



UNIVERSITAS DIPONEGORO

**PERANCANGAN SISTEM *DATABASE DAN SERVER* PADA
SISTEM KEAMANAN KUNCI PINTU GEDUNG DENGAN
*ACCESS CONTROL***

TUGAS AKHIR

MUHAMMAD KHOIRIL WAFI

21060119140133

**FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRO
PROGRAM STUDI SARJANA**

**SEMARANG
SEPTEMBER 2023**



UNIVERSITAS DIPONEGORO

**PERANCANGAN SISTEM DATABASE DAN SERVER PADA
SISTEM KEAMANAN KUNCI PINTU GEDUNG DENGAN
*ACCESS CONTROL***

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik

MUHAMMAD KHOIRIL WAFI

21060119140133

**FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRO
PROGRAM STUDI SARJANA**

**SEMARANG
SEPTEMBER 2023**

HALAMAN PERNYATAAN ORISINALITAS

**Tugas Akhir ini adalah karya saya sendiri,
dan semua sumber baik yang dikutip maupun yang dirujuk
telah saya nyatakan dengan benar**

NAMA : MUHAMMAD KHOIRIL WAFI

NIM : 21060119140133

Tanda Tangan :

Tanggal : 6 September 2023

HALAMAN PENGESAHAN

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Diponegoro, saya yang bertanda tangan di bawah ini:

Nama : Muhammad Khoiril Wafi
NIM : 21060119140133
Program Studi : SARJANA (S1)
Departemen : TEKNIK ELEKTRO
Fakultas : TEKNIK
Jenis Karya : TUGAS AKHIR

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Diponegoro **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul: **PERANCANGAN SISTEM DATABASE DAN SERVER PADA SISTEM KEAMANAN KUNCI PINTU GEDUNG DENGAN ACCESS CONTROL**. Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti/Noneksklusif Universitas Diponegoro berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Semarang

Pada Tanggal : 6 September 2023

Yang menyatakan,

(Muhammad Khoiril Wafi)

NIM. 21060119140133

ABSTRAK

Banyak gedung yang masih menggunakan sistem penguncian manual dengan menggunakan kunci fisik sehingga dalam satu gedung akan mempunyai banyak kunci untuk masing-masing pintu hal tersebut menjadikan proses pengelolaan akses pintu kurang optimal. Untuk mengatasi hal tersebut maka dilakukan perancangan mengenai sistem penguncian pintu gedung yang dapat mengoptimalkan pengelolaan akses pintu serta meningkatkan efisiensi penguncian pintu gedung. Sistem yang dibuat menggunakan konsep IoT dimana terdapat beberapa perangkat kunci untuk masing-masing pintu yang terhubung ke sebuah server sebagai kendali dan data yang mengatur semua pintu serta memberikan informasi kepada pihak pengelola gedung tersebut mengenai kondisi pintu pada gedung tersebut. Beberapa fitur seperti pengawasan langsung, pindai kode QR, penjadwalan dan kendali jarak jauh akan disediakan untuk mengoptimalkan pengawasan dan pengelolaan pintu. Sistem yang dibangun terdiri dari backend API menggunakan Laravel, database menggunakan MySQL, penjadwalan menggunakan Crobjob serta komunikasi menggunakan Websocket dimana semua komponen tersebut akan dipasang pada sebuah Virtual Private Server dengan menggunakan sistem operasi Ubuntu.

Kata Kunci: Akses Pintu, kode QR, IoT, Server, Database, API.

ABSTRACT

Many buildings still use a manual locking system using physical keys so that in one building there will be many keys for each door, making the door access management process less than optimal. To overcome this, a design is carried out regarding a building door locking system that can optimize door access management and increase the efficiency of locking building doors. The system created uses the concept of IoT where there are several key devices for each door connected to a server as control and data that regulates all doors and provides information to the building manager about the condition of the doors in the building. Several features such as live monitoring, QR code scanning, scheduling and remote control will be provided to optimize door monitoring and management. The system consists of a backend API using Laravel, a database using MySQL, scheduling using Crobjob and communication using Websocket where all components will be installed on a Virtual Private Server using Ubuntu operating system.

Keywords: *Door Access, QR code, IoT, Server, Database, API.*

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah Subhanahu Wa Ta'ala atas rahmat, karunia, taufik, dan hidayah-Nya, sehingga penulis dapat menyelesaikan tugas akhir dan penyusunan laporan ini. Tugas Akhir berjudul “PERANCANGAN SISTEM DATABASE DAN SERVER PADA SISTEM KEAMANAN KUNCI PINTU GEDUNG DENGAN ACCESS CONTROL” ini diajukan sebagai syarat akhir untuk menyelesaikan program Sarjana di Departemen Teknik Elektro, Fakultas Teknik, Universitas Diponegoro, Semarang. Penyusunan dan penyelesaian Tugas Akhir ini tidak lepas dari bantuan dan dukungan dari semua pihak, baik secara langsung maupun tidak langsung. Oleh karena itu, pada kesempatan kali ini penulis mengucapkan terima kasih kepada:

1. Bapak Aghus Sofwan, S.T., M.T., Ph.D., selaku Ketua Departemen Teknik Elektro Fakultas Teknik Universitas Diponegoro Semarang.
2. Bapak Dr. Munawar Agus Riyadi, S.T., M.T., selaku Ketua Program Studi S1 Teknik Elektro Universitas Diponegoro Semarang.
3. Bapak Yuli Christiyono, S.T., M.T., selaku Sekretaris Program Studi S1 Teknik Elektro Universitas Diponegoro.
4. Bapak M. Arfan, S. Kom., M. Eng., selaku Dosen Pembimbing Utama yang selalu memberikan bimbingan, arahan, dan motivasi dalam menyelesaikan tugas akhir ini.
5. Bapak Imam Santoso, S.T., M.T., selaku Dosen Pendamping yang selalu memberikan motivasi, bimbingan, serta arahan yang membangun kepada penulis.
6. Bapak Yuli Christiyono, S.T., M.T., selaku dosen wali penulis yang selalu memberikan semangat dan mendengarkan dengan baik permasalahan penulis selama menempuh studi di Departemen Teknik Elektro Universitas Diponegoro.
7. Orang tua dan keluarga besar penulis yang selalu memberikan kasih sayang, doa, motivasi, dan semangat kepada penulis.
8. Sahabat kelompok tugas akhir Henric Dhiki Wicaksono dan Novi Dianasari yang banyak membantu penulis dalam menyelesaikan tugas

akhir ini.

9. Keluarga Eternity S1 Teknik Elektro Universitas Diponegoro Angkatan 2019 khususnya konsentrasi Teknologi Informasi, terimakasih atas semuanya selama penulis menimba ilmu di S1 Teknik Elektro Universitas Diponegoro.
10. Dan semua pihak yang tidak dapat penulis sebutkan satu per satu yang telah membantu dari awal hingga akhir.

Penulis menyadari bahwa dalam penulisan laporan Tugas Akhir ini tentunya ada kekurangan. Oleh karena itu, kritik dan saran yang dapat membangun diperlukan demi kebaikan dan kesempurnaan penyusunan laporan di masa yang akan datang. Semoga Tugas Akhir ini dapat memberikan manfaat dan menambahkan pengetahuan bagi kita.

Semarang, 23 September 2023

(Muhammad Khoiril Wafi)

NIM. 21060119140133

DAFTAR ISI

HALAMAN PERNYATAAN ORISINALITAS	ii
HALAMAN PENGESAHAN	iii
ABSTRAK	v
ABSTRACT	vi
KATA PENGANTAR	vii
DAFTAR ISI	ix
DAFTAR GAMBAR	xiii
DAFTAR TABEL	xvi
DAFTAR SINGKATAN	xvii
BAB I	1
1.1 Latar Belakang	1
1.2 Tujuan	1
1.3 Batasan Masalah	2
1.4 Sistematika Penulisan	2
BAB II	4
2.1 Kendali Akses (Access Control)	4
2.2 Internet of Things	5
2.3 <i>Database MySQL</i>	7
2.4 <i>Virtual Private Server</i>	8
2.5 Sistem Operasi Ubuntu	9
2.6 Nginx	11
2.7 <i>Application Programming Interface</i>	11
2.8 <i>Framework Laravel</i>	12
2.9 <i>Pusher Websocket</i>	14
2.10 Penelitian Terdahulu	16
BAB III	17
3.1 Metode Perancangan	17
3.2 Kebutuhan Fungsional dan Non-Fungsional	19
3.3 Perancangan <i>Database</i>	21
3.4 Perancangan <i>Backend API</i>	23

3.3.1	Diagram <i>Use Case API</i>	23
3.3.2	<i>Endpoint API</i>	25
3.3.3	<i>API Login</i>	27
3.3.4	<i>API Logout</i>	28
3.3.5	<i>API Verifikasi Email</i>	28
3.3.6	<i>API Reset Password</i>	29
3.3.7	<i>API Ganti Password</i>	31
3.3.8	<i>API Ganti Profil</i>	31
3.3.9	<i>API Lihat Avatar</i>	32
3.3.10	<i>API Ganti Avatar</i>	33
3.3.11	<i>API Verifikasi Akses</i>	34
3.3.12	<i>API Daftar Akses</i>	35
3.3.13	<i>API Riwayat Akses</i>	36
3.3.14	<i>API Daftar Pintu</i>	37
3.3.15	<i>API Remote Pintu</i>	37
3.3.16	<i>API Door Login</i>	38
3.3.17	<i>API Door Logout</i>	39
3.3.18	<i>API Door Register</i>	40
3.3.19	<i>API Door Signature</i>	41
3.3.20	<i>API Door Update Status</i>	42
3.3.21	<i>API Door Alert</i>	43
3.5	Perancangan Penjadwalan	43
3.4.1	Pengecekan Jadwal	44
3.4.2	Atur Ulang Jadwal	45
3.6	Perancangan Komunikasi Websocket	46
3.7	Perancangan Server	47
BAB IV	50
4.1	Kajian Hasil Perancangan	50
4.1.1	<i>API Login</i>	50
4.1.2	<i>API Logout</i>	52
4.1.3	<i>API Verifikasi Email</i>	52
4.1.4	<i>API Reset Password</i>	54

4.1.5	API Ganti <i>Password</i>	55
4.1.6	API Ganti Profil	56
4.1.7	API Lihat Avatar.....	56
4.1.8	API Ganti Avatar	57
4.1.9	API Verifikasi Akses	58
4.1.10	API Daftar Akses	59
4.1.11	API Riwayat Akses	60
4.1.12	API Daftar Pintu	61
4.1.13	API <i>Remote</i> Pintu.....	62
4.1.14	API <i>Door Login</i>	64
4.1.15	API <i>Door Logout</i>	65
4.1.16	API <i>Door Register</i>	65
4.1.17	API <i>Door Signature</i>	66
4.1.18	API <i>Door Update Status</i>	67
4.1.19	API <i>Door Alert</i>	68
4.1.20	Pengecekan Jadwal	70
4.1.21	Atur Ulang Jadwal	71
4.1.22	Komunikasi Websocket	72
4.1.23	Koneksi HTTPS	73
4.2	Pengujian	74
4.3	Pengujian Fungsional	74
4.3.1	Fungsi <i>Login</i>	75
4.3.2	Fungsi <i>Logout</i>	76
4.3.3	Fungsi Verifikasi Email	77
4.3.4	Fungsi Ganti <i>Password</i>	78
4.3.5	Fungsi Reset <i>Password</i>	79
4.3.6	Fungsi <i>Update Profil</i>	80
4.3.7	Fungsi <i>Update Avatar</i>	81
4.3.8	Fungsi Lihat Akses	81
4.3.9	Fungsi Lihat Pintu.....	82
4.3.10	Fungsi Lihat Riwayat Akses	83
4.3.11	Fungsi Verifikasi Akses.....	83

4.3.12	Fungsi <i>Signature</i>	84
4.3.13	Komunikasi Websocket	85
4.3.14	Fungsi <i>Update Status Pintu</i>	86
4.3.15	Fungsi Peringatan Pintu	86
4.4	Pengujian Non-Fungsional	87
4.4.1	Pengujian Performa Verifikasi Akses.....	88
4.4.2	Pengujian Performa Update Status Pintu.....	89
4.4.3	Pengujian Performa Penjadwalan	89
4.5	Pembahasan	90
BAB V	92
5.1	Kesimpulan.....	92
5.2	Saran	93
DAFTAR PUSTAKA	94
BIODATA MAHASISWA	97
LAMPIRAN A	98
LAMPIRAN B	99
LAMPIRAN C	120

DAFTAR GAMBAR

Gambar 2.1 Contoh Pengendalian Akses	5
Gambar 2.2 Arsitektur Sistem IoT	6
Gambar 2.3 Struktur Tabel MySQL	8
Gambar 2.4 Perbedaan <i>Server</i> Tradisional dan Virtual	9
Gambar 2.5 Contoh Konfigurasi <i>Firewall</i> Ubuntu.....	10
Gambar 2.6 Contoh Konfigurasi Cronjob	10
Gambar 2.7 <i>Virtual Host</i> pada Nginx	11
Gambar 2.8 Cara Kerja API	12
Gambar 2.9 Konsep MVC pada Laravel	13
Gambar 2.10 Diagram Pusher <i>Channel</i>	15
Gambar 3.1 Alur Perancangan	17
Gambar 3.2 ERD Database.....	22
Gambar 3.3 Diagram Use Case API.....	24
Gambar 3.4 <i>Sequence Diagram</i> API <i>Login</i>	27
Gambar 3.5 <i>Sequence Diagram</i> API <i>Logout</i>	28
Gambar 3.6 <i>Sequence Diagram</i> API Verifikasi Email	29
Gambar 3.7 <i>Sequence Diagram</i> API Reset <i>Password</i>	30
Gambar 3.8 <i>Sequence Diagram</i> API Ganti Password.....	31
Gambar 3.9 <i>Sequence Diagram</i> API Ganti Profil.....	32
Gambar 3.10 <i>Sequence Diagram</i> API Lihat Avatar	32
Gambar 3.11 <i>Sequence Diagram</i> API Ganti Avatar	33
Gambar 3.12 <i>Sequence Diagram</i> API Verifikasi Akses	34
Gambar 3.13 <i>Sequence Diagram</i> API Daftar Akses.....	35
Gambar 3.14 <i>Sequence Diagram</i> API Riwayat Akses.....	36
Gambar 3.15 <i>Sequence Diagram</i> API Daftar Pintu	37
Gambar 3.16 <i>Sequence Diagram</i> API Remote Pintu.....	38
Gambar 3.17 <i>Sequence Diagram</i> API <i>Door Login</i>	39
Gambar 3.18 <i>Sequence Diagram</i> API <i>Logout</i>	39
Gambar 3.19 <i>Sequence Diagram</i> API <i>Door Register</i>	40
Gambar 3.20 <i>Sequence Diagram</i> API <i>Door Signature</i>	41

Gambar 3.21 Sequence Diagram API <i>Door Update Status</i>	42
Gambar 3.22 Sequence Diagram API <i>Door Alert</i>	43
Gambar 3.23 Sequence Diagram Pengecekan Jadwal	44
Gambar 3.24 Sequence Diagram Atur Ulang Jadwal	45
Gambar 3.25 Konfigurasi Websocket	46
Gambar 3.26 Konfigurasi Server.....	47
Gambar 4.1 Hasil API <i>Login</i>	50
Gambar 4.2 Hasil API <i>Login</i> Email Belum Terverifikasi	51
Gambar 4.3 Email Kode OTP	51
Gambar 4.4 Hasil API <i>Logout</i>	52
Gambar 4.5 Hasil API Verifikasi Email	53
Gambar 4.6 Kode OTP Kadaluarsa	53
Gambar 4.7 Hasil API <i>Reset Password</i>	54
Gambar 4.8 Email Reset Password.....	54
Gambar 4.9 Hasil API Ganti <i>Password</i>	55
Gambar 4.10 API Ganti Profil	56
Gambar 4.11 Hasil API Lihat Avatar	57
Gambar 4.12 Hasil API Ganti Avatar.....	58
Gambar 4.13 Hasil API Verifikasi Akses.....	59
Gambar 4.14 Hasil API Daftar Akses	60
Gambar 4.15 API Riwayat Akses.....	61
Gambar 4.16 Hasil API Daftar Pintu.....	62
Gambar 4.17 Hasil API <i>Remote</i> Pintu	63
Gambar 4.18 Aktifitas <i>Door Remote</i>	63
Gambar 4.19 Hasil API <i>Door Login</i>	64
Gambar 4.20 Hasil API <i>Door Logout</i>	65
Gambar 4.21 Hasil API <i>Door Register</i>	66
Gambar 4.22 Hasil API <i>Door Signature</i>	67
Gambar 4.23 Hasil API <i>Door Update Status</i>	68
Gambar 4.24 Hasil API <i>Door Alert</i>	69
Gambar 4.25 Aktifitas <i>Door Alert</i>	69
Gambar 4.26 Pengecekan Jadwal	70

Gambar 4.27 Atur Ulang Jadwal	71
Gambar 4.28 Koneksi Websocket Melalui <i>Endpoint</i>	72
Gambar 4.29 Proses <i>Subscribe</i> ke <i>Channel</i> Websocket	72
Gambar 4.30 <i>Request</i> Menggunakan HTTPS	73
Gambar 4.31 <i>Request</i> Menggunakan HTTP.....	74
Gambar 4.32 Hasil Pengujian Performa API.....	87
Gambar 4.33 Grafik Hasil Pengujian Performa Verifikasi Akses.....	88
Gambar 4.34 Hasil Pengujian Performa Update Status Pintu	89
Gambar 4.35 Hasil Pengujian Performa Penjadwalan.....	90

DAFTAR TABEL

Tabel 3.1 Kebutuhan Fungsional	20
Tabel 3.2 Kebutuhan Non-Fungsional	21
Tabel 3.3 <i>Endpoint API</i>	25
Tabel 4.1 Hasil Pengujian Fungsional.....	75
Tabel 4.2 Hasil Pengujian Fungsi <i>Login</i>	75
Tabel 4.3 Hasil Pengujian Fungsi <i>Logout</i>	77
Tabel 4.4 Hasil Pengujian Fungsi Verifikasi Email	77
Tabel 4.5 Hasil Pengujian Fungsi Ganti <i>Password</i>	78
Tabel 4.6 Hasil Pengujian Fungsi Reset <i>Password</i>	79
Tabel 4.7 Hasil Pengujian Fungsi <i>Update Profil</i>	80
Tabel 4.8 Hasil Pengujian Fungsi <i>Update Avatar</i>	81
Tabel 4.9 Hasil Pengujian Fungsi Lihat Akses	82
Tabel 4.10 Hasil Pengujian Fungsi Lihat Pintu.....	82
Tabel 4.11 Hasil Pengujian Lihat Riwayat Akses	83
Tabel 4.12 Hasil Pengujian Fungsi Verifikasi Akses.....	84
Tabel 4.13 Hasil Pengujian Fungsi <i>Signature</i>	84
Tabel 4.14 Hasil Pengujian Websocket.....	85
Tabel 4.15 Hasil Pengujian Fungsi Update Status Pintu	86
Tabel 4.16 Hasil Pengujian Fungsi Peringatan Pintu	87

DAFTAR SINGKATAN

Singkatan	Arti
API	<i>Application Programming Interface</i>
HTTP	<i>Hypertext Transfer-Transfer Protocol</i>
QR	<i>Quick Response</i>
ACL	<i>Access Control List</i>
IoT	<i>Internet of Things</i>
SQL	<i>Structured Query Language</i>
DB	<i>Database</i>
VPS	<i>Virtual Private Server</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SSL	<i>Secure Socket Layer</i>
ERD	<i>Entity Relation Diagram</i>

BAB I

PENDAHULUAN

1.1 Latar Belakang

Keamanan menjadi hal yang harus diperhatikan dalam sebuah gedung atau bangunan. Pada saat ini sistem penguncian masih banyak menggunakan penguncian tradisional dengan menggunakan kunci fisik yang tidak efisien mengingat jumlah ruangan yang banyak, kunci fisik juga mempunyai tingkat keamanan yang kurang dikarenakan kunci rentan untuk dicuri atau diduplikasi[1]. Mengelola sejumlah besar ruangan menjadi tantangan tersendiri. Seiring bertambahnya ruangan yang harus dikelola maka bertambah juga kompleksitas tugas yang harus dikerjakan seperti mengatur siapa saja yang dapat mengakses ruangan tersebut, bertambahnya ruangan juga meningkatkan resiko kunci hilang atau tertukar sehingga ruangan tidak bisa diakses[2].

Masalah keamanan ruangan dalam sebuah gedung dan efektifitas dapat diselesaikan dengan menggunakan sebuah sistem penguncian cerdas yang terorganisasi dan kerkoneksi ke sebuah sistem manajemen kunci pintu (*access control*) yang memiliki tingkat keamanan yang lebih tinggi, adaptif dan fleksibel[3], [4].

Untuk mendukung kinerja dari sistem penguncian yang terorganisasi maka diperlukan sebuah *server* dan penyimpanan data. sebuah *server* akan menjalankan kode program yang bertugas untuk mengatur dan mengawasi semua aktifitas sistem penguncian dan sebuah penyimpanan data digunakan untuk menyimpan data-data seperti data pengguna, kunci, dan *backup*[5].

Dengan demikian, perancangan sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan *access control* diharapkan dapat memberikan solusi yang efektif dalam meningkatkan keamanan kunci pintu gedung dan memberikan kenyamanan serta kemudahan dalam pengelolaannya.

1.2 Tujuan

Tujuan tugas akhir ini adalah untuk merancang sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan *access control*.

1.3 Batasan Masalah

Dalam tugas akhir ini, telah ditentukan batasan-batasan masalah sebagai berikut:

1. Perancangan sistem *database* dan *server* akan difokuskan pada aplikasi *access control* untuk kunci pintu gedung. Hal ini meliputi perancangan struktur *database* untuk menyimpan informasi pengguna, akses pintu gedung, dan riwayat aktivitas akses serta perancangan server untuk menjalankan sistem penguncian.
2. Perancangan sistem *server* akan berfokus pada pemrosesan data dan komunikasi antara sistem *access control* dan perangkat keras.
3. Sistem keamanan kunci pintu gedung dengan *access control* akan mencakup teknologi seperti perangkat lunak *access control* untuk mengatur akses pengguna.
4. Sistem keamanan akan diimplementasikan pada gedung yang memiliki pintu masuk terbatas dengan akses terbatas pada karyawan atau pihak tertentu.
5. Penyimpanan data pengguna dan aktivitas akses akan disimpan dengan cara yang aman dan terenkripsi untuk memastikan keamanan informasi.
6. Pengembangan sistem *access control* harus memenuhi standar keamanan dan regulasi yang berlaku dalam industri atau lingkungan operasional gedung.
7. Sistem keamanan *access control* harus mudah dioperasikan dan dikelola oleh administrator gedung dengan tingkat keamanan dan kontrol yang optimal.

1.4 Sistematika Penulisan

Sistematika penulisan dalam laporan Tugas Akhir ini adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi latar belakang, tujuan, batasan masalah, dan sistematika penulisan.

BAB II DASAR TEORI

Bab ini berisi teori yang melandasi dalam perancangan.

BAB III PERANCANGAN SISTEM

Bab ini berisi tentang perancangan sistem *database* dan *server* serta sistem keamanan kunci pintu gedung dengan *access control*.

BAB IV PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan pengujian dan analisis hasil pengujian.

BAB V PENUTUP

Bab ini berisi kesimpulan dan saran terhadap hasil pengujian dan analisis yang telah dibuat.

BAB II

DASAR TEORI

2.1 Kendali Akses (Access Control)

Kendali akses atau *access control* merupakan sebuah mekanisme pengaturan kebijakan yang digunakan untuk membatasi dan mengatur hak akses pengguna terhadap suatu sumber daya atau fasilitas tertentu. Kendali akses dapat diaplikasikan secara *hardware* maupun *software* yang akan menjadi pintu masuk pengguna sebelum mengakses suatu fasilitas. Dengan menggunakan kendali akses kita bisa mengatur dan membatasi akses pengguna sehingga hanya pengguna tertentu yang diijinkan yang bisa mengakses sumber daya yang dilindungi[6].

Terdapat beberapa pendekatan yang biasanya digunakan untuk membatasi akses pengguna untuk mengakses suatu sumber daya yang dimiliki, diantaranya yaitu :

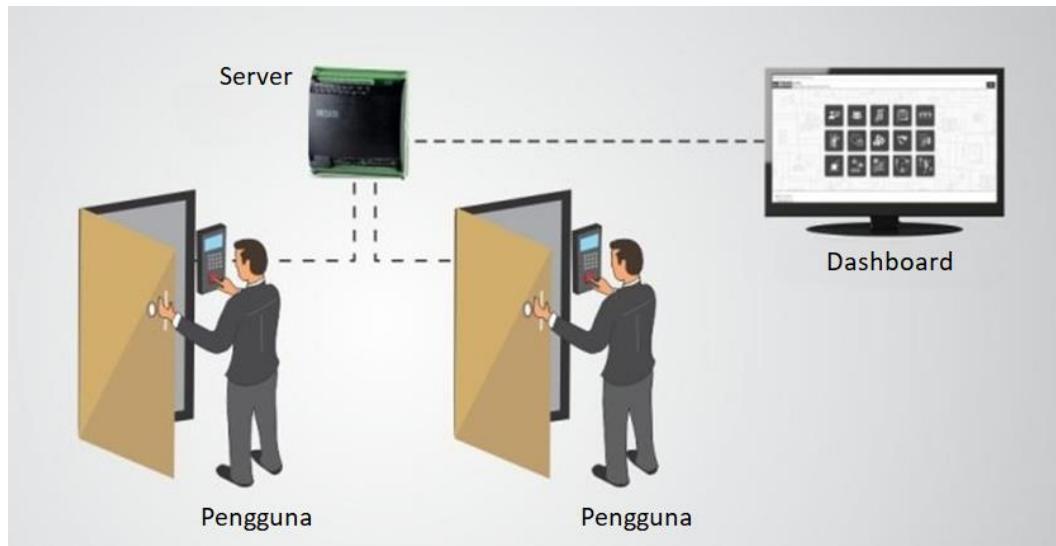
1. Access Control List

ACL atau *Access Control List* merupakan sebuah daftar yang berisi aturan-aturan atau daftar pengguna yang diijinkan untuk mengakses sebuah sumber daya yang dilindungi[7]. Dengan menggunakan ACL kita dapat membatasi hak akses dengan hanya mengijinkan pengguna yang terdapat didalam daftar untuk mengakses sumber daya atau fasilitas tertentu. ACL akan memberikan pembatasan akses pada setiap pengguna secara ketat dan hanya mengijinkan akses jika pengguna memiliki hak akses yang tersimpan didalam daftar akses.

2. Role-based Access Control

RBAC atau *Role-based Access Control* merupakan pendekatan kendali akses yang tidak memberikan hak akses langsung ke pengguna melainkan memberikan hak akses berdasarkan peran yang dimiliki oleh pengguna seperti *admin*, *operator*, *supervisor*, dan lain sebagainya. Metode pengendalian akses berbasis peran akan memberikan aturan-aturan terkait akses berdasarkan peran pengguna tanpa mengetahui identitas pengguna secara spesifik dan hanya menggunakan peran pengguna sebagai

parameter yang digunakan untuk melakukan pengendalian akses sumber daya.



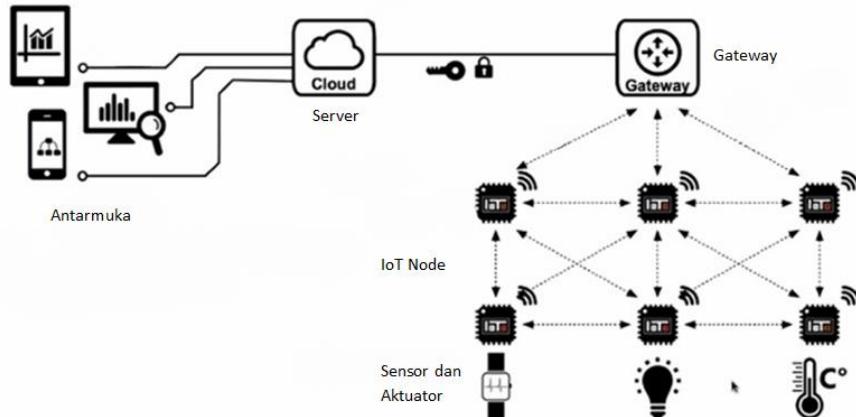
Gambar 2.1 Contoh Pengendalian Akses

Gambar 2.1 merupakan contoh pengendalian akses pada pintu untuk membatasi akses pengguna ke suatu ruangan. Setiap pengguna memiliki identitas unik yang digunakan untuk menentukan tindakan apakah pengguna tersebut diijinkan atau tidak, proses perijinan dilakukan secara otomatis dan dikendalikan oleh sebuah *server* sebagai pusat kendali dan pusat data. Pada sistem pengendalian akses juga terdapat sebuah *software* yang digunakan sebagai antar muka untuk melakukan pengelolaan hak akses seperti mengijinkan pengguna baru atau menghapus ijin pengguna dan juga melihat riwayat akses pengguna.

2.2 Internet of Things

Internet of Things atau IoT merupakan sebuah konsep yang digunakan untuk mengembangkan konesifitas internet, IoT juga memberikan gambaran mengenai kemampuan dari berbagai perangkat elektronik yang saling terhubung dengan membentuk sebuah jaringan komunikasi baik melalui internet maupun komunikasi lainnya seperti bluetooth. Dengan menggunakan konsep IoT kita dapat menghubungkan peralatan seperti sensor dan aktuator yang terhubung ke ke

sebuah jaringan menjadi sebuah kesatuan sistem yang dapat dikendalikan secara efektif dan efisien dengan tingkat kerumitan yang rendah[8].



Gambar 2.2 Arsitektur Sistem IoT

Gambar 2.2 menjelaskan arsitektur dari sistem IoT, sebuah sistem IoT dibangun dari beberapa bagian yang membentuk suatu sistem yang dapat berkomunikasi antara perangkat IoT dan juga server. Bagian-bagian dari sistem IoT diantaranya :

1. Sensor dan Aktuator

Sensor dan aktuator akan menjadi penghubung yang menjembatani interaksi antara sistem dengan dunia luar. Sebuah sensor akan melakukan akuisisi data yang merupakan sumber data dari sebuah sistem IoT sedangkan aktuator akan menerima data dan memberikan keluaran mekanis seperti menyalaakan lampu, membuka kunci dan lain sebagainya.

2. IoT Node

IoT node bertindak sebagai penghubung antara perangkat keras seperti mikrokontroler kedalam sebuah jaringan IoT dengan adanya IoT node memungkinkan antar perangkat IoT untuk saling berkomunikasi satu dengan lainnya secara independen[8].

3. Gateway

Pada sistem IoT umumnya akan terhubung ke sebuah server sebagai pusat kendali dan pusat data. Gateway menjadi penghubung antara IoT node dengan jaringan internet global, sebuah gateway misalnya router dan

modem akan menyediakan koneksi internet yang dapat digunakan oleh perangkat IoT untuk terhubung ke *server* melalui koneksi internet[8].

4. *Server*

Server akan menjadi pusat kendali dan penyimpanan data terpusat pada sistem IoT. Dengan adanya sebuah *server* memungkinkan perangkat IoT untuk saling saling berkoordinasi dengan *server* sebagai pengaturnya, dengan adanya *server* juga memungkinkan perangkat IoT melakukan penyimpanan data dan informasi secara *online* pada *database*[8].

5. Antarmuka

Antarmuka pada sistem IoT digunakan untuk melakukan proses pemantauan aktifitas dan data yang terjadi didalam sistem IoT. Antarmuka akan memberikan akses kepada pengguna untuk berinteraksi dengan sistem IoT seperti melihat data dan memberikan perintah ke perangkat IoT.

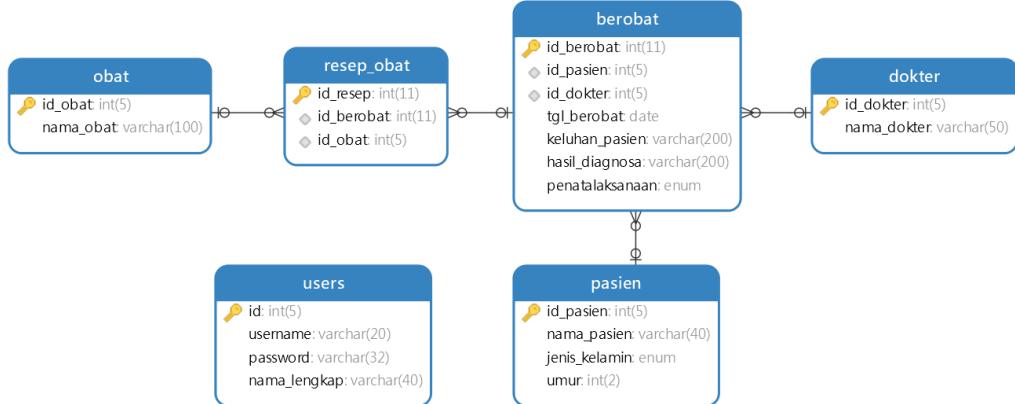
2.3 *Database MySQL*

Database atau basisdata merupakan sekumpulan data yang terintegrasi dan diatur sedemikian rupa sehingga data tersebut dapat dicari, diambil, ditambahkan, dan diolah dengan tepat[6]. Selain berisikan data, *database* juga berisikan informasi lainnya seperti tipe data, nama-nama kolom dan baris. Adapun fungsi lain dari *database* yaitu[9] :

1. Mempermudah identifikasi data dengan cara pengelompokan data, salah satu contohnya dengan pembuatan beberapa tabel atau *field* yang berbeda-beda.
2. Meminimalisir suatu data ganda.
3. Penyimpanan secara digital.
4. Menjadi alternatif lain terkait masalah penyimpanan ruang dalam suatu aplikasi.

MySQL merupakan singkatan dari *Structured Query Language*. SQL merupakan bahasa terstruktur yang khusus digunakan untuk mengolah database. MySQL merupakan sistem manajemen *database* yang bersifat relational. Artinya,

data yang dikelola dalam *database* akan diletakkan pada beberapa tabel yang terpisah sehingga manipulasi data akan jauh lebih cepat[10].

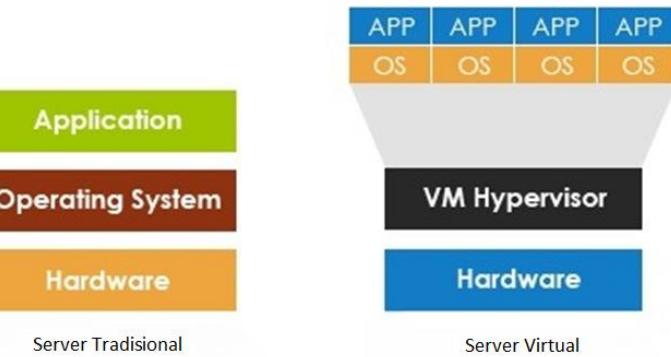


Gambar 2.3 Struktur Tabel MySQL

MySQL merupakan sistem database relasional yang berarti pada MySQL data akan disimpan dalam tabel-tabel yang terdiri dari beberapa kolom, Gambar 2.3 menjelaskan strukur data yang tersimpan didalam *database* MySQL, data yang disimpan dapat berupa teks, angka, *boolean*, waktu, hari dan lain sebagainya tergantung dengan tipe data pada setiap kolomnya. Dalam MySQL terdapat relasi antar tabel yang memberikan informasi hubungan antara tabel satu dengan tabel lainnya serta perlakunya terhadap perubahan yang terjadi pada tabel induknya.

2.4 *Virtual Private Server*

Virtual Private Server atau VPS adalah jenis server yang secara eksklusif diperuntukkan bagi satu pengguna, sehingga seluruh sumber daya yang ada di dalamnya tidak dipengaruhi atau dibagi dengan pengguna lain. Dengan menggunakan teknologi VPS, sebuah mesin fisik dapat menjalankan beberapa sistem operasi secara bersamaan. Pengguna VPS memiliki kendali penuh untuk mengatur seluruh konfigurasi sesuai kebutuhan. Teknologi yang digunakan dalam VPS adalah virtualisasi hardware pada server fisik yang memungkinkan pembagian sumber daya menjadi beberapa bagian yang berbeda, sehingga setiap VPS berfungsi seperti server pribadi yang terisolasi dari pengguna lainnya[11].



Gambar 2.4 Perbedaan *Server* Tradisional dan Virtual

Gambar 2.4 memperlihatkan perbedaan antara *server* tradisional dengan *server* virtual, pada *server* tradisional satu *server* fisik hanya bisa digunakan untuk menjalankan satu sistem operasi dan satu aplikasi sedangkan pada *server* virtual memungkinkan menjalankan beberapa sistem operasi dan aplikasi yang banyak didalam satu *server* fisik. Perbedaan utama yaitu adanya virtualisasi pada *server* virtual yang bertugas untuk membagi sumber daya *server* fisik menjadi beberapa bagian untuk menjalankan sistem operasi. Proses pembagian ini dilakukan menggunakan perangkat lunak, sehingga satu *server* fisik dapat menampung beberapa VPS yang berjalan secara terpisah. Setiap VPS memiliki akses penuh (*Full Root Access*) untuk mengatur sistem operasi dan konfigurasi sesuai keinginannya, termasuk pengaturan *init script*, pengguna, pemrosesan data, sistem file, serta sumber daya server seperti CPU dan RAM yang beroperasi secara independen.

2.5 Sistem Operasi Ubuntu

Ubuntu adalah salah satu distribusi Linux yang didasarkan pada Debian. Distro ini menyediakan sistem operasi berbasis Debian dengan jadwal rilis yang teratur dan dukungan yang luas, baik untuk pengguna perorangan maupun perusahaan. Selain digunakan sebagai sistem operasi desktop, Ubuntu juga populer digunakan sebagai sistem operasi pada *server*, karena memiliki karakteristik yang ringan dan dapat diandalkan dalam menangani berbagai tugas *server*[12]. Ubuntu menyediakan fitur yang cukup lengkap untuk diaplikasikan pada *server* seperti adanya *firewall* dan penjadwalan menggunakan cronjob.

```

user@localhost:~$ 
user@localhost:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                      Action    From
--                      --        --
22/tcp                  ALLOW IN  Anywhere
Anywhere on eth0        ALLOW IN  192.168.0.0/16
25/tcp                  ALLOW IN  Anywhere
80/tcp                  DENY IN   Anywhere
22/tcp (v6)             ALLOW IN  Anywhere (v6)
25/tcp (v6)             ALLOW IN  Anywhere (v6)
80/tcp (v6)             DENY IN   Anywhere (v6)

10.0.0.0/8              ALLOW OUT  Anywhere on eth1

```

Gambar 2.5 Contoh Konfigurasi *Firewall* Ubuntu

Gambar 2.5 merupakan contoh konfigurasi *firewall* pada sistem operasi ubuntu. Di dalam *firewall* semua komunikasi yang keluar dan masuk akan difilter atau dikontrol, *port-port* yang rentan servisnya dapat di tutup atau diblokir, sehingga hanya pihak yang diizinkan saja yang boleh lewat. Cara ini merupakan salah satu pengamanan jaringan yang sering digunakan[13]. Selain *firewall* ubuntu juga menyediakan sistem penjadwalan dengan menggunakan cronjob.

```

# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
* * * * * php /var/www/html/yii2advanced/yii makefile/make

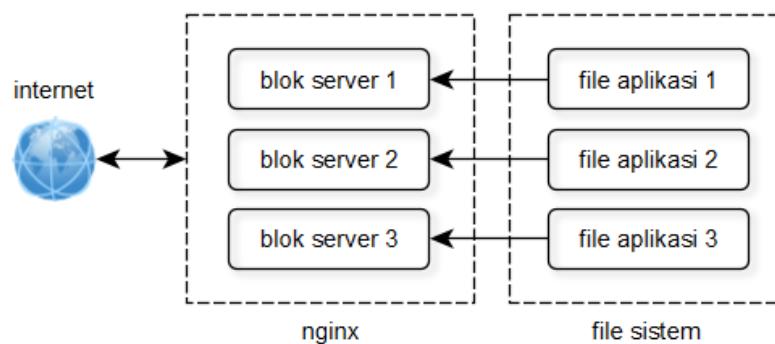
```

Gambar 2.6 Contoh Konfigurasi Cronjob

Gambar 2.6 merupakan contoh konfigurasi cronjob pada sistem operasi ubuntu, dengan menggunakan cronjob maka dimungkinkan untuk mengeksekusi perintah pada terminal secara berulang setiap rentang waktu tertentu seperti setiap 1 minggu sekali atau 1 jam sekali.

2.6 Nginx

Nginx merupakan perangkat lunak *open-source* yang memiliki performa tinggi sebagai server HTTP dan *reverse proxy*. Nginx memiliki keunggulan dalam memberikan konten statis secara cepat, sambil menggunakan sumber daya sistem dengan efisien. Selain itu, Nginx dapat mendistribusikan konten HTTP dinamis melalui jaringan menggunakan FastCGI *handler* untuk menjalankan *script*[14]. Nginx dibangun secara modular, dengan demikian nginx mampu mendukung berbagai fitur seperti *Load Balancing* dan *Reverse Proxying*, *Virtual hosts*.

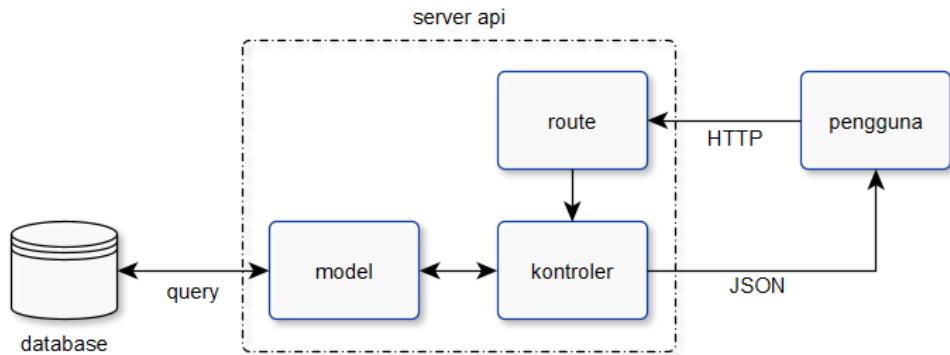


Gambar 2.7 *Virtual Host* pada Nginx

Gambar 2.7 menjelaskan penggunaan virtual host pada nginx, dengan menggunakan virtual host nginx dapat menangani beberapa aplikasi yang sudah dikonfigurasi pada masing-masing *port*-nya. Nginx menggunakan pendekatan *asynchronous-event* untuk menangani permintaan sehingga nginx tidak sepenuhnya bergantung pada ‘*thread*’ untuk menangani permintaan (*request*). Arsitektur ini memberikan hasil kinerja nginx pada saat dibebani hanya memerlukan sedikit memori dalam jumlah yang bisa diprediksikan[14].

2.7 Application Programming Interface

Application Programming Interface atau API merupakan sebuah protokol yang digunakan untuk menampilkan layanan atau data yang disediakan oleh sebuah aplikasi melalui sumber daya yang telah ditentukan. Dengan menggunakan API, aplikasi lain dapat mengakses layanan atau data tanpa harus mengimplementasikan prosedur atau objek yang mendasarinya[15].



Gambar 2.8 Cara Kerja API

Gambar 2.8 menjelaskan cara kerja dari API, pengguna (*client*) akan mengirimkan permintaan HTTP melalui sebuah *endpoint*, *endpoint* merupakan sebuah URL yang digunakan untuk berinteraksi dan mengakses sumber daya yang disediakan oleh API tersebut. API akan mengirimkan data dengan format JSON sebagai respon dari permintaan *client*. JSON atau Java Script Object Notation merupakan sebuah format data yang ditemukan oleh Douglas Crockford pada tahun 2006, JSON menggunakan sintak objek javascript sebagai format data berbasis teks, JSON memiliki ukuran data yang lebih kecil serta waktu proses yang lebih cepat jika dibandingkan dengan format data lainnya[16].

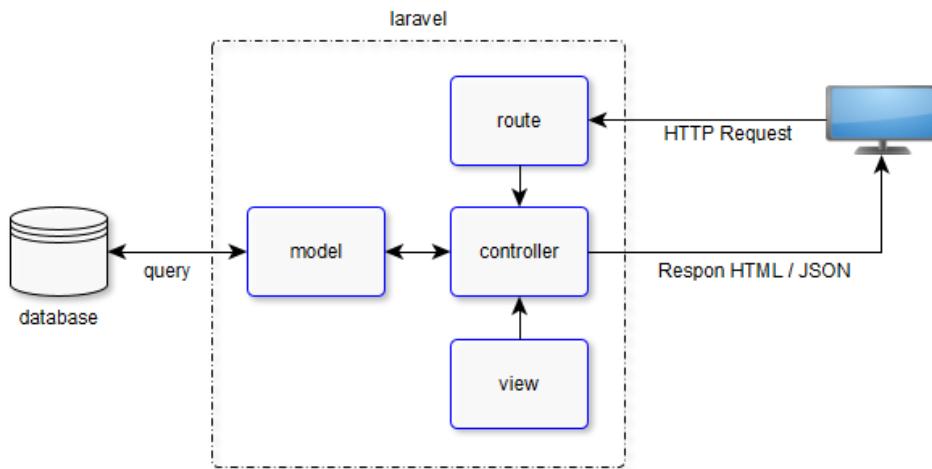
Dengan menggunakan sebuah API maka pengguna tidak akan terhubung secara langsung dengan *database*, sehingga akan meningkatkan keamanan dari sistem, jika sewaktu waktu terjadi hal-hal yang tidak di inginkan seperti rusaknya data atau sistem di retas oleh pihak yang tidak bertanggung jawab data asli tetap aman tersimpan pada database tanpa mengalami gangguan sedikitpun[16].

2.8 Framework Laravel

Framework adalah struktur kerja yang terdiri dari sekumpulan program, terutama berisi kelas dan fungsi, yang dapat membantu para pengembang (*developer*) dalam menangani berbagai permasalahan yang sering dihadapi dalam pemrograman. Dengan menggunakan framework, para pengembang dapat lebih terfokus dan lebih cepat dalam membangun aplikasi karena berbagai tugas rutin

seperti koneksi ke database, pemanggilan variabel, pengelolaan file, dan sebagainya telah diatur dan disediakan dalam framework tersebut[17].

Laravel merupakan kerangka kerja pemrograman web yang menggunakan bahasa pemrograman PHP yang terbuka dan gratis, laravel diperuntukkan untuk pengembangan aplikasi berbasis *website* maupun API dengan menggunakan pendekatan pola MVC atau Model View Controller[18]. Arsitektut MVC memiliki business logic yang terpisah dari model dan *presentation*, sehingga saat melakukan modifikasi pada program tidak mempengaruhi komponen lain yang tidak diubah, dan proses pengembangan yang lebih cepat, serta dapat menggunakan *reuse of code* dimana fungsi ini berguna dalam pengembangan website tanpa harus melakukan coding dari awal[17].



Gambar 2.9 Konsep MVC pada Laravel

Gambar 2.9 menjelaskan kosep MVC yang ada pada Laravel, saat pengguna melakukan *request* ke Laravel maka *request* tersebut akan ditangani oleh *route*, *route* disini bertugas sebagai pintu masuk yang menyediakan *url/enpoint* mana saja yang dapat diakses oleh *client*. Jika tidak ditemukan masalah maka *route* akan melanjutkan *request* tersebut ke modul MVC dengan tugas sebagai berikut :

1. Model

Model merupakan sebuah modul yang digunakan untuk melakukan proses *query* ke *database*, pada laravel model menyediakan mekanisme *query*

builder yang menjamin proses *query* berjalan dengan mudah dan aman dengan berbagai filter untuk menghindari kesalahan.

2. *View*

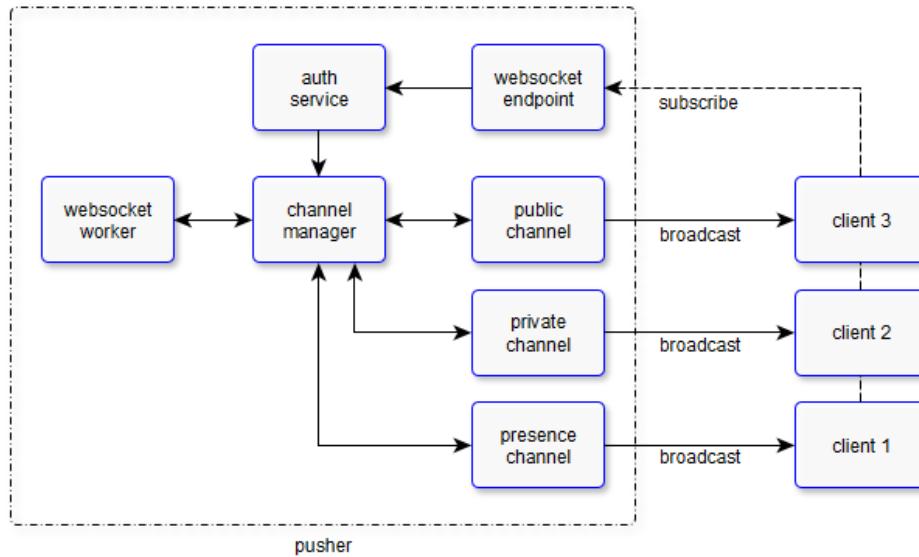
View merupakan bagian dari Laravel yang berfungsi untuk menyediakan tampilan antarmuka kepada pengguna, antarmuka akan tampil pada *browser* pengguna. Pada implementasi Laravel sebagai API maka *view* tidak mendapat tugas khusus karena pada API *controller* akan mengembalikan respon dalam format JSON dan tidak mengembalikan tampilan dalam HTML.

3. *Controller*

Controller digunakan untuk mengolah data yang diminta oleh pengguna, sebuah *controller* tidak langsung menerima permintaan dari pengguna melainkan *controller* akan dipanggil oleh bagian routing pada saat ada permintaan. *Controller* juga akan mengirimkan respon ke *client* sesuai dengan permintaan yang dikirimkan.

2.9 *Pusher Websocket*

Websocket merupakan sebuah protokol komunikasi web berbasis *client-server*, keberadaan websocket dinilai dapat menggantikan teknologi AJAX sebagai pendahulu komunikasi *client-server*. *Websocket* merupakan teknologi yang mampu memberikan performa terbaik ketika diimplementasikan dalam sistem dengan *rate-request* tinggi, dibandingkan dengan teknologi komunikasi lain termasuk AJAX[19]. *Websocket* memungkinkan komunikasi dua arah antara *client* dan *server* dengan menggunakan koneksi yang sudah terjalin, hal ini dikarenakan pada *websocket*, koneksi akan terus terjalin selama tidak terjadi error atau ada permintaan pemutusan koneksi. Salah satu layanan yang menyediakan koneksi *websocket* yaitu Pusher, Pusher menyediakan komunikasi *realtime* antara *server* dan *client* melalui *channel-channel* yang telah tersedia.



Gambar 2.10 Diagram Pusher *Channel*

Gambar 2.10 di atas menjelaskan bagian-bagian dari pusher channel, pusher hanya mendukung komunikasi satu arah yaitu dari server ke client dengan menggunakan metode broadcasting. Untuk bisa mendapatkan pesan dari broadcasting maka client harus terhubung ke salah satu channel dengan cara melakukan subscribe ke channel tersebut. Didalam pusher terdapat 3 channel yaitu :

1. *Public Channel*

Public Channel merupakan *channel* yang tersedia dan dapat di-*subscribe* oleh semua pengguna tanpa adanya proses autentikasi.

2. *Private Channel*

Private Channel merupakan *channel* yang hanya tersedia untuk *client* tertentu, sebelum terhubung ke *private channel* *client* harus melakukan proses autentikasi untuk mendapatkan kode *signature channel* yang digunakan untuk melakukan *subscribe* ke *channel* tersebut.

3. *Presence Channel*

Presence Channel merupakan pengembangan dari *channel private*, pada *presence channel* perilaku *client* dapat dipantau seperti ada client baru yang melakukan subscribe atau ada client yang keluar.

2.10 Penelitian Terdahulu

Beberapa penelitian terdahulu yang dijadikan referensi dalam perancangan sistem database dan server pada sistem keamanan kunci pintu gedung menggunakan access control adalah :

Tabel 2.1 Ringkasan Penelitian Terdahulu

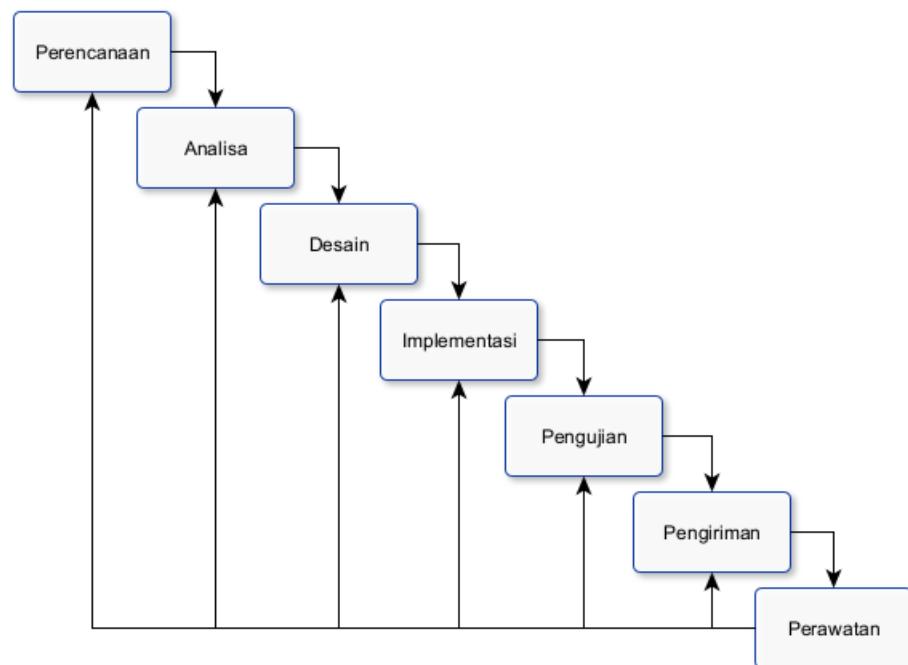
Judul	Tahun	Penulis	Pembahasan
<i>Perancangan Database IoT Berbasis Cloud dengan Restful API</i>	2021	M. Kasyful Anwar	Dalam penelitian ini dibahas mengenai rancangan database berbasis cloud dengan Restful API untuk IoT agar data IoT aman dan memiliki <i>throughput</i> yang bagus dengan struktur data yang diatur pada database.
<i>Development of Intelligent Door Lock System for Room Management Using Multi Factor Authentication</i>	2023	I. Hermawan, D. Arnaldy, P. Oktivasari, dan D. A. Fachrudin	Penelitian ini menjelaskan mengenai permasalahan yang muncul pada manajemen kunci secara tradisional seperti jumlah kunci yang banyak, mudah hilang dan mudah diduplikasi.
<i>RFID and GSM Based Attendance Monitoring System using door locking / unlocking system and Its Hardware Implementation</i>	2015	A. Jain, V. L. Kalyani, dan B. Nogiya	Penelitian ini menjelaskan penggunaan RFID untuk melakukan absesnsi siswa sekaligus sebagai kunci untuk membuka pintu ruang kelas.

BAB III

PERANCANGAN SISTEM

3.1 Metode Perancangan

Dalam perancangan tugas akhir ini menggunakan pendekatan *waterfall* dengan pertimbangan yang matang. Metode *waterfall* memberikan struktur dan ketertiban yang sangat diperlukan dalam pengembangan proyek ini. Setiap tahapan akan dilaksanakan secara berurutan sesuai dengan Gambar 3.1.



Gambar 3.1 Alur Perancangan

Keputusan ini diambil karena proyek tugas akhir ini memiliki batasan waktu yang jelas dan persyaratan yang sudah ditetapkan dengan baik. Dengan menggunakan metode *waterfall*, kita dapat memastikan bahwa setiap tahapan diselesaikan dengan cermat sebelum melanjutkan ke tahap berikutnya, sehingga mengurangi risiko kesalahan dan memastikan kelancaran jalannya proyek. Selain itu, pendekatan ini juga memungkinkan untuk melakukan perencanaan yang lebih terperinci, memahami kebutuhan proyek secara menyeluruh, dan mengidentifikasi risiko potensial dengan lebih baik. Dengan demikian, metode *waterfall* menjadi pilihan yang tepat untuk memastikan keberhasilan dan kualitas dari tugas akhir

ini. Pada Gambar 3.1 terlihat alur perancangan dimulai dari proses perencanaan samapi dengan proses pemeliharaan dengan penjelasan sebagai berikut :

1. Perencanaan

Pada tahap perencanaan dilakukan pengumpuna informasi mengenai tujuan serta batasan-batasan proyek yang harus terpenuhi. Pada tahap perencanaan juga mendefinisikan beberapa kebutuhan fungsional dan kebutuhan non-fungsional pada rancangan sistem untuk dikembangkan lebih lanjut.

2. Analisa

Pada tahap analisa dilakukan studi literatur mengenai tujuan dan batasan-batasan proyek serta melakukan pengumpulan informasi mengenai sistem yang akan dibuat dengan melihat dan mempelajari penelitian/perancangan terdahulu. Analisa dilakukan bersama kelompok tugas akhir dengan melakukan penelusuran pustaka serta referensi yang relevan dan disertai dengan diskusi sehingga menghasilkan keluaran berupa gambaran umum mengenai bagian-bagian dari sistem yang akan digunakan pada tahap selanjutnya.

3. Desain

Setelah mendapatkan gambaran mengenai sistem yang akan dibuat, selanjutnya dilakukan pemodelan sistem dengan menggunakan diagram pemodelan. Diagram pemodelan digunakan untuk menggambarkan hubungan dan interaksi antar komponen didalam sistem, dengan menggunakan diagram maka akan menghasilkan gambar sistem yang lebih rinci yang digunakan untuk membangun sistem.

4. Implementasi

Pada tahap implementasi, gambar pemodelan dari sistem kemudian diterjemahkan atau diimplementasikan menggunakan kode program untuk membuat sistem dapat bekerja sesuai dengan dasain awal yang telah ditentukan.

Implementasi dilakukan dengan menggunakan bahasa pemrograman PHP dengan menggunakan *framework* Laravel. Laravel mempunyai karakteristik RAD atau *Rapid Application Development* sehingga

implementasi dapat dilaksanakan dengan cepat dan efisien, Laravel juga menyediakan komponen-komponen yang akan digunakan didalam sistem seperti autentikasi, antrian, notifikasi pesan, *websocket*, dan lain sebagainya sehingga dapat mempermudah proses implementasi.

5. Pengujian

Pada tahap pengujian, sistem yang telah dibangun diuji kinerjanya dengan mengirimkan beberapa permintaan data dan menganalisa hasil keluaran yang diberikan apakah sesuai atau tidak. Proses pengujian bertujuan untuk mengetahui kesesuaian hasil implementasi dengan desain awal dan juga untuk mengetahui performa dari sistem yang telah dibuat.

6. Pengiriman

Pada tahap pengiriman, aplikasi yang sudah selesai dilakukan pengujian dan dinyatakan lolos uji selanjutkan akan dipasang atau diinstall pada server. Pada proses deployment juga akan dilakukan pemasangan aplikasi tambahan seperti *database*, penjadwalan, *firewall*, dan lain sebagainya. Pada tahap ini sistem yang sudah dibuat dapat digunakan oleh pengguna umum.

7. Perawatan

Pada tahap pemeliharaan atau perawatan dilakukan pengecekan secara berkala pada aplikasi yang sudah berjalan pada server. dengan adanya perawatan secara teratur jika ditemukan permasalahan atau komponen yang tidak bekerja maka akan langsung diperbaiki sehingga tidak mengganggu kinerja dari server.

3.2 Kebutuhan Fungsional dan Non-Fungsional

Kebutuhan fungsional menggambarkan apa saja yang dimiliki dan bisa dilakukan oleh sistem yang akan dibuat sedangkan kebutuhan non-fungsional lebih menekankan keunggulan yang dimiliki oleh sistem yang akan dibuat. Dengan adanya analisa kebutuhan fungsional dan non-fungsional akan memberikan gambaran tentang batasan dan pencapaian yang harus dimiliki oleh sistem agar sistem tersebut dapat dikatakan sesuai dengan rangcangan awal. Bedasarkan analisa pada tahap perancangan maka didapat hasil kebutuhan

fungsional sesuai pada Tabel 3.1 dan kebutuhan non fungsional sesuai dengan Tabel 3.2.

Tabel 3.1 Kebutuhan Fungisional

No	Kebutuhan	Penjelasan
1	Autentikasi login dan logout	Sistem memiliki metode autentikasi untuk melakukan login dan logout
2	Ganti password	Sistem memiliki metode untuk mengganti password
3	Reset password	Sistem memiliki metode untuk menangani kondisi lupa password
4	Verifikasi email	Sistem memiliki metode untuk melakukan verifikasi email
5	Ganti profil	Sistem memiliki metode untuk memperbarui profil pengguna
6	Lihat daftar akses	Sistem memiliki metode untuk melihat daftar akses yang dimiliki oleh pengguna
7	Verifikasi akses	Sistem memiliki metode untuk melakukan verifikasi akses pengguna
8	Lihat riwayat aktifitas	Sistem memiliki metode untuk melihat riwayat aktifitas
9	Lihat daftar pintu	Sistem memiliki metode untuk melihat daftar pintu yang ada
10	Membuka pintu jarak jauh	Sistem memiliki metode untuk membuka atau mengunci pintu dari jarak jauh
11	Register pintu baru	Sistem memiliki metode untuk menambahkan pintu baru ke sistem penguncian
12	Koneksi websocket untuk pintu	Sistem memiliki jalur komunikasi menggunakan websocket
13	Mendapatkan signature	Sistem memiliki metode untuk memperoleh signature untuk melakukan subscribe ke channel websocket
14	Update status pintu	Sistem memiliki metode untuk melakukan update status pintu
15	Peringatan pintu	Sistem memiliki metode oleh pintu untuk mengirim peringatan

Tabel 3.1 menjelaskan kebutuhan fungsional yang harus disediakan oleh sistem yang akan dibuat. Sistem *database* dan *server* yang dibuat akan menyediakan layanan untuk 2 jenis *client*, pertama yaitu perangkat aplikasi mobile yang dipakai oleh operator dan pengguna, yang kedua yaitu perangkat kunci pintu berbasis IoT. Untuk jenis client ini sistem yang dibuat akan memberikan layanan API yang digunakan untuk melakukan beberapa kegiatan seperti login, logout, ganti password, reset password, verifikasi email, verifikasi akses dan lain sebagainya.

Tabel 3.2 Kebutuhan Non-Fungsional

No	Kebutuhan	Penjelasan
1	Tersedia	Sistem dapat beroperasi penuh selama 7 hari dalam satu minggu dan 24 jam dalam satu hari
2	Cepat	Sistem memiliki waktu respon yang cepat, tidak lebih dari 5 detik
3	Aman	Sistem dapat beroperasi dengan aman baik dalam penyimpanan data maupun pengiriman data.

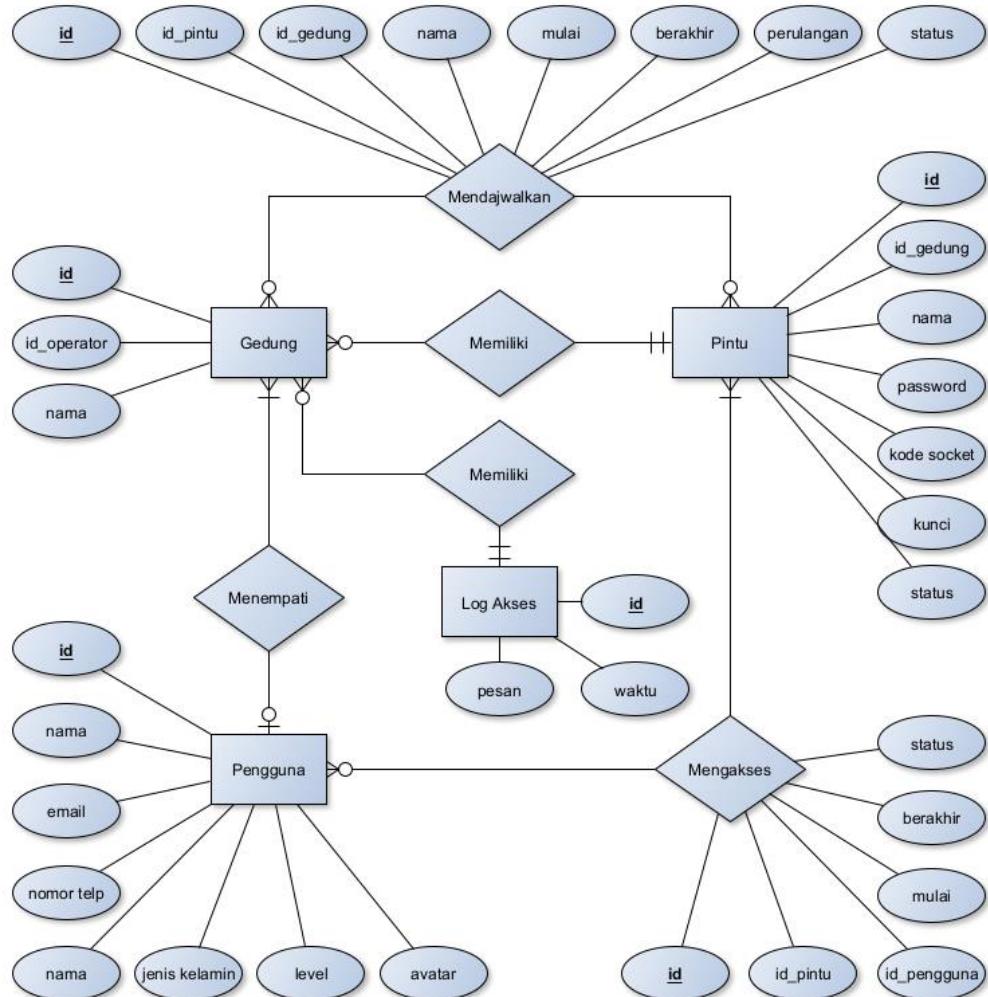
Tabel 3.2 menjelaskan kebutuhan non-fungsional pada sistem, kebutuhan non-fungsional menggambarkan keunggulan dari sistem seperti sistem harus dapat bekerja 24 jam penuh, sistem harus memberikan respon dengan cepat dan juga sistem harus menyimpan dan mengirim data dengan aman.

3.3 Perancangan *Database*

Database digunakan untuk menyimpan data yang digunakan dalam sistem penguncian pintu gedung seperti data pengguna, data pintu, data akses dan lain sebagainya. *Database* yang digunakan harus dapat menunjang kinerja sistem dengan karakteristik respon yang cepat dan pengelolaan data terstruktur.

Berdasarkan penelitian [20] yang membandingkan kinerja dari berbagai tipe dan jenis *database* didapatkan hasil penggunaan MySQL menunjukkan hasil kinerja yang bagus dalam hal waktu eksekusi permintaan, dengan sistem penyimpanan data bersifat relasional dan terstruktur maka MySQL dapat

diterapkan pada sistem penguncian pintu gedung ini. Diagram ER (entity relation) dari database pada sistem penguncian gedung ini dapat dilihat pada Gambar 3.2.



Gambar 3.2 ER Database

Pada Gambar 3.2 terlihat relasi antar entitas didalam *database* sistem penguncian pintu gedung. Pada database sistem penguncian pintu gedung terdapat beberapa entitas seperti pengguna, pintu, gedung dan *log* akses, entitas tersebut digunakan untuk meyimpan data yang akan digunakan didalam pengelolaan sistem dengan isi data sesuai dengan atribut dari masing-masing entitas. Dalam sistem database juga terdapat relasi, relasi menunjukkan hubungan antar entitas seperti menempati, memiliki, menjadwalkan dan mengakses, khusus untuk relasi menjadwalkan dan mengakses pada Gambar 3.2 terlihat bahwa relasi tersebut memiliki aktribut yang menandakan bahwa relasi tersebut juga bertindak sebagai

entitas yang menjadi penghubung antara 2 entitas yang berbeda, pada umumnya kondisi seperti ini berarti relasi tersebut menunjukkan kardinalitas *many-to-many* dimana relasi tersebut bertindak sebagai entitas pivot untuk menghubungkan relasi *many-to-many* dari 2 entitas lainnya.

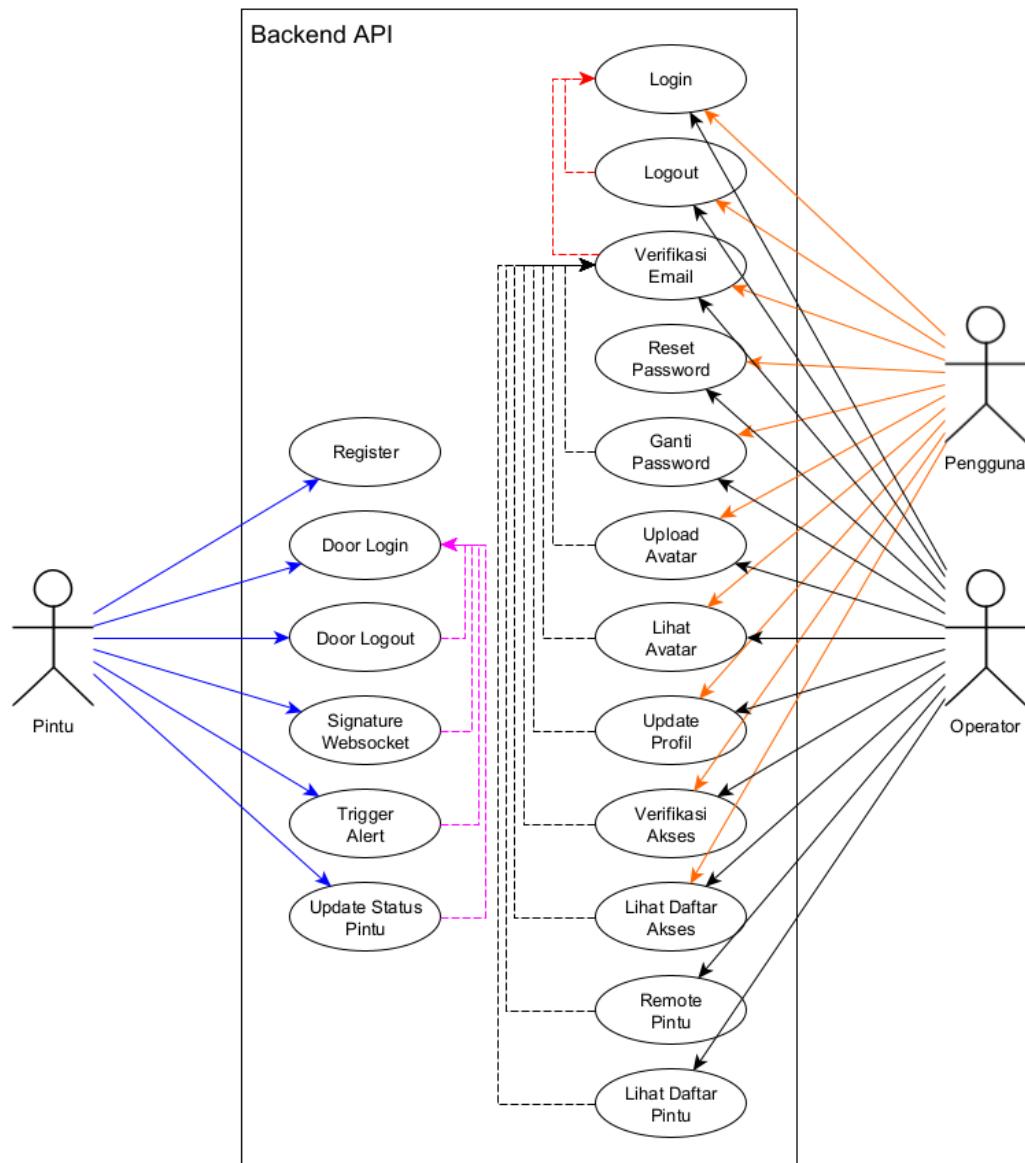
3.4 Perancangan *Backend API*

Backend API digunakan sebagai salah satu jalur komunikasi yang menghubungkan antara perangkat *mobile* dan perangkat kunci pintu berbasis IoT dengan *server*. Selain terdapat *website* yang digunakan sebagai antarmuka utama dalam mengelola sistem ini juga terdapat aplikasi berbasis mobile yang digunakan untuk menunjang kinerja dari sistem terutama pada bagian yang membutuhkan teknologi yang belum disediakan oleh *browser* seperti koneksi *bluetooth* dan pindai kode QR menggunakan kamera.

Backend API yang dikembangkan menggunakan bahasa pemrograman PHP dengan menggunakan kerangka kerja dari Laravel. Laravel menyediakan beberapa fitur yang siap digunakan seperti autentikasi *login*, reset *password*, notifikasi email dan lain sebagainya sehingga mempermudah dan mempercepat proses pengkodean.

3.3.1 Diagram *Use Case API*

Tidak semua fitur yang ada pada website diimplementasikan dalam bentuk API, hal ini berkaitan dengan kompleksitas dari sistem yang dikembangkan, fitur yang berkaitan dengan manajemen penguncian seperti menambah pintu baru, membuat jadwal, menambahkan pengguna baru, memberikan akses ke pengguna dan lain sebagainya hanya bisa dilakukan melalui website hal ini dimaksudkan untuk mempertahankan kontrol yang lebih ketat terhadap akses dan perubahan yang dilakukan pada sistem. Dengan membatasi akses langsung melalui API, administrator sistem dapat memastikan bahwa hanya proses yang telah melalui lapisan keamanan dan validasi yang benar yang dapat mempengaruhi manajemen penguncian dan pengaturan lainnya. Diagram *Use Case* dari *Backend API* dapat dilihat pada Gambar 3.3 di bawah.



Gambar 3.3 Diagram Use Case API

Dapat dilihat pada Gambar 3.3 di atas, terdapat 3 aktor yang berinteraksi dengan sistem melalui API yaitu pintu, pengguna dan operator. Pintu merupakan perangkat IoT yang digunakan untuk melakukan penguncian pada gedung, sedangkan pengguna dan operator merupakan aplikasi mobile yang digunakan sebagai antarmuka sistem.

Pada Gambar 3.3 juga terlihat beberapa fungsi seperti *update profil*, *lihat avatar*, *verifikasi akses* dan lain sebagainya merupakan *extends* dari fungsi *verifikasi email* yang menandakan bahwa fungsi-fungsi tersebut hanya bisa diakses jika *client* sudah melakukan proses verifikasi email. Fungsi verifikasi

email juga merupakan *extends* dari fungsi login sehingga fungsi verifikasi email hanya bisa diakses jika client sudah melakukan autentikasi login. Hal ini dimaksudkan supaya hanya client yang terautentikasi dan terverifikasi yang dapat mengakses sumber daya yang disediakan.

3.3.2 *Endpoint API*

Dalam perancangan *Backend API* ini terdapat dua bagian API yang digunakan, yaitu API untuk aplikasi mobile dan API untuk perangkat kunci pintu IoT dengan penjabaran setiap *endpoint* dan fungsinya dapat dilihat pada Tabel 3.1 diabawah.

Tabel 3.3 *Endpoint API*

Tipe	Endpoint	Auth	Keterangan
POST	/api/login	-	login pengguna dan operator
POST	/api/reset-password	-	reset password menggunakan email
POST	/api/verify-email	sanctum	verifikasi email
GET	/api/logout	sanctum	logout pengguna dan operator
POST	/api/update-profile	sanctum, verified	update profile nama, email, dan lainnya
GET	/api/avatar	sanctum, verified	mengambil gambar avatar
POST	update-avatar	sanctum, verified	mengubah gambar avatar
POST	/api/change-password	sanctum, verified	mengubah password pengguna atau operator
GET	/api/my-access	sanctum, verified	mengambil daftar akses
GET	/api/get-doors	sanctum, verified	mengambil daftar pintu
GET	/api/my-history	Sanctum, verified	mengambil daftar riwayat akses

Lanjutan Tabel 3.4 Endpoint API

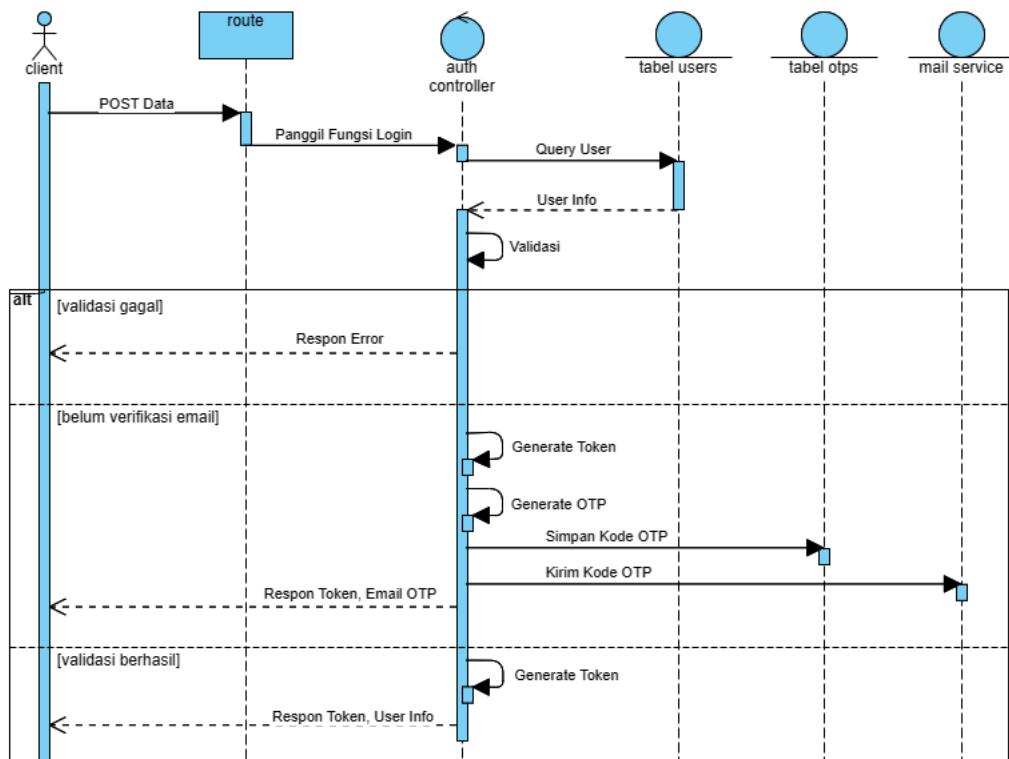
Tipe	Endpoint	Auth	Keterangan
GET	/api/verify-access/{door_id}	sanctum, verified	verifikasi akses dari kode QR pintu
POST	/api/remote-access	sanctum, verified	membuka atau mengunci pintu jarak jauh
POST	/door/login	-	login perangkat kunci pintu
POST	/door/register	-	menambahkan perangkat kunci pintu baru
GET	/door/logout	sanctum	logout perangkat kunci pintu
POST	/door/get-signature	sanctum	mengambil kode signature channel pusher
POST	/door/update-status	sanctum	update status pintu
POST	/door/alert	sanctum	peringatan pada pintu

Dapat dilihat pada Tabel 3.1 di atas, API terbagi menjadi 2 bagian yang ditandai dengan awalan (*prefix*) “/api” dan “/door”. Awalan “/api” merupakan *endpoint* API yang khusus ditujukan untuk penggunaan aplikasi mobile pengguna dan operator sedangkan awalan “/door” merupakan API yang ditujukan untuk perangkat kunci IoT sehingga dengan menggunakan API yang terpisah maka API akan semakin terorganisir dengan baik.

Pada Tabel 3.1 juga terlihat bahwa beberapa *endpoint* memerlukan autentikasi *sanctum* dan *verified*. Autentikasi *sanctum* merupakan sebuah metode autentikasi berbasis token yang digunakan untuk mengamankan sumber daya API dari *client* dengan hanya mengijinkan pengguna yang sudah terautentikasi yang dapat mengakses API tersebut. Sedangkan *verified* merupakan autentikasi tambahan yang digunakan untuk memastikan bahwa *client* (pengguna dan operator) sudah melakukan verifikasi email sehingga dapat meningkatkan keamanan API.

3.3.3 API Login

API *login* digunakan untuk melakukan autentikasi *client* melalui *username* dan *password* yang dikirimkan, API *login* juga memeriksa apakah pengguna dan operator sudah melakukan verifikasi email, jika belum maka *login* akan tertahan sampai pengguna melakukan verifikasi email. Diagram fungsi login dapat dilihat pada Gambar 3.4 di bawah.



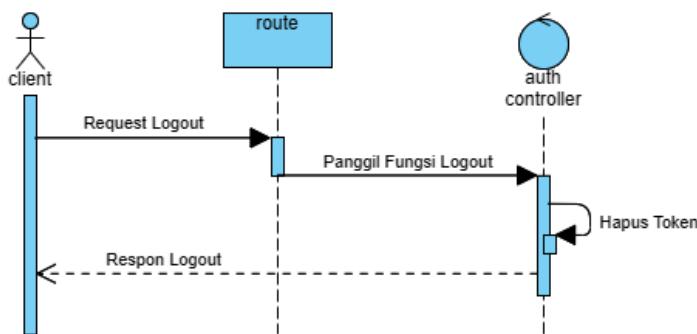
Gambar 3.4 Sequence Diagram API Login

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 melakukan verifikasi email maka

kontroler akan membuat kode OTP atau *One Time Password* yang merupakan 6 digit angka acak dan mengirimkan kode tersebut ke email *client*. Jika autentikasi yang dilakukan gagal, maka kontroler akan mengembalikan respon error ke *client*.

3.3.4 API Logout

Fungsi *logout* digunakan untuk mengakhiri sesi *login* pengguna dan operator dengan cara menghapus semua token yang dimilikinya. Diagram dari API *logout* dapat dilihat pada Gambar 3.5 di bawah.



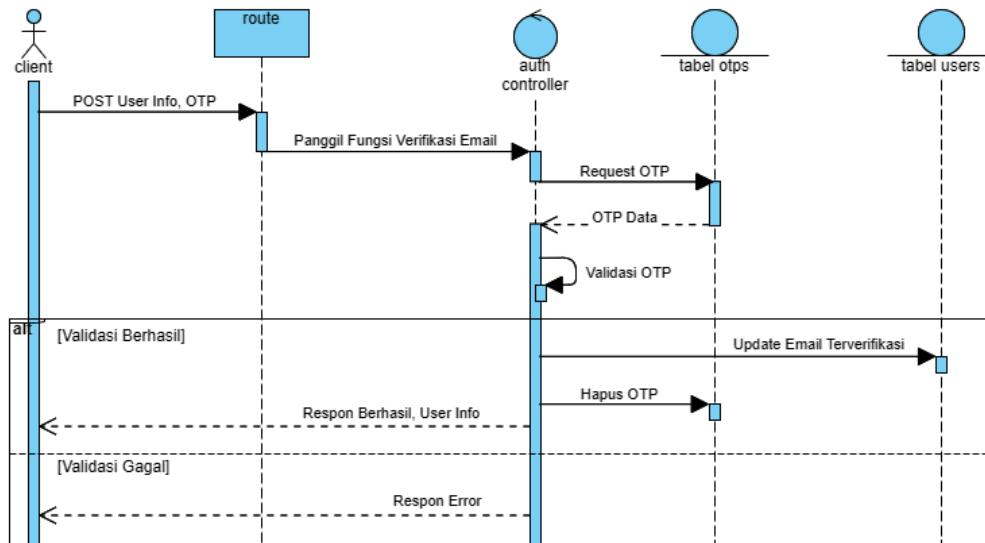
Gambar 3.5 Sequence Diagram API Logout

Pada Gambar 3.5 di atas, 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.

3.3.5 API Verifikasi Email

API verifikasi email digunakan untuk memastikan bahwa pengguna dan operator memiliki email yang valid dan aktif, dengan adanya verifikasi email maka akan meningkatkan keamanan dengan hanya mengijinkan pengguna dan operator yang terpercaya untuk mengakses sumber daya yang ada. Verifikasi email dilakukan dengan cara mengirimkan kode OTP (*One Time Password*) ke

email yang telah didaftarkan. Diagram dari API verifikasi email dapat dilihat pada Gambar 3.6 di bawah.



Gambar 3.6 Sequence Diagram API Verifikasi Email

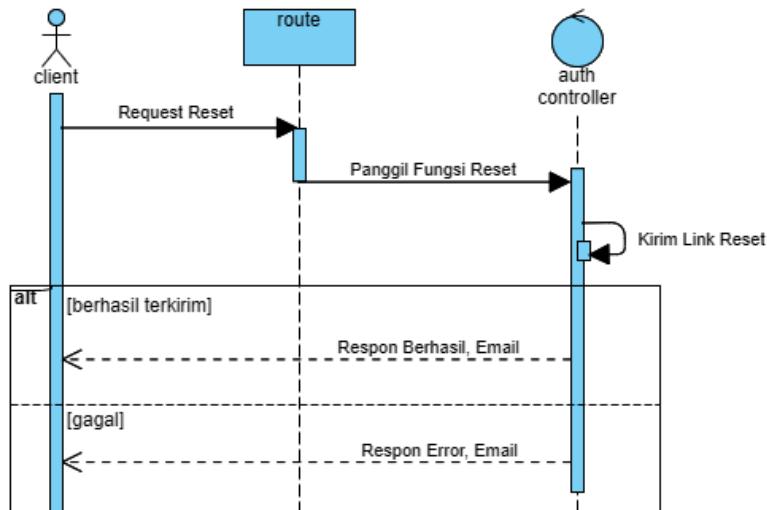
Pada Gambar 3.6 di atas dapat dilihat bahwa pengguna melakukan verifikasi email dengan mengirimkan kode OTP yang sudah diterima melalui email disertai dengan detail client seperti id, nama dan email ke endpoint “/api/verify-email”, kemudian pada kontroler akan memeriksa kode yang diterima dengan kode 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 kontroler akan mengembalikan respon error.

3.3.6 API Reset Password

API reset *password* adalah fitur penting dalam sistem keamanan yang memungkinkan pengguna dan operator untuk mengatasi masalah lupa *password* dengan cepat dan mudah. Dengan menggunakan API ini, pengguna dan operator dapat menghindari kehilangan akses akun dan data yang berharga.

API reset *password* digunakan oleh pengguna dan operator untuk memperbarui *password* jika mereka lupa dengan cara memasukkan email yang sudah terdaftar dan menggunakan *link* yang telah dikirimkan ke email untuk

memperbarui *password*. Diagram dari API reset *password* dapat dilihat pada Gambar 3.7 di bawah.



Gambar 3.7 Sequence Diagram API Reset Password

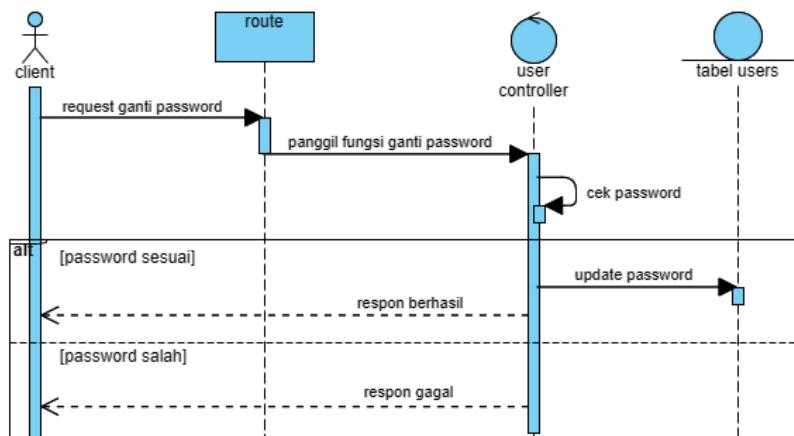
Pada Gambar 3.7 di atas, proses reset *password* menggunakan metode yang terstruktur dan efisien. *Client* memulai proses ini dengan melakukan permintaan (*request*) ke *endpoint* "/api/reset-password" pada *server*. Permintaan ini berisi data email yang sudah terdaftar sebagai parameter supaya *server* 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.

3.3.7 API Ganti Password

API ganti password digunakan oleh pengguna dan operator untuk mengganti password mereka melalui aplikasi mobile. Berbeda dengan reset *password*, API ganti *password* digunakan untuk mengganti *password* pengguna tanpa melalui link yang dikirimkan ke email. Diagram dari API ganti password dapat dilihat pada Gambar 3.8 di bawah.

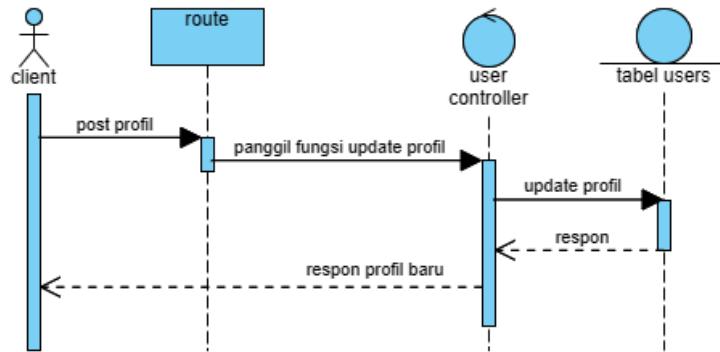


Gambar 3.8 Sequence Diagram API Ganti Password

Dapat dilihat pada Gambar 3.8 di atas, untuk mengganti *password* pertama client mengirimkan permintaan ganti password ke endpoint “/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. Jika kedua *password* cocok maka kontroler akan memperbarui *password* pada tabel users dan mengembalikan respon berhasil, jika *password* tidak sesuai maka kontroler akan mengembalikan respon error.

3.3.8 API Ganti Profil

API ganti profil digunakan untuk mengganti data pengguna dan operator seperti nama, email, nomor hp dan lain sebagainya melalui aplikasi *mobile*. Diagram dari API ganti profil dapat dilihat pada gamabr 3.9 di bawah.

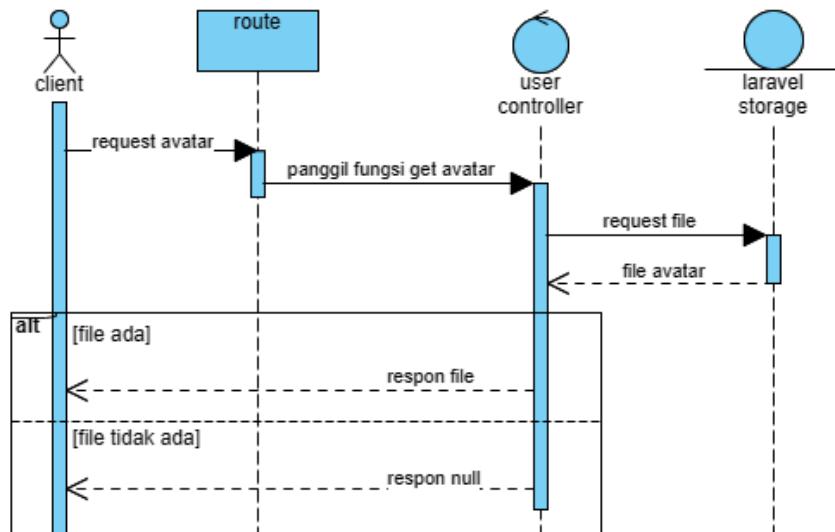


Gambar 3.9 Sequence Diagram API Ganti Profil

Dapat dilihat pada Gambar 3.9 di atas, untuk mengganti profil pertama pengguna atau operator mengirimkan data profil mereka ke endpoint “/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.

3.3.9 API Lihat Avatar

API lihat avatar digunakan untuk mendapatkan gambar avatar (foto profil) dari pengguna dan operator untuk ditampilkan didalam aplikasi mobile. Diagram dari API lihat avatar dapat dilihat pada Gambar 3.10 di bawah.

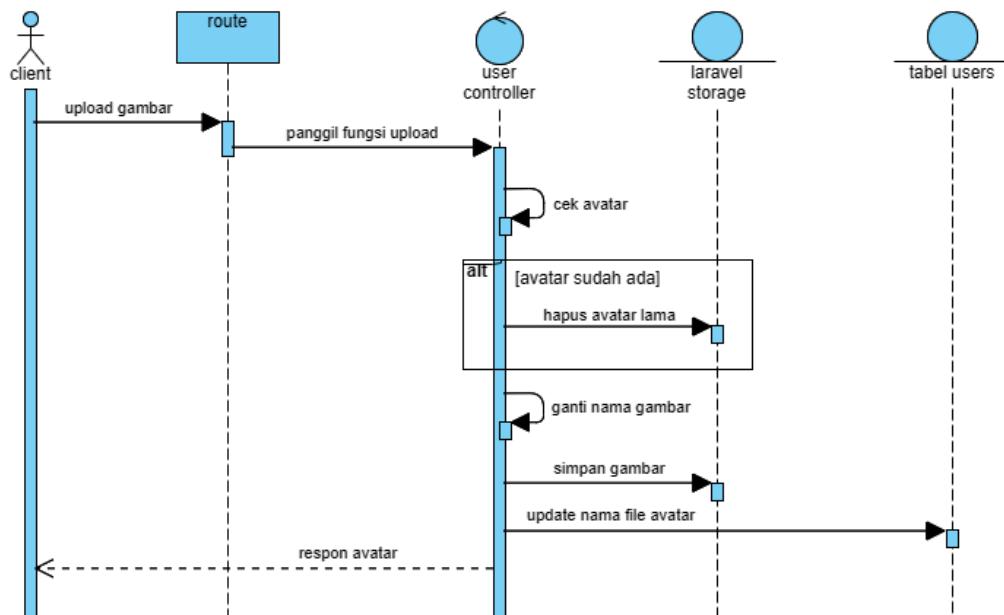


Gambar 3.10 Sequence Diagram API Lihat Avatar

Dapat dilihat pada Gambar 3.10 di atas, untuk mendapatkan gambar avatar pertama pengguna atau operator melakukan permintaan data avatar melalui endpoint “/api/avatar” kemudian kontroler akan mengambil gambar avatar didalam penyimpanan Laravel (*storage*). *Storage* merupakan salah satu fitur yang disediakan laravel untuk melakukan penyimpanan file seperti gambar, log dan berkas lainnya, dengan menggunakan *storage* kita bisa mengatur siapa saja yang boleh mengakses penyimpanan tersebut sehingga lebih aman. Jika file telah ditemukan maka kontroler akan mengembalikan respon file dan jika tidak ditemukan maka kontroler akan mengembalikan nilai null atau kosong.

3.3.10 API Ganti Avatar

API ganti avatar digunakan untuk mengganti gambar avatar pengguna atau operator, pengguna dapat mengganti gambar avatar melalui aplikasi mobile. Diagram dari API ganti avatar dapat dilihat pada Gambar 3.11 di bawah.



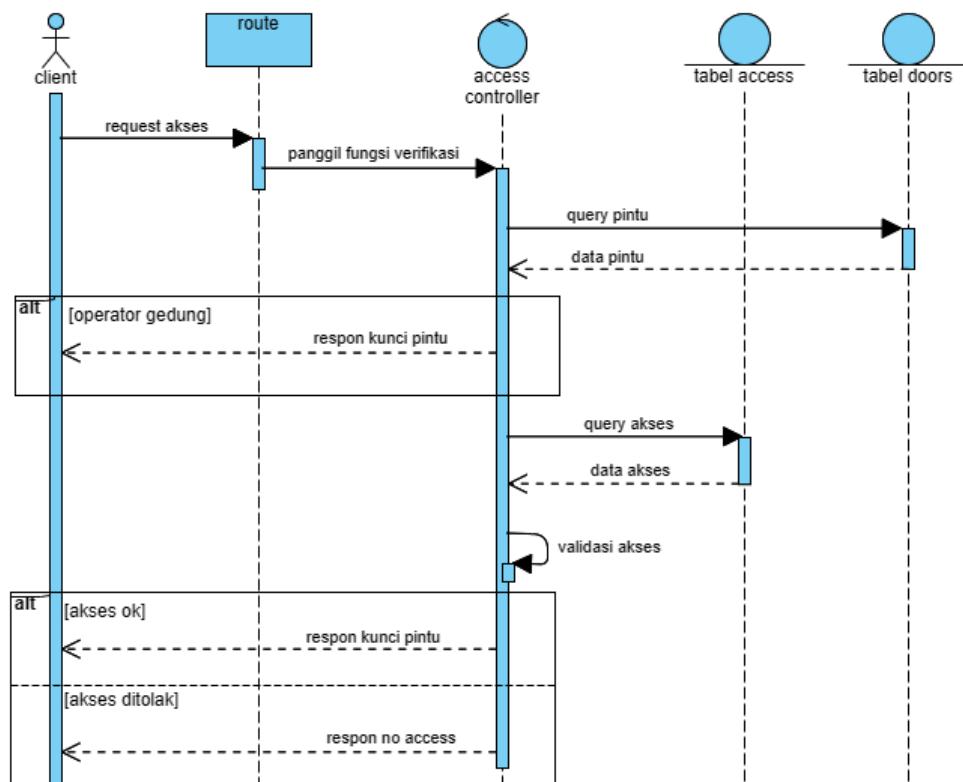
Gambar 3.11 Sequence Diagram API Ganti Avatar

Dapat dilihat pada Gambar 3.11 di atas, untuk mengganti avatar pertama pengguna atau operator akan mengirimkan file avatar yang baru melalui *endpoint* “/api/update-avatar”, kemudian kontroler akan melakukan pemeriksaan apakah

sebelumnya pengguna atau operator sudah memiliki gambar avatar, jika sudah maka file avatar sebelumnya akan dihapus dari *storage*, kemudian file avatar yang baru akan disesuaikan namanya sesuai dengan format *server*, setelah namanya diganti kemudian file akan disimpan didalam *storage* dan kontroler juga akan memperbarui nama avatar didalam tabel *users*, terakhir kontorler akan mengembalikan respon yang menginformasikan avatar telah diganti.

3.3.11 API Verifikasi Akses

Verifikasi akses digunakan oleh pengguna dan operator untuk memverifikasi diri mereka dan untuk mendapatkan akses terhadap suatu pintu dengan cara memindai kode QR dengan perangkat mobile. Diagram dari API verifikasi akses dapat dilihat pada Gambar 3.12 di bawah.



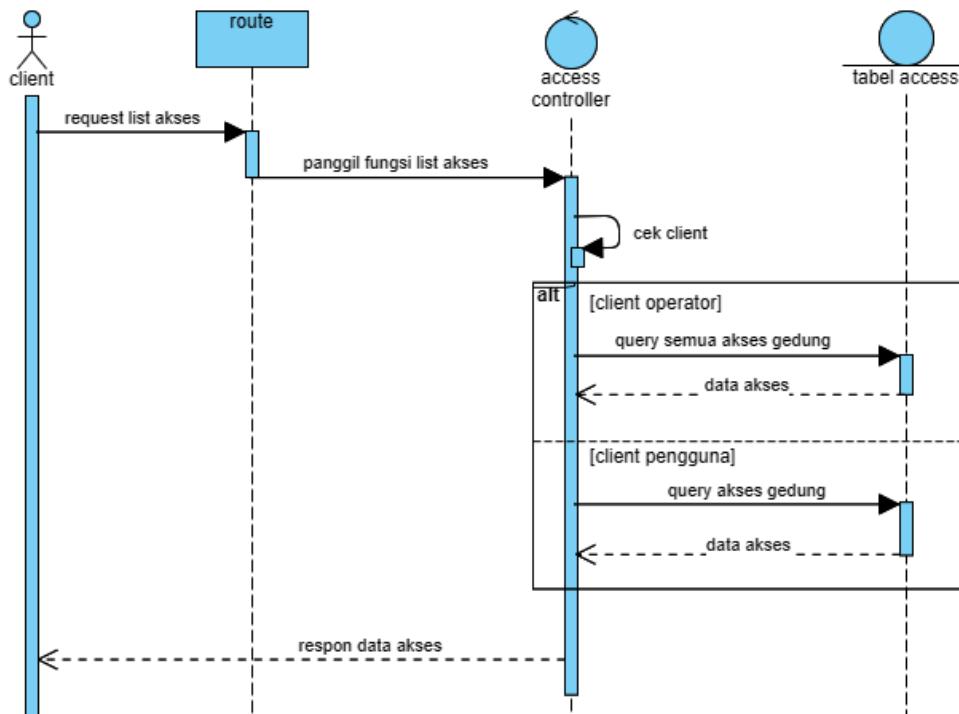
Gambar 3.12 Sequence Diagram API Verifikasi Akses

Untuk mendapatkan akses 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 mengembalikan 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 masih berlaku maka kontroler akan mengembalikan respon akses diijinkan dan mengirimkan kode kunci untuk membuka pintu.

3.3.12 API Daftar Akses

API daftar akses digunakan untuk mendapatkan daftar dari pintu mana saja yang dapat diakses oleh pengguna, sehingga pengguna dapat mengetahui pintu mana saja yang bisa diakses oleh dirinya. Diagram dari API daftar akses dapat dilihat pada Gambar 3.13 di bawah.



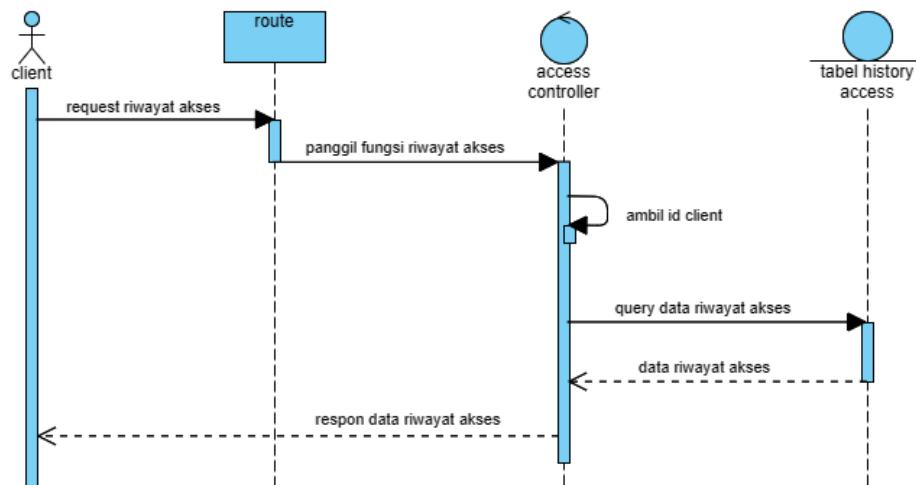
Gambar 3.13 Sequence Diagram API Daftar Akses

Dapat dilihat pada Gambar 3.13 di atas, untuk mendapatkan daftar akses yang dimiliki pengguna mengirimkan request endpoint “/api/my-access”, kemudian kontroler akan melakukan pemeriksaan apakah permintaan datang dari

seorang operator atau pengguna biasa. Jika permintaan berasal dari seorang operator maka kontroler akan mengembalikan semua akses pintu atau semua daftar pintu yang ada, namun jika permintaan datang dari pengguna biasa maka kontroler akan mengembalikan data akses sesuai dengan akses yang dimiliki pengguna tersebut.

3.3.13 API Riwayat Akses

API riwayat akses digunakan untuk memberikan informasi pencatatan akses pengguna, riwayat akses pengguna dapat dilihat pada masing-masing aplikasi *mobile* pengguna. Diagram dari API riwayat akses dapat dilihat pada Gambar 3.14 di bawah.

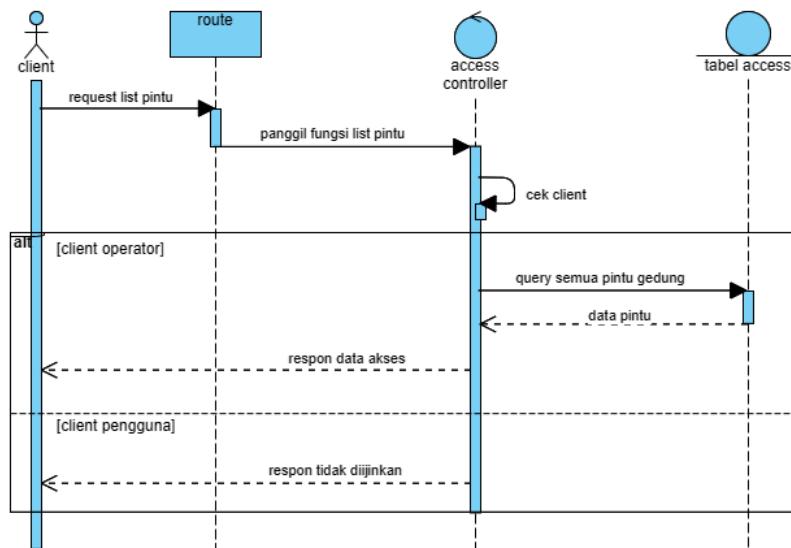


Gambar 3.14 Sequence Diagram API Riwayat Akses

Dapat dilihat pada Gambar 3.14 untuk mendapatkan data riwayat akses maka pengguna akan melakukan permintaan ke *endpoint* “/api/my-history”. Kemudian kontroler akan mengambil data riwayat akses pengguna dengan melakukan *query* ke database 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*.

3.3.14 API Daftar Pintu

API daftar pintu digunakan oleh operator untuk menampilkan daftar pintu yang ada pada suatu gedung, sehingga operator dapat mengetahui jumlah dan status pada setiap pintu. Diagram dari API daftar pintu dapat dilihat pada Gambar 3.15 di bawah.

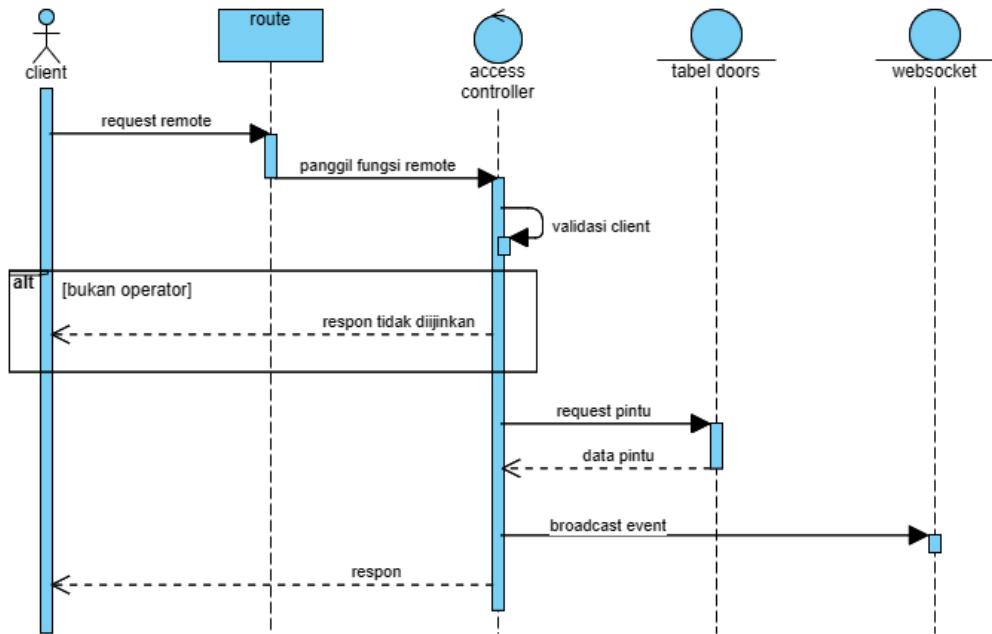


Gambar 3.15 Sequence Diagram API Daftar Pintu

Dapat dilihat pada Gambar 3.15 di atas, 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 sebagai respon ke operator.

3.3.15 API Remote Pintu

API *remote* pintu digunakan oleh operator untuk membuka atau mengunci pintu secara jarak jauh melalui aplikasi *mobile*. Dengan adanya fitur ini operator dapat mengendalikan pintu melalui aplikasi *mobile* dimana saja dan kapan saja tanpa harus berada di ruangan operasional dengan menggunakan komputer. Diagram dari API *remote* pintu dapat dilihat pada Gambar 3.16 di bawah.

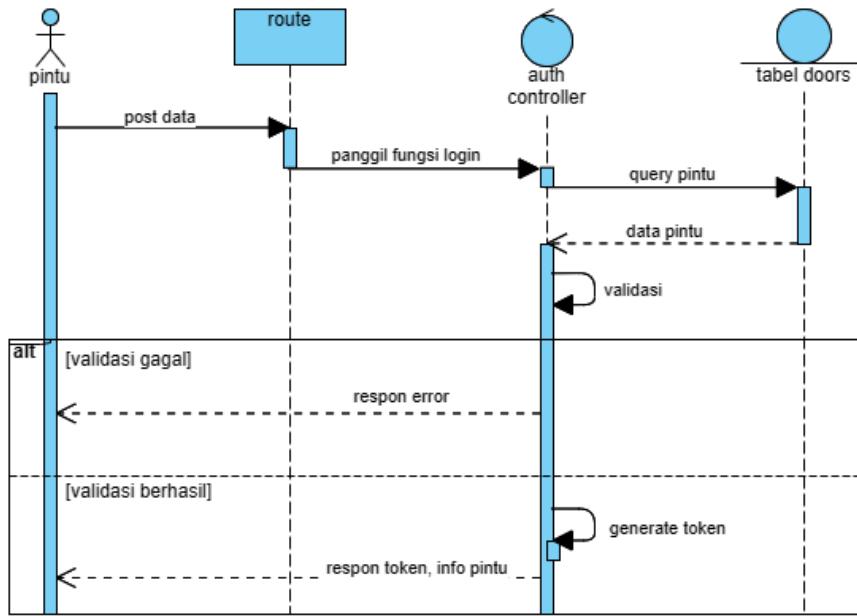


Gambar 3.16 Sequence Diagram API Remote Pintu

Dapat dilihat pada gambar 3.16 di atas, 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 berasal dari pengguna biasa maka kontroler akan mengebalikan 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.

3.3.16 API Door Login

API *door login* digunakan untuk proses autentikasi perangkat kunci pintu, berbeda dengan *login* pada pengguna dan operator pada *login* pintu hanya terdapat pengecekan *username* dan *password* saja tanpa ada pengecekan verifikasi email dan lainnya hal ini dikarenakan pada perangkat kunci pintu tidak menggunakan email sebagai identitasnya. Diagram dari API door login dapat dilihat pada Gambar 3.17 di bawah.

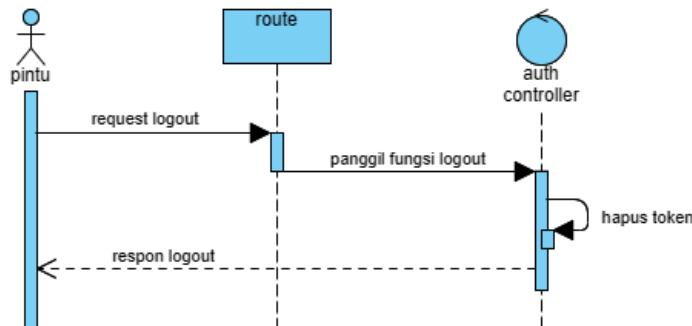


Gambar 3.17 Sequence Diagram API Door Login

Dapat dilihat pada Gambar 3.17 di atas, perangkat kunci pintu akan mengirimkan data *username* dan *password* melalui endpoint “/door/login”, kemudian kontroler akan memeriksa dan membandingkan dengan data pada tabel *doors*, jika data sesuai maka kontroler akan membuat token baru menggunakan sanctum dan mengembalikan respon token yang menandakan *login* berhasil, jika data tidak sesuai maka kontroler akan mengembalikan respon error.

3.3.17 API Door Logout

API *door logout* digunakan untuk mengakhiri sesi *login* perangkat kunci pintu dengan cara menghapus semua token yang dimilikinya. Diagram dari API door logout dapat dilihat pada Gambar 3.18 di bawah.

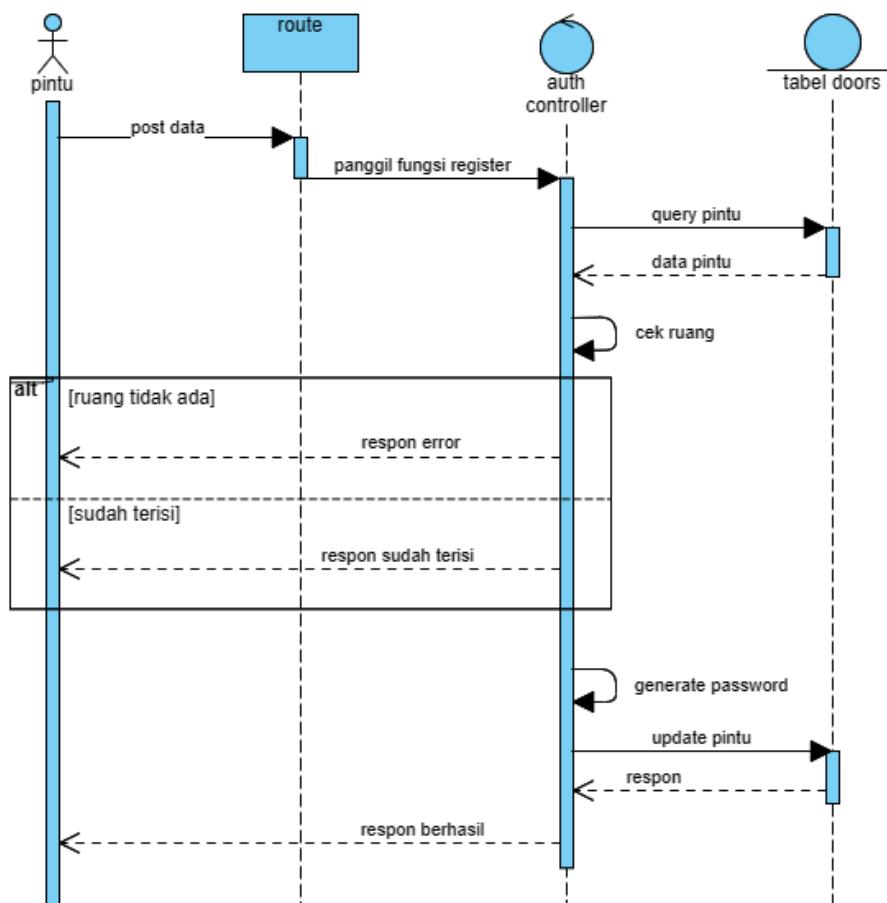


Gambar 3.18 Sequence Diagram API Logout

Pada Gambar 3.18 di atas, sebuah metode logout didalam kontroler akan dipanggil oleh route jika ada perangkat kunci pintu yang melakukan request ke endpoint “/door/logout”. Kemudian kontroler akan menghapus token dari perangkat kunci pintu sesuai dengan token yang dilampirkan didalam header pada saat request diterima. Terakhir kontroler akan mengembalikan respon logout.

3.3.18 API Door Register

API *door register* digunakan untuk menambahkan perangkat penguncian yang baru kedalam pintu. Pada saat operator menambahkan pintu baru melalui *dashboard website* operator maka pintu tersebut belum terpasang perangkat kunci pintu sehingga harus ditambahkan secara manual melalui prosedur pendaftaran. Diagram dari API door register dapat dilihat pada Gambar 3.19 di bawah.



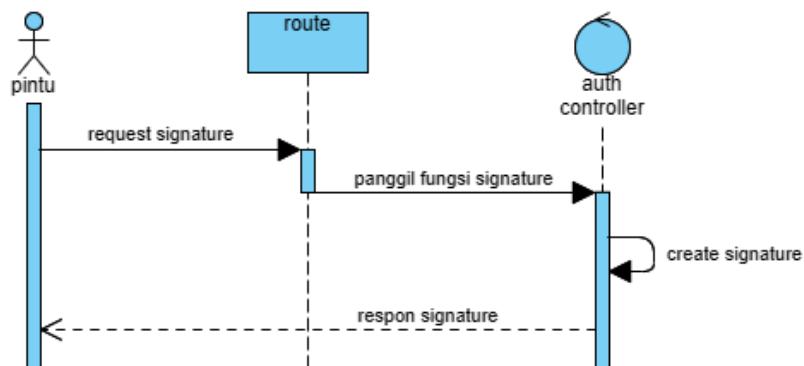
Gambar 3.19 Sequence Diagram API Door Register

Dapat dilihat pada Gambar 3.19 di atas, untuk menambahkan perangkat penguncian baru maka perangkat kunci pintu akan mengirimkan data nama

perangkat yang akan didaftarkan dan kode pintu yang akan ditempati melalui endpoint “/door/register” kemudian kontroler akan mengambil data pada tabel *doors* untuk memastika bahwa pintu masih kosong, jika pintu sudah ada perangkat pengunciannya maka kontroler akan mengembalikan respon sudah terisi dan kontroler akan mengembalikan respon error jika pintu yang dituju tidak ditemukan. Selanjutnya kontroler akan membuat *password* acak yang akan digunakan untuk proses autentikasi, dengan menggunakan metode ini maka akan meningkatkan keamanan karena *password* bersifat acak dan hanya diketahui oleh perangkat kunci pintu dan server. Terakhir kontroler akan mengembalikan respon berhasil direstasi dengan password tadi untuk disimpan pada perangkat kunci pintu.

3.3.19 API Door Signature

API *door signature* digunakan untuk mendapatkan kode unik yang digunakan untuk melakukan *subscribe* ke *channel* pusher. Pusher merupakan sebuah protokol komunikasi notifikasi yang dibangun menggunakan websocket, didalam pusher terdapat *channel-channel* yang dapat di-*subscribe* yang nantinya client dapat menunggu *event* yang disiarkan melalui *broadcast* pada setiap *channel*. Diagram dari API door signature dapat dilihat pada Gambar 3.20 di bawah.



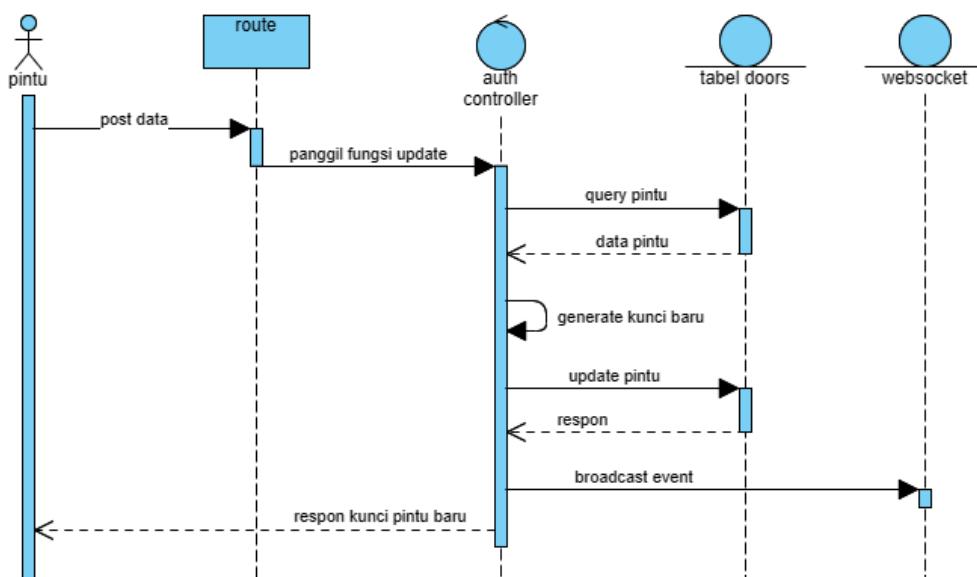
Gambar 3.20 Sequence Diagram API Door Signature

Dapat dilihat pada gamabr 3.20 di atas, 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. Setelah mendapatkan nilai *signature* kemudian kontroler akan mengembalikan respon kode *signature* ke perangkat kunci pintu.

3.3.20 API Door Update Status

API *door update status* digunakan oleh perangkat kunci pintu untuk memperbarui status pintu seperti pintu terbuka, pintu terkunci atau pintu terkoneksi. Pada setiap proses update ini kunci pintu juga akan diperbarui sehingga meningkatkan keamanan karena kode kunci selalu berubah secara dinamis. Diagram dari API door update status dapat dilihat pada Gambar 3.21 di bawah.

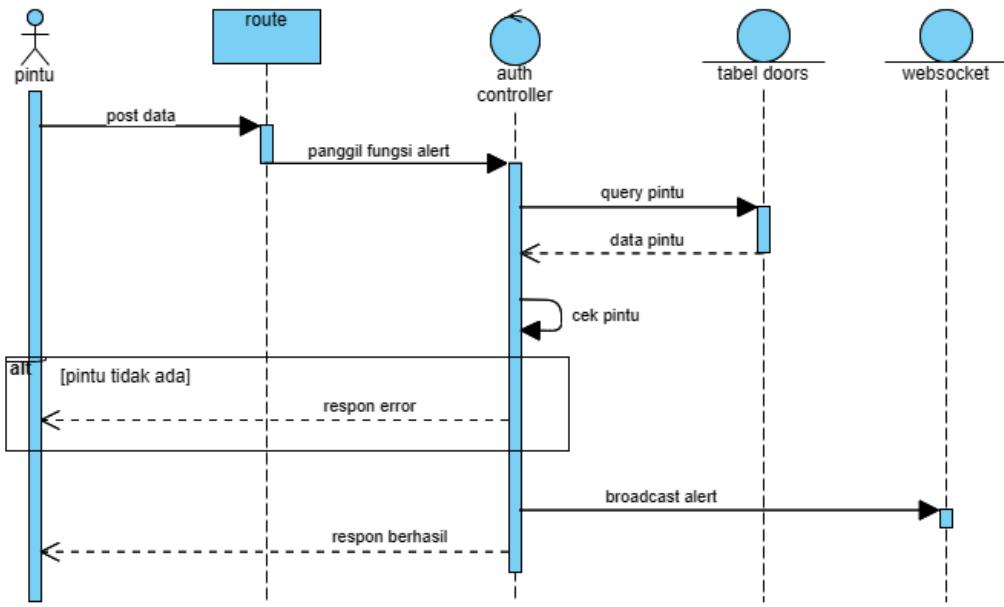


Gambar 3.21 Sequence Diagram API Door Update Status

Terlihat pada Gambar 3.21 di atas, 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.

3.3.21 API Door Alert

API *door alert* digunakan oleh perangkat kunci pintu untuk memberikan peringatan kepada operator bahwa pintu dalam kondisi yang tidak aman seperti terbuka tanpa autentikasi yang sah. Diagram dari API *door alert* dapat dilihat pada Gambar 3.22 di bawah.



Gambar 3.22 Sequence Diagram API Door Alert

Dapat dilihat pada Gambar 3.22 di atas, untuk memberikan peringatan pertama perangkat kunci pintu akan mengirimkan 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 tidak ditemukan maka kontroler akan mengembalikan respon error. Selanjutnya kontroler akan menyiarakan peringatan melalui websocket, terakhir kontroler akan mengembalikan respon peringatan sudah dilaksanakan.

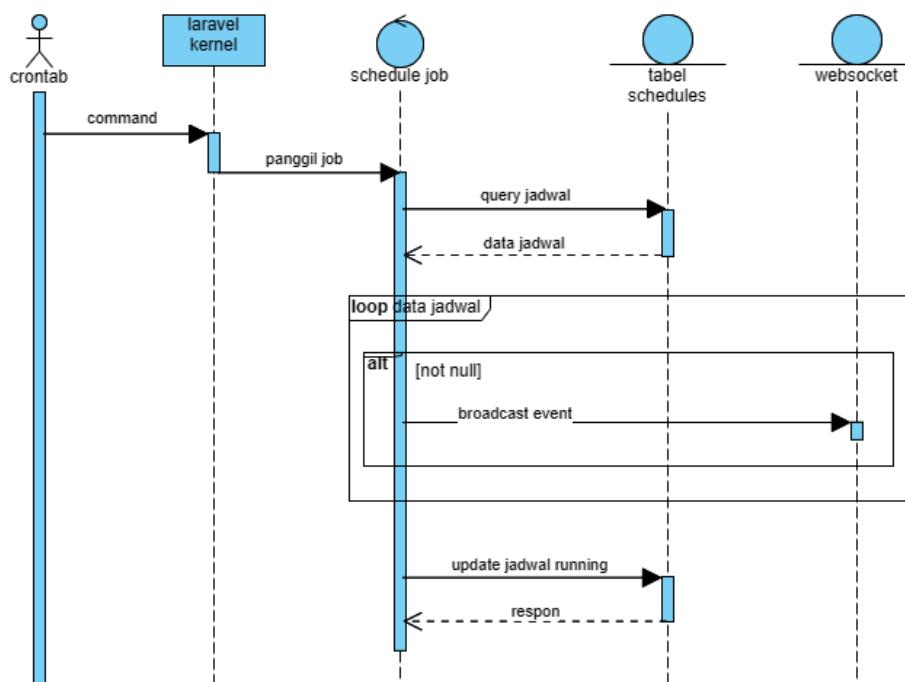
3.5 Perancangan Penjadwalan

Penjadwalan digunakan untuk memeriksa data jadwal yang ada pada tabel schedules. Penjadwalan bekerja dengan menggunakan *kernel* yang disediakan oleh laravel, kernel ini nantinya akan dijalankan oleh laravel dengan menggunakan crontab yang diatur pada sistem operasi server. Crontab akan diatur

untuk menjalankan perintah “*schedule:run*” setiap 1 menit sekali yang akan menjalankan fungsi pengecekan jadwal yang ada pada kernel.

3.4.1 Pengecekan Jadwal

Pengecekan jadwal dilakukan dengan cara memeriksa data jadwal pada tabel *schedules* setiap satu menit sekali. *Kernel* akan menjalankan *job* setiap satu menit sekali untuk memeriksa jadwal, jika ada jadwal yang harus dilaksanakan seperti membuka pintu atau mengunci pintu maka perintah akan dikirimkan ke perangkat kunci pintu melalui websocket. Diagram dari pengecekan jadwal dapat dilihat pada Gambar 3.23 di bawah.



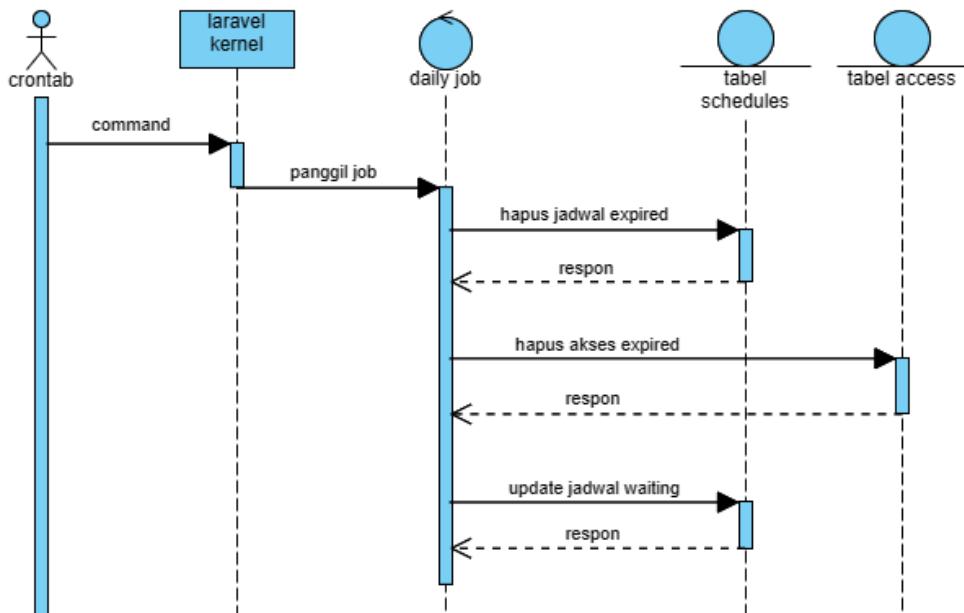
Gambar 3.23 Sequence Diagram Pengecekan Jadwal

Terlihat pada Gambar 3.23 di atas, *crontab* akan mengirimkan perintah ke *kernel* setiap 1 menit sekali dan *kernel* akan menjalankan sebuah *job* yang digunakan untuk memeriksa jadwal. Pertama *job* tersebut akan mengambil data jadwal hari ini yang akan dilaksanakan tentunya dengan menggunakan *query* untuk mengambil data yang sesuai dengan memperhatikan tanggal, waktu dan status jadwal, setelah data didapatkan kemudian *job* akan mengirimkan event ke setiap perangkat kunci pintu yang ada pada data jadwal tersebut, setelah semua

event dikirimkan kemudian *job* akan memperbarui status jadwal menjadi “*running*” atau sedang berjalan. Terakhir *job* akan memperbarui semua jadwal yang sudah dijalankan dan sudah melewati waktu pelaksanaan menjadi “*done*” atau sudah selesai dilaksanakan.

3.4.2 Atur Ulang Jadwal

Setiap jadwal yang sudah dilaksanakan harus dihapus dari *database* dan jadwal yang berulang harus diatur statusnya seperti awal mula sehingga memerlukan proses atur ulang jadwal untuk membersihkan dan mengatur ulang jadwal. Jadwal akan dibersihkan dan diatur ulang setiap 24 jam sekali dengan menggunakan sebuah *job* pada *kernel* Laravel. Diagram dari *job* atur ulang jadwal dapat dilihat pada Gambar 3.24 di bawah.



Gambar 3.24 Sequence Diagram Atur Ulang Jadwal

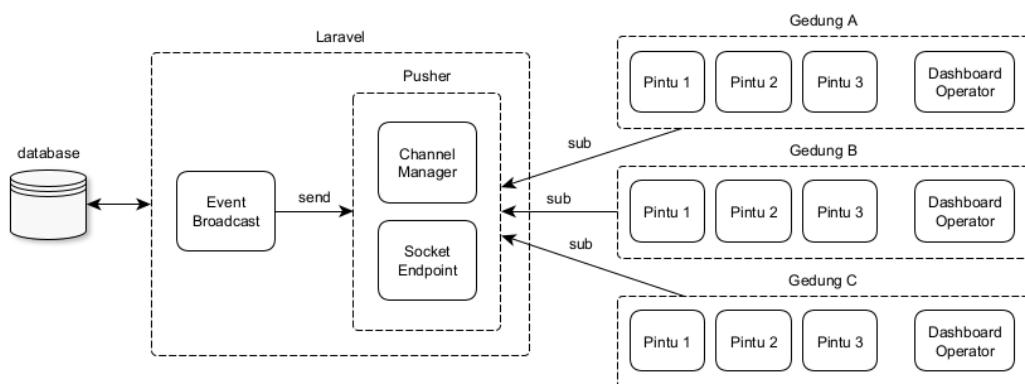
Dapat dilihat pada Gambar 3.24 di atas, *crontab* akan mengirimkan perintah ke *kernel* untuk menjalankan job atur ulang jadwal. Didalam *job* tersebut pertama *job* akan menghapus data jadwal yang sudah kadaluarsa pada tabel *schedules* yaitu jadwal yang sudah dilaksanakan dan tidak berulang, kemudian pada *job* ini juga akan menghapus kartu akses pengguna yang sudah kadaluarsa,

kemudian job akan memperbarui setiap jadwal yang sudah pernah dilaksanakan menjadi “*waiting*” atau menunggu untuk dilaksanakan kembali.

3.6 Perancangan Komunikasi Websocket

Websocket digunakan sebagai jalur komunikasi yang menghubungkan antara *server* dengan perangkat kunci pintu, dengan adanya komunikasi websocket maka perangkat kunci pintu dan *server* akan selalu terhubung sehingga dapat berkomunikasi secara langsung.

Pada perancangan *backend server* untuk mendukung kinerja perangkat kunci pintu ini menggunakan pusher sebagai protokol komunikasi websocket yang menghubungkan antara perangkat kunci pintu dengan *server* dengan konfigurasi seperti yang terlihat pada Gambar 3.25 di bawah.



Gambar 3.25 Konfigurasi Websocket

Dapat dilihat pada Gambar 3.25 di atas, setiap pintu akan dikelompokkan berdasarkan dengan gedung dengan satu orang operator, setiap gedung akan memiliki satu *channel* pusher yang dapat di-*subscribe* oleh perangkat penguncian didalam gedung tersebut, untuk mengawasi semua aktifitas yang terhubung ke *channel* dengan mudah maka konfigurasi channel menggunakan *presence channel*, dengan menggunakan *presence channel* maka aktifitas semua perangkat kunci pintu seperti perangkat kunci melakukan *subscribe* dan koneksi terputus dapat diketahui secara langsung.

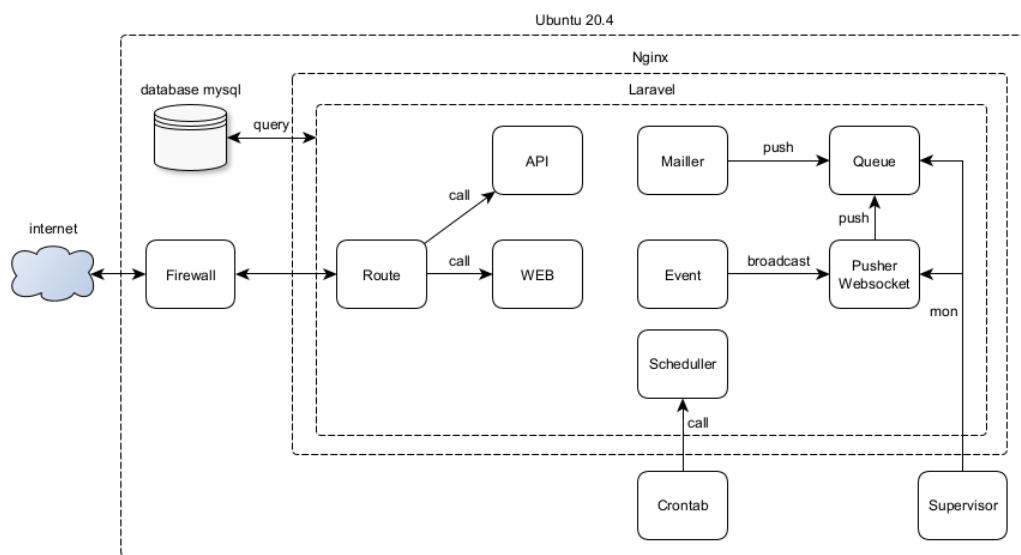
Laravel mengirimkan data atau perintah ke perangkat kunci pintu melalui pusher menggunakan sebuah *event*, setiap *event* seperti kunci atau buka pintu

dipanggil baik melalui website maupun API maka *event* tersebut akan disiarkan ke *channel* sesuai dengan kode perintah dan *channel* yang dituju.

Didalam pusher sendiri terdapat 2 komponen utama yaitu *channel manager* dan *socket endpoint*, *channel manager* digunakan untuk mengelola semua channel yang ada serta mengelola semua *subsciber* dan melakukan *broadcast event*, sedangkan *socket endpoint* digunakan sebagai pintu masuk berupa URL yang digunakan untuk memulai koneksi dengan websocket.

3.7 Perancangan 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 3.26 di bawah.



Gambar 3.26 Konfigurasi Server

Dapat dilihat pada Gambar 3.26 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, 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. Sertifikat tersebut disediakan oleh pihak ke-3 yaitu menggunakan letsencrypt sebagai penyedia sertifikat HTTPS gratis.

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 didalam 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 suka 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.

BAB IV

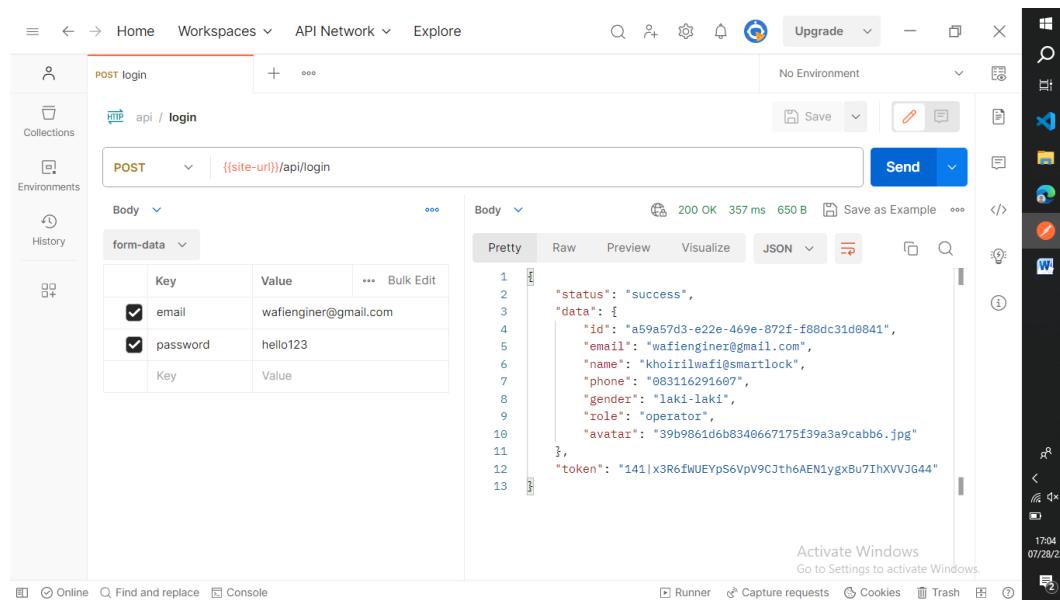
HASIL DAN PEMBAHASAN

4.1 Kajian Hasil Perancangan

Untuk melihat hasil dari perancangan API aplikasi mobile dan perangkat kunci pintu maka dilakukan uji coba menggunakan postman. Postman merupakan sebuah software yang digunakan untuk melakukan request ke API, dengan menggunakan postman maka akan terlihat hasil respon yang dikirimkan oleh server sesuai dengan permintaan *client* melalui *endpoint* tertentu.

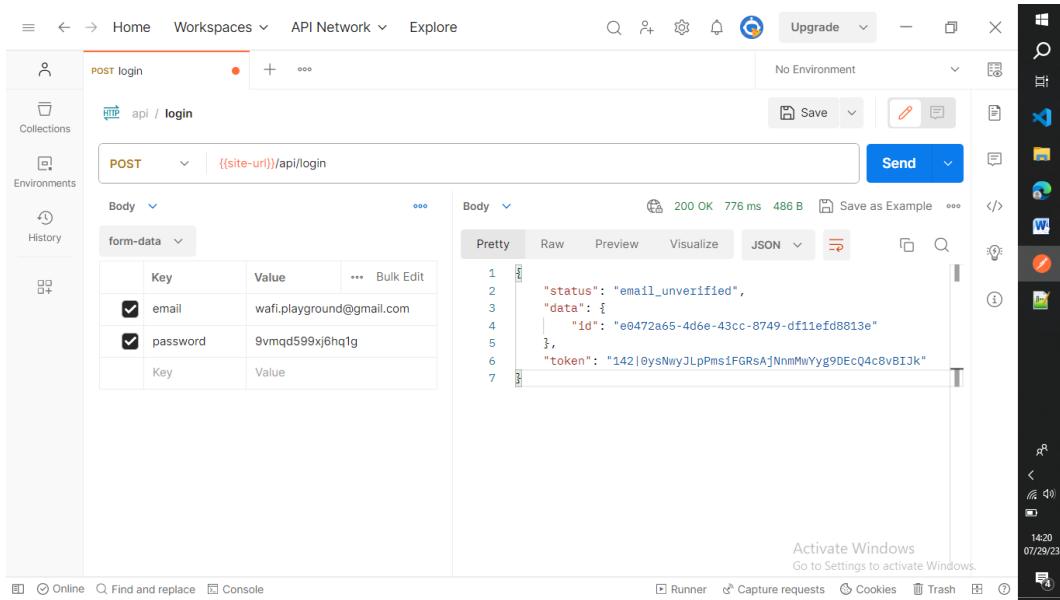
4.1.1 API Login

API *login* akan menerima alamat email dan *password* pengguna untuk melakukan autentikasi. Hasil dari perancangan dari API *login* dapat dilihat pada Gambar 4.1 di bawah.



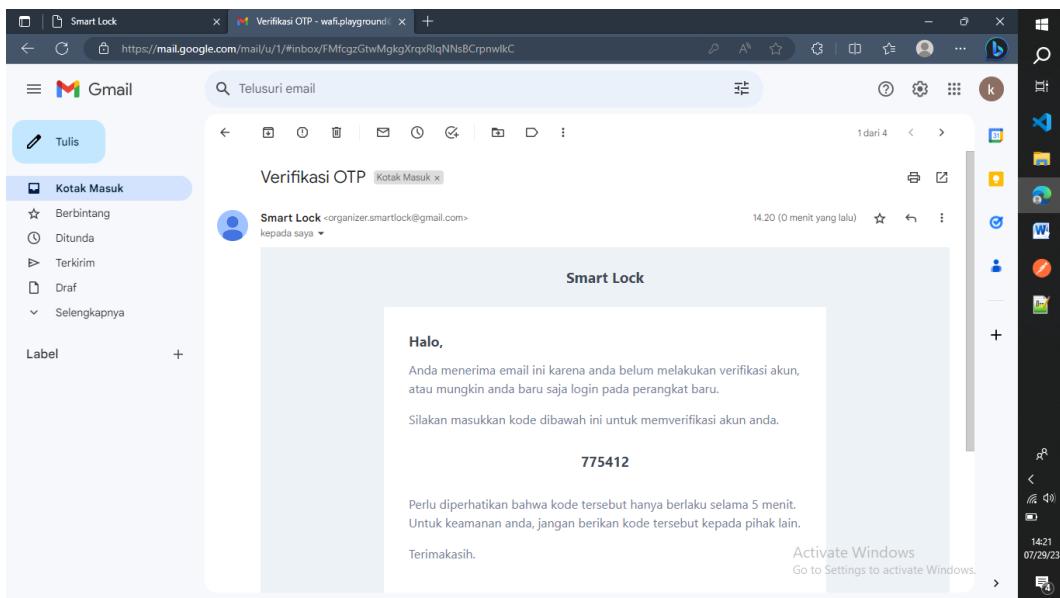
Gambar 4.1 Hasil API Login

Gambar 4.1 menampilkan hasil *request login* yang dikirimkan melalui *endpoint* “/api/login”. API *login* akan memberikan respon status *login*, data pengguna yang melakukan *login* serta token yang akan digunakan untuk mengakses API lainnya yang membutuhkan autentikasi token.



Gambar 4.2 Hasil API *Login* Email Belum Terverifikasi

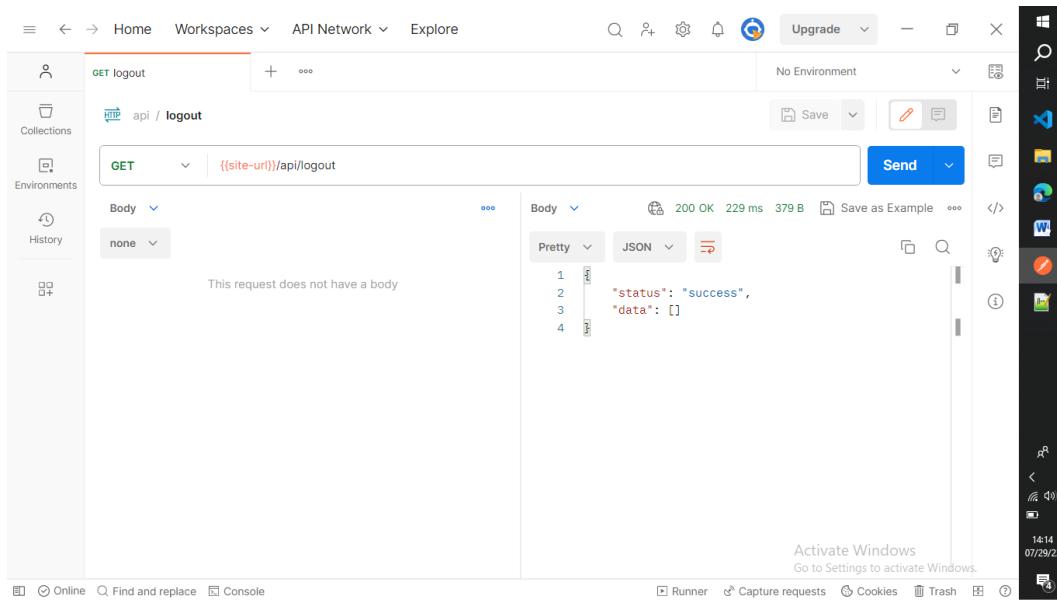
Pada API *login* juga dilakukan pengecekan email pengguna, seperti yang terlihat pada Gambar 4.2, pengguna yang belum melakukan verifikasi email akan menerima respon status email belum terverifikasi dan pengguna akan menerima kode OTP yang dikirimkan ke email pengguna seperti yang terlihat pada Gambar 4.3. Pada kondisi ini API juga memberikan token yang akan digunakan oleh pengguna untuk melakukan verifikasi email karena API untuk melakukan verifikasi email membutuhkan akses *login*.



Gambar 4.3 Email Kode OTP

4.1.2 API Logout

API *logout* digunakan untuk mengakhiri sesi *login* pengguna dengan menghapus semua token yang dimiliki oleh pengguna. API *logout* akan menerima permintaan *logout* pengguna dengan melampirkan token yang dimiliki. Hasil dari API *logout* dapat dilihat pada Gambar 4.4 di bawah.

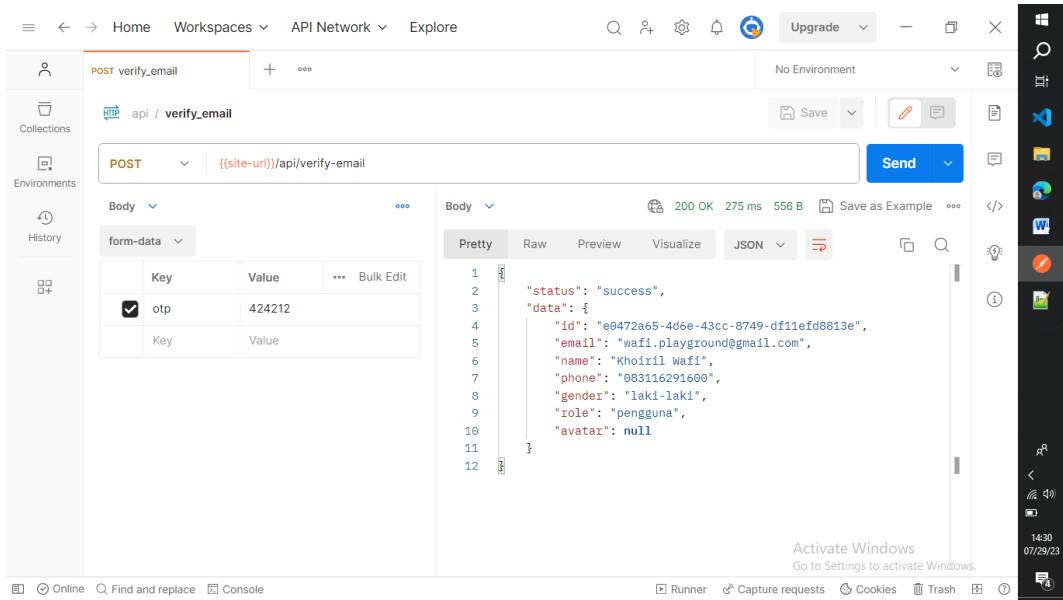


Gambar 4.4 Hasil API *Logout*

Gambar 4.4 menampilkan hasil *request* dari API *logout* yang dikirimkan melalui *endpoint* “/api/logout”. API *logout* akan memberikan respon status *logout* yang dilakukan oleh pengguna dengan melampirkan *token* yang dimiliki oleh pengguna yang bersangkutan pada bagian *header request*.

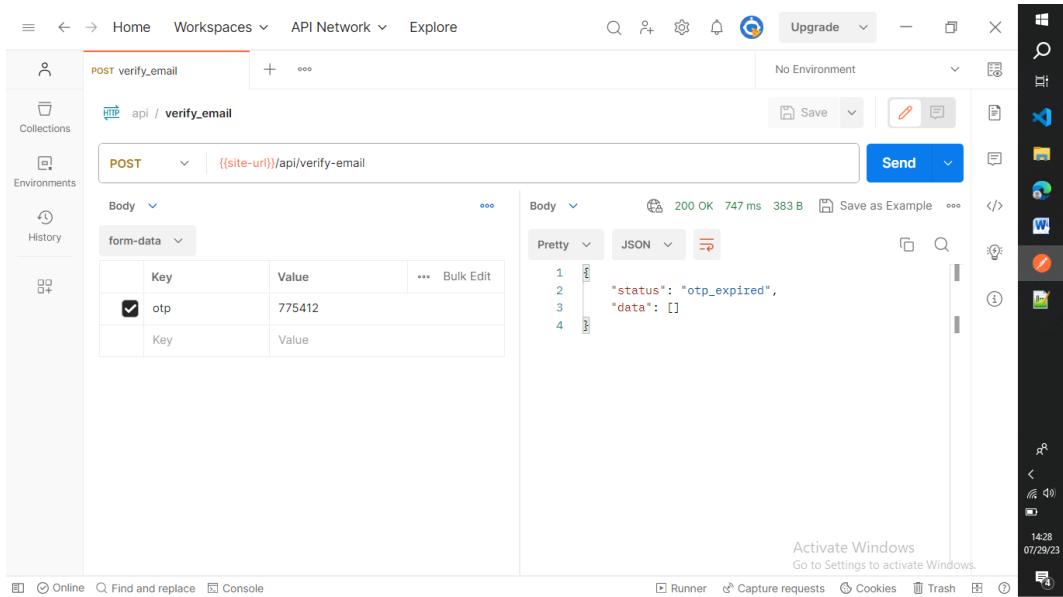
4.1.3 API Verifikasi Email

Pada saat pengguna melakukan login dan mendapatkan respon email belum terverifikasi maka pengguna akan mendapatkan sebuah kode OTP yang dikirimkan ke email pengguna. API verifikasi email akan menerima kode OTP yang telah diterima oleh pengguna untuk melakukan verifikasi email. Dengan menggunakan langkah verifikasi ini maka hanya pengguna atau operator dengan email yang valid dan aktif yang dapat mengakses API sehingga akan meningkatkan keamanan dari API tersebut. Hasil dari API verifikasi email dapat dilihat pada Gambar 4.5 di bawah.



Gambar 4.5 Hasil API Verifikasi Email

Gambar 4.5 menampilkan hasil dari API verifikasi email yang dikirimkan melalui *endpoint* “api/verify-email”. Pengguna akan mengirimkan kode OTP melalui *body* serta melampirkan token *login* pada *header request*, jika verifikasi berhasil maka API akan memberikan respon status berhasil dan mengirimkan data pengguna seperti pada saat *login*. API verifikasi email juga akan memeriksa waktu kadaluarsa dari kode OTP, jika kode OTP sudah kadaluarsa maka verifikasi akan gagal seperti yang terlihat pada Gambar 4.6 di bawah.



Gambar 4.6 Kode OTP Kadaluarsa

4.1.4 API Reset Password

API *reset password* akan menerima email dari pengguna dan mengirimkan url yang dapat digunakan untuk membuat *password* baru ke email pengguna. Hasil dari API *reset password* dapat dilihat pada Gambar 4.7 di bawah.

The screenshot shows the Postman interface. On the left, there's a sidebar with 'Collections', 'Environments', and 'History'. The main area shows a POST request to 'http://api/reset_password'. The request body is set to 'form-data' and contains a single key-value pair: 'email' with the value 'wafi.playground@gmail.com'. The response tab shows a 200 OK status with a response time of 768 ms and a size of 414 B. The response body is displayed in JSON format:

```

1  "status": "success",
2  "data": {
3    "email": "wafi.playground@gmail.com"
4  }
5
6

```

Gambar 4.7 Hasil API Reset Password

The screenshot shows a Gmail inbox. The left sidebar shows 'Kotak Masuk' with labels: Berbintang, Ditunda, Terkirim, Draf, Selengkapnya. The inbox lists an email from 'Smart Lock <organizer.smartlock@gmail.com>' with the subject 'Reset Password'. The email body reads:

Smart Lock

Halo,

Anda menerima email ini karena kami menerima permintaan reset password untuk akun Anda.

Silakan klik tombol di bawah ini untuk mereset password Anda

Reset Password

Perlu diperhatikan bahwa link reset password anda hanya berlaku selama 60 menit. Jika Anda tidak meminta reset password, silahkan abaikan email ini.

Terimakasih.

Gambar 4.8 Email Reset Password

Gambar 4.7 merupakan hasil API *reset password* yang dikirimkan melalui endpoint “/api/reset-password”. Pengguna akan mengirimkan alamat email pada *body request* dan API akan memberikan respon status beserta data alamat email.

Pada proses ini API juga akan mengirimkan url *reset password* ke email pengguna seperti yang terlihat pada Gambar 4.8, pengguna dapat menggunakan url tersebut untuk membuat *password* baru melalui halaman *website*.

4.1.5 API Ganti Password

API ganti password akan menerima *password* lama dan *password* baru pengguna untuk memperbarui *password* pengguna. API ganti *password* juga akan memeriksa token *login* yang dikirimkan untuk memastikan bahwa *pengguna* sudah melakukan proses autentikasi *login*. Hasil dari API ganti *password* dapat dilihat pada Gambar 4.9 di bawah.

The screenshot shows the Postman interface with the following details:

- Request URL:** POST {{site-url}}/api/change-password
- Body:** form-data

Key	Value
password_now	hello123
password	admin@smartlock
password_confirmation	admin@smartlock
- Response Status:** 200 OK (711 ms, 405 B)
- Response Body (Pretty JSON):**

```

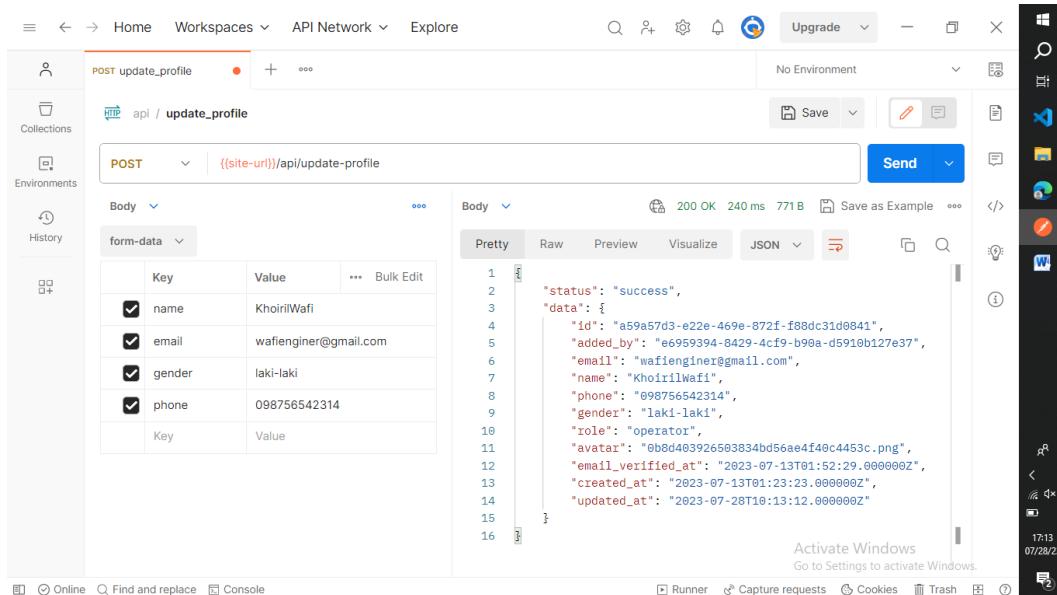
1  {
2   "status": "success",
3   "data": {
4     "message": "password saved"
5   }
6 }
```

Gambar 4.9 Hasil API Ganti Password

Gambar 4.9 merupakan hasil dari API ganti *password* yang dikirimkan melalui *endpoint* “/api/change-password”. Pengguna akan mengirimkan *password* lama dan *password* baru serta konfirmasi *password* baru pada *body request*, pengguna juga memerlukan token *login* untuk mengakses *endpoint* ini. API ganti *password* akan memberikan respon status dan pesan apakah *password* baru berhasil tersimpan atau tidak. Dengan adanya API ganti password maka pengguna dan operator dapat mengganti *password* secara berkala sesuai dengan kehendak masing-masing untuk meningkatkan kemanan akun mereka.

4.1.6 API Ganti Profil

API ganti profil akan menerima data pengguna yang akan diganti seperti nama, email, jenis kelamin dan nomor hp, API ganti profil juga memerlukan token *login* untuk memastikan pengguna sudah melakukan autentikasi. Hasil dari API ganti profil dapat dilihat pada Gambar 4.10 di bawah.

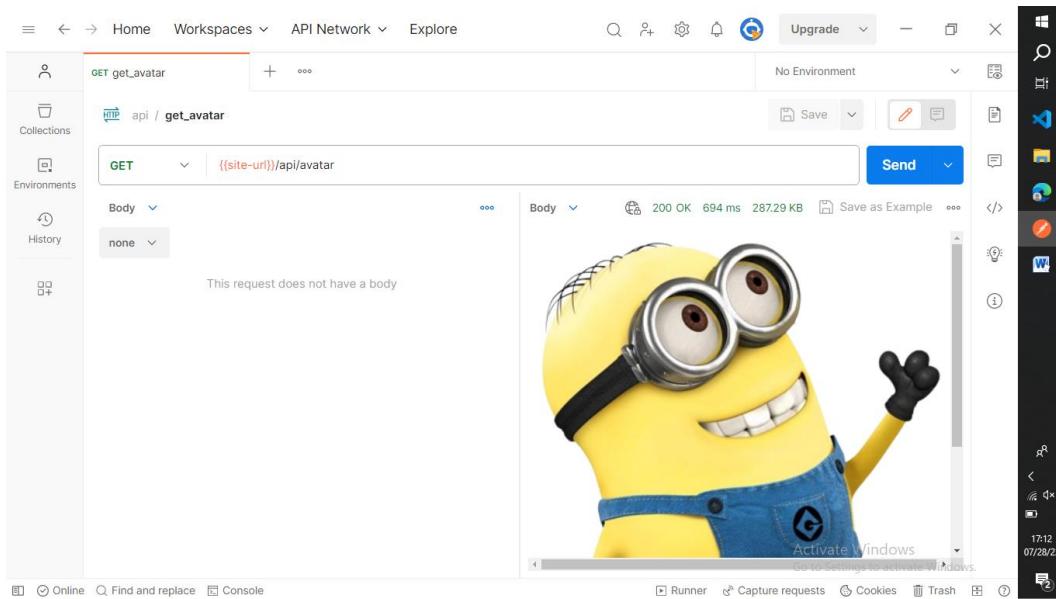


Gambar 4.10 API Ganti Profil

Gambar 4.10 merupakan hasil dari API ganti profil yang dikirimkan melalui *endpoint* “/api/update-profile”. Pengguna akan mengirimkan data yang akan diganti seperti nama, email, jenis kelamin dan nomor hp pada *body request*, pengguna juga menyertakan token *login* pada *header request*. API akan memberikan respon berupa status dan data pengguna yang sudah dirubah seperti yang terlihat pada Gambar 4.10 di atas.

4.1.7 API Lihat Avatar

Setiap gambar avatar pengguna tidak tersedia untuk publik, pengguna akan melakukan *request* untuk mendapatkan gambar avatar dengan melampirkan token *login* ke API lihat avatar. API lihat avatar akan memberikan respon berupa gambar avatar dari setiap pengguna atau operator untuk ditampilkan pada aplikasi masing-masing mobile. Hasil dari API lihat avatar dapat dilihat pada Gambar 4.11 di bawah.

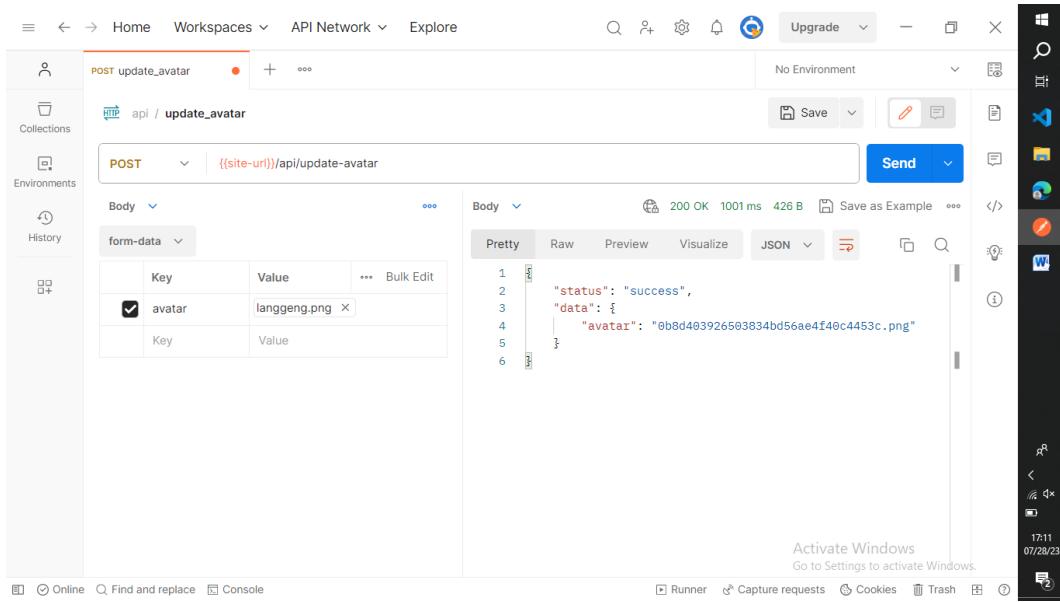


Gambar 4.11 Hasil API Lihat Avatar

Gambar 4.11 merupakan hasil dari API lihat avatar yang dikirimkan melalui *endpoint* “/api/avatar”, pengguna akan mengirimkan permintaan disertai dengan token *login* dan API akan memberikan respon berupa gambar avatar masing-masing pengguna dalam format file gambar. Jika pengguna belum memiliki avatar maka API akan mengembalikan respon berupa gambar *default* untuk avatar yang terletak di server.

4.1.8 API Ganti Avatar

Pengguna dan operator dapat memperbarui gambar avatar melalui aplikasi mobile. Untuk mengganti avatar melalui aplikasi mobile pengguna akan mengirimkan gambar avatar melalui API ganti avatar, pengguna juga memerlukan akses *login* untuk mengakses API ganti avatar. Pada API Ganti avatar pengguna hanya diijinkan untuk mengunggah file gambar dengan ukuran maksimal 1 MB hal ini bertujuan untuk mencegah pengguna mengunggah file lain seperti dokumen atau pdf dan juga untuk membatasi ukuran file proses pengiriman gambar dapat berjalan dengan lancar. Hasil dari API ganti avatar dapat dilihat pada Gambar 4.12 di bawah ini.

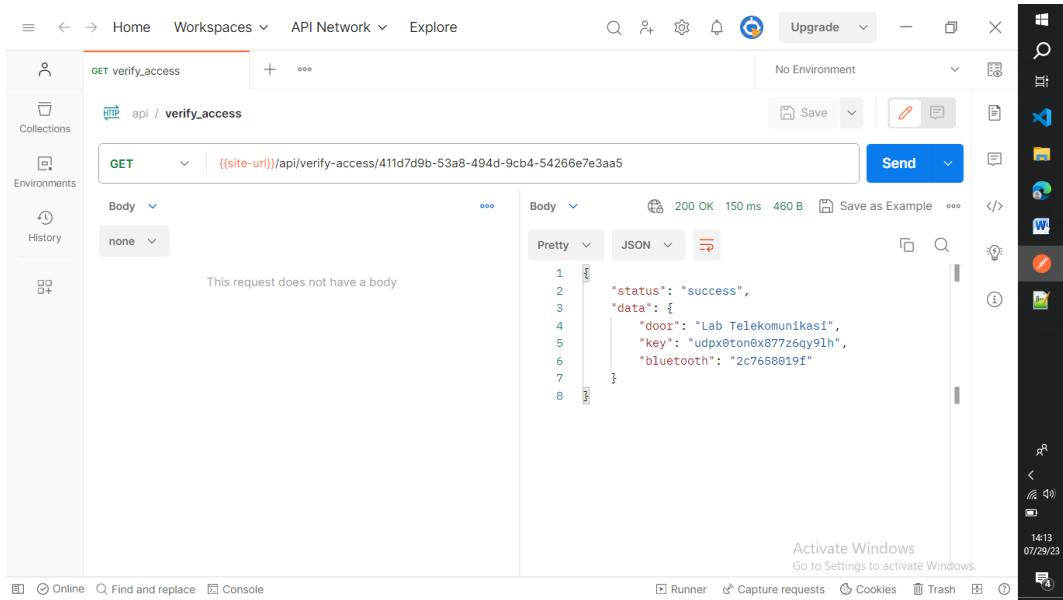


Gambar 4.12 Hasil API Ganti Avatar

Gambar 4.12 merupakan hasil dari API ganti avatar yang dikirimkan melalui *endpoint* “/api/update-avatar”. Pengguna akan mengirimkan file gambar avatar ke API disertai dengan token *login* yang dimiliki, API akan memberikan respon status dan nama file avatar yang berhasil tersimpan didalam server.

4.1.9 API Verifikasi Akses

Pada setiap pintu akan terpasang kode QR yang menjadi identitas dari pintu tersebut. Pengguna dan operator dapat menggunakan kode QR tersebut untuk membuka pintu. Ketika pengguna atau operator melakukan pemindaian kode QR untuk membuka pintu maka perangkat mobile akan mengirimkan request ke API verifikasi akses untuk mendapatkan kunci dari pintu tersebut. Kode QR yang terpasang nantinya akan menyimpan informasi identitas dari pintu yang digunakan untuk membedakan pintu satu dengan yang lainnya. Dengan menggunakan metode ini maka akses pengguna dapat diatur secara langsung karena setiap ingin membuka kunci pintu maka pengguna harus meminta kunci pintu melalui API verifikasi akses sehingga akan meningkatkan keamanan sistem penguncian pintu gedung ini. Hasil dari API verifikasi akses dapat dilihat pada Gambar 4.13 di bawah.

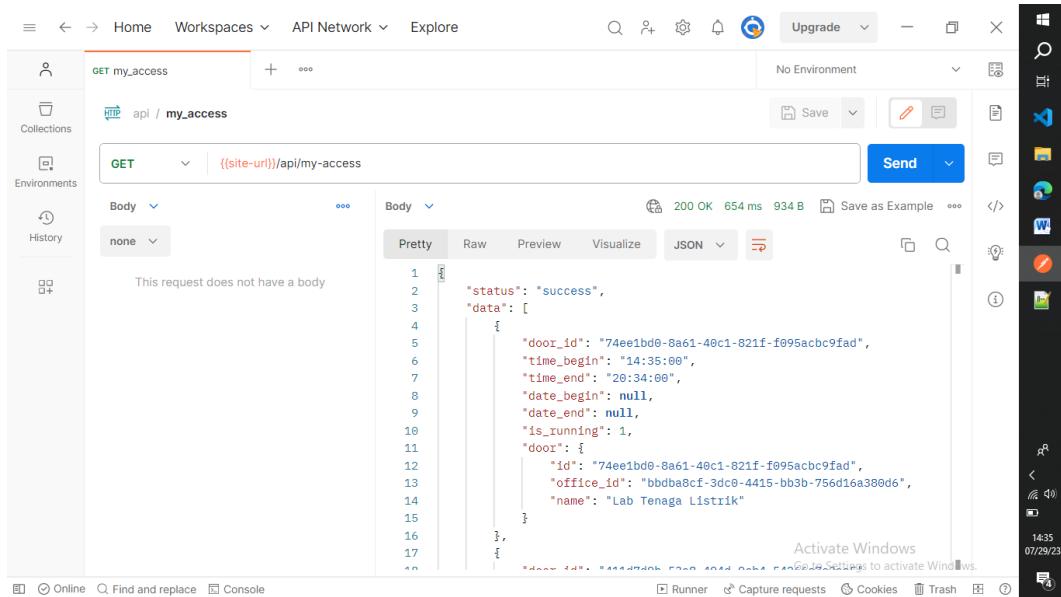


Gambar 4.13 Hasil API Verifikasi Akses

Gambar 4.13 merupakan hasil dari API verifikasi akses yang dikirimkan melalui *endpoint* “/api/verify-access/{id}” dimana id merupakan identitas dari pintu. Pengguna akan mengirimkan token *login* yang dimiliki dan API akan memberikan respon terkait akses yang dimiliki pengguna tersebut, jika diizinkan maka API akan memberikan respon status diijinkan dan API akan memberikan respon data yang digunakan untuk membuka pintu seperti nama pintu, kunci pintu dan nama bluetooth untuk berkomunikasi dengan perangkat kunci pintu. Namun jika akses pengguna tidak ditemukan atau akses pengguna sedang diblokir maka API verifikasi akses akan memberikan respon gagal atau tidak diijinkan dan tidak akan memberikan data untuk membuka pintu.

4.1.10 API Daftar Akses

API daftar akses akan memberikan informasi pintu mana saja yang dapat diakses oleh pengguna. Setiap pengguna dengan pengguna lain mungkin memiliki akses yang berbeda, pemberian akses pintu kepada pengguna tertentu akan diatur oleh seorang operator. Seorang operator dapat memberikan akses atau menghapus akses pengguna pada pintu tertentu, operator juga dapat menghentikan sementara akses pengguna untuk keperluan tertentu. Hasil dari API daftar akses dapat dilihat pada Gambar 4.14 di bawah.

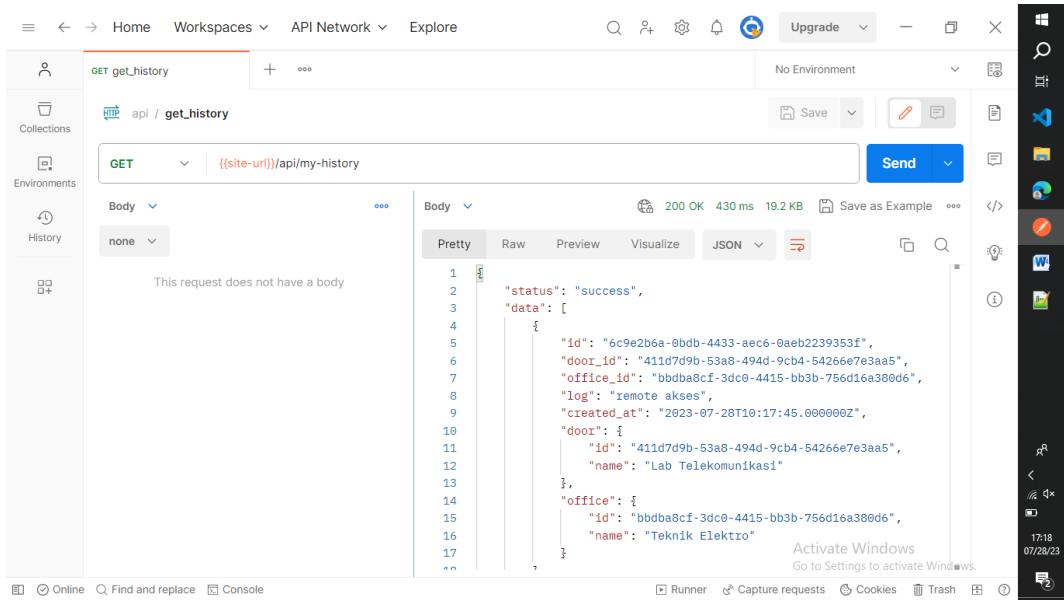


Gambar 4.14 Hasil API Daftar Akses

Gambar 4.14 merupakan hasil dari API daftar akses yang dikirimkan oleh pengguna melalui endpoint “/api/my-access”. API daftar akses akan memberikan respon berupa daftar informasi pintu yang dapat diakses oleh pengguna tersebut yaitu berupa nama pintu, nama gedung, rentang waktu akses, rentang tanggal akses dan status akses. Dengan menggunakan data akses yang dimiliki pengguna dapat mengetahui pintu mana saja yang dapat diakses oleh pengguna tersebut dan juga mengetahui jika akses pengguna masih aktif atau sedang ditangguhkan. Dengan adanya informasi ini pengguna dapat mengelola akses mereka dengan efisien sesuai dengan kebutuhan masing-masing pengguna.

4.1.11 API Riwayat Akses

API riwayat akses akan memberikan informasi kepada pengguna terkait dengan catatan aktifitas pengguna didalam sistem penguncian gedung ini dengan menampilkan data riwayat aktifitas pada aplikasi pengguna. API ini dapat diakses oleh pengguna maupun operator untuk melihat aktifitas mereka didalam sistem penguncian pintu gedung ini. Pencatatan aktifitas dilakukan berdasarkan kegiatan pengguna dan operator seperti memindai kode QR, membuka kunci pintu melalui akses *remote*, membuat jadwal pintu otomatis dan lain sebagainya. Hasil dari API dapat dilihat pada Gambar 4.15 di bawah.

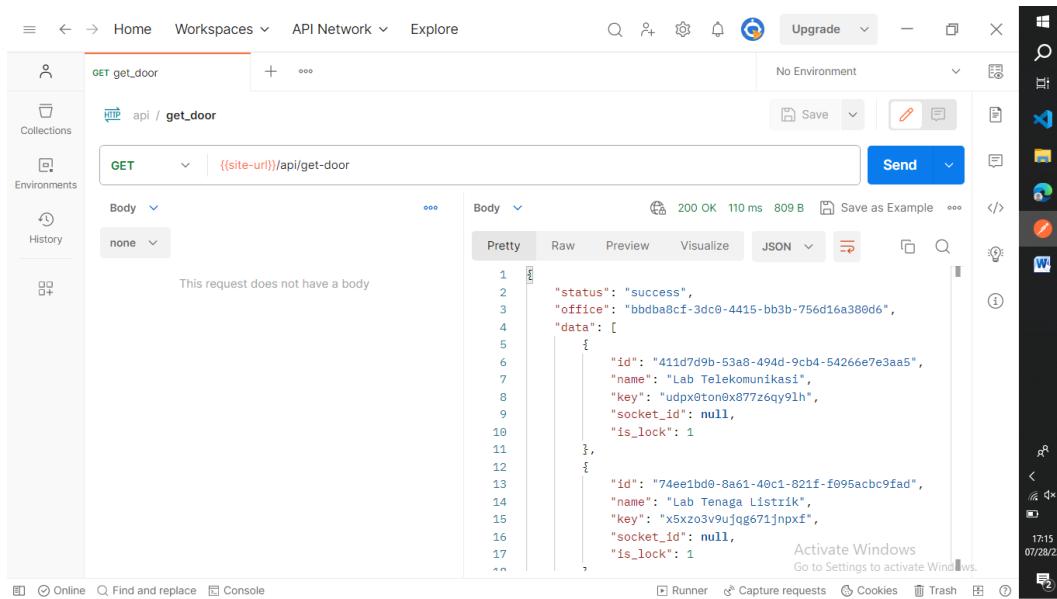


Gambar 4.15 API Riwayat Akses

Gambar 4.15 merupakan hasil dari API riwayat akses. API riwayat akses akan memberikan data pencatatan aktifitas pengguna seperti melakukan pindai kode QR dan membuka pintu disertai dengan waktu dan tanggal aktifitas tersebut. Setiap aktifitas penguncian akan tersimpan didalam *database* sebagai riwayat aktifitas, seorang operator dapat melihat semua aktifitas yang terjadi didalam sistem penguncian tetapi pengguna hanya dapat melihat aktifitas dirinya sendiri. Pada Gambar 4.15 terlihat bahwa data riwayat aktifitas berupa nama pintu, waktu, dan jenis aktifitas yang dilakukan. Data riwayat akses hanya tersedia dalam kurun waktu 1 bulan dan setiap awal bulan maka data riwayat akses akan di pindahkan kedalam penyimpanan server (*backup*) dan tabel riwayat akses dalam *database* akan dibersihkan.

4.1.12 API Daftar Pintu

API daftar pintu digunakan untuk mendapatkan daftar pintu yang ada pada sebuah gedung, data tersebut kemudian ditampilkan pada aplikasi mobile yang akan memberikan informasi terkait kondisi pintu apakah terkunci, *offline* atau kondisi lainnya. API ini hanya dapat diakses oleh operator gedung karena hak pengelolaan pintu gedung berada pada tangan operator. Hasil dari API daftar pintu dapat dilihat pada Gambar 4.16 di bawah.

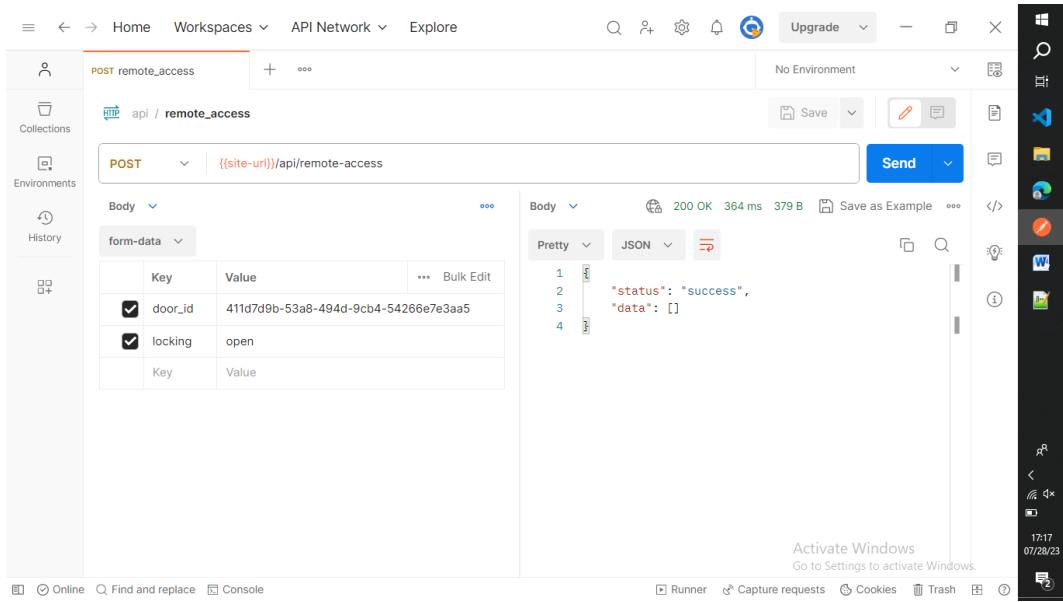


Gambar 4.16 Hasil API Daftar Pintu

Gambar 4.16 merupakan hasil dari API daftar gedung yang dikirimkan melalui *endpoint* “/api/get-door”. Pengguna akan mengirimkan token login yang dimiliki, jika pengguna merupakan seorang operator maka API akan memberikan respon daftar pintu yang ada pada gedung tersebut beserta dengan informasi tambahan seperti nama pintu, kondisi penguncian, koneksi, dan kunci pintu. Namun jika pengguna bukan seorang operator maka API akan memberikan respon bahwa pengguna tersebut tidak diijinkan untuk mengakses API ini.

4.1.13 API *Remote Pintu*

Pada aplikasi yang digunakan oleh operator terdapat tombol yang dapat digunakan untuk membuka dan mengunci pintu sehingga memungkinkan operator dapat melakukan pengelolaan pintu secara jarak jauh melalui koneksi internet. Untuk membuka atau mengunci pintu secara jarak jauh melalui koneksi internet, maka seorang operator akan melakukan *request* dengan mengirimkan identitas pintu beserta token *login* yang dimiliki, jika sesuai maka server akan mengirimkan perintah ke perangkat kunci pintu melalui koneksi websocket yang sudah terhubung. Namun jika terjadi permasalahan maka API akan memberikan respon error. Hasil dari API remote pintu dapat dilihat pada Gambar 4.17 di bawah.



Gambar 4.17 Hasil API Remote Pintu

Gambar 4.17 merupakan hasil dari API *remote* pintu yang dikirimkan melalui *endpoint* “/api/remote-access”. Untuk membuka atau mengunci pintu secara jarak jauh melalui koneksi internet operator akan mengirimkan identitas pintu yang dituju serta aktifitas yang dilakukan seperti *lock* atau *open* yang menjelaskan aktifitas membuka dan mengunci pintu, kemudian server akan memeriksa permintaan dan mengirimkan perintah ke perangkat kunci pintu melalui koneksi websocket seperti yang terlihat pada Gambar 4.18 di bawah.

The screenshot shows a Notepad window with a log file titled 'websocket.log'. The log contains several lines of JSON messages related to a websocket connection. The messages show the exchange between a client and a server. Key lines include:

- Line 877: 'Pc8LLXsNgnzmq2x9vV2U: connection id 350611825.967385339 received message: {"event": "pusher:ping", "data": {}}.'
- Line 878: 'Connection id 350611825.967385339 sending message {"event": "pusher:pong"}'
- Line 879: 'Connection id 123981025.486971429 sending message {"channel": "presence-office.bbdba8cf-3dc0-4415-bb3b-756d16a380d6", "event": "door-command", "data": {"user_id": "\\"a59a57d3-e22e-469e-872f-f88dc31d0841\\", "door_id": "\\"411d7d9b-53a8-494d-9cb4-54266e7e3aa5\\", "locking": "\\"open\\", "key": "\\"18fyj6fibzrf910yxwcs\\\""}'}
- Line 880: 'Connection id 350611825.967385339 sending message {"channel": "presence-office.bbdba8cf-3dc0-4415-bb3b-756d16a380d6", "event": "door-command", "data": {"user_id": "\\"a59a57d3-e22e-469e-872f-f88dc31d0841\\", "door_id": "\\"411d7d9b-53a8-494d-9cb4-54266e7e3aa5\\", "locking": "\\"open\\", "key": "\\"18fyj6fibzrf910yxwcs\\\""}' (repeated)

The bottom of the window shows the status bar with 'Normal text file', 'length: 2,401,926', 'line: 21,359', 'Ln: 879', 'Col: 1', 'Pos: 122,311', 'Unix (LF)', 'UTF-8', and 'INS'.

Gambar 4.18 Aktifitas Door Remote

Gambar 4.18 di atas memperlihatkan aktifitas websoket pada *server*, pada saat API remote pintu dipanggil maka *server* akan mengirimkan perintah ke perangkat kunci pintu pada *channel* yang sudah ditentukan. WebSocket akan mengirimkan data identitas dari pintu, aktifitas yang harus dilakukan serta kode kunci dari pintu tersebut. Sebagai tabahan untuk pencatatan riwayat aktifitas maka ditambahkan identitas aktor yang melakukan request ke API tersebut.

4.1.14 API Door Login

API *door login* digunakan oleh setiap perangkat kunci pintu untuk mendapatkan token akses. Setiap pintu akan mengirimkan *username* dan *password* yang dimiliki untuk mendapatkan token akses. Hasil dari API *door login* dapat dilihat pada Gambar 4.19 di bawah.

The screenshot shows the Postman interface with a successful API call to the 'door_api / login' endpoint. The request method is POST, and the URL is {{site-url}}/door/login. The request body is a form-data object with two fields: 'device_name' (cc:db:a7:5a:34:80) and 'device_pass' (t6ux4kqv9wichpuo3ivn). The response status is 200 OK, with a response time of 904 ms and a response size of 489 B. The response body is a JSON object:

```

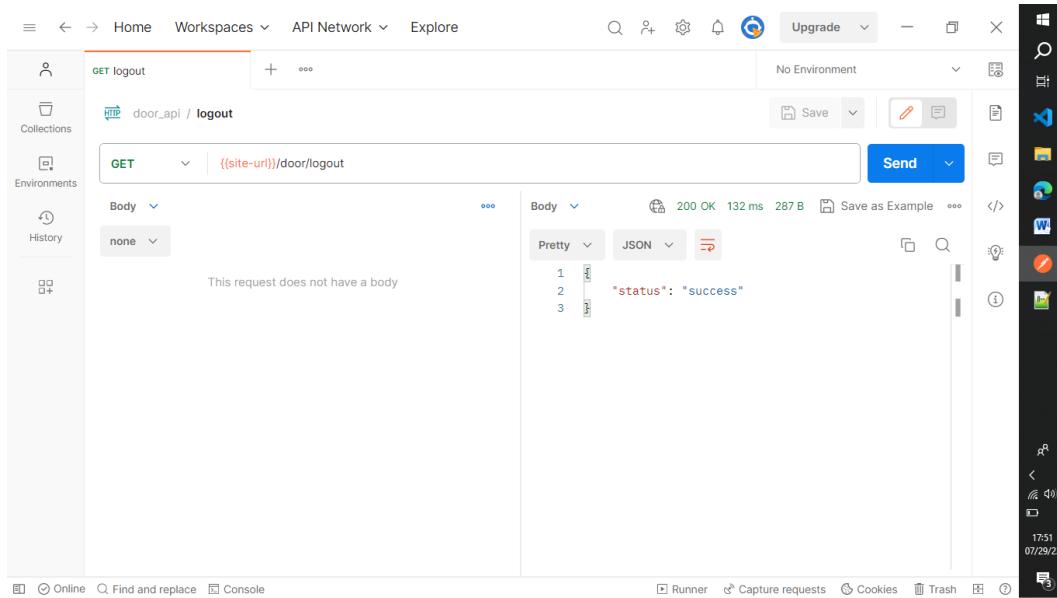
1 "status": "success",
2 "data": {
3     "office_id": "bbdba8cf-3dc0-4415-bb3b-756d16a380d6",
4     "door_id": "bf2e7662-fc30-42b8-b6ab-7513a1d34b2c",
5     "door_name": "Lab Teknologi Informatika"
6 },
7 "token": "145|0nXnY1inGyY8mpHwXsBA7bpvmJ6GZyAfjUs2Qq37"
8
9
  
```

Gambar 4.19 Hasil API Door Login

Gambar 4.19 di atas merupakan hasil dari API *door login* yang dikirimkan melalui *endpoint* “/door/login”. Setiap pintu akan mengirimkan *device_name* sebagai *username* dan *device_pass* sebagai *password* untuk mendapatkan token akses. Jika *login* berhasil maka API akan memberikan respon berupa token akses disertai dengan data pendukung seperti identitas gedung dan identitas pintu. Namun jika gagal maka API akan memberikan respon *login* gagal.

4.1.15 API Door Logout

API *door logout* digunakan untuk menghapus semua token yang dimiliki oleh pintu tersebut. Sesekali pintu akan melakukan permintaan *logout* untuk menghapus semua token yang disimpan didalam *database*. Hasil dari API *door logout* dapat dilihat pada Gambar 4.20 di bawah.

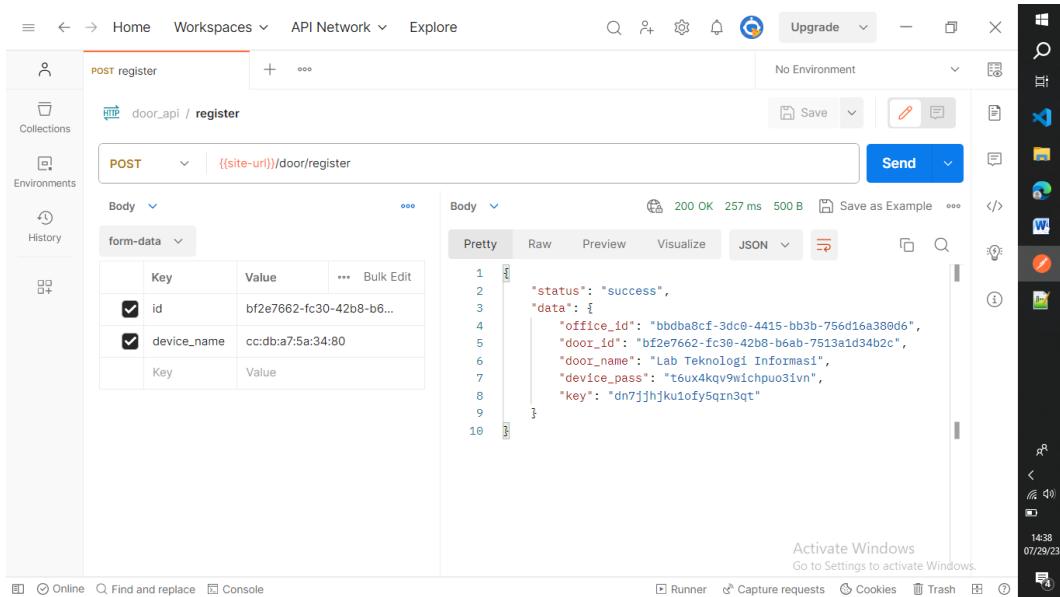


Gambar 4.20 Hasil API *Door Logout*

Gambar 4.20 di atas merupakan hasil dari API *door logout* yang dikirimkan melalui *endpoint* “/door/logout”. Pintu akan mengirimkan permintaan dengan melampirkan token akses yang digunakan untuk autentikasi, jika autentikasi berhasil maka API akan memberikan respon berhasil atau sukses.

4.1.16 API Door Register

API *door register* digunakan oleh perangkat kunci pintu untuk mendaftarkan perangkat penguncian ke *server* sehingga perangkat tersebut dapat dikenali dan dapat diatur kinerjanya oleh *server*. Pada proses pendaftaran *server* akan memperbarui data pintu pada *database*, pada proses pendaftaran *server* juga akan memberikan *password* acak yang digunakan oleh perangkat penguncian untuk melakukan proses autentikasi login. Hasil dari API *door register* dapat dilihat pada Gambar 4.21 di bawah.

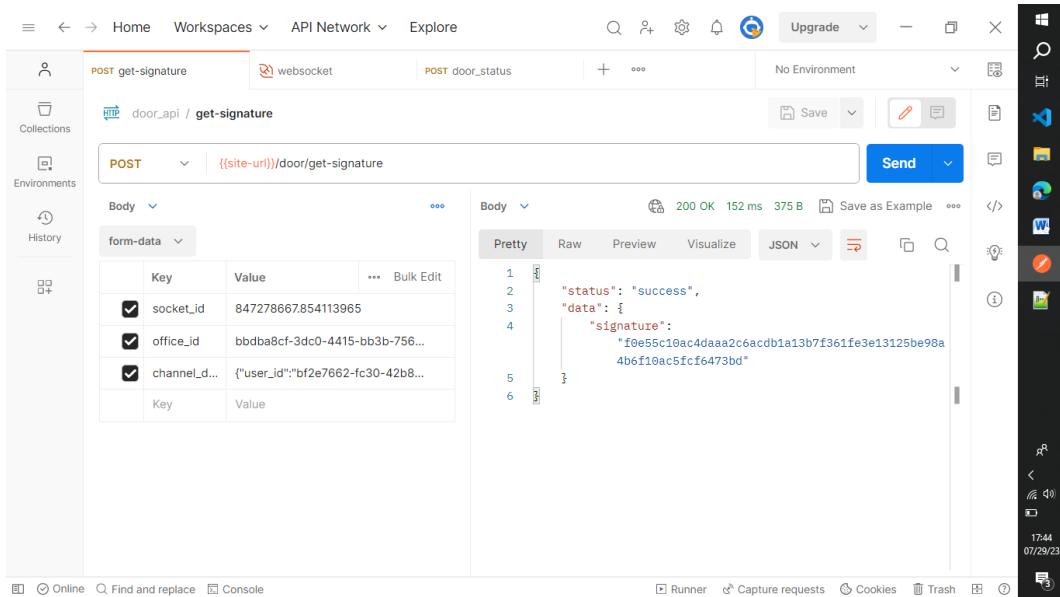


Gambar 4.21 Hasil API Door Register

Gambar 4.21 di atas merupakan hasil dari API *door register* yang dikirimkan melalui *endpoint* “/door/register”. Perangkat kunci pintu akan mengirimkan identitas unik yang nantinya akan digunakan sebagai nama perangkat yang dikenali oleh server. Jika proses pendaftaran berhasil maka API akan memberikan respon sukses dan mengirimkan data seperti identitas pintu, identitas gedung dan juga *password* acak untuk melakukan proses login. Dengan menggunakan *password* acak maka *password* tersebut hanya diketahui oleh *server* dan perangkat penguncian sehingga akan meningkatkan keamanan sistem.

4.1.17 API Door Signature

API *door signature* digunakan oleh perangkat kunci pintu untuk mendapatkan kunci websocket. Komunikasi antara *server* ke perangkat kunci pintu dilakukan menggunakan websocket, dengan menggunakan websocket maka *server* dapat langsung memberikan perintah ke perangkat kunci pintu tanpa harus menunggu *polling* yang dilakukan oleh perangkat kunci pintu. Untuk mengamankan jalur komunikasi websocket ini menggunakan kunci *signature* yang harus dikirimkan oleh perangkat kunci untuk bisa terhubung dan menerima perintah melalui koneksi websokcet. Hasil dari API door signature dapat dilihat pada Gambar 4.22 di bawah.

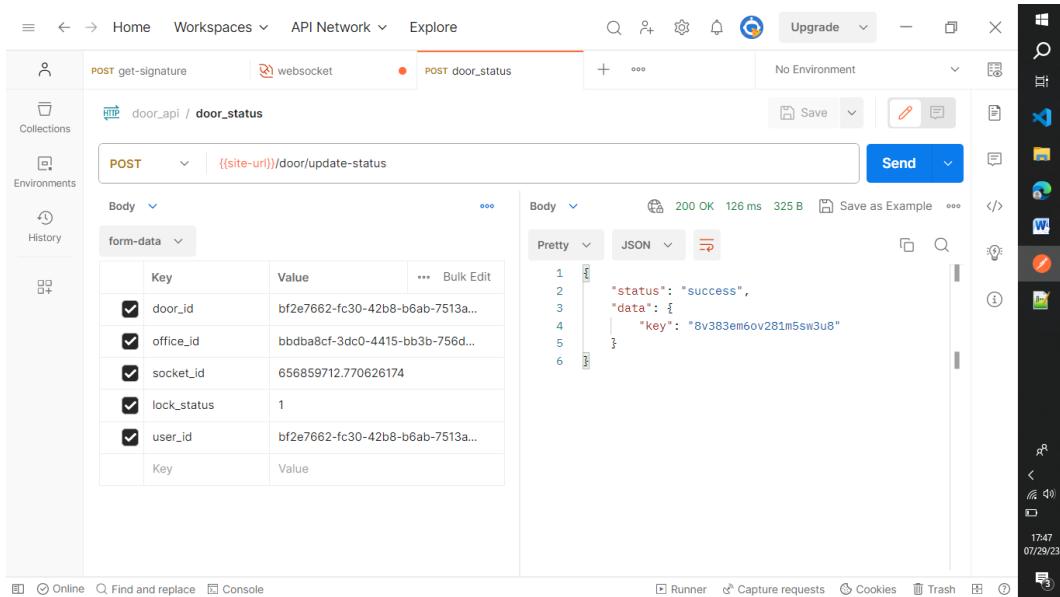


Gambar 4.22 Hasil API Door Signature

Gambar 4.22 di atas merupakan hasil dari API *door signature* yang dikirimkan melalui *endpoint* “/api/signature”. Perangkat kunci pintu akan mengirimkan identitas *socket* yang sudah terjalin disertai dengan data pendukung seperti identitas gedung, identitas pintu dan tentunya token akses, kemudian *server* akan membuat kunci *signature* yang dapat digunakan untuk terhubung ke *channel* websocket. Dengan adanya proses autentikasi dengan menggunakan *signature* maka jalur komunikasi websocket akan lebih aman dikarenakan hanya pengguna yang memiliki kode *signature* yang sesuai yang dapat men-*subscribe* ke *channel* untuk mendapatkan data pada jalur websocket.

4.1.18 API Door Update Status

API *door update* status digunakan oleh perangkat kunci pintu untuk memperbarui status penguncian pintu, jika ada perubahan status pintu seperti terkunci atau terbuka maka perangkat kunci pintu akan mengirimkan status tersebut ke *server*. Proses *update* status ini akan memastikan adanya perubahan kondisi pada pintu sehingga jika ada perintah yang dikirimkan ke perangkat kunci pintu maka perangkat kunci pintu hanya akan memberikan respon jika perubahan benar-benar terjadi dan memastikan bahwa perintah sudah benar-benar dikerjakan. Hasil dari API update status dapat dilihat pada Gambar 4.23 di bawah.



Gambar 4.23 Hasil API Door Update Status

Gambar 4.23 di atas merupakan hasil dari API *update* status yang dikirimkan melalui *endpoint* “/door/update-status”. Perangkat kunci pintu akan mengirimkan data pendukung seperti identitas pintu, identitas gedung, identitas socket, status penguncian dan identitas aktor yang melakukan kegiatan tersebut. Jika data yang dikirimkan sesuai maka server akan mengembalikan respon status sukses yang mengirimkan kunci pintu baru. Setiap melakukan update status maka *server* akan memberikan kunci pintu baru sehingga kunci pada setiap pintu bersifat dinamis, hal ini dimaksudkan untuk memperkuat keamanan sistem penguncian dengan menggunakan kode kunci pintu yang berubah ubah untuk melakukan autentikasi.

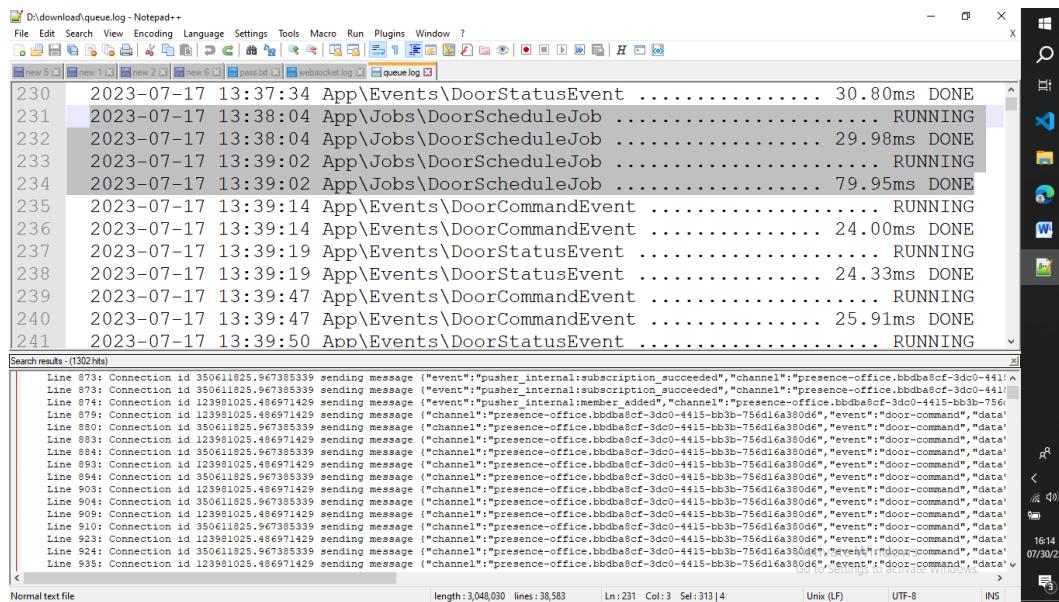
4.1.19 API Door Alert

API *door alert* digunakan oleh perangkat kunci pintu untuk mengirimkan status peringatan ke *server*, peringatan yang dikirmkan dapat berupa pintu yang terbuka tanpa autentikasi yang sah atau juga pintu dalam kondisi terbuka sehingga menghalangi perintah penguncian untuk dijalankan. Hasil dari API door alert dapat dilihat pada Gambar 4.24 di bawah. Dengan adanya sistem peringatan ini maka sistem dapat dengan cepat kondisi yang tidak normal yang terjadi pada pintu-pintu yang sudah terpasang sehingga informasi tersebut dapat tersampaikan dengan cepat untuk sedera ditindaklanjuti lebih dalam.

Gambar 4.25 di atas merupakan aktifitas dari peringatan pintu pada koneksi websocket. Setiap ada peringatan yang dikirmkan oleh perangkat kunci pintu maka server akan mengirimkan peringatan ke operator baik website maupun aplikasi mobile melalui koneksi websocket sehingga perngatan tersebut dapat diterima oleh operator secara *realtime*.

4.1.20 Pengecekan Jadwal

Pada sistem penguncian pintu gedung terdapat fitur penjadwalan yang memungkinkan operator untuk mengatur jadwal membuka atau mengunci pintu secara otomatis pada rentang waktu tertentu. Proses pengecekan jadwal akan dilakukan secara otomatis oleh *server*, jika temukan jadwal yang harus dilaksanakan maka server akan mengirimkan perintah melalui koneksi websosket seperti yang terlihat pada Gambar 4.26 di bawah.



```

D:\download\queue.log - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
New 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 310 queue.log

230 2023-07-17 13:37:34 App\Events\DoorStatusEvent ..... 30.80ms DONE
231 2023-07-17 13:38:04 App\Jobs\DoorScheduleJob ..... RUNNING
232 2023-07-17 13:38:04 App\Jobs\DoorScheduleJob ..... 29.98ms DONE
233 2023-07-17 13:39:02 App\Jobs\DoorScheduleJob ..... RUNNING
234 2023-07-17 13:39:02 App\Jobs\DoorScheduleJob ..... 79.95ms DONE
235 2023-07-17 13:39:14 App\Events\DoorCommandEvent ..... RUNNING
236 2023-07-17 13:39:14 App\Events\DoorCommandEvent ..... 24.00ms DONE
237 2023-07-17 13:39:19 App\Events\DoorStatusEvent ..... RUNNING
238 2023-07-17 13:39:19 App\Events\DoorStatusEvent ..... 24.33ms DONE
239 2023-07-17 13:39:47 App\Events\DoorCommandEvent ..... RUNNING
240 2023-07-17 13:39:47 App\Events\DoorCommandEvent ..... 25.91ms DONE
241 2023-07-17 13:39:50 App\Events\DoorStatusEvent ..... RUNNING

Search results - (1302 hits)
Line 873: Connection id 350611825.967385339 sending message {"event":"pusher_internal:subscription_succeeded","channel":"presence-office.bbdbba8cf-3dc0-4411-9673:967385339"
Line 873: Connection id 350611825.967385339 sending message {"event":"pusher_internal:subscription_succeeded","channel":"presence-office.bbdbba8cf-3dc0-4411-9673:967385339"
Line 874: Connection id 123991025.486971429 sending message {"event":"pusher_internal:member_added","channel":"presence-office.bbdbba8cf-3dc0-4411-bb3b-756d16a380d6"
Line 875: Connection id 123991025.486971429 sending message {"event":"pusher_internal:member_removed","channel":"presence-office.bbdbba8cf-3dc0-4411-bb3b-756d16a380d6"
Line 880: Connection id 350611825.967385339 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 883: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 884: Connection id 350611825.967385339 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 893: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 894: Connection id 350611825.967385339 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 904: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 909: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 910: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 933: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 944: Connection id 350611825.967385339 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"
Line 955: Connection id 123991025.486971429 sending message {"channel":"presence-office.bbdbba8cf-3dc0-4415-bb3b-756d16a380d6","event":"door-command","data"

1614
07/07/23
    
```

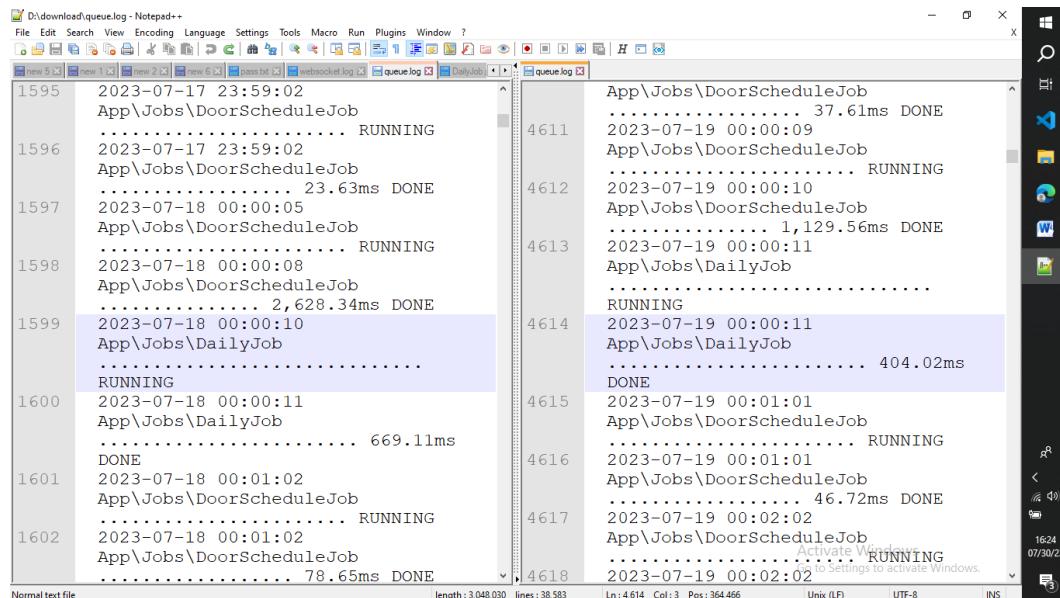
Gambar 4.26 Pengecekan Jadwal

Pada Gambar 4.26 terlihat bahwa pengecekan jadwal menggunakan sebuah *job* dari laravel yaitu *DoorSceduleJob* yang akan memeriksa jadwal pada database setiap 1 menit. Terlihat pada baris 231 (Gambar 4.26) pada pukul 13:38:04 server melakukan pengecekan jadwal dan proses pengecekan selanjutkan dilakukan pada pukul 13:39:02 dan terus berulang hal ini menunjukkan bahwa pengecekan jadwal benar benar dilakukan setiap 1 menit.

Pada Gambar 4.26 juga memperlihatkan bahwa saat proses pengecekan jadwal server menemukan jadwal yang harus dilaksanakan maka server akan mengirimkan perintah ke perangkat kunci pintu. Seperti yang terlihat pada baris 235 setelah ditemukan jadwal yang harus dilaksanakan maka server akan mengirimkan perintah ke perangkat kunci pintu dengan menggunakan sebuah event yaitu *DoorCommandEvent* yang akan disiarkan melalui *channel* websoket.

4.1.21 Atur Ulang Jadwal

Setiap jadwal yang sudah dilaksanakan akan dihapus dari database, namun jika jadwal tersebut berulang maka akan diatur ulang sehingga dapat dijalankan kembali. Proses pengeturan jadwal dilaksanakan setiap 24 jam sekali seperti yang terlihat pada Gambar 4.27 di bawah.



```
D:\download\queue.log - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
New 1 New 2 New 3 New 4 New 5 New 6 parse log queue log DailyJob queue.log

1595 2023-07-17 23:59:02 App\Jobs\DoorScheduleJob ..... RUNNING
1596 2023-07-17 23:59:02 App\Jobs\DoorScheduleJob ..... 23.63ms DONE
1597 2023-07-18 00:00:05 App\Jobs\DoorScheduleJob ..... RUNNING
1598 2023-07-18 00:00:08 App\Jobs\DoorScheduleJob ..... 2,628.34ms DONE
1599 2023-07-18 00:00:10 App\Jobs\DailyJob ..... RUNNING
1600 2023-07-18 00:00:11 App\Jobs\DailyJob ..... 669.11ms DONE
1601 2023-07-18 00:01:02 App\Jobs\DoorScheduleJob ..... RUNNING
1602 2023-07-18 00:01:02 App\Jobs\DoorScheduleJob ..... 78.65ms DONE
4611 2023-07-19 00:00:09 App\Jobs\DoorScheduleJob ..... 37.61ms DONE
4612 2023-07-19 00:00:10 App\Jobs\DoorScheduleJob ..... RUNNING
4613 2023-07-19 00:00:11 App\Jobs\DoorScheduleJob ..... 1,129.56ms DONE
4614 2023-07-19 00:00:11 App\Jobs\DailyJob ..... RUNNING
4615 2023-07-19 00:01:01 App\Jobs\DoorScheduleJob ..... 404.02ms DONE
4616 2023-07-19 00:01:01 App\Jobs\DoorScheduleJob ..... RUNNING
4617 2023-07-19 00:02:02 App\Jobs\DoorScheduleJob ..... 46.72ms DONE
4618 2023-07-19 00:02:02 App\Jobs\DoorScheduleJob ..... RUNNING

```

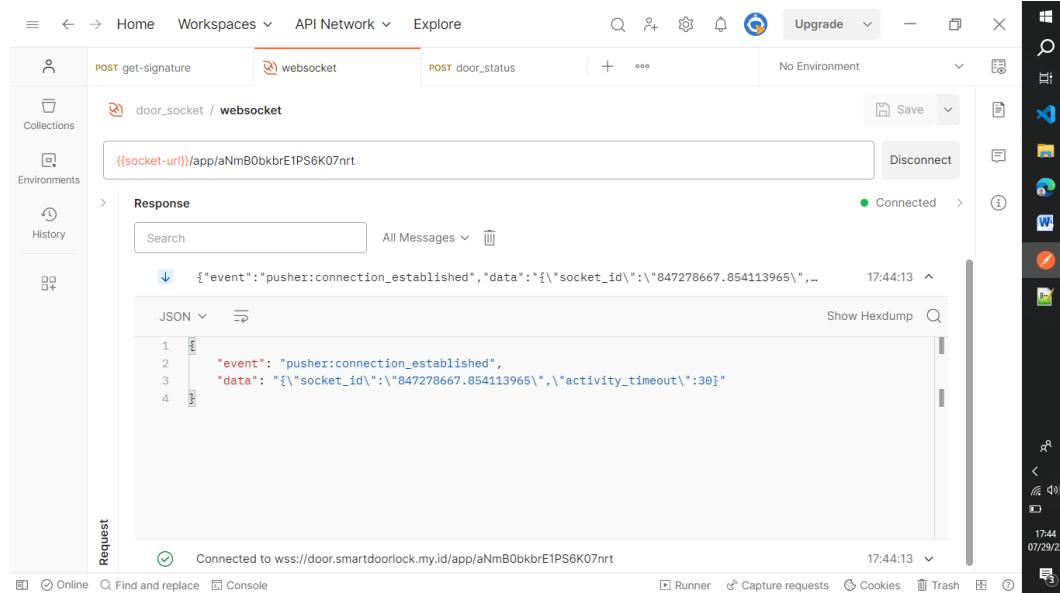
Normal text file length: 3,048,030 lines: 38,583 Ln: 4,614 Col: 3 Pos: 364,466 Unix (LF) UTF-8 INS

Gambar 4.27 Atur Ulang Jadwal

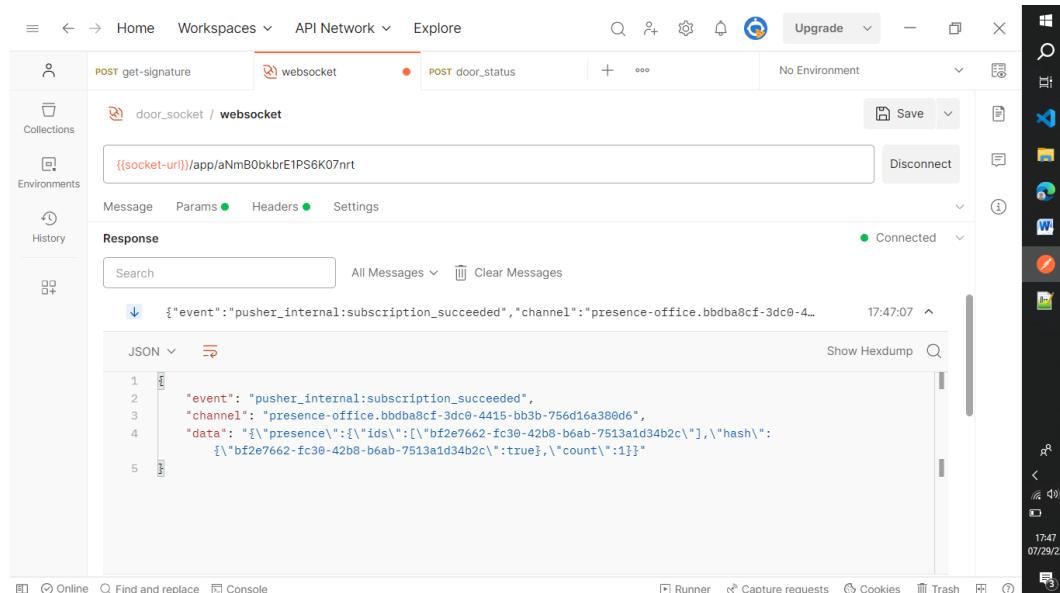
Pada Gambar 4.27 terlihat bahwa pengaturan jadwal dilakukan menggunakan sebuah job laravel yaitu DailyJob. Job ini digunakan untuk menghapus jadwal yang sudah dilaksanakan atau mengatur ulang jadwal untuk kembali dilaksanakan pada hari tersebut. Pada Gambar 4.27 juga terlihat pada baris 1599 dan baris 4614 memperlihatkan proses pengeturan jadwal dilaksanakan setiap 24 jam dimulai pada tanggal 18 Juli 2023 dan prose pengaturan selanjutnya dilaksanakan pada tanggal 19 Juli 2023.

4.1.22 Komunikasi Websocket

Komunikasi websoket dilakukan dengan melakukan permintaan koneksi ke endpoint websoket yang telah disediakan oleh server seperti yang terlihat pada Gambar 4.28.



Gambar 4.28 Koneksi Websocket Melalui *Endpoint*



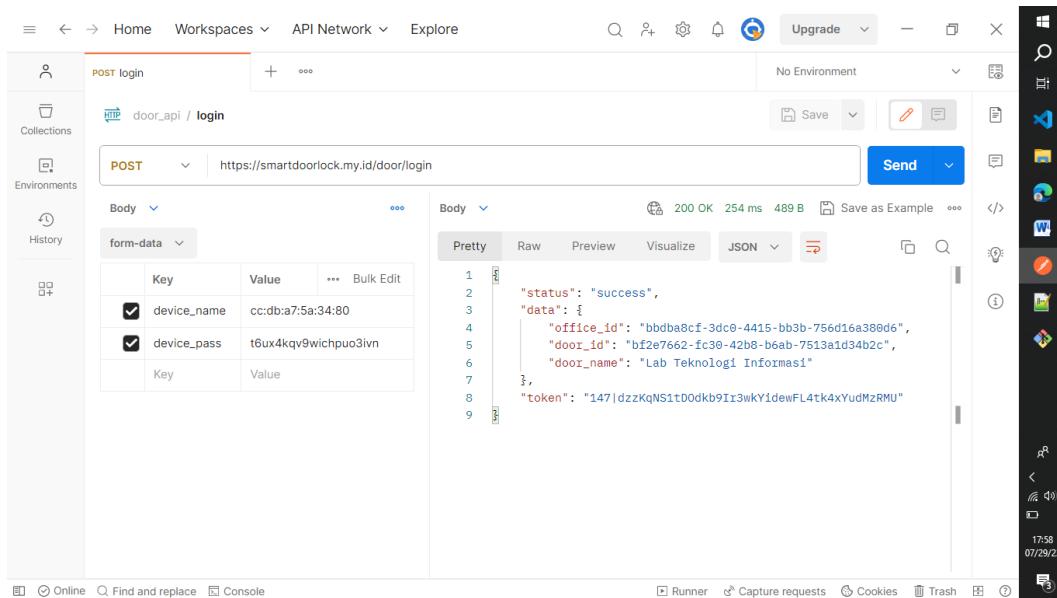
Gambar 4.29 Proses *Subscribe* ke *Channel Websocket*

Pada Gambar 4.28 di atas terlihat proses koneksi websoket dimulai dengan mengirimkan permintaan ke endpoint websocket. Jika koneksi berhasil maka client akan menerima pesan berupa *event:connection_establised* yang

menandakan koneksi websoket sudah berhasil. Sampai tahap ini *client* tidak akan menerima data perintah yang dikirimkan melalui websocket. *Client* harus men-*subscribe* channel untuk bisa mendapatkan data perintah tersebut. Pada Gambar 4.29 diperlihatkan proses *subscribe* ke *channel* dengan mengirimkan pesan *subscriber* melalui websoket dengan melampirkan kode *signature*, jika kode *signature* sesuai maka *server* akan mengirimkan pesan *pusher_internal:subscription_succeeded* yang menandakan proses *subscribe* berhasil.

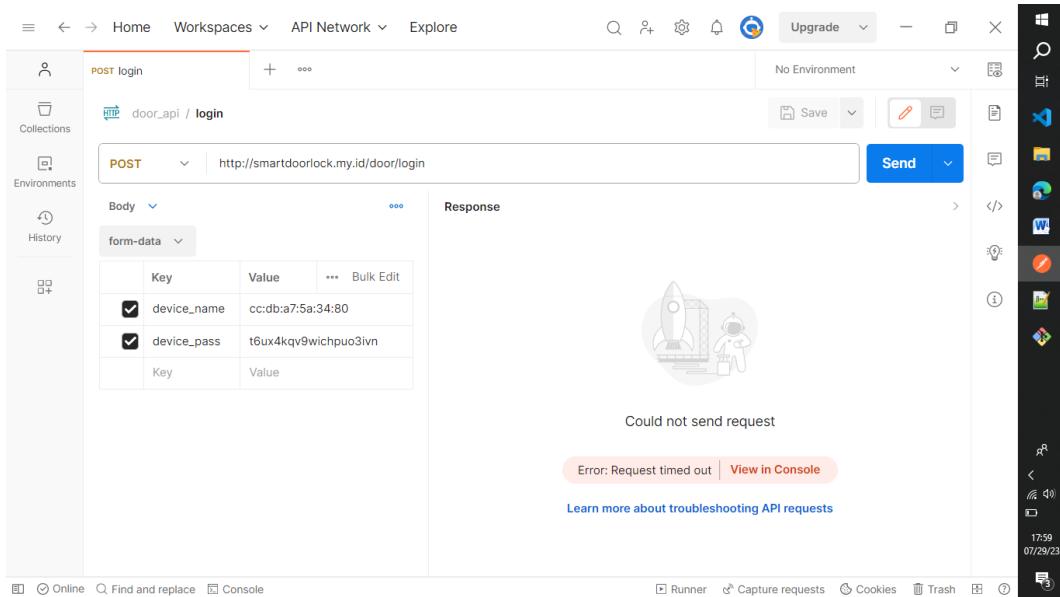
4.1.23 Koneksi HTTPS

Setiap pertukaran data pada sistem penguncian pintu gedung ini akan menggunakan koneksi HTTPS yang merupakan koneksi yang aman dengan enkripsi SSL, hal ini untuk menjamin keamanan proses transfer data. Setiap permintaan yang masuk hanya diijinkan melalui port 443 atau koneksi HTTPS selain melalui port tersebut maka permintaan akan ditolak.



Gambar 4.30 Request Menggunakan HTTPS

Pada Gambar 4.30 terlihat request pada berhasil diterima oleh server dan server memberikan respon status terhadap permintaan yang dikirimkan. Dengan mengirimkan permintaan ke <https://smartdoorlock.my.id/api/login> berarti kita melakukan request menggunakan HTTPS sehingga request tersebut ijinkan oleh server.



Gambar 4.31 Request Menggunakan HTTP

Pada Gambar 4.31 terlihat bahwa permintaan dikirimkan menggunakan protokol HTTP, dengan menggunakan HTTP maka *server* akan menolak permintaan tersebut dan tidak akan ditanggapi oleh *server* sehingga permintaan tersebut akan *timeout* karena *server* tidak memberikan respon terhadap permintaan tersebut.

4.2 Pengujian

Pada tahap ini dilakukan proses pengujian pada sistem yang telah dibuat. Pengujian yang dilakukan meliputi pengujian terhadap kebutuhan fungsional maupun kebutuhan nonfungsional. Pengujian kebutuhan fungsional berfokus pada ketersediaan fitur-fitur yang telah dibuat serta kinerja dari fitur tersebut apakah sesuai atau tidak. Pengujian non-fungsional berkokus pada keandalan sistem yang telah dibuat seperti waktu respon dan tingkat keamanan.

4.3 Pengujian Fungsional

Pada pengujian fungsional berfokus pada fungsi dari fitur-fitur yang ada pada sistem. Pengujian fungsional dilakukan menggunakan metode *blackbox* dengan bantuan postman sebagai alat pengujian. Pengujian *blackbox* merupakan sebuah metode pengujian perangkat lunak yang dilakukan tanpa memperhatikan

struktur kode program didalamnya. Tabel 4.1 menjelaskan beberapa fitur hasil implementasi dari kebutuhan fungsional pada sistem penguncian pintu gedung ini.

Tabel 4.1 Hasil Pengujian Fungsional

No	Kebutuhan	Hasil
1	Autentikasi <i>login</i> dan <i>logout</i>	Tersedia
2	Ganti <i>password</i>	Tersedia
3	Reset <i>password</i>	Tersedia
4	Verifikasi email	Tersedia
5	Ganti profil	Tersedia
6	Lihat daftar akses	Tersedia
7	Verifikasi akses	Tersedia
8	Lihat riwayat aktifitas	Tersedia
9	Lihat daftar pintu	Tersedia
10	Membuka pintu jarak jauh	Tersedia
11	Register pintu baru	Tersedia
12	Koneksi websocket untuk pintu	Tersedia
13	Mendapatkan <i>signature</i>	Tersedia
14	Update status pintu	Tersedia
15	Peringatan pintu	Tersedia

4.3.1 Fungsi *Login*

Pengujian pada fungsi *login* dilaksanakan untuk memeriksa kinerja dari fungsi *login*, *login* dikatakan berhasil jika client mendapatkan respon *success* dari server disertai dengan dikirimkannya data *client* dan token akses. Hasil dari pengujian fungsi login dapat dilihat pada Tabel 4.2.

Tabel 4.2 Hasil Pengujian Fungsi *Login*

No	Nama	Bentuk Pengujian	Respon	Hasil
1	<i>Login</i> dengan data benar	Melakukan <i>login</i> menggunakan email dan <i>password</i> yang sesuai	<i>success</i>	Berhasil

Lanjutan Tabel 4.3 Hasil Pengujian Fungsi *Login*

No	Nama	Bentuk Pengujian	Respon	Hasil
2	<i>Login</i> dengan data salah	Melakukan <i>login</i> dengan menggunakan email atau <i>password</i> yang salah	<i>failed</i>	Gagal
3	<i>Login</i> dengan data kurang	Melakukan <i>login</i> dengan menggunakan email saja atau <i>password</i> saja	<i>missing_parameter</i>	Gagal
4	<i>Login</i> dengan data tidak terdaftar	Melakukan login dengan menggunakan email yang belum terdaftar	<i>missing_parameter</i>	Gagal
5	<i>Login</i> dengan format tidak sesuai	Melakukan <i>login</i> dengan menggunakan <i>username</i> bukan email	<i>missing_parameter</i>	Gagal
6	<i>Login</i> dengan email belum berlum terverifikasi	Melakukan <i>login</i> dengan menggunakan email yang berlum terverifikasi	<i>email_unverified</i>	Gagal

Dapat dilihat pada Tabel 4.2 proses *login* akan berhasil jika menggunakan email dan *password* yang sesuai, proses *login* juga memastikan semua parameter yang digunakan pada autentikasi tersedia dan juga sesuai. Pada proses pengujian menggunakan email yang belum terverifikasi *login* akan tertahan dengan status *email_unverified* dan menunggu *client* untuk melakukan verifikasi email.

4.3.2 Fungsi *Logout*

Pengujian pada fungsi *logout* dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. *Logout* dikatakan berhasil jika *client* menerima respon *success* dari *server*. Hasil dari pengujian fungsi *logout* dapat dilihat pada Tabel 4.3.

Tabel 4.4 Hasil Pengujian Fungsi *Logout*

No	Nama	Bentuk Pengujian	Respon	Hasil
1	<i>Logout</i> dengan token	Melakukan <i>logout</i> menggunakan token yang sesuai	<i>success</i>	Berhasil
2	<i>Logout</i> tanpa token	Melakukan <i>logout</i> tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
3	<i>Login</i> dengan token salah	Melakukan <i>logout</i> dengan menggunakan token yang sudah terhapus	<i>Unauthenticated</i>	Gagal

Dapat dilihat pada Tabel 4.3 proses *logout* hanya berhasil jika menggunakan token yang sesuai, jika *client* melakukan *logout* tanpa menggunakan token atau menggunakan token yang sudah terhapus maka *logout* akan gagal dengan respon *unauthenticated* atau tidak terautentikasi.

4.3.3 Fungsi Verifikasi Email

Pengujian pada fungsi verifikasi email dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Verifikasi email dikatakan berhasil jika *client* mendapatkan respon *success* dari server. Hasil dari pengujian fungsi verifikasi email dapat dilihat pada Tabel 4.4.

Tabel 4.5 Hasil Pengujian Fungsi Verifikasi Email

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Verifikasi email tanpa token	Melakukan verifikasi email tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
2	Verifikasi email tidak sesuai	Melakukan verifikasi email menggunakan kode OTP yang salah	<i>otp_not_match</i>	Gagal
3	Verifikasi email kadaluarsa	Melakukan verifikasi email menggunakan kode OTP yang sudah kadaluarsa	<i>otp_expired</i>	Gagal

Lanjutan Tabel 4.6 Hasil Pengujian Fungsi Verifikasi Email

No	Nama	Bentuk Pengujian	Respon	Hasil
4	Verifikasi email sesuai	Melakukan verifikasi email menggunakan kode OTP yang sesuai	<i>success</i>	Berhasil
5	Verifikasi email token salah	Melakukan verifikasi email menggunakan token yang sudah terhapus	<i>Unauthenticated</i>	Gagal

Dapat dilihat pada Tabel 4.4 proses verifikasi email hanya berhasil jika client mengirimkan kode OTP yang sesuai disertai dengan token yang sesuai. Jika proses verifikasi email menggunakan kode OTP yang salah atau sudah kadaluarsa maka proses verifikasi email akan gagal.

4.3.4 Fungsi Ganti Password

Pengujian pada fungsi ganti password dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses ganti password dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian ganti password dapat dilihat pada Tabel 4.5.

Tabel 4.7 Hasil Pengujian Fungsi Ganti Password

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Ganti password tanpa token	Melakukan penggantian password tanpa token menggunakan token	<i>Unauthenticated</i>	Gagal
2	Ganti password dengan token salah	Melakukan penggantian password menggunakan token yang sudah terhapus	<i>Unauthenticated</i>	Gagal
3	Ganti password sesuai	Melakukan penggantian password sesuai	<i>success</i>	Berhasil
4	Ganti password minimal	Melakukan penggantian password dengan 3 karakter	<i>missing_parameter</i>	Gagal

Lanjutan Tabel 4.8 Hasil Pengujian Fungsi Ganti Password

No	Nama	Bentuk Pengujian	Respon	Hasil
5	Ganti password maksimal	Melakukan penggantian password dengan 100 karakter	<i>missing_parameter</i>	Gagal
6	Ganti password konfirmasi salah	Melakukan penggantian password dengan konfirmasi password berbeda	<i>missing_parameter</i>	Gagal

Pada Tabel 4.5 terlihat proses ganti password berhasil jika *client* mengirimkan password dan konfirmasi password sesuai, *client* juga harus mengirimkan token yang sesuai juga. Jika salah satu parameter tidak terpenuhi maka proses ganti password akan gagal.

4.3.5 Fungsi Reset Password

Pengujian pada fungsi reset password dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses reset password dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian reset password dapat dilihat pada Tabel 4.6.

Tabel 4.9 Hasil Pengujian Fungsi Reset Password

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Reset password email sesuai	Melakukan reset password menggunakan email dengan format sesuai dan terdaftar	<i>success</i>	Berhasil
2	Reset password email tidak sesuai	Melakukan reset password menggunakan email yang belum terdaftar	<i>failed</i>	Gagal
3	Reset password format email salah	Melakukan reset password menggunakan email dengan format salah	<i>missing_parameter</i>	Gagal

Pada Tabel 4.6 terlihat proses reset password berhasil jika *client* mengirimkan alamat email yang sesuai, Jika email tidak sesuai atau belum terdaftar maka reset password akan gagal.

4.3.6 Fungsi *Update Profil*

Pengujian pada fungsi update profil dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses update profil dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian update profil dapat dilihat pada Tabel 4.7.

Tabel 4.10 Hasil Pengujian Fungsi *Update Profil*

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Update nama minimal	Melakukan update profil menggunakan nama 3 karakter	<i>missing_parameter</i>	Gagal
2	Update nama sesuai	Melakukan update profil menggunakan nama sesuai	<i>success</i>	Berhasil
3	Update email sesuai	Melakukan update profil menggunakan email yang sesuai	<i>success</i>	Berhasil
4	Update email format salah	Melakukan update profil menggunakan email dengan format tidak sesuai	<i>missing_parameter</i>	Gagal
5	Update jenis kelamin sesuai	Melakukan update jenis kelamin dengan format sesuai	<i>success</i>	Berhasil
6	Update nomor hp sesuai	Melakukan update nomor hp dengan format sesuai	<i>success</i>	Berhasil
7	Update nomor hp tidak sesuai	Melakukan update nomor hp dengan format salah (digit angka kurang/lebih)	<i>missing_parameter</i>	Gagal

Pada Tabel 4.7 terlihat proses update profil akan berhasil jika semua data yang dikirimkan sesuai dengan format sehingga mendapatkan respon success, jika ada salah satu data yang tidak sesuai format misalnya format email tidak sesuai atau nomor hp kurang maka update profil akan gagal.

4.3.7 Fungsi *Update Avatar*

Pengujian pada fungsi update avatar dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses update avatar dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian reset password dapat dilihat pada Tabel 4.8.

Tabel 4.11 Hasil Pengujian Fungsi *Update Avatar*

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Update avatar format sesuai	Melakukan update gambar avatar sesuai dengan format	<i>success</i>	Berhasil
2	Update avatar file besar	Melakukan update gambar menggunakan gambar dengan ukuran lebih dari 1 MB	<i>request entity too large</i>	Gagal
3	Update avatar bukan gambar	Melakukan update avatar menggunakan file salin gambar	<i>missing_parameter</i>	Gagal

Pada Tabel 4.8 terlihat proses update avatar hanya berhasil jika file yang dikirim adalah gambar dengan ukuran kurang dari 1 MB, jika data yang dikirim bukan merupakan gambar atau ukuran gambar lebih dari 1 mB maka proses update avatar akan gagal.

4.3.8 Fungsi Lihat Akses

Pengujian pada fungsi update avatar dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses update avatar dikatakan berhasil jika *client*

mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi lihat akses dapat dilihat pada Tabel 4.9.

Tabel 4.12 Hasil Pengujian Fungsi Lihat Akses

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Lihat akses dengan token	Melakukan <i>request</i> lihat akses menggunakan token yang sesuai	<i>success</i>	Berhasil
2	Lihat akses tanpa token	Melakukan <i>request</i> lihat akses tanpa menggunakan token	<i>Unauthenticated</i>	Gagal

Pada Tabel 4.9 terlihat bahwa proses lihat akses akan berhasil jika permintaan dilakukan dengan menambahkan token, jika permintaan tidak menggunakan token maka permintaan akan ditolak.

4.3.9 Fungsi Lihat Pintu

Pengujian pada fungsi lihat pintu dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses lihat pintu dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi lihat pintu dapat dilihat pada Tabel 4.10.

Tabel 4.13 Hasil Pengujian Fungsi Lihat Pintu

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Lihat pintu tanpa token	Melakukan <i>request</i> lihat akses tanpa menggunakan token yang sesuai	<i>Unauthenticated</i>	Gagal
2	Lihat pintu dengan token pengguna	Melakukan <i>request</i> lihat akses menggunakan token yang dimiliki pengguna	<i>Unauthenticated</i>	Gagal
3	Lihat Pintu dengan token operator	Melakukan <i>request</i> lihat akses menggunakan token yang dimiliki operator	<i>success</i>	Berhasil

Pada Tabel 4.10 terlihat bahwa proses lihat pintu berhasil jika permintaan disertai dengan token operator, jika permintaan tidak menggunakan token atau menggunakan token yang dimiliki oleh pengguna biasa maka permintaan akan ditolak.

4.3.10 Fungsi Lihat Riwayat Akses

Pengujian pada fungsi lihat riwayat akses dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses lihat riwayat akses dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi lihat riwayat akses dapat dilihat pada Tabel 4.11.

Tabel 4.14 Hasil Pengujian Lihat Riwayat Akses

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Lihat riwayat dengan token	Melakukan <i>request</i> lihat riwayat menggunakan token yang sesuai	<i>success</i>	Berhasil
2	Lihat riwayat tanpa token	Melakukan <i>request</i> lihat riwayat tanpa token menggunakan token	<i>Unauthenticated</i>	Gagal

Pada Tabel 4.11 terlihat bahwa proses lihat riwayat akses berhasil jika permintaan disertai dengan token yang sesuai, jika permintaan tidak menggunakan token maka permintaan akan ditolak.

4.3.11 Fungsi Verifikasi Akses

Pengujian pada fungsi verifikasi akses dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses verifikasi akses dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi lihat riwayat akses dapat dilihat pada Tabel 4.12.

Tabel 4.15 Hasil Pengujian Fungsi Verifikasi Akses

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Verifikasi akses tanpa token	Melakukan <i>request</i> lihat verifikasi akses tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
2	Verifikasi akses dengan token	Melakukan <i>request</i> verifikasi akses menggunakan token yang sesuai	<i>success</i>	Berhasil
3	Verifikasi akses dengan id pintu salah	Melakukan <i>request</i> verifikasi akses menggunakan identitas pintu yang salah	<i>no_data</i>	Gagal

Pada Tabel 4.12 terlihat bahwa proses verifikasi akses berhasil jika permintaan disertai dengan token dan identitas pintu sesuai, jika identitas pintu tidak sesuai maka proses verifikasi akses akan gagal karena pintu tidak ditemukan. Jika permintaan tidak disertai dengan token maka permintaan tersebut akan ditolak.

4.3.12 Fungsi *Signature*

Pengujian pada fungsi *signature* dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses pembuatan *signature* dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi buat *signature* dapat dilihat pada Tabel 4.13.

Tabel 4.16 Hasil Pengujian Fungsi *Signature*

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Signature tanpa token	Melakukan <i>request</i> <i>signature</i> tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
2	Signature dengan token	Melakukan <i>request</i> <i>signature</i> menggunakan token dan data lengkap	<i>success</i>	Berhasil

Lanjutan Tabel 4.17 Hasil Pengujian Fungsi Signature

No	Nama	Bentuk Pengujian	Respon	Hasil
3	Signature data kurang	Melakukan <i>request signature</i> menggunakan data yang kurang	<i>missing_parameter</i>	Gagal

Pada Tabel 4.13 terlihat bahwa proses fungsi *signature* berhasil jika permintaan yang dikirimkan disertai dengan token dan data yang dikirimkan lengkap, jika permintaan yang dikirimkan tanpa menggunakan token atau ada data yang kurang maka permintaan akan gagal.

4.3.13 Komunikasi Websocket

Pengujian komunikasi websocket dilakukan untuk mengetahui kinerja dari koneksi *subscription* pada *channel* websocket. Hasil dari pengujian websocket dapat dilihat pada Tabel 4.14.

Tabel 4.18 Hasil Pengujian Websocket

No	Nama	Bentuk Pengujian	Respon	Hasil
1	<i>Subscription dengan signature</i>	Melakukan <i>subscription</i> ke <i>channel</i> websocket menggunakan sgnature yang sesuai	<i>Subscription Succeed</i>	Berhasil
2	<i>Subscription tanpa signature</i>	Melakukan <i>subscription</i> ke <i>channel</i> websocket tanpa menggunakan sgnature	<i>Invalid Signature</i>	Gagal
3	<i>Subscription dengan signature salah</i>	Melakukan <i>subscription</i> ke <i>channel</i> websocket menggunakan sgnature yang tidak sesuai	<i>Invalid Signature</i>	Gagal

Pada Tabel 4.14 terlihat bahwa proses *subscription* pada *channel* websocket berhasil jika menggunakan kode *signature* yang sesuai, jika tidak menggunakan kode *signature* atau menggunakan kode *signature* yang salah maka proses *subscription* akan gagal.

4.3.14 Fungsi *Update* Status Pintu

Pengujian pada fungsi *update* status pintu dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses *update* status pintu dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi update status pintu dapat dilihat pada Tabel 4.15.

Tabel 4.19 Hasil Pengujian Fungsi Update Status Pintu

No	Nama	Bentuk Pengujian	Respon	Hasil
1	<i>Update</i> status tanpa token	Melakukan <i>update</i> status pintu tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
2	<i>Update</i> status dengan token	Melakukan <i>update</i> status pintu dengan menggunakan token dan data lengkap	<i>success</i>	Berhasil
3	<i>Update</i> status data tidak lengkap	Melakukan <i>update</i> status pintu dengan menggunakan data yang tidak lengkap	<i>missing_parameter</i>	Gagal
4	<i>Update</i> status id pintu salah	Melakukan <i>update</i> status pintu menggunakan data identitas pintu yang salah	<i>failed</i>	Gagal

Pada Tabel 4.15 terlihat bahwa proses *update* status pintu berhasil jika *request* dikirimkan dengan token dan data yang lengkap, jika ada data yang kurang lengkap atau tidak disertai dengan token akan *request* tersebut akan gagal.

4.3.15 Fungsi Peringatan Pintu

Pengujian pada fungsi peringatan pintu dilaksanakan untuk memeriksa kinerja dari fungsi tersebut. Proses peringatan pintu dikatakan berhasil jika *client* mendapatkan respon *success* dari *server*. hasil dari pengujian fungsi peringatan pintu dapat dilihat pada Tabel 4.16.

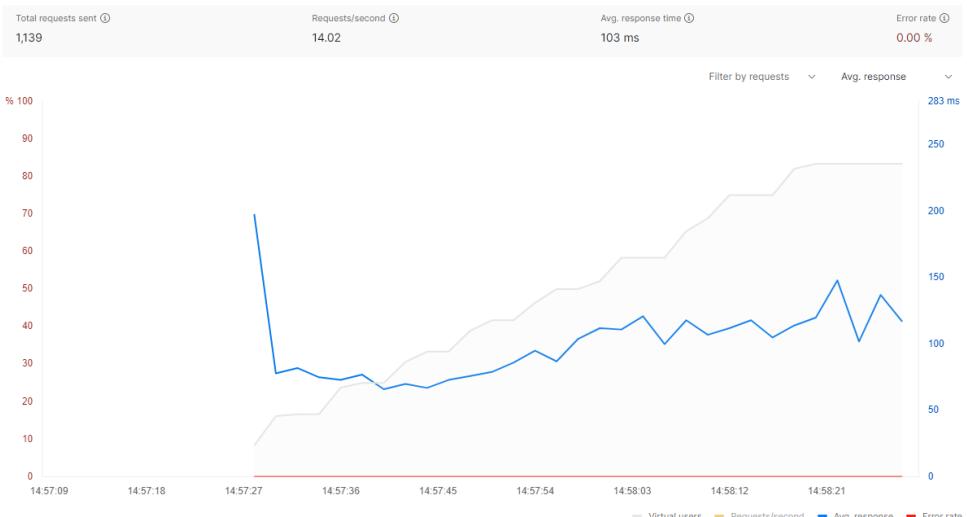
Tabel 4.20 Hasil Pengujian Fungsi Peringatan Pintu

No	Nama	Bentuk Pengujian	Respon	Hasil
1	Peringatan pintu tanpa token	Mengirim peringatan pintu tanpa menggunakan token	<i>Unauthenticated</i>	Gagal
2	Peringatan pintu sesuai	Mengirim peringatan pintu dengan menggunakan token dan data lengkap	<i>success</i>	Berhasil
3	Peringatan pintu data tidak lengkap	Mengirim peringatan pintu dengan menggunakan data yang tidak lengkap	<i>missing_parameter</i>	Gagal
4	Peringatan pintu id pintu salah	Mengirim peringatan pintu menggunakan data identitas pintu yang salah	<i>failed</i>	Gagal

Pada Tabel 4.16 terlihat bahwa proses peringatan pintu berhasil jika *request* dikirimkan dengan token dan data yang lengkap, jika ada data yang kurang lengkap atau tidak disertai dengan token akan *request* tersebut akan gagal.

4.4 Pengujian Non-Fungsional

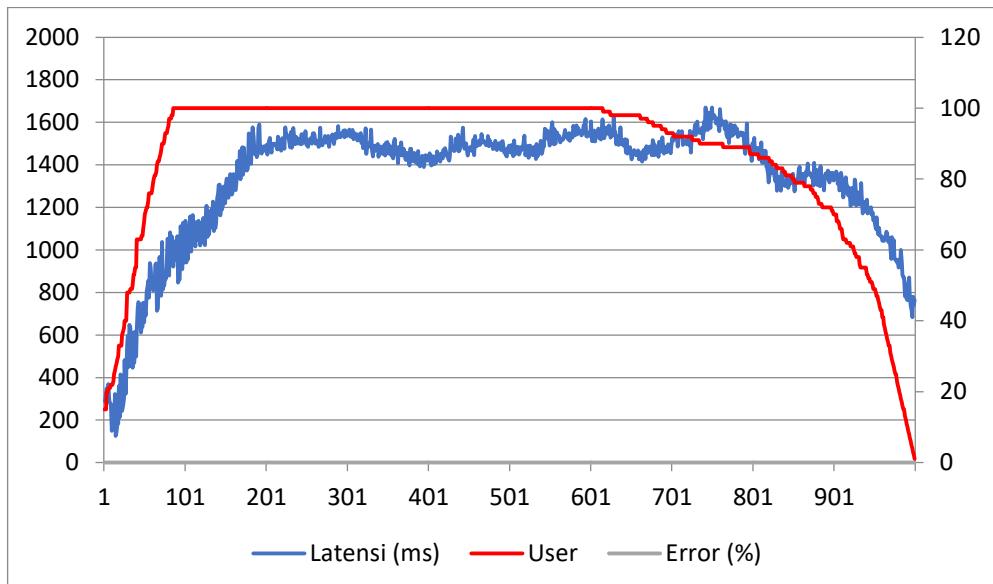
Pengujian non-fungsional berfokus pada karakteristik dari sistem yang telah dibuat. Pengujian dilakukan dengan menggunakan Postman dan Jmeter dengan cara memberikan beberapa *request* secara bersamaan.

**Gambar 4.32** Hasil Pengujian Performa API

Dapat dilihat pada Gambar 4.32, pada pengujian API secara keseluruhan dengan menggunakan 10 pengguna secara bersamaan menunjukkan waktu respon rata-rata 104 milidetik dengan waktu respon tertinggi 148 milidetik dan rasio error 0%.

4.4.1 Pengujian Performa Verifikasi Akses

Pada sistem penguncian pintu gedung ini kemungkinan pengguna melakukan permintaan akses secara bersamaan, secara teori semakin banyak pengguna yang mengirimkan *request* maka waktu respon akan semakin meningkat. Oleh karena itu dilakukan pengujian untuk melihat kemampuan sistem dalam menangani permintaan akses tersebut.

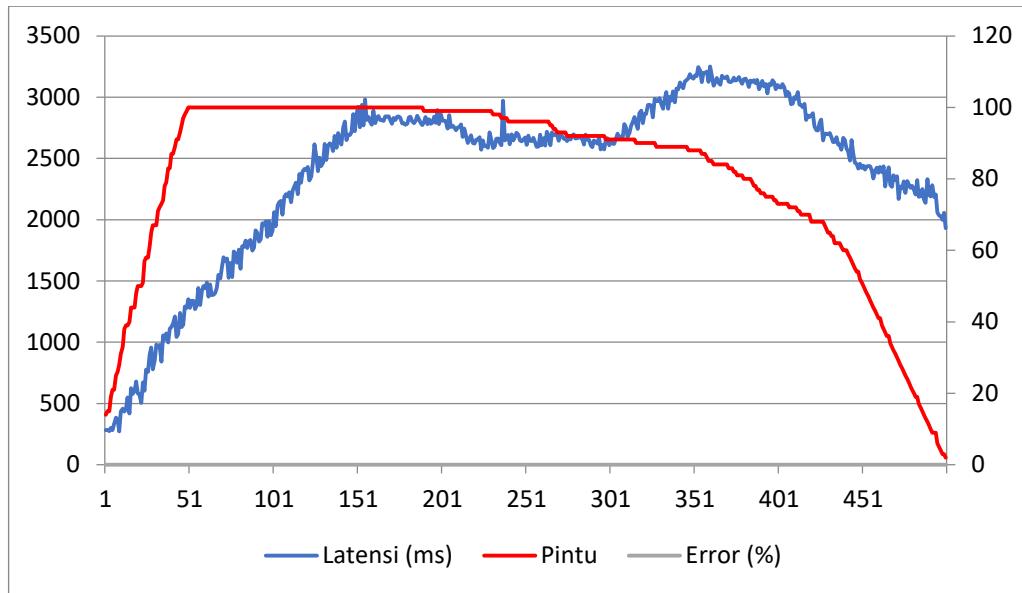


Gambar 4.33 Grafik Hasil Pengujian Performa Verifikasi Akses

Gambar 4.32 merupakan hasil pengujian fungsi verifikasi akses menggunakan Jmeter. Pengujian dilakukan dengan mensimulasikan beban permintaan verifikasi akses sebanyak 100 pengguna berbeda secara bersamaan. Dari hasil pengujian didapatkan bahwa pada puncak jumlah pengguna sistem membutuhkan waktu rata-rata 1.5 detik untuk memberikan respon ke pengguna dengan rasio error 0.0%.

4.4.2 Pengujian Performa Update Status Pintu

Setiap terjadi perubahan status pada pintu maka perangkat penguncian akan langsung mengirimkan status perubahan ke server. Dalam prosesnya dimungkinkan beberapa pintu mengirimkan status perubahan secara bersamaan sehingga dapat mempengaruhi kinerja dari sistem.

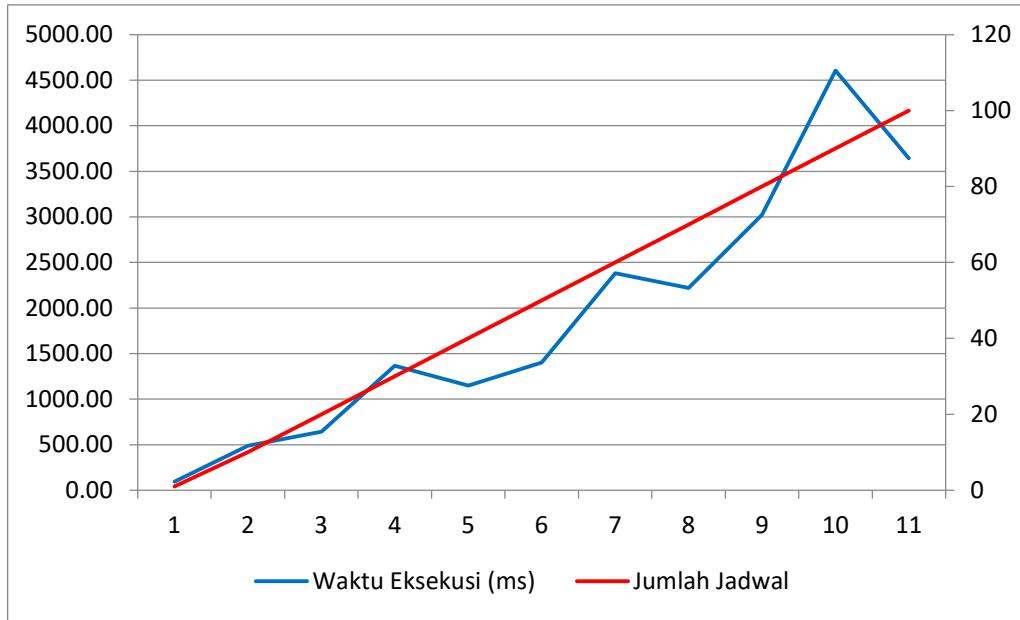


Gambar 4.34 Hasil Pengujian Performa Update Status Pintu

Gambar 4.33 merupakan hasil pengujian performa update status pintu menggunakan Jmeter. Proses pengujian mendapatkan hasil bahwa pada beban 100 pintu berbeda melakukan update status secara bersamaan maka sistem memerlukan waktu rata-rata 2.5 detik dengan rasio error 0.0%.

4.4.3 Pengujian Performa Penjadwalan

Pada proses penjadwalan maka sistem akan melakukan pemeriksaan data jadwal pada *database* setiap 1 menit, oleh karena itu metode pengecekan jadwal diharuskan selesai dilaksanakan sebelum proses pemeriksaan selanjutnya dijalankan.



Gambar 4.35 Hasil Pengujian Performa Penjadwalan

Pengujian dilakukan dengan mencatat waktu pemeriksaan untuk setiap jadwal, yaitu terdapat 11 titik pengujian dimulai dari 1 jadwal sampai 100 jadwal dengan masing-masing jadwal terdapat 20 pintu. Dapat dilihat pada Gambar 4.34 di atas, semakin besar jumlah jadwal yang ada maka waktu yang diperlukan pada proses penjadwalan akan semakin lama dimana untuk 1 jadwal memerlukan waktu 96.00 milidetik dan 100 jadwal memerlukan waktu 3641.14 milidetik. Dari hasil pengujian menggunakan 100 jadwal waktu yang diperlukan yaitu 3.6 detik dimana nilai tersebut masih di bawah dari periode pengecekan jadwal (1 menit) sehingga sistem masih dapat menangani 100 proses penjadwalan dengan aman.

4.5 Pembahasan

Pengembangan sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan akses kontrol ini merupakan bagian dari pengembangan sistem penguncian gedung berbasis IoT yang dibangun dengan tujuan untuk meningkatkan keamanan dan efisiensi pengelolaan kunci pada sebuah gedung dimana didalam satu gedung tersebut terdapat banyak ruangan. Sistem yang dibangun dapat meningkatkan keamanan dan efisiensi penguncian dengan menggunakan beberapa fitur seperti kode QR, kendali jarak jauh, peringatan penerobosan dan penjadwalan.

Perancangan *database* dan *server* pada sistem keamanan kunci pintu menggunakan metode waterfall, pengembangan dilakukan secara berurutan dimulai dari proses pengumpulan kriteria kebutuhan, implementasi sampai ke pengujian dan pengiriman. Sistem server dibangun menggunakan sistem operasi ubuntu 20.04 dengan *database* MySQL dan *backend* API Laravel. Penggunaan ubuntu sebagai *server* memberikan banyak kemudahan seperti adanya dukungan Cronjob sebagai mekanisme penjadwalan dan Nginx sebagai HTTP *server* yang ringan dan cepat. Laravel dan MySQL digunakan sebagai bagian dari *backend* dengan memanfaatkan beberapa fitur seperti autentikasi, penyiaran, penjadwalan yang disediakan oleh Laravel serta *database* relasional yang disediakan oleh MySQL. Pada tahap pengujian menggunakan alat pengujian seperti Jmeter dan Postman, sistem dapat memberikan kinerja yang diinginkan dan juga dapat menangani banyak pengguna secara bersamaan.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang telah dilakukan, dapat disimpulkan beberapa hal sebagai berikut :

1. Meskipun pada awalnya Laravel diperuntukan sebagai kerangka kerja pengembangan *website* menggunakan bahasa PHP, namun dengan berbagai fitur yang telah disediakan Laravel juga dapat digunakan sebagai kerangka kerja pengembangan *backend API*.
2. Proses autentikasi pada *backend API* dilakukan menggunakan paket *sanctum* yaitu sebuah fitur dari Laravel yang melakukan autentikasi menggunakan sebuah token akses.
3. Penggunaan MySQL pada sistem kunci pintu didasarkan pada kebutuhan tempat penyimpanan data yang terstruktur dan cepat serta dukungan *driver* koneksi *database* pada kerangka kerja Laravel.
4. Server yang dikembangkan memiliki kemampuan untuk menyiarkan pesan secara langsung menggunakan komunikasi Websocket. Pesan akan disiarkan ke setiap *channel* menggunakan sebuah *event*.
5. Untuk meningkatkan kinerja dari server maka setiap aktifitas rutin dan aktifitas yang memerlukan waktu tunggu maka akan dimasukkan ke dalam antrian untuk dijalankan di latar belakang.
6. Nginx memiliki kemampuan untuk menjalankan lebih dari 1 aplikasi dalam satu *server* seperti pada sistem kunci pintu yang dikembangkan akan menjalankan API dan Websocket secara bersamaan dalam 1 *server* Nginx dengan menggunakan blok aplikasi yang telah diatur.
7. Supervisor pada *server* digunakan untuk memantau proses yang berjalan di latar belakang pada Laravel seperti antrian dan Websocket.
8. Proses penjadwalan akan dijalankan secara otomatis oleh Cronjob dengan menjalankan Job Laravel setiap 1 menit sekali untuk memeriksa data jadwal di dalam *database*.

9. API yang telah dikembangkan secara umum dapat menangani permintaan secara bersamaan dengan waktu respon sekitar 140 milidetik dengan rasio error 0%.
10. Pada *endpoint* API yang kemungkinan besar akan diakses secara bersamaan seperti verifikasi akses memerlukan waktu respon 1.5 detik pada beban 100 pengguna sedangkan untuk *endpoint update* status pintu memerlukan waktu 2.5 detik untuk beban 100 pintu.
11. Proses penjadwalan 100 jadwal pintu memerlukan waktu eksekusi sebesar 3.6 detik sehingga proses penjadwalan dapat berjalan dengan aman dengan periode pengecekan jadwal 1 menit sekali.

5.2 Saran

Setelah dilakukan proses perancangan dan pengujian didapatkan beberapa hal yang dapat diperbaiki atau ditingkatkan, yaitu :

1. Perlu adanya perancangan lebih lanjut untuk mempercepat respon *server* terutama pada beban *request* yang berat seperti kondisi adanya banyak permintaan secara bersamaan.
2. Diperlukan perancangan lebih lanjut untuk mengembangkan komunikasi Websocket dua arah yang bisa diigunakan untuk mengirim dan menerima data secara langsung.

DAFTAR PUSTAKA

- [1] K. Y. Sun, Y. Pernando, and M. I. Safari, “Perancangan Sistem IoT pada Smart Door Lock Menggunakan Aplikasi BLYNK,” *JUTSI (Jurnal Teknol. dan Sist. Informasi)*, vol. 1, no. 3, pp. 289–296, 2021, doi: 10.33330/jutsi.v1i3.1360.
- [2] I. Hermawan, D. Arnaldy, P. Oktivasari, and D. A. Fachrudin, “Development of Intelligent Door Lock System for Room Management Using Multi Factor Authentication,” vol. 16, no. 1, pp. 1–14, 2023.
- [3] F. As *et al.*, “Design and Construction of a Smart Door Lock With an Embedded Spy-Camera,” *J. Multidiscip. Eng. Sci. Technol.*, vol. 8, no. October, pp. 2458–9403, 2021, [Online]. Available: <https://www.researchgate.net/publication/354872757>
- [4] A. Jain, V. L. Kalyani, and B. Nogiya, “RFID and GSM Based Attendance Monitoring System using door locking / unlocking system and Its Hardware Implementation,” *J. Manag. Eng. Inf. Technol.*, vol. 2, no. 3, pp. 5–10, 2015.
- [5] M. Kasyful Anwar, “Perancangan Database IoT Berbasis Cloud dengan Restful API Cloud-Based IoT Database Design with Restful API,” vol. 20, no. 2, pp. 268–279, 2021.
- [6] S. Artikel, “Jurnal Nasional Teknologi dan Sistem Informasi Pembangunan Auto Backup SQL Database Server Menggunakan Raspberry Pi : Studi Kasus,” vol. 03, pp. 130–137, 2018.
- [7] P. Simanjuntak, C. E. Suharyanto, and Jamilah, “Analisis Penggunaan Access Control List (Acl) Dalam Jaringan Komputer Di Kawasan,” *Isd*, vol. 2, no. 2, pp. 122–128, 2017.
- [8] Y. Efendi, “Internet Of Things (Iot) Sistem Pengendalian Lampu Menggunakan Raspberry Pi Berbasis Mobile,” *J. Ilm. Ilmu Komput.*, vol. 4, no. 2, pp. 21–27, 2018, doi: 10.35329/jiik.v4i2.41.
- [9] R. F. Ramadhan and R. Mukhaiyar, “Penggunaan Database Mysql dengan

- Interface PhpMyAdmin sebagai Pengontrolan Smarhome Berbasis Raspberry Pi,” vol. 1, no. 2, pp. 129–134, 2020.
- [10] Nirsal, Rusmala, and Syafriadi, “Desain Dan Implementasi Sistem Pembelajaran Berbasis E-Learning Pada Sekolah Menengah Pertama Negeri 1 Pakue Tengah,” *J. Ilm. d'Computare*, vol. 10, pp. 30–37, 2020, [Online]. Available: <http://www.elsevier.com/locate/scp>
 - [11] C. Bestari Gea, K. Juri Damai Lase, M. Syamsudin, P. Studi Informatika, F. Sains dan Komputer, and U. Kristen Immanuel Yogyakarta, “Implementasi Virtual Private Server untuk Mini Hosting,” *J. InFact Sains dan Komput.*, vol. 7, no. 02, pp. 5–9, 2023.
 - [12] B. Robert and E. B. Brown, “Pengembangan Distro Ubuntu untuk Aplikasi Game Center,” no. 1, pp. 1–14, 2010.
 - [13] F. Nabawi and A. B. Susanto, “Perancangan Sistem Keamanan Server Linux Ubuntu 18 . 04 dengan Metode Ufw Firewall , Hardening , Chmod dan Chown pada UNUSIA Jakarta Abstrak Pendahuluan,” vol. 7, no. 4, 2022.
 - [14] A. Aziz and T. Tampati, “Analisis Web Server untuk Pengembangan Hosting Server Institusi: Pembandingan Kinerja Web Server Apache dengan Nginx,” vol. 1, no. 2, pp. 12–20, 2015.
 - [15] M. Meng, S. Steinhardt, and A. Schubert, “Application Programming Interface Documentation : Application Programming Interface Documentation : What Do Software Developers Want ?,” no. March, 2019, doi: 10.1177/0047281617721853.
 - [16] A. B. Warsito, A. Ananda, and D. Triyanjaya, “Penerapan Data JSON Untuk Mendukung Pengembangan Aplikasi Pada Perguruan Tinggi Dengan Teknik Restfull Dan Web Service,” *Technomedia J.*, vol. 2, no. 1, pp. 26–36, 2017, doi: 10.33050/tmj.v2i1.313.
 - [17] S. Kosasi, I. D. Ayu, E. Yuliani, and G. Syarifudin, “Implementasi Arsitektur Model View Controller pada Website Toko Online Implementation of Model View Controller Architecture on Online Store

- Website,” vol. 3, no. 2, pp. 135–150, 2021, doi: 10.30812/bite.v3i2.1566.
- [18] D. Purnama Sari and R. Wijanarko, “Implementasi Framework Laravel pada Sistem Informasi Penyewaan Kamera (Studi Kasus di Rumah Kamera Semarang),” *J. Inform. dan Rekayasa Perangkat Lunak*, vol. 2, no. 1, p. 32, 2020, doi: 10.36499/jnrpl.v2i1.3190.
- [19] A.-I. A.B., “Implementasi Teknologi Websocket dalam Pengembangan Sistem Berbagi Lokasi Berbasis Web,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 9, pp. 950–959, 2017, [Online]. Available: <http://j-ptiik.ub.ac.id>
- [20] C. Asiminiidis, G. Kokkonis, and S. Kontogiannis, “Database Systems Performance Evaluation for IoT Applications,” *SSRN Electron. J.*, no. November 2019, 2019, doi: 10.2139/ssrn.3360886.

BIODATA MAHASISWA



Nama : Muhammad Khoiril Wafi
NIM : 21060119140133
Konsentrasi : Teknologi Informasi
Tempat Tanggal Lahir : Demak, 4 Maret 2001
Alamat Sekarang : Jl. Baskoro Raya No. 61, RT.03 /
RW.07, Tembalang, Kec.
Tembalang, Kota Semarang,
Jawa Tengah 50275
Alamat E-mail : khoirilwafi123@gmail.com
Nama Orang Tua : Muarifin
Alamat Orang Tua : Jl. Putrajaya No. 06 02/06 Desa
Karangrejo Kec. Bonang Kab.
Demak
IPK : 3.54

Pengalaman dan Prestasi yang Pernah Diraih:

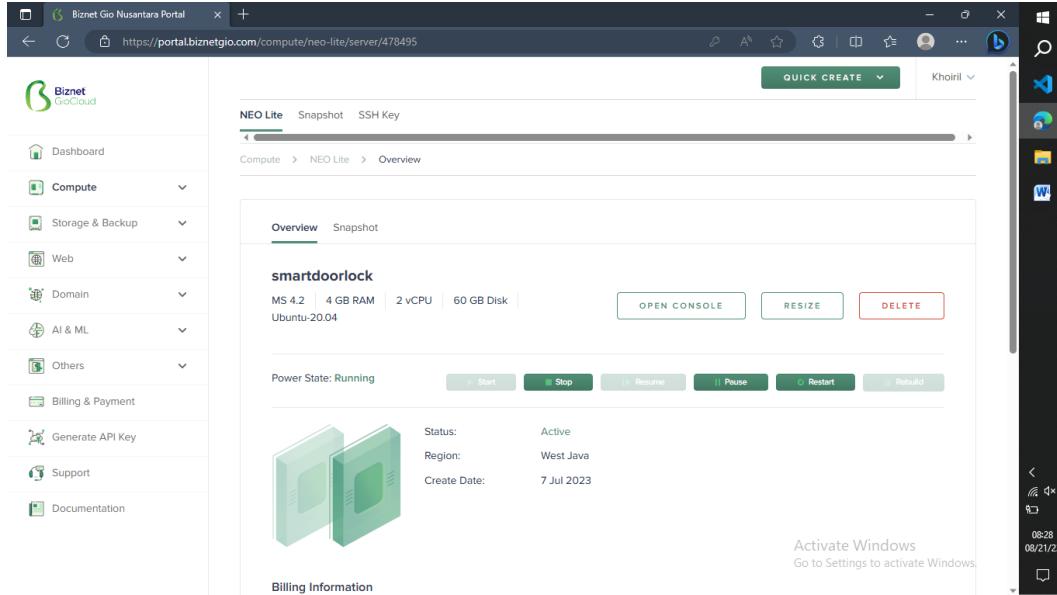
Nama pengalaman / prestasi	Tahun
Sertifikasi Huawei Datacom	2021
Kerja Praktek Kominfo Demak	2022

Semarang, 6 September 2023

(Muhammad Khoiril Wafi)
NIM. 21060119140133

LAMPIRAN A

SERVER



Gambar 1. Tampilan Dashboard VPS

The screenshot shows a terminal window with a black background and white text. It displays a list of active processes using the top command. The output includes columns for PID, USER, NI, VIRT, RES, SHR, %CPU, %MEM, TIME, and Command. The terminal also shows system statistics at the top:

```
2.0% Tasks: 41, 74 thr: 1 running
4.6% Load average: 0.17 0.07 0.06
722M/3.83G Uptime: 13 days, 20:36:47
0K/0K
```

At the bottom right of the terminal window, there's a message: "Activate Windows Go to Settings to activate Windows." The status bar at the bottom right shows the date and time: 08:30 08/21/23.

Gambar 2. Tampilan Server Aktif

LAMPIRAN B

KODE PROGRAM

A. Migrasi Database

```
Schema::create('users', function (Blueprint $table) {
    // identifier
    $table->uuid('id')->primary();
    $table->uuid('added_by')->nullable();

    // authentication parameter
    $table->string('email')->unique();
    $table->string('password');
    $table->rememberToken();

    // profile
    $table->string('name')->unique();
    $table->string('phone')->unique();
    $table->enum('gender', ['laki-laki', 'perempuan']);
    $table->enum('role', ['moderator', 'operator',
        'pengguna']);
    $table->string('avatar')->nullable();

    // timestamp data
    $table->timestamp('email_verified_at')->nullable();
    $table->timestamps();
}) ;

Schema::table('users', function (Blueprint $table) {
    // self-reference relation
    $table->foreign('added_by')->references('id')
        ->on('users')
        ->onUpdate('cascade')
        ->onDelete('restrict');
}) ;

Schema::create('offices', function (Blueprint $table) {
    // identifier
    $table->uuid('id')->primary();

    // data
    $table->string('name');
    $table->uuid('user_id');

    // timestamp
    $table->timestamps();
});
```

```

Schema::table('offices', function (Blueprint $table) {

    // relation to users table
    $table->foreign('user_id')->references('id')
        ->on('users')
        ->onDelete('restrict')
        ->onUpdate('cascade');
});

Schema::create('doors', function (Blueprint $table) {

    // identifier
    $table->uuid('id')->primary();
    $table->uuid('office_id');
    $table->uuid('socket_id')->unique()->nullable();

    // data
    $table->string('name');
    $table->string('device_name')->unique()->nullable();
    $table->string('device_pass')->unique()->nullable();
    $table->string('key')->nullable();
    $table->boolean('is_lock')->default(1);

    // timestamp
    $table->timestamps();
});

Schema::table('doors', function (Blueprint $table) {

    // relation to offices table
    $table->foreign('office_id')->references('id')
        ->on('offices')
        ->onDelete('restrict')
        ->onUpdate('cascade');
});

Schema::create('access', function (Blueprint $table) {

    // identifier
    $table->uuid('id')->primary();
    $table->uuid('user_id');
    $table->uuid('door_id');

    // data
    $table->time('time_begin');
    $table->time('time_end');
    $table->date('date_begin')->default(null)->nullable();
    $table->date('date_end')->default(null)->nullable();
    $table->boolean('is_temporary')->nullable();
    $table->boolean('is_running')->nullable();

    // timestamp
    $table->timestamps();
});

```

```

Schema:::table('access', function (Blueprint $table) {

    // relation to users table
    $table->foreign('user_id')->references('id')
        ->on('users')
        ->onUpdate('cascade')
        ->onDelete('cascade');

    // relation to doors table
    $table->foreign('door_id')->references('id')
        ->on('doors')
        ->onUpdate('cascade')
        ->onDelete('cascade');
});

Schema:::create('schedules', function (Blueprint $table) {

    // identifier
    $table->uuid('id')->primary();
    $table->uuid('office_id');

    // data
    $table->string('name');
    $table->date('date_begin');
    $table->date('date_end');
    $table->time('time_begin');
    $table->time('time_end');
    $table->boolean('is_repeating')->default(0);
    $table->string('day_repeating');
    $table->enum('status', ['waiting', 'running', 'done'])
        ->default('waiting');

    // timestamp
    $table->timestamps();
});

Schema:::table('schedules', function (Blueprint $table) {

    // relation to offices table
    $table->foreign('office_id')->references('id')
        ->on('offices')
        ->onUpdate('cascade')
        ->onDelete('cascade');
});

Schema:::create('door_schedule', function (Blueprint $table)
{

    // identifier
    $table->uuid('id')->primary();
    $table->uuid('schedule_id');
    $table->uuid('door_id');

    // timestamp
    $table->timestamps();
});

```

```

Schema::table('door_schedule', function (Blueprint $table) {

    // relation to schedules table
    $table->foreign('schedule_id')->references('id')
        ->on('schedules')
        ->onUpdate('cascade')
        ->onDelete('cascade');

    // relation to doors table
    $table->foreign('door_id')->references('id')
        ->on('doors')
        ->onUpdate('cascade')
        ->onDelete('cascade');
});
```

B. Routing API

```

// login
Route::post('/login', [AuthController::class,
    'authenticate'])->middleware('guest');

// verify otp via email
Route::post('/verify-email', [AuthController::class,
    'verifyEmail'])->middleware('auth:sanctum');

// reset password
Route::post('/reset-password', [AuthController::class,
    'resetPassword'])->middleware('guest');

// logout
Route::get('/logout', [AuthController::class, 'logout'])
->middleware('auth:sanctum');

Route::middleware(['auth:sanctum', 'verified'])
->group(function () {

    // update user profile
    Route::post('/update-profile', [UserController::class,
        'updateProfile']);

    // get user avatar
    Route::get('/avatar/', [UserController::class,
        'getAvatar']);

    // update avatar
    Route::post('/update-avatar', [UserController::class,
        'updateAvatar']);

    // change password
    Route::post('/change-password', [UserController::class,
        'changePassword']);

    // get user access
    Route::get('/my-access', [AccessController::class,
```

```

'myAccess']);

// get door at office
Route::get('/get-door', [AccessController::class,
'getOfficeDoor']);

// check access
Route::get('/verify-access/{door_id}', [AccessController::class, 'verifyAccess']);

// remote access
Route::post('/remote-access', [AccessController::class,
'remoteAccess']);

// get signature
Route::post('/get-signature', [AuthController::class,
'signature']);

// history access
Route::get('/my-history', [AccessController::class,
'getHistory']);
});

```

C. Routing Door

```

Route::post('/login', [AuthController::class,
'authenticate'])->middleware('guest');

Route::post('/register', [AuthController::class,
'register'])->middleware('guest');

Route::middleware('auth:sanctum')->group(function () {

    // get signature
    Route::post('/get-signature', [AuthController::class,
'signature']);

    // logout
    Route::get('/logout', [AuthController::class,
'logout']);

    // update door status
    Route::post('/update-status',
[DoorEventController::class, 'update_status']);

    // door alert
    Route::post('/alert', [DoorEventController::class,
>alert']);
});

```

D. Kontroler API

```

class AccessController extends Controller
{
    public function myAccess(Request $request)
    {
        $user = $request->user();

        if ($user->role == 'operator') {

            $office = Office::where('user_id', $user->id)
                ->select('id')->first();

            return response()->json([
                'status' => 'success',
                'data' => Door::with(
                    ['office' => function ($query) {
                        $query->select(['id', 'name']);
                    }])->where('office_id', $office->id)
                    ->select(['id', 'office_id', 'name'])
                    ->get(),
            ], 200);
        }

        $access = Access::with(['door' => function ($query) {
            $query->select(['id', 'office_id', 'name']);
        }])->where('user_id', $user->id)
            ->select(['door_id', 'time_begin', 'time_end',
            'date_begin', 'date_end', 'is_running'])->get();

        if ($access) {
            Log::info('user request access using api',
                ['user' => $user]);
            return response()->json([
                'status' => 'success',
                'data' => $access
            ], 200);
        }

        return response()->json([
            'status' => 'no_data',
            'data' => []
        ], 200);
    }

    public function getOfficeDoor(Request $request)
    {
        $user = $request->user();

        if ($user->role != 'operator') {
            return response()->json([
                'message' => 'Not Authorized',
            ], 401);
        }

        $office = Office::select('id')->where('user_id',
        $user->id)->first();
    }
}

```

```

Log::info('user request door using api', ['user' =>
$user]);

return response()->json([
    'status' => 'success',
    'office' => $office->id,
    'data' => Door::where('office_id', $office->id)
        ->select(['id', 'name', 'key', 'socket_id',
        'is_lock'])->get()
], 200);
}

public function verifyAccess(Request $request, $door_id)
{
    $user = $request->user();
    $door = Door::with('office')->where('id', $door_id)
        ->whereNotNull('device_name')->first();

    if (!$door) {
        return response()->json([
            'status' => 'no_data',
            'data' => []
        ], 200);
    }

    $bt = hash_hmac('sha256', $door->device_name,
        config('broadcasting.connections.pusher.secret'));
    $bt_name = substr($bt, 0, 5) . substr($bt, -5);

    if ($user->role == 'operator' && $user->id == $door
        ->office->user_id) {
        return response()->json([
            'status' => 'success',
            'data' => [
                'door' => $door->name,
                'key' => $door->key,
                'bluetooth' => $bt_name,
            ]
        ], 200);
    }

    $now = Carbon::now();

    $date = $now->toDateString();
    $time = $now->toTimeString();

    $access = Access::with('door')
        ->where('user_id', $user->id)
        ->where('door_id', $door_id)
        ->where('is_running', 1)
        ->where('time_begin', '<=', $time)
        ->where('time_end', '>=', $time)
        ->first();
}

```

```

if ($access) {
    if ($access['is_temporary'] == 1) {
        if ($access['date_begin'] <= $date &&
            $access['date_end'] >= $date) {
            new CustomLog($access->user_id,
                $access->door_id, $access->door
                ->office_id, 'mendapatkan kunci akses');

            return response()->json([
                'status' => 'success',
                'data' => [
                    'door' => $access->door->name,
                    'key' => $access->door->key,
                    'bluetooth' => $bt_name,
                ],
            ], 200);
        }
    } else {
        new CustomLog($access->user_id, $access
            ->door_id, $access->door->office_id,
            'mendapatkan kunci akses');

        return response()->json([
            'status' => 'success',
            'data' => [
                'door' => $access->door->name,
                'key' => $access->door->key,
                'bluetooth' => $bt_name,
            ],
        ], 200);
    }
}

return response()->json([
    'status' => 'failed',
    'data' => []
], 200);
}

public function remoteAccess(Request $request)
{
    $user = $request->user();

    if ($user->role != 'operator') {
        return response()->json([
            'status' => 'not authorized',
            'data' => []
        ], 200);
    }

    $data = $request->only(['door_id', 'locking']);

    $validator = Validator::make($data, [
        'door_id' => ['required', 'string'],
        'locking' => ['required', 'string']
    ]);
}

```

```

if ($validator->fails()) {
    return response()->json([
        'status' => 'missing_parameter',
        'data' => $validator->messages()
    ], 200);
}

$door = Door::where('id', $data['door_id'])
->first();

if (!$door) {
    return response()->json([
        'status' => 'no_data',
        'data' => []
    ], 200);
}

event(new DoorCommandEvent($door->office_id, $user
->id, $door->id, $data['locking'], $door->key));
new CustomLog($user->id, $door->id, $door
->office_id, 'remote akses');

return response()->json([
    'status' => 'success',
    'data' => []
], 200);
}

public function getHistory(Request $request)
{
    $user = $request->user();

    $history = AccessLog::with(['door' => function
    ($query) {
        $query->select(['id', 'name']);
    }, 'office' => function ($query) {
        $query->select(['id', 'name']);
    }])->select(['id', 'door_id', 'office_id', 'log',
    'created_at'])->where('user_id', $user->id)
    ->orderBy('created_at', 'DESC')->get();

    return response()->json([
        'status' => 'success',
        'data' => $history
    ], 200);
}

class AuthController extends Controller
{
    protected function generateOTP($id)
    {
        $data = array(
            'code_otp' => rand(123456, 999999),
            'valid_until' => Carbon::now()->addMinutes(5),
        );
    }
}

```

```

        Otp::updateOrCreate(['user_id' => $id], $data);

        return $data['code_otp'];
    }

    public function authenticate(Request $request)
    {
        $data = $request->only(['email', 'password']);

        $validator = Validator::make($data, [
            'email' => ['required', 'string', 'email'],
            'password' => ['required', 'string']
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 'missing_parameter',
                'data' => $validator->messages()
            ], 200);
        }

        $user = User::where('email', $data['email'])
        ->first();

        // cek jika email valid dan password sesuai
        if (!$user || !Hash::check($data['password'], $user
        ->password)) {
            return response()->json([
                'status' => 'failed',
                'data' => []
            ], 200);
        }

        // generate token
        $token = $user->createToken('api-token')
        ->plainTextToken;
        Log::info('user login with api', ['user' => $user]);

        // cek apakah user sudah verifikasi email
        if ($user->email_verified_at == null) {

            // get otp
            $last_otp = Otp::where('user_id', $user->id)
            ->first();

            // kirim kode otp
            if ($last_otp == null) {
                $otp = $this->generateOTP($user->id);
                $user->notify(new
                    SendOTPNotification($otp));
            }
        }
    }
}

```

```

    // send email notification
    return response()->json([
        'status' => 'email_unverified',
        'data' => [
            'id' => $user->id
        ],
        'token' => $token
    ], 200);
}

return response()->json([
    'status' => 'success',
    'data' => [
        "id" => $user->id,
        "email" => $user->email,
        "name" => $user->name,
        "phone" => $user->phone,
        "gender" => $user->gender,
        "role" => $user->role,
        "avatar" => $user->avatar,
    ],
    'token' => $token
], 200);
}

public function logout(Request $request)
{
    $user = $request->user();
    Log::info('user logout using api', ['user' => $user]);

    $user->tokens()->delete();
    return response()->json(['status' => 'success',
    'data' => []], 200);
}

public function verifyEmail(Request $request)
{
    $now = Carbon::now();

    $data = $request->only(['otp']);
    $user = $request->user();

    $validator = Validator::make($data, [
        'otp' => ['required', 'numeric', 'digits:6'],
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $otp = Otp::where('user_id', $user->id)
->where('code_otp', $data['otp'])->first();
}

```

```

if ($otp && $now->lessThanOrEqualTo($otp
->valid_until)) {
    User::where('id', $user->id)
    ->update(['email_verified_at' => $now]);
    Otp::where('user_id', $user->id)->delete();

    Log::info('user verify email using api', ['user'
=> $user]);

    return response()->json([
        'status' => 'success',
        'data' => User::where('id', $user->id)
            ->select(['id', 'email', 'name', 'phone',
            'gender', 'role', 'avatar'])->first(),
    ], 200);
} else if ($otp) {
    Otp::where('user_id', $user->id)->delete();
    return response()->json([
        'status' => 'otp_expired',
        'data' => []
    ], 200);
}

return response()->json([
    'status' => 'otp_not_match',
    'data' => []
], 200);
}

public function resetPassword(Request $request)
{
    $data = request()->only('email');

    $validator = Validator::make($data, [
        'email' => ['required', 'email:dns'],
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $status = Password::sendResetLink($data);

    // check status
    if ($status === Password::RESET_LINK_SENT) {
        Log::info('reset link send', ['email' =>
        $data['email']]);
        return response()->json([
            'status' => 'success',
            'data' => [
                'email' => $data['email']
            ]
        ], 200);
    }
}

```

```

// return error
return response()->json([
    'status' => 'failed',
    'data' => [
        'message' => 'email not found',
        'email' => $data['email'],
    ]
], 200);
}

public function signature(Request $request)
{
    $data = $request->only(['socket_id', 'office_id',
    'channel_data']);

    $validator = Validator::make($data, [
        'socket_id' => ['required', 'string'],
        'office_id' => ['required', 'string'],
        'channel_data' => ['required', 'string']
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $signature = $data['socket_id'] . ':presence
-office.' . $data['office_id'] . ':' .
$data['channel_data'];
$hash = hash_hmac('sha256', $signature,
config('broadcasting.connections.pusher.secret'));

Log::info('mobile app signature', ['app' => $data]);

return response()->json([
    'status' => 'success',
    'data' => [
        'signature' => $hash
    ]
], 200);
}

class UserController extends Controller
{
    public function updateProfile(Request $request)
    {
        $data = $request->only(['name', 'email', 'gender',
        'phone']);
        $user = $request->user();

        $validator = Validator::make($data, [
            'name' => ['required', 'min:4',
            Rule::unique('users')->ignore($user->id)],
            'email' => ['required', 'email:dns',
        ]);
    }
}

```

```

        Rule::unique('users')->ignore($user->id)],
        'gender' => ['required'],
        'phone'  => ['required', 'numeric'],
        Rule::unique('users')->ignore($user->id),
        'digits_between:11,13'],
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data'   => $validator->messages()
        ], 200);
    }

    $user = User::where('id', $user->id)->first();

    if ($user) {
        $user->name    = $data['name'];
        $user->email   = $data['email'];
        $user->gender  = $data['gender'];
        $user->phone   = $data['phone'];

        $status = $user->save();

        if ($status) {
            Log::info('user update profile using api',
                ['user' => $user]);

            return response()->json([
                'status' => 'success',
                'data'   => $user
            ], 200);
        }
    }

    return response()->json([
        'status' => 'failed',
        'data'   => []
    ], 200);
}

public function getAvatar(Request $request)
{
    $file_name = request()->user()->avatar;

    if ($file_name == null) {
        $file = storage_path(
            '/app/images/02943e5368adf6cc72f4a2e0a435090b.png');
    } else {
        $file = storage_path('/app/images/' .
            $file_name);
    }

    return response()->file($file);
}

```

```

public function updateAvatar(Request $request)
{
    $data = $request->only('avatar');

    $validator = Validator::make($data, [
        'avatar' => ['required', 'image',
        'mimes:jpg,jpeg,png', 'max:1024'],
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $user = User::where('id', $request->user()->id)
->first();

    if ($user->avatar != null) {
        Storage::disk('local')->delete('/images/' .
        $user->avatar);
    }

    $name = md5(Carbon::now()) . '.' . $data['avatar']
->getClientOriginalExtension();

    try {

        // store avatar
        $data['avatar']->storeAs('images', $name);
        $user->avatar = $name;
        $user->save();
        Log::info('avatar change', ['user' => $user]);

        return response()->json([
            'status' => 'success',
            'data' => [
                'avatar' => $name,
            ]
        ], 200);
    } catch (Exception $e) {
        Log::error('avatar change failed', ['user' =>
        $user, 'error' => $e]);
    }

    return response()->json(['status' => 'error'], 200);
}

public function changePassword(Request $request)
{
    $user = request()->user();
    $data = request()->only(['password_now', 'password',
    'password confirmation']);
}

```

```

$validator = Validator::make($data, [
    'password_now'          => ['required',
        'string', 'min:8', 'max:50'],
    'password'              => ['required',
        'string', 'min:8', 'max:50', 'confirmed'],
    'password_confirmation' => ['required',
        'string', 'min:8', 'max:50'],
]);
}

if ($validator->fails()) {
    return response()->json([
        'status' => 'missing_parameter',
        'data' => $validator->messages()
    ], 200);
}

if (!Hash::check($data['password_now'], $user
->password)) {
    return response()->json([
        'status' => 'failed',
        'data' => [
            'message' => 'password not match',
        ]
    ], 200);
}

try {
    // update password
    $user->forceFill(['password' =>
    Hash::make($data['password'])]);
    $user->save();

    Log::info('change password', ['user' => $user]);

    return response()->json([
        'status' => 'success',
        'data' => [
            'message' => 'password saved'
        ]
    ], 200);
} catch (Exception $e) {
    Log::error('change password failed', ['user' =>
    $user, 'error' => $e]);
}

return response()->json([
    'status' => 'failed',
    'data' => [
        'message' => 'password save failed',
    ]
], 200);
}
}

```

E. Kontroler Door

```

class AuthController extends Controller
{
    public function authenticate(Request $request)
    {
        $data = $request->only(['device_name',
        'device_pass']);

        $validator = Validator::make($data, [
            'device_name' => ['required', 'string'],
            'device_pass' => ['required', 'string']
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 'missing_parameter',
                'data' => $validator->messages()
            ], 200);
        }

        $door = Door::where('device_name',
        $data['device_name'])->first();

        if (!$door || !Hash::check($data['device_pass'],
        $door->device_pass)) {
            return response()->json([
                'status' => 'failed',
                'data' => []
            ], 200);
        }

        $token = $door->createToken('auth_token')
        ->plainTextToken;
        Log::info('door device login', ['door' => $door]);

        return response()->json([
            'status' => 'success',
            'data' => [
                'office_id' => $door->office_id,
                'door_id' => $door->id,
                'door_name' => $door->name,
            ],
            'token' => $token
        ], 200);
    }

    public function register(Request $request)
    {
        $data = $request->only(['id', 'device_name']);

        $validator = Validator::make($data, [
            'id' => ['required', 'string'],
            'device_name' => ['required', 'string'],
        ]);
    }
}

```

```

if ($validator->fails()) {
    return response()->json([
        'status' => 'missing_parameter',
        'data' => $validator->messages()
    ], 200);
}

$door = Door::where('id', $data['id'])->first();

if (!$door) {
    return response()->json([
        'status' => 'no_data',
        'data' => []
    ], 200);
}

if ($door->device_name != null) {
    return response()->json([
        'status' => 'already_exist',
        'data' => []
    ], 200);
}

$pass = Random::generate(20);
$device_key = Random::generate(20);

$door->device_name = $data['device_name'];

$door->forceFill(['device_pass' =>
Hash::make($pass)]);
$door->key = $device_key;

try {
    $door->save();
    Log::info('door device register', ['door' =>
$door]);

    return response()->json([
        'status' => 'success',
        'data' => [
            'office_id' => $door->office_id,
            'door_id' => $door->id,
            'door_name' => $door->name,
            'device_pass' => $pass,
            'key' => $device_key,
        ],
    ], 200);
} catch (Exception $e) {
    Log::error('door device register failed',
['door' => $door, 'error' => $e]);

    return response()->json([
        'status' => 'failed',
        'data' => []
    ], 200);
}
}

```

```

public function signature(Request $request)
{
    $data = $request->only(['socket_id', 'office_id',
    'channel_data']);

    $validator = Validator::make($data, [
        'socket_id' => ['required', 'string'],
        'office_id' => ['required', 'string'],
        'channel_data' => ['required', 'string']
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $signature = $data['socket_id'] . ':presence-
    office.' . $data['office_id'] . ':' .
    $data['channel_data'];
    $hash = hash_hmac('sha256', $signature,
    config('broadcasting.connections.pusher.secret'));

    Log::info('door device signature', ['door' =>
    $data]);

    return response()->json([
        'status' => 'success',
        'data' => [
            'signature' => $hash
        ]
    ], 200);
}

public function logout(Request $request)
{
    $user = $request->user();

    Log::info('door device logout', ['door' => $user]);

    $user->tokens()->delete();
    return response()->json(['status' => 'success'],
    200);
}

class DoorEventController extends Controller
{
    public function update_status(Request $request)
    {
        $data = $request->only(['door_id', 'office_id',
        'socket_id', 'user_id', 'lock_status']);

        $validator = Validator::make($data, [
            'door_id' => ['required', 'string'],
            'office_id' => ['required', 'string'],
            'socket_id' => ['required', 'string'],
            'user_id' => ['required', 'string'],
            'lock_status' => ['required', 'string']
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 'missing_parameter',
                'data' => $validator->messages()
            ], 200);
        }

        $signature = $data['socket_id'] . ':presence-
        office.' . $data['office_id'] . ':' .
        $data['channel_data'];
        $hash = hash_hmac('sha256', $signature,
        config('broadcasting.connections.pusher.secret'));

        Log::info('door device signature', ['door' =>
        $data]);

        return response()->json([
            'status' => 'success',
            'data' => [
                'signature' => $hash
            ]
        ], 200);
    }
}

```

```

'socket_id'    => ['required', 'string'],
'user_id'      => ['required', 'string'],
'lock_status'  => ['required', 'integer']
]);

if ($validator->fails()) {
    return response()->json([
        'status' => 'missing_parameter',
        'data'   => $validator->messages()
    ], 200);
}

$door = Door::where('id', $data['door_id'])
->first();

// generate key
$key = Random::generate(20);

if ($door) {
    $door->socket_id = $data['socket_id'];
    $door->is_lock = $data['lock_status'];
    $door->key = $key;

    $status = $door->save();

    if ($status) {

        // broadcast event
        event(new
            DoorStatusEvent($data['office_id']));

        // save log
        if ($data['door_id'] == $data['user_id']) {
            Log::info('door self update', ['door' =>
                $door]);
        } else {
            new CustomLog($data['user_id'],
                $data['door_id'], $door->office_id,
                ($data['lock_status'] == 0) ? 'membuka
                    pintu' : 'mengunci pintu');
        }
    }

    return response()->json([
        'status' => 'success',
        'data'   => [
            'key' => $key
        ]
    ], 200);
}
}

return response()->json([
    'status' => 'failed',
    'data'   => []
], 401);
}

```

```
public function alert(Request $request)
{
    $data = $request->only(['door_id', 'office_id',
    'message']);

    $validator = Validator::make($data, [
        'office_id' => ['required', 'string'],
        'door_id' => ['required', 'string'],
        'message' => ['required', 'string']
    ]);

    if ($validator->fails()) {
        return response()->json([
            'status' => 'missing_parameter',
            'data' => $validator->messages()
        ], 200);
    }

    $door = Door::where('id', $data['door_id'])
    ->first();

    if (!$door) {
        return response()->json([
            'status' => 'no_data',
            'data' => []
        ], 200);
    }

    event(new DoorAlertEvent($data['office_id'], $door
    ->name, $data['message']));
    Log::alert('door alert', ['door' => $door, 'message'
    => $data['message']]);

    return response()->json([
        'status' => 'success',
        'data' => []
    ], 200);
}
```

LAMPIRAN C
MAKALAH TUGAS AKHIR

PERANCANGAN SISTEM DATABASE DAN SERVER PADA SISTEM KEAMANAN KUNCI PINTU GEDUNG DENGAN ACCESS CONTROL

Muhammad Khoiril Wafi *), M. Arfan, S.Kom., M.Eng. dan Imam Santoso, S.T., M.T.

Program Studi Sarjana Departemen Teknik Elektro, Universitas Diponegoro
Jl. Prof Sudharto, SH, Kampus UNDIP Tembalang, Semarang 50275, Indonesia

**)E-mail : khoirilwafi@students.undip.ac.id*

Abstrak

Banyak gedung yang masih menggunakan sistem penguncian manual dengan menggunakan kunci fisik sehingga dalam satu gedung akan mempunyai banyak kunci untuk masing-masing pintu hal tersebut menjadikan proses pengelolaan akses pintu kurang optimal. Untuk mengatasi hal tersebut maka dilakukan perancangan mengenai sistem penguncian pintu gedung yang dapat mengoptimalkan pengelolaan akses pintu serta meningkatkan efisiensi penguncian pintu gedung. Sistem yang dibuat menggunakan konsep IoT dimana terdapat beberapa perangkat kunci untuk masing-masing pintu yang terhubung ke sebuah server sebagai kendali dan data yang mengatur semua pintu serta memberikan informasi kepada pihak pengelola gedung tersebut mengenai kondisi pintu pada gedung tersebut. Beberapa fitur seperti pengawasan langsung, pindai kode QR, penjadwalan dan kendali jarak jauh akan disediakan untuk mengoptimalkan pengawasan dan pengelolaan pintu. Sistem yang dibangun terdiri dari backend API menggunakan Laravel, database menggunakan MySQL, penjadwalan menggunakan Crobjob serta komunikasi menggunakan WebSocket dimana semua komponen tersebut akan dipasang pada sebuah Virtual Private Server dengan menggunakan sistem operasi Ubuntu.

Kata Kunci: Akses Pintu, kode QR, IoT, Server, Database, API.

Abstract

Many buildings still use a manual locking system using physical keys so that in one building there will be many keys for each door, making the door access management process less than optimal. To overcome this, a design is carried out regarding a building door locking system that can optimize door access management and increase the efficiency of locking building doors. The system created uses the concept of IoT where there are several key devices for each door connected to a server as control and data that regulates all doors and provides information to the building manager about the condition of the doors in the building. Several features such as live monitoring, QR code scanning, scheduling and remote control will be provided to optimize door monitoring and management. The system consists of a backend API using Laravel, a database using MySQL, scheduling using Crobjob and communication using WebSocket where all components will be installed on a Virtual Private Server using Ubuntu operating system.

Keywords: Door Access, QR code, IoT, Server, Database, API.

I. Pendahuluan

1.1 Latar Belakang

Keamanan menjadi hal yang harus diperhatikan dalam sebuah gedung atau bangunan. Pada saat ini sistem penguncian masih banyak menggunakan penguncian tradisional dengan menggunakan kunci fisik yang tidak efisien mengingat jumlah ruangan yang banyak, kunci fisik juga mempunyai tingkat keamanan yang kurang dikarenakan kunci rentan untuk dicuri atau diduplikasi[1]. Masalah keamanan ruangan dalam sebuah gedung dan efektifitas dapat diselesaikan dengan menggunakan

sebuah sistem penguncian cerdas yang terorganisasi dan kerkoneksi ke sebuah sistem manajemen kunci pintu (*access control*) yang memiliki tingkat keamanan yang lebih tinggi, adaptif dan fleksibel[2], [3].

Untuk mendukung kinerja dari sistem penguncian yang terorganisasi maka diperlukan sebuah *server* dan penyimpanan data. sebuah *server* akan menjalankan kode program yang bertugas untuk mengatur dan mengawasi semua aktifitas sistem penguncian dan sebuah penyimpanan data digunakan untuk menyimpan data-data seperti data pengguna, kunci, dan *backup*[4]. Dengan

demikian, perancangan sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan *access control* diharapkan dapat memberikan solusi yang efektif dalam meningkatkan keamanan kunci pintu gedung dan memberikan kenyamanan serta kemudahan dalam pengelolaannya.

1.2 Tujuan

Tugas akhir ini bertujuan untuk merancang sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan *access control*.

1.3 Batasan Masalah

Tugas akhir ini hanya akan membahas tentang perancangan *database* dan *server* pada sistem keamanan kunci pintu gedung dengan *access control* dengan memberikan solusi perancangan yang sesuai dengan kaidah keilmuan rekayasa.

II. Kajian Pustaka

2.1 Access Control

Kendali akses atau *access control* merupakan sebuah mekanisme pengaturan kebijakan yang digunakan untuk membatasi dan mengatur hak akses pengguna terhadap suatu sumber daya atau fasilitas tertentu. Kendali akses atau *access control* merupakan sebuah mekanisme pengaturan kebijakan yang digunakan untuk membatasi dan mengatur hak akses pengguna terhadap suatu sumber daya atau fasilitas tertentu. Dengan menggunakan kendali akses kita bisa mengatur dan membatasi akses pengguna sehingga hanya pengguna tertentu yang diijinkan yang bisa mengakses sumber daya yang dilindungi[5].

2.2 Internet of Things

IoT merupakan sebuah konsep yang digunakan untuk mengembangkan koneksi internet, IoT juga memberikan gambaran mengenai kemampuan dari berbagai perangkat elektronik yang saling terhubung dengan membentuk sebuah jaringan komunikasi baik melalui internet maupun komunikasi lainnya seperti bluetooth. Dengan menggunakan konsep IoT kita dapat menghubungkan peralatan seperti sensor dan aktuator yang terhubung ke sebuah jaringan menjadi sebuah kesatuan sistem yang dapat dikendalikan secara efektif dan efisien dengan tingkat kerumitan yang rendah[6].

2.3 Database MySQL

Database atau basisdata merupakan sekumpulan data yang terintegrasi dan diatur sedemikian rupa sehingga data tersebut dapat dicari, diambil, ditambahkan, dan diolah dengan tepat[5].

MySQL merupakan singkatan dari *Structured Query Language*. SQL merupakan bahasa terstruktur yang khusus digunakan untuk mengolah *database*. MySQL merupakan sistem manajemen *database* yang

bersifat *relational*. Artinya, data yang dikelola dalam *database* akan diletakkan pada beberapa tabel yang terpisah sehingga manipulasi data akan jauh lebih cepat[7].

2.4 Laravel

Laravel merupakan kerangka kerja pemrograman web yang menggunakan bahasa pemrograman PHP yang terbuka dan gratis, laravel diperuntukkan untuk pengembangan aplikasi berbasis *website* maupun API dengan menggunakan pendekatan pola MVC atau *Model View Controller*[8]. Arsitektur MVC memiliki *business logic* yang terpisah dari *model* dan *presentation*, sehingga saat melakukan modifikasi pada program tidak mempengaruhi komponen lain yang tidak diubah, dan proses pengembangan yang lebih cepat, serta dapat menggunakan *reuse of code* dimana fungsi ini berguna dalam pengembangan *website* tanpa harus melakukan *coding* dari awal[9].

2.5 Websocket

Websocket merupakan sebuah protokol komunikasi web berbasis *client-server*, keberadaan websocket dinilai dapat menggantikan teknologi AJAX sebagai pendahulu komunikasi *client-server*. Websocket merupakan teknologi yang mampu memberikan performa terbaik ketika diimplementasikan dalam sistem dengan *rate-request* tinggi, dibandingkan dengan teknologi komunikasi lain termasuk AJAX[10]. Websocket memungkinkan komunikasi dua arah antara *client* dan *server* dengan menggunakan koneksi yang sudah terjalin, hal ini dikarenakan pada websocket, koneksi akan terus terjalin selama tidak terjadi error atau ada permintaan pemutusan koneksi. Salah satu layanan yang menyediakan koneksi websocket yaitu Pusher, Pusher menyediakan komunikasi *realtime* antara *server* dan *client* melalui *channel-channel* yang telah tersedia.

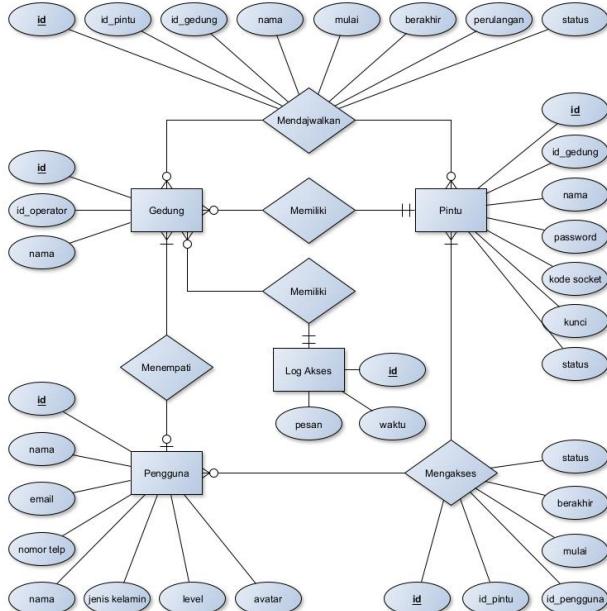
2.6 Virtual Private Server

VPS adalah jenis *server* yang secara eksklusif diperuntukkan bagi satu pengguna, sehingga seluruh sumber daya yang ada di dalamnya tidak dipengaruhi atau dibagi dengan pengguna lain. Dengan menggunakan teknologi VPS, sebuah mesin fisik dapat menjalankan beberapa sistem operasi secara bersamaan. Pengguna VPS memiliki kendali penuh untuk mengatur seluruh konfigurasi sesuai kebutuhan. Teknologi yang digunakan dalam VPS adalah virtualisasi hardware pada server fisik yang memungkinkan pembagian sumber daya menjadi beberapa bagian yang berbeda, sehingga setiap VPS berfungsi seperti *server* pribadi yang terisolasi dari pengguna lainnya[11].

III. Perancangan

3.1 Database

Berdasarkan penelitian [12] yang membandingkan kinerja dari berbagai tipe dan jenis *database* didapatkan hasil penggunaan MySQL menunjukkan hasil kinerja yang bagus dalam hal waktu eksekusi permintaan, dengan sistem penyimpanan data bersifat relasional dan terstruktur maka MySQL dapat diterapkan pada sistem penguncian pintu gedung ini.



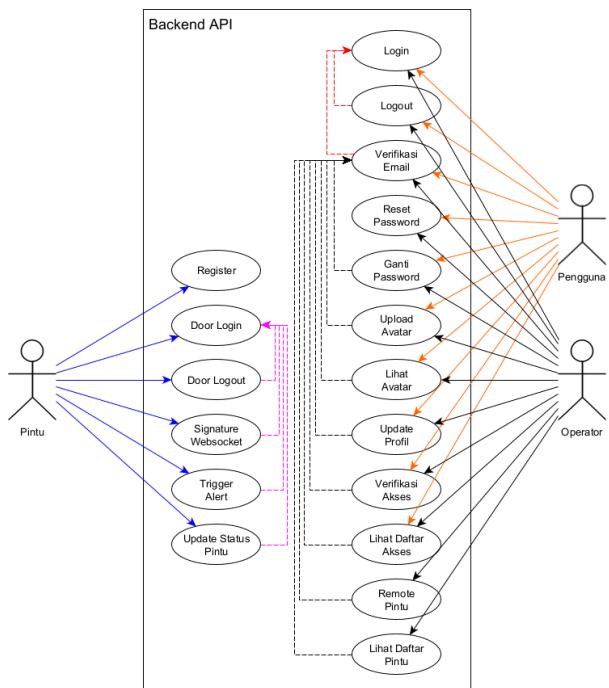
Gambar 1 ERD Database

Pada Gambar 1 terlihat relasi antar entitas didalam *database* sistem penguncian pintu gedung. Pada *database* sistem penguncian pintu gedung terdapat beberapa entitas seperti pengguna, pintu, gedung dan log akses, entitas tersebut digunakan untuk menyimpan data yang akan digunakan didalam pengelolaan sistem dengan isi data sesuai dengan atribut dari masing-masing entitas. Dalam sistem *database* juga terdapat relasi, relasi menunjukkan hubungan antar entitas.

3.2 Backend API

3.2.1 Use Case

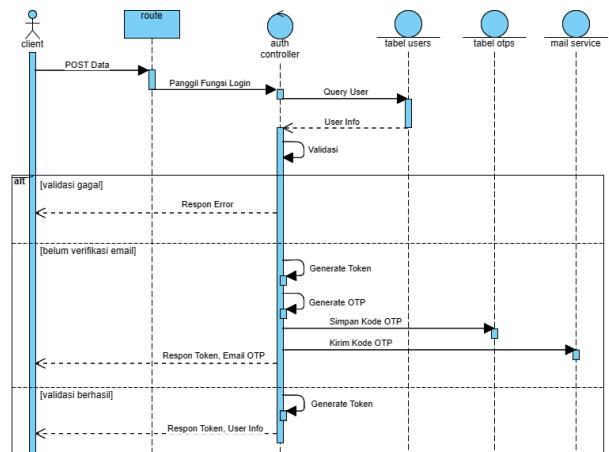
Terdapat 3 aktor yang berinteraksi dengan sistem melalui API yaitu pintu, pengguna dan operator. Pintu merupakan perangkat IoT yang digunakan untuk melakukan penguncian pada gedung, sedangkan pengguna dan operator merupakan aplikasi *mobile* yang digunakan sebagai antarmuka sistem, beberapa metode pada API mungkin membutuhkan akses *login* untuk autentikasinya. Diagram *Use Case* dari *Backend API* dapat dilihat pada Gambar 2 di bawah.



Gambar 2 Diagram Use Case API

3.2.2 Login

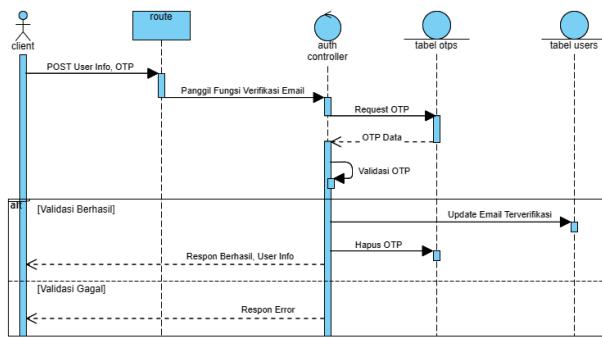
API *login* digunakan untuk melakukan autentikasi *client* melalui *username* dan *password* yang dikirimkan, API *login* juga memeriksa apakah pengguna dan operator sudah melakukan verifikasi email, jika belum maka *login* akan tertahan sampai pengguna melakukan verifikasi email. Diagram fungsi *login* dapat dilihat pada Gambar 3.



Gambar 3 API Login

3.2.3 Verifikasi Email

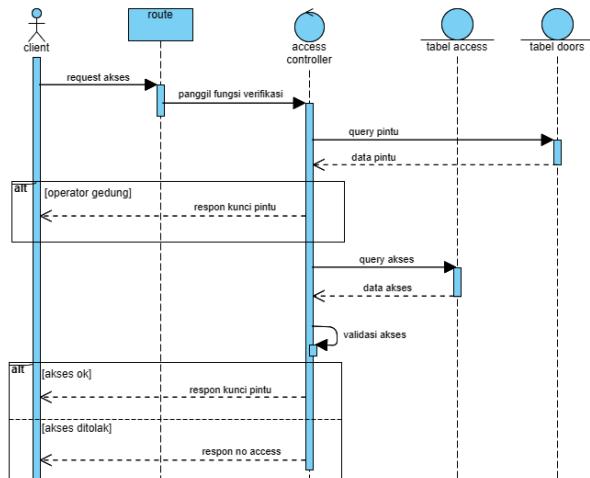
API verifikasi email digunakan untuk memastikan bahwa pengguna dan operator memiliki email yang valid dan aktif, dengan adanya verifikasi email maka akan meningkatkan keamanan dengan hanya mengijinkan pengguna dan operator yang terpercaya untuk mengakses sumber daya yang ada. Verifikasi email dilakukan dengan cara mengirimkan kode OTP (*One Time Password*) ke email yang telah didaftarkan. Diagram dari API verifikasi email dapat dilihat pada Gambar 4.



Gambar 4 API Verifikasi Email

3.2.4 Verifikasi Akses

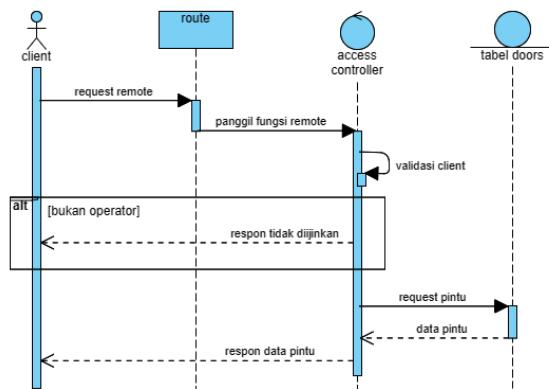
Verifikasi akses digunakan oleh pengguna dan operator untuk memverifikasi diri mereka dan untuk mendapatkan akses terhadap suatu pintu dengan cara memindai kode QR dengan perangkat *mobile*. Diagram dari API verifikasi akses dapat dilihat pada Gambar 5.



Gambar 5 API Verifikasi Akses

3.2.5 Remote Pintu

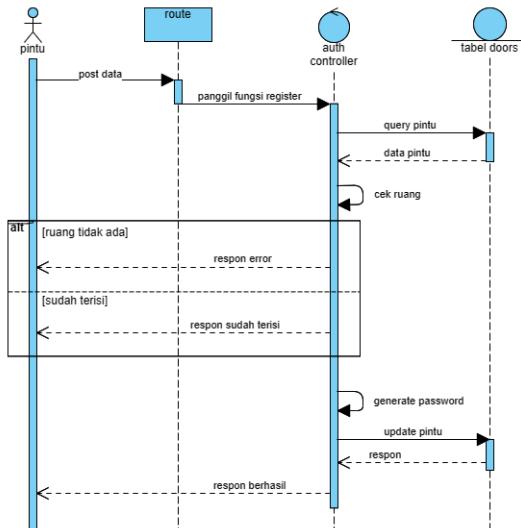
API *remote* pintu digunakan oleh operator untuk membuka atau mengunci pintu secara jarak jauh melalui aplikasi *mobile*. Dengan adanya fitur ini operator dapat mengendalikan pintu melalui aplikasi *mobile* dimana saja dan kapan saja tanpa harus berada di ruangan operasional dengan menggunakan komputer. Diagram dari API *remote* pintu dapat dilihat pada Gambar 6.



Gambar 6 API Remote Pintu

3.2.6 Door Register

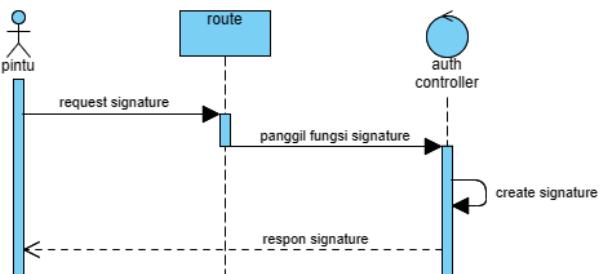
API *door register* digunakan untuk menambahkan perangkat penguncian yang baru kedalam pintu. Pada saat operator menambahkan pintu baru melalui *dashboard website* operator maka pintu tersebut belum terpasang perangkat kunci pintu sehingga harus ditambahkan secara manual melalui prosedur pendaftaran. Diagram dari API *door register* dapat dilihat pada Gambar 7.



Gambar 7 API Door Register

3.2.7 Door Signature

API *door signature* digunakan untuk mendapatkan kode unik yang digunakan untuk melakukan *subscribe* ke *channel* pusher. 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. Setelah mendapatkan nilai *signature* kemudian kontroler akan megembalikan respon kode *signature* ke perangkat kunci pintu. Diagram dari API *door signature* dapat dilihat pada Gambar 8.

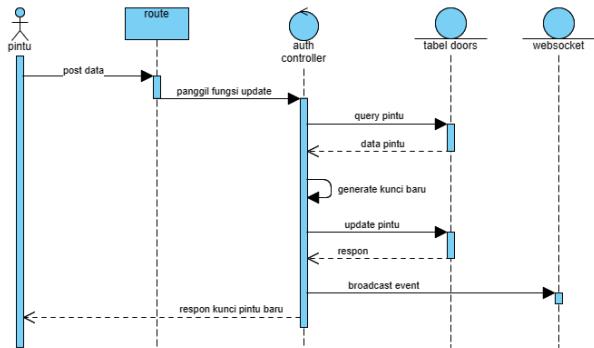


Gambar 8 API Door Signature

3.2.8 Door Update Status

API *door update status* digunakan oleh perangkat kunci pintu untuk memperbarui status pintu seperti pintu terbuka, pintu terkunci atau pintu terkoneksi. Pada setiap proses update ini kunci pintu juga akan diperbarui sehingga meningkatkan keamanan karena kode kunci

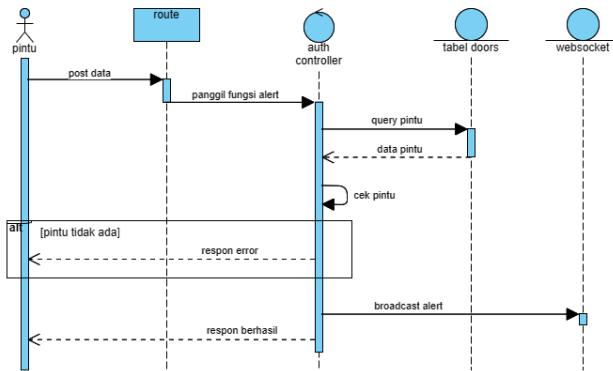
selalu berubah secara dinamis. Diagram dari API door update status dapat dilihat pada Gambar 9.



Gambar 9 API Door Update Status

3.2.9 Door Alert

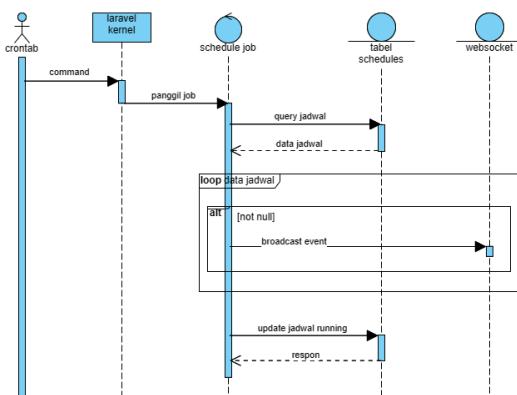
API door alert digunakan oleh perangkat kunci pintu untuk memberikan peringatan kepada operator bahwa pintu dalam kondisi yang tidak aman seperti terbuka tanpa autentikasi yang sah. Diagram dari API door alert dapat dilihat pada Gambar 10.



Gambar 10 API Door Alert

3.3 Penjadwalan

Proses penjadwalan dilakukan dengan cara memeriksa data jadwal pada tabel schedules setiap satu menit sekali. *Kernel* akan menjalankan *job* setiap satu menit sekali untuk memeriksa jadwal, jika ada jadwal yang harus dilaksanakan seperti membuka pintu atau mengunci pintu maka perintah akan dikirimkan ke perangkat kunci pintu melalui websocket. Diagram dari pengecekan jadwal dapat dilihat pada Gambar 11.

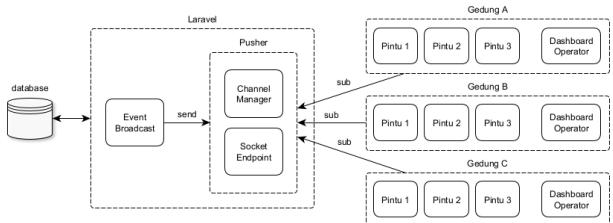


Gambar 11 Proses Penjadwalan

3.4 Komunikasi Websocket

Websocket digunakan sebagai jalur komunikasi yang menghubungkan antara *server* dengan perangkat kunci pintu, dengan adanya komunikasi websocket maka perangkat kunci pintu dan *server* akan selalu terhubung sehingga dapat berkomunikasi secara langsung.

Pada perancangan *backend server* untuk mendukung kinerja perangkat kunci pintu ini menggunakan pusher sebagai protokol komunikasi websocket yang menghubungkan antara perangkat kunci pintu dengan *server* dengan konfigurasi seperti yang terlihat pada Gambar 12.

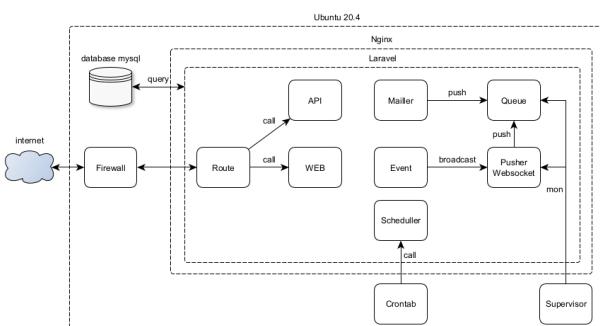


Gambar 12 Komunikasi Websocket

Setiap pintu akan dikelompokkan berdasarkan dengan gedung dengan satu orang operator, setiap gedung akan memiliki satu channel pusher yang dapat *subscribe* oleh perangkat penguncian didalam gedung tersebut, untuk mengawasi semua aktifitas yang terhubung ke channel dengan mudah maka konfigurasi *channel* menggunakan *presence channel*, dengan menggunakan *presence channel* maka aktifitas semua perangkat kunci pintu seperti perangkat kunci melakukan *subscribe* dan koneksi terputus dapat diketahui secara langsung.

3.5 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 13.



Gambar 13 Konfigurasi Server

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. Didalam sistem operasi ubuntu 20.04 dipasang nginx yang digunakan sebagai *web server* untuk menjalankan aplikasi laravel. 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.

IV. Hasil dan Pembahasan

4.1 Pengujian

Pengujian dilakukan untuk memeriksa hasil dari perancangan dan untuk memastikan semua bagian berfungsi secara baik.

Tabel 1 Pengujian Login

Nama	Pengujian	Respon
<i>Login</i> dengan data benar	Melakukan <i>login</i> menggunakan email dan <i>password</i> yang sesuai	<i>success</i>
<i>Login</i> dengan data salah	Melakukan <i>login</i> dengan menggunakan email atau <i>password</i> salah	<i>failed</i>
<i>Login</i> dengan data kurang	Melakukan <i>login</i> dengan menggunakan email saja atau <i>password</i> saja	<i>missing_parameter</i>
<i>Login</i> dengan data tidak terdaftar	Melakukan <i>login</i> dengan menggunakan email yang belum terdaftar	<i>missing_parameter</i>
<i>Login</i> dengan format tidak sesuai	Melakukan <i>login</i> dengan menggunakan <i>username</i> bukan email	<i>missing_parameter</i>
<i>Login</i> dengan email belum terverifikasi	Melakukan <i>login</i> dengan menggunakan email yang belum berlum terverifikasi	<i>email_unverified</i>

Proses *login* akan berhasil jika menggunakan email dan *password* yang sesuai, proses *login* juga memastikan semua parameter yang digunakan pada autentifikasi

tersedia dan juga sesuai. Pada proses pengujian menggunakan email yang belum terverifikasi *login* akan tertahan dengan status *email_unverified* dan menunggu *client* untuk melakukan verifikasi email.

Tabel 2 Pengujian Verifikasi Email

Nama	Pengujian	Respon
Verifikasi email tanpa token	Melakukan verifikasi email tanpa menggunakan token	<i>Unauthenticated</i>
Verifikasi email tidak sesuai	Melakukan verifikasi email menggunakan kode OTP yang salah	<i>otp_not_match</i>
Verifikasi email kadaluarsa	Melakukan verifikasi email menggunakan kode OTP yang sudah kadaluarsa	<i>otp_expired</i>
Verifikasi email sesuai	Melakukan verifikasi email menggunakan kode OTP yang sesuai	<i>success</i>
Verifikasi email token salah	Melakukan verifikasi email menggunakan token yang salah	<i>Unauthenticated</i>

Proses verifikasi email hanya berhasil jika client mengirimkan kode OTP yang sesuai disertai dengan token yang sesuai. Jika proses verifikasi email menggunakan kode OTP yang salah atau sudah kadaluarsa maka proses verifikasi email akan gagal.

Tabel 3 Pengujian Verifikasi Akses

Nama	Pengujian	Respon
Verifikasi akses tanpa token	Melakukan <i>request</i> lihat verifikasi akses tanpa menggunakan token	<i>Unauthenticated</i>

Lanjutan Tabel 3 Pengujian Verifikasi Akses

Nama	Pengujian	Respon
Verifikasi akses dengan token	Melakukan <i>request</i> verifikasi akses menggunakan token yang sesuai	<i>success</i>
Verifikasi akses dengan id pintu salah	Melakukan <i>request</i> verifikasi akses menggunakan identitas pintu yang salah	<i>no_data</i>

Proses verifikasi akses berhasil jika permintaan disertai dengan token dan identitas pintu sesuai, jika identitas pintu tidak sesuai maka proses verifikasi akses akan gagal karena pintu tidak ditemukan. Jika permintaan tidak disertai dengan token maka permintaan tersebut akan ditolak.

Tabel 4 Pengujian *Signature*

Nama	Pengujian	Respon
<i>Signature</i> tanpa token	Melakukan <i>request signature</i> tanpa menggunakan token	<i>Unauthenticated</i>
<i>Signature</i> dengan token	Melakukan <i>request signature</i> menggunakan token dan data lengkap	<i>success</i>
<i>Signature</i> data kurang	Melakukan <i>request signature</i> menggunakan data yang kurang	<i>missing_parameter</i>

Proses fungsi *signature* berhasil jika permintaan yang dikirimkan disertai dengan token dan data yang dikirimkan lengkap, jika permintaan yang dikirimkan tanpa menggunakan token atau ada data yang kurang maka permintaan akan gagal.

Tabel 5 Pengujian Websocket

Nama	Pengujian	Respon
<i>Subscrition</i> dengan <i>signature</i>	Melakukan <i>subscrition</i> ke <i>channel</i> websocket menggunakan <i>signature</i> sesuai	<i>Subscrition Succeed</i>

Lanjutan Tabel 5 Pengujian Websocket

Nama	Pengujian	Respon
<i>Subscrition</i> tanpa <i>signature</i>	Melakukan <i>subscrition</i> ke <i>channel</i> websocket tanpa menggunakan <i>signature</i>	<i>Invalid Signature</i>
<i>Subscrition</i> dengan <i>signature</i> salah	Melakukan <i>subscrition</i> ke <i>channel</i> websocket menggunakan <i>signature</i> yang salah	<i>Invalid Signature</i>

Proses *subscrition* pada *channel* websocket berhasil jika menggunakan kode *signature* yang sesuai, jika tidak menggunakan kode *signature* atau menggunakan kode *signature* yang salah maka proses *subscription* akan gagal.

Tabel 6 Pengujian *Update Status Pintu*

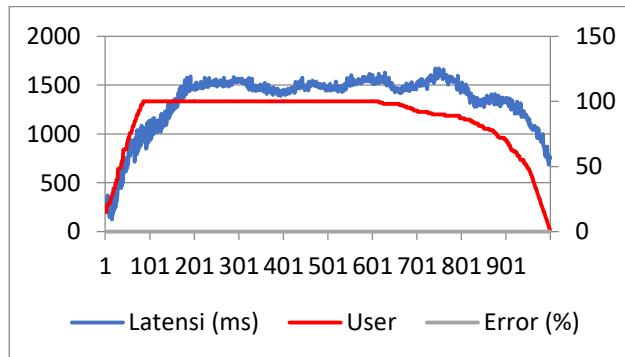
Nama	Pengujian	Respon
<i>Update</i> status tanpa token	Melakukan <i>update</i> status pintu tanpa token	<i>Unauthenticated</i>
<i>Update</i> status dengan token	Melakukan <i>update</i> status pintu dengan menggunakan token dan data lengkap	<i>success</i>
<i>Update</i> status data tidak lengkap	Melakukan <i>update</i> status pintu dengan menggunakan data yang tidak lengkap	<i>missing_parameter</i>
<i>Update</i> status id pintu salah	Melakukan <i>update</i> status pintu menggunakan data identitas pintu yang salah	<i>failed</i>

Proses *update* status pintu berhasil jika request dikirimkan dengan token dan data yang lengkap, jika ada data yang kurang lengkap atau tidak disertai dengan token akan request tersebut akan gagal.

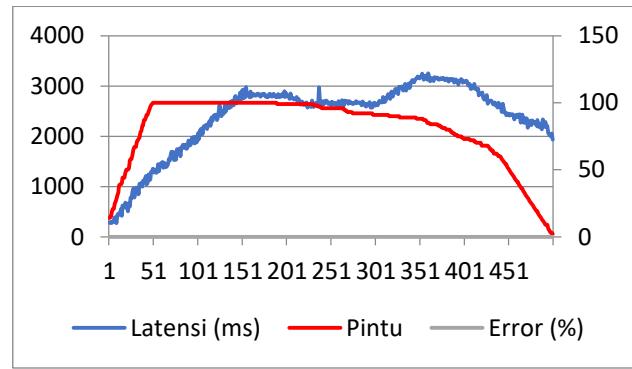
Tabel 7 Pengujian Peringatan Pintu

Nama	Pengujian	Respon
Peringatan pintu tanpa token	Mengirim peringatan pintu tanpa menggunakan token	<i>Unauthenticated</i>
Peringatan pintu sesuai	Mengirim peringatan pintu dengan menggunakan token dan data lengkap	<i>success</i>
Peringatan pintu data tidak lengkap	Mengirim peringatan pintu dengan menggunakan data yang tidak lengkap	<i>missing_parameter</i>
Peringatan pintu id pintu salah	Mengirim peringatan pintu menggunakan data identitas pintu yang salah	<i>failed</i>

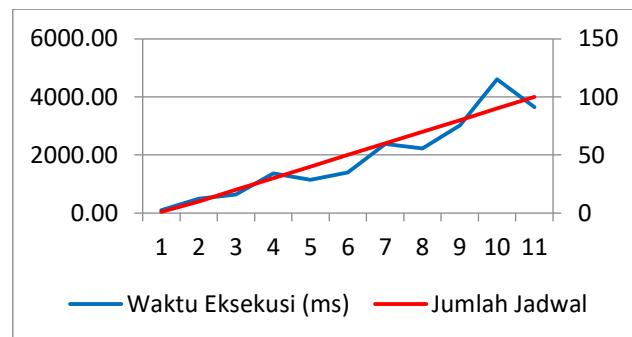
Proses peringatan pintu berhasil jika *request* dikirimkan dengan token dan data yang lengkap, jika ada data yang kurang lengkap atau tidak disertai dengan token akan request tersebut akan gagal.

**Gambar 14** Performa Verifikasi Akses

Pengujian performa verifikasi akses dilakukan dengan mensimulasikan beban permintaan verifikasi akses sebanyak 100 pengguna berbeda secara bersamaan. Dari hasil pengujian didapatkan bahwa pada puncak jumlah pengguna sistem membutuhkan waktu rata-rata 1.5 detik untuk memberikan respon ke pengguna dengan rasio error 0.0%.

**Gambar 15** Performa Update Status Pintu

Pengujian performa update *status* pintu dilakukan menggunakan Jmeter. Proses pengujian mendapatkan hasil bahwa pada beban 100 pintu berbeda melakukan update status secara bersamaan maka sistem memerlukan waktu rata-rata 2.5 detik dengan rasio *error* 0.0%.

**Gambar 16** Performa Penjadwalan

Pengujian dilakukan dengan mencatat waktu pemeriksaan untuk setiap jadwal, yaitu terdapat 11 titik pengujian dimulai dari 1 jadwal sampai 100 jadwal dengan masing-masing jadwal terdapat 20 pintu. Dapat dilihat pada Gambar 4.34 di atas, semakin besar jumlah jadwal yang ada maka waktu yang diperlukan pada proses penjadwalan akan semakin lama dimana untuk 1 jadwal memerlukan waktu 96.00 milidetik dan 100 jadwal memerlukan waktu 3641.14 milidetik. Dari hasil pengujian menggunakan 100 jadwal waktu yang diperlukan yaitu 3.6 detik dimana nilai tersebut masih di bawah dari periode pengecekan jadwal (1 menit) sehingga sistem masih dapat menangani 100 proses penjadwalan dengan aman.

4.2 Pembahasan

Pengembangan sistem *database* dan *server* pada sistem keamanan kunci pintu gedung dengan akses kontrol ini merupakan bagian dari pengembangan sistem penguncian gedung berbasis IoT yang dibangun dengan tujuan untuk meningkatkan keamanan dan efisiensi pengelolaan kunci pada sebuah gedung dimana didalam satu gedung tersebut terdapat banyak ruangan. Sistem yang dibangun dapat meningkatkan keamanan dan efisiensi penguncian dengan menggunakan beberapa fitur seperti

kode QR, kendali jarak jauh, peringatan penerobosan dan penjadwalan.

Perancangan *database* dan *server* pada sistem keamanan kunci pintu menggunakan metode *waterfall*, pengembangan dilakukan secara berurutan dimulai dari proses pengumpulan kriteria kebutuhan, implementasi sampai ke pengujian dan pengiriman. Sistem server dibangun menggunakan sistem operasi ubuntu 20.04 dengan *database* MySQL dan *backend* API Laravel. Penggunaan ubuntu sebagai *server* memberikan banyak kemudahan seperti adanya dukungan Cronjob sebagai mekanisme penjadwalan dan Nginx sebagai HTTP *server* yang ringan dan cepat. Laravel dan MySQL digunakan sebagai bagian dari *backend* dengan memanfaatkan beberapa fitur seperti autentikasi, penyiaran, penjadwalan yang disediakan oleh Laravel serta *database* relasional yang disediakan oleh MySQL. Pada tahap pengujian menggunakan alat pengujian seperti Jmeter dan Postman, sistem dapat memberikan kinerja yang diinginkan dan juga dapat menangani banyak pengguna secara bersamaan.

V. Penutup

5.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang telah dilakukan, dapat disimpulkan beberapa hal sebagai berikut :

1. Meskipun pada awalnya Laravel diperuntukan sebagai kerangka kerja pengembangan website menggunakan bahasa PHP, namun dengan berbagai fitur yang telah disediakan Laravel juga dapat digunakan sebagai kerangka kerja pengembangan *backend* API.
2. Penggunaan MySQL pada sistem kunci pintu didasarkan pada kebutuhan tempat penyimpanan data yang terstruktur dan cepat serta dukungan driver koneksi database pada kerangka kerja Laravel.
3. Server yang dikembangkan memiliki kemampuan untuk menyiarkan pesan secara langsung menggunakan komunikasi Websocket. Pesan akan disiarkan ke setiap channel menggunakan sebuah event.
4. Untuk meningkatkan kinerja dari server maka setiap aktifitas rutin dan aktifitas yang memerlukan waktu tunggu maka akan dimasukkan ke dalam antrian untuk dijalankan di latar belakang.
5. Nginx memiliki kemampuan untuk menjalankan lebih dari 1 aplikasi dalam satu server seperti pada sistem kunci pintu yang dikembangkan akan menjalankan API dan Websocket secara bersamaan dalam 1 server Nginx dengan menggunakan blok aplikasi yang telah diatur.

6. Proses penjadwalan akan dijalankan secara otomatis oleh Cronjob dengan menjalankan Job Laravel setiap 1 menit sekali untuk memeriksa data jadwal di dalam database.
7. API yang telah dikembangkan secara umum dapat menangani permintaan secara bersamaan dengan waktu respon sekitar 140 milidetik dengan rasio error 0%.
8. Pada endpoint API yang kemungkinan besar akan diakses secara bersamaan seperti verifikasi akses memerlukan waktu respon 1.5 detik pada beban 100 pengguna sedangkan untuk endpoint update status pintu memerlukan waktu 2.5 detik untuk beban 100 pintu.
9. Proses penjadwalan 100 jadwal pintu memerlukan waktu eksekusi sebesar 3.6 detik sehingga proses penjadwalan dapat berjalan dengan aman dengan periode pengecekan jadwal 1 menit sekali.

5.2 Saran

Setelah dilakukan proses perancangan dan pengujian didapatkan beberapa hal yang dapat diperbaiki atau ditingkatkan, yaitu :

1. Perlu adanya perancangan lebih lanjut untuk mempercepat respon *server* terutama pada beban *request* secara bersama-sama.
2. Diperlukan perancangan lebih lanjut untuk mengembangkan komunikasi Websocket dua arah yang bisa digunakan untuk mengirim dan menerima data.

DAFTAR PUSTAKA

- [1] K. Y. Sun, Y. Pernando, and M. I. Safari, “Perancangan Sistem IoT pada Smart Door Lock Menggunakan Aplikasi BLYNK,” *JUTSI (Jurnal Teknol. dan Sist. Informasi)*, vol. 1, no. 3, pp. 289–296, 2021, doi: 10.33330/jutsi.v1i3.1360.
- [2] F. As *et al.*, “Design and Construction of a Smart Door Lock With an Embedded Spy-Camera,” *J. Multidiscip. Eng. Sci. Technol.*, vol. 8, no. October, pp. 2458–9403, 2021, [Online]. Available: <https://www.researchgate.net/publication/354872757>
- [3] A. Jain, V. L. Kalyani, and B. Nogiya, “RFID and GSM Based Attendance Monitoring System using door locking / unlocking system and Its Hardware Implementation,” *J. Manag. Eng. Inf. Technol.*, vol. 2, no. 3, pp. 5–10, 2015.
- [4] M. Kasyful Anwar, “Perancangan Database IoT Berbasis Cloud dengan Restful API Cloud-Based IoT Database Design with Restful API,” vol. 20, no. 2, pp. 268–279, 2021.

- [5] S. Artikel, “Jurnal Nasional Teknologi dan Sistem Informasi Pembangunan Auto Backup SQL Database Server Menggunakan Raspberry Pi : Studi Kasus,” vol. 03, pp. 130–137, 2018.
- [6] Y. Efendi, “Internet Of Things (Iot) Sistem Pengendalian Lampu Menggunakan Raspberry Pi Berbasis Mobile,” *J. Ilm. Ilmu Komput.*, vol. 4, no. 2, pp. 21–27, 2018, doi: 10.35329/jiik.v4i2.41.
- [7] Nirsal, Rusmala, and Syafriadi, “Desain Dan Implementasi Sistem Pembelajaran Berbasis E-Learning Pada Sekolah Menengah Pertama Negeri 1 Pakue Tengah,” *J. Ilm. d'Computare*, vol. 10, pp. 30–37, 2020, [Online]. Available: <http://www.elsevier.com/locate/scp>
- [8] D. Purnama Sari and R. Wijanarko, “Implementasi Framework Laravel pada Sistem Informasi Penyewaan Kamera (Studi Kasus di Rumah Kamera Semarang),” *J. Inform. dan Rekayasa Perangkat Lunak*, vol. 2, no. 1, p. 32, 2020, doi: 10.36499/jinrpl.v2i1.3190.
- [9] S. Kosasi, I. D. Ayu, E. Yuliani, and G. Syarifudin, “Implementasi Arsitektur Model View Controller pada Website Toko Online Implementation of Model View Controller Architecture on Online Store Website,” vol. 3, no. 2, pp. 135–150, 2021, doi: 10.30812/bite.v3i2.1566.
- [10] A.-I. A.B., “Implementasi Teknologi Websocket dalam Pengembangan Sistem Berbagi Lokasi Berbasis Web,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 9, pp. 950–959, 2017, [Online]. Available: <http://j-ptiik.ub.ac.id>
- [11] C. Bestari Gea, K. Juri Damai Lase, M. Syamsudin, P. Studi Informatika, F. Sains dan Komputer, and U. Kristen Immanuel Yogyakarta, “Implementasi Virtual Private Server untuk Mini Hosting,” *J. InFact Sains dan Komput.*, vol. 7, no. 02, pp. 5–9, 2023.
- [12] C. Asiminidis, G. Kokkonis, and S. Kontogiannis, “Database Systems Performance Evaluation for IoT Applications,” *SSRN Electron. J.*, no. November 2019, 2019, doi: 10.2139/ssrn.3360886.

BIODATA



Muhammad Khoiril Wafi Lahir di Kabupaten Demak, 4 Maret 2001. Telah menempuh pendidikan mulai dari SD Negeri Karangrejo 2 selama 6 tahun, SMP Negeri 1 Bonang selama 3 tahun dan SMK Negeri 2 Demak selama 3 tahun. Saat ini penulis sedang menyelesaikan pendidikan S1-di Fakultas Teknik Universitas Diponegoro, Konsentrasi Teknologi Informasi angkatan 2019.

Teknik Elektro

Diponegoro, Konsentrasi Teknologi Informasi angkatan

2019.

Saya menyatakan bahwa segala informasi yang tersedia di makalah ini adalah benar, merupakan hasil karya sendiri, bebas dari plagiat dan semua karya orang lain telah dikutip dengan benar.

Muhammad Khoiril Wafi

NIM. 21060119140133

Telah disetujui untuk diajukan pada Seminar Tugas Akhir
Semarang, 6 September 2023

Pembimbing 1

Pembimbing 2

M.Arfan, S.Kom., M.Eng.
NIP. 198408172015041002

Imam Santoso, S.T., M.T.
NIP. 197012031997021001