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
generator client {
  provider = "prisma-client-js"
}

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

model User {
  id        Int      @id @default(autoincrement())
  nama      String
  email     String   @unique
  password  String
  role      Role     @default(USER)
  orders    Order[]
  createdAt DateTime @default(now())
}

model Order {
  id          Int      @id @default(autoincrement())
  judul       String
  deskripsi   String
  status      String   @default("MENUNGGU")
  isPaid      Boolean  @default(false)
  tracking    String?
```

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

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.

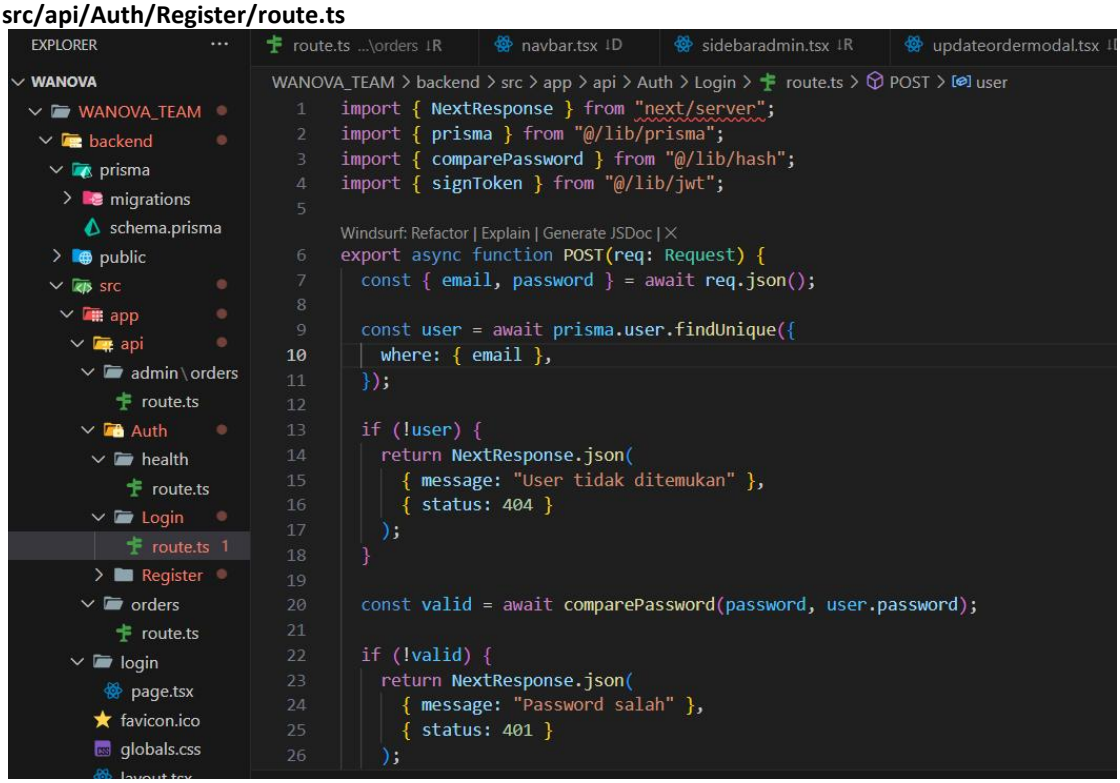
4. Validasi Data

```
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



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,
});
```

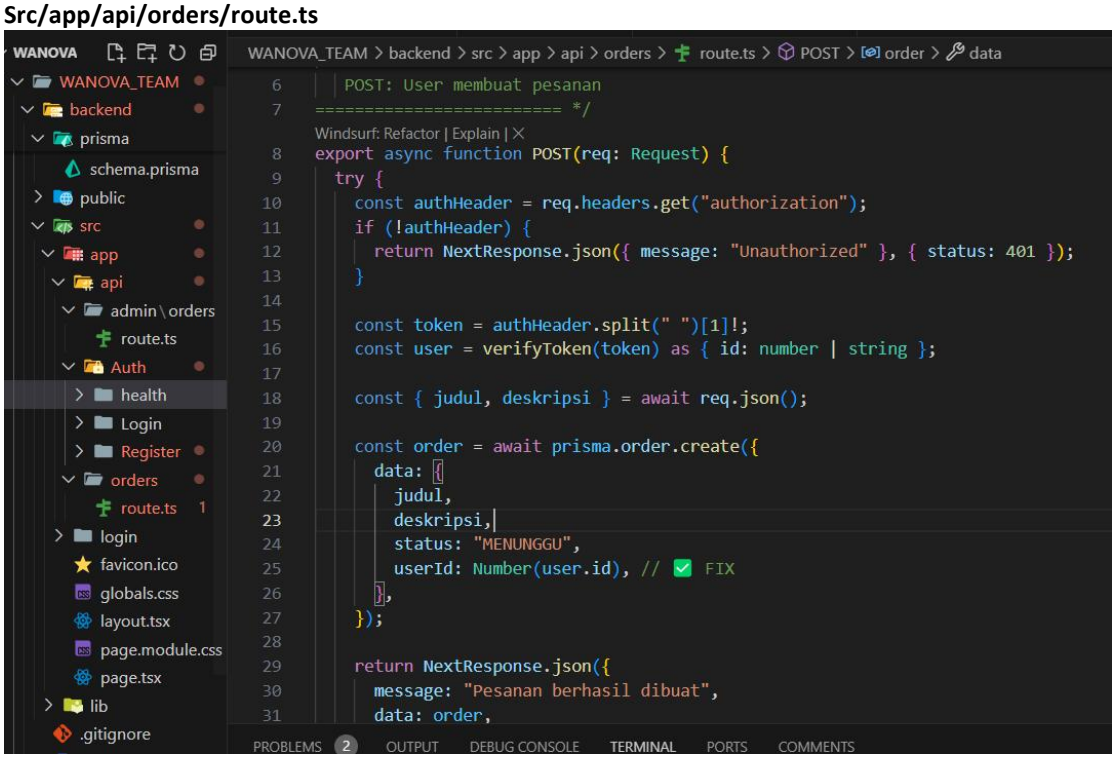
Jika login berhasil, buat token JWT yang berisi:
id: ID user (untuk identifikasi)
role: peran user (admin/user/guest) → berguna untuk otorisasi

13. Kirim Respons Sukses

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

Status default: 200 OK

Src/app/api/orders/route.ts



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.

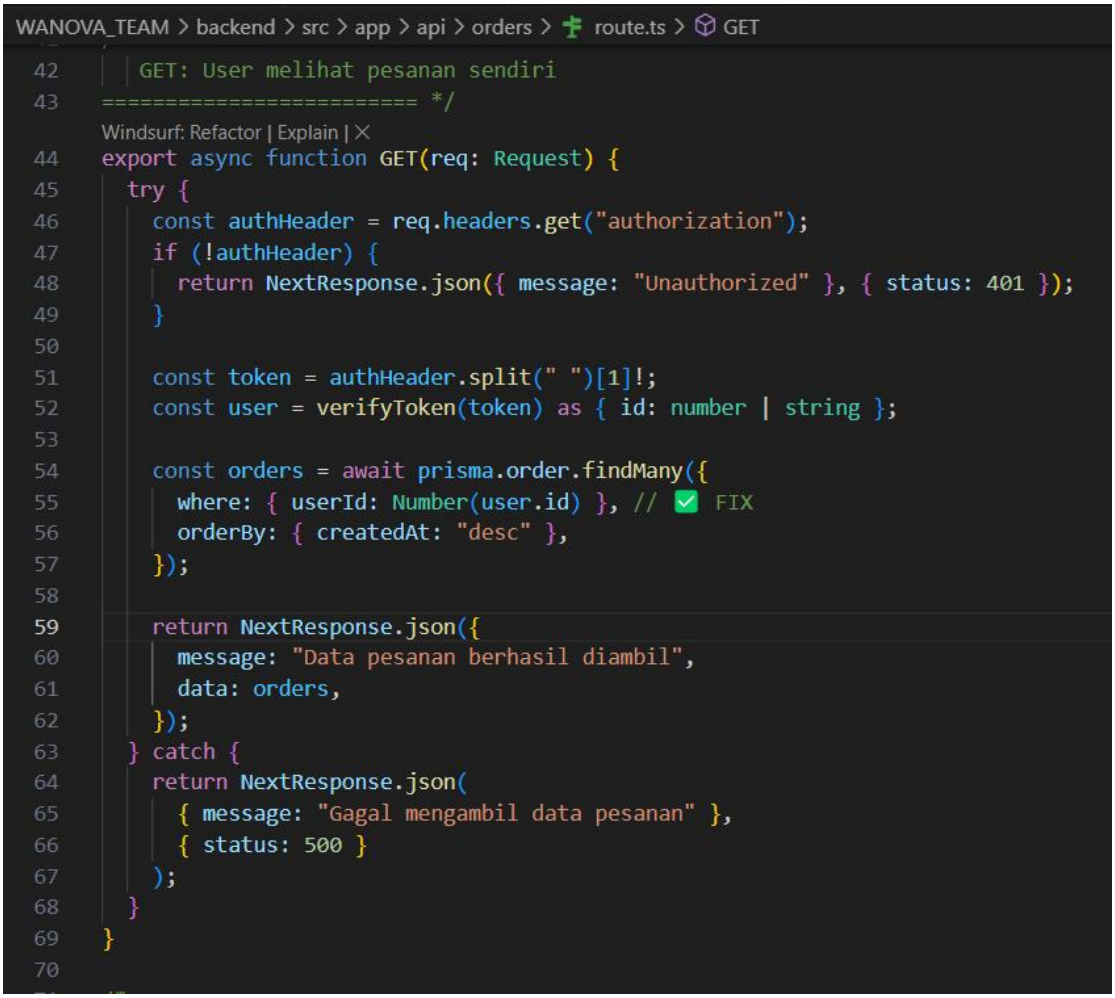
16. Verifikasi Token
const user = verifyToken(token) as { id: number | string };
Fungsi verifyToken (biasanya dari @/lib/jwt) memverifikasi dan mendekode JWT.
Hasilnya adalah payload token, misal: { id: 1, role: "user", iat: ..., exp: ... }
Kita hanya ambil id untuk menyimpan ke database.

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

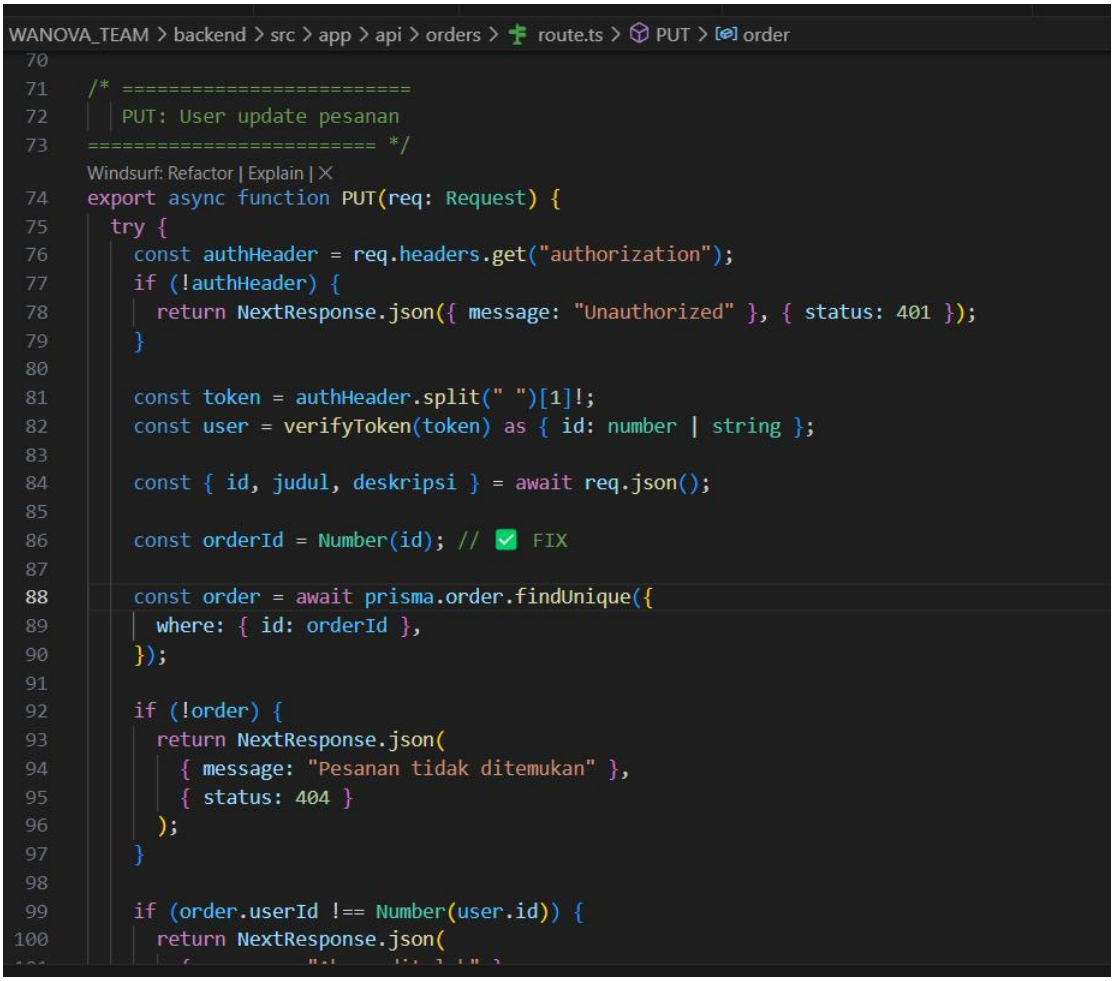
20. Error Handling Global
```
} catch (error) {
  return NextResponse.json(
    { message: "Gagal membuat pesanan" },
    { status: 500 }
  );
}
```
Jika terjadi error (misal: database down, token invalid, input salah), kirim respons HTTP 500 Internal Server Error.

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

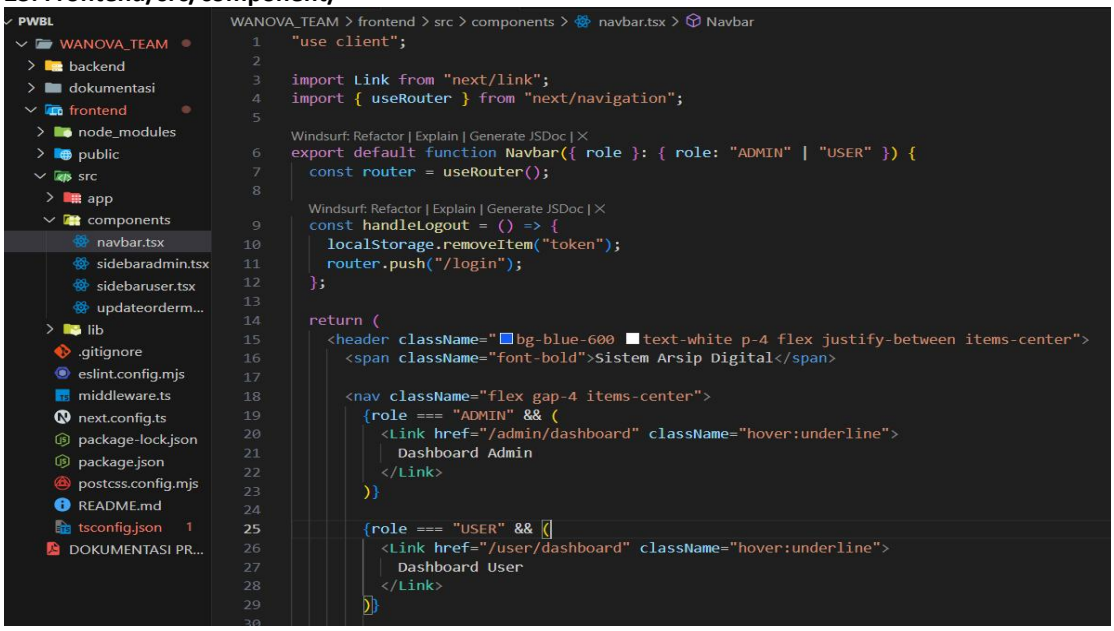
22.



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/



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

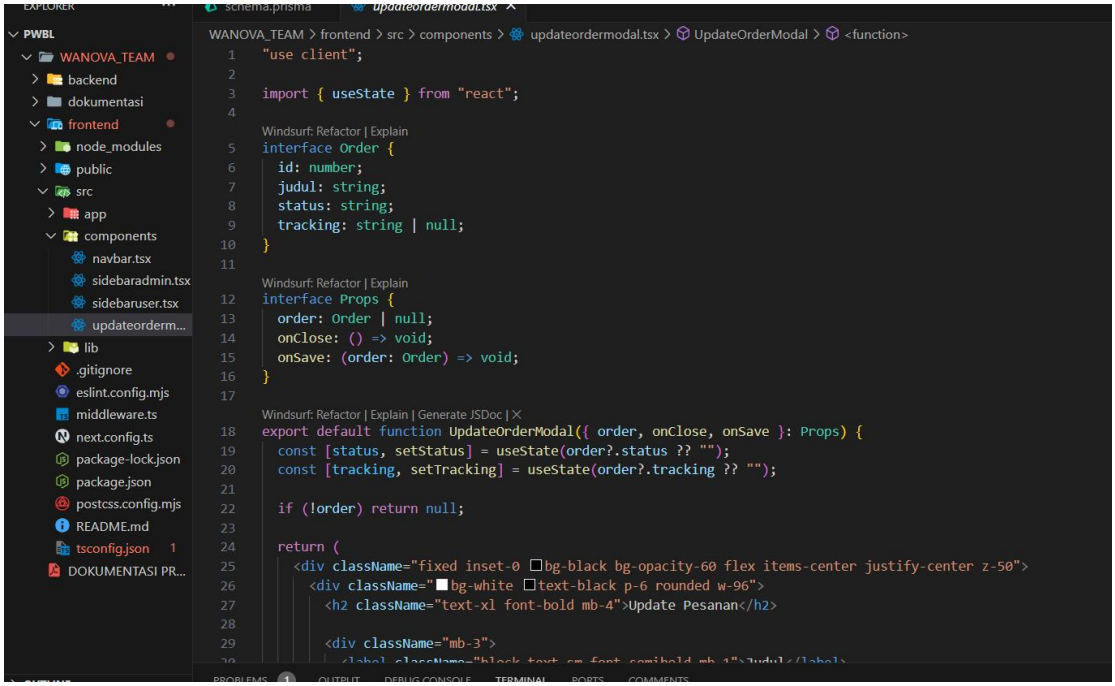
```
const handleLogout = () => {
  localStorage.removeItem("token");
  router.push("/login");
};
```

- 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”.

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.