

Tugas Sistem Basis Data

Membuat REST API dengan Node.js melalui URL

Kelompok 12:

- Jonathan Frederick Kosasih (2306225981)
- Bonifasius Raditya Pandu Hendrianto (2306242350)
- Wesley Frederick Oh (23062027630)

Link github ke files (jika diperlukan):

<https://github.com/sleepingpolice-afk/kelompoksbdb>

Requirements:

- **npm install pg dotenv**
- **npm install cors**

Cara menghidupkan server:

ketik **node index.js** dalam root folder.

Tools yang digunakan:

- Postman: Untuk testing REST API dalam URL.
- Visual Studio Code: Code Editor untuk JavaScript.
- NeonTech Serverless Postgres: Untuk menunjukkan interaksi antara database non-local dan API.
- Postgresql
- Node.js

DOKUMENTASI

Query yang digunakan:

```
CREATE TYPE categories AS ENUM ('Makanan', 'Minuman', 'Snack', 'Perlengkapan Rumah',
'Perlengkapan Dapur', 'Perlengkapan Kamar Mandi', 'Perlengkapan Tidur',
'Perlengkapan Sekolah', 'Perlengkapan Kantor', 'Perlengkapan Elektronik',
'Perlengkapan Olahraga', 'Perlengkapan Hewan', 'Perlengkapan Bayi', 'Perlengkapan
Anak', 'Perlengkapan Dewasa', 'Perlengkapan Lansia', 'Perlengkapan Daur Ulang',
'Perlengkapan Kesehatan', 'Perlengkapan Fashion', 'Perlengkapan Kecantikan',
'Perlengkapan Hobi', 'Perlengkapan Lainnya');
CREATE TYPE status_pesanan AS ENUM ('Menunggu Pembayaran', 'Diproses', 'Dikirim',
'Selesai', 'Dibatalkan');
CREATE TYPE status_pembayaran AS ENUM('Menunggu Konfirmasi', 'Diterima', 'Ditolak');
CREATE TYPE methods AS ENUM('Transfer Bank', 'Kartu Kredit', 'E-wallet');

CREATE TABLE produk (
    id_produk SERIAL PRIMARY KEY,
    nama_produk VARCHAR(255) NOT NULL,
    kategori categories DEFAULT 'Perlengkapan Lainnya',
    harga BIGINT CHECK(harga >= 0),
    stok INT CHECK(stok >= 0),
    deskripsi TEXT
);

CREATE TABLE users (
    id_user SERIAL PRIMARY KEY,
    nama VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    password VARCHAR(255),
    alamat TEXT,
    no_telepon VARCHAR(15)
);

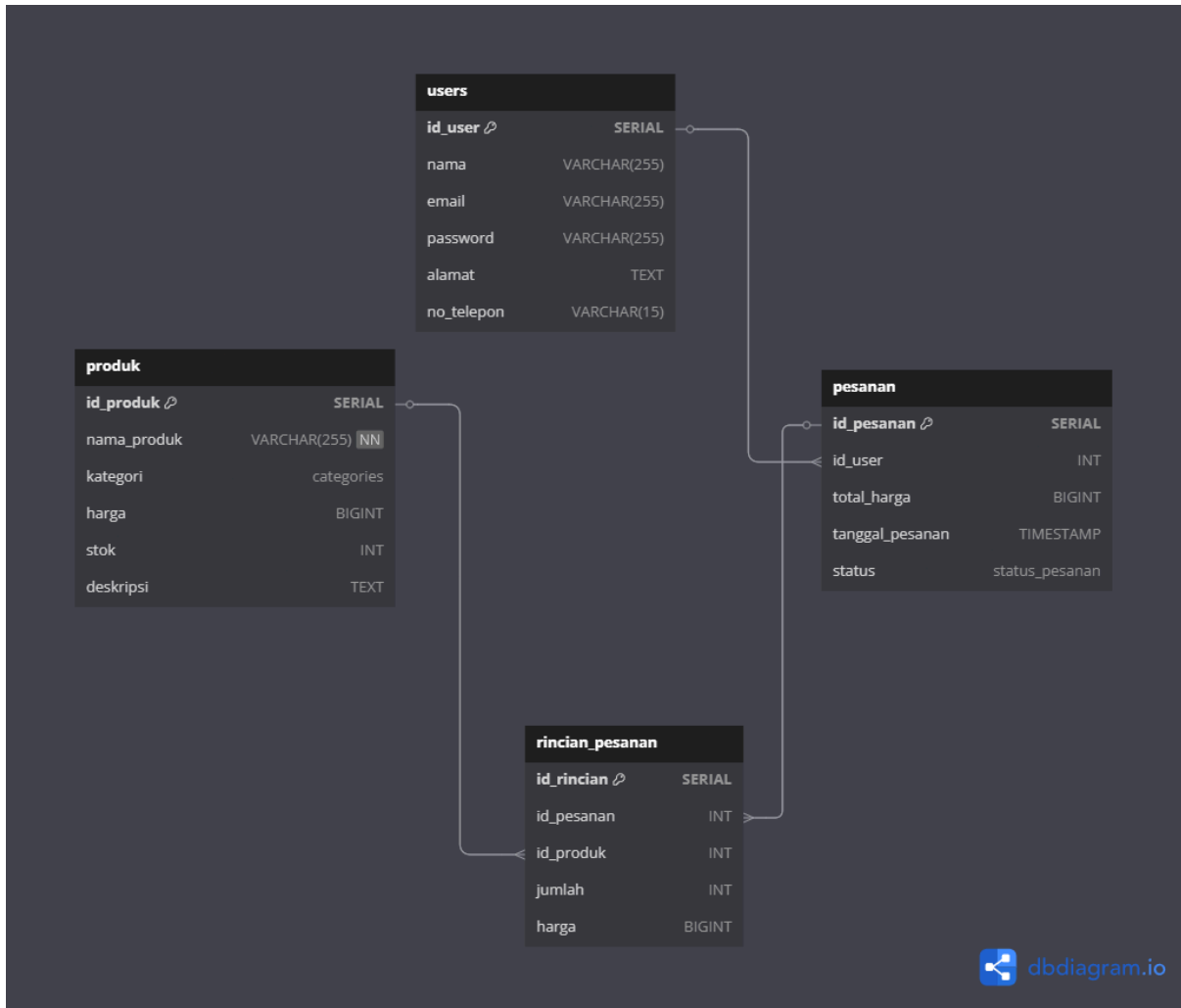
CREATE TABLE pesanan (
    id_pesanan SERIAL PRIMARY KEY,
    id_user INT,
    total_harga BIGINT CHECK(total_harga >= 0),
    tanggal_pesanan TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status status_pesanan DEFAULT 'Menunggu Pembayaran',
    FOREIGN KEY (id_user) REFERENCES users(id_user) ON DELETE CASCADE
);

CREATE TABLE rincian_pesanan (
    id_rincian SERIAL PRIMARY KEY,
    id_pesanan INT,
    id_produk INT,
    jumlah INT CHECK(jumlah >= 0),
```

```

harga BIGINT CHECK(harga >= 0),
FOREIGN KEY (id_pesanan) REFERENCES pesanan(id_pesanan) ON DELETE CASCADE,
FOREIGN KEY (id_produk) REFERENCES produk(id_produk) ON DELETE CASCADE
);

```



Penjelasan

Jadi dalam database transaksi toko online yang kami buat, terdapat 4 buah table, yaitu:

- **produk**
Table produk digunakan untuk menyimpan informasi tentang produk yang terdapat di toko, mulai dari nama produk, kategorinya, harga, stok, dan deskripsi produk.
- **users**
Table users, seperti namanya yaitu menyimpan informasi atas user yang teregister dalam database ini. Table ini memiliki beberapa kolom, yaitu nama, email, password, alamat, dan nomor telepon user.
- **pesanan**
Untuk table pesanan, table ini menyimpan informasi atas pesanan yang dibuat oleh seorang user. Oleh karena itu, table ini memiliki relation dengan table users, serta

memiliki kolom seperti total harga, tanggal pesanan, dan status pesanan tersebut apakah sudah dibayar atau belum.

- rincian_pesanan

Table ini berperan sebagai junction table antara table produk dan table pesanan.

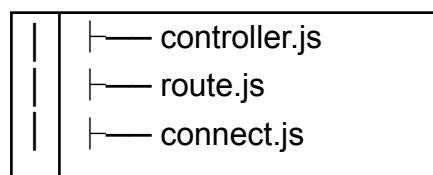
Alasan perlu dibuatnya junction table ini adalah karena adanya hubungan many-to-many antara pesanan dan produk, yang artinya sebuah produk bisa dipesan oleh beberapa pesanan sekaligus, dan sebuah pesanan juga bisa memesan beberapa produk sekaligus. Jadi, dibuatlah table rincian_pesanan yang menyimpan informasi atas pesanan dan produk yang dipesan sehingga table ini memiliki beberapa kolom seperti jumlah, harga, dan beberapa foreign key seperti id_pesanan dan id_produk untuk menghubungkan table ini dengan table pesanan dan table produk.

Struktur Backend

Struktur penempatan file dan folder adalah seperti berikut:

/SBD_Kelompok12_REST API.zip

├── src/



├── .env

├── SBD_Kelompok12_REST API.pdf

├── query.sql

├── index.js

├── package.json

└── package.lock.json

Keterangan:

- **controller.js**: Adalah file yang berisi seluruh function API yang digunakan untuk memasukkan query postgresql, serta mereturn hasilnya, apakah error ataupun berhasil.
- **route.js**: Adalah file yang menspesifikasikan URL yang digunakan untuk menggunakan function yang bersangkutan dengan URL tersebut. Misalnya route /product/get akan memanggil function getProduct di controller dan kemudian mereturn seluruh isi table produk, ataupun pesanan/update yang memanggil function

updatePesanan dan menggunakan query UPDATE untuk memperbarui sebuah pesanan di table pesanan.

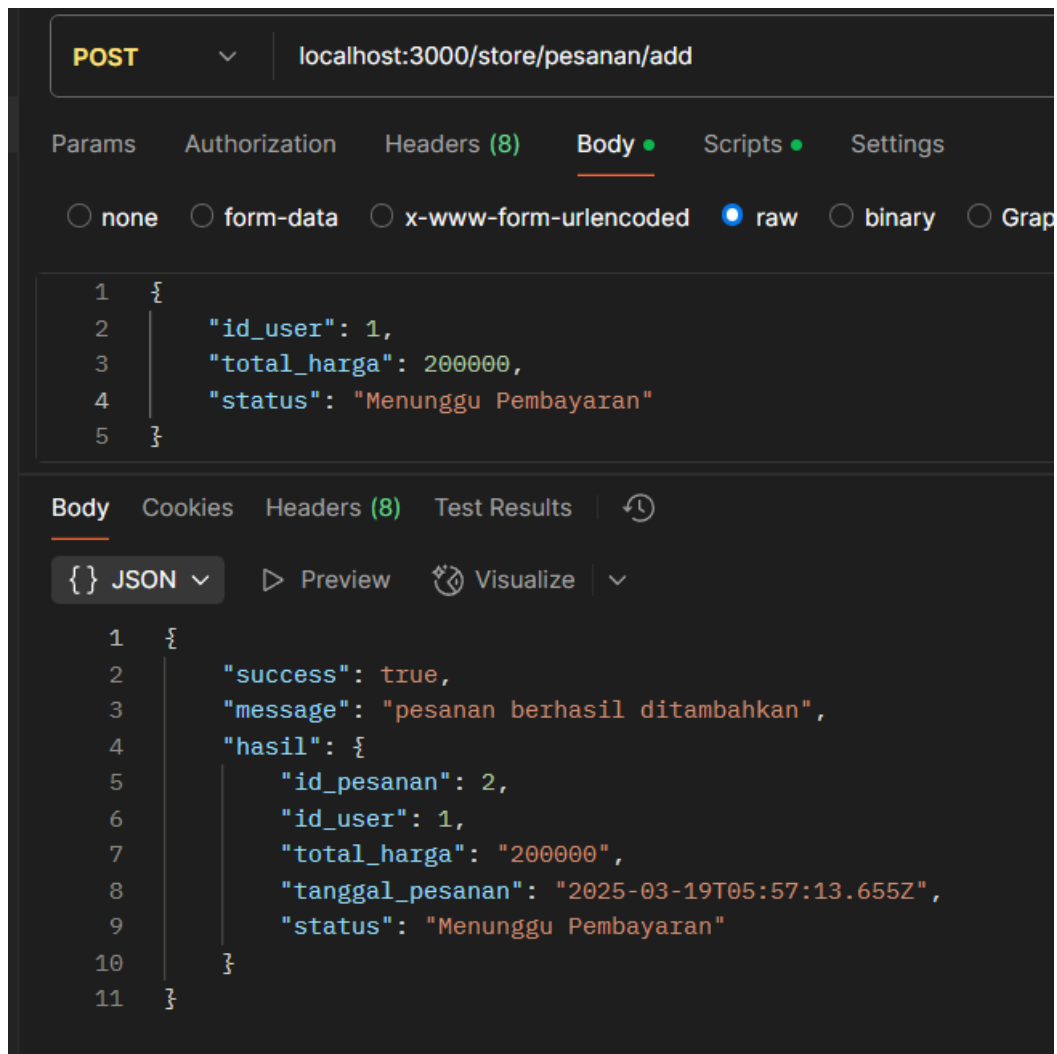
- **connect.js**: Adalah file yang menyambungkan seluruh file backend ke database online (NeonTech) dengan menggunakan metode Connection String.
- **.env**: Adalah file yang menyimpan Connection String yang digunakan pada connect.js, serta port yang bersangkutan.
- **index.js**: Adalah file kunci yang menggabungkan kinerja keseluruhan backend. File ini memanggil dependency dan middleware seperti express.js, body parser, cors, pg, dan dotenv. File ini juga melakukan setup pada routes.js sehingga route bisa digunakan saat mengetik URL di postman.
- **package-lock.json dan package.json**: package.json adalah file bawaan dari npm installer, yang menjadi configuration file untuk node.js. Sedangkan package-lock.json adalah file yang menyimpan lokasi beserta banyak informasi lainnya untuk semua dependency (biasanya di node_modules) agar nantinya dependency bisa digunakan dalam backend.
- **query.sql**: Adalah file yang berisi seluruh query sql yang digunakan dalam tugas ini.
- **SBD_Kelompok12_REST API.pdf**: File dokumentasi ini.

TESTING API

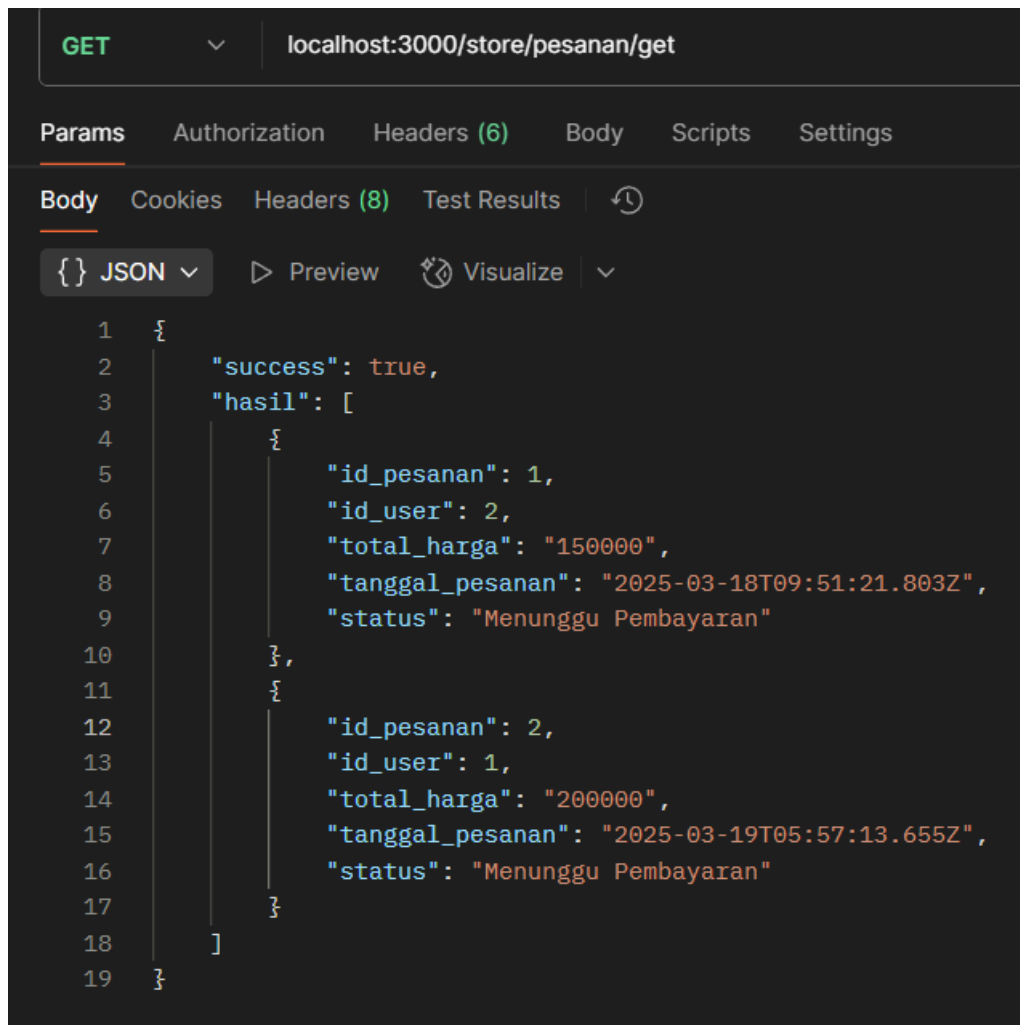
Untuk mengetes API, kita di sini menggunakan konsep CRUD, yaitu Create, Read, Update, dan Delete, yang dalam nodejs, kita melihat ini sebagai method POST, GET, PUT, dan DELETE.

Sebenarnya ada banyak sekali function-function CRUD yang digunakan dalam REST API ini. Namun untuk menjaga dokumentasi tetap sederhana, kita akan menunjukkan beberapa contoh saja yang merepresentasikan masing-masing CRUD.

- Create
Contoh : Membuat entry baru di table pesanan
Query SQL : INSERT INTO pesanan (id_user, total_harga, status)
VALUES (id, harga, status)
Method : POST
URL : localhost:3000/store/pesanan/add
Contoh output



- Read
Contoh : Membaca seluruh isi table pesanan.
Query SQL : `SELECT * FROM produk`
Method : GET
URL : `localhost:3000/store/pesanan/get`
Contoh output



```
GET localhost:3000/store/pesanan/get

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (8) Test Results ↺

{} JSON Preview Visualize

1  {
2      "success": true,
3      "hasil": [
4          {
5              "id_pesanan": 1,
6              "id_user": 2,
7              "total_harga": "150000",
8              "tanggal_pesanan": "2025-03-18T09:51:21.803Z",
9              "status": "Menunggu Pembayaran"
10         },
11         {
12             "id_pesanan": 2,
13             "id_user": 1,
14             "total_harga": "200000",
15             "tanggal_pesanan": "2025-03-19T05:57:13.655Z",
16             "status": "Menunggu Pembayaran"
17         }
18     ]
19 }
```

- Update
Contoh : Mengupdate salah satu entry produk
Query SQL : UPDATE produk SET nama_produk = 'namaproduk', kategori = 'kategori', harga = harga, stok = stok
Method : PUT
URL : localhost:3000/store/product/update/
Contoh output

PUT localhost:3000/store/product/update/

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```

1  {
2    "nama_produk": "sabun",
3    "kategori": "Perlengkapan Kamar Mandi",
4    "harga": 15000,
5    "stok": 200,
6    "deskripsi": "sabun adalah perlengkapan mandi",
7    "id_produk": 1
8  }

```

Body Cookies Headers (8) Test Results ↺

{ } JSON ▾ ▶ Preview ↻ Visualize ▾

```

1  {
2    "success": true,
3    "message": "produk berhasil diupdate",
4    "hasil": {
5      "id_produk": 1,
6      "nama_produk": "sabun",
7      "kategori": "Perlengkapan Kamar Mandi",
8      "harga": "15000",
9      "stok": 200,
10     "deskripsi": "sabun adalah perlengkapan mandi"
11   }
12 }

```

Bukti update:

Cek kolom ketiga dimana entry itu diupdate menjadi yang baru.

```

gatau=> select * from produk;
id_produk | nama_produk | kategori | harga | stok | deskripsi
-----+-----+-----+-----+-----+-----
2 | payung | Perlengkapan Dewasa | 6969969696 | 69 | Payung adalah sesuatu yang anak kecil tidak bo
leah tahu
3 | Tempe | Makanan | 5000 | 10 | tempe lebih enak
1 | sabun | Perlengkapan Kamar Mandi | 15000 | 200 | sabun adalah perlengkapan mandi
(3 rows)

```

- Delete

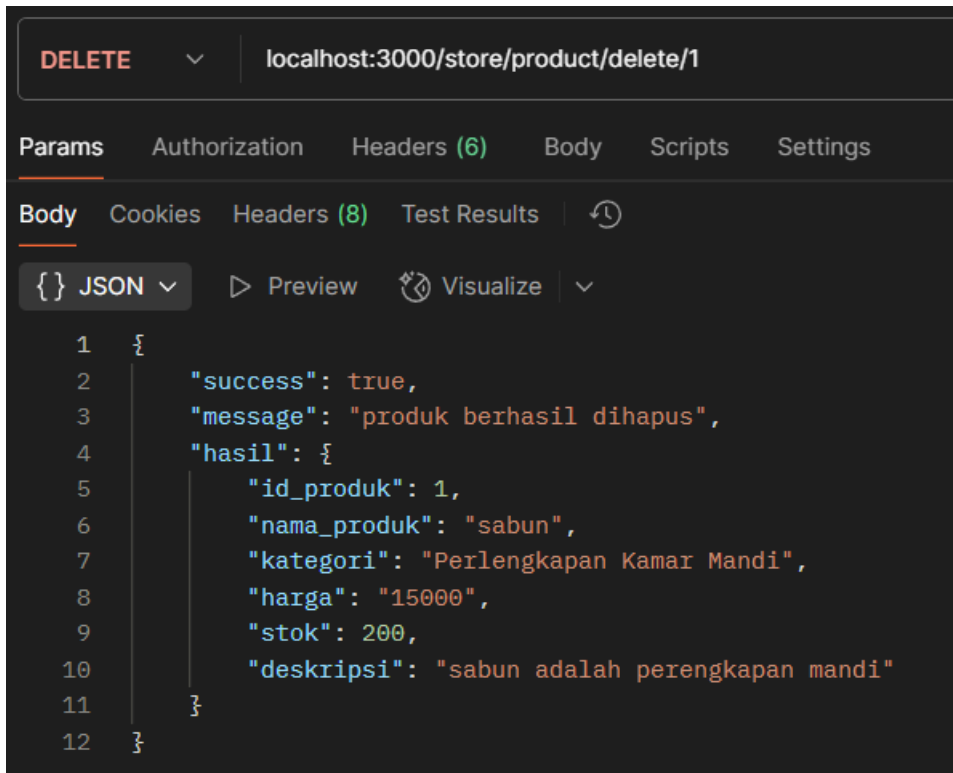
Contoh : Menghapus salah satu produk

Query SQL : DELETE FROM produk WHERE id_produk = id

Method : DELETE

URL : localhost:3000/store/product/delete/:id

Contoh output:



Bukti terhapus:

Perhatikan bahwa produk id 1, yaitu sabun (yang diupdate di bagian Update) telah terhapus.

```
gatau=> select * from produk;
id_produk | nama_produk | kategori | harga | stok | deskripsi
-----+-----+-----+-----+-----+-----
2 | payung | Perlengkapan Dewasa | 6969969696 | 69 | Payung adalah sesuatu yang anak kecil tidak boleh
3 | Tempe | Makanan | 5000 | 10 | tempe lebih enak
(2 rows)
```