Sebelum aku memberikan struktur schema.prisma ku, apakah bisa dijelaskan lagi mengapa aku harus ada tabel atau model untuk temp\_order dan temp\_order\_detail ini? Karena nanti akan ditanyakan juga oleh dosen pembimbing.

**ChatGPT said:**

Pertanyaanmu sangat tepat dan logis — dan kamu **wajib tahu alasan fungsional + teknis** dari dibuatnya temp\_order dan temp\_order\_detail, apalagi ini akan menjadi pertanyaan dari dosen pembimbing.

**📌 Konteks Masalah:**

Ketika kamu menggunakan **Midtrans Snap**, sistem kamu:

* **Tidak bisa langsung menyimpan data ke tabel transaction**, karena:
  + Kamu belum tahu apakah user **benar-benar akan membayar**
  + Midtrans baru akan mengonfirmasi **melalui callback** (bisa sukses, pending, gagal)
* Jadi kamu perlu tempat **sementara** untuk menyimpan **data pembelian sebelum transaksi dikonfirmasi**

**✅ Jawaban Akademik: Mengapa Perlu temp\_order & temp\_order\_detail?**

**1. 🔄 Menjembatani alur asynchronous antara frontend dan callback Midtrans**

* Saat user klik “Beli Sekarang”, frontend langsung redirect ke Snap Midtrans.
* Tapi Snap tidak langsung menyimpan transaksi ke database.
* Maka dibutuhkan **tabel penyimpanan sementara** untuk mencatat:
  + Siapa yang beli (user\_id)
  + Tiket apa yang dibeli (ticket\_id)
  + Jumlah tiket (quantity)
  + Siapa yang akan datang (visitor\_name)
  + Dan kapan kunjungan dilakukan (visit\_date)

**2. ⛔ Tanpa temp\_order, kamu akan kehilangan konteks**

* Midtrans hanya kirim callback dengan order\_id, gross\_amount, dan transaction\_status.
* Tidak ada user\_id, tidak ada ticket\_id, tidak ada visitor\_name.
* Artinya, kamu tidak bisa membangun kembali data transaksi secara utuh.

**3. 🧾 temp\_order\_detail memungkinkan pencatatan data per individu**

* Jika 1 user membeli 3 tiket untuk 3 orang:
  + Sistem perlu tahu siapa saja yang akan datang.
  + Informasi ini penting untuk verifikasi QR atau sistem check-in di lokasi.

**4. 🔒 Memisahkan data yang valid vs tidak valid**

* Transaksi yang berhasil → data temp\_order dipindahkan ke transaction & transaction\_detail
* Transaksi gagal / expired → temp\_order bisa dihapus atau diabaikan

**5. 📚 Alasan sesuai teori Basis Data & Praktik Web:**

| **Prinsip** | **Penjelasan** |
| --- | --- |
| **Normalisasi** | temp\_order\_detail digunakan agar detail pengunjung tidak dicampur dalam satu kolom |
| **Reliabilitas Sistem** | Sistem tetap bisa menyimpan data transaksi yang valid tanpa tergantung frontend |
| **Auditability** | Bisa dilacak siapa yang memesan, dan siapa saja yang akan datang |

**🏛 Contoh Penjelasan ke Dosen:**

Untuk menjaga integritas data transaksi, saya menggunakan tabel temp\_order dan temp\_order\_detail sebagai buffer.  
Ini karena sistem pembayaran Midtrans bersifat asynchronous — artinya data hanya bisa dipastikan sah setelah ada callback sukses dari Midtrans.  
Dengan menyimpan data pesanan sementara, sistem saya tetap bisa melacak siapa pemesan, tiket apa yang dibeli, dan siapa saja pengunjung yang akan datang. Setelah pembayaran dikonfirmasi, data dari temp\_order dipindahkan ke tabel utama transaction, dan temp\_order dapat dihapus otomatis.

**✅ Apakah Tabel Ini Akan Membebani Database?**

Tidak. Karena:

* temp\_order hanya berisi data sementara
* Setelah transaksi sukses, datanya bisa dihapus
* Jumlah datanya tidak akan besar karena hanya menyimpan transaksi sebelum “paid”

Kalau kamu siap, aku bisa bantu:

* Menuliskan model temp\_order & temp\_order\_detail di schema.prisma
* Membuat logika migrasi dari temp\_order ke transaction saat callback

Apakah dengan menggunakan temp\_order juga bisa melakukan logika seperti ini?   
- 1 tiket untuk satu orang   
- 1 akun bisa membeli lebih dari satu tiket, tapi jika membeli lebih dari satu tiket maka harus input data seperti nama saja selain si pemilik akun   
- Ketika pembayaran ditunda atau masih pending maka akan ada notifikasi itu segera melakukan pembayaran agar tidak expried, kemudian saat pending maka tiket tidak bisa di-download

**ChatGPT said:**

🧠 Pertanyaan kamu sangat cerdas dan realistis — cocok untuk sistem tiket wisata skala serius.

Jawaban singkat:

✅ **YA** — dengan menggunakan **tabel temp\_order + temp\_order\_detail**, kamu **bisa mewujudkan semua logika tersebut.**

Mari kita bedah satu per satu:

**✅ 1. 1 tiket untuk 1 orang**

Artinya: setiap orang (pengunjung) harus punya 1 tiket

**Implementasi:**

* Di temp\_order\_detail, kamu simpan **1 row per orang**
* Contoh:
  + Tiket: TICKET001
  + Quantity: 3
  + Maka simpan 3 data temp\_order\_detail atau 1 data dengan quantity = 3 + form untuk input 3 nama

**✅ 2. 1 akun bisa beli banyak tiket, tapi harus input nama per orang**

Ini adalah kebutuhan **registrasi tamu**

**Implementasi:**

* Tambahkan kolom visitor\_name di temp\_order\_detail

|  |
| --- |
| prisma  visitor\_name String? |

* Di form frontend, jika user pilih **2 tiket**, tampilkan 2 field input nama:
  + Nama Pemesan (auto dari akun)
  + Nama Tamu 1
  + Nama Tamu 2
* Simpan ke temp\_order\_detail per nama (1 row per nama)

**✅ 3. Jika status pending, munculkan notifikasi untuk segera bayar**

Penting untuk menghindari tiket expired / tidak dibayar

**Implementasi:**

* Setelah redirect dari Snap, tampilkan halaman **"Menunggu Pembayaran"**
* Di situ:
  + Cek status transaksi via polling (fetch ke API status setiap 10 detik)
  + Jika pending, munculkan alert seperti:

"Segera selesaikan pembayaran Anda sebelum 15 menit berakhir."

* Kamu bisa hitung waktu expired dari data Midtrans (biasanya Snap token aktif 15 menit)

**✅ 4. Tiket hanya bisa didownload jika status = paid**

Ini sudah kamu implementasikan di halaman /user/payment:

|  |
| --- |
| tsx  {trx.transaction\_status === 'paid' && (  <a href={`/api/ticket/pdf?id=${trx.transaction\_code}`}>Download Tiket</a>  )} |

📌 **Jadi ini sudah OK**

**🏗 STRUKTUR DISARANKAN:**

**Tabel temp\_order**

| **Field** | **Tipe** |
| --- | --- |
| order\_id | string (PK) |
| user\_id | string |
| visit\_date | date |
| created\_at | timestamp |

**Tabel temp\_order\_detail**

| **Field** | **Tipe** |
| --- | --- |
| temp\_order\_id | FK |
| ticket\_id | FK |
| visitor\_name | string |
| quantity | int |
| price | decimal |
| subtotal | decimal |

⚠️ Kamu bisa pilih 1 row per orang (dengan 1 quantity), atau 1 row per jenis tiket.

**✅ BONUS IDE**

* Setelah pembayaran sukses → kirim email QR Code ke masing-masing visitor\_name (jika email disertakan)
* Buat "Tiket Grup" dengan banyak QR dalam 1 file PDF

Kalau kamu ingin, aku bisa bantu:

* Membuat struktur Prisma-nya
* Membuat api/temp\_order POST
* Integrasi ke form frontend yang menerima banyak nama

Sebentar, aku jadi penasaran. Jika memang payment\_method dan payment\_provider ini dihapus, kemudian ditambahkan dengan temp\_order dan temp\_order\_detail. Bagaimana alur keseluruhanya dari awal sampai akhir? Karena schema.prisma ku sudah pasti ada perombakan lagi.

Jawaban:

Pertanyaanmu sangat tepat — dan kamu **harus tahu gambaran besar alur sistem secara end-to-end** terutama setelah kita menyederhanakan payment\_\* dan menambahkan temp\_order. Ini juga sangat penting untuk dokumentasi dan sidang.

**🔁 ALUR KESELURUHAN SISTEM (FULL CYCLE)**

**🎯 Fokus: Transaksi tiket wisata menggunakan Midtrans + temp\_order**

**🧑 1. User Login dan Masuk ke Halaman Tiket**

* User melihat daftar tiket dari database ticket.

**🛒 2. User Klik “Beli Tiket”**

* User memilih tiket dan memasukkan jumlah & tanggal kunjungan.
* Jika membeli lebih dari 1, user juga bisa mengisi nama tamu lain (visitor\_name).

✅ **Yang dilakukan sistem:**

* Generate order\_id = ORDER-${timestamp}
* Simpan ke:
  + temp\_order → user\_id, visit\_date, order\_id
  + temp\_order\_detail → tiket, quantity, harga, visitor\_name
* Redirect ke Snap Midtrans menggunakan token dari /api/payment/tokenize

**💳 3. User Membayar Lewat Snap Midtrans**

* Midtrans akan:
  + Menyimpan status transaksi (success, pending, failed)
  + Mengirim **callback** ke pages/api/admin/transaction/callback.ts

**📨 4. Midtrans Kirim Callback**

Callback akan membawa:

|  |
| --- |
| json  {  "order\_id": "ORDER-...",  "transaction\_status": "settlement",  "gross\_amount": "5000",  "payment\_type": "qris"  } |

✅ **Yang dilakukan sistem (di callback.ts):**

1. Cari temp\_order berdasarkan order\_id
2. Ambil user\_id, visit\_date, dan semua detail tiket dari temp\_order\_detail
3. Buat entri baru ke:
   * transaction (1 baris)
   * transaction\_detail (per tiket)
4. Buat entri ke:
   * payment dengan payment\_type, amount, dan status = success
5. Generate QR dan simpan ke ticket.url\_qr
6. Opsional: Hapus temp\_order & temp\_order\_detail

**📃 5. User Melihat Halaman Riwayat Transaksi**

* Menampilkan data dari tabel transaction (yang status-nya paid)
* Dapat melihat:
  + Tanggal
  + Nama tiket
  + Status
  + Tombol Download Tiket (jika status = paid)

**🧾 6. Admin Melihat Halaman Pembayaran**

* Menampilkan data dari tabel payment
* Dapat melihat:
  + Reference (order\_id)
  + Jenis pembayaran (payment\_type)
  + Nominal
  + Status (success/pending)
  + Tanggal

**🧱 Struktur Tabel yang Digunakan:**

| **Fungsi** | **Tabel** |
| --- | --- |
| Data tiket wisata | ticket, spot\_tourism |
| Data user | users |
| Penyimpanan awal transaksi | temp\_order, temp\_order\_detail |
| Data final transaksi | transaction, transaction\_detail |
| Log pembayaran | payment |

**✍️ Penjelasan untuk Dokumentasi Tugas Akhir:**

Sistem ini dirancang dengan pendekatan transaksi dua fase: fase sementara (temp\_order) dan fase final (transaction). Karena integrasi pembayaran menggunakan Midtrans bersifat asinkron (callback-based), maka data transaksi awal disimpan ke dalam temp\_order dan temp\_order\_detail sebelum redirect ke Snap.  
Setelah pembayaran berhasil, sistem akan menerima callback dan menyelesaikan transaksi ke tabel transaction dan payment. Dengan pola ini, integritas data tetap terjaga dan sistem dapat memberikan log pembayaran yang akurat kepada admin maupun pengguna.

Kalau kamu mau, aku bisa bantu gambar diagram alir (flowchart) alur ini atau versi **urutan CRUD + tabel** agar bisa langsung kamu masukkan ke laporan tugas akhir. Mau dibantu juga?

Oke, aku kirimkan schema.prisma lagi, tapi schema.prisma ini masih yang lama belum ada perubahan. Jadi bagaimana alurnya jika memang penghapusan payment\_method dan payment\_provider, dan ditambahnya temp\_order dan temp\_order\_detail nanti?

**ChatGPT said:**

Jawaban:

|  |
| --- |
| generator client {  provider = "prisma-client-js"  output = "@/src/generated/prisma"  }  datasource db {  provider = "mysql"  url = env("DATABASE\_URL")  }  model category\_tourism {  category\_id String @id @db.Char(5)  category\_name String @db.VarChar(20)  category\_desc String? @db.VarChar(150)  category\_icon String? @db.VarChar(255)  spots spot\_tourism[]  }  model facility {  facility\_id String @id @db.Char(10)  facility\_name String @db.VarChar(100)  facility\_icon String? @db.VarChar(255)  facility\_tourisms facility\_tourism[]  }  model facility\_tourism {  facility\_tourism\_id String @id @db.Char(10)  spot\_id String? @db.Char(10)  facility\_id String? @db.Char(10)  spot spot\_tourism? @relation(fields: [spot\_id], references: [spot\_id], onDelete: Cascade)  facility facility? @relation(fields: [facility\_id], references: [facility\_id], onDelete: Cascade)  @@index([facility\_id])  @@index([spot\_id])  }  model gallery {  gallery\_id String @id @db.Char(10)  spot\_id String? @db.Char(10)  gallery\_img String? @db.VarChar(255)  gallery\_caption String? @db.VarChar(255)  spot spot\_tourism? @relation(fields: [spot\_id], references: [spot\_id], onDelete: Cascade)  @@index([spot\_id])  }  model operating\_hours {  operating\_id String @id @db.Char(10)  spot\_id String? @db.Char(10)  operating\_day OperatingDay  hours\_open DateTime @db.Time(0)  hours\_closed DateTime @db.Time(0)  spot spot\_tourism? @relation(fields: [spot\_id], references: [spot\_id], onDelete: Cascade)  @@index([spot\_id])  }  model payment {  payment\_id Int @id @default(autoincrement())  transaction\_id Int  reference\_number String @unique @db.VarChar(50)  payment\_type String @db.VarChar(50)  payment\_amount Decimal @db.Decimal(10, 2)  payment\_status PaymentStatus @default(pending)  payment\_time DateTime @default(now()) @db.Timestamp(0)  response\_data Json  transaction transaction @relation(fields: [transaction\_id], references: [transaction\_id], onDelete: Cascade)  @@index([transaction\_id])  }  model reviews {  reviews\_id String @id @db.Char(10)  spot\_id String? @db.Char(10)  user\_id String? @db.Char(10)  reviews\_rating Int? @db.TinyInt  reviews\_desc String? @db.Text  reviews\_created\_at DateTime? @default(now()) @db.Timestamp(0)  spot spot\_tourism? @relation(fields: [spot\_id], references: [spot\_id], onDelete: Cascade)  user users? @relation(fields: [user\_id], references: [user\_id], onDelete: Cascade)  @@index([spot\_id])  @@index([user\_id])  }  model spot\_tourism {  spot\_id String @id @db.Char(10)  category\_id String? @db.Char(5)  village\_id String? @db.Char(10)  spot\_name String @db.VarChar(60)  spot\_desc String? @db.Text  spot\_address String? @db.VarChar(200)  spot\_maps String? @db.VarChar(255)  spot\_thumbnail String? @db.VarChar(100)  spot\_rating Decimal? @default(0.00) @db.Decimal(3, 2)  spot\_contact String? @db.VarChar(15)  facility\_tourisms facility\_tourism[]  galleries gallery[]  operating\_hours operating\_hours[]  reviews reviews[]  category category\_tourism? @relation(fields: [category\_id], references: [category\_id])  village village? @relation(fields: [village\_id], references: [village\_id])  tickets ticket[]  @@index([category\_id])  @@index([village\_id])  }  model ticket {  ticket\_id String @id @db.Char(10)  spot\_id String? @db.Char(10)  ticket\_name String? @db.VarChar(100)  ticket\_price Decimal @db.Decimal(10, 2)  ticket\_desc String? @db.Text  ticket\_stock Int? @default(0)  code String @db.VarChar(10)  url\_qr String @db.Text  spot spot\_tourism? @relation(fields: [spot\_id], references: [spot\_id], onDelete: Cascade)  transaction\_details transaction\_detail[]  temp\_order\_details temp\_order\_detail[]  @@index([spot\_id])  }  model transaction {  transaction\_id Int @id @default(autoincrement())  user\_id String? @db.Char(10)  transaction\_code String @unique @db.VarChar(50)  total\_price Decimal @db.Decimal(10, 2)  transaction\_status TransactionStatus? @default(pending)  transaction\_date DateTime? @default(now()) @db.Timestamp(0)  visit\_date DateTime @db.Date  payments payment[]  user users? @relation(fields: [user\_id], references: [user\_id], onDelete: Cascade)  transaction\_details transaction\_detail[]  @@index([user\_id])  }  model transaction\_detail {  transaction\_detail\_id Int @id @default(autoincrement())  transaction\_id Int?  ticket\_id String? @db.Char(20)  detail\_quantity Int  detail\_price Decimal @db.Decimal(10, 2)  detail\_subtotal Decimal? @db.Decimal(10, 2)  is\_used TicketStatus @default(unverified)  transaction transaction? @relation(fields: [transaction\_id], references: [transaction\_id], onDelete: Cascade)  ticket ticket? @relation(fields: [ticket\_id], references: [ticket\_id], onDelete: Cascade)  @@index([ticket\_id])  @@index([transaction\_id])  }  model users {  user\_id String @id @db.Char(10)  name String @db.VarChar(100)  email String @unique @db.VarChar(100)  password String @db.VarChar(255)  phone\_number String? @db.VarChar(20)  address String? @db.Text  role Role  reviews reviews[]  transactions transaction[]  temp\_orders temp\_order[]  }  model village {  village\_id String @id @db.Char(10)  village\_name String @db.VarChar(100)  village\_city String @db.VarChar(100)  spots spot\_tourism[]  }  model temp\_order {  temp\_order\_id Int @id @default(autoincrement())  order\_id String @unique @db.VarChar(50)  user\_id String @db.Char(10)  visit\_date DateTime @db.Date  created\_at DateTime @default(now()) @db.Timestamp(0)  user users @relation(fields: [user\_id], references: [user\_id], onDelete: Cascade)  details temp\_order\_detail[]  @@index([user\_id])  }  model temp\_order\_detail {  temp\_order\_detail\_id Int @id @default(autoincrement())  temp\_order\_id Int  ticket\_id String @db.Char(20)  visitor\_name String? @db.VarChar(100)  quantity Int @default(1)  price Decimal @db.Decimal(10, 2)  subtotal Decimal @db.Decimal(10, 2)  temp\_order temp\_order @relation(fields: [temp\_order\_id], references: [temp\_order\_id], onDelete: Cascade)  ticket ticket @relation(fields: [ticket\_id], references: [ticket\_id], onDelete: Cascade)  @@index([temp\_order\_id])  @@index([ticket\_id])  }  enum OperatingDay {  Senin  Selasa  Rabu  Kamis  Jumat  Sabtu  Minggu  }  enum PaymentStatus {  pending  success  failed  }  enum TransactionStatus {  pending  paid  cancelled  }  enum Role {  admin  user  tourism\_owner  }  enum TicketStatus {  unverified  verified  } |

Dokumen schema.prisma milikmu telah berhasil direvisi sesuai kebutuhan integrasi Midtrans dan penggunaan temp\_order. Berikut perubahan utamanya:

**✅ Perubahan Utama:**

1. **Menghapus:**
   * payment\_method
   * payment\_provider
2. **Memodifikasi:**
   * payment → menambahkan payment\_type, menyederhanakan strukturnya
3. **Menambahkan:**
   * temp\_order
   * temp\_order\_detail

**🔁 Alur Sistem (Sesuai Struktur Baru)**

| **Tahap** | **Penjelasan** | **Tabel yang Digunakan** |
| --- | --- | --- |
| 1 | User memilih tiket dan jumlah | ticket |
| 2 | Simpan data order sementara + detail | temp\_order, temp\_order\_detail |
| 3 | Redirect ke Snap Midtrans (menggunakan order\_id) | - |
| 4 | Midtrans kirim callback ke server | transaction, transaction\_detail, payment |
| 5 | QR Code digenerate dan disimpan | ticket.url\_qr |
| 6 | Riwayat transaksi muncul di halaman user | transaction, payment |
| 7 | Admin memantau pembayaran | payment, transaction |

Kalau kamu siap, kita bisa lanjut ke:

* Membuat API penyimpanan ke temp\_order
* Menyesuaikan callback transaction/callback.ts agar ambil dari temp\_order
* Atau lanjut ke pembuatan halaman admin & user

Silakan pilih arah selanjutnya.