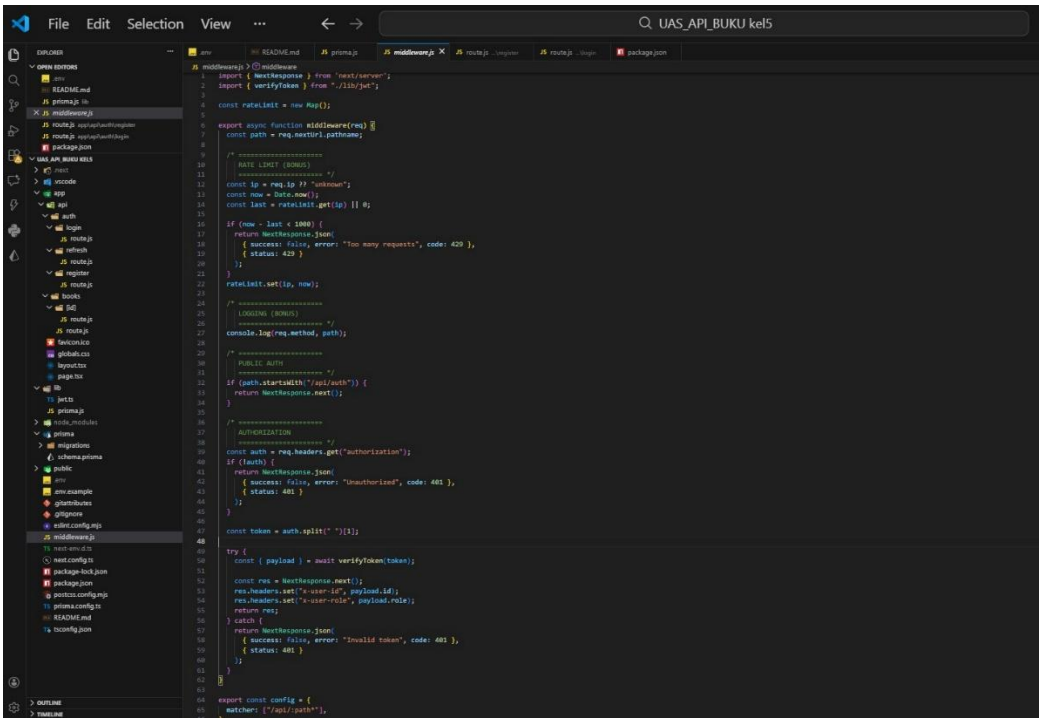
c. JWT



- **SignJWT & jwtVerify**: Dua fungsi utama; satu untuk membuat tanda tangan digital (tanda pengenal), satu lagi untuk memeriksa apakah tanda tangan tersebut asli atau palsu.
- **TextEncoder().encode(...)**: Library jose membutuhkan kunci dalam bentuk *Unit8Array* (deretan angka biner), bukan teks biasa. Fungsi ini mengubah teks dari file .env kamu menjadi format tersebut.

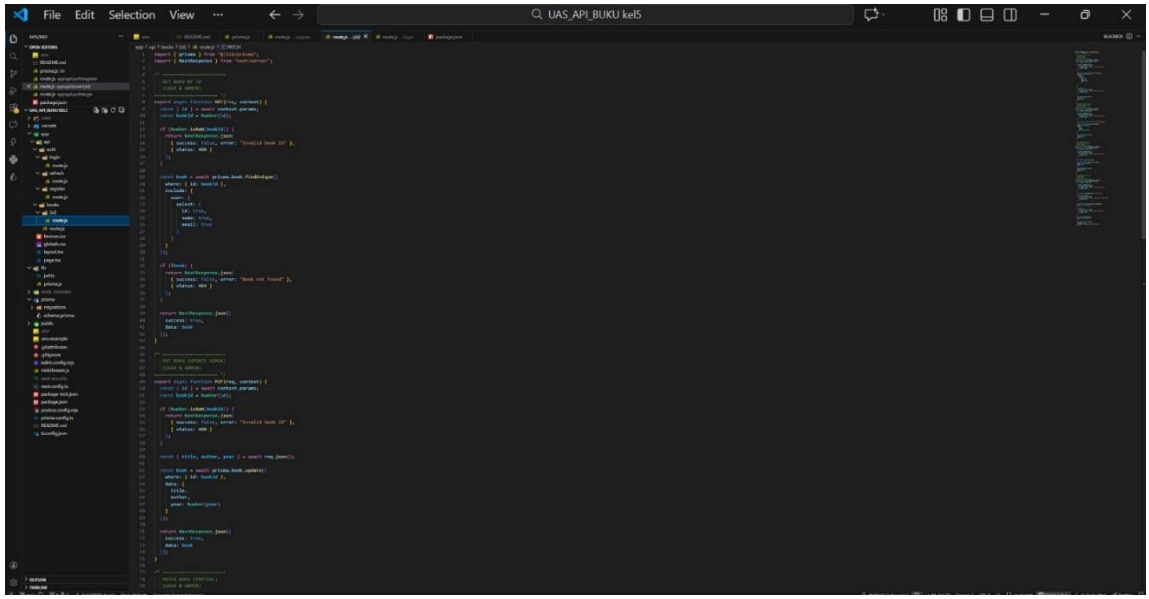
- **process.env.JWT_SECRET!**: Ini mengambil kode rahasia dari variabel lingkungan. Tanda seru (!) di akhir memberitahu TypeScript bahwa kita yakin variabel ini pasti ada.
- **payload**: Berisi data user (seperti ID dan Role). Ini adalah informasi yang "dibungkus" di dalam token.
- **.setProtectedHeader({ alg: "HS256" })**: Menentukan algoritma enkripsi. HS256 adalah standar industri yang aman.
- **.setExpirationTime("15m")**: Menentukan masa berlaku. Di sini disetel 15 menit. Access token sengaja dibuat singkat agar jika dicuri, pencuri hanya punya waktu sedikit untuk menggunakannya.
- .**sign(secret)**: Tahap terakhir di mana token dikunci menggunakan kunci rahasia.

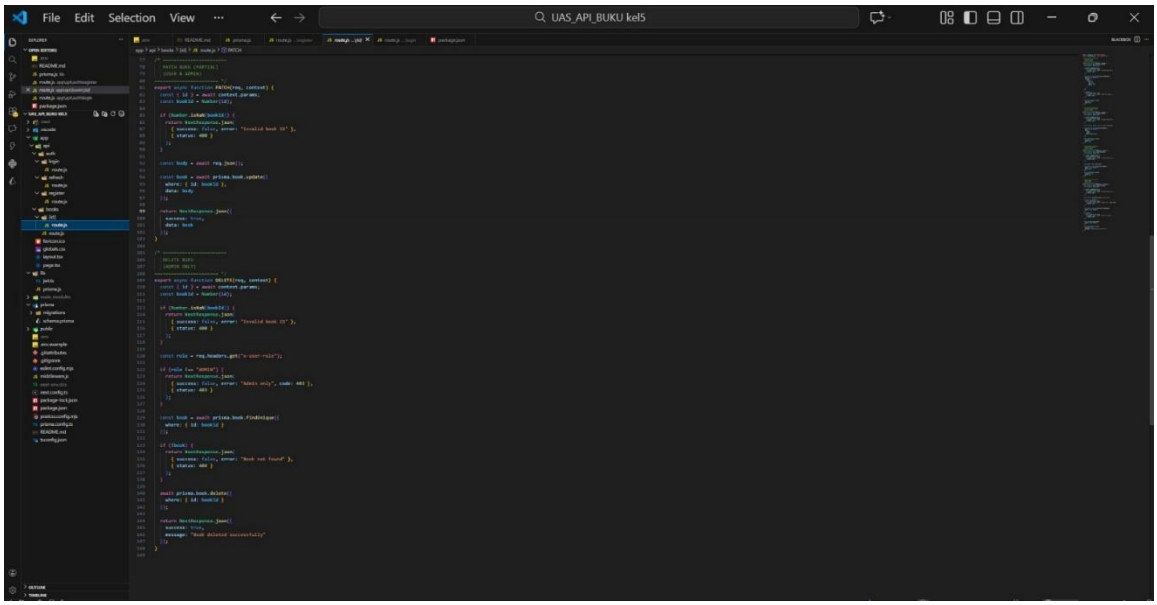
2. Middleware



- **rateLimit**: Menggunakan Map untuk menyimpan data IP sementara. Tujuannya membatasi jumlah klik/request.
- **config.matcher**: Menentukan di mana satpam ini bekerja. Di sini, ia hanya akan memeriksa rute yang dimulai dengan /api/. Halaman biasa (seperti /about) tidak akan diperiksa.
- Jika jaraknya kurang dari 1 detik (**1000 ms**), server menolak dengan kode **429 (Too Many Requests)**. Ini berguna mencegah serangan *Brute Force* atau bot.
- Middleware mencari Header bernama `Authorization`.
- Biasanya formatnya adalah `Bearer <token_disini>`. Kode `.split(" ")[1]` mengambil tokennya saja dan membuang kata "Bearer".
- **`verifyToken(token)`** : Memeriksa apakah token tersebut asli, belum kedaluwarsa, dan dibuat oleh server kita.
- **Custom Headers**: Jika valid, ID dan Role user "ditempelkan" ke header request baru (`x-user-id` & `x-user-role`).
- **Keuntungan**: API di rute selanjutnya (misal: `/api/profile`) tidak perlu membedah token lagi, cukup baca dari header saja.

3. Books API (route buku dan route id)

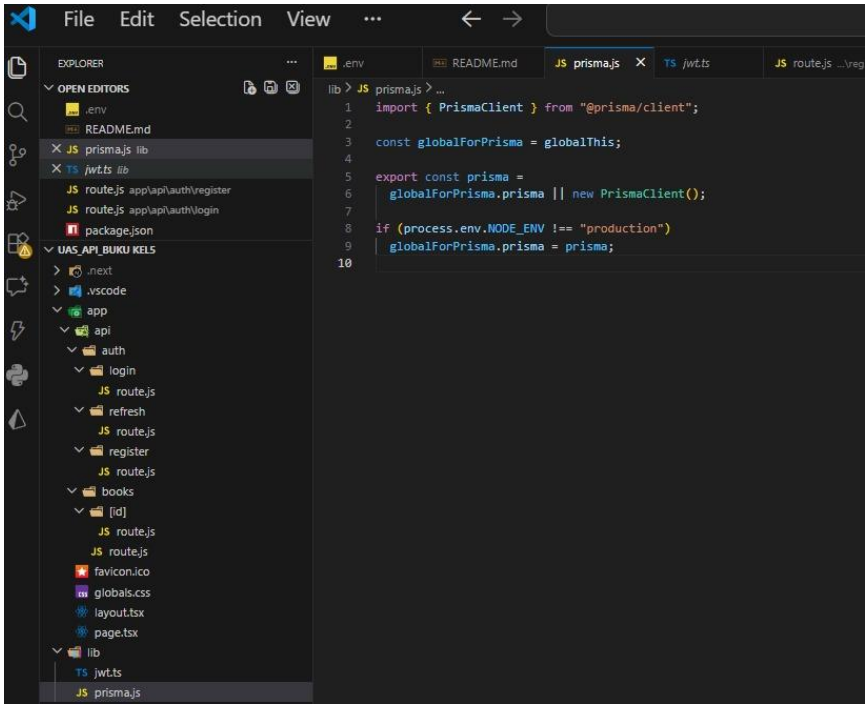




- **export**: Mengizinkan file lain (mesin Next.js) menggunakan fungsi ini.
- **async**: Memberitahu komputer bahwa fungsi ini butuh waktu untuk selesai (karena akses database).
- **function**: Penanda bahwa ini adalah blok kode yang melakukan tugas tertentu.
- **PATCH**: Nama metode HTTP. Digunakan untuk memperbarui data secara **sebagian** (misal hanya judul saja).
- **req**: (*Request*) Objek yang membawa data dari pengirim (seperti isi teks/body).
- **context**: Wadah informasi tambahan dari rute, seperti ID yang ada di alamat URL.
- **const**: Membuat tempat simpan data tetap (variabel) yang tidak bisa diubah isinya nanti.
- **{ id }**: Mengambil hanya data bernama id dari dalam context.params (teknik *destructuring*).
- **await**: Berhenti sejenak sampai data parameter URL siap dibaca.
- **body**: Nama variabel untuk menampung data yang dikirim user (misal: { "title": "Buku Baru" }).
- **req.json()**: Perintah untuk menerjemahkan paket kiriman dari format teks mentah menjadi format objek yang dipahami JavaScript.
- **book**: Menyimpan hasil data yang sudah berhasil diperbarui.
- **prisma**: Jembatan penghubung ke database kamu.
- **.book**: Nama tabel database yang dituju.
- **.update**: Perintah database untuk mengubah data yang sudah ada.
- **where**: Kata kunci "yang mana?".

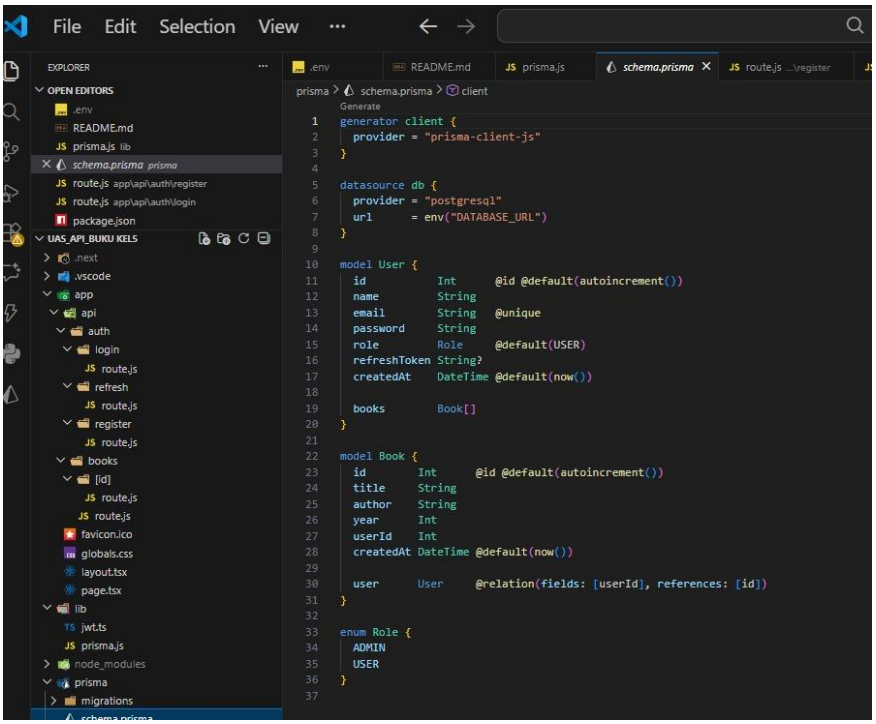
- **id: bookId**: "Ubah data yang kolom ID-nya cocok dengan nomor ini".
- **data**: Apa yang ingin diubah?
- **body**: Isi perubahannya (apa pun yang dikirim user tadi).
- **role**: Variabel penyimpan jabatan user (Admin/User).
- **req.headers**: Bagian "amplop" surat request yang berisi metadata.
- **.get("x-user-role")**: Mengambil label jabatan yang sebelumnya sudah ditempelkan oleh **Middleware**.
- **if**: "Jika".
- **role**: Isi jabatannya.
- **!==**: "Tidak sama dengan".
- **"ADMIN"**: Target jabatan yang diperbolehkan.
- **return**: Selesaikan fungsi sekarang juga dan kirim jawaban ke user.
- **403**: Kode status HTTP untuk **Forbidden** (Dilarang). Artinya: "Server tahu siapa kamu, tapi kamu tidak punya izin menghapus ini".

4. Database / Prisma



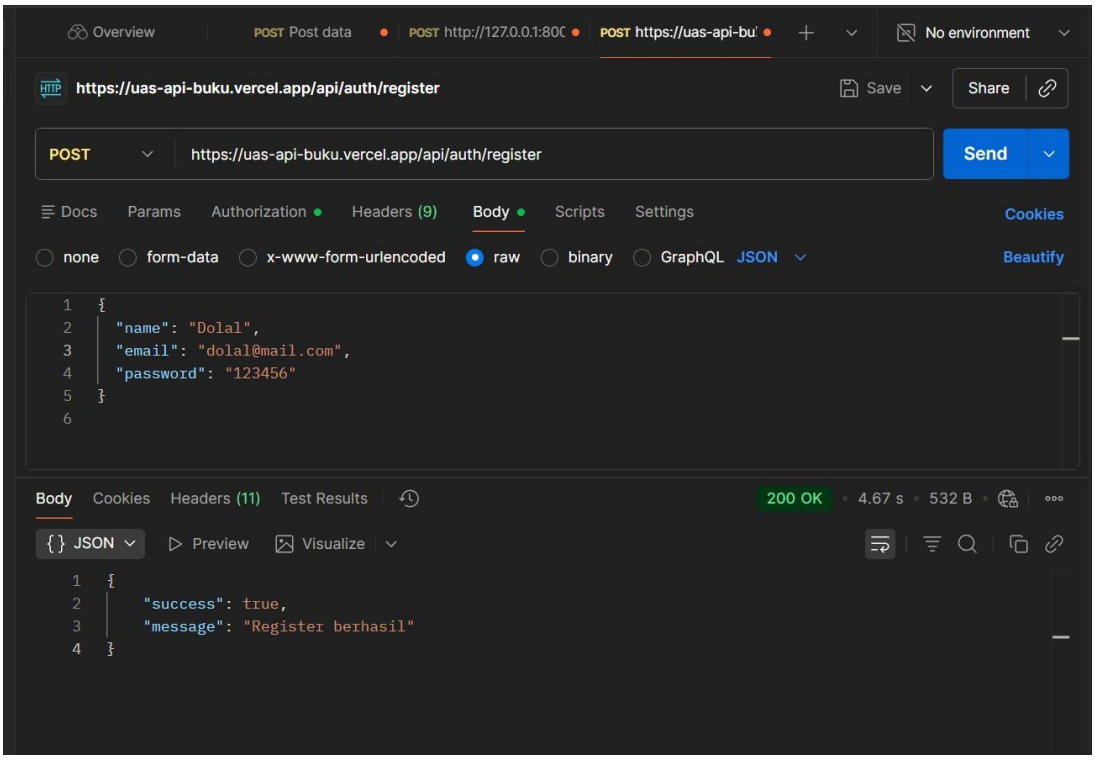
- **import**: Mengambil alat dari folder lain.
- **{ PrismaClient }**: Nama "alat utama" dari Prisma yang berfungsi untuk mengobrol dengan database.
- **from "@prisma/client"**: Lokasi tempat alat tersebut berada.
- **const**: Membuat variabel tetap.
- **globalForPrisma**: Nama yang kita berikan untuk tempat penyimpanan global.
- **globalThis**: Objek spesial di JavaScript yang selalu ada di mana pun (seperti gudang penyimpanan pusat di server).
- **`export`**: Agar variabel `prisma` bisa dipanggil dan dipakai di file lain (seperti file registrasi atau login kamu).
- **`prisma`**: Nama variabel yang akan kita pakai untuk perintah seperti `prisma.user.create`.
- **`||`**: Simbol "ATAU".
- **`new PrismaClient()`**: Perintah untuk membuat koneksi baru ke database.
- **`if`**: "Jika".
- **`process.env.NODE_ENV`**: Mengecek status aplikasi kita (sedang tahap pembuatan atau sudah jadi/online).
- **`!==`**: "Tidak sama dengan".
- **`"production"`**: Status aplikasi yang sudah online secara resmi.

- **`globalForPrisma.prisma = prisma`**: Menyimpan koneksi yang baru dibuat tadi ke gudang global.



- **generator**: Instruksi untuk Prisma agar membuatkan "alat bantu" (SDK).
- **provider**: Memberitahu Prisma untuk menghasilkan kode dalam bahasa **JavaScript/TypeScript** agar bisa kamu pakai di folder lib/prisma.
- **datasource**: Menentukan di mana data disimpan.
- **provider = "postgresql"**: Kamu memberitahu Prisma bahwa kamu menggunakan database **PostgreSQL**.
- **url = env(...)**: Alamat database tidak ditulis langsung di sini demi keamanan, melainkan diambil dari file rahasia .env.
- **model**: Kata kunci untuk membuat tabel baru di database.
- **id**: Nama kolom.
- **Int**: Tipe datanya adalah angka bulat.
- **@id**: Menandakan ini adalah **Primary Key** (tanda pengenal unik).
- **@default(autoincrement())**: Angka akan bertambah otomatis (1, 2, 3...) setiap ada pendaftar baru.
- **@unique**: Memastikan tidak boleh ada dua orang yang daftar dengan email yang sama.
- **Role**: Mengambil pilihan dari enum Role (ADMIN atau USER).
- **String?**: Tanda tanya berarti **Optional** (boleh kosong). User yang belum login tidak punya refreshToken.
- **userId**: Kolom yang menyimpan ID siapa pemilik buku ini.
- **@relation**: Menjelaskan hubungan (relasi).
- **fields: [userId]**: Kolom di tabel ini yang menyambung ke tabel User.
- **references: [id]**: Menyambung ke kolom mana di tabel User? Jawabannya kolom id.
- **`enum`**: Singkatan dari *Enumeration*. Digunakan untuk membuat daftar pilihan yang sudah pasti. Ini mencegah adanya role aneh seperti "SUPER_USER" secara tidak sengaja, karena hanya ADMIN dan USER yang diizinkan.

5. **Testing**
    a. **Register**



Berdasarkan gambar yang Anda unggah, berikut adalah penjelasan mengenai hasil testing API yang dilakukan menggunakan Postman (atau tool serupa):

1. Endpoint dan Metode HTTP

- **Metode:** POST. Ini berarti Anda sedang mengirimkan data baru ke server.
- **URL:** https://uas-api-buku.vercel.app/api/auth/register. Ini adalah endpoint yang digunakan untuk melakukan **registrasi (pendaftaran)** pengguna baru.

2. Request Body (Data yang Dikirim)

Kami mengirimkan data dalam format **JSON** melalui tab Body (raw):

- name: "Dolal"
- email: "dolal@mail.com"
- password: "123456"

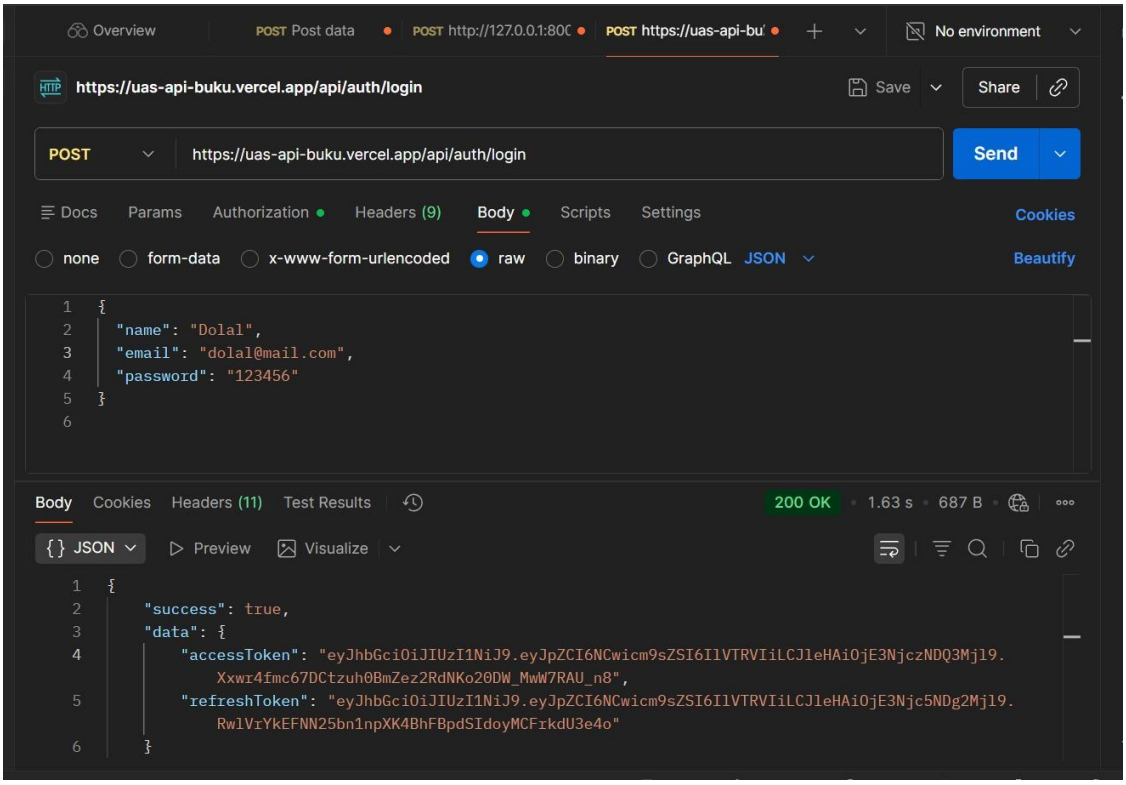
Data inilah yang diproses oleh sistem untuk membuat akun baru di database mereka.

3. Hasil Respon (Response Output)

Di bagian bawah gambar, kita bisa melihat hasil kembalian dari server:

- **Status Code: 200 OK**. Ini menunjukkan bahwa permintaan berhasil diproses oleh server tanpa kendala teknis.
- **Body Response (JSON):**
    - "success": true: Menandakan operasi berhasil secara logika bisnis.
    - "message": "Register berhasil": Konfirmasi teks dari server bahwa proses pendaftaran user "Dolal" telah selesai.

b. Login



Berdasarkan gambar diatas yang Anda unggah, berikut adalah penjelasan mengenai hasil testing API untuk proses **Login**:

1. Endpoint dan Perubahan Metode

- **Metode:** Tetap menggunakan **POST** untuk mengirimkan kredensial secara aman.
- **URL:** https://uas-api-buku.vercel.app/api/auth/login. Endpoint ini khusus digunakan untuk memverifikasi pengguna yang sudah terdaftar.

2. Request Body (Kredensial Login)

Anda mengirimkan data yang sama dengan saat registrasi:

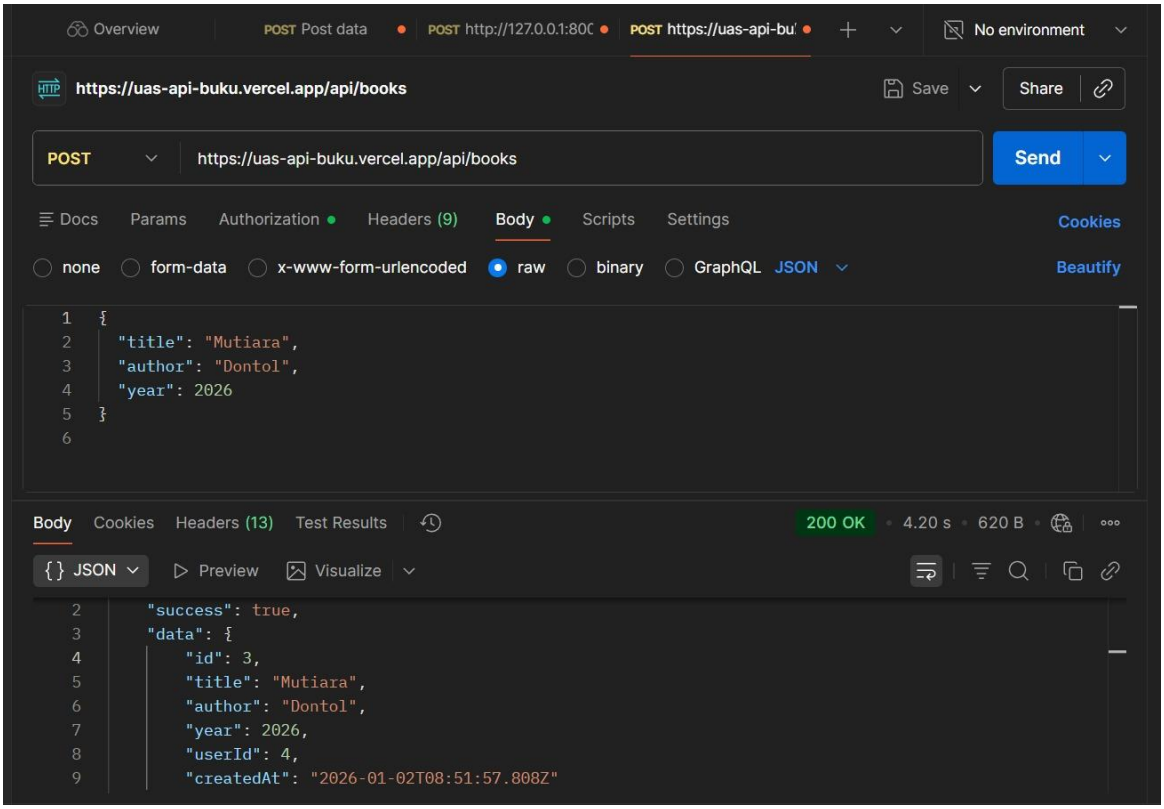
- name: "Dolal"
- email: "dolal@mail.com"
- password: "123456"

3. Hasil Respon (Data Autentikasi)

Respon kali ini jauh lebih penting karena berisi kunci akses untuk masuk ke sistem:

- **Status Code: 200 OK**: Menandakan login berhasil dan kredensial cocok.
- **JSON Data:**
  - "success": true: Mengonfirmasi identitas Anda divalidasi.
  - "accessToken": Ini adalah token JWT (JSON Web Token) yang bersifat sementara. Token ini harus Anda sertakan di **Header Authorization** untuk mengakses API lain yang bersifat privat (misalnya: menambah buku atau melihat profil).
  - "refreshToken": Digunakan untuk mendapatkan accessToken baru jika yang lama sudah kedaluwarsa tanpa harus login ulang.

c. Tambah data buku



1. Proses Login (Gambar Pertama)
Pada tahap ini, Anda melakukan verifikasi identitas untuk mendapatkan izin akses ke sistem.

- **Endpoint:** POST .../api/auth/login.
- **Request:** Mengirimkan data email dan password milik "Dolal".
- **Response:**
  - **Status 200 OK:** Login berhasil.
  - **accessToken**: Ini adalah "kunci digital" (JWT) yang sangat penting. Token ini harus digunakan pada request berikutnya agar server mengenali siapa Anda.
  - **refreshToken**: Digunakan untuk memperbarui akses jika token utama habis masa berlakunya.

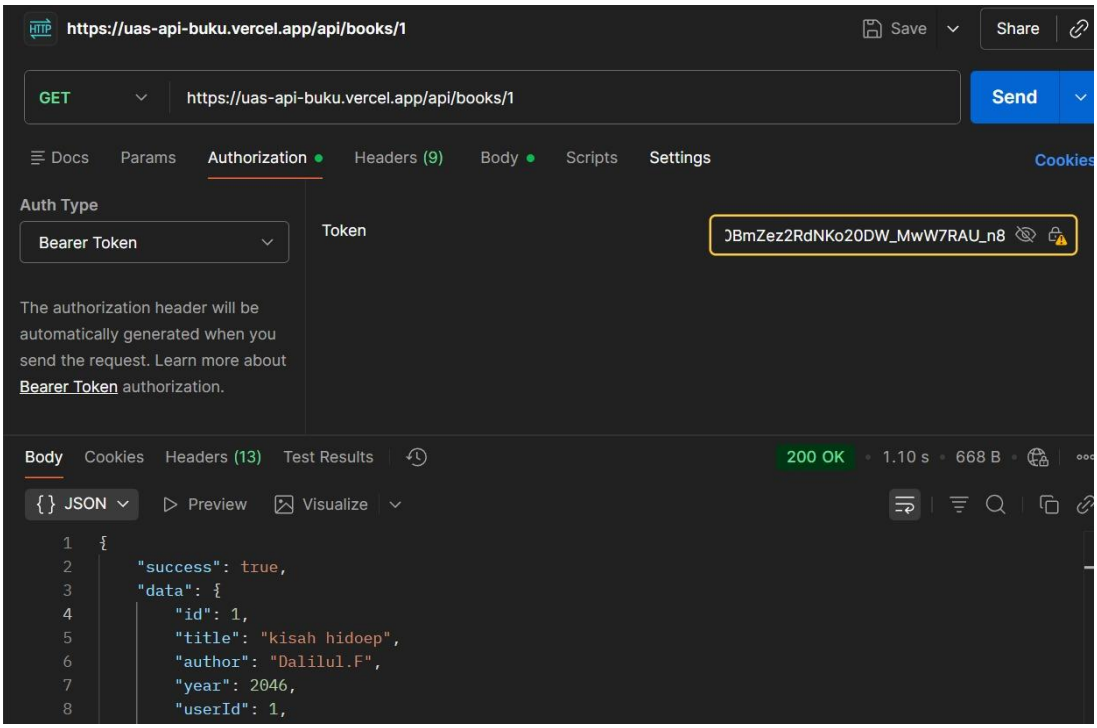
2. Proses Tambah Buku (Gambar Kedua)
Setelah mendapatkan token dari proses login, Anda mencoba menambahkan data buku baru ke dalam database.

- **Endpoint:** POST .../api/books.
- **Request Body:** Anda mengirimkan data buku berupa:
  - title: "Mutiara"
  - author: "Dontol"
  - year: 2026
- **Response (Hasil Input):**
  - **Status 200 OK:** Data berhasil disimpan.
  - **id: 3**: Sistem otomatis memberikan ID unik untuk buku tersebut.
  - **userId: 4**: Ini menunjukkan bahwa buku ini dicatat sebagai milik user dengan ID 4 (user yang sedang login).
  - **createdAt**: Timestamp yang menunjukkan waktu tepat data tersebut dibuat di server.

Secara teknis, Anda telah berhasil melewati alur **Autentikasi** hingga **Operasi Data**.

Gambar kedua bisa berhasil (Status 200 OK) karena Anda kemungkinan besar sudah memasukkan accessToken dari gambar pertama ke dalam tab **Authorization** (Bearer Token) pada Postman.

d. Melihat daftar buku



1. Login untuk Mendapatkan Akses (Gambar 1)
Langkah pertama adalah proses autentikasi.

- **Metode & URL**: POST ke /api/auth/login.
- **Input**: Anda memasukkan kredensial berupa nama, email, dan password.
- **Hasil**: Server memberikan respons **200 OK** yang berisi accessToken. Token ini adalah "kunci" yang diperlukan untuk melakukan operasi lain pada sistem.

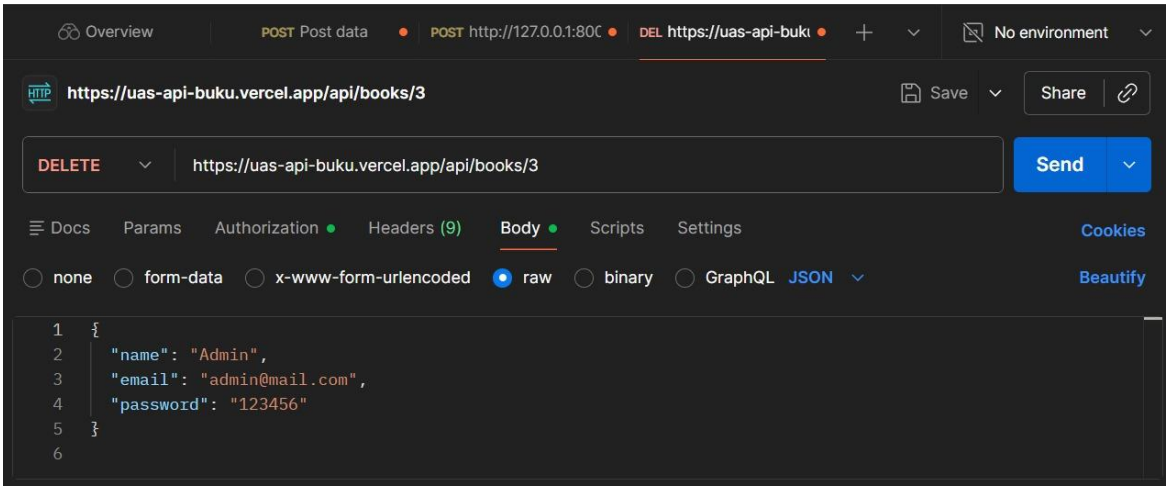
2. Menambahkan Data Buku Baru (Gambar 2)
Setelah memiliki akses, Anda mencoba menambahkan data ke database.

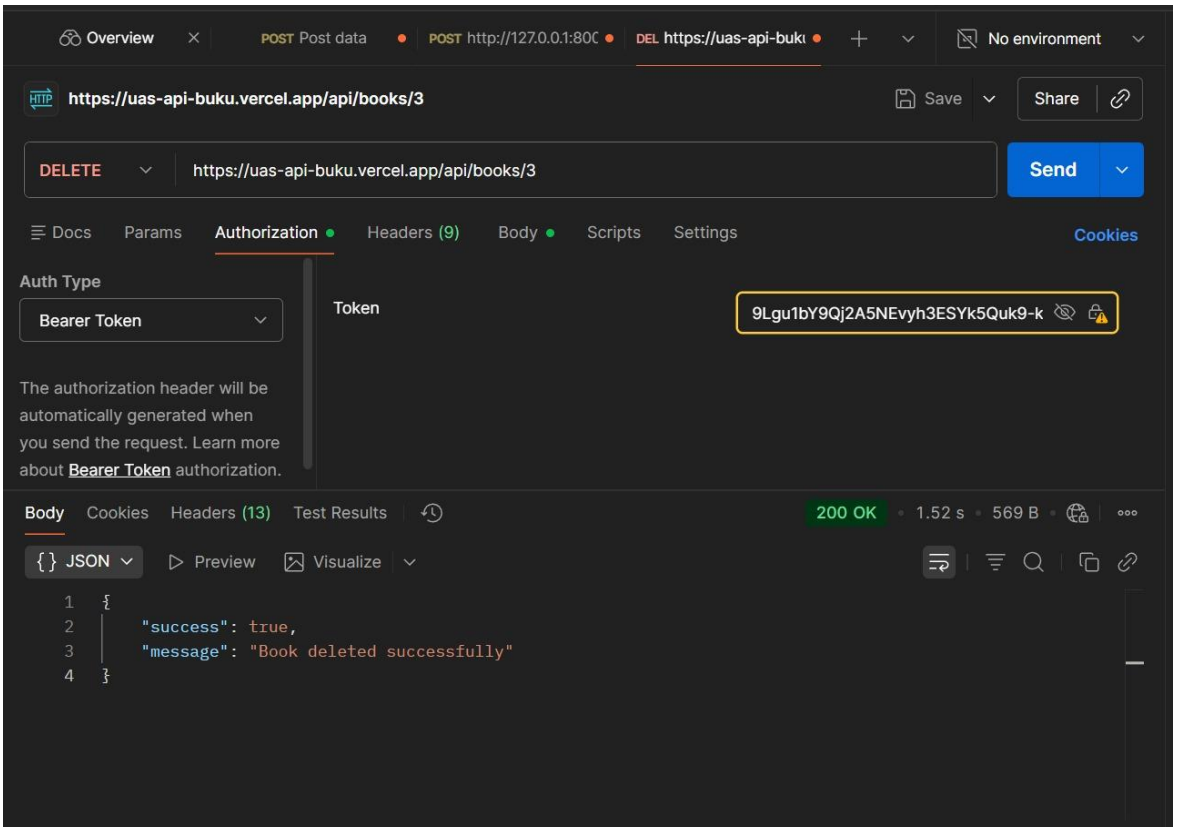
- **Metode & URL**: POST ke /api/books.
- **Input Data**: Anda mengirimkan objek JSON berisi judul buku "Mutiara", penulis "Dontol", dan tahun "2026".
- **Hasil**: Data berhasil disimpan dengan **ID 3** dan terhubung secara otomatis ke **userId 4** (identitas pengguna yang sedang login).

3. Mengambil Data Berdasarkan ID (Gambar 3)
Terakhir, Anda melakukan pengujian untuk mengambil detail data tertentu menggunakan token yang sudah didapat.

- **Metode & URL**: GET ke /api/books/1. Ini berarti Anda secara spesifik meminta data buku yang memiliki **ID 1**.
- **Otorisasi**: Di bagian tab **Authorization**, terlihat Anda menggunakan tipe **Bearer Token**. Anda telah menempelkan accessToken yang didapat dari proses login sebelumnya agar permintaan ini diizinkan oleh server.
- **Hasil**: Server mengembalikan data buku dengan ID 1 berjudul "kisah hidoep" oleh "Dalilul.F".

e. DELETE

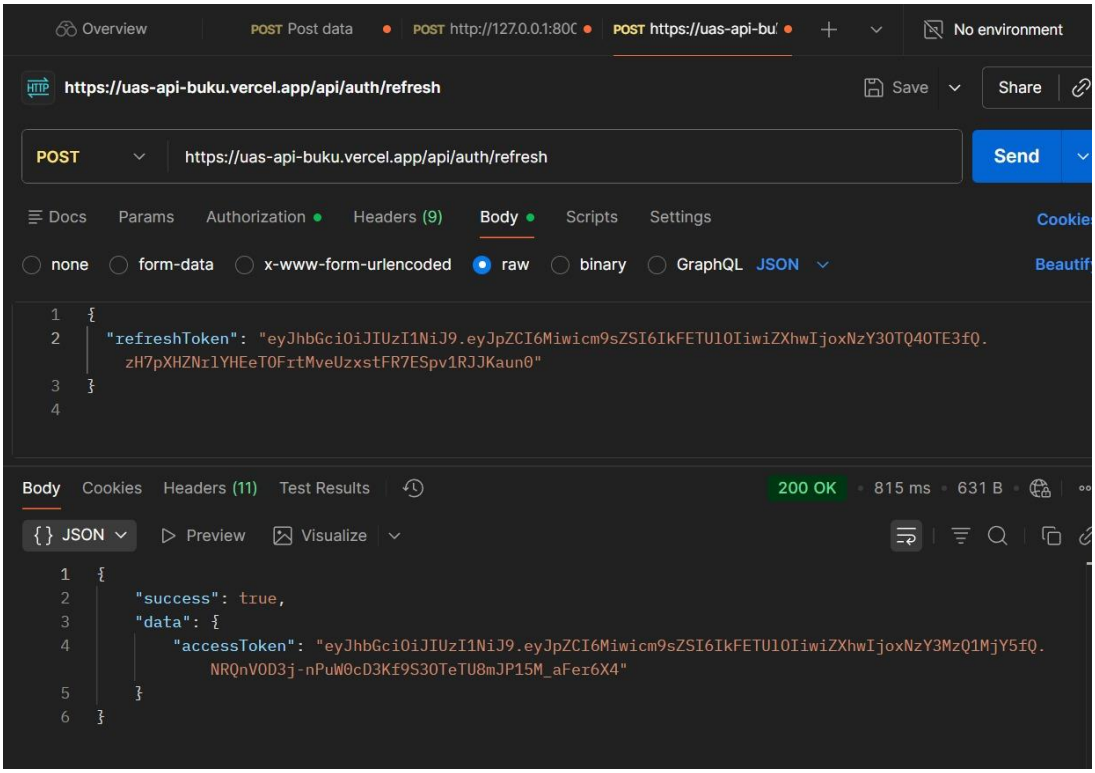




a) **Metode & URL**: DELETE ke /api/books/3.
b) **Proses**:
   - Pengujian dilakukan dengan mencoba menghapus buku yang baru saja dibuat (ID 3).
   - Tab **Authorization** menunjukkan penggunaan **Bearer Token** baru untuk memvalidasi perintah hapus tersebut.

c) **Hasil**: Status **200 OK** dengan pesan "success": true dan "message": "Book deleted successfully". Ini menandakan data buku ID 3 telah resmi dihapus dari database.

f. Refresh



Berdasarkan tampilan Postman tersebut, berikut adalah rincian teknisnya:

- **Metode HTTP:** POST
- **URL API:** https://uas-api-buku.vercel.app/api/auth/refresh
- **Request Body (Input):** Anda mengirimkan sebuah objek JSON yang berisi refreshToken. Token ini biasanya memiliki masa berlaku yang lama dan disimpan secara aman di sisi klien.
- **Response (Output):**
    - **Status Code:** 200 OK, yang berarti permintaan berhasil diproses oleh server.
    - **Data:** Server mengembalikan accessToken yang baru. Token baru inilah yang nantinya akan digunakan untuk mengakses fitur-fitur lain yang memerlukan otorisasi (seperti mengambil data buku atau menambah data).

BAB 3
PENUTUP

Setelah melalui seluruh tahapan pengerjaan, mulai dari perancangan skema database, pengembangan kode program, hingga proses *deployment* dan pengujian, maka dapat disimpulkan bahwa proyek **API Manajemen Buku** ini telah memenuhi target fungsionalitas yang ditetapkan. Beberapa poin utama yang dapat disimpulkan adalah:

1. **Integrasi Teknologi yang Efektif:** Penggunaan *stack* teknologi yang terdiri dari Node.js sebagai *runtime*, Express sebagai *framework*, dan Supabase sebagai database PostgreSQL telah menghasilkan sistem backend yang tangguh dan mudah dikelola. Pemilihan Vercel sebagai platform *hosting* memberikan kemudahan dalam aksesibilitas API secara *real-time* di lingkungan produksi.
2. **Keamanan Berbasis Token:** Penerapan sistem keamanan menggunakan **JSON Web Token (JWT)** telah berhasil memisahkan hak akses antara data publik dan data terproteksi. Mekanisme *Refresh Token* yang diuji melalui Postman membuktikan bahwa sistem dapat menangani siklus hidup sesi pengguna dengan aman, meminimalisir risiko kebocoran data namun tetap menjaga kenyamanan pengalaman pengguna (*user experience*).
3. **Standarisasi RESTful API:** Arsitektur API yang dibangun telah mengikuti standar RESTful dengan penggunaan metode HTTP (GET, POST, PUT, DELETE) yang tepat serta format respons JSON yang konsisten. Hal ini memudahkan integrasi di masa depan apabila aplikasi ini akan dikembangkan ke sisi *client-side* baik berbasis Web maupun Mobile.
4. **Validitas Fungsional:** Berdasarkan hasil uji coba menyeluruh, seluruh *endpoint* mampu menangani permintaan sesuai logika bisnis yang diinginkan, termasuk validasi input dan penanganan error (*error handling*), sehingga sistem minim akan *crash* saat menerima data yang tidak sesuai.