

Aplikasi CRUD dengan React, Express, dan PostgreSQL

Aplikasi ini merupakan proyek latihan untuk mengimplementasikan fitur CRUD menggunakan **React (Vite)** di frontend, **Express.js** di backend, dan **PostgreSQL** sebagai database.

Developed by:

- Raisha Alika Irwandira (10231077)
 - Yosan Pratiwi (10231091)
-

📝 Fitur Aplikasi

- Menampilkan daftar produk dengan tampilan responsif
 - Menambahkan produk baru dengan validasi input
 - Mengedit produk yang sudah ada dengan modal form
 - Menghapus produk dengan konfirmasi sebelum eksekusi
 - Menerapkan notifikasi sukses/gagal pada operasi CRUD
 - Menghubungkan frontend dan backend menggunakan API
 - Menampilkan data dalam tabel dengan fitur pencarian dan pagination
 - Menggunakan state management dengan React Hooks (`useState, useEffect`)
-

📝 Cara Install dan Menjalankan Aplikasi

1. Clone Repository

```
git clone https://github.com/yosanpratiwi/ecommerce-kecil.git  
cd repository-name
```

2. Install Dependensi

Backend

```
cd backend  
npm install
```

Frontend

```
cd frontend  
npm install
```

3. Konfigurasi Database

1. Pastikan PostgreSQL sudah terinstall.
2. Buat database dengan nama `ecommerce_kecil`.
3. Sesuaikan konfigurasi di file `db.js`.
4. Jalankan migrasi database:

```
cd backend  
node migrate.js
```

4. Menjalankan Aplikasi

Menjalankan Backend

```
cd backend  
node index.js
```

Backend akan berjalan di `http://localhost:3001`.

Menjalankan Frontend

```
cd frontend  
npm run dev
```

Frontend akan berjalan di `http://localhost:5173`.

❖ Daftar Endpoint API

1. Endpoint Produk

Method	Endpoint	Deskripsi
GET	/api/produk	Mendapatkan semua data produk
GET	/api/produk/:id	Mendapatkan satu produk berdasarkan ID
POST	/api/produk	Menambahkan produk baru
PUT	/api/produk/:id	Mengupdate produk berdasarkan ID

Method	Endpoint	Deskripsi
DELETE	/api/produk/:id	Menghapus produk berdasarkan ID

2. Contoh Payload untuk POST dan PUT

```
{
  "nama": "Produk A",
  "harga": 10000
}
```

⌚ Teknologi yang Digunakan

- **Frontend:** React (Vite), Tailwind CSS, Axios, React Hooks
- **Backend:** Express.js, Node.js, CORS
- **Database:** PostgreSQL, pgAdmin
- **Tools:** Postman, GitHub, Railway, Vercel
- **State Management:** useState, useEffect, Context API
- **Deployment:** GitHub Actions, Vercel, Railway

📷 Screenshot Aplikasi

1. Tampilan Tambah Produk



Selamat Datang di Aplikasi E-Commerce Sederhana

Tambah Produk

Nama Produk:

Jeruk

Harga:

9000

Simpan

Daftar Produk

Nama Produk	Harga	Aksi
Bawang Merah	Rp10000	Edit Delete
Alpukat	Rp35000	Edit Delete
Anggur	Rp25000	Edit Delete

2. Tampilan Read Data dalam Tabel

The screenshot shows a web browser window with the URL `localhost:5173`. At the top right, there is a green notification bar that says "Produk berhasil ditambahkan!". Below it, the main content area displays a welcome message "Selamat Datang di Aplikasi E-Commerce Sederhana". Underneath, there is a "Tambah Produk" (Add Product) form with fields for "Nama Produk" (Product Name) and "Harga" (Price), and a "Simpan" (Save) button. To the right of the form is a "Daftar Produk" (Product List) table:

Nama Produk	Harga	Aksi
Bawang Merah	Rp10000	<button>Edit</button> <button>Delete</button>
Alpukat	Rp35000	<button>Edit</button> <button>Delete</button>
Anggur	Rp25000	<button>Edit</button> <button>Delete</button>

3. Tampilan Edit Produk

The screenshot shows the same web browser window. The "Edit Produk" (Edit Product) dialog is open over the "Tambah Produk" (Add Product) form. In the dialog, the "Nama Produk" field contains "Jeruk" and the "Harga" field contains "12000". The "Simpan" (Save) button is visible at the bottom of the dialog. The background "Daftar Produk" (Product List) table now includes a new row for "Jeruk" with a price of Rp9000.

Nama Produk	Harga	Aksi
Bawang Merah	Rp10000	<button>Edit</button> <button>Delete</button>
Alpukat	Rp35000	<button>Edit</button> <button>Delete</button>
Anggur	Rp25000	<button>Edit</button> <button>Delete</button>
Jeruk	Rp9000	<button>Edit</button> <button>Delete</button>

The screenshot shows a web application interface for an e-commerce platform. At the top, a green notification bar displays the message "Produk berhasil diperbarui!". Below it, a header says "Selamat Datang di Aplikasi E-Commerce Sederhana". A form titled "Tambah Produk" contains fields for "Nama Produk" and "Harga", with a "Simpan" button. To the right, a table titled "Daftar Produk" lists four items: Bawang Merah (Rp10000), Alpukat (Rp35000), Anggur (Rp25000), and Jeruk (Rp12000), each with "Edit" and "Delete" buttons.

4. Tampilan Hapus Produk

This screenshot shows the same application after a product has been deleted. The green notification bar now displays "Produk berhasil dihapus!". The "Daftar Produk" table shows only three items: Bawang Merah (Rp10000), Alpukat (Rp35000), and Jeruk (Rp12000), each with "Edit" and "Delete" buttons.

🔧 Troubleshooting

1. Pesan "Cannot GET /" pada localhost:3001



localhost:3001



Gmail



YouTube



Maps



Adobe Acrobat

Cannot GET /

- Pastikan server Node.js atau Express berjalan dengan benar.
- Jika menggunakan Express, tambahkan rute di server.js atau app.js:

```
app.get('/', (req, res) => {
  res.send('Hello, World!');
});
```

- Pastikan ada script untuk menjalankan server dengan benar.
- Jika sudah ada rute yang benar, coba restart server:

```
ctrl + c  
npm start
```

2. Nama dan harga tidak tersimpan dengan benar di database

POST | http://localhost:3001/api/produk

Params | Authorization | Headers (9) | **Body** | Pre-request Script | Tests

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

```

1 {
2   "nama": "Produk 0",
3   "harga": 50000
4 }
5

```

Body | Cookies | Headers (8) | Test Results | Status: 200 OK

Pretty | Raw | Preview | **JSON** |

```

1 {
2   "id": 55,
3   "nama": null,
4   "harga": null
5 }

```

- Jika menggunakan Express, pastikan middleware express.json() sudah ada di server.js atau app.js:

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

- Pastikan nama kolom di database cocok dengan yang dikirim dari frontend (nama, harga).
- Pastikan query INSERT ke database sudah benar. Contoh untuk PostgreSQL dengan pg:

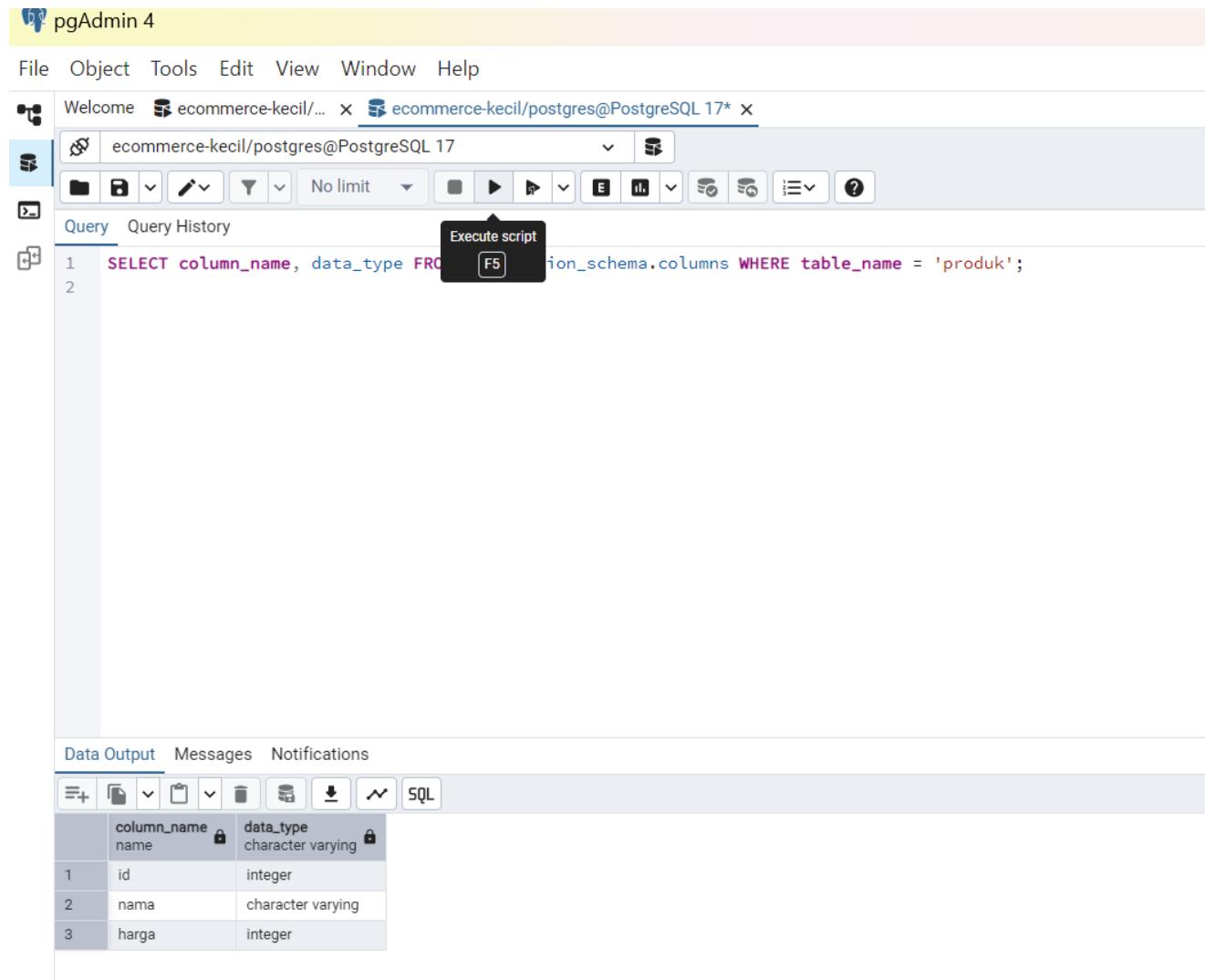
```
const insertQuery = 'INSERT INTO produk (nama, harga) VALUES ($1, $2) RETURNING *';
const values = [req.body.nama, req.body.harga];

const result = await pool.query(insertQuery, values);
res.json(result.rows[0]);
```

- Periksa apakah request dikirim dalam format JSON (di Postman pilih raw JSON).
- Tambahkan log untuk memastikan data diterima dengan benar di backend:

```
console.log(req.body);
```

3. Data nama dan harga tersimpan sebagai null



The screenshot shows the pgAdmin 4 interface. At the top, there's a toolbar with various icons. Below it is a menu bar with File, Object, Tools, Edit, View, Window, and Help. The main area has two tabs: Welcome and ecommerce-kecil/postgres@PostgreSQL 17*. The Welcome tab is currently active. In the Welcome tab, there's a sub-tab for ecommerce-kecil/postgres@PostgreSQL 17*. Below the tabs is a toolbar with icons for file operations like Open, Save, and Print. To the right of the toolbar is a dropdown menu set to 'No limit'. Further right are icons for zooming in and out, and a search bar. Below the toolbar is a status bar with 'Query' and 'Query History' tabs, and an 'Execute script' button with an F5 icon. The main query editor window contains the following SQL code:

```
1 SELECT column_name, data_type FROM information_schema.columns WHERE table_name = 'produk';
2
```

Below the query editor is a 'Data Output' tab, which is currently selected. It displays a table with three rows of data:

	column_name	data_type
1	id	integer
2	nama	character varying
3	harga	integer

- Pastikan Express bisa membaca request body JSON:

```
app.use(express.json());
```

- Pastikan query INSERT di backend benar dan menggunakan parameterized query:

```
const insertQuery = 'INSERT INTO produk (nama, harga) VALUES ($1, $2) RETURNING *';
const values = [req.body.nama, req.body.harga];
```

```
const result = await pool.query(insertQuery, values);
res.json(result.rows[0]);
```

- Kirim request dalam format JSON (di Postman: raw JSON).
- Cek apakah req.body berisi data yang benar:

```
console.log(req.body);
```

- Jalankan SQL berikut di pgAdmin untuk memastikan tidak ada constraint yang mencegah insert:

```
INSERT INTO produk (nama, harga) VALUES ('Produk Test', 50000);
```

- Setelah perbaikan, restart server Node.js:

4. Data tidak tersimpan dengan benar saat insert

The screenshot shows the pgAdmin 4 application window. The title bar says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". There are two tabs open in the main area: "ecommerce-kecil/..." and "ecommerce-kecil/postgres@Po". The second tab is selected. Below the tabs is a toolbar with icons for file operations like New, Open, Save, and a dropdown for "No limit". The main pane is titled "Query" and contains the following SQL code:

```
1 SELECT * FROM produk;
```

Data Output Messages Notifications

SQL

	id [PK] integer	nama character varying (50)	harga integer
2	39	Produk D	200000
3	41	Produk d	5000000
4	42	Produk E	300000
5	43	hrugeugtu	-1
6	47	hrugeugtu	-1
7	44	Produk C	230000
8	48	Produk F	99999
9	49	[null]	[null]
10	50	[null]	[null]
11	51	[null]	[null]
12	52	[null]	[null]
13	53	[null]	[null]
14	54	[null]	[null]
15	55	[null]	[null]

Total rows: 15 Query complete 00:00:00.056

- Pastikan express.json() Aktif di Backend

```
app.use(express.json());
```

- Gunakan console.log(req.body); untuk memastikan data yang dikirim ke backend sudah sesuai.
- Validasi Data Sebelum Insert Tambahkan validasi sebelum menyimpan data:

```
if (!req.body.nama || req.body.harga < 0) {
    return res.status(400).json({ error: "Data tidak valid" });
}
```

- Pastikan Query INSERT Menggunakan Parameter yang Benar

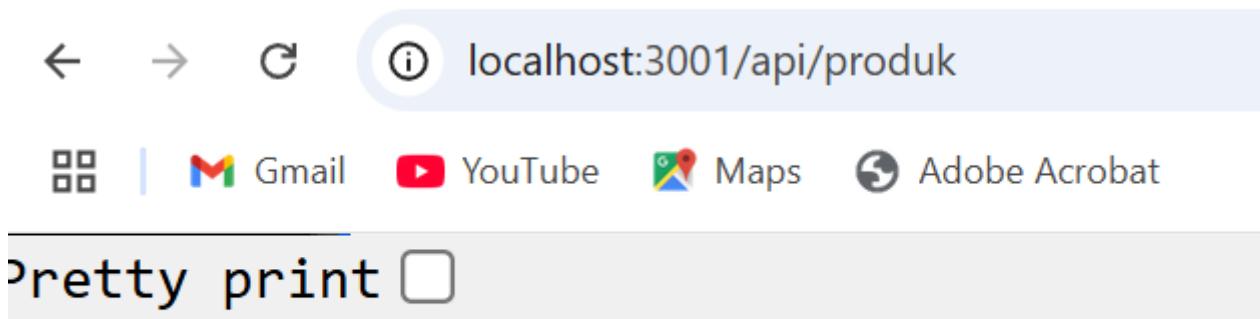
```
const insertQuery = 'INSERT INTO produk (nama, harga) VALUES ($1, $2) RETURNING *';
```

```
const values = [req.body.nama, req.body.harga];
const result = await pool.query(insertQuery, values);
res.json(result.rows[0]);
```

- Hapus Data Null untuk Membersihkan Database

```
DELETE FROM produk WHERE nama IS NULL;
```

5. Kesalahan di backend



- Tambahkan console.error(err); di bagian error handler:

```
app.use((err, req, res, next) => {
    console.error(err);
    res.status(500).json({ error: "Server error" });
});
```

- Pastikan Database PostgreSQL Terhubung
- Tes Koneksi ke Database dengan menambahkan ini sebelum query:

```
pool.query('SELECT 1', (err, res) => {
    if (err) {
        console.error("Database connection error", err);
    } else {
        console.log("Database connected!");
    }
});
```

- Pastikan tidak ada kesalahan pada query SQL (INSERT, SELECT, dll.).

6. Database PostgreSQL Tidak Terhubung

- Pastikan PostgreSQL sudah berjalan dan user `postgres` memiliki akses.
- Cek konfigurasi di `db.js`.
- Jalankan perintah berikut di PostgreSQL untuk memastikan database sudah ada:

```
SELECT * FROM produk;
```

7. Masalah Fetch Data di Frontend

- Pastikan backend berjalan dengan benar di `http://localhost:3001`.
- Gunakan `npm install axios` jika Axios belum terinstall.
- Pastikan CORS sudah diaktifkan di backend.

❖ Best Practices dalam Pengembangan

- **Gunakan Environment Variables:** Simpan kredensial database di `.env`.
- **Gunakan GitHub Issues & Projects:** Kelola tugas dengan jelas.
- **Gunakan Middleware di Express:** Untuk menangani error dan autentikasi.
- **Optimasi Query di PostgreSQL:** Gunakan indeks jika perlu untuk performa tinggi.
- **Pastikan UI Responsif:** Gunakan media queries dengan Tailwind CSS.

❖ Commit dan Push ke GitHub

```
git add .
git commit -m "Menambahkan dokumentasi dan fitur baru"
git push origin main
```

Inspirasi Hari Ini

"Mengetahui saja tidak cukup; kita harus menerapkannya. Berkeinginan saja tidak cukup; kita harus melakukannya." – Goethe