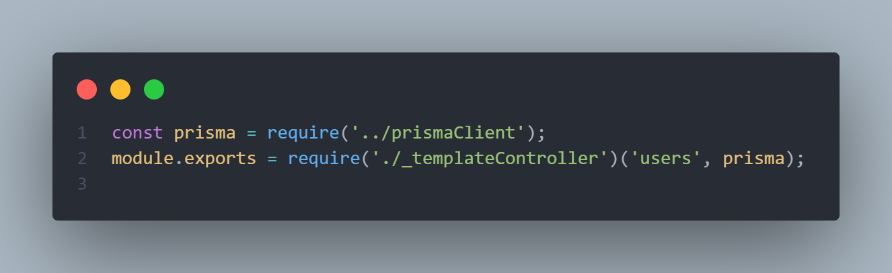
**LAPRAK PRAKTIKUM**

**PLATFROM**

**SYAHDAYANA ARIFIN**

1. **user controller**

Kode ini digunakan untuk **membuat controller CRUD khusus untuk model users**,  
dengan **memanfaatkan controller template** (\_templateController.js) yang sudah dibuat sebelumnya. Jadi, kamu **tidak perlu menulis ulang fungsi CRUD** untuk users — cukup satu baris ini saja.

1. **vucher controller**

Kode ini digunakan untuk **membuat controller CRUD otomatis untuk model vouchers** dengan memanfaatkan **template controller** yang sudah kamu buat di \_templateController.js. Jadi, dengan satu baris ini, kamu **tidak perlu lagi menulis ulang kode CRUD untuk vouchers** seperti getAll, getById, create, update, dan delete.

1. **rating controller**

Kode ini berfungsi untuk **membuat controller CRUD otomatis untuk model ratings**  
dengan menggunakan **template controller** yang sudah dibuat di file \_templateController.js. Dengan kata lain, kamu **tidak perlu menulis ulang fungsi CRUD** (getAll, getById, create, update, delete) untuk ratings, karena semua itu sudah disediakan secara generik oleh \_templateController.

1. **product controller**

Kode ini membuat **controller otomatis untuk model products**  
dengan memanfaatkan **template controller** yang sudah ada di file \_templateController.js. Dengan satu baris ini, kamu langsung punya semua fungsi **CRUD** (getAll, getById, create, update, delete) untuk tabel products, tanpa menulisnya satu per satu.

1. **order controller**

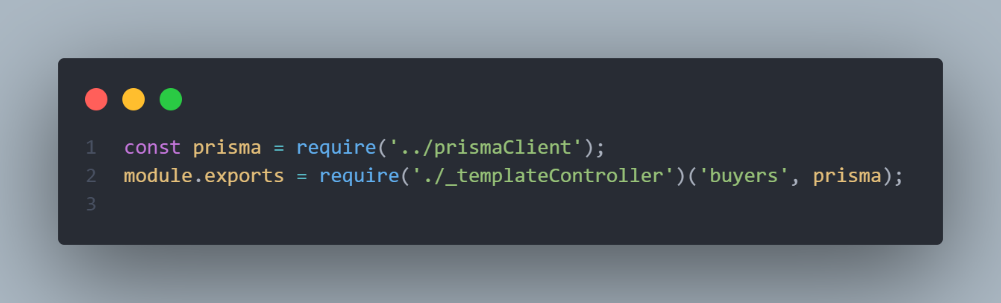
kode ini membuat **controller otomatis untuk model orders** menggunakan **template CRUD** yang sudah kamu buat di \_templateController.js. Dengan hanya satu baris ini, kamu sudah punya semua fungsi dasar:

* getAll → ambil semua pesanan
* getById → ambil satu pesanan berdasarkan id
* create → buat pesanan baru
* update → ubah data pesanan
* delete → hapus pesanan. Semua itu otomatis disediakan oleh template controller.

1. **detail\_orders controller**

Kode ini membuat **controller otomatis untuk model detail\_orders**  
menggunakan **template CRUD** yang sudah ada di file \_templateController.js. Jadi dengan hanya **dua baris kode ini**, kamu sudah mendapatkan semua fungsi untuk melakukan operasi pada tabel detail\_orders, yaitu:

* + getAll → ambil semua data detail pesanan
  + getById → ambil data detail tertentu berdasarkan id
  + create → tambah data detail pesanan baru
  + update → ubah data detail pesanan
  + delete → hapus data detail pesanan

1. **buyers controller**

Kode ini berfungsi untuk membuat **controller otomatis untuk model buyers**,  
menggunakan **template CRUD umum** dari file \_templateController.js. Dengan dua baris ini saja, kamu langsung punya endpoint lengkap untuk mengelola data **pembeli (buyers)**:

* + GET /buyers → ambil semua data buyers
  + GET /buyers/:id → ambil buyer berdasarkan id
  + POST /buyers → tambah buyer baru
  + PUT /buyers/:id → ubah data buyer
  + DELETE /buyers/:id → hapus buyer

Code diatas menjelaskan fungsi umumnya dari code controller yang digunakan, Selanjutnya ada code routers:

1. **users routers**

Kode tersebut berfungsi untuk membuat **rute REST API khusus untuk data users** menggunakan **Express.js**. Di dalamnya, setiap jenis permintaan HTTP — seperti GET, POST, PUT, dan DELETE — dihubungkan langsung dengan fungsi CRUD (getAll, getById, create, update, delete) yang ada di file userController.js. Router ini kemudian diekspor agar bisa digunakan di server.js, sehingga aplikasi dapat menerima dan memproses permintaan dari client (misalnya melalui Thunder Client atau Postman). Dengan kata lain, kode ini menjadi **jembatan antara client dan controller** yang menangani operasi database untuk tabel users.

1. **Buyers routers**

Kode tersebut berfungsi untuk membuat rute REST API untuk data buyers menggunakan Express.js. Di dalamnya, setiap permintaan HTTP seperti GET, POST, PUT, dan DELETE dihubungkan langsung dengan fungsi CRUD (getAll, getById, create, update, delete) yang ada di file buyerController.js. Dengan cara ini, aplikasi dapat menangani berbagai operasi database untuk tabel buyers, seperti menampilkan seluruh data pembeli, mengambil data berdasarkan ID, menambah pembeli baru, memperbarui data, atau menghapus data. Router ini kemudian diekspor agar bisa digunakan di server.js, sehingga client dapat mengakses endpoint /buyers untuk berinteraksi dengan data pembeli.

1. **order routers**

Kode tersebut digunakan untuk membuat **router API khusus untuk data orders** menggunakan **Express.js**. Pada kode ini, setiap rute HTTP dihubungkan dengan fungsi yang ada di orderController.js, yaitu:

* + GET / untuk menampilkan semua data pesanan,
  + GET /:id untuk mengambil data pesanan berdasarkan ID,
  + POST / untuk menambah pesanan baru,
  + PUT /:id untuk memperbarui pesanan berdasarkan ID, dan
  + DELETE /:id untuk menghapus pesanan berdasarkan ID.

Setelah router ini dikonfigurasi, kode mengekspornya dengan module.exports = router, agar bisa digunakan di file utama (seperti server.js). Dengan begitu, endpoint /orders akan berfungsi penuh untuk melakukan operasi CRUD pada data pesanan di database.

1. **detail\_order routers**

Kode ini mendefinisikan rute API untuk menangani operasi pada data detail pesanan (detail\_orders) menggunakan Express.js. File ini memanggil fungsi-fungsi dari detailOrderController untuk menjalankan setiap operasi, yaitu:

* + GET / menampilkan semua data detail pesanan,
  + GET /:id menampilkan data detail pesanan berdasarkan ID,
  + POST / menambahkan data detail pesanan baru,
  + PUT /:id memperbarui data detail pesanan berdasarkan ID, dan
  + DELETE /:id menghapus data detail pesanan berdasarkan ID.

Akhirnya, router ini diekspor dengan module.exports = router agar bisa digunakan di file utama (misalnya server.js) dan dihubungkan ke endpoint /detail\_orders.

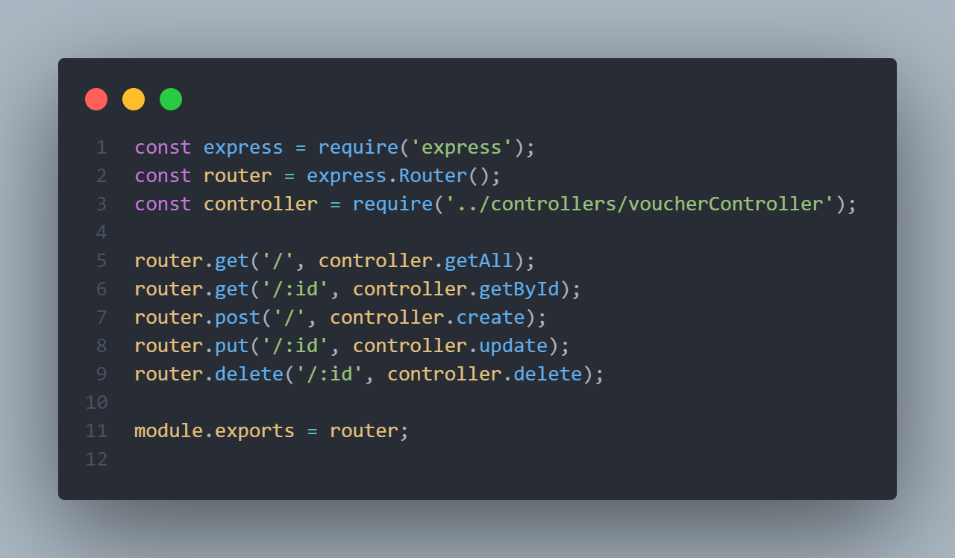
1. **ratings controller**

Kode tersebut digunakan untuk membuat **router Express.js untuk data ratings**, yaitu bagian yang mengatur endpoint CRUD untuk data penilaian atau ulasan pengguna. Berikut rangkuman penjelasannya dalam bentuk kalimat:

Kode ini berfungsi untuk mengatur rute API yang menangani operasi terhadap data ratings menggunakan Express.js. Setiap rute dihubungkan dengan fungsi dari ratingController, yaitu:

* + GET / untuk mengambil semua data penilaian,
  + GET /:id untuk mengambil penilaian berdasarkan ID,
  + POST / untuk menambahkan data penilaian baru,
  + PUT /:id untuk memperbarui data penilaian berdasarkan ID, dan
  + DELETE /:id untuk menghapus penilaian berdasarkan ID.

Terakhir, module.exports = router digunakan agar router ini bisa dipakai di file utama (misalnya server.js) dan dihubungkan ke endpoint /ratings.

1. **voucher routers**

Kode tersebut berfungsi untuk membuat **router Express.js khusus untuk data vouchers**, yang menangani berbagai operasi CRUD (Create, Read, Update, Delete) pada data voucher di aplikasi. Berikut rangkuman penjelasannya dalam bentuk kalimat:

Kode ini digunakan untuk mendefinisikan rute API yang mengelola data vouchers dengan Express.js. Setiap rute dihubungkan dengan fungsi dari voucherController, yaitu:

* + GET / untuk menampilkan semua data voucher,
  + GET /:id untuk menampilkan data voucher berdasarkan ID,
  + POST / untuk menambahkan voucher baru,
  + PUT /:id untuk memperbarui data voucher berdasarkan ID, dan
  + DELETE /:id untuk menghapus voucher berdasarkan ID.

Terakhir, module.exports = router berfungsi mengekspor router agar bisa digunakan di file utama (seperti server.js) dan dihubungkan ke endpoint /vouchers.

1. **Products routers**

Kode tersebut digunakan untuk membuat **router Express.js untuk data products**, yaitu bagian yang mengatur rute API untuk melakukan operasi CRUD (Create, Read, Update, Delete) terhadap data produk. Berikut rangkuman penjelasannya dalam bentuk kalimat:

Kode ini mendefinisikan rute API yang mengelola data products menggunakan Express.js. Setiap rute terhubung dengan fungsi pada productController, yaitu:

* + GET / untuk menampilkan semua data produk,
  + GET /:id untuk menampilkan data produk berdasarkan ID,
  + POST / untuk menambahkan produk baru,
  + PUT /:id untuk memperbarui data produk berdasarkan ID, dan
  + DELETE /:id untuk menghapus produk berdasarkan ID.

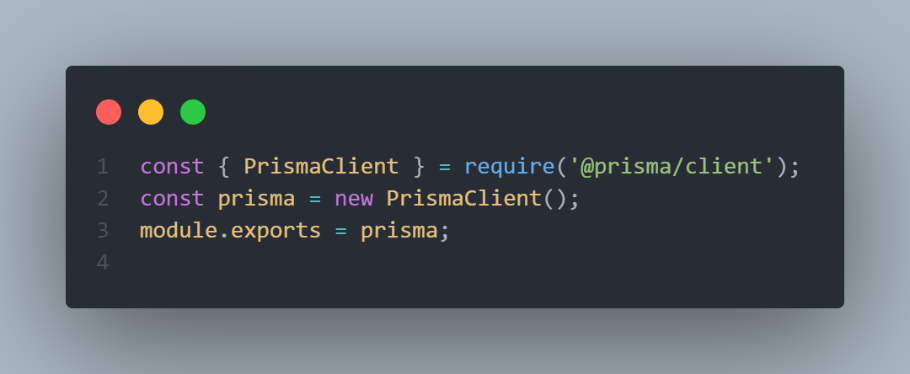
Terakhir, module.exports = router digunakan agar router ini bisa dipakai di file utama (seperti server.js) dan dihubungkan ke endpoint /products.

Penjelasan diatas cukup menjelaskan fungsi dari code yang ada diatas, namun selain code controller dan routres ada beberapa code yang memiliki fungsi yang untuk menjalankan atau menghubungkan code tersebut dengan data.

1. **Package.json**

File package.json ini berfungsi untuk:

* + Mengatur **informasi proyek Node.js** kamu.
  + Menentukan **dependensi dan dev-dependensi** yang diperlukan.
  + Menyediakan **script otomatis** agar kamu tidak perlu mengetik perintah panjang di terminal.
  + Menentukan **file utama** (server.js) dan tipe modul yang digunakan (commonjs).

1. **prisma client.js**

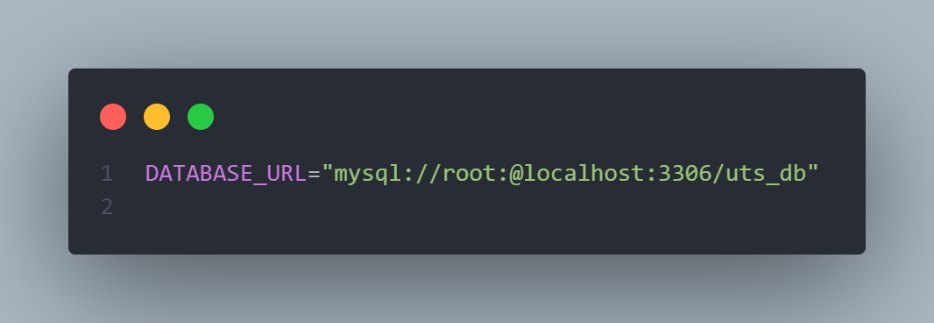
Kode tersebut berfungsi untuk **menghubungkan aplikasi Node.js dengan database** menggunakan **Prisma ORM**. Penjelasannya sebagai berikut:

* + 1. Baris const { PrismaClient } = require('@prisma/client'); digunakan untuk **mengimpor PrismaClient** dari pustaka @prisma/client. PrismaClient adalah kelas utama yang disediakan oleh Prisma untuk melakukan berbagai operasi pada database seperti membuat, membaca, memperbarui, dan menghapus data.
    2. Kemudian, const prisma = new PrismaClient(); membuat **instance baru dari PrismaClient**, yang berfungsi sebagai koneksi aktif antara aplikasi dan database. Dengan instance ini, kamu dapat menjalankan query seperti prisma.user.findMany() atau prisma.product.create() untuk mengelola data.
    3. Terakhir, module.exports = prisma; digunakan untuk **mengekspor instance Prisma** tersebut agar bisa digunakan di file lain. Dengan begitu, file lain cukup memanggil const prisma = require('./prisma'); untuk langsung menggunakan koneksi database tanpa harus membuat instance baru setiap kali.
    4. Secara ringkas, kode ini adalah **konfigurasi dasar untuk inisialisasi dan ekspor koneksi Prisma Client**, sehingga aplikasi dapat berinteraksi dengan database secara efisien dan terpusat.

1. **server.js**

Kode tersebut adalah **struktur utama server Node.js** menggunakan **Express.js** yang berfungsi untuk menjalankan API dan mengatur berbagai rute (routes) dari aplikasi. Berikut penjelasan lengkapnya dalam bentuk kalimat:

* + Bagian awal kode, yaitu const express = require('express');, digunakan untuk **mengimpor framework Express.js**, yang menjadi dasar dalam membuat server HTTP dan REST API. Selanjutnya, const cors = require('cors'); mengimpor middleware **CORS (Cross-Origin Resource Sharing)**, yang berfungsi untuk mengizinkan akses dari domain lain, misalnya saat frontend dan backend berada di server berbeda. Kemudian const bodyParser = require('body-parser'); digunakan untuk **memproses data JSON** yang dikirim melalui body request, sehingga data tersebut dapat dibaca dan digunakan oleh server.
  + Baris require('dotenv').config(); berfungsi untuk **memuat variabel lingkungan dari file .env**, seperti konfigurasi database, port, atau secret key, agar tidak disimpan langsung di kode.
  + Setelah itu dibuat instance Express melalui const app = express();. Middleware app.use(cors()); mengaktifkan fitur CORS, sedangkan app.use(bodyParser.json()); memungkinkan server membaca data JSON dari permintaan klien.
  + Pada bagian berikutnya, terdapat proses **import rute** dari berbagai file seperti userRoutes, buyerRoutes, voucherRoutes, productRoutes, orderRoutes, detailOrderRoutes, dan ratingRoutes. Setiap file route tersebut berisi definisi endpoint (misalnya GET, POST, PUT, DELETE) untuk mengatur data tertentu di database.
  + Kemudian, semua route tersebut **didaftarkan** ke aplikasi menggunakan app.use(), misalnya app.use('/users', userRoutes); artinya semua endpoint yang terkait pengguna akan diawali dengan /users. Begitu juga dengan /buyers, /vouchers, /products, dan seterusnya, sehingga setiap entitas dalam sistem memiliki jalur API yang terpisah dan rapi.
  + Bagian app.get('/', (req, res) => { res.send('🚀 UTS Prisma API is running...'); }); membuat route utama (root) untuk menampilkan pesan sederhana ketika server diakses melalui browser atau aplikasi klien.
  + Terakhir, baris const PORT = process.env.PORT || 3000; menentukan port tempat server dijalankan, dengan nilai default **3000** jika tidak ada pengaturan di file .env. Server kemudian dijalankan dengan app.listen(PORT, ...), dan ketika berhasil aktif, akan menampilkan pesan “Server running on port ...” di terminal.
  + Secara keseluruhan, kode ini berfungsi untuk **membangun server backend berbasis Express.js** yang terhubung dengan berbagai route API, mendukung pengiriman data dalam format JSON, aman diakses melalui CORS, serta siap digunakan untuk berinteraksi dengan database melalui Prisma.

1. **Env**

Baris kode tersebut adalah **konfigurasi koneksi database** yang disimpan di file .env. Penjelasannya dalam bentuk kalimat: DATABASE\_URL="mysql://root:@localhost:3306/uts\_db" digunakan untuk **mengatur alamat koneksi (connection string)** ke database MySQL yang akan digunakan oleh Prisma atau aplikasi Node.js. Nilai ini menunjukkan bahwa aplikasi akan terhubung ke **database MySQL lokal** (localhost) melalui **port 3306**, dengan **username “root”** dan **tanpa password**. Nama database yang digunakan adalah **uts\_db**. Bagian-bagiannya dapat dijelaskan sebagai berikut:

* + **mysql://** → menunjukkan jenis database yang digunakan, yaitu MySQL.
  + **root** → adalah nama pengguna (username) MySQL.
  + **:** → setelah username, seharusnya diikuti password (dalam contoh ini kosong).
  + **@localhost** → menunjukkan bahwa database berada di komputer lokal.
  + **:3306** → nomor port default MySQL.
  + **/uts\_db** → nama database yang akan diakses.

Secara keseluruhan, baris ini memungkinkan aplikasi untuk mengetahui **di mana database berada dan bagaimana cara mengaksesnya**. Nilai ini disimpan di file .env agar **tidak ditulis langsung di dalam kode**, sehingga lebih aman dan mudah diubah tanpa perlu memodifikasi program utama.

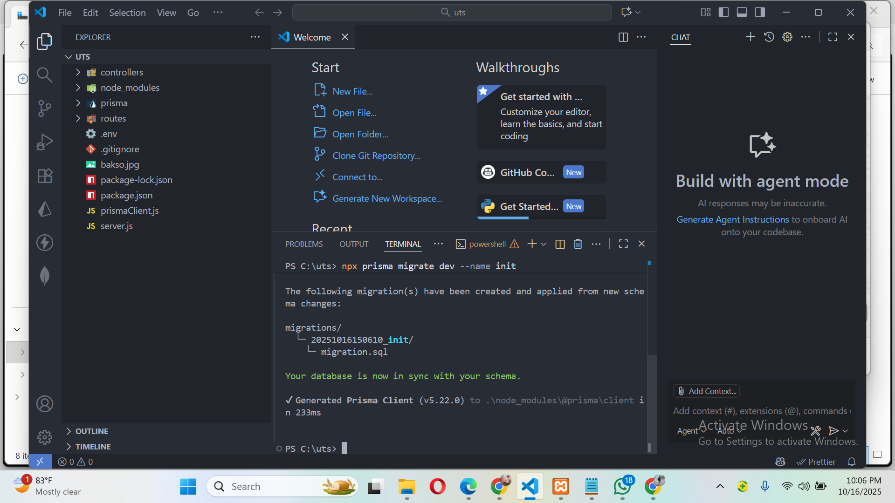
1. **Scema prisma**

Kode tersebut adalah **schema Prisma** yang digunakan untuk mendefinisikan struktur database aplikasi. File ini memberi tahu Prisma bagaimana tabel, kolom, dan hubungan antar data akan dibuat di MySQL. Bagian generator client menunjukkan bahwa Prisma akan membuat **client JavaScript** yang bisa digunakan di Node.js untuk mengakses database. Bagian datasource db menentukan bahwa database yang digunakan adalah **MySQL**, dan alamat koneksinya diambil dari **variabel lingkungan** DATABASE\_URL di file .env. Terdapat dua **enum** (daftar nilai tetap), yaitu Gender (male, female, other) untuk jenis kelamin pengguna, dan ProductType (food, drink, other) untuk jenis produk. Kemudian, bagian **model** berisi tabel-tabel utama dalam database:

* + **users**: menyimpan data pengguna seperti email, username, nomor telepon, kata sandi, jenis kelamin, tanggal lahir, dan alamat. Satu pengguna bisa memiliki banyak **orders** (pesanan).
  + **buyers**: berisi data pembeli yang memiliki nomor telepon, username, kode aktivasi, serta tanggal kedaluwarsa akun. Seorang pembeli dapat memiliki banyak **orders** dan **ratings**.
  + **vouchers**: menyimpan data kupon diskon seperti nama, kode, waktu kedaluwarsa, dan jumlah penggunaan.
  + **products**: berisi data produk seperti nama, gambar, jenis produk, harga, dan deskripsi. Satu produk bisa memiliki banyak **detail\_orders** (rincian pesanan) dan **ratings** (penilaian).
  + **orders**: menyimpan data pesanan, termasuk kode pesanan, total harga, deskripsi, dan diskon. Pesanan terhubung ke **users** dan **buyers**, serta memiliki banyak **detail\_orders** dan **ratings**.
  + **detail\_orders**: merupakan rincian setiap pesanan, berisi informasi produk, jumlah, harga, dan total per item.
  + **ratings**: menyimpan penilaian pembeli terhadap produk dalam suatu pesanan.

Secara sederhana, schema ini menggambarkan **sistem e-commerce atau pemesanan online** yang memiliki pengguna, pembeli, produk, pesanan, rincian pesanan, kupon, dan penilaian. Semua tabel saling terhubung dengan relasi yang teratur, sehingga data dapat dikelola dan diakses dengan mudah melalui Prisma.

Nah selanjutnya ada proses penambahan data dan beberapa proses lainnya:

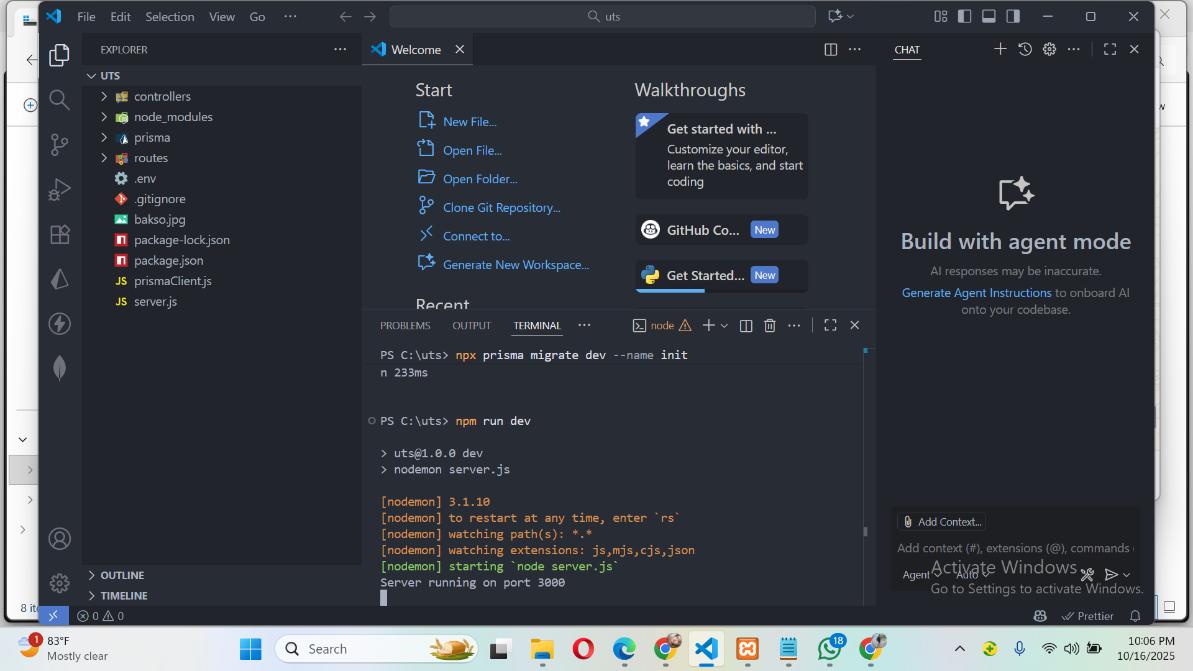
1. 

* Pada proses ini ada perintah “npm prisma migrate dev –name init” perintah ini fungsinya untuk **menyinkronkan (sinkronisasi)** antara model data di Prisma dengan tabel di database MySQL.
* **--name init** adalah nama dari migrasi tersebut. Kata “init” biasanya digunakan untuk menandai bahwa ini adalah **migrasi pertama atau inisialisasi awal** database.

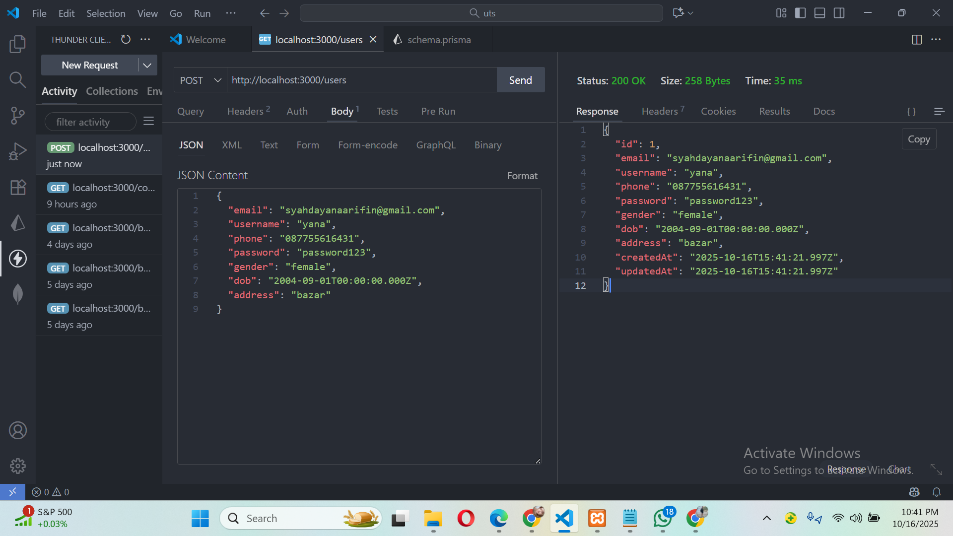
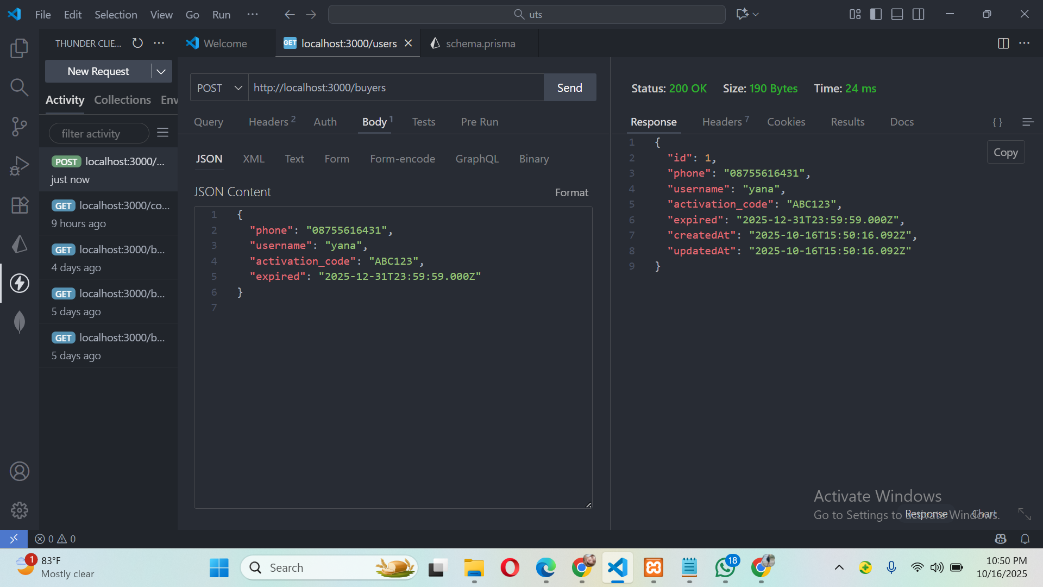
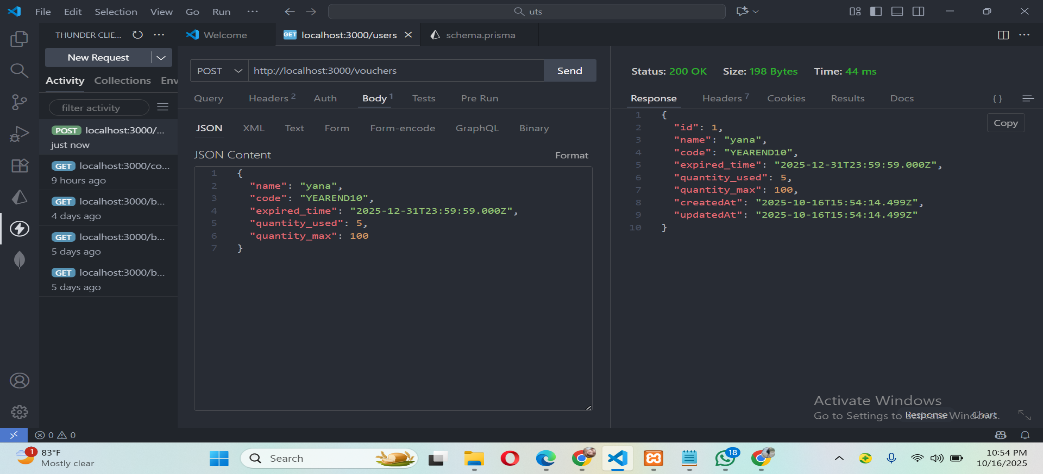
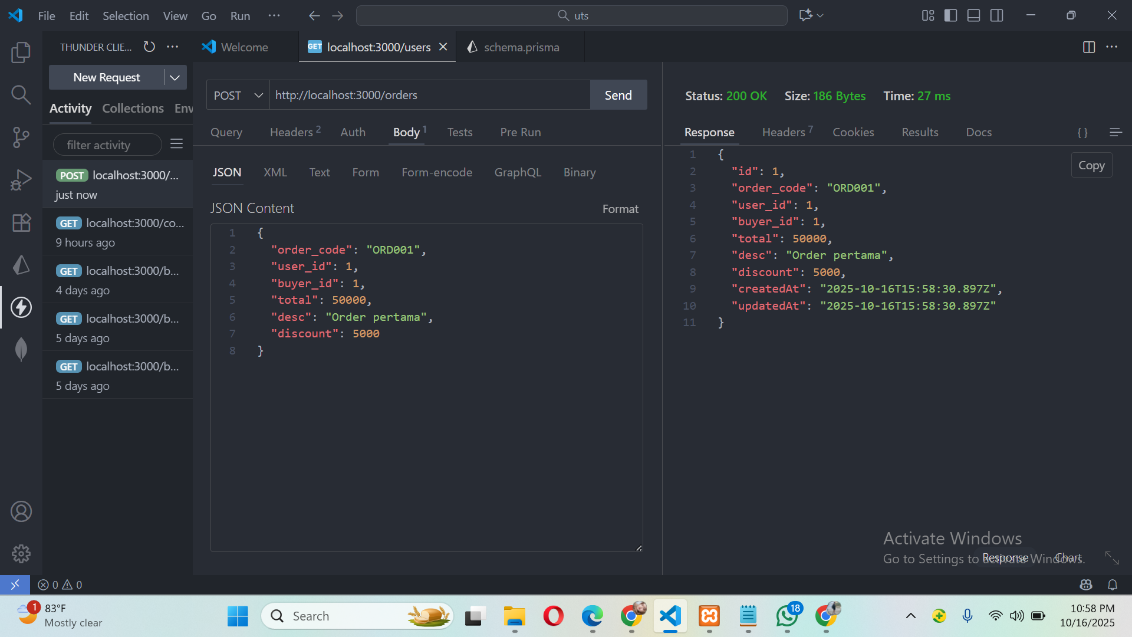
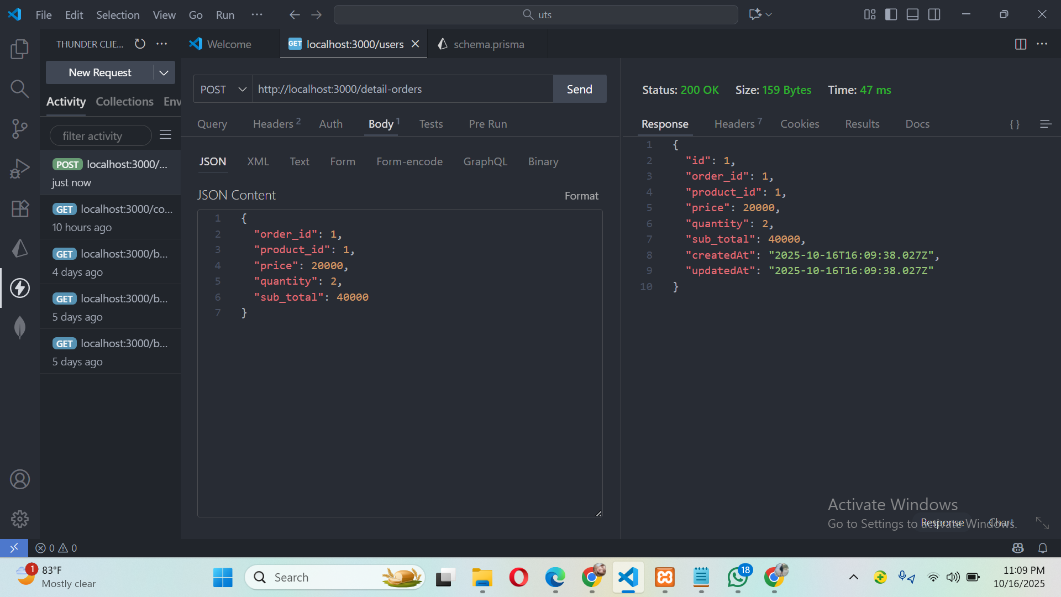
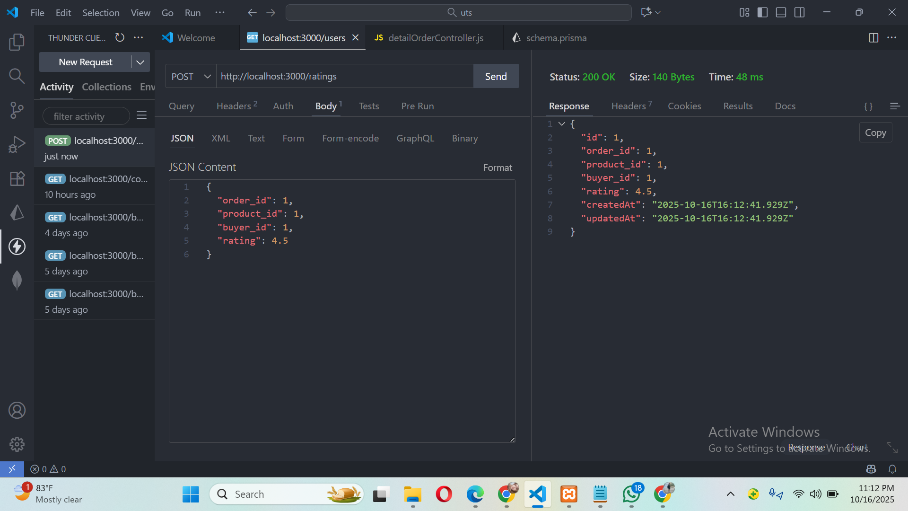
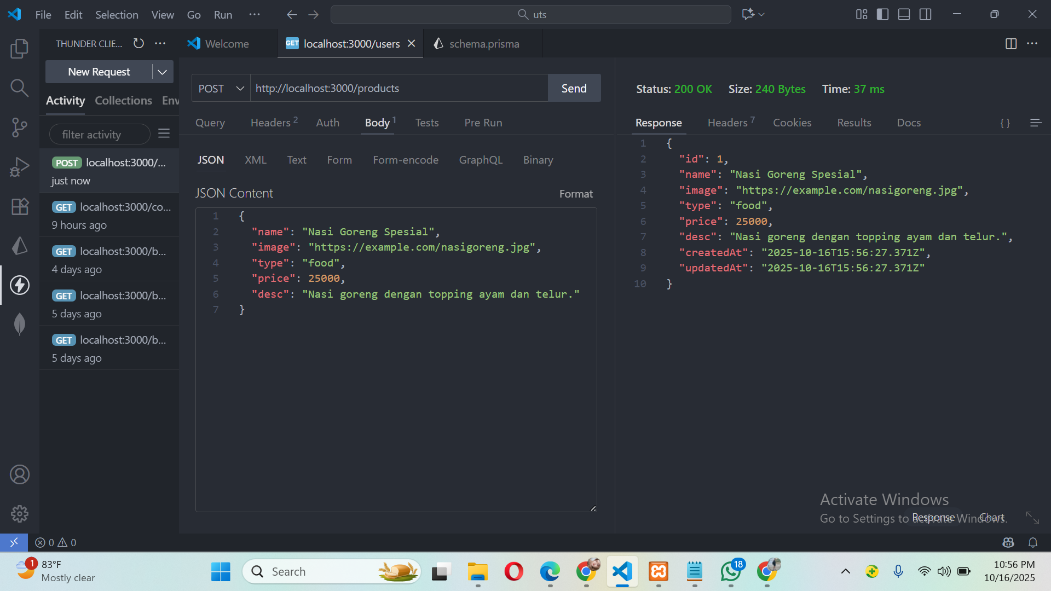
Saat kamu menjalankan perintah ini, Prisma akan:

1. Membuat folder bernama **prisma/migrations** yang berisi file SQL untuk membuat tabel sesuai model di schema.prisma.
2. Menjalankan file SQL tersebut ke dalam database (misalnya MySQL) agar tabel benar-benar terbentuk.
3. Mengupdate **Prisma Client** agar bisa digunakan langsung di kode JavaScript untuk melakukan query (seperti create, read, update, delete).

Singkatnya, perintah ini digunakan untuk **membangun struktur tabel database berdasarkan desain model Prisma** dan menyimpan catatan perubahan (migrasi) agar versi database selalu sesuai dengan kode program.

1. 

Pada perintah yang kedua ada “npm run dev” yang berfungsi untukun **memudahkan proses pengembangan** agar lebih cepat dan efisien, karena setiap kali kamu menyimpan perubahan kode, server akan langsung memuat ulang secara otomatis dan menampilkan hasilnya tanpa harus menjalankan ulang secara manual.

1. **menambahkan data pada users**
2. **Menambahkan Data buyers**
3. **Menambahkan data voucher**
4. **Menambahkan data orders**
5. **Menambahkan data pada detail\_orders**
6. **menambahkan data pada ratings**
7. **Menambahkan data pada product**