TUGAS AKHIR K141502

RANCANG BANGUN APLIKASI PUSH NOTIFICATION TERPUSAT

DEWANGGA OKTA WAHYUDIANTO
NRP 05111440000005

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Fajar Baskoro, S.Kom., MT.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR KI1502

RANCANG BANGUN APLIKASI PUSH NOTIFICATION TERPUSAT

DEWANGGA OKTA WAHYUDIANTO
NRP 05111440000005

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Fajar Baskoro, S.Kom., MT.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS KI1502

DESIGN AND IMPLEMENTATION OF CENTRALIZED PUSH NOTIFICATION APPLICATION

DEWANGGA OKTA WAHYUDIANTO
NRP 05111440000005

Supervisors
Rizky Januar Akbar, S.Kom., M.Eng.
Fajar Baskoro, S.Kom., MT.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI PUSH NOTIFICATION TERPUSAT

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Rekayasa Perangkat Lunak
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
DEWANGGA OKTA WAHYUDIANTO
NRP : 05111440000005

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Rizky Januar Akbar, S.Kom., M.Eng. .....
NIP: 19870103 201404 1 001 (pembimbing 1)

Fajar Baskoro, S.Kom., M.T. .....
NIP: 19740403 199903 1 002 (pembimbing 2)

SURABAYA
JULI 2018

[Halaman ini sengaja dikosongkan]

RANCANG BANGUN APLIKASI PUSH NOTIFICATION TERPUSAT

Nama Mahasiswa	:	Dewangga Okta Wahyudianto
NRP	:	05111440000005
Jurusan	:	Departemen Informatika FTIK-ITS
Dosen Pembimbing 1	:	Rizky Januar Akbar, S.Kom., M.Eng.
Dosen Pembimbing 2	:	Fajar Baskoro, S.Kom., M.T.

ABSTRAK

Aplikasi push notification terpusat merupakan aplikasi yang digunakan untuk melakukan pengiriman pesan notifikasi dengan kapabilitas melakukan pemusatan pengiriman notifikasi bagi seluruh aplikasi yang ada di Institut Teknologi Sepuluh Nopember Surabaya. Aplikasi ini mampu untuk mengirim notifikasi pada jenis perangkat bergerak Android maupun Apple. Pengiriman notifikasi diklasifikasikan berdasarkan jenis aplikasi, role dan unit pengguna. Aplikasi push notification terpusat juga memiliki kapabilitas untuk melakukan moderasi sebelum pesan notifikasi dikirim kepada pengguna, mengimplementasikan message queuing sebelum mengirim notifikasi untuk mengoptimalkan penggunaan sumber daya yang tersedia dan juga melakukan pengiriman notifikasi secara terjadwal.

Pada tugas akhir ini dibuat dua aplikasi. Aplikasi pertama adalah aplikasi push notification terpusat yang berfungsi sebagai web service yang dibuat menggunakan bahasa pemrograman java dengan kerangka kerja maven. Aplikasi ini mengimplementasikan pustaka jersey restful web services. Aplikasi kedua adalah aplikasi pengelola konten notifikasi yang berfungsi untuk mengelola aplikasi utama. Aplikasi penunjang dibuat dengan kerangka kerja Codeigniter.

Pengiriman notifikasi menuju pengguna menggunakan backend as a service milik google yaitu FCM dan apple APNs. Uji coba pada tugas akhir ini meliputi pengujian fungsionalitas keseluruhan aplikasi berdasarkan kasus penggunaan. Dari hasil uji coba, seluruh fungsionalitas aplikasi dapat dipenuhi.

Kata kunci: *Push Notification, Google FCM, Java, Android, Scheduler, Message Queuing*

DESIGN AND IMPLEMENTATION OF CENTRALIZED PUSH NOTIFICATION APPLICATION

Name	:	Dewangga Okta Wahyudianto
NRP	:	05111440000005
Major	:	Informatics Department FTIK-ITS
Supervisor I	:	Rizky Januar Akbar, S.Kom., M.Eng.
Supervisor II	:	Fajar Baskoro, S.Kom., M.T.

ABSTRACT

Centralized Push Notification Application is an application which is made to manage the distribution of notification messages in information system environments owned by Institut Teknologi Sepuluh Nopember Surabaya. this application is designed to execute notification broadcasting to both Android and Apple mobile devices. Notification broadcasts are classified by application target type, user role and user unit. This application is also capable for doing pre-distribution notification moderation, manage notification distribution floods with message queuing and doing scheduled noitification broadcast that executed by a scheduler system inside.

There are two parts of applications in this final project. The main part of this system is centralized push notification application which is made as a web service that implemented with java programming language and maven framework, accompanied by jersey restful web services library.

The second one is notification content management application, which is made as a management interface to manage the main application. This application is made with codeigniter framework.

Google's FCM and Apple's APNS backend as a service is utilized by this application to transmit the notification broadcasts to the target users. Evaluation of this final project application implementation includes a functionality testing of its features based on use case specifications. The result of this evaluation indicates that this application succeeded to fulfill its functional requirements.

.

Keyword: *Push Notification, Google FCM, Java, Android, Scheduler, Message Queuing*

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul:

RANCANG BANGUN
APLIKASI PUSH NOTIFICATION TERPUSAT

Melalui lembar ini, penulis ingin menyampaikan ucapan terimakasih dan penghormatan yang sebesar-besarnya kepada:

1. Allah SWT atas segala nikmat dan rahmat yang telah diberikan selama ini.
2. Keluarga penulis khususnya orang tua penulis yang tiada henti-hentinya mencurahkan kasih sayang, perhatian dan doa kepada penulis selama ini.
3. Bapak Rizky Januar Akbar, S.Kom, M.Eng. selaku dosen pembimbing I yang selalu memberikan motivasi dan membimbing penulis selama pengerjaan tugas akhir.
4. Bapak Fajar Baskoro, S.Kom, M.T. selaku dosen pembimbing II yang telah memberikan nasihat, arahan, dan bantuan sehingga penulis dapat menyelesaikan tugas akhir ini.
5. Bapak dan Ibu dosen Teknik Informatika ITS yang telah membina dan memberikan ilmu kepada penulis selama menempuh studi di Teknik Informatika ITS.
6. Ary Kusuma Wardhani yang senantiasa menemani dan memotivasi penulis, serta pihak lain yang namanya tidak dapat penulis sebutkan satu-persatu.
7. Teman-teman angkatan 2014 khususnya Alfitrah N. S. yang telah memberikan semangat, serta berbagi ilmu selama penulis berkuliah di Informatika ITS.

Penulis menyadari sepenuhnya bahwa tugas akhir ini masih memiliki kekurangan. Oleh karena itu, dengan tangan terbuka, penulis menerima segala saran dan kritik dari pembaca untuk perbaikan ke depannya.

Surabaya, Juli 2018

Dewangga Okta Wahyudianto

DAFTAR ISI

{Halaman ini sengaja dikosongkan}

DAFTAR GAMBAR

DAFTAR TABEL

{Halaman ini sengaja dikosongkan}

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Institut Teknologi Sepuluh Nopember Surabaya (ITS) memiliki banyak aplikasi sistem informasi untuk mendukung berbagai kegiatan pada lingkungan internal kampus. Target pengguna sistem informasi yang digunakan di ITS adalah dosen, mahasiswa, orang tua atau wali, tenaga kependidikan dan alumni.

Aplikasi-aplikasi yang ada di ITS belum terintegrasi antara satu dengan yang lain. Seperti dalam kasus pengiriman notifikasi, pada setiap aplikasi yang ada saat ini masih menerapkan mekanisme pengiriman notifikasinya sendiri-sendiri. Hal tersebut menyebakan perawatan dan pengembangan aplikasi kedepannya menjadi susah.

Untuk mengatasi permasalahan pengiriman notifikasi yang belum terintegrasi, dibuatlah aplikasi *push notification* terpusat. Aplikasi ini memiliki kapabilitas untuk mengirim notifikasi menuju pengguna yang berbeda-beda pada lingkungan ITS dengan data yang terintegrasi. Sehingga mempermudah perawatan serta pengembangan aplikasi untuk kedepannya.

Aplikasi *push notification* terpusat dibuat menggunakan bahasa pemrograman *java*. REST juga digunakan sebagai mekanisme interaksi aplikasi *push notification* terpusat dengan aplikasi klien. Selain itu, aplikasi ini juga dapat melakukan pengiriman notifikasi secara terjadwal (*scheduling*) serta mengimplementasikan mekanisme *message queuing* pengiriman notifikasi menuju BaaS (Backend as a Service) secara *asynchronous*.

Pengiriman notifikasi menuju target pengguna yang berbeda dicontohkan dalam beberapa kasus sebagai berikut.

1. Notifikasi ajakan untuk mengikuti acara kuliah tamu yang diadakan di aula Departemen Informatika hanya diterima pengguna dengan peran sebagai mahasiswa dan unit S1 Informatika.
2. Notifikasi pengisian kuesioner mata kuliah diterima oleh seluruh pengguna dengan peran mahasiswa.
3. Notifikasi batas terakhir pengisian nilai pada aplikasi Integra hanya diterima pengguna dengan peran dosen.

Selain itu terdapat juga aplikasi kedua berbasis web dengan nama aplikasi pengelola konten notifikasi yang berfungsi untuk melakukan konfigurasi dan mengelola aplikasi *push notification* terpusat. Aplikasi ini dibuat menggunakan kerangka kerja CodeIgniter dengan struktur MVC.

1.2 Rumusan Masalah

Berdasarkan latar belakang, terdapat beberapa rumusan masalah yang terdapat pada tugas akhir ini, antara lain adalah sebagai berikut.

1. Bagaimana implementasi REST API untuk Aplikasi *Push Notification* Terpusat?
2. Bagaimana implementasi *queuing* pada Aplikasi *Push Notification* Terpusat?
3. Bagaimana implementasi segmentasi pengiriman notifikasi pada Aplikasi *Push Notification* Terpusat?
4. Bagaimana implementasi *scheduling* pada Aplikasi *Push Notification* Terpusat?
5. Bagaimana implementasi pengiriman data pesan notifikasi secara *asynchronous* pada BaaS (Backend as a Service) APNs dan Firebase?
6. Bagaimana implementasi Aplikasi Pengelola Konten Notifikasi?

1.3 Batasan Masalah

Batasan masalah yang terdapat pada tugas akhir ini adalah sebagai berikut.

1. Aplikasi *Push Notification* Terpusat digunakan khusus untuk ruang lingkup ITS.
2. Aplikasi *Push Notification* Terpusat digunakan untuk mempermudah penyebaran informasi melalui perangkat bergerak (Android dan iOS).
3. Teknologi yang digunakan dalam pembuatan aplikasi ini adalah bahasa pemrograman web (PHP, HTML, CSS JavaScript), Java, kerangka kerja Model-View-Controller, Jax RS web service REST API.
4. Aplikasi ini harus membatasi akses konten notifikasi pada level pengguna yang berbeda.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah membuat sebuah aplikasi pengiriman *push notification* terpusat kepada pengguna dengan jenis yang berbeda-beda pada Institut Teknologi Sepuluh Nopember.

1.5 Manfaat

Aplikasi *Push Notification* Terpusat dan Aplikasi Pengelola Konten Notifikasi diharapkan dapat memudahkan penyebaran informasi di lingkungan ITS sebagai pengganti media cetak. Aplikasi *Push Notification* Terpusat menyediakan web service dari berbagai aplikasi yang ada di ITS (seperti myITS) untuk melakukan *broadcast* notifikasi ke target pengguna yang berbeda sesuai level yang telah ditentukan pada Tabel 1. Aplikasi Pengelola Konten Notifikasi dapat menjadi tempat menyimpan konten notifikasi mana saja yang akan di-*broadcast* sehingga tugas akhir ini diharapkan menjadi sebuah sistem yang menjadi pendukung media penyebaran informasi di lingkungan ITS secara mudah, cepat, dan *realtime* serta dapat diintegrasikan dengan berbagai jenis sistem yang ada.

1.6 Metodologi Pembuatan Tugas Akhir

Adapun beberapa tahap dalam proses pengerjaan tugas akhir ini, yaitu sebagai berikut:

1. Penyusunan proposal tugas akhir

Tahap pertama dalam proses pengerjaan tugas akhir ini adalah menyusun proposal tugas akhir.

2. Studi literatur

Pada tahap ini, akan dicari studi literatur yang relevan untuk dijadikan referensi dalam pengerjaan tugas akhir.

3. Analisis dan desain perangkat lunak

Tahap ini meliputi perumusan kebutuhan fungsional, kebutuhan non-fungsional, kasus penggunaan, diagram aktivitas, diagram kelas, dan diagram sekuens.

4. Implementasi

Aplikasi ini diimplementasikan dengan menggunakan kakas bantu:

1. Basis data yang digunakan adalah Microsoft SQL Server.
2. Aplikasi android dibuat dengan spesifikasi minimal versi 4.0 (*Ice Cream Sandwich*).
3. Pengerjaan aplikasi android mengunakan IDE Android Studio.
4. Bahasa pemrograman untuk pembuatan aplikasi pengelola konten notifikasi adalah PHP.
5. Kerangka kerja yang digunakan untuk membuat aplikasi pengelola konten notifikasi adalah CodeIgniter versi 3.1.7.
6. Bahasa pemrograman untuk pembuatan *web service* adalah Java.

7. Pembuatan aplikasi pengelola konten notifikasi menggunakan IDE Sublime Text.
8. Kerangka kerja yang digunakan untuk pembuatan *web service* adalah Maven dengan archetype Jersey versi 2.16.
9. Pembuatan *web service* menggunakan IDE Eclipse Kepler Service Release 2.

5. Uji coba dan evaluasi

Model pengujian dibagi menjadi dua bagian. Model pengujian pertama adalah pengujian fungsionalitas untuk aplikasi *push notification* terpusat dan aplikasi pengirim konten notifikasi. Model pengujian kedua adalah pengujian REST API. Pengujian pada REST API dilakukan untuk menguji kebenaran proses API dengan aplikasi *Postman*. Skenario pengujian yang dilakukan berdasarkan data yang ada pada data *my.its.ac.id*. Setiap fungsionalitas dari aplikasi ini akan diujikan dan dievaluasi berdasarkan berhasil atau tidaknya pengguna melakukan fungsionalitas tersebut.

6. Penyusunan buku tugas akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini. Pada tahap ini juga disertakan hasil dari implementasi perangkat lunak yang telah dibuat. Sistematika penulisan buku tugas akhir ini secara garis besar antara lain:

1. Pendahuluan
    a. Latar Belakang
    b. Rumusan Masalah
    c. Batasan Masalah
    d. Tujuan
    e. Manfaat
    f. Metodologi Pembuatan Tugas Akhir
    g. Sistematika Penulisan
2. Dasar Teori

3. Analisis dan Perancangan Sistem
4. Implementasi
5. Pengujian dan Evaluasi
6. Kesimpulan dan Saran
7. Daftar Pustaka

1.7 Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan
Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori
Memaparkan hasil studi literatur yang digunakan menyelesaikan Tugas Akhir ini, terdiri atas,

Bab III Analisis dan Perancangan Sistem
Membahas tentang analisis permasalahan, deskripsi umum sistem, spesifikasi kebutuhan perangkat lunak, lingkungan perancangan, perancangan arsitektur sistem, diagram kelas, dan struktur data.

Bab IV Implementasi
Bab ini berisi implementasi dari perancangan dan implementasi fitur-fitur penunjang aplikasi termasuk implementasi sistem dan implementasi antarmuka pengguna.

Bab V Pengujian dan Evaluasi
Bab ini membahas pengujian dengan metode kotak hitam (*black box testing*) untuk mengetahui aspek nilai fungsionalitas dari perangkat lunak dan nilai kegunaan yang dibuat dengan juga memperhatikan

ketertarikan pada calon partisipan untuk menggunakan aplikasi ini.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan pada Tugas Akhir ini. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar kode yang ada pada aplikasi ini.

{Halaman ini sengaja dikosongkan}

BAB II
DASAR TEORI

Pada Bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir ini. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap perangkat lunak yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Android

Android adalah sistem operasi yang dirancang untuk perangkat bergerak dengan layar sentuh seperti *smartphone* atau *tablet* dengan didukung kernel yang berbasis Linux. Android menjadi populer sejak dirilis pada tahun 2007 karena merupakan *platform* tak terbatas dimana aplikasi berbasis Android sangat banyak dan dapat diakses secara gratis [1].

2.2 Android Studio

Android Studio adalah platform kerja Android berupa *Integrated Development Environment* (IDE) untuk pengembangan aplikasi Android yang berbasis bahasa pemrograman java. Platform ini memiliki fitur yaitu sistem versi berbasis *gradle* yang fleksibel, emulator yang cepat dan teratur, dapat langsung *instant run* sehingga tidak perlu membuat apk baru, ada dukungan C++, NDK dan Google Cloud Platform untuk mempermudah pengintegrasian Google Cloud Messaging serta App Engine [2].

2.3 Java

Java adalah bahasa pemrograman berbasis objek atau disebut "OOP" yang merupakan singkatan dari *object oriented rogramming* atau pemrograman yang berorientasi pada objek. Bahasa ini merupakan adopsi dari bahasa pemrograman C dan C++ dimana sintaks pada Java dan kedua bahasa pemrograman ini hampir sama namun Java memiliki keunggulan yaitu dapat

dijalankan pada beberapa platform sistem operasi yang berbeda [3] [4].

2.4 PHP (Hypertext Preprocessor)

PHP merupakan bahasa pemrograman *server-side* yang digunakan untuk pengembangan *web* dan bahasa yang digunakan secara luas dalam pengembangan *website*. PHP dikembangkan pertama kali oleh Rasmus Lerdorf pada tahun 1995. Secara resmi dirilis pada tahun 1997. Fungsionalitas dasar pemrograman berorientasi objek ditambahkan pada PHP 3 dan ditingkatkan pada PHP 4. PHP telah digunakan lebih dari 200 juta *website* yang berbeda, dari *website* personal biasa sampai perusahaan raksasa seperti Facebook, Wikipedia, Tumblr, Slack dan Yahoo.

PHP [5] didesain untuk membuat halaman *web* yang dinamis. PHP berguna sebagai bahasa pemrograman scripting yang dikhususkan untuk pengembangan *web server-side*. PHP memiliki peran utama sebagai filter, yaitu mengambil *file* atau *stream* yang mengandung teks dan atau instruksi PHP dan menghasilkan data *stream* lain. PHP dijalankan pada komputer untuk digunakan oleh seorang pengguna, biasanya dijalankan pada *web server* dan diakses oleh banyak pengguna yang menggunakan *web browser*. PHP memiliki kelebihan yaitu menyembunyikan kompleksitas.

2.5 Kerangka Kerja CodeIgniter

CodeIgniter adalah sebuah kerangka kerja berbasis PHP yang dibangun dengan konsep MVC (Model, View, Controller) yang memiliki keunggulan yaitu memiliki *loading time* yang ringan, tersedianya sintaks yang ekspresif dan jelas sehingga dapat menghemat waktu, biaya awal, biaya pemeliharaan dan dapat meningkatkan kualitas perangkat lunak [6] [7]. Pada tugas akhir ini, PHP dengan kerangka kerja CodeIgniter akan digunakan untuk membangun aplikasi pengelola konten notifikasi.

2.6 SQL Server

Microsoft SQL Server adalah sebuah sistem manajemen basis data relasional (RDBMS) produk Microsoft. Bahasa kueri utamanya adalah Transact-SQL yang merupakan implementasi dari SQL standar ANSI/ISO yang digunakan oleh Microsoft dan Sybase. Umumnya SQL Server digunakan di dunia bisnis yang memiliki basis data berskala kecil sampai dengan menengah, tetapi kemudian berkembang dengan digunakannya SQL Server pada basis data besar.

Microsoft SQL Server dan Sybase/ASE dapat berkomunikasi lewat jaringan dengan menggunakan protokol TDS (Tabular Data Stream). Selain itu, Microsoft SQL Server juga mendukung ODBC (Open Database Connectivity), dan mempunyai driver JDBC untuk bahasa pemrograman Java. Fitur yang lain dari SQL Server ini adalah kemampuannya untuk membuat basis data *mirroring* dan *clustering* [8] [9].

2.7 REST Web Service

REST (Representational State Transfer) merupakan standar arsitektur komunikasi berbasis web yang sering diterapkan dalam pengembangan layanan berbasis web. Umumnya menggunakan HTTP (*Hypertext Transfer Protocol*) sebagai protocol untuk komunikasi data. Pada arsitektur REST, REST server menyediakan resources (sumber daya/data) dan REST client mengakses dan menampilkan resource tersebut untuk penggunaan selanjutnya. Setiap resource diidentifikasi oleh URIs (Universal Resource Identifiers) atau global ID. Resource tersebut direpresentasikan dalam bentuk format teks, JSON atau XML. Pada umumnya format yang digunakan, berupa JSON dan XML [10].

Arsitektur REST yang *decoupled* (terpisah) serta beban komunikasi yang ringan antara produsen dan konsumen membuatnya populer di dunia *cloud-based* API, seperti yang disajikan oleh Amazon, Microsoft, dan Google. Layanan

berbasis *web* yang menggunakan arsitektur REST semacam itu dinamakan RESTful APIs (*Application Programming Interfaces*) atau REST APIs [11].

Tabel 2.1 Metode HTTP dan Penggunaannya dalam REST

Metode	Deskripsi
GET	Mendapatkan (*read*) sebuah sumber daya (*resource*) yang diidentifikasi dengan URI (*Uniform Resource Identifier*)
POST	Mengirimkan sumber daya (*resource*) ke server. Digunakan untuk membuat (*create*) sumber daya baru
PUT	Mengirimkan sumber daya (*resource*) ke server. Digunakan untuk memasukkan (*insert*) atau memperbarui (*update*) sumber daya yang tersimpan.
DELETE	Menghapus (*delete*) sumber daya (*resource*) yang diidentifikasi dengan URI

2.8 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) adalah solusi pengiriman pesan lintas platform yang memungkinkan untuk mengirimkan pesan dengan tepercaya tanpa biaya. FCM dapat dintegrasikan dengan sistem APNs (*Apple Push Notification service*) milik Apple. FCM memiliki kemampuan utama sebagai berikut [12].

1. **Mengirim pesan notifikasi atau pesan data**
   Mengirim pesan notifikasi yang ditampilkan kepada pengguna. Atau mengirim pesan data dan menentukan sepenuhnya apa yang terjadi dalam kode aplikasi.
2. **Penargetan pesan serbaguna**
   Mendistribusikan pesan ke aplikasi klien dengan salah satu dari tiga cara — ke satu perangkat, ke grup perangkat, atau ke perangkat yang berlangganan topik.
3. **Mengirim pesan dari aplikasi klien**

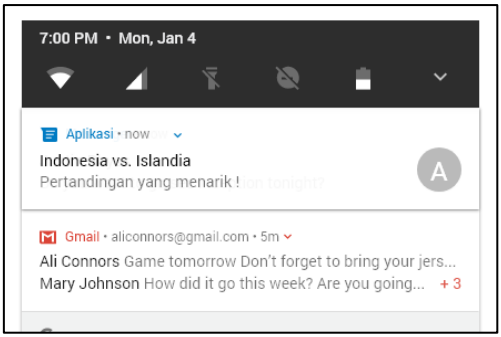
Mengirim notifikasi, *chat*, dan pesan lain dari perangkat ke server melalui saluran koneksi FCM yang andal dan hemat baterai.

Tabel 2.2 Alur Implementasi FCM

No.	Deskripsi
1.	**Menyiapkan FCM SDK**
	Menyiapkan *Firebase* dan FCM pada aplikasi sesuai petunjuk penyiapan untuk platform Android dan iOS.
2.	**Mengembangkan aplikasi klien**
	Menambahkan penanganan pesan, logika langganan topik, atau fitur opsional lain ke aplikasi klien. Selama pengembangan, dapat dilakukan pengujian untuk mengirim pesan dengan mudah menggunakan *Firebase Console*.
3.	**Mengembangkan server aplikasi**
	Tentukan apakah pengembangan nantinya menggunakan *Admin SDK* atau salah satu *server protocol* untuk membuat logika pengiriman, yaitu logika untuk mengautentikasi, membuat permintaan pengiriman, menangani respons, dan sebagainya.

Contoh kode program untuk mengirim pesan notifikasi dengan *FCM protocols* yang dibungkus dalam format JSON terdapat pada lampiran 1.

Dengan menggunakan kode di atas, *user* akan melihat notifikasi pada perangkat bergerak Android dengan pratinjau pada Gambar 2.1.



Gambar 2.1 Pratinjau Notifikasi pada Android

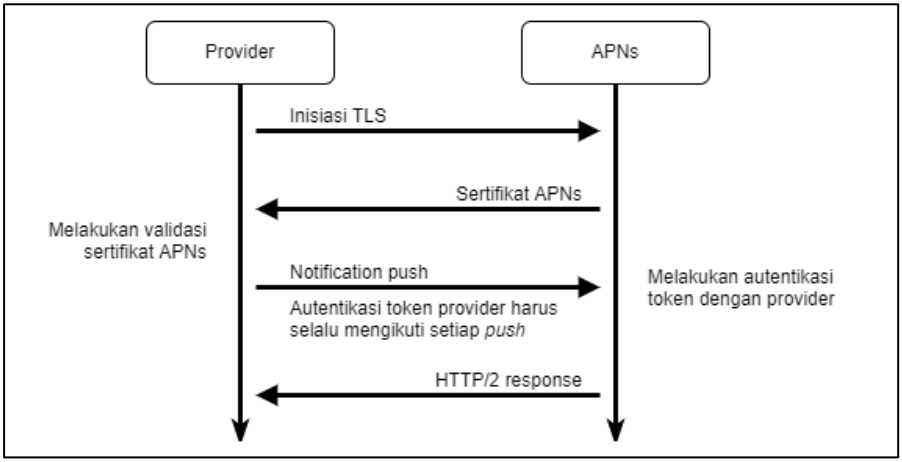
2.9 APNs (Apple Push Notification service)

APNs merupakan sebuah platform *notification service* yang dibuat oleh *Apple Inc*. yang memungkinkan aplikasi *third-party* milik pengembang untuk mengirim data notifikasi ke aplikasi yang ter-*install* pada perangkat-perangkat Apple. Informasi notifikasi yang dapat dikirim termasuk *badges*, suara, berita terbaru ataupun teks peringatan. Mac OS X API mendukung kapabilitas APNs untuk aplikasi-aplikasi lokal. Jumlah *payload* notifikasi yang dapat dikirim *binary interface* sejumlah 4 Kb. Pada APNs juga diwajibkan untuk menggunakan protokol TLS untuk mengirim notifikasi [13].

untuk dapat menerima dan mengontrol *remote notification*, aplikasi harus memenuhi syarat sebagai berikut.

1. Mengaktifkan kemampuan *push notifications* dengan cara memindah switch posisi *off* menjadi *on* pada jendela *Project editor* → *Target* → *Capabilities*.
2. Melakukan registrasi aplikasi kepada *APNs* dengan cara menerima token yang bersifat unik untuk tiap perangkat yang berbeda.
3. Sistem akan menampilkan perangkat ke aplikasi Anda dengan memanggil metode di delegasi aplikasi.

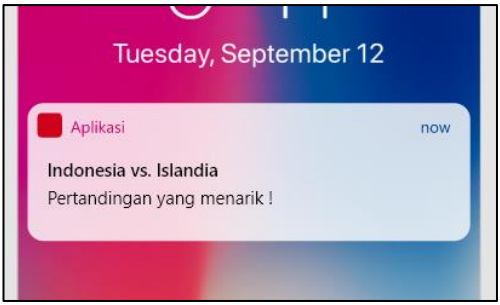
4. Aplikasi mengirimkan token perangkat ke penyedia yang terkait dengan aplikasi.



Gambar 2.2 Menetapkan Koneksi Provider Berbasis Token

Berikut adalah contoh kode program untuk mengirim pesan notifikasi dengan *APNs* yang dibungkus dalam format JSON. Dengan menggunakan kode di atas, *user* akan melihat notifikasi pada perangkat bergerak iOS dengan pratinjau pada Gambar 2.3.

```
{
    "aps" : {
        "alert" : {
            "title" : "Indonesia vs. Islandia",
            "body"      :      "Pertandingan      yang
menarik !",
        },
        "badge" : 5
    }
}
```



Gambar 2.3 Pratinjau Notifikasi pada iOS

2.10 Jersey RESTful Web Services

Jersey *RESTful Web Services* merupakan kerangka kerja *open source* untuk mengembangkan sebuah *RESTful Web Services* pada Java. Jersey menyediakan dukungan untuk JAX-RS API dan mengimplementasikan JAX-RS (JSR-311 & JSR 339) [14].

Jersey memiliki beberapa komponen sebagai berikut :

1. *Core Server*
   Komponen utama untuk membuat *RESTful services* berdasarkan anotasi (*jersey-core*, *jersey-server*, *jsr311-api*).
2. *Core Client*
   Komponen yang dibutuhkan agar bisa berkomunikasi dengan *REST services* (jersey-client).
3. *JAXB support*
   Komponen pendukung ini disediakan oleh Jersey yang digunakan untuk mengkonversi *Object* ke dalam format *XML* dan sebaliknya.
4. *JSON support*

Komponen pendukung ini disediakan oleh Jersey yang digunakan untuk mengkonversi Object ke dalam format *JSON* dan sebaliknya [15].

2.11 Message Queue

Message Queue merupakan metode dimana proses dapat bertukar data menggunakan atau melalui sebuah interface untuk sistem. *Message queue* merupakan sebuah *asynchronous communications protocol*. Sebuah pesan antrian dapat dibuat oleh satu proses dan digunakan oleh banyak proses yang membaca dan/atau menulis pesan ke antrean. Misalnya, server proses dapat membaca dan menulis pesan dari antrean dan ke pesan antrean dibuat untuk klien proses. Jenis pesan dapat digunakan untuk menghubungkan pesan dengan klien tertentu proses meskipun semua pesan pada antrean yang sama.

Pada sebuah implementasi *message queue* secara umum, sebuah administrator sistem melakukan instalasi dan konfigurasi sebuah perangkat lunak *message queue*. Umumnya disebut sebagai *queue manager* atau *broker*. Tabel 2.3 adalah tabel perbandingan sebuah *web service* jika tanpa menggunakan *message queue* dan dengan menggunakan *message queue* [16].

Tabel 2.3 Perbandingan Penggunaan *Message Queue*

Tanpa menggunakan *message queue*	**Dengan menggunakan *message queue***
Jika *server* gagal menerima request (*fails*) atau *down*. Klien harus turun tangan untuk mengatasi kesalahan yang terjadi.	Jika *server* gagal menerima request (*fails*) atau *down*. *Queue* tetap menyimpan pesan meskipun sistem target sedang mati.
Jika *server* telah normal kembali, klien bertanggung jawab untuk mengirim ulang pesn yang dikirimnya lagi.	Jika *server* telah normal kembali, *server* menerima pesan yang *pending* sebelumnya.

Jika *server* memberi tanggapan terhadap panggilan ke klien, tetapi klien gagal menerima panggilan dari server maka operasi dikatakan hilang begitu saja.	Jika *server* memberi tanggapan terhadap panggilan ke klien, tetapi klien gagal menerima panggilan dari server maka operasi yang dikirim oleh server tetap tersedia untuk dikirim ulang kembali.
Jika jutaan klien memanggil layanan web di satu server dalam satu detik, maka kemungkinan besar server akan *down*.	Jika jutaan klien memanggil layanan web di satu server dalam satu detik, maka dapat diputuskan seberapa banyak permintaan yang ditangani oleh *server* (*worker*).
Didapatkan tanggapan langsung dari server dan dapat menangani panggilan *asynchronous*.	Tidak didapatkan respons sinkron langsung, tapi dapat menerapkan atau mensimulasikan panggilan *synchronous*.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini dibahas analisis dan perancangan perangkat lunak untuk mencapai tujuan dari tugas akhir. Perancangan ini meliputi perancangan data, perancangan proses, dan perancangan antar muka, serta juga akan dijelaskan tentang analisis implementasi metode secara umum pada sistem.

3.1 Analisis

Pada subbab berikut akan dijelaskan analisis perangkat lunak untuk aplikasi *Push Notification* terpusat sebagai *RESTful Web Services* dan aplikasi pengelola konten notifikasi. Analisis yang dilakukan meliputi analisis permasalahan, kebutuhan umum perangkat lunak, deskripsi umum sistem, arsitektur dan kebutuhan fungsional sistem.

3.1.1 Analisis Permasalahan

Permasalahan utama yang diangkat pada pembuatan tugas akhir ini adalah karena tidak adanya sebuah *web service* untuk melakukan *broadcasting* push notifikasi di ITS secara terpusat dan terintegrasi untuk memudahkan pengembangan dan pengelolaannya. Selama ini pengiriman push notifikasi pada berbagai aplikasi yang ada di ITS masih terpisah secara sendiri-sendiri pada masing-masing aplikasi yang ada sehingga tidak dapat dilakukan monitoring secara terpusat.

3.1.2 Deskripsi Umum Sistem

Tugas akhir ini bertujuan untuk membuat Aplikasi *Push Notification* Terpusat dan Aplikasi Pengelola Konten Notifikasi yang nantinya diharapkan dapat memudahkan penyebaran informasi di lingkungan ITS sebagai pengganti media cetak. Dengan adanya aplikasi ini, diharapkan warga ITS tidak tertinggal dalam mendapatkan informasi secara mudah, cepat, dan

realtime. Aplikasi *push notification* terpusat dibuat menggunakan bahasa pemrograman *java*. REST juga digunakan sebagai mekanisme interaksi aplikasi *push notification* terpusat dengan aplikasi klien. Selain itu, aplikasi ini juga dapat melakukan pengiriman notifikasi secara terjadwal (*scheduling*) serta mengimplementasikan mekanisme *message queuing* pengiriman notifikasi menuju BaaS (Backend as a Service) secara *asynchronous*. Penjelasan secara detail untuk proses *scheduling* dan *message queue* dapat dilihat pada bagian perancangan proses halaman 98. Aplikasi ini juga dapat menerima konten notifikasi yang dikirim dari API yang telah dipasang pada aplikasi-aplikasi lain.

Aplikasi Pengelola Konten Notifikasi merupakan aplikasi yang digunakan oleh administrator myITS serta memiliki kemampuan untuk mengirimkan konten notifikasi yang akan di-*broadcast* kepada aplikasi Aplikasi *Push Notification* Terpusat. Aplikasi ini menyediakan konten notifikasi yang dapat berupa teks judul, teks konten, gambar, jenis notifikasi untuk ditampilkan di aplikasi myITS sesuai level pengguna maupun *client* yang berbeda.

3.1.3 Spesifikasi Kebutuhan Perangkat Lunak

Subbab ini membahas spesifikasi kebutuhan perangkat lunak dari hasil analisis yang telah dilakukan. Bagian ini berisi kebutuhan perangkat lunak yang direpresentasikan dalam bentuk kebutuhan fungsional, diagram kasus penggunaan, dan diagram aktivitas.

3.1.3.1 Aktor

Pengertian pengguna adalah pihak-pihak, baik manusia maupun sistem atau perangkat lain yang terlibat dan berinteraksi secara langsung dengan sistem. Aktor pada sistem ini akan dijelaskan pada Tabel 3.1.

Tabel 3.1 Aktor pada Sistem

Aktor	Tugas	Hak Akses ke aplikasi
Aplikasi *Push Notification* Terpusat		
Client (Aplikasi)	Melihat data dan mengirim packet untuk pengiriman notifikasi	1. Melakukan *register device* token 2. Melakukan penambahan data notifikasi yang akan dikirim 3. Mendapatkan data tujuan pengiriman notifikasi
Administrator	Mengelola sistem	4. Mendapatkan daftar status aktif *device token* 5. Melakukan perintah untuk menjalankan *scheduler* 6. Melakukan perintah untuk menjalankan *message queuing*
Aplikasi Pengelola Konten Notifikasi		
Administrator	Melakukan monitoring, moderasi, serta mengelola data notifikasi	1. Menampilkan data pengguna serta device token yang teregistrasi 2. Menampilkan daftar peran pengguna (*user role*) 3. Menampilkan daftar *unit pengguna* 4. Menampilkan daftar *client* 5. Mengirim notifikasi dengan antarmuka pengguna 6. Menampilkan hasil operasi *check device token* 7. Menampilkan halaman *packet monitoring* 8. Menampilkan daftar moderasi notifikasi 9. Menampilkan daftar pengguna 10. Melakukan manajemen klien

Aktor	Tugas	Hak Akses ke aplikasi
Aplikasi *Push Notification* Terpusat		
		11. Menampilkan halaman daftar akses token untuk aplikasi klien 12. Melakukan pengaturan tujuan notifikasi 13. Menampilkan halaman pengaturan 14. Melakukan manajemen sertifikat/key untuk BaaS (Backend as a Service)

3.1.3.2 Kebutuhan Fungsional

Kebutuhan fungsional mendefinisikan layanan yang harus dimiliki oleh perangkat lunak, reaksi dari perangkat lunak terhadap suatu masukan, hasil yang dilakukan perangkat lunak pada situasi khusus. Kebutuhan fungsional dari perangkat lunak untuk mengoperasikan komputer menggunakan suara dijelaskan pada Tabel 3.2.

Tabel 3.2 Kebutuhan Fungsional Sistem

No	Kebutuhan Fungsional	Deskripsi
Aplikasi *Push Notification* Terpusat		
F01	Melakukan registrasi *device token*	Kapabilitas aplikasi untuk melakukan registrasi *device token* untuk disimpan kedalam basis data sebagai parameter pengiriman *push notification*. Selain itu juga dapat melakukan pengecekan apakah *device token* telah teregistrasi ataupun belum pada basis data.
F02	Melakukan segmentasi pengiriman notifikasi	Kapabilitas aplikasi untuk melakukan pengiriman notifikasi kepada *user* dengan data *user* yang diambil dari basis data sso ITS (*my.its.ac.id*).

No	Kebutuhan Fungsional	Deskripsi
		Pengiriman notifikasi dibagi menjadi tiga jenis: 1. Single, menggunakan *username, client-id* sebagai parameter pengiriman notifikasi 2. Multiple, berdasarkan *user-id* atau *username* dan *client-id* sebagai parameter pengiriman notifikasi 3. Targeted, berdasarkan mapping *user role* dan *user unit*, *client-id* sebagai parameter pengiriman notifikasi
F03	Mendapatkan data tujuan pengiriman notifikasi	Kapabilitas aplikasi untuk memberikan daftar *role* yang dapat digunakan oleh *client* sebagai parameter pengiriman notifikasi.
F04	Melakukan pengecekan status aktif *device token*	Kapabilitas aplikasi untuk melakukan *check token status* apakah masih aktif atau tidak. Jika sudah tidak aktif maka akan melakukan update status aktif *device token* pada database.
F05	Melakukan *scheduling*	Kapabilitas aplikasi untuk melakukan pengiriman notifikasi berdasarkan waktu pengiriman yang telah ditentukan.
F06	Melakukan *queuing*	Kapabilitas aplikasi untuk melakukan *queuing* data notifikasi sebelum dikirim menuju BaaS *(Backend as a Service)* Firebase dan APNs guna menjaga ketersediaan *service* dengan sumber daya yang terbatas.
Aplikasi Pengelola Konten Notifikasi		
F07	Menampilkan data pengguna serta *device token* yang teregistrasi	Menampilkan data pengguna beserta mapping pengguna berdasarkan *role* dan *unit*nya, serta *device token* yang teregistrasi. Terletak di halaman *dashboard*.

No	Kebutuhan Fungsional	Deskripsi
F08	Menampilkan daftar peran pengguna (*user role*)	Menampilkan daftar peran pengguna (*user role*) berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman *dashboard*.
F09	Menampilkan daftar *unit* pengguna	Menampilkan daftar *unit* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman *dashboard*.
F10	Menampilkan daftar *client*	Menampilkan daftar *client* berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman *dashboard*.
F11	Mengirim notifikasi dengan antarmuka pengguna	Menampilkan antarmuka pengguna untuk mengirim notifikasi kepada tiga jenis target notifikasi. 1. Single 2. Multiple 3. Targeted Target *post* url adalah *API endpoint* pada aplikasi *push notification* terpusat. Terletak di halaman kirim notifikasi.
F12	Menampilkan hasil operasi *check device token*	Mengirim request ke Aplikasi *Push Notification* Terpusat untuk melakukan *check device token status* lalu menampilkan hasil dari check status. Terletak di halaman cek status token.
F13	Menampilkan halaman *packet monitoring*	Menampilkan data notifikasi yang diterima dari *client*. Akan terlihat status pengiriman dan konten yang dikirim. Terletak di halaman monitoring paket.
F14	Menampilkan daftar moderasi notifikasi	Menampilkan data notifikasi yang diterima dari *client* yang memiliki flag moderasi. untuk kemudian notifikasi akan dikirim setelah dilakukan moderasi konten. Setelah itu, data notifikasi akan dikirim dengan dibaca oleh *scheduler*. Terletak di halaman moderasi.
F15	Menampilkan	Menampilkan daftar pengguna yang

No	Kebutuhan Fungsional	Deskripsi
	daftar pengguna	diambil dari basis data SSO ITS (my.its.ac.id). Terletak di halaman daftar pengguna.
F16	Melakukan manajemen klien	Menampilkan daftar klien, melakukan edit status moderasi untuk klien. Terletak di halaman manajemen klien.
F17	Menampilkan halaman daftar akses token untuk aplikasi klien	Menampilkan daftar *access token* untuk aplikasi klien
F18	Melakukan pengaturan tujuan notifikasi	Melakukan manajemen tujuan pengiriman notifikasi yang dikelompokkan berdasarkan pasangan (*pair*) dari peran (*role*) dan unit pengguna.
F19	Menampilkan halaman pengaturan	Menampilkan antarmuka pengguna untuk melakukan perintah mulai dan berhentinya: 1. *Scheduler* 2. *Message queuing*
F20	Melakukan manajemen sertifikat/key untuk BaaS (Backend as a Service)	Melakukan manajemen sertifikat/key BaaS (*Backend as a Service*) untuk masing-masing aplikasi klien. Dikelompokkan berdasarkan *access token*.

3.1.3.3 Kebutuhan Non Fungsional

Kebutuhan non-fungsional yang harus dipenuhi oleh sistem dapat dilihat pada Tabel 3.3.

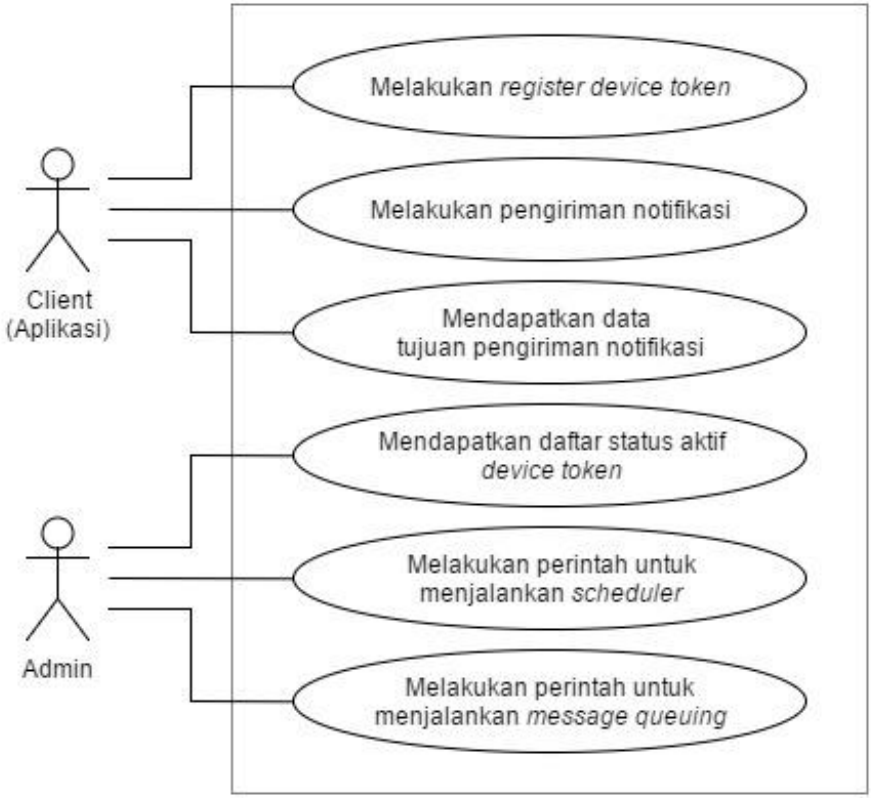
Tabel 3.3 Kebutuhan Non-Fungsional pada Sistem

No	Parameter	Deskripsi
Aplikasi *Push Notification* Terpusat		
1	Ketersediaan	Aplikasi harus dapat berjalan pada sistem operasi yang sesuai dengan platform yang telah disebutkan.
2	Koneksi internet	Aplikasi harus terpasang pada server yang selalu terhubung dengan jaringan internet.
Aplikasi Pengelola Konten Notifikasi		
1	Ketersediaan	Aplikasi harus dapat berjalan pada sistem operasi yang sesuai dengan platform yang telah disebutkan.
2	Tingkat kualitas	Aplikasi dibangun dengan antarmuka pengguna yang konsisten, mudah dipahami dan mudah dioperasikan
3	*Portability*	Aplikasi mudah untuk dioperasikan pada semua perangkat digital dengan mengakses *website*.
4	Bahasa	Bahasa yang digunakan pada antarmuka merupakan bahasa Indonesia.

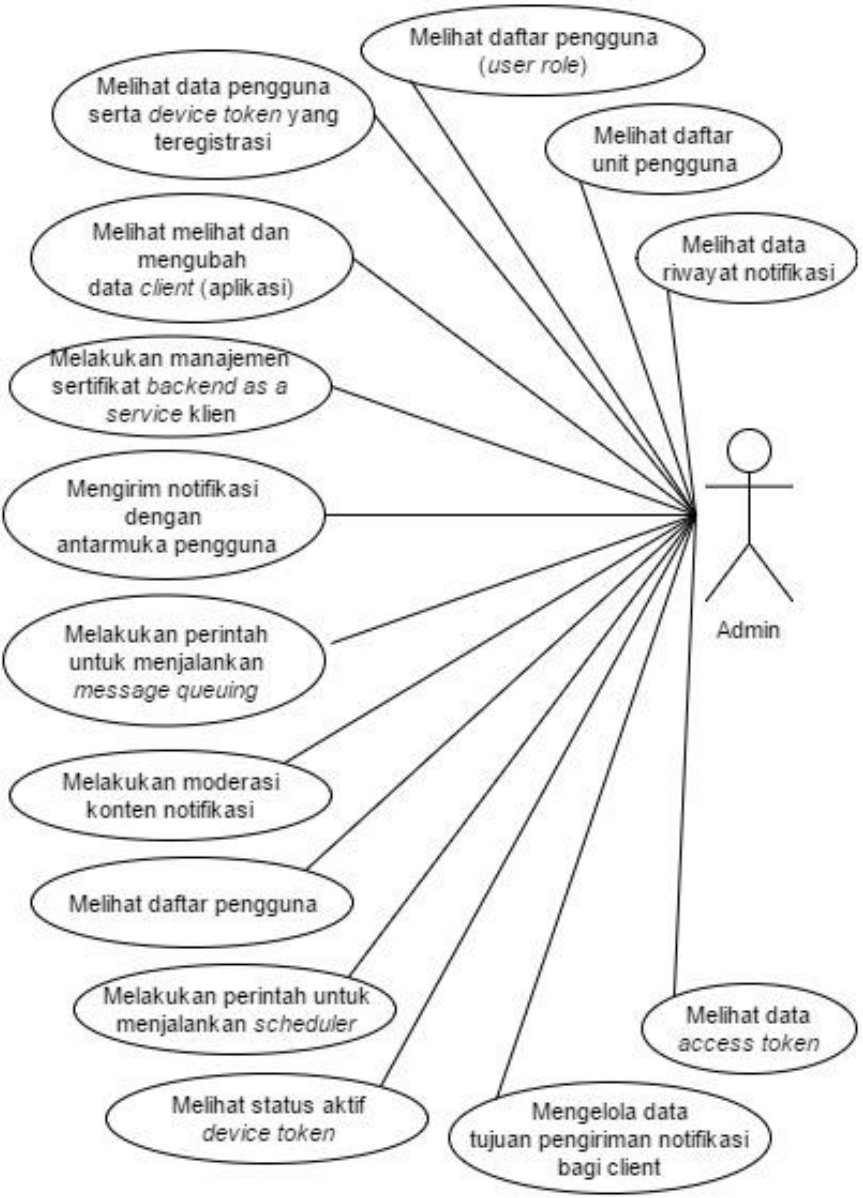
3.1.3.4 Diagram Kasus Penggunaan

Kasus penggunaan perangkat lunak dijelaskan secara rinci dalam subbab ini. Kasus penggunaan didasarkan pada hasil analisis kebutuhan fungsional dari perangkat lunak. Arsitektur sistem dapat dilihat pada bagian perancangan arsitektur halaman 69.

Terdapat dua diagram kasus penggunaan, masing-masing untuk Aplikasi *Push Notification* Terpusat ditunjukkan pada Gambar 3.1, dan Aplikasi Pengelola Konten Notifikasi ditunjukkan pada Gambar 3.2.



Gambar 3.1 Diagram Kasus Penggunaan Aplikasi *Push Notification* Terpusat



Gambar 3.2 Diagram Kasus Penggunaan Aplikasi Pengelola Konten Notifikasi

Penjelasan lengkap mengenai kasus penggunaan dapat dilihat pada Tabel 3.4.

Tabel 3.4 Deskripsi Kasus Penggunaan Sistem

Kode Use Case	Nama Use Case	Aktor
Aplikasi *Push Notification* Terpusat		
UC001	Melakukan *register device token*	*Client* (aplikasi)
UC002	Melakukan pengiriman notifikasi	
UC003	Mendapatkan data tujuan pengiriman notifikasi	
UC004	Mendapatkan daftar status aktif device token	Administrator
UC005	Melakukan perintah untuk menjalankan *scheduler*	
UC006	Melakukan perintah untuk menjalankan *message queuing*	
Aplikasi Pengelola Konten Notifikasi		
UC007	Melihat data pengguna serta *device token* yang teregistrasi	Administrator
UC008	Melihat daftar peran pengguna (*user role*)	Administrator
UC009	Melihat daftar *unit* pengguna	Administrator
UC010	Melihat dan mengubah data *client*	Administrator
UC011	Melakukan pengiriman notifikasi dengan antarmuka pengguna	Administrator
UC012	Melihat data riwayat notifikasi (*packet monitoring*)	Administrator
UC013	Melihat status aktif *device token*	Administrator
UC014	Melakukan moderasi konten notifikasi	Administrator
UC015	Melihat daftar pengguna	Administrator
UC016	Melihat data *access token*	Administrator
UC017	Mengelola (menambah, mengubah, menghapus) data tujuan pengiriman notifikasi bagi *client*	Administrator
UC018	Melakukan perintah untuk menjalankan *scheduler*	Administrator
UC019	Melakukan perintah untuk menjalankan *message queuing*	Administrator

Kode Use Case	Nama Use Case	Aktor
UC020	Mengelola sertifikat klien	Administrator

3.1.3.4.1 Kasus Penggunaan Melakukan Register Device Token

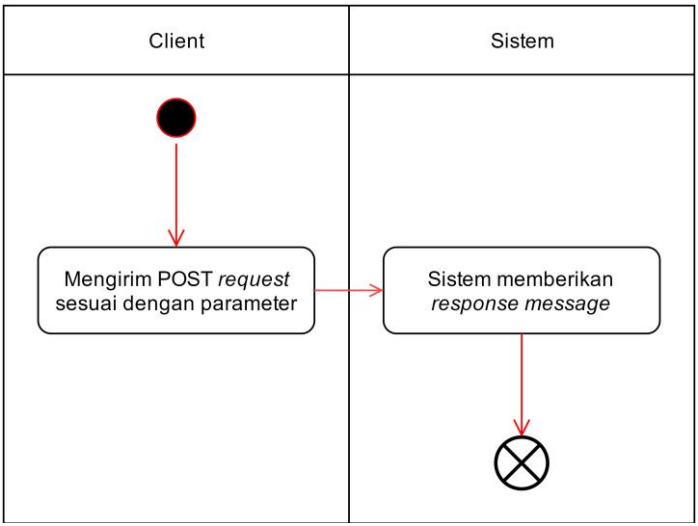
Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk melakukan registrasi *device token* untuk disimpan kedalam basis data sebagai parameter pengiriman *push notification*. Pada kasus penggunaan ini juga dapat dilakukan *checking* apakah *device token* telah teregistrasi ataupun belum pada basis data. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.5.

Tabel 3.5 Kasus Penggunaan Melakukan *Register Device Token*

Komponen	Deskripsi
Nama	Melakukan *register device token*
Nomor	UC001
Deskripsi	Kasus penggunaan ini digunakan untuk melakukan *checking* apakah *device token* telah teregistrasi ataupun belum pada basis data.
Tipe	Fungsional
Aktor	*Client*
Kondisi Awal	Aplikasi menerima POST request dari *client* dengan *message body* format JSON. Contoh request ditunjukkan pada kode sumber 3.4.
Kondisi Akhir	Aplikasi mengirim *response message* JSON. Ditunjukkan pada kode sumber 1, 2 dan 3.
Alur Normal	1. Aktor mengirim POST request ke endpoint API (‘/register/device/’) 2. Sistem menampilkan *response message* dengan *message body*

Komponen	Deskripsi
	format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.5. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.3.



Gambar 3.3 Diagram Aktivitas Melakukan *Register Device Token*

3.1.3.4.2 Kasus Penggunaan Melakukan Pengiriman Notifikasi

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk melakukan pengiriman notifikasi kepada *user* dengan data *user* yang diambil dari basis data SSO ITS (*my.its.ac.id*).
Pengiriman notifikasi dibagi menjadi tiga jenis.

1. Single, menggunakan *username, client-id* sebagai parameter mendapatkan *device token.*

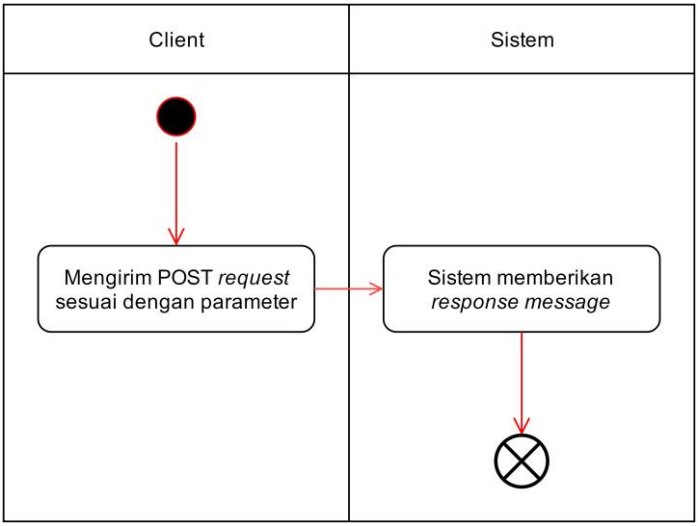
2. Multiple, berdasarkan *user-id* atau *username* dan *client-id* sebagai parameter mendapatkan *device token*
3. Targeted, berdasarkan mapping *user role* dan *user unit*, *client-id* sebagai parameter mendapatkan *device token.*

Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.6.

Tabel 3.6 Kasus Penggunaan Melakukan Pengiriman Notifikasi

Komponen	Deskripsi
Nama	Melakukan pengiriman notifikasi
Nomor	UC002
Deskripsi	Kasus penggunaan ini digunakan untuk melakukan pengiriman notifikasi kepada *user* dengan data *user* yang diambil dari basis data sso ITS (*my.its.ac.id*).
Tipe	Fungsional
Aktor	*Client*
Kondisi Awal	Aplikasi menerima POST request dari *client* dengan *message body* format JSON. Contoh request ditunjukkan pada kode sumber .
Kondisi Akhir	Aplikasi mengirim *response message* JSON. Ditunjukkan pada kode sumber 4, 5 dan 6
Alur Normal	1. Aktor mengirim POST request ke salah satu endpoint API (‘/send/single /’), (‘/send/multiple/’), (‘/send/targeted/’) 2. Sistem menampilkan *response message* dengan *message body* format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.6. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.4.



Gambar 3.4 Diagram Aktivitas Melakukan Pengiriman Notifikasi

3.1.3.4.3 Kasus Penggunaan Mendapatkan Data Tujuan Pengiriman Notifikasi

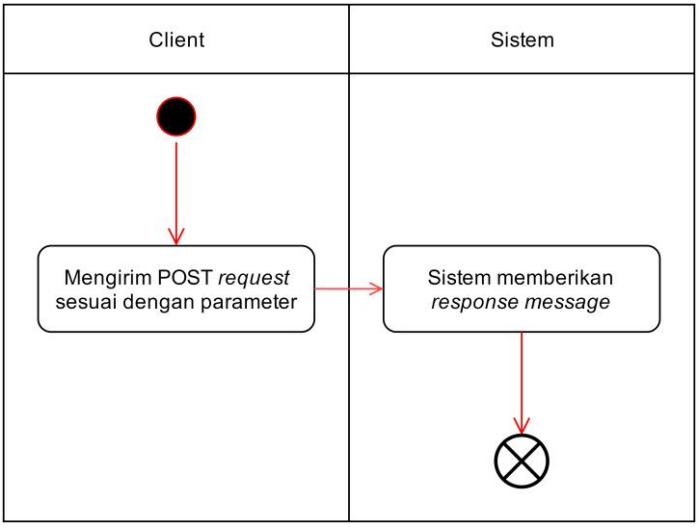
Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk mendapatkan data tujuan pengiriman notifikasi yang dapat digunakan oleh *client* sebagai parameter pengiriman notifikasi. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.7.

Tabel 3.7 Kasus Penggunaan Mendapatkan Data Tujuan Pengiriman Notifikasi

Komponen	Deskripsi
Nama	Mendapatkan data tujuan pengiriman notifikasi
Nomor	UC003
Deskripsi	Kasus penggunaan ini digunakan untuk

Komponen	Deskripsi
	mendapatkan data tujuan pengiriman notifikasi yang dapat digunakan oleh *client* sebagai parameter pengiriman notifikasi
Tipe	Fungsional
Aktor	*Client*
Kondisi Awal	Aplikasi menerima GET request dari *client*.
Kondisi Akhir	Aplikasi mengirim *response message* JSON. Ditunjukkan pada kode sumber 8
Alur Normal	1. Aktor mengirim GET request ke endpoint API sebagai berikut (‘ /send/destination?access_token=’..) 2. Sistem menampilkan *response message* dengan *message body* format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.7. Selanjutnya skenario tersebut digambarkan ke dalam diagram pada Gambar 3.5.



Gambar 3.5 Diagram Aktivitas Mendapatkan Data Tujuan Pengiriman Notifikasi

3.1.3.4.4 Kasus Penggunaan Mendapatkan Daftar Status Aktif Device Token

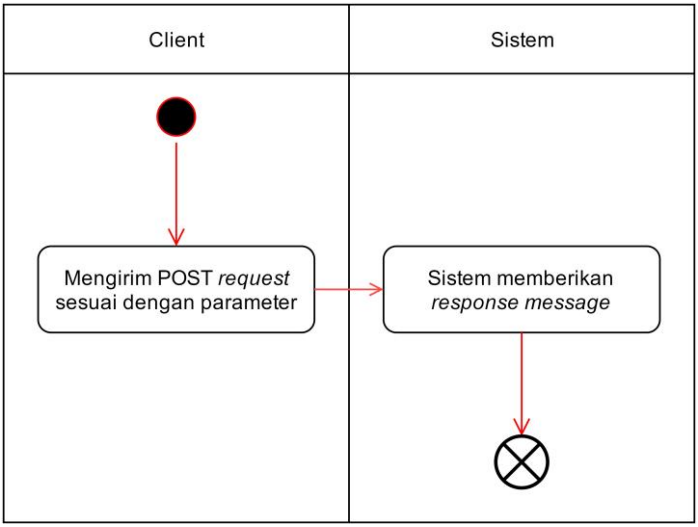
Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk melakukan *check token status* apakah masih aktif atau tidak. Jika sudah tidak aktif maka akan menghapus *device token* dari database.

Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.8.

Tabel 3.8 Kasus Penggunaan Mendapatkan Daftar Status Aktif *Device Token*

Komponen	Deskripsi
Nama	Mendapatkan daftar status aktif device token
Nomor	UC004
Deskripsi	Kasus penggunaan ini digunakan untuk melakukan *check token status* apakah masih aktif atau tidak. Jika sudah tidak aktif maka akan menghapus *device token* dari database.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aplikasi menerima GET request dari *client*.
Kondisi Akhir	Aplikasi mengirim *response message* JSON. Ditunjukkan pada kode sumber 10
Alur Normal	1. Aktor mengirim GET request ke endpoint API sebagai berikut (‘/check/token?access_token=’..) 2. Sistem menampilkan *response message* dengan *message body* format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.8. Selanjutnya skenario tersebut digambarkan ke dalam diagram pada Gambar 3.6.



Gambar 3.6 Diagram Aktivitas Mendapatkan Daftar Status Aktif *Device Token*

3.1.3.4.5 Kasus Penggunaan Melakukan Perintah untuk Menjalankan Scheduler

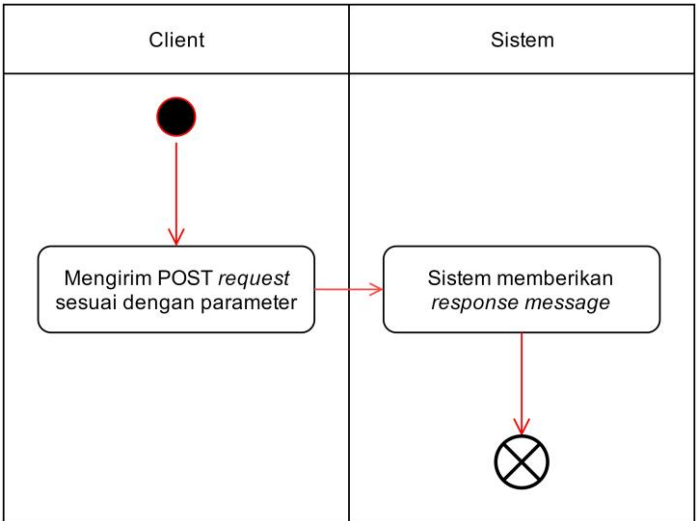
Pada kasus penggunaan ini, sistem memiliki kapabilitas untuk melakukan pengiriman notifikasi berdasarkan waktu pengiriman yang telah ditentukan.

Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.9.

Tabel 3.9 Kasus Penggunaan *Scheduling*

Komponen	Deskripsi
Nama	Melakukan perintah untuk menjalankan scheduler
Nomor	UC005
Deskripsi	Kapabilitas sistem untuk melakukan pengiriman notifikasi berdasarkan waktu pengiriman yang telah ditentukan.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Scheduling dijalankan oleh aktor
Kondisi Akhir	Aplikasi mengirim data *push notification* menuju *push queue*
Alur Normal	1. Aktor mengirim POST request untuk menjalankan *scheduler* ke endpoint API sebagai berikut (‘/scheduler/start’) 2. Sistem menampilkan *response message* dengan *message body* format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.9. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.7.



Gambar 3.7 Diagram Aktivitas Melakukan Perintah untuk Menjalankan *Scheduler*

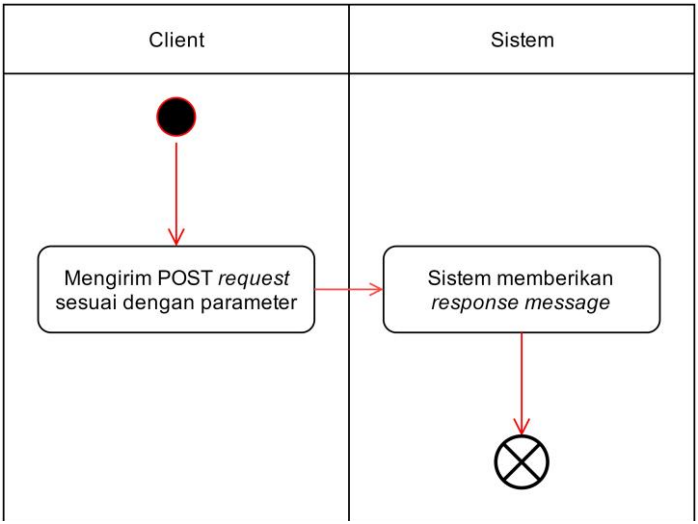
3.1.3.4.6 Kasus Penggunaan Melakukan Perintah untuk Menjalankan Message Queue

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menjalankan *message queue* yang terdapat pada aplikasi *push notification* terpusat melalu REST API. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.10.

Tabel 3.10 Kasus Penggunaan Melakukan Perintah untuk Menjalankan *Message Queuing*

Komponen	Deskripsi
Nama	Melakukan perintah untuk menjalankan *message queue*
Nomor	UC006
Deskripsi	Kapabilitas sistem untuk melakukan *queuing* sebelum mengirim notifikasi guna menjaga ketersediaan *service* dengan sumber daya yang terbatas.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Queuing di jalankan
Kondisi Akhir	Aplikasi mengirim data *push notification* dari *thread push queue* yang telah ditentukan untuk kemudian dikirim ke FCM dan/atau APNS.
Alur Normal	1. Aktor mengirim POST request untuk menyalakan *message queuing* ke endpoint API sebagai berikut (‘/queue/start/’) 2. Sistem menampilkan *response message* dengan *message body* format JSON.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.10. Selanjutnya skenario tersebut digambarkan ke dalam diagram pada Gambar 3.8.



Gambar 3.8 Diagram Aktivitas Melakukan Perintah untuk Menjalankan *Message Queuing*

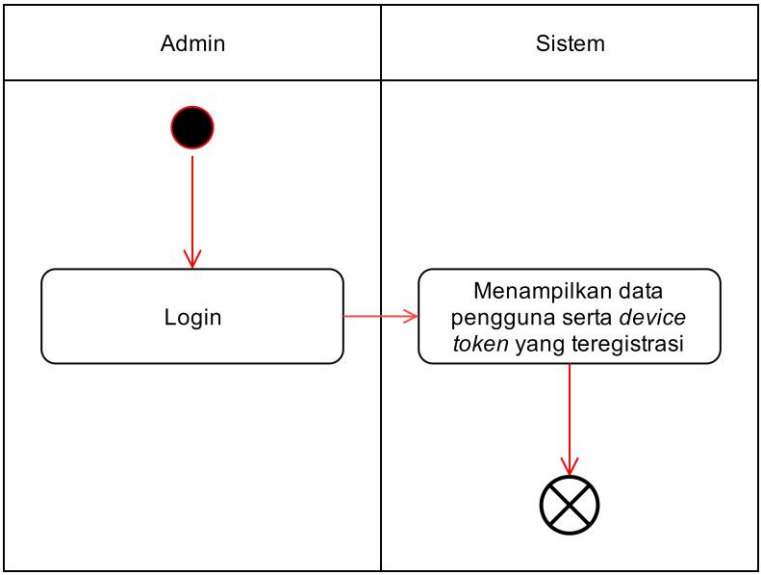
3.1.3.4.7 Kasus Penggunaan Melihat Data Pengguna Serta Device Token yang Teregistrasi

Pada kasus penggunaan ini, aktor memiliki kapabilitas menampilkan data pengguna beserta mapping pengguna berdasarkan *role* dan *unit*nya, serta *device token* yang teregistrasi. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.11.

Tabel 3.11 Kasus Penggunaan Melihat Data Pengguna Serta Device Token yang Teregistrasi

Komponen	Deskripsi
Nama	Melihat data pengguna serta *device token* yang teregistrasi.
Nomor	UC007
Deskripsi	Kasus penggunaan ini menampilkan data pengguna beserta mapping pengguna berdasarkan *role* dan *unit*nya, serta *device token* yang teregistrasi. Terletak di halaman dashboard.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor login ke aplikasi
Kondisi Akhir	Aplikasi dapat menampilkan data pengguna serta *device token* yang teregistrasi.
Alur Normal	1. Aktor login 2. Aplikasi menampilkan data pengguna serta *device token* yang teregistrasi secara otomatis (*dashboard*).
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.11. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.9.



Gambar 3.9 Diagram Aktivitas Melihat Data Pengguna Serta *Device Token* yang Teregistrasi

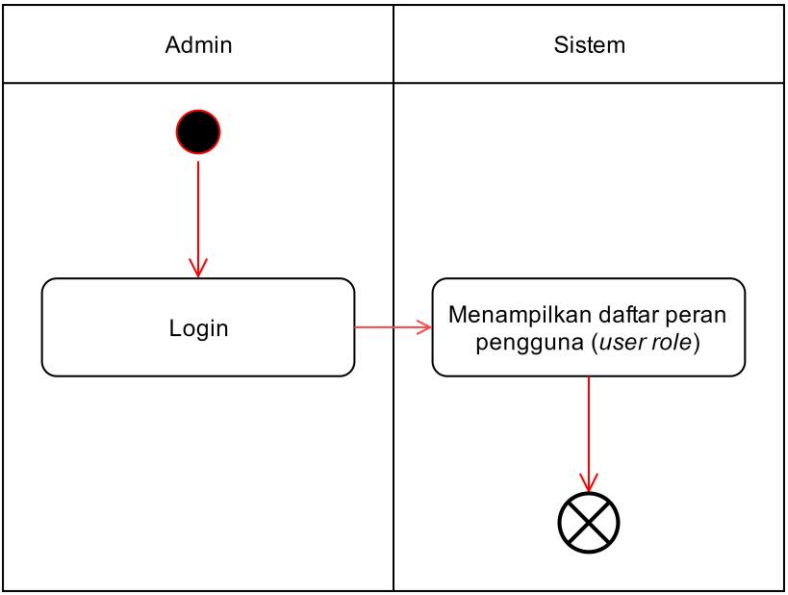
3.1.3.4.8 Kasus Penggunaan Menampilkan Daftar Peran Pengguna (User Role)

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan daftar *role* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.12.

Tabel 3.12 Kasus Penggunaan Menampilkan Daftar Peran Pengguna

Komponen	Deskripsi
Nama	Menampilkan daftar peran pengguna
Nomor	UC008
Deskripsi	Kasus penggunaan ini menampilkan daftar *role* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor login ke aplikasi
Kondisi Akhir	Aplikasi dapat menampilkan daftar *role*
Alur Normal	1. Aktor login 2. Aplikasi menampilkan daftar *role*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.12. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.10.



Gambar 3.10 Diagram Aktivitas Menampilkan Daftar Peran Pengguna (*User Role*)

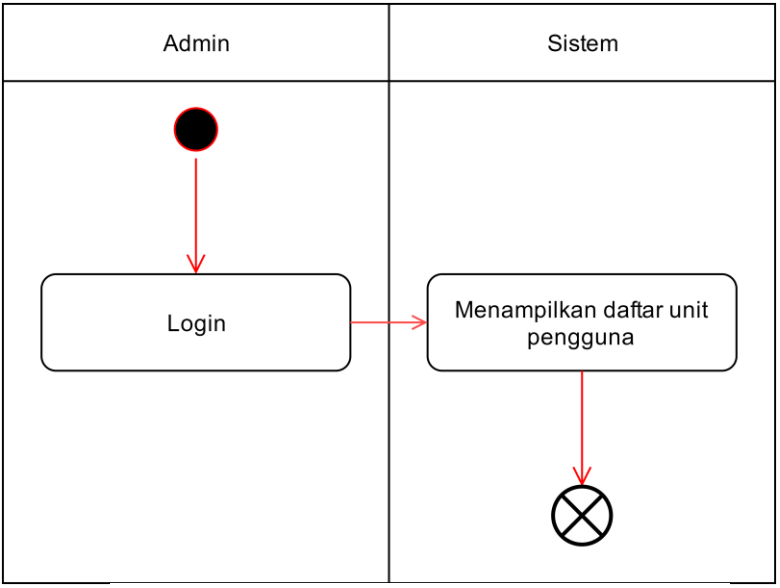
3.1.3.4.9 Kasus Penggunaan Menampilkan Daftar Unit Pengguna

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan daftar *unit* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.13.

Tabel 3.13 Kasus Penggunaan Menampilkan Daftar Unit Pengguna

Komponen	Deskripsi
Nama	Menampilkan daftar *unit* pengguna
Nomor	UC009
Deskripsi	Kasus penggunaan ini menampilkan daftar *unit* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard.
Tipe	Fungsional
Aktor	*Client*
Kondisi Awal	Aktor login ke aplikasi
Kondisi Akhir	Aplikasi dapat menampilkan daftar *unit*
Alur Normal	1. Aktor login 2. Aplikasi menampilkan daftar *unit*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.13. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.11.



Gambar 3.11 Diagram Aktivitas Menampilkan Daftar Unit Pengguna

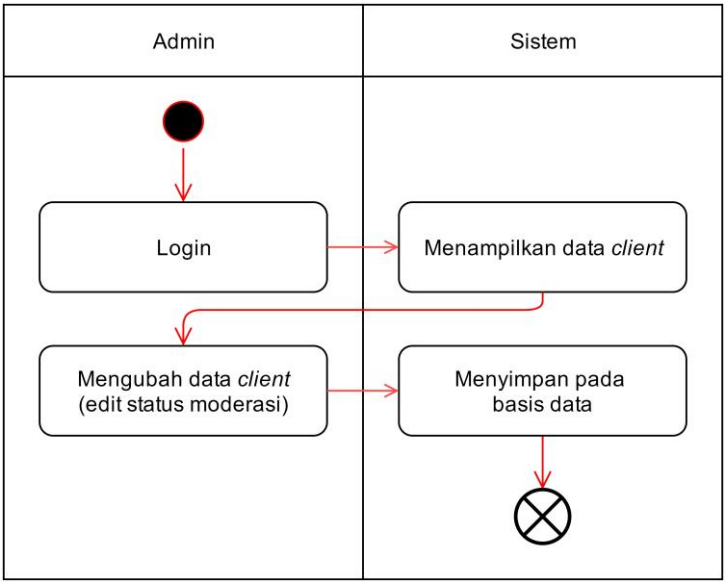
3.1.3.4.10 Kasus Penggunaan Melihat dan Mengubah Data Klien

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan daftar *client* berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.14.

Tabel 3.14 Kasus Penggunaan Melihat dan Mengubah Daftar *Client*

Komponen	Deskripsi
Nama	Melihat dan mengubah data klien
Nomor	UC010
Deskripsi	Kasus penggunaan ini menampilkan daftar *client* pengguna berdasarkan basis data SSO ITS (my.its.ac.id). Terletak di halaman dashboard.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor login ke aplikasi
Kondisi Akhir	Aplikasi dapat menampilkan daftar *client*
Alur Normal	1. Aktor login 2. Aplikasi menampilkan daftar *client*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.14. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.12.



Gambar 3.12 Diagram Aktivitas Melihat dan Mengubah Data Klien

3.1.3.4.11 Kasus Penggunaan Pengiriman Notifikasi dengan Antarmuka Pengguna

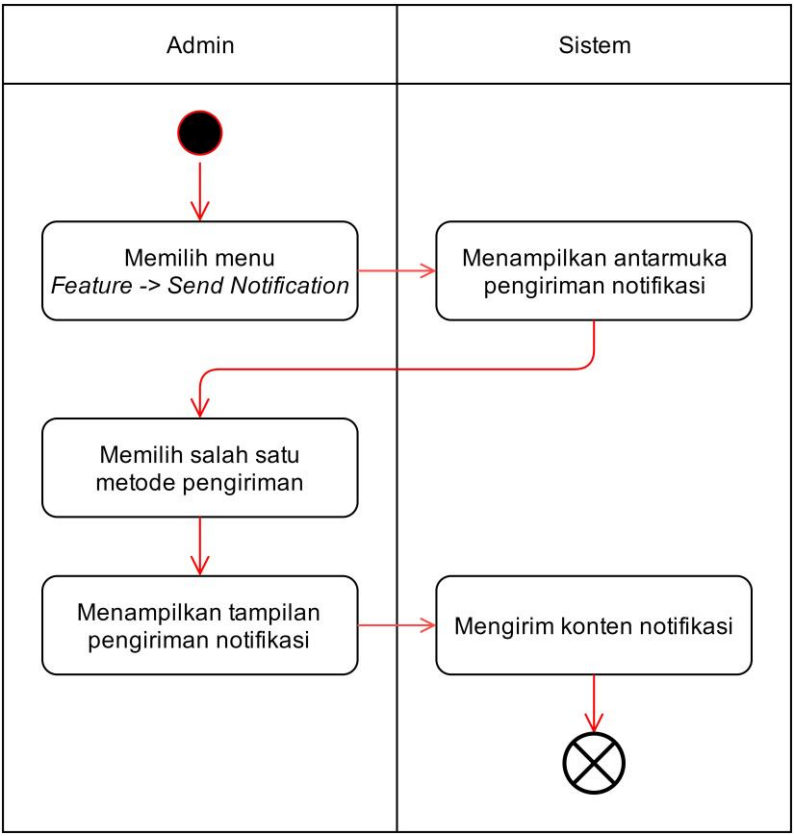
Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan antarmuka pengguna untuk mengirim notifikasi kepada tiga jenis target notifikasi yaitu *single*, *multiple* dan *targeted*. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.15.

Tabel 3.15 Kasus Penggunaan Mengirim Notifikasi dengan Antarmuka Pengguna

Komponen	Deskripsi
Nama	Mengirim Notifikasi dengan Antarmuka Pengguna
Nomor	UC011
Deskripsi	Kasus penggunaan ini menampilkan antarmuka pengguna untuk mengirim notifikasi kepada tiga jenis target notifikasi. 1. Single 2. Multiple 3. Targeted
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *send notification*
Kondisi Akhir	Aplikasi mengirim notifikasi
Alur Normal	1. Aktor memilih menu *Feature* → *Send notification* 2. Aplikasi menampilkan antarmuka pengguna tiga jenis mode pengiriman notifikasi. 3. Aktor memilih salah satu mode pengiriman. 4. Aktor mengisi konten notifikasi yang akan dikirim lalu mengirim notifikasi menuju target yang ditentukan.

Komponen	Deskripsi
	5. Sistem mengirim notifikasi
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.15. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.13.



Gambar 3.13 Diagram Aktvitas Mengirim Notifikasi dengan Antarmuka Pengguna

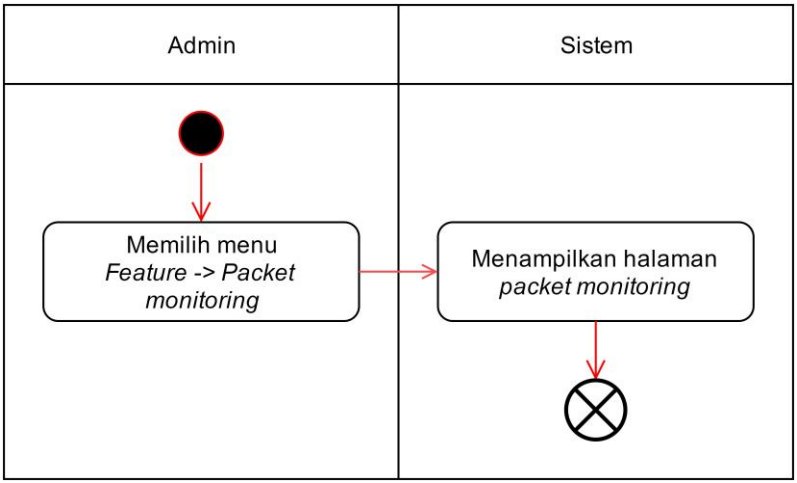
3.1.3.4.12 Kasus Penggunaan Melihat Data Riwayat Notifikasi

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan seluruh data *request* pengiriman *push notification* dari *client* yang teregistrasi. Akan terlihat status pengiriman dan konten yang dikirim. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.16.

Tabel 3.16 Kasus Penggunaan Menampilkan Data Riwayat Notifikasi

Komponen	Deskripsi
Nama	Melihat data riwayat notifikasi
Nomor	UC012
Deskripsi	Kasus penggunaan ini menampilkan seluruh data *request* pengiriman *push notification* dari *client* yang teregistrasi. Akan terlihat status pengiriman dan konten yang dikirim
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *request monitoring*
Kondisi Akhir	Aplikasi dapat menampilkan daftar *client*
Alur Normal	1. Aktor memilih menu *Feature* → *Request monitoring* 2. Aplikasi menampilkan daftar *request*.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.16. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.14.



Gambar 3.14 Diagram Aktvitas Melihat Data Riwayat Notifikasi

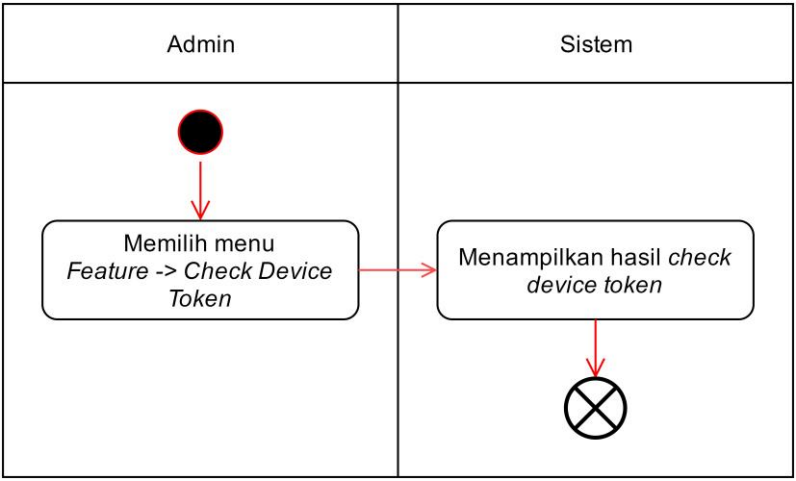
3.1.3.4.13 Kasus Penggunaan Melihat Status Aktif Device Token

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk mengirim request ke Aplikasi *Push Notification* Terpusat untuk melakukan *check device token status* lalu menampilkan hasil dari check status. untuk kemudian dilanjutkan dengan menghapus device token yang tidak aktif. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.17.

Tabel 3.17 Kasus Penggunaan Melihat Status Aktif *Device Token*

Komponen	Deskripsi
Nama	Melihat status aktif *device token*
Nomor	UC013
Deskripsi	Kasus penggunaan ini mengirim request ke Aplikasi *Push Notification* Terpusat untuk melakukan *check device token status* lalu menampilkan hasil dari *check status*.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *check device token*
Kondisi Akhir	Aplikasi dapat menampilkan daftar *client*
Alur Normal	1. Aktor memilih menu *Feature* → *Check Device Token* 2. Aplikasi menampilkan antarmuka pengguna hasil *check device token*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.17. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.15.



Gambar 3.15 Diagram Aktvitas Melihat Status Aktif *Device Token*

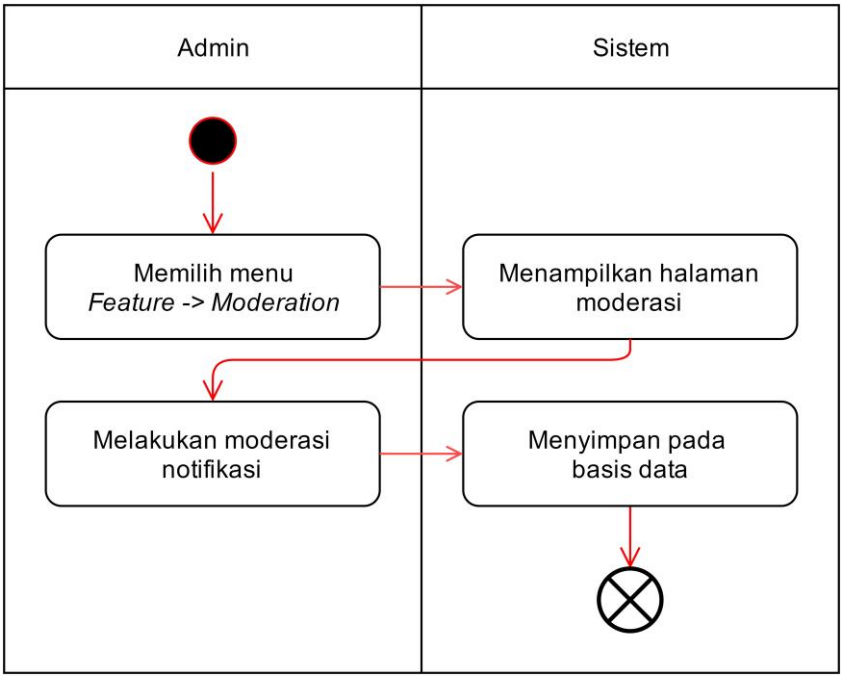
3.1.3.4.14 Kasus Penggunaan Melakukan Moderasi Konten Notifikasi

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan seluruh *request* pengiriman *push notification* dari *client* yang memiliki flag moderasi. untuk kemudian notifikasi akan dikirim setelah dilihat terlebih dahulu preview kontennya. Setelah dilakukan moderasi, maka *request* akan dikirim dengan dibaca oleh *scheduler*. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.18.

Tabel 3.18 Kasus Penggunaan Melakukan Moderasi Konten Notifikasi

Komponen	Deskripsi
Nama	Melakukan moderasi konten notifikasi
Nomor	UC014
Deskripsi	Kasus penggunaan ini menampilkan seluruh *request* pengiriman *push notification* dari *client* yang memiliki flag moderator. untuk kemudian notifikasi akan dikirim setelah dilihat terlebih dahulu preview kontennya. Setelah dilakukan moderasi, maka *request* akan dikirim dengan dibaca oleh *scheduler*.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman moderator
Kondisi Akhir	Aplikasi dapat menampilkan daftar paket yang perlu dilakukan moderasi
Alur Normal	1. Aktor memilih menu *Feature* → *Moderation* 2. Aplikasi menampilkan daftar paket yang harus dilakukan moderasi.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.18. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.16.



Gambar 3.16 Diagram Aktivitas Melakukan Moderasi Konten Notifikasi

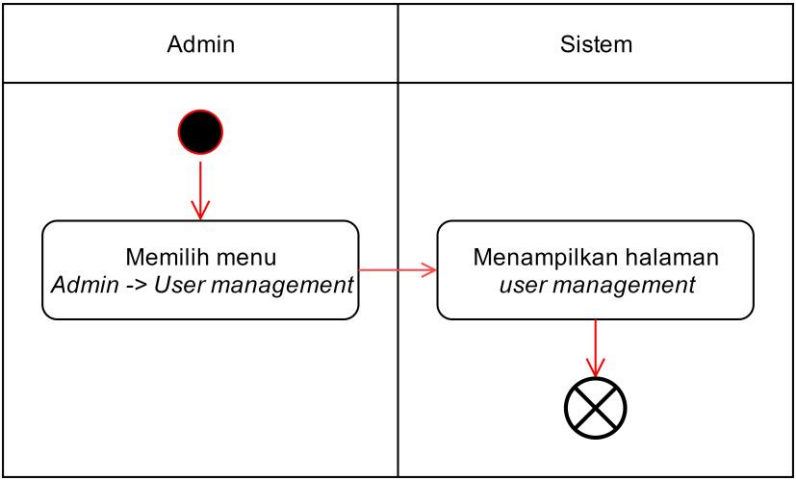
3.1.3.4.15 Kasus Penggunaan Melihat Daftar Pengguna

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan daftar pengguna. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.19.

Tabel 3.19 Kasus Penggunaan Melihat Daftar Pengguna

Komponen	Deskripsi
Nama	Melihat daftar pengguna
Nomor	UC015
Deskripsi	Kasus penggunaan ini menampilkan daftar pengguna
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *user management*
Kondisi Akhir	Aplikasi menampilkan daftar *user*
Alur Normal	1. Aktor memilih menu *Administrator* → *User management* 2. Aplikasi menampilkan daftar pengguna.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.19. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.17.



Gambar 3.17 Diagram Aktvitas Melihat Daftar Pengguna

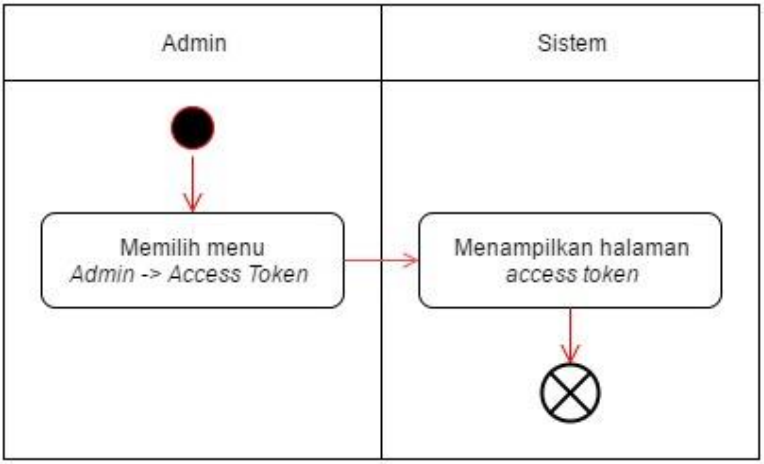
3.1.3.4.16 Kasus Penggunaan Melihat Data Access Token

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menampilkan daftar *access token*. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.20.

Tabel 3.20 Kasus Penggunaan Melihat Data *Access Token*

Komponen	Deskripsi
Nama	Melihat data *access token*
Nomor	UC016
Deskripsi	Kasus penggunaan ini menampilkan daftar pengguna sekaligus dapat melakukan operasi CRUD *access token*.
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *access token management*
Kondisi Akhir	Aplikasi menampilkan daftar *access token*
Alur Normal	1. Aktor memilih menu *Administrator* → *Access token management* 2. Aplikasi menampilkan daftar access token.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.20. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.18.



Gambar 3.18 Diagram Aktvitas Melihat Data *Access Token*

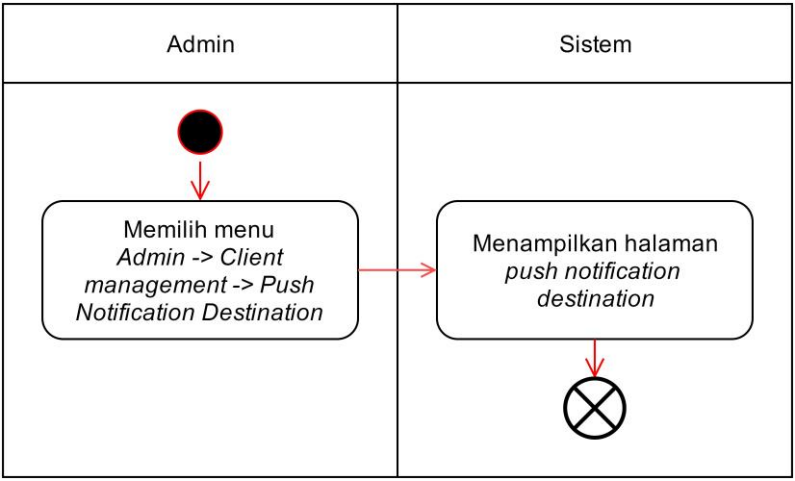
3.1.3.4.17 Kasus Penggunaan Mengelola Data Tujuan Pengiriman Notifikasi Klien

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk melakukan operasi CRUD daftar tujuan pengiriman notifikasi bagi klien. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.21.

Tabel 3.21 Kasus Penggunaan Mengelola Data Tujuan Pengiriman Notifikasi Klien

Komponen	Deskripsi
Nama	Mengelola data tujuan pengiriman notifikasi client
Nomor	UC017
Deskripsi	Kasus penggunaan ini menampilkan daftar pengguna sekaligus dapat melakukan operasi CRUD *client*
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *client management*
Kondisi Akhir	Aplikasi menampilkan daftar *client*
Alur Normal	1. Aktor memilih menu *Administrator* → *Access Token* → *Push Notification Destination* 2. Aplikasi menampilkan daftar tujuan pengiriman notifikasi bagi klien.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.21. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.19.



Gambar 3.19 Diagram Aktvitas Mengelola Data Tujuan Pengiriman Notifikasi Klien

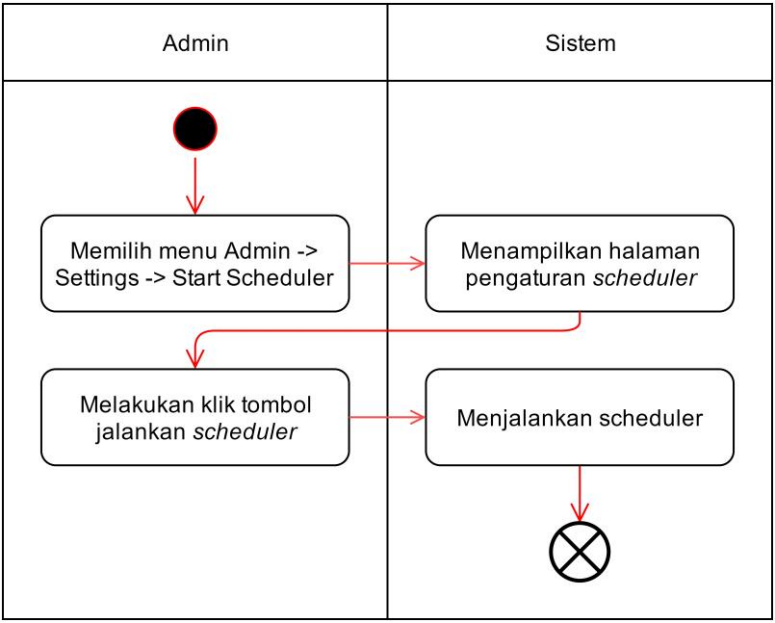
3.1.3.4.18 Kasus Penggunaan Menjalankan Scheduler

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menjalankan *scheduler* aplikasi *push notification* terpusat. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.22.

Tabel 3.22 Kasus Penggunaan Menjalankan *Scheduler*

Komponen	Deskripsi
Nama	Menjalankan *scheduler*
Nomor	UC018
Deskripsi	Kasus penggunaan ini menampilkan antarmuka pengguna untuk melakukan perintah menjalankan *scheduler* aplikasi *push notification* terpusat
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *settings* → *scheduler*
Kondisi Akhir	Aplikasi menampilkan halaman pengaturan *scheduler*
Alur Normal	1. Aktor memilih menu *Administrator* → *Settings* → *Scheduler* 2. Aplikasi menampilkan halaman pengaturan *scheduler* 3. Aktor melakukan klik tombol *start scheduler*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.22. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.20.



Gambar 3.20 Diagram Aktvitas Menjalankan *Scheduler*

3.1.3.4.19 Kasus Penggunaan Konfigurasi Message Queue

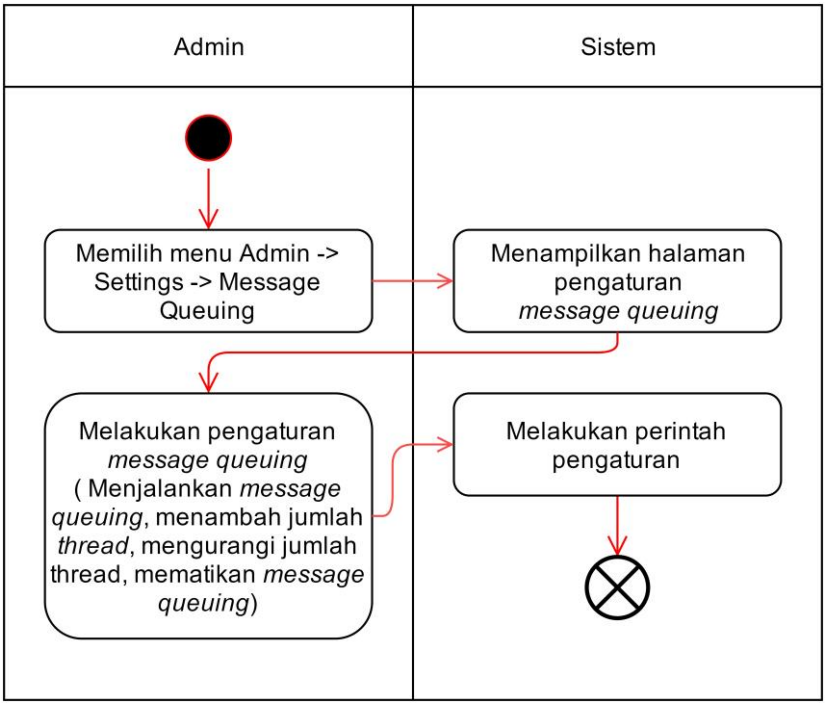
Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk menjalankan *message queuing* aplikasi *push notification* terpusat. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.22.

Tabel 3.23 Kasus Penggunaan Melakukan Perintah untuk Menjalankan *Message Queue*

Komponen	Deskripsi
Nama	Konfigurasi message *queue*
Nomor	UC019
Deskripsi	Kasus penggunaan ini menampilkan antarmuka pengguna untuk melakukan perintah konfigurasi *message queuing* aplikasi *push notification* terpusat
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *settings* → *message queue*
Kondisi Akhir	Aplikasi menampilkan halaman pengaturan *message queue*
Alur Normal	1. Aktor memilih menu *Administrator* → *Settings* → *Message Queuing* 2. Aplikasi menampilkan halaman pengaturan *message queuing* 3. Aktor melakukan konfigurasi *message queuing* Konfigurasi yang dapat dilakukan adalah sebagai berikut: 1. Menjalankan *message queue* 2. Menambah jumlah *worker thread message queue* 3. Mengurangi jumlah *worker*

Komponen	Deskripsi
	thread message queue 4. Mematikan *message queue* 5. Melihat status *message queue*
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.22. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.20.



Gambar 3.21 Melakukan Perintah untuk Menjalankan *Message Queue*

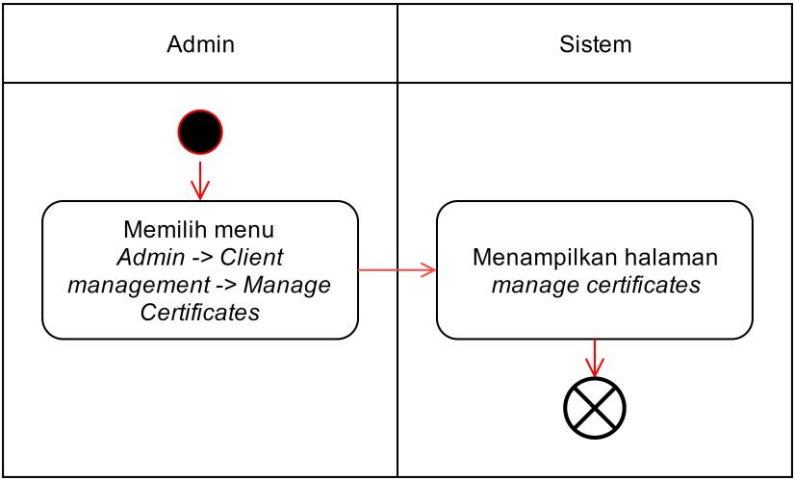
3.1.3.4.20 Kasus Penggunaan Mengelola Sertifikat Klien

Pada kasus penggunaan ini, aktor memiliki kapabilitas untuk melakukan operasi CRUD data sertifikat untuk klien. Penjelasan skenario yang lebih rinci dapat dilihat pada Tabel 3.24.

Tabel 3.24 Kasus Penggunaan Mengelola Sertifikat Klien

Komponen	Deskripsi
Nama	Mengelola sertifikat klien
Nomor	UC020
Deskripsi	Kasus penggunaan ini menampilkan daftar sertifikat/key klien sekaligus dapat melakukan operasi CRUD sertifikat/key klien
Tipe	Fungsional
Aktor	Administrator
Kondisi Awal	Aktor memilih halaman *client list* → *certificate management*
Kondisi Akhir	Aplikasi menampilkan daftar *certificate*
Alur Normal	1. Aktor memilih menu *Administrator* → *Client List* → *Push Notification Destination* 2. Aplikasi menampilkan daftar tujuan pengiriman notifikasi bagi klien.
Alur Alternatif	-

Berdasarkan skenario kasus penggunaan pada Tabel 3.24. Selanjutnya skenario tersebut digambarkan pada diagram pada Gambar 3.22.



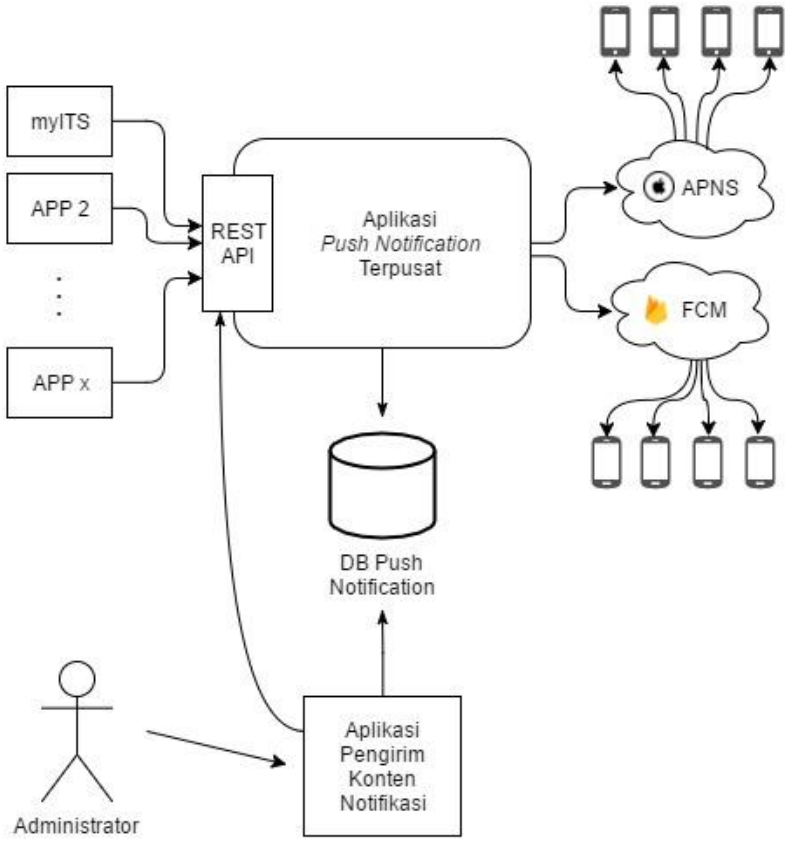
Gambar 3.22 Diagram Aktvitas Mengelola Sertifikat Klien

3.2 Perancangan Sistem

Pada subbab ini dijelaskan mengenai tahapan perancangan sistem. Perancangan sistem ini dibagi menjadi beberapa bagian yang meliputi perancangan data, perancangan arsitektur sistem, perancangan proses aplikasi, dan perancangan antarmuka pengguna.

3.2.1 Perancangan Arsitektur

Dalam tugas akhir ini akan dirancang dan diaplikasikan sebuah aplikasi *push notification* terpusat berbasis Java Maven dengan archetype Jersey, dan sebuah aplikasi pengelola konten notifikasi berbasis PHP dengan framework CodeIgniter. Secara garis besar aplikasi ini memiliki rancangan arsitektur sistem yang dapat dilihat pada Gambar 3.23.



Keterangan	
	Aktor aplikasi/*client*
	Aktor *administrator*

Gambar 3.23 Arsitektur Sistem

3.2.2 Perancangan Basis Data

Pada subbab ini akan mmbahas bagaimana rancangan basis data yang digunakan pada Aplikasi *Push Notification* Terpusat dan Aplikasi Pengelola Konten Notifikasi. Basis data yang digunakan adalah SQL Server 14. *Conceptual Data Model* (CDM) dapat dilihat pada Lampiran A. Sedangkan untuk *Physical Data Model* (PDM) dapat dilihat pada Lampiran B.

3.2.2.1 Perancangan Tabel Push Notification Packet

Tabel *push notification packet* digunakan untuk menyimpan seluruh konten data packet untuk push notifikasi. Detail atribut tabel *push notification packet* ditunjukkan pada Tabel 3.25.

Tabel 3.25 Tabel *Push Notification Packet*

Nama Atribut	Tipe Data	Deskripsi
ID Push Notification packet	uniqueidentifier	*Primary key* tabel *push notification packet*
User ID	uniqueidentifier	*Foreign key* tabel *user*
Access token	varchar(40)	*Foreign key* tabel *access token*
Method	varchar(150)	Jenis method request (GET, POST)
Arrival time	datetime	Waktu kedatangan *packet* notifikasi
Sent time	datetime	Waktu pengiriman notifikasi
Content type	varchar(150)	Tipe konten *request* (application/json)
Content	text	Data notifikasi dalam format JSON
Is sent	numeric(1)	Status kirim notifikasi

3.2.2.2 Perancangan Tabel User Account

Tabel *user account* digunakan untuk menyimpan data *user*. Detail atribut tabel *user account* ditunjukkan pada Tabel 3.26.

Tabel 3.26 Tabel *User Account*

Nama Atribut	Tipe Data	Deskripsi
User ID	Uniqueidentifier	*Primary key* tabel *user account*
Type ID	Int	*Foreign key* tabel *type*
Name	Varchar(150)	Nama pengguna
Nickname	Varchar(20)	Nama pendek pengguna
Username	Varchar(255)	*Username* pengguna
Password	Char(128)	Kata sandi pengguna
Salt	Char(128)	*Salt* pengguna
Email	Varchar(255)	Surel pengguna
Email verified	Numeric(1)	Status verifikasi surel pengguna
Alternate email	Varchar(255)	Surel alternatif pengguna
Alternate email verified	Numeric(1)	Status verifikasi surel alternatif pengguna
Phone	Varchar(18)	Nomor telepon pengguna
Phone verified	Numeric(1)	Status verifikasi nomor telepon pengguna
Enabled	Numeric(1)	Status aktif pengguna
Picture	Varbinary(max)	Data *profile picture* pengguna
Gender	Char(1)	Jenis kelamin pengguna
Birhdate	Date	Tanggal lahir pengguna
Zoneinfo	Varchar(40)	*Zona* pengguna
Locale	Varchar(10)	*Locale* pengguna
Integra ID	Bigint	ID Integra pengguna
Must change password	Numeric(1)	Status untuk penggantian kata sandi (harus diganti atau tidak)
Sandbox	Numeric(1)	Status akun *sandbox*
Locked	Datetime	Status terkunci akun
Suspended	Datetime	Status *suspended* akun
Has suspended	Datetime	Riwayat *suspended* akun
Create date	Datetime	Status akun dibuat
Last update	Datetime	Status akun terakhir diganti

3.2.2.3 Perancangan Tabel OAuth Client

Tabel *client* digunakan untuk menyimpan data *client*. Detail atribut tabel *client* ditunjukkan pada Tabel 3.27.

Tabel 3.27 Tabel *Oauth Client*

Nama Atribut	Tipe Data	Deskripsi
Device token ID	Uniqueidentifier	*Primary key* tabel *OAuth Client*
User ID	Uniqueidentifier	*Foreign key* tabel *user*
Provider ID	Uniqueidentifier	*Foreign key* tabel *provider*
Client name	Varchar(100)	Nama klien
Client description	Varchar(250)	Deskripsi klien
Client secret	Varchar(255)	Kode *secret* klien
Issued at	Datetime	Tanggal pembuatan klien
Expires at	Datetime	Tanggal mati klien
Client IP adderss/CIDR	Varchar(255)	IP klien
Logo	Varchar(100)	Logo klien
Redirect URI	Varchar(255)	*Redirect URI* klien
Base URI	Varchar(255)	*Base URI* klien
Api Base URI	Varchar(255)	*Base URI* untuk *API* klien
Application Type	Char(1)	Tipe aplikasi klien
Contact name	Varchar(255)	Nama kontak klien
Contact email	Varchar(255)	Email klien
Preauthorized	Numeric(1)	Status autorisasi klien
Grant types	Varchar(80)	Tipe hak akses klien
Scope	Varchar(4000)	Cakupan hak akses klien
Sandbox	Numeric(1)	Status *sandbox* klien
Is Moderated	Numeric(1)	Status moderasi klien

3.2.2.4 Perancangan Tabel Device Token

Tabel *device token* digunakan untuk menyimpan data *device token*. Detail atribut tabel *device token* ditunjukkan pada Tabel 3.28.

Tabel 3.28 Tabel *Device Token*

Nama Atribut	Tipe Data	Deskripsi
Device token ID	Uniqueidentifier	*Primary key* tabel *device token*
Client ID	Uniqueidentifier	*Foreign key* tabel *client*
User ID	Uniqueidentifier	*Foreign key* tabel *user*
Device token	Varchar(255)	Data *device token*
Device type	Char(1)	Tipe perangkat
Active	Numeric(1)	Status aktif
Register date	Datetime	Tanggal registrasi *device token*
Invalidate date	Datetime	Tanggal invalidasi *device token*

3.2.2.5 Perancangan Tabel Push Notification Certificate

Tabel *push notification certificate* digunakan untuk menyimpan kode sertifikat push notifikasi. Detail atribut tabel *push notification certificate* ditunjukkan pada Tabel 3.29.

Tabel 3.29 Tabel *Push Notification Certificate*

Nama Atribut	Tipe Data	Deskripsi
Cert ID	Uniqueidentifier	*Primary key* tabel *certificate*
Client ID	Uniqueidentifier	*Foreign key* tabel klien
Cert/key	text	Terdapat dua tipe penyimpanan masing-masing: 1. untuk Google FCM yang disimpan adalah *key* 2. untuk APNs yang disimpan adalah *URL* file .pem *certificate*.
Type	Char(1)	Tipe perangkat. “A” untuk android, “I” untuk apple

3.2.2.6 Perancangan Tabel *OAuth Access Token*

Tabel *oauth access token* digunakan untuk menyimpan data *client access token*. Detail atribut tabel *oauth access token* ditunjukkan pada Tabel 3.30.

Tabel 3.30 Tabel *OAuth Access Token*

Nama Atribut	Tipe Data	Deskripsi
Access Token	Varchar(40)	*Primary key* tabel *access token*
Client ID	Uniqueidentifier	*Foreign key* tabel klien
User ID	Uniqueidentifier	*Foreign key* tabel *user*
Expires	datetime	Tanggal kadaluarsa *access token*
Scope	Varchar(4000)	Cakupan akses token

3.2.2.7 Perancangan Tabel *Push Notification Destination*

Tabel *push notification destination* digunakan untuk menyimpan mapping antara *OAuth Access Token* dengan *Role* dan *Unit*. Detail atribut tabel *push notification destination* ditunjukkan pada Tabel 3.31.

Tabel 3.31 Tabel *Push Notification Destination*

Nama Atribut	Tipe Data	Deskripsi
ID Push Notification Destination	Uniqueidentifier	*Primary key* tabel *push notification destination*
Access Token	Varchar(40)	*Foreign key* tabel *access token*
Role ID	Int	*Foreign key* tabel *role*
Unit ID	Int	*Foreign key* tabel *unit*

3.2.2.8 Perancangan Tabel *Role*

Tabel *role* digunakan untuk menyimpan data peran pengguna (*user role*). Detail atribut tabel *role* ditunjukkan pada Tabel 3.32.

Tabel 3.32 Tabel *Role*

Nama Atribut	Tipe Data	Deskripsi
Role ID	Int	*Primary key* tabel *role*
Unit ID	Int	*Foreign key* tabel *unit*
Name	Varchar(80)	Nama peran pengguna
Description	Varchar(255)	Deskripsi peran pengguna
Scope	Varchar(4000)	Cakupan peran pengguna
Create date	Datetime	Waktu pembuatan data
Last update	Datetime	Waktu perubahan data terakhir

3.2.2.9 Perancangan Tabel *Unit*

Tabel *unit* digunakan untuk menyimpan data *unit* pengguna. Detail atribut tabel *unit* ditunjukkan pada Gambar 3.3.

Tabel 3.33 Tabel *Unit*

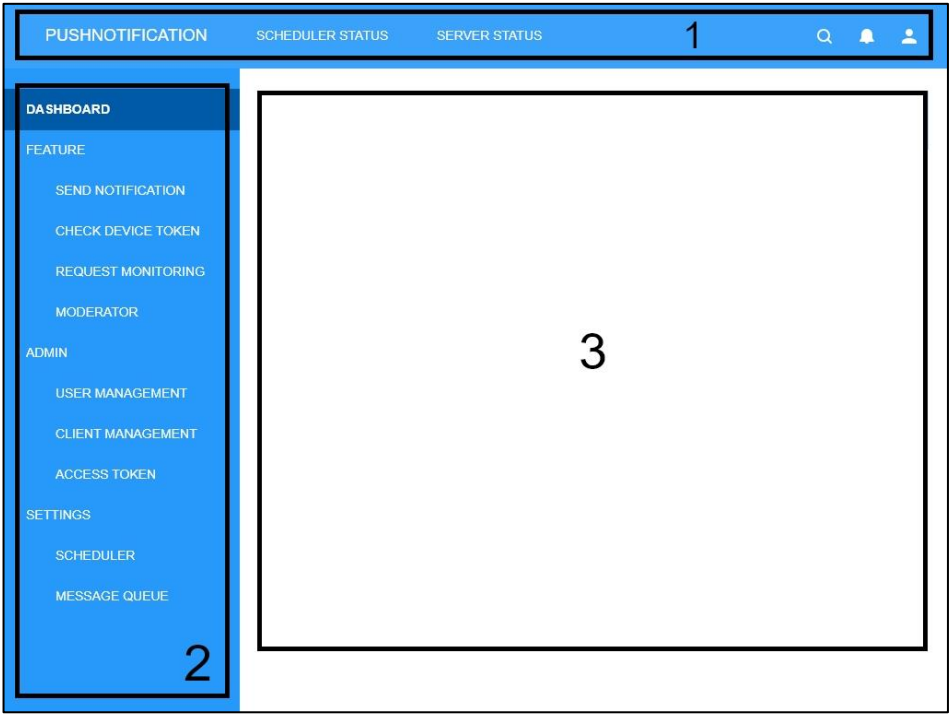
Nama Atribut	Tipe Data	Deskripsi
Unit ID	Int	*Primary key* tabel *unit*
Parent ID	Int	*Foreign key* tabel *unit*. Menyatakan relasi apakah sebuah unit memiliki parent atau tidak
Name	Varchar(80)	Nama unit pengguna
Description	Varchar(255)	Deskripsi unit pengguna
Create date	Datetime	Waktu pembuatan data
Last update	Datetime	Waktu perubahan data terakhir

3.2.3 Perancangan Antarmuka

Pada subbab ini akan dibahas dengan terperinci dari rancangan antarmuka untuk aplikasi pengelola konten notifikasi.

3.2.3.1 Tata Letak Dasar Antarmuka

Pada bagian ini akan dijelaskan susunan tata letak atau *layout* dasar yang akan diimplementasikan di semua halaman. Detail tata letak dasar antarmuka dapat dilihat pada Gambar 3.24 dan Tabel 3.34



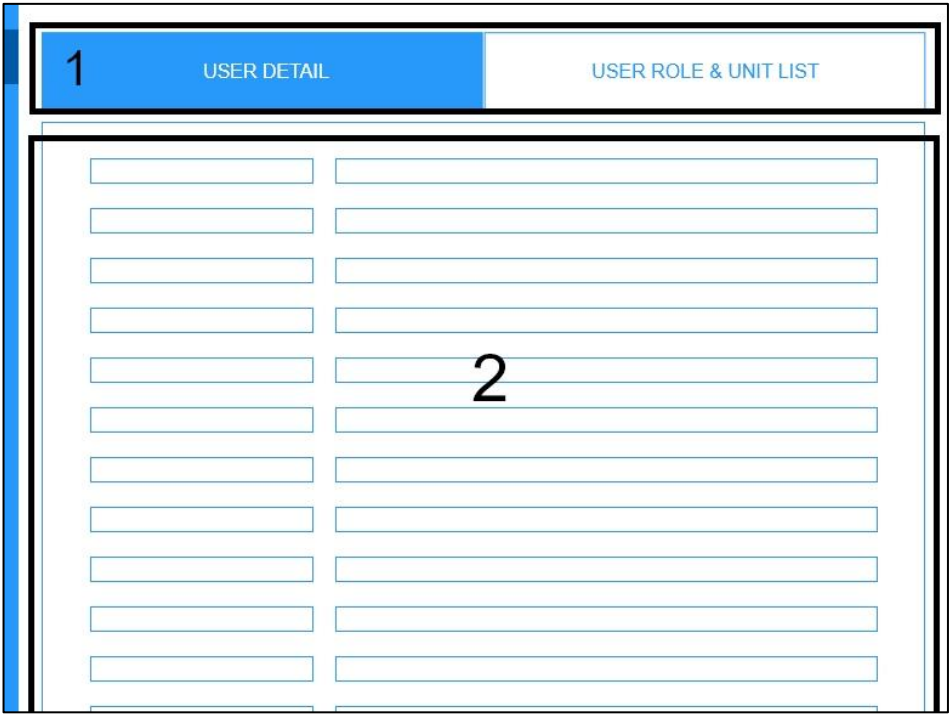
Gambar 3.24 Rancangan Tata Letak Dasar Antarmuka

Tabel 3.34 Deskripsi Tata Letak Dasar Antarmuka

No.	Nama Bagian Antarmuka	Deskripsi
1	*Navigation Bar*	Bagian *graphical user interface* (*GUI*) yang berisikan logo aplikasi, status server, status *message queuing* dan status *scheduler*. Juga terdapat menu untuk keluar dan mengatur profil pengguna. Terletak pada bagian atas layar.
2	*Sidebar Menubar*	Bagian *GUI* yang berisi daftar menu yang dapat diakses oleh aktor aplikasi.
3	*Content*	Bagian *GUI* yang berisi konten dari fitur aplikasi. Seluruh operasi aplikasi pengelola konten notifikasi dilakukan pada bagian ini.

3.2.3.2 Antarmuka Halaman Pembuka (Dashboard)

Antarmuka halaman pembuka merupakan antarmuka yang ditampilkan ketika aplikasi pertama kali dijalankan. Detail rancangan antarmuka halaman pembuka dapat dilihat pada Gambar 3.25 dan Tabel 3.35



Gambar 3.25 Rancangan Antarmuka Halaman Pembuka

Tabel 3.35 Atribut Antarmuka Halaman Pembuka (*Dashboard)*

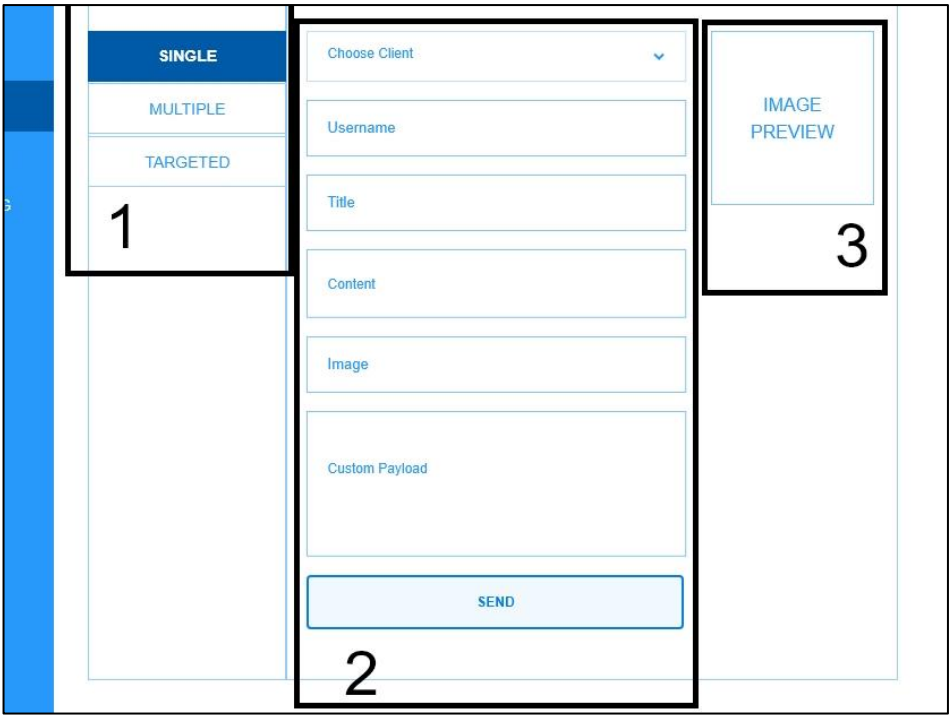
No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Tab_panel*	*Button*	Tombol untuk memindahkan *tab panel*	*Button click*
2	*Content*	*Div*	Berisi data detail pengguna dan detail *user role and unit*	-

3.2.3.3 Antarmuka Halaman Kirim Notifikasi (Send Notification)

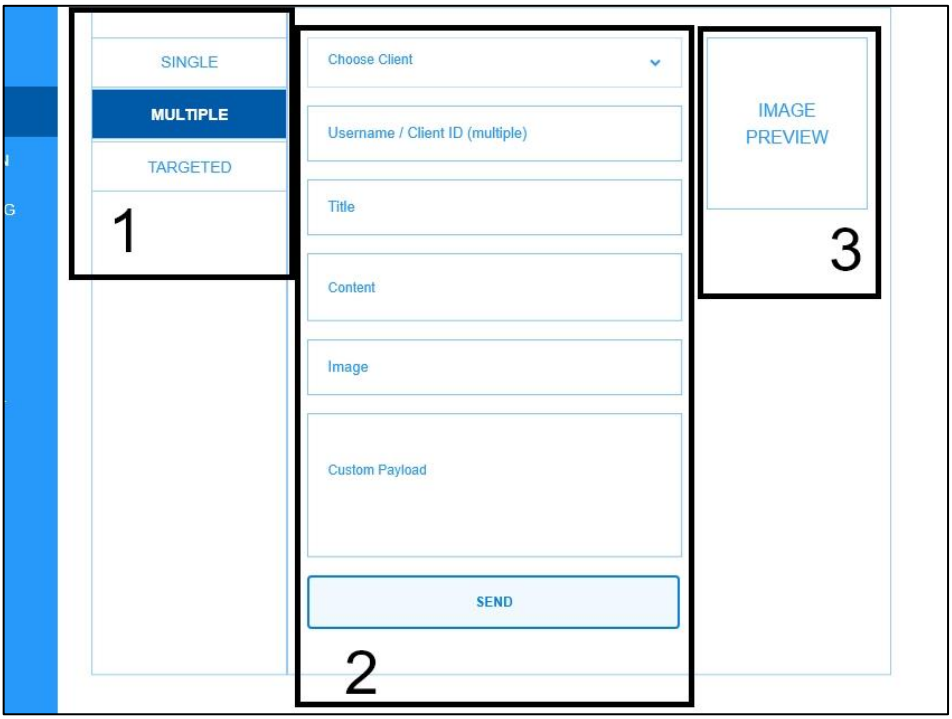
Antarmuka halaman kirim notifikasi merupakan antarmuka untuk mengirim notifikasi. Detail rancangan antarmuka dapat dilihat pada Gambar 3.26, Gambar 3.27, Gambar 3.28 dan Tabel 3.36

Tabel 3.36 Atribut Antarmuka Halaman Kirim Notifikasi

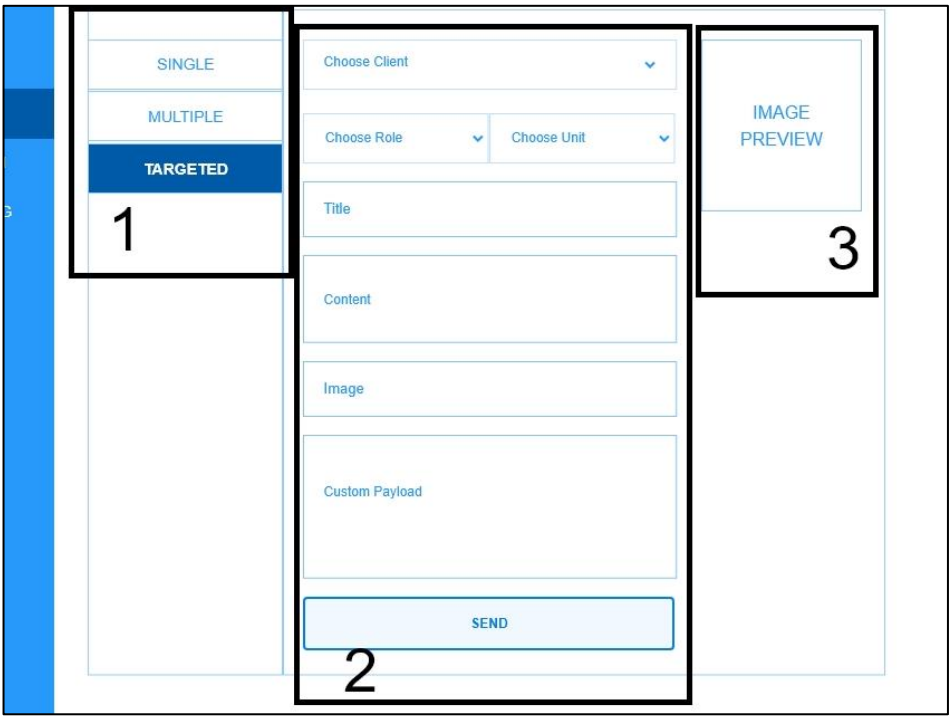
No.	Nama Atribut	Jenis	Kegunaan	Masukan/ Keluaran
1	*Tab_panel*	*Button*	Tombol untuk memindahkan *tab panel* sesuai metode pengiriman notifikasi 1. Single 2. Multiple (Integra ID atau User ID) 3. Targeted	*Button click*
2	*Form Input*	*Div*	Berisi *GUI* untuk memasukkan data dan parameter notifikasi	Data yang di*input* oleh aktor
3	*Image preview*	*Div*	GUI untuk menampilkan pratinjau gambar notifikasi	-



Gambar 3.26 Rancangan Antarmuka Halaman Kirim Notifikasi dengan Tipe *Single*



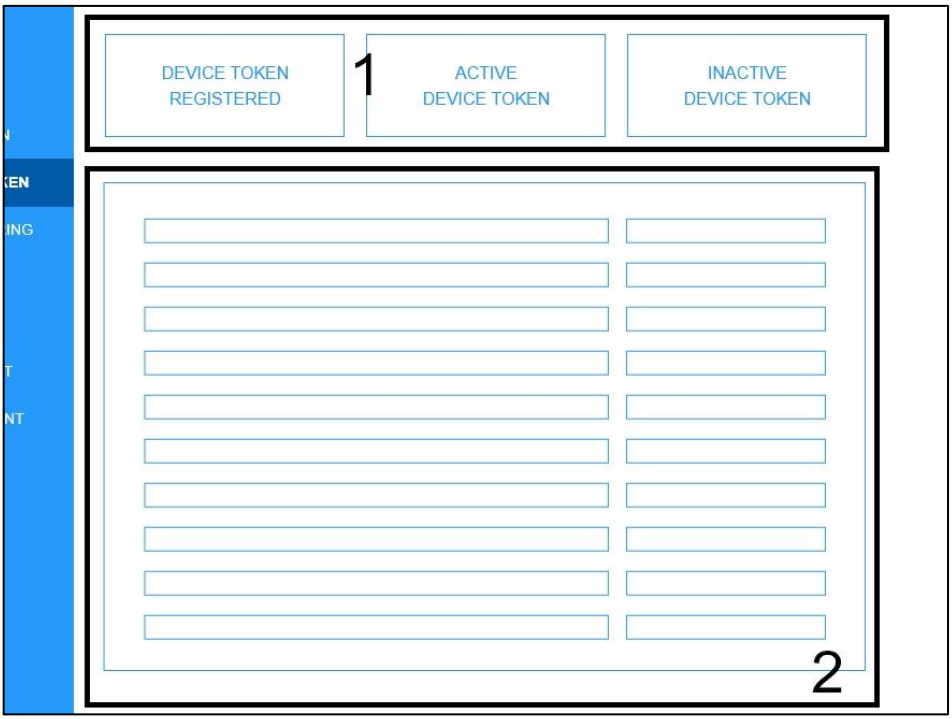
Gambar 3.27 Rancangan Antarmuka Halaman Kirim Notifikasi dengan Tipe *Multiple*



Gambar 3.28 Rancangan Antarmuka Halaman Kirim Notifikasi dengan Tipe *Targeted*

3.2.3.4 Antarmuka Halaman Cek Status Device Token (Check Device Token)

Antarmuka halaman cek status token perangkat menampilkan hasil status aktif *device token* pengguna. Detail rancangan antarmuka dapat dilihat pada Gambar 3.29 dan Tabel 3.37



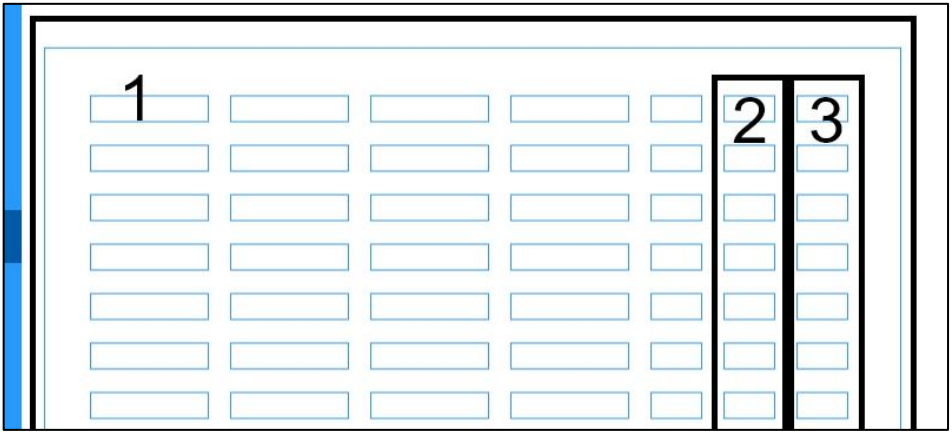
Gambar 3.29 Rancangan Antarmuka Halaman Cek Status *Device Token*

Tabel 3.37 Atribut Antarmuka Halaman Cek Status *Device Token*

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Widget*	*Div*	Berisi informasi: 1. Jumlah *Device token* yang teregistrasi 2. Jumlah *device token* yang aktif 3. Jumlah *device token* yang tidak aktif	-
2	*Content Table*	*Div*	Berisi data hasil cek status keaktifan *device token* (device token, status aktif (404/500))	-

3.2.3.5 Antarmuka Halaman Packet Monitoring

Antarmuka halaman *packet monitoring* merupakan antarmuka untuk melihat seluruh riwayat paket notifikasi yang masuk. Detail rancangan antarmuka dapat dilihat pada Gambar 3.30 dan Tabel 3.38.



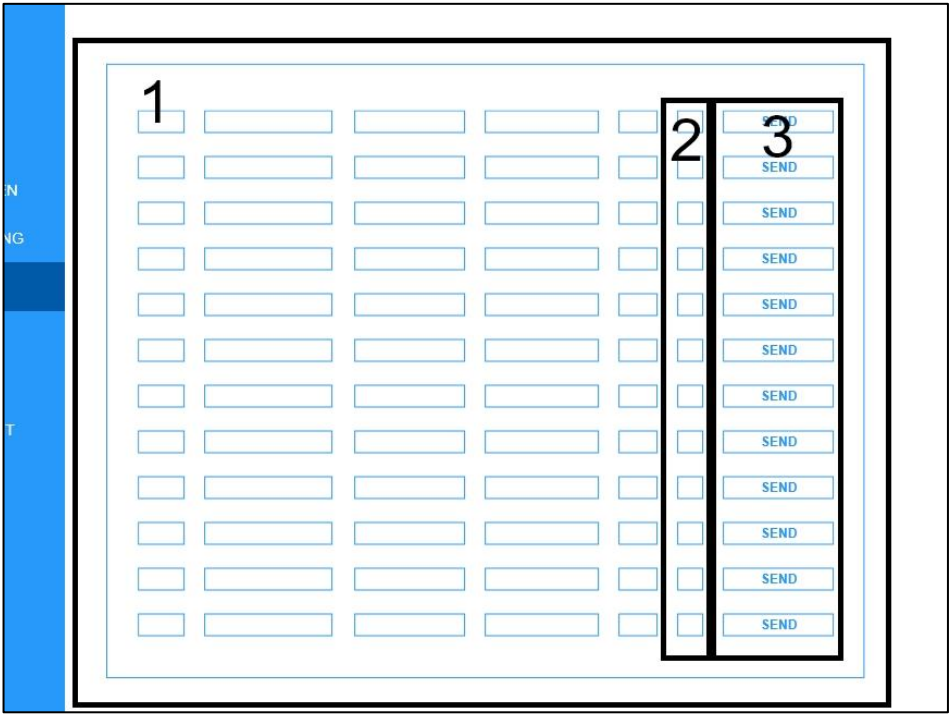
Gambar 3.30 Rancangan Antarmuka Halaman *Packet Monitoring*

Tabel 3.38 Atribut Antarmuka Halaman *Packet Monitoring*

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi detail packet notifikasi	-
2	Tombol pratinjau notifikasi	*Button*	Menampilkan modal berisikan pratinjau notifikasi yang akan/telah dikirim	*Button click*
3	Tombol hapus notifikasi	*Button*	Menghapus data notifikasi	*Button click*

3.2.3.6 Antarmuka Halaman Moderasi

Antarmuka halaman moderasi merupakan antarmuka yang ditampilkan untuk melakukan moderasi konten notifikasi. Detail rancangan antarmuka dapat dilihat pada Gambar 3.31 dan Tabel 3.39.



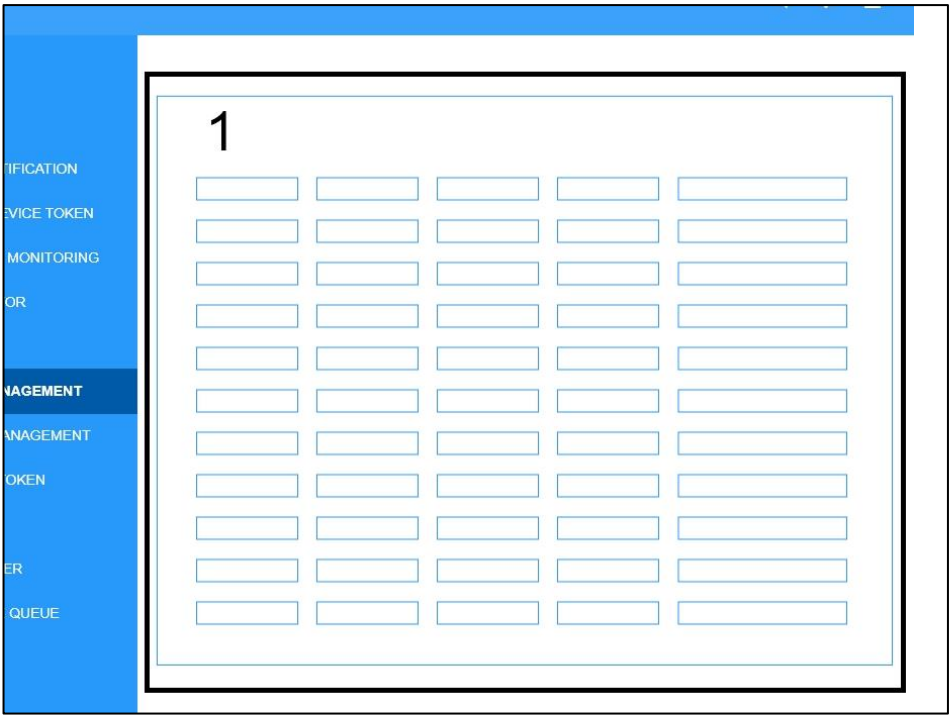
Gambar 3.31 Rancangan Antarmuka Halaman Moderasi

Tabel 3.39 Atribut Antarmuka Halaman Moderasi

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi detail packet notifikasi yang akan di moderasi	-
2	Tombol pratinjau notifikasi	*Button*	Menampilkan modal berisikan pratinjau notifikasi yang akan/telah dikirim	*Button click*
3	Tombol moderasi notifikasi	*Button*	Menampilkan modal berisikan pengaturan waktu dan tanggal pengiriman notifikasi yang telah dilakukan peninjauan konten (moderasi)	*Button click*

3.2.3.7 Antarmuka Halaman Pengguna

Antarmuka halaman pengguna merupakan antarmuka untuk melihat data pengguna. Detail rancangan antarmuka dapat dilihat pada Gambar 3.32 dan Tabel 3.40.



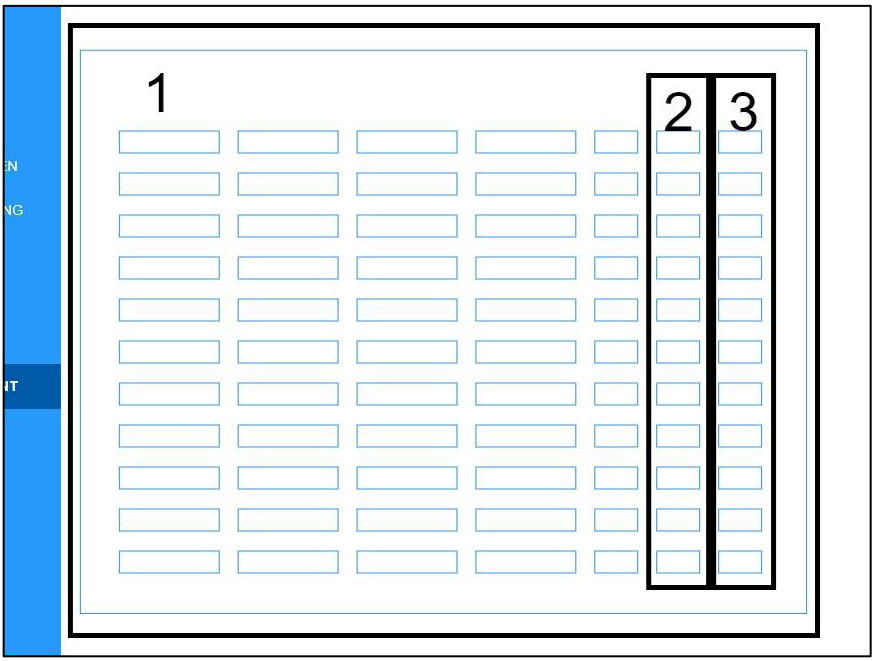
Gambar 3.32 Rancangan Antarmuka Halaman Manajemen Pengguna

Tabel 3.40 Atribut Antarmuka Halaman Pengguna

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi data pengguna	-

3.2.3.8 Antarmuka Halaman Manajemen Klien

Antarmuka halaman manajemen klien merupakan antarmuka untuk mengubah status moderasi klien, serta untuk menuju halaman manajemen sertifikat. Rancangan antarmuka dapat dilihat pada Gambar 3.33 dan Tabel 3.35.



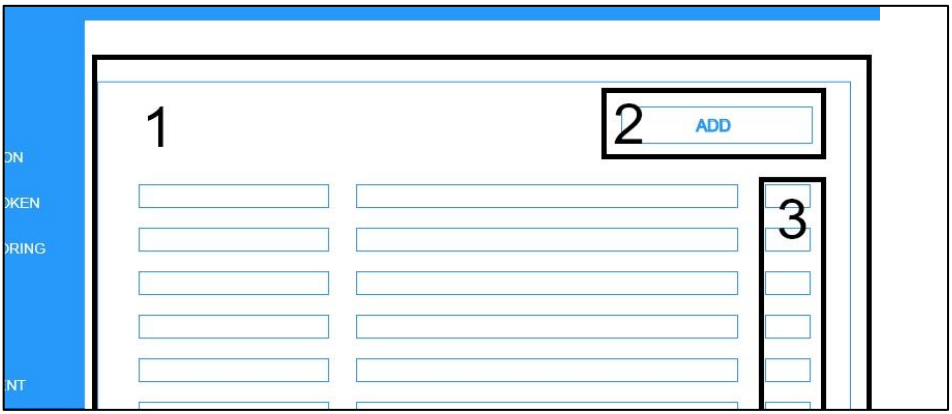
Gambar 3.33 Rancangan Antarmuka Halaman Manajemen Klien

Tabel 3.41 Atribut Antarmuka Halaman Manajemen Klien

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi data klien	-
2	Tombol edit data klien	*Button*	Menampilkan modal untuk mengubah status moderasi klien	*Button click*
3	Tombol manajemen sertifikat	*Button*	Menampilkan halaman manajemen sertifikat untuk klien	*Button click*

3.2.3.9 Antarmuka Halaman Manajemen Sertifikat

Antarmuka halaman manajemen sertifikat merupakan halaman dimana bisa dilakukan operasi *create*, *read* dan *delete* data sertifikat bagi klien. Rancangan antarmuka dapat dilihat pada Gambar 3.34 dan Tabel 3.42.



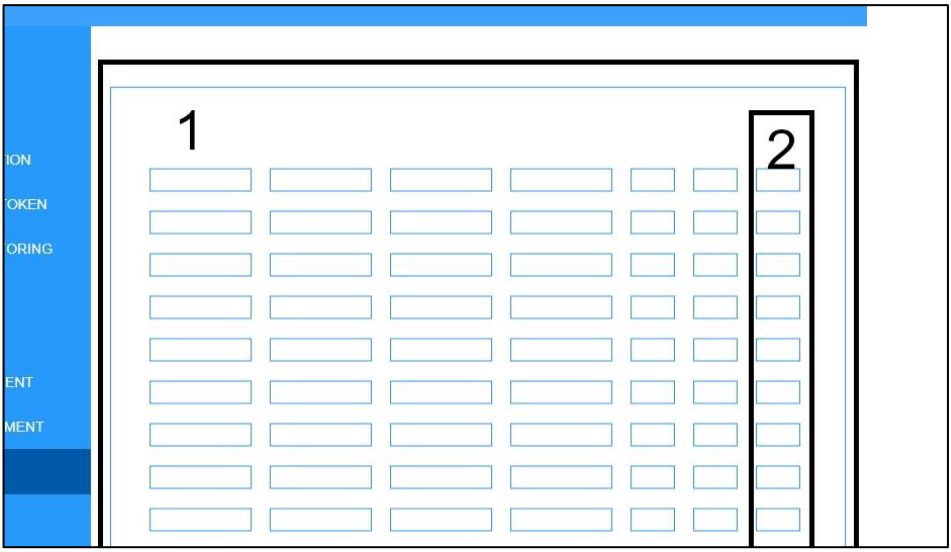
Gambar 3.34 Rancangan Antarmuka Halaman Manajemen Sertifikat

Tabel 3.42 Atribut Antarmuka Halaman Manajemen Sertifikat

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi data sertifikat per klien	-
2	Tombol tambah data sertifikat	*Button*	Menampilkan modal untuk menambah data sertifikat	*Button click*
3	Tombol hapus data sertifikat	*Button*	Menghapus data sertifikat	*Button click*

3.2.3.10 Antarmuka Halaman Manajemen Token Akses

Antarmuka halaman manajemen token akses. Rancangan antarmuka dapat dilihat pada Gambar 3.35.



Gambar 3.35 Rancangan Antarmuka Halaman Manajemen Token Akses

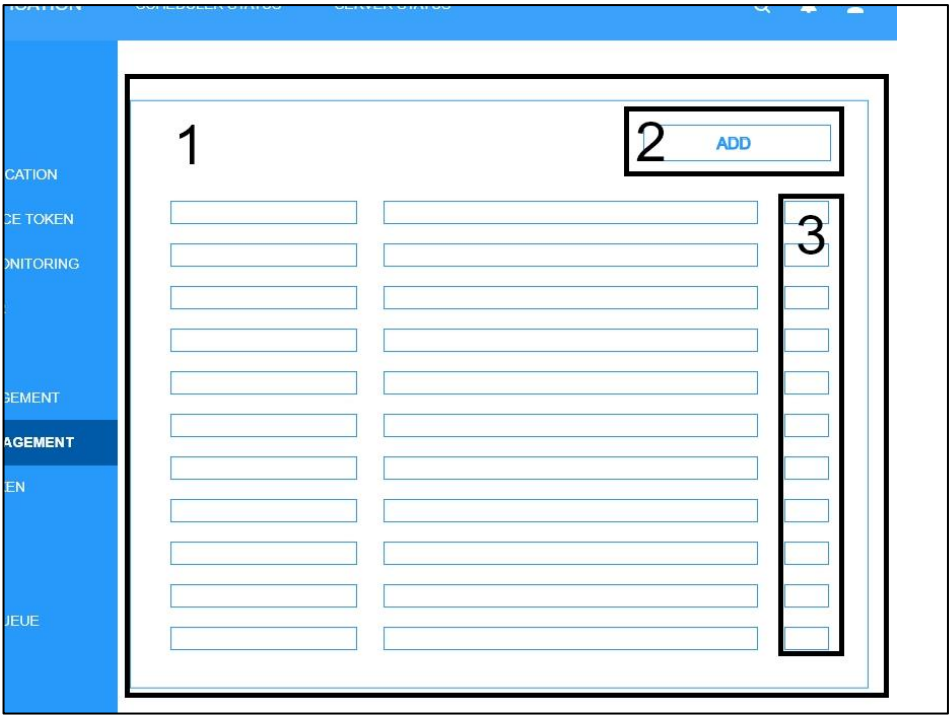
Tabel 3.43 Atribut Antarmuka Halaman Manajemen Sertifikat

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi data sertifikat per klien	-
2	Tombol tambah data sertifikat	*Button*	Menampilkan modal untuk menambah data sertifikat	*Button click*

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
3	Tombol hapus data sertifikat	*Button*	Menghapus data sertifikat	*Button click*

3.2.3.11 Antarmuka Halaman Manajemen Target Pengiriman Notifikasi

Antarmuka halaman manajemen target pengiriman notifikasi merupakan antarmuka untuk menambahkan target pengiriman notifikasi yang dikelompokkan per *access token*. Target pengiriman notifikasi berupa *pair-value role* pengguna dan *unit* pengguna. Detail rancangan antarmuka dapat dilihat pada Gambar 3.36 dan Tabel 3.44.



Gambar 3.36 Rancangan Antarmuka Halaman Manajemen Target Pengiriman Notifikasi

Tabel 3.44 Atribut Antarmuka Halaman Manajemen Target Pengiriman Notifikasi

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Masukan/ Keluaran
1	*Content table*	*Div*	Berisi informasi data target pengiriman notifikasi per *access token*	-
2	Tombol hapus data target pengiriman notifikasi	*Button*	Menghapus data notifikasi	*Button click*

3.2.3.12 Antarmuka Halaman *Scheduler* dan *Message Queuing*

Antarmuka halaman *scheduler* merupakan halaman untuk menjalankan *scheduler* pada aplikasi *push notification* terpusat. Terdapat sebuah tombol yang mengirim post request untuk melakukan *start scheduler*. Rancangan antarmuka dapat dilihat pada Gambar 3.37.

Gambar 3.37 Rancangan Antarmuka Halaman *Scheduler*

Antarmuka halaman *message queuing* merupakan halaman untuk menjalankan *message queuing* pada aplikasi *push notification* terpusat. Terdapat sebuah tombol yang mengirim

post request untuk melakukan *start message queue*. Rancangan antarmuka dapat dilihat pada Gambar 3.38

Gambar 3.38 Rancangan Antarmuka Halaman *Message Queuing*

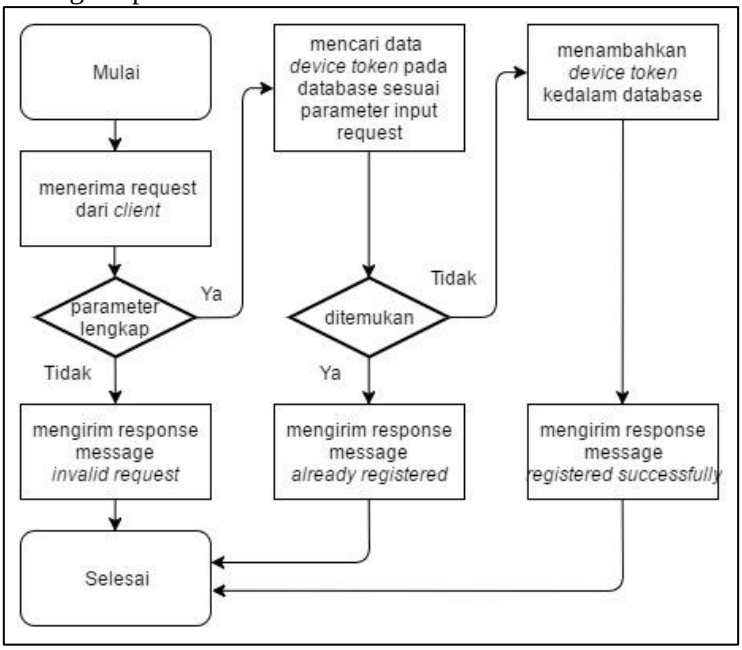
3.2.4 Perancangan Proses

Pada Subbab ini menjelaskan tentang perancangan proses-proses yang ada pada perangkat lunak. Berikut ini merupakan rancangan proses proses yang ada pada pengembangan Tugas Akhir ini.

3.2.4.1 Proses Pada Aplikasi *Push Notification* Terpusat

3.2.4.1.1 Proses *Register Device Token*

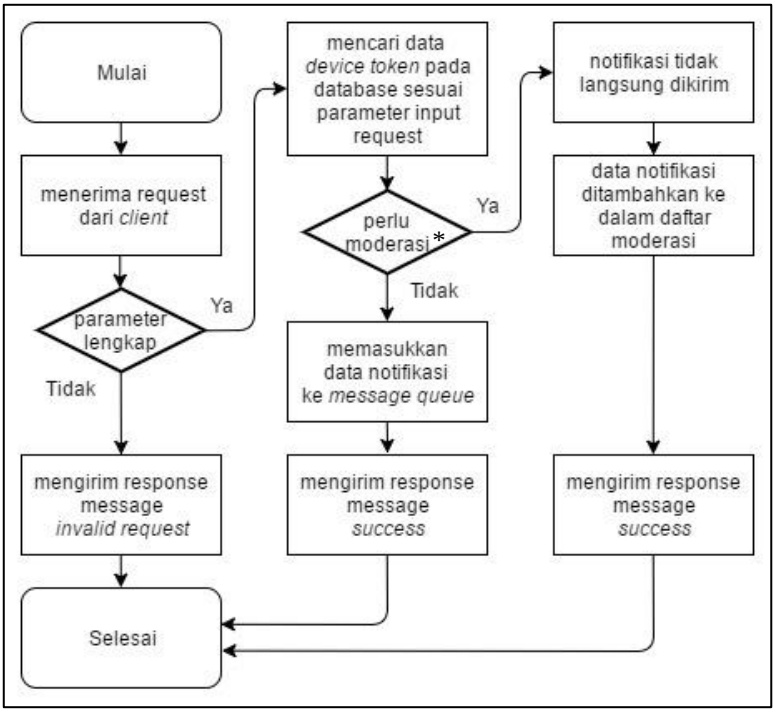
Proses ini Proses *register device token* bertujuan untuk melakukan registrasi kode *token* perangkat berdasarkan *client ID* serta *user ID* agar nantinya aplikasi *client* dapat melakukan penerimaan push notifikasi. Terdapat beberapa proses yang dilakukan untuk melakukan *register device token* sebagaimana ditunjukkan dalam diagram alir pada Gambar 3.39. Proses ini mengacu pada UC001.



Gambar 3.39 Diagram Alir Proses *Register Device Token*

3.2.4.1.2 Proses Segmentasi Pengiriman Notifikasi

Proses ini bertujuan untuk melakukan pengiriman notifikasi menuju perangkat pengguna secara spesifik yaitu dengan tipe *single*, *multiple* atau *targeted*. Terdapat beberapa proses yang dilakukan untuk melakukan segmentasi pengiriman notifikasi sebagaimana ditunjukkan dalam diagram alir pada Gambar 3.40. Proses ini mengacu pada UC002.



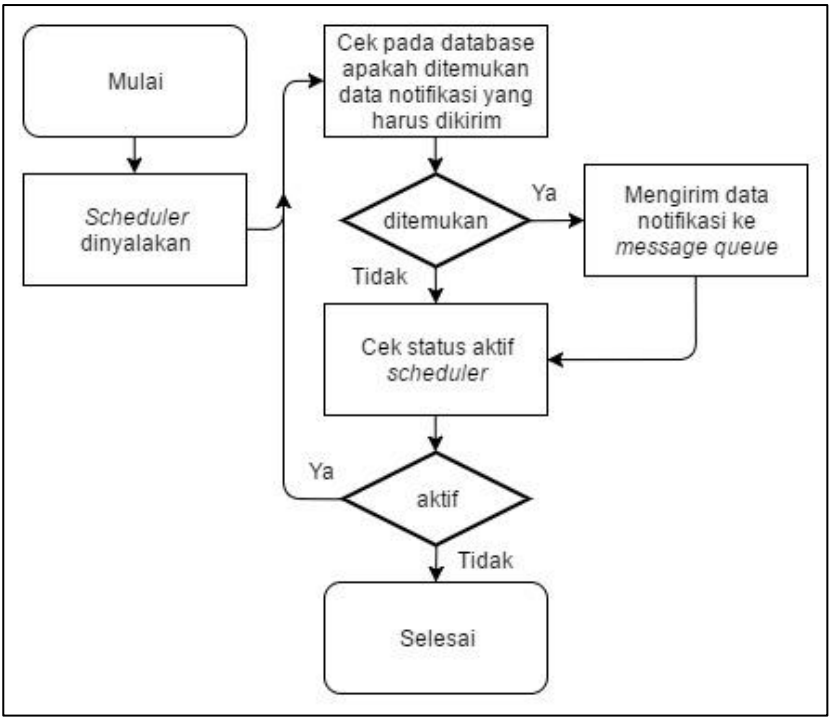
Gambar 3.40 Diagram Alir Proses Segmentasi Pengiriman Notifikasi

Keterangan

* Moderasi adalah peninjauan konten notifikasi oleh *administrator* sebelum notifikasi dikirim menuju pengguna

3.2.4.1.3 Proses *Scheduling*

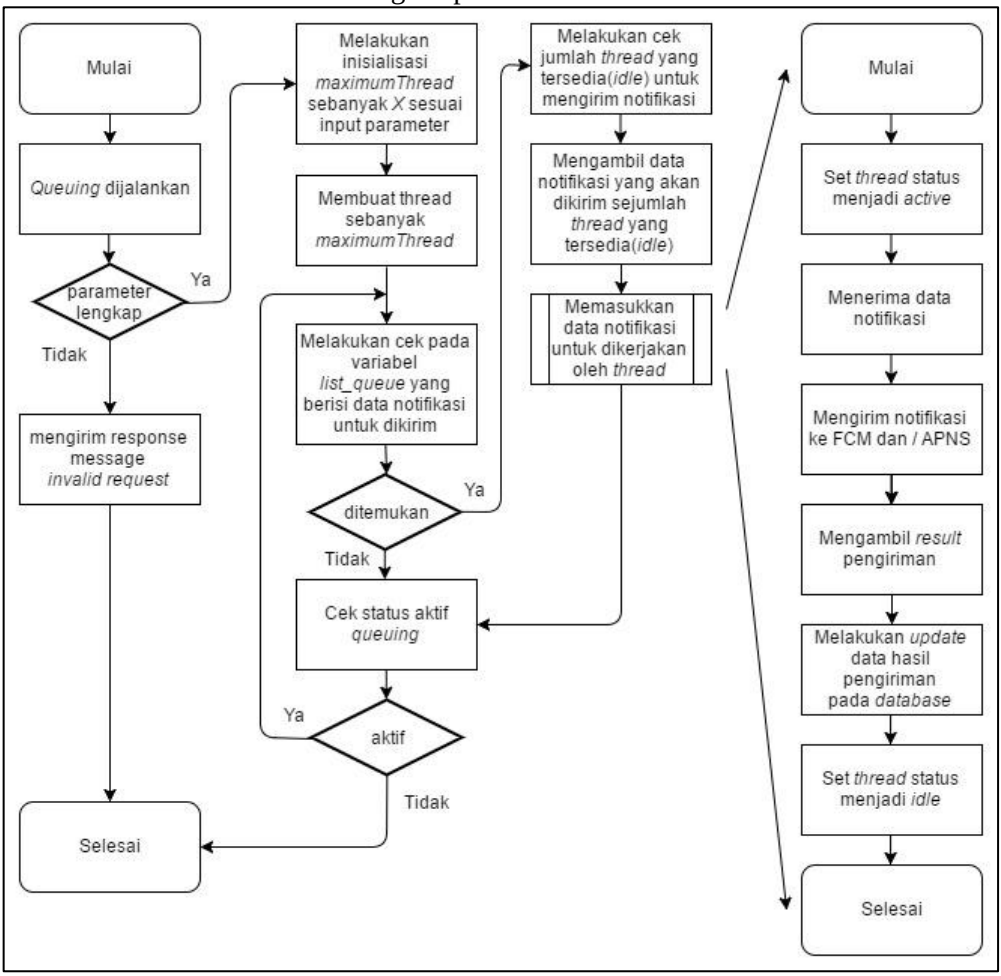
Proses *scheduling* bertujuan untuk melakukan perintah pengiriman notifikasi sesuai dengan waktu yang telah ditentukan. Terdapat beberapa proses yang dilakukan untuk melakukan *scheduling* sebagaimana ditunjukkan dalam diagram alir pada Gambar 3.41. Proses ini mengacu pada UC005.



Gambar 3.41 Diagram Alir Proses *Scheduling*

3.2.4.1.4 Proses *Message Queue*

Proses message queue bertujuan untuk melakukan *queuing* data notifikasi yang akan dikirim. Terdapat beberapa proses yang dilakukan untuk melakukan proses *queuing* sebagaimana ditunjukkan dalam diagram alir pada Gambar 3.42. Proses ini mengacu pada UC006.



Gambar 3.42 Diagram Alir Proses *Message Queue*

3.2.4.2 Proses Pada Aplikasi Pengelola Konten Notifikasi

Aplikasi Pengelola Konten Notifikasi merupakan aplikasi berupa situs web yang berfungsi untuk melakukan manajemen sekaligus monitoring Aplikasi Push Notification Terpusat yang berbasis Java. Secara keseluruhan proses data pada Aplikasi Pengelola Konten Notifikasi dapat dikelompokkan menjadi dua jenis proses. Penjelasannya adalah sebagai berikut.

1. Proses data melalui interaksi langsung dengan *database*.
2. Proses data melalui interaksi dengan API.
   Pada proses data jenis ini, dibutuhkan sebuah *library* tersendiri untuk melakukan interaksi dengan API.

3.2.5 Perancangan *API Endpoint*

Perancangan API Endpoint untuk Aplikasi *Push Notification Terpusat* dijelaskan pada.

Tabel 3.45 Perancangan *API Endpoint*

No	Endpoint URI	Method	Deskripsi
Mengirim Notifikasi			
1	/send/destination?access_token = {access_token}	GET	Mendapatkan daftar tujuan pengiriman notifikasi
2	/send/single	POST	Mengirim data notifikasi dengan tipe *single*
3	/send/multiple	POST	Mengirim data notifikasi dengan tipe *multiple*
4	/send/targeted	POST	Mengirim data notifikasi dengan tipe *targeted*
Cek *device token* status			

No	Endpoint URI	Method	Deskripsi
5	/check/token/{device_type}	POST	Melakukan perintah untuk melakukan cek status *device token*
Scheduler			
6	/scheduler/start	POST	Menyalakan *scheduler*
7	/scheduler/status?access_token = {access_token}	GET	Melihat status *scheduler*
Message Queuing			
8	/queue/start/{thread_number}	POST	Menyalakan *message queue*
9	/queue/add_thread	POST	Menambah jumlah *worker thread* untuk mengirim notifikasi
10	/queue/remove_thread	POST	Mengurangi jumlah *worker thread* untuk mengirim notifikasi
11	/queue/stop	POST	Memberhentikan *message queue*
12	/queue/show_status?access_token= {access_token}	GET	Melihat status *message queue*.
Register Device Token			
13	/register/device	POST	Melakukan registrasi device token aplikasi klien

{Halaman ini sengaja dikosongkan}

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi dari perancangan sistem sesuai dengan perancangan yang telah dibuat. Bahasa pemrograman yang digunakan untuk implementasi aplikasi *push notification* terpusat adalah bahasa pemrograman Java, sedangkan untuk aplikasi pengelola konten notifikasi dengan menggunakan bahasa pemrograman PHP.

4.1 Lingkungan Implementasi Perangkat Lunak

Lingkungan implementasi sistem yang digunakan untuk mengembangkan tugas akhir memiliki spesifikasi perangkat keras dan perangkat lunak seperti ditampilkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Sistem

Perangkat	Spesifikasi	
Perangkat keras	Server Prosesor	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
	Server Memori	512MB
	Perangkat Bergerak Android	Versi minimal SDK 19
	Perangkat Bergerak Apple	Dengan versi minimal iOS 11
Perangkat lunak	Sistem Operasi	Debian 9.4
	Perangkat Pengembang	Eclipse Kepler Service Versi 2
		Android Studio
	Perangkat Pembantu	Microsoft Word 365 Sublime Text 3 XAMPP

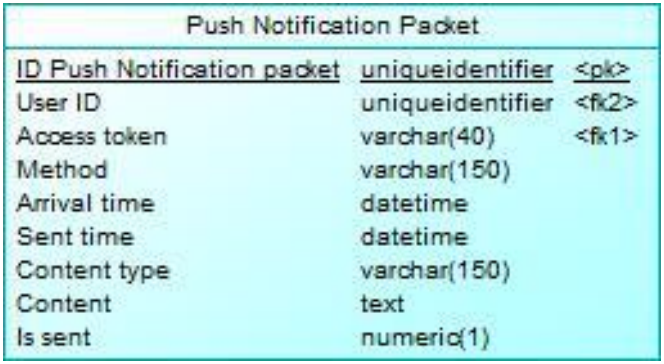
4.2 Implementasi Basis Data

4.2.1 Implementasi Tabel Push Notification Packet

Tabel *push notification packet* digunakan untuk menyimpan seluruh konten data packet untuk push notifikasi.

```
CREATE TABLE [dbo].[pn_packet](
	[id_pn_packet] [uniqueidentifier] NOT NULL,
	[user_id] [uniqueidentifier] NULL,
	[access_token] [varchar](40) NOT NULL,
	[method] [varchar](150) NULL,
	[arrival_time] [datetime] NULL,
	[sent_time] [datetime] NULL,
	[content_type] [varchar](150) NULL,
	[content] [text] NULL,
	[is_sent] [numeric](1, 0) NULL,
 CONSTRAINT [PK_PN_PACKET] PRIMARY KEY NONCLUSTERED
(
	[id_pn_packet] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
)ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Kode Sumber 4.1 Tabel *Push Notification Packet*



Gambar 4.1 Tabel *Push Notification Packet*

4.2.2 Implementasi Tabel *User Account*

Tabel *user account* digunakan untuk menyimpan data *user*.

```
CREATE TABLE [dbo].[user_account](
	[user_id] [uniqueidentifier] NOT NULL,
	[type_id] [int] NOT NULL,
	[name] [varchar](150) NOT NULL,
	[nicknames] [varchar](20) NULL,
	[username] [varchar](255) NOT NULL,
	[password] [char](128) NOT NULL,
	[salt] [char](128) NULL,
	[email] [varchar](255) NULL,
	[email_verified] [numeric](1, 0) NOT NULL,
	[alternate_email] [varchar](255) NULL,
	[alternate_email_verified] [numeric](1, 0) NOT
NULL,
	[phone] [varchar](18) NULL,
	[phone_verified] [numeric](1, 0) NOT NULL,
	[enabled] [numeric](1, 0) NULL,
	[picture] [varbinary](max) NULL,
	[gender] [char](1) NULL,
	[birthdate] [date] NULL,
	[zoneinfo] [varchar](40) NULL,
	[locale] [varchar](10) NULL,
	[integra_id] [bigint] NULL,
	[must_change_pwd] [numeric](1, 0) NULL,
	[sandbox] [numeric](1, 0) NULL,
	[locked] [datetime] NULL,
	[suspended] [datetime] NULL,
	[has_suspended] [numeric](1, 0) NULL,
	[create_date] [datetime] NULL,
	[last_update] [datetime] NULL,
 CONSTRAINT [PK_USER_ACCOUNT] PRIMARY KEY CLUSTERED
(
	[user_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Kode Sumber 4.2 Tabel *User Account*

User Account		
User ID	uniqueidentifier	<pk>
Type ID	int	<fk>
Name	varchar(150)	
Nickname	varchar(20)	
Username	varchar(255)	
Password	char(128)	
Salt	char(128)	
Email	varchar(255)	
Email verified	numeric(1)	
Alternate email	varchar(255)	
Alternate email verified	numeric(1)	
Phone	varchar(18)	
Phone verified	numeric(1)	
Enabled	numeric(1)	
Picture	varbinary(max)	
Gender	char(1)	
Birthdate	date	
Zoneinfo	varchar(40)	
Locale	varchar(10)	
Integra ID	bigint	
Must change password	numeric(1)	
Sandbox	numeric(1)	
Locked	datetime	
Suspended	datetime	
Has suspended	numeric(1)	
Create date	datetime	
Last update	datetime	

Gambar 4.2 Tabel *User Account*

4.2.3 Implementasi Tabel *OAuth Client*

Tabel *client* digunakan untuk menyimpan data *client*.

```
CREATE TABLE [dbo].[oauth_client](
	[client_id] [uniqueidentifier] NOT NULL,
	[user_id] [uniqueidentifier] NULL,
	[provider_id] [uniqueidentifier] NOT NULL,
	[client_name] [varchar](100) NOT NULL,
	[client_description] [varchar](250) NULL,
	[client_secret] [varchar](255) NOT NULL,
	[issued_at] [datetime] NULL,
	[expires_at] [datetime] NULL,
	[client_ip_cidr] [varchar](255) NULL,
	[logo] [varchar](100) NULL,
	[redirect_uri] [varchar](255) NULL,
	[base_uri] [varchar](255) NULL,
	[api_base_uri] [varchar](255) NULL,
	[app_type] [char](1) NULL,
	[contact_name] [varchar](255) NULL,
	[contact_email] [varchar](255) NULL,
	[preauthorized] [numeric](1, 0) NULL,
	[grant_types] [varchar](80) NULL,
	[scope] [varchar](4000) NULL,
	[sandbox] [numeric](1, 0) NULL,
	[is_moderated] [numeric](1, 0) NULL,
 CONSTRAINT [PK_OAUTH_CLIENT] PRIMARY KEY CLUSTERED
(
	[client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Kode Sumber 4.3 Tabel *OAuth Client*

OAuth Client		
Client ID	uniqueidentifier	<pk>
User ID	uniqueidentifier	<fk2>
Provider ID	uniqueidentifier	<fk1>
Client name	varchar(100)	
Client description	varchar(250)	
Client secret	varchar(255)	
Issued at	datetime	
Expires at	datetime	
Client IP address/CIDR	varchar(255)	
Logo	varchar(100)	
Redirect URI	varchar(255)	
Base URI	varchar(255)	
Api Base URI	varchar(255)	
Application type	char(1)	
Contact name	varchar(255)	
Contact email	varchar(255)	
Preauthorized	numeric(1)	
Grant types	varchar(80)	
Scope	varchar(4000)	
Sandbox	numeric(1)	
Is moderated	numeric(1)	

Gambar 4.3 Tabel *Oauth Client*

4.2.4 Implementasi Tabel *Device Token*

Tabel *device token* digunakan untuk menyimpan data *device token*.

```
CREATE TABLE [dbo].[device_token](
	[device_token_id] [uniqueidentifier] NOT NULL,
	[client_id] [uniqueidentifier] NOT NULL,
	[user_id] [uniqueidentifier] NOT NULL,
	[device_token] [varchar](255) NULL,
	[device_type] [char](1) NULL,
	[active] [numeric](1, 0) NULL,
	[reg_date] [datetime] NULL,
	[invalidate_date] [datetime] NULL,
 CONSTRAINT [PK_DEVICE_TOKEN] PRIMARY KEY CLUSTERED
(
	[device_token_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Kode Sumber 4.4 Tabel *Device Token*

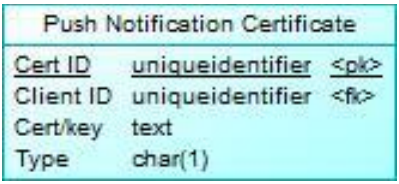
Gambar 4.4 Tabel *Device Token*

4.2.5 Implementasi Tabel Push Notification Certificate

Tabel *push notification certificate* digunakan untuk menyimpan kode sertifikat push notifikasi.

```
CREATE TABLE [dbo].[pn_certificate](
	[cert_id] [uniqueidentifier] NOT NULL,
	[client_id] [uniqueidentifier] NOT NULL,
	[cert_key] [text] NULL,
	[type] [char](1) NULL,
 CONSTRAINT [PK_PN_CERTIFICATE] PRIMARY KEY
NONCLUSTERED
(
	[cert_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Kode Sumber 4.5 Tabel *Push Notification Certificate*



Gambar 4.5 Tabel *Push Notification Certificate*

4.2.6 Implementasi Tabel OAuth Access Token

Tabel *oauth access token* digunakan untuk menyimpan data *client access token*.

```
CREATE TABLE [dbo].[oauth_access_token](
	[access_token] [varchar](40) NOT NULL,
	[client_id] [uniqueidentifier] NOT NULL,
	[user_id] [uniqueidentifier] NULL,
	[expires] [datetime] NOT NULL,
	[scope] [varchar](4000) NULL,
 CONSTRAINT [PK_OAUTH_ACCESS_TOKEN] PRIMARY KEY
CLUSTERED
(
	[access_token] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Kode Sumber 4.6 Tabel *OAuth Access Token*

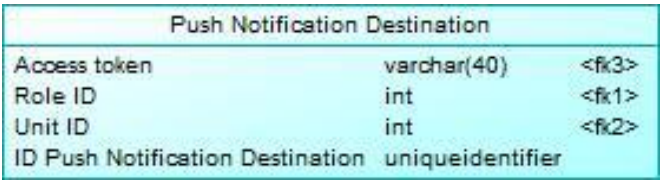
Gambar 4.6 Tabel *OAuth Access Token*

4.2.7 Implementasi Tabel Push Notification Destination

Tabel *push notification destination* digunakan untuk menyimpan mapping antara *OAuth Access Token* dengan *Role* dan *Unit*.

```
CREATE TABLE [dbo].[pn_destination](
	[access_token] [varchar](40) NOT NULL,
	[role_id] [int] NULL,
	[unit_id] [int] NULL,
	[id_pn_dest] [uniqueidentifier] NOT NULL
) ON [PRIMARY]
GO
```

Kode Sumber 4.7 Tabel *Push Notification Destination*



Gambar 4.7 Tabel *Push Notification Destination*

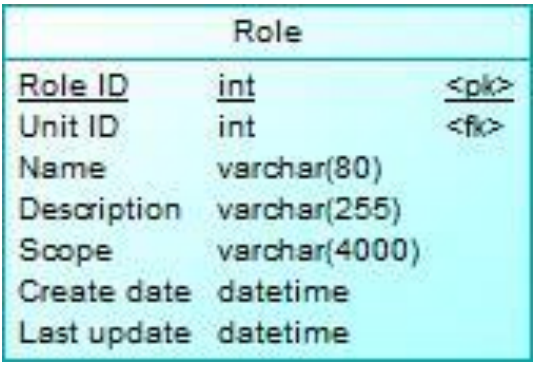
4.2.8 Implementasi Tabel *Role*

Tabel *role* digunakan untuk menyimpan data *role* pengguna.

```
CREATE TABLE [dbo].[user_role](
	[user_id] [uniqueidentifier] NOT NULL,
	[role_id] [int] NOT NULL,
	[unit_id] [int] NOT NULL,
	[client_id] [uniqueidentifier] NOT NULL,
	[issued_at] [datetime] NOT NULL,
	[expired_at] [datetime] NULL,
	[is_default] [numeric](1, 0) NULL,
	[create_date] [datetime] NULL,
	[last_update] [datetime] NULL,
 CONSTRAINT [PK_USER_ROLE] PRIMARY KEY CLUSTERED
(
	[user_id] ASC,
```

```
        [role_id] ASC,
        [unit_id] ASC,
        [client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Kode Sumber 4.8 Tabel *Push Notification Destination*



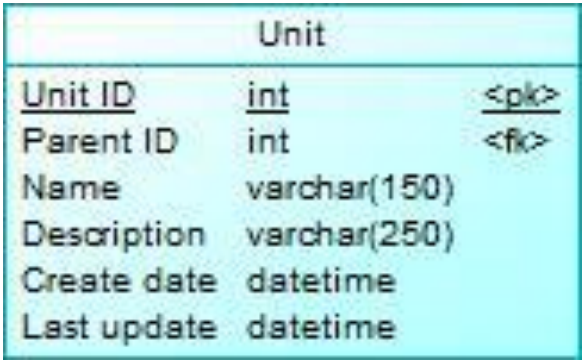
Gambar 4.8 Tabel *Role*

4.2.9 Implementasi Tabel *Unit*

Tabel *unit* digunakan untuk menyimpan data *unit* pengguna.

```
CREATE TABLE [dbo].[unit](
	[unit_id] [int] NOT NULL,
	[parent_id] [int] NULL,
	[name] [varchar](150) NOT NULL,
	[description] [varchar](250) NULL,
	[create_date] [datetime] NULL,
	[last_update] [datetime] NULL,
 CONSTRAINT [PK_UNIT] PRIMARY KEY CLUSTERED
(
	[unit_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Kode Sumber 4.9 Tabel *Unit*



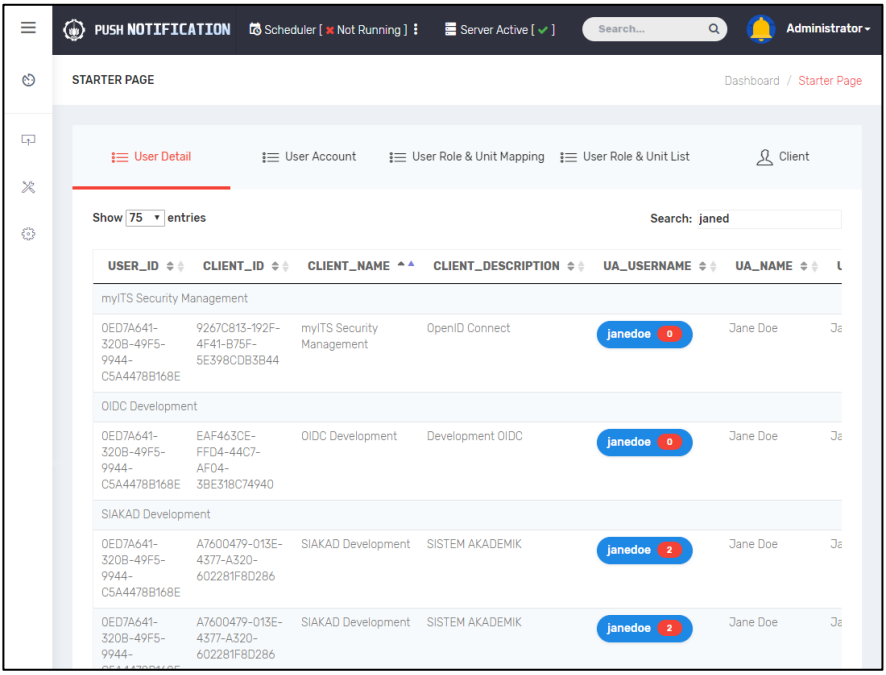
Gambar 4.9 Tabel *Unit*

4.3 Implementasi Antarmuka Perangkat Lunak

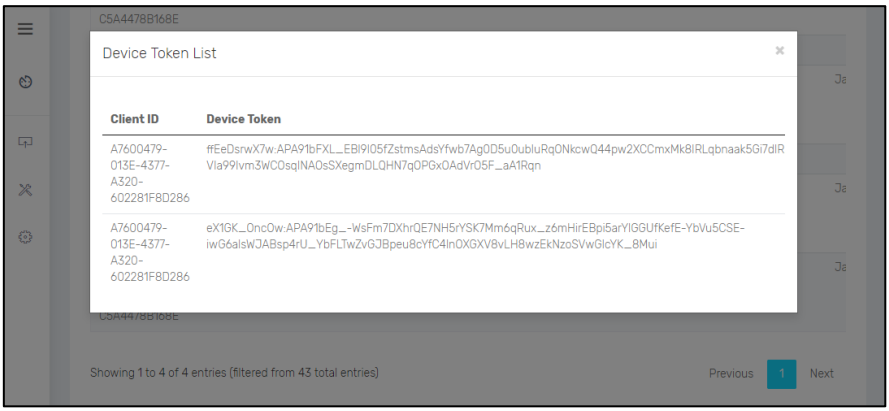
Implementasi tampilan antarmuka pengguna pada perangkat lunak hanya terdapat pada Aplikasi Pengelola Konten Notifikasi

4.3.1 Antarmuka Halaman Pembuka (Dashboard)

Implementasi halaman utama dapat dilihat pada Gambar 4.10. Pada *tab user detail* admin dapat melihat daftar *user* sekaligus mapping client dengan unitnya. Tombol berwarna biru jika di klik akan menampilkan *device_token* unik yang teregistrasi oleh user lalu dikelompokkan berdasarkan jenis perangkat (Android atau iOS) dan berdasarkan *client* nya. Antarmuka daftar *device_token* yang teregistrasi bagi pengguna dapat dilihat pada Gambar 4.11.



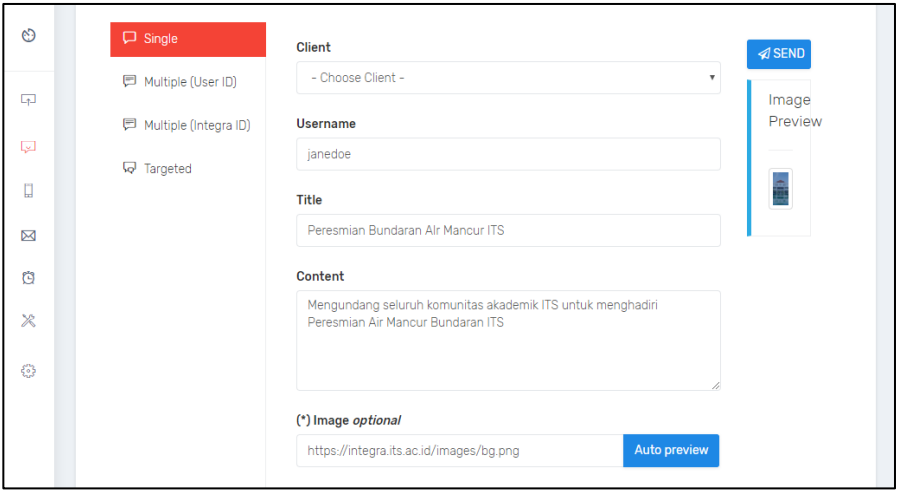
Gambar 4.10 Antarmuka Halaman Pembuka



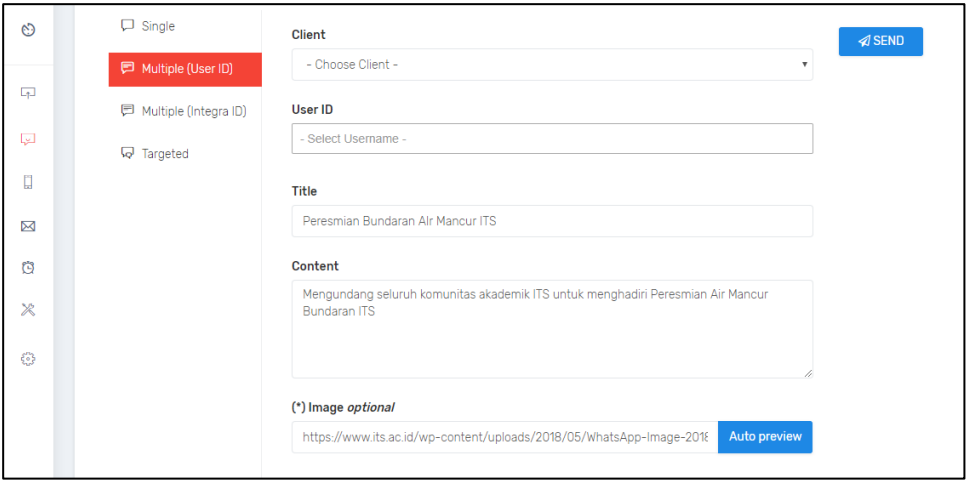
Gambar 4.11 Antarmuka Halaman Utama - Daftar *Device Token* per Pengguna

4.3.2 Antarmuka Halaman Kirim Notifikasi (*Send Notification*)

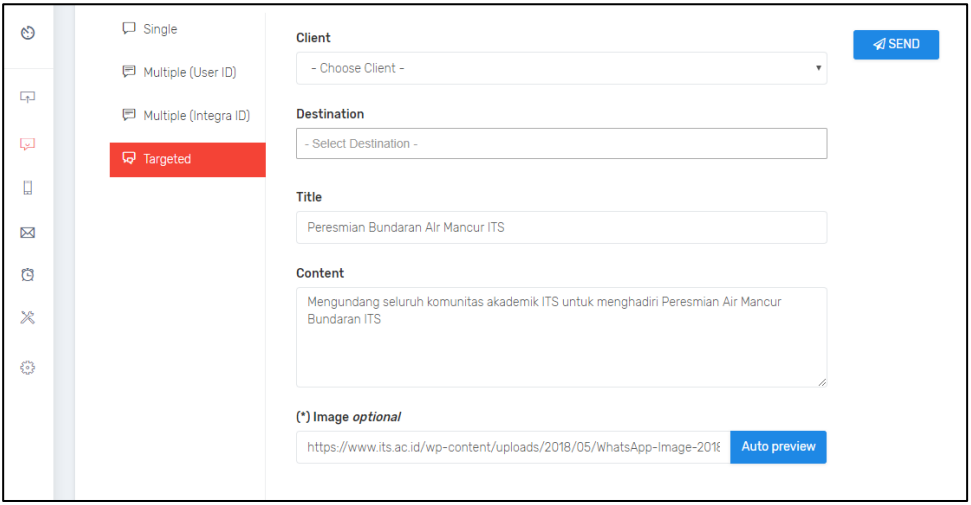
Implementasi halaman kirim notifikasi dapat dilihat pada Gambar 4.12, Gambar 4.13 dan Gambar 4.14.



Gambar 4.12 Antarmuka Halaman Send Notification - Tipe *Single*



Gambar 4.13 Antarmuka Halaman Send Notification - Tipe *Multiple*

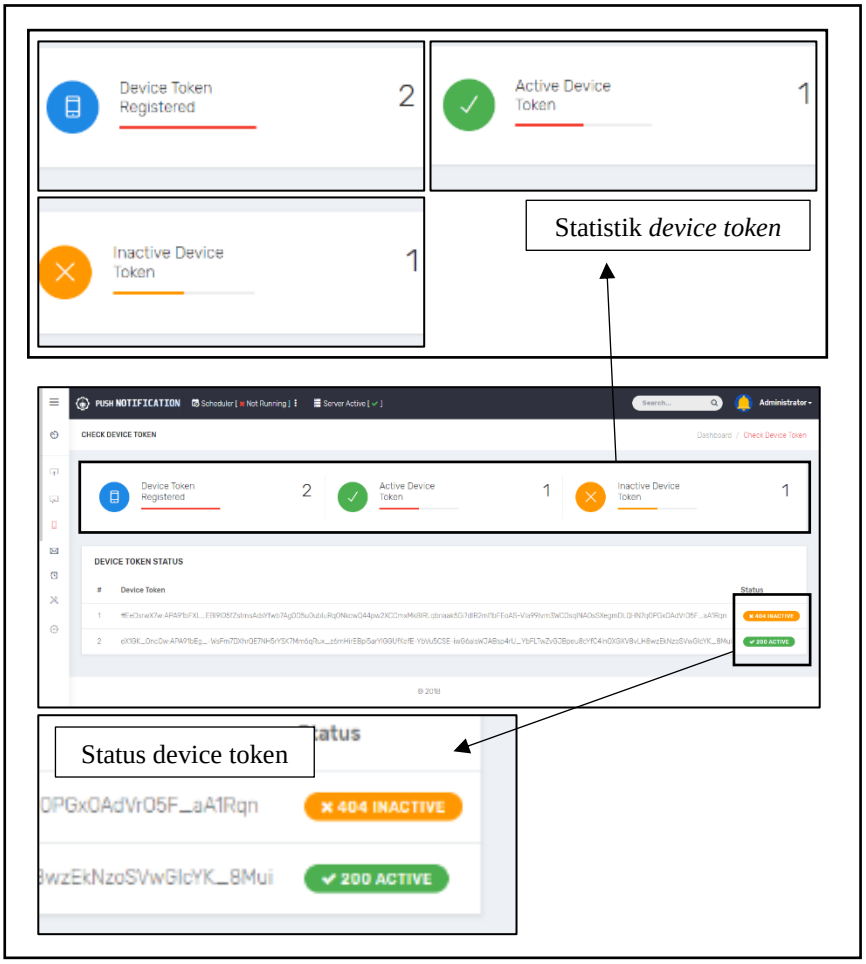


Gambar 4.14 Antarmuka Halaman Send Notification - Tipe *Targeted*

4.3.3 Antarmuka Halaman Cek Status *Device Token* (Check Device Token)

Implementasi halaman cek status *device token* dapat dilihat pada

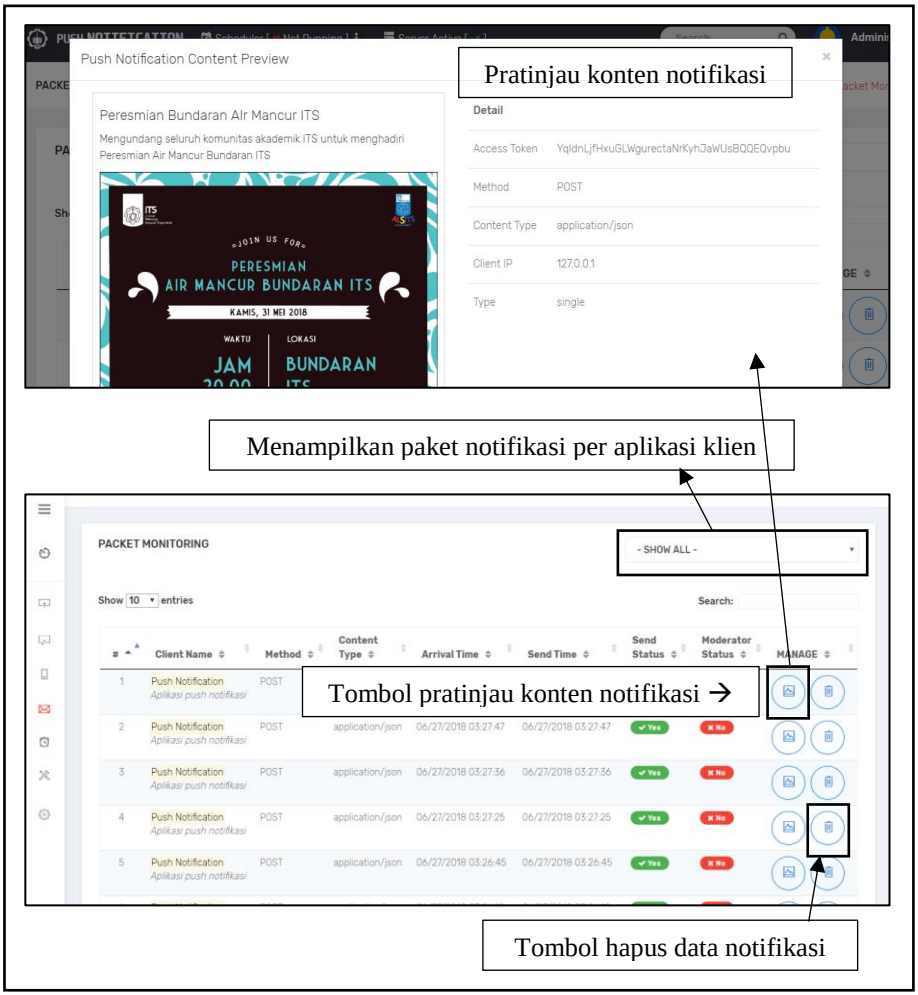
Gambar **4.15**.



Gambar 4.15 Antarmuka Halaman Cek Status *Device Token*

4.3.4 Antarmuka Halaman Packet Monitoring

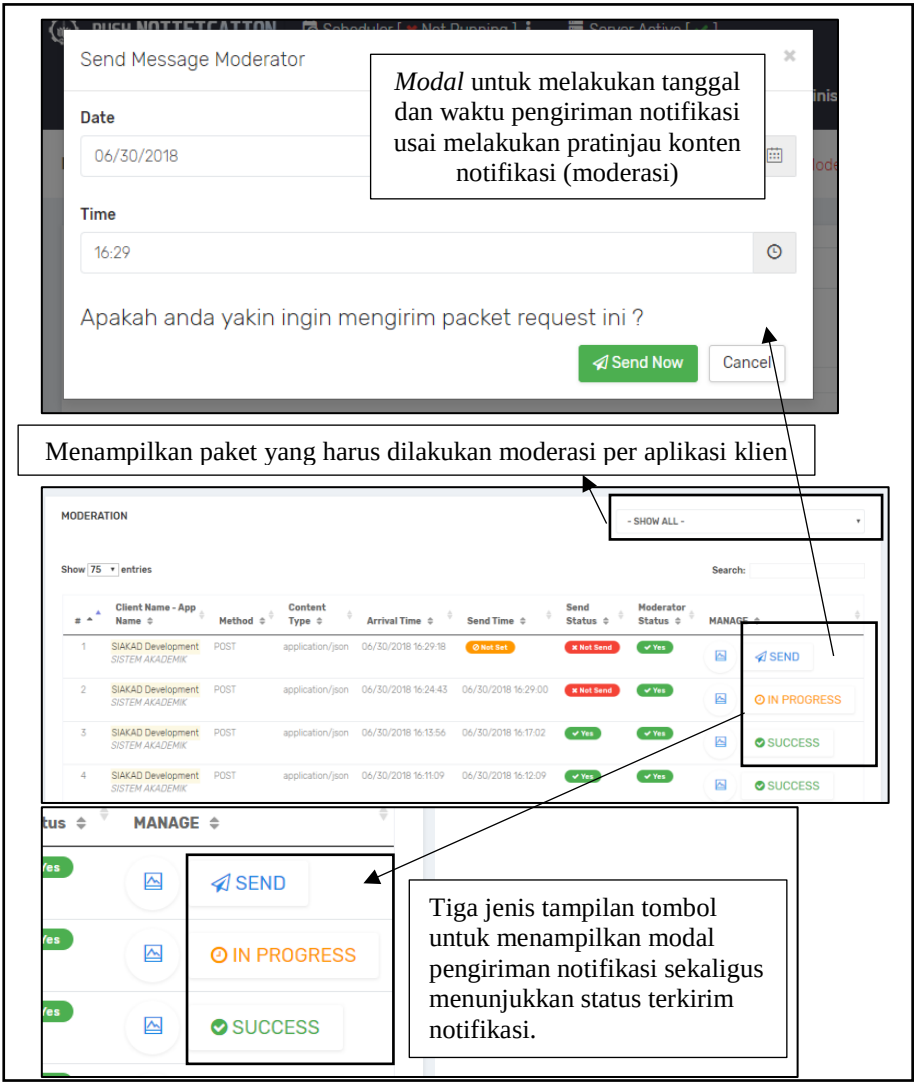
Implementasi halaman *packet monitoring* dapat dilihat pada Gambar 4.16.



Gambar 4.16 Antarmuka Halaman *Packet Monitoring*

4.3.5 Antarmuka Halaman Moderasi

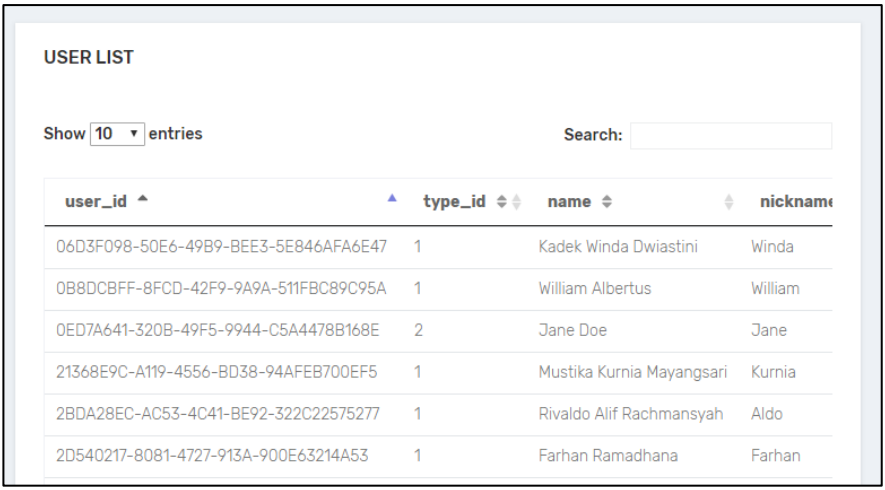
Implementasi dapat dilihat pada Gambar 4.17.



Gambar 4.17 Antarmuka Halaman Moderasi

4.3.6 Antarmuka Halaman Pengguna

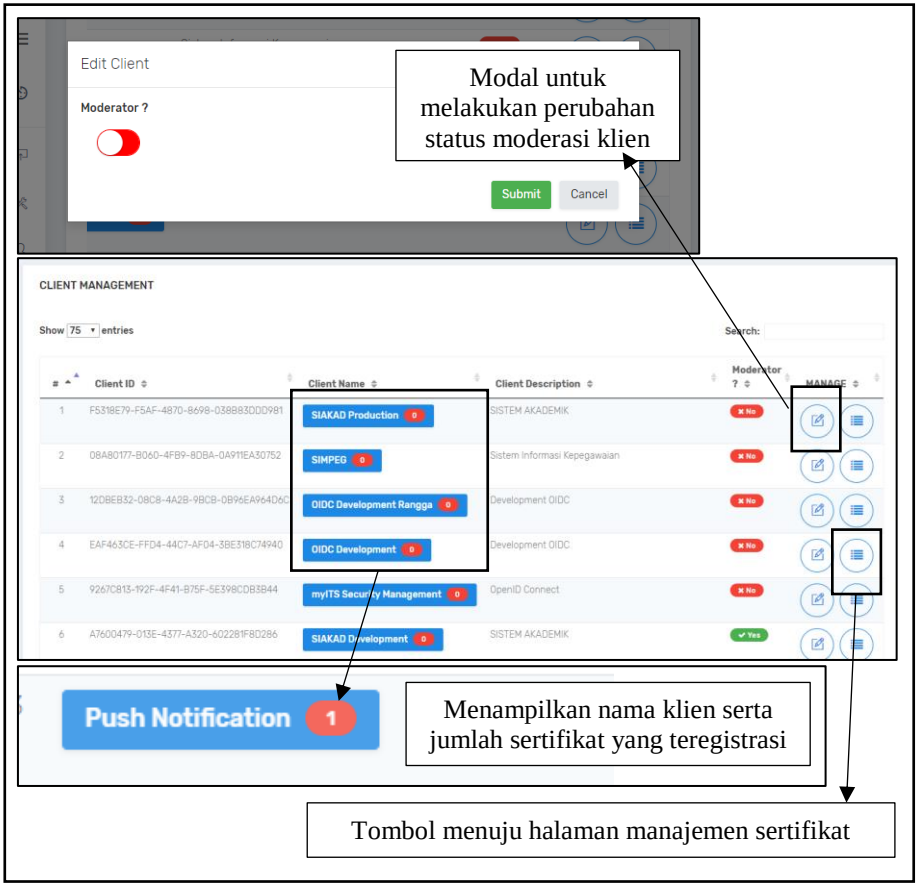
Implementasi halaman pengguna dapat dilihat pada Gambar 4.18.



Gambar 4.18 Antarmuka Halaman Pengguna

4.3.7 Antarmuka Halaman Manajemen Klien

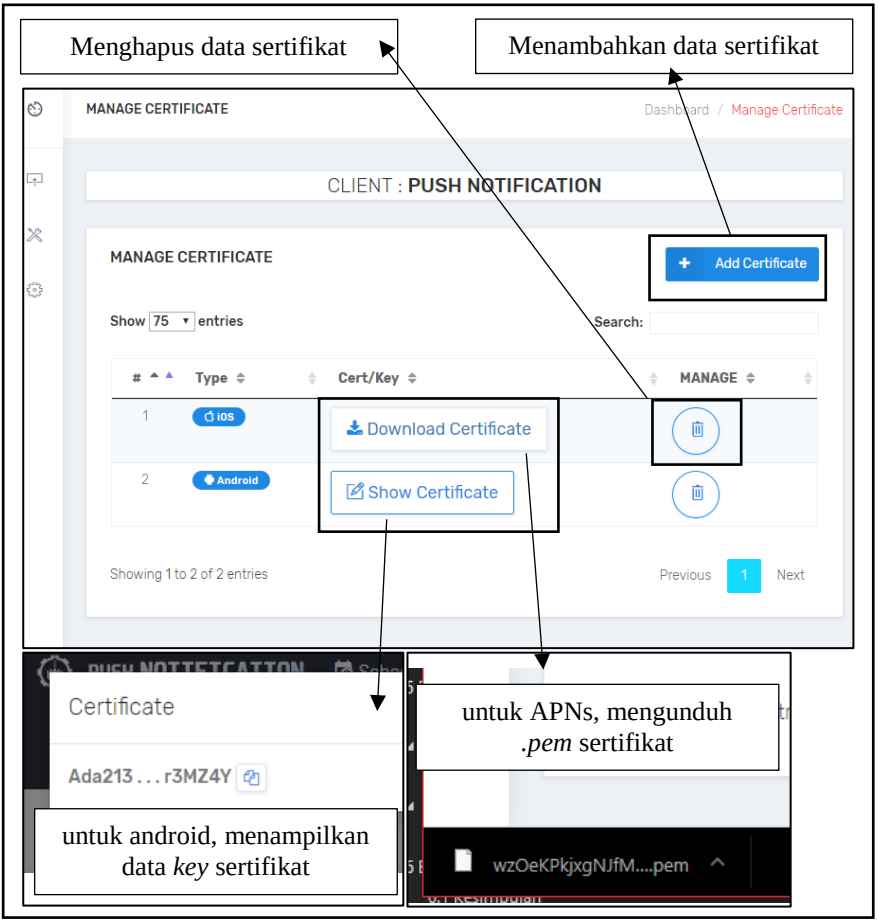
Implementasi halaman manajemen klien dapat dilihat pada Gambar 4.19.



Gambar 4.19 Antarmuka Halaman Manajemen Klien

4.3.8 Antarmuka Halaman Manajemen Sertifikat

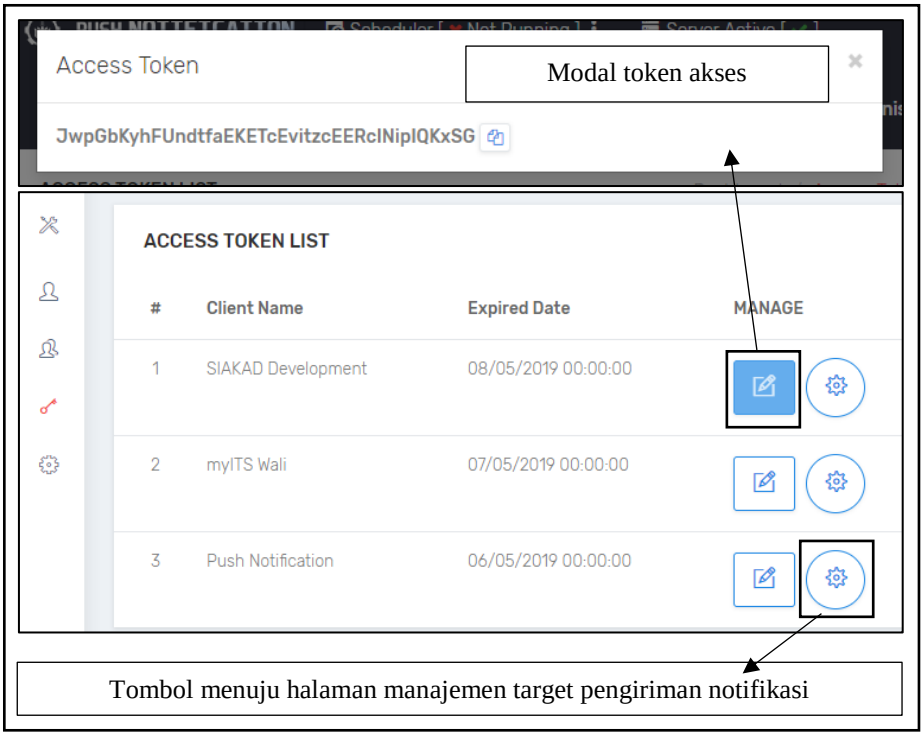
Implementasi halaman manajemen sertifikat dapat dilihat pada Gambar 4.20.



Gambar 4.20 Antarmuka Halaman Manajemen Sertifikat

4.3.9 Antarmuka Halaman Manajemen Token Akses

Implementasi halaman manajemen token akses dapat dilihat pada Gambar 4.21.

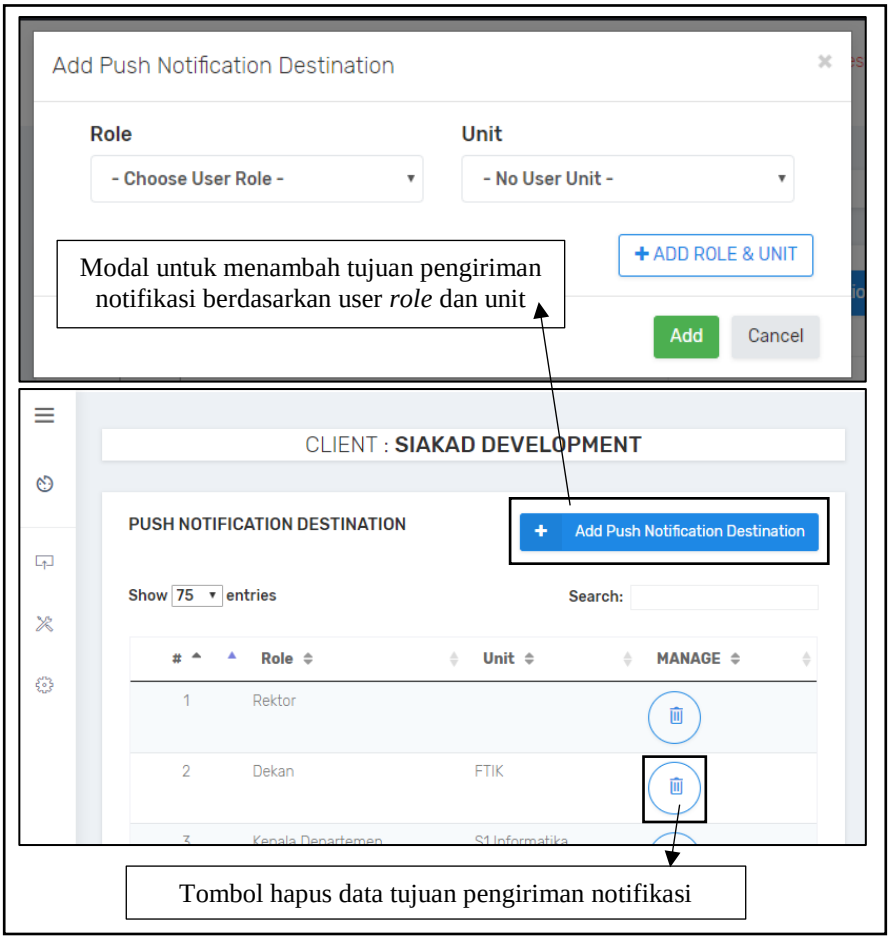


Gambar 4.21 Antarmuka Halaman Manajemen Token Akses

4.3.10 Antarmuka Halaman Manajemen Target Pengiriman Notifikasi

Implementasi halaman manajemen token akses role dan unit dapat dilihat pada

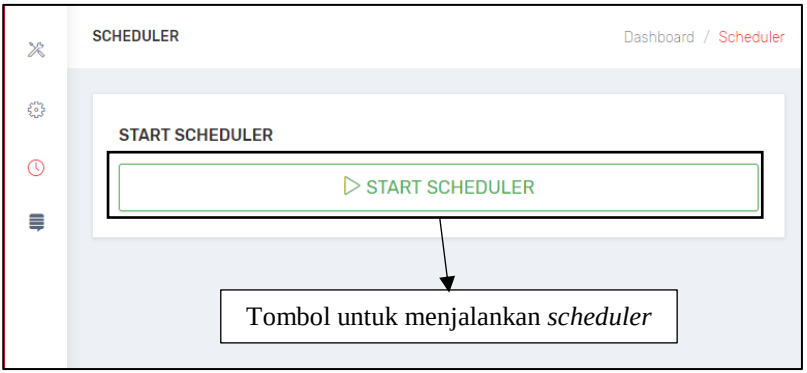
Gambar 4.22.



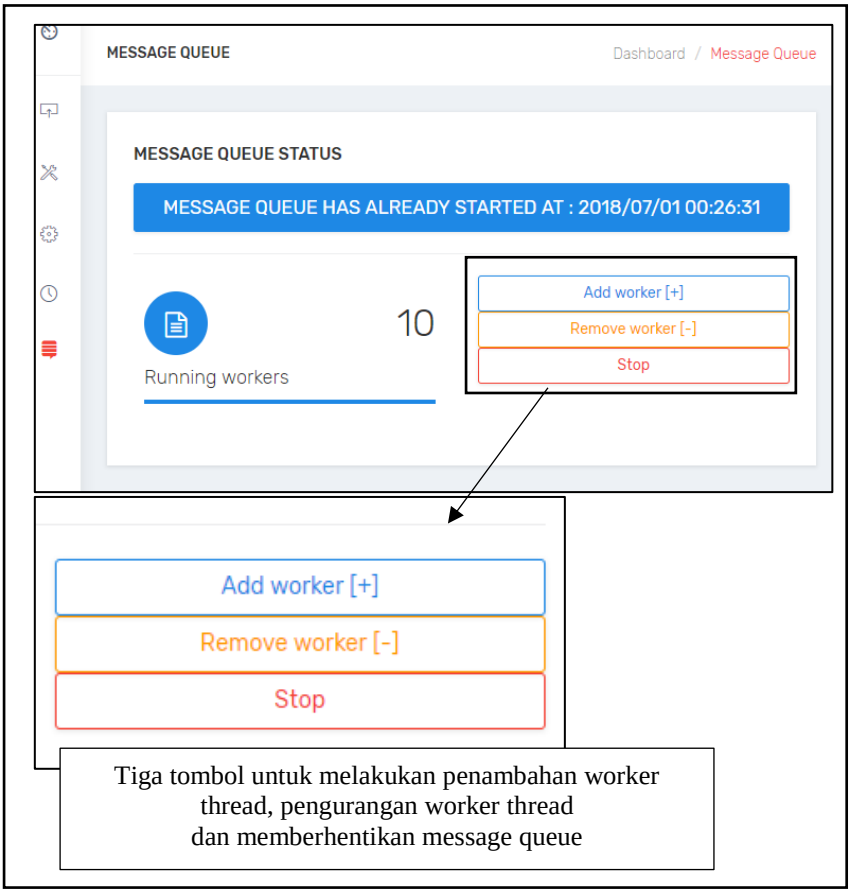
Gambar 4.22 Antarmuka Halaman Manajemen Target Pengiriman Notifikasi

4.3.11 Antarmuka Halaman Scheduler dan Message Queuing

Implementasi halaman *Scheduler* dan *Message Queuing* dapat dilihat pada Gambar 4.23 dan Gambar 4.24.



Gambar 4.23 Antarmuka Halaman *Scheduler*



Gambar 4.24 Antarmuka Halaman *Message Queuing*

4.4 Implementasi Proses dan Kasus Penggunaan

Implementasi proses dilakukan berdasarkan perancangan proses yang sudah dijelaskan pada bab analisis dan perancangan.

4.4.1 Implementasi Aplikasi Push Notification Terpusat

Sebelum menuju pada pembahasan implementasi proses dan kasus penggunaan untuk aplikasi *push notification* terpusat, akan dijelaskan struktur *class package* pada aplikasi *push notification* terpusat pada Tabel 4.2.

Tabel 4.2 Struktur *Class Package* Pada Aplikasi *Push Notification* Terpusat

No	Nama *Package*	Deskripsi
1	*resources*	• *Package* untuk *class* yang melakukan deklarasi serta mapping API *endpoint* • *Class* didalam *package* ini memanggil *class* yang berada dalam *package service* dan *package model*
2	*service*	• *Package* untuk *class* yang melakukan proses pengolahan data • *Class* didalam *package* ini memanggil *class* yang berada dalam *package model*, *package database* dan *package external*
3	*model*	• *Package* untuk *class* yang berfungsi sebagai data model
4	*database*	• *Package* untuk *class* yang berfungsi melakukan koneksi dengan basis data
5	*external*	• Package untuk *class* yang berfungsi sebagai *class* yang berhubungan dengan fitur Backend as a Service. • *Class* didalam *package* ini memanggil *class* yang berada dalam *package service* dan *package model*

Alasan utama dibuatnya klasifikasi class (*package*) dengan struktur seperti diatas adalah untuk memudahkan pengelolaan serta pengembangan aplikasi secara jangka panjang.

4.4.1.1 Implementasi Register Device Token

Subbab ini membahas mengenai implementasi register device token berdasarkan kasus penggunaan UC001. Dimulai dari diterimanya *request* dari *client*. Program akan melakukan pengecekan apakah parameter *request* lengkap atau tidak. Jika tidak lengkap akan melakukan response message *invalid request*. Namun jika parameter lengkap, maka proses berlanjut. Proses selanjutnya adalah mencari data *device token* pada database sesuai parameter *input request*. Jika data *device token* telah ditemukan pada database, berarti *device token* telah teregistrasi. Program mengirim *response message already registered*. Namun jika tidak ditemukan, maka program akan menambahkan *device token* kedalam *database*. Lalu mengirim response message *registered successfully*.

```
1   @Consumes(MediaType.APPLICATION_JSON)
2   @Produces(MediaType.APPLICATION_JSON)
3   public class RegisterDeviceResource {
4     RegisterDeviceService = new();
5
6     @Path("/device")
7     @POST
8     public checkUser() {
9       registerDeviceService.checkDevice()
10      return registerDeviceResponse;
11    }
.   }
.   .
.   .
    .
```

Kode Sumber 4.10 Deklarasi Endpoint Class Register Device Token

```
1   public class RegisterDevice {
2
```

```
3         private String username;
4         private String deviceToken;
5         private String deviceType;
6         private String clientID;
7   }
8
9   public RegisterDevice() {
10
11  }
.   .
.   .
.   .
```

Kode Sumber 4.11 Deklarasi Model Class Register Device Token

Kode Semu 4.1 Register Device Token

1	dataRegisterDeviceToken ← data register device token yang diterima dari aplikasi klien
2	accessToken ← token akses klien

3	If accessToken = true AND dataRegisterDeviceToken ← cek parameter
	  prosesData() ← menuju fungsi prosesData
	Else
	  Melakukan *return response message* "Tidak ada hak akses"

4	prosesData(dataRegisterDeviceToken)
	  Cek ada atau tidaknya data *device token* pada basis data
	  IF *device token* = true ← jika data *device token* ditemukan
	  return response message "perangkat telah teregistrasi"
	  ELSE
	  insertDeviceToken() ← memasukkan device token pada basis data
	  return response message "Berhasil melakukan registrasi"

4.4.1.2 Implementasi Segmentasi Pengiriman Notifikasi

Subbab ini membahas mengenani implementasi segmentasi pengiriman notifikasi berdasarkan kasus penggunaan UC002. Dimulai dari diterimanya *request* dari *client.* Program akan melakukan pengecekan apakah parameter *request* lengkap atau tidak. Jika tidak lengkap akan melakukan *response message invalid request*. Jika parameter lengkap maka proses berlanjut.

Pada selanjutnya akan dilakukan pengecekan tipe notifikasi yang akan dikirim yaitu *single*, *multiple*, atau *targeted*. Proses selanjutnya adalah mencari data *device token* pada basis data sesuai parameter input *request*. Setelah data *device token* pada basis data ditemukan, kemudian dilakukan pengecekan moderasi. Pengecekan status moderasi ditentukan dari *input* parameter *access token* dari *request* yang masuk. Jika status moderasi *access token* adalah *true*, maka sistem melakukan pencatatan pada basis data dengan flag status kirim adalah nol (0). Namun jika status moderasi *access token* adalah *false*, maka sistem menambah post notifikasi menuju *worker thread message queue* untuk selanjutnya dikirim menuju BaaS (*Backend as a Service*). Pengiriman menuju *BaaS Google FCM* dapat dilihat pada

```
1  @Consumes(MediaType.APPLICATION_JSON)
2  @Produces(MediaType.APPLICATION_JSON)
3
4  public class PushResource {
5
6     PushService = new();
7
8     @POST
9     @Path("/single")
10    public PushResourceResponse sendSingle(){
11        pushService.sendSingle();
12    }
13
14    @POST
15    @Path("/multiple")
16    public PushResourceResponse sendMultiple(){
17        pushService.sendMultiple();
18    }
19
20    @POST
21    @Path("/targeted")
22    public PushResourceResponse sendTargeted(){
23        pushService.sendTargeted();
24    }
```

```
25
26    @GET
27    @Path("/destination")
28    public List<> getDestination(){
29        pushService.getDestination();
30    }
.   .
.   .
.   .
```

Kode Sumber 4.12 Deklarasi Endpoint Segmentasi Pengiriman Notifikasi

```
1   public class PushMessage
2   {
3     private String access_token;
4     private String method;
5     private String content_type;
6     private String client_ip;
7     private String title;
8     private String message;
9     private String image;
10    private String username;
11    private ArrayList<String> id_pn_dest;
12    private ArrayList<String> user_id;
13    private ArrayList<String> integra_id;
14    private String type;
15    private String clientID;
16
17    public PushMessage() {
18
19    }
20  }
.   .
.   .
.   .
```

Kode Sumber 4.13 Deklarasi Model Class Segmentasi Pengiriman Notifikasi

Kode Semu 4.2 Segmentasi Pengiriman Notifikasi

1	dataKontenNotifikasi ← data konten notifikasi yang diterima dari aplikasi klien
2	accessToken ← token akses klien
3	IF accessToken = true AND dataRegisterDeviceToken ← cek parameter IF tipeNotifikasi = single prosesData.sendSingle() ← menuju fungsi sendSingle ELSE IF tipeNotifikasi = multiple prosesData.sendMultiple() ← menuju fungsi sendMultiple ELSE IF tipeNotifikasi = targeted prosesData.sendTargeted() ← menuju fungsi sendTargeted ELSE Melakukan *return response message* "Tidak ada hak akses"
4	prosesData.sendSingle() ← mengirim notifikasi dengan tipe single dataDeviceTokenSingle ← Data device token pada basis data processer(dataDeviceToken, dataKontenNotifikasi) ← Menuju fungsi untuk melakukan proses data
5	prosesData.sendMultiple() ← mengirim notifikasi dengan tipe multiple dataDeviceTokenMultiple ← Data device token pada basis data processer(dataDeviceToken, dataKontenNotifikasi) ← Menuju fungsi untuk melakukan proses data
6	prosesData.sendTargeted() ← mengirim notifikasi dengan tipe targeted dataDeviceTokenTargeted ← Data device token pada basis data processer(dataDeviceToken, dataKontenNotifikasi) ← Menuju fungsi untuk melakukan proses data
7	processer(dataDeviceToken, dataKontenNotifikasi) arrayList notifikasiAndroid ← Menyimpan data notifikasi untuk perangkat android (Google Firebase) arrayList notifikasiApple← Menyimpan data notifikasi untuk perangkat APNs while (dataDeviceToken) ← Loop data *device token* IF device_type = 'A' ← Jika tipe perangkat adalah android notifikasiAndroid.Add ← menambahkan data pada variabel notifikasiAndroid ELSE IF device_type = 'I' ← Jika tipe perangkat adalah apple notifikasiApple.Add ← menambahkan data pada variabel notifikasiApple isModerated = checkModerator() ← Melakukan cek status moderasi packetLogger(dataNotifikasi, isModerated) ← Melakukan pencatatan notifikasi dengan memsukkan data pada basis data

	packet logger IF isModerated = *false* ← Jika status moderasi adalah *false* maka notifikasi akan dikirim langsung dan dicatat dalam *packet logger* putToWorker(dataNotifikasi) ← Mengirim data notifikasi menuju *message queue worker thread* ELSE ← Jika status moderasi adalah *true* maka data notifikasi dimasukkan pada basis data *packet logger* dengan flag isModerated adalah *true*

```
1  public String pushFCMNotification(
2   ArrayList<String> userDeviceIdKey,
3   String title,
4   String message,
5   String image) throws Exception
6  {
7
8     String authKey = AUTH_KEY_FCM;
9     String FMCurl = API_URL_FCM;
10
11    URL url = new URL(FMCurl);
12    HttpURLConnection conn =
13     (HttpURLConnection)
14 url.openConnection();
15
16    conn.setUseCaches(false);
17    conn.setDoInput(true);
18    conn.setDoOutput(true);
19    conn.setRequestMethod("POST");
20    conn.setRequestProperty
21      ("Authorization","key="+authKey);
22    conn.setRequestProperty
23      ("Content-Type","application/json");
24
25    JSONObject json = new JSONObject();
26    JSONObject json_content =
27      new JSONObject();
28
29    JSONObject info = new JSONObject();
30    // Notification Title
```

```
31      info.put("title", title);
32      // Notification Body
33      info.put("message", message);
34      // Notification Image
35      if(image==null)
36        info.put("image", JSONObject.NULL );
37      else info.put("image", image);
38
39      json_content.put("notification", info);
40      json.put("data", json_content);
41      json.put("registration_ids",
42          userDeviceIdKey);
43
44      OutputStreamWriter wr =
45        new OutputStreamWriter(
46            conn.getOutputStream());
47      wr.write(json.toString());
48      wr.flush();
49
50      // read the response
51      DataInputStream input =
52        new DataInputStream(
53          conn.getInputStream());
54
55      int c;
56      StringBuilder resultBuf =
57        new StringBuilder();
58
59      while ( (c = input.read()) != -1) {
60        resultBuf.append((char) c);
61      }
62      input.close();
63
64      return resultBuf.toString();
65  }
```

Kode Sumber 4.14 Class untuk Mengirim Notifikasi Menuju Google FCM

4.4.1.3 Implementasi Mendapatkan Data Tujuan Pengiriman Notifikasi

Subbab ini membahas implementasi fitur untuk mendapatkan data tujuan pengiriman notifikasi berdasarkan *access_token* berdasarkan kasus penggunaan UC003. GET *request* dikirim oleh client dengan *headers Content-type* adalah *application/json*. URI endpoint GET adalah /send/destination?access_token=%s.

Kode Sumber 4.15 *Response Message* Mendapatkan Data Tujuan Pengiriman Notifikasi

```
[
    {
        "id_pn_dest": "064 . . . AD7",
        "r_name": "Rektor",
        "role_id": "8"
    },

    .
    .
    .

    {
        "id_pn_dest": "95D . . . E26",
        "r_name": "Mahasiswa",
        "role_id": "3",
        "u_name": "S2 Informatika",
        "unit_id": "4"
    }
]
```

4.4.1.4 Implementasi Mendapatkan Status Aktif Device Token

Subbab ini membahas implementasi fitur untuk mendapatkan status aktif *device token* berdasarkan kasus penggunaan UC004. GET request dikirim oleh client dengan headers Content-type adalah application/json. URI endpoint GET adalah /check/token/{deviceType}?access_token={accessToken}.

```
1   @Consumes(MediaType.APPLICATION_JSON)
2   @Produces(MediaType.APPLICATION_JSON)
3
4   public class CheckTokenResource {
5
6      V1_CheckTokenService = new();
7
8      @GET
9      @Path("/token/{deviceType}")
10     public List<> checkToken(){
11
12     }
13  }
.   .
.   .
.   .
```

Kode Sumber 4.16 Deklarasi *Endpoint Class* Mendapatkan Status Aktif Device Token

```
1   public class CheckToken {
2
3      private String status;
4      private String device_token_id;
5      private String device_token;
6      private String message;
7
8      public V1_CheckToken() {
9
10     }
.   .
.   .
.   .
```

Kode Sumber 4.17 Deklarasi Model Class Mendapatkan Status Aktif Device Token

4.4.1.5 Implementasi Scheduling

Subbab ini membahas mengenai implementasi *scheduling* berdasarkan kasus penggunaan UC005. Dimulai dari menyalakan *scheduler* yaitu dengan cara melakukan GET *request* menuju

endpoint "/scheduler/start". Jika scheduler berhasil dinyalakan maka akan mendapatkan response message ditunjukkan pada Kode Sumber 4.18.

Kode Sumber 4.18 *Response Message* Melakukan Perintah untuk Menjalankan *Scheduler*

```
{
    "scheduler_startTime": "2018/06/02 20:15:40",
    "scheduler_status": 1
}

// Scheduler status 1 = Aktif,
// Scheduler status 0 = Tidak Aktif
```

Ketika scheduler dinyalakan, dibuatlah sebuah thread yang berjalan untuk selalu melakukan *looping* per satu menit. Setiap satu menit, dilakukan pengecekan pada basis data apakah ada notifikasi dengan flag terkirim adalah *false* dan *sent_time* sama dengan *now_time* (waktu yang sekarang). Jika ditemukan, maka mengirimkan notifikasi ke *worker thread message queue*. Namun jika tidak ditemukan, akan dilakukan cek status aktif *scheduler*. Jika status aktif *scheduler* adalah true, maka kembali melakukan *loop* per menit. Jika status aktif *scheduler* adalah *false* maka thread akan dimatikan kemudian proses selesai.

```
 1  package its.pushnotification.service;
 2
 3  public class SchedulerService {
 4
 5    public static void main() {
 6       Calendar calendar = Calendar.getInstance();
 7
 8       // Instantiate Timer Object
 9       Timer time = new Timer();
10       time.schedule(new Scheduler(),
11       calendar.getTime(),
12       TimeUnit.SECONDS.toMillis(10));
13    }
14  }
 .  .
```

.	.
.	.

Kode Sumber 4.19 ***Service Class Scheduler***

```
1   public class Scheduler extends TimerTask{
2
3     public void run() {
4         try {
5              this.pn_scheduler();
6         }
7         catch (Exception ex) {
8             System.out.println(ex.getMessage());
9         }
10    }
11
11    class runthread implements Runnable
12    {
13       public String id_pn_packet=null;
14       String type;
15       JsonObject content;
16       public runthread(res, content,type)
17       {
18          this.id_pn_packet=res;
19          this.type=type;
20          this.content=content;
21       }
22
23       @Override
24       public void run()
25       {
26          if(type.equals("single"))
27          {
28             try {
29                this.executeSendSingle(id_pn_packet,
30                content);
313            } catch (Exception e) {
2                 e.printStackTrace();
33             }
34          }
35          else if(type.equals("multiple"))
36          {
37             try {
38                this.executeSendMultiple(id_pn_packet,
```

```
39                content);
40             } catch (Exception e) {
41                 e.printStackTrace();
42             }
43          }
444         else
5           {
46             try {
47                this.executeSendTargeted(id_pn_packet,
48                content);
49             } catch (Exception e) {
50                 e.printStackTrace();
51             }
52          }
53       }
54
55       public String executeSendSingle(id_pn_packet,
56       content) throws Exception
57       {
58          PushService pushService = new PushService();
59          Gson gson = new Gson();
60          PushMessage pushMessage =
61          gson.fromJson(content, PushMessage.class);
62
63          pushService.sendSingle(pushMessage,1);
64          this.updateDB(id_pn_packet);
656         return "ok";
6        }
676
8        public String executeSendMultiple(id_pn_packet,
69       content) throws Exception
70       {
71          PushService pushService = new PushService();
72
73          Gson gson = new Gson();
74          PushMessage pushMessage =
75          gson.fromJson(content, PushMessage.class);
76
77          pushService.sendMultiple(pushMessage,1);
78          this.updateDB(id_pn_packet);
79          return "ok";
80       }
81
82       public String executeSendTargeted(id_pn_packet,
```

```
83        content) throws Exception
84        {
85           PushService pushService = new PushService();
86
87           Gson gson = new Gson();
88           PushMessage pushMessage =
89           gson.fromJson(content, PushMessage.class);
90
91           pushService.sendTargeted(pushMessage,1);
92           this.updateDB(id_pn_packet);
93           return "ok";
94        }
95
96        public String updateDB(String id_pn_packet){
97           String result = null;
98           String sqlUpdate =
99           "UPDATE [sso].[dbo].[pn_packet] SET is_sent =
100          1,
102          sent_time = CURRENT_TIMESTAMP WHERE
103          id_pn_packet = ?";
104
105          PreparedStatement psInsert;
106          try {
107            psInsert
108             DBUtils.getPreparedStatement(sqlUpdate);
109            psInsert.setString(1, id_pn_packet);
110            psInsert.executeUpdate();
111          } catch (ClassNotFoundException | SQLException
112             ex) {
113
114                Logger.getLogger(Scheduler.class.getName(
115             )).
116             log(Level.SEVERE, null, ex);
117           }
118          return result;
119       }
120
121       public Map<String,String> split_content(content)
122       {
123          Map<String,String> result =
124          new HashMap<String,String>();
125          for (String keyString : content.keySet()) {
126            result.put(keyString,content.get(keyString)
127            .toString());
```

```
128         }
129         return result;
130      }
131   }
132
133   public void pn_scheduler() throws
134   ClassNotFoundException, SQLException
135   {
136      String timeStamp =
137      new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
138      .format(Calendar.getInstance().getTime());
139
140      String sql_getRequestList =
141       "SELECT p.id_pn_packet,p.content FROM
142       [sso].[dbo].[pn_packet] p WHERE DATEADD(MINUTE,
143       DATEDIFF(MINUTE, 0, \'"+timeStamp+"\'),
144       0)=DATEADD(MINUTE, DATEDIFF(MINUTE, 0,
145       p.sent_time), 0) AND p.is_sent = 0";
146      ResultSet rs = DBUtils.
147      getPreparedStatement(sql_getRequestList)
148      .executeQuery();
149
150      while (rs.next())
151      {
152         String res=rs.getString("id_pn_packet");
153         JsonObject content=(JsonObject)new JsonParser()
154         .parse(rs.getString("content"));
155
156         String type=((JsonObject)new JsonParser()
157      .parse(rs.getString("content")))
158      .get("type").getAsString();
159
160         Runnable r=new runthread(res,content,type);
161         Thread thread = new Thread(r);
162           thread.start();
163           thread.interrupt();
164      }
165   }
```

Kode Sumber 4.20 ***External Class Scheduler***

4.4.1.6 Implementasi Message Queue

Subbab ini membahas mengenai implementasi *message queue* berdasarkan kasus penggunaan UC006. Dimulai dari

menyalakan *message queue* yaitu dengan cara melakukan GET *request* menuju *endpoint* "/queue/start/%s". Program akan melakukan pengecekan apakah parameter *request* lengkap atau tidak. Jika tidak lengkap akan melakukan *response message invalid request*. Jika parameter lengkap maka proses berlanjut.

Proses selanjutnya adalah melakukan inisialisasi jumlah *maximum thread* yang dapat berjalan untuk kemudian sebagai *worker thread* mengirim POST *request* menuju Backend as a Service. Setelah melakukan inisialisasi thread sebanyak *maximum thread*, dilakukan sebuah *looping* untuk melakukan pengecekan pada variabel *list_queue* yang berisi data notifikasi untuk dikirim. Variabel ini merupakan variabel dengan tipe data *list* pada java. Jika data notifikasi ditemukan pada *list_queue* maka akan dilakukan cek ketersediaan *worker thread* dengan membaca *thread_status*. Jika *thread_status* adalah 0 maka *thread* tersebut sedang dalam kondisi *idle*. Jika ditemukan *thread* dalam kondisi *idle*, maka data notifikasi akan diproses untuk dikirim melalui thread tersebut.

Setelah data notifikasi masuk kedalam *thread*, proses dalam *thread* adalah mengubah *thread_status* menjadi 1 yang berarti *thread* dalam kondisi *busy*. Lalu mengirim notifikasi menuju *Backend as a Service*. Menerima *return message* hasil pengiriman notifikasi, lalu melakukan *update* data hasil pengiriman pada basis data. Dan yang terakhir adalah mengembalikan status *thread* menjadi 0 (*idle*).

Message queue ini menerapkan adalah mekanisme FIFO (*First In First Out*) pada stack *list_queue* untuk mekanisme pengambilan dan pengiriman data.

Kode Sumber 4.21 *Response Message* Menjalankan *Message Queue*

```
// URI : /queue/start

{
    "error": "false",
    "message": "Message Queue Started",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 1,
```

```
    "thread_number": 5
}

// Status 1 = Aktif, Status 0 = Tidak Aktif
```

Kode Sumber 4.22 ***Response Message*** **Menambahkan** ***Worker Thread Message Queue***

```
// URI : /queue/add_thread

{
    "error": "false",
    "message": "Thread no. 6 started",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 1
}
// Status 1 = Aktif, Status 0 = Tidak Aktif
```

Kode Sumber 4.23 ***Response Message*** **Mengurangi** ***Worker Thread Message Queue***

```
// URI : /queue/remove_thread

{
    "error": "false",
    "message": "Thread no. 6 stopped",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 1
}
// Status 1 = Aktif, Status 0 = Tidak Aktif
```

Kode Sumber 4.24 ***Response Message*** **Memberhentikan** ***Message Queue***

```
// URI : /queue/stop

{
    "error": "false",
    "message": "All thread stopped.",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 0
}

// Status 1 = Aktif, Status 0 = Tidak Aktif
```

Kode Sumber 4.25 *Response Message* Menampilkan Status *Message Queue*

```
// URI : /queue/show_status

{
    "error": "false",
    "message": "You have 5 worker thread",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 1,
    "thread_number": 5
}
// Status 1 = Aktif, Status 0 = Tidak Aktif
```

```
1  @Consumes(MediaType.APPLICATION_JSON)
2  @Produces(MediaType.APPLICATION_JSON)
3
4  public class QueueResource {
5
6     @GET
7     @Path("/start/{threadNumber}")
8     public Queue startEngine() {}
9
10    @GET
11    @Path("/stop")
12    public Queue stopEngine() {}
13
14    @GET
15    @Path("/show_status")
16    public Queue show_threads() {}
17
18    @GET
19    @Path("/add_thread")
20    public Queue add_thread() {}
21
22    @GET
23    @Path("/remove_thread")
24    public Queue remove_thread() {}
25
26 }
```

Kode Sumber 4.26 Deklarasi *Endpoint Class Message Queue*

```
1  public class Queue {
2
```

```
3     private int msg_queue_status;
4     private int thread_number;
5     private String msg_queue_startTime;
6     private String error;
7     private String message;
8
9     public Queue() {
10
11    }
.   .
.   .
.   .
```

Kode Sumber 4.27 ***Deklarasi Model Class Message Queue***

```
1   package its.pushnotification.service;
2
3   import its.pushnotification.external.Fcm;
4
5   import java.util.ArrayList;
6   import java.util.HashMap;
7   import java.util.LinkedList;
8   import java.util.Map;
9
10  public class V1_QueueService {
11
12    public static int thread_max=5;
13    public static int thread_count=0;
14    public static Thread[] thread_pool =
15      new Thread[1000];
16    public static int[] thread_status =
17      new int[1000];
18    public static int[] thread_active =
19      new int[1000];
20    public static LinkedList<Map<String,Object>>
21      queue_list =
22      new LinkedList<Map<String,Object>>();
23    public String put_work(ArrayList<String>
24      userDeviceIdKey, String title, String
25      message, String image)
```

```
26      {
27        Map<String,Object> work_block =
28          new HashMap<String, Object>();
29
30        work_block.put("userDeviceIdKey",
31          userDeviceIdKey);
32        work_block.put("title", title);
33        work_block.put("message", message);
34        work_block.put("image", image);
35        this.enqueue(work_block);
36      }
37
38      public void threadpool_start(int threadNumber)
39      {
40        for (int i=0;i<threadNumber;i++)
41        {
42          Runnable r = new runthread(i);
43          System.out.println(i);
44          thread_pool[i] = new Thread(r);
45          thread_pool[i].start();
46          thread_active[i]=1;
47          thread_status[i]=0;
48          thread_count++;
49        }
50      }
51
52      public void clear_threadpool()
53      {
54        for (int i=0;i<thread_count; i++)
55        {
56          thread_active[i]=0;
57        }
58        thread_count=0;
59      }
60
61      public int add_thread()
62      {
63        Runnable r = new runthread(thread_count);
64        thread_pool[thread_count] = new Thread(r);
65        thread_pool[thread_count].start();
```

```
66        thread_status[thread_count]=0;
67        thread_count++;
68        return thread_count;
69      }
70
71      public int remove_thread()
72      {
73        thread_active[thread_count-1]=0;
74        thread_count--;
75        return thread_count+1;
76      }
77
78
79      public void enqueue(
80        Map<String,Object> inputObject)
81      {
82        if(queue_list.size()==0)
83        {
84          queue_list.addFirst(inputObject);
85        }
86        else
87        {
88          queue_list.addLast(inputObject);
89        }
90      }
91
92      public static Map<String,Object>dequeue()
93      {
94        Map <String,Object> result =
95          new HashMap<String,Object>();
96        if(queue_list.size()>0)
97        {
98          result=queue_list.pollFirst();
99        }
100       return result;
101     }
102
103     class runthread implements Runnable
104     {
105       Map<String,Object> workData;
```

```
106       int thread_number;
107       public runthread(int number)
108       {
109         this.thread_number=number;
110       }
111
112       @Override
113       public void run()
114       {
115         while (thread_active[thread_number]==1)
116         {
117           workData=dequeue();
118           if(workData!=null)
119           if(workData.size()>0)
120           {
121           thread_status[thread_number]=1;
122             Fcm fcmNotif = new Fcm();
123             @SuppressWarnings("unchecked")
124             ArrayList<String> userDeviceIdKey =
125               ArrayList<String>)workData
126               .get("userDeviceIdKey");
127             String title = (String)workData
128               .get("title");
129             String message = (String)workData
130               .get("message");
131             String image = (String)workData
132               .get("image");
133             try{
134               String result = fcmNotif.
135                 pushFCMNotification(
136                    userDeviceIdKey, title,
137                    message, image);
138             }
139             catch (Exception e){
140               System.out.println (e);
141             }
142             thread_status[thread_number]=0;
143           }
144           try {
145             Thread.sleep(1000);
```

```
146             } catch (InterruptedException e) {
147                e.printStackTrace();
148             }
149          }
150       }
151    }
152 }
```

Kode Sumber 4.28 ***Service Class Message Queue***

4.4.2 Implementasi Aplikasi Pengelola Konten Notifikasi

Aplikasi ini digunakan oleh admin untuk mengelola aplikasi *push notification* terpusat. Aplikasi ini diimplementasikan menggunakan *framework* CodeIgniter (CI) dengan struktur MVC (Model View Controller).

Agar aplikasi pengelola konten notifikasi dapat berinteraksi dengan aplikasi *push notification* terpusat, maka dibuatlah sebuah library bernama *My_its_pushnotification_api*. Potongan kode sumber *library* ditunjukkan pada Kode Sumber 4.1

```
1  <?php
2  if ( ! defined('BASEPATH')) exit('No direct script
3  access allowed');
4
5  class Myits_push_notification_api {
6
7    /* Variable to hold an instance of CodeIgniter
8       so we can access CodeIgniter features */
9    protected $CI;
10
12   /* Create an array of the urls to be used in
13         api calls
14      The urls contain conversion specifications
15         that will be replaced by sprintf in the
16         functions @var string
17    */
18   protected $_api_urls = array(
19     'get_destination'  => '[ SERVER URL ]
20         /send/destination?access_token=%s',
```

```
21      'send_single'      => '[ SERVER URL ]
22         /send/single?access_token=%s',
23      'send_multiple'    => '[ SERVER URL ]
24         /send/multiple?access_token=%s',
25      'send_targeted'    => '[ SERVER URL ]
26         /send/targeted?access_token=%s',
27      'check_token'      => '[ SERVER URL ]
28         /check/token/%s?access_token=%s',
29      'start_scheduler'  => '[ SERVER URL ]
30         /scheduler/start?access_token=%s',
31      'scheduler_status' => '[ SERVER URL ]
32         /scheduler/status?access_token=%s',
33      'start_queue'      => '[ SERVER URL ]
34         /queue/start/%s?access_token=%s',
35      'add_thread'       => '[ SERVER URL ]
36         /queue/add_thread?access_token=%s',
37      'remove_thread'    => '[ SERVER URL ]
38         /queue/remove_thread?access_token=%s',
39      'stop_queue'       => '[ SERVER URL ]
40         /queue/stop?access_token=%s',
41      'queue_status'     => '[ SERVER URL ]
42         /queue/show_status?access_token=%s',
43    );
44    /*
45     * Construct function
46     * Sets the codeigniter instance variable and
47     * loads the lang file
48     */
49    public function __construct() {
50
51      // Set the CodeIgniter instance variable
52      $this->CI =& get_instance();
53
54      // Load the Instagram API language file
55      $this->CI->load->
56        Config('myits_push_notification_api');
57    }
58
59    /*
60     * The api call function is used by all other
61     * functions
62     * It accepts a parameter of the url to use
63     * And an optional string of post parameters
64     * @param string api url
```

```
65      * @param array post parameters for curl call
66      * @return std_class data returned form curl
67      * call
68      */
69     public function __apiCal
70       ($url, $post_parameters) {
71
72       // Initialize the cURL session
73       $curl_session = curl_init();
74
75       // Set the URL of api call
76       curl_setopt($curl_session,
77          CURLOPT_URL, $url);
78
79       //set the content type to application/json
80       curl_setopt($curl_session,CURLOPT_HTTPHEADER,
81       array('Content-Type:application/json'));
82
83       // If there are post fields add them to the
84  call
85       if($post_parameters !== FALSE) {
86
87     curl_setopt($curl_session,CURLOPT_POSTFIELDS,
88             $post_parameters);
89       }
90
91       // Return the curl results to a variable
92       curl_setopt($curl_session,
93          CURLOPT_RETURNTRANSFER, 1);
94
95       // There was issues with some servers not being
96          able to retrieve the data through https
97       // The config variable is set to TRUE by
98          default. If you have this problem set the
99          config variable to FALSE
100
101      curl_setopt($curl_session,
102         CURLOPT_SSL_VERIFYPEER,$this->CI->config->
103         item('server_ssl_verify'));
104
105      // Execute the cURL session
106      $contents = curl_exec($curl_session);
107
108      // Close cURL session
```

```
109       curl_close ($curl_session);
110
111       // Return the response
112       return json_decode($contents);
113
114     }
115
116     /*
117      * Get device token status
118      * Accepts token type
119      *  (A for Android device, I for iOS device)
120      * @param Char device_type
121      * @return std_class data token ID and token
122      * status (200 for Active token,
123      * 404 for Inactive token)
124      */
125
126     public function get_token_status($device_type) {
127
128       $token_status_request_url =
129          sprintf($this->_api_urls['check_token'],
130          $device_type, $this->CI->config->
131          item('access_token'));
132
133       return $this->__apiCall
134          ($token_status_request_url, $data = FALSE);
135
136     }
137
138     /*
139      * Get push notification send destination
140      * Accepts access token
141      * @param string access_token
142      * @return std_class data returned form curl
143      * call
144      */
145     public function get_destination() {
146       $get_role_url =
147          sprintf($this>_api_urls['get_destination'],
148          $this->CI->config->item('access_token'));
149       return $this->__apiCall($get_role_url, FALSE);
150     }
151
152   .
```

```
153 .
154 .
155 // Fungsi-fungsi lain sesuai API Endpoint
156 .
157 .
158
159 }
```

Kode Sumber 4.29 *Library* untuk CodeIgniter

4.4.2.1 Implementasi Melihat Data Pengguna Serta Device Token yang Teregistrasi

Implementasi melihat data pengguna beserta *device token* yang teregistrasi berdasarkan kasus penggunaan UC007.

Kode Semu 4.3 Melihat Data Pengguna Serta *Device Token* yang Teregistrasi

1	listUserToken ← Data pengguna serta device token yang teregistrasi diambil dari basis data
2	Menampilkan halaman *Dashboard*

4.4.2.2 Implementasi Menampilkan Daftar Peran Pengguna (User Role)

Implementasi menampilkan daftar peran pengguna berdasarkan kasus penggunaan UC008.

Kode Semu 4.4 Menampilkan Daftar Peran Pengguna

1	listUserRole ← Data peran pengguna diambil dari basis data
2	Menampilkan halaman *dashboard*

4.4.2.3 Implementasi Menampilkan Daftar Unit Pengguna

Implementasi menampilkan daftar unit pengguna berdasarkan kasus penggunaan UC009.

Kode Semu 4.5 Menampilkan Daftar Unit Pengguna

1	listUserUnit ← Data unit pengguna diambil dari basis data
2	Menampilkan halaman *dashboard*

4.4.2.4 Implementasi Melihat dan Mengubah Data Client

Implementasi menampilkan daftar unit pengguna berdasarkan kasus penggunaan UC010.

Kode Semu 4.6 Melihat Data Client

1	listClient ← Data klien diambil dari basis data
2	Menampilkan halaman daftar klien

Kode Semu 4.7 Mengubah Data Client

1	isModerated ← Mengambil data status moderasi klien dari form
2	dataInsert ← Array yang berisi atribut tabel yang akan ditambah beserta nilai dari atribut tersebut yang didapatkan dari view
3	Mengubah data klien ← Mengubah status moderasi klien
4	*Redirect* ke halaman daftar klien

4.4.2.5 Implementasi Pengiriman Notifikasi dengan Antarmuka Pengguna

Implementasi pengiriman notifikasi dengan antarmuka pengguna berdasarkan kasus penggunaan UC011.

Kode Semu 4.8 Menampilkan Halaman Notifikasi dengan Antarmuka Pengguna

1	clientList ← Data klien diambil dari basis data
2	integraIdList ← Data Integra ID diambil dari basis data
3	userIdList ← Data User ID diambil dari basis data
4	destinationList ← Data tujuan pengiriman notifikasi diambil dari *My ITS Push Notification API*
5	Menampilkan halaman kirim notifikasi

Kode Semu 4.9 Melakukan Pengiriman Notifikasi dengan Antarmuka Pengguna

1	postData ← Data _POST yang didapatkan usai dilakukan submit form kirim notifikasi pada halaman kirim notifikasi
2	extract postData ← Melakukan ekstraksi _POST data
3	IF postTipeNotifikasi = single

	Array dataNotifikasi ← membuat array untuk pengiriman push notifikasi sesuai dengan parameter send_single pada *endpoint rest api* aplikasi *push notification* terpusat result = postSendSingle(dataNotifikasi) ← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* ELSE IF postTipeNotifikasi = multiple_user Array dataNotifikasi ← membuat array untuk pengiriman push notifikasi sesuai dengan parameter send_multiple pada *endpoint rest api* aplikasi *push notification* terpusat result = postSendMultiple(dataNotifikasi) ← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* ELSE IF postTipeNotifikasi = targeted Array dataNotifikasi ← membuat array untuk pengiriman push notifikasi sesuai dengan parameter send_targeted pada *endpoint rest api* aplikasi *push notification* terpusat result = postSendTargeted(dataNotifikasi) ← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api*

4.4.2.6 Implementasi Melihat Data Riwayat Notifikasi

Implementasi melihat data riwayat notifikasi berdasarkan kasus penggunaan UC012.

Kode Semu 4.10 Melihat Data Riwayat Notifikasi

1	listPacket ← Data riwayat notifikasi / *packet* diambil dari basis data
2	Menampilkan halaman *packet monitoring*

4.4.2.7 Implementasi Melihat Status Aktif Device Token

Implementasi melihat status aktif *device token* berdasarkan kasus penggunaan UC013.

Kode Semu 4.11 Melihat Status Aktif Device Token

1	tokenData ← Data status aktif *device token* diambil dari *My ITS Push Notification API*
2	Menampilkan halaman cek token

4.4.2.8 Implementasi Melakukan Moderasi Konten Notifikasi

Implementasi melakukan moderasi konten notifikasi berdasarkan kasus penggunaan UC014.

Kode Semu 4.12 Melihat Data Moderasi Notifikasi

1	listModeration ← Data notifikasi yang harus dilakukan moderasi, diambil dari basis data
2	Menampilkan halaman moderator

Kode Semu 4.13 Melakukan Moderasi Notifikasi

1	id_packet ← Mengambil data id packet notifikasi dari form
2	date ← Mengambil data tanggal pengiriman dari form
3	time ← Mengambil data waktu pengiriman dari form
4	dataUpdate ← Melakukan perubahan data pada basis data
5	*Redirect* ke halaman moderator

4.4.2.9 Implementasi Melihat Daftar Pengguna

Implementasi melihat daftar pengguna berdasarkan kasus penggunaan UC015.

Kode Semu 4.14 Melihat Daftar Pengguna

1	listUserAccount ← Data pengguna diambil dari basis data
2	Menampilkan halaman daftar pengguna

4.4.2.10 Implementasi Melihat Data Access Token

Implementasi melihat data *access token* berdasarkan kasus penggunaan UC016.

Kode Semu 4.15 Melihat Daftar Pengguna

1	listAccessToken ← Data *access token* diambil dari basis data
2	Menampilkan halaman access token

4.4.2.11 Implementasi Mengelola Data Tujuan Pengiriman Notifikasi Client

Implementasi mengelola data tujuan pengiriman notifikasi bagi client berdasarkan kasus penggunaan UC017.

Kode Semu 4.16 Melihat Data Tujuan Pengiriman Notifikasi Klien

1	listDestination ← Data tujuan pengiriman notifikasi diambil dari basis data
2	Menampilkan halaman tujuan pengiriman notifikasi

Kode Semu 4.17 Mengubah Data Tujuan Pengiriman Notifikasi Klien

1	id_role ← Mengambil data ID peran pengguna dari form
2	id_unit ← Mengambil data ID unit pengguna dari form
2	dataInsert ← Array yang berisi atribut tabel yang akan ditambah beserta nilai dari atribut tersebut yang didapatkan dari view
3	Menambahkan data tujuan pengiriman notifikasi pada basis data
4	*Redirect* ke halaman daftar tujuan pengiriman notifikasi

Kode Semu 4.18 Menghapus Data Tujuan Pengiriman Notifikasi Klien

1	idPnDestination ← Mengambil data ID tujuan pengiriman notifikasi dari form
2	Menghapus data tujuan pengiriman notifikasi dari basis data
4	*Redirect* ke halaman daftar tujuan pengiriman notifikasi

4.4.2.12 Implementasi Menjalankan Scheduler

Implementasi menjalankan *scheduler* berdasarkan kasus penggunaan UC018.

Kode Semu 4.19 Menjalankan *Scheduler*

1	Menerima perintah untuk menjalankan *scheduler*
2	startScheduler()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api*
3	*Redirect* ke halaman *scheduler*

4.4.2.13 Implementasi Konfigurasi Message Queue

Implementasi konfigurasi *message queue* berdasarkan kasus penggunaan UC019.

Kode Semu 4.20 Melihat *Status Message Queue*

1	queueStatus()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* untuk mendapatkan status *message queue*
2	Menampilkan halaman *message queue*

Kode Semu 4.21 Menjalankan Message Queue

1	Menerima perintah untuk menjalankan *message queue*
2	startMessageQueue()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* untuk menjalankan *message queue*
3	*Redirect* ke halaman *message queue*

Kode Semu 4.22 Menambah Jumlah *Worker Thread Message Queue*

1	Menerima perintah untuk menambah jumlah *worker thread message queue*
2	addThread()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* untuk menambah *worker thread message queue*
3	*Redirect* ke halaman *message queue*

Kode Semu 4.23 Mengurangi Jumlah *Worker Thread Message Queue*

1	Menerima perintah untuk mengurangi jumlah *worker thread message queue*
2	removeThread()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* untuk mengurangi *worker thread message queue*
3	*Redirect* ke halaman *message queue*

Kode Semu 4.24 Mematikan *Message Queue*

1	Menerima perintah untuk mematikan *message queue*
2	stopQueue()← melakukan post request menuju aplikasi *push notification* terpusat melalui *library My_its_pushnotification_api* untuk mematikan *message queue*
3	*Redirect* ke halaman *message queue*

4.4.2.14 Implementasi Mengelola Sertifikat Klien

Implementasi mengelola sertifikat klien berdasarkan kasus penggunaan UC020.

Kode Semu 4.25 Melihat Data Sertifikat Klien

1	listCertificate ← Data sertifikat klien diambil dari basis data
2	Menampilkan halaman kelola sertifikat klien

Kode Semu 4.26 Menambah Data Sertifikat Klien

1	IF tipe Backend as a Service = FCM Menambahkan key pada basis data ELSE IF tipe Backend as a Service = APNs Menguplod file .pem pada direktori yang telah ditentukan Menambahkan filename pada basis data
2	*Redirect* ke halaman kelola sertifikat klien

Kode Semu 4.27 Menghapus Data Sertifikat Klien

1	cert_ID ← Mengambil data ID sertifikat klien dari form
2	Menghapus data sertifikat klien dari basis data
3	IF tipe Backend as a Service = APNs Menghapus file *.pem* pada direktori unggah sesuai dengan filename yang disimpan pada basis data
4	*Redirect* ke halaman kelola sertifikat klien

4.4.3 Implementasi *API Endpoint*

Implementasi API Endpoint untuk Aplikasi *Push Notification* Terpusat dijelaskan pada tabel dibawah ini.

Tabel 4.3 Implementasi API Endpoint

No	Endpoint URI	Method	Deskripsi
Mengirim Notifikasi			
1	/send/destination?access_token = {access_token}	GET	Mendapatkan daftar tujuan pengiriman notifikasi
	Get parameter:		

No	Endpoint URI	Method	Deskripsi
	• *Access Token*		
2	/send/single	POST	Mengirim data notifikasi dengan tipe *single*
	Post parameter: • *Access Token* • JSON: *clientID, username, title, message, image*		
3	/send/multiple	POST	Mengirim data notifikasi dengan tipe *multiple*
	Post parameter: • *Access Token* • JSON: *clientID, userID, title, message, image* **atau** • JSON: *clientID, integraID, title, message, image*		
4	/send/targeted	POST	Mengirim data notifikasi dengan tipe *targeted*
	Post parameter: • *Access Token* • JSON: *clientID, roleID, unitID, title, message, image*		
Cek *device token* status			
5	/check/token/{device_type}	POST	Melakukan perintah untuk melakukan cek status *device token*
	Post parameter: • *Access Token*		
Scheduler			
6	/scheduler/start	POST	Menyalakan

No	Endpoint URI	Method	Deskripsi
			scheduler
7	/scheduler/status?access_token = {access_token}	GET	Melihat status *scheduler*
	Post parameter: • *Access Token*		
Message Queuing			
8	/queue/start/{thread_number}	POST	Menyalakan *message queue*
	Post parameter: • *Access Token* • Jumlah *thread*		
9	/queue/add_thread	POST	Menambah jumlah *worker thread* untuk mengirim notifikasi
	Post parameter: • Access Token		
10	/queue/remove_thread?access_token=%s	POST	Mengurangi jumlah *worker thread* untuk mengirim notifikasi
	Post parameter: • Access Token		
11	/queue/stop	POST	Memberhentikan *message queue*
	Post parameter: • Access Token		
12	/queue/show_status?access_token= {access_token}	GET	Melihat status *message queue*.
	Get parameter:		

No	Endpoint URI	Method	Deskripsi
	• Access Token		
Register Device Token			
13	/register/device	POST	Melakukan registrasi *device token* aplikasi klien
	Post parameter: • JSON : *username, deviceToken, deviceType, clientID*		

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas uji coba dan evaluasi terhadap perangkat lunak yang telah dikembangkan dari implementasi aplikasi pendeteksi perintah suara berbahasa Indonesia untuk mengoperasikan komputer berbasis android.

5.1 Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas yang ditunjukkan pada Tabel 5.1

Tabel 5.1 Lingkungan Pengujian

Jenis	Spesifikasi	
Server (1 buah)	Prosesor	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
	Memori	512 MB
	Sistem Operasi	Debian 9.4
	Perangkat Pengembang	Eclipse Kepler
		Android Studio
	Perangkat Pembantu	Microsoft Office Word 365
Smartphone	Android (1 perangkat)	Dengan versi minimal SDK 19
	Apple (iPhone) (1 perangkat)	Dengan versi minimal iOS 11

5.2 Uji Coba Fungsionalitas

Uji coba fungsionalitas ini adalah pengujian fungsi-fungsi yang berjalan pada aplikasi berdasarkan kasus penggunaan.

5.2.1 Uji Coba Fungsionalitas Aplikasi Push Notification Terpusat

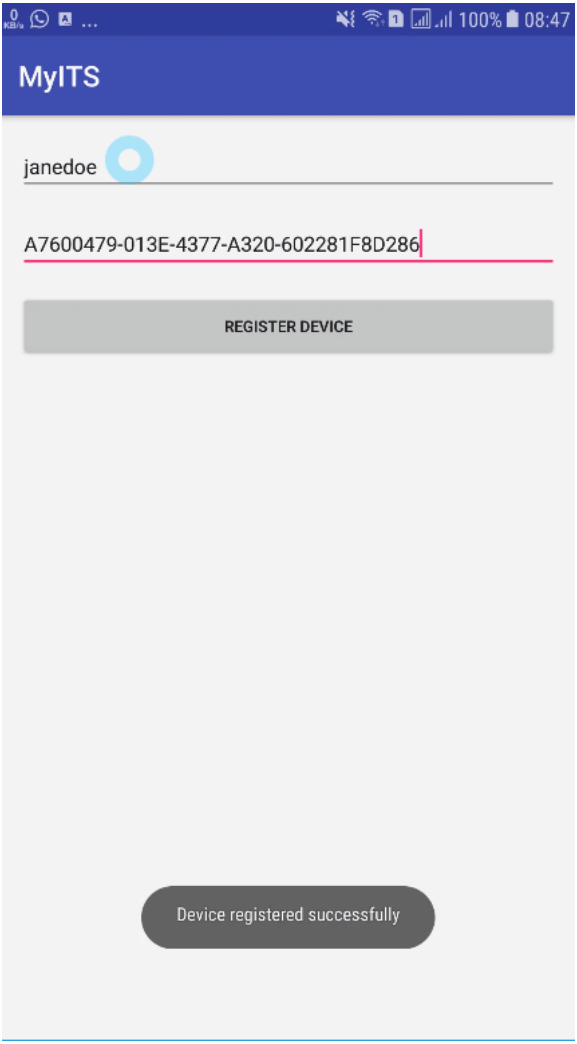
5.2.1.1 Pengujian Melakukan Register Device Token

Pengujian *register device token* merupakan pengujian terhadap kemampuan program untuk menerima *request* dari client lalu melakukan penambahan *device token* pada database yang dikelompokkan sesuai dengan *clientID* dan *username*. Pengujian ini dilakukan menggunakan sebuah aplikasi android yang mengirim post request kepada *endpoint API*. Ketika berhasil maka akan muncul *toast* pada aplikas android yang berisi *response message*. Rincian skenario pada kasus penggunaan ini bisa dilihat pada tabel Tabel 5.2.

Tabel 5.2 Pengujian Melakukan Register Device Token

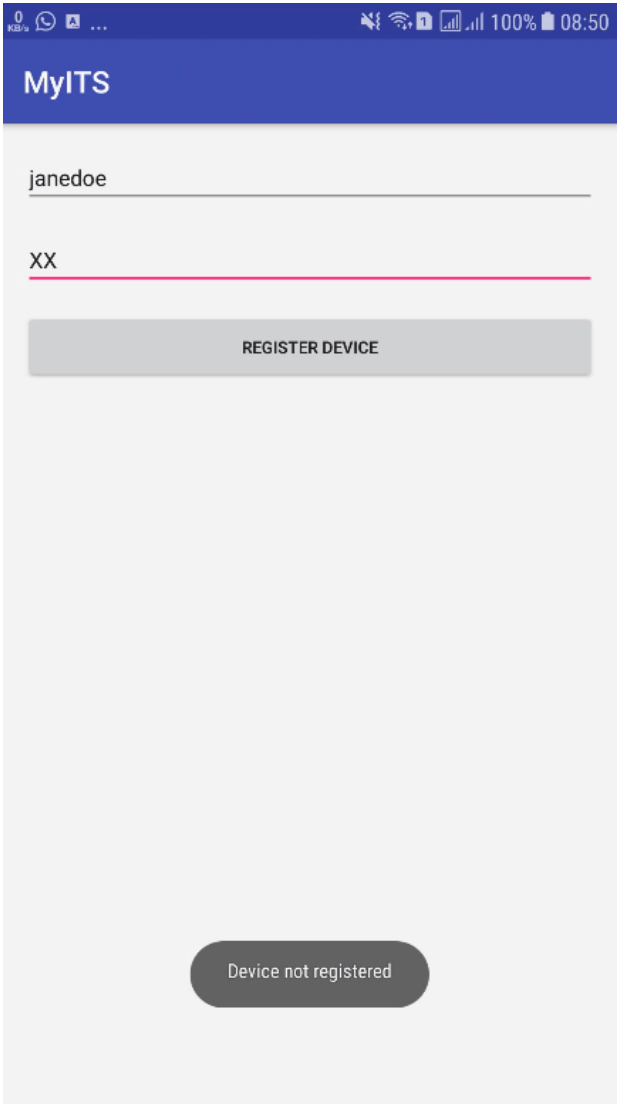
No. Pengujian	**UJ001**
Referensi Kasus Penggunaan	UC001
Nama	Pengujian melakukan *register device token*
Tujuan Pengujian	Memasukkan data *device token* kedalam basis data yang dikelompokkan berdasarkan clientID dan username.
Kondisi Awal	Pengguna membuka interface aplikasi *dummy* android
Data Uji	• Data username • Data clientID
Langkah Pengujian	1. Pengguna memasukkan data username dan clientID pada field yang ada pada aplikasi android 2. Pengguna menekan tombol *submit*
Hasil yang Diharapkan	• Program memasukkan data *device*

	token pada *database* • Aplikasi android memunculkan response message
Hasil yang Didapatkan	• Data *device token* masuk ke *database.* • Aplikasi android memunculkan response message
Hasil Pengujian	Berhasil
Kondisi Akhir	Data *device token* ditambahkan pada *database*



Gambar 5.1 *Response message* ketika *device token* berhasil teregistrasi

Gambar 5.2 *Response message* ketika *device token* telah teregistrasi sebelumnya



Gambar 5.3 *Response message* ketika parameter input salah

Kode Sumber 5.1 ***Post Request Register Device Token***

```
{
    "username": "dwnggkt",
    "clientID": "3E1R...DAEQ",
    "deviceToken": "f2Vfm...bsYs",
    "deviceType": "A"
}
```

Kode Sumber 5.2 Response ***Register Device Token*** **dengan** ***Error False***

```
{
    "error": "false",
    "message": "Device registered succesfully"
}
```

Kode Sumber 5.3 Response ***Register Device Token*** **dengan** ***Error False***

```
{
    "error": "false",
    "message": "Device already registered"
}
```

Kode Sumber 5.4 Response ***Register Device Token*** **dengan** ***Error True***

```
{
    "error": "true",
    "message": "Device not registered"
}
```

5.2.1.2 Pengujian Mengirim Notifikasi

Pengujian mengirim notifikasi merupakan pengujian terhadap kemampuan program untuk melakukan pengiriman notifikasi. Pengujian mengirim notifikasi dibagi menjadi tiga skenario, sesuai dengan tipe pengiriman notifikasi:

1. Single, menggunakan *username*, *client-id* sebagai parameter mendapatkan *device token.*

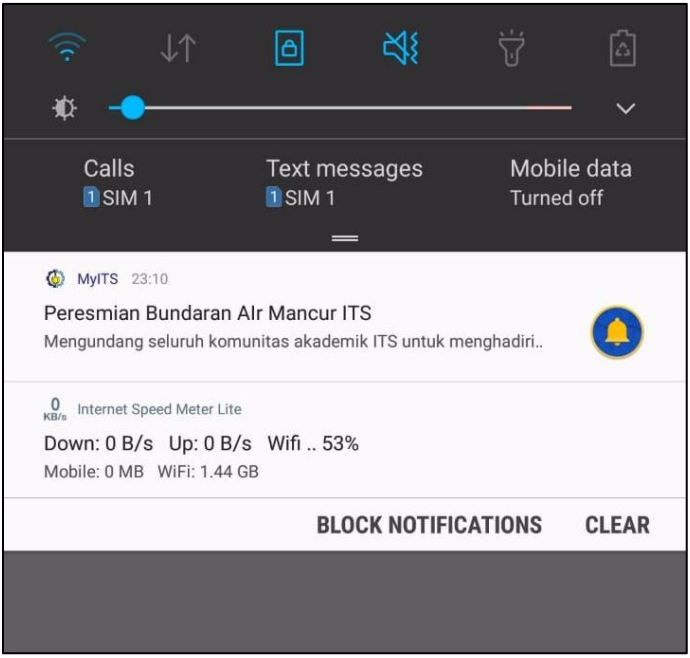
2. Multiple, berdasarkan *user-id* atau *username* dan *client-id* sebagai parameter mendapatkan *device token*
3. Targeted, berdasarkan mapping *user role* dan *user unit*, *client-id* sebagai parameter mendapatkan *device token.*

Rincian skenario pada kasus penggunaan ini bisa dilihat pada Tabel 5.3

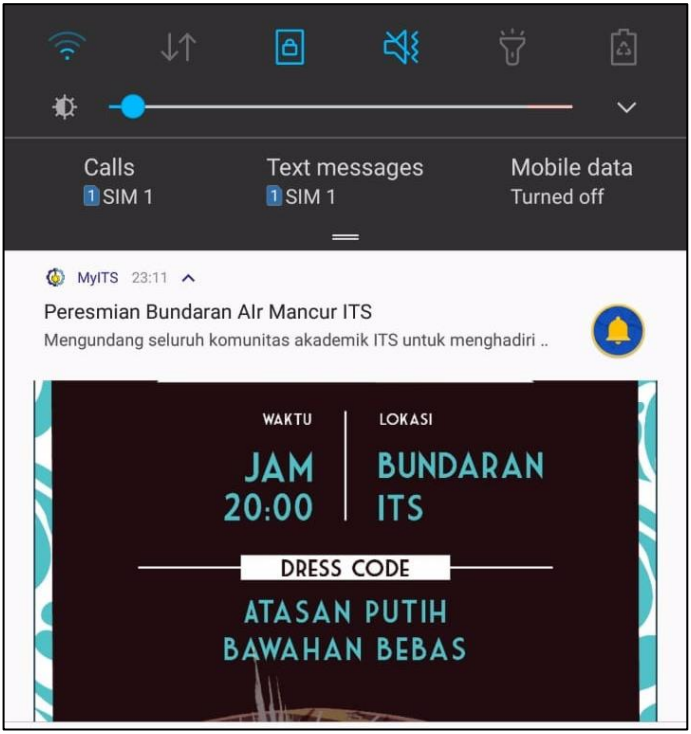
Tabel 5.3 Pengujian Mengirim Notifikasi

No.Pengujian	**UJ002**
Referensi Kasus Penggunaan	UC002
Nama	Pengujian mengirim notifikasi
Aktor	Client
Tujuan Pengujian	Menguji fungsionalitas untuk mengirim notifikasi
Skenario Pengujian	Mengirim post request menggunakan *postman*
Kondisi Awal	Aplikasi *dummy* pada perangkat bergerak belum menerima notifikasi
Data Uji	-
Langkah Pengujian	1. Administrator mengirim post *request* menggunakan postman dengan headers dan message body sesuai dengan parameter. 2. Sistem mengirim notifikasi menuju *Backend as a Service* 3. Administrator melihat perangkat bergerak
Hasil yang diharapkan	1. Notifikasi masuk pada perangkat bergerak. *2.* Pesan notifikasi tercatat pada tabel *pn_packet* atau *packet monitoring*
Hasil yang didapatkan	1. Notifikasi berhasil masuk pada perangkat bergerak 2. Pesan notifikasi berhasil tercatat tabel *pn_packet*

Hasil Pengujian	Berhasil
Kondisi Akhir	Muncul notifikasi pada perangkat bergerak



Gambar 5.4 Pesan Notifikasi yang Masuk Pada Perangkat Bergerak (Tanpa Gambar)



Gambar 5.5 Pesan Notifikasi yang Masuk Pada Perangkat Bergerak (Dengan Gambar)

5.2.1.3 Pengujian Mendapatkan Data Tujuan Pengiriman Notifikasi

Data tujuan pengiriman notifikasi berbeda untuk setiap token akses. Maka akan dilakukan pengujian dengan melakukan *request* menuju *endpoint api* dengan token akses yang berbeda. Rincian skenario pada kasus penggunaan ini bisa dilihat pada Tabel 5.4

Tabel 5.4 Pengujian Mendapatkan Data Tujuan Pengiriman Notifikasi

No.Pengujian	**UJ003**
Referensi Kasus Penggunaan	UC003

Nama	Pengujian mendapatkan data tujuan pengiriman notifikasi
Aktor	Client
Tujuan Pengujian	Menguji fungsionalitas untuk mendapatkan data tujuan pengiriman notifikasi
Skenario Pengujian	Mengirim post request menggunakan *postman*
Kondisi Awal	Data tujuan pengiriman notifikasi belum didapatkan
Data Uji	-
Langkah Pengujian	Administrator mengirim post *request* menggunakan postman dengan headers dan message body sesuai dengan parameter.
Hasil yang diharapkan	Mendapatkan data tujuan pengiriman notifikasi dalam format JSON
Hasil yang didapatkan	Data pengiriman notifikasi dalam format JSON berhasil didapatkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Mendapatkan data tujuan pengiriman notifikasi dalam format JSON

Kode Sumber 5.5 *Response Message* Data Tujuan Pengiriman Notifikasi

```
// Token Akses
// JwpGbKyhFUndtfaEKETcEvitzcEERcINipIQKxSG

[
    {
        "id_pn_dest": "06425290-5B9A-426A-9BE8-
ACBA50C23AD7",
        "r_name": "Rektor",
        "role_id": "8"
    },
    {
        "id_pn_dest": "C3257BD9-F7AD-48F6-A15B-
B0C7A57E175C",
```

```
        "r_name": "Dekan",
        "role_id": "2",
        "u_name": "FTIK",
        "unit_id": "2"
    },
    {
        "id_pn_dest": "09D4CBB0-3BDD-41F6-B605-
B64AEBFAD8F7",
        "r_name": "Kepala Departemen",
        "role_id": "1",
        "u_name": "S1 Informatika",
        "unit_id": "3"
    },
    {
        "id_pn_dest": "95C16487-224A-47BC-97D4-
548C9CA83274",
        "r_name": "Kepala Prodi",
        "role_id": "12",
        "u_name": "S1 Informatika",
        "unit_id": "3"
    },
    {
        "id_pn_dest": "8029E79A-D3F2-498B-BE2B-
4D4A9C8E2A31",
        "r_name": "Mahasiswa",
        "role_id": "3",
        "u_name": "S1 Informatika",
        "unit_id": "3"
    },
    {
        "id_pn_dest": "95D0766C-82A3-4768-9A79-
6A06364C6E26",
        "r_name": "Mahasiswa",
        "role_id": "3",
        "u_name": "S2 Informatika",
        "unit_id": "4"
    }
]
```

Kode Sumber 5.6 *Response Message* Data Tujuan Pengiriman Notifikasi

```
// Token Akses
// YqIdnLjfHxuGLWgurectaNrKyhJaWUsBQQEQvpbu
```

```
[
    {
        "id_pn_dest": "5913F134-FF00-423F-9C65-
D3CD4C11D549",
        "r_name": "Rektor",
        "role_id": "8"
    },
    {
        "id_pn_dest": "65DA800A-742B-4904-A67C-
75A907DF2540",
        "r_name": "Kepala Prodi",
        "role_id": "12",
        "u_name": "S1 Informatika",
        "unit_id": "3"
    }
]
```


5.2.1.4 Pengujian Mendapatkan Status Aktif Device Token

Data tujuan pengiriman notifikasi berbeda untuk setiap token akses. Maka akan dilakukan pengujian dengan melakukan *request* menuju *endpoint api* dengan token akses yang berbeda. Rincian skenario pada kasus penggunaan ini bisa dilihat pada Tabel 5.4

Tabel 5.5 Pengujian Mendapatkan Status Aktif Device Token

No.Pengujian	**UJ004**
Referensi Kasus Penggunaan	UC004
Nama	Pengujian mendapatkan status aktif *device token*
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk mendapatkan status aktif device token
Skenario Pengujian	Mengirim post request menggunakan *postman*
Kondisi Awal	Data status aktif device token belum didapatkan
Data Uji	-

Langkah Pengujian	Administrator mengirim post *request* menggunakan postman dengan headers dan message body sesuai dengan parameter.
Hasil yang diharapkan	Mendapatkan data status aktif device token dalam format JSON
Hasil yang didapatkan	Data status aktif device token dalam format JSON berhasil didapatkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Mendapatkan data status aktif device token dalam format JSON

Kode Sumber 5.7 Response Message Mendapatkan Status Aktif Device Token

```
[
    {
        "device_token": "A1Rdq32...q34343n",
        "device_token_id": "09FDF...F39FA",
        "status": "200"
    },
    {
        "device_token": "f2vFMX...bsYs",
        "device_token_id": "BC7...9f354F6",
        "status": "404"
    }
]
```

5.2.1.5 Pengujian Menjalankan Scheduler

Pengujian menjalankan *scheduler* merupakan pengujian terhadap kemampuan aplikasi untuk menjalankan *scheduler*. *Scheduler* harus dinyalakan terlebih dahulu untuk melakukan pengiriman notifikasi sesuai dengan jadwal yang telah ditentukan.

Tabel 5.6 Pengujian Scheduling

No.Pengujian	**UJ005**
Referensi Kasus Penggunaan	UC005
Nama	Pengujian menjalankan scheduler
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk

	menjalankan *scheduling* pada sistem
Skenario Pengujian	Mengirim post request menggunakan *postman*
Kondisi Awal	*Scheduler* belum nyala
Data Uji	-
Langkah Pengujian	Administrator mengirim post *request* menggunakan postman dengan headers dan message body sesuai dengan parameter.
Hasil yang diharapkan	Berhasil menyalakan *scheduler* pada sistem
Hasil yang didapatkan	*Scheduler* pada sistem berhasil dinyalakan
Hasil Pengujian	Berhasil
Kondisi Akhir	*Scheduler* pada sistem nyala dan siap menangani notifikasi yang dikirim secara terjadwal

Kode Sumber 5.8 *Response message* ketika *Scheduler* Berhasil Dijalankan

```
{
    "scheduler_startTime": "2018/06/02 20:15:40",
    "scheduler_status": 1
}

// Scheduler status 1 = Aktif,
// Scheduler status 0 = Tidak Aktif
```

5.2.1.6 Pengujian Menjalankan Message Queue

Pengujian menjalankan *message queue* merupakan pengujian terhadap kemampuan aplikasi untuk menjalankan *message queue*. *Message queue* harus dinyalakan terlebih dahulu agar sistem dapat melakukan pengiriman notifikasi menuju *Backend as a Service*.

Tabel 5.7 Pengujian Menjalankan Message Queue

No.Pengujian	**UJ006**
Referensi Kasus Penggunaan	UC006
Nama	Pengujian menjalankan *message queue*
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk menjalankan *message queue* pada sistem
Skenario Pengujian	Mengirim post request menggunakan *postman*
Kondisi Awal	*Message queue* belum nyala
Data Uji	-
Langkah Pengujian	Administrator mengirim post *request* menggunakan postman dengan headers dan message body sesuai dengan parameter.
Hasil yang diharapkan	Berhasil menyalakan *message queue* pada sistem
Hasil yang didapatkan	*Message queue* pada sistem berhasil dinyalakan
Hasil Pengujian	Berhasil
Kondisi Akhir	*Message queue* pada sistem nyala dan siap menampung *packet* notifikasi untuk kemudian dikirim menuju *Backend as a Service*

Kode Sumber 5.9 *Response Message* ketika *Message Queue* Berhasil Dijalankan

```
// URI : /queue/start

{
    "error": "false",
    "message": "Message Queue Started",
    "msg_queue_startTime": "2018/07/01 16:58:52",
    "msg_queue_status": 1,
    "thread_number": 5
}
```

```
// Status 1 = Aktif, Status 0 = Tidak Aktif
```

5.2.2 Uji Coba Fungsionalitas Aplikasi Pengelola Konten Notifikasi

5.2.2.1 Pengujian Melihat Data Pengguna Serta Device Token yang Teregistrasi

Pengujian melihat data pengguna serta device token yang teregistrasi merupakan pengujian terhadap kemampuan aplikasi untuk melihat data pengguna serta device token yang teregistrasi.

Tabel 5.8 Pengujian Melihat Data Pengguna Serta Device Token yang Teregistrasi

No.Pengujian	**UJ007**
Referensi Kasus Penggunaan	UC007
Nama	Pengujian melihat data pengguna serta device token yang teregistrasi
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat data pengguna serta device token yang teregistrasi
Kondisi Awal	Terdapat data pengguna serta data *device token*
Langkah Pengujian	1. Membuka halaman dashboard 2. Sistem menampilkan data pengguna serta *device token* yang teregistrasi
Hasil yang diharapkan	Data pengguna dan *device token* yang teregistrasi dapat ditampilkan
Hasil yang didapatkan	Data pengguna dan *device token* yang teregistrasi berhasil ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman dashboard

5.2.2.2 Pengujian Menampilkan Daftar Peran Pengguna (User Role)

Pengujian menampilkan daftar peran pengguna (*user role*) merupakan pengujian terhadap kemampuan aplikasi untuk menampilkan daftar peran pengguna (*user role*).

Tabel 5.9 Pengujian Menampilkan Daftar Peran Pengguna

No.Pengujian	**UJ008**
Referensi Kasus Penggunaan	UC008
Nama	Pengujian menampilkan daftar peran pengguna (*user role*)
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk menampilkan daftar peran pengguna (*user role*)
Kondisi Awal	Terdapat data peran pengguna
Data Uji	-
Langkah Pengujian	1. Membuka halaman dashboard 2. Melakukan klik pada tab peran pengguna 3. Sistem menampilkan daftar peran pengguna
Hasil yang diharapkan	Data peran pengguna dapat ditampilkan
Hasil yang didapatkan	Data peran pengguna berhasil ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman dashboard

5.2.2.3 Pengujian Menampilkan Daftar Unit Pengguna

Pengujian menampilkan daftar unit pengguna merupakan pengujian terhadap kemampuan aplikasi untuk menampilkan daftar unit pengguna.

Tabel 5.10 Pengujian Menampilkan Daftar Unit Pengguna

No.Pengujian	**UJ009**
Referensi Kasus Penggunaan	UC009
Nama	Pengujian menampilkan daftar unit pengguna
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk menampilkan daftar unit pengguna
Kondisi Awal	Terdapat data unit pengguna
Langkah Pengujian	1. Membuka halaman dashboard 2. Melakukan klik pada tab daftar unit 3. Sistem menampilkan daftar unit pengguna
Hasil yang diharapkan	Data unit pengguna dapat ditampilkan
Hasil yang didapatkan	Data unit pengguna berhasil ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Data unit pengguna ditampilkan

5.2.2.4 Pengujian Melihat dan Mengubah Data Client

Pengujian melihat dan mengubah data client merupakan pengujian terhadap kemampuan aplikasi untuk melihat dan mengubah data client.

Tabel 5.11 Pengujian Melihat dan Mengubah Data Client

No.Pengujian	**UJ010**
Referensi Kasus Penggunaan	UC010
Nama	Pengujian melihat dan mengubah data client
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat dan mengubah data client
Kondisi Awal	Terdapat data klien
Data Uji	Klien "Siakad *Development*"
Langkah Pengujian	1. Membuka halaman daftar

	klien 2. Menampilkan daftar klien 3. Melakukan edit status moderasi pada salah satu klien 4. Menyimpan perubahan data 5. Status moderasi pada data klien berubah pada sistem 6. Menampilkan daftar klien dengan data yang telah diperbarui
Hasil yang diharapkan	1. Menampilkan daftar klien 2. Memperbarui status moderasi klien pada basis data
Hasil yang didapatkan	1. Berhasil Menampilkan daftar klien 2. Data berhasil diperbarui pada basis data
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan daftar klien

5.2.2.5 Pengujian Pengiriman Notifikasi dengan Antarmuka Pengguna

Pengujian pengiriman notifikasi dengan antarmuka pengguna merupakan pengujian terhadap kemampuan aplikasi untuk pengiriman notifikasi dengan antarmuka pengguna.

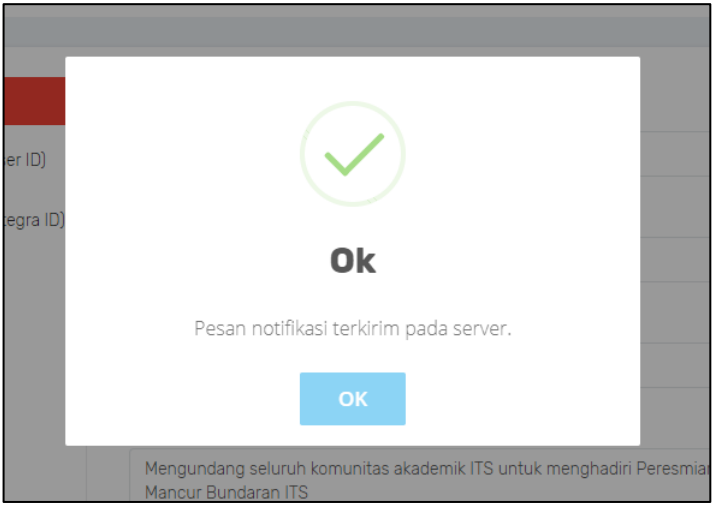
Tabel 5.12 Pengujian Pengiriman Notifikasi dengan Antarmuka Pengguna

No.Pengujian	**UJ011**
Referensi Kasus Penggunaan	UC011
Nama	Pengujian pengiriman notifikasi dengan antarmuka pengguna
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk pengiriman notifikasi dengan antarmuka pengguna
Skenario Pengujian 1	Mengirim notifikasi dengan tipe

	single
Kondisi Awal	Terdapat data *device token*, data tujuan pengiriman notifikasi dan data pengguna
Data Uji	-
Langkah Pengujian	1. Membuka halaman kirim notifikasi 2. Melakukan klik pada tab *single* 3. Mengisi form input (klien, username, title, message, image) 4. Melakukan klik tombol *send* 5. Menampilkan pesan *response* dari aplikasi *push notification* terpusat
Hasil yang diharapkan	Terdapat pesan response yang tampil setelah melakukan kirim notifikasi
Hasil yang didapatkan	Pesan *response* muncul
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kirim notifikasi kemudian pesan response tampil
Skenario Pengujian 2	Mengirim notifikasi dengan tipe *multiple* → *user_id*
Kondisi Awal	Terdapat data *device token*, data tujuan pengiriman notifikasi dan data pengguna
Data Uji	-
Langkah Pengujian	1. Membuka halaman kirim notifikasi 2. Melakukan klik pada tab *multiple* → *user id* 3. Mengisi form input (klien, user_id (*multiple*), title, message, image) 4. Melakukan klik tombol *send* 5. Menampilkan pesan *response*

	dari aplikasi *push notification* terpusat
Hasil yang diharapkan	Terdapat pesan response yang tampil setelah melakukan kirim notifikasi
Hasil yang didapatkan	Pesan *response* tampil
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kirim notifikasi kemudian pesan response muncul
Skenario Pengujian 3	Mengirim notifikasi dengan tipe *multiple → integra_id*
Kondisi Awal	Terdapat data *device token*, data tujuan pengiriman notifikasi dan data pengguna
Data Uji	-
Langkah Pengujian	1. Membuka halaman kirim notifikasi 2. Melakukan klik pada tab *multiple → integra id* 3. Mengisi form input (klien, integra_id (*multiple*), title, message, image) 4. Melakukan klik tombol *send* 5. Menampilkan pesan *response* dari aplikasi *push notification* terpusat
Hasil yang diharapkan	Terdapat pesan response yang tampil setelah melakukan kirim notifikasi
Hasil yang didapatkan	Pesan *response* tampil
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kirim notifikasi kemudian pesan response muncul
Skenario Pengujian 4	Mengirim notifikasi dengan tipe *multiple → targeted*
Kondisi Awal	Terdapat data *device token*, data

	tujuan pengiriman notifikasi dan data pengguna
Langkah Pengujian	1. Membuka halaman kirim notifikasi 2. Melakukan klik pada tab *multiple* → *targeted* 3. Mengisi form input (klien, destination (multiple), title, message, image) 4. Melakukan klik tombol *send* 5. Menampilkan pesan *response* dari aplikasi *push notification* terpusat
Hasil yang diharapkan	Terdapat pesan response yang tampil setelah melakukan kirim notifikasi
Hasil yang didapatkan	Pesan *response* tampil
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kirim notifikasi kemudian pesan response tampil



Gambar 5.6 Pesan Response Saat Mengirim Notifikasi

5.2.2.6 Pengujian Melihat Data Riwayat Notifikasi

Pengujian melihat data riwayat notifikasi merupakan pengujian terhadap kemampuan aplikasi untuk melihat data riwayat notifikasi.

Tabel 5.13 Pengujian Melihat Data Riwayat Notifikasi

No.Pengujian	**UJ012**
Referensi Kasus Penggunaan	UC012
Nama	Pengujian melihat data riwayat notifikasi
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat data riwayat notifikasi
Kondisi Awal	Terdapat data riwayat notifikasi
Langkah Pengujian	1. Membuka halaman *packet monitoring* 2. Sistem menampilkan data riwayat notifikasi
Hasil yang diharapkan	Data riwayat notifikasi dapat ditampilkan
Hasil yang didapatkan	Data riwayat notifikasi berhasil

	ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman riwayat notifikasi/*packet monitoring*

5.2.2.7 Pengujian Melihat Status Aktif Device Token

Pengujian melihat status aktif device token merupakan pengujian terhadap kemampuan aplikasi untuk melihat status aktif device token.

Tabel 5.14 Pengujian Melihat Status Aktif Device Token

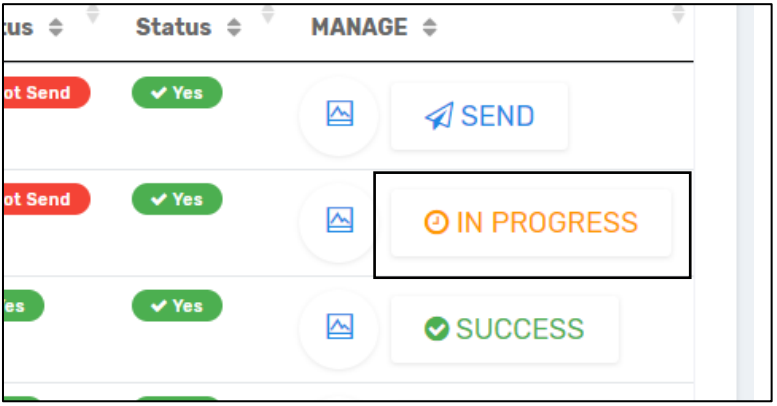
No.Pengujian	**UJ013**
Referensi Kasus Penggunaan	UC013
Nama	Pengujian melihat status aktif device token
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat status aktif *device token*
Kondisi Awal	Terdapat data *device token*
Langkah Pengujian	1. Membuka halaman *check token* 2. Sistem menampilkan status aktif *device token*
Hasil yang diharapkan	Menampilkan daftar status aktif *device token*
Hasil yang didapatkan	Berhasil menampilkan daftar status aktif *device token*
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan data status aktif *device token*

5.2.2.8 Pengujian Melakukan Moderasi Konten Notifikasi

Pengujian melakukan moderasi konten notifikasi merupakan pengujian terhadap kemampuan aplikasi untuk melakukan moderasi konten notifikasi.

Tabel 5.15 Pengujian Melakukan Moderasi Konten Notifikasi

No.Pengujian	**UJ014**
Referensi Kasus Penggunaan	UC014
Nama	Pengujian melakukan moderasi konten notifikasi
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melakukan moderasi konten notifikasi
Kondisi Awal	Terdapat data notifikasi yang harus dilakukan moderasi
Langkah Pengujian	1. Membuka halaman moderasi 2. Melakukan klik pada tombol *view detail* 3. Melihat konten notifikasi 4. Melakukan klik pada tombol *send* 5. Melakukan pengaturan waktu dan tanggal pengiriman 6. Tombol *send* berubah menjadi *in progress*. Paket notifikasi menunggu untuk dikirim oleh *scheduler*.
Hasil yang diharapkan	1. Waktu dan tanggal pengiriman ter-*update* 2. Tombol send berubah menjadi *in progress*
Hasil yang didapatkan	1. Waktu dan tanggal pengiriman berhasil ter-*update* 2. Tombol *send* berhasil berubah menjadi *in progress*
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman moderasi dengan data terbaru setelah dilakukan moderasi



Gambar 5.7 Perubahan Tombol Setelah Dilakukan Moderasi

5.2.2.9 Pengujian Melihat Daftar Pengguna

Pengujian melihat daftar pengguna merupakan pengujian terhadap kemampuan aplikasi untuk melihat daftar pengguna.

Tabel 5.16 Pengujian Melihat Daftar Pengguna

No.Pengujian	**UJ015**
Referensi Kasus Penggunaan	UC015
Nama	Pengujian melihat daftar pengguna
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat daftar pengguna
Kondisi Awal	Terdapat data pengguna
Langkah Pengujian	1. Membuka halaman *user* 2. Sistem menampilkan halaman *user*/daftar pengguna
Hasil yang diharapkan	Daftar pengguna dapat ditampilkan
Hasil yang didapatkan	Daftar pengguna berhasil ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman *user*

5.2.2.10 Pengujian Melihat Data Access Token

Pengujian melihat data access token merupakan pengujian terhadap kemampuan aplikasi untuk melihat data access token.

Tabel 5.17 Pengujian Melihat Data Access Token

No.Pengujian	**UJ016**
Referensi Kasus Penggunaan	UC016
Nama	Pengujian melihat data *access token*
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melihat data *access token*
Kondisi Awal	Terdapat data *access token*
Langkah Pengujian	1. Membuka halaman *access token* 2. Sistem menampilkan halaman *access token*
Hasil yang diharapkan	Daftar *access token* dapat ditampilkan
Hasil yang didapatkan	Daftar *access token* berhasil ditampilkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman *access token*

5.2.2.11 Pengujian Mengelola Data Tujuan Pengiriman Notifikasi Client

Pengujian mengelola data tujuan pengiriman notifikasi pada klien merupakan pengujian terhadap kemampuan aplikasi untuk mengelola data tujuan pengiriman notifikasi pada klien.

Tabel 5.18 Pengujian Mengelola Data Tujuan Pengiriman Notifikasi Client

No.Pengujian	**UJ017**
Referensi Kasus Penggunaan	UC017
Nama	Pengujian mengelola data tujuan pengiriman notifikasi *client*
Tujuan Pengujian	Menguji fungsionalitas untuk

	mengelola data tujuan pengiriman notifikasi klien
Skenario 1	Administrator melakukan penambahan data tujuan pengiriman notifikasi klien
Kondisi Awal	1. Tidak ada data tujuan pengiriman notifikasi klien 2. Halaman tujuan pengiriman notifikasi terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *add destination* 2. Administrator mengisi *pair role* dan *unit* tujuan pengiriman notifikasi 3. Administrator melakukan klik tombol *save* untuk menyimpan data. 4. Data tujuan pengiriman notifikasi berhasil ditambahkan.
Hasil yang diharapkan	Data tujuan pengiriman notifikasi dapat tersimpan oleh sistem.
Hasil yang didapatkan	Data tujuan pengiriman notifikasi berhasil tersimpan oleh sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman tujuan pengiriman notifikasi
Skenario 2	Administrator melakukan penghapusan data tujuan pengiriman notifikasi klien
Kondisi Awal	1. Terdapat data tujuan pengiriman notifikasi klien 2. Halaman tujuan pengiriman notifikasi terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *hapus* pada salah satu data tujuan notifikasi 2. Data tujuan pengiriman notifikasi berhasil terhapus.

Hasil yang diharapkan	Data tujuan pengiriman notifikasi dapat terhapus pada sistem.
Hasil yang didapatkan	Data tujuan pengiriman notifikasi berhasil terhapus pada sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman tujuan pengiriman notifikasi

5.2.2.12 Pengujian Menjalankan Scheduler

Pengujian menjalankan *scheduler* merupakan pengujian terhadap kemampuan aplikasi untuk menjalankan *scheduler* melalui antarmuka aplikasi pengelola konten notifikasi.

Tabel 5.19 Pengujian Menjalankan *Scheduler*

No.Pengujian	**UJ018**
Referensi Kasus Penggunaan	UC018
Nama	Pengujian menjalankan *scheduler*
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk menjalankan *scheduler*
Kondisi Awal	*Scheduler* belum dijalankan
Langkah Pengujian	1. Membuka halaman pengaturan → *Scheduler* 2. Menampilkan halaman scheduler 3. Melakukan klik pada tombol *start scheduler*
Hasil yang diharapkan	Scheduler dapat dijalankan, tombol berganti warna menjadi biru dan terdapat waktu kapan scheduler dinyalakan
Hasil yang didapatkan	Scheduler berhasil dijalankan, tombol berganti warna menjadi biru dan terdapat waktu kapan scheduler dinyalakan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman *scheduler* serta tampil notifikasi bahwa *scheduler* berhasil dinyalakan

5.2.2.13 Pengujian Konfigurasi Message Queue

Pengujian konfigurasi *message queue* merupakan pengujian terhadap kemampuan aplikasi untuk melakukan konfigurasi *message queue*.

Tabel 5.20 Pengujian Konfigurasi *Message Queue*

No.Pengujian	**UJ019**
Referensi Kasus Penggunaan	UC019
Nama	Pengujian konfigurasi *message queue*
Aktor	Administrator
Tujuan Pengujian	Menguji fungsionalitas untuk melakukan konfigurasi *message queue*
Skenario 1	Administrator menyalakan *message queue*
Kondisi Awal	1. *Message queue* belum berjalan pada sistem 2. Halaman pengaturan *message queue* ditampilkan
Langkah Pengujian	1. Administrator memasukkan jumlah *worker thread* 2. Administrator melakukan klik pada tombol *start scheduler* 3. *Scheduler* berhasil dinyalakan 4. Tampil notifikasi bahwa *scheduler* berhasil dinyalakan 5. Terlihat jumlah *worker thread* yang sedang berjalan
Hasil yang diharapkan	Scheduler dapat berjalan sejumlah *worker thread* yang dimasukkan
Hasil yang didapatkan	Scheduler berhasil berjalan sejumlah *worker thread* yang dimasukkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman konfigurasi *message queue*
Skenario 2	Administrator menambah *worker thread*

Kondisi Awal	1. *Message queue* telah berjalan pada sistem 2. Halaman pengaturan *message queue* ditampilkan
Langkah Pengujian	1. Administrator melakukan klik pada tombol *add thread* 2. *Worker thread* berhasil ditambahkan 3. Tampil notifikasi bahwa *worker thread* berhasil ditambahkan *3.* Terlihat jumlah *worker thread* yang sedang berjalan saat ini
Hasil yang diharapkan	*Worker thread* dapat ditambahkan
Hasil yang didapatkan	*Worker thread* berhasil ditambahkan
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman konfigurasi *message queue*
Skenario 3	Administrator mengurangi *worker thread*
Kondisi Awal	1. *Message queue* telah berjalan pada sistem 2. Halaman pengaturan *message queue* ditampilkan
Langkah Pengujian	1. Administrator melakukan klik pada tombol *remove thread* 2. *Worker thread* berhasil dikurangi 3. Tampil notifikasi bahwa *worker thread* berhasil dikurangi 4. Terlihat jumlah *worker thread* yang sedang berjalan saat ini
Hasil yang diharapkan	*Worker thread* dapat dikurangi
Hasil yang didapatkan	*Worker thread* berhasil dikurangi
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman konfigurasi

	message queue
Skenario 4	Administrator memberhentikan *message queue*
Kondisi Awal	1. *Message queue* telah berjalan pada sistem 2. Halaman pengaturan *message queue* ditampilkan
Langkah Pengujian	1. Administrator melakukan klik pada tombol *stop message queue* 2. *Message queue* berhenti
Hasil yang diharapkan	*Message queue* berhenti
Hasil yang didapatkan	*Message queue* berhasil berhenti
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman konfigurasi *message queue*

5.2.2.14 Pengujian Mengelola Sertifikat Klien

Pengujian mengelola sertifikat klien merupakan pengujian terhadap kemampuan aplikasi untuk mengelola data sertifikat pada klien.

Tabel 5.21 Pengujian Mengelola Sertifikat Klien

No.Pengujian	**UJ020**
Referensi Kasus Penggunaan	UC020
Nama	Pengujian mengelola sertifikat klien
Tujuan Pengujian	Menguji fungsionalitas untuk mengelola data sertifikat klien
Skenario 1	Administrator melakukan penambahan data sertifikat klien dengan tipe *backend as a service* APNs
Kondisi Awal	1. Tidak ada data sertifikat klien 2. Halaman kelola sertifikat klien terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *add certificates* 2. Administrator memilih tipe *backend as a service* APNs

	3. Administrator mengunggah file *.pem* 4. File *.pem* berhasil diunggah 5. Data sertifikat berhasil ditambahkan.
Hasil yang diharapkan	File *.pem* dapat diunggah, data sertifikat tersimpan oleh sistem
Hasil yang didapatkan	File *.pem* berhasil diunggah, data sertifikat berhasil tersimpan pada sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kelola sertifikat klien
Skenario 2	Administrator melakukan penambahan data sertifikat klien dengan tipe *backend as a service Google FCM*
Kondisi Awal	1. Tidak ada data sertifikat klien 2. Halaman kelola sertifikat klien terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *add certificates* 2. Administrator memilih tipe *backend as a service* Google FCM 3. Administrator memasukkan *key* 4. Data *key* berhasil ditambahkan.
Hasil yang diharapkan	Data key tersimpan oleh sistem
Hasil yang didapatkan	Data key berhasil tersimpan pada sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kelola sertifikat klien
Skenario 3	Administrator melakukan penghapusan sertifikat klien dengan tipe *backend as a service* APNs
Kondisi Awal	1. Ada data sertifikat klien 2. Halaman kelola sertifikat klien

	terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *delete* pada salah satu data 2. Data sertifikat berhasil dihapus.
Hasil yang diharapkan	Data sertifikat terhapus dalam sistem
Hasil yang didapatkan	Data sertifikat berhasil terhapus pada sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kelola sertifikat klien
Skenario 4	Administrator melakukan penghapusan data sertifikat klien dengan tipe *backend as a service Google FCM*
Kondisi Awal	1. Ada data sertifikat klien 2. Halaman kelola sertifikat klien terbuka
Langkah Pengujian	1. Administrator melakukan klik pada tombol *delete* pada salah satu data 2. Data sertifikat berhasil dihapus
Hasil yang diharapkan	Data sertifikat terhapus dalam sistem
Hasil yang didapatkan	Data sertifikat berhasil terhapus pada sistem.
Hasil Pengujian	Berhasil
Kondisi Akhir	Menampilkan halaman kelola sertifikat klien

5.3 Uji Coba REST API

Pengujian REST API dibagi menjadi dua bagian yaitu pengujian *response body* dan *respond time*. Kedua pengujian ini dilakukan untuk mengetahui kesesuaian *response body* dari endpoint REST API dan performa REST API ketika diakses oleh banyak pengguna.

5.3.1 Pengujian Response Body

Pengujian *response body* dilakukan pada *endpoint* di modul-modul REST API. Setiap *endpoint* diuji kesesuaian *response body* berdasarkan skenario yang telah ditentukan. Pengujian ini dilakukan dengan aplikasi *Postman*.

5.3.1.1 Skenario Pengujian

Terdapat dua skenario pengujian yang diterapkan pada setiap *endpoint* yaitu skenario A dan B, yang didesain untuk menguji parameter yang dikirim ketika mengakses *endpoint*.

Tabel 5.22 Skenario Pengujian *Response Body*

Kode Skenario	Nama Skenario	Deskripsi
A	Parameter Lengkap	• Pengujian dilakukan dengan mengirim seluruh parameter yang dibutuhkan oleh *endpoint.* • Pengujian berhasil jika *status code* dari *response body* bernilai 200 (*ok*)
B	Parameter Tidak Lengkap	• Pengujian dilakukan dengan mengirim sebagian parameter yang dibutuhkan oleh *endpoint*. Pengujian berhasil jika *status code* dari *response body* bernilai 400 (bad request)

5.3.1.2 Hasil Pengujian

Tabel 5.23 Hasil Pengujian *Response Body*

Kode Uji	Endpoint URI	Method	Deskripsi	Kasus Penggunaan	Skenario	
					A	B
Mengirim Notifikasi						
RA001	/send/destination?access_token = {access_token}	GET	Mendapatkan daftar tujuan pengiriman notifikasi	UC003	✔	✔
RA002	/send/single	POST	Mengirim data notifikasi dengan	UC002	✔	✔

Kode Uji	Endpoint URI	Me-thod	Deskripsi	Kasus Peng-gunaan	Skenario	
					A	B
			tipe *single*			
RA003	/send/multiple	POST	Mengirim data notifikasi dengan tipe *multiple*	UC002	✔	✔
RA004	/send/targeted	POST	Mengirim data notifikasi dengan tipe *targeted*	UC002	✔	✔
Cek *device token* status						
RA005	/check/token/ {device_type}	POST	Melakukan perintah untuk melakukan cek status *device token*	UC004	✔	✔
Scheduler						
RA006	/scheduler/start	POST	Menyalakan *scheduler*	UC005	✔	✔
RA007	/scheduler/status? access_token = {access_token}	GET	Melihat status *scheduler*	UC005	✔	✔
Message Queue						
RA008	/queue/start/ {thread_number}	POST	*Menyalakan message queue*	UC006	✔	✔
RA009	/queue/add_thread	POST	Menambah jumlah *worker thread* untuk mengirim notifikasi	UC006	✔	✔
RA010	/queue/ remove_thread	POST	Mengurangi jumlah *worker thread* untuk mengirim notifikasi	UC006	✔	✔
RA011	/queue/stop	POST	Memberhentikan *message queue*	UC006	✔	✔
RA012	/queue/show_stat	GET	Melihat status	UC006	✔	✔

Kode Uji	Endpoint URI	Me-thod	Deskripsi	Kasus Peng-gunaan	Skenario	
					A	B
	us?access_token = {access_token}		*message queue*.			
Register Device Token						
RA013	/register/device	POST	Melakukan registrasi *device token* aplikasi klien	UC001	✔	✔

Keterangan

✔ : Berhasil

5.3.2 Pengujian Response Time

Evaluasi Pengujian *response time* dilakukan untuk mengetahui performa server dan REST API ketika diakses oleh banyak pengguna dalam waktu yang hampir bersamaan. Pengujian ini diujikan dengan jumlah pengguna yang berbeda yaitu 100, 500, 1000 dan 2000. Aplikasi Apache JMeter digunakan untuk melakukan pengujian ini.

Thread Group digunakan untuk menyimulasikan jumlah pengguna di aplikasi JMeter. Satu thread diasumsikan sebagai satu pengguna. Sehingga pengujian dengan 100 pengguna akan menggunakan 100 *thread*. Parameter Ramp-Up-Period menentukan waktu yang dibutuhkan untuk mempersiapkan *thread*. Karena pengujian ini mengukur performa dan *response time* server ketika diakses hampir bersamaan, maka nilai Ramp-Up-Period di jadikan satu detik. Parameter Loop Count akan diisi dengan nilai satu karena pengujian hanya dilakukan satu kali.

Fitur HTTP Request digunakan untuk melakukan *request* ke server. Pengujian ini akan menggunakan *endpoint get* tujuan pengiriman notifikasi klien untuk melakukan *request* ke server. Penjelasan dari *endpoint* tersebut dapat dilihat pada diagram kasus dengan kode UC003.

untuk melihat laporan dari pengujian *JMeter*, dibutuhkan beberapa *Listener*. Pengujian ini menggunakan tiga *Listener* yaitu *Summary Report*, *View Results Tree* dan *View Results in Table*. *Summary Report* digunakan untuk melihat ringkasan dari pengujian. Dari laporan tersebut dapat dilihat rata-rata, minimum dan maksimum *response time*, *request* yang dapat ditangani oleh server per detik (*throughput*) dan persentase *error*. Informasi lebih detail mengenai hasil pengujian dapat dilihat di *View Results in Table*. Sedangkan *View Results Tree* menyajikan hasil informasi *response* server dari setiap *request* yang dilakukan. Hasil pengujian yang ditampilkan pada bab ini mengacu pada *Summary Report*.

5.3.2.1 Pengujian 100 Pengguna

Berdasarkan Tabel 5.24, pengujian 100 pengguna pada server menghasilkan tingkat keberhasilan 100 persen dan rata-rata *response time* 0,3 detik. *Response time* tersebut dapat dikatakan cepat dan dapat diandalkan.

Tabel 5.24 Hasil Pengujian *Response Time* 100 Pengguna

Kode Pengujian		**RT001**
Jumlah Pengguna		100
Persentase Keberhasilan		100%
***Response Time* (Detik)**	**Minimum**	0,205
	Maksimum	0,562
	Rata-Rata	0,334
Throughput		79

5.3.2.2 Pengujian 500 Pengguna

Berdasarkan Tabel 5.25, pengujian 500 pengguna pada server menghasilkan tingkat keberhasilan 100 persen dan rata-rata *response time* 2,3 detik. *Response time* tersebut dapat dikatakan cepat dan dapat diandalkan.

Tabel 5.25 Hasil Pengujian *Response Time* 500 Pengguna

Kode Pengujian	**RT002**
Jumlah Pengguna	500

Persentase Keberhasilan		100%
***Response Time* (Detik)**	**Minimum**	0,197
	Maksimum	3,599
	Rata-Rata	2,326
Throughput		112

5.3.2.3 Pengujian 1000 Pengguna

Berdasarkan Tabel 5.26, pengujian 1000 pengguna pada server menghasilkan tingkat keberhasilan 100 persen dan rata-rata *response time* 4,6 detik. *Response time* tersebut dapat dikatakan cepat dan dapat diandalkan.

Tabel 5.26 Hasil Pengujian *Response Time* 1000 Pengguna

Kode Pengujian		**RT003**
Jumlah Pengguna		1000
Persentase Keberhasilan		100%
***Response Time* (Detik)**	**Minimum**	0,220
	Maksimum	10,396
	Rata-Rata	4,664
Throughput		88

5.3.2.4 Pengujian 2000 Pengguna

Berdasarkan Tabel 5.26, pengujian 2000 pengguna pada server menghasilkan tingkat keberhasilan 80,7 persen dan rata-rata *response time* 8,8 detik. Tingkat keberhasilan 80,7 persen menandakan bahwa ada beberapa *request* yang gagal menerima *response* dari server.

Tabel 5.27 Hasil Pengujian *Response Time* 2000 Pengguna

Kode Pengujian		**RT004**
Jumlah Pengguna		2000
Persentase Keberhasilan		80,7%
***Response Time* (Detik)**	**Minimum**	0,582
	Maksimum	21,014
	Rata-Rata	8,889
Throughput		90

5.3.2.5 Pengujian 3000 Pengguna

Berdasarkan Tabel 5.28, pengujian 3000 pengguna pada server menghasilkan tingkat keberhasilan 63,8 persen dan rata-rata *response time* 10,5 detik. Tingkat keberhasilan 63,8 persen menandakan bahwa ada banyak *request* yang gagal menerima *response* dari server.

Tabel 5.28 Hasil Pengujian *Response Time* 3000 Pengguna

Kode Pengujian		**RT005**
Jumlah Pengguna		3000
Persentase Keberhasilan		63,8%
***Response Time* (Detik)**	**Minimum**	1,330
	Maksimum	19,253
	Rata-Rata	10,552
Throughput		107

5.4 Evaluasi Pengujian

5.4.1 Evaluasi Pengujian Fungsionalitas

Rangkuman mengenai evaluasi pengujian fungsionalitas dapat dilihat pada tabel. Dari semua data tersebut dapat dilihat bahwa semua skneario pengujian berhasil dan program berjalan dengan baik. Sehingga bisa disimpulkan bahwa fungsionalitas program ini bekerja dengan baik.

Tabel 5.29 Rangkuman Hasil Pengujian

Nama	Kode Uji	Hasil
Melakukan *register device token*	UJ001	Berhasil
Melakukan pengiriman notifikasi	UJ002	Berhasil
Mendapatkan data tujuan pengiriman notifikasi	UJ003	Berhasil
Mendapatkan daftar status aktif device token	UJ004	Berhasil
Melakukan perintah untuk menjalankan *scheduler*	UJ005	Berhasil
Melakukan perintah untuk	UJ006	Berhasil

menjalankan *message queue*		
Melihat data pengguna serta *device token* yang teregistrasi	UJ007	Berhasil
Melihat daftar peran pengguna (*user role*)	UJ008	Berhasil
Melihat daftar *unit* pengguna	UJ009	Berhasil
Melihat dan mengubah data *client*	UJ010	Berhasil
Melakukan pengiriman notifikasi dengan antarmuka pengguna	UJ011	Berhasil
Melihat data riwayat notifikasi (*packet monitoring*)	UJ012	Berhasil
Melihat status aktif *device token*	UJ013	Berhasil
Melakukan moderasi konten notifikasi	UJ014	Berhasil
Melihat daftar pengguna	UJ015	Berhasil
Melihat data *access token*	UJ016	Berhasil
Mengelola (menambah, mengubah, menghapus) data tujuan pengiriman notifikasi bagi *client*	UJ017	Berhasil
Melakukan perintah untuk menjalankan *scheduler*	UJ018	Berhasil
Melakukan perintah untuk menjalankan *message queuing*	UJ019	Berhasil
Mengelola Sertifikat Klien	UJ020	Berhasil

5.4.2 Evaluasi Pengujian Response Time

Perbandingan hasil pengujian *response time* REST API dapat dilihat pada Tabel 5.30. Berdasarkan data pada tabel tersebut, tingkat keberhasilan pengujian semakin menurun ketika jumlah pengguna bertambah. Jumlah pengguna 100, 500 dan 1000 memiliki tingkat keberhasilan sempurna. Jumlah pengguna 2000 memiliki tingkat keberhasilan 80,7 persen. Sedangkan jumlah pengguna 3000 memiliki tingkat keberhasilan 63,8 persen.

Dilihat dari segi *response time*, jumlah pengguna 100 memiliki *response time* yang sangat cepat yaitu 0,3 detik untuk

setiap *request*. Sedangkan jumlah pengguna 500 dan 1000 memiliki *response time* masing-masing 2,3 detik dan 4,6 detik. *Response time* untuk jumlah pengguna 2000 dan 3000 memiliki *response time* masing-masing 8,8 detik dan 10,5 detik. Dapat dikatakan bahwa *response time* dari jumlah pengguna 2000 dan 3000 sangat tinggi. *Response time* yang tinggi akan menghambat penggunaan aplikasi lain yang menggunakan REST API sehingga membuat penggunaan aplikasi tidak efisien.

Berdasarkan analisis di atas, dapat diketahui bahwa ketika terdapat lebih dari 2000 pengguna yang mengakses REST API dalam waktu yang hampir bersamaan menyebabkan tingkat keberhasilan lebih rendah dan *response time* yang lebih lama. Terjadinya hal tersebut disebabkan karena keterbatasan sumber daya perangkat keras khususnya kapasitas RAM sejumlah 512 MB pada *server* yang digunakan untuk pengujian. Spesifikasi *server* ditunjukkan pada bagian lingkungan pengujian halaman 167. Semakin banyak jumlah pengguna yang mengakses REST API, maka semakin tinggi pula penggunaan *CPU*, *memory*, serta *network* pada *server*. Jika dilakukan implementasi aplikasi secara *real* agar tidak terjadi penurunan *response time*, maka harus digunakan server dengan spesifikasi yang lebih tinggi dan juga mengakitfkan *cache* pada sistem sehingga jika ada *request* yang sama maka data diambil hanya dari *cache*, tidak perlu dilakukan proses ulang data dari awal.

Tabel 5.30 Perbandingan Hasil Pengujian *Response Time*

Kode Pengujian *Response Time*	Jumlah Pengguna	Tingkat Keberhasilan	Rata-Rata *Response Time* (Detik)
RT001	100	100%	0,334
RT002	500	100%	2,326
RT003	1000	100%	4,664
RT004	2000	80,7%	8,889
RT005	3000	63,8%	10,552

{Halaman ini sengaja dikosongkan}

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diperoleh selama pengerjaan Tugas Akhir dan saran mengenai pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dalam proses pengerjaan tugas akhir dari tahap pendahuluan, kajian pustaka, analisis, perancangan, implementasi dan pengujian aplikasi *push notification* terpusat diperoleh kesimpulan sebagai berikut.

1. Aplikasi *push notification* terpusat mampu melakukan pengiriman notifikasi menuju *backend as a service* Google FCM.
2. REST API untuk aplikasi *push notification* terpusat berhasil diimplementasikan menggunakan *jersey restful web services*. Untuk melakukan keamanan dalam pertukaran data antara aplikasi klien dan API, *access token* disisipkan di dalam *header* setiap kali aplikasi klien melakukan *request* API.
3. REST API telah memenuhi fungsionalitas aplikasi berdasarkan dari hasil pengujian *response body* dengan dua skenario. Dalam segi performa, REST API dapat diandalkan dengan kemampuan menangani 1000 pengguna yang mengakses REST API dalam waktu yang hampir bersamaan. Selebih dari angka tersebut, terjadi penurunan kapabilitas server untuk menangani *request*.
4. *Scheduler* dan *message queue* berhasil diimplementasikan pada aplikasi *push notification* terpusat. Untuk menjalankan *scheduler* dan *message queue* harus dilakukan oleh admin setelah *web service* dijalankan pada *server*.

5. Aplikasi pengelola konten notifikasi berhasil diterapkan untuk mengelola aplikasi *push notification* terpusat menggunakan kerangka kerja Codeigniter.

6.2 Saran

Berikut merupakan beberapa saran untuk pengembangan aplikasi *push notification* terpusat berdasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Menambah kapabilitas aplikasi *push notification* terpusat untuk melakukan pengiriman notifikasi menuju *backend as a service* APNs.
2. Menambah kapabilitas aplikasi *push notification* terpusat untuk melakukan pengiriman *web push notification* menggunakan *backend as a service* FCM.
3. Menambah kapabilitas aplikasi *push notification* terpusat untuk menghapus data *device token* yang sudah tidak aktif.
4. Perlu dilakukan ujicoba pada jumlah pengguna sesuai kondisi yang sebenarnya dengan dukungan infrastruktur yang memadai.

DAFTAR PUSTAKA

[1] H. H. Wilmer, L. E. Sherman and J. M. Chein, "Smartphones and Cognition: A Review of Research Exploring the Links between Mobile Technology Habits and Cognitive Functioning," Department of Psychology, Temple University, Philadelphia, PA, USA, 25 April 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5403814/. [Accessed 29 December 2017].

[2] A. Developer, "Local and Remote Notification Programming Guide," Apple, 13 November 2017. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/. [Accessed 22 December 2017].

[3] D. Rubio, "Google Cloud Messaging for Android (GCM) Unveiled, to Replace C2DM Framework," C4Media Inc., 13 August 2012. [Online]. Available: https://www.infoq.com/news/2012/08/GoogleCMReplacesC2Dm. [Accessed 21 December 2017].

[4] Apple, "Local and Remote Notifications Overview," Apple, 13 November 2017. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/. [Accessed 3 January 2018].

[5] "Google's Android OS: Past, Present, and Future," PhoneArena, 18 August 2011. [Online]. Available: https://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future_id21273. [Accessed 29 December 2017].

[6] N. Langley, "Write once, run anywhere?," Computer Weekly, [Online]. Available: http://www.computerweekly.com/feature/Write-once-run-

anywhere. [Accessed 8 June 2017].

[7] Sun Microsystems, Inc., "The Java Language Environment," Sun Microsystems, Inc., 1997. [Online]. Available: http://www.oracle.com/technetwork/java/intro-141325.html. [Accessed 8 June 2017].

[8] Android Studio, "Mengenal Android Studio," Android Studio, [Online]. Available: https://developer.android.com/studio/intro/index.html?hl=id. [Accessed 8 June 2017].

[9] Microsoft, "Microsoft Releases SQL Server 2012 to Help Customers Manage "Any Data, Any Size, Anywhere"," MIcrosoft News Center, 7 March 2012. [Online]. Available: http://www.microsoft.com/Presspass/press/2012/mar12/03-06SQLServer12PR.mspx. [Accessed 5 January 2018].

[10] Microsoft, "SQL Server Documentation," Microsoft, 10 October 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation. [Accessed 5 January 2018].

[11] "CodeIgniter," EllisLab, [Online]. Available: https://github.com/bcit-ci/CodeIgniter. [Accessed 29 December 2017].

[12] W. Bowman, "PHP MVC Framework Showdown: 7.1 Performance," Medium Corporation, 8 January 2017. [Online]. Available: https://medium.com/@asked_io/php-mvc-framework-showdown-7-1-performance-2da52ac9fcba. [Accessed 29 December 2017].

[13] Feridi, "Mengenal RESTful Web Services," CodePolitan, 25 February 2016. [Online]. Available: https://www.codepolitan.com/mengenal-restful-web-services. [Accessed 29 December 2017].

[14] W3C®, "Web Services Architecture," W3C®, 11 February 2004. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[Accessed 8 June 2017].

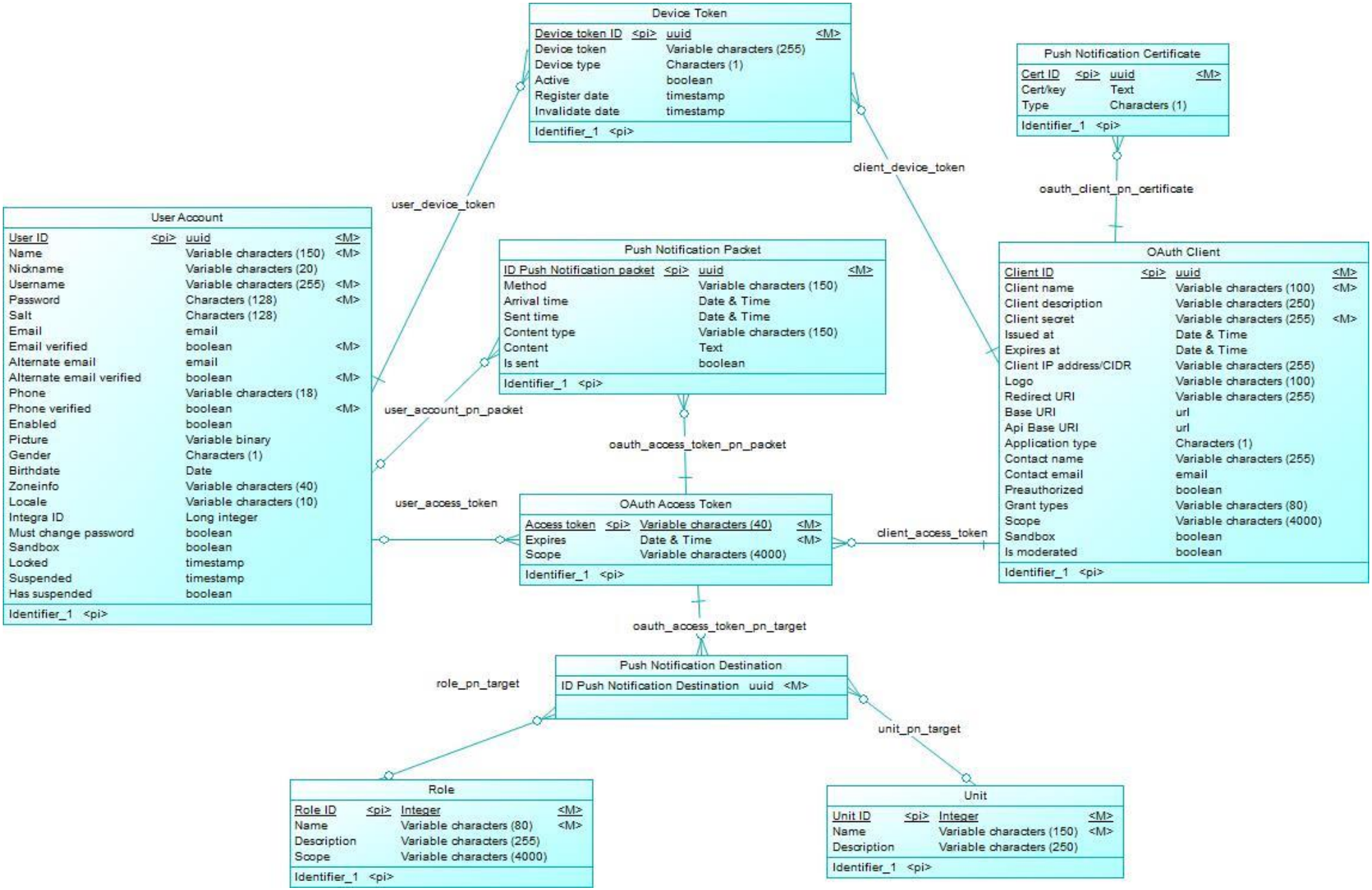
[15] Google, "Firebase Cloud Messaging," Google, 2018. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/. [Accessed 4 January 2018].

[16] Apple, "APNs Overview," Apple, 13 November 2017. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html. [Accessed 4 January 2018].

[17] Microsoft, "About Messages and Message Queues," Microsoft Developer Network, 2018. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms644927(VS.85).aspx. [Accessed 4 January 2018].

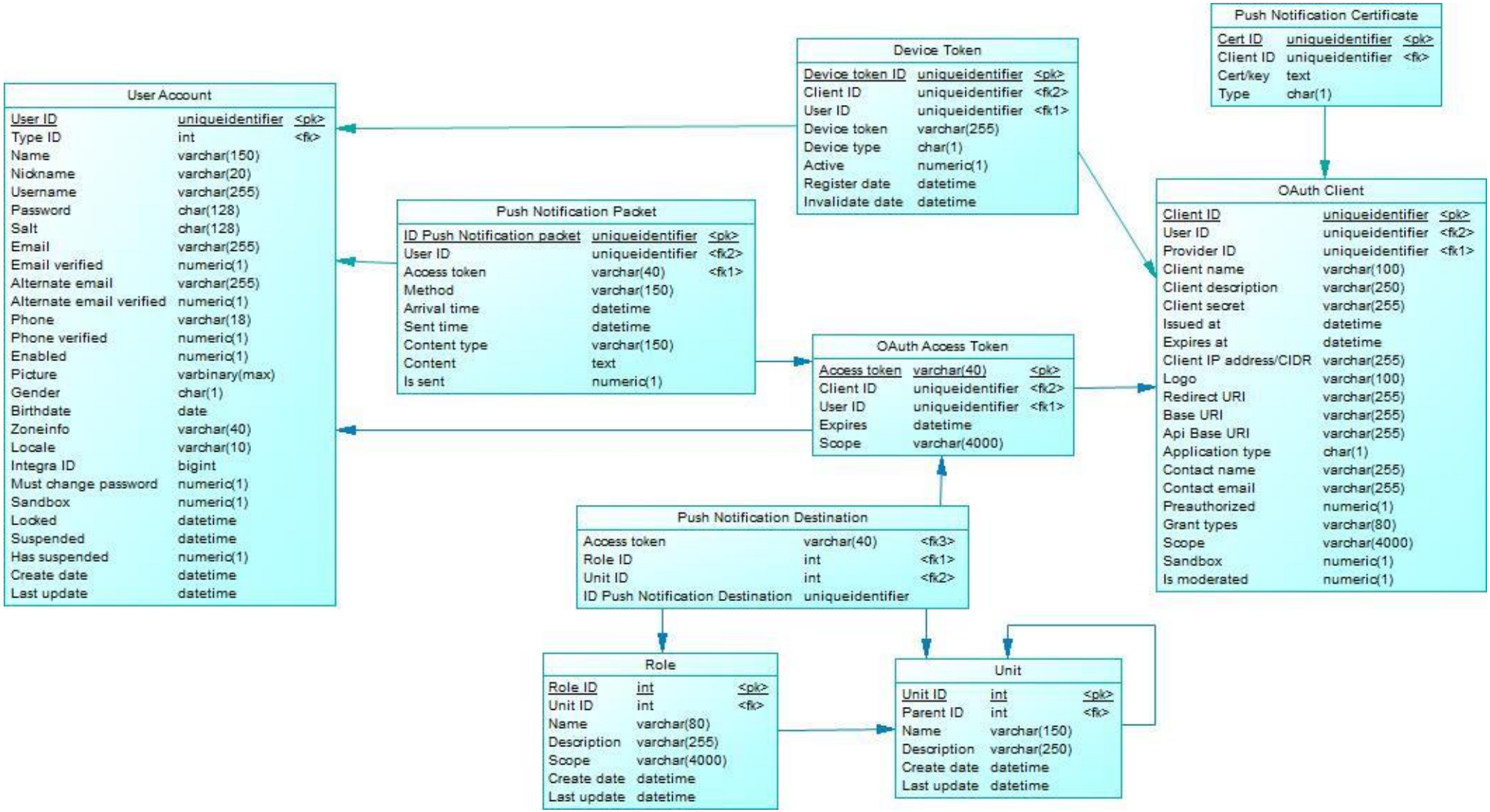
{Halaman ini sengaja dikosongkan}

Lampiran A Conceptual Data Model



{Halaman ini sengaja dikosongkan}

Lampiran B Physical Data Model



{Halaman ini sengaja dikosongkan}

BIODATA PENULIS

Penulis, **Dewangga Okta Wahyudianto**, lahir di Malang, 8 Oktober 1997. Penulis menempuh pendidikan sekolah dasar di Madrasah Ibtidaiyah Negeri 1 Malang (2003-2009). Melanjutkan pendidikan sekolah menengah pertama di SMP Negeri 1 Malang (2009-2012) dan selanjutnya di SMA Negeri 4 Malang (2012-2014). Selanjutnya penulis melanjutkan pendidikan sarjana di Jurusan Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya. Penulis aktif mengerjakan beberapa proyek di antaranya adalah Sistem Informasi Keuangan Institut Pertanian Bogor. Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat rekayasa perangkat lunak (RPL). Penulis dapat dihubungi melalui akun instagram dengan nama pengguna "dwnggkt" atau melalui surel *dewanggaokta@gmail.com*.

{Halaman ini sengaja dikosongkan}