1. Struktur Endpoint Sinkronisasi Backend

Semua endpoint sinkronisasi di backend Anda sudah tersedia di:

- **Produk:**
  - POST /sync/products (kirim produk dari lokal ke server)
  - GET /sync/products?store_id=... (ambil produk dari server ke lokal)
- **Transaksi:**
  - POST /sync/transactions
  - GET /sync/transactions?store_id=...
- **User:**
  - POST /sync/users
  - GET /sync/users?store_id=...
- **Store:**
  - POST /sync/stores
  - GET /sync/stores?owner_id=...
- **Owner:**
  - POST /sync/owners
  - GET /sync/owners?id=...

2. Kode Sinkronisasi di Frontend Electron JS

a. Setup Database Lokal (SQLite)

```
// database.js

const Database = require('better-sqlite3');

const db = new Database('app_data.db');

module.exports = db;
```

b. Konfigurasi URL API Server

```
const API_BASE = 'http://103.126.116.119:8001/sync';
```

c. Fungsi Sinkronisasi Produk

```
const db = require('./database');
const fetch = require('node-fetch'); // atau global fetch jika sudah tersedia

// Kirim produk yang belum tersinkron ke server
async function syncProdukKeServer() {
 const unsynced = db.prepare('SELECT * FROM products WHERE is_synced = 0').all();
 if (!unsynced.length) return;
 const res = await fetch(`${API_BASE}/products`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(unsynced)
 });
 if (res.ok) {
  const stmt = db.prepare('UPDATE products SET is_synced = 1 WHERE id = ?');
  for (const p of unsynced) stmt.run(p.id);
 }
}

// Ambil produk terbaru dari server ke lokal
async function syncProdukDariServer(store_id) {
 const res = await fetch(`${API_BASE}/products?store_id=${store_id}`);
 const products = await res.json();
 db.prepare('DELETE FROM products WHERE store_id = ?').run(store_id);
 const insert = db.prepare(`
  INSERT OR REPLACE INTO products (id, store_id, name, sku, barcode, price,
cost_price, stock, category, description, image_url, is_active, created_at, updated_at,
jenis_diskon, nilai_diskon, diskon_bundle_min_qty, diskon_bundle_value, buy_qty,
free_qty, is_synced)
```

```
  VALUES (@id, @store_id, @name, @sku, @barcode, @price, @cost_price, @stock,
@category, @description, @image_url, @is_active, @created_at, @updated_at,
@jenis_diskon, @nilai_diskon, @diskon_bundle_min_qty, @diskon_bundle_value,
@buy_qty, @free_qty, 1)
  `);
 const insertMany = db.transaction((prods) => {
  for (const p of prods) insert.run(p);
 });
 insertMany(products);
}
```

d. Fungsi Sinkronisasi Transaksi

```
// Kirim transaksi yang belum tersinkron ke server
async function syncTransaksiKeServer() {
  const unsynced = db.prepare('SELECT * FROM transactions WHERE is_synced = 0').all();
  for (const trx of unsynced) {
    // Ambil item transaksi
    const items = db.prepare('SELECT * FROM transaction_items WHERE transaction_id = ?').all(trx.id);
    trx.items = items;
  }
  if (!unsynced.length) return;
  const res = await fetch(`${API_BASE}/transactions`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(unsynced)
  });
  if (res.ok) {
    const stmt = db.prepare('UPDATE transactions SET is_synced = 1 WHERE id = ?');
    for (const trx of unsynced) stmt.run(trx.id);
  }
}

// Ambil transaksi dari server ke lokal
async function syncTransaksiDariServer(store_id) {
  const res = await fetch(`${API_BASE}/transactions?store_id=${store_id}`);
  const transactions = await res.json();
  db.prepare('DELETE FROM transactions WHERE store_id = ?').run(store_id);
```

```
  db.prepare('DELETE FROM transaction_items WHERE transaction_id NOT IN (SELECT id FROM transactions)').run();

  const insertTrx = db.prepare(`

    INSERT OR REPLACE INTO transactions (id, store_id, user_id, total_cost, payment_type, payment_method, received_amount, change_amount, customer_name, customer_phone, payment_status, created_at, updated_at, tax, tax_percentage, role, is_owner, is_synced)

    VALUES (@id, @store_id, @user_id, @total_cost, @payment_type, @payment_method, @received_amount, @change_amount, @customer_name, @customer_phone, @payment_status, @created_at, @updated_at, @tax, @tax_percentage, @role, @is_owner, 1)

  `);

  const insertItem = db.prepare(`

    INSERT OR REPLACE INTO transaction_items (id, transaction_id, product_id, product_name, qty, price, subtotal)

    VALUES (@id, @transaction_id, @product_id, @product_name, @qty, @price, @subtotal)

  `);

  const insertMany = db.transaction((trxs) => {

    for (const trx of trxs) {

      insertTrx.run(trx);

      if (Array.isArray(trx.items)) {

        for (const item of trx.items) insertItem.run(item);

      }

    }

  });

  insertMany(transactions);

}
```

e. Fungsi Sinkronisasi User

```
async function syncUserKeServer() {
 const unsynced = db.prepare('SELECT * FROM users WHERE is_synced = 0').all();
 if (!unsynced.length) return;
 const res = await fetch(`${API_BASE}/users`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(unsynced)
 });
 if (res.ok) {
  const stmt = db.prepare('UPDATE users SET is_synced = 1 WHERE id = ?');
  for (const u of unsynced) stmt.run(u.id);
 }
}

async function syncUserDariServer(store_id) {
 const res = await fetch(`${API_BASE}/users?store_id=${store_id}`);
 const users = await res.json();
 db.prepare('DELETE FROM users WHERE store_id = ?').run(store_id);
 const insert = db.prepare(`
  INSERT OR REPLACE INTO users (id, owner_id, store_id, name, email, username,
password, role, is_active, created_at, is_synced)
  VALUES (@id, @owner_id, @store_id, @name, @email, @username, @password,
@role, @is_active, @created_at, 1)
 `);
 const insertMany = db.transaction((usrs) => {
  for (const u of usrs) insert.run(u);
 });
 insertMany(users);
```

```
}
```

f. Fungsi Sinkronisasi Store & Owner

```
async function syncStoreKeServer() {
  const unsynced = db.prepare('SELECT * FROM stores WHERE is_synced = 0').all();
  if (!unsynced.length) return;
  const res = await fetch(` ${API_BASE}/stores ` , {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(unsynced)
  });
  if (res.ok) {
    const stmt = db.prepare('UPDATE stores SET is_synced = 1 WHERE id = ?');
    for (const s of unsynced) stmt.run(s.id);
  }
}

async function syncStoreDariServer(owner_id) {
  const res = await fetch(` ${API_BASE}/stores?owner_id=${owner_id}` );
  const stores = await res.json();
  db.prepare('DELETE FROM stores WHERE owner_id = ?').run(owner_id);
  const insert = db.prepare(`
    INSERT OR REPLACE INTO stores (id, type, owner_id, name, business_name, address,
phone, receipt_template, created_at, updated_at, tax_percentage, is_synced)
    VALUES (@id, @type, @owner_id, @name, @business_name, @address, @phone,
@receipt_template, @created_at, @updated_at, @tax_percentage, 1)
  ` );
  const insertMany = db.transaction((sts) => {
    for (const s of sts) insert.run(s);
  });
  insertMany(stores);
```

```
}

async function syncOwnerKeServer() {
  const unsynced = db.prepare('SELECT * FROM owners WHERE is_synced = 0').all();
  if (!unsynced.length) return;
  const res = await fetch(` ${API_BASE}/owners ` , {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(unsynced)
  });
  if (res.ok) {
    const stmt = db.prepare('UPDATE owners SET is_synced = 1 WHERE id = ?');
    for (const o of unsynced) stmt.run(o.id);
  }
}

async function syncOwnerDariServer(id) {
  const res = await fetch(` ${API_BASE}/owners?id=${id}` );
  const owners = await res.json();
  db.prepare('DELETE FROM owners WHERE id = ?').run(id);
  const insert = db.prepare(`
    INSERT OR REPLACE INTO owners (id, business_name, email, phone, password,
package_id, package_expired_at, created_at, address, is_synced)
    VALUES (@id, @business_name, @email, @phone, @password, @package_id,
@package_expired_at, @created_at, @address, 1)
  ` );
  const insertMany = db.transaction((os) => {
    for (const o of os) insert.run(o);
  });
```

```
  insertMany(owners);
}
```

g. Fungsi Sinkronisasi Utama

```
async function syncAllData() {
  const store_id = localStorage.getItem('store_id');
  const owner_id = localStorage.getItem('owner_id');
  const ownerId = owner_id || 1; // fallback

  await syncOwnerKeServer();
  await syncOwnerDariServer(ownerId);

  await syncStoreKeServer();
  await syncStoreDariServer(ownerId);

  await syncUserKeServer();
  await syncUserDariServer(store_id);

  await syncProdukKeServer();
  await syncProdukDariServer(store_id);

  await syncTransaksiKeServer();
  await syncTransaksiDariServer(store_id);

  // Tambahkan entitas lain jika ada
}
```

3. Penyesuaian di Frontend Electron JS

- **Panggil syncAllData()** saat aplikasi start, login, atau saat user klik tombol "Sync".
- **Pastikan field is_synced** ada di semua tabel lokal (SQLite) untuk menandai data yang sudah tersinkron.
- **Gunakan endpoint**:
    - Produk: ${API_BASE}/products
    - Transaksi: ${API_BASE}/transactions
    - User: ${API_BASE}/users
    - Store: ${API_BASE}/stores
    - Owner: ${API_BASE}/owners
- **Gunakan fetch** dengan method POST (untuk kirim data) dan GET (untuk ambil data).
- **Pastikan struktur data** yang dikirim/diterima sesuai dengan skema backend.

---

4. Tips & Catatan

- **Jalankan sinkronisasi secara periodik** (misal setiap 5 menit) atau manual (tombol sync).
- **Tangani error** pada proses fetch agar aplikasi tetap stabil saat offline.
- **Pastikan field yang wajib** (misal store_id, owner_id) selalu terisi.
- **Gunakan transaction** pada SQLite untuk batch insert/update agar lebih cepat dan aman.
- **Jika ada perubahan skema**, update juga skema di SQLite (offline_db.sql).

5. Contoh Penggunaan di Frontend

```
// Di main process atau preload Electron
const { syncAllData } = require('./sync'); // file berisi semua fungsi di atas

// Panggil saat aplikasi start atau user klik tombol sync
syncAllData().then(() => {
  console.log('Sinkronisasi selesai!');
}).catch(err => {
  console.error('Gagal sinkronisasi:', err);
});
```

6. Dokumentasi Endpoint Backend

Endpoint	Method	Keterangan
/sync/products	POST	Kirim produk dari lokal ke server
/sync/products?store_id=	GET	Ambil produk dari server ke lokal
/sync/transactions	POST	Kirim transaksi dari lokal ke server
/sync/transactions?store_id=	GET	Ambil transaksi dari server ke lokal
/sync/users	POST	Kirim user dari lokal ke server
/sync/users?store_id=	GET	Ambil user dari server ke lokal
/sync/stores	POST	Kirim store dari lokal ke server
/sync/stores?owner_id=	GET	Ambil store dari server ke lokal
/sync/owners	POST	Kirim owner dari lokal ke server
/sync/owners?id=	GET	Ambil owner dari server ke lokal

---

7. Kesimpulan

- **Frontend Electron JS**:
    - Simpan data di SQLite.
    - Sinkronkan data ke backend menggunakan endpoint /sync/* di server.
    - Gunakan kode sinkronisasi di atas untuk semua entitas utama.
- **Backend Node.js**:
    - Sudah siap menerima dan mengirim data melalui endpoint /sync/*.