




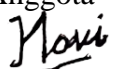



UNIVERSITAS DIPONEGORO – FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRO

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 Gedung Berbasis <i>Internet of Things</i>
Jenis Dokumen	IMPLEMENTASI <small>Catatan: Dokumen ini dikendalikan penyebarannya oleh Dept. Teknik Elektro Undip</small>
Nomor Dokumen	B400-01-TA2223.2.19012
Nomor Revisi	01
Nama File	B400-2-TA2223
Tanggal Penerbitan	8 September 2023
Unit Penerbit	Departemen Teknik Elektro Undip
Jumlah Halaman	67 (termasuk lembar sampul ini)

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

DAFTAR ISI

1.	PENDAHULUAN	4
1.1	Ringkasan Isi Dokumen.....	4
1.2	Aplikasi Dokumen	4
1.3	Daftar Singkatan	4
2.	IMPLEMENTASI.....	5
2.1	Perangkat Keras Kunci Pintu.....	5
2.2	Perangkat Lunak Kunci Pintu	10
2.3	<i>Database</i>	21
2.4	<i>Backend API</i>	26
2.5	<i>Server</i>	35
2.6	Tampilan <i>Website</i>	37
2.7	Aplikasi <i>Mobile</i>	52
3.	PENUTUP	67

Catatan Sejarah Perbaikan Dokumen

VERSI, TGL, OLEH	PERBAIKAN
01, 0 September 2023, oleh Henric Dhiki Wicaksono, Novi Dianasari, dan Muhammad Khoiril Wafi.	<i>Draft</i> Dokumen B400

1. PENDAHULUAN

1.1 Ringkasan Isi Dokumen

Dokumen ini berisi 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 kunci pintu berupa *hardware* dan *software*-nya, pembuatan tampilan baik itu berupa *website* maupun aplikasi *mobile*, serta pembuatan *database* dan *backend server*. Proses implementasi yang dilakukan akan mengikuti desain yang telah dibuat dan dijelaskan pada dokumen desain (B300).

1.2 Aplikasi Dokumen

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

- 1) Sebagai penjelasan proses implementasi dari sistem yang telah 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.

1.3 Daftar Singkatan

Tabel 1.1 Daftar singkatan

SINGKATAN	ARTI
IoT	<i>Internet of Things</i>
WiFi	<i>Wireless Fidelity</i>
JSON	<i>Javascript Object Notation</i>
HTTPS	<i>Hypertext Transfer Protocol Secure</i>
PC	<i>Personal Computer</i>
IDE	<i>Integrated Development Environment</i>
UI	<i>User Interface</i>

Tabel 1.1 (lanjutan)

SINGKATAN	ARTI
QR-Code	<i>Quick Response Code</i>
SSL	<i>Secure Sockets Layer</i>
ESP	<i>Espressif</i>
LED	<i>Light Emitting Diode</i>
SSID	<i>Service Set Identifier</i>
EEPROM	<i>Electrically Erasable Programmable Read-Only Memory</i>
MySQL	<i>My Structured Query Language</i>
HP	<i>Handphone</i>
API	<i>Application Programming Interface</i>
PHP	<i>Hypertext Preprocessor</i>
OTP	<i>One Time Password</i>
SMTP	<i>Simple Mail Transfer Protocol</i>

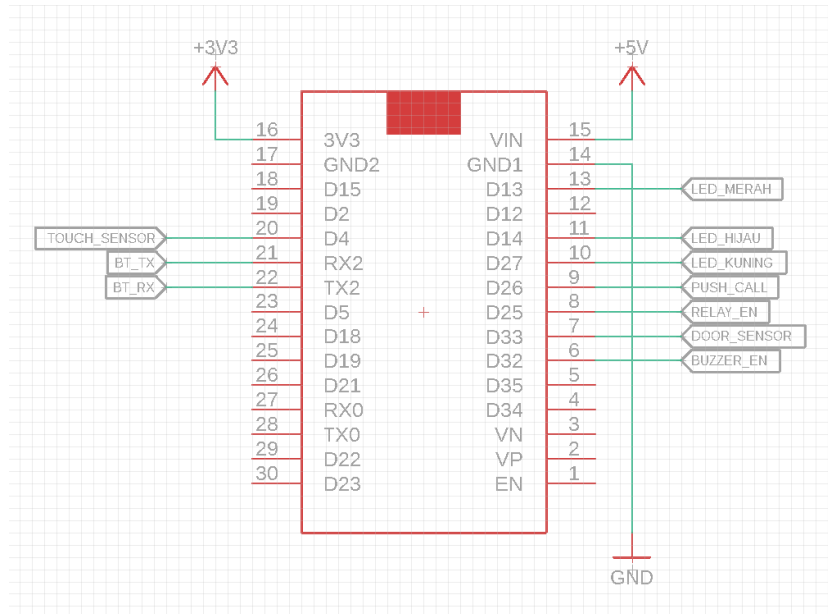
2. IMPLEMENTASI

2.1 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 aktivitas pada perangkat kunci pintu. Pada proses desain dijelaskan penggunaan ESP32 sebagai kontroler utama dengan memanfaatkan modul komunikasi WiFi yang sudah tersedia di dalamnya.



Gambar 2.1 Konfigurasi ESP32

Gambar 2.1 di atas memperlihatkan penggunaan ESP32 sebagai kontroler utama pada perangkat kunci pintu. ESP32 akan mengatur semua *input* dan *output* pada perangkat kunci pintu dengan penjelasan sesuai dengan Tabel 2.1 di bawah ini.

Tabel 2.1 Konfigurasi pin ESP32

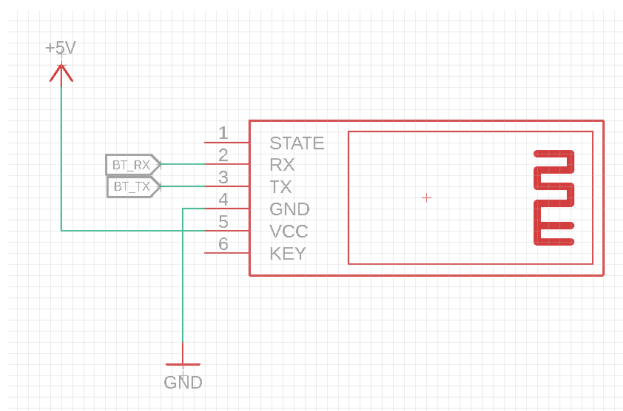
Pin	Mode	Keterangan
D4	<i>Input</i>	Digunakan sebagai sensor sentuh untuk mendeteksi interaksi pengguna.
RX2	<i>Input</i>	Terhubung dengan modul <i>bluetooth</i> untuk menerima data.
TX2	<i>Output</i>	Terhubung dengan modul <i>bluetooth</i> untuk mengirim data.
D13	<i>Output</i>	Terhubung dengan LED indikator status penguncian.
D14	<i>Output</i>	Terhubung dengan LED indikator proses pengiriman data.
D27	<i>Output</i>	Terhubung dengan LED indikator status koneksi WiFi.
D26	<i>Input</i>	Terhubung dengan tombol untuk membuka pintu dari dalam ruangan.
D25	<i>Output</i>	Terhubung dengan modul <i>relay</i> untuk menggerakkan solenoid kunci pintu.

Tabel 2.1 (lanjutan)

Pin	Mode	Keterangan
D33	<i>Input</i>	Terhubung dengan saklar magnetik untuk membaca kondisi pintu.
D32	<i>Output</i>	Terhubung dengan <i>buzzer</i> untuk memberikan suara <i>alarm</i> peringatan.
Vin	<i>Power</i>	Terhubung dengan sumber catu daya.

2. *Bluetooth* HC-05

Modul HC-05 digunakan untuk melakukan komunikasi dengan aplikasi *mobile* menggunakan koneksi *bluetooth*. ESP32 sudah dilengkapi dengan modul *bluetooth*, tetapi pada implementasinya ESP32 tidak bisa menjalankan modul WiFi dan *bluetooth* secara bersamaan dikarenakan keduanya terhubung ke satu modul radio yang sama dan juga keterbatasan *memory* yang dimiliki.



Gambar 2.2 Konfigurasi *bluetooth* HC-05

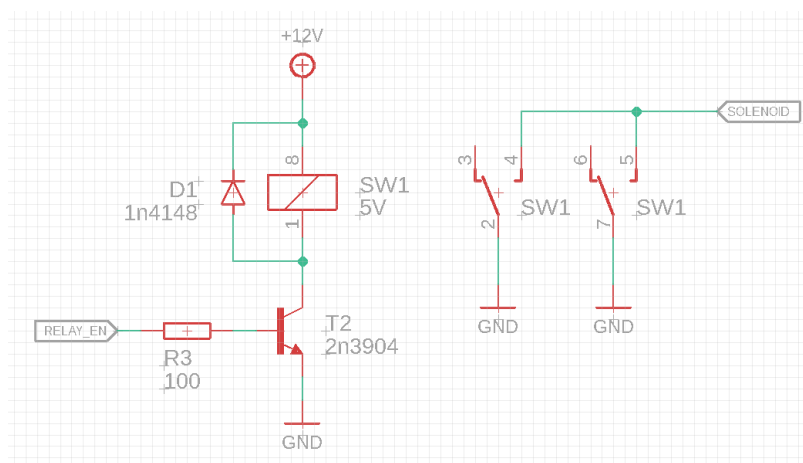
Pada Gambar 2.2 di atas terlihat penggunaan modul *bluetooth* HC-05 pada perangkat kunci pintu. Modul *bluetooth* HC-05 menggunakan komunikasi serial untuk berkomunikasi dengan mikrokontroler dengan *bautrate* yang telah ditentukan. Pada modul *bluetooth* terdapat 4 pin yang digunakan sesuai dengan penjelasan pada Tabel 2.2 di bawah ini.

Tabel 2.2 Konfigurasi pin *bluetooth* HC-05

Pin	Mode	Keterangan
VCC	Power	Terhubung ke sumber catu daya untuk menghidupkan modul <i>bluetooth</i> .
GND	Power	Referensi 0 volt
TX	Output	Terhubung ke mikrokontroler untuk mengirimkan data.
RX	Input	Terhubung ke mikrokontroler untuk menerima data.

3. Solenoid

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

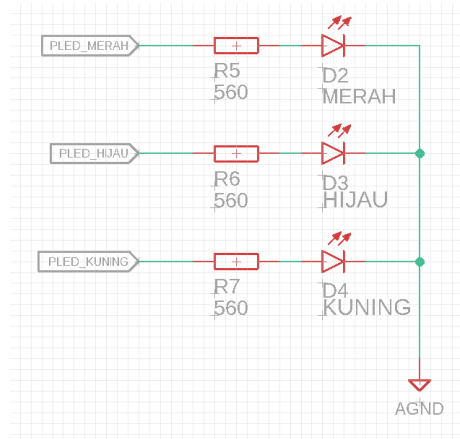


Gambar 2.3 Konfigurasi solenoid

Pada Gambar 2.3 di atas, untuk menggerakkan solenoid maka mikrokontroler memerlukan bantuan dari sebuah modul *relay*. Hal tersebut dikarenakan mikrokontroler hanya mampu mengalirkan arus beberapa mA saja sehingga tidak mampu untuk menggerakkan solenoid secara langsung.

4. LED

LED digunakan untuk memberikan informasi tentang kondisi perangkat kunci pintu.

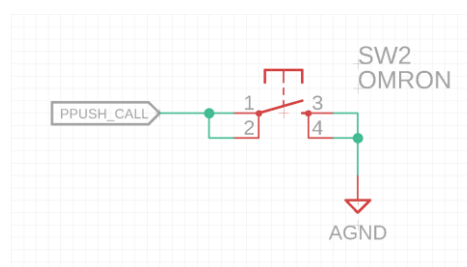


Gambar 2.4 Konfigurasi LED

Pada Gambar 2.4 di atas, perangkat kunci pintu memiliki 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.

5. Tombol

Karena solenoid tidak dapat 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 ruangan.

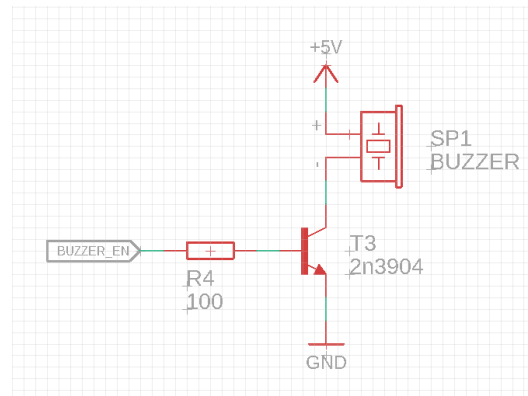


Gambar 2.5 Konfigurasi tombol

Pada Gambar 2.5 di atas terlihat bahwa 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.

6. 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*

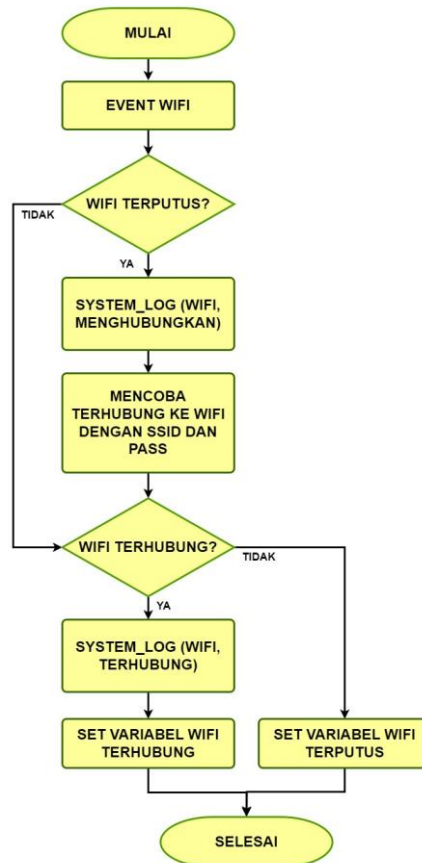
Pada Gambar 2.6 di atas, penggunaan *buzzer* pada perangkat kunci pintu ini dengan menggunakan sebuah transistor untuk membantu mikrokontroler dalam mengaktifkan *buzzer*. Hal tersebut dikarenakan arus yang dikeluarkan mikrokontroler tidak cukup untuk menyalakan *buzzer* secara maksimal.

2.2 Perangkat Lunak Kunci Pintu

Implementasi perangkat lunak pada perangkat kunci pintu ini dilakukan dengan menggunakan Arduino. Arduino merupakan sebuah lingkungan pengembangan program mikrokontroler dengan menggunakan bahasa C++ yang sudah dipermudah. Beberapa metode yang terdapat pada perangkat kunci pintu ini 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, perangkat penguncian mampu beradaptasi seperti melakukan koneksi ulang ke jaringan WiFi saat terjadi masalah.

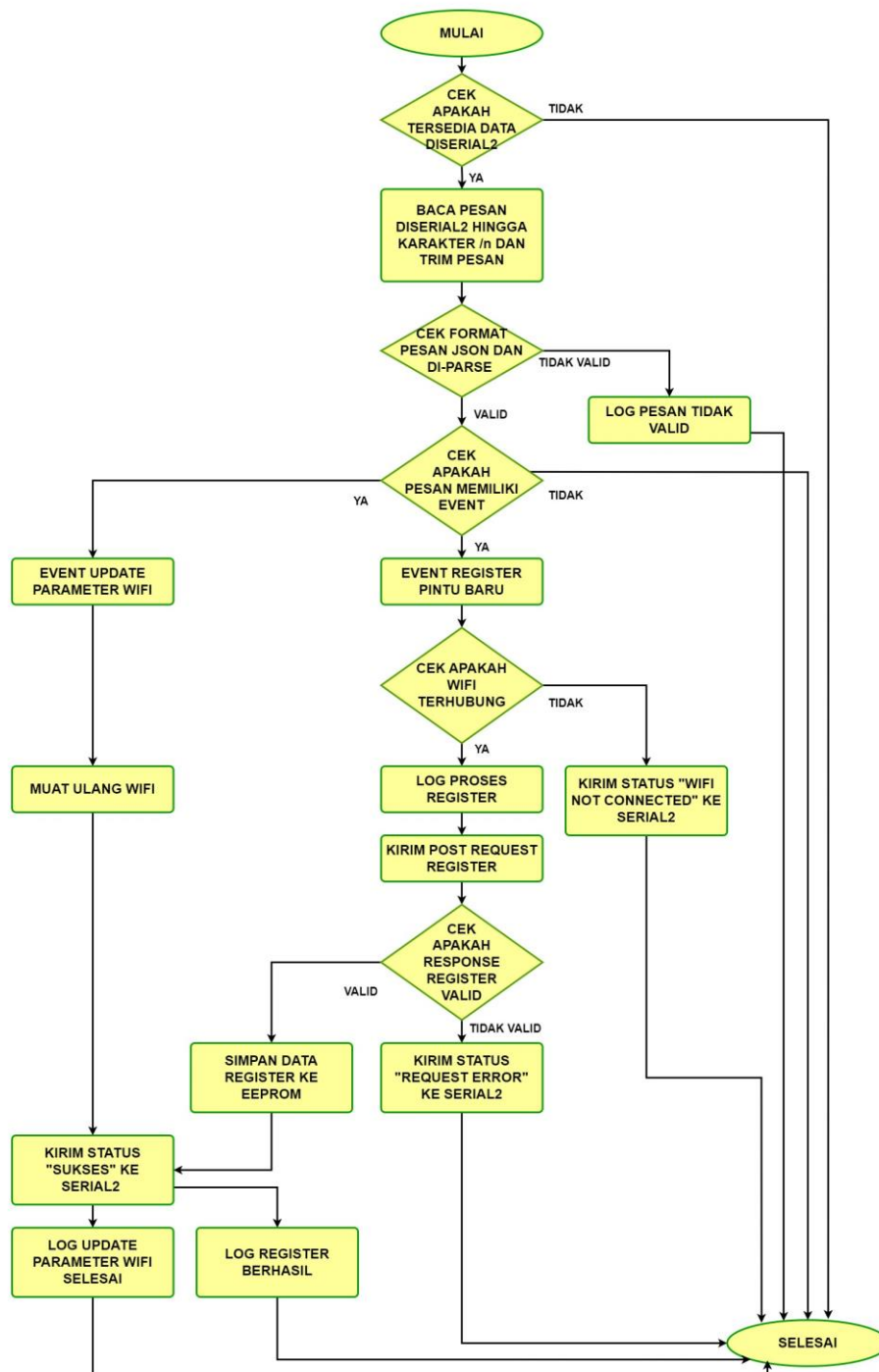


Gambar 2.7 Flowchart koneksi WiFi

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 memperbarui status koneksi WiFi menggunakan sebuah variabel. Variabel tersebut digunakan untuk menyimpan kondisi WiFi sehingga diketahui secara global. Hal tersebut untuk memastikan bahwa WiFi terhubung sebelum berkomunikasi dengan *server*. Pada saat kondisi WiFi terputus maka variabel status akan diperbarui dan program akan mencoba menghubungkan ulang. Pada saat kondisi WiFi baru saja terhubung maka status koneksi juga akan diperbarui.

2. Pengaturan perangkat kunci pintu

Proses pengaturan kunci pintu seperti mengatur kredensial WiFi serta mendaftarkan perangkat baru ke *server* dilakukan dengan menggunakan aplikasi *mobile* melalui koneksi *bluetooth*.



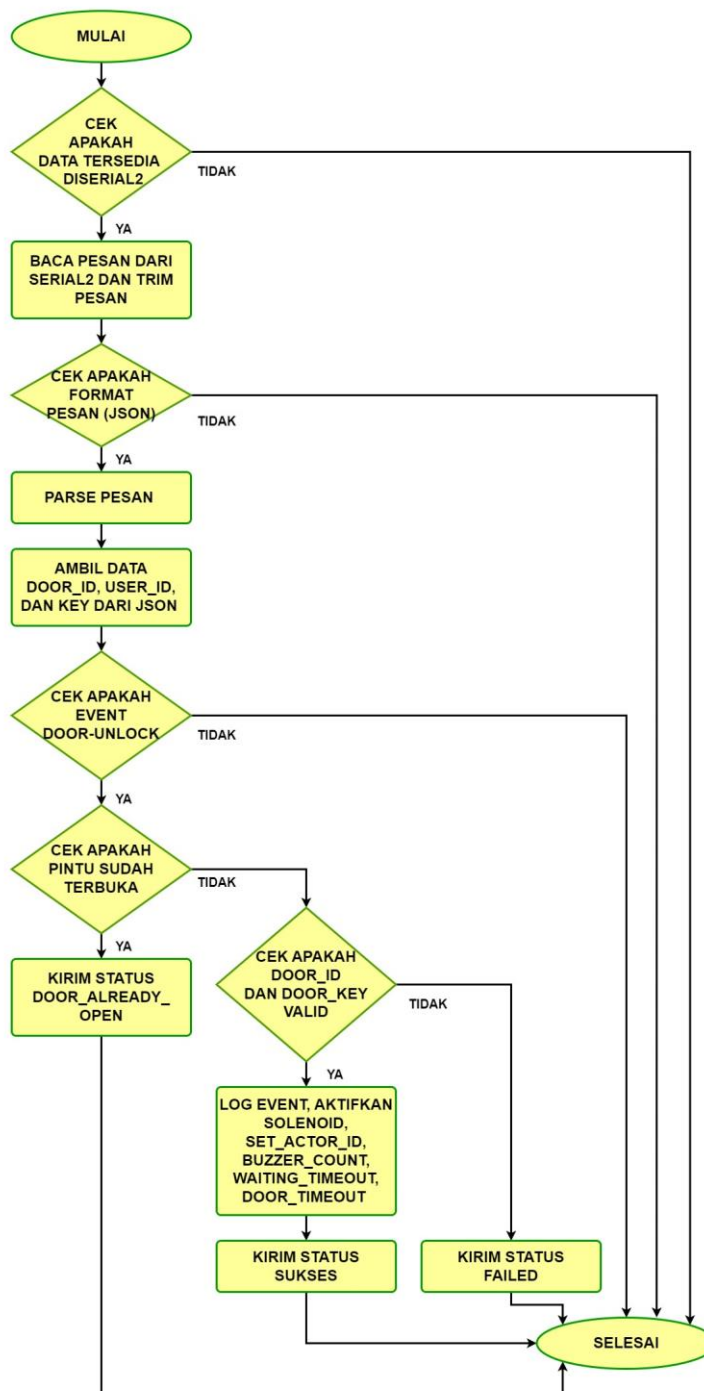
Gambar 2.8 Flowchart fungsi `config_loop()`

Berdasarkan pada Gambar 2.8 di atas, 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"` melalui 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*.

3. Perintah penguncian menggunakan *bluetooth*

Untuk membuka kunci pintu, setiap pengguna akan menggunakan aplikasi *mobile* untuk berkomunikasi dengan perangkat kunci pintu. Pada kondisi tersebut, 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 di bawah ini.



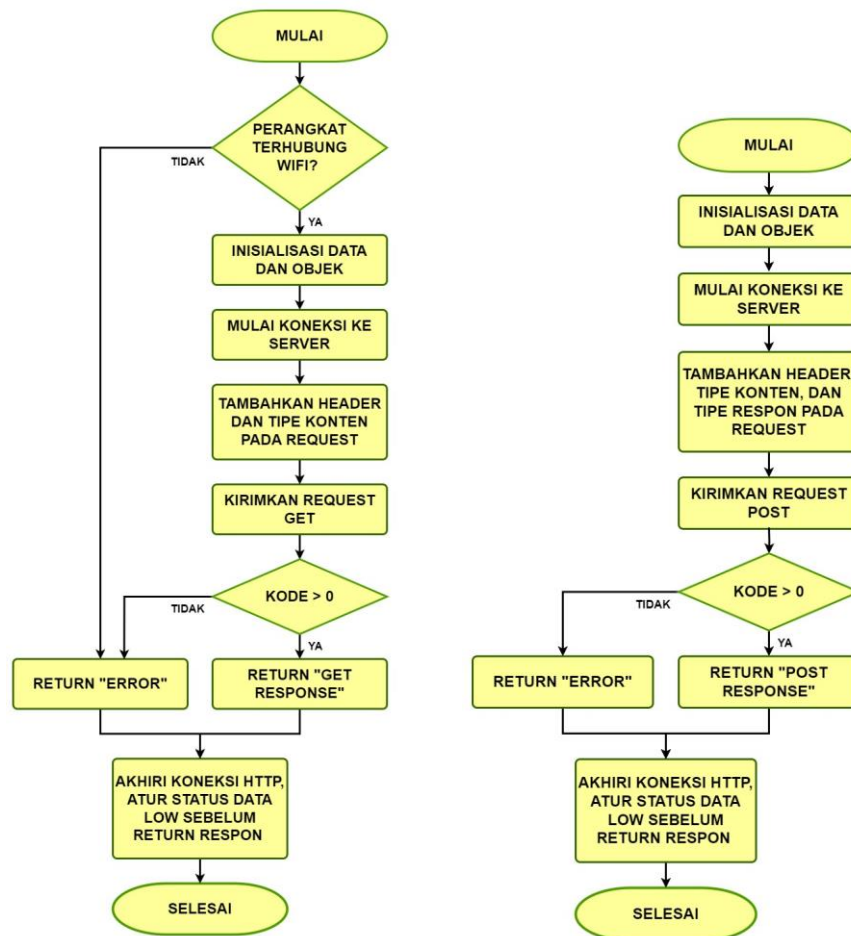
Gambar 2.9 Flowchart fungsi *bt_door_command()*

Berdasarkan pada Gambar 2.9 di atas, 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.

4. Komunikasi dengan *server*

Perangkat kunci pintu akan mengirimkan informasi terkait dengan kondisi pintu dan melakukan autentikasi untuk terhubung ke sistem penguncian. Implementasi komunikasinya menggunakan protokol HTTPS dengan proses enkripsi SSL 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*.

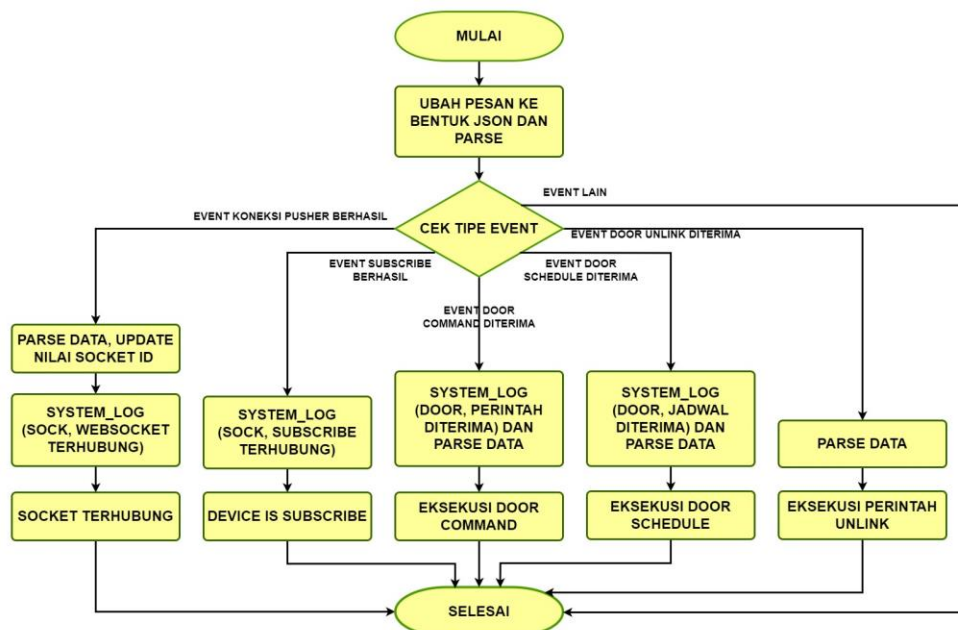
Fungsi `get_request(String endpoint)` digunakan untuk melakukan permintaan HTTP GET ke *endpoint* yang ditentukan pada *server*. Fungsi tersebut memeriksa apakah koneksi WiFi tersedia (`wifi_is_connected`). Jika tidak tersedia, maka fungsi tersebut akan mengembalikan *string* "error". Fungsi tersebut juga melakukan inisialisasi objek *HttpClient* (`http_get`) dan variabel *String* (`get_response`) untuk menyimpan respons 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 respons yang diharapkan (`Accept: application/json`). Permintaan GET

dikirim menggunakan `http_get.GET()` yang mengembalikan kode respon HTTP. Jika kode respons lebih besar dari 0, menandakan permintaan berhasil, fungsi tersebut membaca isi respons dari *server* ke dalam variabel `get_response` menggunakan `http_get.getString()`. Namun, jika permintaan gagal (kode respons ≤ 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 respons 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.

5. 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.

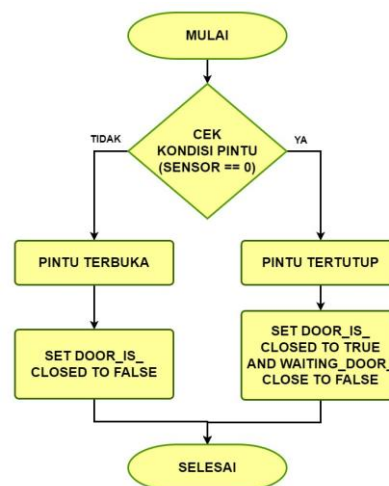
Jika *event* adalah "door-schedule", itu berarti pesan "door-schedule" telah diterima. Fungsi `websocket_message` memanggil fungsi `system_log` dengan parameter "DOOR" dan "jadwal diterima" yang menandakan "schedule received", mengurai/mem-*parse field* "data" dari pesan yang merupakan *string* format 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.

6. 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 di bawah ini.



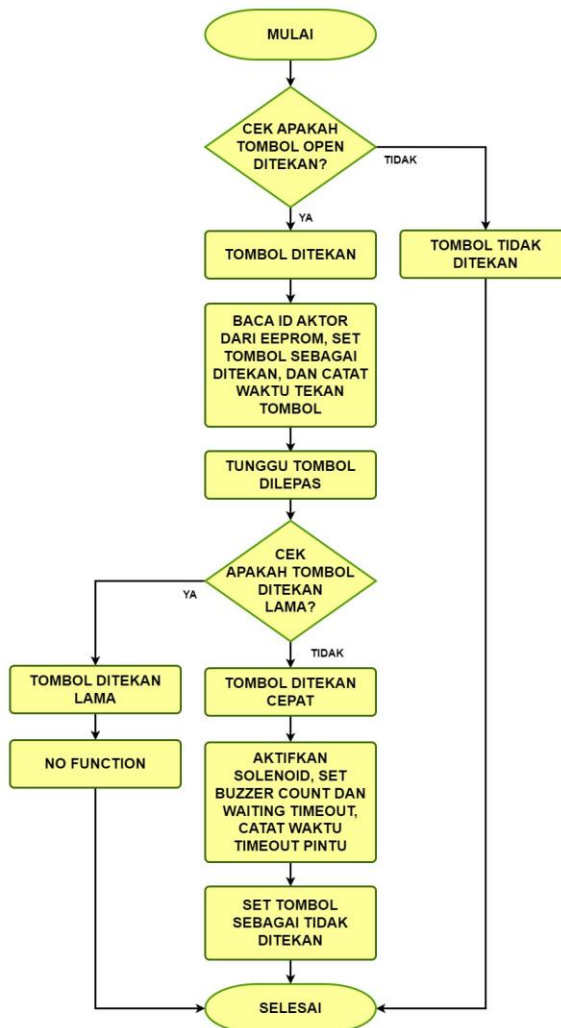
Gambar 2.12 Flowchart pembacaan kondisi pintu

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.

7. 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 menggunakan tombol dapat dilihat pada Gambar 2.13 di bawah ini.



Gambar 2.13 Flowchart penanganan interaksi pengguna menggunakan tombol

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.

2.3 Database

Implementasi dilakukan dengan membuat sebuah *database* menggunakan MySQL sesuai dengan diagram yang telah ditentukan. *Database* 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 diubah dengan cara mengubah kode migrasi. Struktur dari setiap tabel dapat dilihat pada Tabel 2.3 sampai Tabel 2.9 di bawah ini.

Tabel 2.3 Struktur tabel *users*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>added_by</i>	<i>char(36)</i>	referensi <i>actor</i>
3	<i>Email</i>	<i>varchar(255)</i>	alamat email pengguna
4	<i>Password</i>	<i>varchar(255)</i>	<i>password</i> pengguna
5	<i>remember_token</i>	<i>varchar(255)</i>	token untuk fitur "ingat saya"
6	<i>Name</i>	<i>varchar(255)</i>	nama pengguna
7	<i>Phone</i>	<i>varchar(255)</i>	nomor hp pengguna
8	<i>Gender</i>	<i>Enum</i>	jenis kelamin pengguna
9	<i>Role</i>	<i>Enum</i>	jabatan pengguna
10	<i>Foto</i>	<i>varchar(255)</i>	foto profil pengguna
11	<i>email_verified_at</i>	<i>Timestamp</i>	waktu verifikasi email
12	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
13	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.4 Struktur tabel *offices*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>user_id</i>	<i>char(36)</i>	referensi operator gedung
3	<i>Name</i>	<i>varchar(255)</i>	nama gedung
4	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
5	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.5 Struktur tabel *doors*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>office_id</i>	<i>char(36)</i>	referensi gedung
3	<i>socket_id</i>	<i>char(36)</i>	identitas <i>websocket</i>
4	<i>Name</i>	<i>varchar(36)</i>	nama pintu
5	<i>device_name</i>	<i>varchar(36)</i>	<i>username</i> pintu
6	<i>device_pass</i>	<i>varchar(36)</i>	<i>password</i> pintu
7	<i>Key</i>	<i>varchar(36)</i>	kunci pintu
8	<i>is_lock</i>	<i>tinyint(1)</i>	status penguncian
9	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
10	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.6 Struktur tabel *access*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>user_id</i>	<i>char(36)</i>	referensi ke pengguna
3	<i>door_id</i>	<i>char(36)</i>	referensi ke pintu
4	<i>time_begin</i>	<i>Time</i>	waktu mulai
5	<i>time_end</i>	<i>Time</i>	waktu berakhir
6	<i>date_begin</i>	<i>Date</i>	tanggal mulai
7	<i>date_end</i>	<i>Date</i>	tanggal berakhir
8	<i>is_temporary</i>	<i>tinyint(1)</i>	status akses sementara
9	<i>is_running</i>	<i>tinyint(1)</i>	status akses sedang berjalan
10	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat

Tabel 2.6 (lanjutan)

No	Nama Kolom	Tipe Data	Keterangan
11	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.7 Struktur tabel *schedules*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>office_id</i>	<i>char(36)</i>	referensi gedung
3	<i>Name</i>	<i>varchar(255)</i>	nama jadwal
4	<i>date_begin</i>	<i>Date</i>	tanggal mulai
5	<i>date_end</i>	<i>Date</i>	tanggal berakhir
6	<i>time_begin</i>	<i>Time</i>	waktu mulai
7	<i>time_end</i>	<i>Time</i>	waktu berakhir
8	<i>is_repeating</i>	<i>tinyint(1)</i>	status perulangan
9	<i>day_repeating</i>	<i>varchar(255)</i>	perulangan hari
10	<i>Status</i>	<i>Enum</i>	status pelaksanaan jadwal
11	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
12	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.8 Struktur Tabel *Door Schedule*

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>schedule_id</i>	<i>char(36)</i>	referensi ke jadwal
3	<i>door_id</i>	<i>char(36)</i>	referensi ke pintu
4	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
5	<i>updated_at</i>	<i>Timestamp</i>	waktu data diedit

Tabel 2.9 Struktur tabel riwayat akses

No	Nama Kolom	Tipe Data	Keterangan
1	<i>Id</i>	<i>char(36)</i>	kunci primer tabel
2	<i>user_id</i>	<i>char(36)</i>	referensi ke pengguna
3	<i>office_id</i>	<i>char(36)</i>	referensi ke gedung
4	<i>door_id</i>	<i>char(36)</i>	referensi ke pintu
5	<i>Log</i>	<i>Text</i>	pesan log
6	<i>created_at</i>	<i>Timestamp</i>	waktu data dibuat
7	<i>updated_at</i>	<i>Timestamp</i>	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 di bawah ini.

Tabel 2.10 Relasi tabel

No	Tabel Induk	Anak		Jenis Hubungan	On Update	On Delete
		Tabel	Kolom			
1	<i>Users</i>	<i>Users</i>	<i>added_by</i>	<i>one to many</i>	<i>cascade</i>	<i>restrict</i>
2	<i>Users</i>	<i>Access</i>	<i>user_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
3	<i>Doors</i>	<i>Access</i>	<i>door_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
4	<i>Users</i>	<i>access_logs</i>	<i>user_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
5	<i>Doors</i>	<i>access_logs</i>	<i>door_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
6	<i>Offices</i>	<i>access_logs</i>	<i>office_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
7	<i>Offices</i>	<i>Doors</i>	<i>office_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>
8	<i>Schedules</i>	<i>door_schedule</i>	<i>schedule_id</i>	<i>one to many</i>	<i>cascade</i>	<i>cascade</i>

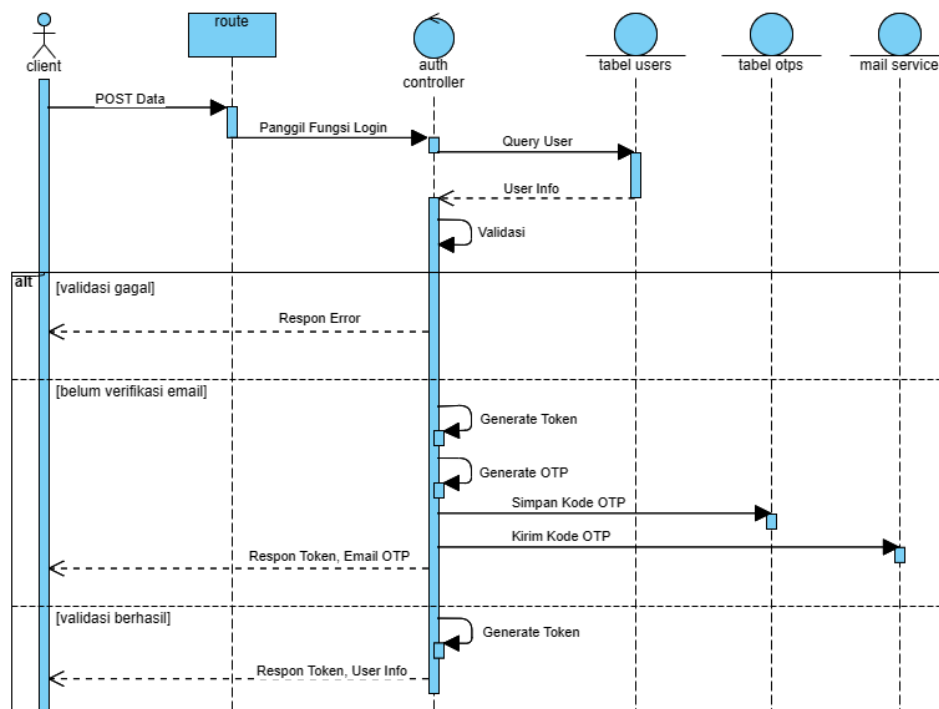
Tabel 2.10 (lanjutan)

No	Tabel Induk	Anak		Jenis Hubungan	On Update	On Delete
		Tabel	Kolom			
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

2.4 Backend API

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

1. Login

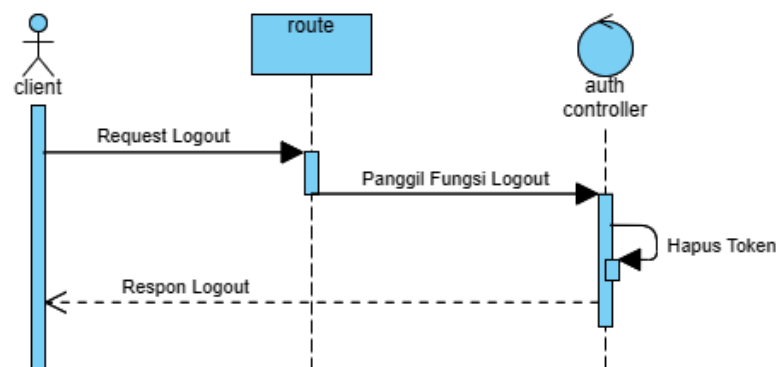


Gambar 2.14 Sequence diagram API login

Pada Gambar 2.14 di atas API *login* dimulai oleh *client*, *client* akan mengirimkan *username* dan *password* melalui *endpoint* “/api/login”. Kemudian, *route*

memanggil fungsi *login* di dalam 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 otorisasi yang disediakan oleh laravel yang dirancang untuk memudahkan implementasi autentikasi API yang sederhana tetapi aman pada aplikasi laravel. Setelah mendapatkan token, kontroler akan mengembalikan token tersebut disertai dengan data *client* seperti nama, email, nomor hp, dan lain sebagainya. Jika *client* terdeteksi belum melakukan verifikasi email maka kontroler membuat kode OTP atau *One Time Password* berupa 6 digit angka acak dan mengirimkan kode tersebut ke email *client*. Jika autentikasi yang dilakukan gagal maka kontroler akan mengembalikan respons *error* ke *client*.

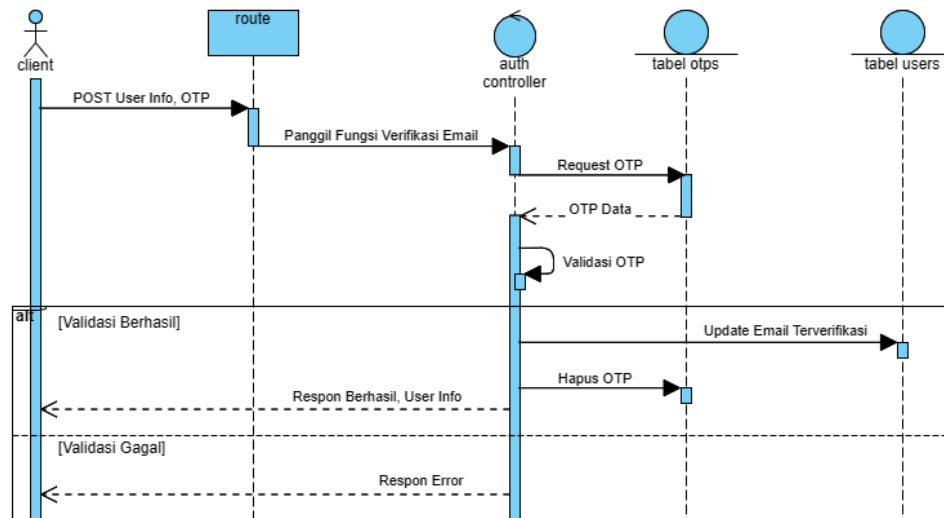
2. Logout



Gambar 2.15 Sequence diagram API *logout*

Pada Gambar 2.15 di atas, sebuah metode *logout* di dalam 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 di dalam *header* pada saat *request* diterima. Proses tersebut sangat penting untuk menjaga keamanan sistem dan melindungi privasi pengguna. Dengan menghapus token saat klien keluar atau *logout*, sistem dapat memastikan bahwa akses yang tidak sah, tidak dapat digunakan untuk mengakses pintu.

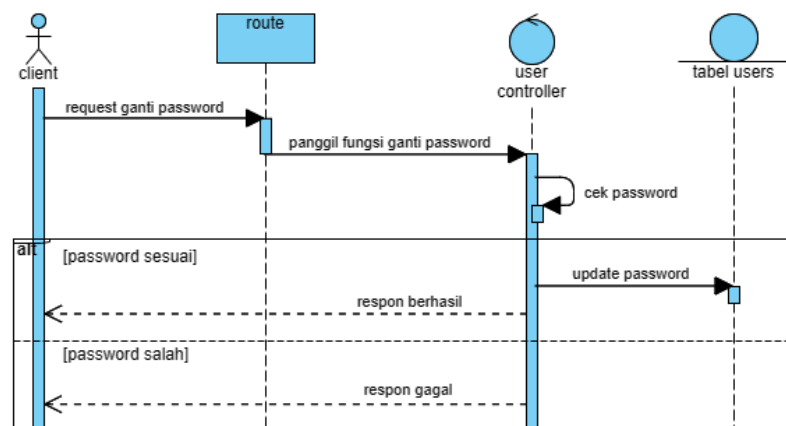
3. Verifikasi email



Gambar 2.16 Sequence diagram API verifikasi email

Pada Gambar 2.16 di atas dapat dilihat bahwa pengguna melakukan verifikasi email dengan mengirimkan kode OTP yang sudah diterima via email disertai dengan detail *client* seperti id, nama, dan email ke *endpoint* “/api/verify-email”. Kemudian, 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, menghapus kode otp yang lama, dan mengembalikan respons berhasil. Jika kode salah atau sudah kedaluwarsa maka kontroler akan mengembalikan respons *error*.

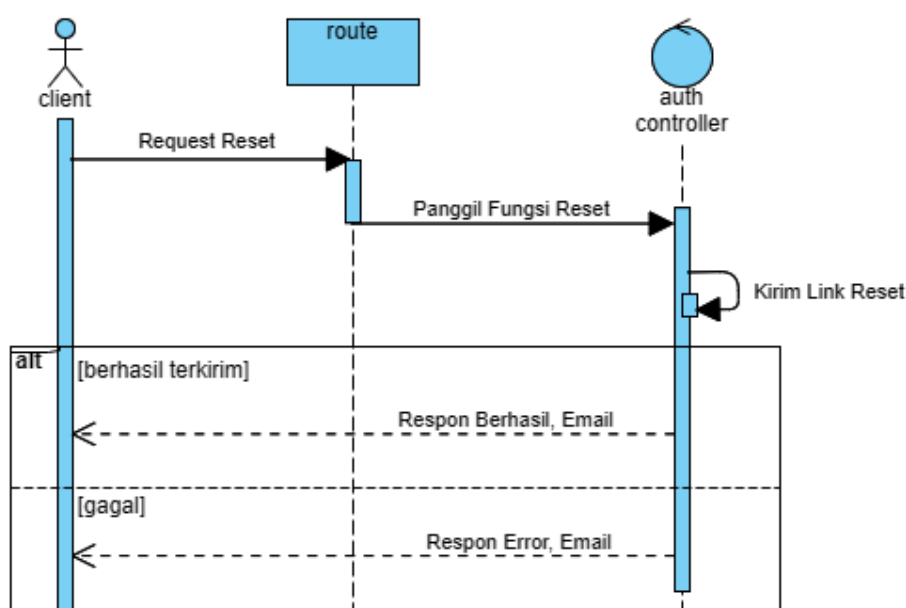
4. Ganti password



Gambar 2. 17 Sequence diagram API ganti password

Dapat dilihat pada Gambar 2.17 di atas, untuk mengganti *password* langkah pertama *client* adalah mengirimkan permintaan ganti *password* ke *endpoint* “/api/change-password” dengan mengirimkan *password* lama, *password* baru, dan konfirmasi *password* baru. Kemudian, di dalam 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 mengembalikan respons berhasil. Jika *password* tidak sesuai maka kontroler akan mengembalikan respons *error*.

5. Reset password

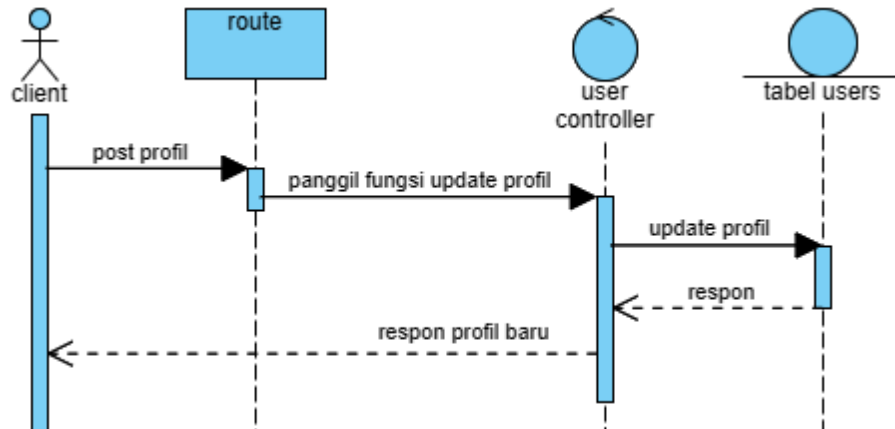


Gambar 2.18 Sequence diagram API reset password

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 tersebut berisi data email yang sudah terdaftar sebagai parameter supaya *server* dapat mengidentifikasi akun yang akan di-*reset 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 tersebut menggunakan layanan email eksternal seperti SMTP yang telah dikonfigurasi pada *server*. Jika email berhasil terkirim dengan sukses maka kontroler akan memberikan respons ke *client* berupa pesan berhasil. Pesan tersebut memberitahukan bahwa email *reset password* telah berhasil dikirim dan pengguna dapat segera memeriksa kotak masuk email-nya 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 tersebut terjadi, kontroler akan memberikan respons *error* ke *client*. Pesan *error* tersebut berisi informasi yang relevan tentang masalah yang terjadi, sehingga pengguna atau operator dapat mengetahui penyebabnya.

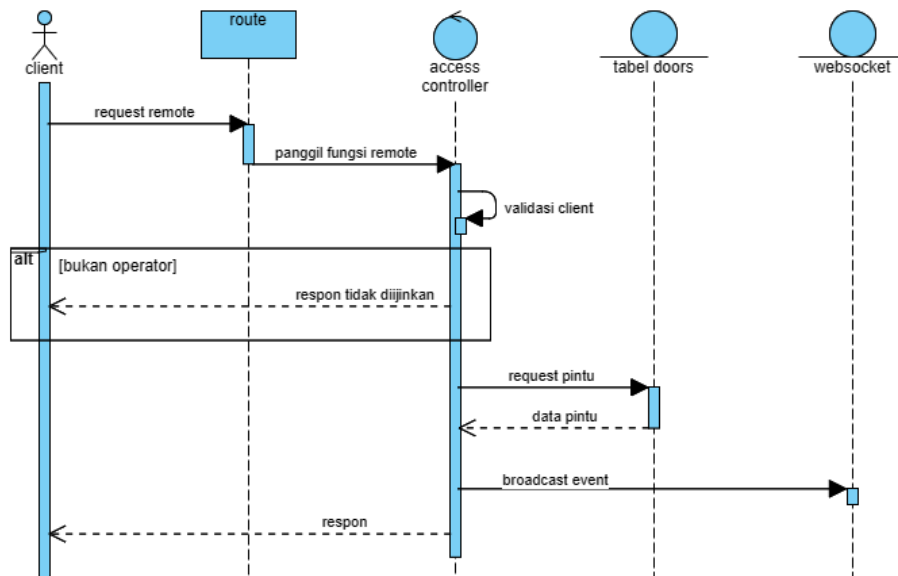
6. Ganti profil



Gambar 2.19 Sequence diagram API ganti profil

Dapat dilihat pada Gambar 2.19 di atas, untuk mengganti profil langkah pertama adalah pengguna atau operator mengirimkan data profil ke *endpoint* *"/api/update-profile"*. Kemudian, di dalam kontroler data yang telah diterima akan dimasukkan ke dalam tabel *users* untuk memperbarui profil dan terakhir kontroler mengembalikan respons bahwa profil berhasil diubah.

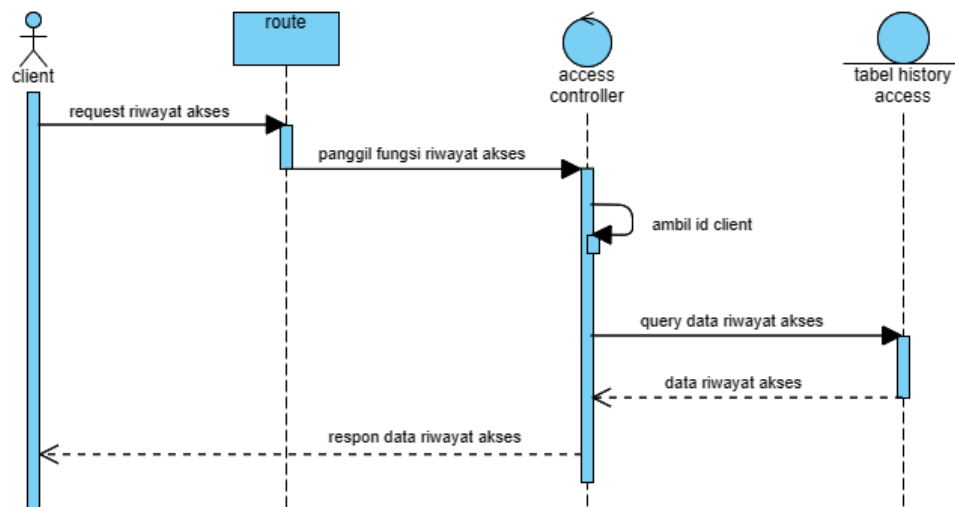
7. Remote pintu



Gambar 2.20 Sequence diagram API remote pintu

Dapat dilihat pada Gambar 2.21 di atas, untuk melakukan *remote* pintu langkah pertama adalah operator melakukan *request* ke *endpoint* “/api/remote-access” dengan mengirimkan identitas pintu yang akan dikendalikan. Kemudian, kontroler memeriksa *client* untuk memastikan permintaan berasal dari operator. Jika berasal dari pengguna biasa maka kontroler akan mengembalikan respons tidak diizinkan. Selanjutnya, kontroler akan mengambil data pintu untuk melengkapi informasi yang dibutuhkan seperti kode kunci, kode gedung, token, dan lain sebagainya. Kemudian, data dikirim ke perangkat kunci pintu melalui koneksi *websocket* yang sudah terhubung. Terakhir, kontroler mengembalikan respons *remote* pintu telah dilaksanakan.

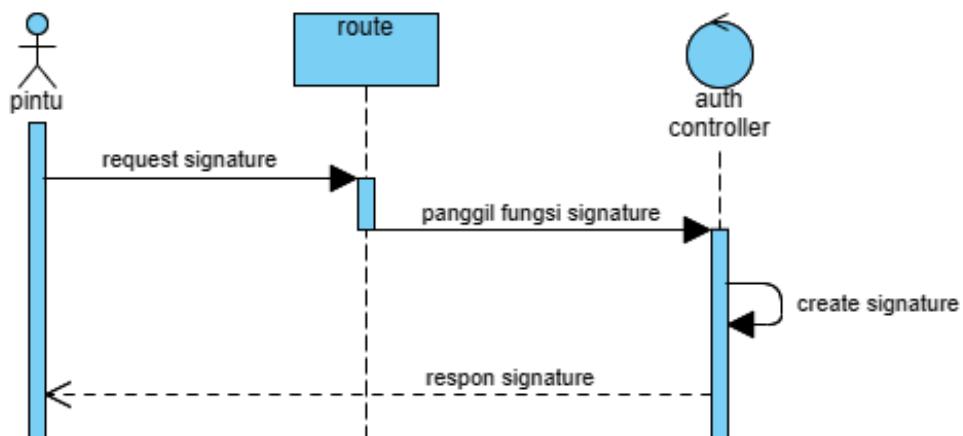
8. Riwayat akses



Gambar 2.21 Sequence diagram API riwayat akses

Dapat dilihat pada Gambar 2.22 untuk mendapatkan data riwayat akses maka pengguna melakukan permintaan ke *endpoint* “/api/my-history”. Kemudian, kontroler mengambil data riwayat akses pengguna dengan melakukan *query* ke *database* dengan menyertakan identitas pengguna sebagai parameter. Selanjutnya, data yang sudah diperoleh seperti nama pintu, nama gedung, waktu aktivitas, dan jenis aktivitas akan dikirimkan kembali ke pengguna sebagai respons untuk diolah pada tampilan aplikasi *mobile*.

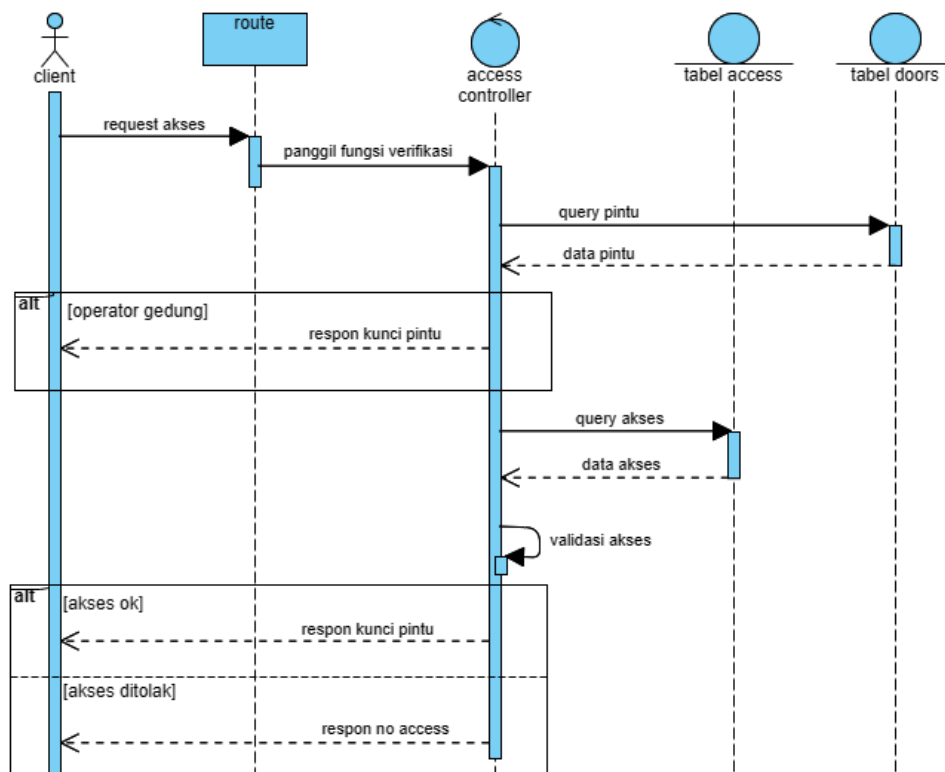
9. Signature



Gambar 2.22 Sequence diagram API door signature

Dapat dilihat pada Gambar 2.23 di atas, untuk mendapatkan kode *signature pusher* langkah pertama adalah 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, kontroler membuat kode *signature* menggunakan metode yang ada pada protokol *pusher*. Setelah mendapatkan nilai *signature*, kontroler mengembalikan respons kode *signature* ke perangkat kunci pintu.

10. Verifikasi akses

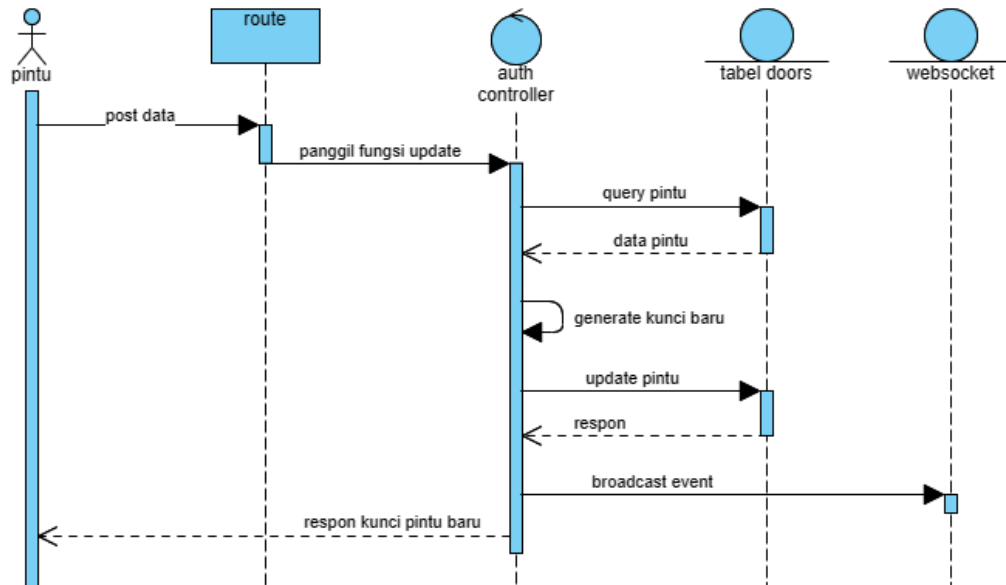


Gambar 2.23 Sequence diagram API verifikasi akses

Pada Gambar 2.24 di atas, untuk mendapatkan akses ke pintu, pengguna memindai kode QR pada pintu untuk mendapatkan informasi terkait pintu tersebut. Kemudian, data tersebut dikirimkan ke *server* melalui *endpoint* “/api/verify-access/{door-id}”. Kemudian, kontroler mengambil data pintu pada tabel *doors*. Jika permintaan berasal dari operator gedung dimana pintu tersebut berada maka kontroler akan mengizinkan dengan mengembalikan respons berupa kode kunci pintu. Jika

permintaan akses dilakukan oleh pengguna biasa maka kontroler akan memeriksa daftar akses di dalam tabel *access*. Jika pengguna memiliki akses dan masih berlaku maka kontroler akan mengembalikan respons akses diizinkan dan mengirimkan kode kunci untuk membuka pintu.

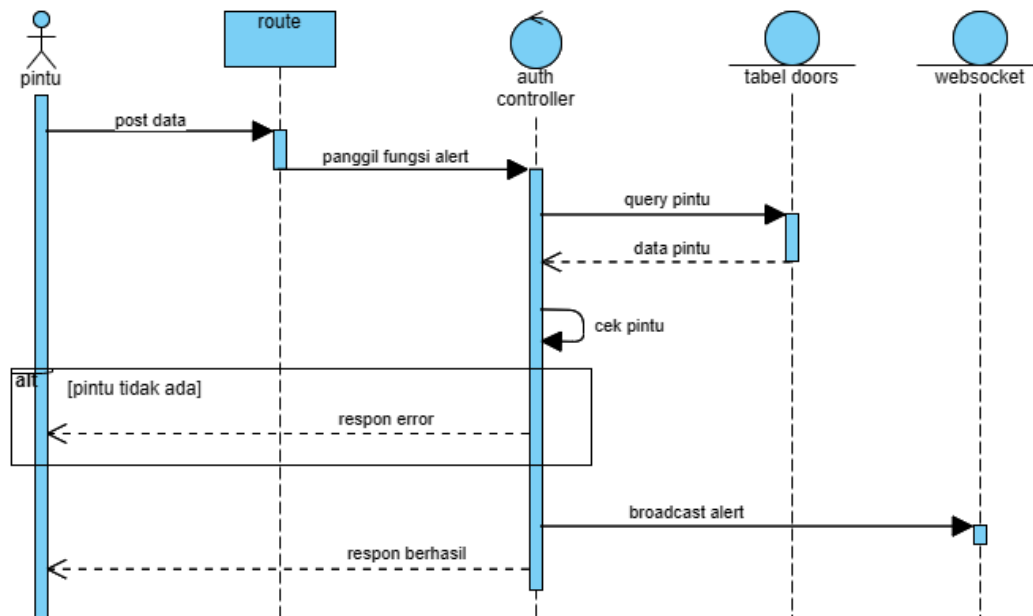
11. *Update* status pintu



Gambar 2.24 Sequence diagram API door update status

Terlihat pada Gambar 2.25 di atas, untuk melakukan *update* status langkah pertama adalah perangkat kunci pintu mengirimkan data-data seperti status penguncian dan *id socket* melalui *endpoint* “/door/update-status”. Kemudian, kontroler mengambil data pintu yang terkait untuk diperbarui menggunakan data status dan kode kunci yang baru. Lalu, kontroler melakukan *broadcasting* untuk menyiarkan bahwa status pintu berubah sehingga setiap informasi perubahan dapat tersampaikan secara langsung. Terakhir, kontroler mengembalikan respons *update* status telah dilaksanakan.

12. Peringatan pintu

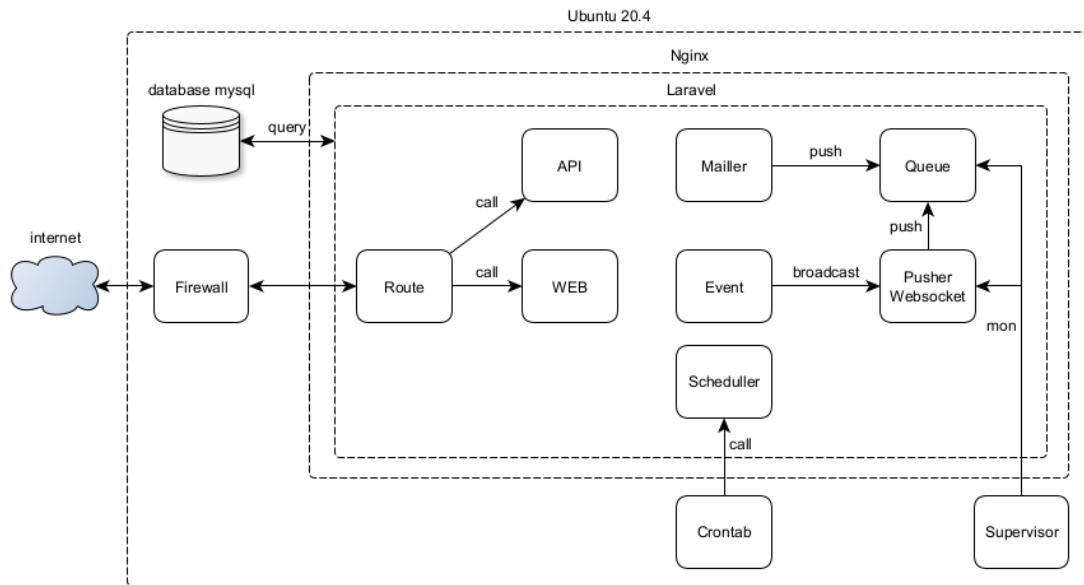


Gambar 2.25 Sequence diagram API door alert

Dapat dilihat pada Gambar 2.26 di atas, untuk memberikan peringatan langkah pertama adalah perangkat kunci pintu mengirimkan data-data seperti id-pintu, id-office, dan status peringatan melalui *endpoint* “/door/alert”. Kemudian, kontroler memeriksa pada tabel pintu untuk memastikan bahwa pintu valid. Jika pintu tidak ditemukan maka kontroler mengembalikan respons *error*. Selanjutnya, kontroler menyiarkan peringatan melalui *websocket*. Terakhir, kontroler mengembalikan respons peringatan sudah dilaksanakan.

2.5 Server

Dengan menggunakan laravel sebagai *backend* yang mengatur kinerja dari perangkat kunci pintu tentunya diperlukan sebuah *server*. *Server* 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 di bawah ini.



Gambar 2.26 Konfigurasi server

Dapat dilihat pada Gambar 2.27 di atas, server dibangun menggunakan sistem operasi ubuntu 20.04. Ubuntu merupakan bagian dari sistem operasi linux yang biasa digunakan baik untuk perangkat dekstop maupun server karena *open source* dan ringan. Dengan menggunakan ubuntu 20.04 sebagai sistem operasi server, pengguna dapat memanfaatkan kestabilan, keamanan, dan dukungan jangka panjang untuk menjalankan aplikasi laravel dengan aman dan efisien.

Di dalam 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 pengguna 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 keamanan data akan terjamin dengan adanya jalur komunikasi yang terenkripsi.

Di dalam 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 tersebut sesuai dengan karakteristik sistem yang dibangun yaitu sistem cepat dan efisien.

Dengan adanya fitur penjadwalan otomatis di dalam sistem kunci pintu maka pengguna 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.

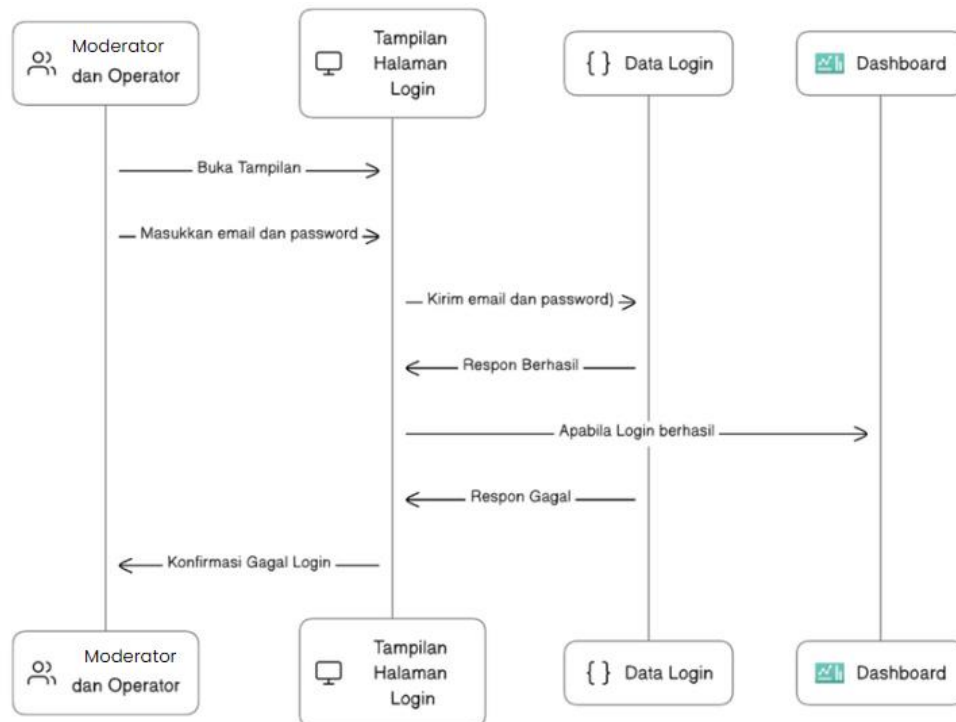
Pada laravel juga mengimplementasikan fitur antrian yang bertujuan untuk meningkatkan kinerja dari sistem sehingga fungsi antrian harus dijaga agar selalu berada dalam kondisi berjalan. Oleh sebab itu, *server* juga dipasang *supervisor* yang digunakan untuk memantau kinerja dari antrian. *Supervisor* merupakan sebuah sistem pengawas atau *process control system* yang digunakan untuk mengontrol dan mengelola proses-proses yang berjalan di dalam 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* agar berjalan secara terus-menerus. *Websocket* memegang peranan penting di dalam 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 juga akan terganggu. Oleh karena itu, diperlukan pengawasan menggunakan *supervisor* untuk menjaga kinerja dari *websocket*.

2.6 Tampilan Website

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

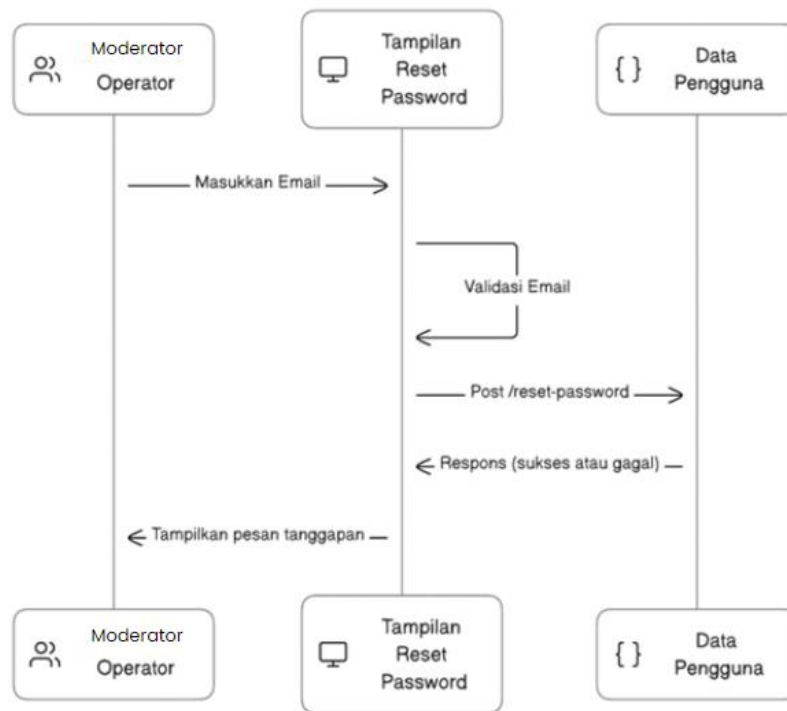
1. Halaman *login*



Gambar 2.28 Diagram *sequence* tampilan halaman *login* website

Gambar 2.28 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah yang terjadi ketika moderator atau operator membuka tampilan halaman *login* dan mencoba melakukan *login* ke dalam sistem. Pertama, moderator atau operator membuka tampilan halaman *login* dengan mengakses antarmuka yang ada pada monitor. Kemudian, moderator atau operator memasukkan email dan *password*. Selanjutnya, email dan *password* tadi dikirim ke data *login*. Jika data *login* yang dikirimkan benar yaitu email dan *password* sesuai dengan yang terdaftar di sistem maka data *login* akan memberikan respons berhasil ke tampilan halaman *login*. Apabila *login* berhasil moderator atau operator diarahkan ke *dashboard*. Namun, jika *login* gagal maka data *login* akan memberikan respons gagal ke tampilan halaman *login*, dilanjutkan mengonfirmasi gagal *login* ke moderator atau operator.

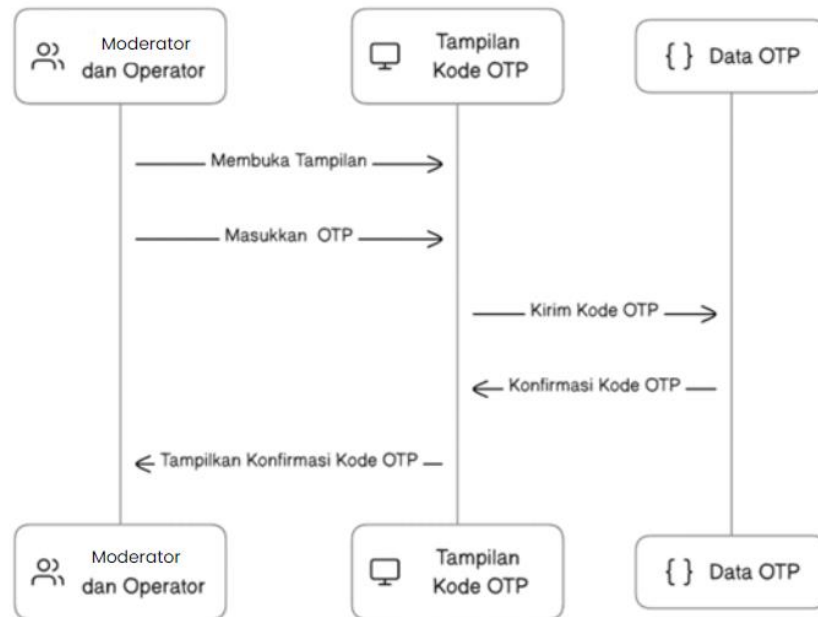
2. Reset Password



Gambar 2.29 Diagram *sequence* tampilan halaman *reset password*

Gambar 2.29 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah saat moderator atau operator menggunakan tampilan *reset password* untuk melakukan *reset* kata sandi dalam sistem. Pertama, moderator atau operator membuka tampilan *reset password* dan memasukkan email. Setelah memasukkan email, tampilan *reset password* melakukan validasi untuk memastikan bahwa email yang dimasukkan valid dan sesuai format. Setelah validasi berhasil, tampilan *reset password* akan mengirim permintaan *reset password* ke data pengguna melalui *endpoint /reset-password*. Data pengguna memproses permintaan tersebut dan memberikan respons (sukses atau gagal) ke tampilan *reset password*. Kemudian, tampilan *reset password* menampilkan pesan tanggapan ke pengguna atau operator.

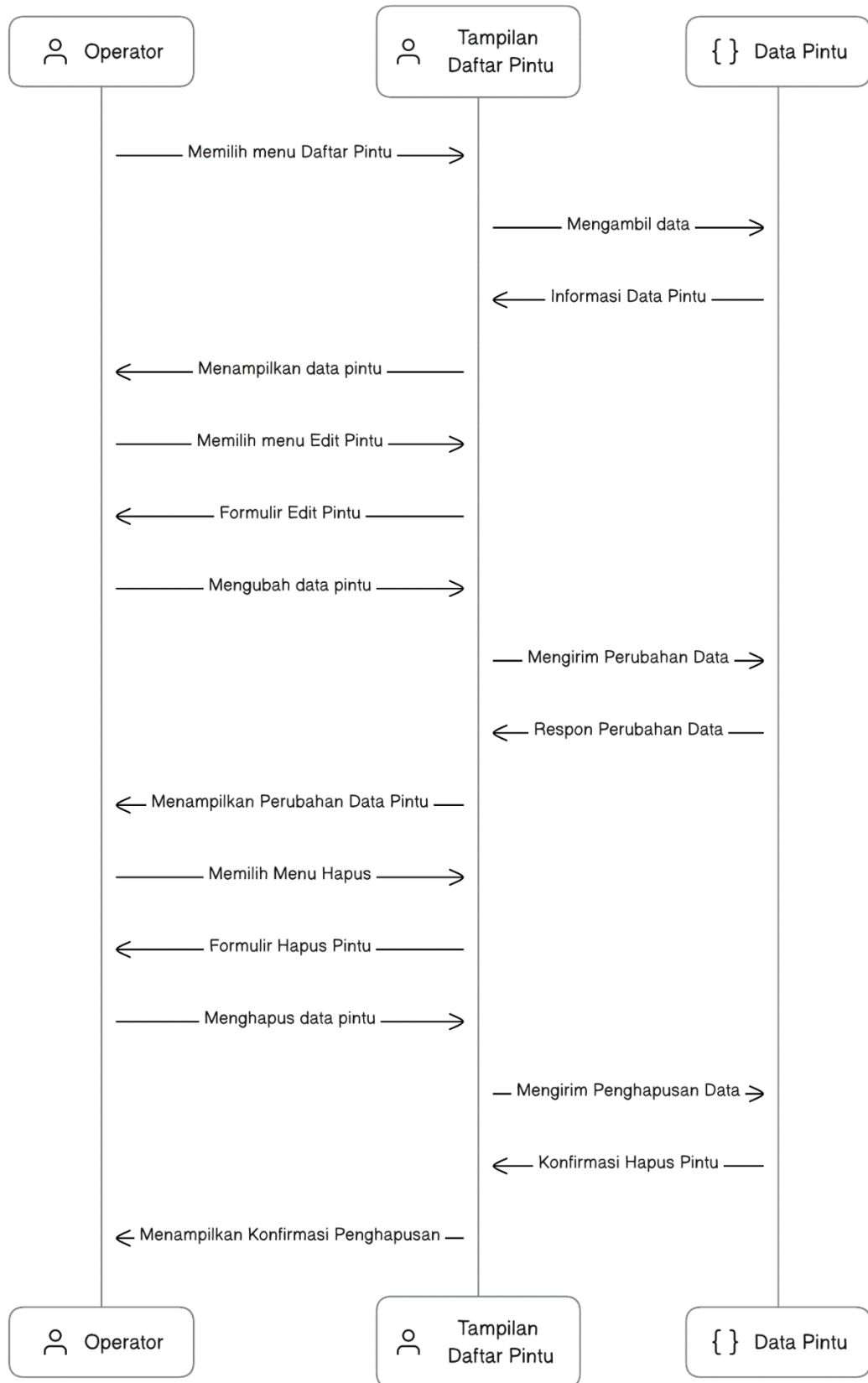
3. Verifikasi *email*



Gambar 2.30 Diagram *sequence* tampilan kode OTP

Gambar 2.30 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah saat moderator atau operator menggunakan tampilan kode OTP untuk memasukkan OTP (*One-Time Password*) dalam sistem. Pertama, moderator atau operator membuka tampilan kode OTP melalui antarmuka yang ada. Kemudian, pengguna atau operator memasukkan OTP yang telah diterima. Setelah memasukkan OTP, tampilan kode OTP akan mengirimkan kode OTP ke data OTP untuk memvalidasi apakah kode yang dimasukkan benar atau tidak. Data OTP akan memproses kode OTP yang diterima dan memberikan konfirmasi apakah kode OTP tersebut valid atau tidak. Jika kode OTP yang dimasukkan valid, tampilan kode OTP akan menampilkan konfirmasi ke moderator atau operator bahwa kode OTP telah berhasil diverifikasi. Namun, jika kode OTP yang dimasukkan tidak valid, tampilan kode OTP akan menampilkan konfirmasi bahwa kode OTP tidak valid dan moderator atau operator perlu memasukkan kode OTP yang benar untuk dapat melanjutkan proses.

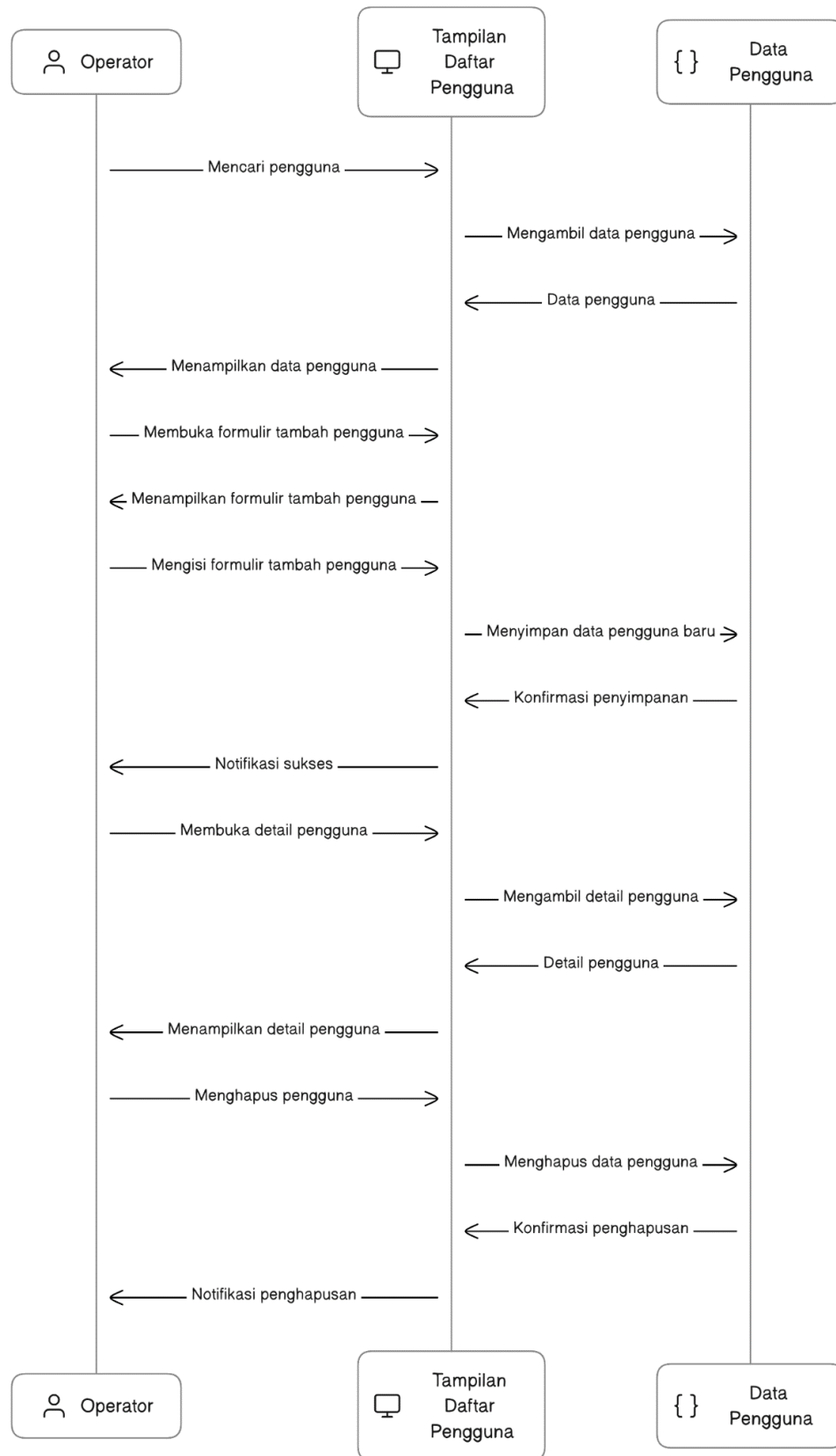
4. Menu Daftar Pintu



Gambar 2.31 Diagram *sequence* tampilan daftar pintu

Gambar 2.31 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika operator menggunakan tampilan daftar pintu untuk mengelola daftar pintu dalam sistem. Pertama, operator memilih menu daftar pintu pada antarmuka yang ada. Kemudian, tampilan daftar pintu mengambil data di data pintu. Data pintu mengirimkan informasi data pintu ke tampilan daftar pintu sehingga operator dapat melihat daftar pintu yang ditampilkan. Selanjutnya, operator dapat memilih menu edit pintu untuk mengubah data pintu tertentu. Tampilan daftar pintu menampilkan formulir edit pintu yang memungkinkan operator untuk mengubah data pintu sesuai kebutuhan. Operator mengisi formulir edit pintu dengan perubahan data yang diinginkan dan menyimpannya. Tampilan daftar pintu mengirim perubahan data ke data pintu. Data pintu mengolah perubahan data yang diterima dan memberikan respons perubahan data. Tampilan daftar pintu menampilkan konfirmasi perubahan data pintu ke operator. Selain itu, operator juga memiliki opsi untuk memilih menu hapus untuk menghapus pintu tertentu. Tampilan daftar pintu menampilkan formulir hapus pintu yang memungkinkan operator untuk melakukan penghapusan data pintu. Operator mengisi formulir hapus pintu untuk mengonfirmasi penghapusan data pintu. Setelah itu, tampilan daftar pintu mengirim permintaan penghapusan data ke data pintu. Data pintu memproses permintaan penghapusan pintu dan memberikan konfirmasi hapus pintu. Tampilan daftar pintu menampilkan konfirmasi penghapusan ke operator.

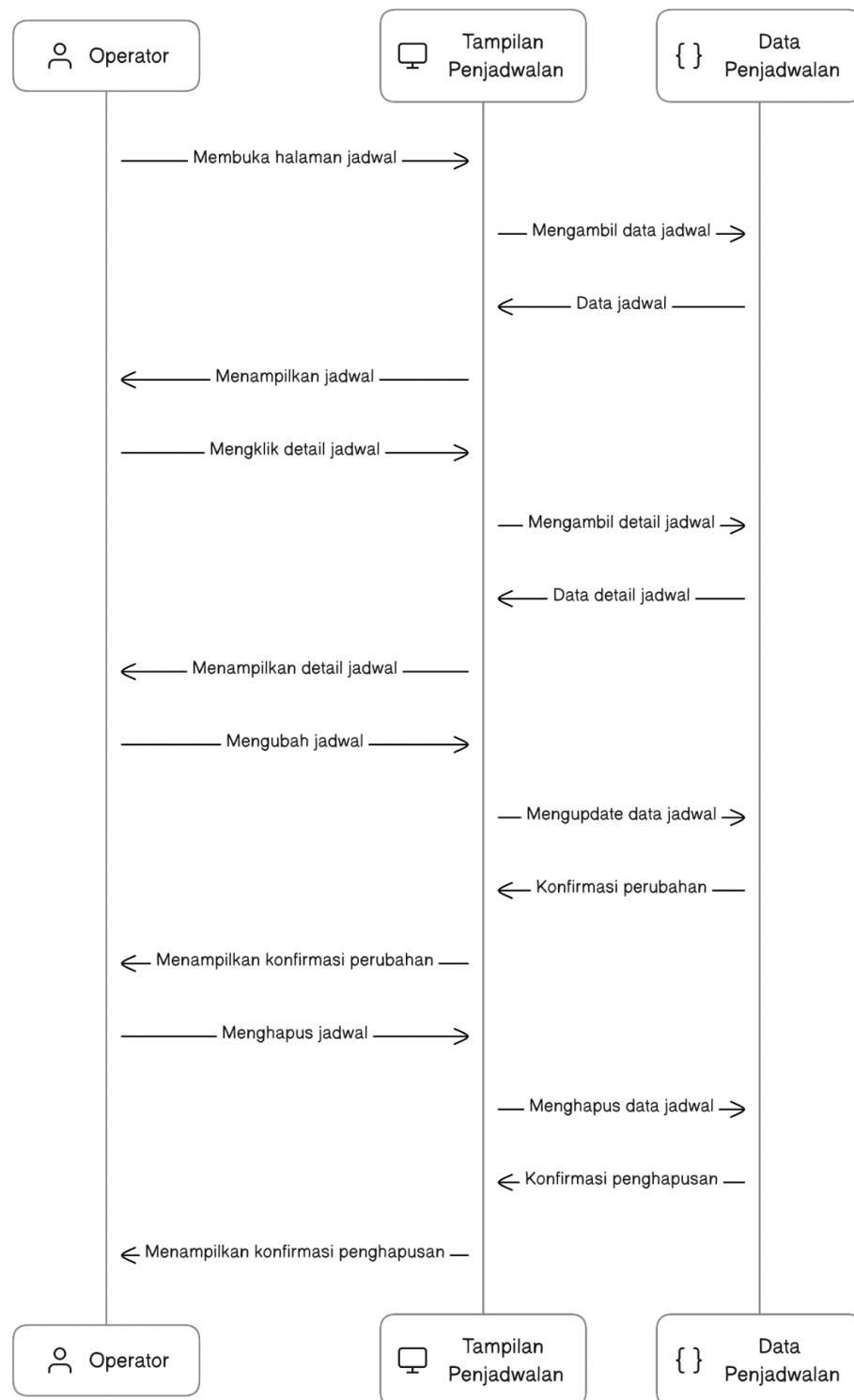
5. Menu Daftar Pengguna



Gambar 2.32 Diagram *sequence* tampilan daftar pengguna

Gambar 2.32 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika operator menggunakan tampilan daftar pengguna untuk mengelola data pengguna dalam sistem. Pertama, operator mencari pengguna dengan membuka tampilan daftar pengguna. Kemudian, tampilan daftar pengguna mengambil data pengguna dari komponen data pengguna untuk menampilkan informasi mengenai daftar pengguna yang ada. Komponen data pengguna mengirimkan data pengguna ke tampilan daftar pengguna sehingga operator dapat melihat daftar pengguna yang ada. Selanjutnya, operator dapat membuka formulir tambah pengguna dengan memilih opsi untuk menambahkan pengguna baru. Tampilan daftar pengguna menampilkan formulir tambah pengguna yang memungkinkan operator untuk mengisi data pengguna baru. Operator mengisi formulir tambah pengguna dengan informasi yang diperlukan dan menyimpannya. Tampilan daftar pengguna menyimpan data pengguna baru ke komponen data pengguna. Komponen data pengguna mengelola data pengguna baru yang diterima dan memberikan konfirmasi bahwa penyimpanan data pengguna telah berhasil dilakukan. Tampilan daftar pengguna menampilkan notifikasi sukses ke operator. Selain itu, operator juga dapat membuka detail pengguna dengan memilih pengguna tertentu dari daftar pengguna. Tampilan daftar pengguna mengambil detail pengguna dari data pengguna untuk menampilkan informasi lebih rinci mengenai pengguna yang dipilih. Data pengguna mengirimkan detail pengguna ke tampilan daftar pengguna sehingga operator dapat melihat informasi detail pengguna tersebut. Operator juga memiliki opsi untuk menghapus pengguna tertentu. Jika memilih untuk menghapus data pengguna, tampilan daftar pengguna mengirim permintaan penghapusan ke data pengguna. Komponen data pengguna mengelola proses penghapusan data pengguna dan memberikan konfirmasi bahwa penghapusan telah berhasil dilakukan. Tampilan daftar pengguna menampilkan notifikasi penghapusan kepada operator.

6. Penjadwalan

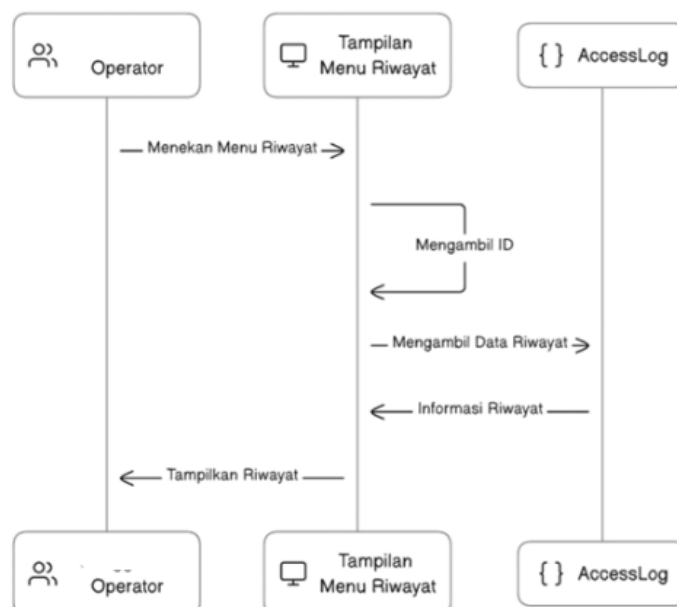


Gambar 2.33 Diagram *sequence* tampilan penjadwalan

Gambar 2.33 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah yang terjadi ketika operator membuka dan mengelola jadwal melalui

tampilan penjadwalan pada *website*. Pertama, operator membuka halaman jadwal yang menampilkan daftar jadwal yang tersedia. Tampilan penjadwalan menghubungi data penjadwalan untuk mengambil data jadwal. Data penjadwalan mengirim data jadwal yang diperlukan ke tampilan penjadwalan. Setelah itu, tampilan penjadwalan menampilkan jadwal kepada operator. Selanjutnya, operator dapat mengklik detail jadwal pada tampilan penjadwalan. Jika operator mengkliknya, tampilan penjadwalan mengambil detail jadwal di data penjadwalan. Data penjadwalan mengirim data detail jadwal ke tampilan penjadwalan. Tampilan penjadwalan menampilkan detail jadwal ke operator. Operator juga memiliki opsi untuk mengubah jadwal. Jika pengguna memilih untuk mengubah jadwal, tampilan penjadwalan menghubungi data penjadwalan untuk meng-*update* data jadwal. Data penjadwalan melakukan konfirmasi perubahan ke tampilan penjadwalan. Tampilan penjadwalan menampilkan konfirmasi perubahan ke operator. Operator juga memiliki opsi untuk menghapus jadwal. Jika operator memilih untuk menghapus jadwal, tampilan penjadwalan menghubungi data penjadwalan untuk menghapus data jadwal. Data penjadwalan melakukan konfirmasi penghapusan ke tampilan penjadwalan. Tampilan penjadwalan menampilkan konfirmasi penghapusan jadwal ke operator.

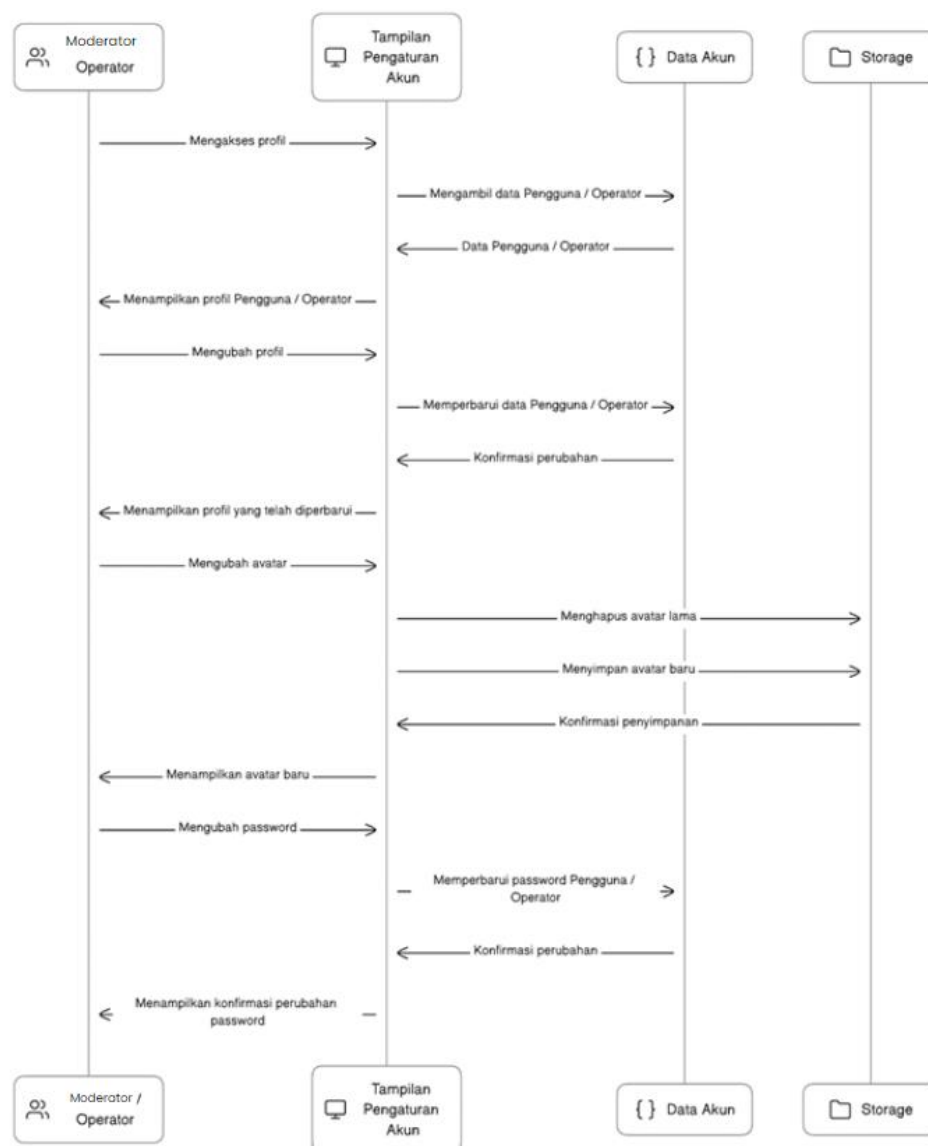
7. Menu Riwayat Akses



Gambar 2.34 Diagram *sequence* tampilan menu riwayat

Gambar 2.34 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika operator menekan menu riwayat pada *website* untuk melihat riwayat akses. Pertama, operator menekan menu riwayat pada tampilan menu riwayat di *website*. Tampilan menu riwayat mengambil ID untuk digunakan dalam pengambilan data riwayat akses di *AccessLog*. *AccessLog* mengirim informasi riwayat ke tampilan menu riwayat. Setelah itu, tampilan menu riwayat menampilkan riwayat akses ke operator.

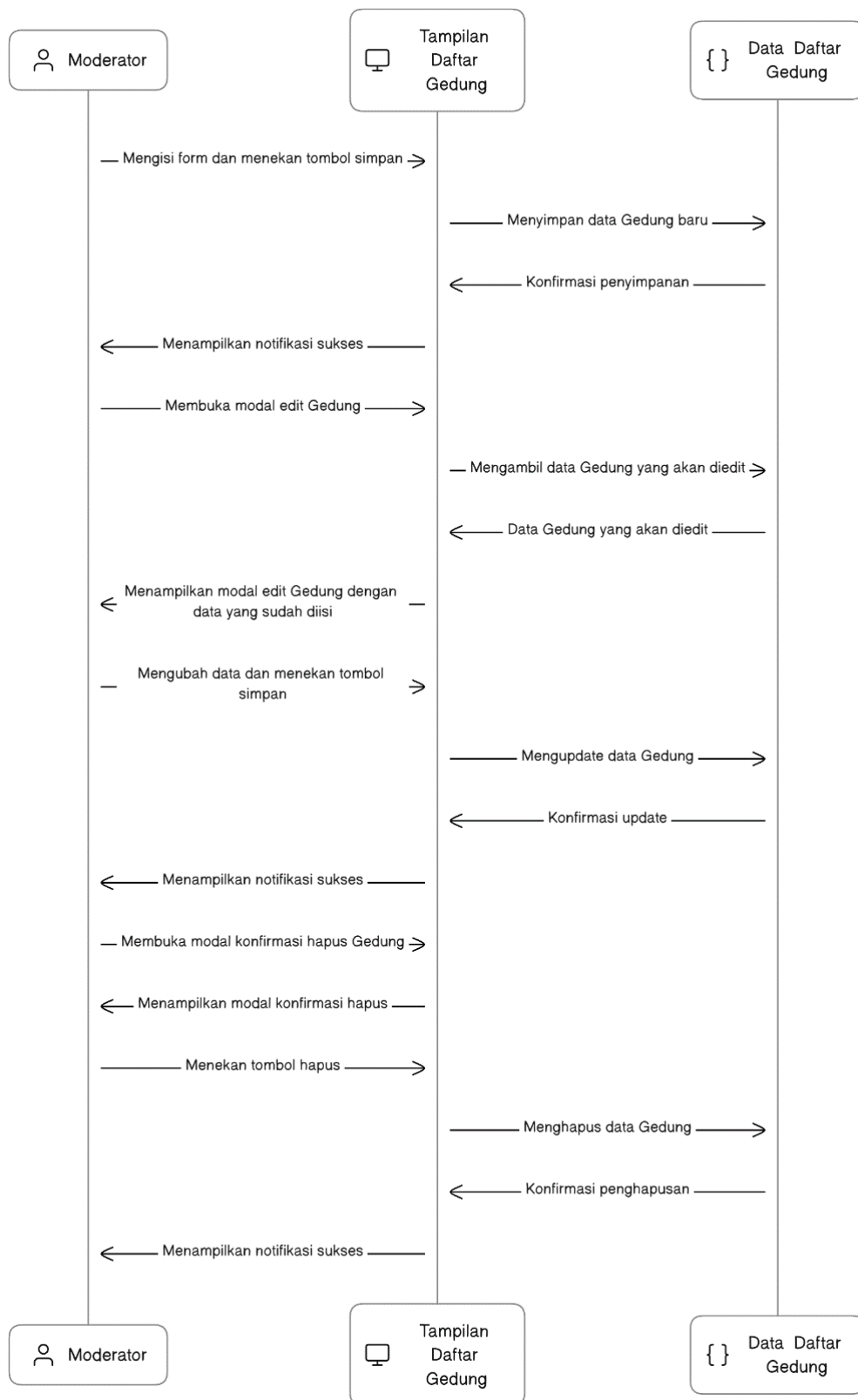
8. Pengaturan Akun



Gambar 2.35 Diagram *sequence* tampilan pengaturan akun

Gambar 2.35 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika moderator atau operator mengakses dan mengelola profil melalui halaman pengaturan akun pada *website*. Pertama, moderator atau operator mengakses profil pada tampilan pengaturan akun. Tampilan pengaturan akun menghubungi data akun untuk mengambil data moderator atau operator. Data akun mengirimkan data moderator atau operator ke tampilan pengaturan akun. Setelah itu, tampilan pengaturan akun menampilkan profil ke moderator atau operator. Selanjutnya, moderator atau operator memiliki opsi untuk mengubah profilnya. Jika pengguna memilih untuk mengubah profil, tampilan pengaturan akun menghubungi data akun untuk memperbarui data moderator atau operator. Data akun mengonfirmasi perubahan ke tampilan pengaturan akun. Tampilan pengaturan akun menampilkan profil yang telah diperbarui ke moderator atau operator. Selain itu, moderator atau operator juga dapat mengubah foto profil. Jika memilih untuk mengubah foto, tampilan pengaturan akun menghubungi *storage* untuk menghapus foto lama dan menyimpan foto baru. *Storage* memberikan konfirmasi penyimpanan foto baru ke tampilan pengaturan akun. Tampilan pengaturan akun menampilkan foto baru ke moderator atau operator. Terakhir, moderator atau operator dapat mengubah *password*. Jika moderator atau operator memilih untuk mengubah *password*, tampilan pengaturan akun menghubungi data akun untuk memperbarui *password* moderator atau operator. Data akun memberikan konfirmasi perubahan ke tampilan pengaturan akun. Tampilan pengaturan akun menampilkan konfirmasi perubahan *password* ke moderator atau operator.

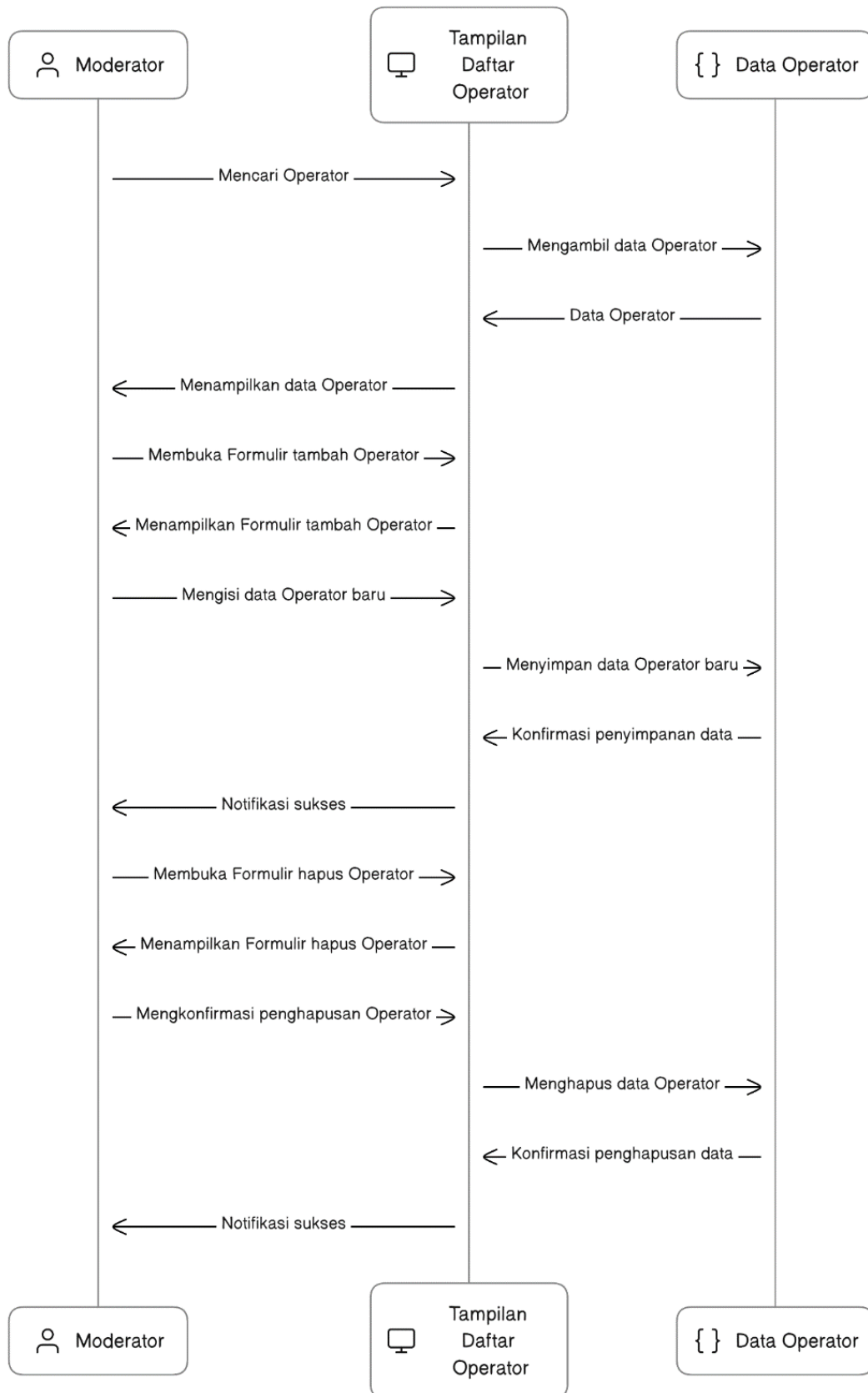
9. Daftar Gedung



Gambar 2.36 Diagram *sequence* tampilan daftar gedung

Gambar 2.36 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah moderator menggunakan halaman daftar gedung untuk menambah, mengubah, dan menghapus data gedung dalam sistem. Pertama, moderator membuka tampilan daftar gedung, mengisi form, dan menekan tombol simpan pada tampilan daftar gedung. Tampilan daftar gedung menyimpan data gedung baru ke dalam data daftar gedung. Data daftar gedung memberikan konfirmasi penyimpanan ke tampilan daftar gedung. Kemudian, tampilan daftar gedung menampilkan notifikasi sukses ke moderator. Selanjutnya, moderator memiliki opsi untuk mengedit gedung dengan membuka modal edit gedung. Tampilan daftar gedung mengambil data gedung yang akan diedit pada data daftar gedung. Data daftar gedung memberikan data gedung yang akan diedit ke tampilan daftar gedung. Tampilan daftar gedung menampilkan modal edit gedung dengan data yang sudah diisi ke moderator. Moderator mengubah data dan menekan tombol simpan pada tampilan daftar gedung. Tampilan daftar gedung meng-*update* data gedung pada data daftar gedung. Data daftar gedung memberikan konfirmasi *update* ke tampilan daftar gedung. Moderator menerima notifikasi sukses dari tampilan daftar gedung. Terakhir, moderator memiliki opsi untuk menghapus gedung dengan membuka modal konfirmasi hapus gedung. Tampilan daftar gedung menampilkan modal konfirmasi hapus gedung ke moderator. Moderator menekan tombol hapus pada tampilan daftar gedung. Tampilan daftar gedung menghubungi data daftar gedung untuk menghapus data gedung yang telah dipilih oleh moderator. Data daftar gedung memberikan konfirmasi penghapusan ke tampilan daftar gedung. Moderator menerima notifikasi sukses dari tampilan daftar gedung setelah data gedung berhasil dihapus.

10. Daftar Operator



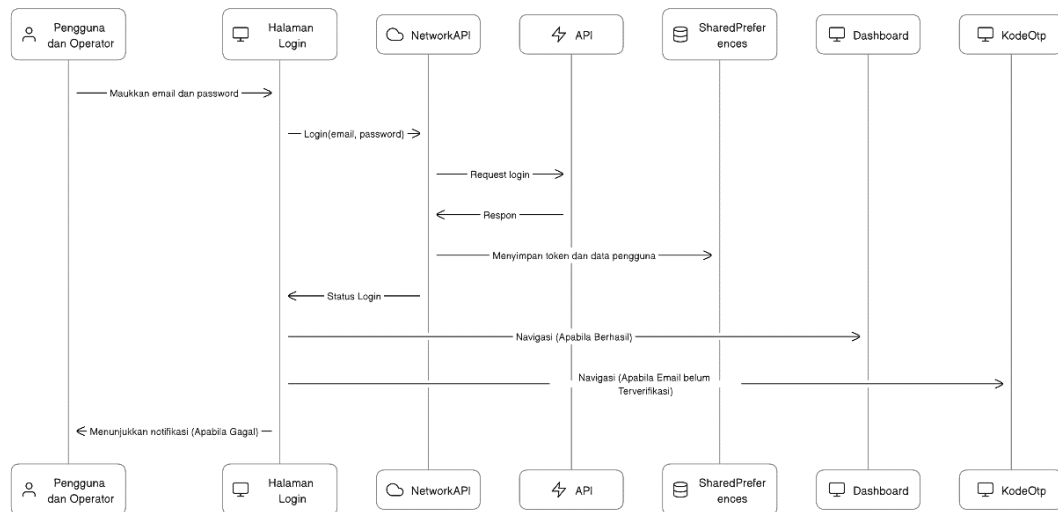
Gambar 2.37 Diagram *sequence* tampilan daftar operator

Gambar 2.37 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah moderator saat menggunakan tampilan daftar operator untuk melakukan pencarian, penambahan, dan penghapusan data operator dalam sistem. Langkah pertama adalah moderator membuka tampilan daftar operator untuk melakukan pencarian operator. Tampilan tersebut akan menghubungkan data operator untuk mengambil data operator yang sesuai dengan kriteria pencarian yang dimasukkan oleh moderator. Setelah itu, komponen data operator memberikan data operator yang sesuai ke tampilan daftar operator. Kemudian, tampilan daftar operator menampilkan data operator ke moderator. Langkah berikutnya adalah ketika moderator ingin menambahkan operator baru melalui formulir tambah operator. Moderator membuka formulir tambah operator pada tampilan daftar operator. Tampilan daftar operator menampilkan formulir tersebut ke moderator. Moderator mengisi data operator baru. Setelah selesai mengisi, tampilan daftar operator menyimpan data operator baru di data operator baru. Data operator memberikan konfirmasi penyimpanan ke tampilan daftar operator. Moderator menerima notifikasi sukses dari tampilan daftar operator setelah data operator berhasil ditambahkan. Selain penambahan, moderator juga dapat melakukan penghapusan operator. Moderator membuka formulir hapus operator pada tampilan daftar operator. Tampilan daftar operator menampilkan formulir tersebut ke moderator. Moderator mengonfirmasi penghapusan operator pada tampilan daftar operator. Setelah itu, tampilan daftar operator menghapus data operator di komponen data operator. Data operator memberikan konfirmasi penghapusan data ke tampilan daftar operator. Moderator menerima notifikasi sukses dari tampilan daftar operator setelah data operator berhasil dihapus.

2.7 Aplikasi Mobile

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

1. Tampilan Halaman *Login*

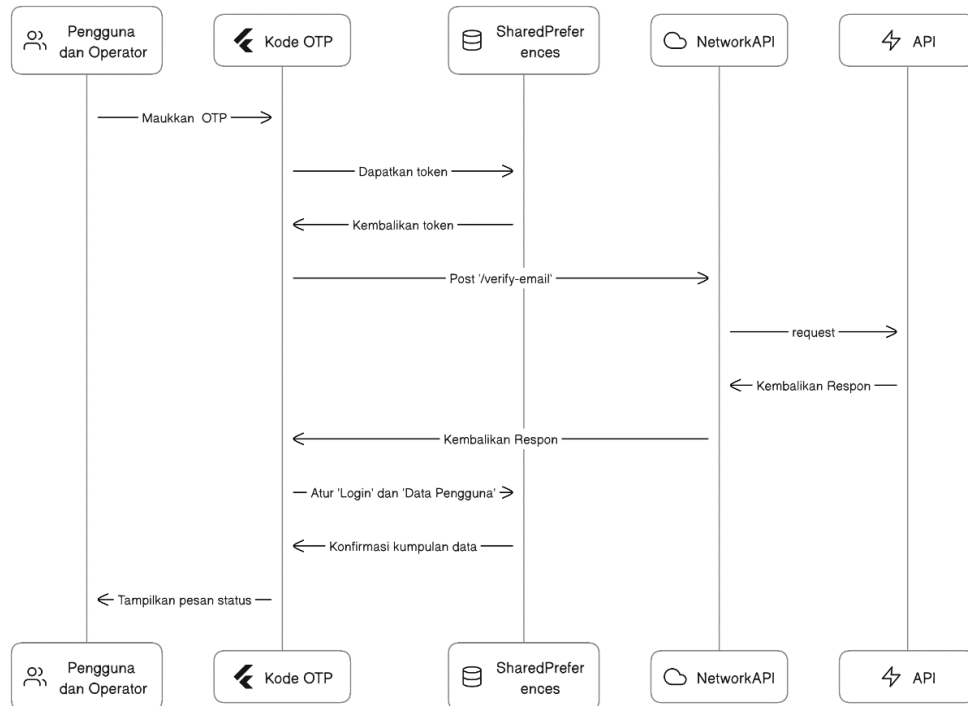


Gambar 2.38 Diagram *sequence* tampilan halaman *login*

Gambar 2.38 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah saat pengguna atau operator melakukan *login* di halaman *login* sistem. Pertama, pengguna atau operator memasukkan email dan *password* melalui antarmuka halaman *login*. Kemudian, antarmuka tersebut menghubungi *NetworkAPI* untuk memproses *login* dengan mengirimkan email dan *password*. Setelah menerima permintaan *login*, *NetworkAPI* berkomunikasi dengan *API* untuk *request login*. *API* memberikan respons ke *NetworkAPI* mengenai status *login* (berhasil atau gagal). Jika *login* berhasil, *NetworkAPI* menyimpan token dan data pengguna yang diperlukan di *SharedPreferences*. Token tersebut akan digunakan untuk mengidentifikasi dan mengautentikasi pengguna atau operator di masa mendatang saat sesi *login*. Setelah menerima respons dari *API*, *NetworkAPI* memberikan status *login* ke halaman *login*. Jika *login* berhasil, halaman *login* akan berlanjut ke halaman *dashboard*. Namun, jika email belum terverifikasi, halaman *login* akan berlanjut ke halaman kodeOtp untuk memverifikasi email. Jika *login* gagal, halaman *login* akan menampilkan notifikasi kegagalan *login* ke pengguna atau operator. Dengan demikian, diagram tersebut memberikan gambaran yang jelas tentang proses *login* dalam sistem serta bagaimana antarmuka dan komponen

lainnya berinteraksi untuk mencapai autentikasi dan navigasi ke halaman yang sesuai berdasarkan status *login*.

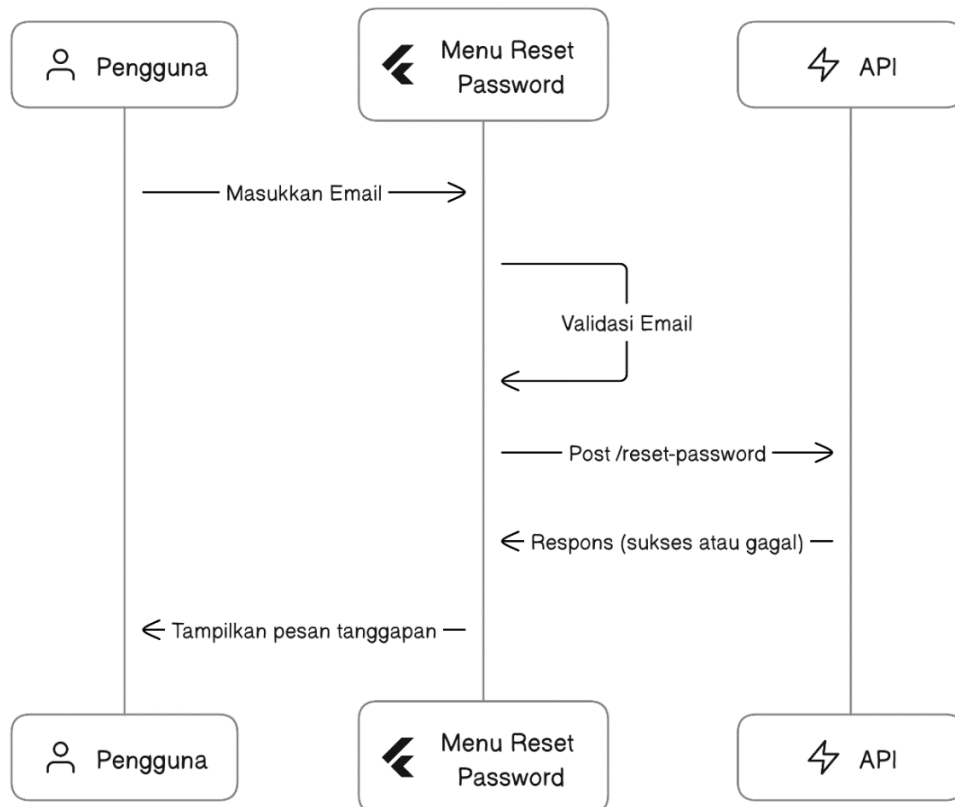
2. Verifikasi Email



Gambar 2.39 Diagram *sequence* tampilan verifikasi email

Gambar 2.39 di atas merupakan diagram *sequence* yang menggambarkan proses verifikasi email menggunakan kode OTP dalam sebuah aplikasi. Pertama, pengguna atau operator memasukkan kode OTP yang diterima melalui aplikasi berbasis *Flutter*. Setelah itu, kode OTP disimpan secara sementara dalam *SharedPreferences*, sebuah tempat penyimpanan lokal pada perangkat. Setelah itu, *SharedPreferences* mengembalikan token ke kode OTP. Kode OTP melakukan *post/verify email* ke *NetworkAPI* untuk memulai proses verifikasi. *NetworkAPI* melakukan *request* ke API untuk memverifikasi kode OTP. Setelah proses verifikasi selesai, API mengirimkan respons ke *NetworkAPI*. Respons tersebut kemudian diteruskan kembali oleh *NetworkAPI* ke kode OTP. Kode OTP mengatur *login* dan data pengguna pada *SharedPreferences*. Terakhir, *SharedPreferences* mengirimkan konfirmasi kumpulan data ke kode OTP dan kode OTP menampilkan pesan status ke operator atau pengguna.

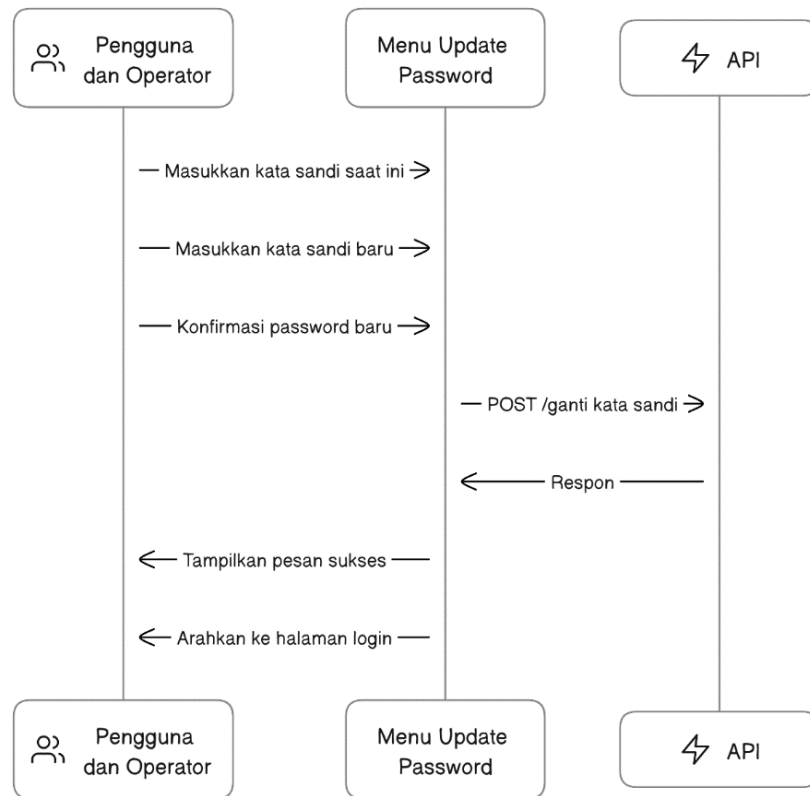
3. Reset Password



Gambar 2.40 Diagram *sequence* tampilan *reset password*

Gambar 2.40 di atas merupakan diagram *sequence* yang menggambarkan proses *reset password* dalam sebuah aplikasi. Pengguna mengajukan permintaan untuk me-*reset password* dengan memasukkan alamat email pada menu *reset password*. Yang dilanjutkan dengan melakukan validasi email yang dimasukkan untuk memastikan bahwa email tersebut valid dan terdaftar di dalam sistem. Setelah email divalidasi, menu *reset password* melakukan *post/reset password* ke API server melalui protokol HTTP POST dengan mengirimkan data permintaan *reset password* yang berisi alamat email pengguna. API server akan memproses permintaan tersebut dan mencoba me-*reset password* untuk akun yang sesuai dengan email yang diberikan. Kemudian, API server akan memberikan respons (sukses atau gagal) ke menu *reset password*. Terakhir, pengguna menerima pesan tanggapan yang ditampilkan oleh menu *reset password* pada aplikasi.

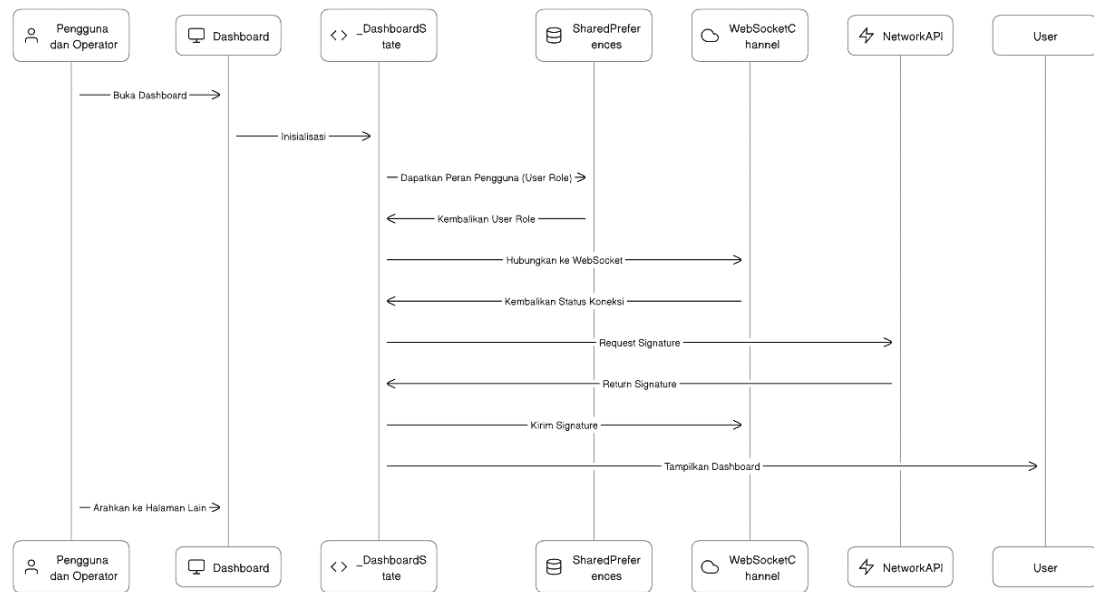
4. Update Password



Gambar 2.41 Diagram *sequence* tampilan menu *update password*

Gambar 2.41 di atas merupakan diagram *sequence* yang menggambarkan alur proses ketika pengguna atau operator ingin mengganti *password* melalui menu *update password* dalam sistem. Pertama, pengguna atau operator memasukkan kata sandi saat ini sebagai langkah verifikasi identitas pada menu *update password*. Setelah itu, pengguna atau operator memasukkan kata sandi baru yang ingin digunakan pada menu *update password*. Untuk memastikan keakuratan, sistem meminta pengguna untuk mengonfirmasi *password* baru yang telah dimasukkan sebelumnya. Kemudian, menu *update password* melakukan *post*/ganti kata sandi ke API. Setelah berhasil melakukan proses penggantian kata sandi, API memberikan respons ke menu *update password* untuk menunjukkan bahwa proses telah berhasil. Kemudian, menu *update password* menampilkan pesan sukses ke pengguna atau operator. Menu *update password* juga mengarahkan pengguna atau operator ke halaman *login*, sehingga pengguna atau operator dapat masuk kembali ke sistem dengan menggunakan kata sandi baru yang telah diperbarui.

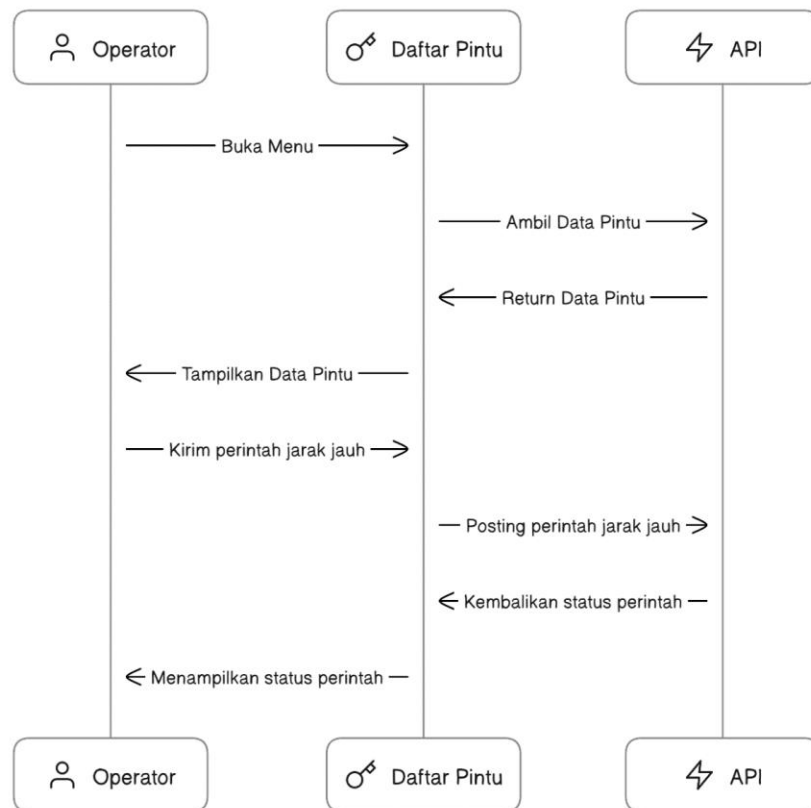
5. Tampilan Halaman *Dashboard*



Gambar 2.42 Diagrams *sequence* tampilan halaman *dashboard*

Gambar 2.42 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika pengguna atau operator membuka *dashboard* dalam sistem. Saat pengguna atau operator membuka *dashboard*, sistem menampilkan tampilan awal. Kemudian, sistem memeriksa peran pengguna atau operator yang sedang aktif untuk menentukan hak aksesnya. Setelah itu, sistem memastikan koneksi ke *WebSocket* yang memungkinkan komunikasi *real-time*. Setelah koneksi berhasil, sistem akan meminta tanda tangan (*signature*) dari *server* melalui *NetworkAPI*. Setelah mendapatkan tanda tangan, sistem mengirimkan tanda tangan melalui *WebSocket*. Setelah menerima tanda tangan, sistem menampilkan *dashboard* dengan konten yang sesuai dengan peran pengguna atau operator. Jika ada halaman lain yang perlu diakses dari *dashboard*, sistem akan mengarahkan pengguna atau operator ke halaman lain. Dengan demikian, diagram tersebut memberikan gambaran tentang proses sederhana saat pengguna atau operator membuka *dashboard* dan bagaimana komponen-komponen dalam sistem berinteraksi untuk memberikan tampilan yang sesuai dengan peran pengguna dan menavigasikan ke halaman lain jika diperlukan.

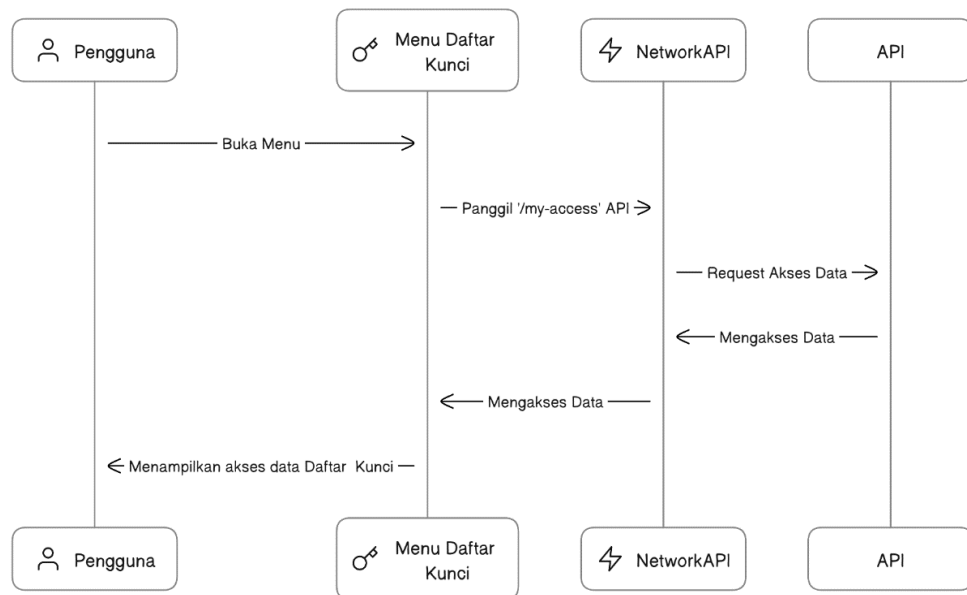
6. Tampilan Menu Daftar Pintu



Gambar 2.43 Diagram *sequence* tampilan menu daftar pintu

Gambar 2.43 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah yang terjadi ketika operator membuka daftar pintu dan menggunakan fitur jarak jauh untuk mengendalikan pintu-pintu dalam sistem. Pertama, operator membuka menu daftar pintu untuk melihat daftar pintu yang terhubung. Daftar pintu menghubungi API untuk mengambil data pintu. Setelah menerima permintaan, API *return* data pintu ke daftar pintu. Daftar pintu menampilkan data pintu tersebut ke operator, sehingga operator dapat melihat informasi tentang pintu-pintu yang ada. Selanjutnya, operator menggunakan fitur perintah jarak jauh untuk mengendalikan salah satu pintu dari jarak jauh. Daftar pintu menghubungi kembali API untuk melakukan *posting* perintah jarak jauh tersebut. Setelah menerima perintah, API memprosesnya dan mengembalikan status perintah ke daftar pintu. Kemudian, daftar pintu menampilkan status perintah tersebut ke operator, sehingga operator dapat melihat apakah perintah telah berhasil atau tidak.

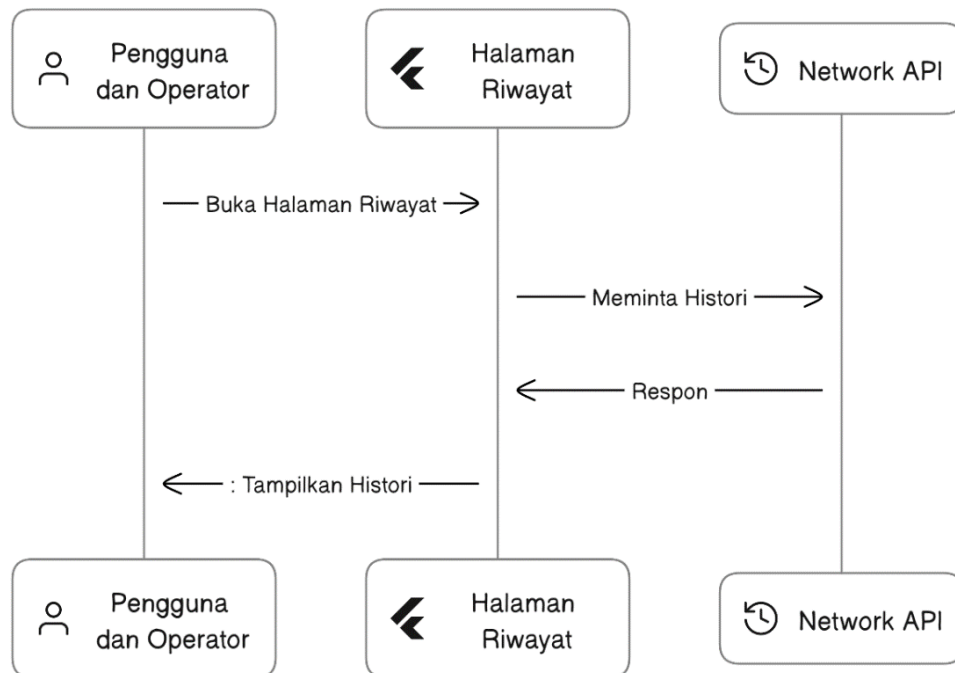
7. Tampilan Menu Daftar Kunci



Gambar 2.44 Diagram *sequence* tampilan menu daftar kunci

Gambar 2.44 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika pengguna membuka menu daftar kunci untuk melihat data akses terkait kunci dalam sistem. Pengguna pertama kali membuka menu daftar kunci yang menampilkan daftar kunci yang ada. Selanjutnya, menu daftar kunci menghubungi *NetworkAPI* untuk mengambil data akses pengguna terkini dengan memanggil API *'/my-access'*. Setelah menerima permintaan, *NetworkAPI* meminta akses data dari API. API mengakses data yang diperlukan dan memberikannya kembali ke *NetworkAPI*. Selanjutnya, *NetworkAPI* memberikan data akses tersebut ke menu daftar kunci. Menu daftar kunci menampilkan data akses tersebut ke pengguna sehingga pengguna dapat melihat daftar kunci yang dimiliki. Proses tersebut memastikan bahwa pengguna dapat dengan mudah melihat informasi tentang akses kunci melalui menu daftar kunci tanpa perlu campur tangan manual. Dengan alur ini, pengguna dapat dengan cepat mengetahui status dan informasi penting terkait kunci yang dimilikinya dalam sistem.

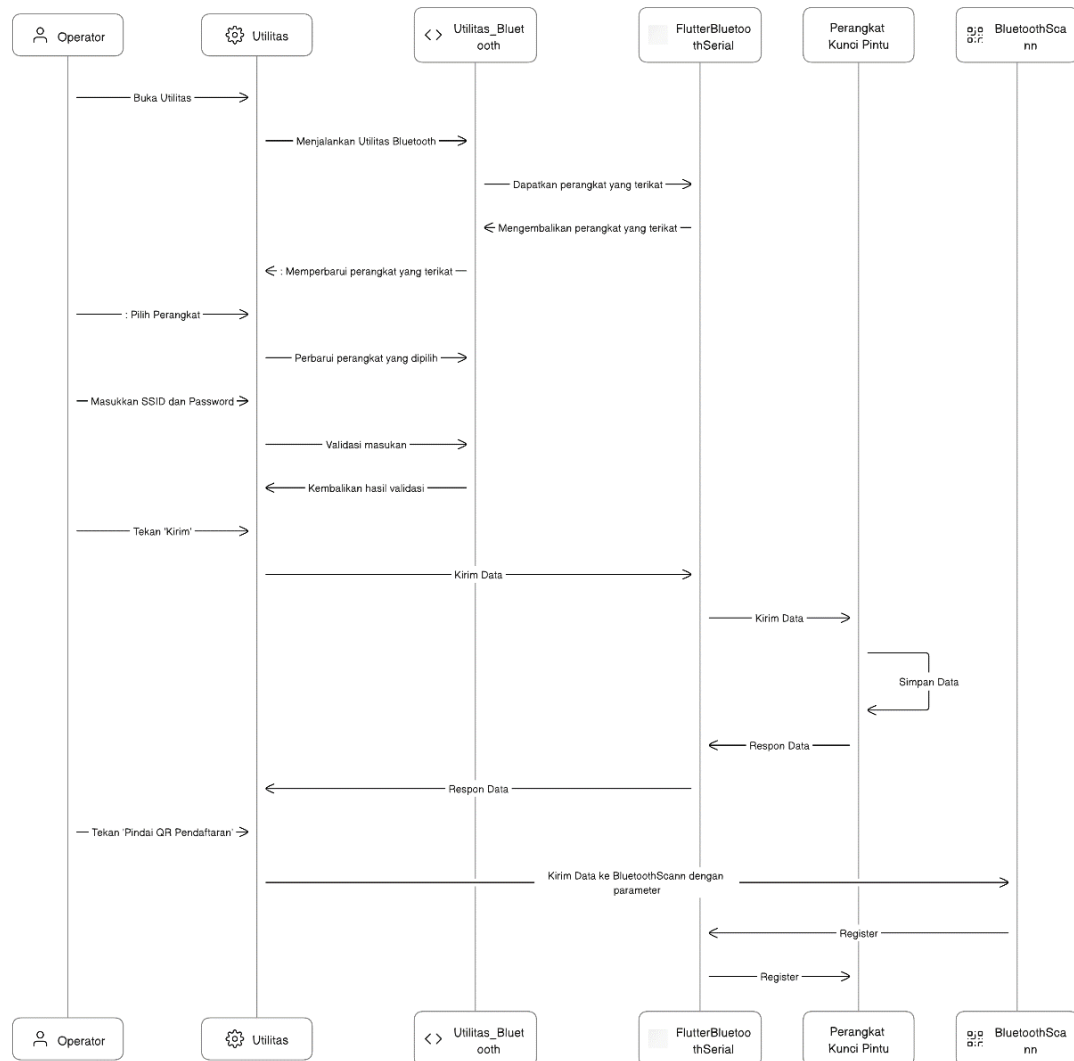
8. Tampilan Menu Riwayat Akses



Gambar 2.45 Diagram *sequence* tampilan halaman riwayat

Gambar 2.45 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah ketika pengguna atau operator membuka halaman riwayat untuk melihat histori aktivitas dalam sistem. Pertama, pengguna atau operator membuka halaman riwayat yang menampilkan daftar histori aktivitas yang ada. Halaman riwayat menghubungi *Network API* untuk meminta histori aktivitas. Setelah menerima permintaan, *Network API* memproses, mengambil data histori dari *server*, dan memberikan respons yang berisi data histori tersebut ke halaman riwayat. Halaman riwayat menampilkan data histori aktivitas ke pengguna atau operator, sehingga pengguna atau operator dapat melihat daftar aktivitas yang telah terjadi dalam sistem. Proses tersebut memastikan bahwa pengguna atau operator dapat dengan mudah melihat dan menelusuri histori aktivitasnya melalui halaman riwayat tanpa kesulitan. Dengan adanya alur tersebut, pengguna atau operator dapat dengan cepat memahami dan melacak aktivitas yang terjadi dalam sistem.

9. Tampilan Menu Utilitas

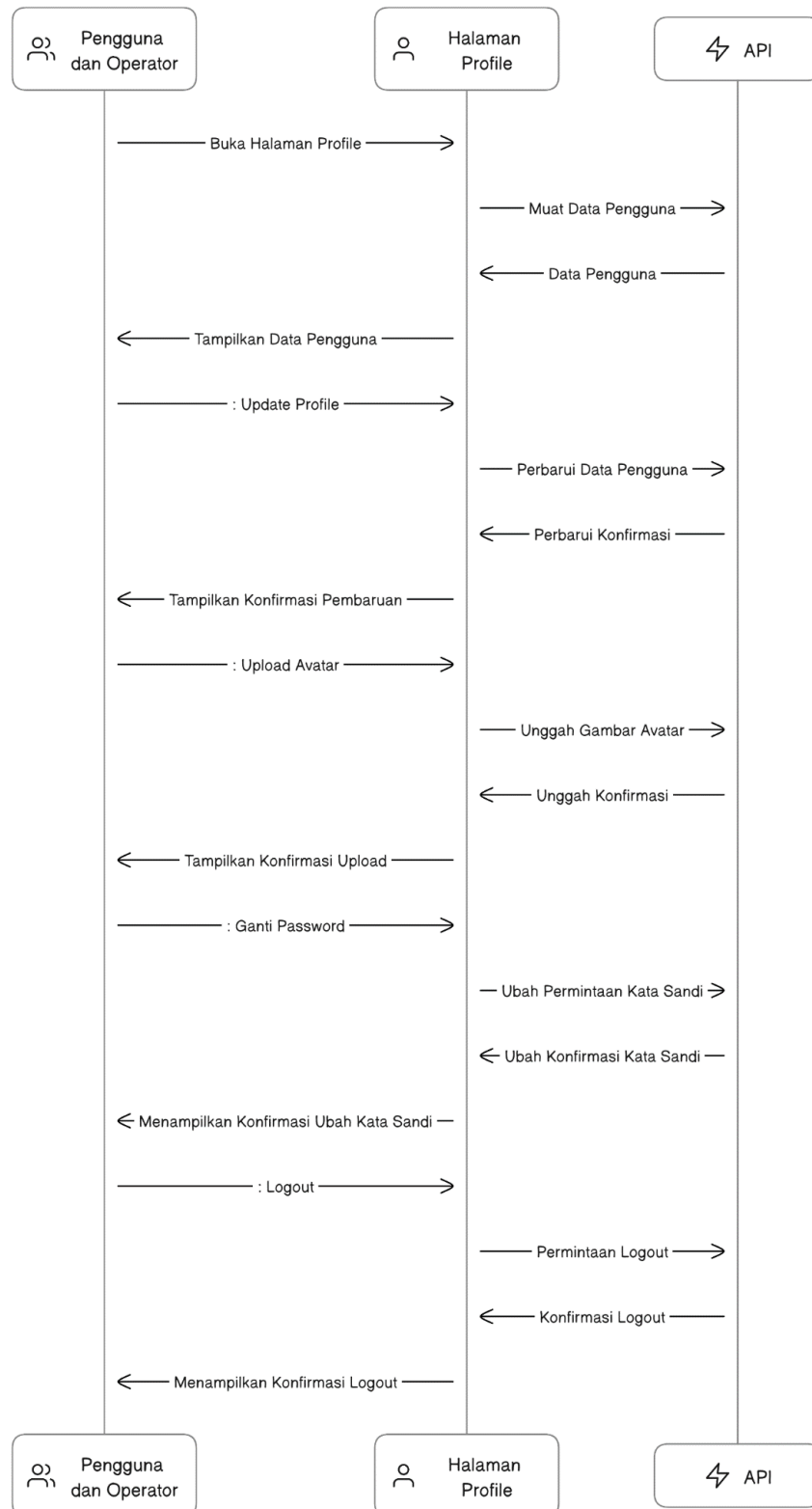


Gambar 2.46 Diagram *sequence* tampilan halaman utilitas

Gambar 2.46 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah yang terjadi ketika operator membuka menu utilitas dan menggunakan fitur utilitas *bluetooth* untuk menghubungkan dan mengatur perangkat kunci pintu. Pertama, operator membuka menu utilitas untuk mengakses fitur utilitas *bluetooth*. Selanjutnya, utilitas *bluetooth* berinteraksi dengan *flutter bluetooth serial* untuk mendapatkan daftar perangkat yang terikat. *Flutter bluetooth serial* mengembalikan daftar perangkat yang terikat ke utilitas *bluetooth*. Kemudian, operator memilih perangkat dari daftar dan menu utilitas meminta utilitas *bluetooth* untuk memperbarui perangkat yang dipilih. Utilitas *bluetooth* berkomunikasi dengan

operator untuk meminta SSID dan *password* untuk konfigurasi perangkat. Setelah operator memasukkan SSID dan *password*, utilitas *bluetooth* melakukan validasi masukan dan mengembalikan hasil validasi ke menu utilitas. Kemudian, operator menekan tombol kirim dan menu utilitas mengirimkan data melalui *flutter bluetooth serial*. *Flutter bluetooth serial* mengirim data ke perangkat kunci pintu untuk melakukan pengaturan. Perangkat kunci pintu menerima data dan menyimpannya. Kemudian, perangkat kunci pintu memberikan respons kembali melalui *flutter bluetooth serial*. *Flutter bluetooth serial* meneruskan respons ke menu utilitas dan operator menerima respons tersebut. Selanjutnya, operator menekan tombol pindai QR pendaftaran, dan menu utilitas mengirim data dengan parameter ke *bluetooth scann*. *Bluetooth scann* melakukan registrasi perangkat kunci pintu. *Flutter bluetooth serial* bertindak sebagai penghubung untuk melakukan registrasi dengan perangkat kunci pintu.

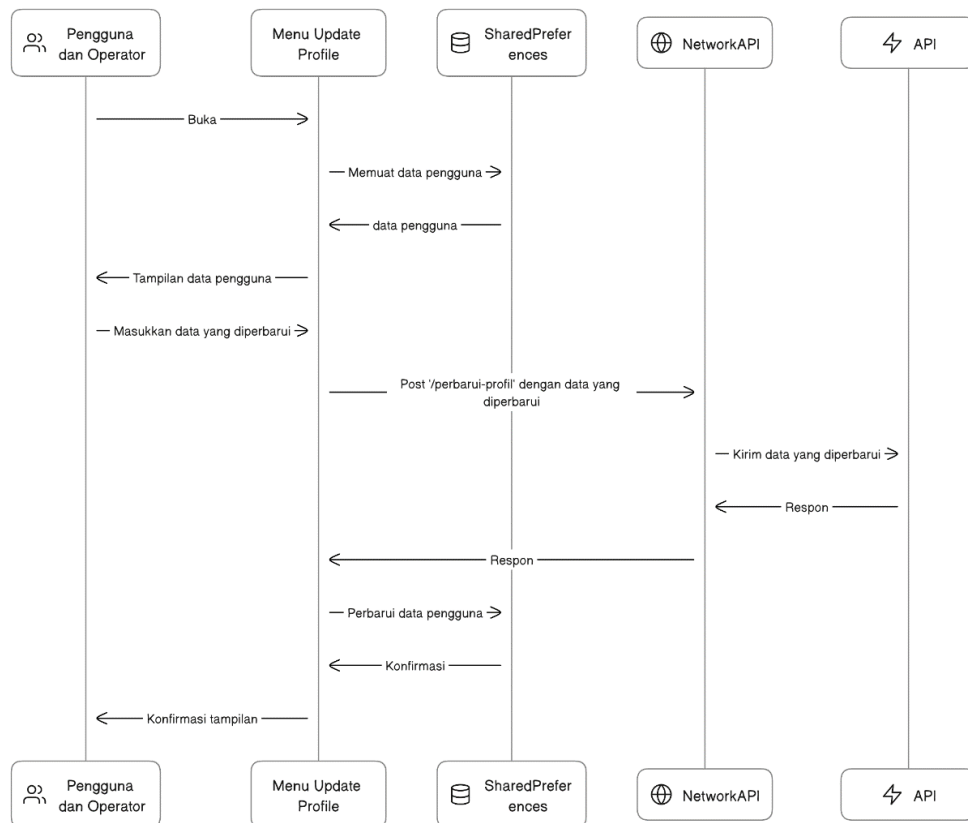
10. Tampilan Menu Profil



Gambar 2.47 Diagram *sequence* tampilan halaman *profile*

Gambar 2.47 di atas merupakan diagram *sequence* yang menggambarkan langkah-langkah yang terjadi ketika pengguna atau operator membuka halaman *profile* untuk mengelola profilnya dalam sistem. Pertama, pengguna atau operator membuka halaman *profile* yang menampilkan informasi profil. Halaman *profile* menghubungi API untuk memuat data pengguna dari *server*. Setelah menerima permintaan, API mengirimkan data pengguna kembali ke halaman *profile*. Halaman *profile* menampilkan data pengguna ke pengguna atau operator sehingga informasi profil ditampilkan. Selanjutnya, pengguna atau operator memiliki opsi untuk memperbarui profil. Jika memilih untuk melakukan perubahan, halaman *profile* akan menghubungi kembali API untuk memperbarui data pengguna. API mengonfirmasi perubahan data pengguna dan mengembalikan konfirmasi ke halaman *profile*. Halaman *profile* menampilkan konfirmasi pembaruan sehingga pengguna dapat melihat bahwa profilnya telah diperbarui. Selain itu, pengguna atau operator juga dapat mengunggah foto untuk profil. Jika memilih untuk melakukannya, halaman *profile* menghubungi API untuk mengunggah gambar foto. API mengonfirmasi proses unggah gambar foto dan mengembalikan konfirmasi ke halaman *profile*. Halaman *profile* menampilkan konfirmasi unggah foto sehingga pengguna atau operator dapat melihat bahwa foto telah berhasil diunggah. Selanjutnya, pengguna atau operator juga dapat mengganti kata sandi. Jika memilih untuk melakukannya, halaman *profile* menghubungi API untuk mengubah kata sandi. API mengonfirmasi proses perubahan kata sandi dan mengembalikan konfirmasi ke halaman *profile*. Halaman *profile* menampilkan konfirmasi perubahan kata sandi sehingga pengguna atau operator dapat melihat bahwa kata sandi telah berhasil diubah. Terakhir, pengguna atau operator dapat melakukan *logout* dari aplikasi. Jika memilih untuk keluar, halaman *profile* menghubungi API untuk melakukan permintaan *logout*. API mengonfirmasi proses *logout* dan mengembalikan konfirmasi ke halaman *profile*. Halaman *profile* menampilkan konfirmasi *logout* sehingga pengguna atau operator dapat melihat bahwa telah berhasil keluar dari aplikasi.

11. Update Profile

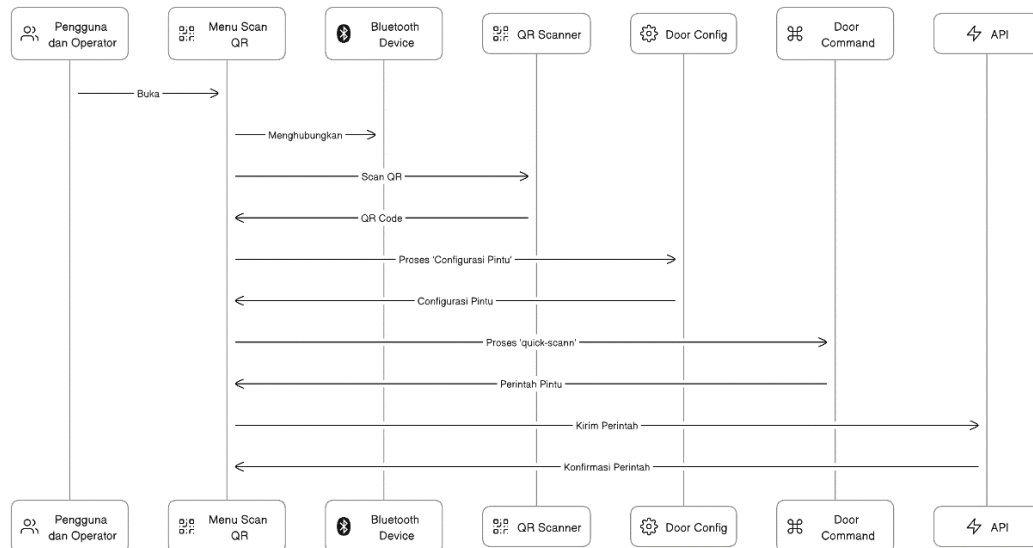


Gambar 2.48 Diagram *sequence* tampilan menu *update profile*

Gambar 2.48 di atas merupakan diagram yang menggambarkan langkah-langkah yang terjadi ketika pengguna atau operator membuka menu *update profile* untuk memperbarui profil dalam sistem. Pertama, pengguna atau operator membuka menu *update profile* yang memuat data pengguna dari *SharedPreferences*. Setelah data pengguna dimuat dari *SharedPreferences*, menu *update profile* menampilkan data tersebut ke menu *update profile* sehingga pengguna atau operator dapat melihat informasi profil yang sudah ada. Selanjutnya, pengguna atau operator memasukkan data yang ingin diperbarui pada menu *update profile*. Setelah data diperbarui, menu *update profile* menghubungi *NetworkAPI* untuk melakukan permintaan POST dengan data yang diperbarui menggunakan *endpoint* *'/perbarui-profil'*. *NetworkAPI* menerima permintaan dari menu *update profile* dan mengirimkan data yang telah diperbarui ke API. API memproses data tersebut dan memberikan respons kembali ke *NetworkAPI*. *NetworkAPI* meneruskan respons dari API ke menu *update profile*. Setelah menerima respons, menu *update profile* memperbarui data pengguna yang

ada di *SharedPreferences* untuk mencerminkan perubahan yang telah dilakukan. Setelah data pengguna diperbarui di *SharedPreferences*, menu *update profile* memberikan konfirmasi kepada pengguna atau operator bahwa data telah berhasil diperbarui.

12. Tampilan Menu *Scan QR Code*



Gambar 2.49 Diagram *sequence* tampilan menu *scan QR Code*

Gambar 2.49 di atas merupakan diagram yang menggambarkan langkah-langkah yang terjadi ketika pengguna atau operator membuka menu *scan QR* untuk melakukan proses *scan QR code* pada aplikasi. Pertama, pengguna atau operator membuka menu *scan QR* yang akan menghubungkan dengan perangkat *bluetooth*. Setelah terhubung dengan perangkat *bluetooth*, menu *scan QR* menggunakan *QR scanner* melakukan pemindaian *QR code*. *QR scanner* mengambil data dari *QR code* yang dipindai dan mengirimkan hasilnya kembali ke menu *scan QR*. Menu *scan QR* menerima data *QR code* dan menghubungkan ke *door config* untuk memproses '*konfigurasi pintu*' berdasarkan *QR code* yang dipindai. *Door config* melakukan proses konfigurasi pintu sesuai dengan data *QR code* dan mengembalikan hasilnya kembali ke menu *scan QR*. Selanjutnya, menu *scan QR* menggunakan *door command* untuk melakukan '*quick-scan*' atau pemindaian cepat pada pintu yang telah dikonfigurasi. *Door command* mengirimkan perintah pintu berdasarkan konfigurasi yang diterima dari *door config* dan mengembalikan

hasilnya kembali ke menu *scan QR*. Menu *scan QR* menggunakan API untuk mengirimkan perintah pintu yang telah dihasilkan oleh *door command*. API memproses perintah tersebut dan memberikan konfirmasi bahwa perintah pintu telah berhasil dikirimkan kembali ke menu *scan QR*.

3. PENUTUP

Dokumen B400 memaparkan proses implementasi yang dilakukan untuk membangun sistem keamanan kunci pintu gedung berbasis IoT. Proses implementasi akan menentukan hasil akhir dari sistem yang dikembangkan. Hasil implementasi juga akan dijadikan acuan untuk proses pengujian sistem selanjutnya.