

**DOKUMENTASI UJI KOMPETENSI KEAHLIAN  
APLIKASI SIPALO (SISTEM PEMINJAMAN ALAT OLAHRAGA)**



**Disusun oleh:**  
FINELYA PUTRI ARDIANATASYA / 19  
XII RPL 2

**PROGRAM KEAHLIAN PENGEMBANGAN PERANGKAT LUNAK DAN GIM  
KOMPETENSI KEAHLIAN REKAYASA PERANGKAT LUNAK  
SMK BRANTAS KARANGKATES  
2026**

a. Analisis Kebutuhan

1. Admin: Admin adalah pengguna dengan hak akses tertinggi dalam sistem. Admin bertanggung jawab atas pengelolaan data master dan pengaturan sistem.

**Tanggung jawab Admin:**

- Login dan Logout ke sistem
- Mengelola data user
- Mengelola data alat
- Mengelola kategori alat
- Mengelola data peminjaman
- Mengelola data pengembalian
- Melihat log aktivitas
- Mengelola laporan

2. Petugas: Petugas adalah pengguna yang menangani operasional peminjaman dan pengembalian alat di lapangan.

**Tanggung jawab Petugas:**

- Login dan Logout ke sistem
- Menyetujui pengajuan peminjaman
- Menolak pengajuan peminjaman
- Memproses pengembalian alat
- Memeriksa kondisi alat
- Menghitung denda keterlambatan
- Mencetak laporan

3. Peminjam: Peminjam adalah pengguna seperti siswa, guru, atau staf yang meminjam alat.

**Tanggung jawab Peminjam:**

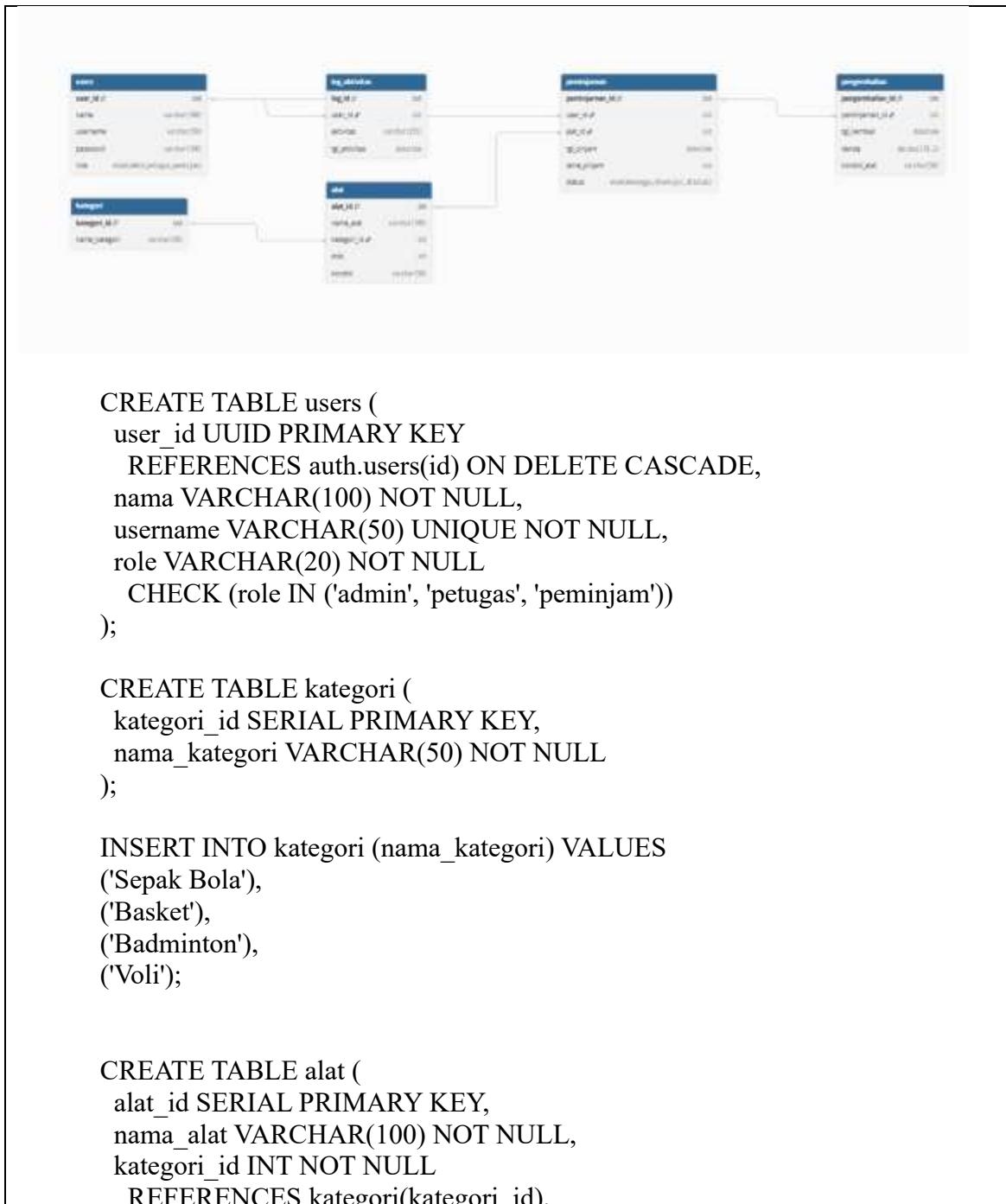
- Login dan Logout ke sistem
- Melihat daftar alat
- Mengajukan peminjaman
- Melihat status peminjaman
- Melakukan pengembalian alat

4. Kebutuhan fungsional:

1. Sistem melakukan autentikasi pengguna (login/logout) dan pengelolaan hak akses berdasarkan role dengan data yang tersimpan di database.
2. Mengelola data master (user, alat, kategori) melalui operasi tambah, ubah, hapus, dan tampil yang tersimpan di database.
3. Menampilkan daftar alat, detail alat, stok, dan kondisi alat kepada pengguna.
4. Sistem memfasilitasi proses pengajuan peminjaman alat oleh peminjam melalui form peminjaman.
5. Sistem memproses persetujuan atau penolakan pengajuan peminjaman oleh petugas.
6. Memproses pengembalian alat termasuk pencatatan tanggal kembali dan kondisi alat.
7. Sistem menghitung denda keterlambatan pengembalian sesuai aturan yang ditetapkan.

8. Sistem menyediakan pembuatan dan pencetakan laporan serta pencatatan log aktivitas.
5. Kebutuhan Non-Fungsional
    1. Tampilan mudah digunakan
    2. Navigasi jelas
    3. Form mudah dipahami
    4. Validasi input berjalan baik
    5. Hak akses berbasis role
    6. Stok alat selalu terupdate

b. ERD



```

CREATE TABLE users (
    user_id UUID PRIMARY KEY
    REFERENCES auth.users(id) ON DELETE CASCADE,
    nama VARCHAR(100) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    role VARCHAR(20) NOT NULL
    CHECK (role IN ('admin', 'petugas', 'peminjam'))
);
  
```

```

CREATE TABLE kategori (
    kategori_id SERIAL PRIMARY KEY,
    nama_kategori VARCHAR(50) NOT NULL
);
  
```

```

INSERT INTO kategori (nama_kategori) VALUES
('Sepak Bola'),
('Basket'),
('Badminton'),
('Voli');
  
```

```

CREATE TABLE alat (
    alat_id SERIAL PRIMARY KEY,
    nama_alat VARCHAR(100) NOT NULL,
    kategori_id INT NOT NULL
    REFERENCES kategori(kategori_id),
);
  
```

```
        stok INT NOT NULL,
        kondisi VARCHAR(50)
    );

INSERT INTO alat (nama_alat, kategori_id, stok, kondisi) VALUES
('Bola Sepak', 1, 5, 'Baik'),
('Bola Basket', 2, 4, 'Baik'),
('Raket Badminton', 3, 10, 'Baik'),
('Bola Voli', 4, 2, 'Baik');
```

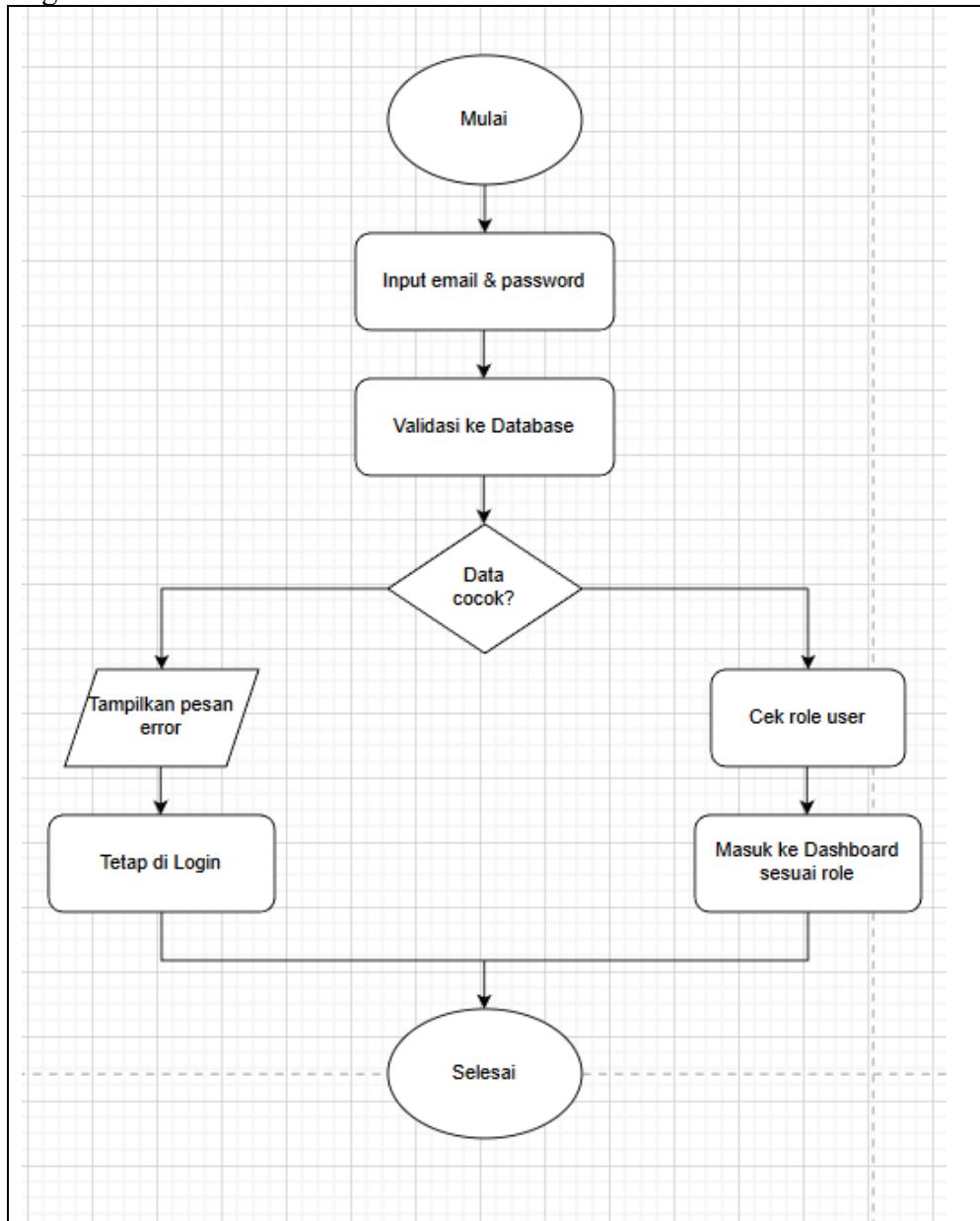
```
CREATE TABLE peminjaman (
    peminjaman_id SERIAL PRIMARY KEY,
    user_id UUID NOT NULL
        REFERENCES users(user_id),
    alat_id INT NOT NULL
        REFERENCES alat(alat_id),
    tgl_pinjam TIMESTAMP DEFAULT NOW(),
    lama_pinjam INT NOT NULL,
    status VARCHAR(20)
);
```

```
CREATE TABLE pengembalian (
    pengembalian_id SERIAL PRIMARY KEY,
    peminjaman_id INT NOT NULL
        REFERENCES peminjaman(peminjaman_id),
    tgl_kembali TIMESTAMP,
    denda NUMERIC,
    kondisi_alat VARCHAR(50)
);
```

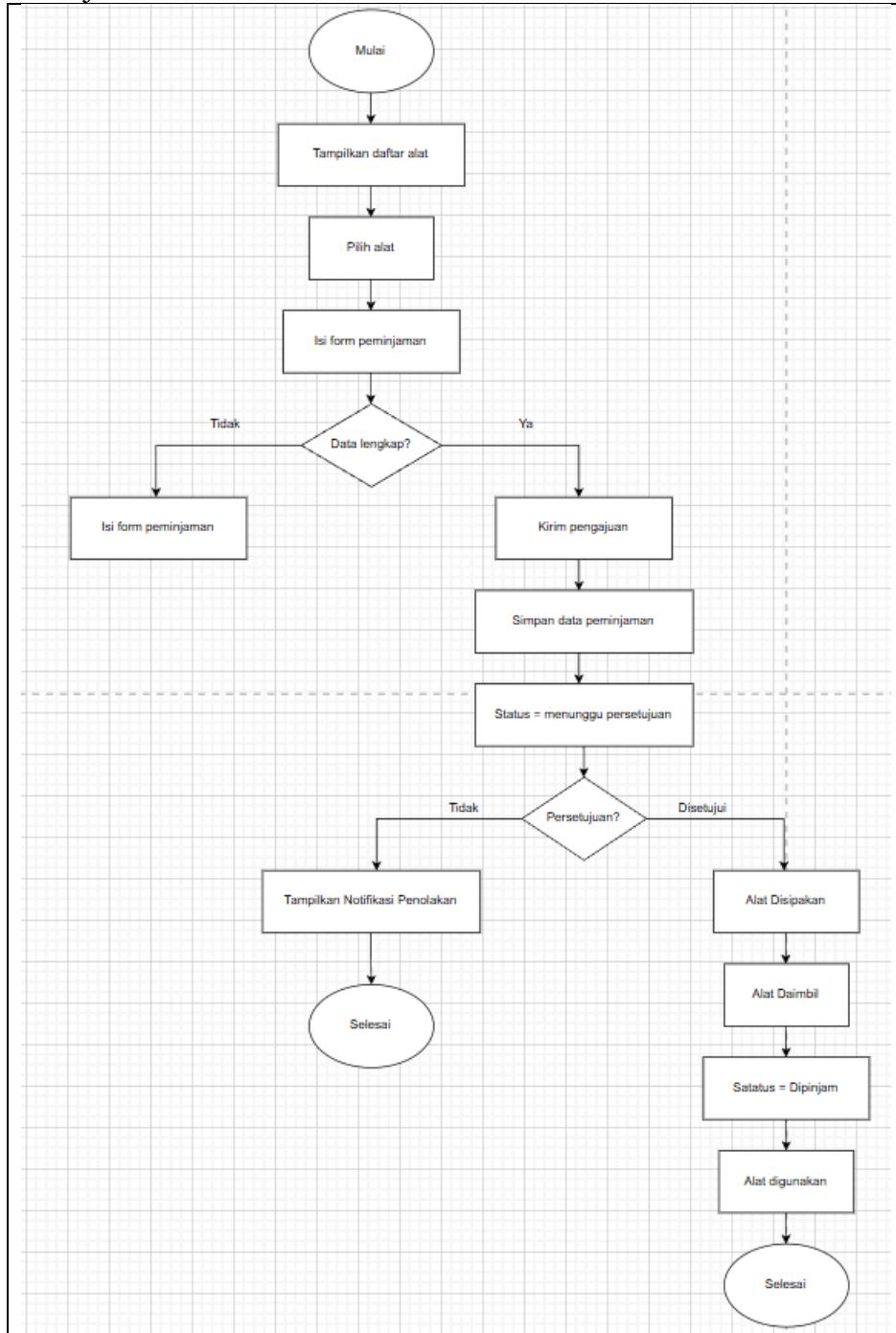
```
CREATE TABLE log_aktivitas (
    log_id SERIAL PRIMARY KEY,
    user_id UUID NOT NULL
        REFERENCES users(user_id),
    aktivitas VARCHAR(100),
    tgl_aktivitas TIMESTAMP DEFAULT NOW()
);
```

c. Flowchart

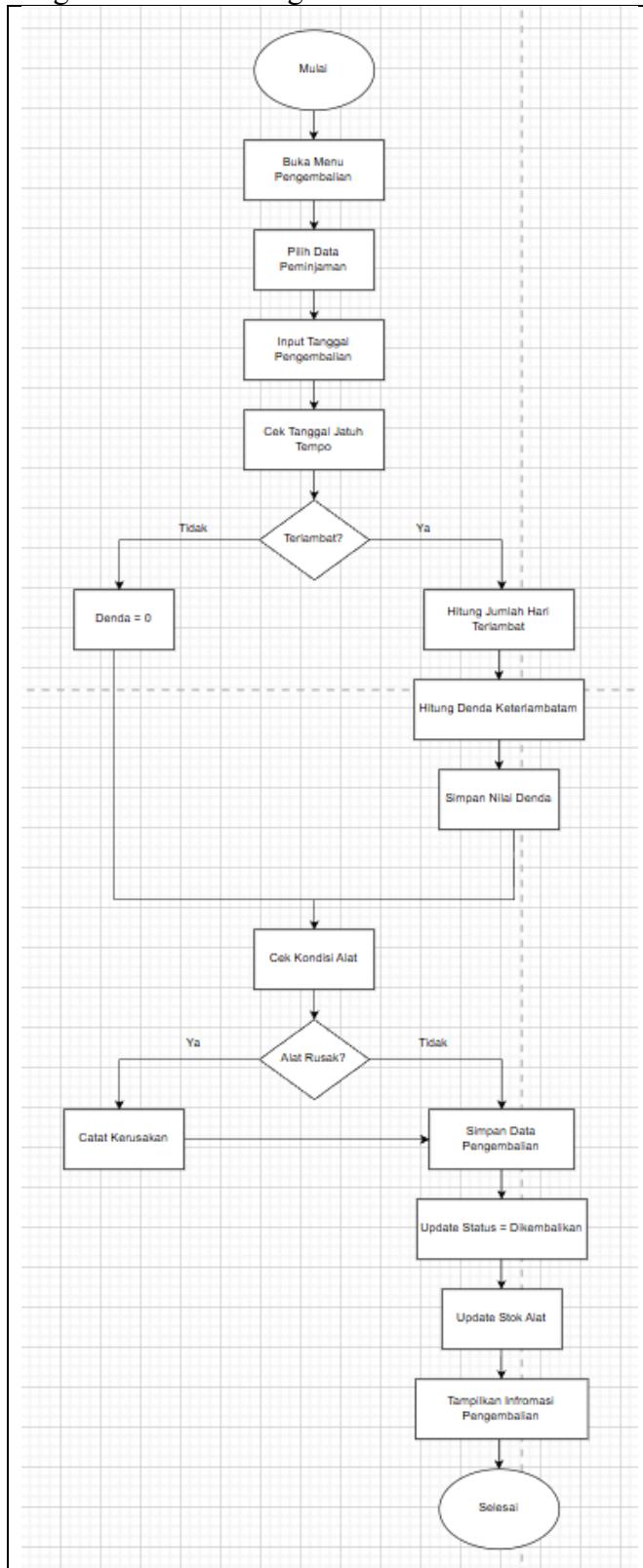
1. Login



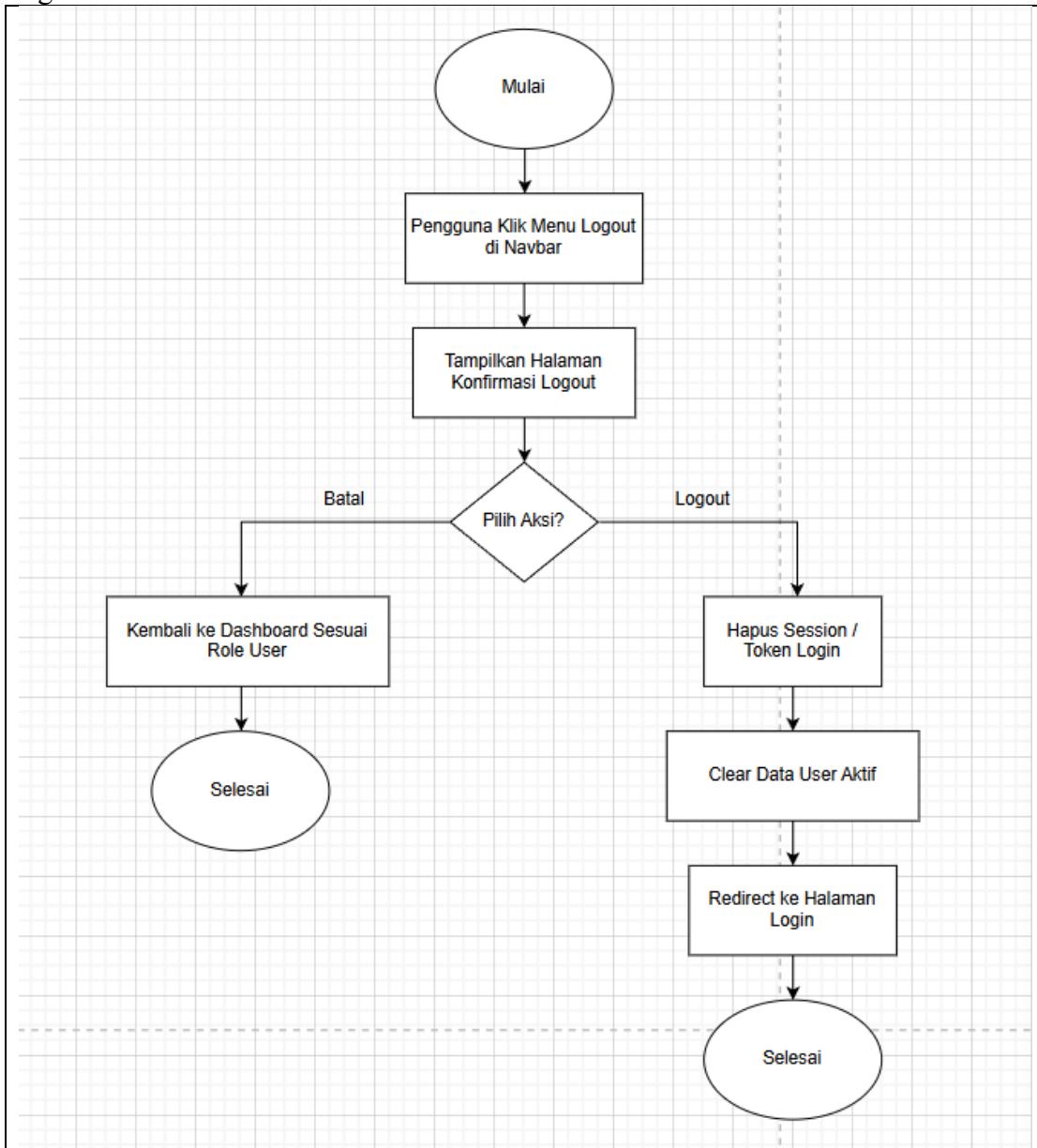
## 2. Peminjaman



### 3. Pengembalian + Hitung Denda



#### 4. Logout



- d. Dokumentasi fungsi / prosedur  
 1. Function & Trigger Peminjaman

Function:

```

CREATE OR REPLACE FUNCTION proses_peminjaman_alat()
RETURNS TRIGGER AS $$
DECLARE
  stok_sekarang INT;
BEGIN
  -- Ambil stok terakhir
  SELECT stok INTO stok_sekarang FROM alat WHERE alat_id =
  NEW.alat_id;

  -- Cek ketersediaan
  IF stok_sekarang <= 0 THEN
    RAISE EXCEPTION 'Stok alat sudah habis!';
  END IF;
END;
  
```

```

END IF;

-- Kurangi stok
UPDATE alat SET stok = stok - 1 WHERE alat_id = NEW.alat_id;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

Trigger:

```

CREATE TRIGGER trg_peminjaman_otomatis
BEFORE INSERT ON peminjaman
FOR EACH ROW
EXECUTE FUNCTION proses_peminjaman_alat();

```

## 2. Function & Trigger Pengembalian

Function:

```

CREATE OR REPLACE FUNCTION proses_pengembalian_alat()
RETURNS TRIGGER AS $$

BEGIN
    -- Tambah stok kembali
    UPDATE alat
    SET stok = stok + 1
        WHERE alat_id = (SELECT alat_id FROM peminjaman WHERE
peminjaman_id = NEW.peminjaman_id);

    -- Update status jadi kembali
    UPDATE peminjaman
    SET status = 'kembali'
        WHERE peminjaman_id = NEW.peminjaman_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

Trigger:

```

CREATE TRIGGER trg_pengembalian_otomatis
AFTER INSERT ON pengembalian
FOR EACH ROW
EXECUTE FUNCTION proses_pengembalian_alat();

```

## 3. Function & Trigger Log\_aktivitas

Function:

```

CREATE OR REPLACE FUNCTION catat_ke_log()
RETURNS TRIGGER AS $$

BEGIN
    INSERT INTO log_aktivitas (user_id, aktivitas, tgl_aktivitas)
    VALUES (NEW.user_id, 'Melakukan transaksi baru di sistem', NOW());

```

```
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

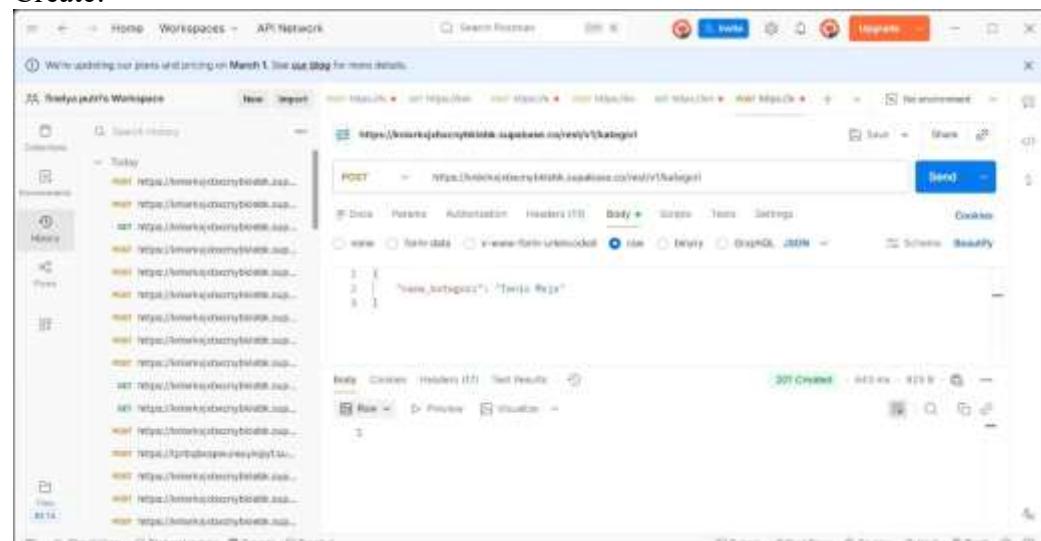
Trigger:

```
CREATE TRIGGER trg_buat_log
AFTER INSERT ON peminjaman
FOR EACH ROW
EXECUTE FUNCTION catat_ke_log();
```

#### e. Test Postman

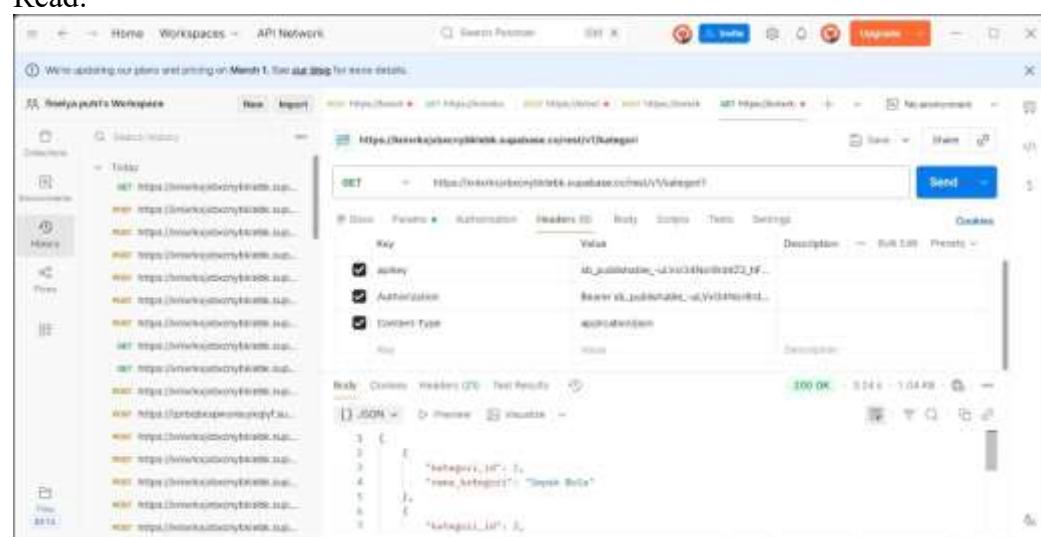
##### 1. Kategori

Create:



The screenshot shows a Postman interface with a POST request to `https://localhost:44386/kategori/api/v1/kategori`. The request body is a JSON object with the key `"name_kategori"` and the value `"Televisi Baru"`. The response status is `201 Created`.

Read:



The screenshot shows a GET request to `https://localhost:44386/kategori/api/v1/kategori`. It includes an `Authorization` header with a Bearer token and a Content-Type header set to `application/json`. The response status is `200 OK`, and the response body is a JSON array containing one category object with the name `Televisi Baru`.

## Update:

The screenshot shows the Postman application interface. The left sidebar displays a workspace named "Praveya's Workspace" with a list of recent requests. The main panel shows a PATCH request to the URL `https://kenneljobscrbeek.database.cosmoseastv1.catelog`. The Headers tab contains the key `Authorization` with the value `Bearer d1234567890123456789012345678901`. The Body tab contains the JSON payload `{"name": "basic_beefegic1", "status": "In Progress"}`. The response status is 204 No Content.

## Delete:

The screenshot shows the Postman application interface. The left sidebar displays a workspace named "Praveya's Workspace" with a list of recent requests. The main panel shows a DELETE request to the URL `https://kenneljobscrbeek.database.cosmoseastv1.catelog/_id=1`. The Headers tab contains the keys `Authorization` (with value `Bearer d1234567890123456789012345678901`) and `Content-Type` (with value `application/json`). The response status is 204 No Content.

## 2. Alat

### Create:

The screenshot shows the Postman application interface. A POST request is being made to `https://www.firebaseio.com/.json`. The request body contains the following JSON data:

```
[{"id": 1, "name": "Baker", "category": "Food", "value": 5, "brand": "Baker"}]
```

The response status is `201 Created`.

### Read:

The screenshot shows the Postman application interface. A GET request is being made to `https://www.firebaseio.com/.json` with the following query parameters:

- orderBy: `value`
- equalTo: `1`
- auth: `anonymouse`

The response status is `200 OK` and the body contains the following JSON data:

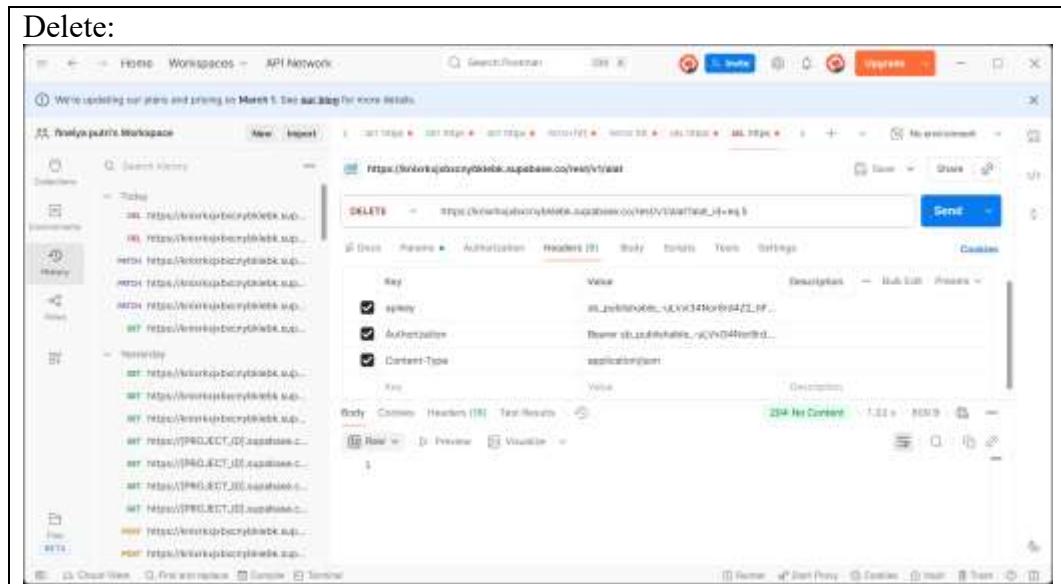
```
[{"id": 1, "name": "Baker", "category": "Food", "value": 5, "brand": "Baker"}]
```

### Update:

The screenshot shows the Postman application interface. A PATCH request is being made to `https://www.firebaseio.com/.json` with the following JSON body:

```
[{"id": 1, "name": "Baker", "category": "Food", "value": 5, "brand": "Baker"}]
```

The response status is `204 No Content`.



#### f. Inner Join

## Inner Join supabase:

```
INSERT INTO participates (user_id, stat_id, ts) VALUES (1, 1, '2020-02-06 22:28:04')
```

The screenshot shows the Supabase SQL Editor interface. The left sidebar displays a tree view of the database schema, including tables like 'participates', 'users', and 'stats'. The main area shows an SQL query being run against the 'participates' table. The results pane below shows two rows of data inserted into the table.

user_id	stat_id	ts
1	1	2020-02-06 22:28:04
1	2	2020-02-06 22:28:04

### Inner Join Postman:

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections, environments, history, and other options. The main area shows a request to `https://kierkojoberry@klebek.supabase.co/rest/v1/pemijaman` using a GET method. The response status is 200 OK, with a size of 1.74 kB. The response body is displayed as JSON, showing two records from the database:

```
[{"id": 1, "pemijaman_id": 2, "tgj_pnjmaw": "2024-02-06T22:26:00", "status": "Kembal", "users": [{"name": ""}], "alat": [{"name_alat": "Bola Sepak"}]}, {"id": 2, "pemijaman_id": 3, "tgj_pnjmaw": "2024-02-06T22:39:27", "status": "Kembal", "users": [{"name": ""}]}]
```

### g. Left Join

#### Left Join Supabase:

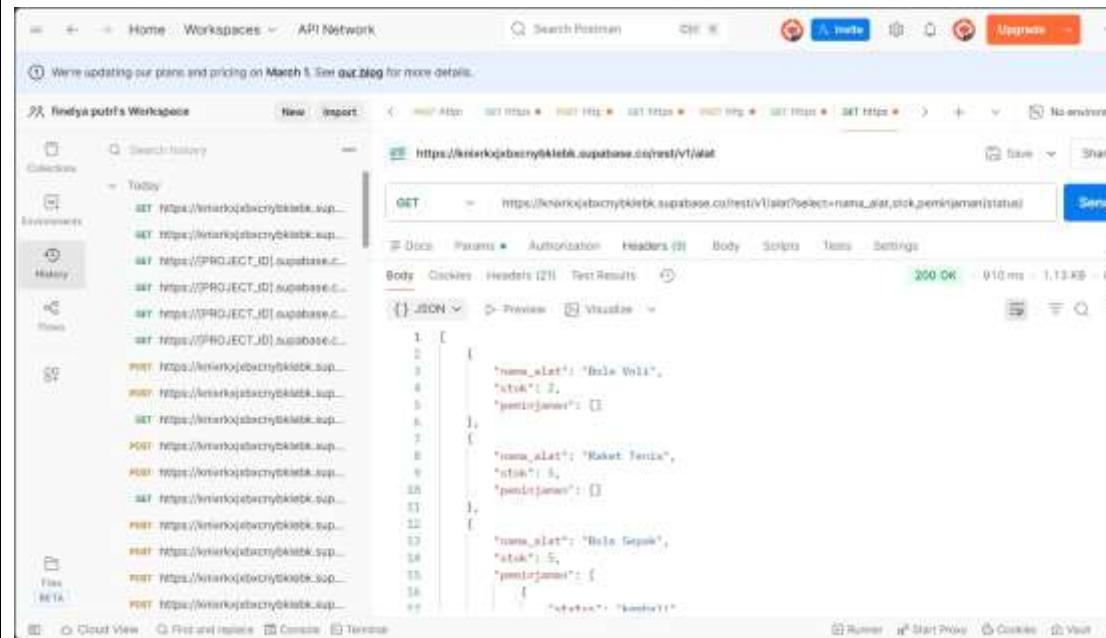
The screenshot shows the Supabase SQL Editor interface. On the left, there's a sidebar with various database objects like triggers, functions, and tables. The main area shows a SQL query being run against the `tbl_pemijaman` table:

```
SELECT * FROM tbl_pemijaman LEFT JOIN users ON users.id = pemijaman.user_id;
```

The results table shows the joined data:

user_id	pemijaman_id	status
1	1	Kembal
2	2	Kembal
3	3	Kembal
4	4	Kembal
5	5	Kembal

## Left Join Postman:

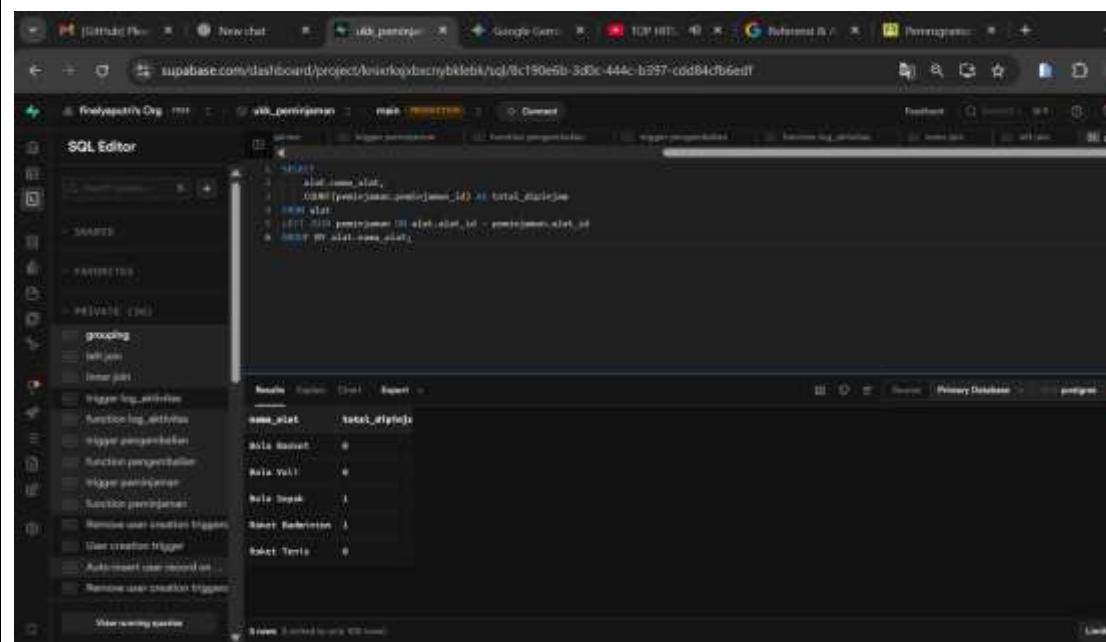


The screenshot shows the Postman interface with a successful API call. The URL is `https://knarkojoberrykibek.supabase.co/test/v1/supabase/test/lat`. The response body is a JSON array with three objects:

```
[{"name_aLat": "Bila Volti", "stok": 2, "peminjaman": []}, {"name_aLat": "Raket Tennis", "stok": 5, "peminjaman": []}, {"name_aLat": "Bola Sepak", "stok": 5, "peminjaman": [{"status": "Kosong"}]}
```

## h. Grouping

### Grouping supabase:



The screenshot shows the Supabase dashboard with the SQL Editor open. The query being run is:

```
SELECT name_aLat, COUNT(peminjaman.id) AS total_peminjaman, total_stok
FROM stat
LEFT JOIN peminjaman ON stat.id = peminjaman.stat_id
GROUP BY name_aLat;
```

The results table shows the grouped data:

name_aLat	total_peminjaman	total_stok
Bila Volti	0	2
Raket Tennis	0	5
Bola Sepak	1	5
Bola Badminton	0	0
Kotak Teman	0	0

## Grouping Postman:

The screenshot shows the Postman application interface. At the top, there are navigation links for Home, Workspaces, API Network, and a search bar labeled 'Search Postman'. Below the header, a message indicates they are updating plans and pricing on March 5, with a link to their blog.

The main area displays a collection titled 'Findya's Workspaces'. On the left sidebar, there are sections for Collections, Environments, History, Posts, and a pinned item 'SF'. The 'History' section lists several recent API requests. The 'Environments' section shows multiple environments, each with a dropdown menu. The 'Body' tab of the selected environment is expanded, showing a JSON response structure:

```
[{"name_stat": "Bola basket", "total_dipinjam": 8}, {"name_stat": "Bola Voli", "total_dipinjam": 6}, {"name_stat": "Bola Sepak", "total_dipinjam": 1}, {"name_stat": "Raket Badminton", "total_dipinjam": 1}]
```