LAPORAN UAS PEMROGRAMAN API

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

Penyusun:

Raya Yuni Setiawan (23104410018)

Ahmad Fathin Naufal H. (23104410032)

Pracitra Ade Saputra (23104410048)

Syahrul Evan Nur Rohman (23104410040)

Mochamad Ichlasul Amal (23104410045)

M. Regi Fabian Nashfi (23104410037)

PROGRAM STUDI TEKNIK INFORMATIKA FAKULTAS

TEKNOLOGI INFORMASI UNIVERSITAS ISLAM BALITAR

2026

KATA PENGANTAR

Dengan rasa syukur yang mendalam, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala berkah dan bimbingan-Nya sehingga laporan ini berhasil dirampungkan. Laporan yang ada di hadapan pembaca ini dibuat dalam rangka memenuhi persyaratan tugas perkuliahan, di mana mahasiswa dituntut untuk merancang dan menerapkan REST API dengan memanfaatkan teknologi Next.js, database PostgreSQL, ORM Prisma, serta menerapkan mekanisme keamanan melalui autentikasi dan otorisasi.

Tujuan dari penyusunan laporan ini adalah untuk memaparkan dasar-dasar teori, proses perancangan, hingga penerapan praktis REST API dalam konteks Sistem Inventori Produk yang meliputi pengelolaan informasi pengguna dan produk. Harapannya, laporan ini dapat menjadi panduan yang berguna untuk memahami bagaimana menerapkan sistem autentikasi, otorisasi, dan pengamanan API dalam membangun aplikasi backend.

Penulis memahami bahwa masih terdapat kekurangan dalam laporan ini. Untuk itu, penulis sangat terbuka dan mengharapkan masukan serta saran konstruktif dari pembaca guna penyempurnaan ke depannya. Semoga laporan ini memberi kontribusi positif bagi para pembaca dan dapat dijadikan acuan dalam mengembangkan sistem yang sejenis.

BAB I

PENDAHULLUAN

1.1 Latar Belakang

Kemajuan pesat di bidang teknologi informasi membawa konsekuensi meningkatnya kebutuhan terhadap sistem aplikasi yang tidak hanya aman dan terstruktur, tetapi juga mudah untuk dikembangkan lebih lanjut. Dalam ekosistem aplikasi modern, backend API memiliki peran vital sebagai pengatur aliran data dan jembatan komunikasi antara sisi client dengan database.

REST API telah menjadi pilihan populer di kalangan developer karena karakteristiknya yang fleksibel, efisien dari segi performa, dan kemampuannya berintegrasi dengan mudah ke berbagai platform. Namun, dalam proses pengembangan REST API, dimensi keamanan tidak boleh diabaikan. Aspek-aspek seperti autentikasi pengguna, otorisasi akses, dan manajemen hak akses menjadi fundamental untuk melindungi sistem dari ancaman penyalahgunaan dan kebocoran data.

Bertolak dari pemahaman tersebut, dalam tugas ini dikembangkan sebuah REST API dengan memanfaatkan teknologi Next.js sebagai framework, PostgreSQL sebagai sistem manajemen database, dan Prisma ORM untuk interaksi database. Implementasi keamanan dilakukan melalui penerapan password hashing menggunakan bcrypt, penggunaan JWT Bearer Token untuk autentikasi, middleware untuk verifikasi akses, serta sistem otorisasi berbasis peran (role-based authorization) dalam konteks Sistem Inventori Produk.

BAB II

PEMBAHASAN

2.1 Code Login/Route

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import bcrypt from "bcryptjs";
import { SignJWT } from "jose"; // Import dari jose
import { success } from "zod";

export async function POST(request) {
  try {
    const { email, password } = await request.json();

    // 1. Cari User
    const user = await prisma.user.findUnique({ where: { email } });
    if (!user) {
      return NextResponse.json({
        success: false,
        message: "User Tidak Sah",
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

    // 2. Cek Password
    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {
      return NextResponse.json({
        success: false,
        message: "Email atau Password Salah",
        code: 401
      }, { status: 401 });
    }

    // 3. Buat Token (Payload: id, email, role)
    const secret = new TextEncoder().encode(process.env.JWT_SECRET);
    const token = await new SignJWT({
      id: user.id,
      email: user.email,
      role: user.role
    })
      .setProtectedHeader({ alg: "HS256" })
      .setExpirationTime("2h") // Token berlaku 2 jam
      .sign(secret);

    // 4. Kirim Token
    return NextResponse.json({
      success: true,
        message: "Login Berhasil",
        token: token
    });

  } catch (error) {
    console.error("Error Login: ", error);
    return NextResponse.json({ message: "Error" }, { status: 500 });
  }
}
```

Inisialisasi dan Pengambilan Data

Pada bagian awal, fungsi POST menerima sebuah objek request yang dikirimkan oleh pengguna melalui metode POST. Kode kemudian mengekstrak data email dan password dari badan permintaan (body) menggunakan fungsi await request.json(). Proses ini berada di dalam blok try...catch untuk memastikan bahwa jika terjadi kesalahan sistem yang tidak terduga, server tidak akan langsung berhenti (crash), melainkan akan mengirimkan respons error 500 sebagai pemberitahuan kepada klien.

Verifikasi Identitas Pengguna

Setelah data diterima, langkah pertama yang dilakukan adalah pencarian pengguna di dalam database menggunakan Prisma ORM. Sistem mencari satu baris data unik yang sesuai dengan email yang

diinputkan. Jika email tersebut tidak ditemukan dalam database, sistem akan langsung menghentikan proses dan mengembalikan respons "Unauthorized" dengan status kode 401. Hal ini memastikan bahwa hanya pengguna terdaftar yang bisa melanjutkan ke tahap berikutnya.

Validasi Keamanan Password

Jika pengguna ditemukan, sistem akan melanjutkan ke tahap pengecekan password. Karena password di database biasanya disimpan dalam bentuk terenkripsi (hash), kode menggunakan library bcryptjs untuk membandingkan password yang diinputkan oleh user dengan password yang ada di database. Jika hasilnya tidak cocok (password salah), sistem kembali mengirimkan respons error 401 dengan pesan bahwa email atau password salah, demi menjaga keamanan agar penyerang tidak tahu secara spesifik mana data yang salah.

Pembuatan Token JWT (Jose)

Apabila password terbukti valid, sistem beralih ke pembuatan JSON Web Token (JWT) menggunakan library jose. Token ini berfungsi sebagai "tiket masuk" digital bagi pengguna untuk mengakses fitur-fitur yang membutuhkan login. Di dalam token tersebut, dimasukkan informasi penting (payload) seperti ID pengguna, email, dan peran (role). Token ini ditandatangani menggunakan algoritma HS256 dengan kunci rahasia (JWT_SECRET) dan diatur agar kedaluwarsa dalam waktu 2 jam guna meminimalisir risiko penyalahgunaan jika token tercuri.

Pengiriman Respons Berhasil

Tahap terakhir adalah mengirimkan respons balik ke pengguna. Jika semua proses di atas berhasil dilewati, server akan mengirimkan objek JSON dengan status success: true dan pesan "Login Berhasil". Di dalam respons tersebut, token JWT yang telah dibuat sebelumnya akan disertakan. Token inilah yang nantinya akan disimpan oleh browser (biasanya di Cookies atau Local Storage) untuk digunakan sebagai alat autentikasi pada permintaan-permintaan berikutnya ke server.

2.2 Code Middleware

```
// middleware.js
import { NextResponse } from "next/server";
import { jwtVerify } from "jose";
import { success } from "zod";

export async function middleware(request) {
  const { pathname } = request.nextUrl;

  // Bypass middleware jika dinonaktifkan (untuk development)
  if (process.env.DISABLE_AUTH_MIDDLEWARE === "true") {
    console.log("Middleware dinonaktifkan sementara");
    return NextResponse.next();
  }

  // 1. Route /api/auth/* adalah PUBLIC (tidak perlu token)
  if (pathname.startsWith("/api/auth")) {
    return NextResponse.next();
  }

  // 2. Ambil token dari Header
  const authHeader = request.headers.get("Authorization");

  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return NextResponse.json(
      {
        success: false,
        message: "Token Tidak Ditemukan",
      code: 401
      },
      { status: 401 }
    );
  }

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

  try {
    // 3. Verifikasi Token
    const secret = new TextEncoder().encode(process.env.JWT_SECRET);
    const { payload } = await jwtVerify(token, secret);

    // 4. Authorization berdasarkan route dan role

    // Route /api/users/* - Hanya Admin
    if (pathname.startsWith("/api/users")) {
      if (payload.role !== "Admin") {
        return NextResponse.json(
          { success: false,
            message: "Hanya Admin Yang Dapat Mengakses Route Ini",
          code: 403 },
          { status: 403 }
        );
      }
    }

    // Route /api/products/* - User dengan token valid (Admin atau User)
    if (pathname.startsWith("/api/products")) {
      // if (payload.role !== "Admin") {
      //   return NextResponse.json(
      //     { success: false,
      //       message: "Hanya Admin Yang Dapat Mengakses Route Ini",
      //     code: 403 },
      //     { status: 403 }
      //   );
      // }
    }

    // 5. Jika semua validasi sukses, lanjutkan request
    return NextResponse.next();

  } catch (error) {
    // 6. Jika token salah/expired
    console.error("Token verification failed:", error);
    return NextResponse.json(
      {
        success: false,
        message: "Token Tidak Valid atau Kadaluwarsa",
        code: 401

      },
      { status: 401 }
    );
  }
}

export const config = {
  // Matcher untuk semua route yang perlu dicek
  matcher: [
    "/api/auth/:path*",      // Public route
    "/api/users/:path*",     // Protected - Admin only
    "/api/products/:path*"   // Protected - Token valid (Admin/User)
  ],
};
```

Inisialisasi dan Pengecualian Rute

Fungsi middleware dimulai dengan memeriksa konfigurasi lingkungan; jika variabel DISABLE_AUTH_MIDDLEWARE bernilai "true", sistem akan melewati semua pengecekan keamanan, yang biasanya digunakan untuk memudahkan proses pengembangan. Selanjutnya, kode melakukan pengecekan terhadap jalur URL (pathname). Rute yang dimulai dengan /api/auth/ dikategorikan sebagai rute publik, sehingga permintaan diizinkan lewat tanpa perlu membawa token autentikasi.

Ekstraksi dan Verifikasi Token

Untuk rute yang dilindungi, middleware mencari token di dalam header permintaan bernama Authorization. Jika header tidak ditemukan atau tidak diawali dengan format "Bearer ", sistem akan langsung menolak akses dengan mengirimkan respons error 401 (Token Tidak Ditemukan). Jika formatnya benar, sistem akan mengambil string token tersebut dan memverifikasinya menggunakan library jose dan kunci rahasia JWT_SECRET untuk memastikan bahwa token tersebut sah dan belum kedaluwarsa.

Validasi Role dan Otorisasi

Setelah token dinyatakan valid, middleware memeriksa informasi peran (role) yang tersimpan di dalam isi token (payload). Sistem menerapkan aturan akses yang berbeda berdasarkan rute: pada rute /api/users/, akses hanya diberikan jika pengguna memiliki peran "Admin", sedangkan permintaan dari peran lain akan ditolak dengan status 403 (Forbidden). Sementara itu, rute seperti /api/products/ dirancang untuk bisa diakses oleh pengguna dengan token valid, baik itu Admin maupun pengguna biasa.

Penanganan Kesalahan dan Akhir Proses

Jika seluruh validasi terpenuhi, sistem menjalankan perintah NextResponse.next() untuk meneruskan permintaan ke tujuan akhirnya. Namun, jika selama proses verifikasi terjadi kegagalan—seperti token yang dimanipulasi atau sudah melewati batas waktu aktif—blok catch akan menangkap kesalahan tersebut dan mengirimkan respons error 401 yang menyatakan bahwa token tidak valid atau sudah kedaluwarsa. Seluruh mekanisme ini dipandu oleh objek config di bagian akhir kode yang menentukan daftar rute mana saja yang harus dipantau oleh middleware ini.

2.3 Code products/[id]/route

Mekanisme Keamanan dan Filter Akses (Middleware)

Sistem ini menggunakan Middleware sebagai lapisan perlindungan pertama untuk menyaring setiap permintaan yang masuk ke API. Secara teknis, middleware ini akan memeriksa apakah sebuah permintaan memerlukan token autentikasi atau tidak; misalnya, rute publik seperti /api/auth/ diizinkan lewat tanpa pemeriksaan. Untuk rute yang dilindungi, middleware mengambil token JWT dari header Authorization dan memverifikasinya menggunakan library jose. Selain memvalidasi keaslian token, middleware juga melakukan otorisasi berbasis peran (role-based access control), di mana rute tertentu seperti /api/users/ secara ketat hanya bisa diakses oleh pengguna dengan peran "Admin". Jika token tidak ditemukan, tidak valid, atau peran pengguna tidak sesuai, middleware akan langsung menolak

permintaan dengan mengirimkan status error 401 atau 403 sebelum permintaan tersebut sampai ke pemrosesan data.

Operasi Data dan Manajemen Produk (API Routes)

Setelah melewati lapisan keamanan middleware, permintaan akan ditangani oleh fungsi API khusus untuk mengelola data produk menggunakan Prisma ORM. Dalam modul ini, terdapat fungsi GET untuk mengambil detail produk spesifik berdasarkan ID, serta fungsi PUT dan PATCH yang memungkinkan pembaruan data produk termasuk pengaturan relasi kategori, apakah kategori tersebut akan diputuskan (disconnect) atau dihubungkan ke kategori baru (connect). Khusus untuk operasi yang bersifat destruktif seperti fungsi DELETE, sistem menerapkan lapisan keamanan tambahan di tingkat fungsi dengan memanggil helper checkAdminRole. Hal ini memastikan bahwa meskipun seseorang berhasil menembus middleware, penghapusan data secara permanen dari database hanya dapat dilakukan jika pengguna terbukti memiliki otoritas Admin yang sah.

2.4 Code products/route

```
import { prisma } from "@/lib/prisma";
import { NextResponse } from "next/server";
import { success } from "zod";

// !dengan include
export async function GET(){
    try{
        const products = await prisma.product.findMany({
            include: {
                category: true //mengambil semua kolom yang ada di table category
            }
        });
        return NextResponse.json(products, {status: 200}); // ← Diubah dari 201 ke 200
    }catch(error){
        console.error("Server error", error);
        return NextResponse.json({error: "Internal Server Error"}, {status:500});
    }
}

//!dengan select
// export async function GET(){
//     try{
//         const products = await prisma.product.findMany({
//             select: {
//                 name: true,
//                 price: true,
//                 category: {
//                     select : {
//                         name : true,
//                     }
//                 }
//             }
//         });
//         return NextResponse.json(products, {status: 200}); // ← Diubah dari 201 ke 200
//     }catch(error){
//         console.error("Server error", error);
//         return NextResponse.json({error: "Internal Server Error"}, {status:500});
//     }
// }

// !dengan reverse query
// export async function GET(){
//     try{
//         const kategori = await prisma.category.findMany({
//             where: {name: "Gaming Gear"},
//             include: {
//                 product: true //menampilkan array dari tabel product
//             }
//         });
//         return NextResponse.json(kategori, {status: 200}); // ← Diubah dari 201 ke 200
//     }catch(error){
//         console.error("Server error", error);
//         return NextResponse.json({error: "Internal Server Error"}, {status:500});
//     }
// }
```

Sistem Keamanan dan Otorisasi (Middleware)

Lapisan pertama adalah Middleware yang berfungsi sebagai penjaga gerbang otomatis untuk setiap permintaan masuk. Sistem ini secara cerdas membedakan rute publik seperti login/registrasi yang diizinkan lewat tanpa syarat, dengan rute terproteksi yang mewajibkan adanya token JWT valid di header Authorization. Selain memverifikasi keaslian token menggunakan library jose, middleware ini juga menerapkan Otorisasi Berbasis Peran (Role-Based Access Control); contohnya, akses ke rute /api/users/ akan ditolak dengan status 403 jika pengguna tidak memiliki peran "Admin". Hal ini memastikan bahwa meskipun pengguna memiliki akun, mereka tidak bisa sembarangan mengakses data sensitif milik administrator.

Manajemen Data Produk (GET dan POST)

Setelah melewati filter keamanan, permintaan akan ditangani oleh API Route yang berinteraksi langsung dengan database melalui Prisma ORM. Fungsi GET dirancang untuk mengambil seluruh daftar produk dengan menyertakan detail kategori terkait secara otomatis melalui fitur include. Sementara itu, fungsi POST digunakan untuk menambahkan produk baru ke sistem. Dalam proses pembuatan produk ini, kode memastikan data seperti nama, harga, dan stok tersimpan dengan benar, serta menghubungkan produk tersebut ke kategori yang sudah ada melalui metode connect pada field categoryId.

Operasi Spesifik dan Perlindungan Ganda (PUT dan DELETE)

Untuk operasi yang lebih spesifik, sistem menyediakan fungsi GET (by ID) untuk melihat detail satu produk, serta fungsi PUT atau PATCH untuk memperbarui data produk yang sudah ada. Dalam proses pembaruan, sistem memiliki logika fleksibel yang dapat memutus hubungan kategori lama (disconnect)

atau menyambungkan ke kategori baru (connect) sesuai kebutuhan data yang dikirimkan. Yang paling krusial adalah fungsi DELETE; meskipun middleware sudah melakukan pengecekan awal, fungsi ini menerapkan lapisan keamanan kedua dengan memanggil helper checkAdminRole secara internal. Ini menjamin bahwa penghapusan data permanen benar-benar hanya bisa dieksekusi oleh Admin, memberikan perlindungan berlapis terhadap integritas data aplikasi Anda.

2.5 Code Register/route

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import bcrypt from "bcryptjs";
import { success } from "zod";

export async function POST(request) {
  try {
    const { name, email, password, role } = await request.json();

    // 1. Hash Password
    // Angka 10 adalah salt rounds (standar keamanan saat ini)
    const hashedPassword = await bcrypt.hash(password, 10);

    // 2. Simpan ke DB
    const newUser = await prisma.user.create({
      data: {
        name,
        email,
        password: hashedPassword, // Simpan yang sudah di-hash!
        role: role || "User", // Default ke "User" jika tidak dikirim
      },
    });

    return NextResponse.json({
      success: true,
      message: "User Berhasil Didaftarkan",
      user: {
        id: newUser.id,
        name: newUser.name,
        email: newUser.email,
        role: newUser.role
      }
    }, { status: 201 });
  } catch (error) {
    console.error("Error Register User: ", error);

    // Handle unique constraint error (email sudah terdaftar)
    if (error.code === 'P2002') {
      return NextResponse.json({
        success: false,
        message: "Email Sudah Terdaftar",
        code: 400
      }, { status: 400 });
    }

    return NextResponse.json({ message: "Error creating user" }, { status: 500 });
  }
}
```

Registrasi dan Keamanan Pengguna

Proses dimulai dari pendaftaran pengguna baru melalui fungsi POST pada route registrasi. Sistem menerima data seperti nama, email, dan password, lalu menggunakan library bcryptjs untuk mengubah password menjadi hash (enkripsi) demi keamanan data di database. Jika pendaftaran berhasil, data disimpan menggunakan Prisma ke tabel user; namun jika email sudah terdaftar, sistem akan menangkap error unik (P2002) dan memberikan informasi bahwa email tersebut sudah digunakan.

Gerbang Keamanan Otomatis (Middleware)

Setelah pengguna memiliki akun, setiap permintaan akses ke API akan dipantau oleh Middleware yang bertindak sebagai filter keamanan global. Middleware ini secara otomatis memeriksa keberadaan token JWT di dalam header permintaan. Jika token tidak ada atau tidak valid, akses akan langsung ditolak sebelum sempat mencapai database. Selain itu, middleware ini menerapkan Otorisasi Berbasis Peran (Role-Based Access Control), di mana akses ke rute tertentu (seperti manajemen user) dikunci khusus hanya untuk pengguna dengan peran "Admin".

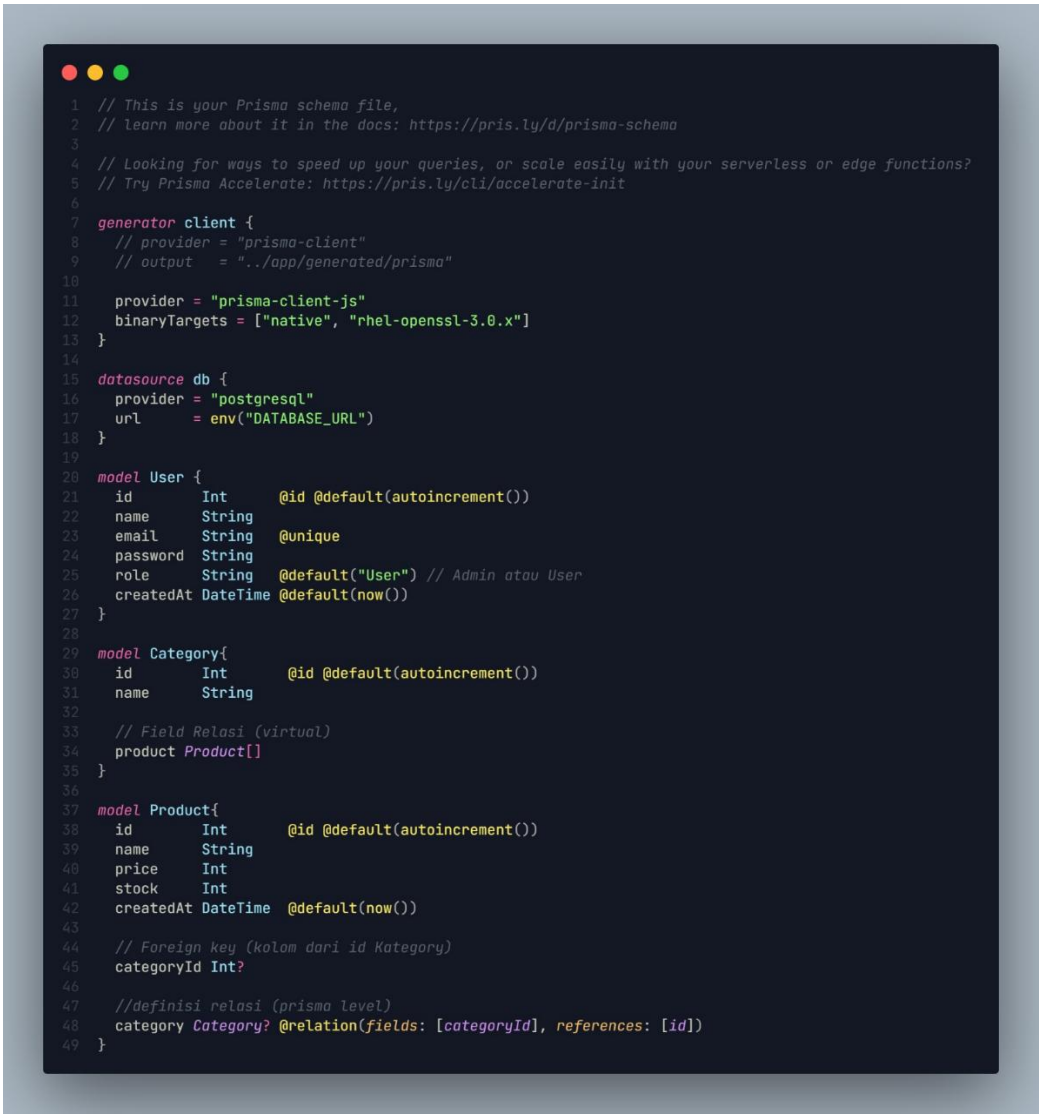
Manajemen Inventori Produk (CRUD)

Setelah melewati filter keamanan, aplikasi menyediakan serangkaian fungsi untuk mengelola data produk menggunakan Prisma ORM. Terdapat fungsi GET untuk menarik semua daftar produk sekaligus detail kategorinya, serta fungsi POST untuk membuat produk baru yang langsung dihubungkan ke kategori tertentu melalui ID. Untuk pembaruan data, fungsi PUT atau PATCH memungkinkan perubahan fleksibel, termasuk kemampuan untuk memutuskan atau menyambungkan kembali relasi produk dengan kategori.

Perlindungan Data Destruktif

Sebagai langkah keamanan terakhir, sistem menerapkan perlindungan berlapis pada operasi yang berisiko tinggi. Pada fungsi DELETE produk, kode tidak hanya mengandalkan pengecekan di middleware, tetapi juga memanggil fungsi pembantu checkAdminRole di dalam fungsi tersebut untuk memverifikasi ulang identitas penghapus. Hal ini menjamin bahwa penghapusan data permanen dari database benar-benar hanya dapat dilakukan oleh administrator yang sah, menjaga integritas dan keamanan data seluruh aplikasi.

2.6 Code Schema.prisma



```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {
  // provider = "prisma-client"
  // output   = "../app/generated/prisma"

  provider = "prisma-client-js"
  binaryTargets = ["native", "rhel-openssl-3.0.x"]
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id        Int      @id @default(autoincrement())
  name      String
  email     String   @unique
  password  String
  role      String   @default("User") // Admin atau User
  createdAt DateTime @default(now())
}

model Category{
  id        Int       @id @default(autoincrement())
  name      String

  // Field Relasi (virtual)
  product Product[]
}

model Product{
  id        Int       @id @default(autoincrement())
  name      String
  price     Int
  stock     Int
  createdAt DateTime  @default(now())

  // Foreign key (kolom dari id Kategory)
  categoryId Int?

  //definisi relasi (prisma level)
  category Category? @relation(fields: [categoryId], references: [id])
}
```

Struktur Basis Data dan Registrasi Pengguna

Dasar dari aplikasi ini adalah skema Prisma yang mendefinisikan tiga model utama: User, Category, dan Product, di mana satu kategori dapat memiliki banyak produk melalui relasi satu-ke-banyak (one-to-many). Pada proses pendaftaran pengguna, sistem menerima data seperti nama, email, dan password, lalu mengamankan password tersebut menggunakan metode hashing melalui library bcryptjs sebelum disimpan ke database PostgreSQL. Jika pendaftaran gagal karena email sudah digunakan, sistem secara cerdas menangkap error kode unik Prisma P2002 untuk memberikan pesan kesalahan yang spesifik kepada pengguna.

Filter Keamanan dan Otorisasi (Middleware)

Setelah pengguna terdaftar, setiap permintaan akses ke rute API diproteksi oleh Middleware sebagai lapisan pertahanan utama. Middleware ini secara otomatis membedakan rute publik seperti login dan registrasi dari rute terproteksi yang mewajibkan adanya token JWT valid di header Authorization. Selain memverifikasi keaslian token, sistem menjalankan Otorisasi Berbasis Peran (Role-Based Access Control); contohnya, akses ke rute manajemen pengguna (/api/users/) akan ditolak jika peran yang tersemat di dalam token bukan "Admin". Keamanan ini bersifat fleksibel karena dapat dinonaktifkan sementara untuk kebutuhan pengembangan melalui variabel lingkungan (environment variable).

Manajemen Data Produk dan Perlindungan Berlapis

Pada sisi operasional data, aplikasi menyediakan fitur lengkap untuk mengelola inventori produk menggunakan Prisma ORM. Terdapat fungsi GET untuk menarik seluruh daftar produk beserta detail kategorinya, serta fungsi POST untuk membuat produk baru yang langsung dihubungkan ke kategori tertentu. Untuk pembaruan data, fungsi PUT memungkinkan modifikasi harga, stok, hingga pengubahan relasi kategori produk. Sebagai perlindungan tambahan pada aksi yang bersifat destruktif, fungsi DELETE menerapkan keamanan lapis kedua dengan memanggil kembali helper checkAdminRole secara internal. Hal ini menjamin bahwa penghapusan data secara permanen hanya benar-benar dapat dieksekusi oleh administrator, memberikan tingkat integritas data yang sangat tinggi.

2.7 Code Users/Route

```
import { prisma } from "@/lib/prisma";
import { NextResponse } from "next/server";
import {z} from "zod";

//!bikin schema untuk zod
const UserSchema = z.object({
    name: z.string().min(3, {message: "Nama minimal harus 3 karakter"}),
    email: z.string().email({message: "Format email tidak vailid"}),
    password: z.string().min(8, {message: "Password minimal 8 karakter"}),
});

// GET: Ambil semua data user
export async function GET() {
  try{
      const users = await prisma.user.findMany();
      return NextResponse.json(users);
  }catch(error){
    console.error("Error get Data: ", error);
    NextResponse.json({message: "Server Error"}, {status: 500});
  }
}

//! POST: Tambah user baru tanpa try-catch dan ZOD
// export async function POST(request) {
//   const data = await request.json();
//   const userBaru = await prisma.user.create({
//     data: {
//       name: data.name,
//       email: data.email,
//       password: data.password,
//     },
//   });
```

```
//   return NextResponse.json(userBaru);
// }

//!POST dengan try-catch dilengkapi ZOD
export async function POST(request){
  try{
    const body = await request.json();
    const validation = UserSchema.safeParse(body);
    if(!validation.success){
      return NextResponse.json({
        message: "Input Tidak Valid",
        error: validation.error.flatten().fieldErrors
      },
      {status: 400}
    );
    }

    const newUser = await prisma.user.create({
      data: body
    });
    return NextResponse.json(newUser, {status: 200});

  }catch(error){
    console.error("Error POST User: ", error);
    NextResponse.json({message: "Server Error"}, {status: 500});
  }
}
```

Pemodelan Data dan Dasar Sistem

Sistem dibangun di atas fondasi Prisma Schema yang mendefinisikan hubungan antar data di database PostgreSQL, mencakup model User untuk autentikasi, serta model Category dan Product yang saling terhubung dengan relasi satu-ke-banyak. Pada proses pendaftaran pengguna, aplikasi menerapkan protokol keamanan tinggi dengan melakukan hashing password menggunakan library bcryptjs sebelum data disimpan. Selain itu, terdapat mekanisme validasi skema menggunakan library Zod yang bertugas memeriksa kualitas input pengguna, seperti memastikan email memiliki format yang valid dan password memenuhi panjang minimal, guna mencegah masuknya data sampah atau serangan injeksi ke dalam sistem.

Lapisan Keamanan dan Otorisasi (Middleware)

Setelah pengguna terdaftar, setiap interaksi dengan API dipantau secara ketat oleh Middleware sebagai gerbang pertahanan utama. Middleware ini bertugas mengekstrak dan memverifikasi token JWT dari header permintaan untuk memastikan identitas pengguna sah. Di sinilah Otorisasi Berbasis Peran (Role-Based Access Control) diterapkan secara otomatis; permintaan yang menuju rute sensitif seperti /api/users/ akan langsung diblokir jika profil pengguna di dalam token tidak memiliki peran "Admin". Sistem ini memberikan fleksibilitas bagi pengembang melalui opsi bypass middleware jika diperlukan dalam lingkungan pengembangan lokal.

Operasional Data dan Proteksi Berlapis

Pada tingkat manajemen data produk, aplikasi menyediakan fungsionalitas CRUD lengkap yang dieksekusi melalui Prisma ORM. Fungsi GET dan POST menangani penarikan daftar produk beserta kategorinya dan pembuatan produk baru dengan relasi yang tepat. Untuk operasi perubahan data, fungsi PUT memungkinkan pembaruan informasi produk sekaligus fleksibilitas untuk mengubah atau

memutuskan hubungan kategorinya. Sebagai perlindungan integritas data yang paling krusial, fungsi DELETE menerapkan keamanan lapis kedua dengan memanggil fungsi pembantu checkAdminRole secara internal. Hal ini menjamin bahwa meskipun lapisan middleware terlewati, penghapusan data secara permanen tetap mutlak membutuhkan otoritas Admin, menciptakan sistem yang aman dan andal.

2.8 Code Users/id/route

Pemodelan Data dan Dasar Sistem

Sistem dibangun di atas fondasi Prisma Schema yang mendefinisikan hubungan antar data di database PostgreSQL, mencakup model User untuk autentikasi, serta model Category dan Product yang saling terhubung dengan relasi satu-ke-banyak. Pada proses pendaftaran pengguna, aplikasi menerapkan protokol keamanan tinggi dengan melakukan hashing password menggunakan library bcryptjs sebelum data disimpan. Selain itu, terdapat mekanisme validasi skema menggunakan library Zod yang bertugas memeriksa kualitas input pengguna, seperti memastikan email memiliki format yang valid dan password memenuhi panjang minimal, guna mencegah masuknya data sampah atau serangan injeksi ke dalam sistem.

Lapisan Keamanan dan Otorisasi (Middleware)

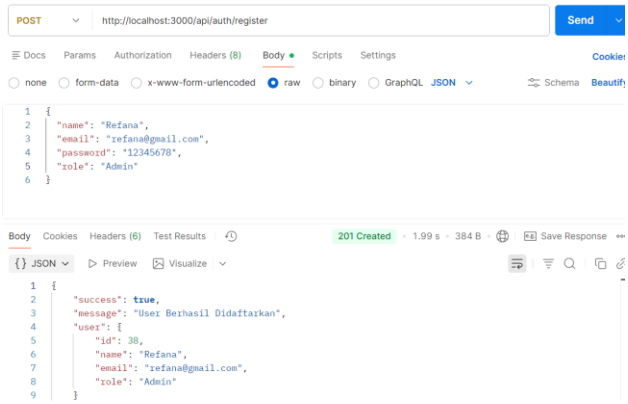
Setelah pengguna terdaftar, setiap interaksi dengan API dipantau secara ketat oleh Middleware sebagai gerbang pertahanan utama. Middleware ini bertugas mengekstrak dan memverifikasi token JWT dari header permintaan untuk memastikan identitas pengguna sah. Di sinilah Otorisasi Berbasis Peran (Role-Based Access Control) diterapkan secara otomatis; permintaan yang menuju rute sensitif seperti /api/users/ akan langsung diblokir jika profil pengguna di dalam token tidak memiliki peran "Admin". Sistem ini memberikan fleksibilitas bagi pengembang melalui opsi bypass middleware jika diperlukan dalam lingkungan pengembangan lokal.

Operasional Data dan Proteksi Berlapis

Pada tingkat manajemen data produk, aplikasi menyediakan fungsionalitas CRUD lengkap yang dieksekusi melalui Prisma ORM. Fungsi GET dan POST menangani penarikan daftar produk beserta kategorinya dan pembuatan produk baru dengan relasi yang tepat. Untuk operasi perubahan data, fungsi PUT memungkinkan pembaruan informasi produk sekaligus fleksibilitas untuk mengubah atau memutuskan hubungan kategorinya. Sebagai perlindungan integritas data yang paling krusial, fungsi DELETE menerapkan keamanan lapis kedua dengan memanggil fungsi pembantu checkAdminRole secara internal. Hal ini menjamin bahwa meskipun lapisan middleware terlewati, penghapusan data secara permanen tetap mutlak membutuhkan otoritas Admin, menciptakan sistem yang aman dan andal.

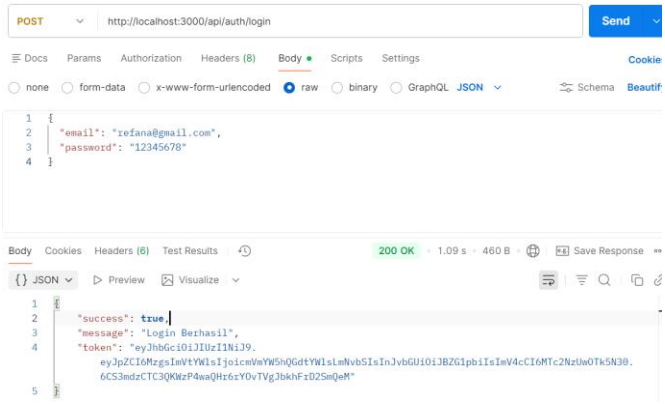
2.9 TESTING POSTMAN

1. Admin Register



Hasil eksekusi API di atas memperlihatkan proses pengiriman data yang sukses menggunakan metode **POST** ke *endpoint* registrasi pada *server* lokal. Permintaan ini mengirimkan muatan data dalam format JSON yang berisi informasi pengguna baru, yakni nama "Refana", alamat email, kata sandi, dan peran sebagai "Admin". Respon yang diterima dari *server* ditandai dengan kode status **201 Created**, yang mengonfirmasi bahwa data pengguna baru telah berhasil dibuat dan disimpan ke dalam *database*. Selain itu, sistem mengembalikan pesan sukses beserta rincian objek pengguna yang baru didaftarkan, termasuk ID unik yang dihasilkan sistem, namun secara otomatis menyembunyikan *field* kata sandi sebagai langkah keamanan standar.

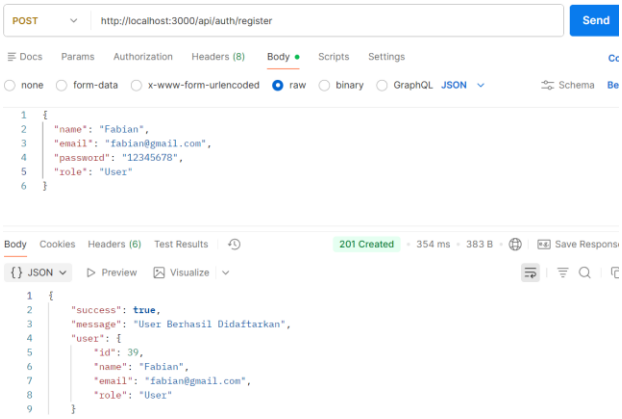
2. Admin Login



Respons API yang ditampilkan mengonfirmasi keberhasilan proses otentikasi pengguna pada sistem. Melalui metode **POST**, kredensial berupa email dan kata sandi dikirimkan ke server untuk diverifikasi. Status **200 OK** yang diterima menandakan bahwa data tersebut valid dan akses diberikan. Poin terpenting dalam balasan JSON ini adalah adanya **token** otentikasi yang dihasilkan. Kode token yang panjang tersebut bertindak sebagai kredensial digital sementara yang harus disimpan oleh sisi klien (*client-side*), karena akan

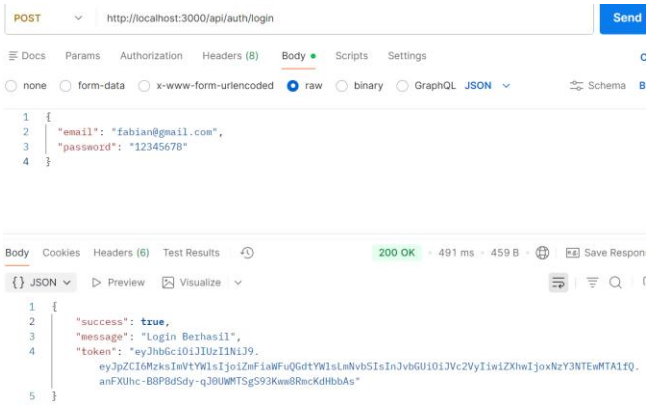
digunakan sebagai izin akses pada permintaan-permintaan selanjutnya untuk membuka fitur atau halaman yang membutuhkan otorisasi khusus (seperti data profil atau halaman admin).

3. User Register



Hasil eksekusi API ini menampilkan proses pembuatan akun untuk pengguna bernama "Fabian" melalui metode **POST** pada *endpoint* registrasi. Permintaan dikirim dengan muatan data JSON yang menetapkan peran (*role*) akun tersebut sebagai "User". Server merespons dengan kode status **201 Created**, yang menandakan bahwa data pengguna baru telah sukses dibuat dan disimpan ke dalam sistem. Dalam badan respon (*body*), server mengembalikan pesan konfirmasi "User Berhasil Didaftarkan" serta objek data pengguna yang baru saja dibuat—lengkap dengan ID unik 39—sebagai bukti bahwa proses registrasi telah selesai dengan sempurna.

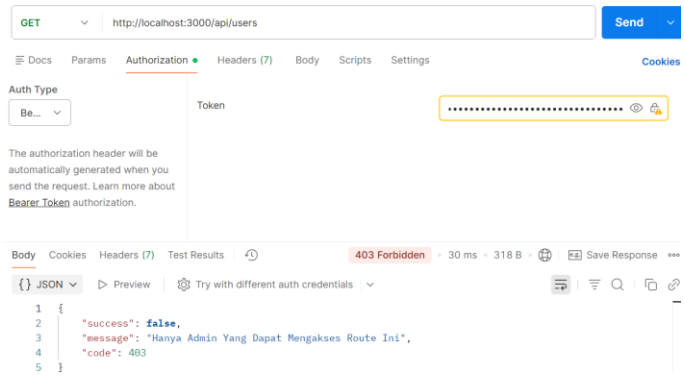
4. User Login



Respons API tersebut menunjukkan keberhasilan proses otentikasi (*login*) untuk akun "Fabian" yang baru saja didaftarkan. Melalui metode **POST** ke *endpoint* login, klien mengirimkan kredensial berupa email "fabian@gmail.com" dan kata sandi. Server memvalidasi data tersebut dan mengembalikan status **200 OK**, menandakan bahwa identitas pengguna benar. Dalam balasan JSON, sistem memberikan konfirmasi "Login

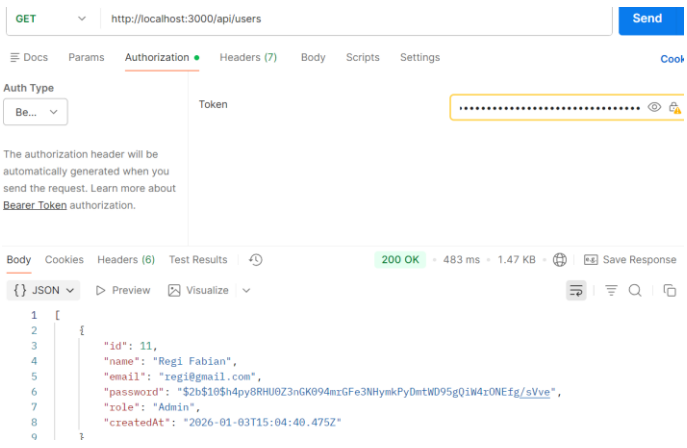
Berhasil" dan menyertakan sebuah **token** otentikasi. Token panjang ini berfungsi sebagai "kunci akses" sementara yang harus disimpan oleh aplikasi klien untuk mengizinkan pengguna mengakses fitur-fitur tertutup di dalam sistem pada permintaan berikutnya.

5. Get “users” dengan token User (hanya bisa dengan admin)



Hasil *output* ini mendemonstrasikan mekanisme keamanan otorisasi atau pembatasan hak akses (*Role-Based Access Control*) pada sistem. Pengguna mencoba mengakses data sensitif di *endpoint* GET /api/users menggunakan **Token Bearer** (kemungkinan besar milik akun "Fabian" yang memiliki peran sebagai "User"). Server menolak permintaan tersebut dengan kode status **403 Forbidden**, bukan 401 (yang berarti token salah/tidak ada). Pesan kesalahan "Hanya Admin Yang Dapat Mengakses Route Ini" menegaskan bahwa token tersebut valid secara identitas, namun akun pemilik token tidak memiliki level izin "Admin" yang diwajibkan untuk melihat daftar seluruh pengguna.

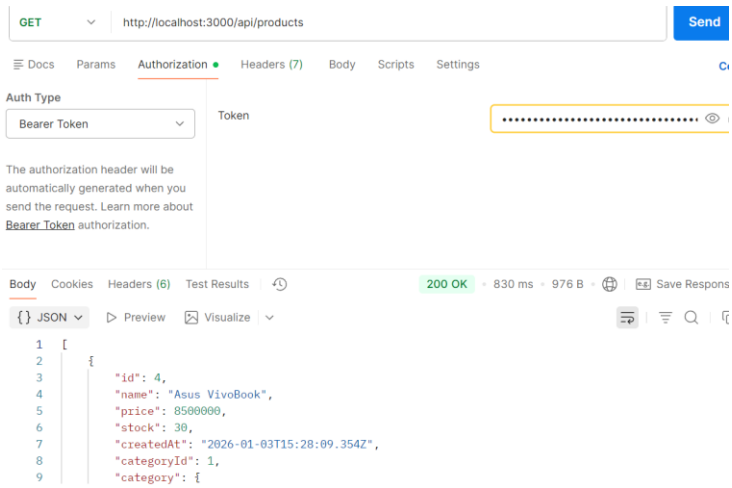
6. Get “users” dengan token Admin



Output ini menunjukkan bahwa permintaan untuk melihat daftar pengguna (GET /api/users) kini **berhasil** dieksekusi, ditandai dengan status kode **200 OK**. Hal ini mengindikasikan bahwa token yang dimasukkan ke dalam *Header Authorization* kali ini adalah token yang valid milik akun dengan peran **"Admin"**.

Karena otorisasi diterima, server membuka akses ke sumber daya yang dilindungi (*protected resource*) dan mengembalikan data dalam format array JSON [...]. Data tersebut berisi daftar lengkap pengguna yang ada di database. Pada potongan gambar, terlihat salah satu pengguna dengan ID 11 ("Regi Fabian"). Perlu dicatat bahwa meskipun data ini terbuka, kolom *password* yang ditampilkan berbentuk teks acak (*hash*), yang berarti sistem telah menerapkan enkripsi agar kata sandi asli tidak terbaca secara langsung.

7. Get “products” dengan token User/Admin



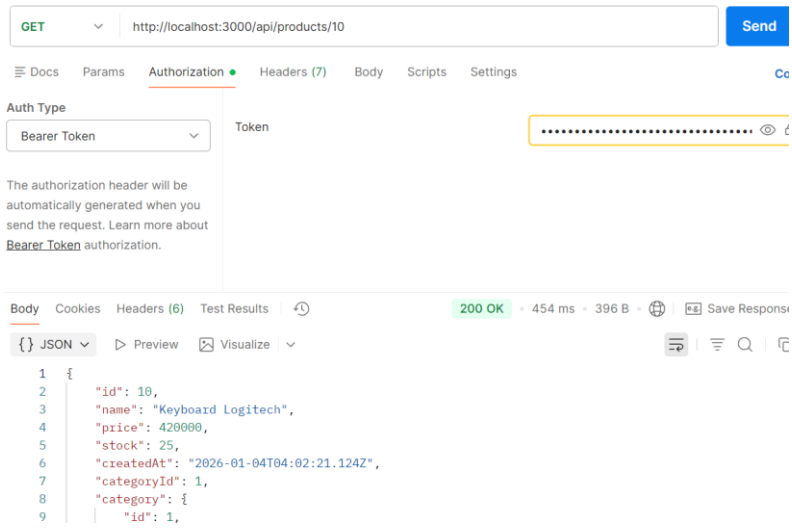
Hasil eksekusi ini menampilkan respon dari permintaan **GET** ke *endpoint* /api/products yang bertujuan untuk mengambil daftar barang yang tersedia. Server memberikan respon sukses dengan kode status **200 OK** dan mengirimkan data dalam format *array* JSON.

Pada potongan data yang terlihat, terdapat produk dengan rincian:

- **Nama:** "Asus VivoBook"
- **Harga:** 8.500.000
- **Stok:** 30 unit

Poin penting dalam struktur respon ini adalah adanya objek **category** yang bersarang (*nested*) di dalam data produk. Ini menunjukkan bahwa *backend* menerapkan teknik *data population* atau *eager loading*. Artinya, saat mengambil data produk, sistem secara otomatis menarik data kategori yang berelasi dengannya (ID kategori 1), sehingga klien mendapatkan informasi lengkap (produk + kategori) hanya dalam satu kali panggilan API.

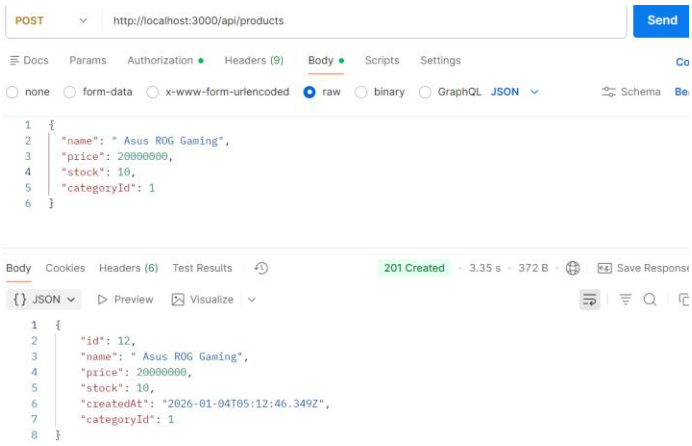
8. Get “products/[id]”



Hasil eksekusi API ini menampilkan respons dari permintaan pengambilan data produk secara spesifik berdasarkan ID-nya. Melalui metode **GET** pada *endpoint* /api/products/10, klien meminta server untuk mencari dan menampilkan detail barang yang memiliki ID unik 10. Server berhasil memproses permintaan tersebut dan mengembalikan status **200 OK**.

Dalam badan respons JSON, ditampilkan informasi rinci produk berupa "Keyboard Logitech" dengan harga 420.000 dan stok tersisa 25 unit. Struktur data ini kembali memperlihatkan objek **category** yang bersarang di dalamnya (terhubung ke categoryId: 1), yang mengonfirmasi bahwa sistem secara konsisten menerapkan teknik relasi (*eager loading*) untuk menyajikan data produk lengkap beserta info kategorinya dalam satu kesatuan respons.

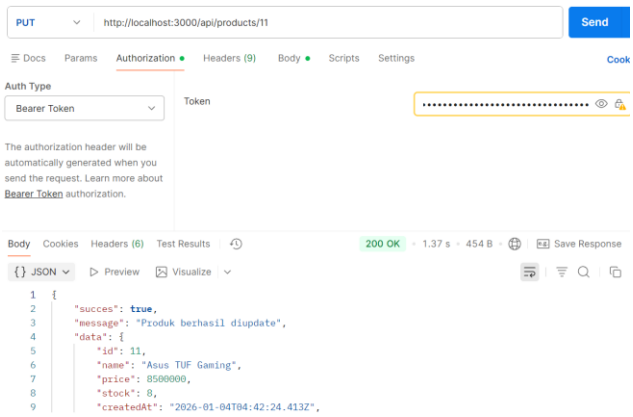
9. Post create product



Hasil eksekusi API di atas menampilkan proses **penambahan data produk baru** yang berhasil. Klien mengirimkan permintaan **POST** ke *endpoint* /api/products dengan menyertakan rincian barang dalam format JSON: nama "Asus ROG Gaming", harga 20.000.000, stok 10 unit, dan dikaitkan dengan kategori ID 1.

Server merespons dengan kode status **201 Created**, yang mengonfirmasi bahwa sumber daya (*resource*) baru telah sukses dibuat di *database*. Dalam badan respons (*body*), sistem mengembalikan objek data yang baru saja disimpan tersebut, namun kini dilengkapi dengan **ID unik 12** (yang dihasilkan otomatis oleh sistem) serta stempel waktu **createdAt** yang mencatat kapan tepatnya data tersebut masuk ke sistem.

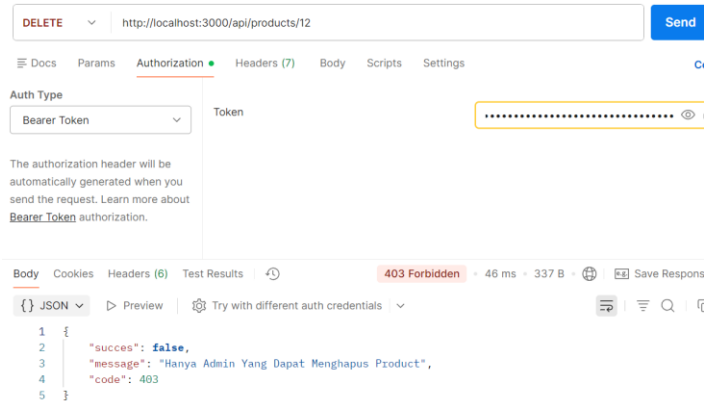
10. Put Product



Hasil eksekusi API ini mengilustrasikan keberhasilan proses **pembaruan data** (*update*) pada sebuah produk yang sudah ada di dalam sistem. Klien menggunakan metode **PUT** yang ditujukan secara spesifik ke *endpoint* /api/products/11, yang berarti instruksi perubahan ini hanya berlaku untuk produk dengan ID unik 11.

Server menanggapi permintaan tersebut dengan status **200 OK**, menandakan bahwa revisi data telah berhasil diproses. Di dalam badan respons (*body*) JSON, sistem memberikan konfirmasi melalui pesan "Produk berhasil diupdate" dan menyertakan objek data yang memuat kondisi terbaru produk tersebut setelah diubah, yakni nama produk menjadi "Asus TUF Gaming" dengan harga 8.500.000 dan stok tersisa 8 unit.

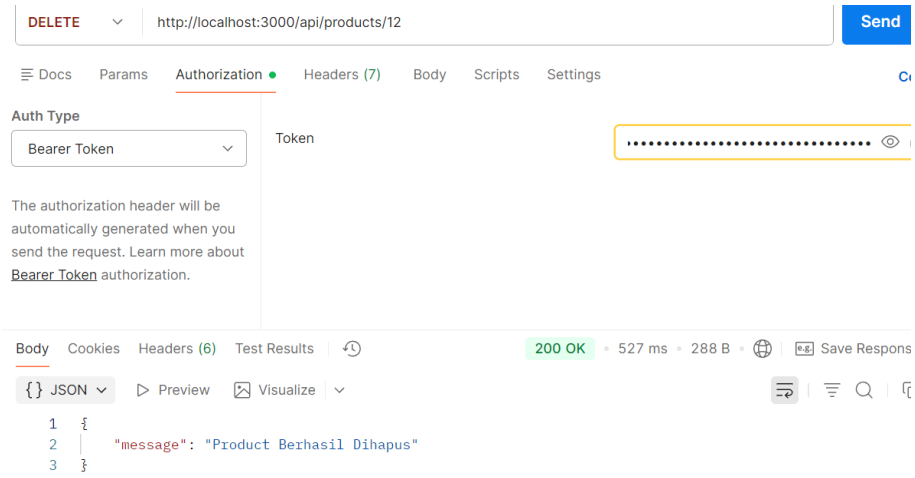
11. Delete dengan token User



Hasil eksekusi API ini menunjukkan percobaan **penghapusan data** yang **gagal** karena masalah perizinan (*authorization*). Klien mengirimkan permintaan dengan metode **DELETE** ke *endpoint* /api/products/12, yang bertujuan untuk menghapus produk dengan ID 12 dari database.

Server menolak permintaan tersebut dengan kode status **403 Forbidden**. Berbeda dengan error *401 Unauthorized* (yang berarti "Anda siapa?"), status 403 ini menyiratkan bahwa server mengenali pengguna (token valid), tetapi pengguna tersebut tidak memiliki hak akses yang cukup untuk melakukan tindakan ini. Pesan error dalam JSON, **"Hanya Admin Yang Dapat Menghapus Product"**, memperjelas bahwa operasi penghapusan sangat dibatasi dan dilindungi oleh *Role-Based Access Control (RBAC)*, di mana hanya akun dengan peran "Admin" yang diizinkan melanjutkannya.

12. Delete dengan token Admin



Respon API ini mengonfirmasi bahwa operasi **penghapusan data** akhirnya **sukses** dijalankan. Permintaan dengan metode **DELETE** yang ditujukan ke *endpoint* /api/products/12 kali ini diterima oleh server dengan kode status **200 OK**. Hal ini mengindikasikan bahwa token otentikasi yang digunakan dalam permintaan ini adalah milik akun dengan level akses **"Admin"**, sehingga hambatan perizinan (Error 403) yang muncul sebelumnya dapat diatasi. Pesan JSON **"Product Berhasil Dihapus"** menegaskan bahwa data produk dengan ID 12 telah dihapus secara permanen dari basis data.

`

BAB III

PENUTUP

Kesimpulan

1. Implementasi Teknologi Modern

Proyek ini berhasil merancang dan menerapkan REST API yang efisien dengan memanfaatkan teknologi terkini, yaitu Next.js sebagai framework utama, PostgreSQL sebagai sistem manajemen basis data, dan Prisma ORM untuk menjembatani komunikasi data. Struktur data dibangun di atas skema Prisma yang mendefinisikan relasi antar model (User, Category, Product), termasuk hubungan *one-to-many* antara kategori dan produk.

2. Mekanisme Keamanan Berlapis

Aspek keamanan menjadi fokus utama dalam pengembangan sistem ini, yang diterapkan melalui beberapa lapisan:

- Enkripsi Data: Sistem memastikan keamanan kredensial pengguna dengan menerapkan *hashing* password menggunakan library bcryptjs sebelum data disimpan ke database.
- Autentikasi JWT: Proses login menghasilkan *JSON Web Token* (JWT) menggunakan library jose, yang berfungsi sebagai tiket akses digital yang aman dan memiliki masa kedaluwarsa.
- Middleware Proteksi: Terdapat lapisan *middleware* yang secara otomatis menyaring setiap permintaan, memverifikasi token pada header *Authorization*, dan menolak akses ilegal sebelum mencapai database.

3. Penerapan *Role-Based Access Control* (RBAC)

Sistem berhasil mengimplementasikan pembagian hak akses yang ketat antara peran Admin dan User:

- Pembatasan Akses: Middleware secara cerdas membatasi rute sensitif. Contohnya, rute untuk melihat seluruh data pengguna (GET /api/users) hanya dapat diakses oleh Admin.
- Proteksi Operasi Destruktif: Fitur krusial seperti penghapusan produk (DELETE) dilindungi oleh validasi ganda. Meskipun pengguna memiliki token valid, sistem akan menolak permintaan dengan status 403 Forbidden jika pengguna tersebut bukan Admin.

4. Validasi dan Integritas Data

Sistem menjamin kualitas data yang masuk melalui penggunaan library Zod untuk memvalidasi input pengguna (seperti format email dan panjang password). Selain itu, fitur *data population* (eager loading) telah berjalan dengan baik, di mana data produk yang diambil secara otomatis menyertakan detail kategorinya dalam satu respons JSON.

5. Hasil Pengujian

Berdasarkan dokumentasi pengujian menggunakan Postman, seluruh fungsionalitas utama (CRUD) dan skenario keamanan telah berjalan sesuai spesifikasi:

- Proses registrasi dan login berhasil menghasilkan token yang valid.
- Sistem berhasil menolak akses tidak sah (User mencoba mengakses fitur Admin) dengan kode status yang tepat.
- Operasi manipulasi data (Create, Update, Delete) oleh Admin tereksekusi dengan sukses dan tersimpan di database.

Secara keseluruhan, laporan ini menunjukkan bahwa sistem backend yang dibangun telah memenuhi standar keamanan modern, memiliki struktur kode yang rapi, dan mampu menangani manajemen inventori dengan pembagian hak akses yang jelas dan aman.