TUGAS AKHIR – IF184802

INTEGRASI WSO2 API *MANAGER* DENGAN MYITS *SINGLE SIGN-ON* BERBASIS OAUTH2

VINCENT MARCELLO DWI TANUJAYA
NRP 05111640000089

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

INTEGRASI WSO2 API *MANAGER* DENGAN MYITS *SINGLE SIGN-ON* BERBASIS OAUTH2

VINCENT MARCELLO DWI TANUJAYA
NRP 05111640000089

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – IF184802

WSO2 API MANAGER INTEGRATION WITH OAUTH2-BASED MYITS SINGLE SIGN-ON

VINCENT MARCELLO DWI TANUJAYA
NRP 05111640000089

Advisor
Rizky Januar Akbar, S.Kom., M.Eng.
Abdul Munif, S.Kom., M.Sc.

INFORMATICS ENGINEERING DEPARTMENT
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

INTEGRASI WSO2 API *MANAGER* DENGAN MYITS *SINGLE SIGN-ON* BERBASIS OAUTH2

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Rekayasa Perangkat Lunak
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
VINCENT MARCELLO DWI TANUJAYA
NRP : 05111640000089

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Rizky Januar Akbar, S.Kom., M.Eng. .....
NIP: 198701032014041001 (pembimbing 1)

Abdul Munif, S.Kom., M.Sc. .....
NIP: 198608232015041004 (pembimbing 2)

SURABAYA
JANUARI 2020

SURABAYA
JANUARI 2020

[Halaman ini sengaja dikosongkan]

INTEGRASI WSO2 API *MANAGER* DENGAN MYITS *SINGLE SIGN-ON* BERBASIS OAUTH2

Nama Mahasiswa	: VINCENT MARCELLO DWI TANUJAYA
NRP	: 05111640000089
Departemen	: Teknik Informatika – ITS
Dosen Pembimbing I	: Rizky Januar Akbar, S.Kom., M.Eng.
Dosen Pembimbing II	: Abdul Munif, S.Kom., M.Sc.

Abstrak

Di dalam organisasi (dalam hal ini DPTSI-ITS), yang mengimplementasikan atau menjalankan WSO2 API Manager sekaligus MyITS Single Sign-On sebagai authorization server, menghadapi masalah di mana WSO2 API Manager memiliki basis data pengguna dan data client (Aplikasi yang mengakses data pengguna). Sedangkan di lain pihak, MyITS Single Sign-On juga memiliki basis data pengguna dan data client sendiri.

Permasalahan basis data yang terpisah ini membuat operasional menjadi sulit, sebagai contoh apabila ada API Resource yang diproteksi oleh WSO2 API Manager, maka API Manager akan mengeluarkan access token secara mandiri untuk dapat mengakses API atau Resource yang diproteksi oleh WSO2 API Manager. Pengguna dan client yang dapat mengakses resource tersebut hanyalah pengguna dan client yang terdaftar di basis data WSO2 API Manager. Padahal MyITS Single Sign-On mempunyai data-data tersebut.

Tugas akhir ini ingin menyelesaikan masalah tersebut dengan melakukan integrasi antara WSO2 API Manager dengan MyITS Single Sign-On di mana MyITS Single Sign-On digunakan sebagai external authorization server oleh WSO2 API Manager. Sehingga, WSO2 API Manager tidak perlu menggunakan basis datanya sendiri dan tidak perlu menggunakan built-in authorization server-nya sendiri melainkan menggunakan MyITS Single Sign-On sebagai external authorization server.

Kata kunci: WSO2 API Manager, Single Sign-On, Authorization Server

[Halaman ini sengaja dikosongkan]

WSO2 API MANAGER INTEGRATION WITH OAUTH2-BASED MYITS SINGLE SIGN-ON

Name	: VINCENT MARCELLO DWI TANUJAYA
NRP	: 05111640000089
Major	: Informatics Engineering Department – ITS
Supervisor I	: Rizky Januar Akbar, S.Kom., M.Eng.
Supervisor II	: Abdul Munif, S.Kom., M.Sc.

Abstract

Within the organization (in this case DPTSI-ITS), which implements or runs the WSO2 API Manager as well as MyITS Single Sign-On as an authorization server, faces problems where the WSO2 API Manager has a user database and client data (Applications that access user data). On the other hand, MyITS Single Sign-On also has its user database and client data.

This separate database problem makes operations difficult, for example, if there is an API Resource protected by WSO2 API Manager, the API Manager will issue an access token independently to be able to access the API or Resource protected by WSO2 API Manager. Users and clients that can access these resources are only users and clients registered in the WSO2 API Manager database. Even though MyITS Single Sign-On has these data.

This final project wants to solve this problem by integrating WSO2 API Manager with MyITS Single Sign-On where MyITS Single Sign-On is used as an external authorization server by WSO2 API Manager. Thus, WSO2 API Manager does not need to use its database and does not need to use its built-in authorization server but instead uses MyITS Single Sign-On as an external authorization server.

Keywords: WSO2 API Manager, Single Sign-On, Authorization Server

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas penyertaan dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **"Integrasi WSO2 API *Manager* dengan MyITS *Single Sign-On* Berbasis OAuth2"**.

Penulis ingin mengucapkan terima kasih kepada semua pihak yang telah memberikan bimbingan, arahan dan dukungan baik secara langsung maupun tidak langsung selama proses pengerjaan tugas akhir ini maupun selama masa studi. Pihak tersebut antara lain:

1. Bapak Mesakh Tanujaya dan Ibu Poedjiwati selaku orang tua penulis dan keluarga besar yang selalu memberikan doa dan dukungan sehingga penulis dapat menyelesaikan tugas akhir dalam waktu yang diharapkan.
2. Bapak Rizky Januar Akbar, S.Kom., M.Eng. selaku dosen wali dan dosen pembimbing tugas akhir yang telah memberikan bimbingan, motivasi dan memberikan banyak masukan dalam pengerjaan tugas akhir ini.
3. Bapak Abdul Munif, S.Kom., M.Sc. selaku dosen pembimbing tugas akhir yang memberikan dukungan berupa ilmu, koreksi serta masukan-masukan yang dapat penulis kembangkan dari tugas akhir ini.
4. Bapak dan Ibu dosen Departemen Informatika ITS yang telah mengajarkan banyak ilmu yang berharga bagi penulis.
5. Bapak dan Ibu karyawan Departemen Informatika ITS atas berbagai bantuan yang telah diberikan kepada penulis selama masa perkuliahan.
6. Teman-teman Pengurus Harian HMTC Garang 2018/2019, yang memberi penulis kesempatan untuk berkembang dan belajar.
7. Muhammad Alam Cahya, Daniel Kurniawan, Ibrahim Tamtama Adi, Ismail Syarief, Fandy Putra Mohammad, Diana Hudani Kisyono, Dewi Sekarini, Almas Aqmarina,

Denise Sonia Rahmadina dan Isye Putri Roselin yang sudah menemani penulis selama menempuh masa studi dan membantu penulis selama pengerjaan tugas akhir.

8. Teman-teman Laboratorium Algoritma dan Pemrograman yang selalu memberikan penulis tempat yang nyaman untuk belajar dan bermain.
9. Teman-teman Laboratorium Rekayasa Perangkat Lunak yang telah memberikan tempat selama pengerjaan tugas akhir.
10. Teman-teman satu angkatan Informatika ITS 2016 yang saling menyemangati satu sama lain.
11. Pihak-pihak lain yang tidak bisa penulis sebutkan satu per satu.

Penulis memohon maaf bila masih ada kekurangan pada tugas akhir ini. Penulis juga berharap bahwa apa yang dihasilkan dari tugas akhir ini dapat memberikan manfaat bagi semua pihak.

Surabaya, Januari 2020

Vincent Marcello Dwi Tanujaya

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan tugas akhir yang meliputi latar belakang, rumusan masalah, batasan masalah, tujuan pembuatan, manfaat, metodologi pembuatan tugas akhir dan sistematika penulisan.

1.1. Latar Belakang

WSO2 API *Manager* merupakan *platform open source* yang menyediakan antarmuka *web* bagi tim *developer* untuk membagikan dan memonitor API, dan bagi *user* dapat mencari dan menggunakan API yang disediakan oleh tim *developer* [1]. Salah satu fungsi dari WSO2 API *Manager* adalah sebagai *server* otorisasi *(Authorization Server)* yang ditangani oleh komponen *key manager*. WSO2 API *Manager* memiliki *built-in authorization server*. Secara arsitektural, WSO2 API *Manager* dapat memanfaatkan *external authorization server* daripada menggunakan *built-in authorization server* [2].

Platform MyITS *Single Sign-On* adalah *authorization server* yang memanfaatkan protokol OAuth2 [3]. MyITS *Single Sign-On* menyimpan data akun pengguna dan data *client* yang dapat diberi otorisasi dan memberikan otorisasi dengan mengeluarkan *access token* dan dapat melakukan verifikasi *access token.*

Di dalam organisasi (dalam hal ini DPTSI-ITS), yang mengimplementasikan atau menjalankan WSO2 API *Manager* sekaligus MyITS *Single Sign-On* sebagai *authorization server*, menghadapi masalah di mana WSO2 API *Manager* memiliki basis data pengguna dan data *client* (Aplikasi yang mengakses data pengguna). Sedangkan di lain pihak, MyITS *Single Sign-On* juga memiliki basis data pengguna dan data *client* sendiri.

Permasalahan basis data yang terpisah ini membuat operasional menjadi sulit, sebagai contoh apabila ada API *Resource* yang diproteksi oleh WSO2 API *Manager*, maka API *Manager* akan mengeluarkan *access token* secara mandiri untuk dapat mengakses API atau *Resource* yang diproteksi oleh WSO2

API *Manager*. Pengguna dan *client* yang dapat mengakses *resource* tersebut hanyalah pengguna dan *client* yang terdaftar di basis data WSO2 API *Manager*. Padahal MyITS *Single Sign-On* mempunyai data-data tersebut.

Tugas akhir ini ingin menyelesaikan masalah tersebut dengan melakukan integrasi antara WSO2 API *Manager* dengan MyITS *Single Sign-On* di mana MyITS *Single Sign-On* digunakan sebagai *external authorization server* oleh WSO2 API *Manager*. Sehingga, WSO2 API *Manager* tidak perlu menggunakan basis datanya sendiri dan tidak perlu menggunakan *built-in authorization server*-nya sendiri melainkan menggunakan MyITS *Single Sign-On* sebagai *external authorization server*.

1.2. Rumusan Masalah

Perumusan masalah yang terdapat pada tugas akhir ini, antara lain adalah:

a. Bagaimana menyatukan basis data *client* dan pengguna yang terpisah antara WSO2 API *Manager* dengan MyITS *Single Sign-On*?
b. Bagaimana mengimplementasikan konektor antara MyITS *Single Sign-On* sebagai *server* otorisasi eksternal dengan WSO2 API *Manager*?
c. Bagaimana mengimplementasikan *client registration endpoint* pada MyITS *Single Sign-On* agar dapat digunakan sebagai *server* otorisasi?
d. Bagaimana mengimplementasikan OAuth2 *token introspection* pada MyITS *Single Sign-On* agar dapat digunakan sebagai *server* otorisasi?
e. Bagaimana melakukan konfigurasi pada WSO2 API *Manager* sehingga dapat mengakses MyITS *Single Sign-On* sebagai *server* otorisasi eksternal?

1.3. Batasan Masalah

Batasan masalah yang terdapat pada tugas akhir ini, sebagai berikut:

a. *Platform* yang digunakan adalah aplikasi MyITS *Single Sign-On* yang berbasis PHP.
b. Bahasa pemrograman yang digunakan untuk implementasi konektor adalah Java.
c. Spesifikasi WSO2 API *Manager* yang digunakan adalah WSO2 API *Manager* versi 2.6.0.
d. *Library* OAuth2 *Server* yang digunakan adalah Bshaffer versi 1.10.0.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah:

a. Menyatukan basis data *client* dan pengguna yang terpisah antara WSO2 API *Manager* dengan MyITS *Single Sign-On*.
b. Mengimplementasi konektor antara MyITS *Single Sign-On* sebagai *server* otorisasi eksternal dengan WSO2 API *Manager*.
c. Mengimplementasi fitur *client registration endpoint* pada MyITS *Single Sign-On* agar dapat digunakan sebagai *server* otorisasi.
d. Mengimplementasi fitur OAuth2 *token introspection* pada MyITS *Single Sign-On* agar dapat digunakan sebagai *server* otorisasi.
e. Melakukan konfigurasi pada WSO2 API *Manager* sehingga dapat mengakses MyITS *Single Sign-On* sebagai *server* otorisasi eksternal.

1.5. Manfaat

Manfaat dari hasil pembuatan tugas akhir ini adalah mempermudah operasional MyITS *Single Sign-On* dan memberikan kemudahan kepada pengguna dan *client* untuk mengakses *resource* pada WSO2 API *Manager*.

1.6. Metodologi Pembuatan Tugas Akhir

1. Penyusunan proposal tugas akhir

   Proposal tugas akhir ini berisikan mengenai deskripsi pendahuluan tentang tugas akhir yang diusulkan.

Pendahuluan ini terdiri dari latar belakang dari dibuatnya tugas akhir ini, rumusan masalah yang diangkat, batasan masalah, tujuan pembuatan tugas akhir, manfaat tugas akhir, metodologi yang berisikan penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Selain itu, terdapat tinjauan pustaka yang digunakan sebagai referensi pendukung dalam pembuatan tugas akhir, ringkasan tugas akhir dan jadwal pengerjaan tugas akhir.

2. Studi literatur

   Pada tahap ini, akan dipelajari beberapa referensi yang diperlukan dalam implementasi tugas akhir ini, antara lain:

   1. OAuth 2.0
   2. WSO2 API *Manager*
   3. Implementasi *third-party key manager* pada WSO2 API *Manager*

3. Analisis dan desain perangkat lunak

   Pada tahap analisis dan desain ini akan dilakukan analisa terhadap aplikasi yang sudah ada sebelumnya, yaitu MyITS *Single Sign-On* untuk melakukan penyesuaian terhadap protokol OAuth 2.0. Kemudian akan dilakukan pola perancangan untuk mekanisme integrasi yang sesuai untuk diterapkan pada *platform* MyITS *Single Sign-On* dan WSO2 API *Manager*.

4. Implementasi perangkat lunak

   Implementasi mekanisme ini akan dibangun menggunakan bahasa pemrograman Java dan menggunakan *Integrated Development Environtment* (IDE) Eclipse.

5. Pengujian dan evaluasi

   Pengujian pada sistem ini akan dilakukan dengan menggunakan metode *blackbox*. Pengujian *blackbox* merupakan metode pengujian untuk mengetahui apakah semua fungsi yang ada pada sistem telah berjalan sesuai kebutuhan fungsional yang telah didefinisikan.

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.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini terdiri atas beberapa bab yang tersusun secara sistematis, yaitu sebagai berikut.

1. BAB I. Pendahuluan

   Bab pendahuluan berisi penjelasan mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi yang digunakan selama penyusunan dan sistematika penulisan tugas akhir.
2. BAB II. Tinjauan Pustaka

   Bab tinjauan pustaka berisi hasil studi literatur yang digunakan sebagai dasar untuk menyelesaikan tugas akhir.
3. BAB III. Analsis dan Perancangan

   Bab ini berisi tentang analisis kebutuhan WSO2 API *Manager* untuk menerapkan *key manager* pihak ketiga dan perancangan kustomisasi *key manager* untuk MyITS *Single Sign-On.*
4. BAB IV. Implementasi Sistem

   Bab ini membahas implementasi dari perancangan terhadap perangkat lunak. Penjelasan berupa baris kode yang telah dibuat dan dapat berfungsi dengan baik.
5. BAB V. Pengujian dan Evaluasi

   Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dari sistem yang telah dibuat.
6. BAB VI. Kesimpulan dan Saran

   Bab ini merupakan bab terakhir yang berisi tentang kesimpulan yang didapat dari proses pembuatan tugas akhir beserta saran-saran untuk pengembangan kedepannya.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini membahas teori-teori yang mendukung pembuatan tugas akhir. Teori yang mendukung tersebut adalah OAuth 2.0, WSO2 API *Manager* dan Implementasi *Third Party Key Manager* pada WSO2 API *Manager*.

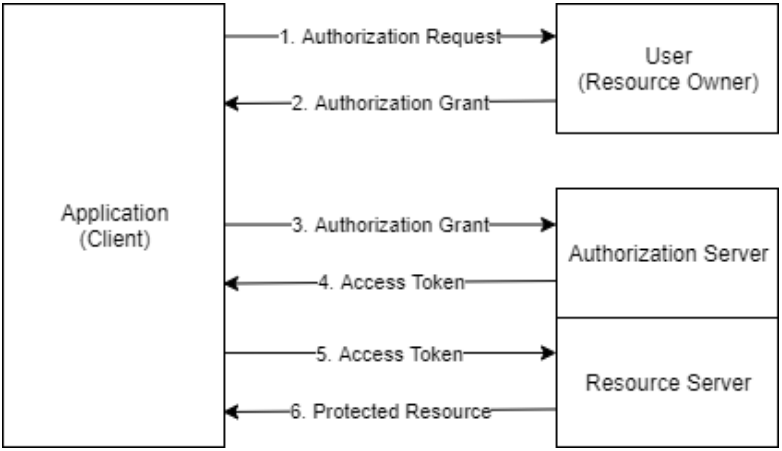
2.1. OAuth 2.0

OAuth2 merupakan kerangka kerja otorisasi yang memungkinkan aplikasi untuk mendapatkan akses terbatas ke akun pengguna pada layanan HTTP. OAuth2 bekerja dengan mendelegasikan autentikasi pengguna kepada layanan yang menyimpan data pengguna dan memberikan otorisasi kepada pihak ketiga untuk mengakses ke akun pengguna. OAuth2 menyediakan aliran otorisasi untuk aplikasi *web* dan aplikasi *mobile* [4].

OAuth2 memiliki beberapa *role* penting dalam menjalankan proses otorisasi, yaitu:

1. *Resource Owner*
   Resource Owner adalah pengguna yang melakukan otorisasi aplikasi untuk dapat mengakses ke akun mereka. Akses aplikasi ke akun pengguna terbatas pada otorisasi yang diberikan.
2. *Client*
   Client adalah aplikasi yang ingin mengakses akun pengguna. Sebelum melakukan akses, aplikasi harus memiliki otorisasi dari pengguna.
3. *Authorization Server*
   Authorization Server berperan untuk melakukan proses otorisasi yang diberikan pengguna kepada *client*.
4. *Resource Server*
   Resource Server berperan sebagai tempat penyimpanan data pengguna. Saat melakukan akses ke *resource server* harus

menyertakan *access token* yang diberikan oleh *Authorization Server*.



Gambar 2.1 Alur Protokol OAuth2

Gambar 2.1 menjelaskan mengenai alur pada protokol OAuth2. Secara umum, alur protokol OAuth2 sebagai berikut:

1. Aplikasi *client* meminta otorisasi untuk mengakses *resource* kepada *resource owner.*
2. Jika *user* melakukan otorisasi pada permintaan *client*, *client* akan menerima *authorization grant*.
3. Aplikasi *client* meminta *access token* kepada *authorization server* dengan menyertakan autentikasi *client* dan *authorization grant* dari *client.*
4. Jika identitas *client* dan a*uthorization grant* tervalidasi, *authorzation server* akan memberikan *access token* kepada *client*.
5. *Client* meminta *resource* kepada *resource server* dengan menyertakan *access token* sebagai autentikasinya.
6. Jika *access token* tervalidasi, *resource server* akan mengirimkan *resource* yang diminta kepada *client*.

2.2. WSO2 API *Manager*

WSO2 API *Manager* merupakan *platform* untuk membuat, mengelola, memakai dan memantau API [1]. Pada WSO2 API *Manager* terdapat beberapa modul dalam implementasinya yaitu:

a. API *Publisher*
API *Publisher* merupakan modul untuk membuat API baru dan mengelolanya.

b. API *Store*
API *Store* merupakan modul untuk mencari API dan melakukan *subscribe* API.

c. API *Gateway*
API *Gateway* merupakan modul yang mendasari API *runtime* yang berdasar pada WSO2 *Enterprise Integrator*.

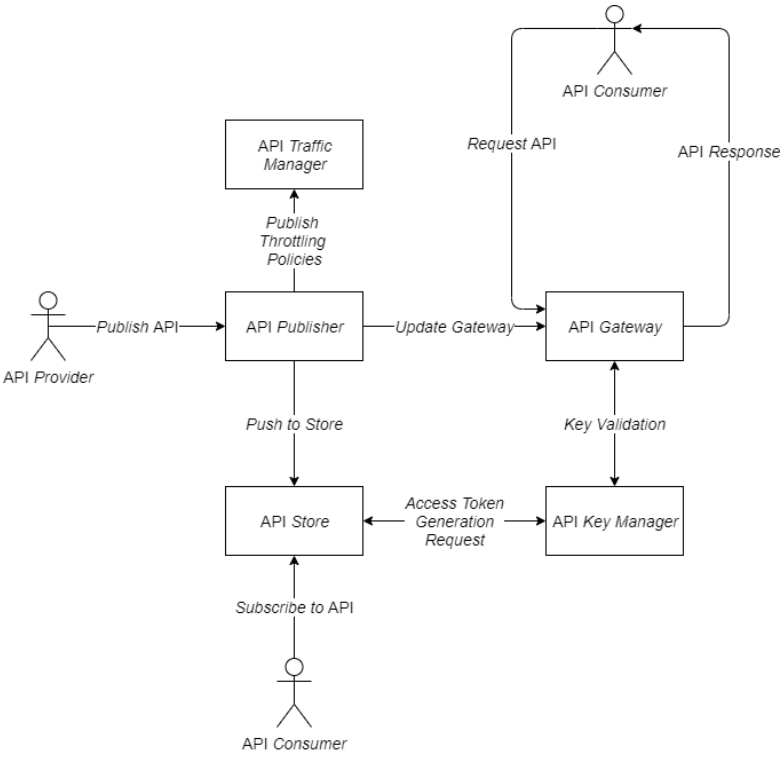
d. API *Key Manager*
API *Key Manager* merupakan modul yang bertanggung jawab pada pembuatan dan validasi *key* dan *access token.*

e. API *Traffic Manager*
API *Traffic Manager* merupakan modul yang melakukan limitasi pada tingkat permintaan API.

Pada Gambar 2.2 dijelaskan mengenai skema komunikasi antar modul pada WSO2 API *Manager*.

1. API *Publisher* berkomunikasi dengan API *Traffic Manager* untuk menerapkan kebijakan *throttling*.
2. API *Publisher* berkomunikasi dengan API *Gateway* untuk melakukan pembaharuan informasi API yang ada.
3. API *Publisher* berkomunikasi dengan API *Store* untuk melakukan penambahan API yang dibuat oleh API *Provider*.
4. API *Gateway* berkomunikasi dengan API *Key Manager* untuk melakukan validasi *key* yang diberikan oleh API *Consumer*.
5. API *Store* berkomunikasi dengan API *Key Manager* untuk melakukan permintaan *access token*.
6. API *Consumer* dapat melakukan *subscribe* ke API pada API *Store* dan melakukan *request* API pada API *Gateway*.

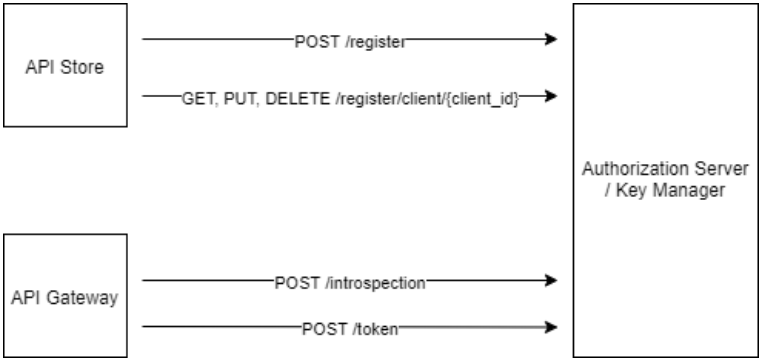
7. API *Provider* dapat memasarkan API yang telah dibuat pada API *Publisher*.



Gambar 2.2 Skema Hubungan antar Modul pada WSO2 API *Manager*

Pada tugas akhir ini, akan mengganti modul API *Key Manager* pada WSO2 API *Manager* dengan MyITS *Single Sign-On*. Modul ini memiliki interaksi dengan beberapa modul lain untuk menjalankan tugasnya seperti yang dapat dilihat pada Gambar 2.3, yaitu:

1. API *Store* dapat menambahkan, mengubah, menghapus dan mengambil data *client* pada *Key Manager*. Hal ini dilakukan dengan melakukan *request* kepada *client registration endpoint* pada Key Manager.
2. API *Gateway* dapat melakukan pengecekan terhadap *access token* yang diterima untuk mengetahui kevalidan dari *access token*. Hal ini dilakukan dengan melakukan *request* kepada *introspection endpoint* pada Key Manager.
3. API *Gateway* dapat melakukan *request access token* kepada Key Manager untuk mendapatkan *access token*. Hal ini dilakukan dengan melakukan *request* kepada *token endpoint* pada Key Manager.



Gambar 2.3 Hubungan Key Manager dengan Modul Lain Pada WSO2 API *Manager*

2.3. Implementasi *Third Party Key Manager* pada *WSO2 API Manager*

Key Manager menangani semua *client*, keamanan, dan operasi terkait *access token*. Pada API *Manager*, komponen-komponen yang berbeda meminta ke *Key Manager* untuk melakukan tugas yang berbeda. API *Gateway* terhubung dengan *Key Manager* untuk melakukan validasi terhadap *token* OAuth, *subscription*, dan

permohonan API. Ketika pelanggan meminta *access token* terhadap aplikasi menggunakan API *Store*, API *Store* melakukan panggilan ke *Key Manager* untuk membuat aplikasi OAuth dan mendapatkan *access token*. Hal yang sama juga berlaku untuk melakukan validasi *token*, API *Gateway* memanggil *Key Manager*, yang mengambil dan melakukan validasi detail *token* dari *database* [2].

Key Manager memisahkan *client* OAuth dan akses manajemen *token*, sehingga memungkinkan untuk menambahkan OAuth pihak ketiga. Dalam implementasinya, diperlukan *Key Manager Interface* untuk menghubungkannya [2]. Dalam *Key Manager Interface* terdapat beberapa *method* yang digunakan untuk melakukan operasi, yaitu:

a. *createApplication*
   Membuat aplikasi OAuth yang baru pada *Authorization Server*.
b. *updateApplication*
   Melakukan *update* pada aplikasi OAuth.
c. *retrieveApplication*
   Mengambil informasi aplikasi OAuth.
d. *deleteApplication*
   Menghapus aplikasi OAuth.
e. *getNewApplicationAccessToken*
   API *Store* memanggil *method* ini untuk mendapatkan *access token* aplikasi yang baru. *Method* ini dipanggil saat pertama kali dan ketika API *Store* membutuhkan pembaharuan dari *token* yang ada.
f. *getTokenMetaData*
   Mengambil detail dari *access token.*
g. *getKeyManagerConfiguration*
   Mengambil implementasi *Key Manager* dari *file api-manager.xml.*
h. *buildAccessTokenRequestFromJSON*
   Method ini akan menguraikan *input* JSON dan menambahkan nilai-nilai ke permintaan *Access Token.*

i. *mapOAuthApplication*
   Membuat aplikasi OAuth dalam mode *semi-manual* ketika *user* memiliki *key* dan *secret* konsumen yang sudah dihasilkan oleh *Key Manager*.
j. *buildAccessTokenRequestFromOAuthApp*
   Membuat *Access Token* menggunakan informasi dari aplikasi OAuth.
k. *registerNewResource*
   Method ini berkomunikasi dengan *resource registration endpoint* dari *authorization server* untuk membuat *resource* baru.
l. *getResourceByApiId*
   Method ini menerima *resource* yang sudah terdaftar melalui ID API.
m. *updateRegisteredResource*
   Method ini melakukan *update resource* yang sudah terdaftar melalui ID API.
n. *deleteRegisteredResourceByAPiId*
   Method ini digunakan untuk menghapus *resource* yang sudah terdaftar melalui ID API.
o. *deleteMappedApplication*
   Menghapus *mapping records* dari aplikasi OAuth.
p. *getActiveTokensByConsumerKey*
   Mendapatkan semua *token* aktif yang dihasilkan berdasarkan *Consumer Key*.
q. *getAccessTokenByConsumerKey*
   Mendapatkan detail dari *access token* yang ditampilkan pada *Store*.

Key Manager tidak terlepas dari proses validasi *key*. Pada saat menambahkan *key manager* pihak ketiga tidak terlepas dari *KeyValidationHandler* yang memiliki beberapa operasi utama [5], yaitu:

a. *validateToken*
   Melakukan validasi terhadap *token.*

b. *validateSubscription*
   Melewati atau melakukan perubahan domain validasi.
c. *validateScopes*
   Melonggarkan atau mengurangi batasan-batasan *scope*.
d. *generateConsumerToken*
   Membuat tipe yang berbeda dari *token.*

2.4. MyITS *Single Sign-On*

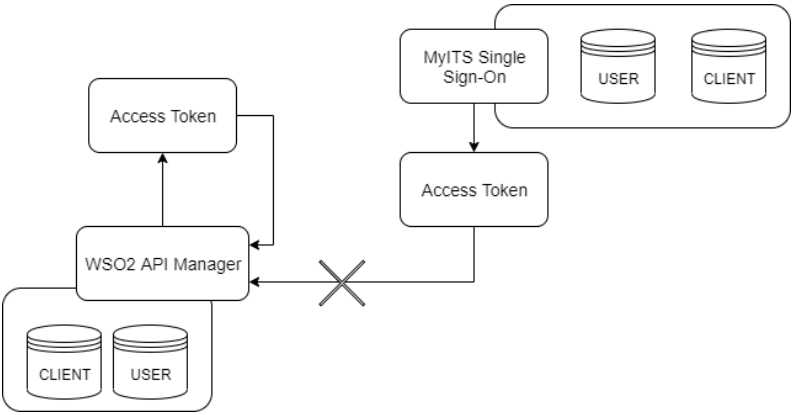
MyITS *Single Sign-On* (SSO) merupakan sistem autentikasi *Single Sign-On* dan otorisasi *role-based-access-control* (RBAC) ITS menggunakan standar OpenID *Connect* [3]. Sistem OpenID *Connect* yang saat ini digunakan oleh ITS diterapkan pada MyITS, dimana sistem-sistem internal pada Direktorat Pengembangan Sistem Informasi (DPTSI) ITS, seperti SI Akademik, SI SKEM, SI Beasiswa, SI Kurikulum dan lainnya terintegrasi dan dapat diakses oleh pengguna hanya dengan satu kali login.

Sistem menggunakan *authorization code flow* dimana pengguna dapat mendapatkan akses data mereka jika memiliki *access token*. *Access token* ini didapatkan ketika pengguna mengirimkan *Client Secret* dan *Client ID* kepada *token endpoint* pada MyITS *Single Sign-On*. Pengguna dapat mengakses *user info* mereka dengan melakukan *request* kepada *userinfo endpoint*.

Untuk saat ini, *access token* yang dikeluarkan oleh MyITS *Single Sign-On* hanya dapat digunakan untuk melakukan validasi terhadap *user info request* pada MyITS *Single Sign-On*. *Access Token* ini tidak dapat digunakan oleh *external resource server* seperti WSO2 API *Manager* yang digunakan oleh DPTSI ITS.

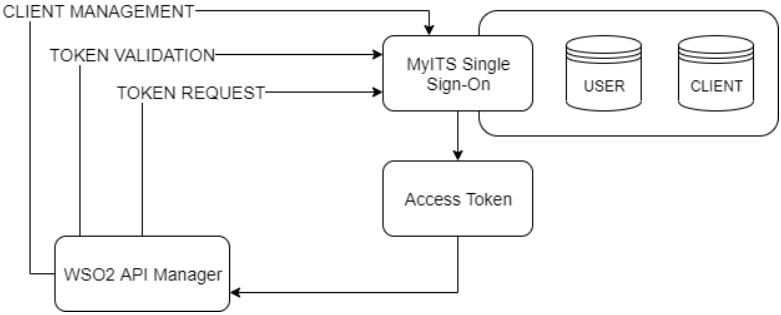
Untuk mengakses *resource* yang diproteksi oleh WSO2 API *Manager*. *User* atau *client* harus melakukan *request access token* kepada WSO2 API *Manager*. *Access token* yang dapat digunakan untuk mengakses *resource* hanya *access token* yang dikeluarkan oleh WSO2 API *Manager*. Sementara *access token* yang dikeluarkan oleh MyITS *Single Sign-On* tidak dapat digunakan oleh pengguna atau *client* untuk mengakses *resource* yang ada pada WSO2 API *Manager*. Gambar 2.4 menunjukkan hubungan

antara MyITS *Single Sign-On* dengan WSO2 API *Manager* saat ini.



Gambar 2.4 Skema hubungan antara MyITS *Single Sign-On* saat ini dengan WSO2 API *Manager*

Pada tugas akhir ini akan dilakukan pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On*. Sehingga *access token* yang didapatkan pada MyITS *Single Sign-On* dapat digunakan untuk mengakses *resource* pada WSO2 API *Manager*. Ketika WSO2 API *Manager* menerima *request resource* yang dengan disertakan *access token* yang didapatkan dari MyITS *Single Sign-On*. WSO2 *Manager* akan mengirimkan *request token validation* kepada MyITS *Single Sign-On*. Jika *access token* tervalidasi, maka *resource* akan diberikan kepada pengguna atau *client*. WSO2 API *Manager* juga dapat menambah, mengubah, menghapus dan mendapatkan detail informasi mengenai *client* pada MyITS *Single Sign-On*. WSO2 API *Manager* juga dapat melakukan *request access token* kepada MyITS *Single Sign-On* dengan berbagai *grant type*. Gambar 2.5 menunjukkan skema hubungan MyITS *Single Sign-On* dengan WSO2 API *Manager* pada tugas akhir ini.



Gambar 2.5 Skema Hubungan MyITS Single Sign-On dengan WSO2 API Manager Setelah Tugas Akhir Ini

Untuk melakukan pengecekan terhadap kevalidan dari *access token* dibutuhkan *introspection endpoint* yaitu *endpoint* yang akan mengecek kebenaran dari sebuah *access token*. Dalam melakukan *client management* dibutuhkan *client registration endpoint*. *Endpoint* ini digunakan untuk menambah, mengubah, menghapus dan mendapatkan informasi dari *client*. Untuk melakukan *request* terhadap *access token* dibutuhkan *token endpoint*. *Token endpoint* ini akan menghasilkan *access token* yang dibutuhkan oleh *client*.

Tabel 2.1 Ketersedian *Endpoint* yang Dibutuhkan pada MyITS *Single Sign-On*

Endpoint yang Dibutuhkan	Status pada MyITS *Single Sign-On*
Token Endpoint	Ada
Introspection Endpoint	Tidak Ada
Client Registration Endpoint	Tidak Ada

Pada Tabel 2.1 dapat dilihat ada beberapa *endpoint* yang belum tersedia pada MyITS *Single Sign-On. Endpoint* yang belum tersedia adalah *introspection endpoint* dan *client registration endpoint*. *Introspection endpoint* digunakan untuk melakukan

pengecekan terhadap keabsahan dari *access token* dan *client registration endpoint* digunakan untuk menambah, mengubah, menghapus dan mendapatkan informasi dari *client*.

[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN

Bab ini membahas tahap analisis permasalahan dan perancangan tugas akhir. Pada bagian awal dibahas mengenai analisis permasalahan yang ingin diselesaikan. Selanjutnya dibahas mengenai perancangan program untuk memberikan gambaran umum mengenai perubahan sistem yang dibuat.

3.1. Analisis

Proses pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On (authorization server)* dengan menerapkan sebuah konektor yang menghubungkan kedua *platform* tersebut. Konektor ini bertujuan untuk menjembatani alur komunikasi antara WSO2 API *Manager* dengan MyITS *Single Sign-On*.

3.1.1. WSO2 API *Manager*

WSO2 API *Manager* adalah *platform* yang digunakan untuk membuat, mengatur, memakai, dan memantau berbagai API [1]. Menggunakan *Service Oriented Architecture* yang telah terbukti untuk mengatasi berbagai tantangan manajemen API seperti penyediaan API, tata kelola API, keamanan API, dan pemantauan API. Pada WSO2 API *Manager* terdapat beberapa modul dalam implementasinya yang disebutkan pada Tabel 3.1.

Tabel 3.1 Modul Dalam WSO2 API *Manager*

Modul	Kegunaan
API *Publisher*	Membuat API baru dan mengelolanya
API *Store*	Mencari API dan berlangganan API
API *Gateway*	Mengamankan, melindungi, mengelola dan mengeskalasi panggilan API
API *Key Manager*	Membuat dan melakukan validasi *key* dan *access token*

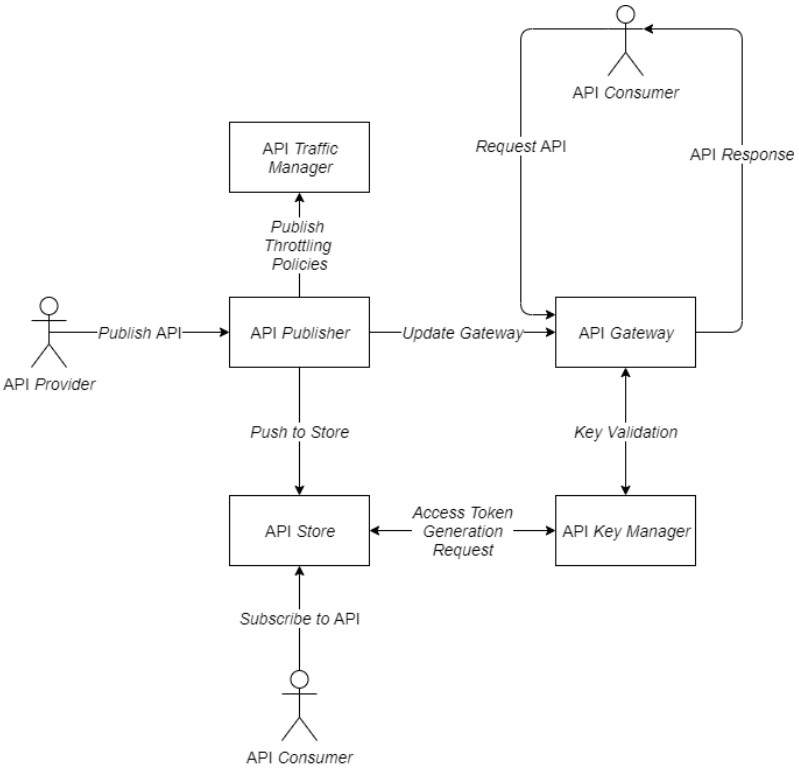
API *Traffic Manager*	Melakukan limitasi pada tingkat permintaan API

WSO2 API *Manager* yang digunakan adalah versi 2.6.0. Versi ini merupakan pengembangan dari versi 2.5.0 yang memiliki beberapa fitur baru dan pengembangan, yaitu:

a. WSO2 API *Manager Analytics* berdasar pada WSO2 *Stream Processor*. Mulai versi 2.6.0 dan seterusnya, WSO2 API *Manager Analytics* akan ditunjang oleh WSO2 *Stream Processor* 4.3.0 dan tidak lagi menggunakan WSO2 *Data Analytics Server*.
b. Dukungan dalam melakukan pengunduhan laporan *Microgateway analytics.*
c. Dukungan dalam pengaturan konfigurasi ambang batas peringatan melalui API *Publisher* dan API *Store.*

Pada Gambar 3.1 dijelaskan mengenai skema komunikasi antar modul pada WSO2 API *Manager*.

1. API *Publisher* berkomunikasi dengan API *Traffic Manager* untuk menerapkan kebijakan *throttling*.
2. API *Publisher* berkomunikasi dengan API *Gateway* untuk melakukan pembaharuan informasi API yang ada pada API *Gateway*.
3. API *Publisher* berkomunikasi dengan API *Store* untuk melakukan penambahan API yang telah dibuat oleh API *Provider*.
4. API *Gateway* berkomunikasi dengan API *Key Manager* untuk melakukan validasi *key* yang diberikan oleh API *Consumer*.
5. API *Store* berkomunikasi dengan API *Key Manager* untuk melakukan permintaan *access token.*
6. API *Consumer* dapat melakukan *subscribe* ke API pada API *Store* dan melakukan *request* API pada API *Gateway.*
7. API *Provider* dapat memasarkan API yang telah dibuat pada API *Publisher*.



Gambar 3.1 Skema Komunikasi Antar Modul Pada WSO2 API *Manager*

Pada tugas akhir ini akan mengganti API *Key Manager* yang ada di WSO2 API *Manager* dengan eksternal *Key Manager* yaitu MyITS *Single Sign-On*.

3.1.1.1. API *Publisher*

Pengembangan API biasanya dilakukan oleh seseorang yang memahami aspek teknis API, antarmuka, dokumentasi, versi dan lain-lain [6]. Sementara manajemen API biasanya dilakukan oleh seseorang yang memahami aspek bisnis API. Di sebagian besar

lingkungan bisnis, pengembangan API adalah tanggung jawab yang berbeda antara publikasi dan manajemen API.

WSO2 API *Manager* menyediakan antarmuka *Web* sederhana yang disebut WSO2 API *Publisher* untuk pengembangan dan manajemen API. Ini adalah GUI terstruktur yang dirancang untuk pembuatan API, mengembangkan, mendokumentasikan, mengeskalasi, melakukan perubahan versi API, penerbitan API, monetisasi, analisis statistik, dan promosi.

Tabel 3.2 menunjukkan aktivitas siklus umum pengembang/pengelola API.

Tabel 3.2 Siklus Umum Pengelola API

Tahapan	Aktivitas
Develop	• Mengembangkan API dan perubahan API • Melakukan *deployment* ke *Application Server*
Publish	• Mendaftarkan API • Mengatur *Service Level Agreement* • Mengatur persyaratan keamanan • Mengatur *Rate Limits/Throttling*
Manage	• Mengelola API *Lifecycle* • Mengelola API *Version* • Mengelola API *Policies* • Mengelola API *Keys*
Monitor	• Mengawasi perilaku API • Mengawasi konsumen API • Mengumpulkan kebutuhan konsumen

3.1.1.2. API *Store*

API *Store* menyediakan antarmuka kolaboratif untuk API *provider* untuk menyajikan dan mengiklankan API mereka dan bagi API *consumer* untuk mendaftar, menemukan, mengevaluasi, berlangganan dan menggunakan API yang diamankan, dilindungi, dan diautentikasi [6].

Tabel 3.3 menunjukkan kegiatan siklus umum dari API *Consumer*.

Tabel 3.3 Siklus Umum Konsumen API

Tahapan	Aktivitas
Find	• Melihat API paling sering dipakai, API baru, dan fitur API. • Mencari berdasarkan nama, *tag*, atau penyedia. • Menyimpan pencarian.
Explore	• Melihat peringkat dan komentar • Mengunduh bantuan dan dokumentasi • Mencoba secara *online* • Memberikan pertanyaan
Subscribe	• Mendaftarkan aplikasi • Mendapatkan *key* • Berlangganan API • Berlangganan ke API *changes*
Evaluate	• Menilai API • Berbagi Komentar

	• Melakukan permintaan fitur • Berpartisipasi pada forum

3.1.1.3. API *Gateway*

API *Gateway* merupakan komponen *backend* (*proxy* API) yang dikembangkan menggunakan WSO2 *Enterprise Service Bus*. API *Gateway* bertugas untuk mengamankan, melindungi, mengelola, dan mengeskalasi panggilan API. API *Gateway* ini mencegah permintaan API dan menerapkan kebijakan seperti pembatasan dan keamanan menggunakan *handlers*, serta mengelola statistik API [6]. Setelah validasi kebijakan, *Gateway* melewati panggilan layanan *web* ke *backend* yang sebenarnya. Jika panggilan layanan adalah permintaan *access token*, API *Gateway* meneruskannya langsung ke API *Key Manager*.

3.1.1.4. API *Traffic Manager*

API *Traffic Manager* membantu pengguna untuk mengatur lalu lintas API, menjadikan API dan aplikasi tersedia bagi konsumen di berbagai tingkat layanan dan mengamankan API dari serangan [6]. API *Traffic Manager* memiliki fitur *dynamic throttling engine* untuk memproses kebijakan pelambatan secara *real-time*, termasuk pembatasan tingkat permintaan API.

3.1.1.5. API *Key Manager*

API *Key Manager* memiliki tugas untuk mengelola semua *client*, keamanan, dan operasi terkait *access token*. API *Gateway* terhubung dengan API *Key Manager* untuk memeriksa validitas *access token*, *subscription*, dan pemanggilan API [6]. Ketika konsumen membuat *client* dan mengeluarkan *access token* ke *client* menggunakan API *Store*. API *Store* membuat panggilan ke API *Gateway*, yang kemudian menghubungkan ke *Key Manager* untuk membuat *client* dan mendapatkan *access token*. Demikian pula, untuk melakukan validasi *access token*, API *Gateway* memanggil *Key Manager*, yang melakukan validasi dan mengambil detail *access token* dari *database*.

Key Manager juga menyediakan API untuk menghasilkan *access token* OAuth yang dapat diakses melalui API *Gateway*. Semua *token* yang digunakan untuk validasi didasarkan pada protokol OAuth 2.0.0. Otorisasi API yang aman disediakan oleh standar OAuth 2.0 untuk *Key Manager*. API *Gateway* mendukung autentikasi API dengan OAuth 2.0, dan memungkinkan organisasi untuk memberlakukan batas penggunaan dan kebijakan pembatasan.

API *Key Manager* memisahkan operasi untuk membuat aplikasi OAuth dan melakukan validasi *access token* sehingga memungkinkan untuk digunakannya *authorization server* pihak ketiga untuk melakukan validasi.

Dalam lingkungan produksi, ada beberapa konfigurasi yang dapat digunakan, yaitu:

a. Konfigurasi dengan menggunakan WSO2 API *Manager* sebagai *Key Manager* di *server* yang berbeda.
b. Konfigurasi dengan menggunakan WSO2 *Identity Server* sebagai *Key Manager*.
c. Konfigurasi dengan menggunakan *authorization server* pihak ketiga sebagai *key validation* dan WSO2 API *Manager* untuk melakukan manajemen API.

3.1.1.6. *Users* dan *Roles*

WSO2 API *Manager* menawarkan 4 peran yang berlaku untuk sebagian besar perusahaan, yaitu:

1. *Admin*
   Penyedia manajemen API yang menyediakan dan mengelola API *Gateway*. *Admin* bertanggung jawab untuk membuat peran pengguna dalam sistem, menetapkan perannya, mengelola basis data, keamanan dan lain-lain.
2. *Creator*
   Creator merupakan orang yang memiliki peran teknis yang memahami aspek teknis API seperti antarmuka, dokumentasi, versi dan lain-lain. Menggunakan API *Publisher* untuk menyediakan API ke dalam API *Store*. *Creator* juga

menggunakan API *Store* untuk berkonsultasi dengan *rate* dan *feedback* yang diberikan oleh API *User*. *Creator* dapat menambahkan API ke API *Store* namun tidak dapat mengelola *lifecycle*-nya.

3. *Publisher*
   Seorang *publisher* mengelola satu set API di perusahaan atau unit bisnis dan mengontrol *lifecycle* API, langganan, dan aspek monetisasi. *Publisher* juga dapat melihat pola penggunaan untuk API dan memiliki akses ke semua statistik API.
4. *Subscriber*
   Subscriber menggunakan API *Store* untuk menemukan API, membaca dokumentasi dan forum, menilai/mengomentari API, berlangganan API, mendapatkan *access token* dan melakukan *request* API.

3.1.1.7. Konfigurasi *Authorization Server* Pihak Ketiga

Key Manager memisahkan *client* OAuth dan manajemen *access token* dari operasinya sehingga memungkinkan untuk menambahkan penyedia OAuth pihak ketiga untuk mengelola *client* dan *access token*. Saat bekerja dengan *key manager* eksternal, diperlukan *Extending Key Manager Interface* [2].

3.1.1.7.1. *Extending Key Manager Interface*

Dalam pelaksanaannya, beberapa komponen berbeda melakukan interaksi dengan *Key Manager* untuk mencapai tugas yang berbeda. Contohnya :

a. API *Gateway* berhubungan dengan API *Key Manager* untuk melakukan pengecekan terhadap *Access Token*. Ketika API *Gateway* menerima *request* dari *client*, API *Gateway* akan melakukan komunikasi dengan *Key Manager* dan mendapatkan validasi *access token*. *Key Manager* memeriksa apakah *access token* aktif, dan apakah *token* dapat digunakan untuk mengakses *resource* yang akan diakses. Jika *token* valid, *key manager* akan mengirimkan detail tambahan

tentang *token* kepada API *Gateway* dalam bentuk *response* [7].

b. API *Store* berhubungan dengan API *Key Manager* untuk melakukan pengelolaan aplikasi dan melakukan *request access token*. Setelah membuat aplikasi pada API *Store*, *user* akan menekan tombol *generate* untuk melakukan pendaftaran aplikasi. Pada titik ini, API *Store* melakukan komunikasi kepada *Key Manager* untuk membuat *client* OAuth dan mendapatkan *consumer key/consumer secret* dan *access token* aplikasi [7].

Oleh karena itu, *Key Manager Interface* bertugas sebagai jembatan antara *authorization server* dengan WSO2 API *Manager*.

Dalam melakukan implementasi *Key Manager Interface*, yang mana merupakan *Java Extension Point* pada WSO2 API *Manager*, ada beberapa *method* yang digunakan *Key Manager Interface* untuk melakukan operasi yang disebutkan pada Tabel 3.4 [7].

Tabel 3.4 *Method* pada *Key Manager Interface*

Method	Kegunaan
createApplication	Membuat aplikasi OAuth baru pada *authorization server*
updateApplication	Melakukan pembaharuan pada aplikasi OAuth
retrieveApplication	Mendapatkan informasi aplikasi OAuth
deleteApplication	Menghapus aplikasi OAuth
getNewApplicationAccessToken	Mendapatkan *access token* baru pada saat pertama kali

getTokenMetaData	Mendapatkan detail dari *access token*
getKeyManagerConfiguration	Mendapatkan implementasi *Key Manager* pada *file api-manager.xml*
buildAccessTokenRequestFromJSON	Membentuk *access token request* menjadi JSON
mapOAuthApplication	Menambahkan aplikasi yang telah terdaftar sebelumnya pada *Key Manager*
buildAccessTokenRequestFromOAuthApp	Membuat *Access Token Request* menggunakan *OAuth Application Information*.
registerNewResource	Menambahkan *Resource* baru
getResourceByApiId	Mendapatkan *Resource* berdasarkan pada ID API
updateRegisteredResource	Melakukan pembaharuan pada *resource*
deleteRegisteredResourceByAPIId	Menghapus *Resource* berdasarkan pada ID API
deleteMappedApplication	Menghapus data aplikasi yang dipetakan
getActiveTokensByConsumerKey	Mendapatkan semua *access token* yang aktif berdasarkan *consumer key*
getAccessTokenByConsumerKey	Mendapatkan detail *access token* yang

	ditampilkan pada API *Store*.

3.1.1.8. *Object* yang Dibutuhkan pada Implementasi *Key Manager Interface*

Ada beberapa *object* yang dibutuhkan untuk melakukan implementasi *Key Manager Interface*, yaitu:

a. *AccessTokenInfo*
b. *AccessTokenRequest*
c. *OauthAppRequest*
d. *OauthApplicationInfo*

3.1.1.8.1. *AccessTokenInfo*

AccessTokenInfo memiliki beberapa atribut dan *method* yang dijelaskan pada Tabel 3.5 dan Tabel 3.6.

Tabel 3.5 Atribut pada *Object AccessTokenInfo*

Nama atribut	Tipe Data
isTokenValid	*Boolean*
isApplicationToken	*Boolean*
consumerKey	*String*
consumerSecret	*String*
scope	*Array Of String*
tokenState	*String*
accessToken	*String*
issuedTime	*Long*
validityPeriod	*Long*
errorcode	*Integer*
endUserName	*String*
parameters	*HashMap<String,Object>*

Tabel 3.6 *Method* pada *Object AccessTokenInfo*

Nama *Method*	*Return Value Data Type*
setScopes	*void*
getErrorcode	*Int*
setErrorcode	*Void*

setScope	*Void*
getTokenState	*String*
setTokenState	*Void*
getAccessToken	*String*
setAccessToken	*Void*
getIssuedTime	*Long*
getValidityPeriod	*Long*
getConsumerKey	*String*
getConsumerSecret	*String*
setConsumerKey	*Void*
setConsumerSecret	*Void*
setIssuedTime	*Void*
setValidityPeriod	*Void*
addParameter	*Void*
getParameter	*Object*
isTokenValid	*Boolean*
setTokenValid	*Void*
isApplicationToken	*Boolean*
setApplicationToken	*Void*
getJSONString	*String*
getEndUserName	*String*
setEndUserName	*Void*

3.1.1.8.2. *AccessTokenRequest*

AccessTokenRequest memiliki beberapa atribut dan *method* yang dijelaskan pada Tabel 3.7 dan Tabel 3.8.

Tabel 3.7 Atribut pada *Object AccessTokenRequest*

Nama atribut	Tipe Data
clientId	*String*
clientSecret	*String*
grantTyoe	*String*
scope	*Array Of String*
callbackURI	*String*
resourceOwnerUsername	*String*
resourceOwnerPassword	*String*
refreshToken	*String*

tenantDomain	*String*
validityPeriod	*Long*
tokenToRevoke	*String*
requestParameters	*HashMap<String,Object>*

Tabel 3.8 *Method* pada *Object AccessTokenRequest*

Nama *Method*	*Return Value Data Type*
getTokenToRevoke	*String*
setTokenToRevoke	*Void*
getClientId	*String*
setClientId	*Void*
getClientSecret	*String*
setClientSecret	*Void*
getGrantType	*String*
setGrantType	*Void*
getScope	*Array Of String*
setScope	*Void*
getCallBackURI	*String*
setCallBackURI	*Void*
getValidityPeriod	*Long*
setValidityPeriod	*Void*
getResourceOwnerUsername	*String*
setResourceOwnerUsername	*Void*
getResourceOwnerPassword	*String*
setResourceOwnerPassword	*Void*
getRefreshToken	*String*
setRefreshToken	*Void*
getTenantDomain	*String*
setTenantDomaint	*Void*
addRequestParam	*Void*
getRequestParam	*Object*

3.1.1.8.3. *OauthAppRequest*

OauthAppRequest memiliki beberapa atribut dan *method* yang dijelaskan pada Tabel 3.9 dan Tabel 3.10.

Tabel 3.9 Atribut pada *Object OAuthAppRequest*

Nama atribut	Tipe Data
mappingId	*String*
oAuthApplicationInfo	*OauthApplicationInfo*

Tabel 3.10 *Method* pada *Object OAuthAppRequest*

Nama *Method*	*Return Value Data Type*
getMappingId	*String*
setMappingId	*Void*
getOAuthApplicationInfo	*OauthApplicationInfo*
setOAuthApplicationInfo	*Void*

3.1.1.8.4. *OauthApplicationInfo*

OauthApplicationInfo memiliki beberapa atribut dan *method* yang dijelaskan pada Tabel 3.11 dan Tabel 3.12.

Tabel 3.11 Atribut pada *Object OauthApplicationInfo*

Nama atribut	Tipe Data
clientId	*String*
clientName	*String*
callBackURL	*String*
clientSecret	*String*
parameters	*Map<String,Object>*
isSaasApplication	*Boolean*
appOwner	*String*
jsonString	*String*

Tabel 3.12 *Method* pada *Object OAuthApplicationInfo*

Nama *Method*	*Return Value Data Type*
setJsonString	*Void*
getClientId	*String*
setClientId	*Void*
getClientSecret	*String*

setClientSecret	*Void*
setClientname	*Void*
setCallBackURL	*Void*
addParameter	*Void*
getParameter	*Object*
getJsonString	*String*
getClientName	*String*
getCallBackURL	*String*
putAll	*Void*
removeParameter	*Void*
getIsSassApplication	*Boolean*
setIsSassApplication	*Void*
setAppOwner	*Void*
getAppOwner	*String*

3.1.1.9. Pengaturan Konfigurasi *Authorization Server* Pihak Ketiga pada WSO2 API *Manager*

Dalam melakukan implementasi *authorization server* pihak ketiga ada beberapa langkah yang diperlukan, yaitu:

1. Melakukan implementasi *key manager interface* sesuai pada subbab 3.1.1.7.1. Melakukan *build* pada kelas tersebut dan menyimpannya dalam *file* berekstensi JAR.
2. Letakkan *file* JAR yang sudah dibuat pada direktori *<API-Manager-Home>/repository/components/lib.*
3. Hilangkan tanda komentar elemen *<APIKeyManager>* yang terletak pada *file* dengan direktori *<API-Manager-Home>/repository/conf/api-manager.xml* dan ubah beberapa *value* pada elemen *<APIKeyManager>* sesuai dengan Tabel 3.13.

Tabel 3.13 Elemen pada *APIKeyManager*

Elemen	*Value*
KeyManagerClientImpl	Nama kelas yang sudah dibuat pada langkah pertama

Configuration	Berisi elemen-elemen tambahan yang berisi nilai yang dibutuhkan saat melakukan *Extending Key Manager Interface*

4. Ubah *value* dari elemen <*http*> pada *file _TokenAPI_.xml* yang terletak pada direktori <*API-Manager-Home*>*/repository/deployment/server/sypnase-configs/default/api/_TokenAPI_.xml*. *Value* diubah menjadi *Token Endpoint* seusai pada *authorization server* pihak ketiga.
5. Mengaktifkan *mapExistingAuthApps* pada *file site.json* yang terletak pada direktori <*API-Manager-Home*>*/repository/deployment/server/jaggeryapps/store/site/conf/site.json*

3.1.2. MyITS *Single Sign-On*

Platform MyITS *Single Sign-On* adalah *authorization server* yang memanfaatkan protokol OAuth2 [6]. MyITS *Single Sign-On* menggunakan *library* OAuth Bshaffer dalam pengimplementasian protokol OAuth2. MyITS *Single Sign-On* menyimpan data akun pengguna, data *client* yang dapat diberi otorisasi, dan memberikan otorisasi dengan mengeluarkan *access token* dan dapat melakukan verifikasi *access token*.

OAuth2 adalah sebuah protokol yang memungkinkan aplikasi pihak ketiga untuk mendapatkan akses terbatas ke layanan HTTP, baik atas nama pemilik *resource* dengan mengatur interaksi persetujuan antara pemilik *resource* dan layanan HTTP, atau dengan memungkinkan aplikasi pihak ketiga untuk mendapatkan akses atas namanya sendiri[8]. Dalam penggunaannya, OAuth2 menggunakan beberapa jenis *endpoint* yaitu *Token Endpoint, Introspection Endpoint,* dan *Client Registration Endpoint*.

- *Token Endpoint*
  Token Endpoint digunakan oleh *client* untuk mendapatkan *access token* dengan mengirimkan *authorization grant.*
- *Introspection Endpoint*
  Resource Server membutuhkan *Introspection Endpoint* untuk melakukan validasi terhadap *access token* yang diberikan oleh *client. Introspection Endpoint* akan memberikan *response* tentang data *access token* jika *access token* yang diberikan dapat divalidasi.
- *Client Registration Endpoint*
  Endpoint ini bertujuan untuk melakukan pengelolaan terhadap *client* yang terdaftar/didaftarkan pada OAuth2.

3.1.2.1. *Token Endpoint*

Token Endpoint digunakan oleh *client* untuk mendapatkan *access token* dengan melampirkan *authorization grant* atau *refresh token* [8].

Saat melakukan *access token request*, disertakan juga *grant type* yang digunakan. *Grant type* yang dapat digunakan yaitu *authorization code, implicit, user credentials, client credentials* dan *refresh token.*

3.1.2.1.1. *Authorization Code*

Grant type ini digunakan untuk memperoleh *access token* dan *refresh token* dan dioptimalkan untuk *confidential client* [8].

3.1.2.1.1.1. Access Token Request

Client membuat *request* kepada *Token Endpoint* dengan mengirimkan beberapa parameter dengan menggunakan format *application/x-www-form-urlencoded*. Parameter yang dikirimkan dan diletakkan pada *body* yaitu:

- *grant_type*
  REQUIRED. Nilainya harus diisi dengan "*authorization_code*".
- *code*
  REQUIRED. Nilainya yaitu *authorization code* yang didapatkan dari *authorization server*.

- *redirect_uri*
  REQUIRED. Nilai yang dikirimkan harus sama dengan nilai yang dikirimkan saat *authorization request*.
- *client_id*
  REQUIRED, jika *client* tidak terautentikasi dengan *authorization server*.

3.1.2.1.1.2. Access Token Response

Jika *access token request* valid dan terotorisasi, *authorization server* akan mengeluarkan *access token* dan *optional refresh token*. *Authorization server* akan memberikan data berupa:

- *access_token*
- *token_type*
- *expires_in*
- *refresh_token*
- *example_parameter*

3.1.2.1.2. *Implicit Grant*

Grant type ini digunakan untuk memperoleh *access* dan dioptimalkan untuk *public client* yang dikenal mengoperasikan URI *redirection* tertentu [8].

3.1.2.1.2.1. Authorization Request

Client membuat *request* ke *authorization endpoint* dengan mengirimkan beberapa parameter dengan menggunakan format *application/x-www-form-urlencoded*. Parameter yang dikirimkan dan diletakkan pada *body* yaitu:

- *response_type*
  REQUIRED. Nilainya harus diisi dengan "*token*".
- *client_id*
  REQUIRED. Nilainya yaitu *client identifier*.
- *redirect_uri*
  OPTIONAL.
- *scope*
  REQUIRED, merupakan *scope* dari *access request*.

- *State*
  RECOMMENDED. Nilai buram yang digunakan oleh *client* untuk mempertahankan keadaan antara *request* dan *callback*.

3.1.2.1.2.2. Access Token Response

Jika *access token request* valid dan terotorisasi, *authorization server* akan mengeluarkan *access token*. *Authorization server* akan memberikan data berupa:

- *access_token*
- *token_type*
- *expires_in*
- *scope*
- *state*

3.1.2.1.3. *User Password Credential*

Grant type ini cocok dalam kasus di mana *resource owner* memiliki hubungan saling percaya dengan *client*, seperti sistem operasi perangkat atau yang sangat istimewa [8].

3.1.2.1.3.1. Access Token Request

Client membuat *request* kepada *Token Endpoint* dengan mengirimkan beberapa parameter dengan menggunakan format *application/x-www-form-urlencoded*. Parameter yang dikirimkan dan diletakkan pada *body* yaitu:

- *grant_type*
  REQUIRED. Nilainya harus diisi dengan "*password.*
- *username*
  REQUIRED. Username dari *resource owner*.
- *password*
  REQUIRED. Password dari *resource owner.*
- *scope*
  OPTIONAL, merupakan *scope* dari *access request*.

3.1.2.1.3.2. Access Token Response

Jika *access token request* valid dan terotorisasi, *authorization server* akan mengeluarkan *access token* dan *optional refresh token*. *Authorization server* akan memberikan data berupa:

- *access_token*
- *token_type*
- *expires_in*
- *refresh_token*
- *example_parameter*

3.1.2.1.4. *Client Credential*

Client dapat meminta *access token* hanya menggunakan *client credential*-nya (atau cara autentikasi lainnya yang didukung) ketika *client* meminta akses ke *resource* yang dilindungi di bawah kontrolnya, atau orang-orang dari pemilik *resource* lain yang sebelumnya telah diatur dengan *authorization server* [8].

3.1.2.1.4.1. Access Token Request

Client membuat *request* kepada *Token Endpoint* dengan mengirimkan beberapa parameter dengan menggunakan format *application/x-www-form-urlencoded*. Parameter yang dikirimkan dan diletakkan pada *body* yaitu:

- *grant_type*
  REQUIRED. Nilainya harus diisi dengan "*client_credentials"*.
- *scope*
  OPTIONAL, merupakan *scope* dari *access request*.

3.1.2.1.4.2. Access Token Response

Jika *access token request* valid dan terotorisasi, *authorization server* akan mengeluarkan *access token*. *Authorization server* akan memberikan data berupa:

- *access_token*
- *token_type*

- *expires_in*
- *example_parameter*

3.1.2.1.5. *Refreshing an Access Token*

Jika *authorization server* menerbitkan *refresh token* kepada *client, Client dapat* membuat *request* kepada *Token Endpoint* dengan mengirimkan beberapa parameter dengan menggunakan format *application/x-www-form-urlencoded*. Parameter yang dikirimkan dan diletakkan pada *body* yaitu:

- *grant_type*
  REQUIRED. Nilainya harus diisi dengan "*refresh_token"*.
- *refresh_token*
  REQUIRED. Refresh token yang diterbitkan untuk *client*.
- *scope*
  OPTIONAL, merupakan *scope* dari *access request.*

3.1.2.2. *Introspection Endpoint*

Introspection endpoint adalah *endpoint* OAuth 2.0 yang mengambil parameter yang mewakili *token* OAuth 2.0 dan mengembalikan dokumen JSON yang mewakili informasi metadata *token*, termasuk apakah *token* ini sedang aktif. Definisi *token* aktif tergantung pada *authorization server*, tetapi biasanya *token* yang telah dikeluarkan oleh *authorization server* ini, tidak kedaluwarsa, belum dicabut, dan valid untuk digunakan pada sumber daya yang dilindungi dan membuat panggilan introspeksi [9]. Pada MyITS *Single Sign-On* belum diimplementasikan *introspection endpoint*.

3.1.2.2.1. *Introspection Request*

Protected resource memanggil *Introspection Endpoint* menggunakan HTTP POST, *request* dikirimkan dengan parameter dengan format "*application/x-www-form-urlencode"*. Parameter yang dikirimkan berupa:

- *token*
  REQUIRED. Access token yang akan diintrospeksi.

- *token_type_hint*
  OPTIONAL. Petunjuk tentang jenis *token* yang diajukan untuk introspeksi.

3.1.2.2.2. *Introspection Response*

Server mengembalikan respons dengan format JSON *object* dengan data-data yang dikembalikan sebagai berikut:

- *active*
  REQUIRED. Status keaktifan *token*.
- *scope*
  OPTIONAL. *Scopes* yang berasosiasi dengan *access token.*
- *client_id*
  OPTIONAL. Client Identifier.
- *username*
  OPTIONAL. Username dari *resource owner.*
- *exp*
  OPTIONAL. Waktu kedaluwarsa dari *Access Token.*
- *iat*
  OPTIONAL. Waktu penerbitan dari *Access Token.*
- *nbf*
  OPTIONAL. Menunjukan bahwa *access token* tidak pernah digunakan sebelumnya.
- *sub*
  OPTIONAL. Merepresentasikan *resource owner* yang melakukan otorisasi pada *access token.*
- *aud*
  OPTIONAL. Merepresentasikan audiens yang dituju oleh *access token* ini.
- *iss*
  OPTIONAL. Merepresentasikan penerbit *access token.*
- *jti*
  OPTIONAL. Identifier dari *access token.*

3.1.2.3. *Client Registration Endpoint*

Client Registration Endpoint adalah *endpoint* OAuth 2.0 yang dirancang untuk memungkinkan *client* terdaftar di *server* otorisasi. *Client Registration Endpoint* harus menerima pesan HTTP POST dengan parameter permintaan yang disandikan di badan entitas menggunakan format "*application/json*" [10]. Pada MyITS *Single Sign-On* belum diimplementasikan *client registration endpoint*.

3.1.2.3.1. *Client Registration Request*

Operasi ini mendaftarkan *client* dengan *authorization server*. *Authorization server* memberikan *client* ini pengidentifikasi *client* yang unik, secara opsional menetapkan *client secret*, dan mengaitkan metadata yang disediakan dalam *request* dengan pengidentifikasi *client* yang dikeluarkan. *Request* mencakup parameter metadata *client* yang ditentukan untuk *client* selama pendaftaran. *Authorization server* mungkin memberikan nilai *default* untuk setiap item yang dihilangkan dalam metadata *client*.

Untuk mendaftar, *client* atau *developer* mengirim POST HTTP ke *Client Registration Endpoint* dengan jenis konten "*application/json*". HTTP *Entity Payload* adalah JSON yang terdiri dari objek JSON dengan semua nilai metadata *client* yang diminta sebagai anggota tingkat atas objek JSON itu.

3.1.2.3.2. *Client Read Request*

Untuk membaca konfigurasi *client* saat ini di *authorization server*, *client* membuat permintaan HTTP GET ke *client configuration endpoint*, melakukan autentikasi dengan [11].

3.1.2.3.3. *Client Update Request*

Untuk memperbarui *client registration* yang sebelumnya terdaftar dengan *authorization server*, *client* membuat permintaan HTTP PUT ke *client configuration endpoint* dengan jenis konten "*application/json*". *Payload entity* HTTP adalah dokumen JSON yang terdiri dari objek JSON dan semua parameter sebagai anggota tingkat atas objek JSON itu. Permintaan ini dikonfirmasi oleh *registration access token* yang dikeluarkan untuk *client* [11].

3.1.2.3.4. *Client Delete Request*

Untuk menghapus *client* pada *authorization* di *authorization server*, *client* membuat *request* HTTP DELETE ke *client configuration endpoint*. Permintaan ini dikonfirmasi oleh *registration access token* yang dikeluarkan untuk *client* [11].

3.1.3. OAuth2 *Server Library* (PHP) Bshaffer

OAuth2 *Server Library* Bshaffer merupakan *library open source* OAuth2 *Server* dan OpenID *Connect* yang dapat implementasikan pada aplikasi berbasis PHP. *Library* ini tersedia pada Git *https://github.com/bshaffer/oauth2-server-php*. Dalam melakukan implementasi *library* ini dibutuhkan versi PHP 5.3.9 atau yang lebih baru [12].

Library Bshaffer ini memiliki beberapa konsep utama, yaitu:

1. *Grant Types*
   Grant Type memungkinkan pengguna untuk mengekspos berbagai cara bagi *client* untuk menerima *access token.*
2. *Controller*
   OAuth *Server* memiliki 3 *endpoint*, yang mana setiap *endpoint* akan diteruskan kepada *controller*.
3. *Storage Object*
   Library ini menggunakan *storages interfaces* untuk memungkinkan interaksi dengan beberapa lapisan data.

3.1.3.1. *Grant Types*

Ada banyak *Grant Types* yang didukung oleh spesifikasi OAuth2. *Library* ini juga memungkinkan untuk melakukan penambahan kustomisasi *grant types*. Beberapa *grant types* yang didukung yaitu:

1. *Authorization Code*
2. *Password*
3. *Client Credentials*
4. *Refresh Token*
5. *Implicit*
6. JWT *Bearer*

3.1.3.2. *Controller*

Sebagian besar OAuth2 API akan memiliki *endpoint* untuk *Authorize Request*, *Token Request* dan *Resource Request*. OAuth2 *Server* memiliki beberapa *method* untuk menangani setiap *request*, yaitu:

1. *Authorize Controller*
2. *Resource Controller*
3. *Token Controller*

3.1.3.2.1. *Authorize Controller*

Controller ini digunakan untuk *authorize endpoint*, yang mana mengharuskan pengguna untuk memberikan autentikasi dan mengembalikan kepada *client* dengan *authorization code* atau *access token*. Ada beberapa *method* pada *authorize controller* ini, yaitu:

- *handleAuthorizeRequest*
  Menerima obyek *request* untuk *authorize request* dan mengembalikan obyek *response* dengan respons yang sesuai.
- *validateAuthorizeRequest*
  Menerima obyek *request* dan mengembalikan nilai *false* jika *request* yang diterima tidak valid. Jika *request* yang diterima valid, akan dikembalikan detail dari *client*.

3.1.3.2.2. *Resource Controller*

Digunakan untuk setiap *resource request* yang membutuhkan autentikasi OAuth2. *Controller* ini akan melakukan validasi pada *request* yang masuk dan mempersilahkan aplikasi untuk menyajikan *resource* yang terproteksi. Ada beberapa *method* yang ada pada *controller* ini, yaitu:

- *verifyResourceRequest*
  Menerima obyek *request* untuk *resource request*, melakukan pengecekan apakah *token* ada atau tidak dan mengembalikan nilai *boolean* jika *request* valid.
- *getAccessTokenData*

Menerima obyek request sebagai argumen dan mengembalikan data token jika berlaku, atau *null* jika *token* tidak valid.

3.1.3.2.3. *Token Controller*

Digunakan untuk *token endpoint*, yang menggunakan *grant type* sesuai pada konfigurasi untuk mengembalikan *access token* kepada *client*. Ada beberapa *method* pada *controller* ini, yaitu:

- *grantAccessToken*
  Menerima obyek *request* untuk *token request*, dan mengembalikan *token* jika *request* valid.
- *handleTokenRequest*
  Menerima obyek *request* untuk *token request*, dan mengembalikan obyek *response* dengan respons yang sesuai.

3.1.3.3. *Storages Object*

Library ini mendukung *adapter* untuk beberapa mesin penyimpanan (*storage engines*). Beberapa diantaranya adalah PDO (untuk MySQL, SQLite, PostgreSQL, dan lain-lain), MongoDB, Redis, dan Cassandra.

3.1.4. Integrasi WSO2 API *Manager* dengan MyITS *Single Sign-On*

Dalam membangun konektor yang menghubungkan WSO2 API *Manager* dengan *authorization server* pihak ketiga membutuhkan beberapa fitur yang dipenuhi oleh *authorization server* eksternal (Dalam hal ini MyITS *Single Sign-On)*. Fitur-fitur yang dibutuhkan dapat dilihat pada Tabel 3.14.

Tabel 3.14 Fitur yang Dibutuhkan WSO2 API *Manager*

Fitur yang dibutuhkan	*Method* terkait
Menambahkan *client* pada *authorization server*.	*createApplication*
Mengubah *client* pada *authorization server*.	*updateApplication*

Mendapatkan data *client* dari *authorization server*.	*retrieveApplication*
Menghapus *client* pada *authorization server*.	*deleteApplication*
Mendapatkan *access token* baru	*getNewApplicationAccessToken*
Mendapatkan detail *access token*	*getTokenMetaData*.

Melihat kebutuhan fitur yang ada pada WSO2 API *Manager*. Dapat dipenuhi dengan *endpoint* yang harus disediakan oleh MyITS *Single Sign-On*. Untuk pemetaan fitur dan *endpoint* seperti yang ditunjukkan pada Tabel 3.15.

Tabel 3.15 Pemetaan Fitur yang Dibutuhkan WSO2 API *Manager* dengan *Endpoint* yang Disediakan MyITS *Single Sign-On*

Fitur yang dibutuhkan WSO2 API *Manager*	*Endpoint* yang disediakan MyITS *Single Sign-On*
Menambahkan *client* pada *authorization server*.	Belum tersedia pada MyITS *Single Sign-On*.
Mengubah *client* pada *authorization server*.	Belum tersedia pada MyITS *Single Sign-On*.
Mendapatkan data *client* dari *authorization server*.	Belum tersedia pada MyITS *Single Sign-On*.
Menghapus *client* pada *authorization server*.	Belum tersedia pada MyITS *Single Sign-On*.
Mendapatkan *access token* baru	*Token Endpoint*
Mendapatkan detail *access token*	Belum tersedia pada MyITS *Single Sign-On*.

Ada beberapa *endpoint* yang belum tersedia pada MyITS *Single Sign-On*, yaitu *client registration endpoint* dan *introspection endpoint*. Kedua *endpoint* ini akan diimplementasikan pada MyITS *Single Sign-On* untuk dapat dilakukan pengintegrasian WSO2 API *Manager* dan MyITS *Single Sign-On*.

3.1.5. *Introspection Endpoint* pada MyITS *Single Sign-On*

MyITS *Single Sign-On* belum memiliki *introspection endpoint*. *Introspection endpoint* sendiri dibutuhkan oleh WSO2 API *Manager* untuk melakukan pengecekan terhadap *access token*. Pengimplementasian *introspection endpoint* pada MyITS *Single Sign-On* dapat dilakukan dengan memperhatikan aspek-aspek pada subbab 3.1.2.2.

3.1.6. *Client Registration Endpoint* pada MyITS *Single Sign-On*

Dalam melakukan integrasi antara WSO2 API *Manager* dengan MyITS *Single Sign-On*. *Client registration endpoint* dibutuhkan oleh WSO2 API *Manager* untuk melakukan penambahan, perubahan, penghapusan dan pengambilan informasi *client* yang tersimpan pada MyITS *Single Sign-On*. Pada saat ini, MyITS *Single Sign-On* belum memiliki *client registration endpoint*. Untuk melakukan penerapan *client registration endpoint*, dapat dilakukan dengan memperhatikan aspek-aspek pada subbab 3.1.2.3.

Namun ada satu hal yang tidak bisa diterapkan sesuai subbab 3.1.2.3, yaitu penerapan *authorization* dengan menggunakan *registration access token*. Hal ini dikarenakan pada WSO2 API *Manager* terdapat beberapa *method* pada *key manager interface* yang berkomunikasi dengan *client registration endpoint*, tidak memiliki parameter berupa *registration access token*, yaitu:

- *Method createApplication*
  Method ini menerima parameter berupa obyek *OauthAppRequest* yang tidak memiliki atribut berupa *registration access token* seperti yang dijelaskan pada subbab 3.1.1.8.3.
- *Method updateApplication*
  Method ini menerima parameter berupa obyek *OauthAppRequest* yang tidak memiliki atribut berupa *registration access token* seperti yang dijelaskan pada subbab 3.1.1.8.3.
- *Method retrieveApplication*

Method ini menerima parameter berupa *client id*. Tidak ada *registration access token* yang diterima oleh *method retrieveApplication*.

- *Method deleteApplication*
  Method ini menerima parameter berupa *client id*. Tidak ada *registration access token* yang diterima oleh *method deleteApplication*.

Permasalahan mengenai *authorization* dengan menggunakan *registration access token* dapat ditangani dengan mengubah *authorization* dari *registration access token* menjadi API *Key*.

3.2. Perancangan

Subbab ini membahas tentang perancangan yang akan diterapkan pada sistem WSO2 API *Manager* agar dapat menggunakan *Key Manager* pihak ketiga.

Tugas Akhir ini hanya berfokus pada implementasi *introspection endpoint* pada MyITS *Single Sign-On*, implementasi *client registration endpoint* pada MyITS *Single Sign-On*, penerapan *library* yang akan menggantikan tugas *Key Manager* bawaan agar dapat menggunakan *Key Manager* pihak ketiga dan pengaturan konfigurasi yang ada pada WSO2 API *Manager*.

3.2.1. Perancangan Implementasi *Introspection Endpoint* pada MyITS *Single Sign-On*.

WSO2 API *Manager* membutuhkan *introspection endpoint* untuk melakukan pengecekan terhadap *access token* yang diterima dari *client*. WSO2 API *Manager* akan mengirimkan data berupa *access token* kepada MyITS *Single Sign-On* dan MyITS *Single Sign-On* akan mengembalikan metadata dari *access token*. *Library* OAuth2 Bshaffer telah menyediakan *method* untuk melakukan introspeksi terhadap *access token*, yaitu *method getAccessTokenData* pada *Resource Controller*. Contoh *instropection request* dan *introspection response* dapat dilihat pada Kode Sumber 3.1 dan Kode Sumber 3.2.

```
POST /introspection HTTP/1.1
   Host: server.example.com
   Authorization: Bearer S1AV23hskdJ
```

Kode Sumber 3.1 Contoh *Introspection Request*

```
HTTP/1.1 200 OK
    Content-Type: application/json
    {
        “access_token”: “S1AV23hskdJ”,
        “client_id”: “clientId”,
        “user_id”: “userId”,
        “expires”: 1578245235,
        “scope”: “test”
    }
```

Kode Sumber 3.2 Contoh *Introspection Response*

3.2.2. Perancangan Implementasi *Client Registration Endpoint* pada MyITS *Single Sign-On*

WSO2 API *Manager* membutuhkan *client registration endpoint* untuk melakukan penambahan, perubahan, penghapusan dan pengambilan data *client* yang tersimpan pada MyITS *Single Sign-On.* Terdapat beberapa *request* pada *client registration endpoint*, yaitu *request* tambah *client*, *request* hapus *client*, *request* ubah *client* dan *request* ambil *client*.

Request tambah *client* digunakan untuk melakukan penambahan *client* pada MyITS *Single Sign-On*. Data yang dikirimkan berformat JSON dan berisi data-data *client* yang akan ditambahkan. *Response* yang diterima yaitu data-data *client* yang telah ditambahkan. Contoh *request* dan *response* dapat dilihat pada Kode Sumber 3.3 dan Kode Sumber 3.4.

```
POST /client-registration HTTP/1.1
   Host: server.example.com
   Content-Type: application/json
   {
       “client_name”: “aplikasiku”,
       “redirect_uri”: “www.example.com”
       “grant_type”: “client_credentials”
   }
```

Kode Sumber 3.3 Contoh *Request* Tambah *Client*

```
HTTP/1.1 200 OK
    Content-Type: application/json
    {
        "client_id": "SFpocsCaSgfG",
        "client_secret":"asdWDasdFDO",
        "redirect_uri": "example.com",
        "grant_types": "client_credentials ",
        "scope": “test”,
        "user_id": “userId”
    }
```

Kode Sumber 3.4 Contoh *Response* Tambah *Client*

Request ubah *client* digunakan untuk melakukan perubahan *client* pada MyITS *Single Sign-On*. Parameter yang dikirimkan berupa *client_id* yang diletakkan pada URL. Data yang dikirimkan berformat JSON dan berisi data-data *client* yang akan diubah. Dan *response* yang didapatkan berupa data *client*. Contoh *request* dan *response* dapat dilihat pada Kode Sumber 3.5 dan Kode Sumber 3.6.

```
PUT /client-registration/SFpocsCaSgfG HTTP/1.1
   Host: server.example.com
   x-api-key : asCsasdXApEI
   Content-Type: application/json
   {
       “client_name”: “aplikasiku”,
       “redirect_uri”: “www.example.com”
       “grant_type”: “password”
   }
```

Kode Sumber 3.5 Contoh *Request* Ubah *Client*

```
HTTP/1.1 200 OK
    Content-Type: application/json
    {
        "client_id": "SFpocsCaSgfG",
        "client_secret":"asdWDasdFDO",
        "redirect_uri": "example.com",
        "grant_types": "password",
        "scope": “test”,
        "user_id": “userId”
    }
```

Kode Sumber 3.6 Contoh *Response* Ubah *Client*

Request hapus *client* digunakan untuk melakukan penghapusan *client* pada MyITS *Single Sign-On*. Contoh *request* dan *response* dapat dilihat pada Kode Sumber 3.8 dan Kode Sumber 3.8.

```
HTTP/1.1 204 OK
```

Kode Sumber 3.7 Contoh *Response* Hapus *Client*

```
DELETE /client-registration/SFpocsCaSgfG HTTP/1.1
   Host: server.example.com
   x-api-key : asCsasdXApEI
```

Kode Sumber 3.8 Contoh *Request* Hapus *Client*

Request ambil *client* digunakan untuk mendapatkan detail *client* pada MyITS *Single Sign-On*. Contoh *request* dan *response* dapat dilihat pada Kode Sumber 3.9 dan Kode Sumber 3.10.

```
GET /client-registration/SFpocsCaSgfG HTTP/1.1
   Host: server.example.com
   x-api-key : asCsasdXApEI
```

Kode Sumber 3.9 Contoh *Request* Ambil *Client*

```
HTTP/1.1 200 OK
    Content-Type: application/json
    {
        "client_id": "SFpocsCaSgfG",
        "client_secret":"asdWDasdFDO",
        "redirect_uri": "example.com",
        "grant_types": "password",
        "scope": “test”,
        "user_id": “userId”
    }
```

Kode Sumber 3.10 Contoh *Response* Ambil *Client*

3.2.3. Perancangan *Method Library* yang Mengimplementasi *Key Manager Interface* pada WSO2 API *Manager*

Pada *key manager interface* disediakan beberapa *method* yang dapat diimplementasi untuk menyediakan kebutuhan yang dibutuhkan WSO2 API *Manager*. *Method* ini nantinya akan memiliki fungsi masing-masing.

3.2.3.1. *Method loadConfiguration*

Method ini bertujuan untuk pemuatan data yang ada pada *file api-manager.xml* yang disediakan oleh WSO2 API *Manager*.

3.2.3.2. *Method createApplication*

Method ini digunakan untuk melakukan registrasi *client* kepada *authorization server* (MyITS *Single Sign-On*). *Method* ini memiliki parameter *OauthAppRequest* yang berisi informasi mengenai *Client* yang akan dibuat. *Method* ini akan mengembalikan data berupa *OauthApplicationInfo* yang berisi data *client* yang tersimpan di MyITS *Single Sign-On*. Pada *method* ini dibutuhkan *client registration endpoint*. Spesifikasi *method createApplication* disebutkan pada Tabel 3.16.

Tabel 3.16 Spesifikasi *Method createApplication*

Tujuan	Menambahkan *client* pada *authorization server*.
Tipe Data *Return Value*	*OAuthAppliactionInfo*
Parameter	*OauthAppRequest*
Data yang diperlukan pada konfigurasi	*Client Registration Endpoint*.

3.2.3.3. *Method updateApplication*

Method ini digunakan untuk melakukan perubahan *client* yang tersimpan pada *authorization server* (MyITS *Single Sign-On*). *Method* ini memiliki parameter *OauthAppRequest* yang berisi informasi mengenai *Client* yang akan dibuat. Dalam melakukan *request* disertakan *authorization basic* menggunakan *client id* dan *client secret*. *Method* ini akan mengembalikan data berupa *OauthApplicationInfo* yang berisi data *client* yang tersimpan di MyITS *Single Sign-On*. Pada *method* ini dibutuhkan *client registration endpoint*. Spesifikasi *method updateApplication* disebutkan pada Tabel 3.17.

Tabel 3.17 Spesifikasi *Method updateApplication*

Tujuan	Melakukan perubahan *client* pada *authorization server*.
Tipe Data *Return Value*	*OauthAppliactionInfo*
Parameter	*OauthAppRequest*
Data yang diperlukan pada konfigurasi	*Client Registration Endpoint*. *Client Id, Client Secret*

3.2.3.4. *Method deleteApplication*

Method ini digunakan untuk melakukan penghapusan *client* pada *authorization server* (MyITS *Single Sign-On*). *Method* ini memiliki parameter *client_id* yang merupakan *id client* pada MyITS *Single Sign-On*. Dalam melakukan *request* disertakan *authorization basic* menggunakan *client id* dan *client secret.* Pada *method* ini dibutuhkan *client registration endpoint.* Spesifikasi *method deleteApplication* disebutkan pada Tabel 3.18.

Tabel 3.18 Spesifikasi *Method deleteApplication*

Tujuan	Menghapus *client* pada *authorization server*.
Tipe Data *Return Value*	*Void*
Parameter	*String – Client_Id*
Data yang diperlukan pada konfigurasi	*Client Registration Endpoint, Client Id, Client Secret*

3.2.3.5. *Method retrieveApplication*

Method ini digunakan untuk mendapatkan informasi mengenai *client* yang dimiliki *authorization server* (MyITS *Single Sign-On*). *Method* ini memiliki parameter *client_id* yang merupakan *id client* pada MyITS *Single Sign-On*. Dalam melakukan *request* disertakan *authorization basic* menggunakan *client id* dan *client secret. Method* ini akan mengembalikan data berupa *OauthApplicationInfo* yang berisi data *client* yang tersimpan di MyITS *Single Sign-On.* Pada *method* ini dibutuhkan

client registration endpoint. Spesifikasi *method retrieveApplication* disebutkan pada Tabel 3.19.

Tabel 3.19 Spesifikasi *Method retrieveApplication*

Tujuan	Mendapatkan detail *client* pada *authorization server*.
Tipe Data *Return Value*	*OauthApplicationInfo*
Parameter	*String – Client_Id*
Data yang diperlukan pada konfigurasi	*Client Registration Endpoint, Client Id, Client Secret*

3.2.3.6. *Method getNewApplicationAccessToken*

Method ini digunakan untuk mendapatkan *access token* baru untuk sebuah *client*. *Method* ini memiliki parameter berupa *AccessTokenRequest* yang berisi *client id* dan *client secret*. *Method* ini akan mengembalikan data berupa *AccessTokenInfo* yang berisi informasi *access token* yang didapatkan. Pada *method* ini dibutuhkan *Token Endpoint.* Spesifikasi *method getNewApplicationAccessToken* disebutkan pada Tabel 3.20.

Tabel 3.20 Spesifikasi *Method getNewApplicationAccessToken*

Tujuan	Mendapatkan *access token* baru.
Tipe Data *Return Value*	*AccessTokenInfo*
Parameter	*AccessTokenRequest*
Data yang diperlukan pada konfigurasi	*Token Endpoint*

3.2.3.7. *Method getTokenMetaData*

Method ini digunakan untuk mendapatkan informasi mengenai detail dari *access token*. Parameter yang diberikan yaitu berupa *access token*. *Method* ini mengembalikan data berupa *AccessTokenInfo* dari *access token* yang diberikan. *Method* ini membutuhkan *Introspection Endpoint.* Spesifikasi *method getTokenMetaData* disebutkan pada Tabel 3.21.

Tabel 3.21 Spesifikasi *Method getTokenMetaData*

Tujuan	Mendapatkan detail dari *access token.*
Tipe Data *Return Value*	*AccessTokenInfo*
Parameter	*String – access token*
Data yang diperlukan pada konfigurasi	*Introspection Endpoint*

3.2.4. Konfigurasi pada WSO2 API *Manager*

Untuk menerapkan *key manager* pihak ketiga dibutuhkan data berupa *Token Endpoint, Client Registration Endpoint, Introspection Endpoint, client id* dan *client secret.* Data ini diletakkan pada *file api-manager.xml*. Selanjutnya dilakukan pergantian URL yang ada pada *file _TokenAPI_.xml* dengan *Token Endpoint* pada MyITS *Single Sign-On.* Serta mengaktifkan *mapExistingAuthApps* yang ada pada *file site.json.* Data yang dibutuhkan dapat dilihat di Tabel 3.22.

Tabel 3.22 Data yang Dibutuhkan pada Konfigurasi WSO2 API *Manager*

Data yang dibutuhkan	Keterangan
Token Endpoint	*Token Endpoint* yang dimiliki oleh MyITS *Single Sign-On*
Introspection Endpoint	*Introspection Endpoint* yang dimiliki oleh MyITS *Single Sign-On*
Client Registration Endpoint	*Client Registration Endpoint* yang dimiliki oleh MyITS *Single Sign-On*
Client Id	*Client Id* WSO2 API *Manager* yang ada pada MyITS *Single Sign-On*

Client Secret	*Client Secret* WSO2 API *Manager* yang ada pada MyITS *Single Sign-On*

BAB IV
IMPLEMENTASI SISTEM

Bab ini membahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijelaskan pada bab sebelumnya. Implementasi yang dijelaskan adalah bagaimana menerapkan Perancangan *Method Library* yang mengimplementasi *Key Manager Interface* pada WSO2 API *Manager*. Bahasa pemrograman yang digunakan untuk implementasi adalah bahasa pemrograman Java.

4.1. Lingkungan Pengembangan Sistem

Lingkungan pengembangan sistem yang digunakan untuk mengembangkan Tugas Akhir ini dilakukan pada lingkungan dan kakas sebagai berikut.

1. Prosesor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz
2. Memori (RAM) 8192MB
3. Sistem operasi menggunakan Windows 10 Enterprise 64 bit.
4. WSO2 API *Manager* versi 2.6.0
5. IDE menggunakan Eclipse Java 2019-09
6. Java JDK versi 1.8.0_201
7. PHP versi 7.3.11
8. XAMPP PhpMyadmin dan SQL *Server* untuk mengelola basis data

4.2. Penerapan *Introspection Endpoint* Pada MyITS *Single Sign-On*

Penerapan *introspection endpoint* pada sistem ini menambahkan kelas *introspection controller* pada MyITS *Single Sign-On* yang memanggil *method getAccessTokenData* kelas *resource controller* pada *library* OAuth2 Bshaffer. Kelas *controller* berisikan fungsi-fungsi yang digunakan untuk menangkap *request* dan mengirimkan *response* dengan memproses terlebih dahulu di dalam kelas *controller* itu sendiri.

Pada kelas *controller* ini terdapat satu *method*. *Pseudocode method* ini dapat dilihat pada Kode Sumber 4.1.

```
1.  method handle()
2.      GET introspectionRequest from request object
3.      CREATE new response
4.      CALL getAccessTokenData(introspectionRequest, r
    esponse) from library
5.      RETURN response
6.  end
```

Kode Sumber 4.1 *Pseudocode* Kelas *Introspection Controller*

Pada *method handle*, akan dibentuk *introspection request* yang diambil dari *request object*. Lalu akan dipanggil *method getAccessTokenData* dari *library* OAuth2 Bshaffer pada *resource controller*. Pemanggilan *method* ini akan mengembalikan respons berupa metadata dari *access token*.

4.3. Penerapan *Client Registration Endpoint* pada MyITS *Single Sign-On*

Penerapan *client registration endpoint* pada sistem ini dengan menambahkan kelas *client controller* pada *library* OAuth2 Bshaffer dan menambahkan *method deleteClient* pada kelas *Client Repository*. *Pseudocode* kelas *deleteClient* pada *Client Repository* dapat dilihat pada Kode Sumber 4.2.

```
1.  method deleteClient(client_id)
2.     query = ‘DELETE from client_table where client_i
    d = ‘ + client_id
3.     return query->execute
4.  end
```

Kode Sumber 4.2 *Pseudocode Method deleteClient*

Pada *method deleteClient* memiliki parameter berupa *client_id*. Lalu akan dibentuk *query* untuk menghapus *client*

dengan *client_id* yang dikirimkan pada *method* ini. *Query* tersebut akan dieksekusi untuk melakukan penghapusan pada *client*.

Pada kelas *client controller* OAuth2 Bshaffer terdapat beberapa *method*, yaitu *getClientInfo, deleteClient* dan *setClient*. *Method getClientInfo* digunakan untuk mengambil informasi dari sebuah *client* dan mengembalikan informasi tersebut kepada *consumer*. *Method deleteClient* digunakan untuk menghapus *client*. *Method setClient* digunakan untuk menambah atau merubah data *client*. *Pseudocode method-method* tersebut dapat dilihat pada Kode Sumber 4.3, Kode Sumber 4.4 dan Kode Sumber 4.5.

```
1.  method getClientInfo(request,response,clientId)
2.     if( client = CALL getClientDetails(clientId,requ
    est,response) from Repository)
3.        response.code = 200
4.        response.parameter = client
5.     end
6.  end
```

Kode Sumber 4.3 *Pseudocode Method getClientInfo* pada Kelas *Client Controller*

Pada *method getClientInfo* akan dilakukan pengecekan terhadap *client_id* yang dikirimkan dan memanggil *method getClientDetails* dari *repository*. *Method getClientDetails* akan mengembalikan data berupa data lengkap dari *client*.

```
1.  method deleteClient(request,response,clientId)
2.     if( client = CALL deleteClient(clientId,request,
    response) FROM Repository)
3.        response.code = 200
4.        response.parameter = client
5.     end
6.  end
```

Kode Sumber 4.4 *Pseudocode Method deleteClient* pada Kelas *Client Controller*

Pada *method deleteClient* akan dilakukan pengecekan terhadap *client_id* yang dikirimkan dan memanggil *method deleteClient* dari *repository*. *Method getClientDetails* akan mengembalikan status keberhasilan dari penghapusan *client*.

```
1.  method setClient(request,response,clientId)
2.     if(clientId = null)
3.        clientId = new Uuid
4.        clientSecret = new Uuid
5.        redirect_uri = request['redirect_uri']
6.        grant_types = request['grant_types']
7.        if(client = CALL setClientDetails(clientId, c
    lientSecret, redirect_uri, grant_types) from Reposi
    tory)
8.           response.code = 201
9.           response.parameter = client
10.       end
11.    else
12.       if(client = CALL getClientDetails(clientId) f
    rom Repository)
13.          clientSecret = client.clientSecret
14.          redirect_uri = request['redirect_uri']
15.          grant_types = request['grant_types']
16.          if(clientUpdate = CALL setClientDetails(cl
    ientId, clientSecret, redirect_uri, grant_types) fr
    om
17.  Repository)
18.          response.code = 201
19.          response.parameter = clientUpdate
20.       end
21.    end
22. end
```

Kode Sumber 4.5 *Pseudocode Method setClient* pada Kelas *Client Controller*

Pada *method setClient* akan dilakukan pengecekan terhadap *client_id* yang dikirimkan apakah *client_id* yang dikirimkan ada atau tidak. Jika tidak ada, maka akan dibentuk *client id* dan *client secret* baru. Lalu akan memanggil *method setClientDetails* dari

repository untuk menambahkan *client* pada basis data. Lalu akan memanggil *method getClientDetails* pada *repository* untuk mengembalikan nilai berupa data dari *client*.

Selanjutnya akan diterapkan *client controller* pada MyITS *Single Sign-On*. Kelas *controller* ini berisikan fungsi-fungsi yang digunakan untuk menangkap *request* dan mengirimkan *response* dengan memproses terlebih dahulu di dalam kelas *controller* itu sendiri. Pada kelas *controller* ini terdapat beberapa *method*, yaitu *createClient, updateClient, getClient* dan *deleteClient*. *Pseudocode method-method* tersebut dapat dilihat pada Kode Sumber 4.6, Kode Sumber 4.7, Kode Sumber 4.8 dan Kode Sumber 4.9.

```
1.  method createClient()
2.      GET clientRequest from request object
3.      CREATE new response
4.      clientId = null
5.      CALL setClient(clientRequest, response, clientI
    d) from library
6.      RETURN response
7.  end
```

Kode Sumber 4.6 *Pseudocode Method createClient* pada *Client Controller* MyITS *Single Sign-On*

Pada *method createClient* akan dibentuk *client request* yang diambil dari *request object*. Lalu akan dipanggil *method setClient* dari *library* OAuth2 Bshaffer pada *client controller*. Pemanggilan *method* ini akan mengembalikan respons berupa data dari *client* yang telah dibuat.

```
1.  method updateClient()
2.      GET clientRequest from request object
3.      CREATE new response
4.      clientId = request.client_id
5.      CALL setClient(clientRequest, response, clientI
    d) from library
6.      RETURN response
```

```
7.  end
```

Kode Sumber 4.7 *Pseudocode Method updateClient* pada Kelas *Client Controller* MyITS *Single Sign-On*

Pada *method updateClient* akan dibentuk *client request* yang diambil dari *request object*. Lalu akan dipanggil *method setClient* dari *library* OAuth2 Bshaffer pada *client controller*. Pemanggilan *method* ini akan mengembalikan respons berupa data dari *client*.

```
1.  method deleteClient()
2.      GET clientRequest from request object
3.      CREATE new response
4.      clientId = request.client_id
5.      CALL deleteClient(clientRequest, response, clie
    ntId) from library
6.      RETURN response
7.  end
```

Kode Sumber 4.8 *Pseudocode Method deleteClient* pada Kelas *Client Controller* MyITS *Single Sign-On*

Pada *method deleteClient* akan dibentuk *client request* yang diambil dari *request object*. Lalu akan dipanggil *method deleteClient* dari *library* OAuth2 Bshaffer pada *client controller*. Pemanggilan *method* ini akan mengembalikan respons tanpa data.

```
1.  method getClient()
2.      GET clientRequest from request object
3.      CREATE new response
4.      clientId = request.client_id
5.      CALL getClientDetails(clientRequest, response,
    clientId) from library
6.      RETURN response
7.  end
```

Kode Sumber 4.9 *Pseudocode Method getClient* pada Kelas *Client Controller* MyITS *Single Sign-On*

Pada *method getClient* akan dibentuk *client request* yang diambil dari *request object*. Lalu akan dipanggil *method getClientDetails* dari *library* OAuth2 Bshaffer pada *client controller*. Pemanggilan *method* ini akan mengembalikan respons data dari *client*.

4.4. Penerapan *Method Library* yang Mengimplementasi *Key Manager Interface*

Penerapan pola perancangan pada sistem ini yaitu mengganti *key manager* bawaan dengan *key manager* yang telah dikustomisasi sehingga dapat terhubung dengan MyITS *Single Sign-On*. Penerapan ini dilakukan dengan melakukan *override* beberapa *method* dari *Key Manager Interface* yang disediakan oleh WSO2 API *Manager* dan melakukan konfigurasi pada WSO2 API *Manager* agar dapat terhubung dengan MyITS *Single Sign-On.*

4.4.1. Penerapan pada *Method loadConfiguration*

Pada *method* ini dilakukan pemuatan *configuration* untuk *key manager* yang disediakan oleh WSO2 API *Manager*. Kode Sumber 4.10 berikut adalah *pseudocode* untuk *method loadConfiguration.*

```
1.  configuration = null
2.  method loadConfiguration(KeyManagerConfiguration)
3.     this.configuration = KeyManagerConfiguration
4.  end
```

Kode Sumber 4.10 *Pseudocode Method loadConfiguration*

Pada *method loadConfiguration* akan diambil data dari *file api-manager.xml*. Data ini akan diletakan pada *global variable* agar dapat diakses oleh *method* lain.

4.4.2. Penerapan pada *Method createApplication*

Pada *method* ini akan dilakukan penambahan aplikasi pada WSO2 API *Manager* dan disimpan juga pada MyITS *Single Sign-*

On. Langkah-langkah yang dilakukan dalam penerapan *method createApplication* adalah:

1. Pemanggilan *Client Registration Endpoint* yang ada pada *configuration.*
2. Membentuk JSON *Object* yang berisi *client nama* dan *redirect uri* yang akan diletakkan pada *body parameter*.
3. Melakukan *request* dengan *method post* terhadap *Client Registration Endpoint* dengan parameter yang sudah dibuat.
4. Pengecekan terhadap kode respons. Jika berhasil, akan dibentuk obyek *OauthApplicationInfo* yang berisi *client id* dan *client secret.*

Kode Sumber 4.11 berikut adalah *pseudocode* untuk *method createApplication.*

```
1.  method createApplication(OAuthAppRequest)
2.     oAuthApplicationInfo = OAuthAppRequest.getOAuthA
    pplicationInfo()
3.     registrationEndpoint = configuration.getParamete
    r("RegistrationEndpoint")
4.
5.     bodyArray = [
6.        client_name : oAuthApplicationInfo.getClientN
    ame(),
7.        redirect_uri : oAuthApplicationInfo.getCallBa
    ckURL()
8.     ]
9.
10.    form_data = [
11.       body : jsonEncode(bodyArray)
12.    ]
13.
14.    request = HTTP.post(registrationEndpoint,form_da
    ta)
15.
16.    if request.code = 201
17.       info = new OAuthApplicationInfo
18.       info.setClientId(request.data.client_id)
```

```
19.       info.setClientSecret(request.data.client_secr
    et)
20.       return info
21.    else
22.       return false
23.    end
24. end
```

Kode Sumber 4.11 *Pseudocode Method createApplication*

4.4.3. Penerapan pada *Method updateApplication*

Pada *method* ini akan dilakukan perubahan pada aplikasi pada WSO2 API *Manager* dan disimpan juga pada MyITS *Single Sign-On*. Langkah-langkah yang dilakukan dalam penerapan *method updateApplication* adalah:

1. Pemanggilan *Client Registration Endpoint* yang ada pada *configuration.*
2. Penambahan *client id* pada URL untuk mengubah aplikasi tertentu. Sehingga URL-nya menjadi *client_registration_endpoint/client_id.*
3. Membentuk JSON *Object* yang berisi *client name* dan *redirect uri* yang akan diletakkan pada *body parameter*.
4. Melakukan *request* dengan *method put* terhadap *Client Registration Endpoint* dengan parameter yang sudah dibuat.
5. Pengecekan terhadap kode respons. Jika berhasil, akan dibentuk obyek *OauthApplicationInfo* dan akan mengembalikan obyek tersebut.

Kode Sumber 4.12 berikut adalah *pseudocode* untuk *method updateApplication.*

```
1.  method updateApplication(OAuthAppRequest)
2.     oAuthApplicationInfo = OAuthAppRequest.getOAuthA
    pplicationInfo()
3.     registrationEndpoint = configuration.getParamete
    r("RegistrationEndpoint")
4.     wso2ClientId = configuration.getParameter("Clien
    tId")
```

```
5.     wso2ClientSecret = configuration.getParameter("C
   lientSecret")
6.     updateEndpoint = registrationEndpoint + "/" +
   oAuthApplicationInfo.getClientId()
7.     bodyArray = [
8.        client_name : oAuthApplicationInfo.getClientN
   ame(),
9.        redirect_uri : oAuthApplicationInfo.getCallBa
   ckURL()
10.    ]
11.
12.    headerArray = [
13.       x-api-
   key : base64encode(wso2ClientId+":"+wso2ClientSecre
   t)
14.    ]
15.
16.    form_data = [
17.       body : jsonEncode(bodyArray),
18.       header : headerArray
19.    ]
20.
21.    request = HTTP.put(updateEndpoint,form_data)
22.
23.    if request.code = 201
24.       info = new OAuthApplicationInfo
25.       info.setClientId(request.data.client_id)
26.       info.setClientSecret(request.data.client_secr
   et)
27.       return info
28.    else
29.       return false
30.    end
31. end
```

Kode Sumber 4.12 *Pseudocode Method updateApplication*

4.4.4. Penerapan pada *Method deleteApplication*

Pada *method* ini akan dilakukan penghapusan aplikasi pada WSO2 API *Manager* dan dihapus juga pada MyITS *Single Sign-*

On. Langkah-langkah yang dilakukan dalam penerapan *method deleteApplication* adalah:

1. Pemanggilan *Client Registration Endpoint* yang ada pada *configuration.*
2. Penambahan *client id* pada URL untuk menghapus aplikasi tertentu. Sehingga URL-nya menjadi *client_registration_endpoint/client_id.*
3. Melakukan *request* terhadap *Client Registration Endpoint* dengan *method delete*.
4. Pengecekan terhadap kode respons. Jika berhasil, akan mengembalikan nilai *true*.

Kode Sumber 4.13 berikut adalah *pseudocode* untuk *method deleteApplication.*

```
1.  method deleteApplication(clientId)
2.     registrationEndpoint = configuration.getParamete
   r("RegistrationEndpoint")
3.     wso2ClientId = configuration.getParameter("Clien
   tId")
4.     wso2ClientSecret = configuration.getParameter("C
   lientSecret")
5.
6.     deleteUrl = registrationEndpoint + "/" + clientI
   d
7.
8.     headerArray = [
9.        x-api-
   key : base64encode(wso2ClientId+":"+wso2ClientSecre
   t)
10.    ]
11.
12.    form_data = [
13.       header : headerArray
14.    ]
15.
16.    request = HTTP.delete(deleteUrl,form_data)
17.
```

```
18.    if request.code = 200
19.       return true
20.    else
21.       return false
22.    end
23. end
```

Kode Sumber 4.13 *Pseudocode Method deleteApplication*

4.4.5. Penerapan pada *Method retrieveApplication*

Pada *method* ini akan dilakukan *request* mengenai informasi aplikasi yang dilakukan kepada MyITS *Single Sign-On.* Langkah-langkah yang dilakukan dalam penerapan *method retrieveApplication* adalah:

1. Pemanggilan *Client Registration Endpoint* yang sudah diatur dalam *configuration.*
2. Penambahan *client id* pada URL untuk mendapatkan informasi aplikasi tertentu. Sehingga URL-nya menjadi *client_registration_endpoint/client_id.*
3. Melakukan *request* terhadap *Client Registration Endpoint* registrasi dengan *method get*.
4. Pengecekan terhadap kode respons. Jika berhasil, akan dibentuk obyek *OauthApplicationInfo* dan akan mengembalikan obyek tersebut.

Kode Sumber 4.14 berikut adalah *pseudocode* untuk *method deleteApplication.*

```
1.  method retrieveApplication(clientId)
2.     registrationEndpoint = configuration.getParamete
    r("RegistrationEndpoint")
3.     wso2ClientId = configuration.getParameter("Clien
    tId")
4.     wso2ClientSecret = configuration.getParameter("C
    lientSecret")
5.
6.     getUrl = registrationEndpoint + "/" + clientId
```

```
7.     headerArray = [
8.        x-api-
   key : base64encode(wso2ClientId+":"+wso2ClientSecre
   t)
9.     ]
10.
11.    form_data = [
12.       header : headerArray
13.    ]
14.    request = HTTP.post(getUrl,form_data)
15.
16.    if request.code = 200
17.       info = new OAuthApplicationInfo
18.       info.setClientId(request.data.client_id)
19.       info.setClientSecret(request.data.client_secr
   et)
20.       return info
21.    else
22.       return false
23.    end
24. end
```

Kode Sumber 4.14 *Pseudocode Method retrieveApplication*

4.4.6. Penerapan pada *Method getNewApplicationAccessToken*

Pada *method* ini akan dilakukan *request access token* untuk sebuah *client*/aplikasi. Langkah-langkah yang dilakukan dalam penerapan *method getNewApplicationAccessToken* adalah:

1. Pemanggilan *Token Endpoint* yang sudah diatur dalam *configuration.*
2. Pembentukan parameter berupa *grant type*, *client id*, dan *client secret* yang diletakkan di *body request*.
3. Melakukan *request* terhadap *Token Endpoint* dengan *method post*.
4. Pengecekan terhadap kode respons. Jika berhasil, akan dibentuk obyek *AccessTokenInfo* yang berisi mengenai informasi *access token* dan dikembalikan.

Kode Sumber 4.15 berikut adalah *pseudocode* untuk *method getNewApplicationAccessToken.*

```
method getNewApplicationAccessToken(AccessTokenRequest)
    tokenEndpoint = configuration.getParameter("TokenEndpoint")
    clientId = AccessTokenRequest.getClientId()
    clientSecret = AccessTokenRequest.getClientSecret()

    bodyArray = [
       grant_type : "client_credentials",
       client_id : clientId,
       client_secret : clientSecret
    ]

    form_data = [
       body : bodyArray
    ]

    request = HTTP.post(tokenEndpoint,form_data)

    if request.code = 200
       info = new AccessTokenInfo
       info.setAccessToken(request.data.access_token)
       info.setValidityPeriod(request.data.expires_time)
       info.setTokenValid(true)
       info.setScope(request.data.scopes)
       info.setConsumerKey(clientId)
       return info
    else
       return false
    end
end
```

Kode Sumber 4.15 *Pseudocode Method getNewApplicationAccessToken*

4.4.7. Penerapan pada *Method getTokenMetaData*

Pada *method* ini akan dilakukan *request* untuk melakukan introspeksi mengenai *access token*. Langkah-langkah yang dilakukan dalam penerapan *method getTokenMetaData* adalah:

1. Pemanggilan *Introspection Endpoint* yang sudah diatur dalam *configuration.*
2. Pembentukan parameter berupa *authorization* yang berisi *access token* yang akan diintrospeksi dan diletakkan pada *header*. Contoh, "*Authorization* : *Bearer access_token"*.
3. Melakukan *request* terhadap *Introspection Endpoint* dengan *method get*.
4. Pengecekan terhadap kode respons. Jika berhasil, akan dibentuk obyek *AccessTokenInfo* yang berisi mengenai informasi *access token* dan dikembalikan.

Kode Sumber 4.16 berikut adalah *pseudocode* untuk *method getTokenMetaData*.

```
1.  method getTokenMetaData(accessToken)
2.     introspectionEndpoint = configuration.getParamet
    er("IntrospectionEndpoint")
3.
4.     headerArray = [
5.        authorization : "Bearer " + accessToken
6.     ]
7.
8.     form_data = [
9.        header : headerArray
10.    ]
11.
12.    request = HTTP.get(introspectionEndpoint,form_da
    ta)
13.
14.    if request.code = 200
15.       info = new AccessTokenInfo
16.       info.setValidityPeriod(request.data.expires_t
    ime - currentTime())
17.       info.setTokenValid(true)
```

```
18.         info.setIssuedTime(currentTime())
19.         info.setConsumerKey(clientId)
20.         return info
21.     else
22.         return false
23.     end
24. end
```

Kode Sumber 4.16 *Pseudocode Method getTokenMetaData*

4.4.8. Penerapan Konfigurasi pada WSO2 API *Manager*

Sebelum melakukan penerapan ini, dilakukan terlebih dahulu *export* berupa JAR hasil dari implementasi *key manager interface* yang telah dibuat. Lalu letakan *file* tersebut pada direktori "*WSO2 API Manager/2.6.0/repository/components/lib*". Pada penerapan ini dilakukan konfigurasi pada beberapa *file* di WSO2 API *Manager*, yaitu:

1. *File api-manager.xml*
   Pada *file* ini ditambahkan *tag* dengan nama *<APIKeyManager>* untuk menambahkan *key manager* pihak ketiga yang terletak pada *api-manager.xml*. Ada beberapa data yang ditambahkan di dalam *tag* tersebut untuk dapat dipanggil pada *library*, diantaranya:
   a. Nama *Class* dari hasil implementasi *key manager interface.*
   b. *ClientRegistrationEndpoint*
   c. *IntrospectionEndpoint*
   d. *TokenEndpoint*
   e. *Client Id* dari WSO2 API *Manager* yang tersimpan pada MyITS *Single Sign-On.*
   f. *Client Secret* dari WSO2 API *Manager* yang tersimpan pada MyITS *Single Sign-On.*

Nilai dari data yang dibutuhkan pada *file api-manager.xml* dapat dilihat pada Tabel 4.1 dan implementasi dapat dilihat pada Kode Sumber 4.17

Tabel 4.1 *Key* dan *Value* yang Ditambahkan pada *api-manager.xml*

Nama *Tag*	Nilai
Nama *Class*	id.ac.its.sso.OAuthClient
ClientRegistrationEndpoint	https://my.its.ac.id/client-registration
IntrospectionEndpoint	https://my.its.ac.id/introspect
TokenEndpoint	https://my.its.ac.id/token
ClientId	wso2apimanager
ClientSecret	wso2apimanager

```
1.  <APIKeyManager>
2.   <KeyManagerClientImpl>id.ac.its.sso.OAuthClient</K
    eyManagerClientImpl>
3.   <Configuration>
4.      <ClientRegistrationEndpoint>https://my.its.ac.id
    /client-registration</ClientRegistrationEndpoint>
5.      <IntrospectionEndpoint>https://my.its.ac.id/intr
    ospect</IntrospectionEndpoint>
6.      <TokenEnpoint>https://my.its.ac.id/token</TokenE
    ndpoint>
7.      <ClientId>wso2apimanager</ClientId>
8.      <ClientSecret>wso2apimanager</ClientSecret>
9.   </Configuration>
10. </APIKeyManager>
```

Kode Sumber 4.17 Konfigurasi pada *File api-manager.xml*

2. *File _TokenAPI_.xml*
   Pada *file* ini dilakukan perubahan pada *endpoint*, *endpoint* yang digunakan adalah *Token Endpoint* pada MyITS *Single Sign-On*. Implementasi dapat dilihat pada Kode Sumber 4.18.

```
1.  <http uri-template="https://my.its.ac.id/token">
```

Kode Sumber 4.18 Perubahan pada *File _TokenAPI_.xml*

3. *File site.json*
   Pada *file* ini dilakukan perubahan pada *key* *mapExistingAuthApps*. Nilai dari *key* tersebut diubah dari *false* menjadi *true*. Hal ini akan memungkin *user* untuk menambah *client* yang sudah ada pada MyITS *Single Sign-On*. Implementasi dapat dilihat pada Kode Sumber 4.19.

```
1.  {
2.    "mapExistingAuthApps" : true,
3.  }
```

Kode Sumber 4.19 Perubahan pada *File site.json*

BAB V
PENGUJIAN DAN EVALUASI

5.1. Lingkungan Pengujian

Pengujian dilakukan dengan menggunakan beberapa lingkungan pengujian yaitu dua *web server* dan satu klien. Lingkungan pengujian dapat dilihat pada Tabel 5.1.

Tabel 5.1 Lingkungan Pengujian Sistem

WSO2 API *Manager (Server)*	
Sistem Operasi	Windows 10 *Enterprise*
Prosesor	Intel I7-6700HQ @2.60GHz
RAM	8 GB
WSO2 API *Manager*	WSO2 API *Manager* versi 2.6.0
Java	JDK versi 1.8.0_201
Basis Data	SQL Server 2016 versi 13.00.5492
Web Server	Java HotSpot(TM) 64-Bit Server VM 25.201-b09
WSO2 API *Manager*	WSO2 API *Manager* versi 2.6.0
MyITS *Single Sign-On (Server)*	
Sistem Operasi	Windows 10 *Enterprise*
Prosesor	Intel I7-6700HQ @2.60GHz
RAM	8 GB
PHP	PHP versi 7.3.11
Basis Data	SQL Server 2016 versi 13.00.5492
Web Server	Apache versi 2.4.41
Komputer Klien *(Client)*	
Sistem Operasi	Windows 10 *Enterprise*

Prosesor	Intel I7-6700HQ @2.60GHz
RAM	8 GB
Browser	Google Chrome versi 79.0.3945.88
CURL	CURL versi 7.55.1

5.2. Skenario Pengujian

Pada bagian ini akan dibahas mengenai proses uji coba yang digunakan. Pengujian dilakukan dengan metode *black box* untuk menguji masing-masing fungsionalitas yang sudah dirancang pada sistem. Metode *black box* merupakan metode pengujian perangkat lunak yang memeriksa fungsionalitas dari suatu perangkat lunak tanpa memandang struktur internalnya.

5.2.1. Pengujian Terhadap Fungsionalitas Sistem

Pengujian fungsionalitas sistem dilakukan dengan menguji kasus-kasus uji tiap kasus penggunaan. Berikut adalah kasus uji dari fungsionalitas sistem.

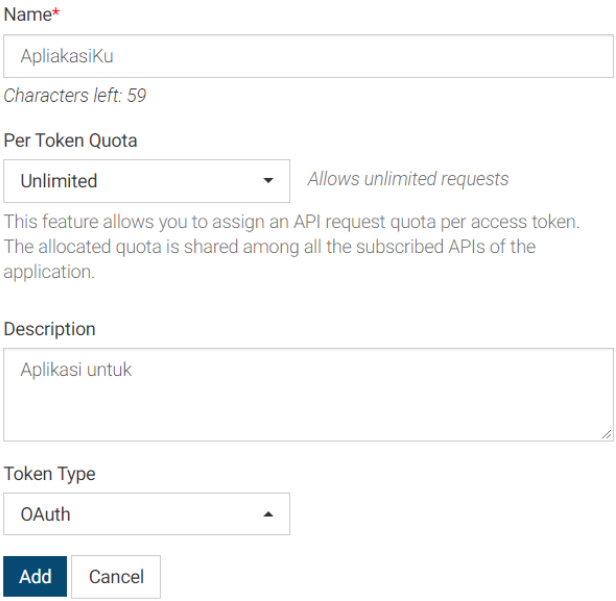
5.2.1.1. Kasus Uji Pendaftaran *Client.*

Pada kasus ini akan diuji dengan melakukan pendaftaran *client* pada WSO2 API *Manager*. Rincian kasus uji ditunjukkan pada Tabel 5.2.

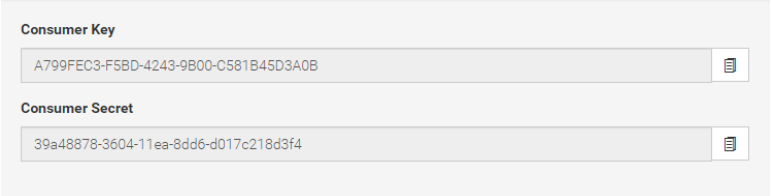
Tabel 5.2 Kasus Uji Pendaftaran *Client*

ID	**TC-001**
Kasus Penggunaan	Pendaftaran *Client.*
Nama	Pengujian pendaftaran *client* pada WSO2 API *Manager.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendaftarkan *client* pada MyITS *Single Sign-On*.
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store.*

Langkah Pengujian	1. Aktor mengakses halaman *add application*. 2. Aktor mengisi data *client* yang akan dibuat. 3. Aktor menekan tombol "*add*". 4. Aktor membuka *tab Production*. 5. Aktor menekan tombol "*Generate Keys*".
Hasil yang diharapkan	Aktor mendapatkan *client id* dan *client secret* dari *client* yang baru saja dibuat.
Kondisi Akhir	Aktor mendapatkan *client id* dan *client secret* dari *client* yang baru saja dibuat.



Gambar 5.1 Proses Penambahan Aplikasi



Gambar 5.2 *Client/Consumer Key* dan *Client/Consumer Secret* yang Didapatkan

Pada Gambar 5.1 menunjukkan proses pengisian data-data *client* pada aplikasi WSO2 API *Store*. Data yang diisi berupa nama *client*, deskripsi *client*, kuota setiap *token* dan tipe *token*. Jika sudah selesai, pengguna menekan tombol *add*. Pada Gambar 5.2 menunjukkan *client/consumer key* dan *client/consumer secret* yang didapatkan pengguna. *Client key* dan *client secret* ini merupakan identitas bagi *client* yang didaftarkan.

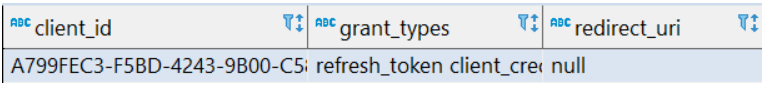
5.2.1.2. Kasus Uji Perubahan Data *Client*

Pada kasus ini akan diuji dengan melakukan perubahan data *Client*. Rincian kasus uji ditunjukkan pada Tabel 5.3.

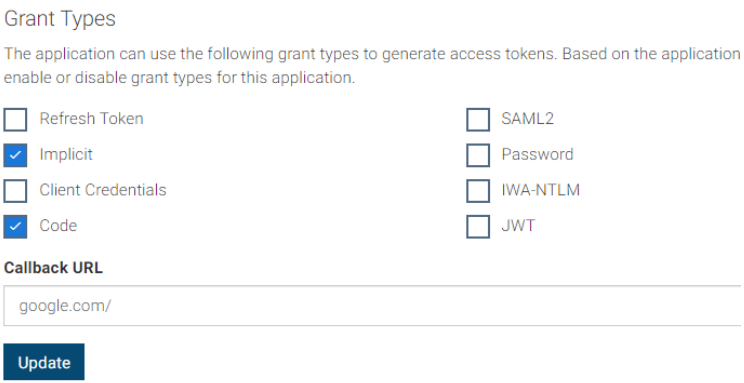
Tabel 5.3 Kasus Uji Perubahan Data *Client*

ID	**TC-002**
Kasus Penggunaan	Perubahan Data *Client.*
Nama	Pengujian perubahan data *client* pada WSO2 API *Manager*.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mengubah data *client* pada MyITS *Single Sign-On*.
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store.*
Langkah Pengujian	1. Aktor mengakses halaman *application.*

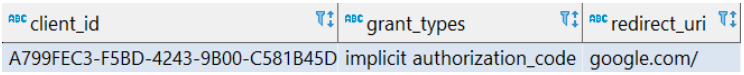
	2. Aktor membuka salah satu *application.* 3. Aktor membuka *tab Production.* 4. Aktor memilih *grant types* dan mengisi *callback* URL. 5. Aktor menekan tombol "*Update*".
Hasil yang diharapkan	*Database client* tersebut berubah.
Kondisi Akhir	*Database client* tersebut berubah.



Gambar 5.3 Data *Client* Sebelum Perubahan



Gambar 5.4 Proses Perubahan Data Client



Gambar 5.5 Data *Client* setelah Dilakukan Perubahan

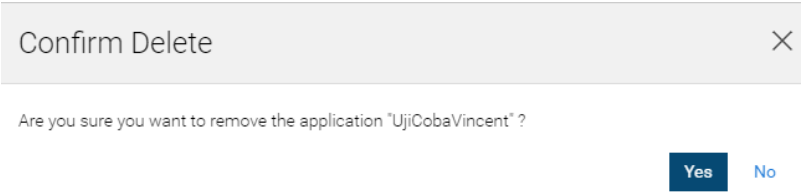
Pada Gambar 5.3 dapat dilihat data *client* sebelum dilakukan perubahan. Gambar 5.4 menunjukkan proses perubahan data *client* yang dilakukan oleh pengguna pada WSO2 API *Store*. Data yang diubah adalah *grant types* dan *redirect uri*. Hasil perubahan data *client* dapat dilihat pada Gambar 5.5.

5.2.1.3. Kasus Uji Penghapusan *Client*

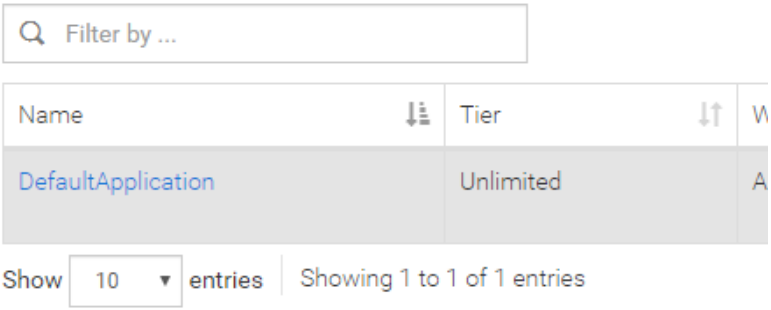
Pada kasus ini akan diuji dengan melakukan penghapusan data *Client.* Rincian kasus uji ditunjukkan pada Tabel 5.4.

Tabel 5.4 Kasus Uji Penghapusan *Client*

ID	**TC-003**
Kasus Penggunaan	Penghapusan *Client.*
Nama	Pengujian penghapusan *client* pada WSO2 API *Manager*.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat menghapus *client* pada MyITS *Single Sign-On*.
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store*
Langkah Pengujian	1. Aktor mengakses halaman *application.* 2. Aktor menekan tombol "*delete"* pada salah satu *application.*
Hasil yang diharapkan	*Application* yang dihapus terhapus pada WSO2 API *Manager* dan MyITS *Single Sign-On.*
Kondisi Akhir	*Application* yang dihapus terhapus pada WSO2 API *Manager* dan MyITS *Single Sign-On.*



Gambar 5.6 Proses Penghapusan *Client*



Gambar 5.7 *Client* Terhapus pada WSO2 API *Manager*

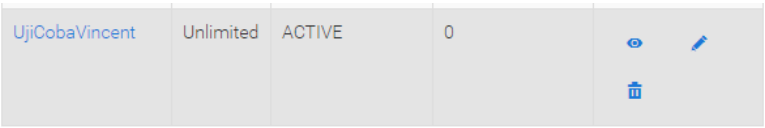
Pada Gambar 5.6 menunjukkan proses penghapusan *client* pada WSO2 API *Store*. Pengguna melakukan konfirmasi penghapusan *client* dengan nama "UjiCobaVincent". Pada Gambar 5.7 dapat dilihat *client* bahwa *client* dengan nama "UjiCobaVincent" sudah tidak ditemukan pada daftar *client* di WSO2 API *Store*.

5.2.1.4. Kasus Uji Pengambilan Data *Client*

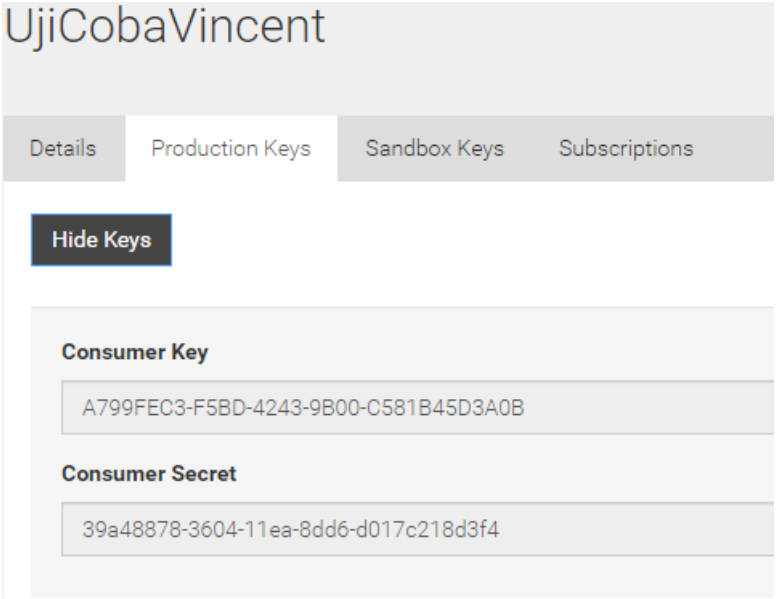
Pada kasus ini akan diuji dengan melakukan *request* data *Client.* Rincian kasus uji ditunjukkan pada Tabel 5.5.

Tabel 5.5 Kasus Uji Pengambilan Data *Client*

ID	**TC-004**
Kasus Penggunaan	Mendapatkan Data *Client.*
Nama	Pengujian mendapatkan data *client* pada WSO2 API *Manager.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendapatkan data *client* pada MyITS *Single Sign-On.*
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store.*
Langkah Pengujian	1. Aktor mengakses halaman *application.* 2. Aktor membuka salah satu halaman *application.* 3. Aktor membuka *tab production.*
Hasil yang diharapkan	Aktor dapat melihat *consumer key* dan *consumer secret* dari aplikasi.
Kondisi Akhir	Aktor dapat melihat *consumer key* dan *consumer secret* dari aplikasi.



Gambar 5.8 Proses Pemilihan *Client*



Gambar 5.9 *Client/Consumer Key* dan *Client/Consumer Secret* yang Didapatkan

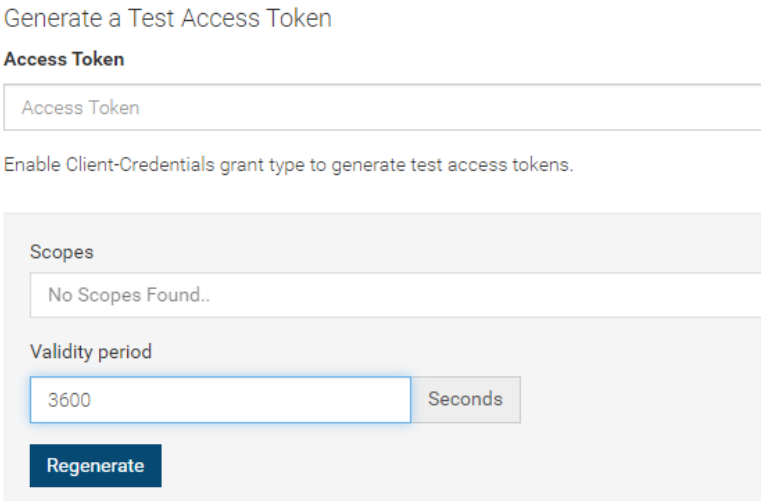
Pada Gambar 5.8 menunjukkan proses pemilihan *client* yang akan dilihat informasinya. *Client* yang dipilih bernama "UjiCobaVincent". Pengguna menekan nama *client* untuk menampilkan informasi dari *client*. Pada Gambar 5.9 dapat dilihat bahwa pengguna telah mendapatkan *consumer/client key* dan *consumer/client secret* dari *client* yang bernama "UjiCobaVincent". *Consumer/client key* dan *consumer/client secret* ini merupakan identitas dari *client*.

5.2.1.5. Kasus Uji *Request Access Token* Melalui Aplikasi WSO2 API *Store*

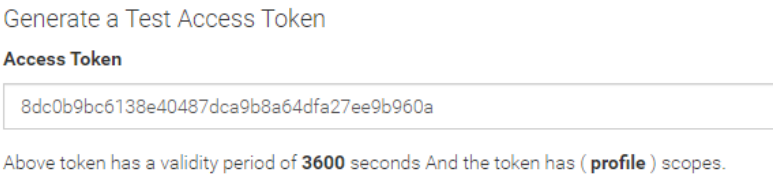
Pada kasus ini akan diuji dengan melakukan *request access token* melalui WSO2 API *Store*. Rincian kasus uji ditunjukkan pada Tabel 5.6.

Tabel 5.6 Kasus Uji *Request Access Token* Melalui Aplikasi WSO2 API *Store*

ID	**TC-005**
Kasus Penggunaan	*Request Access Token* melalui Aplikasi WSO2 API *Store.*
Nama	Pengujian *Request Access Token* melalui Aplikasi WSO2 API *Store.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendapatkan *Access Token* melalui Aplikasi WSO2 API *Store.*
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store.*
Langkah Pengujian	1. Aktor mengakses halaman *application.* 2. Aktor membuka salah satu halaman *application.* 3. Aktor membuka *tab production.* 4. Aktor menekan tombol "*regenerate*".
Hasil yang diharapkan	Aktor mendapatkan *access token*.
Kondisi Akhir	Aktor mendapatkan *access token*.



Gambar 5.10 Proses Permintaan *Access Token* Pada WSO2 API *Store*



Gambar 5.11 *Access Token* yang Didapatkan dari WSO2 API *Manager*

Pada Gambar 5.10 menunjukkan proses permintaan *access token* melalui WSO2 API *Store*. Pengguna mengisi *validity period* dan menekan tombol *regenerate*. *Access token* yang didapatkan oleh pengguna ditunjukan pada Gambar 5.11.

5.2.1.6. Kasus Uji *Request Access Token* Melalui URL

Pada kasus ini akan diuji dengan melakukan *request access token* melalui URL. Rincian kasus uji ditunjukkan pada Tabel 5.7.

Tabel 5.7 Kasus Uji *Request Access Token* Melalui URL

ID	**TC-006**
Kasus Penggunaan	*Request Access Token* melalui URL.
Nama	Pengujian *Request Access Token* melalui URL.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendapatkan *Access Token* melalui URL.
Skenario 1	Menggunakan *Grant Type Client Credentials.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request access token* melalui URL yang disediakan WSO2 API *Store* menggunakan *client credential*
Hasil yang diharapkan	Aktor mendapatkan *access token*.
Kondisi Akhir	Aktor mendapatkan *access token*.
Skenario 2	Menggunakan *Grant Type User Credentials.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request access token* melalui URL yang disediakan WSO2 API *Store* menggunakan *user credential.*
Hasil yang diharapkan	Aktor mendapatkan *access token*.
Kondisi Akhir	Aktor mendapatkan *access token*.

Skenario 3	Menggunakan *Grant Type Refresh Token.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request access token* melalui URL yang disediakan WSO2 API *Store* menggunakan *refresh token.*
Hasil yang diharapkan	Aktor mendapatkan *access token*.
Kondisi Akhir	Aktor mendapatkan *access token*.

```
C:\Users\ASUS>curl -k -d "grant_type=client_credentials" -H "Authorization: Ba
sic MzlCMjM1MkYtREFBNS00MDM4LUIzMTUtODAxQUE4OUM1NTBBOjJmZWFmMjRhLTM1ZWItMTFlYS
1iZWM0LWQwMTdjMjE4ZDNmNA==" https://api.its.ac.id:8243/token
```

Gambar 5.12 Contoh *Request Access Token* dengan *Client Credentials*

```
{"access_token":"85adc34c799a950fc5a5b74375aaeedc23ef3b13","expires_in":3600,"
token_type":"Bearer","scope":"profile"}
```

Gambar 5.13 Contoh *Response Access Token* Menggunakan *Client Credentials*

```
C:\Users\ASUS>curl -k -d "grant_type=password&username=5116100089&password=tes
ting" -H "Authorization: Basic MzlCMjM1MkYtREFBNS00MDM4LUIzMTUtODAxQUE4OUM1NTB
BOjJmZWFmMjRhLTM1ZWItMTFlYS1iZWM0LWQwMTdjMjE4ZDNmNA==" https://api.its.ac.id:8
243/token
```

Gambar 5.14 Contoh *Request Access Token* Menggunakan *User Credentials*

```
{"access_token":"844fd4cc1b49684bdbdc4d00a3faa2b0dd59d16c","expires_in":3600,"
token_type":"Bearer","scope":"profile","refresh_token":"61424d2bb9a10947c13eb7
3739844f51c27d76b4"}
```

Gambar 5.15 Contoh *Response Access Token* Menggunakan *User Credentials*

```
C:\Users\ASUS>curl -k -d "grant_type=refresh_token&refresh_token=61424d2bb9a10
947c13eb73739844f51c27d76b4" -H "Authorization: Basic MzlCMjM1MkYtREFBNS00MDM4
LUIzMTUtODAxQUE4OUM1NTBBOjJmZWFmMjRhLTM1ZWItMTFlYS1iZWM0LWQwMTdjMjE4ZDNmNA==" 
https://api.its.ac.id:8243/token
```

Gambar 5.16 Contoh *Request Access Token* Menggunakan *Refresh Token*

```
{"access_token":"f6a7b5092a787c05f4734117d9063d7e6087bd40","expires_in":3600,"
token_type":"Bearer","scope":"profile"}
```

Gambar 5.17 Contoh *Response Access Token* Menggunakan *Refresh Token*

Pada Gambar 5.12, Gambar 5.14 dan Gambar 5.16 ditunjukkan proses permintaan *access token* melalui URL. Proses permintaan *access token* ini dilakukan pada *command prompt* dengan menggunakan *client credentials, user credentials* dan *refresh token*. *Access token* yang didapatkan ditampilkan pada Gambar 5.13, Gambar 5.15 dan Gambar 5.17.

5.2.1.7. Kasus Uji *Request Resource* Menggunakan *Access Token* yang Benar Melalui URL

Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang benar melalui URL. Rincian kasus uji ditunjukkan pada Tabel 5.8.

Tabel 5.8 Kasus Uji *Request Resource* Menggunakan *Access Token* yang Benar Melalui URL

ID	**TC-007**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang benar melalui URL.
Nama	Pengujian *Request Resource* menggunakan *Access Token* yang benar melalui URL.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendapatkan *resource* melalui *Request*

	Resource menggunakan *Access Token* yang benar melalui URL.
Skenario 1	Menggunakan *access token* yang didapat melalui *Grant Type Client Credentials.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui URL yang disediakan WSO2 API *Store* menggunakan *access token.*
Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*
Skenario 2	Menggunakan *access token* yang didapat melalui *Grant Type User Credential.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui URL yang disediakan WSO2 API *Store* menggunakan *access token.*
Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*
Skenario 3	Menggunakan *access token* yang didapat melalui *Grant Type Authorization Code*.
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui URL yang disediakan WSO2 API *Store* menggunakan *access token.*

Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*

```
C:\Users\ASUS>curl -k -X GET "https://192.168.1.8:8243/v1/1.0.3/store/inventor
y" -H "accept: application/json" -H "Authorization: Bearer 8dc0b9bc6138e40487d
ca9b8a64dfa27ee9b960a"
```

Gambar 5.18 Contoh *Request Resource* Menggunakan URL


```
{"Done":1,"UniqueStatusForTest":4,"string":17561,"alive":1,"unavailable":1,"pe
nding":91,"available":2521,"b-ok":2,"secret":1,"sleeping":3,"ckwtcjyk":1,"swim
ming":1,"111":2,"|ping -c2 -i91 localhost|":2,"not available":32,"Available":3
,"\"pending\"":1,"ee":1,"sold":1523,"ddd":1,"Nonavailable":1,"testing":1,"retr
o":1,"XXXX":1,"YYYY":1,"PENDING":2,"UniquePetStatusForTest":1,"dfff":1,"dement
ed":1,"hhhh":1,"0":1,"1":6,"Ready":1,"ssss":1,"Ok":1,"InactiveTest":1,"4756":1
,"Pending":1}
```


Gambar 5.19 Contoh *Response Resource* menggunakan URL

Pada Gambar 5.18 dilakukan permintaan *resource* melalui URL. Proses permintaan *resource* ini dilakukan pada *command prompt*. *Resource* yang didapatkan dapat dilihat pada Gambar 5.19. *Response* yang diterima berisi data-data yang telah diminta.

5.2.1.8. Kasus Uji *Request Resource* Menggunakan *Access Token* yang Benar Melalui Aplikasi WSO2 API *Store*

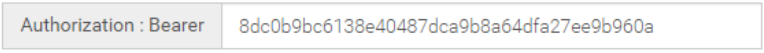
Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang benar melalui WSO2 API Store. Rincian kasus uji ditunjukkan pada Tabel 5.9.

Tabel 5.9 Kasus Uji *Request Resource* Menggunakan *Access Token* yang Benar Melalui Aplikasi WSO2 API *Store*

ID	**TC-008**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang benar melalui WSO2 API *Store.*
Nama	Pengujian *Request Resource* menggunakan *Access Token*

	yang benar melalui WSO2 API *Store.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendapatkan *resource* melalui *Request Resource* menggunakan *Access Token* yang benar melalui WSO2 API *Store.*
Skenario 1	Menggunakan *access token* yang didapat melalui *Grant Type Client Credentials.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor membuka halaman API. 2. Aktor melakukan *request resource* melalui WSO2 API *Store* menggunakan *access token.*
Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*
Skenario 2	Menggunakan *access token* yang didapat melalui *Grant Type User Credential.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor membuka halaman API. 2. Aktor melakukan *request resource* melalui WSO2 API *Store* menggunakan *access token.*
Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*

Skenario 3	Menggunakan *access token* yang didapat melalui *Grant Type Refresh Token*.
Kondisi Awal	-
Langkah Pengujian	1. Aktor membuka halaman API. 2. Aktor melakukan *request resource* melalui WSO2 API *Store* menggunakan *access token.*
Hasil yang diharapkan	Aktor mendapatkan data *resource.*
Kondisi Akhir	Aktor mendapatkan data *resource.*



Gambar 5.20 Contoh *Access Token* yang Digunakan pada WSO2 API *Store*


```
{
  "id": 21435431254680,
  "name": "Snoop",
  "photoUrls": [
    "http://image/dog85.jpg"
  ],
  "tags": [
    {
      "id": 58,
      "name": "black"
    }
  ],
  "status": "available"
}
```


Gambar 5.21 Contoh *Response Resource* pada WSO2 API *Store*

Access token yang digunakan untuk melakukan permintaan *resource* dapat dilihat pada Gambar 5.20. *Response request resource* melalui WSO2 API *Store* dapat dilihat Gambar 5.21. *Response* yang diterima berisi data-data yang telah diminta.

5.2.1.9. Kasus Uji *Request Resource* Menggunakan *Access Token* yang Salah Melalui URL

Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang salah melalui URL. Rincian kasus uji ditunjukkan pada Tabel 5.10.

Tabel 5.10 Kasus Uji *Request Resource* Menggunakan *Access Token* yang Salah Melalui URL

ID	**TC-009**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang salah melalui URL.
Nama	Pengujian *Request Resource* menggunakan *Access Token* yang salah melalui URL.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat menolak memberikan *resource* melalui *Request Resource* jika menggunakan *Access Token* yang salah melalui URL.
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui URL menggunakan *access token* yang salah.
Hasil yang diharapkan	Aktor mendapatkan penolakan
Kondisi Akhir	Aktor mendapatkan penolakan

```
C:\Users\ASUS>curl -k -X GET "https://api.its.ac.id:8243/v1/1.0.3/store/invent
ory" -H "accept: application/json" -H "Authorization: Bearer 8dc0b9bc6138e4048
7dca9b8a64dfa27ee9b960s"
```

Gambar 5.22 Contoh *Request Resource* Menggunakan *Access Token* yang Salah Melalui URL

```
{"fault":{"code":900904,"message":"Access Token Inactive","description":"Acces
s failure for API: /v1/1.0.3, version: 1.0.3 status: (900904) - Access Token I
nactive. Generate a new access token and try again"}}
```

Gambar 5.23 Contoh *Response Resource* Menggunakan *Access Token* yang Salah Melalui URL

Pada Gambar 5.22 menunjukkan proses permintaan *resource* dengan *access token* yang salah. Permintaan *resource* ini dilakukan menggunakan *command prompt*. *Response* yang diterima dapat dilihat pada Gambar 5.23. *Response* yang diterima menampilkan pesan bahwa *access token* yang digunakan tidak aktif atau tidak dapat digunakan.

5.2.1.10. Kasus Uji *Request Resource* Menggunakan *Access Token* yang Salah Melalui WSO2 API *Store*

Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang salah melalui WSO2 API *Store*. Rincian kasus uji ditunjukkan pada Tabel 5.11.

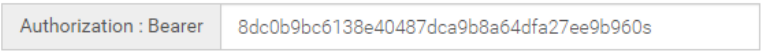
Tabel 5.11 Kasus Uji *Request Resource* Menggunakan *Access Token* yang Salah Melalui WSO2 API *Store*

ID	**TC-010**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang salah melalui WSO2 API *Store.*
Nama	Pengujian *Request Resource* menggunakan *Access Token* yang salah melalui WSO2 API *Store.*

Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat menolak memberikan *resource* melalui *Request Resource* jika menggunakan *Access Token* yang salah melalui WSO2 API *Store.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui WSO2 API *Store* menggunakan *access token* yang salah.
Hasil yang diharapkan	Aktor mendapatkan penolakan
Kondisi Akhir	Aktor mendapatkan penolakan


```
{
  "fault": {
    "code": 900904,
    "message": "Access Token Inactive",
    "description": "Access failure for API: /v1/1.0.3,
version: 1.0.3 status: (900904) - Access Token
Inactive. Generate a new access token and try again"
  }
}
```


Gambar 5.24 Contoh *Response Resource* Menggunakan *Access Token* yang Salah pada WSO2 API *Manager*



Gambar 5.25 Contoh *Access Token* yang Salah

Pada Gambar 5.24 menunjukkan *response resource* yang diterima jika menggunakan *access token* yang salah. *Response* berisi pesan bahwa *access token* yang digunakan tidak aktif. *Access token* yang digunakan dapat dilihat pada Gambar 5.25.

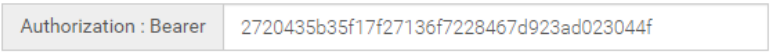
5.2.1.11. Kasus Uji *Request Resource* menggunakan *Access Token* yang *Expired* melalui WSO2 API *Store*

Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang *expired* melalui WSO2 API *Store*. Rincian kasus uji ditunjukkan pada Tabel 5.12.

Tabel 5.12 Kasus Uji *Request Resource* Menggunakan *Access Token* yang *Expired* Melalui WSO2 API *Store*

ID	**TC-011**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang *expired* melalui WSO2 API *Store.*
Nama	Pengujian *Request Resource* menggunakan *Access Token* yang *expired* melalui WSO2 API *Store.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat menolak memberikan *resource* melalui *Request Resource* jika menggunakan *Access Token* yang *expired* melalui WSO2 API *Store.*
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui WSO2 API *Store* menggunakan *access token* yang *expired*.
Hasil yang diharapkan	Aktor mendapatkan penolakan

Kondisi Akhir	Aktor mendapatkan penolakan



Gambar 5.26 Contoh *Access Token* yang *Expired*

```
{
  "fault": {
    "code": 900904,
    "message": "Access Token Inactive",
    "description": "Access failure for API: /v1/1.0.3,
version: 1.0.3 status: (900904) - Access Token
Inactive. Generate a new access token and try again"
  }
}
```

Gambar 5.27 Contoh *Response* Menggunakan *Access Token* yang *Expired* pada WSO2 API *Store*

Pada Gambar 5.26 menunjukkan *access token expired* yang digunakan untuk melakukan *request resource*. *Response* yang diterima dapat dilihat pada Gambar 5.27. *Response* menampilkan pesan bahwa *access token* yang digunakan tidak aktif.

5.2.1.12. Kasus Uji *Request Resource* Menggunakan *Access Token* yang Expired Melalui URL

Pada kasus ini akan diuji dengan melakukan *request resource* dengan menggunakan *access token* yang *expired* melalui URL. Rincian kasus uji ditunjukkan pada Tabel 5.13.

Tabel 5.13 Kasus Uji *Request Resource* Menggunakan *Access Token* yang *Expired* Melalui URL

ID	**TC-012**
Kasus Penggunaan	*Request Resource* menggunakan *Access Token* yang *expired* melalui URL.

Nama	Pengujian *Request Resource* menggunakan *Access Token* yang *expired* melalui URL.
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat menolak memberikan *resource* melalui *Request Resource* jika menggunakan *Access Token* yang *expired* melalui URL.
Kondisi Awal	-
Langkah Pengujian	1. Aktor melakukan *request resource* melalui URL menggunakan *access token* yang *expired*.
Hasil yang diharapkan	Aktor mendapatkan penolakan
Kondisi Akhir	Aktor mendapatkan penolakan

```
C:\Users\ASUS>curl -k -X GET "https://api.its.ac.id:8243/v1/1.0.3/pet/1" -H "a
ccept: application/json" -H "Authorization: Bearer 2720435b35f17f27136f7228467
d923ad023044f"
```

Gambar 5.28 Contoh *Request Resource* Menggunakan *Access Token Expired* Melalui URL

```
{"fault":{"code":900904,"message":"Access Token Inactive","description":"Acces
s failure for API: /v1/1.0.3, version: 1.0.3 status: (900904) - Access Token I
nactive. Generate a new access token and try again"}}
```

Gambar 5.29 Contoh *Response* Menggunakan *Access Token* yang *Expired* Melalui URL

Pada Gambar 5.28 menunjukkan *request resource* menggunakan *access token* yang *expired*. *Request* ini dilakukan menggunakan *command prompt*. *Response* yang didapatkan dapat dilihat pada Gambar 5.29. *Response* menampilkan pesan bahwa *access token* yang digunakan tidak aktif.

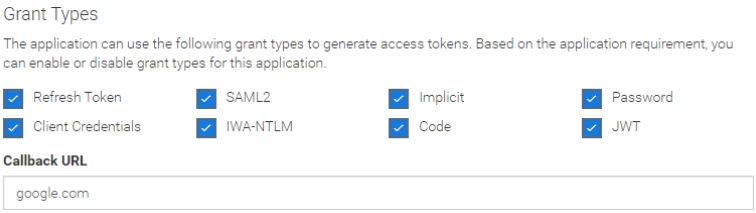
5.2.1.13. Kasus Uji Tambah *Client* Dengan Menggunakan Semua *Grant Type*

Pada kasus ini akan diuji dengan melakukan penambahan *client* menggunakan semua *grant type* pada WSO2 API *Store*. Rincian kasus uji ditunjukan pada Tabel 5.14.

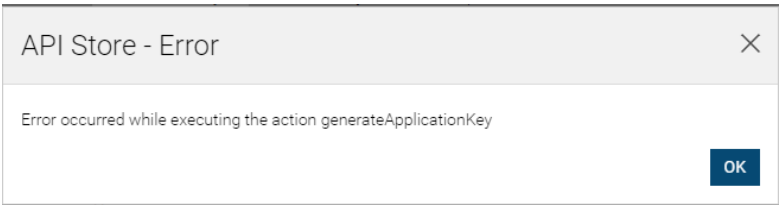
Tabel 5.14 Kasus Uji Tambah *Client* Dengan Menggunakan Semua *Grant Type*

ID	**TC-013**
Kasus Penggunaan	Pendaftaran *Client* dengan menggunakan semua *grant type.*
Nama	Pengujian pendaftaran *client* pada WSO2 API *Manager* dengan menggunakan semua *grant type.*
Tujuan Pengujian	Menguji apakah WSO2 API *Manager* dapat mendaftarkan/menambahkan *client* pada MyITS *Single Sign-On* dengan menggunakan semua *grant type.*
Kondisi Awal	Aktor telah *login* pada WSO2 API *Store.*

Langkah Pengujian	1. Aktor mengakses halaman *add application.* 2. Aktor mengisi data *client* yang akan dibuat. 3. Aktor menekan tombol "*add*". 4. Aktor membuka *tab Production.* 5. Aktor memilih semua *grant type.* 6. Aktor mengisi *redirect uri* dari *client* 7. Aktor menekan tombol "*Generate Keys*".
Hasil yang diharapkan	Aktor mendapatkan *client id* dan *client secret* dari *client* yang baru saja dibuat.
Kondisi Akhir	Aktor mendapatkan pesan notifikasi *error.*



Gambar 5.30 Proses Penambahan *Client* dengan Menggunakan Semua *Grant Type*



Gambar 5.31 Pesan *Error* yang Didapatkan Setelah Menambahkan *Client* dengan Menggunakan Semua *Grant Type*

Pada Gambar 5.30 ditunjukkan proses penambahan *client* dengan menyertakan semua *grant type*. *Grant type* yang disertakan yaitu *refresh token, client credentials,* SAML2, IWA-NTLM, *implicit, code, password* dan JWT. Sistem menampilkan pesan *error* yang dapat dilihat pada Gambar 5.31.

5.3. Evaluasi

Pada Subbab ini dijelaskan hasil dari pengujian yang dilakukan pada Subbab sebelumnya.

5.3.1. Evaluasi Fungsionalitas Sistem

Evaluasi ini adalah hasil dari pengujian kasus uji pada Subbab 5.2.1. Hasil dinyatakan dalam status terpenuhi atau tidak. Hasil tersebut ditunjukkan pada Tabel 5.15.

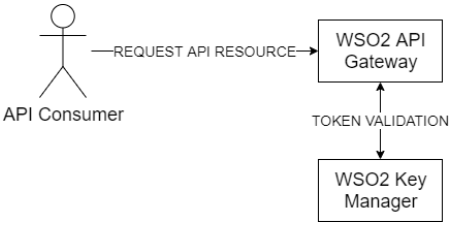
Tabel 5.15 Tabel Evaluasi Kasus Pengujian Fungsionalitas Sistem

No.	Kode Kasus Pengujian	Terpenuhi
1	TC-001	√
2	TC-002	√
3	TC-003	√
4	TC-004	√
5	TC-005	√
6	TC-006	√
7	TC-007	√
8	TC-008	√
9	TC-009	√

10	TC-010	√
11	TC-011	√
12	TC-012	√
13	TC-013	×

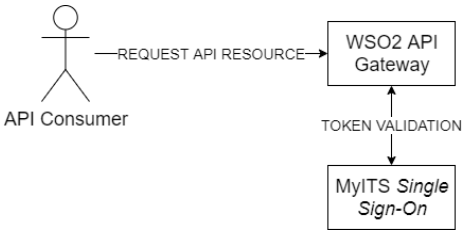
Terdapat satu kasus pengujian yang tidak terpenuhi, yaitu kasus pengujian TC-013. Hal ini dikarenakan adanya *grant type* pada WSO2 API *Manager* yang tidak didukung oleh MyITS *Single Sign-On*. *Grant type* yang tidak didukung oleh MyITS *Single Sign-On* yaitu SAML2 dan IWA-NTLM. Hal ini tidak mengganggu proses pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On* jika kedua *grant type* tersebut tidak digunakan pada WSO2 API *Manager*.

Pada saat melakukan uji coba aplikasi WSO2 API Manager terdapat beberapa perbedaan alur saat melakukan *request* API.



Gambar 5.32 Skema API *Request* Sebelum Dilakukan Pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On*

Sebelum melakukan integrasi WSO2 API *Manager* dengan MyITS *Single Sign-On*. Pada saat API *Consumer* melakukan *request* API kepada WSO2 API *Gateway*. API *Gateway* akan melakukan *request token validation* kepada WSO2 *Key Manager*. Jika *token* tervalidasi maka *resource* akan diberikan kepada API *Consumer*. Gambar 5.32 menunjukan skema API *Request* sebelum dilakukan pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On*.



Gambar 5.33 Skema API *Request* Setelah Dilakukan Pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On*

Setelah melakukan integrasi antara WSO2 API *Manager* dengan MyITS *Single Sign-On*. Ketika API *Consumer* melakukan *request* API kepada WSO2 API *Gateway*. API *Gateway* akan melakukan *request token validation* kepada MyITS *Single Sign-On*. Jika *token* tervalidasi oleh MyITS *Single Sign-On*, *resource* akan diberikan kepada API *Consumer*. Gambar 5.33 menunjukkan skema API *Request* setelah dilakukan pengintegrasian WSO2 API *Manager* dengan MyITS *Single Sign-On*.

Perbedaan ini akan mengakibatkan beban pada MyITS *Single Sign-On* akan semakin tinggi dikarenakan pada setiap API *Request*. WSO2 API *Manager* akan berkomunikasi dengan MyITS *Single Sign-On* untuk melakukan pengecekan terhadap keabsahan suatu *token*. Hal ini akan meningkatkan jumlah pemrosesan data pada *server* MyITS *Single Sign-On.* Masalah ini akan dapat ditangani dengan meningkatkan performa *server* pada MyITS *Single Sign-On*.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1. Kesimpulan

Berikut merupakan kesimpulan yang dapat diambil dari proses pengembangan dan uji coba:

1. Basis data *client* dan pengguna pada WSO2 API *Manager* dapat disatukan dengan basis data *client* dan pengguna pada MyITS *Single Sign-On.*
2. Konektor antara MyITS *Single Sign-On* dan WSO2 API *Manager* dapat diimplementasikan sehingga MyITS *Single Sign-On* dapat digunakan sebagai server otorisasi.
3. *Client Registration Endpoint* dapat diimplementasikan pada MyITS *Single Sign-On* sehingga MyITS *Single Sign-On* dapat digunakan sebagai server otorisasi.
4. OAuth2 *Token Introspection* dapat diimplementasikan pada MyITS *Single Sign-On* sehingga dapat digunakan sebagai server otorisasi.
5. WSO2 API *Manager* dapat dikonfigurasi sehingga WSO2 API *Manager* dapat mengakses MyITS *Single Sign-On* sebagai server otorisasi eksternal.
6. *Grant-type* SAML2 dan IWA-NTLM tidak dapat digunakan pada saat menggunakan MyITS *Single Sign-On* sebagai *external authorization server* pada WSO2 API *Manager.*

6.2. Saran

Saran untuk pengembangan sistem di masa yang akan datang adalah melakukan analisis beban kerja pada MyITS *Single Sign-*

On untuk menangani *request* yang meningkat dari WSO2 API *Manager*.

GLOSARIUM

Access Token	**:**	Sebuah kata yang tidak bermakna yang digunakan untuk mengidentifikasi pengguna atau aplikasi dan dapat digunakan untuk mengakses API. Contoh *access token* : 8dv0b9bc6135e45e54568d1c95ddsasda5d

API	**:**	*Application Programming Interface* yang merupakan penerjemah komunikasi antara *client* dan *server*. Contohnya ketika aplikasi *mobile* ingin mengambil data user dari *server* autorisasi. Diperlukan suatu standar untuk dapat membaca pesan yang disampaikan *server* autorisasi. Contoh pesan yang disampaikan server autorisasi:
		{
		“username”: “Vincent”,
		“email” : “vincent@gmail.com”
		}

API *Consumer*	**:**	*Developer* atau *client* yang menggunakan API. Contoh: Developer aplikasi *mobile*, aplikasi Presensi ITS, dan lain-lain.
API *Provider*	**:**	*Developer* yang menyediakan API Contoh: Developer yang mengembangkan API Presensi ITS.
GUI	**:**	*Graphical User Interface* merupakan metode interaksi secara grafis. *User* dapat berinteraksi dengan sistem secara grafis. Contoh: Halaman Website dan Aplikasi *Mobile*
Deployment	**:**	Tahapan di mana sistem dibuat tersedia bagi pengguna. Sebagai contoh ketika *user* dapat mengakses halaman suatu *website*,

maka *website* tersebut sudah dalam tahapan *deployment.*

Application Server **:** Aplikasi *server* merupakan sebuah aplikasi komputer dengan fungsi untuk melayani permintaan *user* terhadap permintaan akses yang berasal dari komputer *client*. Contoh : *Web Server*.

Service Level Agreement **:** Kontrak dari penyedia layanan dengan kita sebagai pengguna yang memberikan jaminan tingkat pelayanan yang dapat diharapkan. Contoh *Service Level Aggrement* : Waktu yang dibutuhkan untuk menyelesaikan pekerjaan adalah 3 bulan.

Lifecycle **:** Siklus hidup dari sebuah produk. Contoh ketika membuat sebuah aplikasi, terdapat siklus hidup mulai dari analisa kebutuhan, desain, implementasi, uji coba dan evolusi.

Throttling **:** Pembatasan pada suatu sistem. Contoh: *Request* suatu API dibatasi hanya 10 *request/*menit.

Client **:** Aplikasi yang mengakses data dari sebuah API. Contoh: Aplikasi MyITS *Mobile*, Aplikasi Integra dan lain-lain.

OAuth2 **:** Suatu protokol terbuka yang memungkinkan pengguna untuk berbagi sumber pribadi mereka yang disimpan di suatu situs *web* dengan situs lain tanpa perlu menyerahkan nama pengguna dan kata sandi mereka.

Resource **:** Suatu data yang tersimpan pada *resource server* yang dapat diakses oleh *client/user* tertentu. Contoh *Resource :* Biodata *user*, Foto *user* dan lain-lain.

Resource Owner **:** Pengguna yang melakukan otorisasi aplikasi untuk dapat mengakses ke akun

mereka. Akses aplikasi ke akun pengguna terbatas pada otorisasi yang diberikan. Contoh: Ketika pengguna *login* ke suatu aplikasi, pengguna memberikan otorisasi dengan memberikan *username* dan *password* kepada *user*.

Grant Type **:** Metode pemberian akses ke resources (sumber data) yang dilindungi dengan berbagai cara dan keamanan data yang berbeda. Contoh *grant type: Client Credentials, Refresh Token* dan lain-lain.

Endpoint **:** URL dimana suatu servis dapat diakses oleh *client*. Contoh : integra.its.ac.id

Client ID **:** Pengidentifikasi suatu *client.* Membedakan *client* yang satu dengan yang lain. Contoh *Client ID* : 9c94b349-d6a3-49b4-bd5e-9dccc7cddcf8

Client Secret **:** Kode rahasia yang dimiliki oleh client. Kode ini digunakan untuk membuktikan keabsahan dari sebuah *client*. Contoh *client secret:* 2123db9bc61f5e45254568d1c95ddsasda5d

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] "About this Release - API Manager 2.6.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/AM260/About+this+Release. [Accessed: 23-Dec-2019].

[2] "Configuring a Third-Party Key Manager - API Manager 2.6.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/AM260/Configuring+a+Third-Party+Key+Manager. [Accessed: 23-Dec-2019].

[3] K. Dwiastini, "Implementasi Otentikasi Single Sign On Dan Otorisasi Role Based Access Control Berbasis Standar OpenId Connect," Institut Teknologi Sepuluh Nopember, 2018.

[4] M. Anicas, "An Introduction to OAuth 2 | DigitalOcean," 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Accessed: 23-Dec-2019].

[5] "Extending Key Validation - API Manager 2.6.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/AM260/Extending+Key+Validation. [Accessed: 27-Dec-2019].

[6] "Key Concepts - API Manager 2.6.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/AM260/Key+Concepts. [Accessed: 23-Dec-2019].

[7] "Extending the Key Manager Interface - API Manager 2.6.0 - WSO2 Documentation." [Online]. Available: https://docs.wso2.com/display/AM260/Extending+the+Key+Manager+Interface. [Accessed: 22-Dec-2019].

[8] D. Hardt, "RFC 6749 - The OAuth 2.0 Authorization Framework," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749#section-3.2. [Accessed: 23-Dec-2019].

[9] J. Richer, “RFC 7662 - OAuth 2.0 Token Introspection,” 2015. [Online]. Available: https://tools.ietf.org/html/rfc7662. [Accessed: 23-Dec-2019].

[10] J. Richer, “RFC 7591 - OAuth 2.0 Dynamic Client Registration Protocol,” 2015. [Online]. Available: https://tools.ietf.org/html/rfc7591#section-3. [Accessed: 23-Dec-2019].

[11] J. Richer, “RFC 7592 - OAuth 2.0 Dynamic Client Registration Management Protocol,” 2015. [Online]. Available: https://tools.ietf.org/html/rfc7592. [Accessed: 23-Dec-2019].

[12] “OAuth2 Server PHP.” [Online]. Available: https://bshaffer.github.io/oauth2-server-php-docs/. [Accessed: 06-Jan-2020].

BIODATA PENULIS

Penulis bernama Vincent Marcello Dwi Tanujaya, lahir di Jepara tanggal 6 Maret 1998 anak kedua dari dua bersaudara. Penulis menempuh pendidikan di SD Kanisius Jepara pada tahun 2004 hingga 2010, SMP Negeri 1 Jepara pada tahun 2010 hingga 2013 dan SMA Negeri 3 Semarang pada tahun 2013 hingga 2016. Pada masa penulisan, penulis sedang menempuh pendidikan masa studi S1 tahun keempat pada Departemen Teknik Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember.

Penulis memiliki ketertarikan mengenai rancang bangun aplikasi *web*, basis data dan rekayasa perangkat lunak. Penulis pernah menjadi asisten dosen pada mata kuliah Manajemen Basis Data dan Pemrograman Berorientasi Objek. Dalam mengembangkan kemampuan, penulis juga pernah bekerja sebagai *backend developer*.

Di luar kesibukan akademis, penulis aktif di organisasi kemahasiswaan di Departemen Informatika. Pada tahun kedua penulis diamanahi sebagai Staff Dalam Negeri Himpunan Mahasiswa Teknik Computer-Informatika Kreasi 2017/2018. Pada tahun ketiga penulis diamanahi sebagai Wakil Ketua Himpunan Mahasiswa Teknik Computer-Informatik Garang 2018/2019. Penulis dapat dihubungi melalui surel di **vincent.tanujaya@gmail.com**