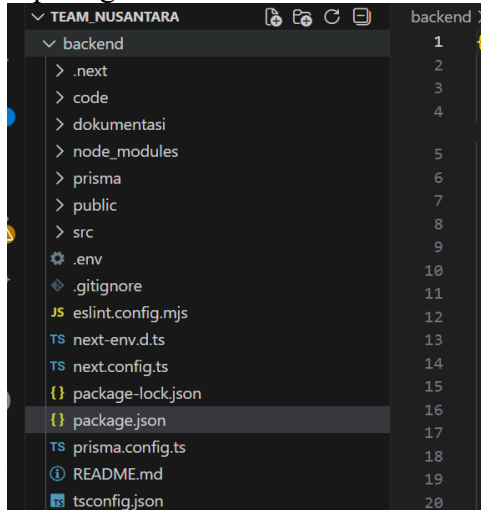


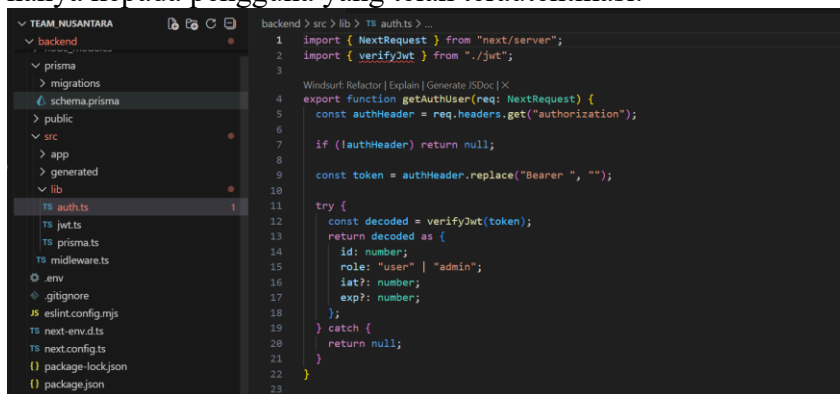
Nama : Muhammad Fabio Andre  
Kelas : TI 23 A  
Npm : 23313023

## DOKUMENTASI PROJECT PWBL

1. Menginstall backend, setelah menginstall ubah port backend pada file package.json bagian scripts menjadi 3001 agar tidak sama dengan frontend nya dan membuat struktur folder seperti gambar dibawah ini:



2. Pada file auth.ts di folder src/app kode ini digunakan untuk mengecek autentikasi pengguna melalui JWT pada setiap permintaan yang masuk. Fungsi getUserAuth mengambil token dari header Authorization, lalu memverifikasinya menggunakan verifyJwt. Jika token valid, data pengguna seperti id dan role akan dikembalikan. Namun, jika token tidak ada atau tidak valid, fungsi akan mengembalikan null, sehingga sistem dapat membatasi akses hanya kepada pengguna yang telah terautentikasi.



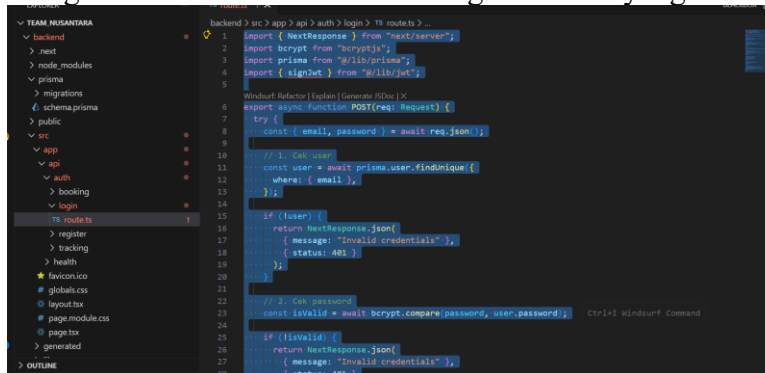
3. Pada file prisma.ts di folder src/app kode ini digunakan untuk menginisialisasi dan mengelola koneksi Prisma agar tidak membuat banyak instance saat aplikasi berjalan, terutama pada mode development. Dengan menyimpan objek PrismaClient di variabel global, aplikasi akan menggunakan satu koneksi yang sama meskipun terjadi reload. Hal

ini mencegah terjadinya error akibat terlalu banyak koneksi database dan membuat penggunaan Prisma lebih efisien serta stabil selama pengembangan:



```
1 import { PrismaClient } from "@prisma/client";
2
3 const globalForPrisma = global as unknown as {
4   prisma: PrismaClient | undefined;
5 };
6
7 export const prisma =
8   globalForPrisma.prisma ??
9   new PrismaClient({
10     log: ["query"],
11   });
12
13 if (process.env.NODE_ENV !== "production") {
14   globalForPrisma.prisma = prisma;
15 }
16
17 export default prisma;
```

4. Pada file login/route.ts di folder src/app/api/auth kode ini merupakan API untuk proses login pengguna. Program menerima data email dan password dari request, lalu mengecek apakah pengguna dengan email tersebut ada di database menggunakan Prisma. Jika pengguna tidak ditemukan, sistem akan mengembalikan respons "Invalid credentials". Selanjutnya, password yang dikirim akan dibandingkan dengan password yang tersimpan menggunakan bcrypt. Jika password benar, sistem akan membuat JWT yang berisi id dan role pengguna. Token ini kemudian dikirim kembali sebagai respons dan digunakan sebagai bukti autentikasi untuk mengakses fitur yang dilindungi:



```
1 import { NextResponse } from "next/server";
2 import bcrypt from "bcryptjs";
3 import prisma from "@prisma/client";
4 import { jwt } from "jsonwebtoken";
5
6 export async function POST(req: Request) {
7   try {
8     const { email, password } = await req.json();
9
10    // 1. Cari user
11    const user = await prisma.user.findUnique({
12      where: { email },
13    });
14
15    if (!user) {
16      return NextResponse.json(
17        { message: "Invalid credentials" },
18        { status: 401 }
19      );
20    }
21
22    // 2. Cek password
23    const isValid = await bcrypt.compare(password, user.password);
24
25    if (!isValid) {
26      return NextResponse.json(
27        { message: "Invalid credentials" },
28        { status: 401 }
29      );
30    }
31  } catch {
32    // ...
33  }
34}
```

5. Pada file register/route.ts di folder src/app/api/auth kode ini digunakan sebagai API untuk proses pendaftaran (register) pengguna baru. Program menerima data name, email, dan password dari request, lalu memeriksa apakah semua data telah diisi. Selanjutnya, sistem mengecek apakah email sudah terdaftar di database. Jika belum, password akan dienkripsi menggunakan bcrypt sebelum disimpan. Data pengguna kemudian disimpan ke database melalui Prisma, dan sistem mengembalikan respons berupa id serta email pengguna yang berhasil didaftarkan:

```

1 import { NextResponse } from "next/server";
2 import bcrypt from "bcrypt";
3 import prisma from "@/lib/prisma";
4
5 // Windows: Refactor | Explain | Generate JSDoc | X
6 export async function POST(req: Request) {
7   const { name, email, password } = await req.json();
8
9   if (!name || !email || !password) {
10    return NextResponse.json(
11      { message: "All fields required" },
12      { status: 400 }
13    );
14   }
15
16   const exist = await prisma.user.findUnique({ where: { email } });
17   if (exist) {
18    return NextResponse.json(
19      { message: "Email already used" },
20      { status: 409 }
21    );
22   }
23
24   const hashed = await bcrypt.hash(password, 10);
25
26   const user = await prisma.user.create({
27     data: {
28       name,
29       email,
30       password: hashed
31     }
32   });
33
34   return NextResponse.json(
35     { message: "User created successfully" },
36     { status: 201 }
37   );
38 }

```

- Kode ini digunakan sebagai API untuk mengelola data pesanan (*order*). Metode GET berfungsi mengambil seluruh data order dari database beserta relasi pengguna dan layanan, sedangkan metode POST digunakan untuk menambahkan data order baru berdasarkan data yang dikirim oleh client. Hasil proses tersebut dikembalikan dalam bentuk JSON agar dapat digunakan oleh aplikasi frontend:

```

1 import prisma from "@/src/lib/prisma";
2 import { NextResponse } from "next/server";
3
4 export async function GET() {
5   const orders = await prisma.order.findMany({
6     include: { user: true, service: true }
7   });
8   return NextResponse.json(orders);
9 }
10
11 export async function POST(req: Request) {
12   const body = await req.json();
13
14   const order = await prisma.order.create({ data: body });
15   return NextResponse.json(order);
16 }

```

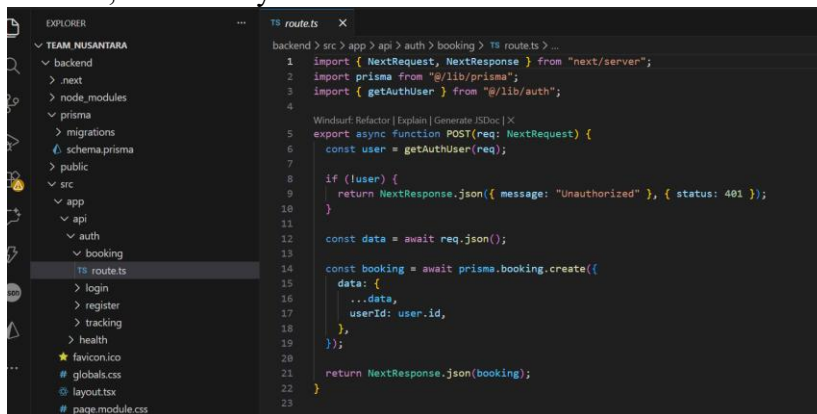
- File ini digunakan untuk menyimpan variabel lingkungan (*environment variables*) yang dibutuhkan aplikasi. DATABASE\_URL berisi konfigurasi koneksi ke database PostgreSQL, sedangkan JWT\_SECRET digunakan sebagai kunci rahasia untuk membuat dan memverifikasi token JWT. Dengan memisahkan data penting ini ke dalam file .env, konfigurasi menjadi lebih aman dan fleksibel tanpa harus menuliskannya langsung di dalam kode program:

```

1 # Environment variables declared in this file are NOT automatically loaded by Prisma.
2 # Please add 'import "dotenv/config";' to your 'prisma.config.ts' file, or use the Prisma CLI with Bu
3 # to load environment variables from .env files: https://pris.ly/prisma-config-env-vars.
4
5 # Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server, Mong
6 # See the documentation for all the connection string options: https://pris.ly/d/connection-strings
7
8 # The following 'prisma:postgres' URL is similar to the URL produced by running a local Prisma Postgr
9 # server with the 'prisma dev' CLI command, when not choosing any non-default ports or settings. The
10 # one found in a remote Prisma Postgres URL, does not contain any sensitive information.
11
12 DATABASE_URL="postgresql://postgres:123@localhost:5432/bengkel_service"
13 JWT_SECRET=supersecretkey123
14
15
16

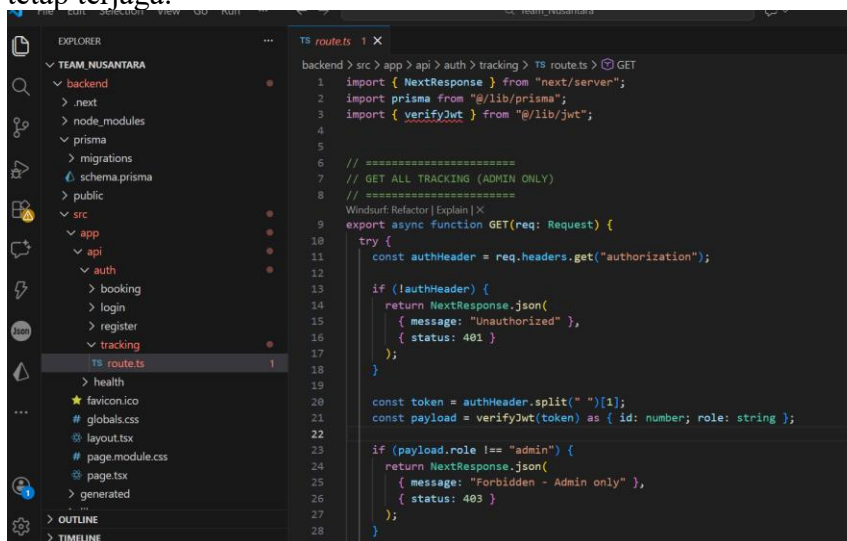
```

8. Kode ini digunakan sebagai API untuk membuat data booking yang hanya dapat diakses oleh pengguna yang telah terautentikasi. Fungsi `getAuthUser` digunakan untuk mengecek apakah request memiliki token yang valid. Jika pengguna belum login, sistem akan mengembalikan status *Unauthorized*. Jika sudah terautentikasi, data booking yang dikirim akan disimpan ke database melalui Prisma dengan menambahkan `userId` dari pengguna tersebut, lalu hasilnya dikembalikan dalam bentuk JSON:



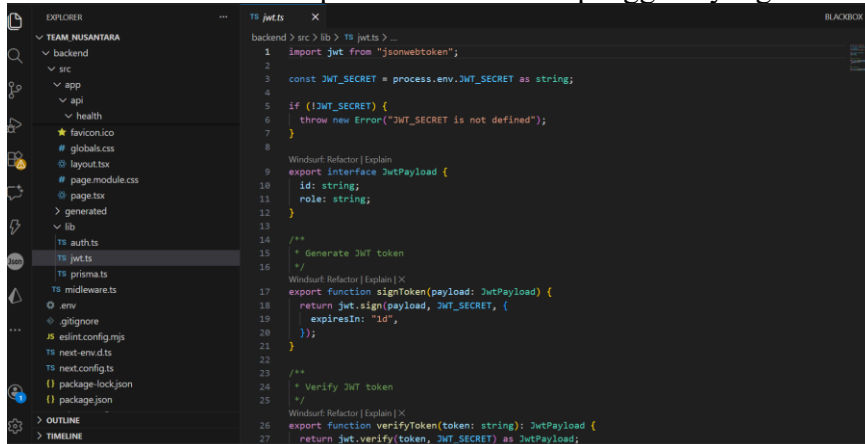
```
1 import { NextRequest, NextResponse } from "next/server";
2 import prisma from "@lib/prisma";
3 import { getAuthUser } from "@lib/auth";
4
5 export async function POST(req: NextRequest) {
6   const user = getAuthUser(req);
7
8   if (!user) {
9     return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
10  }
11
12  const data = await req.json();
13
14  const booking = await prisma.booking.create({
15    data: {
16      ...data,
17      userId: user.id,
18    },
19  });
20
21  return NextResponse.json(booking);
22 }
23
```

9. Kode ini merupakan API untuk mengelola data *tracking* yang hanya dapat diakses oleh admin. Pada metode GET, sistem memverifikasi token JWT dari header dan memastikan bahwa pengguna memiliki peran sebagai admin sebelum mengambil seluruh data tracking beserta informasi booking dan user. Pada metode POST, sistem juga memverifikasi bahwa permintaan berasal dari admin, kemudian membuat atau memperbarui data tracking berdasarkan `bookingId` dan `progress` yang dikirim. Dengan mekanisme ini, hanya admin yang dapat melihat dan mengubah status tracking, sehingga keamanan dan kontrol data tetap terjaga:



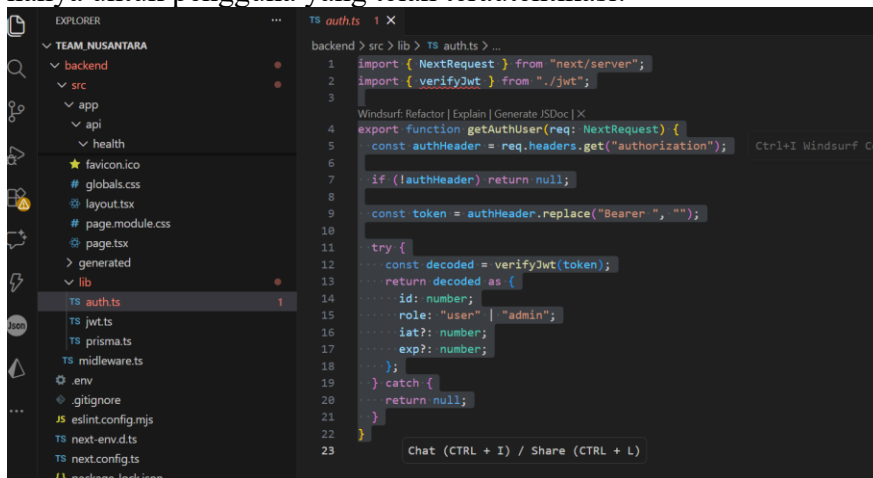
```
1 import { NextResponse } from "next/server";
2 import prisma from "@lib/prisma";
3 import { verifyJwt } from "@lib/jwt";
4
5 // =====
6 // GET ALL TRACKING (ADMIN ONLY)
7 // =====
8
9 export async function GET(req: Request) {
10   try {
11     const authHeader = req.headers.get("authorization");
12
13     if (!authHeader) {
14       return NextResponse.json(
15         { message: "Unauthorized" },
16         { status: 401 }
17       );
18     }
19
20     const token = authHeader.split(" ")[1];
21     const payload = verifyJwt(token) as { id: number; role: string };
22
23     if (payload.role !== "admin") {
24       return NextResponse.json(
25         { message: "Forbidden - Admin only" },
26         { status: 403 }
27       );
28     }
29
30     // GET ALL TRACKING (ADMIN ONLY)
31     const tracking = await prisma.tracking.findMany();
32     return NextResponse.json(tracking);
33   } catch (error) {
34     console.error(error);
35     return NextResponse.json(
36       { message: "Internal Server Error" },
37       { status: 500 }
38     );
39   }
40 }
41
```

10. Kode ini berfungsi untuk membuat dan memverifikasi token JWT sebagai bagian dari sistem autentikasi. Variabel `JWT_SECRET` digunakan sebagai kunci rahasia untuk menjaga keamanan token. Fungsi `signToken` digunakan untuk menghasilkan token berisi data pengguna dengan masa berlaku satu hari, sedangkan `verifyToken` digunakan untuk memeriksa keaslian dan validitas token tersebut. Dengan mekanisme ini, sistem dapat memastikan bahwa setiap akses berasal dari pengguna yang sah dan terautentikasi:



```
1 import jwt from "jsonwebtoken";
2
3 const JWT_SECRET = process.env.JWT_SECRET as string;
4
5 if (!JWT_SECRET) {
6   throw new Error("JWT_SECRET is not defined");
7 }
8
9 export interface JwtPayload {
10   id: string;
11   role: string;
12 }
13
14 /**
15  * Generate JWT token
16  */
17 export function signToken(payload: JwtPayload) {
18   return jwt.sign(payload, JWT_SECRET, {
19     expiresIn: "1d",
20   });
21 }
22
23 /**
24  * Verify JWT token
25  */
26 export function verifyToken(token: string): JwtPayload {
27   return jwt.verify(token, JWT_SECRET) as JwtPayload;
```

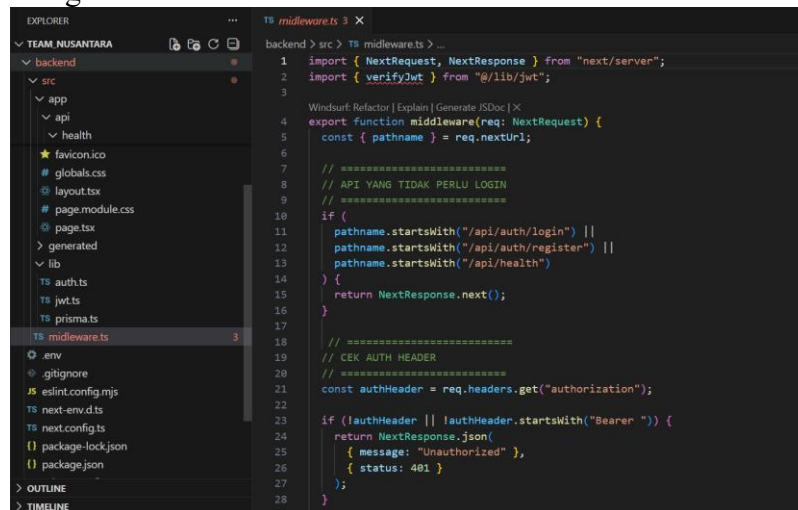
11. Kode ini digunakan untuk mengambil dan memverifikasi data pengguna dari token JWT pada setiap request. Fungsi `getAuthUser` membaca header `Authorization`, mengambil token yang dikirim, lalu memverifikasinya menggunakan `verifyJwt`. Jika token valid, fungsi mengembalikan informasi pengguna seperti `id` dan `role`. Namun, jika token tidak ada atau tidak valid, fungsi akan mengembalikan `null`, sehingga sistem dapat membatasi akses hanya untuk pengguna yang telah terautentikasi:



```
1 import { NextRequest } from "next/server";
2 import { verifyJwt } from "../jwt";
3
4 export function getAuthUser(req: NextRequest) {
5   const authHeader = req.headers.get("authorization");
6
7   if (!authHeader) return null;
8
9   const token = authHeader.replace("Bearer ", "");
10
11   try {
12     const decoded = verifyJwt(token);
13     return decoded as {
14       id: number;
15       role: "user" | "admin";
16       iat?: number;
17       exp?: number;
18     };
19   } catch {
20     return null;
21   }
22 }
```

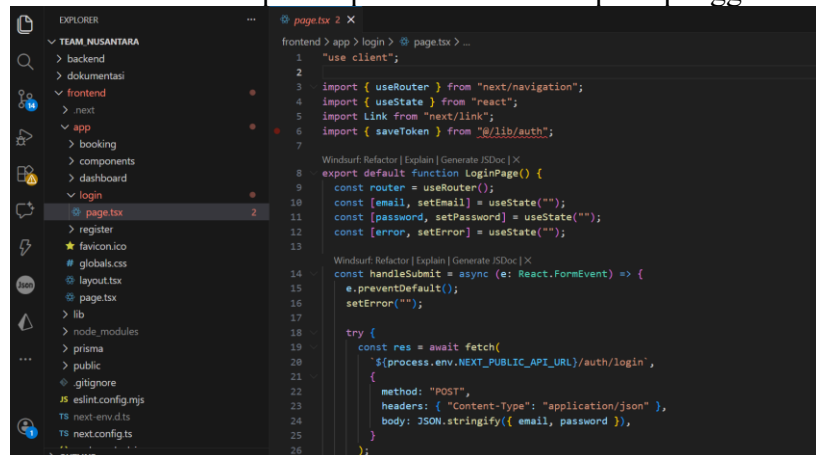
12. Kode middleware ini berfungsi sebagai pengaman API dengan memeriksa autentikasi menggunakan JWT. Middleware akan melewati beberapa endpoint publik seperti `login`, `register`, dan `health` tanpa pengecekan. Untuk endpoint lainnya, sistem memeriksa header

Authorization dan memverifikasi token. Jika token tidak ada atau tidak valid, akses akan ditolak. Selain itu, rute tertentu seperti /api/tracking dibatasi khusus untuk pengguna dengan peran admin. Dengan mekanisme ini, hanya pengguna yang berhak yang dapat mengakses API tertentu:



```
1 import { NextRequest, NextResponse } from "next/server";
2 import { verifyJwt } from "@lib/jwt";
3
4 export function middleware(req: NextRequest) {
5   const { pathname } = req.nextUrl;
6
7   // =====
8   // API YANG TIDAK PERLU LOGIN
9   // =====
10  if (
11    pathname.startsWith("/api/auth/login") ||
12    pathname.startsWith("/api/auth/register") ||
13    pathname.startsWith("/api/health")
14  ) {
15    return NextResponse.next();
16  }
17
18  // =====
19  // CEK AUTH HEADER
20  // =====
21  const authHeader = req.headers.get("authorization");
22
23  if (!authHeader || !authHeader.startsWith("Bearer ")) {
24    return NextResponse.json(
25      { message: "Unauthorized" },
26      { status: 401 }
27    );
28  }
```

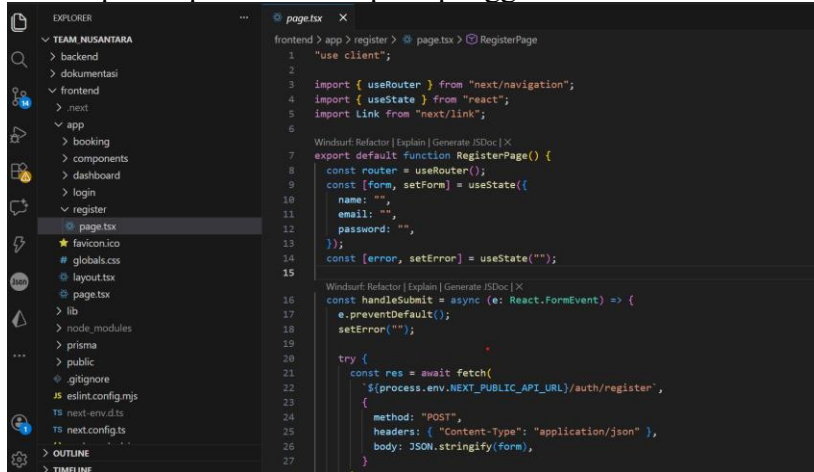
13. Kode ini merupakan komponen halaman login pada sisi frontend. Komponen ini sudah dihubungkan ke backend dan menampilkan form untuk memasukkan email dan password, lalu mengirimkannya ke API saat tombol login ditekan. Jika proses login berhasil, token yang diterima akan disimpan menggunakan fungsi saveToken, kemudian pengguna diarahkan ke halaman dashboard. Apabila login gagal atau server tidak dapat dihubungi, sistem akan menampilkan pesan kesalahan kepada pengguna:



```
1 "use client";
2
3 import { useRouter } from "next/navigation";
4 import { useState } from "react";
5 import Link from "next/link";
6 import { saveToken } from "@lib/auth";
7
8 export default function LoginPage() {
9   const router = useRouter();
10  const [email, setEmail] = useState("");
11  const [password, setPassword] = useState("");
12  const [error, setError] = useState("");
13
14  const handleSubmit = async (e: React.FormEvent) => {
15    e.preventDefault();
16    setError("");
17
18    try {
19      const res = await fetch(
20        `${process.env.NEXT_PUBLIC_API_URL}/auth/login`,
21        {
22          method: "POST",
23          headers: { "Content-Type": "application/json" },
24          body: JSON.stringify({ email, password }),
25        }
26      );
```

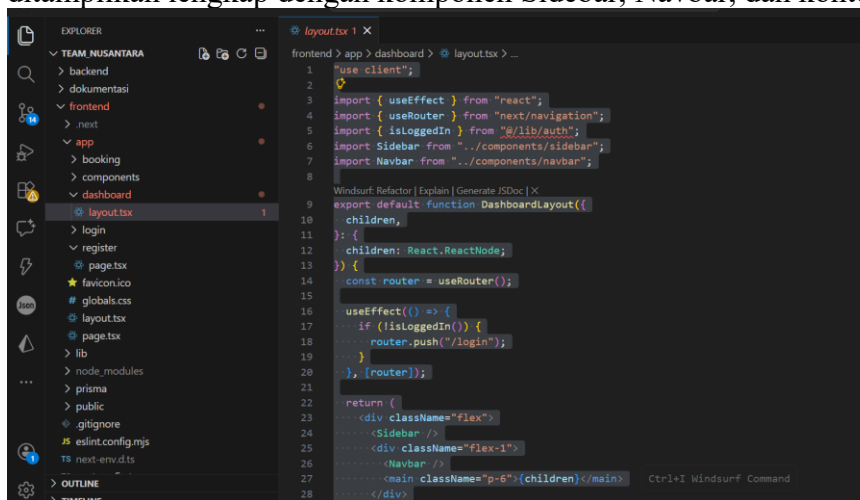
14. Kode ini merupakan komponen halaman pendaftaran pengguna pada sisi frontend. Codingan ini sudah di hubunngkan ke backend dan ini halaman menyediakan form untuk memasukkan nama, email, dan password, kemudian mengirimkan data tersebut ke API register saat form disubmit. Jika registrasi berhasil, pengguna akan diarahkan ke halaman

login. Namun, jika terjadi kesalahan atau server tidak dapat dihubungi, sistem akan menampilkan pesan error kepada pengguna:



```
1 "use client";
2
3 import { useRouter } from "next/navigation";
4 import { useState } from "react";
5 import Link from "next/link";
6
7 export default function RegisterPage() {
8   const router = useRouter();
9   const [form, setForm] = useState({
10     name: "",
11     email: "",
12     password: "",
13   });
14   const [error, setError] = useState("");
15
16   const handleSubmit = async (e: React.FormEvent) => {
17     e.preventDefault();
18     setError("");
19
20     try {
21       const res = await fetch(
22         `${process.env.NEXT_PUBLIC_API_URL}/auth/register`,
23         {
24           method: "POST",
25           headers: { "Content-Type": "application/json" },
26           body: JSON.stringify(form),
27         }
28       );
29     } catch {
30       setError("Gagal mendaftar. Silakan coba lagi.");
31     }
32   };
33 }
```

15. Kode ini digunakan sebagai *layout* halaman dashboard yang hanya dapat diakses oleh pengguna yang sudah login. Saat komponen dimuat, sistem akan mengecek status login menggunakan fungsi `isLoggedIn`. Jika pengguna belum terautentikasi, maka akan diarahkan kembali ke halaman login. Jika sudah login, halaman dashboard akan ditampilkan lengkap dengan komponen Sidebar, Navbar, dan konten utama di dalamnya.



```
1 "use client";
2
3 import { useEffect } from "react";
4 import { useRouter } from "next/navigation";
5 import { isLoggedIn } from "@/lib/auth";
6 import Sidebar from "@/components/sidebar";
7 import Navbar from "@/components/navbar";
8
9 export default function DashboardLayout({
10   children,
11 }) {
12   const router = useRouter();
13
14   useEffect(() => {
15     if (!isLoggedIn()) {
16       router.push("/login");
17     }
18   }, [router]);
19
20   return (
21     <div className="flex">
22       <Sidebar />
23       <div className="flex-1">
24         <Navbar />
25         <main className="p-6">{children}</main>
26       </div>
27     </div>
28   );
29 }
```