PEMROGRAMAN API

REST API MENGGUNAKAN NEXT.JS

Dosen Pengampu:

Saiful Nur Budiman, M.Kom

Nama Kelompok:

Rizqi Harisma Uchrowi	**23104410001**
Dimas Akbar Maulana	**23104410056**
Angga Novantara	**24104410044**
Andrecaesar Setiawan	**23104410041**
Intan Lusiana	**23104410106**
Enjing Dwi Retno Wulan	**23104410049**

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ISLAM BALITAR

2025

Pendahuluan

Proyek ini merupakan pengembangan API backend menggunakan framework Next.js versi 16.0.5. Nama proyek adalah "project-api-nextjs" dengan versi 0.1.0. Tujuan proyek ini adalah membangun sebuah sistem API yang aman, dengan fitur autentikasi dan otorisasi, serta integrasi database menggunakan Prisma. Proyek ini dirancang untuk mendukung manajemen data seperti students dan users, dengan penekanan pada keamanan akses berdasarkan role pengguna (misalnya, role ADMIN).

Proyek ini dibuat sebagai bagian dari Ujian Akhir Semester (UAS) kelompok, yang menunjukkan penerapan konsep-konsep seperti server-side rendering, middleware untuk keamanan, dan konfigurasi lingkungan development. Semua file konfigurasi dan dependencies telah disesuaikan untuk mendukung pengembangan yang efisien dan skalabel.

Deskripsi Proyek

Proyek ini adalah aplikasi API berbasis Next.js yang fokus pada backend. Fitur utama meliputi:

1. Autentikasi dan Otorisasi: Menggunakan JWT (JSON Web Token) untuk verifikasi token pengguna.
2. Proteksi Route API: Middleware memastikan hanya pengguna terautentikasi yang dapat mengakses route tertentu, dan role ADMIN diperlukan untuk akses ke /api/users.
3. Integrasi Database: Menggunakan Prisma sebagai ORM untuk berinteraksi dengan database (misalnya PostgreSQL via adapter PG).
4. Rate Limiting: Untuk mencegah abuse API.
5. Validasi Data: Menggunakan Zod untuk schema validation.

Proyek ini private (tidak untuk publikasi langsung) dan dioptimalkan untuk environment development, build, dan production.

Teknologi dan Dependencies

Berdasarkan file package.json dan package-lock.json, berikut adalah daftar dependencies utama yang digunakan:

Dependencies Utama (Runtime):

Nama Package	Versi	Deskripsi
@prisma/adapter-pg	^7.1.0	Adapter Prisma untuk PostgreSQL.
@prisma/client	^7.0.1	Client untuk Prisma ORM, digunakan untuk query database.
bcryptjs	^3.0.3	Library untuk hashing password (keamanan autentikasi).
jose	^6.1.2	Library untuk JWT verification (menggantikan jsonwebtoken untuk performa lebih baik).
next	16.0.5	Framework utama untuk building API dan aplikasi web.
rate-limiter-flexible	^9.0.0	Untuk membatasi request API guna mencegah DDoS atau abuse.
react	19.2.0	Library UI (meskipun fokus backend, mungkin untuk komponen hybrid).
react-dom	19.2.0	DOM renderer untuk React.
zod	^4.1.13	Library untuk schema validation dan parsing data.

DevDependencies (Development):

Nama Package	Versi	Deskripsi
@prisma/adapter-pg	^7.1.0	Adapter Prisma untuk PostgreSQL.
@tailwindcss/postcss	^4	Plugin PostCSS untuk Tailwind CSS.
@types/node	^20	Type definitions untuk Node.js.
@types/react	^19	Type definitions untuk React.
@types/react-dom	^19	Type definitions untuk React DOM.
baseline-browser-mapping	^2.8.32	Mapping untuk compatibility browser.
dotenv	^17.2.3	Untuk loading environment variables dari .env.
eslint	^9	Tool untuk linting code.
eslint-config-next	16.0.5	Konfigurasi ESLint khusus Next.js.

prisma	^7.0.1	CLI Prisma untuk migrasi dan generate schema.
tailwindcss	^4	Framework CSS utility-first.
typescript	^5	Bahasa superset JavaScript untuk type safety.

Total packages terinstal: Lebih dari 100 (berdasarkan package-lock.json), dengan lockfile version 3 untuk konsistensi instalasi.

Struktur dan Konfigurasi Proyek

Proyek ini memiliki konfigurasi standar Next.js dengan penambahan custom untuk keamanan dan database. Berikut ringkasan file utama:

1. **middleware.js**:
    1. Fungsi middleware untuk memverifikasi token JWT pada route API tertentu (/api/students/* dan /api/users/*).
    2. Menggunakan jose untuk verifikasi token dengan secret dari environment variable JWT_SECRET.
    3. Menambahkan header x-user-role dari payload token.
    4. Memblokir akses jika token invalid atau role bukan ADMIN untuk /api/users.
    5. Error handling: Mengembalikan response JSON dengan status 401 atau 403 jika gagal.
2. **eslint.config.mjs**:
    1. Konfigurasi ESLint menggunakan eslint-config-next untuk core web vitals dan TypeScript.
    2. Override ignores untuk file seperti .next/**, out/**, dll., agar linting fokus pada source code.
3. **.env.example**:
    1. Contoh environment variables: DATABASE_URL untuk koneksi database dan JWT_SECRET untuk JWT.
4. **next.config.ts**:

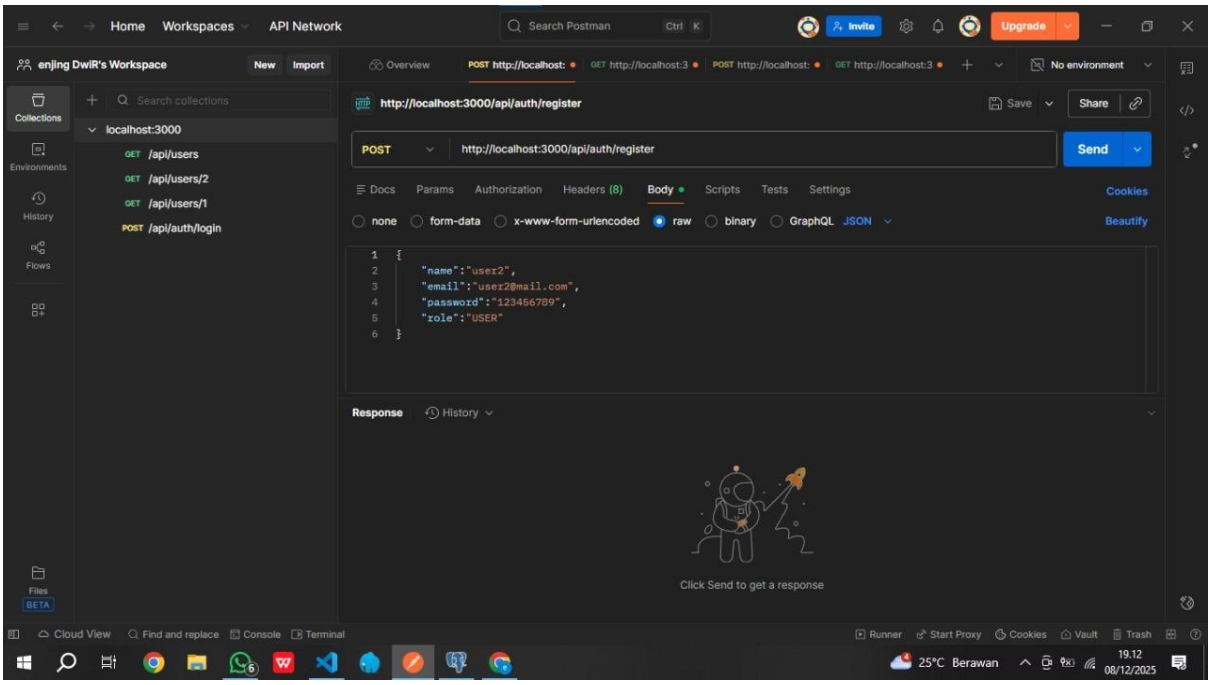
    1. File konfigurasi Next.js dasar (kosong, siap untuk custom seperti rewrites atau images).
5. **.gitignore**:
    1. Mengabaikan file seperti node_modules, .env, .next, dll., untuk menjaga repo bersih dari file sensitif atau generated.
6. **README.md**:
    1. Instruksi dasar untuk menjalankan proyek: npm run dev untuk development server di localhost:3000.
    2. Referensi ke dokumentasi Next.js dan deployment di Vercel.
7. **postcss.config.mjs**:
    1. Konfigurasi PostCSS dengan plugin Tailwind CSS untuk processing CSS.
8. **tsconfig.json**:
    1. Konfigurasi TypeScript: Target ES2017, include JSX, paths alias (@/* untuk root), dan exclude node_modules.
    2. Mendukung incremental compilation untuk performa.
9. **prisma.config.ts**:
    1. Konfigurasi Prisma: Schema di prisma/schema.prisma, migrations di prisma/migrations, dan datasource dari DATABASE_URL.

Cara Menjalankan Proyek

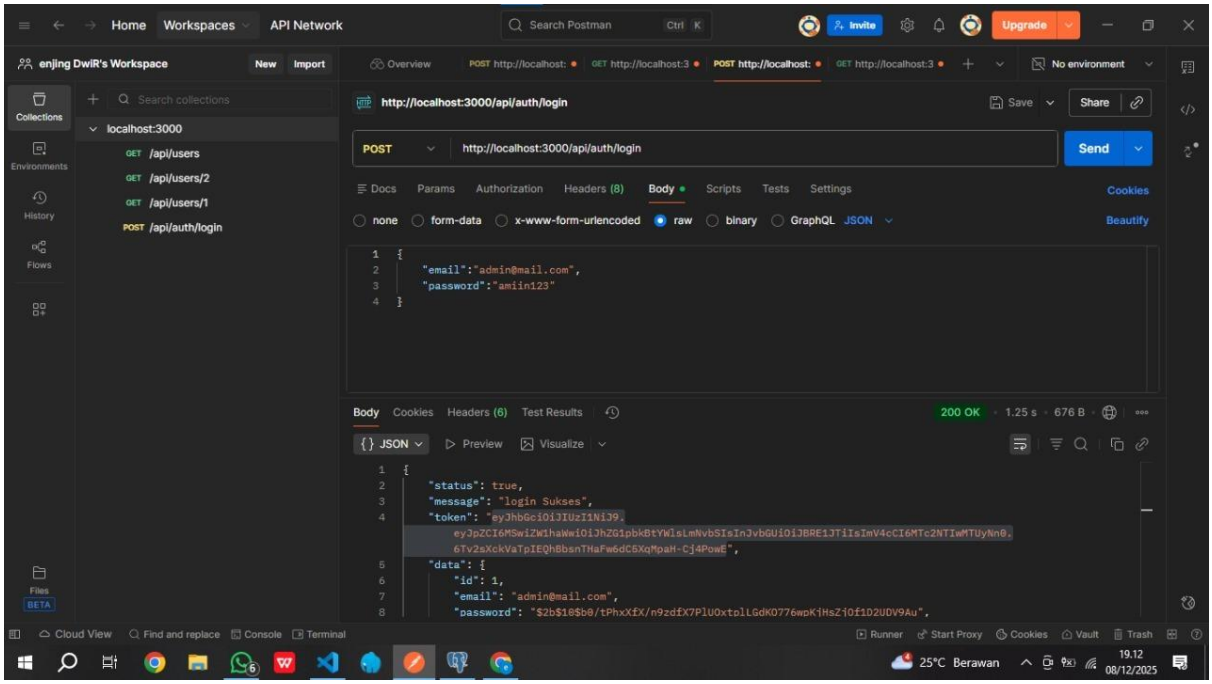
1. Install dependencies: npm install.
2. Setup environment: Copy .env.example ke .env dan isi variabel.
3. Jalankan Prisma: npx prisma generate dan npx prisma migrate dev.
4. Development: npm run dev (server di http://localhost:3000).

Screenshot Hasil Testing

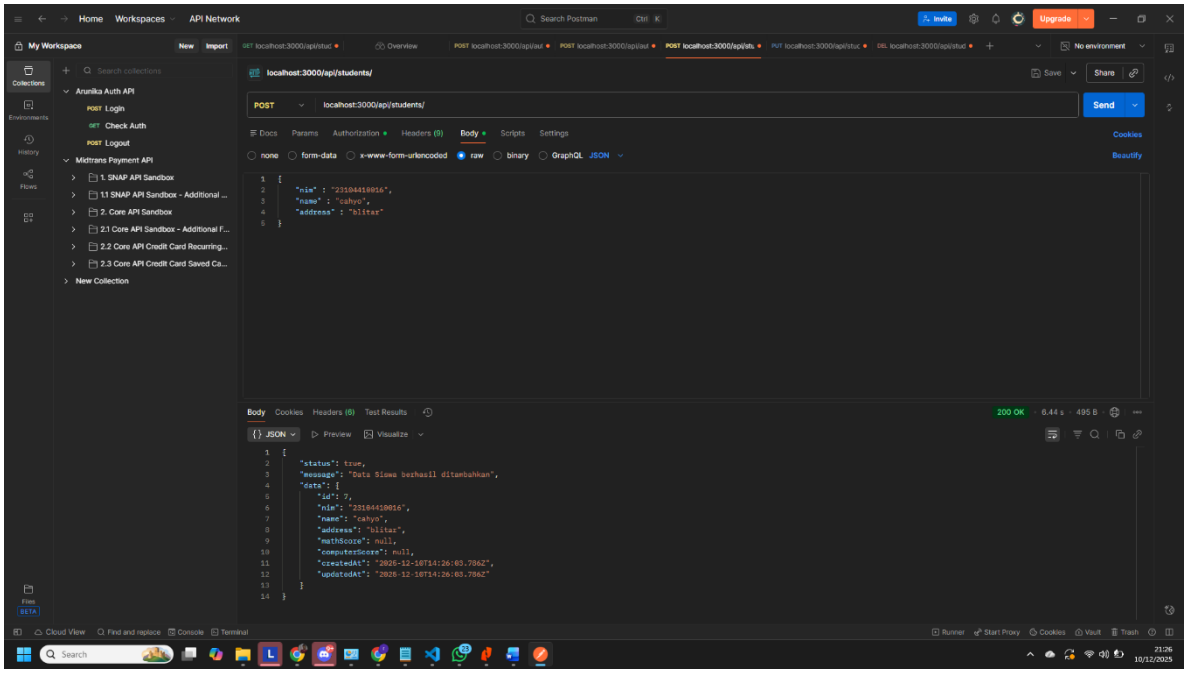
1. Menambahkan / mendaftar sebagai admin dan user



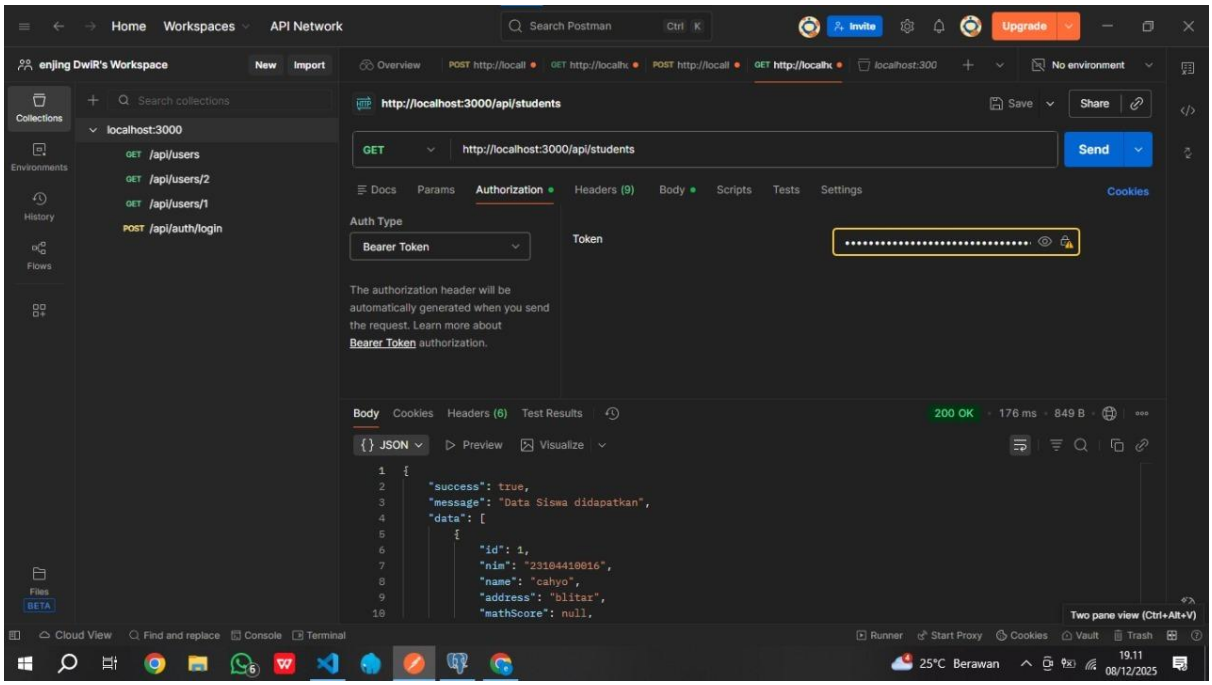
2. Login sebagai admin untuk mendapatkan token



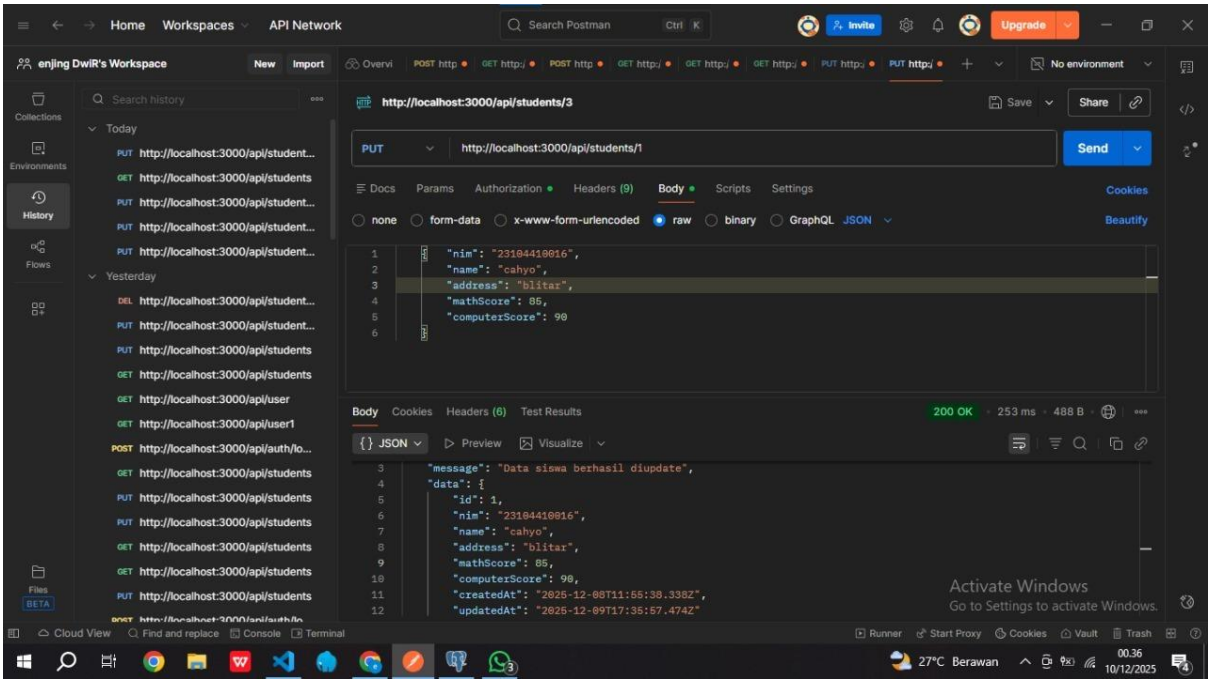
3. Menambahkan data student



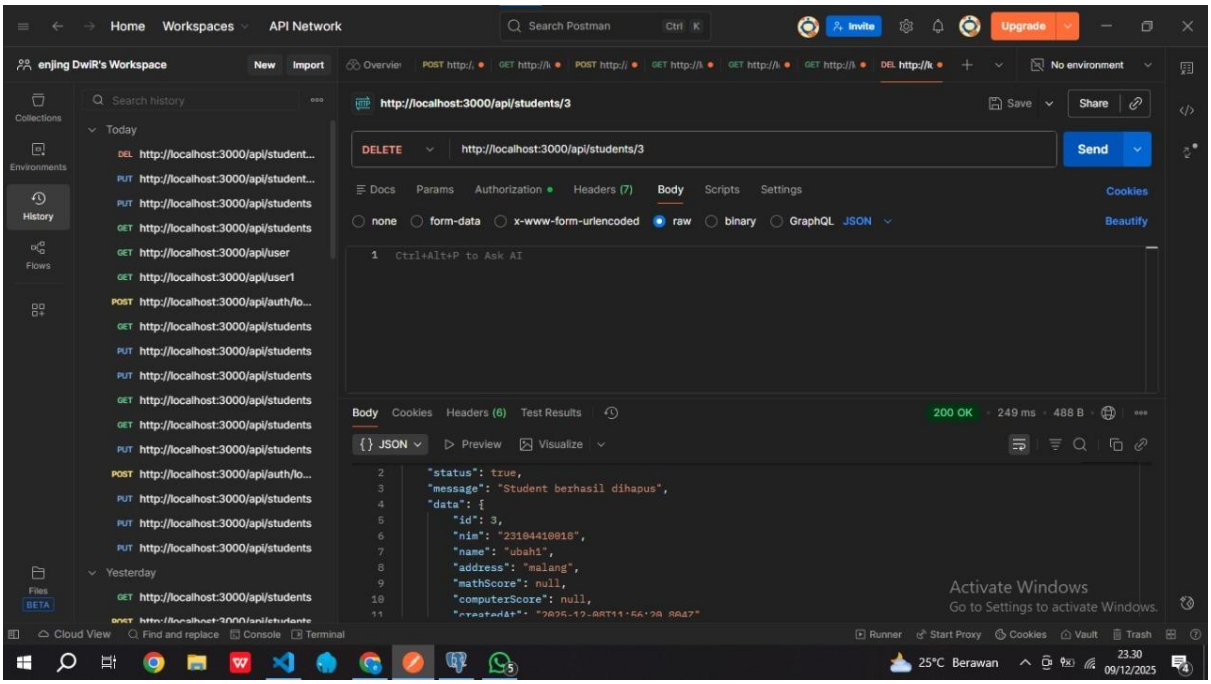
4. Mendapatkan Data student Keseluruhan



5. Memperbarui (Update) data student



6. Delete data student



Penjelasan Coding

A. schema.prisma

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

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init
generator client {
  provider = "prisma-client"
  output   = "../app/generated/prisma"
}
datasource db {
  provider = "postgresql"
}
model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String
  name      String
  role      UserRole @default(USER)
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@map("users")
}
model Student {
  id           Int      @id @default(autoincrement())
  nim          String   @unique
  name         String
  address      String
  mathScore    Int?     @map("math_score")
  computerScore Int?    @map("computer_score")
  createdAt    DateTime @default(now()) @map("created_at")
```

```
  updatedAt     DateTime @updatedAt @map("updated_at")

  @@map("students")
}
enum UserRole {
  ADMIN
  USER
}
```

Ada 2 hal utama tujuan dairi scema prisma:

1. Sistem Login / Pengguna Ada tabel users untuk menyimpan:
    - Email + password (buat login)
    - Nama orangnya
    - Role: apakah dia ADMIN atau USER biasa → Jadi hanya orang yang punya akun bisa masuk ke aplikasi ini, dan admin punya hak lebih tinggi.
2. Data Mahasiswa & Nilainya Ada tabel students untuk menyimpan:
    - NIM (unik, tidak boleh sama)
    - Nama mahasiswa
    - Alamat
    - Nilai Matematika
    - Nilai Ilmu Komputer → Ini intinya aplikasi untuk mengelola data dan nilai mahasiswa

B. Prisma.js

```
        import { PrismaClient } from "../app/generated/prisma/client";
import { PrismaPg } from "@prisma/adapter-pg";

const globalForPrisma = globalThis;

const adapter = new PrismaPg({
  connectionString: process.env.DATABASE_URL,
});
```

```
const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    adapter,
  });

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

export default prisma;
```

Kode ini membuat satu koneksi Prisma ke database PostgreSQL yang sama untuk seluruh aplikasi, biar tidak boros memori dan tetap cepat, terutama saat development (hot-reload).

C. api/auth/register

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

const userSchema = z.object({
  name: z.string().min(3, { message: "Nama minimal 3 karakter" }),
  email: z.string().email({ message: "Email tidal valid" }),
  password: z.string().min(8, { message: "Password minimal 8 karakter" }),
});

export async function POST(req) {
  try {
    const data = await req.json();
    const validation = userSchema.safeParse(data);
    if (!validation.success) {
      return NextResponse.json(
        {
```

```
        status: false,
        message: "Input anda tidak valid",
        error: validation.error.flatten().fieldErrors,
        code: 400,
      },
      { status: 400 }
    );
  }
  const hashedPassword = await bcrypt.hash(data.password, 10);

  const newUser = await prisma.user.create({
    data: {
      ...data,
      password: hashedPassword,
    },
    select: {
      id: true,
      name: true,
      email: true,
      role: true,
      createdAt: true,
      updatedAt: true,
    },
  });
  return NextResponse.json(
    {
      status: true,
      message: "User berhasil dibuat",
      data: newUser,
    },
    { status: 201 }
  );
} catch (error) {
  console.error("Error GET User: ", error);
```

```
    return NextResponse.json(
      {
        status: false,
        error: "server error",
        code: 500,
      },
      { status: 500 }
    );
  }
}
```

API untuk DAFTAR AKUN BARU (register) di aplikasi yng di buatt.”
Ketika ada orang kirim data (nama, email, password) lewat POST ke route ini (misal: /api/register), maka:

1. Validasi input pake Zod
    - Nama minimal 3 huruf
    - Email harus bener (contoh: intan@gmail.com)
    - Password minimal 8 karakter Kalau salah → langsung kasih error yang jelas.
2. Enkripsi password Pakai bcrypt biar password ga disimpan mentah-mentah (aman dari hacker).
3. Simpan user baru ke database Pakai Prisma → buat user baru di tabel users.
4. Balikin respon:
    - Kalau berhasil → status 201 + data user (tanpa password)
    - Kalau gagal → kasih pesan error yang rapi

D. api/auth/login

```
        import prisma from "../../../../lib/prisma";
import { NextResponse } from "next/server";
import bcrypt from "bcryptjs";
import { SignJWT } from "jose";
import { RateLimiterMemory } from "rate-limiter-flexible";

// cek untuk rate limiter
```

```
const rateLimiter = new RateLimiterMemory({
  points: 5, // 5 attempts
  duration: 15 * 60, // 15 menit
});

export async function POST(req) {
  try {
    const { email, password } = await req.json();
    const ip = req.ip || req.headers.get("x-forwarded-for") || "unknown";
    const key = `${email}:${ip}`;

    try {
      await rateLimiter.consume(key);
    } catch (rateLimitError) {
      const retrySeconds = Math.ceil(rateLimitError.msBeforeNext / 1000);

      return NextResponse.json(
        {
          status: false,
          error: "Terlalu banyak percobaan login. Coba lagi dalam 15 menit",
          code: 429,
        },
        {
          status: 429,
          headers: {
            "Retry-After": retrySeconds.toString(),
          },
        }
      );
    }

    // get user data dari db dengan email (unique)
    const user = await prisma.user.findUnique({ where: { email } });
```

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

// cek password
const isPasswordValid = await bcrypt.compare(password, user.password);

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

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);
```

```
    return NextResponse.json({
      status: true,
      message: "login Sukses",
      token: token,
      data: user,
    });
  } catch (error) {
    console.error("Error Login: ", error);
    return NextResponse.json(
      {
        status: false,
        error: "Error Server Login",
        code: 500,
      },
      { status: 500 }
    );
  }
}
```

ini adalah API LOGIN (masuk akun) untuk aplikasi .

Ketika user kirim email + password ke route ini (misal: /api/login), maka:

1. Cegah brute-force Maksimal 5 kali salah login per email + IP dalam 15 menit. Kalau lebih dari 5 → dikunci 15 menit (anti hacker).
2. Cari user berdasarkan email Kalau email tidak ada → “Email tidak ditemukan”
3. Cocokkan password Pakai bcrypt (aman). Kalau salah → “Password anda salah”
4. Kalau berhasil → buat JWT token (tanda masuk) yang berlaku 2 jam Token ini berisi: id, email, dan role user
5. Balikin respon sukses:

E. user

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

const userSchema = z.object({
  name: z.string().min(3, { message: "Nama minimal 3 karakter" }),
  email: z.string().email({ message: "Email tidal valid" }),
  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({
      select: {
        id: true,
        name: true,
        email: true,
        role: true,
        createdAt: true,
        updatedAt: true,
      },
    });

    return NextResponse.json(
      {
        success: true,
        message: "Data Users didapatkan",
        data: users,
      },
      { status: 200 }
```

```
    );
  } catch (error) {
    console.log("Error GET all User: ", error);
    return NextResponse.json(
      {
        status: false,
        error: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}

// POST menambah data user
export async function POST(req) {
  try {
    const data = await req.json();
    const validation = userSchema.safeParse(data);
    if (!validation.success) {
      return NextResponse.json(
        {
          status: false,
          message: "Input anda tidak valid",
          error: validation.error.flatten().fieldErrors,
          code: 400,
        },
        { status: 400 }
      );
    }
    const hashedPassword = await bcrypt.hash(data.password, 10);
    const newUser = await prisma.user.create({
      data: {
        ...data,
```

```
        password: hashedPassword,
      },
      select: {
        id: true,
        name: true,
        email: true,
        role: true,
        createdAt: true,
        updatedAt: true,
      },
    });
    return NextResponse.json(
      {
        status: true,
        message: "User berhasil dibuat",
        data: newUser,
      },
      { status: 201 }
    );
  } catch (error) {
    console.log("Error Add User: ", error);
    return NextResponse.json(
      {
        status: false,
        error: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}
```

Ini adalah satu file API Next.js (biasanya di app/api/users/route.js) yang punya 2 fungsi utama:

Method	Fungsi	Siapa yang boleh pakai?
GET	Ambil semua data user (daftar semua akun)	Biasanya hanya Admin
POST	Buat akun user baru (registrasi)	Bisa semua orang atau hanya Admin

Detail masing-masing:

1. GET → Lihat semua user
   - Mengambil semua data user dari database
   - Hanya menampilkan: id, name, email, role, createdAt, updatedAt → Password tidak menampilkan password (aman!)
   - Balikkan dalam bentuk JSON rapi
2. POST → Daftar / Tambah user baru
   - Validasi input pakai Zod:
     - Nama ≥ 3 karakter
     - Email harus valid
     - Password ≥ 8 karakter
   - Password otomatis di-hash pakai bcrypt (aman banget)
   - Simpan ke database pakai Prisma
   - Balikkan data user yang baru dibuat (tanpa password)

Contoh penggunaan:

- GET /api/users → lihat semua user (hanya admin)

POST /api/users → daftar akun baru (bisa dipakai publik atau hanya admin)

F. user/id

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

export async function GET(req, { params }) {
  try {
```

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

if (isNaN(id)) {
  return NextResponse.json(
    {
      status: false,
      error: "ID tidak valid",
      code: 400,
    },
    { status: 400 }
  );
}

const user = await prisma.user.findUnique({
  where: { id: id },
  select: {
    id: true,
    name: true,
    email: true,
    role: true,
    createdAt: true,
    updatedAt: true,
  },
});

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

```
      );
    }

    return NextResponse.json(
      {
        status: true,
        message: "user ditemukan",
        data: user,
      },
      { status: 200 }
    );
  } catch (error) {
    console.log("Error GET User: ", error);
    return NextResponse.json(
      {
        status: false,
        error: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}

export async function PUT(req, { params }) {
  try {
    const resolvedParams = await params;
    const id = parseInt(resolvedParams.id);
    const data = await req.json();

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

    if (!userExist) {
      return NextResponse.json(
```

```
      {
        status: false,
        error: "ID User tidak ditemukan",
        code: 404,
      },
      { status: 404 }
    );
  }

  const updateData = { ...data };

  if (data.password) {
    updateData.password = await bcrypt.hash(data.password, 10);
  } else {
    delete updateData.password;
  }

  const updateUser = await prisma.user.update({
    where: { id: id },
    data: updateData,
    select: {
      id: true,
      name: true,
      email: true,
      role: true,
      createdAt: true,
      updatedAt: true,
    },
  });
  return NextResponse.json(
    {
      status: true,
      message: "User berhasil diupdate",
      data: updateUser,
```

```
      },
      { status: 200 }
    );
  } catch (error) {
    return NextResponse.json(
      {
        status: false,
        error: "Server Error",
        code: 500,
      },
      { status: 500 }
    );
  }
}

export async function DELETE(req, { params }) {
  try {
    const resolvedParams = await params;
    const id = parseInt(resolvedParams.id);

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

    if (!userExist) {
      return NextResponse.json(
        { message: "ID User tidak ditemukan" },
        { status: 404 } // 404 Not Found lebih tepat
      );
    }

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

```
      status: "success",
      message: "User berhasil dihapus",
      data: deletedUser,
    },
    { status: 200 }
  );
 } catch (error) {
  return NextResponse.json(
    {
      status: false,
      error: "Server Error",
      code: 500,
    },
    { status: 500 }
  );
 }
}
```

Ini adalah API untuk mengelola 1 user berdasarkan ID

Method	URL contoh	Fungsi utama
GET	/api/users/5	Lihat detail user dengan ID = 5
PUT	/api/users/5	Edit / update data user ID = 5
DELETE	/api/users/5	Hapus user dengan ID = 5

Apa yang dilakukan masing-masing:

GET → Cari user berdasarkan ID → Kalau ada → tampilkan data (tanpa password) → Kalau tidak → "user tidak ditemukan"

PUT (Update) → Cek dulu apakah user ada → Kalau kirim password baru → otomatis di-hash (aman) → Update nama, email, role, atau password → Balikin data user yang sudah diubah

DELETE → Cek apakah user ada → Hapus permanen dari database → Kasih pesan sukses

G. students

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

// GET ambil semua data students
export async function GET() {
  try {
    const students = await prisma.student.findMany();
    return NextResponse.json(
      {
        success: true,
        message: "Data Siswa didapatkan",
        data: students,
      },
      { status: 200 }
    );
  } catch (error) {
    console.log("Error GET all students: ", error);
    return NextResponse.json(
      {
        status: false,
        error: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}

export async function POST(req) {
  try {
    const data = await req.json();
```

```
    const newStudent = await prisma.student.create({
      data: data,
    });
    return NextResponse.json({
      status: true,
      message: "Data Siswa berhasil ditambahkan",
      data: newStudent,
    });
  } catch (error) {
    console.log("Error Add Students: ", error);
    return NextResponse.json(
      {
        status: false,
        message: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}
```

File ini adalah API untuk mengelola data mahasiswa (students) (Lokasi biasanya: app/api/students/route.js)

Method	Fungsi
GET	Ambil semua data mahasiswa dari database
POST	Tambah mahasiswa baru ke database

Contoh penggunaan:

- GET /api/students → dapat semua daftar mahasiswa + nilai
- POST /api/students → kirim JSON seperti ini:

JSON

```
{
  "nim": "23104410106",
  "name": "intan",
```

"address": "panggungrejo",
"mathScore": 85,
"computerScore": 90
}
→ langsung masuk ke tabel students

H. students/id

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

export async function GET(req, { params }) {
 try {
  const resolvedParams = await params;
  const id = parseInt(resolvedParams.id);

  if (isNaN(id)) {
   return NextResponse.json({ message: "ID tidak valid" }, { status: 400 });
  }

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

  if (!student) {
   return NextResponse.json(
    {
     status: false,
     error: "Siswa tidak ditemukan",
     code: 401,
    },
    { status: 401 }
   );
```

```
    }

    return NextResponse.json(
      {
        status: true,
        message: "Data siswa ditemukan",
        data: student,
      },
      { status: 200 }
    );
  } catch (error) {
    console.log("Error GET Student: ", error);
    return NextResponse.json(
      {
        status: false,
        message: "Error Server",
        code: 500,
      },
      { status: 500 }
    );
  }
}

export async function PUT(req, { params }) {
  const resolvedParams = await params;
  const id = parseInt(resolvedParams.id);
  const data = await req.json();

  const studentExists = await prisma.student.findUnique({
    where: { id: id },
  });

  if (!studentExists) {
    return NextResponse.json(
```

```
      {
        status: false,
        message: "ID siswa tidak ditemukan",
        code: 404,
      },
      { status: 404 }
    );
  }

  const updateStudent = await prisma.student.update({
    where: { id: id },
    data: data,
  });
  return NextResponse.json(
    {
      status: true,
      message: "Data siswa berhasil diupdate",
      data: updateStudent,
    },
    { status: 200 }
  );
}

export async function DELETE(req, { params }) {
  try {
    const userRole = req.headers.get("x-user-role");

    if (userRole !== "ADMIN") {
      return Response.json(
        {
          status: false,
          error: "Unauthorized: Role anda bukan admin",
          code: 403,
        },
```

```
    { status: 403 }
  );
}

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

const studentExists = await prisma.student.findUnique({
  where: { id: id },
});

if (!studentExists) {
  return NextResponse.json(
    {
      status: false,
      message: "ID siswa tidak ditemukan",
      code: 404,
    },
    { status: 404 }
  );
}

const deletedStudent = await prisma.student.delete({
  where: { id: id },
});

return NextResponse.json(
  {
    status: true,
    message: "Student berhasil dihapus",
    data: deletedStudent,
  },
  { status: 200 }
);
```

```
  } catch (error) {
    return NextResponse.json(
      {
        status: false,
        message: "Server Database Error",
        code: 500,
      },
      { status: 500 }
    );
  }
}
```

File ini adalah API untuk mengelola 1 data mahasiswa berdasarkan ID (

Method	Contoh URL	Fungsi
GET	/api/students/7	Lihat detail 1 mahasiswa (ID = 7)
PUT	/api/students/7	Edit / update data mahasiswa ID = 7
DELETE	/api/students/7	Hapus mahasiswa ID = 7 → HANYA ADMIN yang boleh!

Poin penting:

- GET & PUT → bisa dipakai siapa saja yang login (belum dicek role)
- DELETE → hanya ADMIN yang boleh (dicek lewat header x-user-role)
- Kalau ID salah/tidak ada → kasih error yang jelas
- Semua respon pakai format JSON rapi

API lengkap untuk lihat, edit, dan hapus 1 mahasiswa, dengan proteksi khusus: hanya admin yang boleh menghapus.

Kesimpulan

Proyek "project-api-nextjs" berhasil mengimplementasikan API backend yang aman dan scalable menggunakan Next.js. Ini mencakup autentikasi, otorisasi, dan integrasi database, yang sesuai dengan tujuan UAS kelompok untuk mendemonstrasikan kemampuan pengembangan web modern. Demikian laporan ini kami buat dengan sebenar-benarnya.