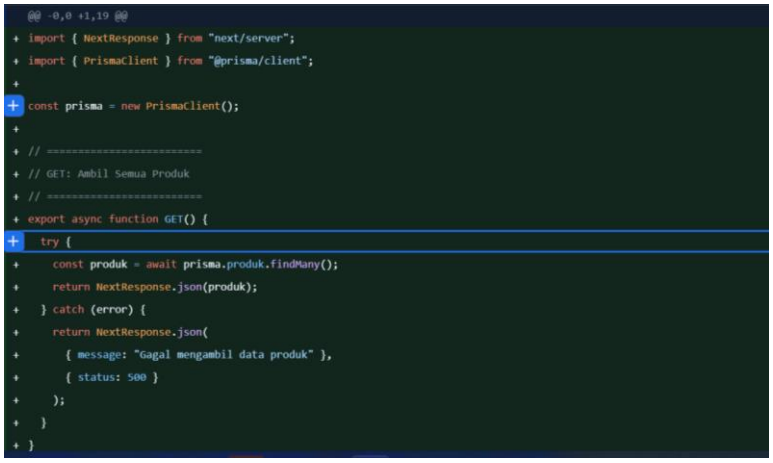
Nama : Diva Afirlia Maheswari

NPM : 23313022

Kelas : TI 23 A

Laporan Dokumentasi Projek

1. File route.ts ini digunakan untuk membuat **API endpoint** pada Next.js (App Router) yang berfungsi **mengambil seluruh data produk dari database** menggunakan Prisma ORM



```
@@ -0,0 +1,19 @@
+ import { NextResponse } from "next/server";
+ import { PrismaClient } from "@prisma/client";
+
+ const prisma = new PrismaClient();
+
+ // ======================
+ // GET: Ambil Semua Produk
+ // ======================
+ export async function GET() {
+   try {
+     const produk = await prisma.produk.findMany();
+     return NextResponse.json(produk);
+   } catch (error) {
+     return NextResponse.json(
+       { message: "Gagal mengambil data produk" },
+       { status: 500 }
+     );
+   }
+ }
```

Kode route.ts tersebut juga digunakan untuk membuat API endpoint **GET** pada Next.js yang berfungsi mengambil seluruh data produk dari database. Pada awal kode, NextResponse digunakan untuk mengatur response API, sedangkan PrismaClient berfungsi sebagai penghubung ke database. Prisma diinisialisasi agar aplikasi dapat menjalankan query database, kemudian fungsi GET dibuat untuk menangani permintaan pengambilan data produk. Data diambil menggunakan prisma.produk.findMany() dan dikembalikan ke client dalam bentuk JSON. Jika terjadi kesalahan saat proses pengambilan data, sistem akan menampilkan pesan error dengan status HTTP 500 sebagai penanda kesalahan pada sisi server.

2. Potongan kode ini digunakan untuk melakukan **validasi username** sebelum data pengguna diproses lebih lanjut. Sistem mengecek apakah nilai name yang dikirim sudah terdaftar di database dengan menggunakan fungsi prisma.user.findFirst() pada tabel user. Jika data dengan username tersebut ditemukan, maka API akan langsung menghentikan proses dan mengembalikan response JSON berisi pesan *"Username sudah digunakan"* dengan status HTTP **409 (Conflict)**, yang menandakan adanya bentrok data karena username harus bersifat unik. Validasi ini bertujuan untuk menjaga konsistensi data serta mencegah duplikasi akun pengguna dalam sistem.

```
    }
+
+     // 4. Cek username (name) sudah ada atau belum
+   const nameExist = await prisma.user.findFirst({
+     where: { name }
+   });
+
+   if (nameExist) {
+     return NextResponse.json(
+       { message: "Username sudah digunakan" },
+       { status: 409 }
+     );
+   }

  }
+
```

3. DEETE : Hapus Produk

```
+
+ / ======================
+ // DELETE: Hapus Produk per ID
+ // ======================
+ export async function DELETE(req: Request) {
+   try {
+     const url = new URL(req.url);
+     const id = Number(url.searchParams.get("id"));
+
+     if (!id) {
+       return NextResponse.json(
+         { message: "ID produk tidak ditemukan" },
+         { status: 400 }
+       );
+     }
+
+     await prisma.produk.delete({
+       where: { id },
+     });
```

Kode ini berfungsi sebagai **API DELETE** untuk menghapus data produk berdasarkan **ID** yang dikirim melalui parameter URL. Pada awal fungsi, sistem mengambil URL dari request dan mengekstrak nilai id menggunakan searchParams, kemudian mengonversinya menjadi tipe angka. Jika ID tidak ditemukan atau tidak valid, API akan mengembalikan response JSON dengan pesan *"ID produk tidak ditemukan"* dan status HTTP **400 (Bad Request)**. Apabila ID valid, sistem akan menghapus data produk dari database menggunakan perintah prisma.produk.delete() dengan kondisi berdasarkan ID tersebut. Implementasi ini memastikan proses penghapusan data berjalan terkontrol dan hanya dilakukan pada data yang valid.

4. Menambahkan response di DELETE

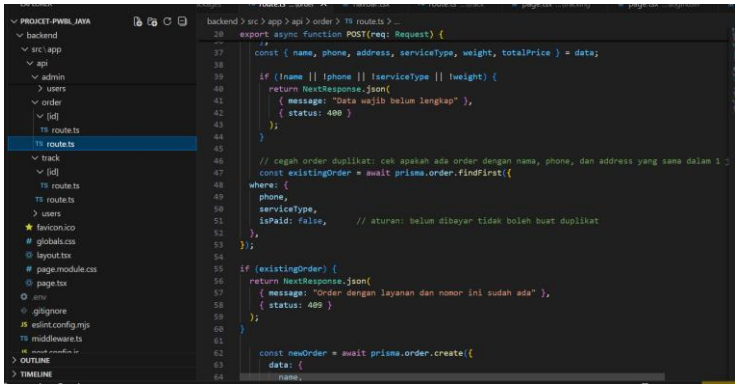
```
       where: { id },
     });
+    return NextResponse.json(
+      { message: "Produk berhasil dihapus" },
+      { status: 200 }
+    );
+  } catch (error) {
+    return NextResponse.json(
+      { message: "Gagal menghapus produk" },
+      { status: 500 }
+    );
+  }
+ }
```

Setelah proses penghapusan data produk berhasil dilakukan menggunakan Prisma, API akan mengembalikan response JSON dengan pesan *"Produk berhasil dihapus"* dan status HTTP **200 (OK)** sebagai penanda bahwa operasi berjalan sukses. Namun, jika terjadi kesalahan selama proses penghapusan, misalnya ID tidak ditemukan di database atau terjadi error pada server, maka blok catch akan dijalankan dan API mengirimkan pesan *"Gagal menghapus produk"* dengan status HTTP **500 (Internal Server Error)**. Mekanisme ini memastikan sistem memberikan informasi yang jelas terkait keberhasilan maupun kegagalan proses penghapusan data.

5. Verif Token

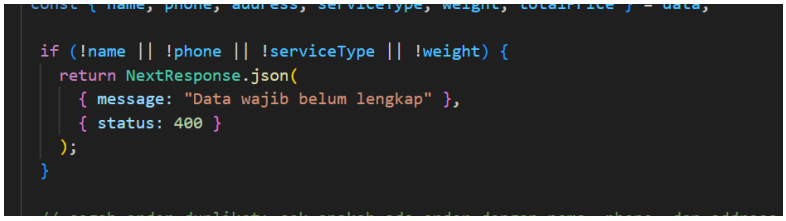
Perubahan pada file package.json dilakukan dengan memperbarui dependency **Prisma** dari versi ^5.22.0 menjadi ^7.0.0 untuk menyesuaikan kebutuhan pengembangan backend yang lebih terbaru. Prisma digunakan sebagai ORM untuk mempermudah pengelolaan database, termasuk proses CRUD dan pengelolaan skema data. Pembaruan versi ini bertujuan meningkatkan stabilitas, keamanan, serta kompatibilitas sistem dengan teknologi yang digunakan, sehingga proses pengembangan dan pengelolaan database dapat berjalan lebih optimal dan terstruktur.

6. Validasi Data



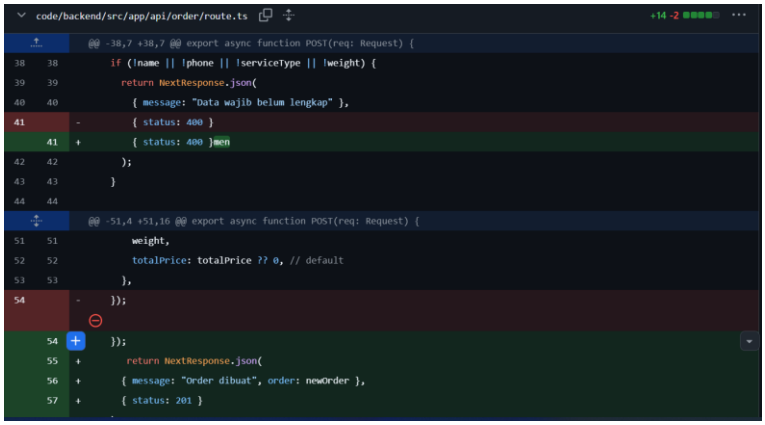
Kode tersebut merupakan API endpoint **POST** pada Next.js App Router yang berfungsi untuk membuat data order baru ke dalam database menggunakan Prisma ORM. Proses diawali dengan menerima data dari request berupa nama, nomor telepon, alamat, jenis layanan, berat, dan total harga. Selanjutnya sistem melakukan validasi untuk memastikan data wajib seperti nama, nomor telepon, jenis layanan, dan berat telah diisi; jika tidak lengkap maka server akan mengembalikan respons error dengan status 400. Setelah itu, sistem melakukan pengecekan order duplikat dengan mencari data order yang memiliki nomor telepon dan jenis layanan yang sama serta berstatus belum dibayar (isPaid = false) untuk mencegah pembuatan order ganda. Jika order tersebut sudah ada, maka server akan mengembalikan pesan bahwa order sudah terdaftar dengan status 409. Apabila tidak ditemukan duplikasi, sistem akan menyimpan data order baru ke dalam database dan mengembalikan respons bahwa pembuatan order berhasil.

7. Eror Handling



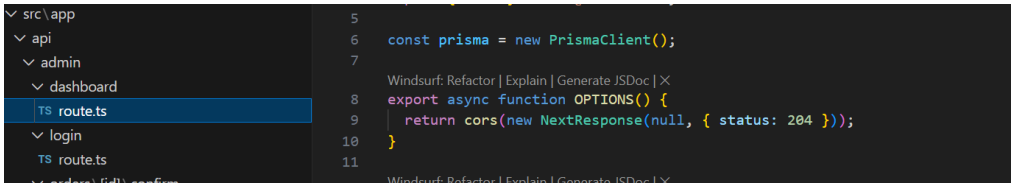
```
if (!name || !phone || !serviceType || !weight) {
  return NextResponse.json(
    { message: "Data wajib belum lengkap" },
    { status: 400 }
  );
}
```

kode ini berfungsi untuk **mengambil dan memvalidasi data input** yang diterima oleh sistem. Data seperti name, phone, address, serviceType, weight, dan totalPrice diambil dari objek data menggunakan teknik *destructuring*. Selanjutnya, sistem melakukan pengecekan terhadap data wajib, yaitu name, phone, serviceType, dan weight. Jika salah satu dari data tersebut tidak diisi atau bernilai kosong, maka proses akan dihentikan dan API mengembalikan response JSON berisi pesan *"Data wajib belum lengkap"* dengan status HTTP **400 (Bad Request)**. Validasi ini bertujuan untuk memastikan bahwa data penting telah diisi sebelum diproses lebih lanjut.

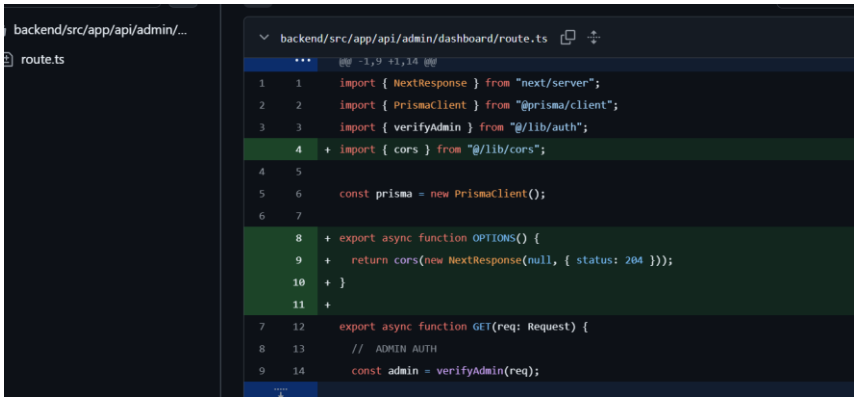


```
code/backend/src/app/api/order/route.ts                                +14 -2
@@ -38,7 +38,7 @@ export async function POST(req: Request) {
38 38     if (!name || !phone || !serviceType || !weight) {
39 39       return NextResponse.json(
40 40         { message: "Data wajib belum lengkap" },
41    -      { status: 400 }
   41 +      { status: 400 }men
42 42       );
43 43     }
44 44
@@ -51,4 +51,16 @@ export async function POST(req: Request) {
51 51         weight,
52 52         totalPrice: totalPrice ?? 0, // default
53 53       },
54    - });
   54 + });
55 +     return NextResponse.json(
56 +     { message: "Order dibuat", order: newOrder },
57 +     { status: 201 }
```

8. Update Admin



kode ini digunakan untuk **menginisialisasi koneksi database dan menangani permintaan CORS** pada API. PrismaClient diinisialisasi agar aplikasi dapat berkomunikasi dengan database menggunakan Prisma ORM. Sementara itu, fungsi OPTIONS dibuat untuk menangani *preflight request* dari browser yang berkaitan dengan CORS (Cross-Origin Resource Sharing). Fungsi ini mengembalikan response kosong dengan status HTTP **204 (No Content)** yang menandakan bahwa permintaan diperbolehkan, sehingga request dari domain lain dapat diproses dengan lancar oleh API.



9. Update admin AUTH

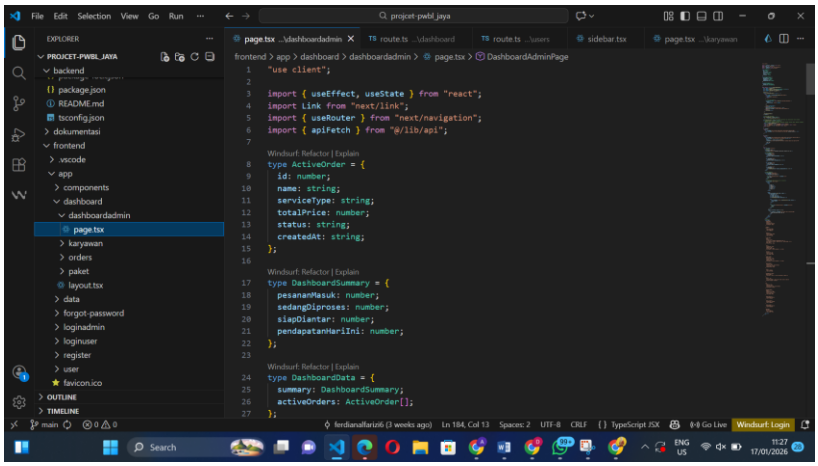
kode ini berfungsi untuk melakukan **autentikasi admin** sebelum suatu proses dijalankan. Fungsi verifyAdmin(req) digunakan untuk memeriksa apakah request yang masuk berasal dari pengguna dengan hak akses admin. Jika hasil verifikasi gagal atau admin tidak valid, maka API akan langsung menghentikan proses dan mengembalikan response JSON berisi pesan *"Unauthorized admin"* dengan status HTTP **401 (Unauthorized)** melalui mekanisme CORS. Validasi ini bertujuan untuk menjaga keamanan sistem dengan memastikan bahwa hanya admin yang berwenang yang dapat mengakses atau menjalankan fitur tertentu.

10. Update Middleware

```
// Fungsi pembantu CORS eksplisit di dalam middleware untuk memastikan header selalu ada.
Windsurf: Refactor | Explain |
function setCorsHeaders(res: NextResponse) {
  res.headers.set("Access-Control-Allow-Origin", "http://localhost:3001");
  res.headers.set("Access-Control-Allow-Methods", "GET,POST,PUT,PATCH,DELETE,OPTIONS");
  res.headers.set("Access-Control-Allow-Headers", "Content-Type, Authorization");
  return res;
}
```

Kode tersebut merupakan **fungsi pembantu untuk mengatur CORS (Cross-Origin Resource Sharing)** pada middleware agar setiap response API selalu memiliki header CORS. Fungsi setCorsHeaders menerima objek NextResponse, lalu menambahkan header Access-Control-Allow-Origin untuk mengizinkan akses dari domain http://localhost:3001, Access-Control-Allow-Methods untuk menentukan metode HTTP yang diperbolehkan (GET, POST, PUT, PATCH, DELETE, dan OPTIONS), serta Access-Control-Allow-Headers untuk mengizinkan penggunaan header Content-Type dan Authorization. Fungsi ini bertujuan mencegah pemblokiran request oleh browser dan memastikan komunikasi antara frontend dan backend berjalan dengan baik.

11. Update page dashboard admin



Jadi dashboard admin tidak bisa di akses begitu saja, tetapi ada fitur log yang mengharuskan admin login terlebih dahulu untuk mengakses dashboard tersebut.

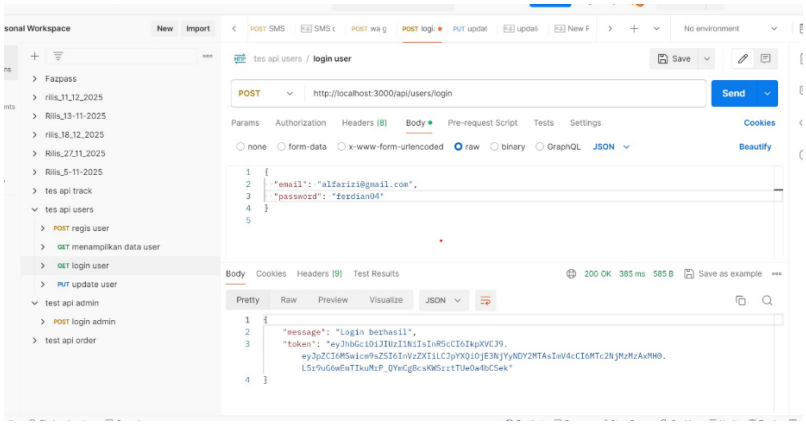
12. Update tampilan order pada page admin agar data yang masuk di dapatkan dari page user nya

```
15
16  /* FORMAT RUPIAH */
    Windsurf: Refactor | Explain | ×
17  const formatRupiah = (value: number) =>
18    "Rp " + value.toLocaleString("id-ID");
19
    Windsurf: Refactor | Explain | Generate JSDoc | ×
20  export default function OrdersPage() {
21    const router = useRouter();
22    const [orders, setOrders] = useState<Order[]>([]);
23    const [loading, setLoading] = useState(true);
24    const [error, setError] = useState("");
25
26    /* FETCH ORDERS */
27    useEffect(() => {
        Windsurf: Refactor | Explain | Generate JSDoc | ×
28      const fetchOrders = async () => {
29        try {
30          const token = localStorage.getItem("admin_token");
31          if (!token) {
32            router.push("/loginadmin");
33            return;
34          }
35
36          const data = await apiFetch("/api/admin/orders", {
37            headers: {
38              Authorization: `Bearer ${token}`,
39            },
40          });
```

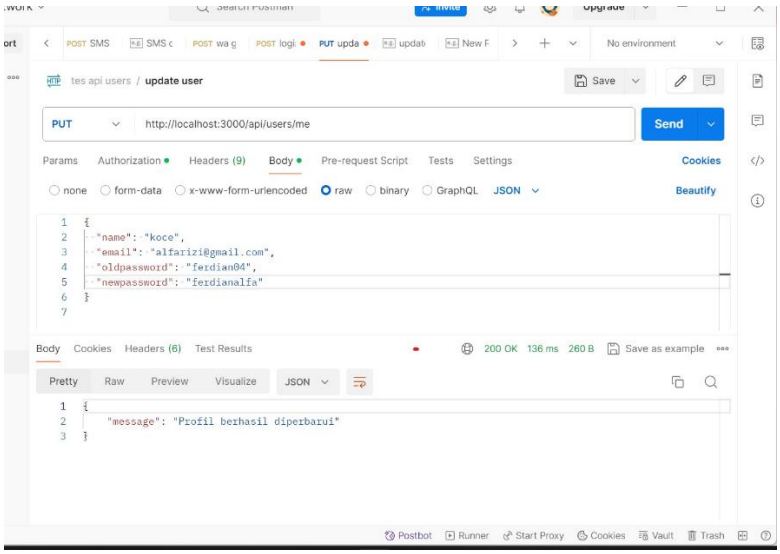
13. Login User Fix

```
    try {
      const data = await apiFetch("/api/users/login", {
        method: "POST",                    (property) password: string
        body: JSON.stringify({ email, password }),
      });

      console.log("LOGIN SUCCESS:", data);

      if (data.token) {
        localStorage.setItem("user_token", data.token);
        router.push("/user");
      } else {
        throw new Error("Token tidak diterima dari server");
      }

    } catch (err: any) {
      console.error("LOGIN ERROR:", err);
      setError(err.message || "Gagal terhubung ke server");
    } finally {
      setLoading(false);
    }
  };

  return (
```

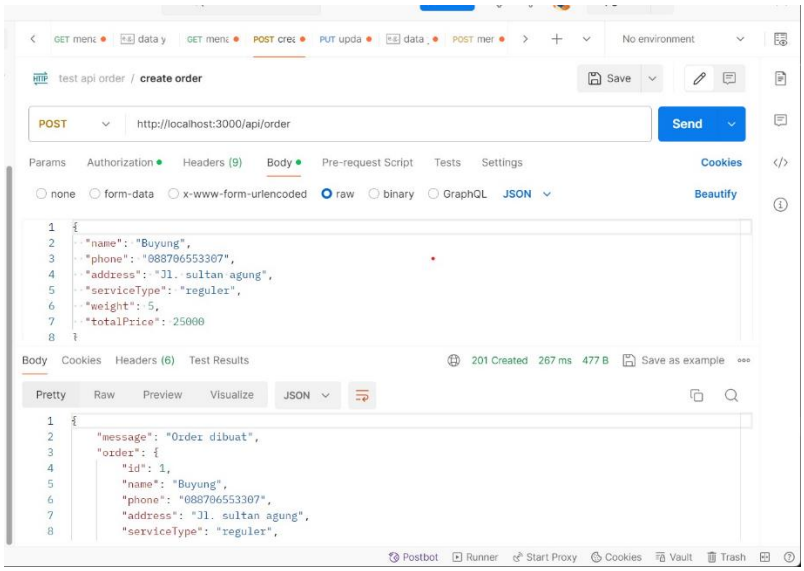
digunakan untuk **menangani proses login yang berhasil pada sisi client**. Ketika server mengembalikan data login, sistem menampilkan pesan *"LOGIN SUCCESS"* beserta data ke console untuk keperluan debugging. Selanjutnya, kode melakukan pengecekan apakah server mengirimkan token. Jika token tersedia, token tersebut disimpan ke dalam localStorage dengan nama token dan user_token sebagai bukti autentikasi pengguna, kemudian pengguna diarahkan ke halaman /user. Namun, jika token tidak diterima dari server, sistem akan melempar error dengan pesan *"Token tidak diterima dari server"* untuk menandakan adanya kegagalan pada proses autentikasi.



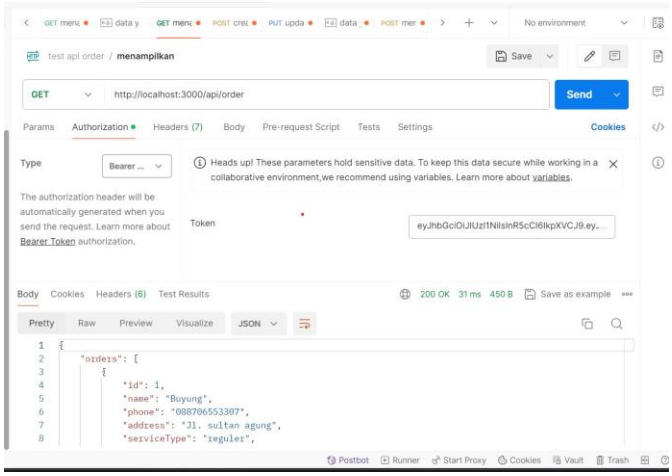
Login user



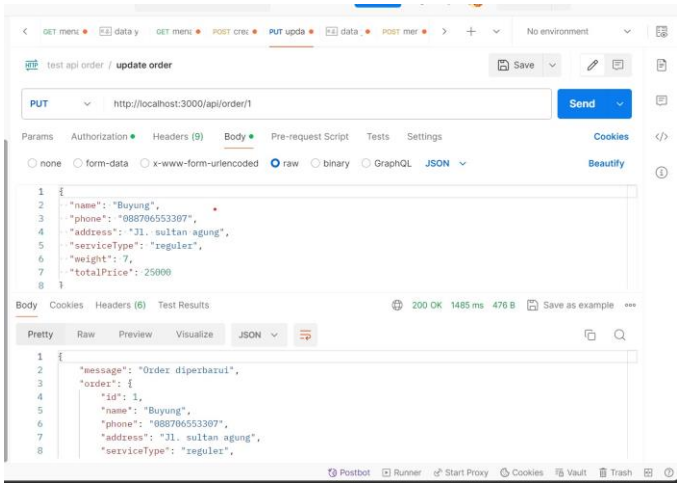
Update user (ganti password)



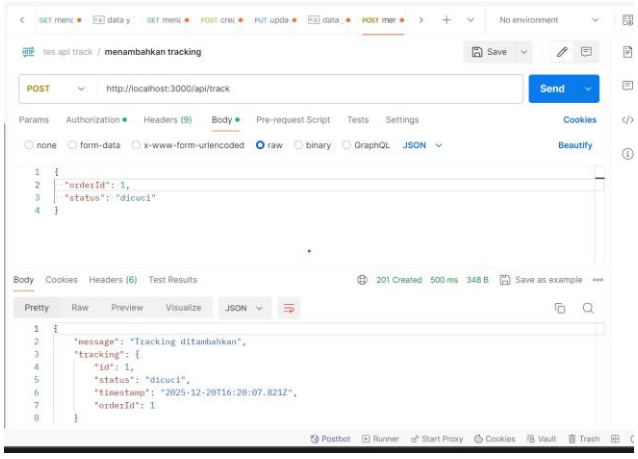
Buat order (buat folder test api folder)



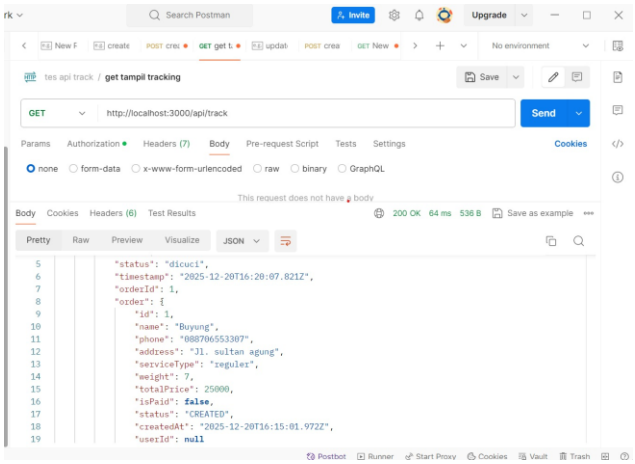
Lihat order



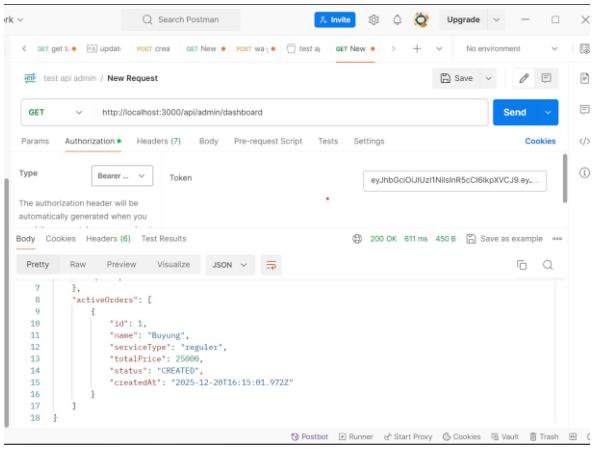
Update order before payment



Add tracking pada ordered



Tampil tracking



Admin lihat pesanan masuk