Pretest - Saiful Anwar

1. Buat sistem sederhana admin page untuk input data pembelian, cancel pembelian oleh admin toko.
Dengan database berikut
- DATABASE Produk (buat 10 produk)
- DATABASE Stock produk
- DATABASE Pembelian

dengan menggunakan:
- Nodejs, express. js (EJS)
- database sql
- desain UI bebas

2. Buat Chatbot sederhana yang diintegrasikan dengan AI seperti chatgpt, deepseek, gemini, ollama, atau lainnya.
- desain UI bebas

Ketentuan pengerjaan:
- Durasi pengerjaan 24 jam sejak arahan pretest dikirim di email
- Upload file hasil tes ke Github dan berikan link repositori kedua tes tsb (beri nama file dan panduan penggunaan file anda)

INFORMASI PENGERJAAN PROJECT ADMIN-PAGE-SEDERHANA

💻 Tugas 1: Sistem Administrasi Pembelian & Chatbot

Komponen Backend & Bahasa Pemrograman

- **Bahasa Pemrograman:** Menggunakan **JavaScript** (khususnya untuk *server-side* programming).
- **Runtime: Node.js** berfungsi sebagai lingkungan *runtime* untuk menjalankan kode JavaScript di luar *browser*.
- **Framework: Express.js** digunakan sebagai *web application framework* utama untuk membangun API dan menangani *routing*.
- **Database:** Menggunakan database relasional **MySQL** untuk menyimpan data.
- **ORM (Object-Relational Mapper): Prisma** digunakan untuk interaksi database yang aman dan terstruktur (seperti penggunaan *transaction* untuk manajemen stok).

Komponen Frontend & View

- **Template Engine (View): EJS (Embedded JavaScript)** digunakan untuk menghasilkan halaman HTML dinamis yang dilihat oleh pengguna (`pembelian.ejs`).
- **UI/UX (Styling):** Antarmuka pengguna (*layout* dan *style*) dikendalikan menggunakan **CSS Murni** yang tertanam langsung di file *view*.
- **Manajemen Sesi:** `express-session` digunakan untuk menyimpan data sesi pengguna, khususnya untuk mempertahankan riwayat percakapan Chatbot.

Integrasi Eksternal

- **API Eksternal:** Mengintegrasikan **Google Gemini API** (melalui *package* `google/genai`) sebagai *engine* untuk fungsionalitas Chatbot.

🤖 Tugas 2: Tools Pengembangan Chatbot

Tugas 2 berfokus pada implementasi dan fungsionalitas Chatbot.

- **Model Inti:** Menggunakan model **Gemini** dari Google untuk memproses masukan teks pengguna dan menghasilkan respons yang relevan.
- **Library API:** `@google/genai` adalah *library* khusus yang digunakan di kode *server* (Node.js) untuk berkomunikasi secara efisien dengan layanan Gemini.
- **Backend Handler: Express.js** dan *controller* yang relevan bertugas menangani permintaan dari *widget* Chatbot dan mengelola komunikasi bolak-balik ke API Gemini.
- **Antarmuka Pengguna (Widget):** Dibangun menggunakan kombinasi **HTML, CSS, dan JavaScript** di sisi *client* untuk membuat *widget floating* yang interaktif.

Panduan Cara pengerjaannya

*Tools menggunakan VS Code (terminal bash)

1. Sistem Admin Page Sederhana (Node.js & MySQL)

Proyek ini memerlukan koneksi ke *database* dan antarmuka *web* (`EJS`).

🛠️ Struktur Project

Pertama, buat *folder* dan struktur dasarnya:

```
mkdir admin-page-sederhana
cd admin-page-sederhana
mkdir public views models controllers routes
mkdir public/css
```

💻 Langkah Awal dan Instalasi Terminal

Jalankan perintah ini di terminal Bash Anda untuk menginisialisasi proyek dan menginstal dependensi:

```
# Inisialisasi proyek Node.js
npm init -y

# Instal dependensi utama
npm install express ejs mysql2

# mysql2 adalah driver untuk koneksi MySQL
```

2. Menggunakan Prisma ORM (Alternatif Modern) ✨

Ya, Anda **sangat bisa** menggunakan **Prisma ORM** sebagai pengganti *raw* SQL untuk mendefinisikan skema (*schema definition*) dan mengelola *database* (*migrations*) serta untuk interaksi CRUD (Create, Read, Update, Delete) di kode Node.js

Anda.

Keuntungan menggunakan Prisma:

- Anda mendefinisikan skema dalam file `schema.prisma` yang jauh lebih mudah dibaca daripada SQL.
- Prisma menangani *migrations* (*database changes*) secara otomatis.
- Interaksi di Node.js menjadi lebih aman (*type-safe*).

Langkah-Langkah Setup Prisma (Jika Anda memilih jalur ini):

1. Instalasi dan Inisialisasi

```
# Instal Prisma CLI sebagai dev dependency
npm install prisma --save-dev

# Instal Prisma Client sebagai runtime dependency
npm install @prisma/client

# Inisialisasi Prisma (akan membuat folder prisma/ dan file .env)
npx prisma init
```

2. Konfigurasi (`prisma/schema.prisma`)

Ubah file `schema.prisma` untuk mendefinisikan model Anda.

Blueprint Skema Prisma:

```
// prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

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

// 1. Tabel Produk
model Produk {
  id      Int @id @default(autoincrement())
  nama    String @db.VarChar(255)
  harga   Int @db.Int // Menggunakan Int atau Decimal sesuai kebutuhan

  stok             Stok? // Relasi 1-ke-1 opsional
  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
}

// 2. Tabel Stok (Relasi 1-ke-1 dengan Produk)
model Stok {
  id        Int @id @default(autoincrement())
  id_produk Int @unique
  jumlah    Int @default(0)

  produk Produk @relation(fields: [id_produk], references: [id])
}
```

```
// 3. Tabel Pembelian (Header Transaksi)
model Pembelian {
  id         Int @id @default(autoincrement())
  tgl_beli   DateTime @default(now())
  total      Int @db.Int
  status     Status @default(Selesai)

  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
}

// Enum untuk status pembelian
enum Status {
  Selesai
  Dibatalkan
}

// 4. Tabel DetailPembelian (Item Transaksi)
model DetailPembelian {
  id             Int @id @default(autoincrement())
  id_pembelian   Int
  id_produk      Int
  jumlah         Int
  harga_satuan   Int @db.Int

  pembelian Pembelian @relation(fields: [id_pembelian], references: [id])
  produk    Produk    @relation(fields: [id_produk], references: [id])

  @@id([id_pembelian, id_produk], name: "ID_Detail") // Opsional: Composite PK
}
```

3. Penerapan Skema dan Pengisian Data Awal

Sekarang saatnya menggunakan Prisma CLI untuk:

1. Membuat koneksi *database* yang benar berdasarkan *model* Anda.
2. Mengisi 10 data awal produk dan stok (karena ini adalah pretest, kita akan gunakan *seeding*).

3.1. Jalankan Migrasi Prisma

Jalankan perintah ini di terminal Anda untuk menerapkan skema (`schema.prisma`) ke *database* `toko_db` di MySQL Workbench Anda.

```
# Perintah ini akan:
# 1. Membandingkan schema.prisma dengan database MySQL Anda.
# 2. Membuat file migrasi baru di prisma/migrations.
# 3. Menerapkan perubahan ke database (membuat semua tabel).

npx prisma migrate dev --name init_database_schema
```

```
Output di Terminalnya
Saiful Anwar@DESKTOP-KEMJLDM MINGW64 /d/admin-page-sederhana
```

```
$ npx prisma migrate dev --name init_database_schema
Loaded Prisma config from prisma.config.ts.
Prisma schema loaded from prisma\schema.prisma
Error: Prisma schema validation - (get-config wasm)
Error code: P1012
error: The datasource property `url` is no
longer supported in schema files. Move connection URLs for Migrate to `prisma.config.ts` and pass
either `adapter` for a direct database connection or `accelerateUrl` for Accelerate to the
`PrismaClient` constructor. See https://pris.ly/d/config-datasource and https://pris.ly/d/prisma7-
client-config
  -->  prisma\schema.prisma:14
   |
13 |   provider = "mysql"
14 |   url      = env("DATABASE_URL")
   |

Validation Error Count: 1
[Context: getConfig]

Prisma CLI Version : 7.1.0
```

Solusi:

1. Ubah Nama File

Jika saat ini nama file Anda adalah `prisma.config.ts`, ganti namanya menjadi `prisma.config.js`:

```
# Di terminal Bash Anda, di dalam folder 'admin-page-sederhana'
mv prisma.config.ts prisma.config.js
```

2. Ubah Isi File (`prisma.config.js`)

Salin dan tempel kode JavaScript murni berikut ini ke dalam file `prisma.config.js`. Kode ini menggunakan sintaks Node.js (`require` dan `module.exports`) dan memastikan variabel lingkungan dimuat dengan benar menggunakan `dotenv`.

```
// prisma.config.js (JavaScript murni)

// Pastikan library dotenv sudah terinstal: npm install dotenv
require('dotenv').config();
const { defineConfig } = require("prisma/config");

// Fungsi helper untuk mendapatkan variabel lingkungan
const env = (key) => {
    const value = process.env[key];
    if (!value) {
        // Logika sederhana untuk memastikan variabel penting ada
        throw new Error(`Variabel lingkungan ${key} tidak didefinisikan. Pastikan file .env sudah
diisi.`);
    }
    return value;
};

// Pastikan DATABASE_URL ada
```

```
if (!env("DATABASE_URL")) {
  throw new Error("DATABASE_URL must be defined in the .env file");
}

module.exports = defineConfig({
  // Path ke file skema Prisma Anda
  schema: "prisma/schema.prisma",

  // Konfigurasi Migrasi
  migrations: {
    path: "prisma/migrations",
  },

  // Konfigurasi Datasource (Menggunakan Prisma 7.x Syntax)
  datasource: {
    // Adapter dan URL koneksi dipindahkan ke sini
    provider: "mysql",
    url: env("DATABASE_URL"),
  },
});
```

3. Pastikan `schema.prisma` Sudah Disesuaikan

Pastikan Anda telah **menghapus baris** **`url = env("DATABASE_URL")`** dari `prisma/schema.prisma`.

```
datasource db {
  provider = "mysql"
  // HANYA ADA 'provider', tanpa 'url' di sini
}
```

4. Perbaikan di Schema.Prisma

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

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

datasource db {
  provider = "mysql"
}

// 1. Tabel Produk
model Produk {
  id      Int @id @default(autoincrement())
  nama    String @db.VarChar(255)
  harga   Int @db.Int // Menggunakan Int atau Decimal sesuai kebutuhan

  stok             Stok? // Relasi 1-ke-1 opsional
  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
```

```
}

// 2. Tabel Stok (Relasi 1-ke-1 dengan Produk)
model Stok {
  id        Int @id @default(autoincrement())
  id_produk Int @unique
  jumlah    Int @default(0)

  produk Produk @relation(fields: [id_produk], references: [id])
}

// 3. Tabel Pembelian (Header Transaksi)
model Pembelian {
  id         Int @id @default(autoincrement())
  tgl_beli   DateTime @default(now())
  total      Int @db.Int
  status     Status @default(Selesai)

  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
}

// Enum untuk status pembelian
enum Status {
  Selesai
  Dibatalkan
}

// 4. Tabel DetailPembelian (Item Transaksi)
model DetailPembelian {
  id_pembelian   Int
  id_produk      Int
  jumlah         Int
  harga_satuan   Int @db.Int

  pembelian Pembelian @relation(fields: [id_pembelian], references: [id])
  produk    Produk    @relation(fields: [id_produk], references: [id])

  @@id([id_pembelian, id_produk], name: "ID_Detail") // Opsional: Composite PK
}
```

5. Periksa File `.env`

```
# .env (Pastikan tidak ada spasi atau karakter yang tidak perlu)
DATABASE_URL="mysql://root:12345@localhost:3306/toko_db"
```

6. Jalankan Migrasi Database

Setelah konfigurasi di atas benar, Anda bisa menerapkan skema ke MySQL.

Jalankan perintah ini di terminal Anda:

```
Saiful Anwar@DESKTOP-KEMJLDM MINGW64 /d/admin-page-sederhana
$ npx prisma migrate dev --name init_schema_toko
```

4. Persiapan File Seeder

1.1. Buat Folder dan File

Jalankan perintah ini di terminal Bash Anda (pastikan Anda berada di *root* folder `admin-page-sederhana`):

Terminal Bash

```
# Membuat folder 'seed' di dalam folder 'prisma'
mkdir -p prisma/seed

# Membuat file 'seed.js' di dalam folder tersebut
touch prisma/seed/seed.js
```

2. Isi Kode Seeder (`prisma/seed/seed.js`)

Salin dan tempel kode JavaScript berikut ke dalam file **`prisma/seed/seed.js`**. Kode ini akan membuat 10 produk dan memberikan stok awal 50 untuk setiap produk, sekaligus menunjukkan cara menggunakan **Prisma Client** di luar Express.

```
// prisma/seed/seed.js

const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

// Daftar 10 produk awal yang akan di-seed
const initialProducts = [
    { nama: 'Laptop Gaming', harga: 15000000, initialStock: 50 },
    { nama: 'Monitor 27 Inch', harga: 3500000, initialStock: 50 },
    { nama: 'Keyboard Mekanik', harga: 1200000, initialStock: 50 },
    { nama: 'Mouse Wireless', harga: 450000, initialStock: 50 },
    { nama: 'Headset Bluetooth', harga: 700000, initialStock: 50 },
    { nama: 'Webcam HD', harga: 550000, initialStock: 50 },
    { nama: 'SSD 512GB', harga: 800000, initialStock: 50 },
    { nama: 'RAM 16GB DDR4', harga: 950000, initialStock: 50 },
    { nama: 'Power Bank 10000mAh', harga: 250000, initialStock: 50 },
    { nama: 'Speaker Portable', harga: 300000, initialStock: 50 },
];

async function main() {
    console.log(`Menghapus data Produk dan Stok lama...`);
    // Opsional: Hapus data lama (Produk harus dihapus setelah Stok)
    await prisma.stok.deleteMany({});
    await prisma.produk.deleteMany({});
    await prisma.pembelian.deleteMany({});

    console.log(`Memulai seeding data Produk dan Stok...`);

    for (const data of initialProducts) {
```

```
        // Membuat Produk baru dan Stok yang terhubung (Nested Write)
        const product = await prisma.produk.create({
            data: {
                nama: data.nama,
                // Pastikan harga adalah Int sesuai skema Prisma Anda
                harga: data.harga,
                stok: {
                    // Membuat entri Stok untuk produk ini
                    create: {
                        jumlah: data.initialStock,
                    },
                },
            },
        });
        console.log(`Berhasil membuat produk ID: ${product.id}, Nama: ${product.nama}`);
    }

    console.log(`Seeding selesai.`);
}

main()
    .catch((e) => {
        console.error("Error saat seeding:", e);
        process.exit(1);
    })
    .finally(async () => {
        await prisma.$disconnect();
    });
```

3. Konfigurasi `package.json`

Pastikan file **`package.json`** Anda memiliki *script* untuk menjalankan *seeder* ini. Jika Anda belum melakukannya, buka `package.json` dan tambahkan baris `"seed": "node prisma/seed/seed.js"` di bagian `"scripts"`:

```
{
  "name": "admin-page-sederhana",
  "version": "1.0.0",
  "description": "Admin Page Sederhana menggunakan Node.js, Express, EJS, dan Prisma (MySQL)",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "seed": "node prisma/seed/seed.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "express",
    "prisma",
    "ejs"
  ],
  "author": "",
  "license": "ISC",
```

```
  "dependencies": {
    "@google/genai": "^1.32.0",
    "@prisma/client": "^6.19.0",
    "dotenv": "^17.2.3",
    "ejs": "^3.1.10",
    "express": "^5.2.1",
    "express-session": "^1.18.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.11",
    "prisma": "^6.19.0"
  }
}
```

⚙ Perintah Instalasi Tambahan

Setelah Anda merevisi `package.json`, jalankan perintah ini di terminal untuk menghapus `mysql2` dan menginstal `nodemon` (jika Anda menambahkannya):

```
# Hapus driver mysql2 yang tidak lagi digunakan
npm uninstall mysql2

# Instal nodemon (opsional, untuk pengembangan)
npm install --save-dev nodemon
```

⚙ Perbaiki Prisma Client

Jalankan perintah ini di terminal Anda:

```
# Perintah ini membaca schema.prisma dan menghasilkan kode klien yang hilang
npx prisma generate
```

Perbaiki seed.js

```
require('dotenv').config();

const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const initialProducts = [
    { nama: 'Laptop Gaming', harga: 15000000, initialStock: 50 },
    { nama: 'Monitor 27 Inch', harga: 3500000, initialStock: 50 },
    { nama: 'Keyboard Mekanik', harga: 1200000, initialStock: 50 },
    { nama: 'Mouse Wireless', harga: 450000, initialStock: 50 },
    { nama: 'Headset Bluetooth', harga: 700000, initialStock: 50 },
    { nama: 'Webcam HD', harga: 550000, initialStock: 50 },
    { nama: 'SSD 512GB', harga: 800000, initialStock: 50 },
    { nama: 'RAM 16GB DDR4', harga: 950000, initialStock: 50 },
    { nama: 'Power Bank 10000mAh', harga: 250000, initialStock: 50 },
    { nama: 'Speaker Portable', harga: 300000, initialStock: 50 },
];
```

```
async function main() {
    console.log(`Menghapus data Produk dan Stok lama...`);
    await prisma.stok.deleteMany({});
    await prisma.produk.deleteMany({});
    await prisma.pembelian.deleteMany({});

    console.log(`Memulai seeding data Produk dan Stok...`);

    for (const data of initialProducts) {
        const product = await prisma.produk.create({
            data: {
                nama: data.nama,
                harga: data.harga,
                stok: {
                    create: {
                        jumlah: data.initialStock,
                    },
                },
            },
        });
        console.log(`Berhasil membuat produk ID: ${product.id}, Nama: ${product.nama}`);
    }

    console.log(`Seeding selesai.`);
}

main()
    .catch((e) => {
        console.error("Error saat seeding:", e);
        process.exit(1);
    })
    .finally(async () => {
        await prisma.$disconnect();
    });
```

Downgrade ke Prisma 6.19

```
# Downgrade Prisma CLI (dev dependency)
npm install prisma@6.19.0 --save-dev

# Downgrade Prisma Client (runtime dependency)
npm install @prisma/client@6.19.0
```

prisma.config.js rename menjadi prisma.config.ts

dan ubah isinya menjadi

```
import "dotenv/config";
import { defineConfig } from "prisma/config";

const env = (key: string): string => {
    const value = process.env[key];
    if (!value) {
        throw new Error(`Variabel lingkungan ${key} tidak didefinisikan.`);
```

```
    }
    return value;
};

if (!env("DATABASE_URL")) {
  throw new Error("DATABASE_URL must be defined in the .env file");
}

export default defineConfig({
  schema: "prisma/schema.prisma",
  migrations: {
    path: "prisma/migrations",
  },

  datasource: {
    url: env("DATABASE_URL"),
  },
});
```

schema.prismanya diperbaiki lagi

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

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

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

// 1. Tabel Produk
model Produk {
  id      Int @id @default(autoincrement())
  nama    String @db.VarChar(255)
  harga   Int @db.Int // Menggunakan Int atau Decimal sesuai kebutuhan

  stok             Stok? // Relasi 1-ke-1 opsional
  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
}

// 2. Tabel Stok (Relasi 1-ke-1 dengan Produk)
model Stok {
  id        Int @id @default(autoincrement())
  id_produk Int @unique
  jumlah    Int @default(0)

  produk Produk @relation(fields: [id_produk], references: [id])
```

```
}

// 3. Tabel Pembelian (Header Transaksi)
model Pembelian {
  id         Int @id @default(autoincrement())
  tgl_beli   DateTime @default(now())
  total      Int @db.Int
  status     Status @default(Baru)

  DetailPembelian DetailPembelian[] // Relasi 1-ke-banyak
}

// Enum untuk status pembelian
enum Status {
  Baru
  Selesai
  Dibatalkan
}

// 4. Tabel DetailPembelian (Item Transaksi)
model DetailPembelian {
  id_pembelian   Int
  id_produk      Int
  jumlah         Int
  harga_satuan   Int @db.Int

  pembelian Pembelian @relation(fields: [id_pembelian], references: [id])
  produk    Produk    @relation(fields: [id_produk], references: [id])

  @@id([id_pembelian, id_produk], name: "ID_Detail") // Opsional: Composite PK
}
```

Generate Ulang

```
npx prisma generate
```

Re-Migrate

```
# Hapus folder migrations lama (JIKA SUDAH ADA)
rm -rf prisma/migrations

# Jalankan migrasi bersih
npx prisma migrate dev --name create_all_tables
```

5. Langkah Selanjutnya: Pengisian Data Awal (Seeding)

Anda sudah memverifikasi `package.json` dan kode *seeder* (`prisma/seed/seed.js`). Sekarang adalah saatnya mengisi 10 data produk dan stok ke *database*.

Jalankan perintah ini di terminal Anda:

```
Saiful Anwar@DESKTOP-KEMJLDM MINGW64 /d/admin-page-sederhana
$ npm run seed
```

Langkah Selanjutnya: Implementasi Aplikasi Node.js Express

Fokus kita sekarang adalah pada kode aplikasi **Admin Page Sederhana**. Kita akan membuat *file-file* penting yang akan menjalankan logika bisnis transaksi Anda.

1. Inisialisasi Prisma Client (`models/prisma.js`)

Buat folder `models` dan buat file `prisma.js` di dalamnya.

```
mkdir models
touch models/prisma.js
```

Isi `models/prisma.js`:

```
// models/prisma.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

// Ekspor instance Prisma untuk digunakan di seluruh controllers
module.exports = prisma;
```

2. File Server Utama (`server.js`)

Anda telah menginstal Express, sekarang buat file utama untuk menjalankan server dan mengarahkan *routing*.

```
touch server.js
```

Isi `server.js`:

```
// server.js

// --- 1. Import Dependencies ---
const express = require('express');
const session = require('express-session'); // Digunakan untuk menyimpan session dan pesan flash
(message)
const dotenv = require('dotenv');
const path = require('path');

// --- Import Controllers & Routes ---
const indexRouter = require('./routes/index'); // Route untuk Tugas 1 (Admin Page)
const chatController = require('./controllers/chatController'); // Controller untuk Tugas 2 (Chatbot)

// --- 2. Konfigurasi ---
dotenv.config(); // Memuat variabel dari file .env
const app = express();
const PORT = process.env.PORT || 3000;
```

```
// --- 3. Setup View Engine (EJS) ---
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// --- 4. Middleware ---

// A. Session Middleware (PENTING untuk menyimpan riwayat chat dan pesan notifikasi)
app.use(session({
    secret: 'kunci_rahasia_super_aman_12345', // Ganti dengan kunci rahasia yang lebih kompleks di
production
    resave: false,
    saveUninitialized: true,
    cookie: { secure: process.env.NODE_ENV === 'production' }
}));

// B. Body Parsing Middleware (PENTING! Mencegah error 'Cannot read properties of undefined')
// Digunakan untuk memproses data dari form POST (seperti input 'message' di chatbot)
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// C. Static Files Middleware
app.use(express.static(path.join(__dirname, 'public')));


// --- 5. Routing ---

// A. Rute untuk Tugas 1: Admin Page (Transaksi Pembelian)
// Menggunakan indexRouter (yang mengarah ke pembelianController)
app.use('/', indexRouter);

// B. Rute untuk Tugas 2: Chatbot Gemini
app.get('/chat', chatController.getChatPage);
app.post('/chat', chatController.sendMessage);


// --- 6. Menjalankan Server ---
app.listen(PORT, () => {
    console.log(`Admin Page Server berjalan di http://localhost:${PORT}`);
});
```

3. Setup Routes (`routes/index.js`)

Buat folder `routes` dan file `index.js` untuk mendefinisikan *endpoint*.

```
mkdir routes
touch routes/index.js
```

Isi `routes/index.js`:

```
// routes/index.js

const express = require('express');
const router = express.Router();
const pembelianController = require('../controllers/pembelianController');
```

```
const chatController = require('../controllers/chatController'); // Harus tetap ada

// ================================================
// ROUTES ADMIN PAGE (PEMBELIAN)
// ================================================

// Route GET: Halaman Utama (memanggil pembelianController.index)
router.get('/', pembelianController.index);

// Route POST: Input Pembelian Baru
// DIPERBAIKI: Mengubah inputPembelian menjadi createPembelian
router.post('/pembelian/input', pembelianController.createPembelian);

// Route POST: Pembatalan Pembelian
router.post('/pembelian/cancel', pembelianController.cancelPembelian);

// Route POST: Selesaikan Pembelian
// DIPERBAIKI: Mengubah completePembelian menjadi markAsSelesai
router.post('/pembelian/selesai', pembelianController.markAsSelesai);

// --- ROUTE BARU UNTUK PEMBERSIHAN ---
// Route POST: Membersihkan transaksi yang dibatalkan
router.post('/clear-cancelled', pembelianController.clearCancelledPurchases);

// ================================================
// ROUTES CHATBOT
// ================================================

// Route GET: Halaman Chatbot (diakses oleh iframe widget)
router.get('/chat', chatController.getChatPage);

// Route POST: Mengirim pesan ke Chatbot (menggunakan AJAX/Fetch)
router.post('/chat', chatController.sendMessage);


module.exports = router;
```

4. Controller Logika Bisnis (`controllers/pembelianController.js`)

Buat folder `controllers` dan file `pembelianController.js`. Ini adalah bagian paling krusial di mana kita akan menerapkan logika **transaksi atomik** menggunakan `prisma.$transaction`.

```
mkdir controllers
touch controllers/pembelianController.js
```

5. 💻 `controllers/pembelianController.js` (Kode Lengkap)

Salin dan tempel kode berikut ke dalam file **`controllers/pembelianController.js`**.

```
// controllers/pembelianController.js

const prisma = require('../models/prisma');
const { PrismaClientKnownRequestError } = require('@prisma/client/runtime/library');
```

```
// Fungsi Helper untuk format harga
const formatRupiah = (number) => {
    return new Intl.NumberFormat('id-ID', {
        style: 'currency',
        currency: 'IDR',
        minimumFractionDigits: 0,
    }).format(number);
};

// ================================================================
// 1. GET: Halaman Utama / (exports.index)
// ================================================================

exports.index = async (req, res) => { // DULU BERNAMA getFormData
    try {
        const produkList = await prisma.produk.findMany({
            include: {
                stok: true,
            },
        });

        const pembelianList = await prisma.pembelian.findMany({
            orderBy: { tgl_beli: 'desc' },
            include: {
                DetailPembelian: {
                    include: {
                        produk: true,
                    },
                },
            },
        });

        // Hitung Total Pembayaran dan Format Tanggal
        const formattedPembelianList = pembelianList.map(pembelian => {
            const totalBayar = pembelian.DetailPembelian.reduce((sum, item) => sum + (item.jumlah *
item.harga_satuan), 0);
            return {
                ...pembelian,
                totalBayar: formatRupiah(totalBayar),
                tanggalFormatted: new Date(pembelian.tgl_beli).toLocaleString('id-ID'),
            };
        });

        res.render('pembelian', {
            produkList,
            pembelianList: formattedPembelianList,
            message: req.session.message,
            formatRupiah: formatRupiah,
        });
        req.session.message = null;

    } catch (error) {
        console.error("Error saat mengambil data form:", error);
        res.status(500).send("Terjadi kesalahan server saat memuat data.");
    }
};
```

```
// ================================================================
// 2. POST: Membuat Pembelian Baru (Status Awal: 'Baru')
// ================================================================

exports.createPembelian = async (req, res) => {
    const { id_produk, jumlah } = req.body;
    const jumlahBeli = parseInt(jumlah);
    const idProduk = parseInt(id_produk);

    if (jumlahBeli <= 0 || !idProduk) {
        req.session.message = { type: 'error', text: 'Input jumlah atau produk tidak valid.' };
        return res.redirect('/');
    }

    try {
        await prisma.$transaction(async (tx) => {
            // 1. Cek Ketersediaan Stok dan Harga
            const produk = await tx.produk.findUnique({
                where: { id: idProduk },
                include: { stok: true },
            });

            if (!produk || !produk.stok || produk.stok.jumlah < jumlahBeli) {
                throw new Error(`Stok produk ${produk.nama} (${produk.stok.jumlah}) tidak mencukupi
untuk ${jumlahBeli} item.`);
            }

            const totalHarga = produk.harga * jumlahBeli;

            // 2. Kurangi Stok
            await tx.stok.update({
                where: { id: produk.stok.id },
                data: {
                    jumlah: {
                        decrement: jumlahBeli,
                    },
                },
            });

            // 3. Buat Entri Pembelian
            const pembelianBaru = await tx.pembelian.create({
                data: {
                    tgl_beli: new Date(),
                    status: 'Baru',
                    total: totalHarga,
                },
            });

            // 4. Buat Entri Detail Pembelian
            await tx.detailPembelian.create({
                data: {
                    id_pembelian: pembelianBaru.id,
                    id_produk: idProduk,
                    jumlah: jumlahBeli,
                    harga_satuan: produk.harga,
```

```
                },
            });

            req.session.message = { type: 'success', text: `Pembelian ID ${pembelianBaru.id} berhasil
dicatat dengan status 'Baru'. Stok berhasil dikurangi.` };
        });

    } catch (error) {
        console.error("Kesalahan Transaksi Pembelian:", error.message);
        req.session.message = { type: 'error', text: `Gagal mencatat pembelian: ${error.message}` };
    }

    res.redirect('/');
};

// ==================================================================
// 3. POST: Pembatalan Pembelian
// ==================================================================

exports.cancelPembelian = async (req, res) => {
    const idPembelian = parseInt(req.body.id_pembelian_cancel);

    if (!idPembelian) {
        req.session.message = { type: 'error', text: 'ID Pembelian untuk pembatalan tidak valid.' };
        return res.redirect('/');
    }

    try {
        await prisma.$transaction(async (tx) => {
            const pembelian = await tx.pembelian.findUnique({
                where: { id: idPembelian },
                include: { DetailPembelian: true },
            });

            if (!pembelian) {
                throw new Error(`Pembelian ID ${idPembelian} tidak ditemukan.`);
            }
            if (pembelian.status === 'Dibatalkan') {
                throw new Error(`Pembelian ID ${idPembelian} sudah dibatalkan sebelumnya.`);
            }

            // Kembalikan Stok
            for (const item of pembelian.DetailPembelian) {
                const stok = await tx.stok.findUnique({
                    where: { id_produk: item.id_produk },
                });

                if (!stok) {
                    throw new Error(`Stok produk ID ${item.id_produk} hilang.`);
                }

                await tx.stok.update({
                    where: { id: stok.id },
                    data: {
                        jumlah: {
                            increment: item.jumlah,
```

```
                        },
                    },
                });
            }

            // Perbarui Status Pembelian ke 'Dibatalkan'
            await tx.pembelian.update({
                where: { id: idPembelian },
                data: { status: 'Dibatalkan' },
            });

            req.session.message = { type: 'success', text: `Pembelian ID ${idPembelian} berhasil
dibatalkan. Stok telah dikembalikan.` };
        });

    } catch (error) {
        console.error("Kesalahan Transaksi Pembatalan:", error.message);
        req.session.message = { type: 'error', text: `Gagal membatalkan pembelian: ${error.message}`
};
    }

    res.redirect('/');
};

// ====================================================================
// 4. POST: Ubah Status menjadi Selesai
// ====================================================================

exports.markAsSelesai = async (req, res) => {
    const idPembelian = parseInt(req.body.id_pembelian_selesai);

    if (!idPembelian) {
        req.session.message = { type: 'error', text: 'ID Pembelian untuk penyelesaian tidak valid.' };
        return res.redirect('/');
    }

    try {
        const result = await prisma.pembelian.updateMany({
            where: { id: idPembelian, status: 'Baru' },
            data: { status: 'Selesai' },
        });

        if (result.count === 0) {

            const pembelianCheck = await prisma.pembelian.findUnique({
                where: { id: idPembelian },
            });

            if (!pembelianCheck) {
                req.session.message = { type: 'error', text: `Pembelian ID ${idPembelian} tidak
ditemukan.` };
            } else {
                req.session.message = { type: 'error', text: `Gagal menyelesaikan pembelian ID
${idPembelian}. Status saat ini adalah '${pembelianCheck.status}', bukan 'Baru'.` };
            }
        } else {
```

```
             req.session.message = { type: 'success', text: `Pembelian ID ${idPembelian} berhasil
diubah status menjadi 'Selesai'.` };
        }

    } catch (error) {
        console.error("Kesalahan saat mengubah status:", error.message);
        req.session.message = { type: 'error', text: `Gagal mengubah status pembelian:
${error.message}` };
    }

    res.redirect('/');
};

// ====================================================================
// 5. POST: Membersihkan Transaksi Dibatalkan (Fungsi Baru)
// ====================================================================
exports.clearCancelledPurchases = async (req, res) => {
    try {
        await prisma.$transaction(async (tx) => {
            // 1. Hapus DetailPembelian yang terkait dengan Pembelian 'Dibatalkan'
            await tx.detailPembelian.deleteMany({
                where: {
                    pembelian: {
                        status: 'Dibatalkan'
                    }
                }
            });

            // 2. Hapus entitas Pembelian yang memiliki status 'Dibatalkan'
            const result = await tx.pembelian.deleteMany({
                where: {
                    status: 'Dibatalkan'
                }
            });

            req.session.message = {
                type: 'success',
                text: `${result.count} transaksi Dibatalkan berhasil dihapus permanen dari riwayat.`
            };
        });

    } catch (error) {
        console.error("Error menghapus riwayat pembatalan:", error);
        req.session.message = {
            type: 'error',
            text: 'Gagal menghapus riwayat pembatalan. Coba periksa koneksi database atau skema
Prisma.'
        };
    } finally {
        res.redirect('/');
    }
};
```

6. ⚠ Persiapan Middleware Penting

Kode *controller* di atas menggunakan **session** (`req.session.message`) untuk menampilkan pesan sukses atau error setelah *redirect*. Anda harus menginstal dan mengkonfigurasi `express-session` di `server.js` Anda:

5.1. Instal `express-session`

```
npm install express-session
```

5.2. Tambahkan ke `server.js`

Tambahkan *middleware* sesi ke `server.js` Anda:

```
// server.js

// --- 1. Import Dependencies ---
const express = require('express');
const session = require('express-session'); // Digunakan untuk menyimpan session dan pesan flash
(message)
const dotenv = require('dotenv');
const path = require('path');

// --- Import Controllers & Routes ---
const indexRouter = require('./routes/index'); // Route untuk Tugas 1 (Admin Page)
const chatController = require('./controllers/chatController'); // Controller untuk Tugas 2 (Chatbot)

// --- 2. Konfigurasi ---
dotenv.config(); // Memuat variabel dari file .env
const app = express();
const PORT = process.env.PORT || 3000;

// --- 3. Setup View Engine (EJS) ---
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// --- 4. Middleware ---

// A. Session Middleware (PENTING untuk menyimpan riwayat chat dan pesan notifikasi)
app.use(session({
    secret: 'kunci_rahasia_super_aman_12345', // Ganti dengan kunci rahasia yang lebih kompleks di
production
    resave: false,
    saveUninitialized: true,
    cookie: { secure: process.env.NODE_ENV === 'production' }
}));

// B. Body Parsing Middleware (PENTING! Mencegah error 'Cannot read properties of undefined')
// Digunakan untuk memproses data dari form POST (seperti input 'message' di chatbot)
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// C. Static Files Middleware
app.use(express.static(path.join(__dirname, 'public')));


// --- 5. Routing ---
```

```
// A. Rute untuk Tugas 1: Admin Page (Transaksi Pembelian)
// Menggunakan indexRouter (yang mengarah ke pembelianController)
app.use('/', indexRouter);

// B. Rute untuk Tugas 2: Chatbot Gemini
app.get('/chat', chatController.getChatPage);
app.post('/chat', chatController.sendMessage);


// --- 6. Menjalankan Server ---
app.listen(PORT, () => {
    console.log(`Admin Page Server berjalan di http://localhost:${PORT}`);
});
```

5.3 `views/pembelian.ejs` (Kode Lengkap)

Salin dan tempel kode berikut ke dalam file **`views/pembelian.ejs`** di folder `views/` Anda:

```
$ touch views/pembelian.ejs
```

```
views/pembelian.ejs

<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Page Sederhana - Transaksi Pembelian</title>

    <style>
        /* Global Reset & Typography */
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 40px 20px;
            background-color: #f4f7f6;
            color: #333;
        }

        .container {
            max-width: 1000px;
            margin: auto;
            background: #ffffff;
            padding: 30px;
            border-radius: 12px;
            box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
        }

        h1 {
            color: #007bff;
            border-bottom: 3px solid #007bff;
            padding-bottom: 10px;
            margin-bottom: 30px;
```

```
    display: flex;
    align-items: center;
    font-size: 24px;
}
h1::before { content: '📦'; margin-right: 15px; font-size: 30px; }

h2 {
    color: #555;
    font-size: 20px;
    margin-top: 30px;
    margin-bottom: 15px;
    padding-bottom: 5px;
    border-bottom: 1px solid #eee;
    display: flex;
    align-items: center;
}
h2.input-form::before { content: '🛒'; margin-right: 10px; }
h2.cancel-form::before { content: '❌'; margin-right: 10px; }
h2.complete-form::before { content: '✅'; margin-right: 10px; }
h2.history::before { content: '📝'; margin-right: 10px; }

/* --- CSS BARU UNTUK PEMBERSIHAN RIWAYAT --- */
h2.cleanup-form::before { content: '🧹'; margin-right: 10px; }
.cleanup-form .clear-button {
    background-color: #ffc107; /* Kuning/Orange */
    color: #333;
    font-weight: bold;
    margin-right: 0;
}
.cleanup-form .clear-button:hover {
    background-color: #e0a800;
}
/* --- AKHIR CSS PEMBERSIHAN --- */


/* Form Styling */
form {
    margin-bottom: 25px;
    padding: 20px;
    border: 1px solid #e0e0e0;
    border-radius: 8px;
    background: #fdfdfd;
}
form > div { margin-bottom: 15px; }

label {
    display: block;
    margin-bottom: 8px;
    font-weight: 600;
    color: #444;
}

input[type="number"], select {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
```

```
            border-radius: 6px;
            box-sizing: border-box;
            transition: border-color 0.3s;
        }
        input[type="number"]:focus, select:focus {
            border-color: #007bff;
            outline: none;
        }

        /* Button Styling */
        button {
            padding: 10px 20px;
            border: none;
            border-radius: 6px;
            cursor: pointer;
            font-weight: bold;
            transition: background-color 0.3s;
            color: white;
            margin-right: 10px;
        }

        /* Warna Tombol Aksi */
        .input-form button { background-color: #28a745; }
        .input-form button:hover { background-color: #218838; }

        /* Tombol Pembatalan */
        .cancel-form button { background-color: #dc3545; }
        .cancel-form button:hover { background-color: #c82333; }

        /* Tombol Penyelesaian */
        .complete-form button { background-color: #007bff; }
        .complete-form button:hover { background-color: #0069d9; }

        /* Alert Messages */
        .alert {
            padding: 12px 20px;
            margin-bottom: 25px;
            border-radius: 6px;
            font-weight: 600;
            box-shadow: 0 2px 4px rgba(0,0,0,0.05);
        }
        .alert-success {
            background-color: #d4edda;
            color: #155724;
            border: 1px solid #c3e6cb;
        }
        .alert-error {
            background-color: #f8d7da;
            color: #721c24;
            border: 1px solid #f5c6cb;
        }

        /* Table Styling (Riwayat Pembelian) */
        table {
            width: 100%;
            border-collapse: separate;
```

```
        border-spacing: 0;
        margin-top: 20px;
        border: 1px solid #ddd;
        border-radius: 8px;
        overflow: hidden;
    }
    th, td {
        padding: 12px;
        text-align: left;
    }
    th {
        background-color: #007bff;
        color: white;
        font-weight: 600;
        border-right: 1px solid rgba(255, 255, 255, 0.1);
    }
    tr:nth-child(even) { background-color: #f9f9f9; }
    tr:hover { background-color: #f1f1f1; }

    /* Detail Produk List */
    td ul { list-style: none; padding: 0; margin: 0; }
    td ul li { padding: 2px 0; font-size: 14px; color: #666; }

    /* Status Visuals */
    .cancelled {
        color: #dc3545;
        font-style: italic;
        background-color: #f8d7da;
    }
    .cancelled td {
        text-decoration: line-through;
    }

    /* Status spesifik */
    td:nth-child(3) { font-weight: bold; }
    .status-Baru { color: #ffc107; }
    .status-Selesai { color: #28a745; }
    .status-Dibatalkan { color: #dc3545; }

    /* --- CSS CHATBOT FLOATING WIDGET --- */
    #chat-toggle-button {
        position: fixed;
        bottom: 20px;
        right: 20px;
        width: 60px;
        height: 60px;
        border-radius: 50%;
        background-color: #007bff;
        color: white;
        text-align: center;
        line-height: 60px;
        font-size: 24px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.4);
        cursor: pointer;
        z-index: 1000;
        transition: background-color 0.3s;
```

```
        }
        #chat-toggle-button:hover {
            background-color: #0056b3;
        }

        #chat-widget {
            position: fixed;
            bottom: 90px;
            right: 20px;
            width: 350px;
            height: 450px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 4px 12px rgba(0, 0, 0, 0.3);
            z-index: 999;
            display: none;
            overflow: hidden;
        }

        #chat-iframe {
            border: none;
            width: 100%;
            height: 100%;
        }
        /* --- AKHIR CSS CHATBOT --- */
    </style>
</head>
<body>
    <div class="container">
        <h1>Administrasi Transaksi Pembelian</h1>

        <% if (message) { %>
            <div class="alert alert-<%= message.type %>">
                <%= message.text %>
            </div>
        <% } %>

        <h2 class="input-form">Input Pembelian Baru</h2>
        <form action="/pembelian/input" method="POST" class="input-form">
            <div>
                <label for="id_produk">Pilih Produk:</label>
                <select id="id_produk" name="id_produk" required>
                    <option value="">-- Pilih Produk --</option>
                    <% produkList.forEach(produk => { %>
                        <option value="<%= produk.id %>">
                            <%= produk.nama %> (<%= produk.stok ? produk.stok.jumlah : 0 %> di Stok) -
<%= new Intl.NumberFormat('id-ID', { style: 'currency', currency: 'IDR', minimumFractionDigits: 0
}).format(produk.harga) %>
                        </option>
                    <% }); %>
                </select>
            </div>
            <div>
                <label for="jumlah">Jumlah Beli:</label>
                <input type="number" id="jumlah" name="jumlah" min="1" required>
            </div>
```

```
            <button type="submit">Catat Pembelian</button>
        </form>

        <h2 class="cancel-form">Batalkan Pembelian</h2>
        <form action="/pembelian/cancel" method="POST" class="cancel-form">
            <div>
                <label for="id_pembelian_cancel">ID Pembelian yang Dibatalkan:</label>
                <input type="number" id="id_pembelian_cancel" name="id_pembelian_cancel" required>
            </div>
            <button type="submit">Batalkan Transaksi</button>
        </form>

        <h2 class="complete-form">Selesaikan Transaksi (Ubah Status ke 'Selesai')</h2>
        <form action="/pembelian/selesai" method="POST" class="complete-form">
            <div>
                <label for="id_pembelian_selesai">ID Pembelian yang Diselesaikan:</label>
                <input type="number" id="id_pembelian_selesai" name="id_pembelian_selesai" required>
            </div>
            <button type="submit">Selesaikan Transaksi</button>
        </form>

        <h2 class="history">Riwayat Pembelian</h2>
        <table>
            <thead>
                <tr>
                    <th>ID Transaksi</th>
                    <th>Tanggal</th>
                    <th>Status</th>
                    <th>Detail Produk</th>
                    <th>Total Bayar</th>
                </tr>
            </thead>
            <tbody>
                <% if (pembelianList && pembelianList.length > 0) { %>
                    <% pembelianList.forEach(pembelian => { %>
                        <tr class="<%= pembelian.status === 'Dibatalkan' ? 'cancelled' : '' %>">
                            <td><%= pembelian.id %></td>
                            <td><%= pembelian.tanggalFormatted %></td>
                            <td class="status-<%= pembelian.status %>">
                                <%= pembelian.status %>
                                <% if (pembelian.status === 'Dibatalkan') { %>
                                    (Stok Kembali)
                                <% } %>
                            </td>
                            <td>
                                <ul>
                                    <% pembelian.DetailPembelian.forEach(item => { %>
                                        <li><%= item.jumlah %> x <%= item.produk.nama %> (<%=
formatRupiah(item.harga_satuan) %>)</li>
                                    <% }); %>
                                </ul>
                            </td>
                            <td><%= pembelian.totalBayar %></td>
                        </tr>
                    <% }); %>
                <% } else { %>
```

```
                    <tr>
                        <td colspan="5" style="text-align: center;">Belum ada riwayat transaksi
pembelian.</td>
                    </tr>
                <% } %>
            </tbody>
        </table>

        <div class="card-header">
            <h2>🧹 Pembersihan Riwayat</h2>
        </div>
        <form action="/clear-cancelled" method="POST" class="cleanup-form"
            onsubmit="return confirm('Apakah Anda yakin ingin MENGHAPUS PERMANEN semua transaksi yang
DIBATALKAN? Tindakan ini tidak dapat diurungkan.');">
            <div>
                <label style="font-style: italic; color: #777;">
                    Tombol ini akan menghapus permanen semua transaksi yang berstatus "Dibatalkan"
dari database.
                </label>
            </div>
            <button type="submit" class="clear-button">Bersihkan Riwayat Pembatalan</button>
        </form>
        </div>

    <div id="chat-toggle-button">🤖</div>

    <div id="chat-widget">
        <iframe id="chat-iframe" src="/chat" title="Chatbot Gemini Widget" width="100%" height="100%"
frameborder="0"></iframe>
    </div>

    <script>
        const chatButton = document.getElementById('chat-toggle-button');
        const chatWidget = document.getElementById('chat-widget');

        chatButton.addEventListener('click', () => {
            // Toggle (sembunyikan/tampilkan) jendela widget
            if (chatWidget.style.display === 'none' || chatWidget.style.display === '') {
                chatWidget.style.display = 'flex';
                chatButton.textContent = '❌'; // Ganti ikon menjadi X
            } else {
                chatWidget.style.display = 'none';
                chatButton.textContent = '🤖'; // Kembalikan ikon menjadi Robot
            }
        });

        // Opsional: Untuk menutup widget saat menekan tombol ESC
        document.addEventListener('keydown', (e) => {
            if (e.key === "Escape" && chatWidget.style.display === 'flex') {
                chatWidget.style.display = 'none';
                chatButton.textContent = '🤖';
            }
        });
    </script>

</body>
```

```
</html>
```

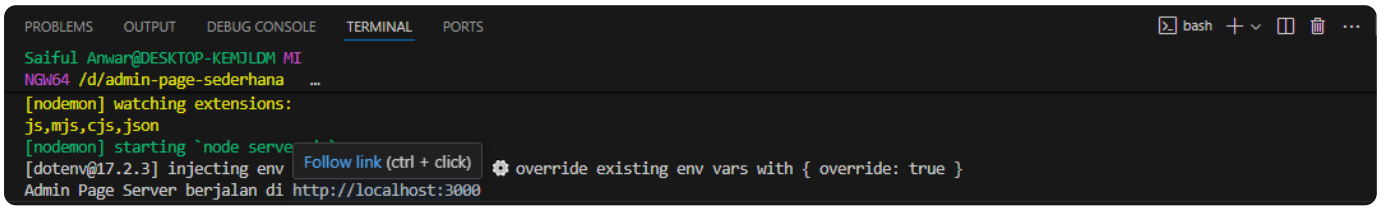
Uji Coba

Jalankan server Anda

Perintah Terminal (Jalankan Server):

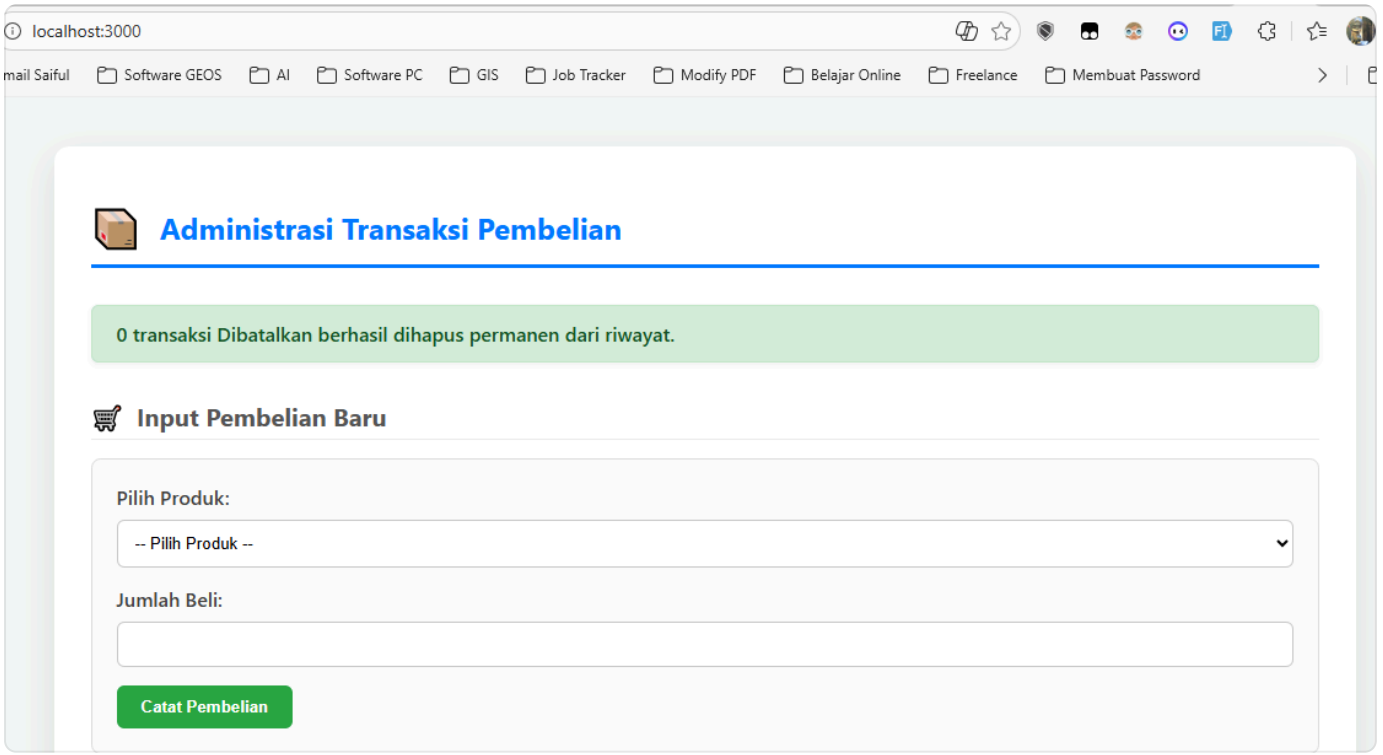
```
npm run dev
```

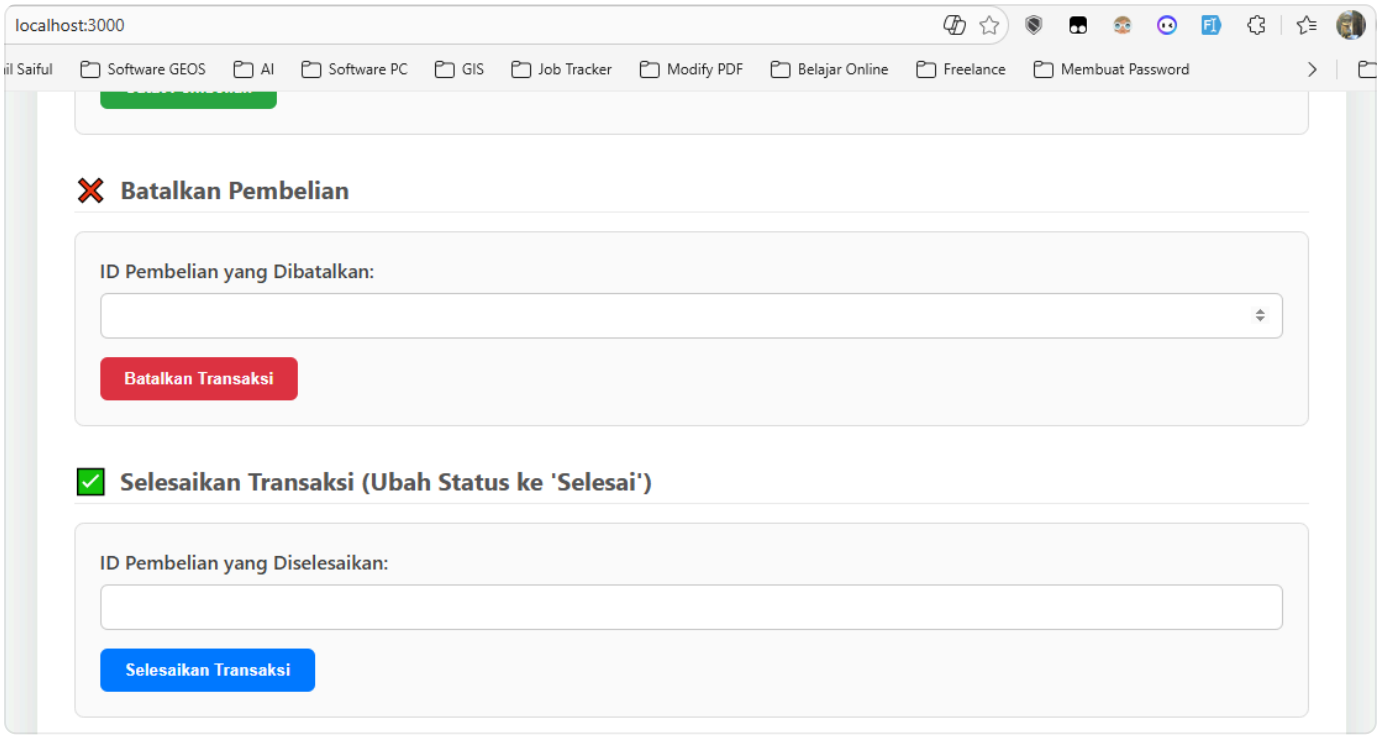
Akan muncul di terminal seperti ini

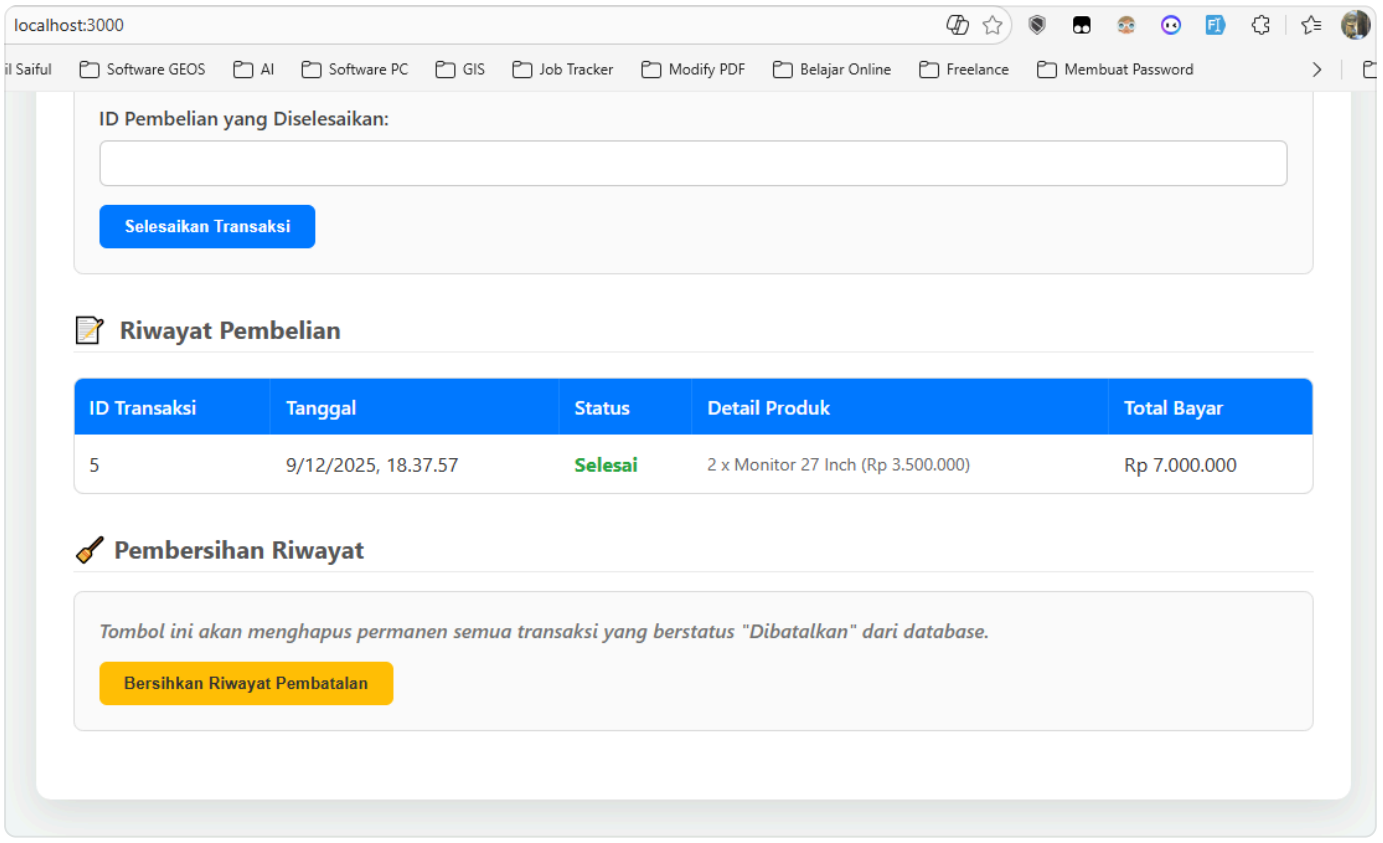


Klik yang http://localhost:3000 (ctrl +click)

lalu kamu akan diarahkan ke server localhost:3000 di web browser anda, dan tampilannya akan seperti ini







---

2. Tugas 2 (Chatbot Gemini)

🛠️ Step 1: Instalasi Library Chatbot

Kita perlu menginstal *package* Google Gemini API (`@google/genai`) yang akan menangani semua interaksi dengan AI.

Perintah Instalasi:

Bash

```
npm install @google/genai
```

🔑 Step 2: Konfigurasi API Key Gemini

Untuk mengizinkan aplikasi Anda berbicara dengan AI Gemini, Anda perlu kunci API.

🔑 Cara Membuat Kunci API Gemini

1. Kunjungi Google AI Studio

- Buka *browser* web Anda dan navigasi ke **Google AI Studio**. Ini adalah *platform* tempat Anda mengelola dan membuat kunci API untuk Gemini.

2. Navigasi ke API Key

- Setelah masuk (Anda mungkin perlu *login* dengan akun Google Anda), cari opsi atau tombol yang bertuliskan **"Get API key"**, **"Create API key"**, atau navigasikan ke bagian **"API access"** atau **"API keys"** di *dashboard*.
    - *Biasanya, ada tombol besar di sudut kiri atas atau di sidebar yang memandu Anda langsung.*

3. Buat Kunci Baru

- Klik tombol untuk **"Create API key"**.
- Setelah Anda mengklik, sistem akan segera membuat string kunci yang panjang dan unik (dimulai dengan `AIzaSy...`).

4. Salin dan Simpan Kunci Anda

- **Sangat Penting:** Setelah kunci API ditampilkan, segera **salin** kodenya secara lengkap.
- **Peringatan:** Kunci ini hanya akan ditampilkan sekali. Setelah Anda menutup jendela *pop-up*, Anda tidak akan bisa melihatnya lagi (Anda hanya bisa menghapusnya dan membuat yang baru).

5. Tambahkan Kunci ke File `.env` Anda

- Buka file `.env` di folder proyek Anda.
- Tempelkan kunci yang telah Anda salin sebagai nilai untuk variabel `GEMINI_API_KEY`:

```
# Environment variables declared in this file are NOT automatically loaded by Prisma.
# Please add `import "dotenv/config";` to your `prisma.config.ts` file, or use the Prisma CLI with Bun
# to load environment variables from .env files: https://pris.ly/prisma-config-env-vars.

# Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server,
MongoDB and CockroachDB.
# See the documentation for all the connection string options: https://pris.ly/d/connection-strings

# The following `prisma+postgres` URL is similar to the URL produced by running a local Prisma
Postgres
# server with the `prisma dev` CLI command, when not choosing any non-default ports or settings. The
API key, unlike the
# one found in a remote Prisma Postgres URL, does not contain any sensitive information.

DATABASE_URL="mysql://root:12345@localhost:3306/toko_db"

GEMINI_API_KEY="AIzaSyA81YW6xuFqsvCgLTeRFy8CxKEmaIHPXUk"
```

📁 Step 3: Membuat File Controller Chatbot

Buat logika *backend* yang spesifik untuk *chatbot*. Ini adalah tempat di mana pesan diproses dan dikirim ke AI.

Perintah Terminal (Membuat File Baru):

Buat file baru di dalam folder `controllers/`:

```
touch controllers/chatController.js
```

Isi File `controllers/chatController.js`:

Salin dan tempel kode *controller* berikut ke dalam file **`controllers/chatController.js`**:

```
// controllers/chatController.js

const { GoogleGenAI } = require('@google/genai');

// Inisialisasi Gemini AI. Kunci akan otomatis dibaca dari GEMINI_API_KEY di .env
const ai = new GoogleGenAI({});
// Riwayat chat di luar fungsi controller agar tetap tersimpan selama server berjalan
const chatHistory = [];

// 1. GET: Menampilkan halaman chat (hanya untuk memuat riwayat awal)
exports.getChatPage = (req, res) => {
    // Karena kita tidak lagi menggunakan pesan session untuk error saat AJAX,
    // kita hanya mengirim history.
    res.render('chatbot', {
        history: chatHistory,
        message: null // Tidak perlu pesan flash message dari session
    });
};

// 2. POST: Mengirim pesan ke AI (HARUS MENGEMBALIKAN JSON)
exports.sendMessage = async (req, res) => {
    const userMessage = req.body.message;

    if (!userMessage) {
        // Jika pesan kosong, kirim status 400 Bad Request
        return res.status(400).json({ success: false, message: 'Pesan tidak boleh kosong.' });
    }

    // Tambahkan pesan pengguna ke riwayat (di sisi server)
    chatHistory.push({ sender: 'user', text: userMessage });

    try {
        // Setup sesi chat dengan riwayat yang ada
        const chat = ai.chats.create({
            model: "gemini-2.5-flash",
            history: chatHistory.map(item => ({
                role: item.sender === 'user' ? 'user' : 'model',
                // Pastikan format parts sesuai: [ {text: "teks"} ]
                parts: [{ text: item.text }]
            }))
        });
```

```
        // Kirim pesan terbaru
        const response = await chat.sendMessage({ message: userMessage });
        const aiResponse = response.text;

        // Tambahkan respons AI ke riwayat (di sisi server)
        chatHistory.push({ sender: 'model', text: aiResponse });

        // Kirim respons JSON ke client
        return res.status(200).json({
            success: true,
            aiResponse: aiResponse
        });

    } catch (error) {
        console.error("Gemini API Error:", error.message);
        // Kirim respons error JSON
        return res.status(500).json({
            success: false,
            message: 'Gagal menghubungi AI. Pastikan API Key valid dan terisi.'
        });
    }
};
```

🖥️ Step 4: Membuat File View Chatbot

Buat antarmuka pengguna (UI) untuk *chatbot*.

Perintah Terminal (Membuat File Baru):

Buat file baru di dalam folder `views/`:

```
touch views/chatbot.ejs
```

Isi File `views/chatbot.ejs`:

Salin dan tempel kode tampilan berikut ke dalam file **`views/chatbot.ejs`**:

```
<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chatbot Gemini Sederhana</title>
    <style>
        /* PERBAIKAN: Hapus margin/padding dari body agar header menyentuh batas kotak iframe */
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: #f0f2f5;
            margin: 0; /* PASTIKAN INI NOL */
            padding: 0; /* PASTIKAN INI NOL */
            display: flex;
            justify-content: center;
            align-items: center;
```

```
            min-height: 100vh;
        }

        /* CHAT CONTAINER: Pastikan memenuhi lebar yang tersedia */
        .chat-container {
            width: 100%; /* Gunakan 100% agar memenuhi lebar iframe/widget */
            max-width: 600px;
            background: white;
            border-radius: 12px;
            box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);
            display: flex;
            flex-direction: column;
            height: 100vh; /* Gunakan 100vh agar memenuhi tinggi iframe/widget */
            overflow: hidden;
        }

        /* CHAT HEADER: Sudut yang benar */
        .chat-header {
            background-color: #007bff;
            color: white;
            padding: 15px;
            text-align: center;
            font-size: 1.2em;
            font-weight: bold;
            border-radius: 12px 12px 0 0; /* Sudut atas */
        }

        /* KODE LAINNYA */
        .chat-box { flex-grow: 1; padding: 20px; overflow-y: auto; background-color: #e9ebee; }
        .message { margin-bottom: 15px; display: flex; }
        .message.user { justify-content: flex-end; }
        .message.model { justify-content: flex-start; }
        .message-bubble { max-width: 80%; padding: 10px 15px; border-radius: 18px; line-height: 1.4;
word-wrap: break-word; }
        .message.user .message-bubble { background-color: #007bff; color: white; border-bottom-right-
radius: 4px; }
        .message.model .message-bubble { background-color: #ffffff; color: #333; border: 1px solid
#ddd; border-bottom-left-radius: 4px; }
        .chat-input-area { padding: 15px; border-top: 1px solid #ccc; background-color: white;
display: flex; }
        .chat-input-area #chatInput { flex-grow: 1; padding: 10px; border: 1px solid #ccc; border-
radius: 20px; margin-right: 10px; font-size: 1em; }
        .chat-input-area #sendButton { background-color: #28a745; color: white; border: none; border-
radius: 20px; padding: 10px 20px; cursor: pointer; font-weight: bold; transition: background-color
0.3s; }
        .chat-input-area #sendButton:hover { background-color: #218838; }
        .alert { padding: 10px; margin-bottom: 10px; border-radius: 6px; font-weight: bold; text-
align: center; }
        .alert-error { background-color: #f8d7da; color: #721c24; }

        /* Loading animation */
        .loading-dots {
            font-weight: bold;
            animation: blink 1.5s infinite;
        }
        @keyframes blink {
```

```
            0% { opacity: 0.2; }
            50% { opacity: 1; }
            100% { opacity: 0.2; }
        }
    </style>
</head>
<body>
    <div class="chat-container">
        <div class="chat-header">🤖 Chatbot Gemini</div>

        <div class="chat-box" id="chatBox">
            <% if (message && message.type === 'error') { %>
                <div class="alert alert-error"><%= message.text %></div>
            <% } %>

            <% history.forEach(msg => { %>
                <div class="message <%= msg.sender %>">
                    <div class="message-bubble">
                        <%= msg.text %>
                    </div>
                </div>
            <% }); %>
        </div>

        <div class="chat-input-area">
            <input type="text" id="chatInput" placeholder="Ketik pesan Anda..." required
autocomplete="off">
            <button id="sendButton">Kirim</button>
        </div>
    </div>

    <script>
        const chatInput = document.getElementById('chatInput');
        const sendButton = document.getElementById('sendButton');
        const chatBox = document.getElementById('chatBox');

        chatBox.scrollTop = chatBox.scrollHeight;

        function appendMessage(text, sender, isLoading = false) {
            const messageDiv = document.createElement('div');
            messageDiv.classList.add('message', sender);

            const bubble = document.createElement('div');
            bubble.classList.add('message-bubble');

            if (isLoading) {
                 bubble.innerHTML = '<span class="loading-dots">...</span>';
            } else {
                 bubble.textContent = text;
            }

            messageDiv.appendChild(bubble);
            chatBox.appendChild(messageDiv);
            chatBox.scrollTop = chatBox.scrollHeight;

            return messageDiv;
```

```
        }

        async function sendMessage() {
            const message = chatInput.value.trim();
            if (message === "") return;

            appendMessage(message, 'user');
            chatInput.value = '';

            const loadingBubbleContainer = appendMessage("", 'model', true);

            try {
                const response = await fetch('/chat', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/x-www-form-urlencoded',
                    },
                    body: new URLSearchParams({
                        message: message
                    })
                });

                const data = await response.json();

                chatBox.removeChild(loadingBubbleContainer);

                if (data.success) {
                    appendMessage(data.aiResponse, 'model');
                } else {
                    appendMessage(data.message || "Terjadi kesalahan saat menghubungi AI.", 'model');
                }

            } catch (error) {
                console.error("Fetch/Koneksi Error:", error);
                if (loadingBubbleContainer && chatBox.contains(loadingBubbleContainer)) {
                    chatBox.removeChild(loadingBubbleContainer);
                }
                appendMessage("Koneksi terputus atau API gagal dihubungi.", 'model');
            }
        }

        sendButton.addEventListener('click', sendMessage);

        chatInput.addEventListener('keypress', (e) => {
            if (e.key === 'Enter') {
                e.preventDefault();
                sendMessage();
            }
        });

    </script>
</body>
</html>
```

🛣️ Step 5: Mendaftarkan Route Baru

Buka file `server.js` Anda dan tambahkan impor serta *route* untuk *chatbot*.

Modifikasi File `server.js`:

Tambahkan baris berikut di `server.js` Anda:

```
// server.js

// --- 1. Import Dependencies ---
const express = require('express');
const session = require('express-session'); // Digunakan untuk menyimpan session dan pesan flash
(message)
const dotenv = require('dotenv');
const path = require('path');

// --- Import Controllers & Routes ---
const indexRouter = require('./routes/index'); // Route untuk Tugas 1 (Admin Page)
const chatController = require('./controllers/chatController'); // Controller untuk Tugas 2 (Chatbot)

// --- 2. Konfigurasi ---
dotenv.config(); // Memuat variabel dari file .env
const app = express();
const PORT = process.env.PORT || 3000;

// --- 3. Setup View Engine (EJS) ---
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// --- 4. Middleware ---

// A. Session Middleware (PENTING untuk menyimpan riwayat chat dan pesan notifikasi)
app.use(session({
    secret: 'kunci_rahasia_super_aman_12345', // Ganti dengan kunci rahasia yang lebih kompleks di
production
    resave: false,
    saveUninitialized: true,
    cookie: { secure: process.env.NODE_ENV === 'production' }
}));

// B. Body Parsing Middleware (PENTING! Mencegah error 'Cannot read properties of undefined')
// Digunakan untuk memproses data dari form POST (seperti input 'message' di chatbot)
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// C. Static Files Middleware
app.use(express.static(path.join(__dirname, 'public')));


// --- 5. Routing ---

// A. Rute untuk Tugas 1: Admin Page (Transaksi Pembelian)
// Menggunakan indexRouter (yang mengarah ke pembelianController)
app.use('/', indexRouter);
```

```
// B. Rute untuk Tugas 2: Chatbot Gemini
app.get('/chat', chatController.getChatPage);
app.post('/chat', chatController.sendMessage);


// --- 6. Menjalankan Server ---
app.listen(PORT, () => {
    console.log(`Admin Page Server berjalan di http://localhost:${PORT}`);
});
```

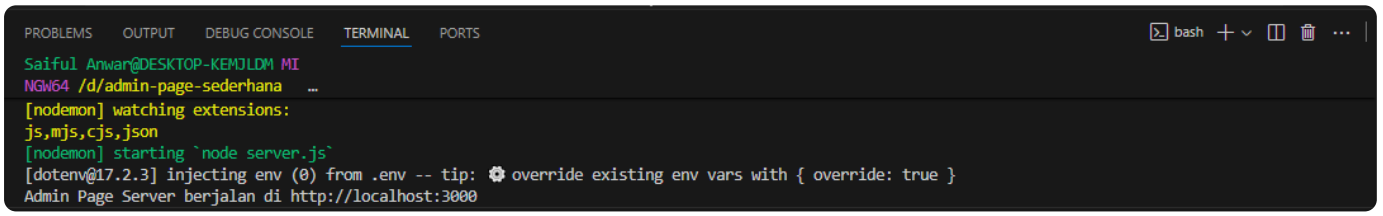
▶ Step 6: Uji Coba

Jalankan server Anda dan uji kedua fitur.

Perintah Terminal (Jalankan Server):

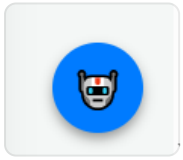
```
npm run dev
```

Lalu di terminal akan muncul seperti ini

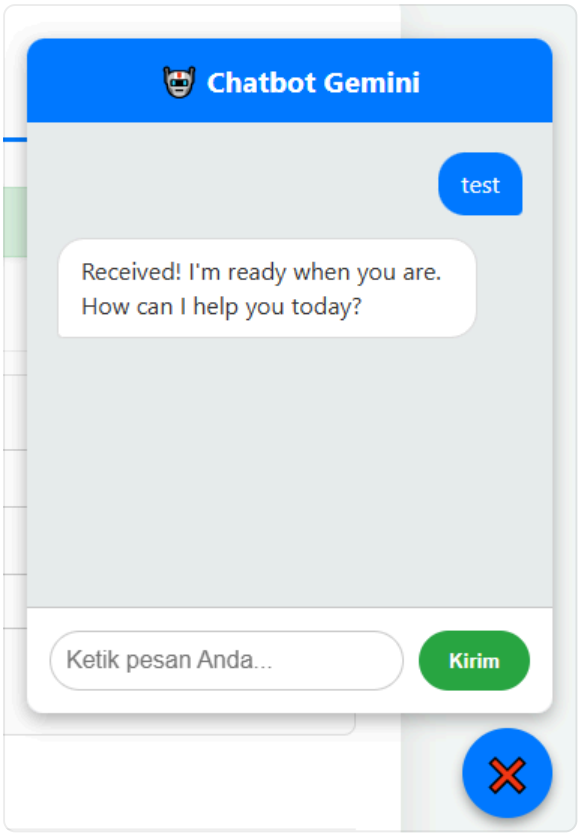


Selanjutkan kamu arahkan pointernya ke http://localhost:3000 (ctrl +click) dan akan otomatis membawa ke localhost:3000 pada web web browser anda

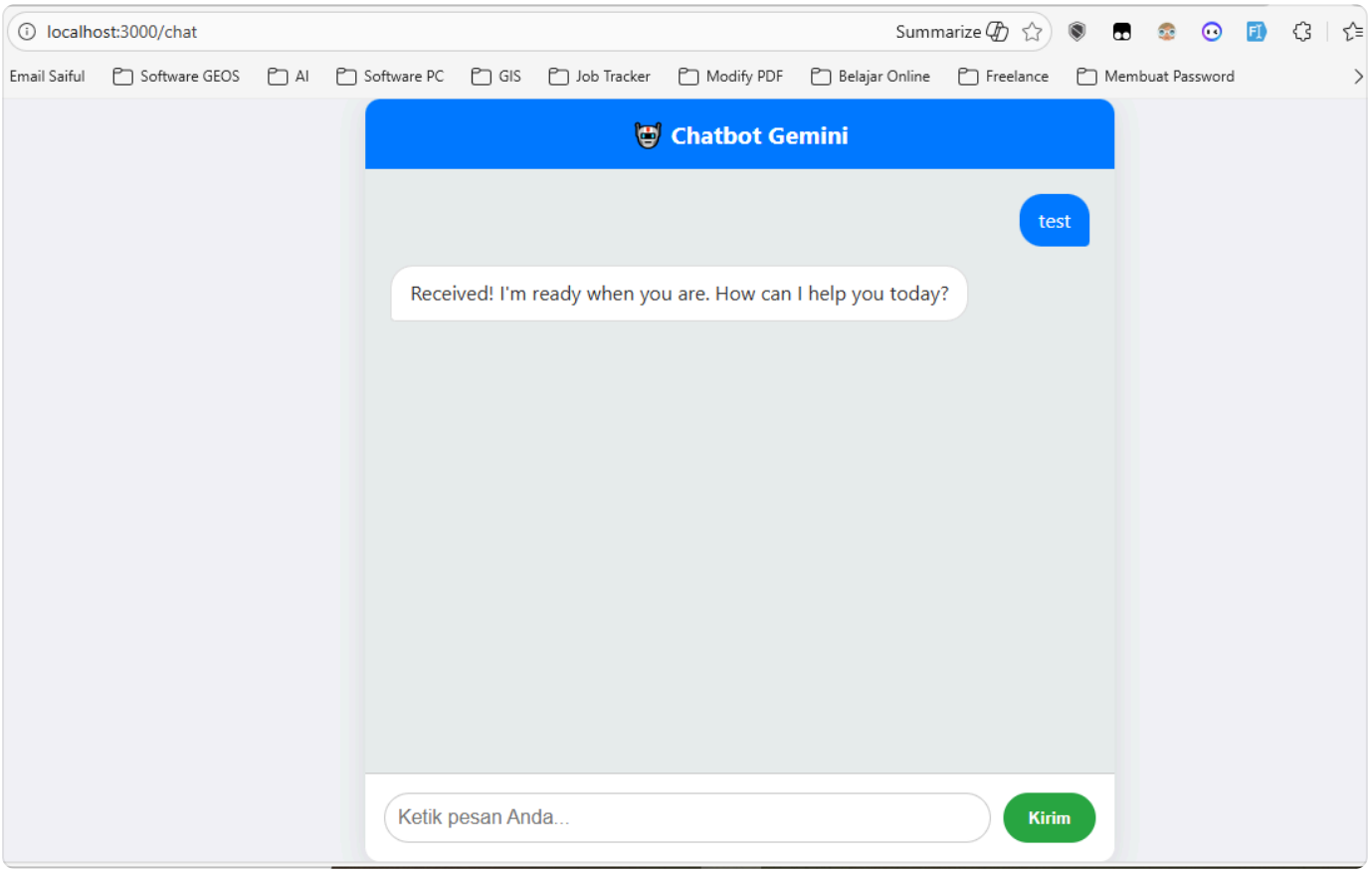
Akan ada ikon chatbot widget dibawah (pojook kanan bawah)



ketika di klik, akan muncul seperti ini

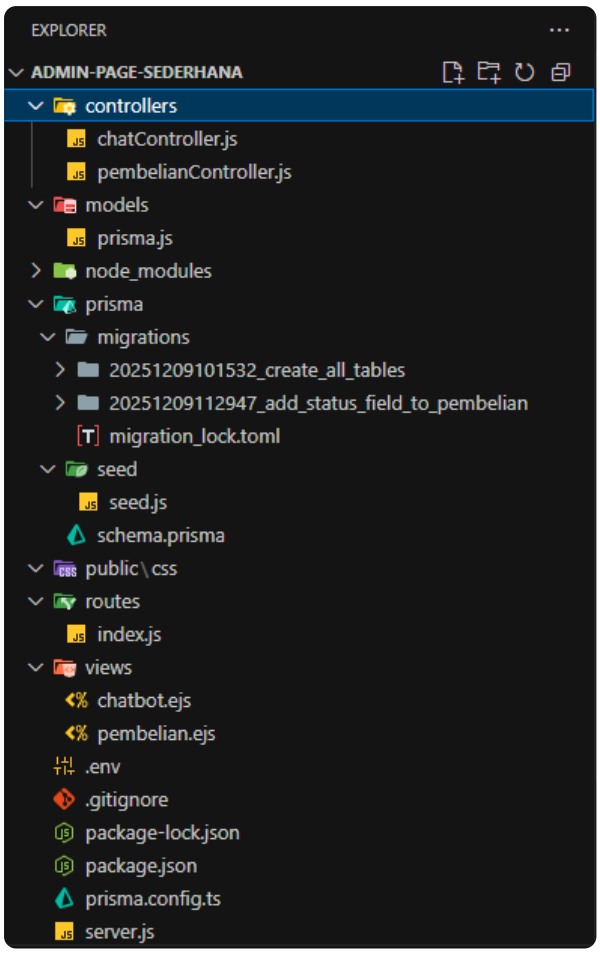


bisa juga di akses di http://localhost:3000/chat

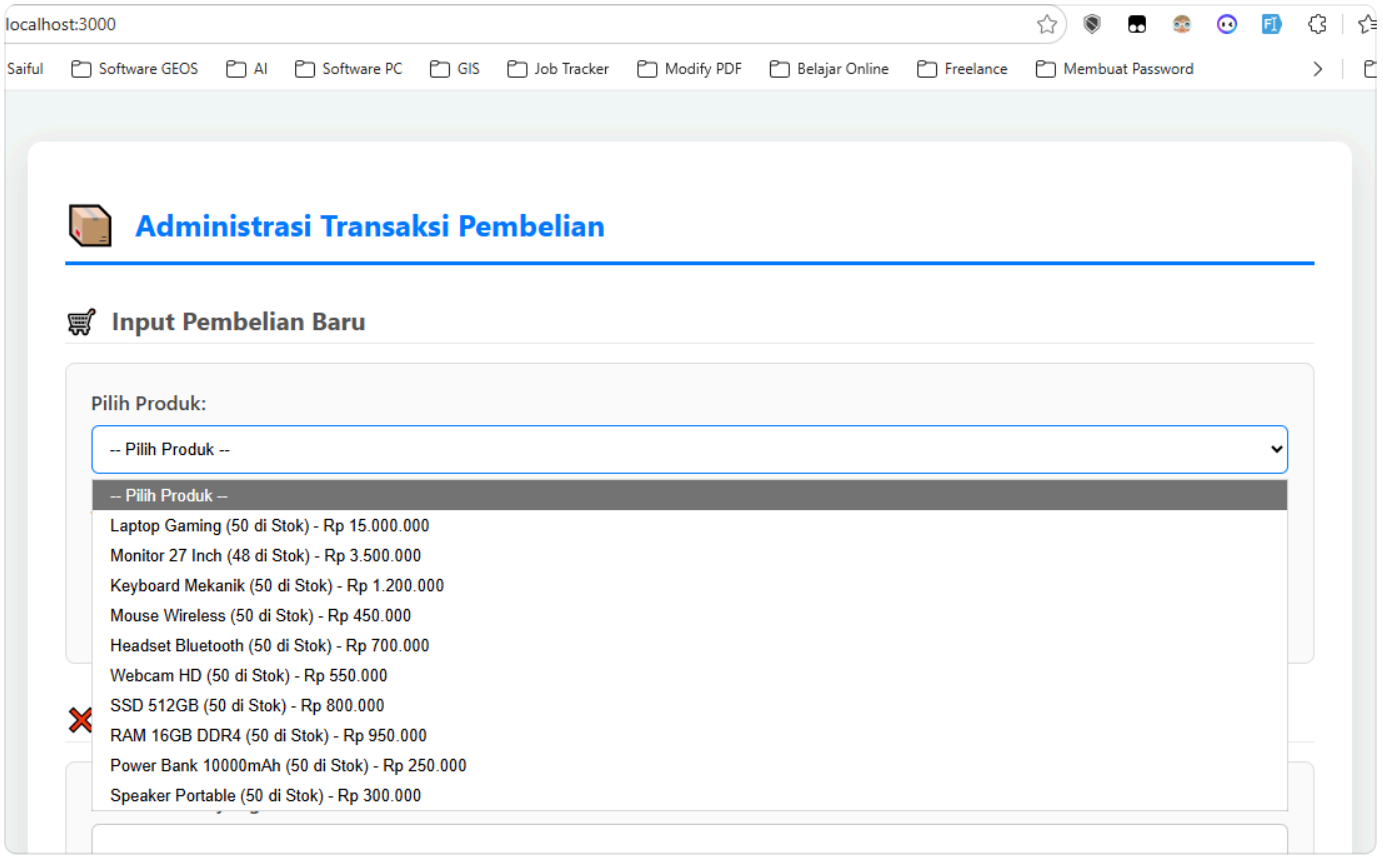


LAMPIRAN

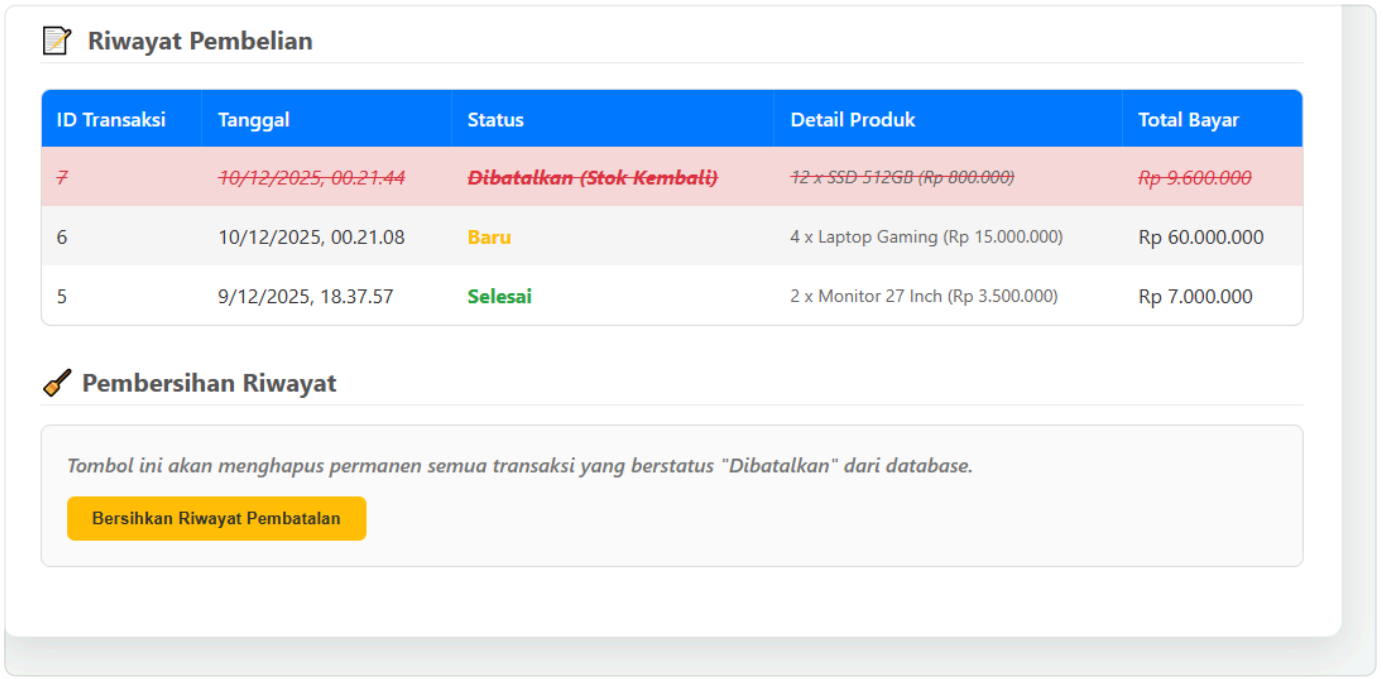
Struktur Project di VS Code



Tampilan 10 Produk beserta jumlah stok yang tersedia

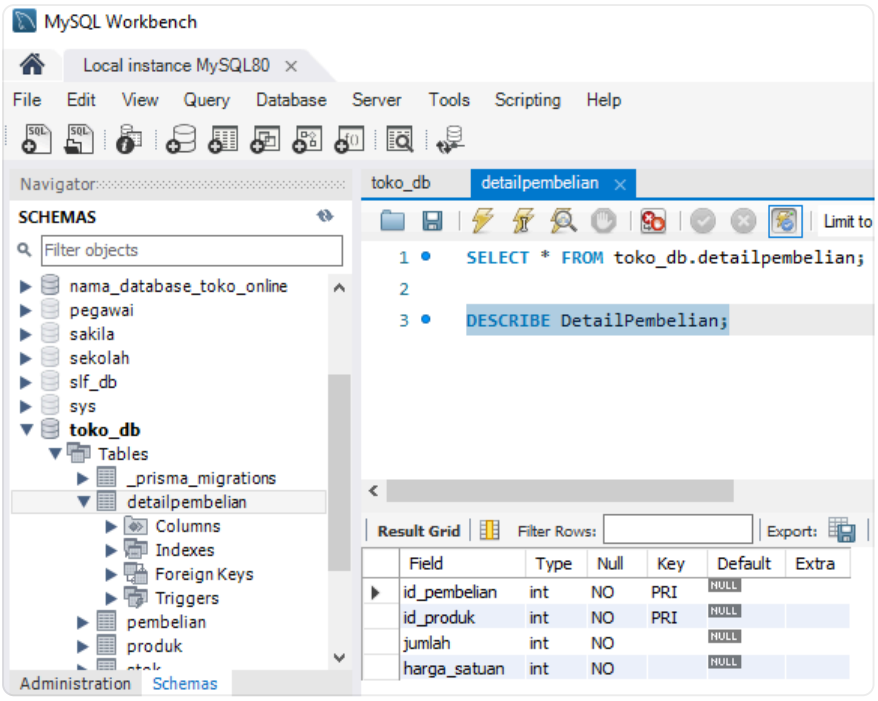


Tampilan Riwayat Pembelian

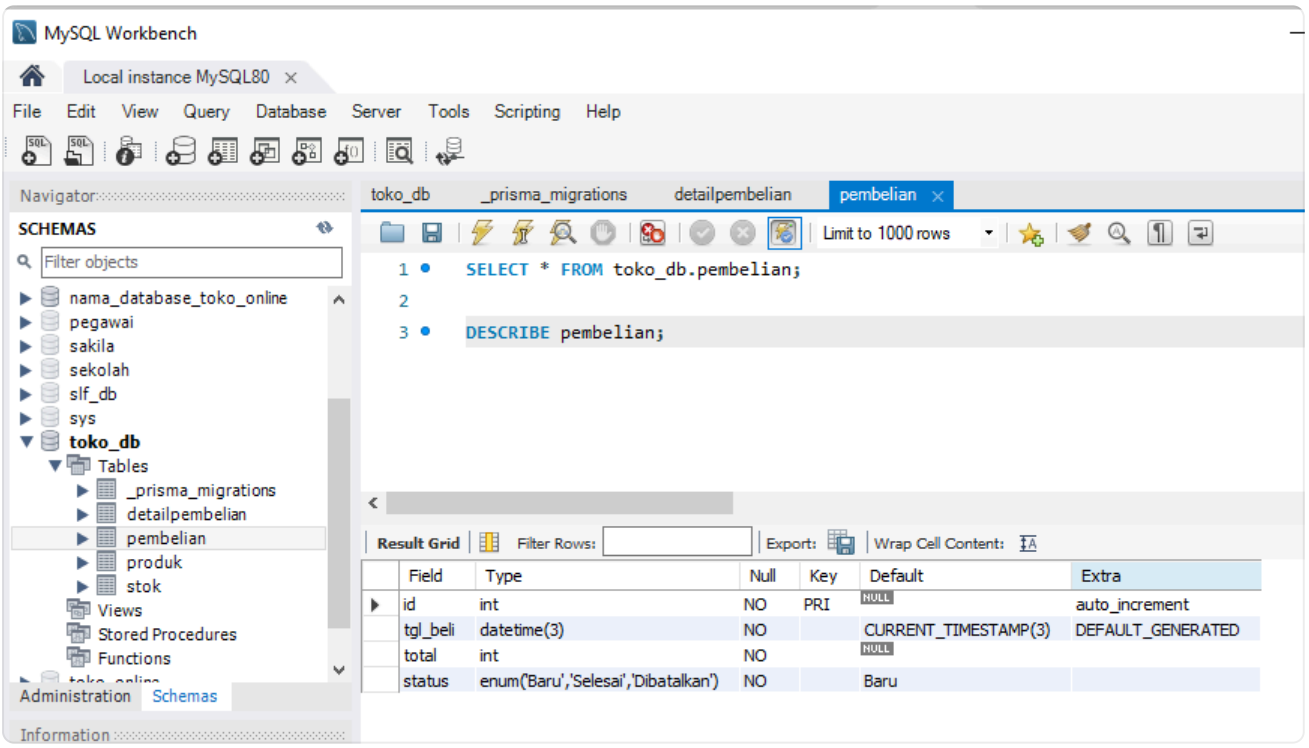


Struktur Database di MySQL

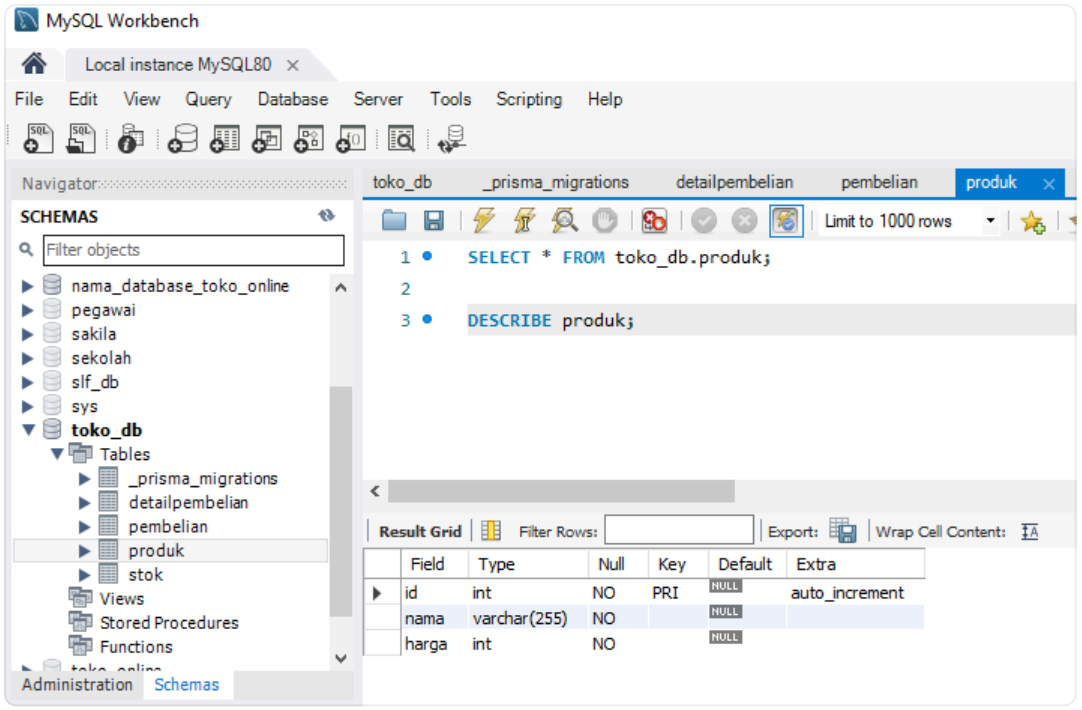
Tabel Data detailpembelian



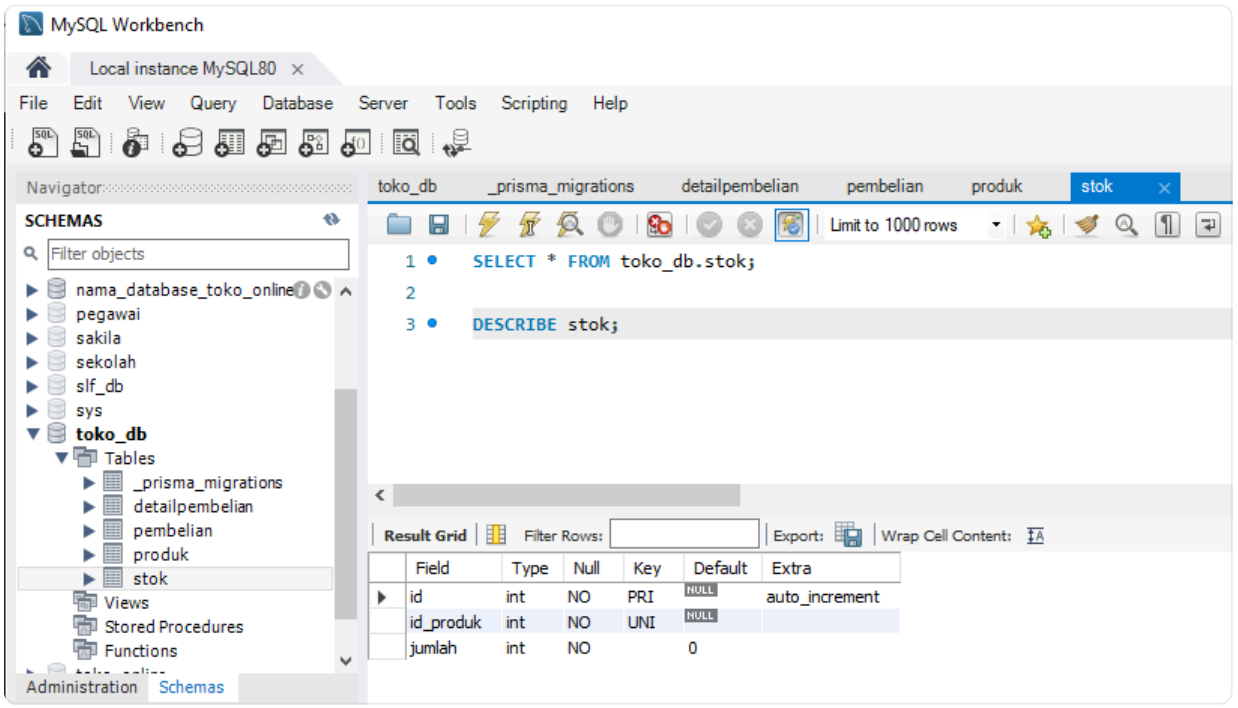
Tabel Data pembelian



Tabel data produk



Tabel data stok



Terimakasih :D