

LAPORAN UAS PEMROGRAMAN

API



Disusun Oleh:

1. Gilbert Wijaya (23104410083)
2. Nanda Nalendra Bagaskara (23104410069)
3. Sandi Widya Permana (23104410053)
4. Rafii Rahmadiansyah (23104410052)
5. Rafli Fattah Nur Ikhsan (23104410055)
6. Danang Sugeng Widagdo (23104410068)

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK DAN INFORMATIKA
UNIVERSITAS ISLAM BALITAR

2025

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era pengembangan aplikasi web modern, arsitektur software telah beralih menuju pendekatan yang lebih terdistribusi, di mana backend dan frontend sering kali dipisahkan untuk meningkatkan skalabilitas dan fleksibilitas. Representational State Transfer (REST) API menjadi standar industri dalam menjembatani komunikasi data antara antarmuka pengguna dan server. Kemampuan untuk membangun REST API yang handal, cepat, dan aman merupakan kompetensi fundamental yang harus dimiliki oleh seorang pengembang perangkat lunak.

Proyek Ujian Akhir Semester (UAS) ini berfokus pada pengembangan backend aplikasi manajemen data menggunakan teknologi Next.js. Meskipun dikenal sebagai kerangka kerja frontend, Next.js memiliki kemampuan server-side yang mumpuni melalui fitur API Routes-nya. Untuk pengelolaan basis data, sistem ini menggunakan PostgreSQL yang dikenal tangguh, dipadukan dengan Prisma ORM untuk mempermudah interaksi database dan menjaga keamanan type-safety dalam kode.

Selain fungsionalitas dasar Create, Read, Update, dan Delete (CRUD), aspek keamanan menjadi prioritas utama dalam pengembangan ini. Ancaman terhadap integritas data menuntut implementasi lapisan keamanan yang ketat. Oleh karena itu, sistem ini menerapkan mekanisme autentikasi dan otorisasi berlapis, mulai dari enkripsi kata sandi (password hashing) menggunakan bcrypt, penggunaan JWT (JSON Web Token) untuk manajemen sesi stateless, hingga penerapan Middleware untuk memvalidasi akses berdasarkan peran pengguna (Role-Based Authorization).

Laporan ini akan membahas perancangan dan implementasi REST API untuk Sistem Manajemen Buku, yang bertujuan untuk menyediakan layanan backend yang aman, efisien, dan terstruktur bagi pengguna dengan peran Admin maupun User biasa.

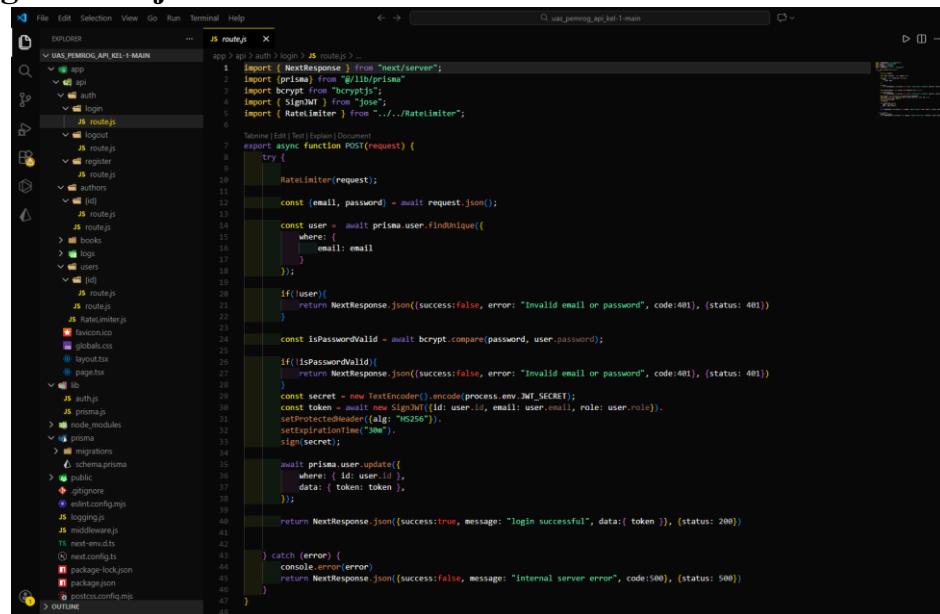
BAB II

PEMBAHASAN

2.1 Penjelasan Coding dan Codingnya

1. app/api/auth/...

A. login/route.js



```
File Edit Selection View Go Run Terminal Help
UAS_PEMBODI_APP_KEL1-1_MAIN ...
route.js
app > api > auth > login > route.js
1 import { NextResponse } from "next/server";
2 import { prisma } from "@lib/prisma"
3 import bcrypt from "bcryptjs";
4 import { NextResponse, json } from "next/server";
5 import { RateLimiter } from "../../../../../RateLimiter";
6
7 Tabnine | Edit | Test | Explain | Document
8
9
10 export async function POST(request) {
11   try {
12     RateLimiter(request);
13
14     const [email, password] = await request.json();
15
16     const user = await prisma.user.findUnique({
17       where: {
18         email: email
19       }
20     });
21
22     if(!user)
23       return NextResponse.json({success:false, error: "Invalid email or password"}, {status: 401});
24
25     const isPasswordValid = await bcrypt.compare(password, user.password);
26
27     if(!isPasswordValid)
28       return NextResponse.json({success:false, error: "Invalid email or password"}, {status: 401});
29
30     const secret = new TextEncoder().encode(process.env.JWT_SECRET);
31     const token = await new SignJWT({id: user.id, email: user.email, role: user.role})
32       .setProtectedHeader("alg: " + "HS256")
33       .setExpirationTime("30m")
34       .sign(secret);
35
36     await prisma.user.update({
37       where: { id: user.id },
38       data: { token: token },
39     });
40
41     return NextResponse.json({success:true, message: "Login successful", data: { token }}, {status: 200});
42
43   } catch (error) {
44     console.error(error)
45     return NextResponse.json({success:false, message: "Internal server error", code:500}, {status: 500})
46   }
47 }
```

Penjelasannya :

1. Impor Modul (Baris 1 - 5)

Bagian ini memanggil pustaka (library) yang diperlukan:

- **NextResponse:** Digunakan untuk mengirimkan respon HTTP (seperti status 200, 401, dll) kembali ke client.
- **prisma:** Interface untuk berinteraksi dengan database.
- **bcrypt:** Pustaka untuk keamanan password (membandingkan password yang diinput dengan yang terenkripsi di database).
- **SignJWT:** Digunakan untuk membuat token keamanan (JWT) sebagai bukti login.
- **RateLimiter:** Fungsi kustom (middleware) untuk membatasi jumlah percobaan login guna mencegah serangan *Brute Force*.

2. Penanganan Request & Rate Limiting (Baris 7 - 10)

- **export async function POST(request):** Mendefinisikan bahwa rute API ini hanya menerima metode **POST**.
- **RateLimiter(request):** Langkah pertama adalah mengecek apakah user sudah terlalu sering mencoba login. Jika ya, proses akan dihentikan di sini.

3. Pengambilan Data & Validasi User (Baris 12 - 22)

- **request.json():** Mengambil data email dan password yang dikirimkan oleh user melalui body request.

- **prisma.user.findUnique**: Mencari data user di tabel database berdasarkan email yang diinput.
- **Pengecekan if (!user)**: Jika email tidak ditemukan di database, sistem langsung mengembalikan error **401 (Unauthorized)**. Pesan dibuat umum ("Invalid email or password") demi alasan keamanan.

4. Verifikasi Password (Baris 24 - 28)

- **bcrypt.compare**: Membandingkan password teks biasa dari user dengan password terenkripsi (hash) yang ada di database.
- **Pengecekan if (!isValidPassword)**: Jika password salah, kembalikan error **401**.

5. Pembuatan JWT Token (Baris 29 - 33)

Setelah user terverifikasi, sistem membuat "tiket masuk" digital:

- **secret**: Mengambil kunci rahasia dari environment variable (JWT_SECRET).
- **SignJWT(...)**: Memasukkan data user (ID, Email, Role) ke dalam token.
- **setExpirationTime("30m")**: Token diatur agar kedaluwarsa dalam 30 menit demi keamanan.
- **sign(secret)**: Menandatangani token secara digital agar tidak bisa dipalsukan.

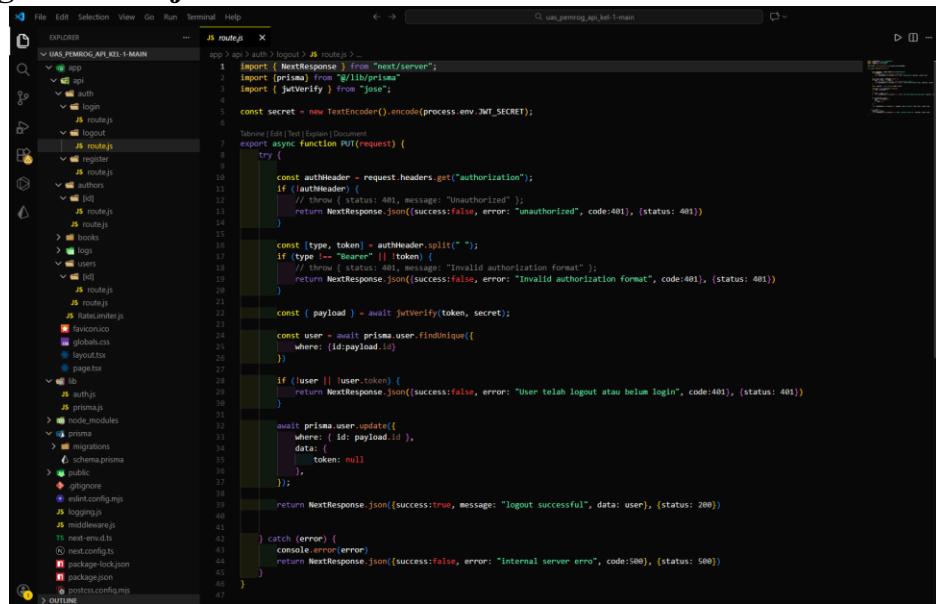
6. Update Database & Respon Berhasil (Baris 35 - 40)

- **prisma.user.update**: Menyimpan token yang baru dibuat ke dalam record user di database (opsional, tergantung arsitektur, biasanya untuk melacak session aktif).
- **NextResponse.json(..., {status: 200})**: Mengirimkan respon sukses ke client beserta token JWT-nya.

7. Error Handling (Baris 43 - 46)

- **catch (error)**: Jika terjadi kesalahan sistem (misal: database mati atau error koding), blok ini akan menangkapnya.
- **Status 500**: Mengembalikan pesan "internal server error" agar aplikasi tidak *crash* di sisi user.

B. logout/route.js



The screenshot shows a code editor with the file `route.js` open. The file is part of a project structure for a Node.js application. The code handles a `PUT` request to the `/auth/logout` endpoint. It first checks if an `authorization` header is present. If not, it returns a `401 Unauthorized` response. Then, it splits the token from the header and checks if it's a `Bearer` token. If not, it returns a `401 Unauthorized` response. It then verifies the token using `jwtVerify` and finds the user in the database. If the user exists, it updates the user's status to `logout`. Finally, it returns a successful `Logout successful` message.

```
app > app > auth > logout > JS route.js > ...
1 import { NextResponse } from "next/server";
2 import { prisma } from "@/lib/prisma";
3 import { jwtVerify } from "jose";
4
5 const secret = new TextEncoder().encode(process.env.JWT_SECRET);
6
7 export async function PUT(request) {
8   try {
9
10     const authHeader = request.headers.get("authorization");
11     if (!authHeader) {
12       // throw { status: 401, message: "Unauthorized" };
13       return NextResponse.json({ success: false, error: "unauthorized", code: 401 }, { status: 401 });
14     }
15
16     const [type, token] = authHeader.split(" ");
17     if (type !== "Bearer" || !token) {
18       // throw { status: 401, message: "Invalid authorization format" };
19       return NextResponse.json({ success: false, error: "Invalid authorization format", code: 401 }, { status: 401 });
20     }
21
22     const { payload } = await jwtVerify(token, secret);
23
24     const user = await prisma.user.findUnique({
25       where: { id: payload.id }
26     });
27
28     if (!user || !user.token) {
29       return NextResponse.json({ success: false, error: "User telah logout atau belum login", code: 401 }, { status: 401 });
30     }
31
32     await prisma.user.update({
33       where: { id: payload.id },
34       data: {
35         token: null
36       }
37     });
38
39     return NextResponse.json({ success: true, message: "Logout successful", data: user }, { status: 200 });
40   } catch (error) {
41     console.error(error);
42     return NextResponse.json({ success: false, error: "internal server error", code: 500 }, { status: 500 });
43   }
44 }
45
46 
```

Penjelasan :

1. Inisialisasi dan Keamanan (Baris 1 - 5)

- **jwtVerify**: Fungsi dari library jose yang digunakan untuk mendekripsi dan memvalidasi keaslian token JWT yang dikirim oleh pengguna.
- **const secret**: Mengonversi kode rahasia (JWT_SECRET) dari file environment menjadi format yang bisa dibaca oleh mesin (Uint8Array) untuk keperluan verifikasi token.

2. Pengambilan Token dari Header (Baris 7 - 14)

- **export async function PUT(request)**: Fungsi ini menggunakan metode **PUT**. Biasanya logout menggunakan POST atau DELETE, namun PUT di sini mungkin dipilih karena tujuannya adalah *mengupdate* status user di database.
- **authHeader**: Sistem mengambil data dari header HTTP bernama `authorization`.
- **Validasi Header**: Jika header kosong, sistem langsung menolak akses dengan status **401 (Unauthorized)**.

3. Ekstraksi dan Validasi Format (Baris 16 - 20)

- **authHeader.split(" ")**: Memisahkan teks "Bearer <token>".
- **Pengecekan Tipe**: Memastikan bahwa token yang dikirim adalah tipe **Bearer**. Jika formatnya salah (misal hanya mengirim token tanpa kata "Bearer"), sistem akan mengembalikan error.

4. Verifikasi Keaslian Token (Baris 22 - 26)

- **jwtVerify(token, secret)**: Langkah paling krusial. Sistem mengecek apakah token tersebut benar-benar dibuat oleh server Anda dan belum kedaluwarsa.
- **payload**: Jika verifikasi berhasil, sistem mengambil data yang tersimpan di dalam token (seperti ID user).

- **prisma.user.findUnique**: Mencari user di database berdasarkan ID yang didapat dari token tersebut.

5. Validasi Status Login (Baris 28 - 30)

- Sistem mengecek apakah user tersebut ada dan apakah di database kolom token-nya masih berisi. Jika kolom token di database sudah kosong (null), berarti user tersebut **sudah logout sebelumnya** atau memang belum login.

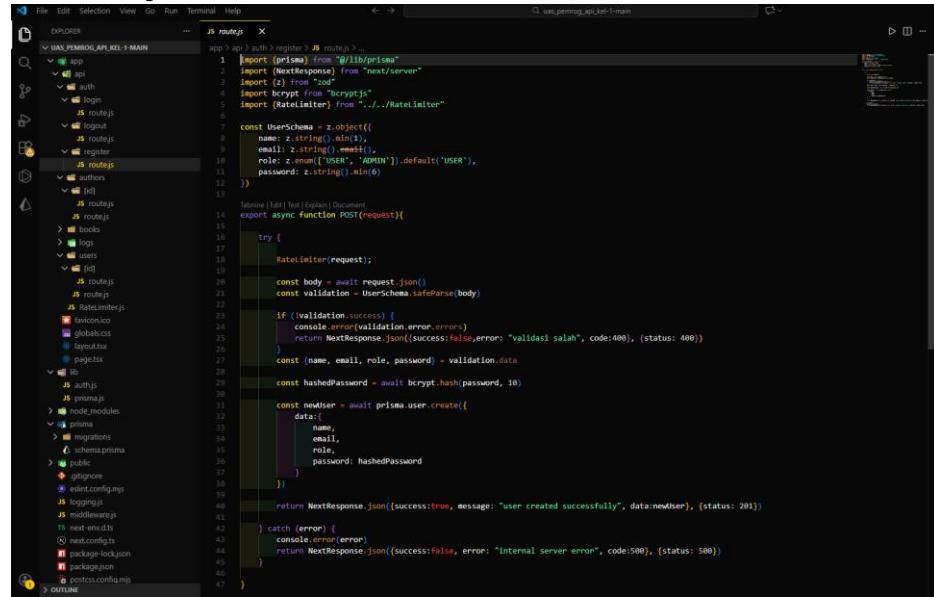
6. Proses Logout - Menghapus Token (Baris 32 - 40)

- **prisma.user.update**: Bagian inti dari logout. Sistem akan mencari user tersebut dan mengubah kolom **token menjadi null**.
- Dengan mengubah menjadi null, token yang dipegang pengguna secara teknis tidak lagi berlaku di sisi server.
- **Respon Berhasil**: Mengirimkan status **200 (OK)** dengan pesan "logout successful".

7. Penanganan Error (Baris 42 - 46)

- Jika terjadi kesalahan (misal token palsu atau database error), bagian catch akan menangkapnya dan mengirimkan respon error **500** atau membiarkan middleware menangani error autentikasi.

C. register/route.js



```
File Edit Selection View Go Run Terminal Help
JS route.js
app> cd auth & npm i
1 import {prisma} from '@lib/prisma'
2 import {NextResponse} from 'next/server'
3 import {z} from 'zod'
4 import bcrypt from 'bcryptjs'
5 import {RateLimiter} from '../..../RateLimiter'
6
7 const UserSchema = z.object({
8   name: z.string().min(1),
9   email: z.string().email(),
10  role: z.enum(['USER', 'ADMIN']).default('USER'),
11  password: z.string().min(6)
12 })
13
14 function validateUser(request) {
15   try {
16     RateLimiter(request);
17
18     const body = await request.json();
19     const validation = UserSchema.safeParse(body);
20
21     if (!validation.success) {
22       console.error(validation.error.message);
23       return NextResponse.json({success: false, error: "validasi salah"}, {status: 400});
24     }
25     const [name, email, role, password] = validation.data;
26
27     const hashedPassword = await bcrypt.hash(password, 10);
28
29     const newUser = await prisma.user.create({
30       data: {
31         name,
32         email,
33         role,
34         password: hashedPassword
35       }
36     });
37
38     return NextResponse.json({success: true, message: "user created successfully", data: newUser}, {status: 201});
39
40   } catch (error) {
41     console.error(error);
42     return NextResponse.json({success: false, error: "internal server error"}, {status: 500});
43   }
44 }
45
46
47
```

Penjelasan :

1. Impor Modul dan Pustaka (Baris 1 - 5)

- **prisma**: Digunakan untuk melakukan operasi *create* data ke database.
- **NextResponse**: Untuk mengirimkan respon HTTP ke client.
- **z (Zod)**: Pustaka untuk validasi skema data guna memastikan input dari user sesuai kriteria (misal: format email harus benar).
- **bcrypt**: Digunakan untuk melakukan *hashing* (enkripsi) password sebelum disimpan ke database agar aman.
- **RateLimiter**: Middleware untuk membatasi jumlah request guna mencegah penyalahgunaan API.

2. Definisi Skema Validasi (Baris 7 - 12)

Bagian ini mendefinisikan aturan main untuk data pendaftaran:

- **name**: Minimal 1 karakter.
- **email**: Harus dalam format email yang valid.
- **role**: Hanya boleh berisi 'USER' atau 'ADMIN', dengan nilai default adalah 'USER'.
- **password**: Minimal 6 karakter untuk keamanan.

3. Ekstraksi dan Validasi Data (Baris 19 - 27)

- **request.json()**: Mengambil data dari body request yang dikirim user.
- **UserSchema.safeParse(body)**: Melakukan validasi data terhadap skema Zod.
- **Pengecekan if (!validation.success)**: Jika data tidak valid (misal: password terlalu pendek), server akan mengirimkan error **400 (Bad Request)** dengan pesan "validasi salah".

4. Enkripsi Password (Baris 29)

- **bcrypt.hash(password, 10):** Password asli dari user tidak boleh disimpan langsung. Baris ini mengubah password menjadi kode acak (hash). Angka 10 adalah *salt rounds* yang menentukan seberapa kuat enkripsinya.

5. Penyimpanan ke Database (Baris 31 - 38)

- **prisma.user.create:** Membuat record baru di tabel user.
- Sistem menyimpan name, email, role, dan hashedPassword yang sudah dienkripsi tadi.

6. Respon Berhasil dan Penanganan Error (Baris 40 - 46)

- **Status 201:** Jika berhasil, sistem mengirimkan status **201 (Created)** yang menandakan data baru telah berhasil dibuat.
- **Error Handling (Catch):** Jika terjadi kesalahan teknis (seperti email duplikat atau database terputus), sistem menangkapnya dan mengirimkan error **500 (Internal Server Error)**.

2. authors/[id]/...

A. route.js

Penjelasan :

1. Impor Modul dan Pustaka (Baris 1 - 5)

- **prisma**: Digunakan untuk melakukan operasi *create* data ke database.
 - **NextResponse**: Untuk mengirimkan respon HTTP ke client.
 - **z (Zod)**: Pustaka untuk validasi skema data guna memastikan input dari user sesuai kriteria (misal: format email harus benar).
 - **bcrypt**: Digunakan untuk melakukan *hashing* (enkripsi) password sebelum disimpan ke database agar aman.
 - **RateLimiter**: Middleware untuk membatasi jumlah request guna mencegah penyalahgunaan API.

2. Definisi Skema Validasi (Baris 7 - 12)

Bagian ini mendefinisikan aturan main untuk data pendaftaran.

- **name**: Minimal 1 karakter.
 - **email**: Harus dalam format email yang valid.

- **role**: Hanya boleh berisi 'USER' atau 'ADMIN', dengan nilai default adalah 'USER'.
- **password**: Minimal 6 karakter untuk keamanan.

3. Ekstraksi dan Validasi Data (Baris 19 - 27)

- **request.json()**: Mengambil data dari body request yang dikirim user.
- **UserSchema.safeParse(body)**: Melakukan validasi data terhadap skema Zod.
- **Pengecekan if (!validation.success)**: Jika data tidak valid (misal: password terlalu pendek), server akan mengirimkan error **400 (Bad Request)** dengan pesan "validasi salah".

4. Enkripsi Password (Baris 29)

- **bcrypt.hash(password, 10)**: Password asli dari user tidak boleh disimpan langsung. Baris ini mengubah password menjadi kode acak (hash). Angka 10 adalah *salt rounds* yang menentukan seberapa kuat enkripsinya.

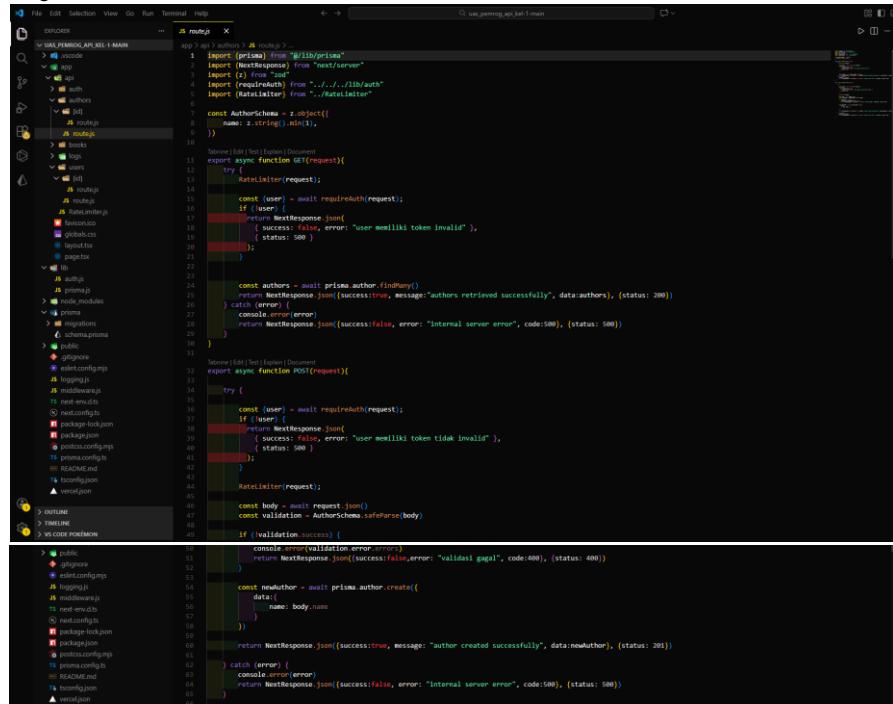
5. Penyimpanan ke Database (Baris 31 - 38)

- **prisma.user.create**: Membuat record baru di tabel user.
- Sistem menyimpan name, email, role, dan hashedPassword yang sudah dienkripsi tadi.

6. Respon Berhasil dan Penanganan Error (Baris 40 - 46)

- **Status 201**: Jika berhasil, sistem mengirimkan status **201 (Created)** yang menandakan data baru telah berhasil dibuat.
- **Error Handling (Catch)**: Jika terjadi kesalahan teknis (seperti email duplikat atau database terputus), sistem menangkapnya dan mengirimkan error **500 (Internal Server Error)**.

B. route.js



The screenshot shows a code editor with the file `route.js` open. The file is part of a Next.js application structure, specifically under the `app/api/authors` directory. The code implements two routes: `GET` and `POST`. The `GET` route uses `prisma.author.findMany()` to retrieve all authors from the database. It includes validation logic using `Zod` (`AuthorSchema`) and `RateLimiter` to handle multiple requests. The `POST` route uses `prisma.author.create()` to add a new author, also validating the input and using `RateLimiter`.

```
import { prisma } from '#lib/prisma'
import { NextResponse } from 'next/server'
import { requireAuth } from '../lib/auth'
import { RateLimiter } from 'rate-limiter-flexible'

const AuthorSchema = Z.object({
  name: Z.string().min(1),
})

const authors = await prisma.author.findMany()
const body = await prisma.author.create({
  data: {
    name: body.name
  }
})

export async function GET(request) {
  try {
    const user = await requireAuth(request)
    if (!user) {
      return NextResponse.json({ success: false, error: "user memiliki token invalid" }, { status: 500 })
    }
    const authors = await prisma.author.findMany()
    if (authors.length === 0) {
      return NextResponse.json({ success: true, message: "authors retrieved successfully", data: authors }, { status: 200 })
    }
    catch (error) {
      console.error(error)
      return NextResponse.json({ success: false, error: "internal server error", code: 500 }, { status: 500 })
    }
  }
}

export async function POST(request) {
  try {
    const user = await requireAuth(request)
    if (!user) {
      return NextResponse.json({ success: false, error: "user memiliki token tidak valid" }, { status: 500 })
    }
    const body = await request.json()
    const validation = AuthorSchema.safeParse(body)
    if (!validation.success) {
      const errors = validation.error.errors
      return NextResponse.json({ success: false, error: "validasi gagal", code: 400 }, { status: 400 })
    }
    const newAuthor = await prisma.author.create({
      data: {
        name: body.name
      }
    })
    return NextResponse.json({ success: true, message: "author created successfully", data: newAuthor }, { status: 201 })
  }
  catch (error) {
    console.error(error)
    return NextResponse.json({ success: false, error: "internal server error", code: 500 }, { status: 500 })
  }
}
```

Penjelasan :

Berikut adalah penjelasan untuk file **Authors Main Route** (`app/api/authors/route.js`). File ini menangani pengelolaan data penulis (author) secara kolektif, yaitu mengambil semua daftar penulis dan menambahkan penulis baru.

Sama seperti rute sebelumnya, file ini sangat mementingkan keamanan dengan mewajibkan login melalui `requireAuth`.

1. Definisi Skema (Baris 7 - 9)

- **AuthorSchema:** Menggunakan library **Zod** untuk memastikan bahwa saat menambah penulis baru, data name harus berupa teks (string) dan tidak boleh kosong (`min(1)`).

2. Fungsi GET - Mengambil Semua Penulis (Baris 11 - 31)

Fungsi ini digunakan untuk menampilkan daftar seluruh penulis yang ada di database.

- **RateLimiter(request):** Membatasi kecepatan permintaan untuk mencegah beban berlebih pada server.
- **requireAuth(request):** Memastikan hanya pengguna terautentikasi yang bisa menarik data ini. Jika token tidak valid, akan muncul error status **500** (walaupun secara logika seharusnya 401, kode Anda diatur ke 500).
- **prisma.author.findMany():** Perintah database untuk mengambil **semua** baris data dari tabel author.
- **Respon:** Jika berhasil, mengirimkan data daftar penulis dengan status **200 (OK)**.

3. Fungsi POST - Menambah Penulis Baru (Baris 32 - 66)

Fungsi ini digunakan untuk mendaftarkan nama penulis baru ke dalam sistem.

- **Proteksi Keamanan:** Kembali menggunakan requireAuth dan RateLimiter sebelum memproses data.
- **Validasi Input:**
 - request.json(): Mengambil data dari body permintaan.
 - AuthorSchema.safeParse(body): Mengecek apakah data sudah sesuai aturan (nama tidak kosong). Jika gagal, mengirim error **400 (Validasi Gagal)**.
- **prisma.author.create:** Memasukkan data nama baru ke kolom name di tabel author database.
- **Respon:** Mengirimkan data penulis yang baru dibuat dengan status **201 (Created)**.

3. users/[id]/...

A. route.js

The screenshot shows three separate code editors or tabs within a terminal window, each displaying a different file from a Next.js application structure:

- File 1 (Top):** routes/api/users/[id].js
- File 2 (Middle):** routes/api/users/[id].js
- File 3 (Bottom):** routes/api/users/[id].js

All three files contain nearly identical code, representing a GET endpoint for a user by ID. The code includes imports for `prisma` and `NextResponse`, defines a schema for the user object, and handles token validation using a RateLimiter. It then checks if the user exists, returns a 404 error if not, and performs a database update. Finally, it returns a success response with the updated user data.

```
1 import {prisma} from '@lib/prisma'
2 import {NextResponse} from 'next/server'
3 import {RateLimiter} from 'rate-limiter'
4 import {requireAuth} from '../../../../../lib/auth'
5
6 const usersSchema = z.object({
7   name: z.string().min(1),
8   email: z.string().email(),
9   role: z.enum(['ADMIN', 'MANAGER']).default('USER'),
10  password: z.string().min(6)
11 })
12
13
14 Talenive | Edit | Test | Explain | Document
15 export async function GET(request, {params}) {
16   try {
17     const [user] = await requireAuth(request);
18     if (!user) {
19       return NextResponse.json(
20         { success: false, error: "user memiliki token tidak valid" },
21         { status: 500 }
22       );
23     }
24     RateLimiter(request);
25
26     const resolvedParams = await params;
27     const id = parseInt(resolvedParams.id);
28
29     if (!isNaN(id)) {
30       return NextResponse.json({success:false, error: "Invalid user ID", code:400}, {status: 400});
31     }
32
33     const User = await prisma.user.findUnique({
34       where: { id: id }
35     });
36
37     if (!User) {
38       return NextResponse.json({success:false, error: "User not found", code:404}, {status: 404});
39     }
40
41     return NextResponse.json({success:true, message:"user found ", data:User}, {status: 200});
42   } catch (error) {
43     console.error(error);
44     return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500});
45   }
46 }
```

```
1 import {prisma} from '@lib/prisma'
2 import {NextResponse} from 'next/server'
3 import {RateLimiter} from 'rate-limiter'
4 import {requireAuth} from '../../../../../lib/auth'
5
6 const usersSchema = z.object({
7   name: z.string().min(1),
8   email: z.string().email(),
9   role: z.enum(['ADMIN', 'MANAGER']).default('USER'),
10  password: z.string().min(6)
11 })
12
13
14 Talenive | Edit | Test | Explain | Document
15 export async function PUT(request, {params}) {
16   try {
17     const [user] = await requireAuth(request);
18     if (!user) {
19       return NextResponse.json(
20         { success: false, error: "user memiliki token tidak valid" },
21         { status: 500 }
22       );
23     }
24     RateLimiter(request);
25
26     const resolvedParams = await params;
27     const id = parseInt(resolvedParams.id);
28
29     const body = await request.json();
30     const validation = usersSchema.safeParse(body);
31
32     if (Validation.success) {
33       console.error(validation.error.error);
34       return NextResponse.json({success:false,error: "validasi gagal", code:400}, {status: 400});
35     }
36
37     const updateUser = await prisma.user.update({
38       where: { id: id },
39       data: {
40         name: body.name,
41         email: body.email,
42         role: body.role,
43         password: body.password
44       }
45     });
46
47     return NextResponse.json({success:true, message: "user updated successfully", data:updateUser}, {status: 200});
48   } catch (error) {
49     console.error(error);
50     return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500});
51   }
52 }
```

```
1 import {prisma} from '@lib/prisma'
2 import {NextResponse} from 'next/server'
3 import {RateLimiter} from 'rate-limiter'
4 import {requireAuth} from '../../../../../lib/auth'
5
6 const usersSchema = z.object({
7   name: z.string().min(1),
8   email: z.string().email(),
9   role: z.enum(['ADMIN', 'MANAGER']).default('USER'),
10  password: z.string().min(6)
11 })
12
13
14 Talenive | Edit | Test | Explain | Document
15 export async function DELETE(request, {params}) {
16   try {
17     const [user] = await requireAuth(request);
18     if (!user) {
19       return NextResponse.json(
20         { success: false, error: "user memiliki token tidak valid" },
21         { status: 500 }
22       );
23     }
24     RateLimiter(request);
25
26     const resolvedParams = await params;
27     const id = parseInt(resolvedParams.id);
28
29     if (!isNaN(id)) {
30       return NextResponse.json({success:false, error: "Invalid user ID", code:400}, {status: 400});
31     }
32
33     const deletedUser = await prisma.user.delete({
34       where: { id: id }
35     });
36
37     // If (deletedUser) {
38     //   return NextResponse.json({success:false, error: "User not found", code:404}, {status: 404});
39     // }
40
41     return NextResponse.json({success:true, message:"user deleted successfully", data:deletedUser}, {status: 200});
42   } catch (error) {
43     console.error(error);
44     return NextResponse.json({success:false, error: "internal server error", code:500}, {status: 500});
45   }
46 }
```

Penjelasan :

1. Skema Validasi Pengguna

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

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

2. Fungsi GET - Mendapatkan Detail Pengguna

Digunakan untuk mengambil informasi profil satu pengguna:

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

3. Fungsi PUT - Memperbarui Data Pengguna

Digunakan untuk mengubah informasi akun pengguna yang sudah ada:

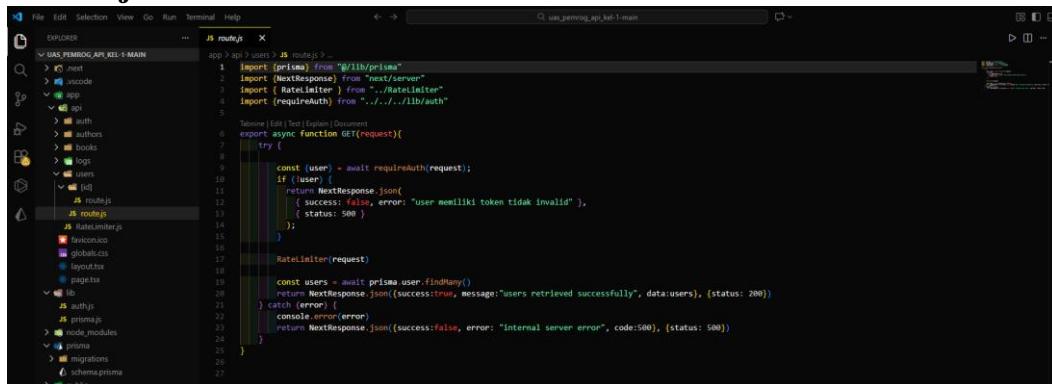
- **Validasi:** Data baru yang dikirim user diperiksa kesesuaianya dengan UserSchema menggunakan safeParse.
- **Update Database:** Menggunakan prisma.user.update untuk memperbarui field nama, email, role, dan password berdasarkan ID pengguna tersebut.
- **Hasil:** Mengembalikan pesan sukses "user updated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

4. Fungsi DELETE - Menghapus Pengguna

Digunakan untuk menghapus akun dari sistem:

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

B. route.js



The screenshot shows a code editor with the file 'route.js' open. The file is located in a directory structure for a 'users' API endpoint. The code implements a GET request handler that first checks for authentication using 'requireAuth'. If successful, it uses 'RateLimiter' to prevent abuse. Then, it retrieves all users from the database using 'prisma.user.findMany'. Finally, it returns a JSON response with status 200 and a success message. If any error occurs, it returns a 500 internal server error.

```
File: route.js
app > api > users > JS route.js
1 import {prisma} from "../../prisma"
2 import {requireAuth} from "./server"
3 import { RateLimiter } from "rate-limiter"
4 import { requireAuth } from "../../lib/auth"
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

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

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

Penjelasan :

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

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

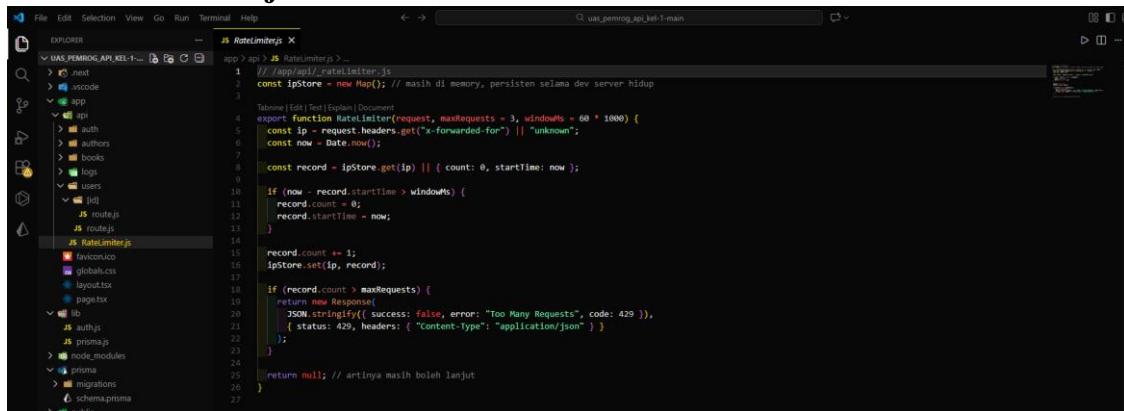
- **requireAuth**: Middleware wajib untuk memastikan bahwa data daftar pengguna hanya bisa diakses oleh orang yang sudah login.
- **RateLimiter**: Mencegah serangan *denial-of-service* atau *scraping* data user secara massal.

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

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

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

C. RateLimiter.js



```
app > api > JS RateLimiter.js
1 // app/api/ratelimiter.js
2 const ipStore = new Map(); // masih di memory, persisten selama dev server hidup
3
4 function RateLimiter(request, maxRequests = 3, windowMs = 60 * 1000) {
5   const ip = request.headers.get("x-forwarded-for") || "unknown";
6   const now = Date.now();
7
8   const record = ipStore.get(ip) || { count: 0, startTime: now };
9
10  if (now - record.startTime > windowMs) {
11    record.count = 0;
12    record.startTime = now;
13  }
14
15  record.count += 1;
16  ipStore.set(ip, record);
17
18  if (record.count > maxRequests) {
19    return new Response(
20      JSON.stringify({ success: false, error: "Too Many Requests", code: 429 }),
21      { status: 429, headers: { "Content-Type": "application/json" } }
22    );
23  }
24
25  return null; // artinya masih boleh lanjut
26}
27
```

Penjelasan :

1. Inisialisasi Penyimpanan (Baris 1 - 2)

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

2. Parameter Fungsi (Baris 4)

Fungsi RateLimiter memiliki aturan default sebagai berikut:

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

3. Identifikasi Pengguna (Baris 5 - 8)

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

4. Logika Reset Waktu (Baris 10 - 13)

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

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

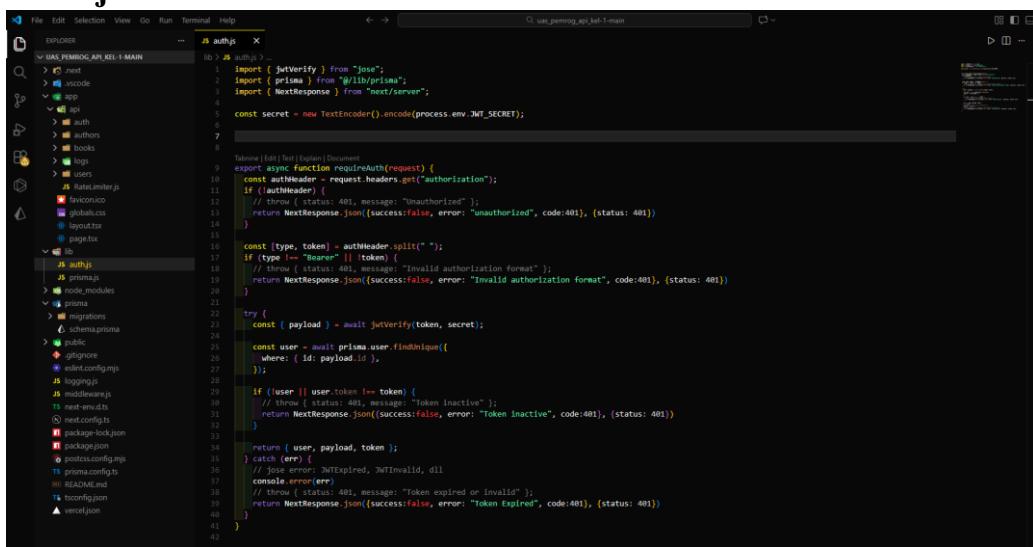
- Setiap kali pengguna mengakses API, nilai record.count akan bertambah 1.
- **Pengecekan Pelanggaran**: Jika count sudah melebihi 3, sistem akan langsung menolak permintaan.
- **Respon Error 429**: Jika ditolak, sistem mengirimkan status **429 (Too Many Requests)** beserta pesan error dalam format JSON.

6. Izin Akses (Baris 25)

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

4. lib/...

A. auth.js



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

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

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

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

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

    if (!user || user.tokens[0] !== token) {
      // throw { status: 401, message: "Token inactive" };
      return NextResponse.json({ success: false, error: "Token inactive", code: 401 }, { status: 401 });
    }

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

Penjelasan :

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

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

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

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

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

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

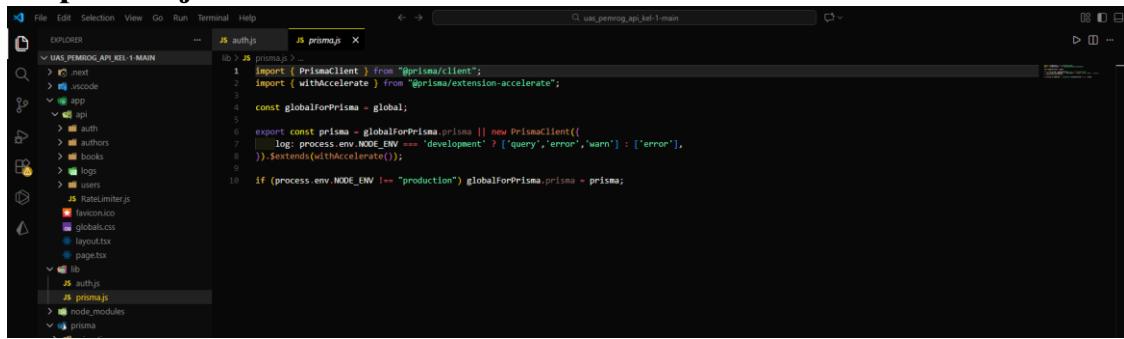
4. Sinkronisasi Sesi Database (Baris 29 - 33)

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

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

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

B. prisma.js



The screenshot shows a code editor with the file 'prisma.js' open. The file contains the following code:

```
lib > JS prisma.js
1 import { PrismaClient } from "@prisma/client";
2 import { withAccelerate } from "@prisma/extension-accelerate";
3
4 const globalForPrisma = global;
5
6 export const prisma = globalForPrisma.prisma || new PrismaClient({
7   log: process.env.NODE_ENV === "development" ? ["query", "error", "warn"] : ["error"],
8 }).$extends(withAccelerate());
9
10 if (process.env.NODE_ENV !== "production") globalForPrisma.prisma = prisma;
```

The code imports PrismaClient and withAccelerate from the @prisma/client and @prisma/extension-accelerate packages respectively. It then creates a global variable globalForPrisma and defines a prisma variable as either the existing global variable or a new PrismaClient instance. The PrismaClient is configured to log queries, errors, and warnings only in development mode. Finally, it checks if the environment is not production and updates the global prisma variable.

Penjelasan :

. Inisialisasi dan Kunci Rahasia (Baris 1 - 5)

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

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

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

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

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

4. Sinkronisasi Sesi Database (Baris 29 - 33)

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

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

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

5. prisma/migrations/schema.prisma

```
File Edit Selection View Go Run Terminal Help ← → C:\Users\pmling\git\api-test-main
prisma schema.prisma > ...
1
2 generator client {
3   provider = "prisma-client"
4   output = ".*/app/generated/prisma"
5   provider = "prisma-client-rs"
6   prismaTargets = ["native", "rhel-openssl-3.0.x"]
7 }
8
9 datasource db {
10   provider = "postgresql"
11   url = env("DATABASE_URL")1 The datasource property "url" is no longer supported in schema files. Move connection URLs for Migrate to "prisma.config.ts"
12 }
13
14 enum Role {
15   USER
16   ADMIN
17 }
18
19 model User {
20   id Int @id @default(autoincrement())
21   name String
22   email String @unique
23   role Role @default(USER)
24   password String
25   token String
26   createdAt Datetime @default(now())
27 }
28
29 model Author {
30   id Int @id @default(autoincrement())
31   name String
32 }
33
34 book Book []
35
36 model Book {
37   id Int @id @default(autoincrement())
38   title String
39   publishedYear Int
40   publisher String
41   createdAt Datetime @default(now())
42
43   authorId Int?
44
45   author Author? @relation(fields: [authorId], references: [id])
46 }
47
48
49 createdAt Datetime @default(now())
50
51 authorId Int?
52
53 author Author? @relation(fields: [authorId], references: [id])
54
55
56 model Log {
57   id Int @id @default(autoincrement())
58   method String
59   path String
60   userRef String
61   role String
62   createdAt Datetime @default(now())
63 }
```

Penjelasan :

1. Konfigurasi Generator & Datasource (Baris 1 - 13)

- **generator client**: Menginstruksikan Prisma untuk membuat "Prisma Client", yaitu alat yang Anda gunakan di kode route.js (seperti prisma.user.findMany) agar bisa berinteraksi dengan database menggunakan bahasa Javascript/Typescript.
 - **datasource db**: Menentukan bahwa database yang digunakan adalah **PostgreSQL**.
 - **url = env("DATABASE_URL")**: Alamat koneksi database diambil dari file lingkungan (.env) agar informasi sensitif seperti password database tidak tersebar di kode program.

2. Definisi Enum Role (Baris 14 - 17)

- **enum Role:** Mendefinisikan pilihan tetap untuk peran pengguna, yaitu **USER** atau **ADMIN**. Ini memastikan tidak ada peran lain di luar dua pilihan tersebut yang bisa masuk ke database.

3. Model User - Tabel Pengguna (Baris 19 - 27)

Ini adalah tabel yang menyimpan data akun:

- **id**: Kunci utama (Primary Key) yang bertambah otomatis setiap ada user baru.
- **email**: Diberi label `@unique`, artinya tidak boleh ada dua pengguna dengan email yang sama.
- **role**: Menggunakan pilihan dari enum Role dengan nilai default adalah USER.
- **token**: Ditandai dengan tanda tanya (String?), artinya kolom ini bersifat **opsional** (bisa kosong/null), yang digunakan untuk menyimpan JWT saat login.

4. Model Author & Book - Relasi Data (Baris 29 - 49)

Bagian ini menjelaskan hubungan antara Penulis dan Buku:

- **Model Author**: Menyimpan nama penulis dan memiliki daftar buku (book Book[]).
- **Model Book**: Menyimpan informasi detail buku seperti judul, tahun terbit, dan penerbit.
- **Relasi (One-to-Many)**: Seorang penulis bisa memiliki banyak buku. Hal ini dihubungkan melalui **authorId** di tabel Book yang merujuk pada **id** di tabel Author.

5. Model Log - Pencatatan Aktivitas (Baris 51 - 58)

Digunakan untuk keperluan audit atau pemantauan sistem:

- Menyimpan metode HTTP yang digunakan (method), alamat URL yang diakses (pathname), serta ID dan peran pengguna yang melakukan aksi tersebut.
- **createdAt**: Mencatat waktu otomatis saat log dibuat menggunakan fungsi now().

6. logging.js



```
/* logging.js */  
const pathname = request.nextUrl.pathname;  
const method = request.method;  
  
console.info(`[API LOG] ${method} ${pathname}`);  
  
if (payload) {  
    console.info(`[USER] ${payload.id} ${payload.role ? `-${payload.role}` : ''}`)  
};  
else {  
    console.info(`[USER] anonymous`);  
}  
  
const data = {  
    method,  
    pathname,  
    UserId: payload.id != null ? String(payload.id) : null,  
    role: payload.role ?? null,  
};  
  
// 🔥 FIRE & FORGET (Tdisk await)  
fetch(`${process.env.NEXT_PUBLIC_BASE_URL}/api/logs`, {  
    method: "POST",  
    headers: {  
        "Content-Type": "application/json",  
        "x-log-secret": process.env.LOG_SECRET,  
    },  
    body: JSON.stringify(data),  
}).catch((e) => {});
```

Penjelasan :

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

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

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

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

3. Persiapan Data Log (Baris 16 - 21)

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

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

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

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

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

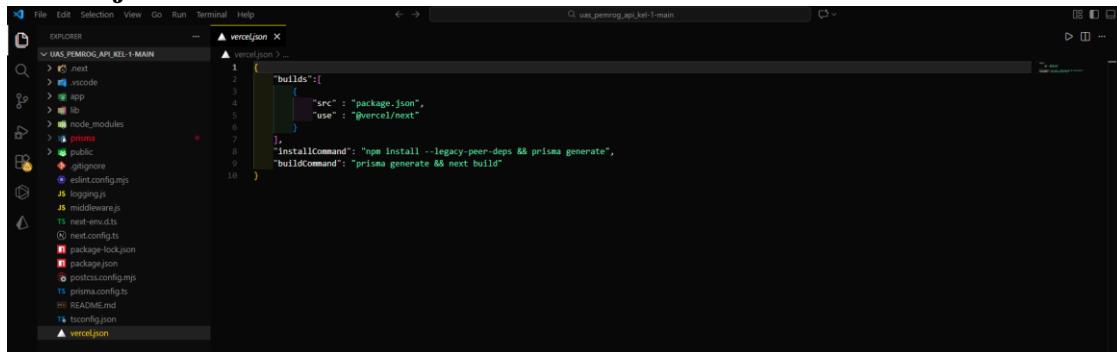
7. middleware.js

```
File Edit Selection View Go Run Terminal Help
EXPLORER -- JS middleware.js
1 import { NextResponse } from "next/server";
2 import { jwtVerify } from "jose";
3 import { logging } from "/logging";
4
5 // 1. Cek header authorization
6 export async function middleware(request) {
7   const authHeader = request.headers.get("authorization");
8
9   // 1.1: Jika header tidak ada
10  If (!authHeader) {
11    return new NextResponse(
12      JSON.stringify({ success: false, error: "Unauthorized", code: 401 }),
13      { status: 401 }
14    );
15  }
16
17  // 2. Format Bearer
18  const [type, token] = authHeader.split(" ");
19
20  If (type !== "Bearer" || !token) {
21    return new NextResponse(
22      JSON.stringify({ success: false, error: "Invalid authorization format", code: 401 }),
23      { status: 401 }
24    );
25  }
26
27  try {
28    const secret = new TextEncoder().encode(process.env.JWT_SECRET);
29
30    const [payload] = await jwtVerify(token, secret);
31
32    // * Logging TAPI side effect lain
33    logging(request);
34
35    const pathname = request.nextUrl.pathname;
36    const method = request.method;
37
38    if (pathname.startsWith("/api/users")) {
39      // Tambahkan logika untuk memeriksa role user di sini
40      if (payload.role === "ADMIN") {
41        return new NextResponse(
42          JSON.stringify({ success: false, error: "Forbidden", code: 403 }),
43          { status: 403 }
44        );
45      }
46
47      if (pathname.startsWith("/api/books") && method === "DELETE") {
48        if (payload.role !== "ADMIN") {
49          return new NextResponse(
50            JSON.stringify({ success: false, error: "Forbidden", code: 403 }),
51            { status: 403 }
52          );
53        }
54
55        return NextResponse.next();
56
57      } catch (error) {
58        console.error(error);
59        return new NextResponse(
60          JSON.stringify({ success: false, error: "Invalid or expired token", code: 401 }),
61          { status: 401 }
62        );
63      }
64    }
65
66    export const config = {
67      matcher: ["/api/books/:path*", "/api/authors/:path*", "/api/users/:path*"]
68    }
69  }
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
209
```

4. Konfigurasi Matcher (Baris 66 - 69)

- **matcher:** Menentukan rute mana saja yang harus melewati "satpam" middleware ini.
- Berdasarkan kode Anda, middleware ini aktif untuk semua rute di bawah:
 - /api/books/:path*
 - /api/authors/:path*
 - /api/users/:path*.

8. vercel.json



```
1 {
2   "builds": [
3     {
4       "src": "package_json",
5       "use": "@vercel/next"
6     }
7   ],
8   "installCommand": "npm install --legacy-peer-deps && prisma generate",
9   "buildCommand": "prisma generate && next build"
10 }
```

Penjelasan :

1. Proses Autentikasi Login (POST)

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

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

2. Sinkronisasi Sesi Database

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

9. README.md

```
File Edit Selection View Go Run Terminal Help Search C:\>Users>M Si>AppData>Local>Packages>5319275AWhatsAppDesktop_cv1g1gvymjm>LocalStorage>sessions>2443f777fb8ee23d7e091143f9ad4c57cd5d0b1>transfers>2025-52>readme.md

1 #!/usr/bin/env node
2 /**
3  * This file contains environment variables for the application.
4  */
5 
6 // Environment Variables
7 
8 // Database URL
9 DATABASE_URL="postgresql://neondb_owner:pg_2myUH0xplnjeP-wispy-field-ais7w3s1-pooler.ap-southeast-1.aws.neon.tech/neondb?sslmode=require"
10 
11 // JWT Secret
12 JWT_SECRET="ini-rahasia-milik-badan-intelejen-cia"
13 
14 // Next.js Public Base URL
15 NEXT_PUBLIC_BASE_URL="https://uas-pemrog-api-kel-1.vercel.app"
16 
17 // Log Secret
18 LOG_SECRET="ini-rahasia-elit-global"
19 
20 // Note
21 note:
22 1. setelah url endpoint yang sudah disiapkan pada folder app next js
23 2. file migrasi terdapat pada prisma silahkan dicoba
24 
25 // Tugas
26 tugas :
27 1. refactor kode yang sudah ada
28 2. cek setup fungsionalitas api
29 
30 
31 
```

Penjelasan :

Bagian 1: Variabel Lingkungan (.env)

Bagian ini menyimpan kunci rahasia yang tidak boleh dipublikasikan ke GitHub agar aplikasi tetap aman.

- **Baris 1 (DATABASE_URL):** Ini adalah *connection string* untuk menghubungkan aplikasi dengan database PostgreSQL yang dihosting di **Neon.tech** (AWS region ap-southeast-1).
 - Mengandung informasi: Protokol (postgresql), username (neondb_owner), password, host server, dan nama database (neondb).
- **Baris 3 (JWT_SECRET):** Kunci rahasia yang digunakan oleh library jose atau jsonwebtoken untuk menandatangani (signing) token login pengguna.
 - Jika kunci ini bocor, orang lain bisa memalsukan identitas admin/user di aplikasi Anda.
- **Baris 5 (NEXT_PUBLIC_BASE_URL):** Alamat dasar (base URL) aplikasi saat sudah dideploy ke **Vercel**.
 - Awalan NEXT_PUBLIC_ menandakan variabel ini bisa diakses oleh sisi Client (browser), bukan hanya server.
- **Baris 7 (LOG_SECRET):** Kunci keamanan khusus yang digunakan oleh sistem logging.js dan API logs.
 - Ini memastikan hanya aplikasi internal Anda yang bisa mengirimkan data aktivitas ke database log.

Bagian 2: Akses & Repositori

- **Baris 10-11 (URL API):** Endpoint utama untuk berinteraksi dengan sistem adalah <https://uas-pemrog-api-kel-1.vercel.app/api/>.
- **Baris 13-14 (GitHub):** Alamat repositori kode sumber. Status "private" menunjukkan bahwa kode ini dilindungi dan hanya bisa diakses oleh tim pengembang.

Bagian 3: Instruksi Operasional (Note)

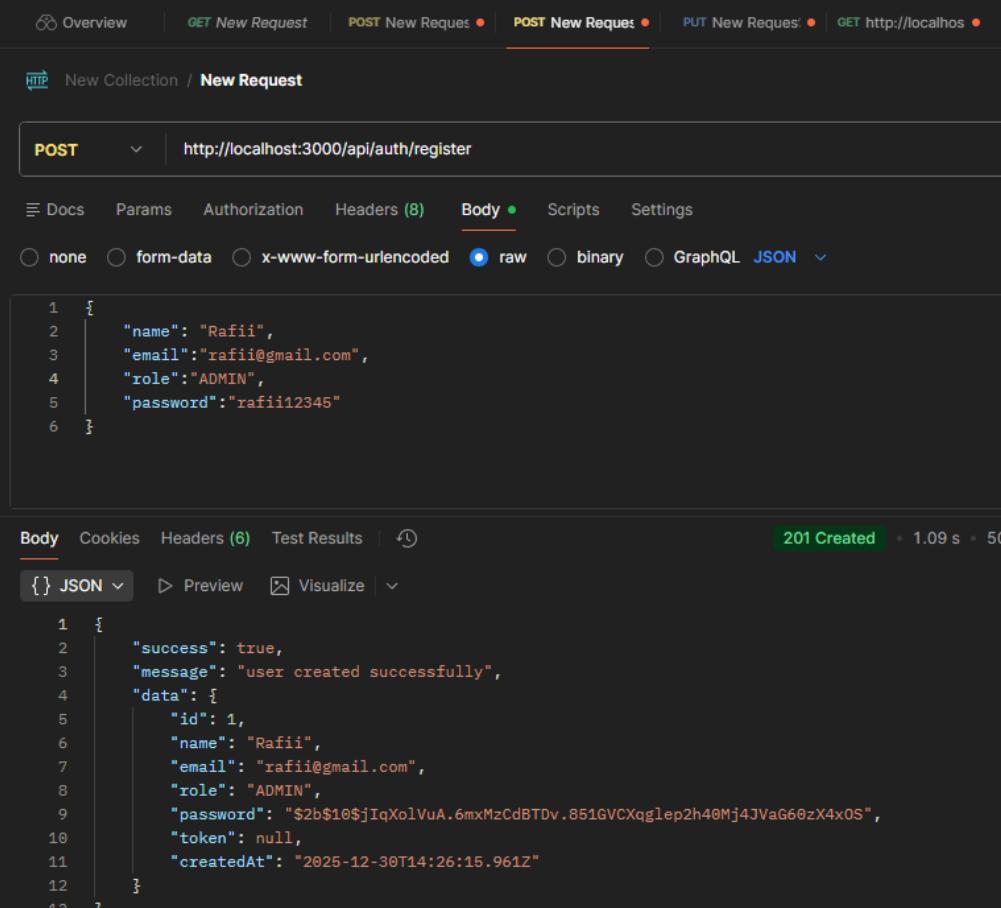
- **Baris 17-18 (Endpoint):** Menjelaskan bahwa struktur API mengikuti struktur folder di dalam folder app Next.js (App Router).
- **Baris 19-20 (Migrasi):** Memberitahu pengembang bahwa definisi tabel sudah ada di folder prisma.
 - Untuk menjalankannya, Anda perlu menggunakan perintah npx prisma migrate dev agar tabel User, Author, Book, dan Log terbentuk di Neon PostgreSQL.

Bagian 4: Rencana Tugas (Task)

- **Baris 22-24 (Refactor):** Perintah untuk membersihkan kode, memperbaiki *typo* (seperti pesan error "token tidak invalid"), dan mengoptimalkan logika agar lebih efisien.
- **Baris 25-26 (Fungsionalitas):** Perintah untuk melakukan testing pada setiap metode (GET, POST, PUT, DELETE) di semua endpoint (auth, authors, books, users, logs).

2.2 HASIL TESTING

1. Register



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/auth/register
- Body:** Raw JSON (selected)


```

1 {
2   "name": "Rafii",
3   "email": "rafi@gmail.com",
4   "role": "ADMIN",
5   "password": "rafi12345"
6 }
```
- Headers:** (8 items listed)
- Response:**
 - Status: 201 Created
 - Time: 1.09 s
 - Size: 50
 - Content-Type: application/json
 - Content-Length: 130
 - Connection: keep-alive
 - Date: Mon, 25 Dec 2023 14:26:15 GMT
 - X-Powered-By: Express

```

1 {
2   "success": true,
3   "message": "user created successfully",
4   "data": {
5     "id": 1,
6     "name": "Rafii",
7     "email": "rafi@gmail.com",
8     "role": "ADMIN",
9     "password": "$2b$10$jIqXolVuA.6mxMzCdBTdV.851GVCXqglep2h40Mj4JVaG60zX4xOS",
10    "token": null,
11    "createdAt": "2025-12-30T14:26:15.961Z"
12  }
13 }
```

Penjelasan:

Dari gambar diatas menunjukkan hasil pengujian fungsionalitas Registrasi Pengguna Baru menggunakan REST Client. Pengujian ini bertujuan untuk memverifikasi apakah sistem mampu menerima data pengguna baru, menyimpannya ke dalam database PostgreSQL, serta menerapkan enkripsi pada password sesuai spesifikasi keamanan yang dirancang.

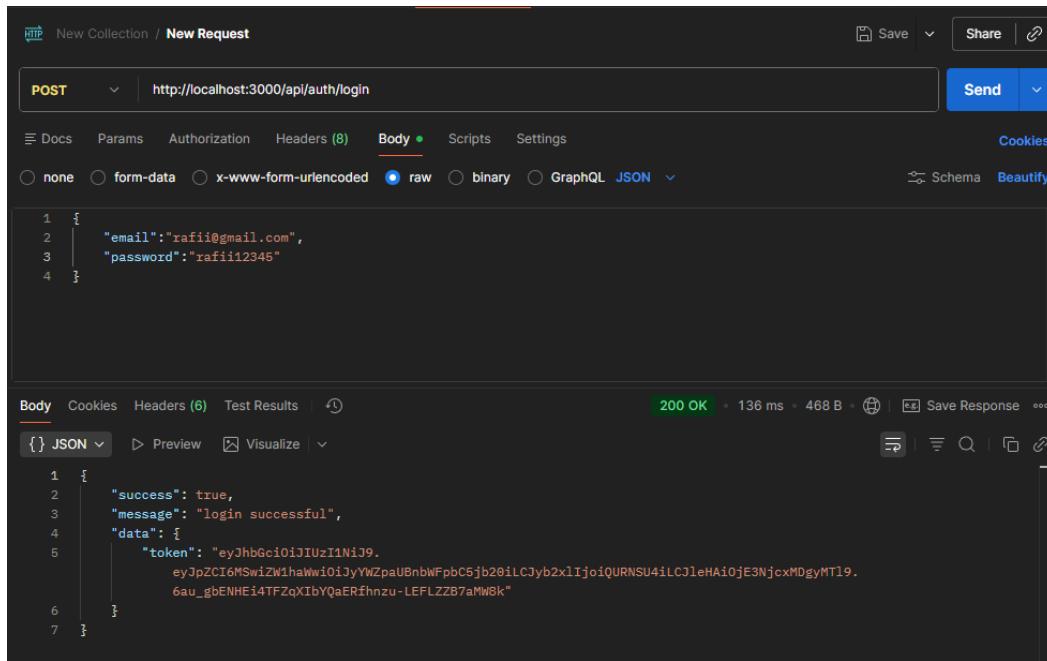
Detail Pengujian:

- Endpoint: <http://localhost:3000/api/auth/register>
- Method: POST
- Payload (Body): Data dikirim dalam format JSON yang berisi name, email, role, dan password (plaintext).

Analisis Hasil:

- Status Code 201 (Created): Server mengembalikan status HTTP 201, yang menandakan bahwa permintaan berhasil diproses dan resource baru (data user) telah sukses dibuat di dalam database.
- Keamanan Password (Bcrypt): Pada bagian Request Body, password yang dikirimkan adalah teks biasa ("rafii12345"). Namun, pada Response Body, field password yang dikembalikan telah berubah menjadi string acak yang diawali dengan \$2b\$10\$.... Hal ini membuktikan bahwa implementasi password hashing menggunakan bcrypt telah berfungsi dengan baik. Password asli tidak disimpan secara mentah di database, menjaga keamanan akun pengguna.
- Validasi Role-Based Authorization (RBAC): Sistem berhasil memproses input role: "ADMIN". Dalam arsitektur aplikasi ini, penetapan peran sangat krusial karena menentukan hak akses terhadap endpoint selanjutnya.
 - Peran ADMIN: Akun yang didaftarkan pada pengujian di atas akan memiliki hak istimewa (privileged access) untuk melakukan mutasi data, yaitu mengakses endpoint POST untuk menambah data Buku (Books) dan data Penulis (Authors).
 - Peran USER: Sebaliknya, jika registrasi dilakukan dengan role: "USER", akun tersebut akan dibatasi hanya pada metode GET (Read-only), yang artinya hanya diperbolehkan untuk melihat daftar buku dan detail penulis tanpa izin untuk mengubah data.
- Response Data: Respons JSON mengembalikan objek user yang baru dibuat beserta id dan createdAt, serta pesan konfirmasi "user created successfully", yang memudahkan sisi frontend untuk memvalidasi keberhasilan proses.

2. Login



Penjelasan:

Gambar di atas memperlihatkan proses pengujian autentikasi pengguna menggunakan REST Client. Pengujian ini bertujuan untuk memverifikasi apakah sistem mampu memvalidasi kredensial pengguna (email dan password) yang telah terdaftar serta menerbitkan JSON Web Token (JWT) sebagai tanda otorisasi yang sah.

Detail Pengujian:

- Endpoint: <http://localhost:3000/api/auth/login>
- Method: POST
- Payload (Body): Mengirimkan kredensial akun yang baru dibuat, yaitu email ("rafi@gmail.com") dan password ("rafi12345").

Analisis Hasil:

- Verifikasi Kredensial (Status 200 OK): Server mengembalikan status HTTP 200, yang menandakan bahwa proses login berhasil. Secara internal, sistem telah melakukan pencocokan antara email yang dikirim dengan data di database, serta memverifikasi kecocokan password (plaintext) dengan hash password yang tersimpan menggunakan algoritma bcrypt.
- Penerbitan JWT (Bearer Token): Pada bagian data di respons JSON, terdapat properti token yang berisi string panjang diawali dengan eyJ.... Ini adalah JWT (JSON Web Token) yang telah di-sign oleh server. Token ini membungkus informasi penting (payload) seperti User ID dan Role (dalam kasus ini: ADMIN) secara terenkripsi.

- Mekanisme Stateless Authentication: Keberadaan token ini krusial karena aplikasi menggunakan arsitektur stateless. Server tidak menyimpan sesi login di memori. Sebagai gantinya, token ini akan digunakan oleh sisi client (Frontend/Postman) sebagai "kunci akses". Untuk setiap permintaan berikutnya (seperti Menambah Buku atau Mengedit Author), token ini wajib disertakan di dalam Header Authorization dengan format Bearer <token>.
- Keamanan Hak Akses: Karena akun yang login adalah akun dengan role ADMIN (berdasarkan pengujian registrasi sebelumnya), token yang dihasilkan ini secara implisit membawa izin administrator. Sistem Middleware nantinya akan mendekode token ini untuk mengizinkan akses ke rute-rute yang diproteksi khusus Admin.

3. Tambah Data Author

The image consists of two vertically stacked screenshots of the Postman application interface.

Screenshot 1 (Top): This screenshot shows a POST request to the URL `http://localhost:3000/api/authors/`. The request body contains the following JSON:

```

1 {
2   "name": "Daizuke Aizawa"
3 }

```

The response status is 201 Created, with a response time of 153 ms and a response size of 351 B. The response body is:

```

1 {
2   "success": true,
3   "message": "author created successfully",
4   "data": {
5     "id": 1,
6     "name": "Daizuke Aizawa"
7   }
8 }

```

Screenshot 2 (Bottom): This screenshot shows the same POST request to `http://localhost:3000/api/authors/`, but with the Authorization tab selected. The Auth Type is set to "Bearer Token", and the Token field contains the value `4WJ3SL4ypNbQLC5mEjmFqalqbZDE`.

Penjelasan:

Dari gambar diatas, sebelum dapat menambahkan data buku, sistem mengharuskan adanya data Penulis (*Author*) terlebih dahulu. Hal ini dikarenakan skema basis data menggunakan prinsip *Relational Integrity*, di mana setiap buku

harus terhubung dengan ID penulis yang valid. Pengujian ini dilakukan oleh pengguna dengan peran ADMIN.

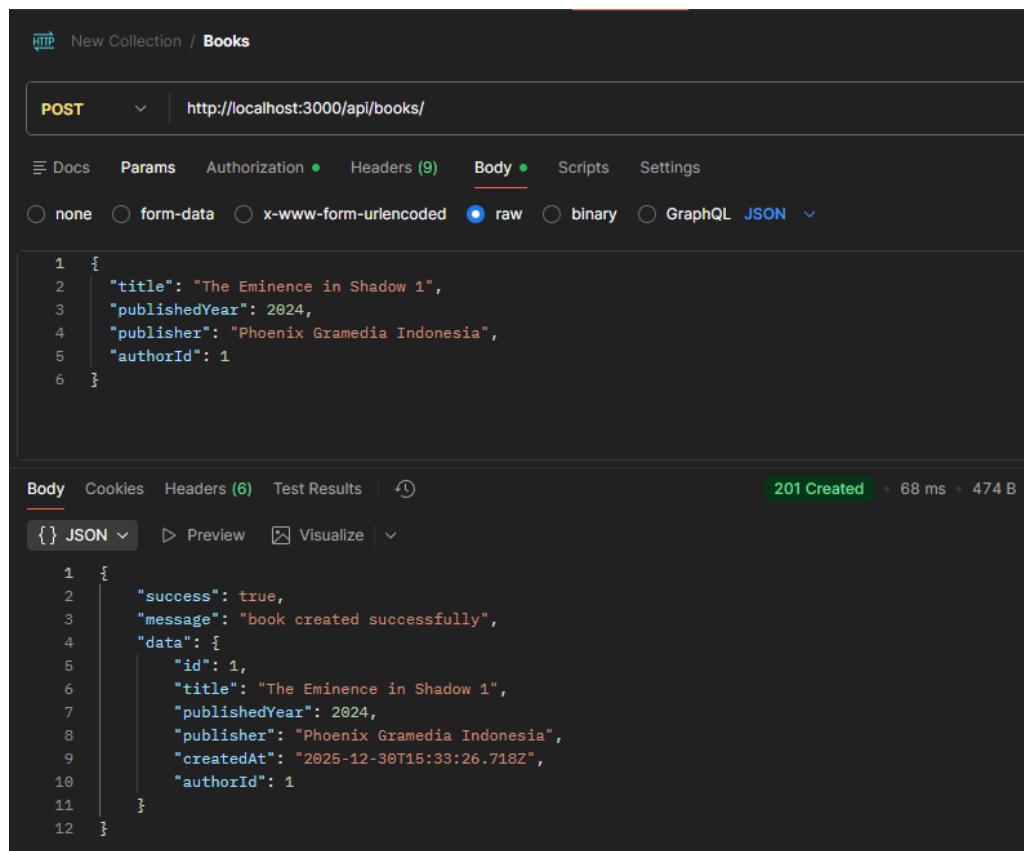
Detail Pengujian:

- Endpoint: <http://localhost:3000/api/authors/>
- Method: POST
- Authorization: Bearer Token (Token Admin dari login sebelumnya).

Analisis Hasil:

- Validasi Keamanan dan Middleware (Authorization): Seperti yang terlihat pada tangkapan layar (Tab Authorization), permintaan ini menyertakan Bearer Token pada Header Authorization. Token ini adalah JWT yang didapatkan saat proses login.
 - Fungsi Middleware: Sebelum data diproses, middleware di sisi server memverifikasi validitas token tersebut. Selain itu, middleware juga memastikan bahwa token tersebut milik pengguna dengan role ADMIN. Jika pengguna biasa (User) mencoba mengakses endpoint ini, akses akan ditolak (Forbidden), menjaga integritas data referensi perpustakaan.
- Pemrosesan Data Referensi: Pada bagian Body, dikirimkan data JSON sederhana berisi { "name": "Daizuke Aizawa" }. Server menerima input ini dan menggunakan Prisma ORM untuk membuat entitas baru di tabel Author.
- Respons Sistem dan Auto-Increment ID: Server mengembalikan status HTTP 201 Created beserta objek data penulis yang baru dibuat.
 - Catatan: Respons mengembalikan "id": 1. ID ini dihasilkan secara otomatis (*Auto-increment*) oleh database PostgreSQL. Nilai **ID inilah yang nantinya wajib digunakan** sebagai *Foreign Key* saat admin ingin menambahkan buku yang ditulis oleh penulis tersebut.

4. Tambah Data Buku



The screenshot shows a POST request to `http://localhost:3000/api/books/`. The request body contains the following JSON payload:

```
1 {
2   "title": "The Eminence in Shadow 1",
3   "publishedYear": 2024,
4   "publisher": "Phoenix Gramedia Indonesia",
5   "authorId": 1
6 }
```

The response status is `201 Created` with a response time of 68 ms and a size of 474 B. The response body is:

```
1 {
2   "success": true,
3   "message": "book created successfully",
4   "data": {
5     "id": 1,
6     "title": "The Eminence in Shadow 1",
7     "publishedYear": 2024,
8     "publisher": "Phoenix Gramedia Indonesia",
9     "createdAt": "2025-12-30T15:33:26.718Z",
10    "authorId": 1
11  }
12 }
```

Penjelasan:

Setelah data referensi penulis tersedia, pengujian dilanjutkan dengan menambahkan data buku ke dalam sistem. Pengujian ini memvalidasi kesesuaian tipe data (*Type Safety*) dan integritas relasi (*Referential Integrity*) antar tabel dalam database.

Detail Pengujian:

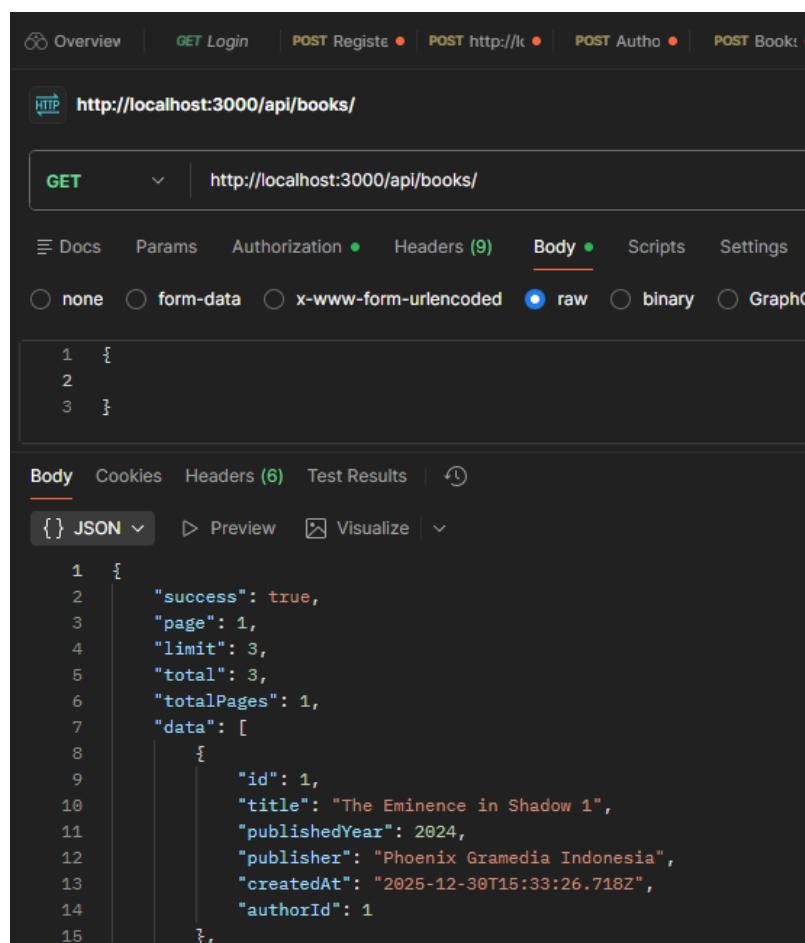
- Endpoint: <http://localhost:3000/api/books/>
- Method: POST
- Payload (Body): Mengirimkan data buku dengan format JSON yang telah disesuaikan dengan skema Prisma, khususnya pada tipe data integer.

Analisis Hasil

- Validasi Tipe Data (Schema Validation): Berbeda dengan percobaan awal yang mengalami kegagalan validasi, pada pengujian ini data `publishedYear` dikirim sebagai Integer (2024) dan bukan String. Hal ini sesuai dengan definisi model di `schema.prisma`. Keberhasilan server memproses data ini menunjukkan bahwa sistem validasi input di sisi backend berfungsi efektif dalam mencegah masuknya data yang tidak sesuai format.

- Implementasi Relasi (Foreign Key Association): Field authorId: 1 yang dikirimkan dalam request body berfungsi sebagai Foreign Key. Sistem berhasil menghubungkan buku "The Eminence in Shadow 1" dengan penulis "Daisuke Aizawa" (yang memiliki ID 1). Jika ID penulis tidak ditemukan di database, Prisma ORM akan menolak permintaan ini. Kesuksesan penyimpanan data membuktikan bahwa relasi One-to-Many antara entitas Author dan Book telah terjalin dengan benar.
- Persistensi Data: Server mengembalikan respons dengan status HTTP 201 Created dan menampilkan objek buku yang baru disimpan lengkap dengan id dan createdAt. Ini mengonfirmasi bahwa data telah tersimpan secara permanen di database PostgreSQL.

5. Pengujian Endpoint Melihat Daftar Buku (GET)



The screenshot shows the Postman application interface. At the top, there's a navigation bar with links for Overview, Login, Register, Auth, and Books. Below that is a header bar with an 'HTTP' icon and the URL 'http://localhost:3000/api/books/'. The main area has a 'GET' method selected and the same URL. Below the method, there are tabs for Docs, Params, Authorization, Headers (9), Body (selected), Scripts, and Settings. Under 'Body', the 'raw' option is chosen, and the JSON response is displayed:

```

1  {
2
3  }

```

Below the JSON response, there are tabs for Body, Cookies, Headers (6), Test Results, and a preview section. The preview section shows the JSON response again:

```

1  {
2      "success": true,
3      "page": 1,
4      "limit": 3,
5      "total": 3,
6      "totalPages": 1,
7      "data": [
8          {
9              "id": 1,
10             "title": "The Eminence in Shadow 1",
11             "publishedYear": 2024,
12             "publisher": "Phoenix Gramedia Indonesia",
13             "createdAt": "2025-12-30T15:33:26.718Z",
14             "authorId": 1
15         },

```

Penjelasan:

Setelah data buku tersedia di database, dilakukan pengujian fitur Read (Melihat Data). Sesuai dengan skenario hak akses, pengujian ini dilakukan menggunakan akun dengan peran USER. Tujuannya adalah memastikan bahwa pengguna umum dapat mengakses katalog buku namun tetap dalam batasan hanya membaca (Read-only). Namun, seharusnya sama saja fungsinya untuk peran ADMIN dibagian GET ini.

Detail Pengujian:

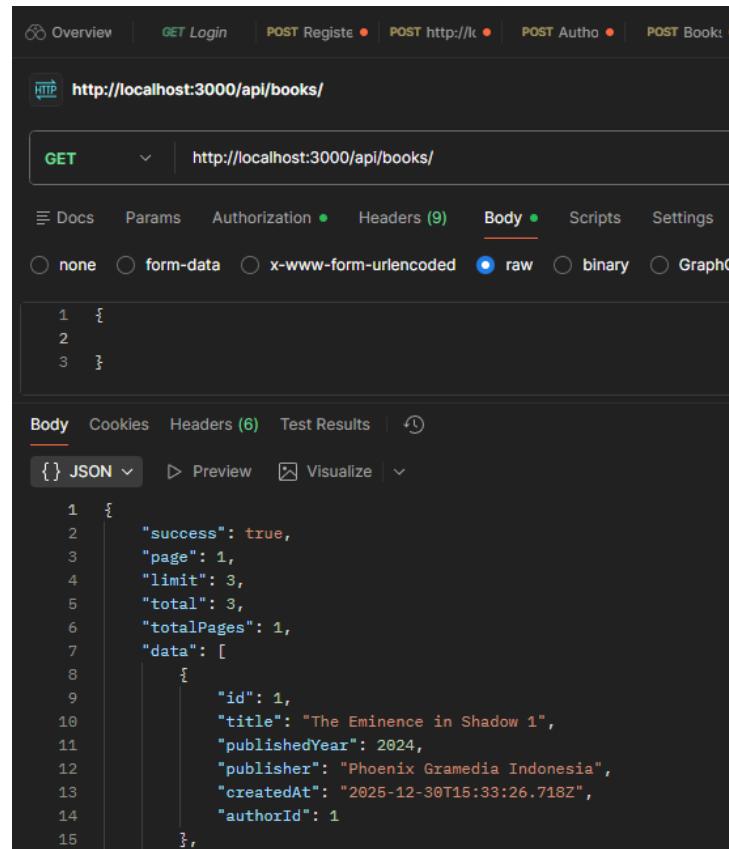
- Endpoint: <http://localhost:3000/api/books/>
- Method: GET
- Authorization: Bearer Token (Menggunakan Token akun User Biasa).

Analisis Hasil:

- Akses Data Sukses (Read Permission): Server memberikan respons dengan status sukses ("success": true). Hal ini membuktikan bahwa middleware mengizinkan role "USER" untuk mengakses metode GET. Berbeda dengan metode POST yang diblokir untuk pengguna biasa, metode ini bersifat publik atau terbuka bagi pengguna terdaftar untuk melihat koleksi perpustakaan.
- Implementasi Pagination: Respons API tidak hanya mengembalikan array data mentah, melainkan menyertakan metadata halaman (Pagination). Terlihat adanya properti "page": 1, "limit": 3, dan "total": 3.
 - Fitur ini sangat krusial untuk performa aplikasi. Jika jumlah buku mencapai ribuan, sistem tidak akan membebani *bandwidth* dengan mengirimkan semua data sekaligus, melainkan membaginya per halaman. Dalam pengujian ini, sistem berhasil mendeteksi total 3 buku yang tersimpan di database.
- Struktur Data Konsisten: Pada array data, objek buku ditampilkan lengkap dengan relasi penulisnya (authorId). Data yang ditampilkan (seperti "The Eminence in Shadow 1") konsisten dengan data yang sebelumnya diinput oleh Admin, memverifikasi bahwa proses pengambilan data (querying) dari database berjalan akurat.

6. Pengujian PUT

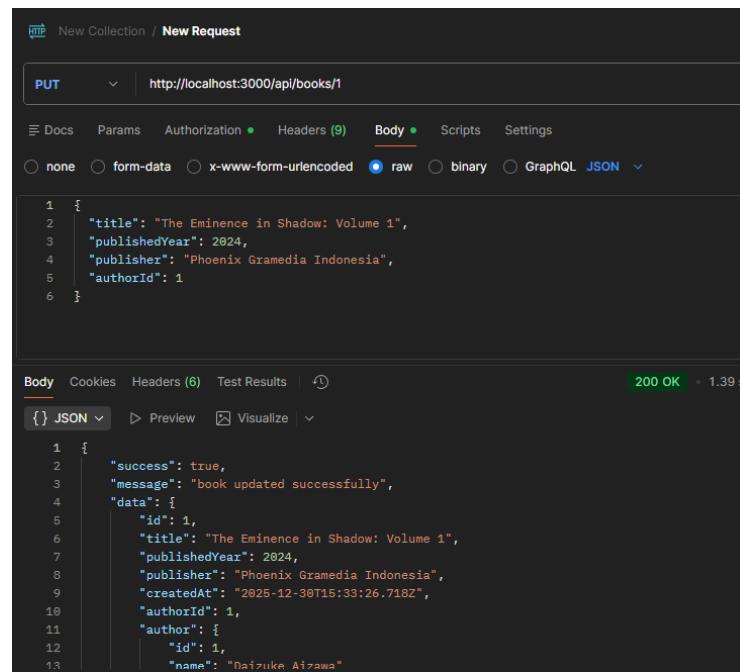
Berikut ini adalah judul buku sebelum diubah



The screenshot shows the Postman interface with a GET request to `http://localhost:3000/api/books/`. The response body is raw JSON:

```
1 {  
2     "success": true,  
3     "page": 1,  
4     "limit": 3,  
5     "total": 3,  
6     "totalPages": 1,  
7     "data": [  
8         {  
9             "id": 1,  
10            "title": "The Eminence in Shadow 1",  
11            "publishedYear": 2024,  
12            "publisher": "Phoenix Gramedia Indonesia",  
13            "createdAt": "2025-12-30T15:33:26.718Z",  
14            "authorId": 1  
15        },
```

Lalu sesudahnya seperti digambar di bawah ini:



The screenshot shows the Postman interface with a PUT request to `http://localhost:3000/api/books/1`. The request body is raw JSON:

```
1 {  
2     "title": "The Eminence in Shadow: Volume 1",  
3     "publishedYear": 2024,  
4     "publisher": "Phoenix Gramedia Indonesia",  
5     "authorId": 1  
6 }
```

The response status is 200 OK with a response time of 1.39 s. The response body is raw JSON:

```
1 {  
2     "success": true,  
3     "message": "book updated successfully",  
4     "data": {  
5         "id": 1,  
6         "title": "The Eminence in Shadow: Volume 1",  
7         "publishedYear": 2024,  
8         "publisher": "Phoenix Gramedia Indonesia",  
9         "createdAt": "2025-12-30T15:33:26.718Z",  
10        "authorId": 1,  
11        "author": {  
12             "id": 1,  
13             "name": "Daizuke Aizawa"  
14         }  
15     }  
16 }
```

Penjelasan:

Pengujian ini bertujuan untuk memverifikasi fitur manipulasi data, yaitu mengubah informasi buku yang sudah tersimpan. Fitur ini menggunakan metode HTTP PUT dan memanfaatkan *Dynamic Routing* untuk menargetkan data spesifik berdasarkan ID buku.

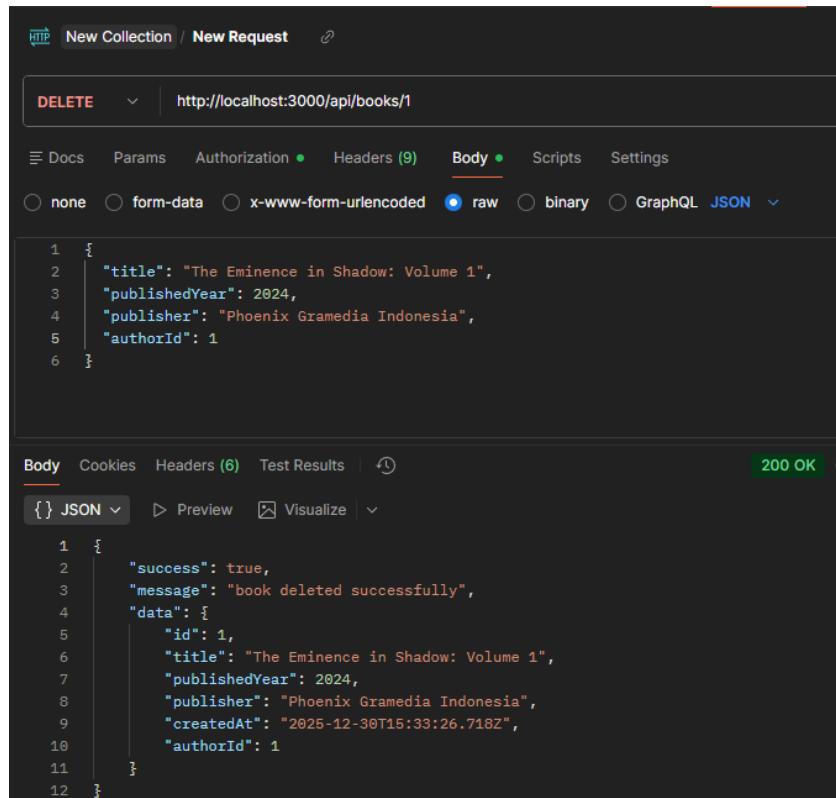
Detail Pengujian:

- Endpoint: `http://localhost:3000/api/books/1` (Angka 1 adalah ID Buku yang akan diedit).
- METHOD: PUT
- Authorization: Bearer Token
- Payload (Body): Data JSON baru yang ingin disimpan (misalnya revisi judul atau tahun terbit).

Analisis Hasil:

- Penerapan Dynamic Routing ([id]): Permintaan dikirim ke URL yang menyertakan ID buku (`/api/books/1`). Next.js menangkap parameter 1 ini melalui mekanisme Dynamic Routes pada folder [id]. Sistem kemudian mencari data buku dengan ID tersebut di database. Jika ID tidak ditemukan, sistem dirancang untuk mengembalikan respon 404 Not Found.
- Validasi Hak Akses (Role Protection): Sebelum perubahan data diproses, kode program melakukan validasi ganda:
 - Autentikasi: Memastikan pengguna memiliki token yang valid.
 - Otorisasi: Memastikan atribut role pada token adalah "ADMIN". Jika pengguna biasa (User) mencoba mengakses endpoint ini, sistem akan menolak permintaan dengan status 403 Forbidden. Keberhasilan proses update pada gambar membuktikan bahwa token Admin yang digunakan memiliki izin yang sah.
- Integritas Pembaruan Data (Prisma Update): Server menerima data revisi dari client (misalnya perubahan judul menjadi "Volume 1"). Prisma ORM menjalankan perintah update secara aman ke PostgreSQL. Respon status 200 OK beserta data terbaru yang dikembalikan menunjukkan bahwa sinkronisasi antara request client dan data fisik di database telah berhasil dilakukan.

7. Pengujian DELETE



The screenshot shows a Postman collection named "New Collection". A new request is being made to the endpoint `http://localhost:3000/api/books/1` using the `DELETE` method. The `Body` tab is selected, showing a raw JSON payload:

```
1 {
2   "title": "The Eminence in Shadow: Volume 1",
3   "publishedYear": 2024,
4   "publisher": "Phoenix Gramedia Indonesia",
5   "authorId": 1
6 }
```

Below the request, the response is displayed under the `Test Results` tab. The status is `200 OK`. The response body is:

```
1 {
2   "success": true,
3   "message": "book deleted successfully",
4   "data": {
5     "id": 1,
6     "title": "The Eminence in Shadow: Volume 1",
7     "publishedYear": 2024,
8     "publisher": "Phoenix Gramedia Indonesia",
9     "createdAt": "2025-12-30T15:33:26.718Z",
10    "authorId": 1
11  }
12 }
```

Penjelasan:

Tahap terakhir dalam pengujian manajemen data adalah memverifikasi fitur penghapusan (*Delete*). Fitur ini merupakan aksi destruktif yang bersifat permanen, sehingga memerlukan validasi hak akses yang ketat untuk mencegah hilangnya data penting oleh pihak yang tidak berwenang.

Detail Pengujian:

- Endpoint: <http://localhost:3000/api/books/1>
- Method: DELETE
- Authorization: Bearer Token (Token Admin).

Analisis Hasil:

- Validasi Otorisasi Tingkat Tinggi: Sistem menerima permintaan penghapusan pada ID buku 1. Sebelum mengeksekusi perintah, backend memvalidasi token JWT untuk memastikan pemohon memiliki role ADMIN. Sesuai dengan skenario keamanan yang dirancang, jika pengguna biasa (User) mencoba mengakses endpoint ini, sistem akan menolak akses tersebut. Keberhasilan operasi ini membuktikan bahwa mekanisme proteksi rute (Route Protection) berfungsi dengan baik.
- Penghapusan Data Permanen (Prisma Client): Sistem menggunakan perintah `prisma.book.delete` untuk mencari dan menghapus baris data yang sesuai

dengan ID yang diberikan. Berbeda dengan fitur Update yang memodifikasi nilai, fitur ini menghilangkan entitas buku beserta relasinya dari tabel basis data.

- Respon Balikan Data (Return Object): Server mengembalikan status HTTP 200 OK. Menariknya, respons JSON tetap mengembalikan objek data buku yang baru saja dihapus ("The Eminence in Shadow..."). Hal ini merupakan perilaku standar Prisma ORM yang berguna sebagai konfirmasi akhir kepada client mengenai detail data apa yang telah dihapus dari sistem.

BAB III

PENUTUP

Berdasarkan serangkaian pengujian yang telah dilakukan menggunakan REST Client (Postman), dapat disimpulkan bahwa API yang dibangun telah memenuhi seluruh spesifikasi teknis yang diminta:

1. Fungsionalitas CRUD: Sistem berhasil menangani operasi Create, Read, Update, dan Delete pada entitas Buku dan Penulis dengan alur data yang valid.
2. Keamanan (Security Layer):
 - Autentikasi: Implementasi JWT berhasil mengamankan akses, di mana pengguna tanpa token tidak dapat mengakses sumber daya yang dilindungi.
 - Otorisasi (RBAC): Pemisahan peran antara Admin dan User berjalan efektif. Admin memiliki kontrol penuh (CRUD), sedangkan User dibatasi hanya pada akses baca (Read-only).
3. Integritas Data: Validasi tipe data dan relasi antar tabel (Foreign Key) berhasil mencegah masuknya data sampah (garbage data) ke dalam sistem.

Dengan demikian, backend aplikasi ini dinyatakan siap untuk diintegrasikan dengan sisi frontend atau digunakan sebagai layanan mikro (microservice) yang mandiri.