LAPORAN UJIAN AKHIR SEMESTER
PEMROGRAMAN API

“ Sistem Manajemen Buku “

Dosen Pengampu :

Saiful Nur Budiman, S.Kom., M.Kom.

Disusun Oleh :

Isra Naswa Reyka Swahili	(23104410006)
Agistha Ardha Sulistyo P.	(23104410009)
Zaki Zakaria Zakse	(23104410010)
Jusafa Ido Ad’hareza	(23104410022)
Jovanda Kelvin Wibawa P.	(23104410035)

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK DAN INFORMATIKA
UNIVERSITAS ISLAM BALITAR

Januari 2026

KATA PENGANTAR

Alhamdulillah, dengan rasa syukur kehadirat Allah SWT. atas rahmat dan hidayah- Nya, serta Rasulullah saw. atas risalah yang dibawanya, sehingga penyusun dapat menyelesaikan Laporan UAS mata kuliah Pemrograman API di Universitas Islam Balitar, Blitar. Terima kasih penyusun sampaikan kepada Bapak Saiful Nur Budiman, S.Kom., M.Kom., selaku Dosen Pengampu.

Segala usaha dan upaya telah penyusun lakukan untuk menyempurnakan penulisan laporan ini, namun penyusun menyadari bahwa laporan ini masih jauh dari kata sempurna. Semoga laporan ini bermanfaat bagi semua pihak yang membacanya.

Blitar, 2 Januari 2026

P e n y u s u n

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Proyek ini mengembangkan REST API untuk sebuah Sistem Manajemen Buku digital guna mengatasi kebutuhan pengelolaan data yang terstruktur dan aman. Aplikasi dibangun dengan Next.js yang memungkinkan pembuatan API endpoint dan antarmuka yang efisien. PostgreSQL digunakan sebagai basis data yang andal untuk menyimpan informasi, sementara Prisma ORM mempermudah dan mengamankan interaksi dengan database melalui kode yang type-safe.

Fokus utama proyek adalah pada keamanan sistem. Implementasi meliputi JWT Authentication untuk otentikasi pengguna yang aman, hashing password dengan bcrypt untuk melindungi kredensial, dan Role-Based Authorization yang membatasi akses berdasarkan peran Admin atau User. Hal ini memastikan bahwa hanya pengguna yang berwenang yang dapat melakukan operasi tertentu, seperti mengelola data lengkap atau hanya mengakses informasi yang diizinkan.

Dengan pendekatan ini, sistem tidak hanya berfungsi untuk manajemen data buku, tetapi juga menyediakan fondasi yang kokoh, aman, dan siap untuk dikembangkan lebih lanjut.

1.2 Tujuan

1. Membangun REST API yang dapat mengelola data pengguna dan buku.
2. Mengimplementasikan sistem keamanan dengan JWT dan bcrypt.
3. Membuat middleware authentication dan authorization.
4. Melakukan pengujian API menggunakan Postman.

BAB II

LANDASAN TEORI

2.1 Teknologi yang Digunakan

Proyek ini memanfaatkan beberapa teknologi utama untuk membangun sistem manajemen buku yang andal, menjaga keamanan data, serta mempermudah proses pengembangan. Setiap komponen teknologi yang digunakan memiliki peran khusus dan saling melengkapi dalam keseluruhan arsitektur aplikasi meliputi :

- Next.js digunakan sebagai fondasi utama dalam pengembangan aplikasi. Sebagai framework berbasis React, Next.js tidak hanya mendukung pembuatan antarmuka pengguna, tetapi juga menyediakan fitur *API Routes* yang memungkinkan proses pembuatan backend secara terintegrasi. Dengan demikian, pengembang tidak perlu menyiapkan server terpisah, karena Next.js telah menyediakan lingkungan lengkap untuk menangani request, response, dan logika API secara langsung dalam satu proyek.

- PostgreSQL berperan sebagai sistem manajemen basis data (DBMS) yang menangani penyimpanan seluruh data buku dan pengguna. PostgreSQL dipilih karena sifatnya yang *powerful* dan konsisten sesuai standar ACID (Atomicity, Consistency, Isolation, Durability) sehingga menjamin keandalan transaksi. Selain itu, PostgreSQL mampu menangani banyak akses secara bersamaan (*concurrent access*) tanpa mengorbankan performa, sehingga cocok untuk aplikasi yang berpotensi berkembang dan melayani banyak pengguna.

- Prisma ORM digunakan untuk menghubungkan aplikasi dengan database PostgreSQL. Prisma menyediakan pendekatan modern dalam manajemen data melalui *type-safe queries*, yang memungkinkan deteksi kesalahan sejak proses pengembangan melalui fitur *auto-completion* dan validasi tipe. Selain itu, Prisma mendukung pengelolaan skema database dan migrasi secara otomatis sehingga perubahan struktur data dapat dilakukan lebih aman dan terorganisir.

- Untuk aspek keamanan dan autentikasi, proyek ini mengimplementasikan JWT (JSON Web Token) sebagai standar pertukaran data aman antar server dan klien. JWT

memiliki struktur yang terdiri dari *Header, Payload,* dan *Signature*, di mana bagian *Signature* ditandatangani secara digital untuk memastikan token tidak dapat diubah tanpa otorisasi. Teknologi ini memudahkan proses validasi identitas pengguna dalam setiap permintaan ke sistem.

- Selanjutnya, Bcrypt digunakan untuk mengamankan kata sandi pengguna sebelum disimpan dalam database. Algoritma hashing ini secara otomatis menambahkan *salt* untuk setiap proses enkripsi sehingga kata sandi tidak tersimpan dalam bentuk asli. Bcrypt juga mendukung *adaptive hashing* yang membuat proses enkripsi dapat ditingkatkan tingkat kesulitannya seiring perkembangan kemampuan perangkat keras, sehingga serangan *brute force* menjadi jauh lebih sulit dilakukan.

BAB III

PERANCANGAN SISTEM

3.1 Database Schema

Tabel Users:

- id (Primary Key, Auto Increment)
- name, email (unique), password (hashed)
- role (Admin/User), createdAt

Tabel Books:

- id (Primary Key, Auto Increment)
- title, author, year
- userId (Foreign Key ke Users)
- createdAt

Relasi: One User to Many Books (1:N)

3.2 API Endpoints

Authentication (Public):

- POST /api/auth/register - Registrasi user baru
- POST /api/auth/login - Login dan generate token

Books (Protected):

- GET /api/books - Ambil semua buku (Token required)
- GET /api/books/:id - Ambil buku by ID (Token required)
- POST /api/books - Tambah buku baru (Token required)
- PUT /api/books/:id - Update buku (Token required)
- DELETE /api/books/:id - Hapus buku (Admin only)

Users (Protected):

- GET /api/users - Ambil semua users (Admin only)

3.3 Response Format

Success:

```
{
  "success": true,
  "message": "Operasi berhasil",
  "data": { }
}
```

Error:

```
{
  "success": false,
  "error": "Unauthorized",
  "code": 401
}
```

BAB IV

IMPLEMENTASI

4.1 Setup Database dan Prisma

Prisma schema mendefinisikan dua model (User dan Book) dengan relasi one-to-many. Migration dilakukan dengan command npx prisma migrate dev untuk generate dan eksekusi SQL ke database.

4.2 Authentication

Register:

- Menerima input: name, email, password, role
- Hash password dengan bcrypt.hash() (salt rounds 10)
- Simpan ke database menggunakan Prisma
- Return user data (password tetap terlihat untuk debugging)

Login:

- Validasi email dan password dengan bcrypt.compare()
- Generate JWT token dengan payload: {id, email, role}
- Token dikirim ke client untuk digunakan pada request selanjutnya

4.3 Middleware

Authentication Middleware:

- Extract token dari header Authorization: Bearer <token>
- Verify token dengan jwt.verify() dan JWT_SECRET
- Attach user data ke request object
- Return 401 jika token invalid/tidak ada

Authorization Middleware:

- Cek role user dari token
- Izinkan akses berdasarkan role requirement (Admin/User)
- Return 403 jika role tidak sesuai

4.4 CRUD Operations

Create Book (POST):

- Input: title, author, year
- Ambil userId dari token (req.user.id)
- Simpan ke database dengan Prisma
- Protected dengan token wajib

Read Books (GET):

- GET /api/books - Return semua buku
- GET /api/books/:id - Return buku spesifik
- Protected dengan token wajib

Update Book (PUT):

- Input: title, author, year (optional)
- Update data buku berdasarkan ID
- Protected dengan token wajib

Delete Book (DELETE):

- Hapus buku berdasarkan ID
- Protected dengan token + role Admin only
- Return error 401/403 jika tidak authorized

4.5 Error Handling

Setiap endpoint dilengkapi dengan try-catch untuk menangani error. Response error konsisten dengan format:

- success: false
- error: <error message>
- code: <http status code>

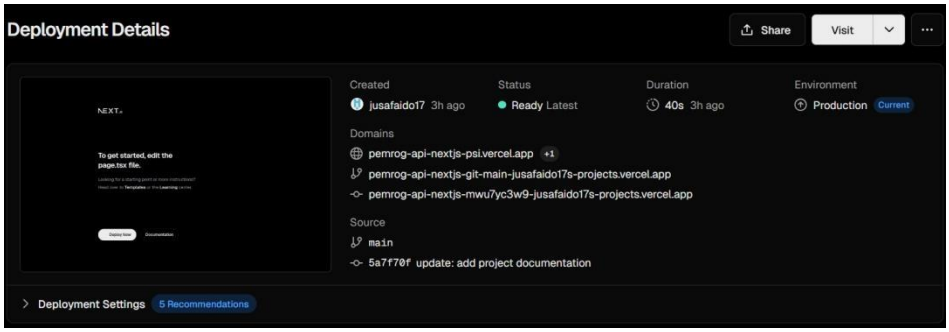
BAB V

HASIL PENGUJIAN

5.1 Deployment Information

Aplikasi telah berhasil di-deploy pada platform Vercel sebagai lingkungan produksi. Proses deployment berjalan lancar dengan waktu eksekusi sekitar 40 detik hingga status dinyatakan siap digunakan. Versi yang aktif saat ini menjalankan konfigurasi Production Environment, sehingga seluruh fitur API dapat diakses secara publik melalui tautan berikut:

Alamat Produksi: https://pemrog-api-nextjs-psi.vercel.app
Status: Siap digunakan *(*Deployment berhasil)
Platform: Vercel
Durasi Deployment: ± 40 detik



[Screenshot 1: Deployment Details di Vercel]

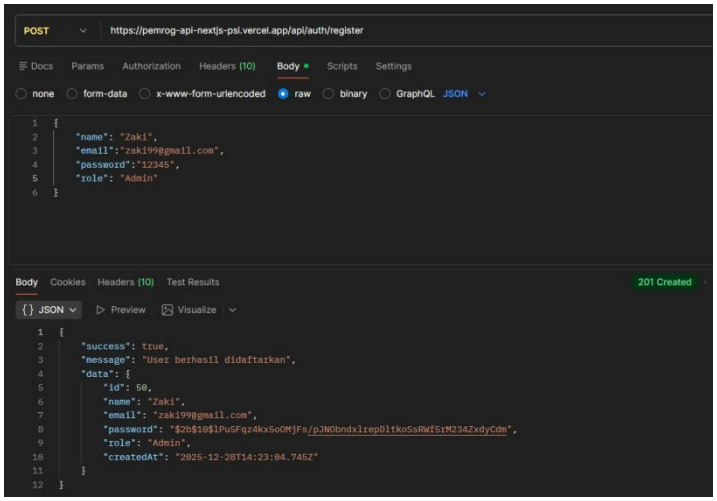
5.2 Testing dengan Postman

Pengujian dilakukan menggunakan Postman untuk memastikan semua endpoint berfungsi dengan baik.

5.2.1 Authentication Testing

1. Register User (Admin)

- Method: POST
- Endpoint: /api/auth/register
- Body: { "name": "Zaki", "email": "zaki99@gmail.com", "password": "12345", "role": "Admin"}
- Status: **201 Created**
- Response: { "success": true, "message": "User berhasil didaftarkan", "data": { "id": 50, "name": "Zaki", "email": "zaki99@gmail.com", "password": "$2b$10$1pU5fGqcz4kx5G0MjFs/pJN0bndxlrepDltkOsSRWf5IrM234ZxdyCdm", "role": "Admin", "createdAt": "2025-12-28T14:23:04.745Z" }}
- **Hasil:** User Admin berhasil dibuat dengan password ter-hash menggunakan bcrypt

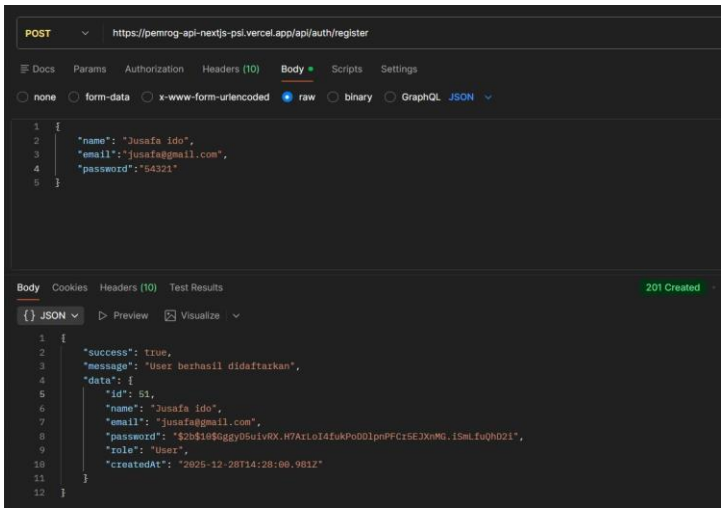


[Screenshot 2: Postman - Register Admin]

2. Register User (Regular User)

- Method: POST
- Endpoint: /api/auth/register
- Body: { "name": "Jusafa ido", "email": "jusafa@gmail.com", "password": "54321"}
- Status: **201 Created**

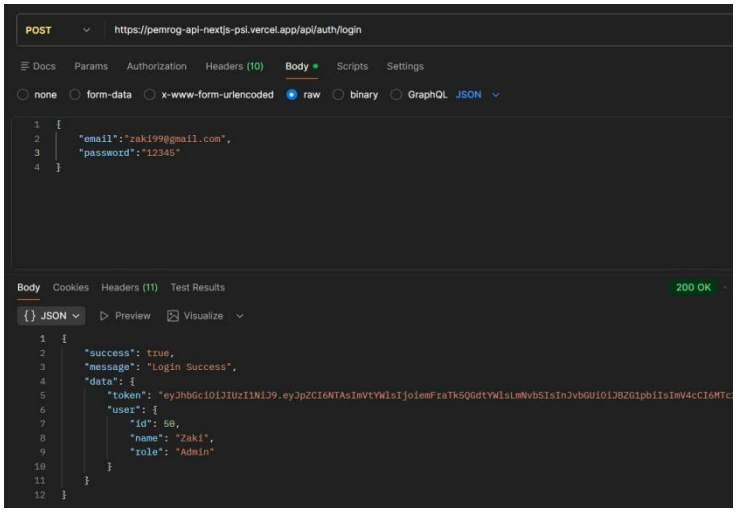
- Response: User ID 51 dengan role "User" (default)
- **Hasil:** User biasa berhasil dibuat



[Screenshot 3: Postman - Register User]

3. Login User Admin

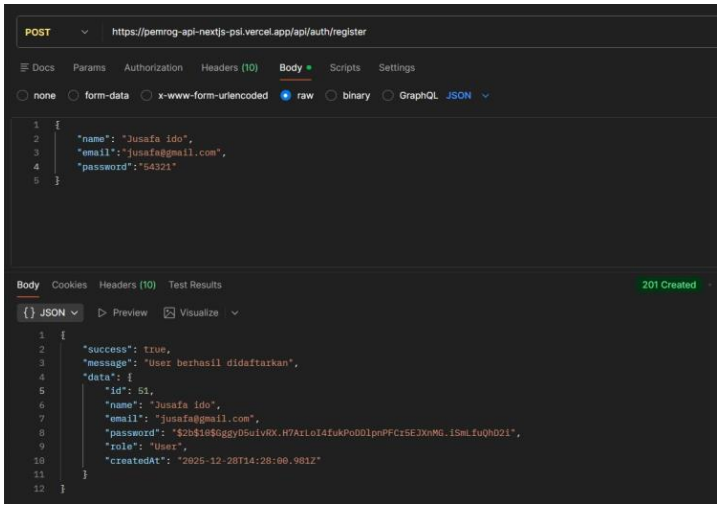
- Method: POST
- Endpoint: /api/auth/login
- Body: { "email": "zaki99@gmail.com", "password": "12345"}
- Status: **200 OK**
- Response: { "success": true, "message": "Login Success", "data": { "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "user": { "id": 50, "name": "Zaki", "role": "Admin" } }}
- **Hasil:** JWT token berhasil di-generate untuk Admin



[Screenshot 4: Postman - Login Admin]

4. Login User Regular

- Method: POST
- Endpoint: /api/auth/login
- Body: { "email": "jusafa@gmail.com", "password": "54321"}
- Status: **200 OK**
- **Hasil:** Token berhasil di-generate untuk user biasa (role: User)

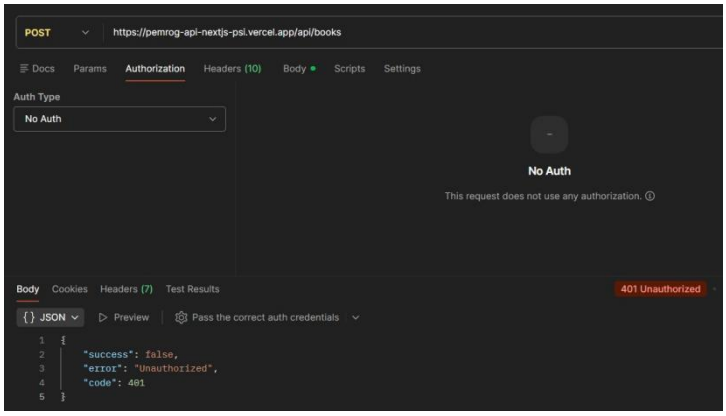


[Screenshot 5: Postman - Login User]

5.2.2 CRUD Books Testing

5. Unauthorized Access - POST Book (Tanpa Token)

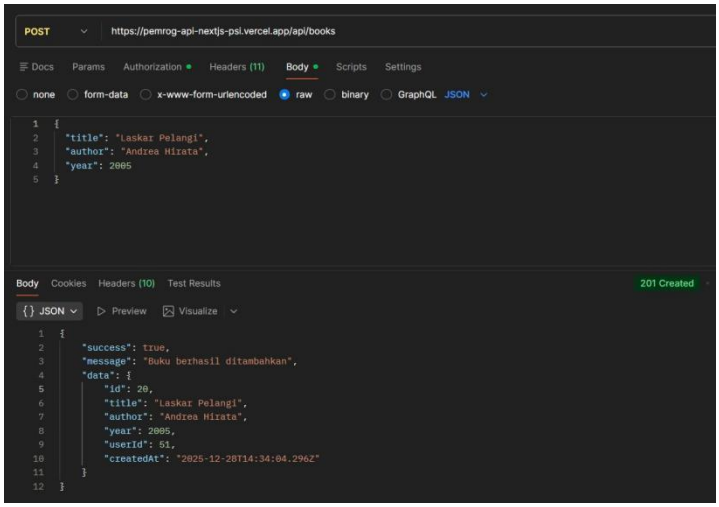
- Method: POST
- Endpoint: /api/books
- Authorization: **No Auth**
- Status: **401 Unauthorized**
- Response: { "success": false, "error": "Unauthorized", "code": 401}
- **Hasil:** Middleware berhasil block request tanpa token



[Screenshot 6: Postman - POST Book Unauthorized]

6. Create Book (With Token)

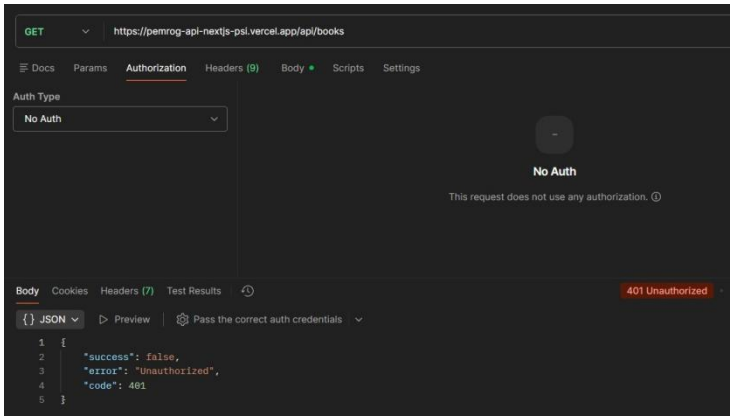
- Method: POST
- Endpoint: /api/books
- Authorization: **Bearer Token** (dari login)
- Body: { "title": "Laskar Pelangi", "author": "Andrea Hirata", "year": 2005}
- Status: **201 Created**
- Response: Book ID 20 berhasil dibuat dengan userId dari token (51)
- **Hasil:** Buku berhasil ditambahkan dengan authentication



[Screenshot 7: Postman - Create Book]

7. Unauthorized Access - GET Books (Tanpa Token)

- Method: GET
- Endpoint: /api/books
- Authorization: **No Auth**
- Status: **401 Unauthorized**
- **Hasil:** Middleware protection berfungsi untuk GET request

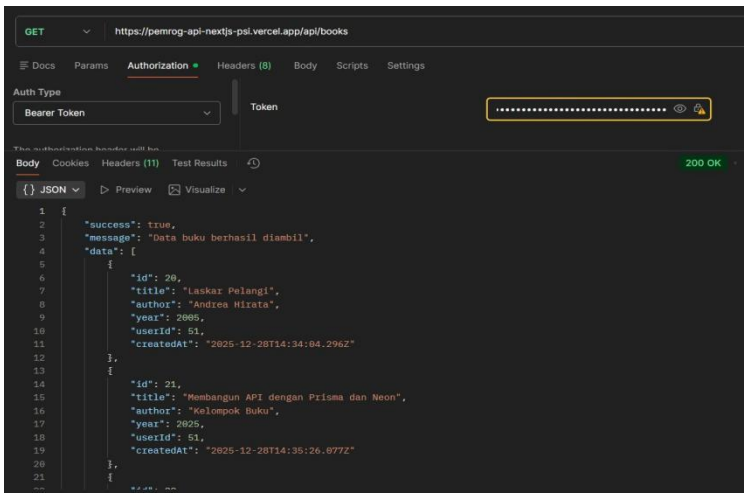


[Screenshot 8: Postman - GET Books Unauthorized]

8. Get All Books (With Token)

- Method: GET
- Endpoint: /api/books
- Authorization: **Bearer Token**

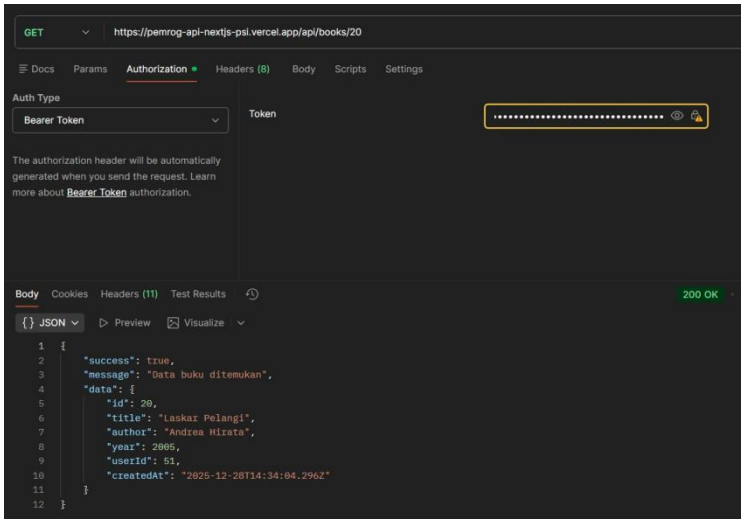
- Status: **200 OK**
- Response: Array berisi 3 buku (ID: 20, 21, 22, 23)
    - Laskar Pelangi - Andrea Hirata (2005)
    - Membangun API dengan Prisma dan Neon - Kelompok Buku (2025)
    - Dan lainnya
- **Hasil:** Semua buku berhasil ditampilkan dengan token valid



[Screenshot 9: Postman - GET All Books]

9. Get Book by ID (With Token)

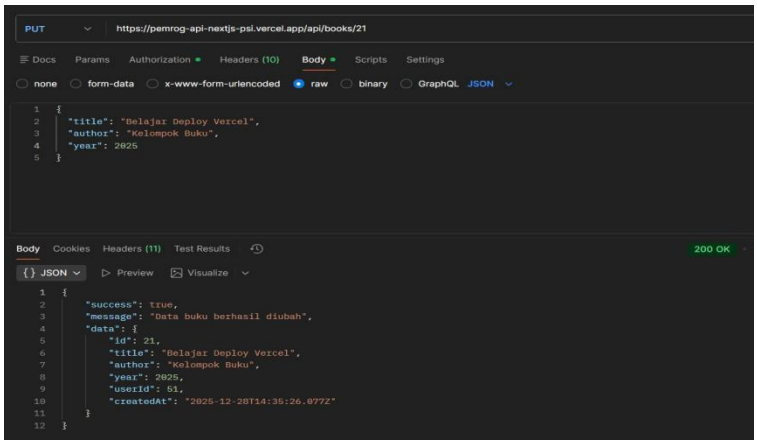
- Method: GET
- Endpoint: /api/books/20
- Authorization: **Bearer Token**
- Status: **200 OK**
- Response: { "success": true, "message": "Data buku ditemukan", "data": { "id": 20, "title": "Laskar Pelangi", "author": "Andrea Hirata", "year": 2005, "userId": 51, "createdAt": "2025-12-28T14:34:04.296Z" }}
- **Hasil:** Detail buku berhasil ditampilkan



[Screenshot 10: Postman - GET Book by ID]

10. Update Book (With Token)

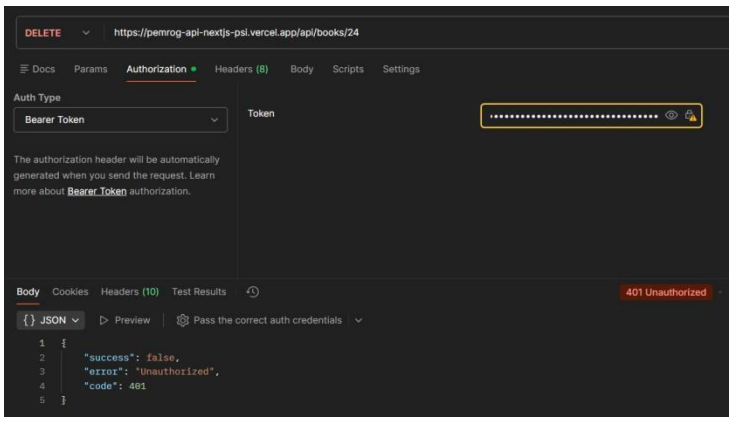
- Method: PUT
- Endpoint: /api/books/21
- Authorization: **Bearer Token**
- Body: { "title": "Belajar Deploy Vercel", "author": "Kelompok Buku", "year": 2025}
- Status: **200 OK**
- Response: Book ID 21 berhasil diupdate
- **Hasil:** Data buku berhasil diubah



[Screenshot 11: Postman - Update Book]

11. Unauthorized Delete (Tanpa Token)

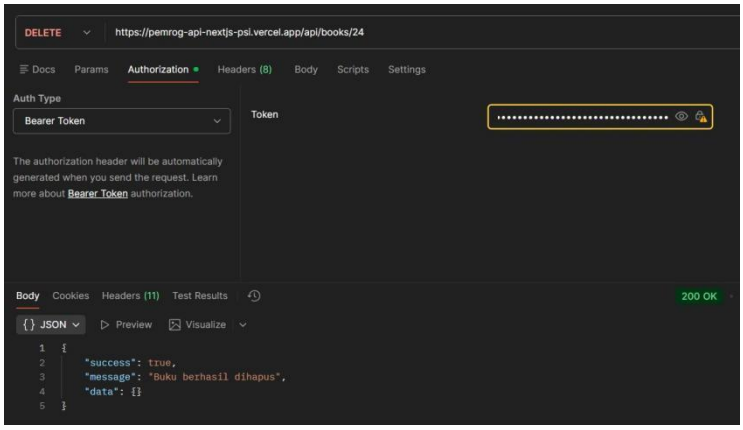
- Method: DELETE
- Endpoint: /api/books/24
- Authorization: **No Auth**
- Status: **401 Unauthorized**
- **Hasil:** Delete operation terproteksi



[Screenshot 12: Postman - DELETE Unauthorized]

12. Delete Book (Admin Token)

- Method: DELETE
- Endpoint: /api/books/24
- Authorization: **Bearer Token** (Admin)
- Status: **200 OK**
- Response: { "success": true, "message": "Buku berhasil dihapus", "data": {}}
- **Hasil:** Buku berhasil dihapus (hanya Admin yang bisa)

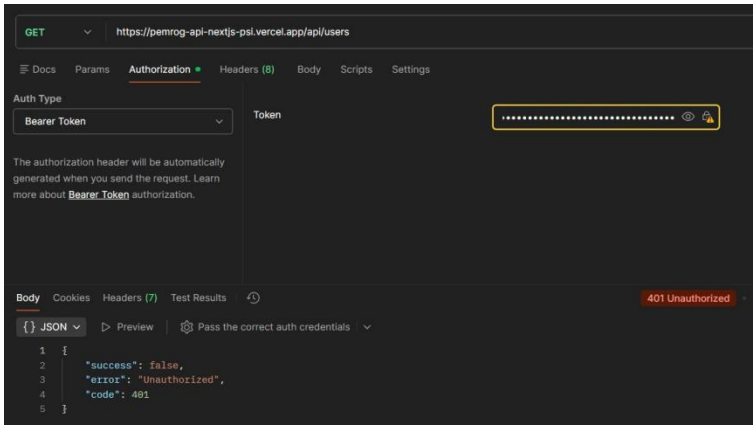


[Screenshot 13: Postman - DELETE Book Success]

5.2.3 Users Management Testing

13. Unauthorized Access - GET Users (Tanpa Token)

- Method: GET
- Endpoint: /api/users
- Authorization: **No Auth**
- Status: **401 Unauthorized**
- **Hasil:** Endpoint users terproteksi

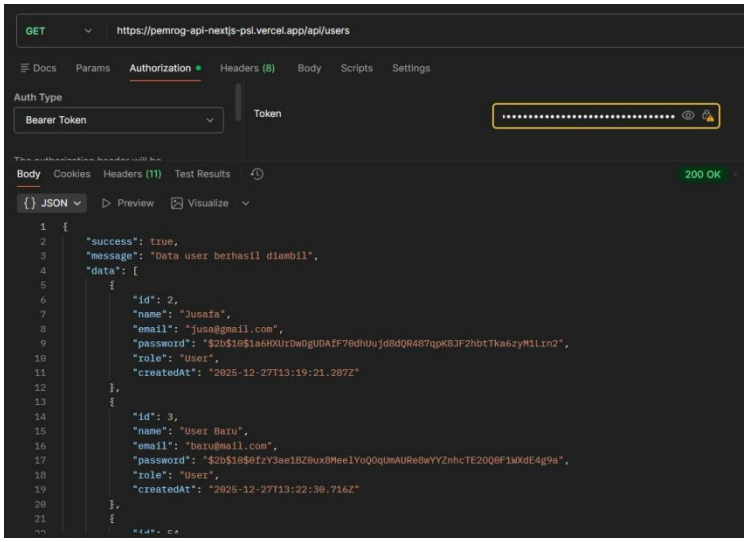


[Screenshot 14: Postman - GET Users Unauthorized]

14. Get All Users (Admin Token)

- Method: GET
- Endpoint: /api/users

- Authorization: **Bearer Token** (Admin)
- Status: **200 OK**
- Response: Array berisi semua users
    - ID 2: Jusafa (User)
    - ID 3: User Baru (User)
    - Dan lainnya dengan password ter-hash
- **Hasil:** Admin berhasil melihat semua user (role-based authorization berfungsi)

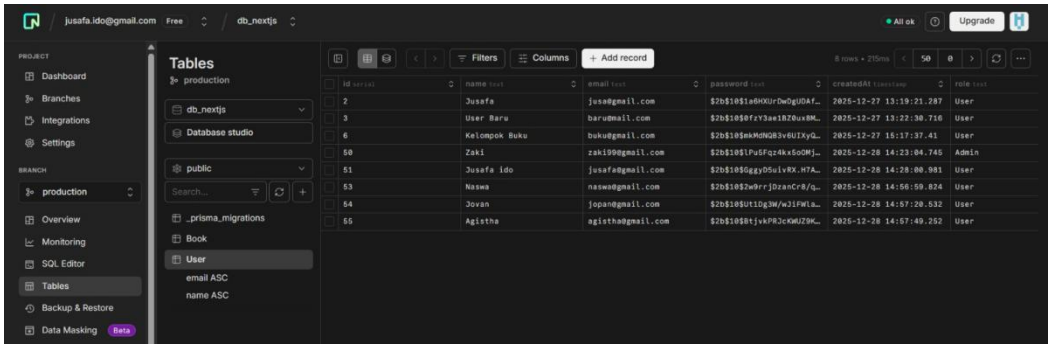


[Screenshot 15: Postman - GET All Users]

5.3 Hasil Pengujian pada Database

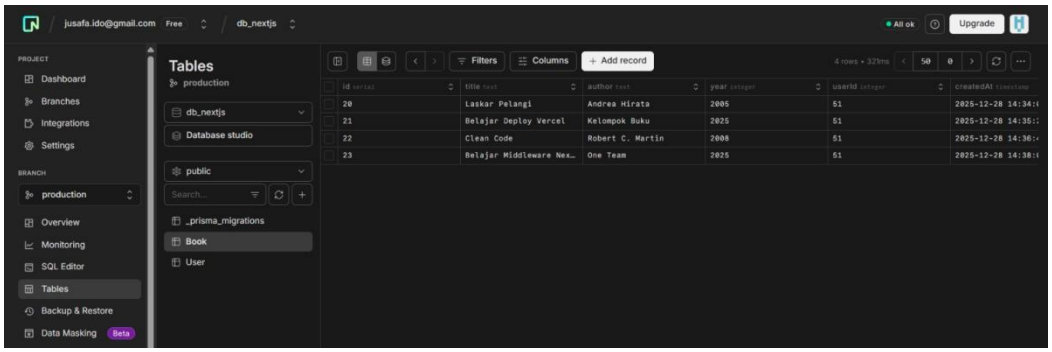
Setelah pengujian API dengan Postman dilakukan pada tahap sebelumnya, langkah lanjutan adalah melakukan pengecekan langsung pada database untuk memastikan bahwa setiap permintaan API (registrasi, login, serta CRUD data buku) benar-benar tersimpan sesuai dengan proses yang dijalankan. Pemeriksaan dilakukan melalui dashboard database PostgreSQL (Neon) dengan melihat isi tabel User dan Book.

Screenshot 1 – Data pada Tabel User



Gambar ini menunjukkan hasil penyimpanan data pada tabel **User**. Semua data yang dibuat melalui endpoint registrasi berhasil tersimpan dengan atribut lengkap, yaitu **id**, **name**, **email**, **password**, **createdAt**, dan **role**. Kolom **password** tampil dalam bentuk *hashed* sebagai hasil enkripsi menggunakan algoritma bcrypt, sehingga tidak ada password yang tersimpan dalam bentuk teks asli. Selain itu, kolom **role** telah terisi sesuai proses otorisasi yang diterapkan, dengan perbedaan hak akses untuk Admin dan User. Tampilan ini membuktikan bahwa proses autentikasi dan otorisasi sistem bekerja sebagaimana mestinya serta sesuai dengan struktur skema pada Prisma ORM.

Screenshot 2 – Data pada Tabel Book



Gambar ini menampilkan isi dari tabel **Book**. Setiap baris berisi informasi buku yang ditambahkan melalui API, meliputi **id**, **title**, **author**, **year**, **userId**, dan **createdAt**. Kolom **userId** menunjukkan relasi antara data buku dengan pengguna yang menambahkan data tersebut. Berdasarkan hasil yang terlihat, seluruh data tersimpan dengan format yang benar dan konsisten. Relasi antara tabel **User** dan **Book** berjalan dengan baik karena **userId** yang tercatat sesuai dengan data pengguna yang ada pada tabel User. Hal ini menjadi indikator bahwa implementasi relasi menggunakan Prisma ORM berfungsi optimal.

Dari kedua hasil verifikasi tersebut dapat disimpulkan bahwa integrasi antara API, Prisma ORM, dan PostgreSQL telah berjalan dengan lancar. Setiap operasi CRUD menghasilkan perubahan data yang valid serta tercermin secara akurat di dalam database.

5.4 Analisis Hasil Testing

Keberhasilan yang telah dilakukan :

1. Authentication berjalan dengan baik (register & login)
2. Password hashing dengan bcrypt berhasil
3. JWT token generation dan verification berfungsi
4. Middleware authentication berhasil protect endpoint
5. Authorization role-based berfungsi (DELETE hanya Admin)
6. Semua operasi CRUD berjalan normal
7. Error handling konsisten untuk unauthorized access

BAB VI

PENUTUP

6.1 Kesimpulan

Proyek berhasil membangun REST API untuk manajemen buku menggunakan Next.js dengan PostgreSQL dan Prisma sebagai pengelola basis data. Sistem autentikasi telah diterapkan menggunakan JWT dan bcrypt, didukung middleware yang memastikan proses authentication serta authorization berjalan untuk melindungi setiap endpoint. Pembagian hak akses berbasis peran (Admin/User) berfungsi efektif dalam membatasi tindakan tertentu sesuai otoritas pengguna. Seluruh fitur CRUD dapat dijalankan sesuai kebutuhan dan tidak ditemukan kendala selama pengujian. API juga telah berhasil di-*deploy* ke Vercel sehingga dapat diakses secara online.

6.2 Saran

Keamanan sistem dapat ditingkatkan dengan menambahkan mekanisme *refresh token* agar proses autentikasi lebih tahan terhadap sesi yang kedaluwarsa. Validasi input juga perlu diperluas, seperti pemeriksaan format email dan kekuatan password, untuk mencegah kesalahan dan potensi eksploitasi. Selain itu, penerapan *logging* direkomendasikan untuk membantu proses pemantauan dan penelusuran kesalahan selama aplikasi berjalan. Langkah lanjutan yang juga disarankan adalah menambahkan *unit testing* dan *integration testing* guna memastikan setiap fitur berfungsi sesuai harapan dan tetap stabil saat terjadi perubahan pengembangan.

DAFTAR PUSTAKA

1. Next.js Documentation. (2025). *API Routes*. Retrieved from https://nextjs.org/docs/api-routes/introduction
2. Prisma Documentation. (2025). *Prisma Schema*. Retrieved from https://www.prisma.io/docs/concepts/components/prisma-schema
3. PostgreSQL Documentation. (2025). *PostgreSQL 15 Documentation*. Retrieved from https://www.postgresql.org/docs/
4. JWT.io. (2025). *JSON Web Tokens Introduction*. Retrieved from https://jwt.io/introduction
5. Bcrypt Documentation. (2025). *Bcrypt for Node.js*. Retrieved from https://www.npmjs.com/package/bcrypt

INFORMASI KONEKSI

- Link Repository

GitHub Repository: https://github.com/jusafaido17/pemrog-api-nextjs.git

- Deployment

URL Production: https://pemrog-api-nextjs-psi.vercel.app

Domain Alternatives:

- https://pemrog-api-nextjs-git-main-jusafaido17s-projects.vercel.app
- https://pemrog-api-nextjs-rmd0vm2ry-jusafaido17s-projects.vercel.app

Lampiran Environment Variables:

JWT_SECRET="rahasia_negara_ini_jangan_disebar"

DISABLE_AUTH_MIDDLEWARE="false"

Neon Database

DATABASE_URL="postgresql://neondb_owner:npg_KHaSXAi43vwN@ep-wandering-recipe-a1jz62aj-pooler.ap-southeast-1.aws.neon.tech/db_nextjs?sslmode=require&channel_binding=require"

LAMPIRAN CODING DAN PENJELASAN

1. .env

```
DATABASE_URL="postgresql://postgres:1234@localhost:5432/db_nextjs?schem
a=public"
JWT_SECRET = pemrog_2026_super_secret_khusus_backend


#neondb
DATABASE_URL="postgresql://neondb_owner:npg_Aktua12fNoDE@ep-curly-leaf-
a1ruyllz-pooler.ap-southeast-
1.aws.neon.tech/neondb?sslmode=require&channel_binding=require"

DISABLE_AUTH_MIDDLEWARE=false
```

File konfigurasi tersebut berfungsi untuk mengatur koneksi database, keamanan autentikasi, dan perilaku middleware pada aplikasi backend. DATABASE_URL digunakan untuk menghubungkan aplikasi ke database PostgreSQL, baik secara lokal maupun ke database cloud Neon saat production. JWT_SECRET berfungsi sebagai kunci rahasia untuk membuat dan memverifikasi token JWT yang dipakai dalam proses login dan autentikasi pengguna. Sementara itu, DISABLE_AUTH_MIDDLEWARE digunakan untuk mengaktifkan atau menonaktifkan middleware autentikasi, misalnya untuk keperluan testing atau debugging.

2. schema.prisma

```
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/User)
  createdAt DateTime @default(now())

  books     Book[]   // Relasi: Satu user bisa punya banyak buku
}

model Book {
  id        Int      @id @default(autoincrement())
  title     String
  author    String
  year      Int
  userId    Int?     // Foreign Key ke User
  user      User?    @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now())
}
```

Kode Prisma Schema tersebut berfungsi untuk mengatur bagaimana Prisma terhubung ke database PostgreSQL dan bagaimana struktur tabelnya dibuat. Bagian generator client digunakan untuk menentukan Prisma Client yang akan di-generate agar bisa dipakai di kode JavaScript/TypeScript. datasource db mengatur sumber database yang diambil dari variabel environment DATABASE_URL. Model User mendefinisikan tabel pengguna dengan atribut seperti nama, email, password, peran, dan relasi ke banyak Book. Model Book mendefinisikan tabel buku yang berelasi ke User melalui userId, sehingga satu user dapat memiliki banyak buku.

3. app/api/users/route.js

```
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();

      // Response Sukses Sesuai Ketentuan
      return NextResponse.json({
        success: true,
        message: "Data user berhasil diambil",
        data: users
      });
  } catch (error) {
    console.error("Error get Data: ", error);

    // Response Error Sesuai Ketentuan (401)
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
  }
}

//! 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) {
```

```
      // Jika validasi gagal, tetap gunakan format Unauthorized 401
sesuai ketentuan
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401,
        details: validation.error.flatten().fieldErrors // Opsional:
tetap sertakan detail Zod
      }, { status: 401 });
    }

    const newUser = await prisma.user.create({
      data: body
    });

    // Response Sukses Sesuai Ketentuan
    return NextResponse.json({
      success: true,
      message: "User berhasil ditambahkan",
      data: newUser
    }, { status: 200 });

  } catch (error) {
    console.error("Error POST User: ", error);

    // Response Error Sesuai Ketentuan (401)
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
  }
}
```

Kode tersebut membuat endpoint API untuk mengelola data **user** menggunakan Next.js, Prisma, dan validasi Zod. Pertama, dibuat UserSchema untuk memastikan data yang dikirim memiliki nama minimal 3 karakter, email valid, dan password minimal 8 karakter. Fungsi **GET** digunakan untuk mengambil semua data user dari database; jika berhasil akan mengembalikan respons sukses berisi data user, dan jika gagal akan mengirim respons error 401. Sementara itu, fungsi **POST** digunakan untuk menambahkan user baru. Data yang diterima divalidasi terlebih dahulu menggunakan Zod; jika tidak valid, server mengembalikan error 401 beserta detail kesalahan. Jika validasi berhasil, data disimpan ke database melalui Prisma dan API mengirim respons

sukses yang berisi informasi user baru. Dengan demikian, kode ini sudah menggabungkan akses database, validasi input, serta penanganan error yang konsisten.

4. **app/api/users/[id]/route.js**

```
// export async function GET(request, {params}) {

//   const {id} = await params;
//   return Response.json({
//     info: `Anda mengakses  ID: ${id}`
//   });
// }

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


// //GET
// export async function GET(request, {params}){

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

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

//     return NextResponse.json(
//       user,
//       {status: 200}
//     );
// }

//!PUT
// export async function PUT(request, {params}){
//     const resolvedParams = await params;
//     const id = parseInt(resolvedParams.id);

//     const data = await request.json();
//     const updateUser = await prisma.user.update({
//         where :{id: id},
//         data: {
//             name: data.name,
```

```
//             email: data.email,
//             password: data.password
//         }
//     });

//     return NextResponse.json({
//         message: "User berhasil diupdate",
//         data: updateUser},
//         {status: 200}
//     );
// }

// //!DELETE
// export async function DELETE(request, {params}){
//     const resolvedParams = await params;
//     const id = parseInt(resolvedParams.id);

//     const deleteUser = await prisma.user.delete({
//         where: {id, id}
//     });

//     return NextResponse.json({
//         message: "User berhasil dihapus",
//         data: deleteUser},
//         {status: 200}
//     );
// }

// ====================== dengan try-catch
=====================================

// GET: Ambil detail satu user berdasarkan ID
export async function GET(request, { params }) {
  try {
    const resolvedParams = await params;
    const id = parseInt(resolvedParams.id);

    if (isNaN(id)) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

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

```
    if (!user) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

    return NextResponse.json({
      success: true,
      message: "User berhasil ditemukan",
      data: user
    }, { status: 200 });

  } catch (error) {
    console.error("Error GET User: ", error);
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
  }
}

// PUT: Mengupdate data user
export async function PUT(request, { params }) {
  try {
    const resolvedParams = await params;
    const id = parseInt(resolvedParams.id);

    if (isNaN(id)) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

    const data = await request.json();
    const updateUser = await prisma.user.update({
      where: { id: id },
      data: {
        name: data.name,
        email: data.email,
        password: data.password
      }
    });
```

```
    return NextResponse.json({
      success: true,
      message: "User berhasil diupdate",
      data: updateUser
    }, { status: 200 });

  } catch (error) {
    console.error("Error PUT User: ", error);
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
  }
}

// DELETE: Menghapus user
export async function DELETE(request, { params }) {
  try {
    const resolvedParams = await params;
    const id = parseInt(resolvedParams.id);

    if (isNaN(id)) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

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

    return NextResponse.json({
      success: true,
      message: "User berhasil dihapus",
      data: deletedUser
    }, { status: 200 });

  } catch (error) {
    console.error("Error DELETE User: ", error);
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
```

```
  }
}
```

Kode tersebut adalah API route Next.js untuk mengelola data user berdasarkan ID menggunakan Prisma. Fungsi GET digunakan untuk mengambil detail satu user berdasarkan parameter id, PUT digunakan untuk memperbarui data user, dan DELETE digunakan untuk menghapus user dari database. Setiap fungsi dilengkapi try-catch untuk menangani error dan memastikan hanya data dengan id yang valid yang diproses, serta mengembalikan respons dalam format JSON dengan status sukses atau error (401) sesuai ketentuan.

5. app/api/books/route.js

```
import { prisma } from "@/lib/prisma";
import { NextResponse } from "next/server";
import jwt from "jsonwebtoken";

// GET: Ambil semua buku dari Neon DB
export async function GET() {
  try {
    const books = await prisma.book.findMany();

    return NextResponse.json({
      success: true,
      message: "Data buku berhasil diambil",
      data: books
    }, { status: 200 }); // Status 200 untuk pengambilan data
  } catch (error) {
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 }); // Error diseragamkan ke 401 sesuai ketentuan
  }
}

// POST: Tambah buku ke Neon DB
export async function POST(request) {
  try {
    const body = await request.json();
    const { title, author, year } = body;
```

```
    const authHeader = request.headers.get("authorization");
    if (!authHeader) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

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

    // Verifikasi token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const newBook = await prisma.book.create({
      data: {
        title,
        author,
        year: parseInt(year),
        userId: decoded.id, // Menyimpan ID user
      },
    });

    return NextResponse.json({
      success: true,
      message: "Buku berhasil ditambahkan",
      data: newBook
    }, { status: 201 }); // Status 201 karena berhasil membuat data
baru

  } catch (error) {
    console.error(error);
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 }); // Error diseragamkan ke 401 sesuai ketentuan
  }
}
```

Kode tersebut merupakan API route Next.js untuk mengelola data buku pada database menggunakan Prisma. Fungsi GET digunakan untuk mengambil seluruh data buku dari database, sedangkan fungsi POST digunakan untuk menambahkan buku baru setelah melakukan verifikasi token JWT dari header Authorization untuk memastikan user sudah login. Jika token valid, data buku akan disimpan beserta userId pemiliknya, dan

jika terjadi kesalahan atau token tidak valid maka API akan mengembalikan respons error dengan status 401 sesuai ketentuan.

6. app/api/books/[id]/route.js

```
import { prisma } from "@/lib/prisma";
import { NextResponse } from "next/server";
import jwt from "jsonwebtoken";

// GET: Ambil detail satu buku berdasarkan ID (Protected)
export async function GET(request, { params }) {
    try {
        const { id } = await params;
        const bookId = parseInt(id);

        // 1. Verifikasi Token (Sesuai syarat Poin 4)
        const authHeader = request.headers.get("authorization");
        if (!authHeader) {
            return NextResponse.json({ success: false, error:
"Unauthorized", code: 401 }, { status: 401 });
        }

        const token = authHeader.split(" ")[1];
        jwt.verify(token, process.env.JWT_SECRET);

        // 2. Cari data di Database
        const book = await prisma.book.findUnique({
            where: { id: bookId }
        });

        if (!book) {
            return NextResponse.json({
                success: false,
                error: "Unauthorized",
                code: 401
            }, { status: 401 });
        }

        return NextResponse.json({
            success: true,
            message: "Data buku ditemukan",
            data: book
        }, { status: 200 });

    } catch (error) {
```

```
        return NextResponse.json({
            success: false,
            error: "Unauthorized",
            code: 401
        }, { status: 401 });
    }
}

// DELETE: Menghapus buku (Admin Only)
export async function DELETE(request, { params }) {
    try {
        const { id } = await params;
        const bookId = parseInt(id);

        // 1. Ambil & Verifikasi Token
        const authHeader = request.headers.get("authorization");
        if (!authHeader) {
            return NextResponse.json({ success: false, error:
"Unauthorized", code: 401 }, { status: 401 });
        }

        const token = authHeader.split(" ")[1];
        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        // 2. Role Based Authorization (Poin 4: Admin Only)
        if (decoded.role !== "Admin") {
            return NextResponse.json({
                success: false,
                error: "Unauthorized",
                code: 401
            }, { status: 401 });
        }

        await prisma.book.delete({
            where: { id: bookId }
        });

        return NextResponse.json({
            success: true,
            message: "Buku berhasil dihapus",
            data: {}
        }, { status: 200 });

    } catch (error) {
        return NextResponse.json({
            success: false,
            error: "Unauthorized",
            code: 401
```

```
        }, { status: 401 });
    }
}

// PUT: Mengupdate data buku (User atau Admin)
export async function PUT(request, { params }) {
    try {
        const { id } = await params;
        const body = await request.json();

        // 1. Ambil & Verifikasi Token (Poin 4: Token Wajib)
        const authHeader = request.headers.get("authorization");
        if (!authHeader) {
            return NextResponse.json({ success: false, error:
"Unauthorized", code: 401 }, { status: 401 });
        }

        const token = authHeader.split(" ")[1];
        jwt.verify(token, process.env.JWT_SECRET);

        const updatedBook = await prisma.book.update({
            where: { id: parseInt(id) },
            data: {
                title: body.title,
                author: body.author,
                year: parseInt(body.year)
            }
        });

        return NextResponse.json({
            success: true,
            message: "Data buku berhasil diubah",
            data: updatedBook
        }, { status: 200 });

    } catch (error) {
        return NextResponse.json({
            success: false,
            error: "Unauthorized",
            code: 401
        }, { status: 401 });
    }
}
```

Kode tersebut merupakan API route Next.js untuk mengelola data buku berdasarkan ID dengan sistem keamanan berbasis JWT. Fungsi GET digunakan untuk mengambil detail satu buku setelah token diverifikasi, fungsi PUT digunakan untuk memperbarui

data buku oleh user yang sudah login, dan fungsi DELETE hanya dapat dilakukan oleh user dengan role Admin. Seluruh endpoint dilindungi dengan verifikasi token dan akan mengembalikan respons error 401 jika token tidak ada, tidak valid, atau user tidak memiliki hak akses.

7. app/api/auth/login/route.js

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

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) {
      // Sesuai ketentuan: success false, error Unauthorized, code 401
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

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

    if (!isPasswordValid) {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }

    // 3. Buat Token
    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")
      .sign(secret);

    // 4. Response Sukses Sesuai Ketentuan
    return NextResponse.json({
        success: true,
        message: "Login Success",
        data: {
            token: token,
            user: {
                id: user.id,
                name: user.name,
                role: user.role
            }
        }
    });

  } catch (error) {
    console.error("Error Login: ", error);
    // Sesuai ketentuan: Gunakan 401 untuk semua error
    return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
    }, { status: 401 });
  }
}
```

Kode tersebut adalah API login pada Next.js yang digunakan untuk melakukan autentikasi user. Endpoint ini akan mencari user berdasarkan email, memverifikasi kecocokan password menggunakan bcrypt, lalu jika valid akan membuat token JWT yang berisi informasi user (id, email, role) dengan masa berlaku dua jam. Jika login berhasil, API mengembalikan token dan data user, sedangkan jika email atau password salah, atau terjadi error lain, maka akan dikembalikan respons error dengan status 401 sesuai ketentuan.

8. app/api/auth/register/route.js

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

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

    // 1. Hash Password
    const hashedPassword = await bcrypt.hash(password, 10);

    // 2. Simpan ke DB
    const newUser = await prisma.user.create({
      data: {
        name,
        email,
        password: hashedPassword,
        role: role || "User",
      },
    });

    // --- RESPONSE SUKSES SESUAI KETENTUAN ---
    return NextResponse.json({
      success: true,
      message: "User berhasil didaftarkan",
      data: newUser
    }, { status: 201 });

  } catch (error) {
    console.error("Error Register: ", error);

    // --- RESPONSE ERROR SESUAI KETENTUAN ---
    return NextResponse.json({
      success: false,
      error: "Unauthorized",
      code: 401
    }, { status: 401 });
  }
}
```

Kode tersebut merupakan endpoint **POST** untuk proses registrasi user. Saat data dikirim oleh client, server terlebih dahulu membaca name, email, password, dan role dari body request. Sebelum disimpan, password tidak disimpan dalam bentuk asli, melainkan di-**hash** menggunakan bcrypt agar lebih aman. Setelah password terenkripsi, data user disimpan ke database melalui Prisma, dengan nilai default role sebagai "User" jika tidak dikirim dari client. Jika proses berhasil, server mengembalikan respons status **201** yang menandakan pembuatan data baru beserta informasi user yang tersimpan. Namun jika terjadi kesalahan — misalnya masalah database atau input tidak sesuai —

server akan menangkapnya di blok catch dan mengirim respons error dengan status **401** dan pesan “Unauthorized”.

9. lib/prisma.js

```
// import { PrismaClient } from "@/app/generated/prisma/client";
//comment client lama
import { PrismaClient } from "@prisma/client";
import { withAccelerate } from "@prisma/extension-accelerate";

const globalForPrisma = global;

export const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    log: process.env.NODE_ENV === 'development' ? ['query', 'error',
'warn'] : ['error'],
  }).$extends(withAccelerate());

if (process.env.NODE_ENV !== "production") {
  globalForPrisma.prisma = prisma;
}
```

Kode tersebut digunakan untuk menginisialisasi Prisma Client agar dapat digunakan di seluruh aplikasi Next.js secara efisien. Prisma disimpan di variabel global agar tidak dibuat ulang setiap kali terjadi hot reload pada mode development, sehingga mencegah terlalu banyak koneksi database. Konfigurasi log hanya diaktifkan detail saat development, dan ekstensi withAccelerate digunakan untuk meningkatkan performa akses database, khususnya saat menggunakan Prisma Accelerate di environment production.

10. middleware.js

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

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

```
  if (process.env.DISABLE_AUTH_MIDDLEWARE === "true") return
NextResponse.next();
  if (pathname.startsWith("/api/auth/")) return NextResponse.next();

  const authHeader = request.headers.get("authorization");
  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return NextResponse.json({ success: false, error: "Unauthorized",
code: 401 }, { status: 401 });
  }

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

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

  // 1. Sesuaikan dengan Login: Login pakai 'role', maka di sini pakai
'role'
  if (pathname.startsWith("/api/users")) {

    // 2. Cek apakah role-nya adalah 'Admin'
    // Pastikan huruf 'A' besar/kecil sesuai dengan yang ada di
database Anda
    if (payload.role !== "Admin") {
      return NextResponse.json({
        success: false,
        error: "Unauthorized",
        code: 401
      }, { status: 401 });
    }
  }

  return NextResponse.next();
} catch (error) {
  return NextResponse.json({
    success: false,
    error: "Unauthorized",
    code: 401
  }, { status: 401 });
  }
}

export const config = {
  matcher: ["/api/books/:path*", "/api/users/:path*"],
};
```

Kode middleware tersebut berfungsi sebagai lapisan keamanan untuk endpoint API dengan memverifikasi token JWT pada setiap request ke /api/books dan /api/users.

Middleware akan melewati proses autentikasi jika dimatikan lewat environment atau jika request menuju endpoint auth, lalu memeriksa header Authorization, memverifikasi token menggunakan JWT_SECRET, dan melakukan pengecekan role Admin khusus untuk route /api/users. Jika token tidak ada, tidak valid, atau role tidak sesuai, maka request akan ditolak dengan respons error 401, sedangkan request yang lolos akan diteruskan ke handler API.