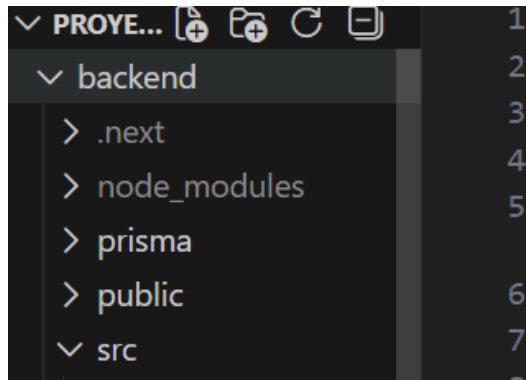


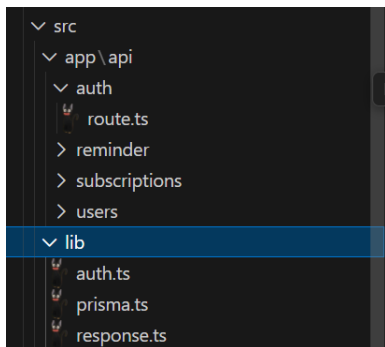
LAPORAN DOKUMENTASI PROJECT PWBL : PROYEK SUBTRACKER

Nama: Reiza Fahlevi Nunyai

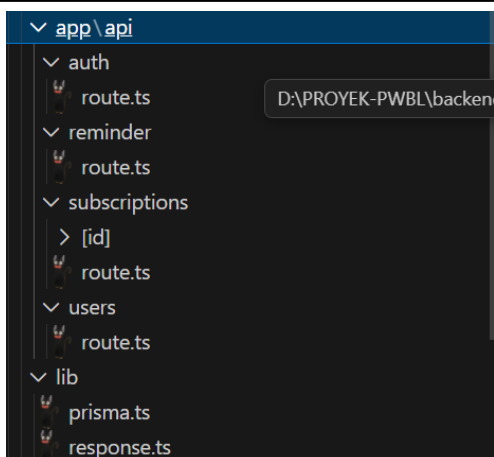
NPM: 23313021



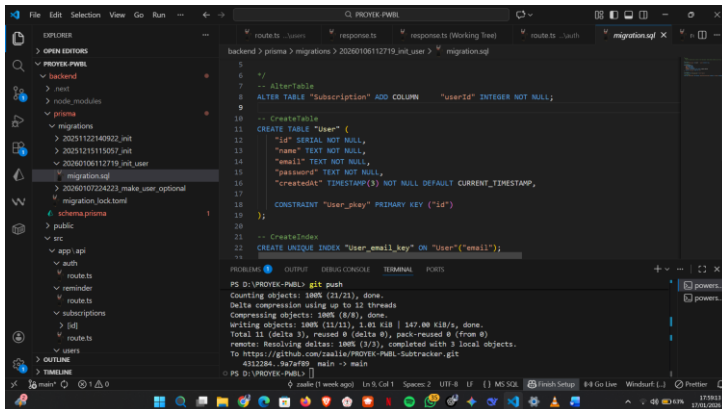
Menginstal **Backend**, **Frontend** dan juga **Prisma**



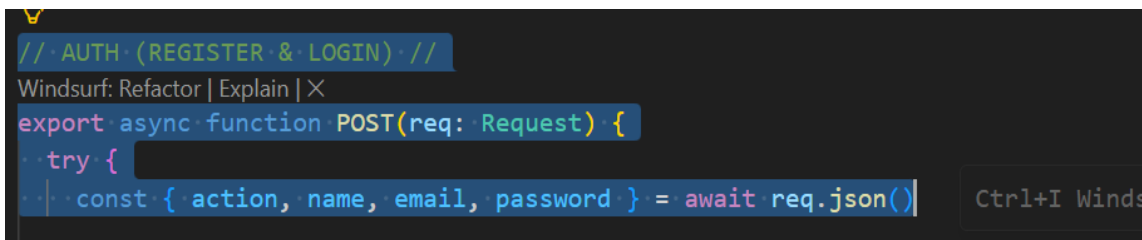
Membuat folder SRC dan api Struktur ini digunakan untuk **membangun project REST API backend** menggunakan **Next.js App Router + Prisma**.



Folder auth digunakan untuk login dan registrasi user, users untuk mengelola data user (CRUD), subscriptions untuk mengelola data langganan user, dan reminder untuk fitur pengingat pembayaran. Folder lib berisi utilitas bersama, di mana prisma.ts mengatur koneksi database dan response.ts menyamakan



Membuat Migrasi database, untuk **membuat struktur database** yang dipakai aplikasi, bukan untuk isi data.



Pada folder auth Kode ini dipakai untuk **mengambil data login atau daftar akun** yang dikirim pengguna, supaya sistem SubTracker tahu apakah harus **mendaftarkan akun baru** atau memproses login.



Kode ini berfungsi untuk **mengecek kelengkapan data** dari pengguna. Jika action, email, atau password tidak diisi, maka sistem akan **menolak request** dan mengirim respon error bahwa data wajib belum lengkap.

```
// REGISTER USER //
if (action === "register") {
  if (!name) {
    return NextResponse.json(
      { success: false, message: "Nama wajib diisi" },
      { status: 400 }
    )
  }

  const exists = await prisma.user.findUnique({ where: { email } })
  if (exists) {
    return NextResponse.json(
      { success: false, message: "Email sudah terdaftar" },
      { status: 409 }
    )
  }

  const user = await prisma.user.create({
    data: {
      name,
      email,
      password: await bcrypt.hash(password, 12),
    },
    select: { id: true, name: true, email: true },
  })

  return NextResponse.json({ success: true, data: user }, { status: 201 })
}
}
```

Kode ini menangani **registrasi pengguna**, mengecek nama dan email, mencegah email ganda, menyimpan data dengan password terenkripsi, lalu mengembalikan status berhasil.

```
// LOGIN USER //
if (action === "login") {
  const user = await prisma.user.findUnique({ where: { email } })
  if (!user || !(await bcrypt.compare(password, user.password))) {
    return NextResponse.json(
      { success: false, message: "Email atau password salah" },
      { status: 401 }
    )
  }

  return NextResponse.json(
    {
      success: true,
      data: { id: user.id, name: user.name, email: user.email },
    },
    { status: 200 }
  )
}

return NextResponse.json(
  { success: false, message: "Action tidak valid" },
  { status: 400 }
)

} catch (error) {
  console.error(error)
  return NextResponse.json(
    { success: false, message: "Internal server error" },
    { status: 500 }
  )
}
```

Kode ini menangani **proses login**, yaitu mencocokkan email dan password dengan data di database. Jika valid, sistem mengembalikan data user; jika salah atau aksi tidak dikenali, sistem mengirim pesan error.

```
// MENAMPILKAN DATA REMINDER //
Windsurf: Refactor | Explain | X
export async function GET() {
  try {
    const today = new Date();
    const reminders = await prisma.subscription.findMany({
      where: {
        nextPayment: {
          lte: new Date(today.getTime() + 3 * 24 * 60 * 60 * 1000),
        },
        isActive: true,
      },
    });

    return apiResponse.ok(reminders);
  } catch {
    return apiResponse.serverError();
  }
}
```

Pada Folder Reminder, di file route.ts kode ini digunakan untuk **menampilkan data reminder tagihan**, yaitu mengambil langganan yang akan jatuh tempo dalam 3 hari ke depan dan masih aktif, lalu mengirimkannya sebagai respon.

```

5
6 // MENAMPILKAN SEMUA DATA SUBSCRIPTION //
Windsurf: Refactor | Explain | X
7 export async function GET() {
8   try {
9     const subscriptions = await prisma.subscription.findMany({
10       orderBy: { createdAt: "desc" },
11     })
12
13     return apiResponse.ok(subscriptions)
14   } catch (error) {
15     console.error("GET /subscriptions error:", error)
16     return apiResponse.serverError()
17   }
18 }

```

Pada Folder API subscriptions ini digunakan untuk **menampilkan seluruh data subscription**, diurutkan dari yang **paling terbaru**, lalu dikirim sebagai respon ke client.

```

19
20 // TAMBAH SUBSCRIPTION //
Windsurf: Refactor | Explain | X
21 export async function POST(req: Request) {
22   try {
23     const {
24       name,
25       price,
26       category,
27       cycle,
28       nextPayment,
29       notes,
30       userId,
31     } = await req.json()
32   }
33 }

```

ini digunakan untuk **menerima data subscription baru** dari client, seperti nama layanan, harga, kategori, siklus tagihan, jadwal pembayaran, catatan, dan user, untuk kemudian diproses dan disimpan ke sistem.

```

33 // VALIDASI WAJIB
34 if (!name || !price || !nextPayment || !userId) {
35   return apiResponse.badRequest("Data wajib belum lengkap")
36 }
37
38 const subscription = await prisma.subscription.create({
39   data: {
40     name,
41     price,
42     category: category ?? Category.OTHER,
43     cycle: cycle ?? Cycle.MONTHLY,
44     nextPayment: new Date(nextPayment),
45     notes,
46     userId,
47   },
48 })
49
50 return apiResponse.created(subscription)
51 } catch (error) {
52   console.error("POST /subscriptions error:", error)
53   return apiResponse.serverError()
54 }
55 }

```

berfungsi untuk **mengecek kelengkapan data subscription**, lalu **menyimpan subscription baru ke database** dengan nilai default jika ada data opsional yang kosong, dan mengembalikan status berhasil.

```

Windsurf: Refactor | Explain
4   type Params = {
5     params: { id: string }
6   }
7   ○
8   // MENAMPILKAN SATU DATA SUBSCRIPTION MENURUT ID //
Windsurf: Refactor | Explain | ✕
9   export async function GET(
10    _: Request,
11    { params }: { params: { id: string } }
12  ) {
13    console.log("PARAMS ID:", params.id)
14
15    const id = Number(params.id)
16
17    if (isNaN(id)) {
18      return apiResponse.badRequest(
19        `ID tidak valid: ${params.id}`
20      )
21    }
22
23    const subscription = await prisma.subscription.findUnique({
24      where: { id },
25    })
26
27    return apiResponse.ok(subscription)
28  }

```

Pada folder Subscriptions [id] kode digunakan untuk **menampilkan satu data subscription berdasarkan ID**, dengan memastikan ID valid sebelum mengambil data dari database dan mengembalikannya sebagai respon.

```

// DELETE SUBSCRIPTION //
Windsurf: Refactor | Explain | ✕
export async function DELETE(_: Request, { params }: Params) {
  try {
    const id = Number(params.id)

    if (isNaN(id)) {
      return apiResponse.badRequest("ID tidak valid")
    }

    await prisma.subscription.delete({
      where: { id },
    })

    return apiResponse.ok({
      message: "Subscription berhasil dihapus",
    })
  } catch (error) {
    console.error("DELETE /subscriptions/[id] error:", error)
    return apiResponse.serverError()
  }
}

```

untuk **mengubah (update) dan menghapus (delete) data subscription berdasarkan ID**, dengan memastikan ID valid sebelum melakukan perubahan atau penghapusan data di database.

```
Windsurf: Refactor | Explain | Generate | JSDoc | X
export async function GET(req: Request) {
  try {
    const { searchParams } = new URL(req.url);
    const id = searchParams.get("id");

    // TAMPILKAN SATU DATA USER MENURUT ID //
    if (id) {
      const user = await prisma.user.findUnique({
        where: { id: Number(id) },
      });

      if (!user) {
        return NextResponse.json(
          { success: false, message: "User tidak ditemukan" },
          { status: 404 }
        );
      }

      return NextResponse.json({
        success: true,
        data: {
          id: user.id,
          name: user.name,
          email: user.email,
          password: user.password,
          createdAt: user.createdAt,
        },
      });
    }
  }
}
```

Pada Folder users route.ts kode digunakan untuk **menampilkan data user berdasarkan ID**, serta **memperbarui data user dengan validasi** agar ID dan user yang diubah benar-benar ada.

```
36 // TAMPILKAN SEMUA DATA USER //
37 const users = await prisma.user.findMany({
38   orderBy: { createdAt: "desc" },
39 });
40
41 return NextResponse.json({
42   success: true,
43   data: users,
44 });
45 } catch (error) {
46   console.error(error);
47   return NextResponse.json(
48     { success: false, message: "Gagal mengambil data user" },
49     { status: 500 }
50 );
51 }
```

Kode ini berfungsi untuk **mengambil dan menampilkan seluruh data user**, diurutkan dari yang **paling baru**, dan menampilkan pesan error jika proses gagal.

```
114 // DELETE USER //
115 export async function DELETE(req: Request) {
116   try {
117     const { id } = await req.json();
118
119     if (!id) {
120       return NextResponse.json(
121         { success: false, message: "ID user wajib diisi" },
122         { status: 400 }
123       );
124     }
125
126     const existingUser = await prisma.user.findUnique({
127       where: { id },
128     });
129
130     if (!existingUser) {
131       return NextResponse.json(
132         { success: false, message: "User tidak ditemukan" },
133         { status: 404 }
134       );
135     }
136
137     await prisma.user.delete({
138       where: { id },
139     });
140
141     return NextResponse.json(
142       { success: true,
143         message: "User berhasil dihapus" },
144       { status: 200 }
145     );
146   } catch (error) {
147     console.error(error);
148     return NextResponse.json(
149       { success: false, message: "Gagal menghapus user" },
150       { status: 500 }
151     );
152   }
153 }
```

```
114 // DELETE USER //
115 export async function DELETE(req: Request) {
116   try {
117     const { id } = await req.json();
118
119     if (!id) {
120       return NextResponse.json(
121         { success: false, message: "ID user wajib diisi" },
122         { status: 400 }
123       );
124     }
125
126     const existingUser = await prisma.user.findUnique({
127       where: { id },
128     });
129
130     if (!existingUser) {
131       return NextResponse.json(
132         { success: false, message: "User tidak ditemukan" },
133         { status: 404 }
134       );
135     }
136
137     await prisma.user.delete({
138       where: { id },
139     });
140
141     return NextResponse.json(
142       { success: true,
143         message: "User berhasil dihapus" },
144       { status: 200 }
145     );
146   } catch (error) {
147     console.error(error);
148     return NextResponse.json(
149       { success: false, message: "Gagal menghapus user" },
150       { status: 500 }
151     );
152   }
153 }
```

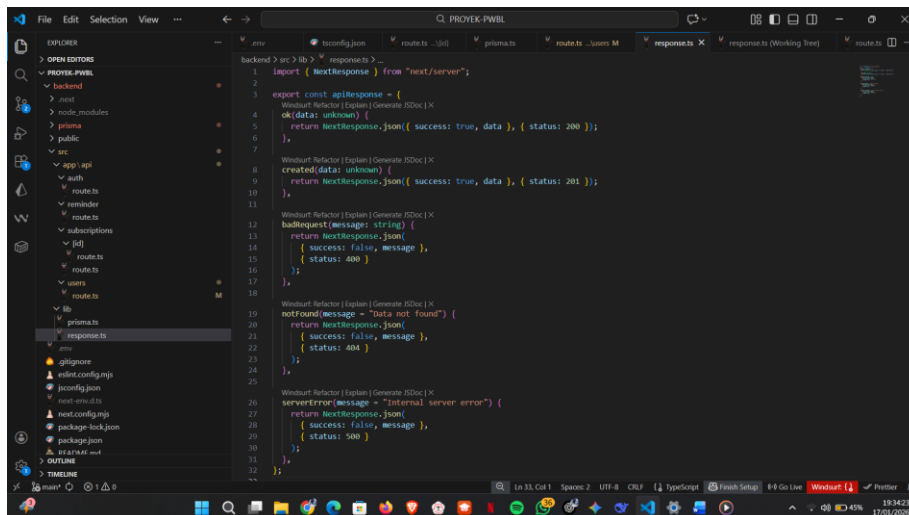
Kode ini digunakan untuk **memperbarui dan menghapus data user**. Sistem mengecek ID dan keberadaan user, mengenkripsi password jika diubah, lalu menyimpan perubahan atau menghapus user dari database.

```

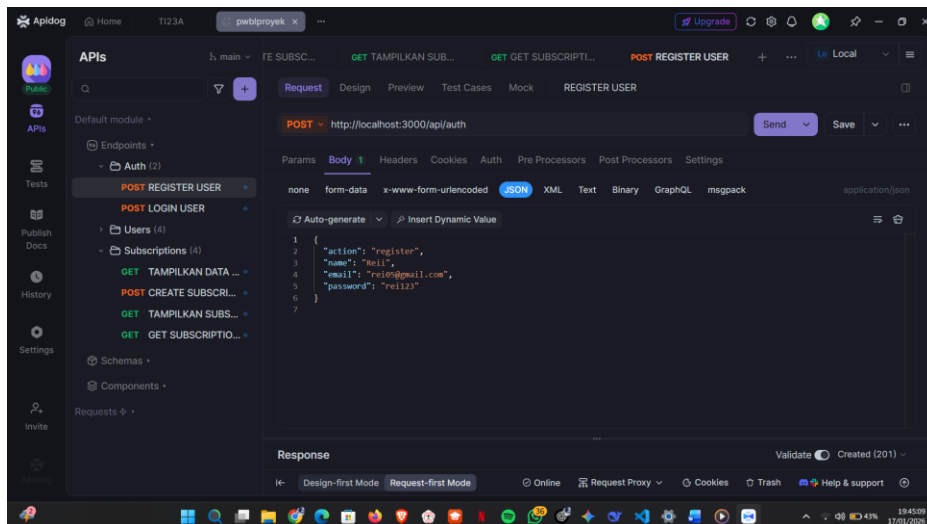
1 import { PrismaClient } from "@prisma/client";
2
3 declare global {
4   var prisma: PrismaClient | undefined;
5 }
6
7 export const prisma =
8   global.prisma ??
9   new PrismaClient({
10     log: ["error"],
11   });
12
13 if (process.env.NODE_ENV !== "production") {
14   global.prisma = prisma;
15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

```

Pada Prisma.ts ini digunakan untuk **menginisialisasi dan mengelola koneksi Prisma ke database** agar tidak membuat koneksi berulang, terutama saat development, sehingga aplikasi lebih stabil dan efisien.

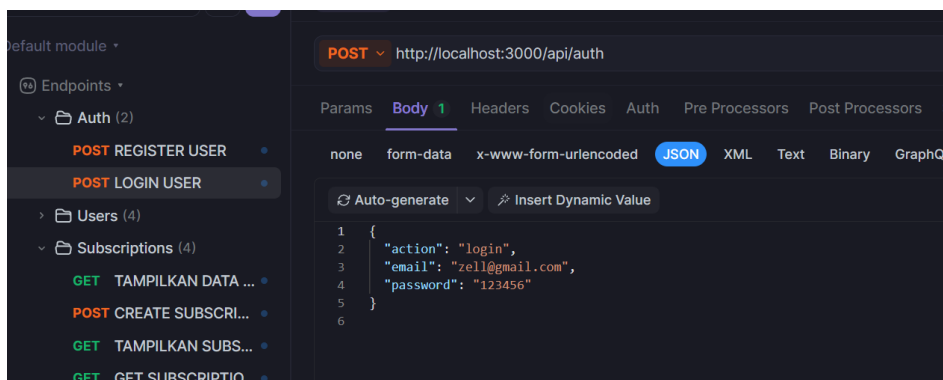


Pada response.ts kode ini berfungsi sebagai **helper untuk menyamakan format response API**, sehingga semua respon (berhasil atau error) memiliki struktur dan status HTTP yang konsisten.

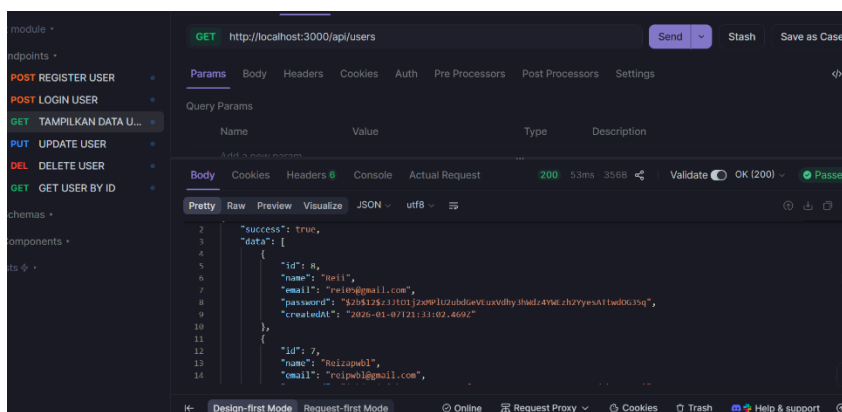


pengujian endpoint REGISTER USER menggunakan Apidog.

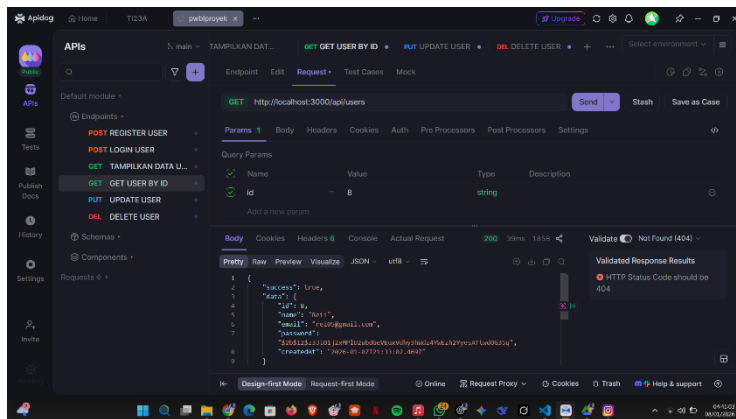
Request yang dikirim adalah **POST** ke `/api/auth` dengan format **JSON**, berisi action: register, name, email, dan password. Data ini digunakan untuk **mendaftarkan user baru** ke sistem SubTracker. Jika berhasil, server mengembalikan **status 201 (Created)** sebagai tanda registrasi sukses.



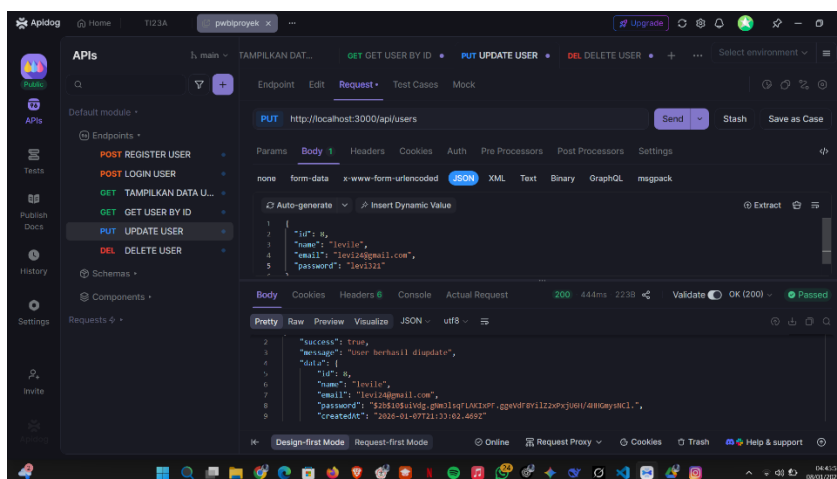
Menguji LOGIN USER pada Apidog



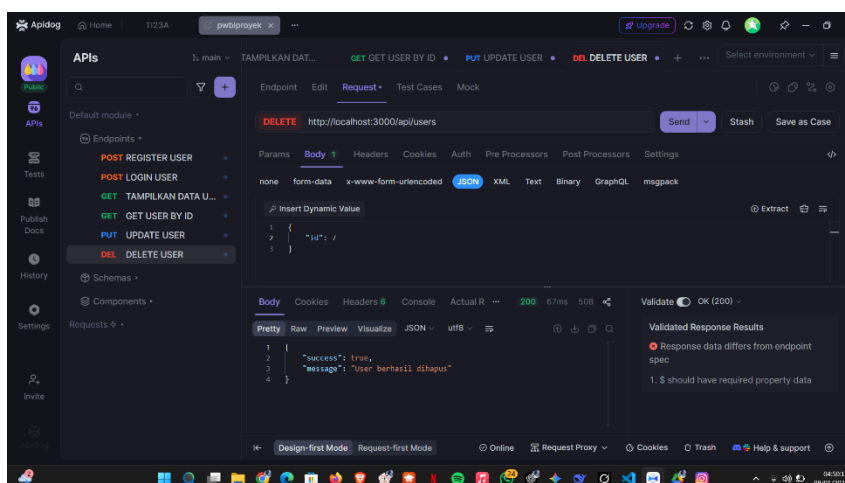
pengujian endpoint GET /api/users di Apidog. Request ini digunakan untuk **menampilkan seluruh data user** yang tersimpan di database. Server berhasil merespons dengan **status 200 (OK)** dan mengembalikan data user dalam format JSON, menandakan endpoint berjalan dengan baik.



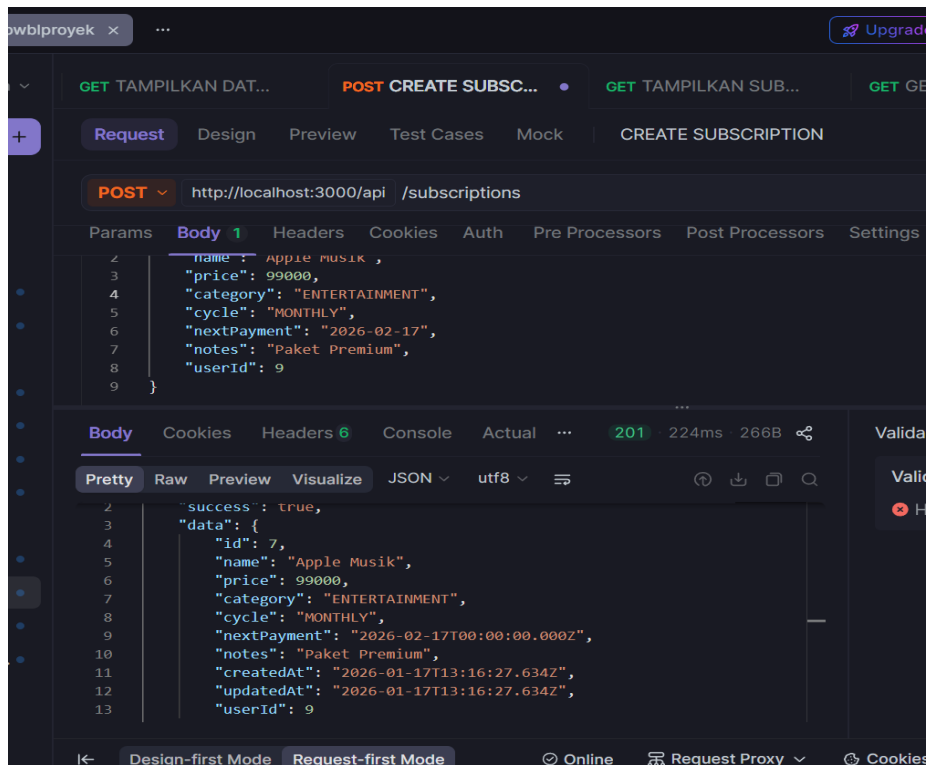
pengujian endpoint GET /api/users by id di Apidog. Request ini digunakan untuk menampilkan data user berdasarkan id yang dipilih untuk ditampilkan yang tersimpan di database.



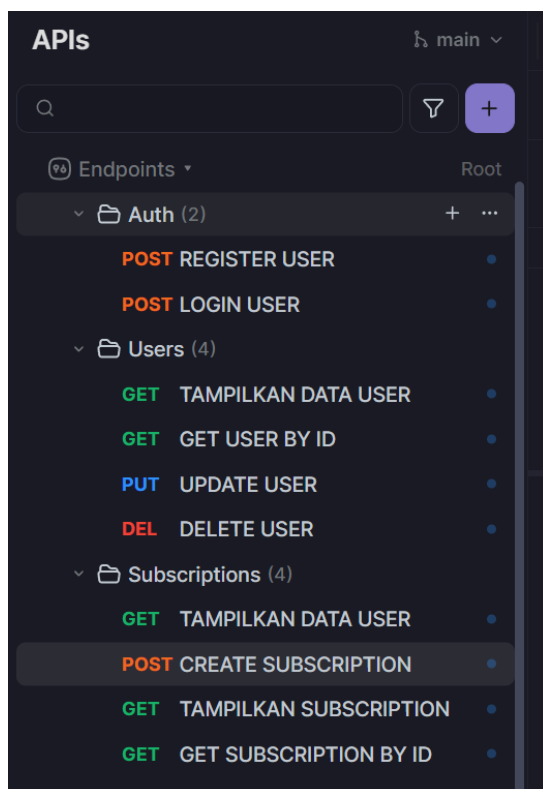
Menguji api update data user pada Apidog



Menguji Api untuk delete user



API create subscription berhasil dijalankan. Data yang dikembalikan adalah **detail langganan yang baru dibuat**, seperti nama layanan, harga, kategori, siklus tagihan, jadwal pembayaran berikutnya, catatan, waktu pembuatan, serta **ID user pemilik subscription**.



Ini adalah struktur backend API project SubTracker