

Nama : Muhammad Naufal Tauhid Rapasya
NPM : 23313020
Kelas : TI23A
PWBL

Dokumentasi M Naufal Tauhid R

1. Instalasi Prisma

```
ANOVA_TEAM > backend > prisma >  schema.prisma >  client
Generate
1 generator client {
2   provider = "prisma-client-js"
3 }
4
5 datasource db {
6   provider = "postgresql"
7   url      = env("DATABASE_URL")
8 }
9
10 model User {
11   id      Int      @id @default(autoincrement())
12   nama    String
13   email   String   @unique
14   password String
15   role    Role     @default(USER)
16   orders  Order[]
17   createdAt DateTime @default(now())
18 }
19
20 model Order {
21   id      Int      @id @default(autoincrement())
22   judul  String
23   deskripsi String
24   status  String   @default("MENUNGGU")
25   isPaid Boolean @default(false)
26   tracking String?
}
```

2. src/api/Auth/Register/route.ts

The screenshot shows the file structure of the ANOVA_TEAM project in the Explorer sidebar. The current file is route.ts located in the Register folder under src/api/Auth. The code in route.ts is as follows:

```
WANNOVA_TEAM > backend > src > app > api > Auth > Register > route.ts ID
1 import { NextResponse } from "next/server";
2 import { prisma } from "@/lib/prisma";
3 import { hashPassword } from "@/lib/hash";
4
5 export async function POST(req: Request) {
6   try {
7     const body = await req.json() as {
8       nama: string;
9       email: string;
10      password: string;
11    };
12
13    const { nama, email, password } = body;
14
15    if (!nama || !email || !password) {
16      return NextResponse.json(
17        { message: "Data tidak lengkap" },
18        { status: 400 }
19      );
20    }
21
22    const existing = await prisma.user.findUnique({
23      where: { email },
24    });
25
26    if (existing) {
```

1. Import Dependencies

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { hashPassword } from "@/lib/hash";
NextResponse: Digunakan untuk mengirim respons HTTP (status code, JSON, dll) dari route handler.
prisma: Instance Prisma Client untuk berinteraksi dengan database.
hashPassword: Fungsi utilitas untuk mengenkripsi password (biasanya menggunakan bcrypt).
```

2. Handler POST Request

```
export async function POST(req: Request) {
  try {
    // ...
  } catch (error) {
    // Error handling (tidak ditampilkan di kode Anda)
  }
}
```

Fungsi ini menangani permintaan POST ke endpoint tertentu (misal: /api/register). Menggunakan `async/await` karena operasi database bersifat asynchronous.

3. Parsing Data Request

```
const body = await req.json() as {
  nama: string;
  email: string;
  password: string;
};
```

```
const { nama, email, password } = body;
```

Membaca data JSON dari body request (misal: `{"nama": "Ali", "email": "ali@example.com", "password": "123456"}`).

Menggunakan Type Assertion (`as {...}`) untuk memberi tipe TypeScript pada data.

4. Validasi Data

```
if (!nama || !email || !password) {
  return NextResponse.json(
    { message: "Data tidak lengkap" },
    { status: 400 }
  );
}
```

Memastikan ketiga field wajib (nama, email, password) terisi.

Jika ada yang kosong, kirim respons HTTP 400 (Bad Request) dengan pesan error.

5. Cek Email Duplikat

```
const existing = await prisma.user.findUnique({
  where: { email },
});
```

Mengecek apakah email sudah terdaftar di database.

`prisma.user.findUnique({ where: { email } })` mencari user berdasarkan kolom email (asumsi kolom email memiliki constraint UNIQUE di database).

src/api/Auth/Register/route.ts

The screenshot shows a code editor with the file `route.ts` open. The file is part of a Next.js application structure. The code handles a POST request to the `/api/Login` endpoint. It imports `NextResponse` from `next/server`, `prisma` from `@/lib/prisma`, `comparePassword` from `@/lib/hash`, and `signToken` from `@/lib/jwt`. It first checks if a user exists with the provided email. If not, it returns a 404 error. Then it checks if the password is valid by comparing it with the hashed password stored in the database. If invalid, it returns a 401 error. Otherwise, it signs a token and returns it.

```
1 import { NextResponse } from "next/server";
2 import { prisma } from "@/lib/prisma";
3 import { comparePassword } from "@/lib/hash";
4 import { signToken } from "@/lib/jwt";
5
6 Windsurf: Refactor | Explain | Generate JSDoc | X
7 export async function POST(req: Request) {
8   const { email, password } = await req.json();
9
10  const user = await prisma.user.findUnique({
11    where: { email },
12  });
13
14  if (!user) {
15    return NextResponse.json(
16      { message: "User tidak ditemukan" },
17      { status: 404 }
18    );
19  }
20
21  const valid = await comparePassword(password, user.password);
22
23  if (!valid) {
24    return NextResponse.json(
25      { message: "Password salah" },
26      { status: 401 }
27    );
28}
```

Ini adalah API route di Next.js 13+ (App Router), yang otomatis menjadi endpoint:

<http://localhost:3000/api/Login>

6. Handler POST Request

```
export async function POST(req: Request) {
```

Hanya menerima method POST.

Menggunakan `async/await` karena operasi database & hashing bersifat asynchronous.

7. Parsing Body Request

```
const { email, password } = await req.json();
```

Membaca data dari body request (misal: `{ "email": "user@example.com", "password": "123456" }`)

Destructure langsung ke variabel email dan password.

8. Cari User Berdasarkan Email

```
const user = await prisma.user.findUnique({
```

```
  where: { email },
```

```
});
```

Query database mencari user dengan email yang dikirim.

Jika tidak ditemukan → return error 404.

9. Validasi User Tidak Ditemukan

```
if (!user) {
```

```
  return NextResponse.json(
```

```
    { message: "User tidak ditemukan" },
```

```
    { status: 404 }
```

```
);
```

```
}
```

Jika user null → kirim respons HTTP 404 Not Found.

10. Verifikasi Password

```
const valid = await comparePassword(password, user.password);
```

Membandingkan password yang dikirim (`password`) dengan password terenkripsi di database (`user.password`).

Fungsi comparePassword biasanya menggunakan bcrypt.compare().

11. Validasi Password Salah

```
if (!valid) {
  return NextResponse.json(
    { message: "Password salah" },
    { status: 401 }
);
}
```

Jika password tidak cocok → kirim HTTP 401 Unauthorized.

12. Buat JWT Token

```
const token = signToken({
  id: user.id,
  role: user.role,
});
```

Jika login berhasil, buat token JWT yang berisi:

id: ID user (untuk identifikasi)

role: peran user (admin/user/guest) → berguna untuk otorisasi

13. Kirim Respon Sukses

```
return NextResponse.json({
  message: "Login berhasil",
  token,
  role: user.role,
});
```

Status default: 200 OK

Src/app/api/orders/route.ts

The screenshot shows a code editor with a sidebar displaying the project structure. The structure includes a `WANOVA` folder containing a `WANOVA_TEAM` folder, which has a `backend` folder. Inside `backend` are `prisma` (with `schema.prisma`), `public`, and `src` folders. The `src` folder contains an `app` folder with an `api` folder, which in turn contains an `admin\orders` folder (containing `route.ts`) and an `Auth` folder (containing `health`, `Login`, and `Register` files). There is also a separate `orders` folder containing a `route.ts` file. Other files in the `src` folder include `login`, `favicon.ico`, `globals.css`, `layout.tsx`, `page.module.css`, and `page.tsx`. The `lib` and `.gitignore` files are also present. The main editor area shows the content of the `route.ts` file under the `admin\orders` route. The code handles a POST request to create an order, checking for an authorization header, verifying the token, and creating a new order in the database. It returns a success response with a message and the created order data.

```
WANOVA TEAM > backend > src > app > api > orders > route.ts > POST > order > data
WANOVA_TEAM > backend > src > app > orders > route.ts > POST > order > data
6 | | POST: User membuat pesanan
7 | ===== */
8 | export async function POST(req: Request) {
9 |   try {
10 |     const authHeader = req.headers.get("authorization");
11 |     if (!authHeader) {
12 |       return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
13 |     }
14 |
15 |     const token = authHeader.split(" ")[1]!;
16 |     const user = verifyToken(token) as { id: number | string };
17 |
18 |     const { judul, deskripsi } = await req.json();
19 |
20 |     const order = await prisma.order.create({
21 |       data: [
22 |         {
23 |           judul,
24 |           deskripsi,
25 |           status: "MENUNGGU",
26 |           userId: Number(user.id), // FIX
27 |         },
28 |       ],
29 |     );
30 |
31 |     return NextResponse.json({
32 |       message: "Pesanan berhasil dibuat",
33 |       data: order,
34 |     });
35 |   }
36 | }
```

14. Validasi Autentikasi (JWT Token)

```
const authHeader = req.headers.get("authorization");
if (!authHeader) {
  return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
}
```

Memeriksa apakah header Authorization ada.
Jika tidak ada → kirim respons HTTP 401 Unauthorized.

15. Ekstrak Token dari Header

```
const token = authHeader.split(" ")[1];  
Format header biasanya: Bearer <token>  
.split(" ")[1] mengambil bagian token setelah kata Bearer.  
! adalah TypeScript non-null assertion — karena kita sudah pastikan authHeader ada sebelumnya.
```

16. Verifikasi Token

```
const user = verifyToken(token) as { id: number | string };  
Fungsi verifyToken (biasanya dari @/lib/jwt) memverifikasi dan mendekode JWT.  
Hasilnya adalah payload token, misal: { id: 1, role: "user", iat: ..., exp: ... }  
Kita hanya ambil id untuk menyimpan ke database.
```

17. Parsing Data Pesanan

```
const { judul, deskripsi } = await req.json();  
Membaca data JSON dari body request:
```

18. Buat Pesanan di Database

```
const order = await prisma.order.create({  
  data: {  
    judul,  
    deskripsi,  
    status: "MENUNGGU",  
    userId: Number(user.id), // ✎ FIX  
  },  
});
```

Menggunakan Prisma untuk membuat record baru di tabel Order.

Kolom yang disimpan:

judul: dari input client
deskripsi: dari input client

status: default "MENUNGGU" (bisa juga "DIPROSES", "SELESAI", dll)
userId: ID pengguna yang login → diambil dari token JWT

19. Kirim Respons Sukses

```
return NextResponse.json({  
  message: "Pesanan berhasil dibuat",  
  data: order,  
});
```

Status default: 200 OK

20. Error Handling Global

```
} catch (error) {  
  return NextResponse.json(  
    { message: "Gagal membuat pesanan" },  
    { status: 500 }  
  );  
}
```

Jika terjadi error (misal: database down, token invalid, input salah), kirim respons HTTP 500 Internal Server Error.

```

WANOVA_TEAM > backend > src > app > api > orders > route.ts > GET
42 |   | GET: User melihat pesanan sendiri
43 ===== */
44 Windsurf: Refactor | Explain | X
44 export async function GET(req: Request) {
45   try {
46     const authHeader = req.headers.get("authorization");
47     if (!authHeader) {
48       return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
49     }
50
51     const token = authHeader.split(" ")[1]!;
52     const user = verifyToken(token) as { id: number | string };
53
54     const orders = await prisma.order.findMany({
55       where: { userId: Number(user.id) }, // ✅ FIX
56       orderBy: { createdAt: "desc" },
57     });
58
59     return NextResponse.json({
60       message: "Data pesanan berhasil diambil",
61       data: orders,
62     });
63   } catch {
64     return NextResponse.json(
65       { message: "Gagal mengambil data pesanan" },
66       { status: 500 }
67     );
68   }
69 }
70

```

21.

Memberi tahu developer bahwa fungsi ini digunakan untuk melihat pesanan pribadi oleh user.

- Cek header Authorization → jika tidak ada → 401
- Ambil token → verifikasi → dapatkan user.id
- Query database: cari semua pesanan dengan userId sama dengan user yang login
- Urutkan dari yang terbaru
- Kirim respons sukses + daftar pesanan
- Jika error → kirim 500

```

WANOVA_TEAM > backend > src > app > api > orders > route.ts > PUT > order
70
71  /* ===== */
72  | PUT: User update pesanan
73  ===== */
74  Windsurf: Refactor | Explain | X
75  export async function PUT(req: Request) {
76    try {
77      const authHeader = req.headers.get("authorization");
78      if (!authHeader) {
79        return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
80      }
81
82      const token = authHeader.split(" ")[1]!;
83      const user = verifyToken(token) as { id: number | string };
84
85      const { id, judul, deskripsi } = await req.json();
86
87      const orderId = Number(id); // ✅ FIX
88
89      const order = await prisma.order.findUnique({
90        where: { id: orderId },
91      });
92
93      if (!order) {
94        return NextResponse.json(
95          { message: "Pesanan tidak ditemukan" },
96          { status: 404 }
97        );
98
99      if (order.userId !== Number(user.id)) {
100        return NextResponse.json(

```

22.

Memberi tahu developer bahwa fungsi ini digunakan untuk memperbarui pesanan oleh user.

- pengguna yang terautentikasi (memiliki token valid) yang bisa memperbarui pesanannya sendiri.
- Pengguna hanya bisa mengedit pesanan miliknya — tidak bisa edit pesanan orang lain.
- Pesanan yang sudah dibayar tidak bisa diubah — fitur proteksi bisnis.
- Update hanya mencakup judul & deskripsi — status, isPaid, dll tetap terkendali sistem.

23. Frontend/src/component/

```

PWBL
  WANOVA_TEAM > frontend > src > components > navbar.tsx > Navbar
    1  "use client";
    2
    3  import Link from "next/link";
    4  import { useRouter } from "next/navigation";
    5
    6  Windsurf: Refactor | Explain | Generate JSDoc | X
    7  export default function Navbar({ role }: { role: "ADMIN" | "USER" }) {
    8    const router = useRouter();
    9
    10   Windsurf: Refactor | Explain | Generate JSDoc | X
    11   const handleLogout = () => {
    12     localStorage.removeItem("token");
    13     router.push("/login");
    14   };
    15
    16   return (
    17     <header className="bg-blue-600 text-white p-4 flex justify-between items-center">
    18       <span className="font-bold">Sistem Arsip Digital</span>
    19
    20       <nav className="flex gap-4 items-center">
    21         {role === "ADMIN" && (
    22           <Link href="/admin/dashboard" className="hover:underline">
    23             Dashboard Admin
    24           </Link>
    25         )}
    26
    27         {role === "USER" && [
    28           <Link href="/user/dashboard" className="hover:underline">
    29             Dashboard User
    30           </Link>
    31         ]}
    32
    33       </nav>
    34     </header>
    35   );
    36 }

```

Menampilkan navbar navigasi yang berubah sesuai role pengguna (admin/user), dan menyediakan tombol Logout.

Menandakan bahwa komponen ini adalah Client Component (bukan Server Component). Diperlukan karena komponen ini:

- Menggunakan useRouter() → hook navigasi client-side.
- Menggunakan localStorage → hanya bisa diakses di browser.
- Memiliki event handler (onClick) → interaktif di sisi klien.

24. Import Dependencies

```
import Link from "next/link";
```

```
import { useRouter } from "next/navigation";
```

- Link: Komponen Next.js untuk navigasi halaman tanpa reload (SPA-like).
- useRouter: Hook untuk mengontrol navigasi programatik (misal: redirect setelah logout).

25. Definisi Komponen dengan Prop role

```
export default function Navbar({ role }: { role: "ADMIN" | "USER" }) {
```

Komponen menerima prop role yang hanya bisa bernilai "ADMIN" atau "USER" — ini adalah type safety dengan TypeScript.

26. Inisialisasi Router

```
const router = useRouter();
```

Digunakan untuk melakukan redirect ke halaman lain (misal: /login setelah logout).

27. Fungsi Logout

```
const handleLogout = () => {
  localStorage.removeItem("token");
  router.push("/login");
};
```

- Menghapus token dari localStorage → menghapus autentikasi lokal.
- Redirect ke halaman login → user harus login ulang.
- Ini adalah cara standar untuk logout di aplikasi SPA/Next.js tanpa server-side session

28. Render JSX (Tampilan Navbar)

```
<header className="bg-blue-600 text-white p-4 flex justify-between items-center">
  <span className="font-bold">Sistem Arsip Digital</span>
```

- Tampilan header dengan latar biru, teks putih.
- Judul aplikasi: "Sistem Arsip Digital".

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "WANOVA_TEAM". The "updateordermodal.tsx" file is selected.
- Schema.prisma**: A file icon in the top bar.
- updateordermodal.tsx**: The active code editor tab.
- Code Content:**

```
1  "use client";
2
3  import { useState } from "react";
4
5  Windsurf: Refactor | Explain
6  interface Order {
7    id: number;
8    judul: string;
9    status: string;
10   tracking: string | null;
11 }
12 Windsurf: Refactor | Explain
13 interface Props {
14   order: Order | null;
15   onClose: () => void;
16   onSave: (order: Order) => void;
17 }
18 Windsurf: Refactor | Explain | Generate JSDoc | X
19 export default function UpdateOrderModal({ order, onClose, onSave }: Props) {
20   const [status, setStatus] = useState(order?.status ?? "");
21   const [tracking, setTracking] = useState(order?.tracking ?? "");
22
23   if (!order) return null;
24
25   return (
26     <div className="fixed inset-0 bg-black bg-opacity-60 flex items-center justify-center z-50">
27       <div className="bg-white p-6 rounded w-96">
28         <h2 className="text-xl font-bold mb-4">Update Pesanan</h2>
29
30         <div className="mb-3">
31           <label className="block text-sm font-medium mb-1">Judul</label>
```
- Bottom Bar:** PROBLEMS (1), OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, COMMENTS.

29.

Frontend/src/component/updateordermodal

Menampilkan modal pop-up yang memungkinkan pengguna (biasanya admin) untuk memperbarui status dan nomor tracking pesanan.

- Menampilkan modal untuk memperbarui status dan nomor tracking pesanan.
- Hanya bisa mengubah status dan tracking — judul tidak bisa diubah.
- Mengirim data yang diperbarui ke parent via callback onSave.
- Tidak ditampilkan jika order null — mencegah error.