|  |  |
| --- | --- |
|  | UNIVERSITAS DIPONEGORO – FAKULTAS TEKNIK  DEPARTEMEN TEKNIK ELEKTRO  *Jl. Prof. H. Soedarto, SH, Tembalang, Semarang 50275*  *Telp/Faks. (024)-7460057 e-mail: departemen@elektro.undip.ac.id* |

**Dokumen Pengembangan Produk**

Lembar Sampul Dokumen

|  |  |  |
| --- | --- | --- |
| Judul Dokumen | **TUGAS AKHIR:**  **Rancang Bangun Sistem Keamanan Kunci Pintu GedungBerbasis *Internet of Things*** | |
|  |  | |
| Jenis Dokumen | **IMPLEMENTASI** | |
|  | Catatan: Dokumen ini dikendalikan penyebarannya oleh Dept. Teknik Elektro Undip | |
| Nomor Dokumen | **B400-01-TA2223.2.19012** | |
|  |  | |
| Nomor Revisi | **01** | |
|  |  | |
| Nama File | **B400-2-TA2223** | |
|  |  | |
| Tanggal Penerbitan | **6 Februari 2023** | |
|  |  | |
| Unit Penerbit | **Departemen Teknik Elektro Undip** | |
|  |  | |
| Jumlah Halaman | **15** | (termasuk lembar sampul ini) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Pengusul | | | | |
| Pengusul | Nama  NIM | **Henric Dhiki Wicaksono**  21060119120011 | Jabatan  Tanda Tangan | Anggota |
|  | Nama  NIM | **Novi Dianasari**  21060119120039 | Jabatan  Tanda Tangan | Anggota |
|  | Nama  NIM | **Muhammad Khoiril Wafi**  21060119140133 | Jabatan  Tanda Tangan | Anggota |
| Pembimbing | Nama | **M. Arfan, S.Kom., M.Eng.** | Tanda Tangan |  |
| Utama | NIP | 198408172015041002 |  |  |
| Pendamping | Nama | **Imam Santoso, S.T., M.T.** | Tanda Tangan |  |
|  | NIP | 197012031997021001 |  |  |

**DAFTAR ISI**

[Catatan Sejarah Perbaikan Dokumen 3](#_Toc143680443)

[1. PENDAHULUAN 4](#_Toc143680444)

[**1.1** **Ringkasan Isi Dokumen** 4](#_Toc143680445)

[**1.2** **Aplikasi Dokumen** 4](#_Toc143680446)

[**1.3** **Referensi** 4](#_Toc143680447)

[**1.4** **Daftar Singkatan** 4](#_Toc143680448)

[2. IMPLEMENTASI 6](#_Toc143680449)

[2.1 Perangkat Keras Kunci Pintu 6](#_Toc143680450)

[2.2 Perangkat Lunak Kunci Pintu 10](#_Toc143680451)

[2.3 Database 22](#_Toc143680452)

[2.4 Backend API 27](#_Toc143680453)

[2.5 Server 36](#_Toc143680454)

[2.6 Tampilan Website 38](#_Toc143680455)

[2.7 Aplikasi Mobile 39](#_Toc143680456)

[3. PENUTUP 39](#_Toc143680457)

# Catatan Sejarah Perbaikan Dokumen

|  |  |
| --- | --- |
| **VERSI, TGL, OLEH** | **PERBAIKAN** |
| 01,  7 April 2023,  oleh Henric Dhiki Wicaksono, Novi Dianasari  dan Muhammad Khoiril Wafi. | *Draft* Dokumen B200 |

1. PENDAHULUAN

## **Ringkasan Isi Dokumen**

Dokumen ini berisikan penjelasan mengenai proses implementasi yang dilakukan untuk mewujudkan produk tugas akhir “Sistem Keamanan Kunci Pintu Gedung Berbasis Internet of Things”. Proses implementasi yang dilakukan meliputi pembuatan perangkat keras kunci pintu berupa hardware dan softwarenya, pembuatan tampilan baik itu berupa website maupun aplikasi mobile, pembuatan database dan backend server. proses implementasi yang dilakukan akan mengikuti desain yang telah dibuat dan dijelaskan pada dokumen desain (B300).

## **Aplikasi Dokumen**

Dokumen ini digunakan dalam proses pengembangan “Rancang Bangun Sistem Keamanan Kunci Pintu Gedung Berbasis *Internet of Things*” untuk:

1. Sebagai pennjelasan proses implementasi dari sistem yang talah dirancang baik dari segi hardware maupun software.
2. Sebagai acuan dalam proses pengujian sistem pada tahap selanjutnya.
3. Sebagai dokumentasi dan pencatatan perubahan.

Dokumen B400 ini diajukan kepada dosen pembimbing tugas akhir dan tim tugas akhir Program Studi Sarjana Teknik Elektro Undip sebagai bahan penilaian tugas akhir.

## **Referensi**

## **Daftar Singkatan**

**Tabel 1.1** Daftar Singkatan

| Singkatan | Arti |
| --- | --- |
| IoT | *Internet of Things* |
| WiFi | *Wireless Fidelity* |
| JSON | *Javascript Object Notation* |
| ACL | *Access Control List* |
| HTTPS | *Hypertext Transfer Protocol Secure* |
| IOS | *iPhone Operating System* |
| PC | *Personal Computer* |
| IDE | *Integrated Development Environment* |
| PCB | *Printed Circuit Board* |
| UI | *User Interface* |
| QR-*Code* | *Quick Response Code* |
| SSL/TLS | *Secure Sockets Layer/Transport Layer Security* |

1. IMPLEMENTASI
2. Perangkat Keras Kunci Pintu

Proses implementasi perangkat keras kunci pintu dilakukan dengan merangkai komponen-komponen yang telah ditentukan sebelumnya menjadi sebuah alat yang akan digunakan untuk mengunci pintu secara digital. Beberapa komponen yang digunakan adalah sebagai berikut :

1. ESP32

Diperlukan sebuah mikrokontroler yang bertugas untuk mengelola semua aktifitas pada perangkat kunci pintu, pada proses desain dijelaskan penggunaan ESP32 sebagai kontroler utama dengan memanfaatkan modul komunikasi WiFi yang sudah tersedia didalamnya.

******

Gambar 2.1 Konfigurasi ESP32

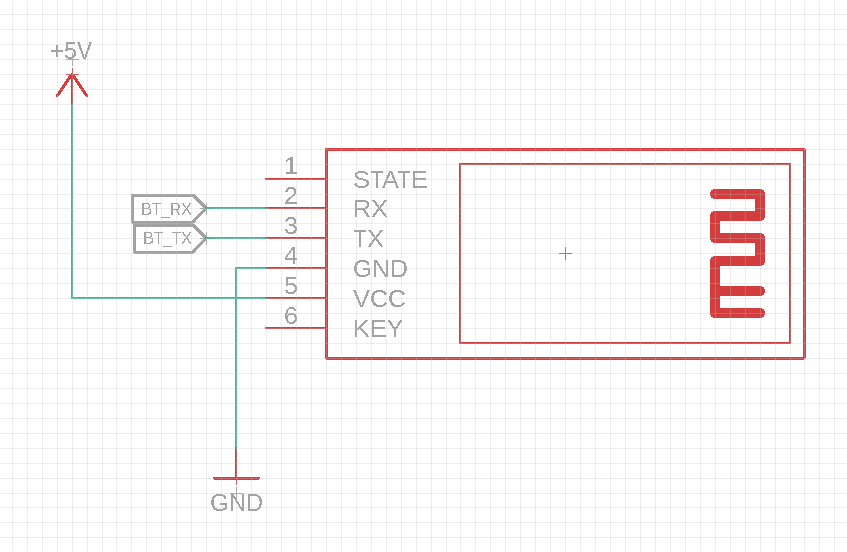
Gambar 2.1 diatas menjelaskan penggunaan ESP32 sebagai kontroler pada perangkat kunci pintu. ESP32 akan mengatur semua input dan output pada perangkat kunci pintu dengan penjelasan sesuai dengan tabel 2.1 dibawah.

Tabel 2.1 Konfigurasi Pin ESP32

|  |  |  |
| --- | --- | --- |
| **Pin** | **Mode** | **Keterangan** |
| D4 | Input | Digunakan sebagai sensor sentuh untuk mendeteksi interaksi pengguna |
| RX2 | Input | Terhubung dengan modul bluetooth untuk menerima data |
| TX2 | Output | Terhubung dengan modul bluetooth untuk mengirim data |
| D13 | Output | Terhubung dengan LED indikator status penguncian |
| D14 | Output | Terhubung dengan LED indikator proses pengiriman data |
| D27 | Output | Terhubung dengan LED indikator status koneksi WiFi |
| D26 | Input | Terhubung dengan tombol untuk membuka pintu dari dalam ruangan |
| D25 | Output | Terhubung dengan modul relay untuk menggerakkan solenoid kunci pintu |
| D33 | Input | Terhubung dengan saklar magnetik untuk membaca kondisi pintu |
| D32 | Output | Terhubung dengan buzzer untuk memberikan suara alarm peringatan |
| Vin | Power | Terhubung dengan sumber catu daya. |

1. Bluetooth HC05

Modul HC05 digunakan untuk melakukan komunikasi dengan aplikasi mobile menggunakan koneksi bluetooth, meskipun ESP32 juga dilengkapi dengan modul bluetooth namun pada implementasinya ESP32 tidak bisa menjalankan modul WiFi dan bluetooth secara bersamaan dikarenakan keduanya terhubung ke satu modul radio yang sama, keterbatasan memory yang dimiliki untuk menjalankan dua proses tersebut secara bersama-sama juga menjadi batasan dalam implementasi keduanya.



Gambar 2.2 Konfigurasi Bluetooth HC05

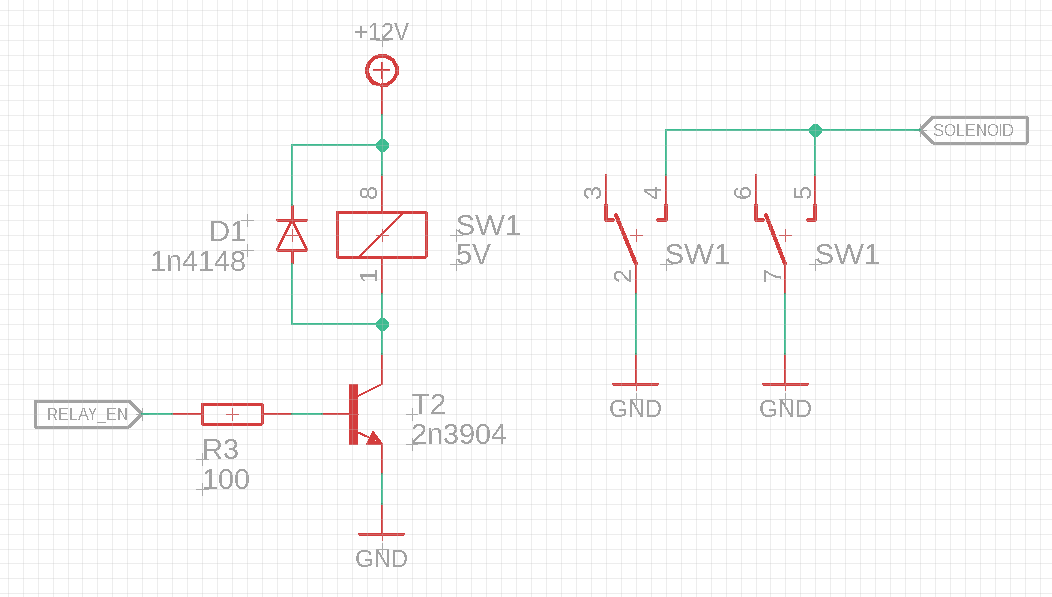
Pada gambar 2.2 diatas terlihat penggunaan modul bluetooth HC05 pada perangkat kunci pintu. Modul bluetooth HC05 menggunakan komunikasi serial untuk berkomunikasi dengan mikrokontroler dengan bautrate yang telah ditentukan. Pada modul bluetooth terdapat 4 pin yang digunakan sesuai dengan penjelasan pada tabel 2.2 dibawah.

Tabel 2.2 Konfigurasi Pin Bluetooth HC05

|  |  |  |
| --- | --- | --- |
| **Pin** | **Mode** | **Keterangan** |
| VCC | Power | Terhubung ke sumber catu daya untuk menghidupkan modul bluetooth |
| GND | Power | Referensi 0 volt |
| TX | Output | Terhubung ke mikrokontroler untuk mengirimkan data |
| RX | Input | Terhubung ke mikrokontroler untuk menerima data |

1. Solenoid

Untuk melakukan penguncian secara mekanik maka diperlukan sebuah solenoid untuk mengubah arus listrik dari kontroller menjadi gerakan mekanik dengan menggunakan prinsip elektromagnetik.

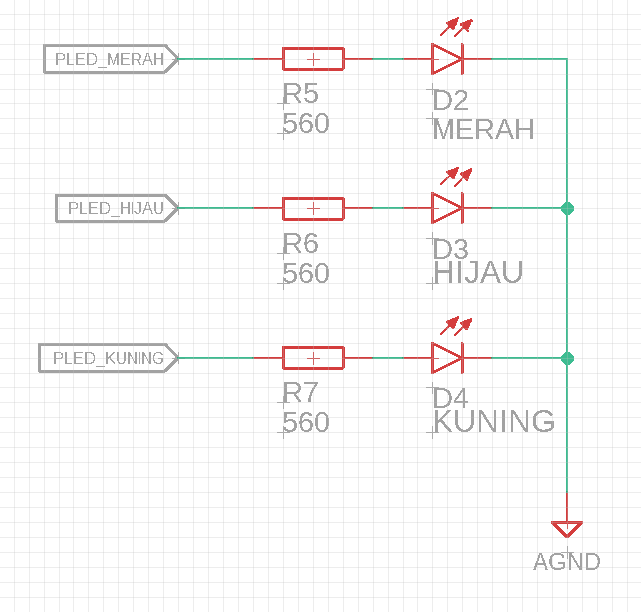


Gambar 2.3 Konfigurasi Solenoid

Pada gambar 2.3 diatas, untuk menggerakkan solenoid mikrokontroler memerlukan bantuan dari sebuah modul relay hal ini dikarenakan mikrokontroler hanya mampu mengalirkan arus beberapa mA sehingga tidak mampu untuk menggerakkan solenoid recara langsung.

1. LED

Untuk memberikan informasi terkait dengan kondisi perangkat kunci pintu maka diperlukan LED untuk memberikan informasi kondisi dari perangkat kunci pintu.

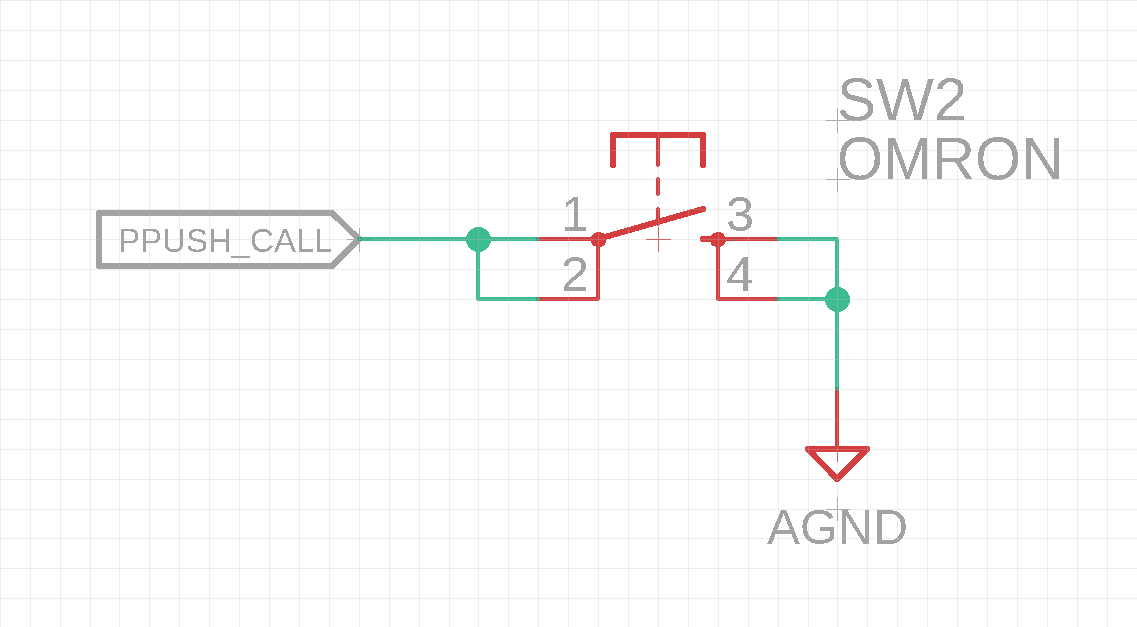


Gambar 2.4 Konfigurasi LED

Pada perangkat kunci pintu terdapat 3 LED status yaitu LED merah untuk memberikan informasi status penguncian, LED hijau untuk memberikan informasi status pengiriman data dan led kuning untuk memberikan status koneksi WiFi.

1. Tombol

Karena solenoid tidak bisa bekerja secara terus menerus maka solenoid hanya akan aktif jika ada interaksi dari pengguna, oleh karena itu kondisi pintu akan selalu terkunci. Untuk membuka pintu pengguna dapat menggunakan tombol yang disediakan untuk membuka pintu dari dalam.

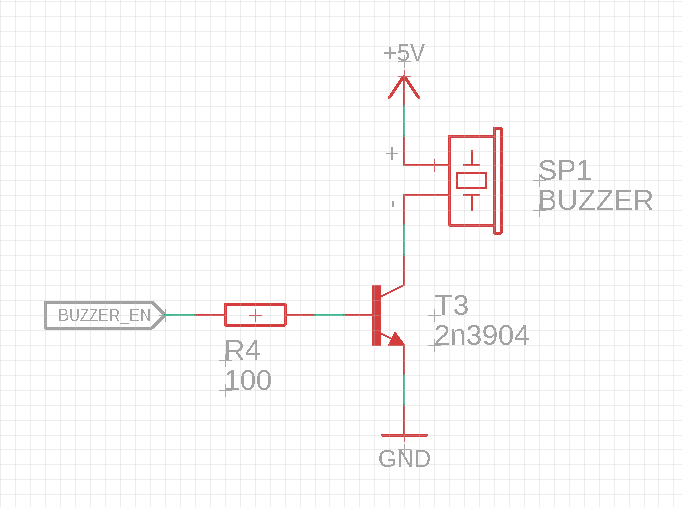


Gambar 2.5 Konfigurasi Tombol

Pada gambar 2.5 diatas terlihat penggunaan tombol menggunakan konfigurasi pull down dengan memanfaatkan kondisi pull-up yang disediakan mikrokontroler. Pada saat ditekan maka tombol akan menghasilkan logika LOW yang dapat dibaca oleh mikrokontroler.

1. Buzzer

Untuk memberikan peringatan suara pada perangkat kunci pintu dapat menggunakan buzzer, buzzer akan mengeluarkan suara “beep” pada saat dialiri listrik.



Gambar 2.6 Konfigurasi Buzzer

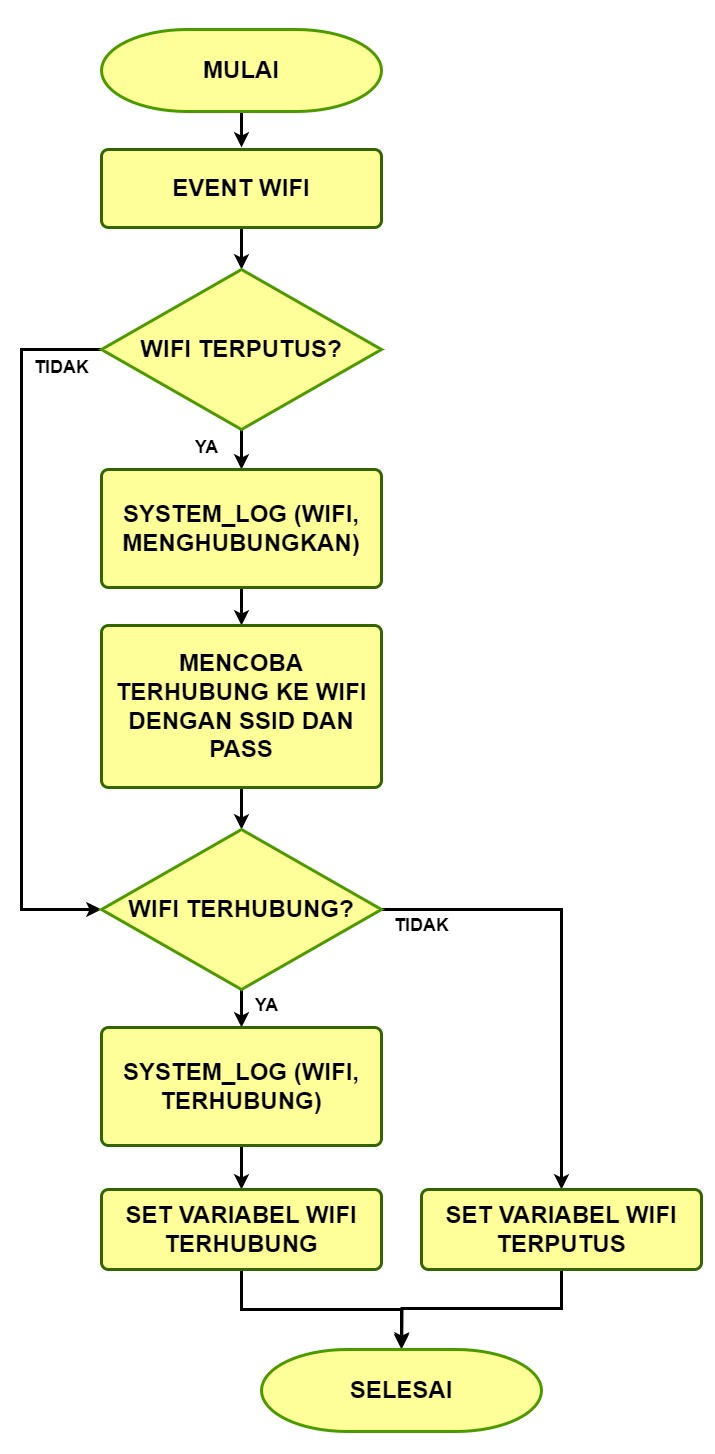
Pada gambar 2.6 diatas, penggunaan buzzer pada perangkat kunci pintu menggunakan sebuah transistor untuk membantu mikrokontroler dalam mengaktifkan buzzer hal ini dikarenakan arus yang dikeluarkan mikrokontroller tidak cukup untuk menyalakan buzzer sacara maksimal.

1. Perangkat Lunak Kunci Pintu

Implementasi perangkat lunak pada perangkat kunci pintu dilakukan dengan menggunakan Arduino. Aduino merupakan sebuah lingkungan pengembangan program mikrokotroler dengan menggunakan bahasa C++ yang sudah dipermudah. Beberapa metode yang terdapat pada perangkat kunci pintu adalah sebagai berikut :

1. Penanganan koneksi WiFi

Perangkat kunci pintu memerlukan koneksi ke internet koneksi tersebut dilakukan melalui modul WiFI yang telah disediakan oleh ESP32. Setiap terjadi perubahan kondisi WiFi maka perangkat penguncian mampu beradaptasi seperti melakukan koneksi ulang jika terjadi masalah pada komunikasi WiFi tersebut.

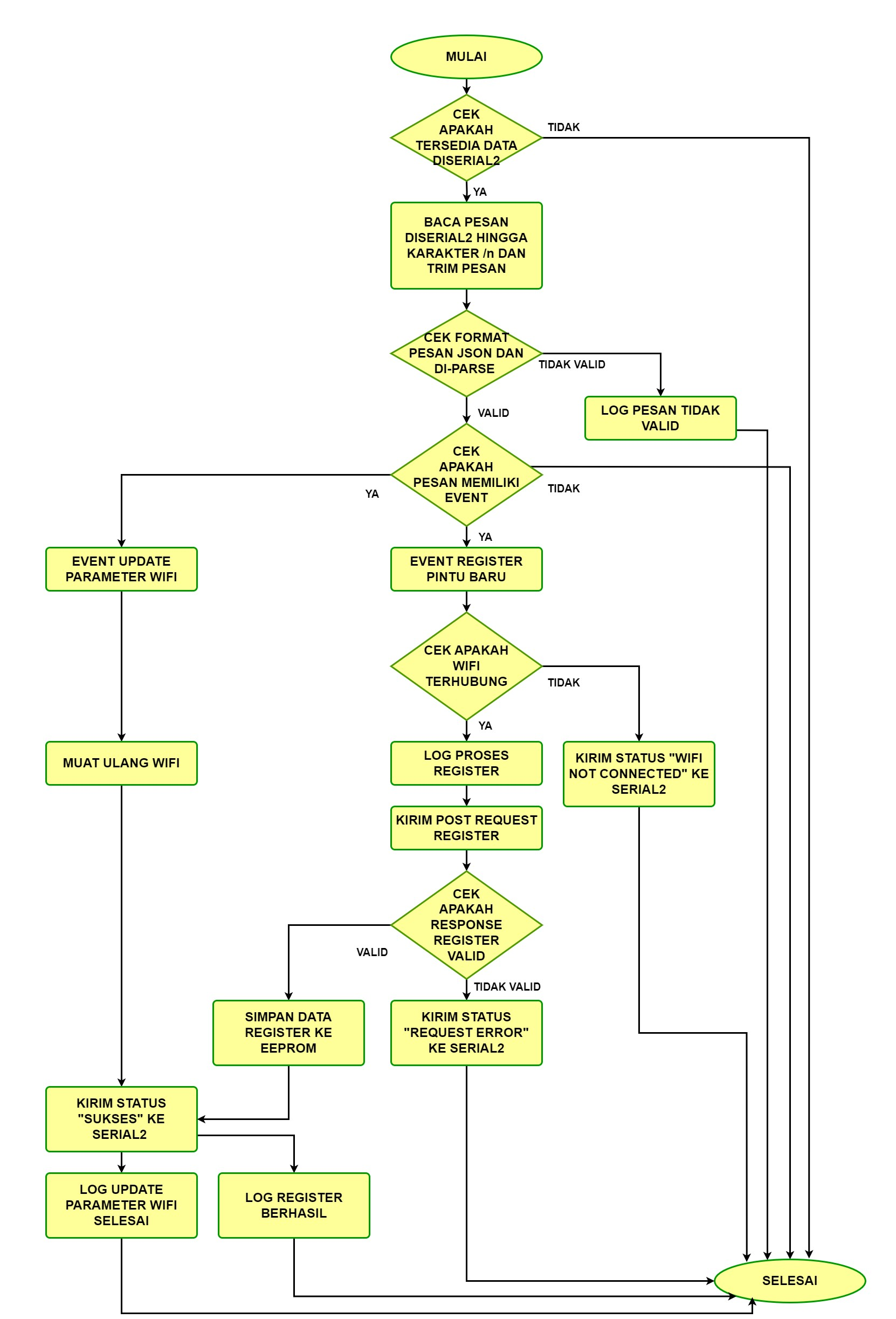


Gambar 2.7 Flowchart Koneksi WiFi

Berdasarkan pada Gambar 2.7 di atas, memperlihatkan cara kerja program untuk menangani kondisi WiFi. Kondisi WiFi akan dipantau menggunakan sebuah event, setiap terjadi perubahan status maka event tersebut juga akan berubah dan akan memperbarui status koneksi WiFi menggunakan sebuah variabel. Variable tersebut akan digunakan untuk menyimpan kondisi WiFi sehingga kondisi WiFi dapat diketauhi secara global untuk memastikan WiFI terhubung sebelum berkomunikasi dengan server. pada kondisi WiFi terputus maka variabel status akan diperbarui dan program akan mencoba menghubungkan ulang, pada kondisi WiFi baru saja terhubung maka status koneksi juga akan diperbarui.

1. Pengaturan perangkat kunci pintu

Proses pengaturan kunci pintu seperti mengatur kredensial WiFi serta mendaftarkan perangkat baru ke server dilakukan menggunakan aplikasi mobile dengan perintah yang dikirimkan melalui koneksi bleutooth.

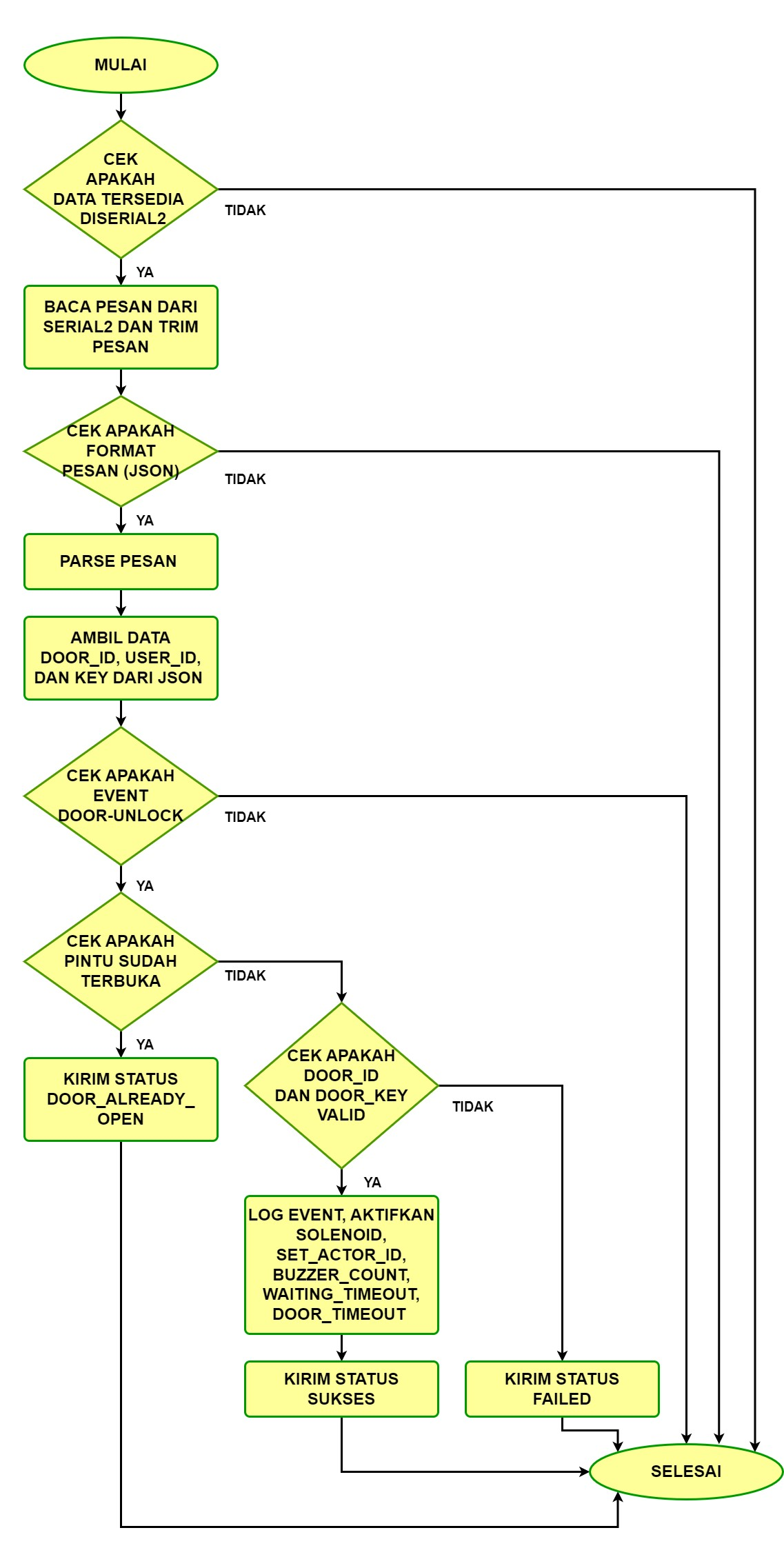


Gambar 2.8 Flowchart Fungsi config\_loop()

Berdasarkan pada Gambar 2.8, Fungsi **config\_loop(void)** akan memeriksa apakah ada data yang tersedia di antarmuka **Serial2** yang terhubung ke modul *bluetooth* HC-05. Data yang diterima di **Serial2** dibaca hingga menemukan karakter *newline* (**'\n'**) dan disimpan dalam variabel **bt\_message**. Pesan yang telah diterima kemudian dipangkas untuk menghilangkan karakter yang tidak diinginkan menggunakan fungsi **trim()**. Selanjutnya, kode ini memeriksa apakah pesan yang diterima merupakan objek JSON yang valid dengan memeriksa apakah pesan dimulai dengan '{' dan diakhiri dengan '}' menggunakan **bt\_message.startsWith("{")** dan **bt\_message.endsWith("}")**. Jika pesan tersebut adalah objek JSON yang valid, selanjutnya dilakukan *parsing* menggunakan fungsi **JSON.parse(bt\_message).** Selanjutnya, fungsi **config\_loop(void)** memeriksa properti "*event*" dari objek JSON yang di-*parsing* untuk mengidentifikasi jenis pembaruan konfigurasi yang diminta menggunakan **bt\_package.hasOwnProperty("event")**. Jika properti "*event*" ada, maka kode akan memeriksa nilai dari properti "*event*" untuk menentukan tindakan selanjutnya. Jika nilai properti *event* adalah "*wifi-update*", maka fungsi **config\_loop(void)** melakukan *update* parameter WiFi berupa data "ssid" dan "*password*" dari objek JSON. Selain itu, fungsi ini mengirim status keberhasilan kembali ke aplikasi *mobile* melalui koneksi *Bluetooth*. Jika *event* adalah "*register*", pertama fungsi ini memeriksa apakah WiFi terhubung. Jika tidak, fungsi ini merespons dengan status yang menunjukkan bahwa WiFi tidak terhubung. Jika WiFi terhubung, fungsi ini melanjutkan proses pendaftaran pintu baru. Proses pendaftaran pintu melibatkan pengiriman permintaan POST ke sebuah *endpoint server* ("/*register*") dengan data *"door\_id"* dan *"device\_name*" melalaui HTTP. Selanjutnya, tanggapan dari *server* di-*parsing*, dan jika pendaftaran berhasil, data relevan (*office* ID, *door* ID, nama pintu, dll) disimpan di dalam EEPROM. Selain itu, status keberhasilan atau kegagalan dikirim kembali ke aplikasi *mobile* melalui koneksi *Bluetooth*.

1. Perintah penguncian menggunakan bluetooth

Untuk membuka kunci pintu, setiap pengguna akan menggunakan aplikasi mobile untuk berkomunikasi dengan perangkat kunci pintu. Pada kondisi ini proses verifikasi pengguna dilakukan dengan memeriksa setiap pesan yang dikirimkan oleh pengguna. Untuk membuka kunci pintu maka pengguna harus mengirimkan pesan sesuai dengan format perintah yang diketahui oleh perangkat kunci pintu. Program yang digunakan untuk melakukan verifikasi tersebut dapat dilihat pada gambar 2.9 dibawah.

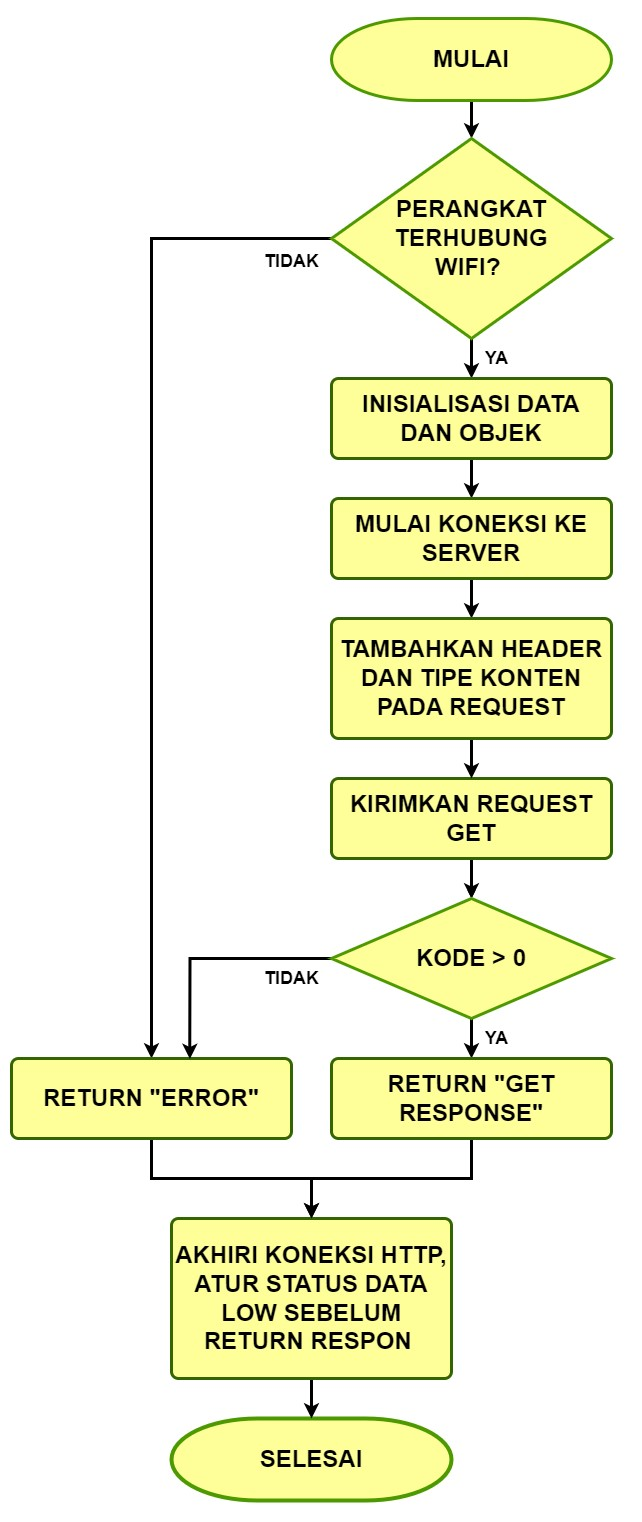
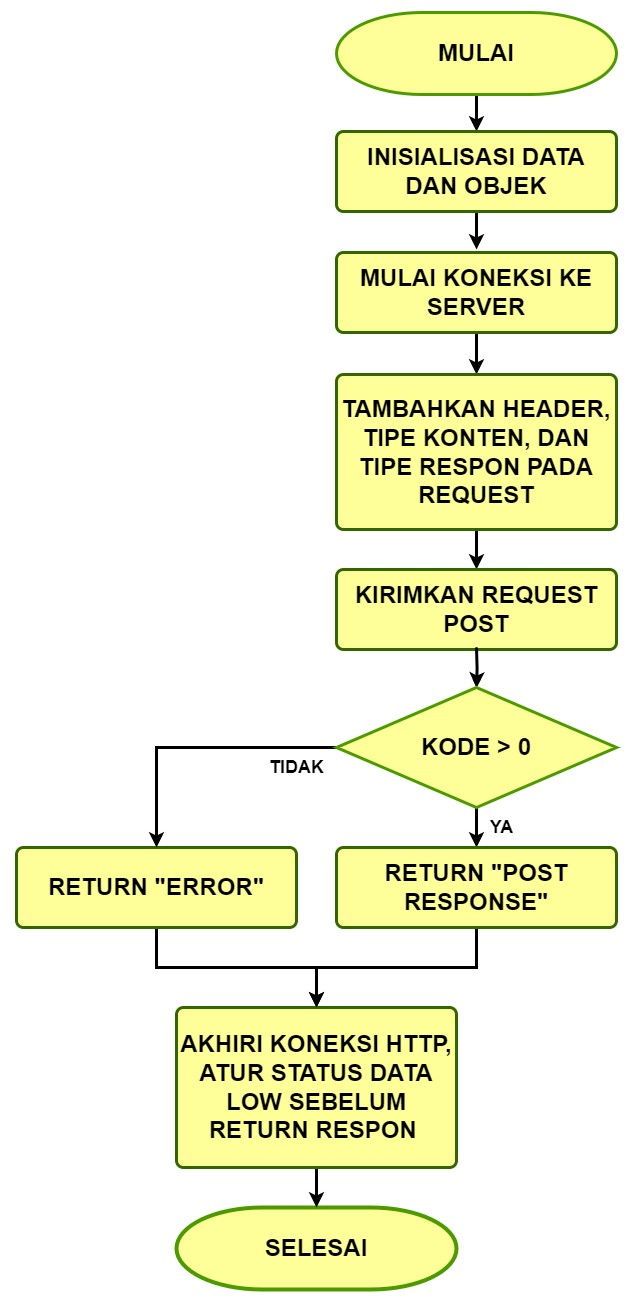


Gambar 2.9 Flowchart Fungsi bt\_door\_command()

Berdasarkan pada Gambar 2.9, Fungsi **bt\_door\_command(void)** memeriksa apakah ada data yang tersedia di antarmuka **Serial2** (*Bluetooth*). Data yang diterima dibaca hingga menemukan karakter *newline* (**'\n'**) dan disimpan dalam variabel **bt\_message.** Pesan yang telah diterima kemudian dipangkas untuk menghilangkan karakter yang tidak diinginkan menggunakan fungsi **trim()**. Fungsi **bt\_door\_command(void)** juga memverifikasi apakah pesan yang diterima memiliki format JSON yang valid (dimulai dengan "{" dan diakhiri dengan "}") menggunakan **bt\_message.startsWith("{")**dan **bt\_message.endsWith("}")**. Jika pesan tersebut adalah objek JSON yang valid, selanjutnya dilakukan *parsing* menggunakan fungsi **JSON.parse(bt\_message)**. Kemudian, fungsi **bt\_door\_command(void)** mengekstrak data "*door\_id," "user\_id*," dan "*key*" dari objek JSON. Jika *event* dalam objek JSON yang di-*parsing* adalah "*door-unlock*," fungsi **bt\_door\_command(void)** memeriksa apakah pintu sudah terbuka. Jika sudah, fungsi **bt\_door\_command(void)** merespons dengan status yang menunjukkan bahwa pintu sudah terbuka. Jika belum, fungsi **bt\_door\_command(void)** memeriksa apakah "*door\_id"* dan *"key*" yang diterima sesuai dengan nilai yang disimpan di dalam EEPROM. Jika sesuai, fungsi **bt\_door\_command(void)** memulai proses membuka kunci pintu. Proses membuka kunci pintu melibatkan mengaktifkan solenoid dan melakukan beberapa tindakan terkait lainnya.

1. Komunikasi dengan server

Perangkat kunci pintu akan mengirimkan informasi terkait dengan kondisi pintu dan melakukan autentikasi untuk terhubung ke ssistem penguncian. Dalam implementasi komunikasi dengan menggunakan protokol HTTPS dengan proses enkripsi SLL sehingga setiap data yang dikirimkan akan dienkripsi dengan aman. Protokol HTTPS yang didukung meliputi permintaan GET dan POST sesuai dengan standar komunikasi HTTP.

Gambar 2.10 Flowchart Fungsi HTTP GET dan POST

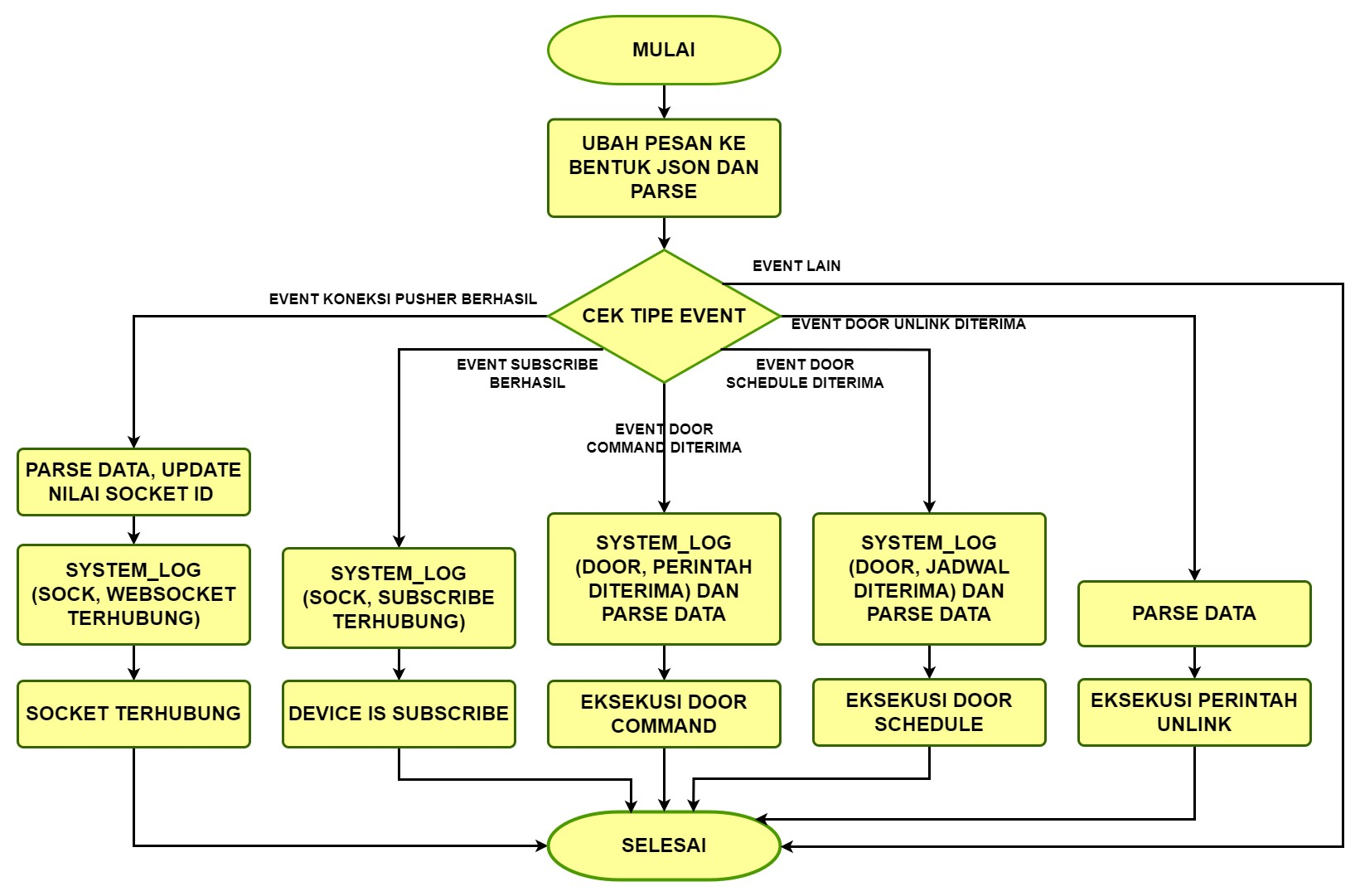
Berdasarkan Gambar 2.10, *flowchart* di atas memperlihatkan diagram alir pemrograman fungsi **post\_request()** dan **get\_request()** yang berfungsi untuk melakukan permintaan HTTP ke *server*.

Fungsi **get\_request(String endpoint)** digunakan untuk melakukan permintaan HTTP GET ke *endpoint* yang ditentukan pada *server*. Fungsi tersebut memeriksa apakah koneksi WiFi tersedia (**wifi\_is\_connected**) sebelum melanjutkan. Jika tidak tersedia, maka fungsi tersebut akan mengembalikan *string* "*error*". Fungsi tersebut juga melakukan inisialisasi objek *HTTPClient* (**http\_get**) dan variabel *String* (**get\_response**) untuk menyimpan respon dari *server*. Fungsi tersebut memulai koneksi dengan menetapkan pin **data\_status** ke nilai HIGH dan memulai permintaan HTTP menggunakan **http\_get.begin(...)**. Fungsi tersebut menambahkan *header* yang diperlukan untuk permintaan, seperti token otorisasi (**Bearer** **token**) dan format respon yang diharapkan (**Accept: application/json**). Permintaan GET dikirim menggunakan **http\_get.GET()** yang mengembalikan kode respon HTTP. Jika kode respon lebih besar dari 0, menandakan permintaan berhasil, fungsi ini membaca isi respon dari *server* ke dalam variabel **get\_response** menggunakan **http\_get.getString().** Namun, jika permintaan gagal (kode respon <= 0), variabel **get\_response** diatur menjadi "*error*". Terakhir, koneksi HTTP ditutup, dan pin **data\_status** diatur kembali ke nilai LOW. Fungsi tersebut mengembalikan *string* **get\_response** yang berisi respon dari *server* atau pesan "*error*".

Fungsi **post\_request(String endpoint, String payload)** digunakan untuk melakukan permintaan HTTP POST ke *endpoint* yang ditentukan pada *server*, dengan mengirimkan data dalam *payload*. Fungsi tersebut mengikuti proses yang mirip dengan fungsi **get\_request**, namun dengan modifikasi yang diperlukan untuk permintaan POST. Fungsi tersebut melakukan inisialisasi objek *HTTPClient* (**http\_post**) dan variabel *String* (**post\_response**) untuk menyimpan respon dari *server*. Fungsi tersebut menetapkan pin **data\_status** ke nilai HIGH dan memulai permintaan HTTP menggunakan **http\_post.begin(...)**. *Header* untuk otorisasi (**Authorization**) dan tipe konten (**Content-Type: application/x-www-form-urlencoded**) ditambahkan ke permintaan, bersama dengan format respon yang diharapkan (**Accept: application/json**). Permintaan POST dengan *payload* yang diberikan dikirim menggunakan **http\_post.POST(payload)** yang mengembalikan kode respon HTTP. Jika kode respon lebih besar dari 0, menandakan permintaan berhasil, fungsi tersebut membaca isi respon dari *server* ke dalam variabel **post\_response** menggunakan **http\_post.getString()**. Jika permintaan gagal (kode respon <= 0), variabel **post\_response** diatur menjadi "*error*". Koneksi HTTP ditutup dan pin **data\_status** diatur kembali ke nilai LOW. Fungsi tersebut mengembalikan *string* **post\_response** yang berisi respon dari *server* atau pesan "*error*".

1. Penanganan perintah dari server

Server akan mengirimkan perintah ke perangkat penguncian menggunakan sebuah event websocket oleh karena itu perangkat kunci pintu harus dapat membaca setiap event yang dikirimkan oleh server.



Gambar 2.11 Flowchart Fungsi websocket\_message()

Berdasarkan Gambar 2.11 di atas fungsi **void websocket\_message()**dipanggil ketika sebuah pesan *WebSocket* diterima. Di dalam fungsi **websocket\_message**, kode pertama kali mengurai pesan yang diterima ke dalam format JSON menggunakan **JSON.parse()**. Objek JSONVar **message** digunakan untuk menyimpan data pesan yang telah di-*parse*. Selanjutnya, fungsi **websocket\_message** mengecek nilai *field* **"event"** dari pesan JSON yang telah di-*parse* untuk menentukan tipe *event* yang diterima. Jika *event* adalah **"pusher:connection\_established"**, berarti koneksi *WebSocket* telah berhasil dibuat. Kemudian, fungsi **websocket\_message** mengurai *field* **"data"** dari pesan yang merupakan *string* format JSON lainnya menggunakan **JSON.parse(),** mengambil nilai *field* **"socket\_id"** dari **pusher\_data** yang telah di-*parse*, memperbarui nilai variabel **socket\_id** dengan **socket\_id** yang telah diambil, memanggil fungsi **system\_log** dengan parameter "SOCK" dan "*websocket* terhubung" yang menandakan "*WebSocket connected*", dan mengeset variabel **socket\_is\_connected** menjadi **true** yang menandakan bahwa koneksi **WebSocket** sekarang terhubung.

Jika *event* adalah **"pusher\_internal:subscription\_succeeded"**, berarti *subscribe/*langganan ke saluran atau topik telah berhasil dibuat, maka fungsi **websocket\_message** memanggil fungsi **system\_log** dengan parameter "SOCK" dan "*subscribe* berhasil" yang menandakan "*subscription succeeded*" dan mengeset variabel **device\_is\_subscribe** menjadi **true** yang menandakan bahwa perangkat telah berhasil berlangganan/*subscribe* ke saluran.

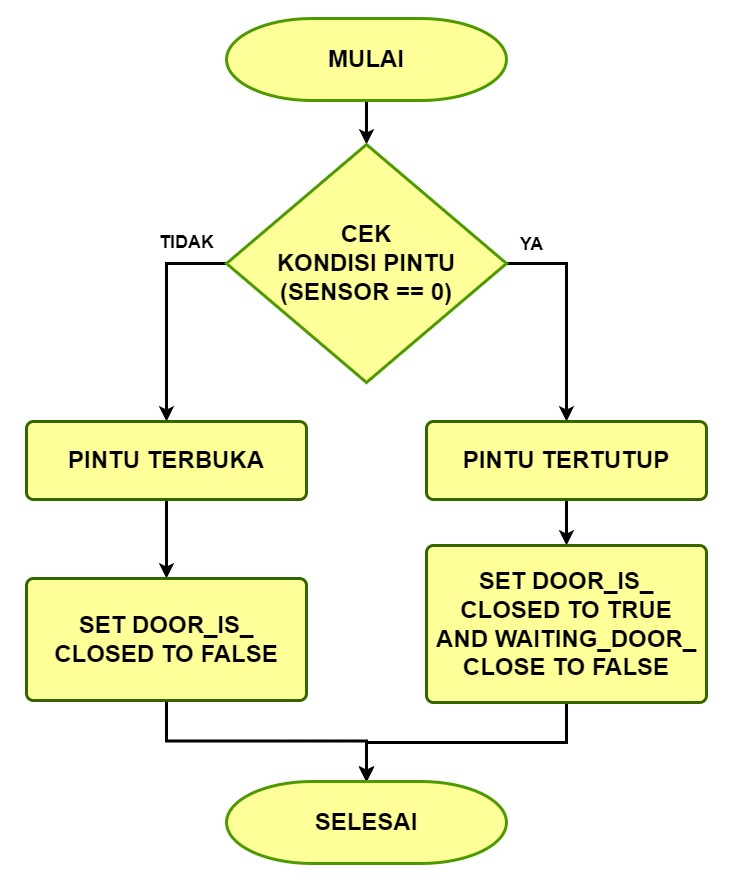
Jika *event* adalah **"door-command"**, berarti pesan "*door-command*" telah diterima. Fungsi **websocket\_message** memanggil fungsi **system\_log** dengan parameter "DOOR" dan "perintah diterima" yang menandakan "*command received*", mengurai *field* **"data"** dari pesan yang merupakan *string* format JSON lainnya menggunakan **JSON.parse()**, dan memanggil fungsi **door\_command** dengan data **command** yang telah di-*parse* sebagai argumen.

Jika *event* adalah **"door-schedule"**, itu berarti pesan "*door-schedule*" telah diterima. Fungsi **websocket\_message** memanggil fungsi **system\_log** dengan parameter "DOOR" dan "jadwal diterima" yang menandakan "*schedule received*", mengurai/mem-*parse* *field* **"data"** dari pesan yang merupakan *string* format JSON lainnya menggunakan **JSON.parse()**, memanggil fungsi **door\_schedule** dengan data **schedule** yang telah di-*parse* sebagai argumen.

Jika *event* adalah **"door-unlink"**, berarti pesan "*door-unlink*" telah diterima. Fungsi **websocket\_message** mengurai/ mem-*parse field* **"data"** dari pesan yang merupakan *string* format JSON lainnya menggunakan **JSON.parse()** dan memanggil fungsi **unlink** dengan data **unlink\_data** yang telah di-*parse* sebagai argumen.

1. Pembacaan kondisi pintu

Kondisi pintu dibaca dengan menggunakan saklar magnetik, saklar magnetik akan menggunakan medan magnet untuk menentukan kondisi pintu sedang terbuka atau tertutup. Setiap terjadi perubahan maka perangkat kunci pintu akan memperbarui status pintu. Cara kerja dari program kondisi pintu dapat dilihat pada gambar 2.12 dibawah .

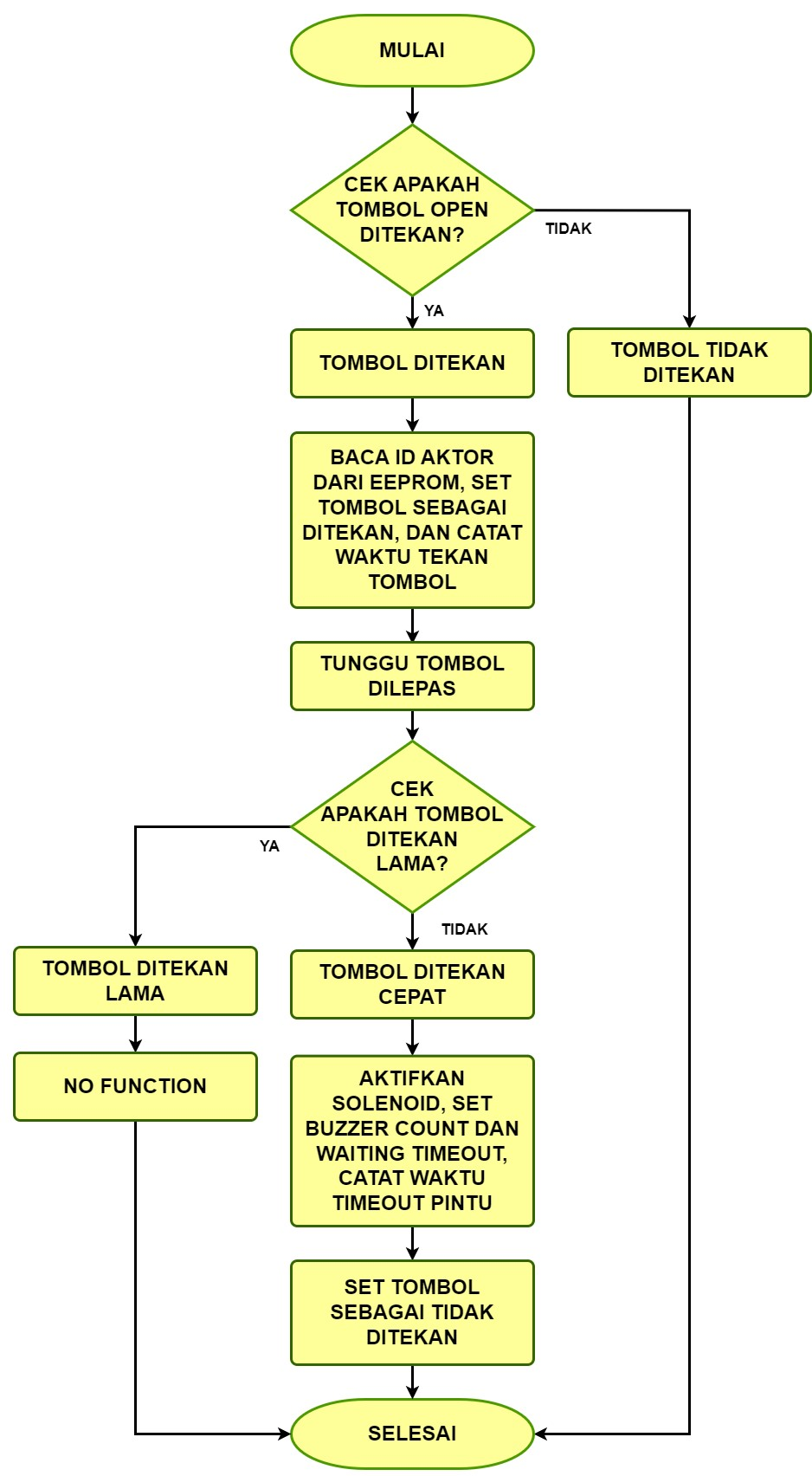


Gambar 2.12 Flowchart Pembacaan Kondisi Pintu

Berdasarkan pada Gambar 2.12 di atas, program dimulai dengan pernyataan pengkondisian menggunakan **if**. Dengan fungsi **digitalRead(sensor)** akan membaca nilai digital dari *input* "sensor". Nilai yang dikembalikan oleh fungsi tersebut berupa **0** jika pintu dalam keadaan tertutup, dan **1** jika pintu dalam keadaan terbuka. Jika hasil bacaan dari sensor adalah **0** (pintu dalam keadaan tertutup), maka blok kode dieksekusi dengan menyetel variabel **door\_is\_closed** ke **true** untuk menandakan bahwa pintu dalam keadaan tertutup dan variabel **waiting\_door\_close** ke **false** untuk digunakan di bagian kode lain yang terkait dengan penanganan pintu dalam keadaan tertutup. Jika hasil bacaan dari sensor bukan 0 (pintu dalam keadaan terbuka), maka blok kode dieksekusi dengan menyetel variabel **door\_is\_closed** ke **false** yang menandakan bahwa pintu dalam keadaan terbuka.

1. Penanganan interaksi pengguna

Pengguna dapat berinteraksi untuk membuka pintu menggunakan berbagai metode. Metode yang mungkin digunakan oleh pengguna yaitu dengan menggunakan tombol yang telah disediakan atau menggunakan aplikasi mobile untuk membuka kunci pintu. Proses penanganan interaksi pengguna dapat dilihat pada gambar 2.13 dibawah.



Gambar 2.13 Flowchart Penanganan Interaksi Pengguna

Berdasarkan pada Gambar 2.13 di atas, pertama program memeriksa apakah tombol "*open*" ditekan menggunakan kode(**digitalRead(button) == LOW**) dengan beberapa kondisi harus terpenuhi yaitu **button\_is\_pressed** bernilai **false**, **waiting\_door\_close** bernilai **false**, dan **door\_is\_closed** bernilai **true.**

Jika semua kondisi tersebut terpenuhi, kode akan melanjutkan untuk menjalankan tindakan-tindakan untuk membaca nilai **door\_id\_addr** dari EEPROM dan menyimpannya dalam variabel **actor\_id**, menetapkan **button\_is\_pressed** menjadi **true**, merekam waktu saat tombol ditekan menggunakan fungsi **millis()** dan menyimpannya dalam variabel **button\_pressed\_time**. Selanjutnya, memeriksa apakah tombol "*open*" dilepaskan dengan menggunakan kode (**digitalRead(button) == HIGH**) dan jika **button\_is\_pressed** bernilai **true**. Jika ya, kode akan memeriksa berapa lama tombol ditekan. Jika tombol ditekan lebih dari 700 milidetik (**(millis() - button\_pressed\_time) > 700**), maka tidak ada tindakan yang dilakukan (**// no function** *comment*) yang berarti bahwa jika tombol ditekan dalam waktu yang lebih lama, beberapa tindakan khusus dapat ditambahkan, tetapi dalam kode tersebut tidak ada tindakan khusus yang dijalankan untuk kasus tersebut. Jika tombol dilepaskan dengan cepat (ditekan selama kurang dari atau sama dengan 700 milidetik), tindakan-tindakan berikut dilakukan yaitu menetapkan solenoid menjadi aktif, menetapkan **buzzer\_count** menjadi 2 (yang digunakan untuk mengatur bunyi *buzzer*), menetapkan **waiting\_timeout** menjadi **true**, dan merekam waktu saat ini menggunakan **millis()**, serta menyimpannya dalam variabel **door\_timeout**. Terakhir, kode menetapkan **button\_is\_pressed** menjadi **false** yang berarti kode siap untuk mendeteksi tekanan tombol kembali.

1. Database

Implementasi dilakukan dengan membuat sebuah datatabase menggunakan MySQL sesuai dengan diagram yang telah ditentukan. Database mungkin akan berisi tabel tambahan yang dibutuhkan oleh kerangka kerja Laravel. Pembuatan skema database dilakukan dengan menggunakan mekanisme migrasi yang disediakan oleh laravel sehingga struktur dari database dapat dengan mudah dirubah dengan cara mengubah kode migrasi. Penjelasan mengenai struktur dari setiap tabel dapat dilihat pada Tabel 2.3 sampai Tabel 2.9.

Tabel 2.3 Struktur Tabel Users

| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- | --- |
| 1 | id | char(36) | kunci primer tabel |
| 2 | added\_by | char(36) | referensi aktor |
| 3 | email | varchar(255) | alamat email pengguna |
| 4 | password | varchar(255) | password pengguna |
| 5 | remember\_token | varchar(255) | token untuk fitur "ingat saya" |
| 6 | name | varchar(255) | nama penggua |
| 7 | phone | varchar(255) | nomor hp pengguna |
| 8 | gender | enum | jenis kelamin pengguna |
| 9 | role | enum | jabatan pengguna |
| 10 | avatar | varchar(255) | foto profil pengguna |
| 11 | email\_verified\_at | timestamp | waktu verifikasi email |
| 12 | created\_at | timestamp | waktu data dibuat |
| 13 | updated\_at | timestamp | waktu data diedit |

Tabel 2.4 Struktur Tabel Offices

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | user\_id | char(36) | referensi operator gedung |
| 3 | name | varchar(255) | nama gedung |
| 4 | created\_at | timestamp | waktu data dibuat |
| 5 | updated\_at | timestamp | waktu data diedit |

Tabel 2.5 Struktur Tabel Doors

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | office\_id | char(36) | referensi gedung |
| 3 | socket\_id | char(36) | identitas websocket |
| 4 | name | varchar(36) | nama pintu |
| 5 | device\_name | varchar(36) | username pintu |
| 6 | device\_pass | varchar(36) | password pintu |
| 7 | key | varchar(36) | kunci pintu |
| 8 | is\_lock | tinyint(1) | status penguncia |
| 9 | created\_at | timestamp | waktu data dibuat |
| 10 | updated\_at | timestamp | waktu data diedit |

Tabel 2.6 Struktur Tabel Access

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | user\_id | char(36) | referensi ke pengguna |
| 3 | door\_id | char(36) | referensi ke pintu |
| 4 | time\_begin | time | waktu mulai |
| 5 | time\_end | time | waktu berakhir |
| 6 | date\_begin | date | tanggal mulai |
| 7 | date\_end | date | tanggal berakhir |
| 8 | is\_temporary | tinyint(1) | status akses sementara |
| 9 | is\_running | tinyint(1) | status akses sedang berjalan |
| 10 | created\_at | timestamp | waktu data dibuat |
| 11 | updated\_at | timestamp | waktu data diedit |

Tabel 2.7 Struktur Tabel Schedules

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | office\_id | char(36) | referensi gedung |
| 3 | name | varchar(255) | nama jadwal |
| 4 | date\_begin | date | tanggal mulai |
| 5 | date\_end | date | tanggal berakhir |
| 6 | time\_begin | time | waktu mulai |
| 7 | time\_end | time | waktu berakhir |
| 8 | is\_repeating | tinyint(1) | status perulangan |
| 9 | day\_repeating | varchar(255) | perulangan hari |
| 10 | status | enum | status pelaksanaan jadwal |
| 11 | created\_at | timestamp | waktu data dibuat |
| 12 | updated\_at | timestamp | waktu data diedit |

Tabel 2.8 Struktur Tabel Door Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | schedule\_id | char(36) | referensi ke jadwal |
| 3 | door\_id | char(36) | referensi ke pintu |
| 4 | created\_at | timestamp | waktu data dibuat |
| 5 | updated\_at | timestamp | waktu data diedit |

Tabel 2.9 Struktur Tabel Riwayat Akses

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Kolom** | **Tipe Data** | **Keterangan** |
| 1 | id | char(36) | kunci primer tabel |
| 2 | user\_id | char(36) | referensi ke pengguna |
| 3 | office\_id | char(36) | referensi ke gedung |
| 4 | door\_id | char(36) | referensi ke pintu |
| 5 | log | text | pesan log |
| 6 | created\_at | timestamp | waktu data dibuat |
| 7 | updated\_at | timestamp | waktu data diedit |

Dengan menggunakan relational database tentunya akan terdapat hubungan atau relasi antara dua tabel atau lebih, relasi memberikan informasi mengenai hubungan antara dua tabel atau lebih yang saling berkaitan beserta dengan perilaku terhadap perubahan data pada tabel induknya. Relasi pada setiap tabel dapat dilihat pada Tabel 2.10 dibawah.

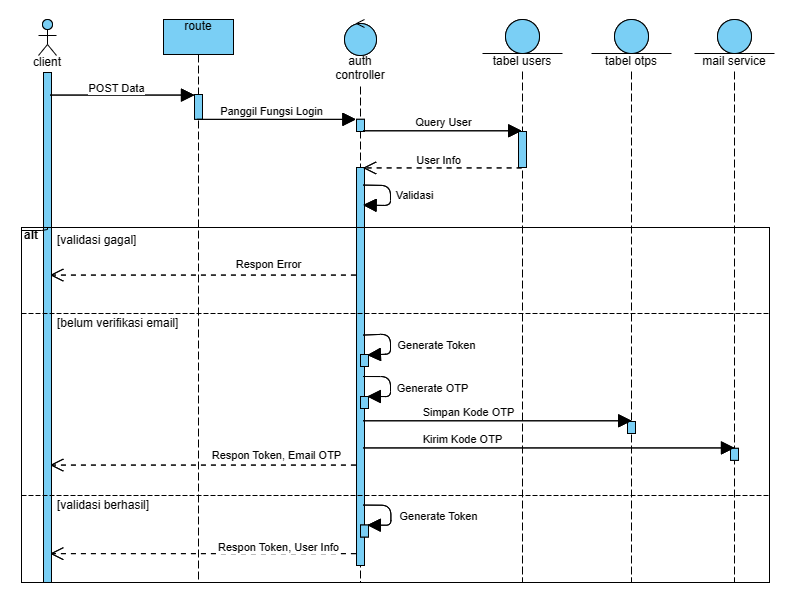
Tabel 2.10 Relasi Tabel

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Tabel Induk** | **Anak** | | **Jenis Hubungan** | **On Update** | **On Delete** |
| **Tabel** | **Kolom** |
| 1 | users | users | added\_by | one to many | cascade | restrict |
| 2 | users | access | user\_id | one to many | cascade | cascade |
| 3 | doors | access | door\_id | one to many | cascade | cascade |
| 4 | users | access\_logs | user\_id | one to many | cascade | cascade |
| 5 | doors | access\_logs | door\_id | one to many | cascade | cascade |
| 6 | offices | access\_logs | office\_id | one to many | cascade | cascade |
| 7 | offices | doors | office\_id | one to many | cascade | cascade |
| 8 | schedules | door\_schedule | schedule\_id | one to many | cascade | cascade |
| 9 | doors | door\_schedule | door\_id | one to many | cascade | cascade |
| 10 | users | offices | user\_id | one to one | cascade | restrict |
| 11 | users | otps | user\_id | one to one | cascade | cascade |
| 12 | offices | schedules | office\_id | one to many | cascade | cascade |

1. Backend API

Implementasi backend API dilakukan dengan menggunakan kerangka kerja Laravel menggunakan bahasa pemrograman PHP. Beberapa metode API yang diimplementasikan untuk mendukung kinerja dari sistem keamanan kunci pintu adalah sebagai berikut :

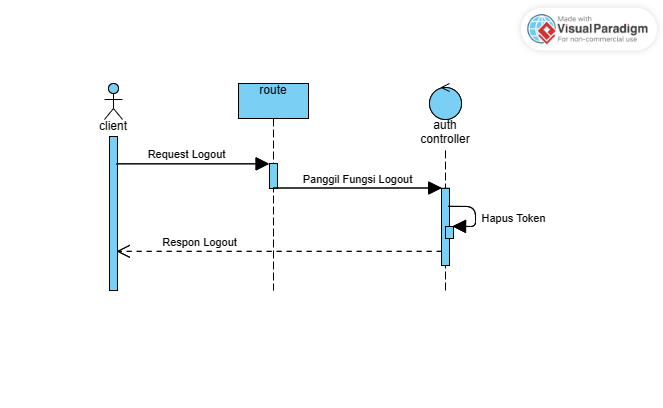
1. Login



Gambar 2.14 Sequence Diagram API Login

API *login* dimulai oleh *client*, *client* akan mengirimkan *username* dan *password* mereka melalui *endpoint* “/api/login” kemudian *route* akan memanggil fungsi *login* didalam kontroler, kontroler akan memeriksa *username* dan *password* dengan melakukan *query* ke tabel *users*, jika cocok maka kontroler akan membuat token menggunakan modul sanctum. Sanctum adalah sebuah paket autentikasi dan autorisasi yang disediakan oleh Laravel yang dirancang untuk memudahkan implementasi otentikasi API yang sederhana namun aman pada aplikasi Laravel. Setelah mendapatkan token kemudian kontroler akan mengembalikan token tersebut disertai dengan data *client* seperti nama, email, nomor hp dan lain sebagainya. Jika terdeteksi *client* belum malakukan verifikasi email maka kontroler akan membuat kode OTP atau *One* *Time* *Password* yang merupakan 6 digit angka acak dan mengirimkan kode terkesebut ke email *client*. Jika autentikasi yang dilakukan gagal, maka kontroler akan mengembalikan respon error ke *client*.

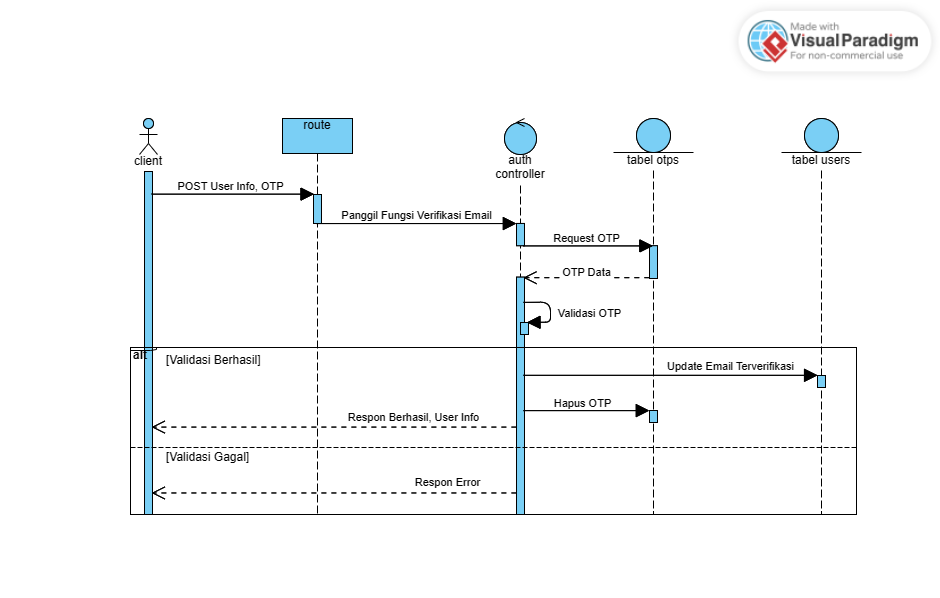
1. Logout



Gambar 2.15 Sequnce Diagram API Logout

Pada gambar 2.15 diatas, sebuah metode *logout* didalam kontroler akan dipanggil oleh *route* jika ada client yang melakukan *request* ke *endpoint* “/api/logout”. Kemudian kontroler akan menghapus token dari client sesuai dengan token yang dilampirkan didalam *header* pada saat *request* diterima. Proses ini sangat penting untuk menjaga keamanan sistem dan melindungi privasi pengguna. Dengan menghapus token saat klien keluar atau log out, sistem dapat memastikan bahwa akses tidak sah atau akses yang sudah tidak digunakan lagi tidak dapat digunakan kembali oleh pihak yang tidak berwenang.

1. Verifikasi email



Gambar 2.16 Sequence Diagram API Verifikasi Email

Pada gambar 2.16 diatas dapat dilihat bahwa pengguna melakukan verifikasi email dengan mengirimkan kode OTP yang sudah diterima melalui email disertai dengan detail client seperti id, nama dan email ke endpoint “/api/verify-email”, kemudian pada kontroler akan memeriksa kode yang diterima dengan kode yang tersimpan pada tabel otps, jika cocok dan masih aktif maka kontroler akan memperbarui status client menjadi terverifikasi dan menghapus kode otp yang lama kemudian mengembalikan respon berhasil. Jika kode salah atau sudah kadaluarsa maka kontroller akan mengembalikan respon error.

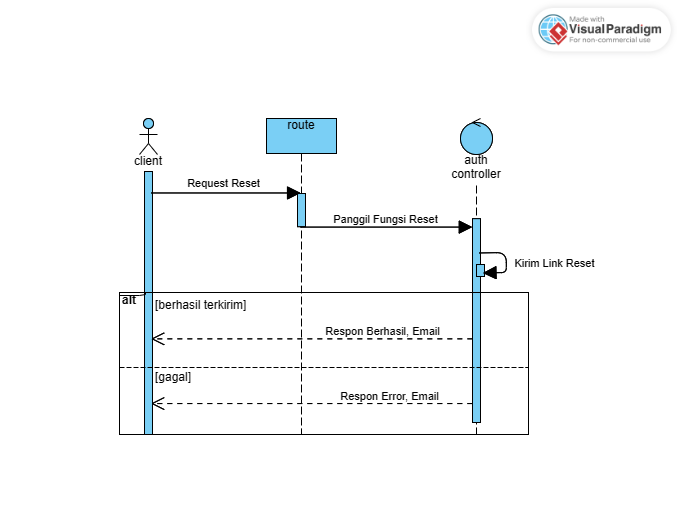
1. Ganti password



Gambar 2. 17 Sequence Diagram API Ganti Password

Dapat dilihat pada gambar 2.17 diatas, untuk mengganti *password* pertama client mengirimkan permintaan ganti password ke enpoint “/api/change-password” dengan mengirimkan *password* lama, *password* baru dan konfirmasi *password* baru, kemudian didalam kontroler *password* lama yang dikirimkan akan dicocokkan dengan *password* client sekarang dengan menggunakan fungsi Hash. Fungsi Hash merupakan sebuah fungsi yang disediakan oleh laravel yang digunakan untuk pengolahan data yang berkaitan dengan enkripsi. Jika kedua *password* cocok maka kontroler akan memperbarui *password* pada tabel users dan mengmbalikan respon berhasil, jika *password* tidak sesuai maka kontroler akan mengembalikan respon error.

1. Reset password

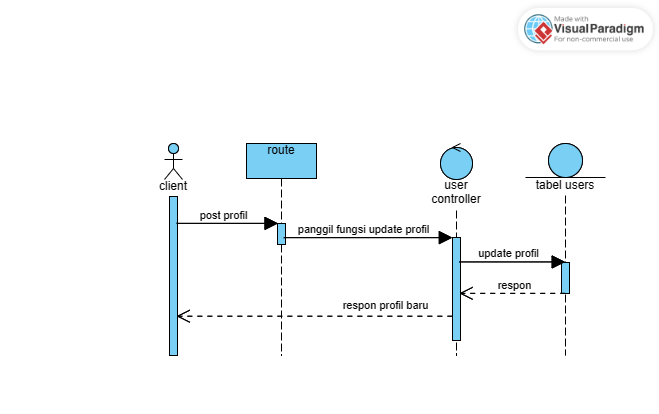


Gambar 2.18 Sequence Diagram API Reset Password

Pada gambar 2.18 di atas, proses reset *password* menggunakan metode yang terstruktur dan efisien. *Client* memulai proses ini dengan melakukan permintaan (*request*) ke *endpoint* "/api/reset-password" pada *server*. Permintaan ini berisi data email yang sudah terdaftar sebagai parameter supaya *server* dapat mengidentifikasi akun yang akan direset *password*-nya.

Setelah permintaan tersebut diterima maka kontroler pada sisi server akan mengeksekusi sebuah fungsi khusus yang telah disediakan oleh Laravel. Fungsi tersebut bertugas untuk mengirimkan *link* reset password ke alamat email yang diberikan oleh *client*. Proses pengiriman email ini dapat penggunaan layanan email eksternal seperti SMTP yang telah dikonfigurasi pada *server*. Jika email berhasil terkirim dengan sukses maka kontroler akan memberikan respon ke client berupa pesan berhasil. Pesan ini memberitahukan bahwa email reset *password* telah berhasil dikirim dan pengguna dapat segera memeriksa kotak masuk email mereka untuk melanjutkan proses selanjutnya. Namun, dalam beberapa situasi, pengiriman email mungkin mengalami kegagalan. Misalnya, alamat email yang diberikan tidak valid, terjadi gangguan jaringan, atau layanan email eksternal mengalami masalah. Jika hal ini terjadi, kontroler akan memberikan respon error ke client. Pesan error ini berisi informasi yang relevan tentang masalah yang terjadi, sehingga pengguna atau operator dapat mengetahui penyebabnya.

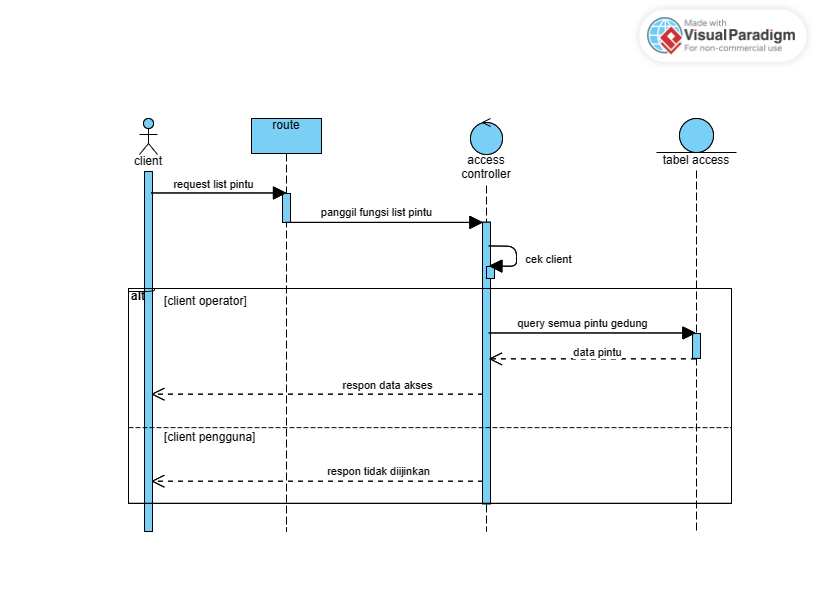
1. Ganti profil



Gambar 2.19 Sequence Diagram API Ganti Profil

Dapat dilihat pada gambar 2.19 diatas, unttuk mengganti profil pertama penguna atau operator mengirimkan data profil mereka ke enpoint “/api/update-profile” kemudian didalam kontroler data yang telah terima akan dimasukkan kedalam tabel *users* untuk memperbarui profil dan terakhir kontroler mengembalikan respon bahwa profil berhasil dirubah.

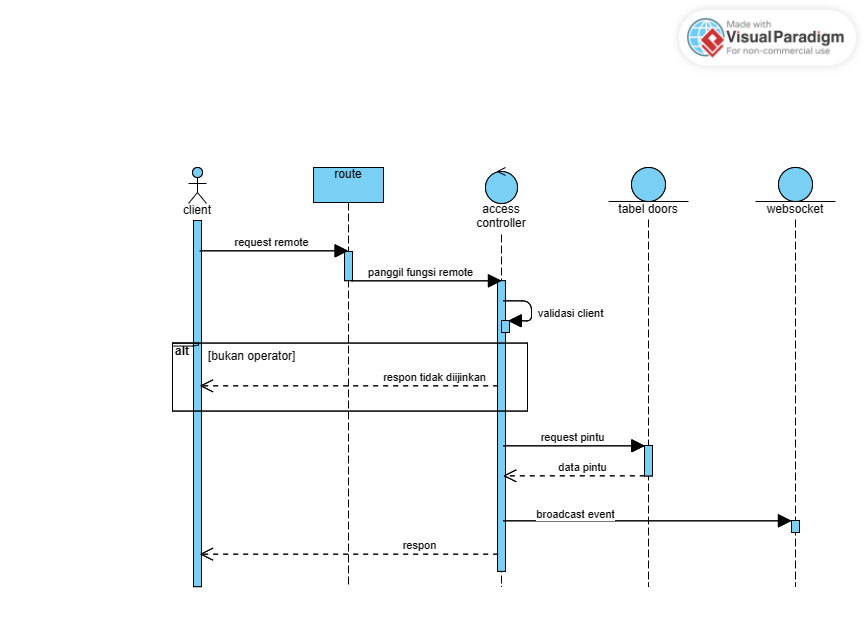
1. Daftar pintu



Gambar 2.20 Sequence Diagram API Daftar Pintu

Dapat dilihat pada gambar 2.20 diatas, untuk mendapatkan daftar pintu maka operator akan melalakukan request ke endpoint “/api/get-door” kemudian didalam kontroler dilakukan pengecekan untuk memastikan permintaan hanya berasal dari operator, jika permintaan berasal dari pengguna biasa maka kontroler akan mengembalikan respon tidak diijinkan. Selanjutnya kontroler akan mengambil data pintu pada tabel *doors* dan mengirimkan rebagai respon ke operator.

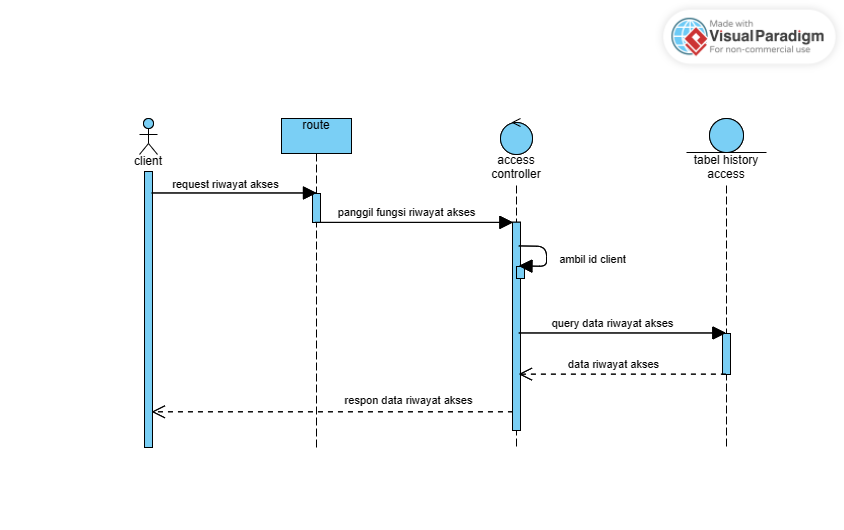
1. Remote pintu



Gambar 2.21 Sequence Diagram API Remote Pintu

Dapat dilihat pada gamabar 2.21 diatas, untuk melakukan *remote* pintu pertama operator akan melakukan *request* ke *endpoint* “/api/remote-access” dengan mengirimkan identitas dari pintu yang akan dikendalikan, kemudian kontroler akan memeriksa *client* untuk memastikan permintaan berasal dari operator, jika barasal dari pengguna biasa maka kontroler akan mengeembalikan respon tidak diijinkan. Selanjutnya kontroler akan mengambil data pintu untuk melengkapi informasi yang dibutuhkan seperti kode kunci, kode gedung, token dan lain sebagainya, kemudian data akan disiarkan ke perangkat kunci pintu melalui koneksi websocket yang sudah terhubung, terakhir kontroler akan mengembalikan respon remote pintu telah dilaksanakan.

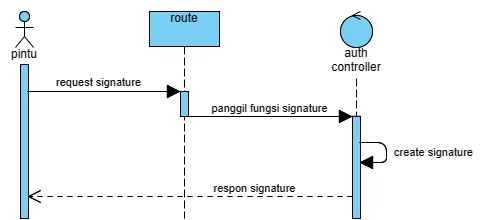
1. Riwayat akses



Gambar 2.22 Sequence Diagram API Riwayat Akses

Dapat dilihat pada gambar 2.22 untuk mendapatkan data riwayat akses maka pengguna akan melakukan permintaan ke *enpoint* “/api/my-history”. Kemudian kontroller akan mengambil data riwayat akses pengguna dengan melakukan *query* ke databse dengan menyertakan identitas pengguna sebagai parameter. Selanjutnya data yang sudah diperoleh seperti nama pintu, mana gedung, waktu aktifitas dan jenis aktifitas akan dikirimkan kembali ke pengguna sebagai respon untuk diolah pada tampilan aplikasi *mobile*.

1. Signature



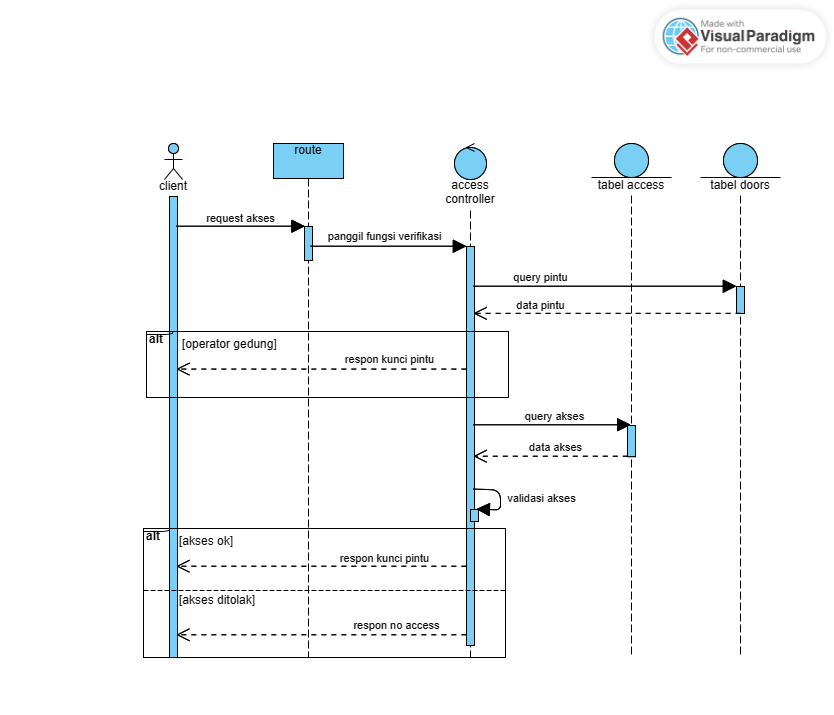
Gambar 2.23 Sequence Diagram API Door Signature

Dapat dilihat pada gamabr 3.20 diatas, untuk mendapatkan kode *signature* pusher pertama perangkat kunci pintu melakukan permintaan ke endpoint “/door/get-signature” dengan mengirimkan data-data seperti *socket*-id, *office*-id dan *channel*-data, dari data tersebut kemudian kontroler akan membuat kode *signature* menggunakan metode yang ada pada protokol pusher yaitu :

sign = hash\_hmac('sha256','socket:channel:data','secret\_key')

dimana ‘*secret*\_*key*’ merupakan kunci pusher yang terdapat pada server. Setelah mendapatkan nilai signature kemudian kontroler akan megembalikan respon kode *signature* ke perangkat kunci pintu.

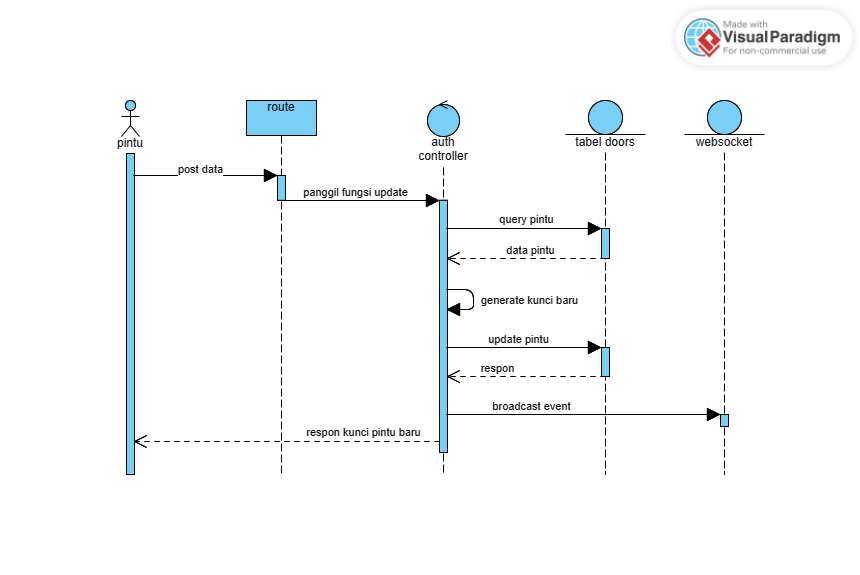
1. Verifikasi akses



Gambar 2.24 Sequence Diagram API Verifikasi Akses

Untuk mendapatkan akases ke pintu, setelah memindai kode QR pada pintu maka pengguna atau operator akan mendapatkan data informasi terkait pintu tersebut, kemudian data tersebut dikirimkan ke server melalui endpoint “/api/verify-access/{door-id}”, kemudian kontroler akan mengambil data pintu pada tabel *doors*, jika permintaan berasal dari operator gedung dimana pintu tersebut berada maka kontroler akan mengijinkan dengan mengambalikan respon berupa kode kunci pintu. Jika pengguna permintaan akses dilakukan oleh pengguna biasa maka kontroler akan memeriksa daftar akses didalam tabel *access*, jika pengguna memiliki akses dan masil berlaku maka kontroler akan mengembalikan respon akses diijinkan dan mengirimkan kode kunci untuk membuka pintu.

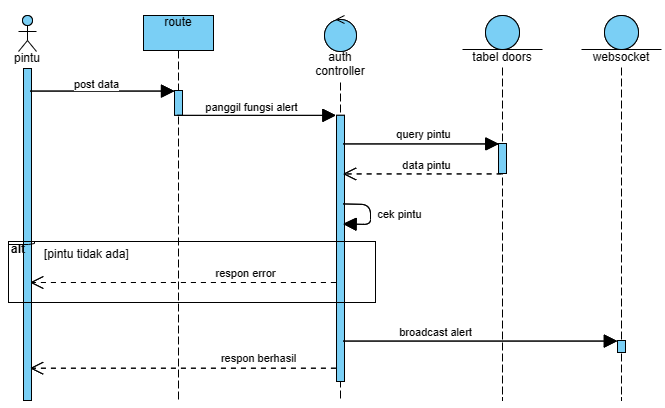
1. Update status pintu



Gambar 2.25 Sequence Diagram API Door Update Status

Terlihat pada gambar 2.25 diatas, untuk melakukan update status pertama perangkat kunci pintu akan mengirimkan data-data seperti status penguncian dan id *socket* melalui *endpoint* “/door/update-status”, kemudian kontroler akan mengambil data pintu yang terkait untuk diperbarui menggunakan data status dan kode kunci yang baru, kemudian kontroler juga akan melakukan *broadcasting* untuk menyiarkan bahwa status pintu berubah sehingga setiap informasi perubahan dapat tersampaikan secara langsung, terakhir kontroler akan mengembalikan respon update status telah dilaksanakan.

1. Peringatan pintu

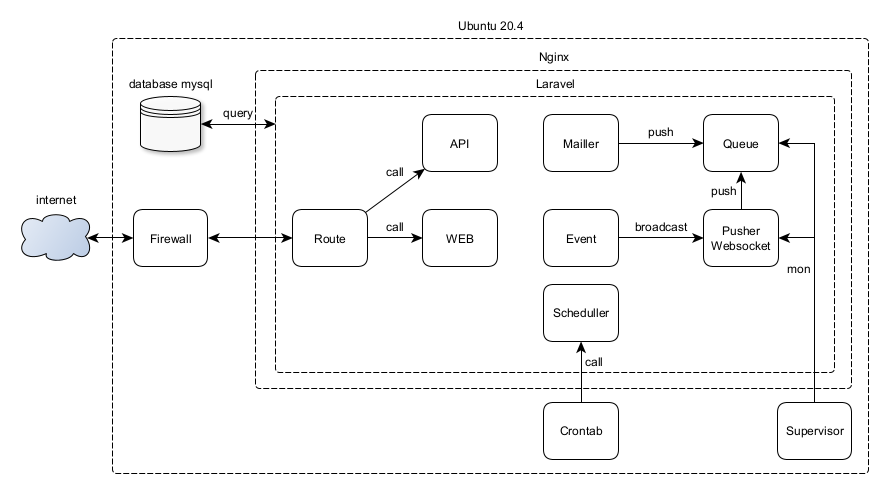


Gambar 2.26 Sequence Diagram API Door Alert

Dapat dilihat pada gambar 2.26 diatas, untuk memberikan peringatan pertama perangkat kunci pintu akan mengirimakan data-data seperti id-pintu, id-*office* dan status peringatan melalui *endpoint* “/door/alert” kemudian kontroler akan memeriksa pada tabel pintu untuk memastika bahwa pintu valid, jika pintu tudak ditemuakan maka kontroler akan mengembalikan respon error. Selanjutnya kontroler akan menyiarkan peringatan melalui websocket, terakhir kontroler akan mengembalikan respon peringatan sudah dilaksanakan.

1. Server

Dengan menggunakan Laravel sebagai backend yang mengatur kinerja dari perangkat kunci pintu tentunya diperlukan sebuah *server*. *Server* ini akan bertindak sebagai pusat pengolahan data dan berfungsi untuk menerima permintaan dari perangkat kunci pintu, mengatur akses, memproses logika bisnis, dan berkomunikasi dengan *database*. Diagram dari *backend* *server* dapat dilihat pada gambar 2.27 dibawah.



Gambar 2.27 Konfigurasi Server

Dapat dilihat pada gambar 3.26 diatas, *server* dibangun menggunakan sistem operasi ubuntu 20.04, ubuntu merupakan bagian dari sistem operasi linux yang biasa digunakan baik untuk perangkat desktop maupun *server* karena *open* *source* dan ringan. Dengan menggunakan Ubuntu 20.04 sebagai sistem operasi server, kita dapat memanfaatkan kestabilan, keamanan, dan dukungan jangka panjang untuk menjalankan aplikasi laravel dengan aman dan efisien.

Didalam sistem operasi ubuntu 20.04 dipasang nginx yang digunakan sebagai web *server* untuk menjalankan aplikasi laravel. Dengan menggunakan nginx maka aplikasi laravel dapat dijalankan dengan efisien dikarenakan nginx memiliki karakteristik ringan dan cepat. Dengan menggunakan nginx kita juga bisa membagi server menjadi beberapa blok yang dapat digunakan untuk menjalankan API dan websocket secara bersamaan. Pada nginx juga dipasang sertifikat SSL yang digunakan untuk mengenkripsi semua komunikasi dengan menggunakan HTTPS sehingga kemanan data akan terjamin dengan adanya jalur komunikasi yang terenkripsi.

Didalam ubuntu juga dipasang MySQL sebagai pusat penyimpanan data yang terhubung ke laravel, dengan menggunakan database yang berjalan pada server yang sama maka kecepatan transfer data akan sangat cepat dengan menghilangkan latensi jaringan hal ini sesuai dengan karakteristik sistem yang dibangun yaitu sistem cepat dan efisien.

Dengan adanya fitur penjadwalan otomatis didaiam sistem kunci pintu maka kita juga harus memasang crontab untuk menjalankan penjadwalan yang ada pada laravel, penjadwalan akan dipanggil setiap 1 menit sekali untuk memeriksa apakah ada jadwal yang harus dilaksanakan atau tidak.

Pada laravel juga mengimplementasikan fitur antrian yang bertujuan untuk meningkatkan kinerja dasi sistem sehingga fungsi antrian harus dijaga agar selalu berada dalam kondisi berjalan, oleh karena itu maka pada server suja dipasang supervisor yang digunakan untuk memantau kinerja dari antrian. Supervisor merupakan sebuah sistem pengawas atau *process* *control* *system* yang digunakan untuk mengontrol dan mengelola proses-proses yang berjalan didalam sistem operasi. Supervisor akan memastikan bahwa proses antrian pada laravel tetap berjalan secara terus-menerus dan akan memulai ulang proses jika terjadi kegagalan serta mengelola jumlah pekerja antrian atau *queue* *workers* yang berjalan secara paralel untuk meningkatkan efisiensi dan kinerja.

Supervisor juga digunakan untuk menjaga websocket berjalan secara terus menerus, websocket memegang peranan penting didalam sistem ini karena websocket menjadi jalur komunikasi yang menghubungkan perangkat kunci pintu dengan *server* pusat, sehingga jika terjadi gangguan pada kinerja websocket maka seluruh kinerja dari perangkat kunci pintu akan terganggu, oleh karena itu diperlukan pengawasan menggunakan supervisor untuk menjaga kinerja dari websocket.

1. Tampilan Website

Implementasi tampilan website menggunakan library livewire sehingga memungkinkan tampilan yang interaktif. Beberapa tampilan yang tersedia pada website adalah sebagai berikut :

1. Autentikasi
2. Profil
3. Pengelolaan pintu
4. Pengelolaan pengguna
5. Pengelolaan penjadwalan
6. Riwayat akses
7. Aplikasi Mobile

Implementasi aplikasi mobile dilakukan dengan menggunakan kerangka kerja Flutter dengan menggunakan bahasa Dart. Beberapa metode yang diimplementasikan pada aplikasi mobile adalah sebagai berikut :

1. Autentikasi
2. Penampilan data dari server
3. Pemindaian kode QR
4. Pengiriman data melalui bluetooth
5. Pengiriman data ke server
6. PENUTUP

Dokumen B400 memaparkan proses implementasi yang dilakukan untuk membangun sistem keamanan kunci pintu gedung berbasi internet of things. Proses implementasi akan menentukan hasil akhir dari sistem yang dikembangkan, hasil implementasi juga akan dijadikan acuan untuk proses pengujian sistem.