BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Program kerja magang di PT Jaya Santoso Teknologi dilaksanakan dibawah naungan divisi IT Development dengan peran sebagai backend developer. Tanggung jawab utama yang diemban selama masa magang adalah merancang dan mengembangkan layanan pembayaran untuk website Minyma yang saat ini tengah dikembangkan oleh perusahaan. Pelaksanaan tugas ini dilakukan di bawah bimbingan dan arahan Jesivinica Santoso selaku supervisor magang.

Seluruh anggota divisi dituntut untuk berkoordinasi secara aktif guna menghindari terjadinya miskomunikasi dalam proses pengerjaan proyek. Koordinasi dilakukan secara daring melalui platform WhatsApp, Discord, dan Google Meet. Selain menyusun laporan harian yang merupakan bagian dari prosedur kerja magang, rapat evaluasi divisi juga dilaksanakan setiap dua minggu sekali, baik secara WFH (*Work From Home*) melalui Google Meet atau secara WFO (*Work From Office*) di kantor, sesuai dengan kesepakatan.

Dalam proses pengembangan layanan pembayaran, digunakan beberapa teknologi pendukung, antara lain Java Spring Boot sebagai *framework* pengembangan dan PostgreSQL sebagai sistem basis data. Tim developer menerapkan sistem *branch* dan *pull request* melalui platform GitHub untuk memantau perkembangan pekerjaan masing-masing anggota tim. Setelah pengembangan pada *branch* tertentu selesai, *pull request* akan diajukan dan kemudian dievaluasi oleh product owner, sebelum digabungkan ke *branch* stg (*staging*) untuk pengujian. Setelah proses pengujian , kode akan digabungkan ke *branch* prod (*production*) dan kemudian di-*deploy*.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama pelaksanaan program kerja magang di PT Jaya Santoso Teknologi berfokus pada perancangan dan pengembangan layanan pembayaran untuk website Minyma. Pengembangan dilakukan dengan mengimplementasikan API (*Application Programming Interface*) untuk menangani proses CRUD (*Create, Read, Update, Delete*) terhadap entitas order, item, invoice, dan order snapshot. API tersebut digunakan untuk mendukung alur transaksi pada

sistem, seperti pembuatan order, pencatatan item yang dipesan, serta pembuatan dan pelacakan invoice.

Struktur kode disusun secara modular untuk memisahkan tanggung jawab ke dalam beberapa lapisan. *controller* digunakan untuk menangani permintaan dari sisi klien, *service* berfungsi sebagai tempat penerapan logika bisnis, *model* merepresentasikan entitas data yang terhubung dengan tabel dalam basis data, serta DTO (*Data Transfer Object*) digunakan sebagai perantara dalam proses antara sistem dan pengguna. Pendekatan ini diterapkan untuk menjaga keteraturan kode, mempermudah proses pemeliharaan, serta meminimalkan kendala dalam proses pengembangan dan *debugging*.

3.3 Uraian Pelaksanaan Magang

Rincian kegiatan magang yang dilakukan di PT Jaya Santoso Teknologi disajikan dalam Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	**Pekerjaan yang dilakukan**
1	Pengenalan dan pengarahan awal terkait proyek, instalasi *software* pendukung (Spring Boot, PostgreSQL, Postman, GitHub), serta *setup environment* pengembangan.
2	Analisis kebutuhan sistem pembayaran, diskusi alur dan skema sistem, serta mempelajari dasar penggunaan Java Spring Boot dan PostgreSQL.
3	Melanjutkan eksplorasi dan pendalaman konsep serta praktik pengembangan menggunakan Java Spring Boot dan PostgreSQL.
4	Perancangan alur sistem serta skema basis data untuk entitas Order dan Item.
5	Implementasi skema basis data yang telah dirancang ke dalam PostgreSQL.
6	Pengembangan *model, repository, service,* dan *controller* untuk entitas Order dan Item (struktur dasar, belum mengandung logika bisnis).
Lanjut pada halaman berikutnya	

Lanjutan Tabel 3.1

Minggu Ke -	Pekerjaan yang dilakukan
7	Revisi alur dan skema sistem; penambahan entitas Quotation serta implementasi *model, repository,* dan *controller*-nya.
8	Mempelajari konsep DTO (*Data Transfer Object*) dan mengimplementasikan DTO dasar untuk *request* dan *r*espons pada Order dan Item.
9	Pengembangan logika bisnis untuk proses CRUD pada entitas Order dan Item, termasuk pemetaan Item pada saat pembuatan Order.
10	Penyesuaian struktur respons menggunakan DTO untuk memastikan format data yang konsisten dan sesuai standar.
11	Revisi alur sistem: penambahan entitas OrderSnapshot, implementasi *model, controller,* dan *repository*-nya, serta integrasi ke dalam proses pembuatan Order.
12	Pembuatan method untuk kalkulasi harga dan integrasi entitas Invoice (yang dikembangkan oleh tim lain) ke dalam alur pembuatan Order.
13	Penambahan mekanisme *error handling* menggunakan *custom exception*, serta pembuatan struktur standar untuk respons sukses dan error.
14	Revisi skema sistem: penghapusan entitas Quotation untuk menghindari redundansi data dengan OrderSnapshot. Optimasi kode dengan memisahkan metode ke dalam *service* yang lebih modular serta revisi struktur DTO.
15	*Refactor* penamaan file dan penyesuaian *endpoint/method* sesuai kebutuhan front-end.
16	Pembuatan dokumentasi API untuk seluruh *endpoint*, serta revisi akhir pada alur pembuatan Order.

3.4 Perangkat Penunjang Pelaksanaan Magang

Bagian ini menyajikan perangkat-perangkat yang digunakan selama pelaksanaan magang di PT Jaya Santoso Teknologi. Informasi yang disampaikan mencakup perangkat lunak, perangkat keras, serta teknologi yang digunakan dalam

proses pengembangan sistem.

3.4.1 Perangkat Lunak yang Digunakan

Tabel 3.2 menyajikan daftar perangkat lunak yang digunakan selama pelaksanaan magang, yang mencakup nama perangkat, versi, kegunaan, dan tautan dokumentasi Perangkat lunak tersebut dipilih untuk mendukung proses pengembangan sistem backend, pengujian API, serta manajemen basis data dan kontrol versi.

Tabel 3.2. Daftar perangkat lunak dan keterangannya

Nama Perangkat	**Keterangan**
IntelliJ IDEA Community Edition	Versi 2024.3.2 – IDE (*Integrated Development Environment*) untuk pengembangan aplikasi Java. https://www.jetbrains.com/idea/documentation
PostgreSQL	Versi 17.3 (x64) – Sistem manajemen basis data relasional (RDBMS) yang digunakan melalui GUI pgAdmin4. https://www.postgresql.org/docs
pgAdmin 4	Versi 9.0 – Antarmuka grafis (GUI) untuk mengelola database PostgreSQL. https://www.pgadmin.org/docs/pgadmin4/latest
Postman	Versi 11.53.0 – Alat untuk melakukan pengujian dan manajemen API secara manual. https://learning.postman.com/docs
GitHub Desktop	Versi 3.5.0 (x64) – Antarmuka grafis untuk sistem kontrol versi Git yang memudahkan kolaborasi pengembangan kode. https://docs.github.com/desktop

3.4.2 Perangkat Keras yang Digunakan

Tabel 3.3 menyajikan perangkat keras yang digunakan selama pelaksanaan magang, meliputi jenis perangkat dan spesifikasi teknisnya. Perangkat keras ini mendukung kebutuhan pengembangan perangkat lunak, pengujian sistem, serta menjalankan aplikasi yang dibutuhkan selama kegiatan magang berlangsung.

Tabel 3.3. Daftar perangkat keras dan spesifikasinya

Nama Perangkat	**Spesifikasi**
Laptop pribadi	HP 255 G8
Processor	AMD Ryzen 5 5500U
RAM (*Random Access Memory*)	16 GB DDR4
Storage	512 GB SSD
Operating System	Windows 11 Home 64-bit
GPU (*Graphics Processing Unit*)	AMD Radeon RX Vega 7

3.4.3 Teknologi yang Digunakan

Tabel 3.4 menampilkan daftar teknologi yang digunakan dalam proses pengembangan sistem selama masa magang. Daftar tersebut mencakup informasi mengenai nama teknologi, versi, fungsi utama, serta tautan menuju dokumentasi resminya. Teknologi yang digunakan meliputi framework, library, dan sistem basis data yang berperan penting dalam membangun, memvalidasi, dan mengelola data pada sisi backend aplikasi.

Tabel 3.4. Daftar teknologi dan keterangannya

Nama Teknologi	**Keterangan**
Spring Boot	Versi 3.4.2 – Framework berbasis Java untuk membangun layanan backend REST API. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
PostgreSQL	Versi 17.3 – Sistem manajemen basis data relasional (RDBMS) yang digunakan untuk menyimpan dan mengelola data aplikasi. https://www.postgresql.org/docs
	Lanjut pada halaman berikutnya

Lanjutan Tabel 3.4

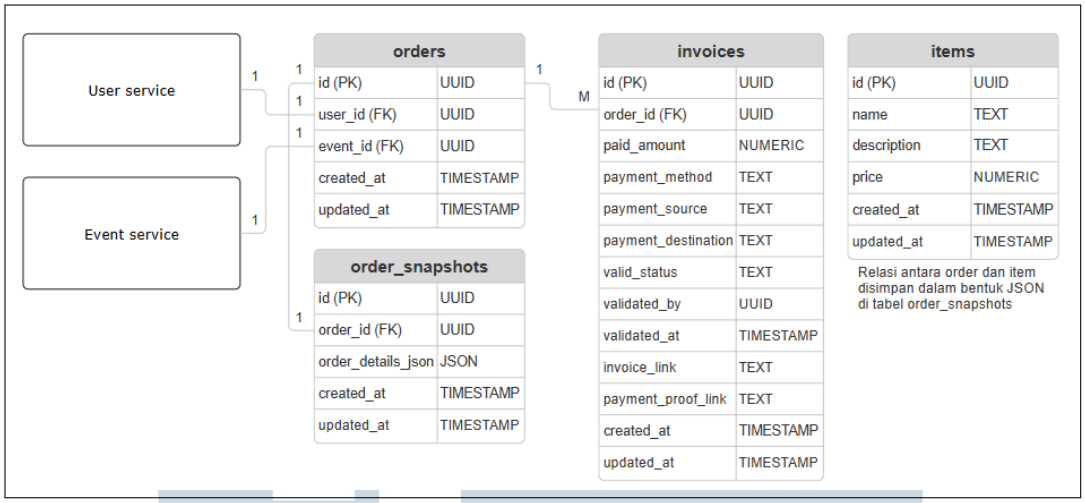
Nama Teknologi	Keterangan
Lombok	Versi mengikuti konfigurasi Spring Boot – Library Java untuk mengurangi kode boilerplate seperti getter, setter, dan constructor melalui anotasi. https://projectlombok.org/api/lombok/package-summary
Spring Boot Starter Validation	Versi mengikuti konfigurasi Spring Boot – Library untuk validasi input menggunakan anotasi seperti `@NotNull`, `@Min`, dan lainnya. https://blog.devops.dev/getting-started-with-validation-in-spring-boot-using-validator-1a7ab6d2d1f5

3.5 Perancangan Sistem

Bagian ini menjelaskan proses perancangan layanan pembayaran untuk website Minyma di PT Jaya Santoso Teknologi. Pembahasan mencakup struktur basis data dan implementasi sistem melalui engujian *endpoint* menggunakan Postman yang disajikan melalui tangkapan layar.

3.5.1 Struktur Tabel

Struktur tabel dalam pengembangan layanan pembayaran website Minyma terdiri atas sejumlah tabel yang saling berelasi untuk mendukung alur transaksi sistem. Setiap tabel dirancang dengan atribut dan *constraint* yang disesuaikan dengan kebutuhan untuk menjaga integritas dan konsistensi data. Hubungan antar tabel divisualisasikan melalui *physical* ERD (*Entity Relationship Diagram*) pada Gambar 3.1 , diikuti penjelasan detail setiap tabel



Gambar 3.1. *Physical* ERD layanan pembayaran website Minyma

A Tabel `orders`

Tabel 3.5 mendeskripsikan struktur tabel `orders`, yang digunakan untuk menyimpan data pesanan yang dibuat dalam website Minyma. Tabel ini berelasi dengan layanan `users` dan layanan `events` melalui *foreign key*, serta mencatat waktu pembuatan dan pembaruan setiap pesanan.

Tabel 3.5. Struktur tabel `orders`

Nama Kolom	Tipe Data	Constraint
`id`	`UUID`	`PRIMARY KEY`
`user_id (FK)`	`UUID`	`FOREIGN KEY, ON DELETE CASCADE`
`event_id (FK)`	`UUID`	`FOREIGN KEY, ON DELETE CASCADE`
`created_at`	`TIMESTAMP`	`NOT NULL`
`updated_at`	`TIMESTAMP`	`NOT NULL`

Penjelasan atribut tabel `orders`:

1. `id`
   Berfungsi sebagai *primary key* yang unik untuk setiap data pesanan. Kolom ini menggunakan tipe data `UUID` yang dihasilkan secara acak untuk menjamin keunikan.

2. `user_id`

   Diambil sebagai *foreign key* dari kolom `id` pada tabel `users`. Berfungsi untuk mereferensikan pengguna yang melakukan pesanan. Jika data pengguna terkait dihapus, maka data pesanan juga akan dihapus secara otomatis melalui aturan `ON DELETE CASCADE`.

3. `event_id`

   Diambil sebagai *foreign key* dari kolom `id` pada tabel `events`. Kolom ini digunakan untuk mereferensikan acara yang dipesan oleh pengguna. Sama seperti `user_id`, kolom ini juga menerapkan aturan `ON DELETE CASCADE`.

4. `created_at`

   Mencatat waktu pembuatan data pesanan. Bersifat `NOT NULL`, sehingga wajib diisi saat data dibuat.

5. `updated_at`

   Mencatat waktu terakhir kali data pesanan diperbarui. Bersifat `NOT NULL`, sehingga wajib diisi setiap kali terjadi perubahan pada data.

B Tabel `items`

Tabel 3.6 mendeskripsikan struktur tabel `items`, yang berfungsi untuk menyimpan informasi barang atau layanan yang dapat dimasukkan ke dalam suatu pesanan. Setiap item memiliki atribut seperti nama, deskripsi, dan harga. Data ini digunakan saat menyusun isi dari sebuah pesanan.

Tabel 3.6. Struktur tabel `items`

Nama Kolom	Tipe Data	Constraint
id (PK)	UUID	PRIMARY KEY
name	TEXT	NOT NULL
description	TEXT	NOT NULL
price	NUMERIC	NOT NULL
created_at	TIMESTAMP	NOT NULL
updated_at	TIMESTAMP	NOT NULL

Penjelasan atribut tabel `items`:

1. `id`

   Berfungsi sebagai *primary key* yang unik untuk setiap data item. Kolom ini menggunakan tipe data `UUID` yang dihasilkan secara acak untuk menjamin keunikan.

2. `name`

   Menyimpan nama item yang dapat dipesan. Kolom ini bersifat `NOT NULL`, sehingga wajib diisi untuk setiap entri data.

3. `description`

   Berfungsi untuk menyimpan informasi deskriptif mengenai item. Kolom ini juga bersifat `NOT NULL` dan menggunakan tipe data `TEXT` untuk memungkinkan penyimpanan teks panjang.

4. `price`

   Menunjukkan harga dari item terkait. Tipe data yang digunakan adalah `NUMERIC` untuk memastikan presisi nilai desimal. Kolom ini bersifat `NOT NULL`, sehingga tidak boleh dikosongkan.

5. `created_at`

   Mencatat waktu pembuatan data item. Kolom ini bersifat `NOT NULL` sehingga wajib diisi.

6. `updated_at`

   Mencatat waktu terakhir kali data item diperbarui. Kolom ini bersifat `NOT NULL` dan diharapkan diperbarui setiap kali terjadi perubahan pada data item.

C Tabel `invoices`

Tabel 3.7 mendeskripsikan struktur tabel `invoices`, yang merekam informasi transaksi pembayaran atas pesanan yang telah dilakukan. Tabel ini berelasi dengan tabel `orders` dan `users` melalui *foreign key*, serta menyimpan data pembayaran seperti jumlah, metode, sumber, tujuan, status validasi, dan bukti pembayaran. Selain itu, tabel ini juga mencatat waktu pembuatan dan pembaruan setiap transaksi.

Tabel 3.7. Struktur tabel `invoices`

Nama Kolom	Tipe Data	Constraint
id (PK)	UUID	PRIMARY KEY
order_id (FK)	UUID	FOREIGN KEY, ON DELETE CASCADE
paid_amount	NUMERIC	NUMERIC
payment_method	TEXT	-
payment_source	TEXT	-
payment_destination	TEXT	-
valid_status	TEXT	DEFAULT "Pending"
validated_by	UUID	-
validated_at	TIMESTAMP	-
invoice_link	TEXT	-
payment_proof_link	TEXT	-
created_at	TIMESTAMP	NOT NULL
updated_at	TIMESTAMP	NOT NULL

Penjelasan atribut tabel `invoices`:

1. `id`

   Berfungsi sebagai *primary key* yang unik untuk setiap data invoice. Kolom ini menggunakan tipe data `UUID` untuk menjamin keunikan dalam sistem.

2. `order_id`

   Merupakan *foreign key* yang mengacu pada kolom `order_id` pada tabel `orders`. Kolom ini berfungsi untuk menghubungkan invoice dengan pesanan tertentu. Jika data pesanan dihapus, maka data invoice yang berelasi juga akan terhapus secara otomatis melalui aturan `ON DELETE CASCADE`.

3. `paid_amount`

   Menyimpan jumlah nominal pembayaran yang dilakukan. Tipe data yang digunakan adalah `NUMERIC`, yang memungkinkan representasi angka desimal untuk keperluan transaksi keuangan.

4. `payment_method`
   Menyimpan informasi metode pembayaran yang digunakan, seperti transfer bank atau dompet digital. Menggunakan tipe data `TEXT`.

5. `payment_source`
   Menyimpan informasi sumber pembayaran (misalnya, nama rekening atau akun dompet digital). Bertipe `TEXT` dan bersifat opsional.

6. `payment_destination`
   Menyimpan informasi tujuan pembayaran, yaitu akun atau rekening tujuan. Kolom ini juga menggunakan tipe data `TEXT`.

7. `valid_status`
   Menyimpan status validasi pembayaran, Bertipe `TEXT` dengan nilai awal (`default`) berupa `"Pending"`. Status ini menunjukkan apakah pembayaran telah divalidasi oleh admin.

8. `validated_by`
   Menyimpan `UUID` dari admin yang melakukan validasi pembayaran. Kolom ini bersifat opsional dan dapat kosong jika belum divalidasi.

9. `validated_at`
   Mencatat waktu saat pembayaran divalidasi. Kolom ini menggunakan tipe data `TIMESTAMP` dan bersifat opsional.

10. `invoice_link`
    Menyimpan tautan (*link*) menuju dokumen invoice digital. Bertipe `TEXT` dan bersifat opsional.

11. `payment_proof_link`
    Menyimpan tautan menuju bukti pembayaran yang diunggah. Bertipe `TEXT` dan bersifat opsional.

12. `created_at`
    Mencatat waktu pembuatan data invoice. Kolom ini bersifat `NOT NULL` dan wajib diisi saat data dibuat.

13. `updated_at`
    Mencatat waktu terakhir kali data invoice diperbarui. Kolom ini juga bersifat `NOT NULL` dan harus diperbarui setiap kali terjadi perubahan pada data.

D Tabel `order_snapshots`

Tabel 3.8 mendeskripsikan struktur tabel `order_snapshots`, yang digunakan untuk menyimpan salinan (*snapshot*) data pesanan pada waktu tertentu. Tabel ini terhubung dengan tabel `orders` melalui *foreign key* dan menyimpan detail lengkap isi pesanan dalam format JSON. Fungsinya adalah untuk menjaga integritas data historis pesanan, terutama jika terjadi perubahan pada item atau struktur pesanan setelahnya.

Tabel 3.8. Struktur tabel `order_snapshots`

Nama Kolom	Tipe Data	Constraint
id (PK)	UUID	PRIMARY KEY
order_id (FK)	UUID	FOREIGN KEY, ON DELETE CASCADE
order_details_json	JSON	NOT NULL
created_at	TIMESTAMP	NOT NULL
updated_at	TIMESTAMP	NOT NULL

Penjelasan atribut tabel `order_snapshots`:

1. `id`
   Berfungsi sebagai *primary key* yang unik untuk setiap entri snapshot. Kolom ini menggunakan tipe data `UUID` guna menjamin keunikan.

2. `order_id`
   Merupakan *foreign key* yang mengacu pada kolom `order_id` di tabel `orders`. Kolom ini berfungsi untuk mengaitkan snapshot dengan pesanan tertentu. Jika data pada tabel `orders` dihapus, maka data yang berelasi pada tabel ini juga akan terhapus secara otomatis melalui aturan `ON DELETE CASCADE`.

3. `order_details_json`
   Menyimpan detail lengkap dari suatu pesanan dalam format JSON. Data yang disimpan mencakup informasi seperti item, harga, kuantitas, dan informasi lain yang berkaitan dengan pesanan pada saat snapshot dibuat. Kolom ini bersifat `NOT NULL`, sehingga wajib diisi pada setiap entri.

4. `created_at`
   Mencatat waktu saat snapshot dibuat. kolom ini bersifat `NOT NULL`, sehingga wajib diisi.

5. `updated_at`
   Mencatat waktu terakhir kali snapshot diperbarui. Kolom ini juga bersifat `NOT NULL`, sehingga wajib diisi.

3.5.2 Hasil Implementasi

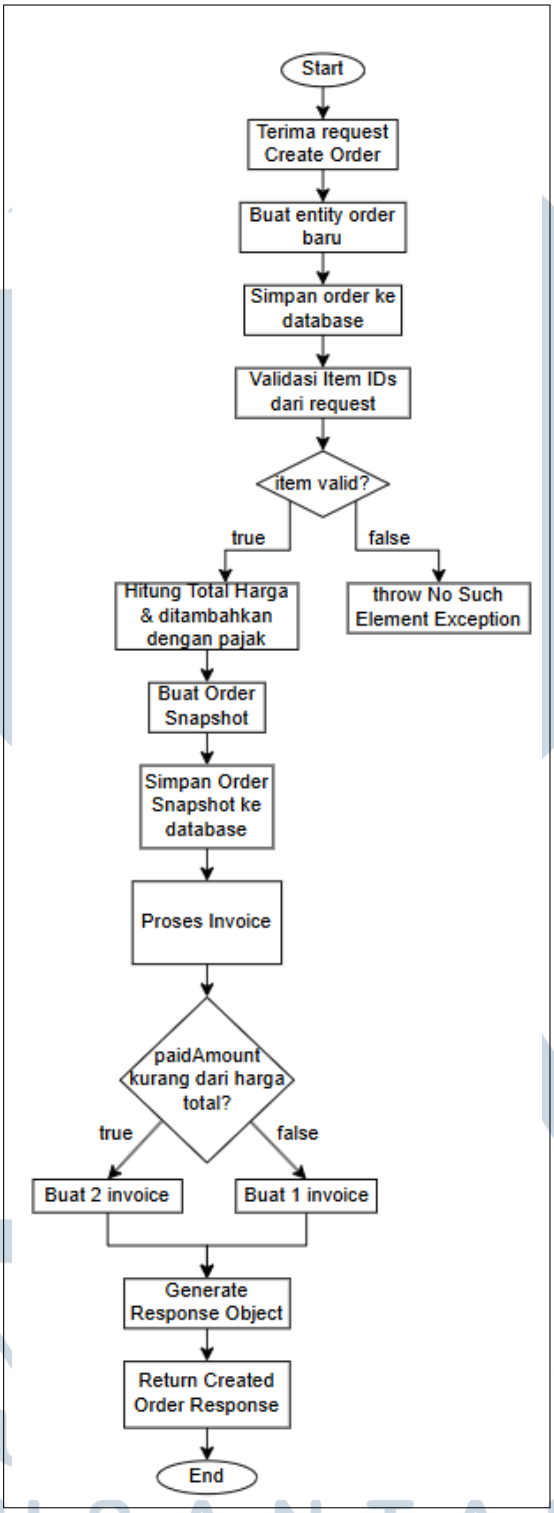
A *Create Order*

Endpoint ini digunakan untuk membuat pesanan baru. Saat permintaan dilakukan, sistem secara otomatis akan membentuk snapshot dari data pesanan serta menghasilkan invoice berdasarkan nominal pembayaran yang diberikan. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/orders/`.

Tabel 3.9 menjelaskan struktur data yang dikirimkan dalam *request body* saat melakukan permintaan untuk membuat pesanan. Setiap *field* memiliki peran penting dalam proses pembuatan pesanan, termasuk informasi mengenai item yang dipesan, jumlah pembayaran, serta detail metode dan tujuan pembayaran.

Tabel 3.9. *Request body* untuk *endpoint* pembuatan order

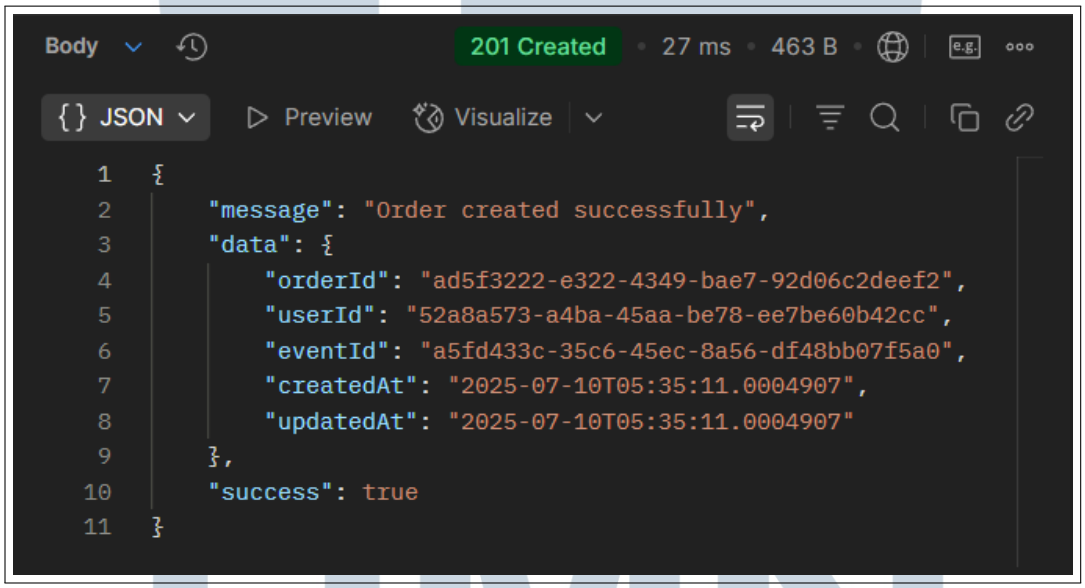
Field	Tipe Data	Deskripsi
`itemIds`	Array of UUID	Daftar ID item yang dipilih untuk dimasukkan ke dalam pesanan. Setiap ID mengacu pada entitas item yang telah tersedia di sistem.
`paidAmount`	Integer	Jumlah uang yang dibayarkan oleh pengguna. Digunakan untuk menentukan apakah klien melakukan pembayaran penuh atau parsial.
`paymentMethod`	String	Metode pembayaran yang digunakan, misalnya transfer bank (`"TF"`), atau `VA`.
`paymentSource`	String	Sumber dana atau rekening asal pembayaran pengguna. format yang digunakan adalah Bank_Rekening Contoh: `"Mandiri_123"`.
`paymentDestination`	String	Bank tujuan pembayaran tanpa menyertakan nomor rekening Contoh: `"BCA"`.



Gambar 3.2. Flowchart alur pembuatan order

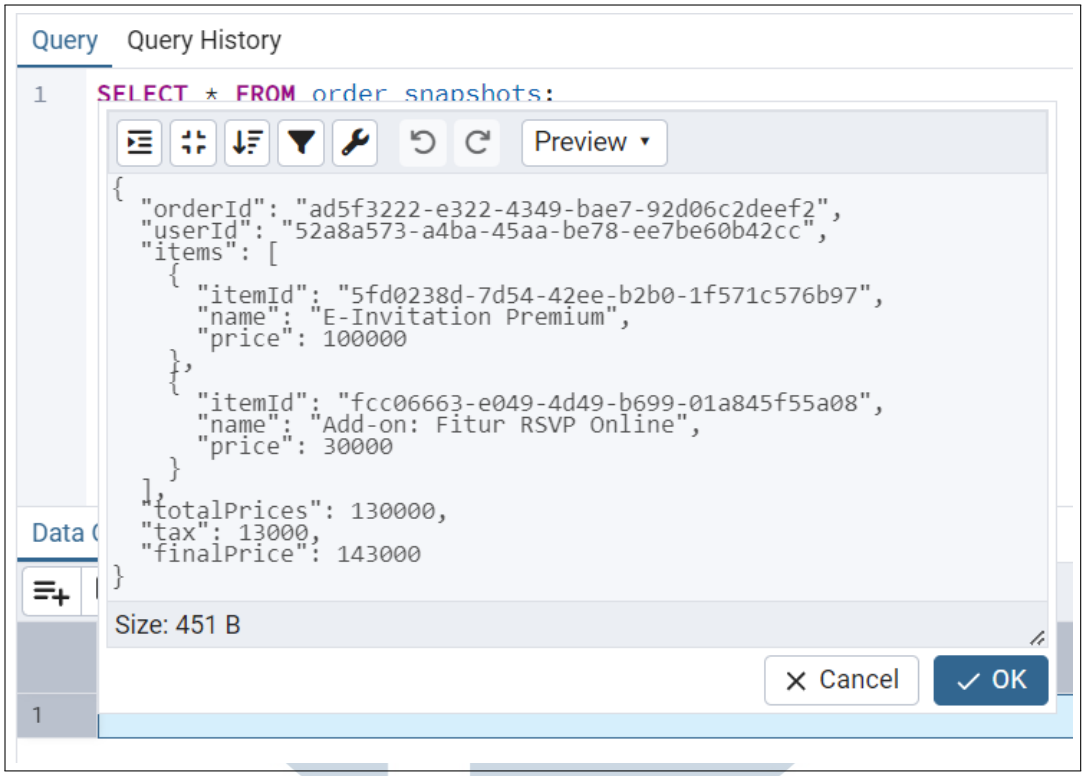
Gambar 3.2 menunjukkan alur proses lengkap pembuatan order dalam sistem. Proses dimulai dengan penerimaan *request* dan pembuatan *entity* order

baru. Setelah order berhasil disimpan ke basis data, sistem melakukan validasi terhadap item-item yang diminta dalam `itemIds`. Jika validasi gagal, sistem akan mengembalikan *No Such Element Exception*, namun jika berhasil, proses dilanjutkan dengan perhitungan total harga termasuk pajak. Tahap selanjutnya melibatkan pembuatan order snapshot yang berfungsi sebagai salinan data item dan harga pada saat order dibuat, memastikan konsistensi data meskipun terjadi perubahan harga item di kemudian hari. Setelah order snapshot tersimpan, sistem memproses pembuatan invoice berdasarkan nilai `paidAmount` yang diberikan. Proses diakhiri dengan pembentukan *response object* yang berisi informasi order yang telah berhasil dibuat.



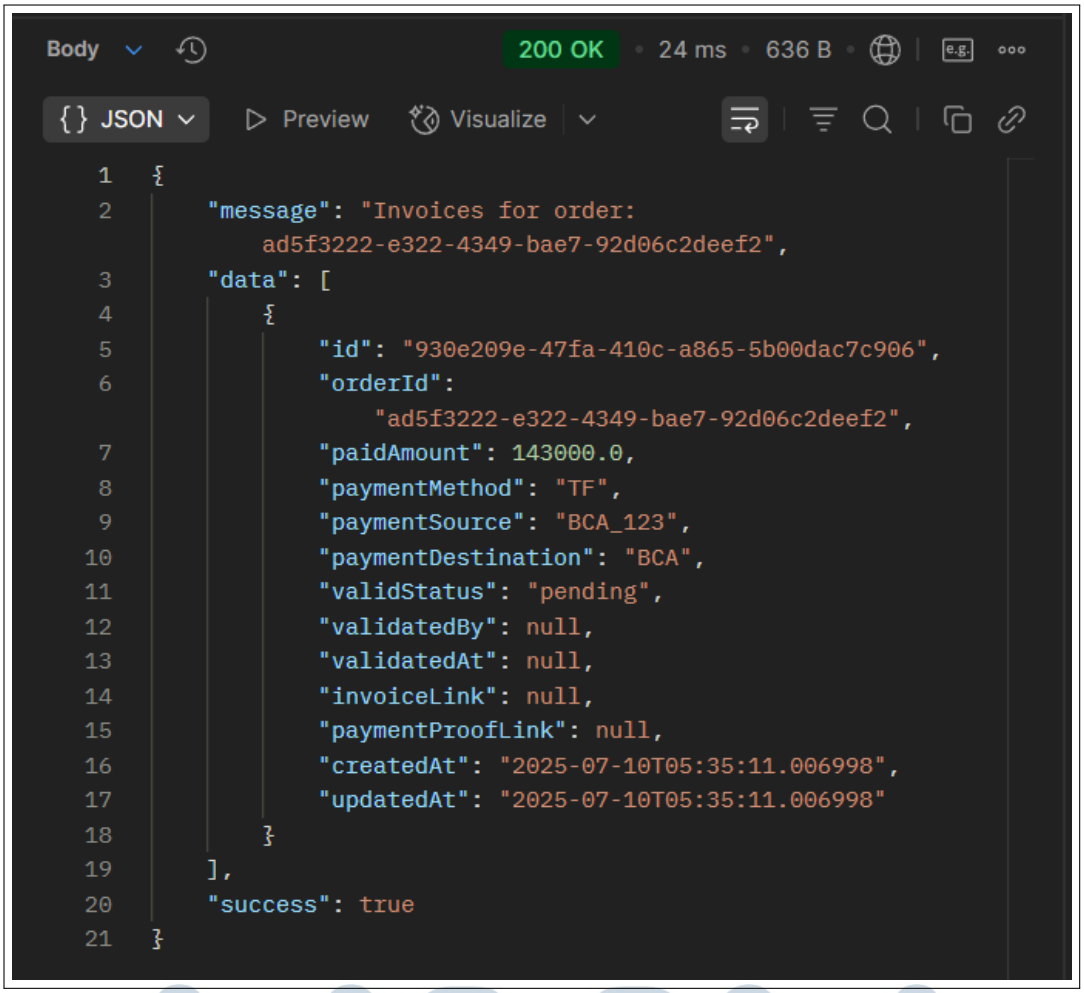
Gambar 3.3. Response untuk *endpoint* pembuatan order

Gambar 3.3 menunjukkan respons sistem setelah permintaan pembuatan order berhasil dilakukan dengan kode HTTP 201 (*Created*). Respons ini berisi informasi seperti ID order, ID pengguna, dan ID event, serta waktu pembuatan dan pembaruan data. Format respons ini akan menjadi standar untuk semua *endpoint* yang mengembalikan pesan, data, dan *success flag* dengan nilai *true* untuk operasi yang berhasil.



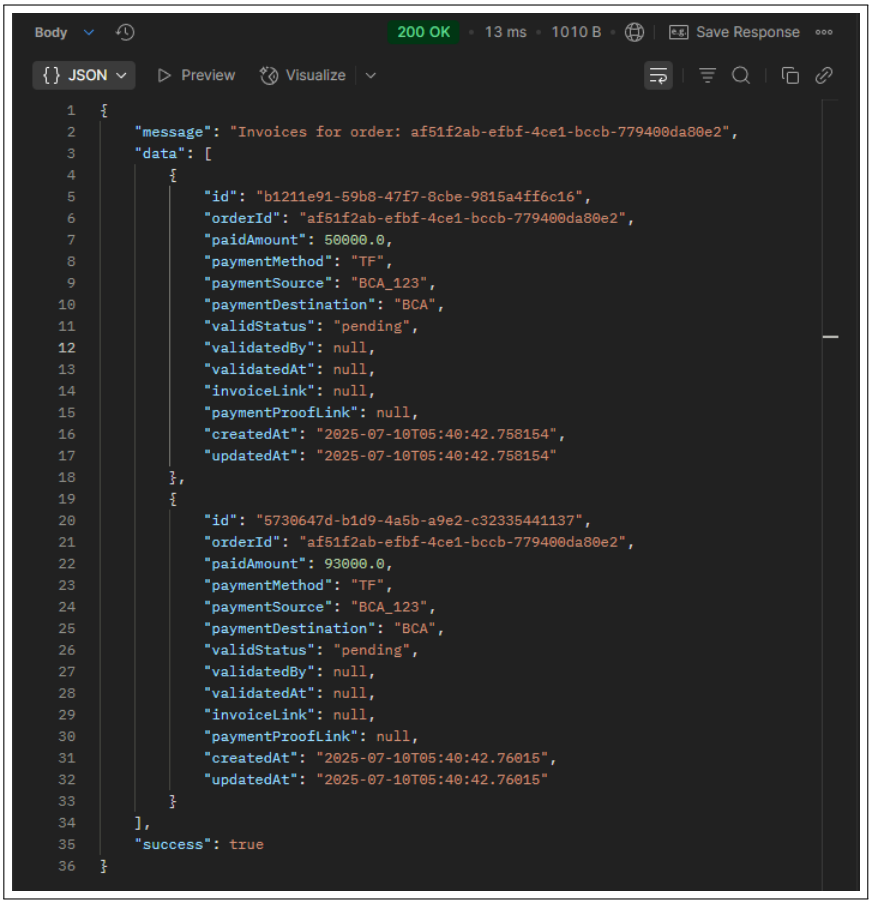
Gambar 3.4. Data snapshot yang dibuat setelah pembuatan order

Gambar 3.4 menunjukkan data order snapshot yang tercatat di basis data setelah pembuatan order, dengan fokus pada isi kolom `order_details_json`.



Gambar 3.5. Data invoice dari sebuah order jika dibayar penuh

Gambar 3.5 memperlihatkan satu invoice yang dihasilkan karena nilai `paidAmount` yang dimasukkan sesuai dengan total harga pesanan setelah pajak, yang menunjukkan bahwa pembayaran dilakukan secara penuh.

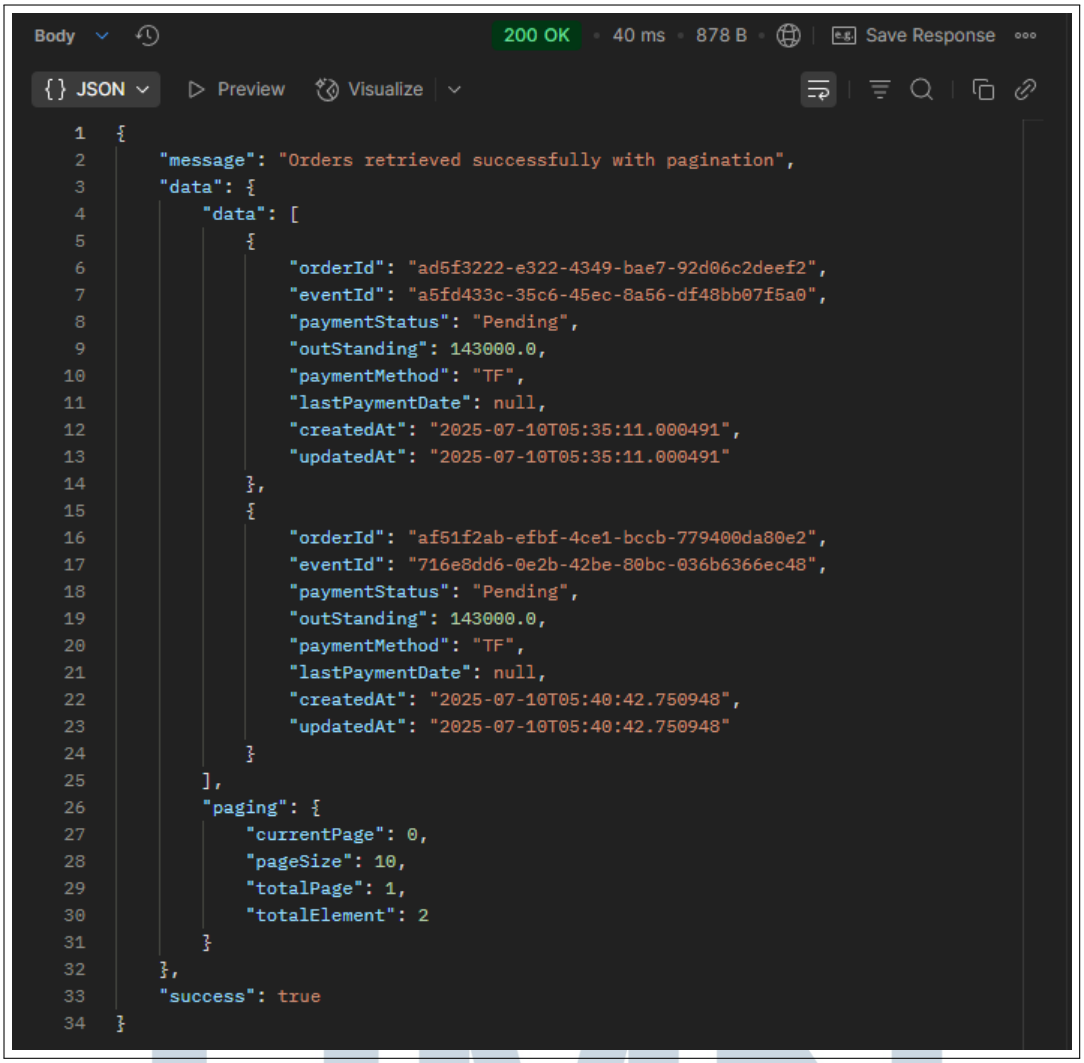


Gambar 3.6. Data invoice dari sebuah order jika dibayar parsial

Sementara itu, Gambar 3.6 menampilkan dua invoice yang terbentuk karena `paidAmount` yang dimasukkan lebih kecil dari total harga pesanan setelah pajak, sehingga menandakan bahwa pembayaran dilakukan secara parsial.

B *Get All Orders with Pagination*

Endpoint ini digunakan untuk mengambil daftar semua pesanan yang ada dalam sistem, dilengkapi dengan fitur *pagination* agar data yang ditampilkan tidak berlebihan dalam satu kali permintaan. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/orders/paginate`. Penggunaan metode POST dipilih untuk memungkinkan pengiriman parameter *page* untuk menentukan indeks halaman yang ingin di akses dan *size* untuk menentukan ukuran elemen dalam satu halaman melalui *request body*.



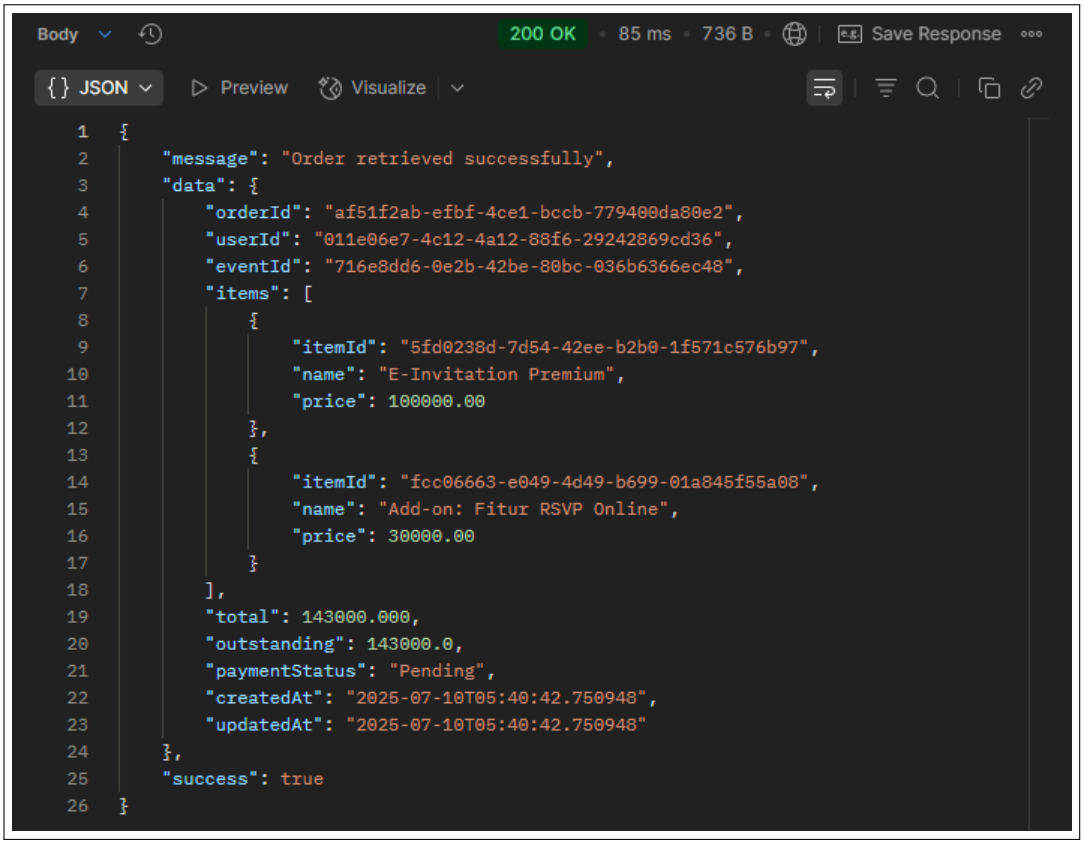
Gambar 3.7. Respons untuk *endpoint* daftar order dengan *pagination*

Gambar 3.7 menunjukkan respons sistem dari *endpoint* daftar order yang dilengkapi dengan fitur *pagination*. Respons tersebut mencakup data order yang terdiri atas ID order, ID event, status pembayaran, jumlah tagihan yang belum dibayar atau divalidasi(*outstanding*), metode pembayaran, tanggal pembayaran terakhir, waktu pembuatan dan pembaruan, serta informasi *pagination* seperti halaman saat ini, ukuran halaman, dan total data.

C *Get Order by ID*

Endpoint ini digunakan untuk menampilkan detail informasi suatu pesanan berdasarkan ID-nya, termasuk informasi terkait seperti data snapshot pesanan dan invoice yang terkait. Metode HTTP yang digunakan adalah `GET` dengan URL

`/api/payment/orders/{id}`.

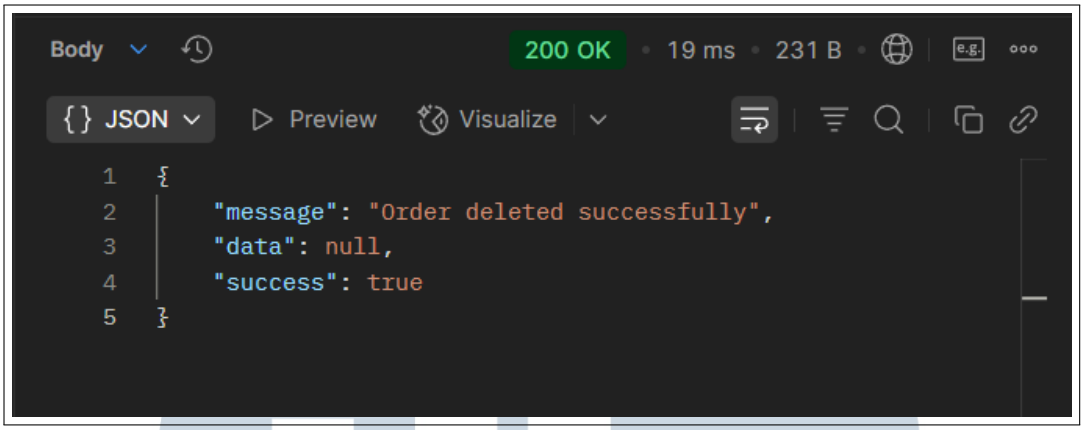


Gambar 3.8. Response untuk *endpoint* data order berdasarkan ID

Gambar 3.8 menampilkan detail lengkap dari suatu order berdasarkan ID tertentu. Data yang ditampilkan mencakup ID order, ID pengguna, ID event, daftar item beserta harga, total tagihan, jumlah tagihan yang belum dibayar atau divalidasi (*outstanding*), status pembayaran, serta informasi waktu pembuatan dan pembaruan.

D *Delete Order*

Endpoint ini digunakan untuk menghapus suatu pesanan berdasarkan ID-nya. Penghapusan ini juga akan menghapus entitas lain yang terkait, seperti invoice dan order snapshot, karena pengaturan `CASCADE` di basis data. Metode HTTP yang digunakan adalah `DELETE` dengan URL `/api/payment/orders/{id}`.



Gambar 3.9. Response untuk *endpoint* menghapus order

Gambar 3.9 menunjukkan respons sistem setelah berhasil melakukan penghapusan order.

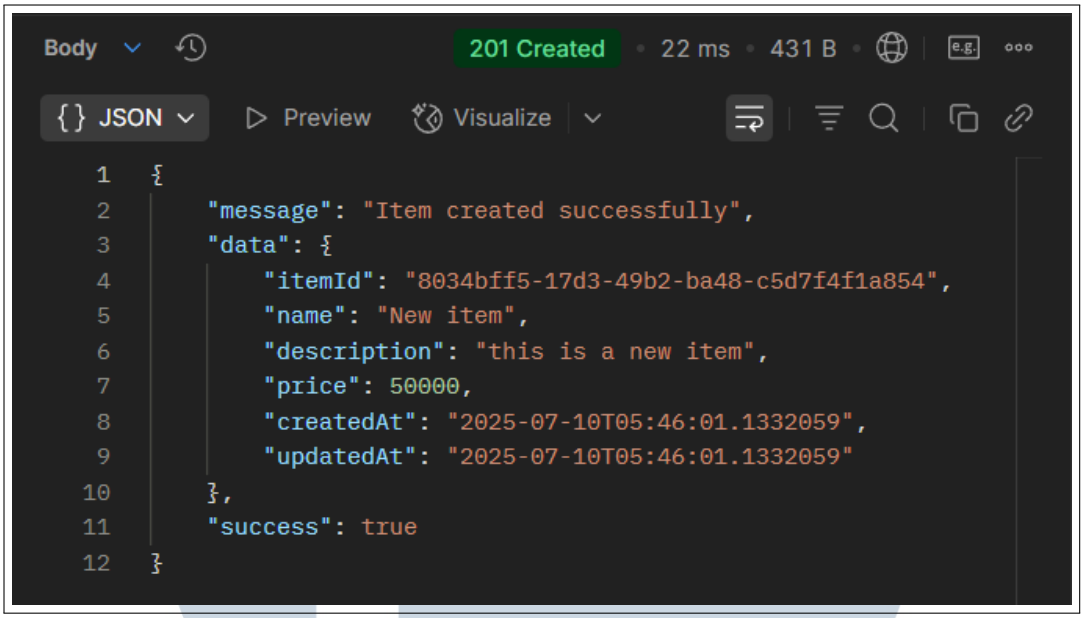
E *Create Item*

Endpoint ini digunakan untuk menambahkan data item baru ke dalam sistem. Item ini nantinya dapat dipilih dalam proses pembuatan order. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/items/`.

Tabel 3.10 menjelaskan struktur data yang dikirimkan dalam *request body* saat melakukan permintaan untuk membuat item baru. Setiap *field* memiliki peran penting dalam proses pembuatan item, yang mencakup nama item, deskripsi lengkap dari item tersebut, dan harga item.

Tabel 3.10. *Request body* untuk *endpoint* pembuatan item

Field	Tipe Data	Deskripsi
`name`	`string`	Nama item atau paket layanan yang ditawarkan.
`description`	`string`	Deskripsi lengkap mengenai layanan dalam item.
`price`	`number`	Harga item dalam satuan rupiah.

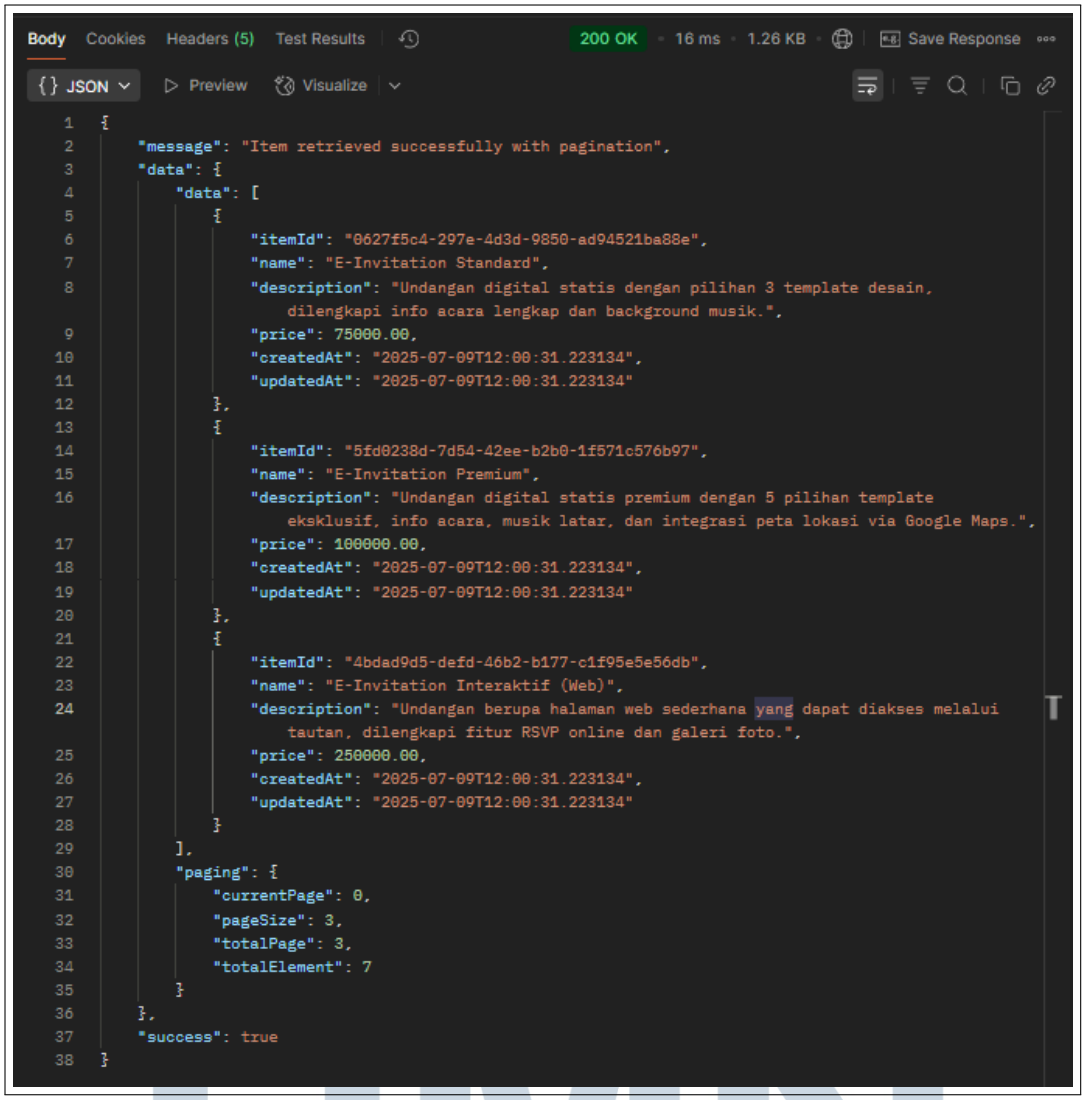


Gambar 3.10. Response untuk *endpoint* pembuatan item

Gambar 3.10 memperlihatkan respons sistem setelah permintaan pembuatan item berhasil dilakukan dengan kode HTTP 201. Respons ini berisi informasi seperti ID item, nama item, deskripsi item, harga item, serta waktu pembuatan dan pembaruan data.

F *Get Items with Pagination*

Endpoint ini digunakan untuk mengambil daftar item yang tersedia di dalam sistem, dengan fitur *pagination* agar proses pengambilan data lebih efisien. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/items/paginate`. Penggunaan metode POST memungkinkan pengiriman parameter *page* dan *size* melalui *request body*.

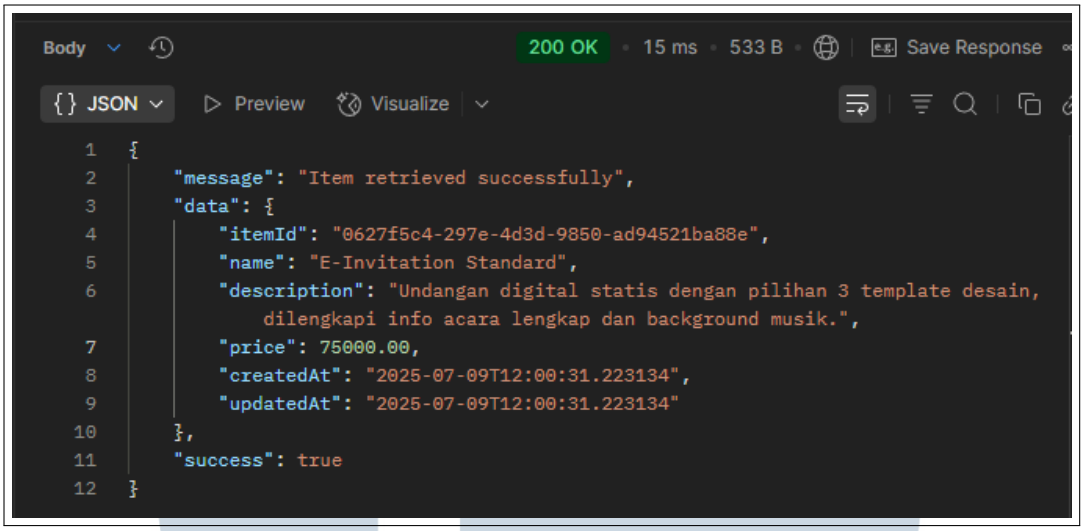


Gambar 3.11. Response untuk *endpoint* daftar item dengan *pagination*

Gambar 3.11 memperlihatkan daftar item yang berhasil diambil dengan *pagination*, beserta informasi *pagination* seperti halaman saat ini, ukuran halaman, dan total data.

G *Get Item by ID*

Endpoint ini digunakan untuk mengambil detail informasi dari suatu item berdasarkan ID-nya. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/items/{id}`.

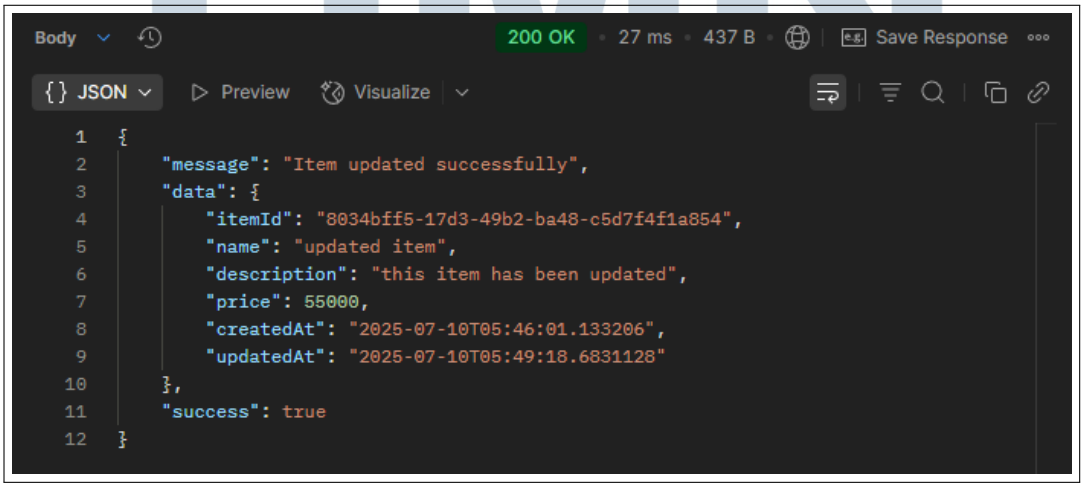


Gambar 3.12. Response untuk *endpoint* data item berdasarkan ID

Gambar 3.12 menunjukkan data item spesifik yang diambil berdasarkan ID-nya yang mencakup ID item, nama item, deskripsi item, harga item, serta waktu pembuatan dan pembaruan.

H *Update Item*

Endpoint ini digunakan untuk memperbarui informasi suatu item berdasarkan ID-nya. Perubahan bisa mencakup nama, deskripsi, atau harga item. Metode HTTP yang digunakan adalah `PUT` dengan URL `/api/payment/items/{id}`.

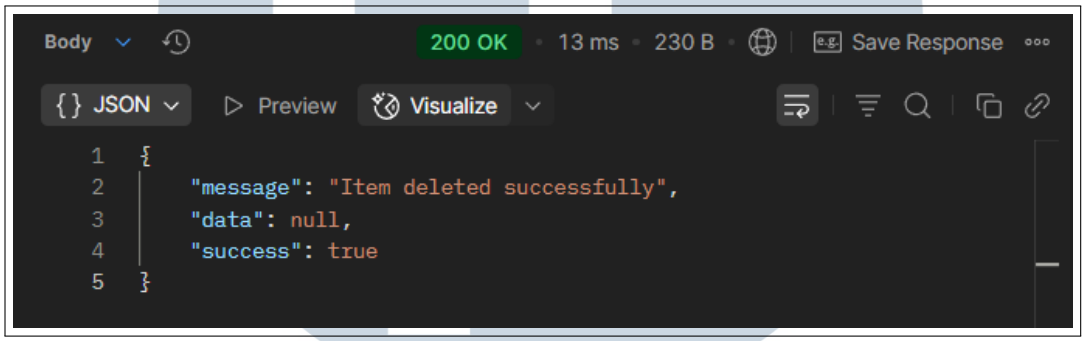


Gambar 3.13. Response untuk *endpoint* pembaruan item

Gambar 3.13 menunjukkan respons sistem setelah data item berhasil diperbarui. Respons mengembalikan data item yang telah diperbarui dengan informasi terbaru.

I *Delete Item*

Endpoint ini digunakan untuk menghapus suatu item dari sistem berdasarkan ID-nya. Metode HTTP yang digunakan adalah `DELETE` dengan URL `/api/payment/items/{id}`

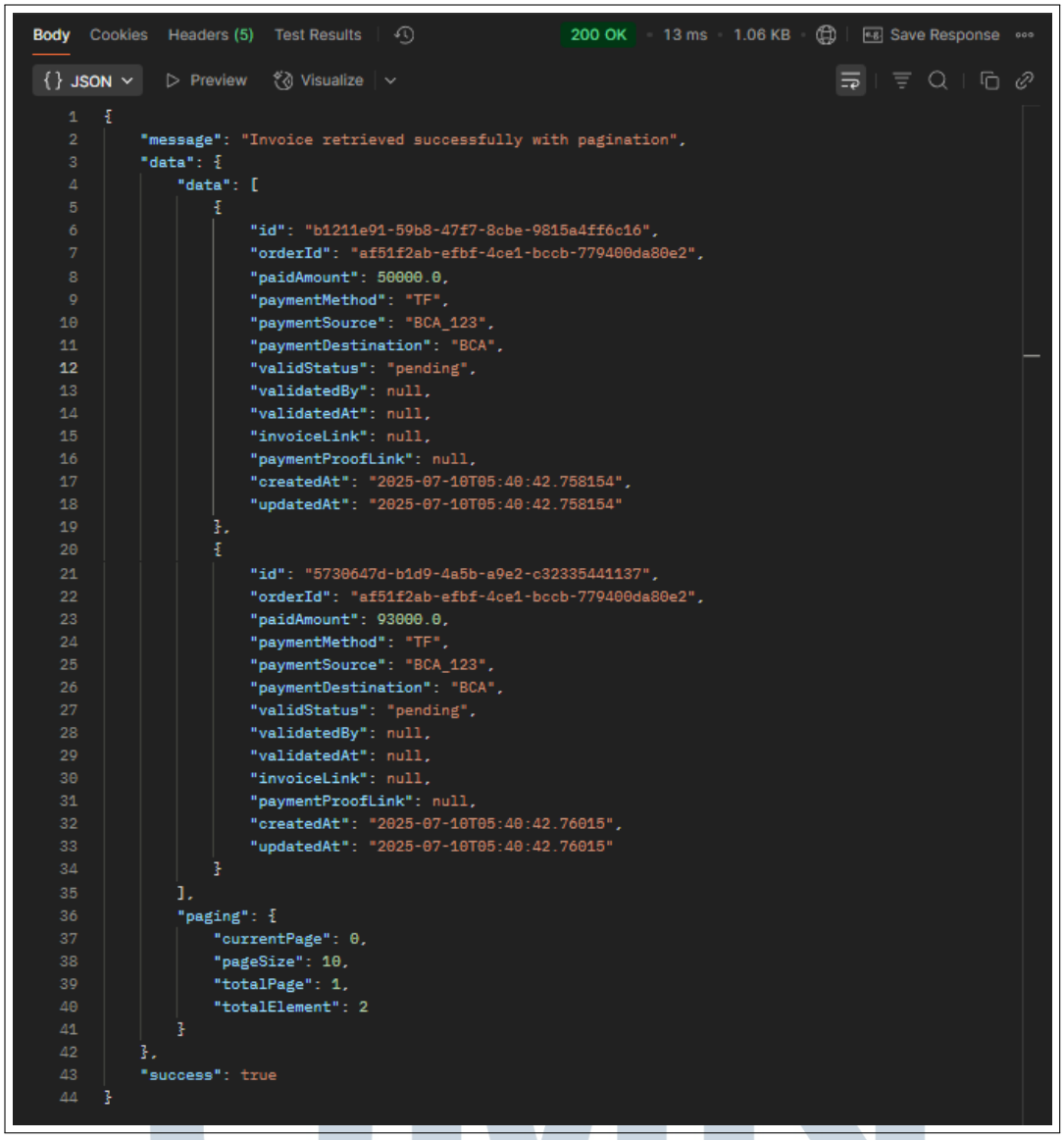


Gambar 3.14. Response untuk *endpoint* penghapusan item

Gambar 3.14memperlihatkan respons setelah proses penghapusan item berhasil dilakukan.

J *Get Invoices with Pagination*

Endpoint ini digunakan untuk mengambil daftar invoice yang tersedia di sistem dengan dukungan *pagination* agar data tidak terlalu banyak dalam satu permintaan. Metode HTTP yang digunakan adalah `POST` dengan URL `/api/payment/invoices/paginate`.

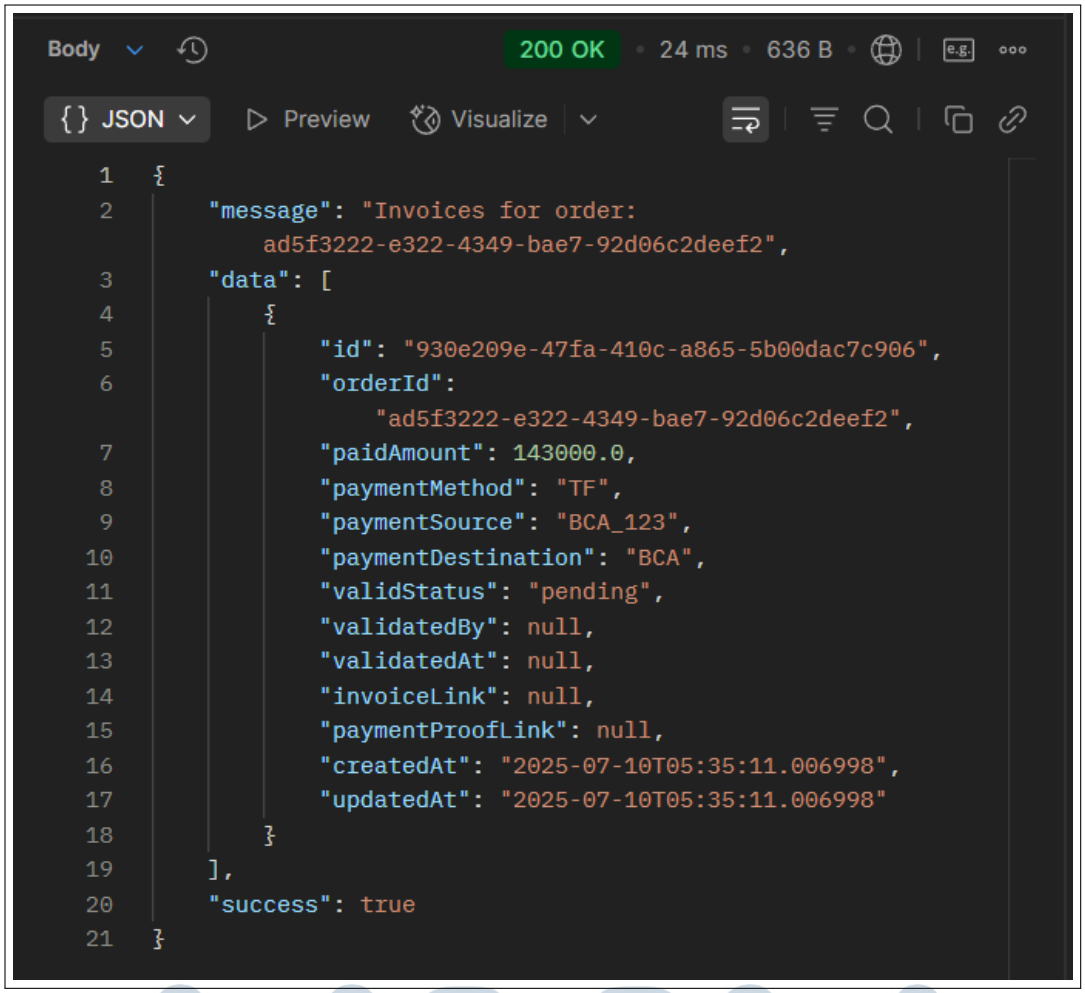


Gambar 3.15. Response untuk *endpoint* daftar invoice dengan *pagination*

Gambar 3.15 menampilkan hasil dari *endpoint* untuk mengambil daftar invoice dengan dukungan *pagination*, beserta informasi *pagination* seperti halaman saat ini, ukuran halaman, dan total data.

K *Get Invoices by Order ID*

Endpoint ini digunakan untuk mengambil invoice yang terkait dengan suatu pesanan tertentu berdasarkan ID order-nya. Metode HTTP yang digunakan adalah `GET` dengan URL `/api/payment/orders/{id}/invoices`.



Gambar 3.16. Response untuk *endpoint* data invoice berdasarkan ID order

Gambar 3.16 menunjukkan respons sistem untuk mengambil daftar invoice yang terkait dengan suatu order ID tertentu. Respons mengembalikan informasi invoice yang berkaitan dengan order yang diminta, dengan setiap invoice memuat data lengkap terkait pembayaran.

Setiap invoice dalam respons berisi informasi seperti ID invoice, ID order terkait, ID pengguna, jumlah tagihan (`paidAmount`), metode pembayaran, sumber dan tujuan pembayaran, serta status validasi. Dalam contoh ini terlihat status validasi masih dalam kondisi `pending`, dengan field `validatedBy`, `validatedAt`, `invoiceLink`, dan `paymentProofLink` yang masih bernilai `null` karena belum melalui proses validasi.

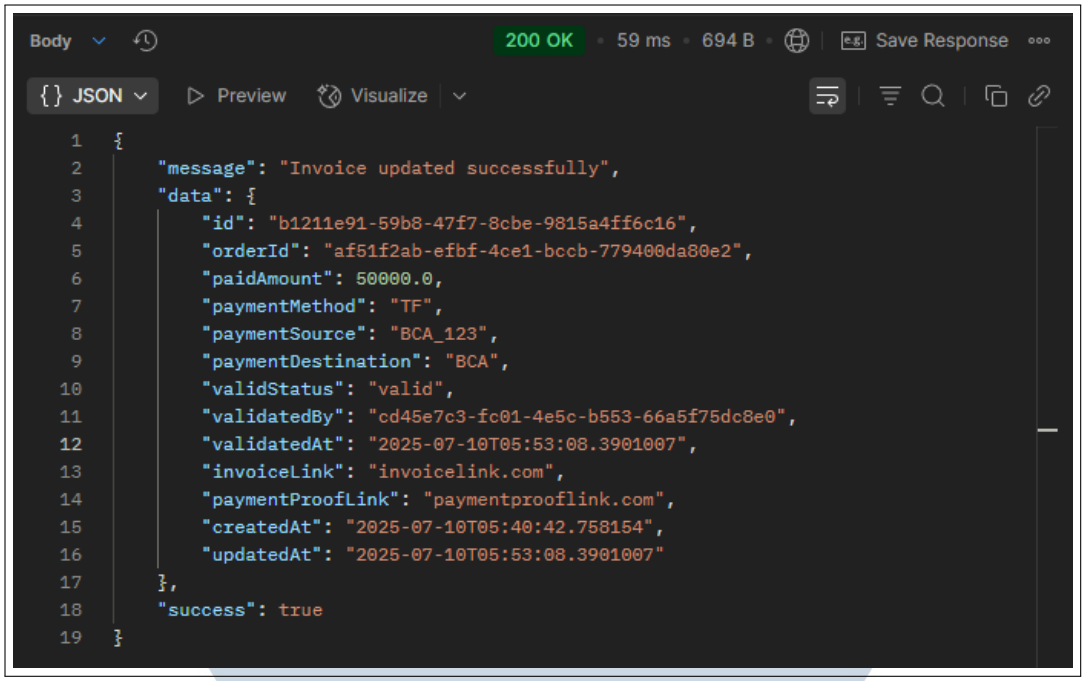
L *Update Invoice*

Endpoint ini digunakan untuk memperbarui data pada suatu invoice, seperti status status validasi, identitas pihak yang melakukan validasi, waktu validasi, serta tautan menuju *file* invoice dan bukti pembayaran. Metode HTTP yang digunakan adalah `PATCH` dengan URL `/api/payment/invoices/{id}`.

Tabel 3.11 menjelaskan struktur data yang dikirimkan dalam *request body* saat melakukan permintaan untuk memperbarui data invoice. Setiap *field* dalam permintaan ini berperan penting dalam proses validasi invoice, seperti identitas pengguna, status validasi, pihak yang melakukan validasi, serta tautan yang mengarah ke invoice dan bukti pembayaran.

Tabel 3.11. *Body request* untuk *endpoint* pembaruan invoice

Field	Tipe Data	Deskripsi
`validStatus`	`string`	Status validasi invoice, seperti `"valid"` atau `"invalid"`.
`validatedBy`	`UUID`	ID pengguna yang melakukan validasi terhadap invoice.
`invoiceLink`	`string`	URL yang mengarah ke file atau tampilan invoice.
`paymentProofLink`	`string`	URL bukti pembayaran yang diunggah oleh pengguna.

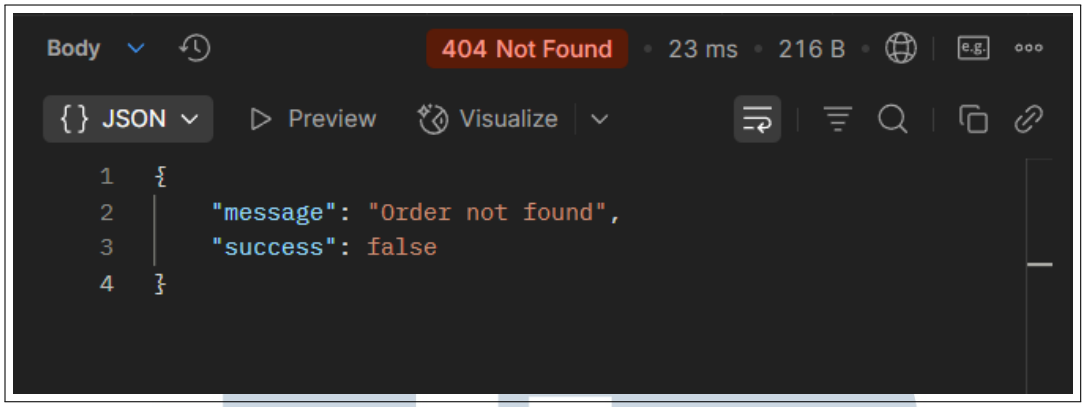


Gambar 3.17. Response untuk *endpoint* pembaruan invoice

Gambar 3.17 menunjukkan respons sistem setelah data invoice berhasil diperbarui, berupa data detail invoice yang telah diperbarui.

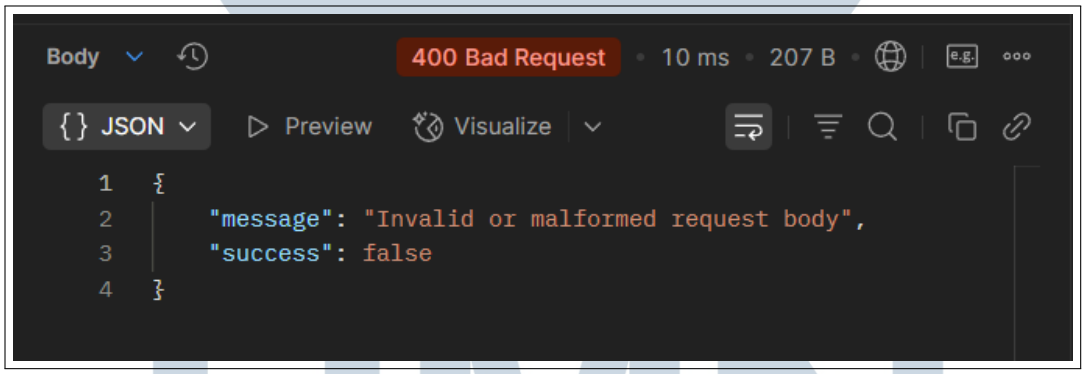
M *Error Handling*

Selain implementasi terhadap berbagai *endpoint*, sistem juga dirancang untuk menangani kasus kesalahan (*error*) secara terpusat menggunakan mekanisme *Global Exception Handler*. Pendekatan ini memastikan bahwa setiap kesalahan yang terjadi selama pemrosesan permintaan akan menghasilkan respons dengan format yang konsisten, dilengkapi kode status HTTP yang sesuai serta pesan kesalahan yang informatif. Struktur respons kesalahan terdiri dari dua atribut utama, yaitu pesan *error* dan *success flag* yang bernilai *false*.



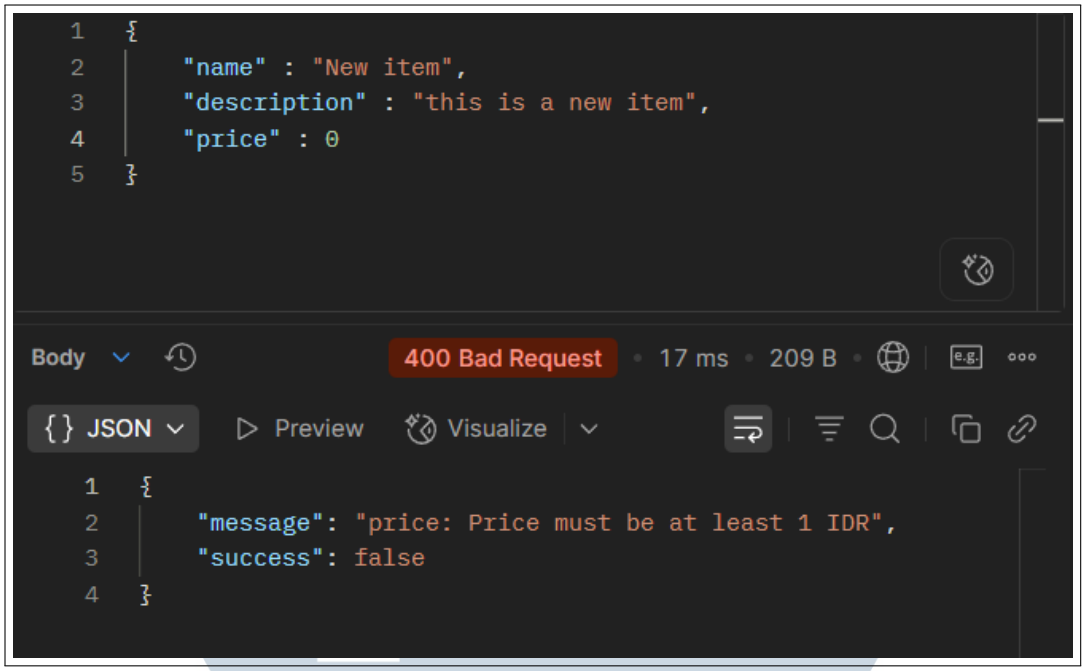
Gambar 3.18. Response error untuk data yang tidak ditemukan

Gambar 3.18 menunjukkan respons yang dihasilkan apabila data yang diminta tidak ditemukan. Dalam kasus ini, sistem akan memberikan pesan yang sesuai dengan entitas yang dicari serta mengembalikan kode status 404 (*Not Found*).



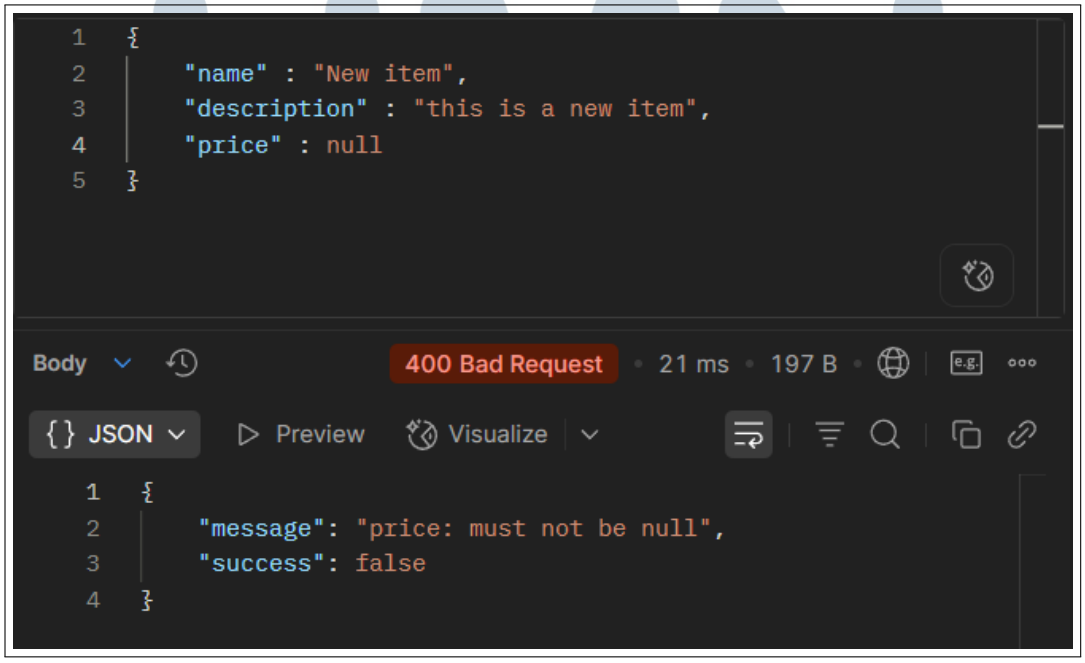
Gambar 3.19. Respons error untuk penulisan request body yang kurang tepat

Gambar 3.19 memperlihatkan respons yang muncul apabila *request body* tidak ditulis dengan benar. Contoh kasus yang ditampilkan adalah adanya tanda koma di akhir struktur JSON, yang menyebabkan sistem mengembalikan pesan "*Invalid or malformed request body*" dengan status kode 400 (*Bad Request*).



Gambar 3.20. Respons error untuk parameter *request body* yang tidak sesuai (1)

Gambar 3.20 menunjukkan respons jika terdapat parameter dalam *request body* yang tidak sesuai dengan ketentuan validasi. Pada kasus ini, parameter `price` memiliki nilai kurang dari 1 IDR. Sistem akan memberikan pesan kesalahan yang menjelaskan masalah tersebut dan mengembalikan status kode 400 (*Bad Request*).



Gambar 3.21. Respons error untuk parameter *request body* yang tidak sesuai (2)

Gambar 3.21 memberikan contoh lain dari kesalahan validasi parameter, yaitu ketika parameter `price` tidak disertakan dalam *request body*. Sama seperti sebelumnya, sistem akan memberikan respons kesalahan yang relevan beserta status kode 400 (*Bad Request*).

3.6 Pengujian Sistem

Bagian ini menyajikan hasil pengujian sistem yang dilakukan menggunakan metode *Black Box Testing* untuk memastikan bahwa setiap endpoint yang diimplementasikan telah berfungsi sesuai dengan spesifikasi yang diharapkan. Pengujian mencakup berbagai skenario, baik untuk kasus yang valid maupun tidak valid. Seluruh pengujian dilakukan secara manual menggunakan Postman oleh Supervisor magang, Jesivinica Santoso. Tabel 3.12 merangkum deskripsi pengujian, hasil yang diharapkan, hasil aktual dari sistem, serta kesimpulan pengujian berdasarkan kesesuaian antara output dan ekspektasi.

Tabel 3.12. Hasil Pengujian Sistem

No	Pengujian	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
1	Pembuatan pesanan baru dengan data valid	HTTP 201 Created + data order	Pesanan berhasil dibuat, HTTP 201 Created	Sesuai
2	Pembuatan pesanan dengan item tidak tersedia	HTTP 404 Not Found + pesan error	Error 404 Not Found, item tidak ditemukan	Sesuai
3	Ambil daftar pesanan dengan *pagination*	HTTP 200 OK + list order	Data list pesanan berhasil diambil, HTTP 200 OK	Sesuai
4	Ambil detail pesanan dengan ID valid	HTTP 200 OK + detail order	Detail pesanan berhasil diambil, HTTP 200 OK	Sesuai

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.12

No	Pengujian	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
5	Ambil detail pesanan dengan ID tidak ditemukan	HTTP 404 Not Found + pesan error	Error 404 Not Found	Sesuai
6	Hapus pesanan dengan ID valid	HTTP 200 OK	Pesanan berhasil dihapus, HTTP 200 OK	Sesuai
7	Tambah item baru dengan data lengkap	HTTP 201 Created + data item	Data berhasil ditambahkan, HTTP 201 Created	Sesuai
8	Tambah item dengan *request body* kosong	HTTP 400 Bad Request + pesan error	Error 400 Bad Request	Sesuai
9	Ambil daftar item dengan *pagination*	HTTP 200 OK + list item	Data list item berhasil diambil, HTTP 200 OK	Sesuai
10	Ambil item berdasarkan ID valid	HTTP 200 OK + detail item	Detail item berhasil diambil, HTTP 200 OK	Sesuai
11	Perbarui item dengan ID valid	HTTP 200 OK + updated item	Item berhasil diperbarui, HTTP 200 OK	Sesuai
12	Hapus item dengan ID tidak valid	HTTP 404 Not Found	Error 404 Not Found	Sesuai
13	Ambil daftar invoice dengan *pagination*	HTTP 200 OK + list invoice	Data list invoice berhasil diambil, HTTP 200 OK	Sesuai

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.12

No	Pengujian	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
14	Ambil invoice berdasarkan ID order valid	HTTP 200 OK + data invoice	Data invoice berhasil diambil, HTTP 200 OK	Sesuai
15	Validasi invoice dengan data lengkap	HTTP 200 OK + updated invoice	Invoice berhasil divalidasi, HTTP 200 OK	Sesuai
16	Validasi invoice dengan `validStatus` kosong	HTTP 400 Bad Request + pesan error	Error 400 Bad Request	Sesuai
17	Kirim request body dengan format JSON tidak valid (syntax error)	HTTP 400 Bad Request + error message	Error 400 Bad Request	Sesuai
18	Kirim nilai *numeric* yang tidak sesuai validasi (misal price kurang dari 1)	HTTP 400 Bad Request + pesan validasi	Error 400 Bad Request	Sesuai

3.7 Kendala dan Solusi yang Ditemukan

A Kendala

Beberapa kendala yang ditemukan selama periode magang adalah sebagai berikut:

1. Perubahan *requirement* yang terjadi secara berkala menyebabkan proses pengembangan harus mengalami penyesuaian ulang, baik pada sisi alur bisnis maupun struktur data, yang kemudian berdampak pada mundurnya progres pengembangan.

2. Kurangnya koordinasi dan komunikasi yang maksimal dengan divisi

lain, khususnya divisi yang bertanggung jawab terhadap sistem eksternal yang akan diintegrasikan, menghambat kejelasan informasi teknis yang dibutuhkan.

3. Terjadi situasi saling menunggu keputusan atau masukan dari pihak lain, yang menyebabkan pekerjaan menjadi stagnan dalam beberapa periode dan tidak dapat dilanjutkan secara mandiri oleh tim *backend*.

B Solusi

Untuk mengatasi kendala-kendala yang ditemukan, beberapa solusi yang diterapkan antara lain:

1. Untuk menghadapi perubahan *requirement*, dilakukan dokumentasi perubahan secara rutin serta pembaruan pada alur pengembangan agar lebih adaptif. Selain itu, setiap perubahan yang bersifat mayor dikomunikasikan lebih awal kepada *supervisor* guna mendapatkan validasi sebelum diterapkan.
2. Koordinasi lintas divisi ditingkatkan dengan cara menjadwalkan sesi diskusi teknis yang lebih rutin dan langsung menghubungi PIC terkait apabila terdapat ketidakjelasan teknis. Pendekatan ini membantu mempercepat klarifikasi dan mencegah penundaan yang berlarut-larut.
3. Untuk menghindari stagnasi akibat menunggu pihak lain, diterapkan prinsip inisiatif aktif, di mana anggota tim berusaha mengambil langkah-langkah yang memungkinkan untuk tetap melanjutkan pekerjaan lain yang tidak bergantung pada input tersebut. Selain itu, dilakukan eskalasi kepada *supervisor* jika diperlukan untuk mempercepat pengambilan keputusan.

Dengan solusi tersebut, proses kerja magang menjadi lebih efisien, tetap bergerak meski di tengah kendala, dan menjaga alur kerja tim tetap produktif.