TUGAS AKHIR – IF184802

IMPLEMENTASI PROTOKOL CLIENT INITIATED BACKCHANNEL AUTHENTICATION (CIBA) DI SISI SERVER PADA MYITS SINGLE SIGN-ON

MUHAMMAD ADISTYA AZHAR
NRP 05111640000103

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Nurul Fajrin Ariyani, S.Kom., M.Sc.

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

TUGAS AKHIR – IF184802

IMPLEMENTASI PROTOKOL CLIENT INITIATED BACKHANNEL AUTHENTICATION (CIBA) DI SISI SERVER PADA MYITS SINGLE SIGN-ON

MUHAMMAD ADISTYA AZHAR
NRP 05111640000103

Dosen Pembimbing
Rizky Januar Akbar, S.Kom., M.Eng.
Nurul Fajrin Ariyani, S.Kom., M.Sc.

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

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF CLIENT INITIATED BACKCHANNEL AUTHENTICATION (CIBA) SERVER SIDE PROTOCOL ON TOP OF MYITS SINGLE SIGN-ON

MUHAMMAD ADISTYA AZHAR
NRP 05111640000103

Supervisor
Rizky Januar Akbar, S.Kom., M.Eng.
Nurul Fajrin Ariyani, S.Kom., M.Sc.

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

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI PROTOKOL CLIENT INITIATED BACKCHANNEL AUTHENTICATION (CIBA) DI SISI SERVER PADA MYITS SINGLE SIGN-ON

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:
MUHAMMAD ADISTYA AZHAR
NRP: 05111640000103

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Nurul Fajrin Ariyani, S.Kom., M.Sc. .....
NIP: 19860722 201504 2 003 (pembimbing 2)

SURABAYA
JUNI 2020

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI PROTOKOL CLIENT INITIATED BACKCHANNEL AUTHENTICATION (CIBA) DI SISI SERVER PADA MYITS SINGLE SIGN-ON

Nama Mahasiswa	: MUHAMMAD ADISTYA AZHAR
NRP	: 05111640000103
Departemen	: Teknik Informatika, FTEIC-ITS
Dosen Pembimbing I	: Rizky Januar Akbar, S.Kom., M.Eng.
Dosen Pembimbing II	: Nurul Fajrin Ariyani, S.Kom., M.Sc.

Abstrak

Berkembangnya instansi pada sektor online banking dan financial technology sangat bergantung pada mekanisme keamanan data pelanggan. Dengan menanggapi itu, telah terbit protokol autentikasi dan autorisasi yang dinamakan dengan Client Initiated Backchannel Authentication (CIBA) yang mengakomodasi kasus penggunaan dimana, Consumption Device, perangkat yang digunakan untuk berinteraksi dengan Relying Party/ Aplikasi Klien, dan Authentication Device, perangkat yang digunakan untuk berautentikasi dengan Authorization Server/ server autorisasi dan memberikan izin hak akses, merupakan perangkat yang terpisah secara fisik. Kasus ini umum ditemukan di layanan pelanggan online banking dan sistem point of sales.

Dalam tugas akhir ini, sistem server CIBA dikembangkan diatas library server autorisasi yang dinamakan dengan Bshaffer OAuth 2.0, yaitu library yang menjalankan server autorisasi milik Institut Teknologi Sepuluh Nopember (ITS), myITS SSO. Sistem ini ditekankan untuk mendukung masalah CIBA yang mendasar seperti pendaftaran aplikasi klien, inisiasi autentikasi, pengambilan access token, dan session binding.

Pengujian terhadap sistem ini dilakukan menggunakan aplikasi Postman pada setiap endpoint CIBA server. Pada setiap endpoint server, diberikan parameter request dengan nilai yang sesuai dan tidak sesuai. Test case yang diaplikasikan diadopsi dari conformance testing Financial Grade API CIBA (FAPI CIBA). Server dapat mematuhi spesifikasi CIBA dengan cara memberi

pesan error beserta penjelasannya ketika parameter request tidak sesuai, dan sebaliknya melanjutkan request jika parameter request sudah sesuai.

Protokol CIBA berhasil dikembangkan pada tugas akhir ini sesuai spesifikasi tanpa menghambat protokol lainnya yang berjalan di sever autorisasi myITS SSO, dikarenakan arsitektur library yang decoupled dengan cara memanfaatkan prinsip pemrograman SOLID. CIBA dapat berjalan secara fungsional dengan fitur pendaftaran aplikasi klien, alur autentikasi dan pengambilan access token. Aplikasi klien yang terdaftar di myITS SSO memiliki pilihan protokol baru, sehingga kasus penggunaan autentikasi yang membutuhkan perangkat terpisah dapat diaplikasikan menggunakan CIBA. Di lingkungan ITS, layanan Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI) help desk dapat memanfaatkan protokol CIBA untuk memenuhi proses bisnis error checking dan debugging apabila ada keluhan dari sivitas akademika.

Kata kunci: *Client Initiated Backchannel Authentication, Authorization Server, Relying Party, Consumption Device, Authentication Device*

IMPLEMENTATION OF CLIENT INITIATED BACKCHANNEL AUTHENTICATION (CIBA) SERVER ON TOP OF MYITS SINGLE SIGN-ON AUTHORIZATION SERVER

Name	: MUHAMMAD ADISTYA AZHAR
NRP	: 05111640000103
Department	: Informatics, Faculty of Electics-ITS
Supervisor I	: Rizky Januar Akbar, S.Kom., M.Eng.
Supervisor II	: Nurul Fajrin Ariyani, S.Kom., M.Sc.

Abstract

The rise of online banking and financial technology sector greatly depend on security mechanism to protect sensitive customer data. A new authentication and authorization protocol named Client Initiated Bakchannel Authentication (CIBA) is released to accommodate use cases where the Consumption Device, the device which is used to interace with the Relying Party/ Client Application, and the Authentication Device, the device which the user authenticates with the Authorization Server and grants consent, are separated physically, which are common in online bank customer service, and point of sales system.

In this final project, a CIBA server system is developed on top of an existing authorization library called Bshaffer OAuth 2.0, which runs Institut Teknologi Sepuluh Nopember's (ITS) Authorization Server, myITS SSO. This system is emphasized to support fundamental CIBA concerns such as client registration, authentication initiation, access token retrieval, and session binding.

Testing was conducted on each CIBA server endpoints using Postman being given correct and incorrect parameters. The test cases applied during testing were adopted from Financial Grade API CIBA (FAPI CIBA) conformance testing provided by OpenID Foundation. The server was able to comply accordingly to CIBA specification by responding with descriptive error messages in abnormal conditions, and continuing requests in the contrary.

The CIBA protocol in this final project is successfully implemented as of specifications without hampering existing protocols that are in myITS SSO Authorization Server, due to a decoupled library architecture. Client Applications that are registed to use myITS SSO have futher options to opt in to various protocols and act accordingly to its context. Therefore, use cases in authentication that requires devices to be separated physically can be done with CIBA. In this case, the help desk service run by Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI) can leverage CIBA to fulfill error checking and debugging remotely when a user report errors.

Keywords: *Client Initiated Backchannel Authentication, Authorization Server, Relying Party, Consumption Device, Authentication Device*

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Segala puji dan syukur kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga dapat di selesaikan tugas akhir ini yang berjudul "*Implementasi Protokol Client Initiated Backchannel Authentication (CIBA) Di Sisi Server Pada MyITS Single Sign-On*".

Terselesaikannya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan banyak pihak, oleh karena itu melalui lembar ini penulis ingin mengucapkan terima kasih dan penghormatan kepada:

1. Ayahanda tersayang Prahara Maghribi , Ibunda tersayang Ratna Rintaningrum, Adik tersayang Muhammad Farhan Dzulfikar, dan keluarga tercinta penulis yang selalu memberikan dukungan doa, moral, serta saran dalam berbagai bentuk agar dapat menyelesaikan tugas akhir ini.
2. Bapak Rizky Januar Akbar, S.Kom., M.Eng. selaku dosen pembimbing tugas akhir pertama yang telah memberikan bimbingan, motivasi, tambahan ilmu dan solusi pada setiap permasalahan atas kesulitan dalam pengerjaan tugas akhir ini.
3. Ibu Nurul Fajrin Ariyani, S.Kom., M.Sc. selaku dosen pembimbing tugas akhir kedua yang dengan sabar telah memberikan arahan selama penyusunan tugas akhir ini.
4. Bapak dan Ibu dosen departemen Teknik Informatika ITS yang telah mengajarkan ilmu pada penulis.
5. Bapak dan Ibu pegawai departemen Teknik Informatika ITS atas berbagai bantuan yang diberikan selama masa perkuliahan.
6. Ghifaroza Rahmadiana yang selalu menemani, mendukung, dan menyemangati selama pengerjaan tugas akhir ini.

7. Sahabat penulis Andika, Dewang, dan Bani yang selalu mengisi hari-hari menjadi sangat menyenangkan.
8. Teman-teman dari ITS Debate Society, Andri, Ghazy, dan Pitra yang telah memberi pandangan baru selama masa perkuliahan.
9. Teman-teman admin Laboratorium Rekayasa Perangkat Lunak yang memberi dukungan moral selama pengerjaan tugas akhir ini.
10. Seluruh mahasiswa Teknik Informatika ITS angkatan 2016 yang telah menjadi teman penulis selama menjalani masa kuliah di Teknik Informatika ITS.
11. Komunitas OpenID, Stackoverflow dan *open source* yang telah menyediakan referensi untuk membantu kelancaran tugas akhir ini.
12. Pihak-pihak yang lain yang tidak sempat penulis sebutkan, yang telah membantu kelancaran pengerjaan tugas akhir ini.

Penulis menyadari bahwa laporan tugas akhir ini masih jauh dari kata sempurna. Oleh karena itu dengan segala kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan penulis kedepannya. Penulis berharap tugas akhir ini bisa bermanfaat bagi semua pihak, khususnya sivitas akademika Teknik Informatika ITS, serta bagi agama, bangsa, dan negara.

Surabaya, Juni 2020

Muhammad Adistya Azhar

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

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

1.1 Latar Belakang

OpenID Connect (OIDC) 1.0 merupakan kerangka kerja yang berperan sebagai *identity layer* yang dibangun di atas protokol *OAuth 2.0* [1]. Fungsi *OAuth 2.0* adalah melakukan autorisasi pengguna agar mendapatkan hak akses data yang terproteksi [2]. OIDC memungkinkan klien, yakni aplikasi yang membutuhkan layanan OIDC untuk melakukan autentikasi identitas pengguna. Tujuan OIDC adalah memberi layanan autentikasi atau *login* untuk situs-situs yang dimiliki oleh suatu organisasi. Setiap kali pengguna membutuhkan *login* pada suatu situs, maka situs itu akan mengalihkan pengguna ke situs yang memiliki server OIDC dan dikembalikan ke situs awal ketika autentikasi berhasil. OIDC terdiri dari tiga alur untuk menangani autentikasi sebagai berikut:

a. *Authorization Code*
b. *Implicit*
c. *Hybrid*

OIDC telah mengeluarkan alur baru yang dinamakan *Client Initiated Backchannel Authentication* (CIBA). Perbedaan CIBA dengan ketiga alur di atas adalah sebagai berikut:

a. CIBA memiliki alur yang terpisah dimana perangkat tempat melakukannya autentikasi dan perangkat berjalannya aplikasi klien terpisah secara fisik. Sedangkan alur selain CIBA mengasumsikan kedua perangkat tersebut sama [3].

b. CIBA tidak membutuhkan *redirect* untuk mendapatkan *access token*. CIBA memiliki tiga standar yang terdefinisi untuk mendapatkan *access token*, yaitu *poll*, *ping*, dan *push* [3].

Alur CIBA memiliki empat entitas yang terdefinisi pada Tabel 1.1.

Tabel 1.1 Entitas Pada Protokol CIBA

Nama Entitas	Deskripsi
OpenID Provider (OP)	Server autorisasi yang mengimplementasi OIDC dan CIBA. Kata OP dan server autorisasi dapat digunakan secara bergantian karena memiliki makna yang sama.
Relying Party (RP)	Aplikasi klien yang menggunakan layanan CIBA. Kata RP dan aplikasi klien dapat digunakan secara bergantian karena memiliki makna yang sama.
Consumption Device (CD)	Perangkat yang digunakan untuk mengoperasikan dan menjalankan aplikasi klien.
Authentication Device (AD)	Perangkat yang digunakan untuk melakukan autentikasi dan autorisasi terhadap permintaan akses data yang terproteksi oleh RP. Pada umumnya, perangkat ini berupa *smartphone* dan dipegang oleh pengguna.

CIBA sangat cocok untuk kasus penggunaan dimana CD dan AD merupakan perangkat yang terpisah. Sebagai contoh pada saat mengonfirmasi identitas diri ke *customer service* bank melalui telepon. Pada umumnya, *customer service* membutuhkan data untuk memverifikasi bahwa orang yang berbicara dengannya

adalah pemegang akun yang sesungguhnya. Oleh karena itu, *customer service* selalu bertanya tentang nama, tempat dan tanggal lahir ibu kandung. Dengan adanya CIBA, *customer service* tidak perlu melakukan hal tersebut. Pengguna sebagai pemegang akun hanya perlu memberi autorisasi serta mengisi kredensial pada *smartphone* yang berperan sebagai AD, sehingga *customer service* dapat lanjut untuk mengakses akun bank pengguna menggunakan aplikasi internal *online banking* yang berperan sebagai RP. Alur CIBA ini dapat dikatan sebagai alur yang memiliki perangkat terpisah.

Dengan penjelasan diatas, maka pada tugas akhir ini akan dilakukan implementasi CIBA pada server autorisasi myITS SSO agar dapat mengakomodir kasus penggunaan dengan kebutuhan perangkat CD dan AD yang terpisah pada lingkunan Institut Teknologi Sepuluh Nopember (ITS). Literatur myITS SSO dapat dibaca pada buku tugas akhir "Implementasi Otentikasi SSO: Single Sign On dan Otorisasi Role Based Access Control Menggunakan Standar OpenID Connect" yang disusun oleh Kadek Winda Dwiastini [4].

1.2 Rumusan Masalah

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

1. Bagaimana mengimplementasi prosedur pendaftaran aplikasi klien agar dapat menggunakan alur CIBA?
2. Bagaimana mengimplementasi alur CIBA pada server autorisasi myITS Single Sign-On?
3. Bagaimana cara mengimplementasi pengambilan *access token* menggunakan mode *poll*, *ping*, dan *push*?
4. Bagaimana mengimplementasi *session binding* antara CD dan AD?

1.3 Batasan Masalah

Permasalahan pada tugas akhir ini memiliki beberapa batasan, diantaranya sebagai berikut:

1. Platform yang digunakan adalah aplikasi MyITS Single Sign-On yang berbasis bahasa pemrograman PHP.
2. Library OAuth 2.0 Server yang digunakan adalah Bshaffer versi 1.10.0.
3. Spesifikasi CIBA menganut pada versi “Core 1.0 draft-03”.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Mengetahui mekanisme implementasi pendaftaran aplikasi klien agar dapat menggunakan *flow* CIBA.
2. Mengetahui mekanisme implementasi *flow* CIBA pada myITS SSO dengan metode pengambilan token *poll*, *ping*, dan *push*.
3. Mengetahui mekanisme implementasi *session binding* antara CD dan AD.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini adalah mempermudah operasional myITS Single Sign-On dengan cara mengakomodir kasus penggunaan dimana *Consumption Device* dan *Authentication Device* harus terpisah.

1.6 Metodologi

Tahap yang dilakukan untuk menyelesaikan tugas akhir ini adalah sebagai berikut:

1. **Penyusunan Proposal Tugas Akhir**

   Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan

masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. **Studi literatur**

   Tahap ini merupakan tahap pengumpulan informasi dan pembelajaran yang akan digunakan pada tugas akhir ini. Studi literatur meliputi diskusi dan pemahaman terkait dengan tugas akhir ini, diantaranya mengenai:

   - *OAuth 2.0.*
   - *OpenID Connect*
   - Protokol CIBA.
   - *JSON Web Token*
   - Prinsip Pemrograman Berbasis Obyek
   - Phalcon sebagai kerangka kerja pengembangan *web application.*
   - REST API
   - SQL Server sebagai basis data utama.
   - Firebase Cloud Messaging sebagai *broker* untuk mengirim *push notification.*

3. **Analisis dan Perancangan**

   Pada tahap ini dilakukan perancangan terhadap perangkat lunak yang akan dikembangkan. Tahapan-tahapan yang dilakukan selama analisis dan perancangan sistem adalah sebagai berikut.

   - Menentukan lingkup permasalahan
   - Mendefinisikan sistem secara umum

- Membuat pemodelan sistem menggunakam diagram kasus penggunaan, diagram aktivitas, dan diagram sekuensial
- Menentukan arsitektur dan kelas
- Membuat rancangan antarmuka perangkat lunak

4. **Implementasi**
   Pada tahap ini dilakukan pembuatan protokol CIBA yang berdasar pada perancangan sistem, dan mengintegrasikan dengan perangkat lunak myITS SSO.

5. **Uji Coba dan Evaluasi**

   Pada tahap ini dilakukan uji coba terhadap protokol baru yang telah dikembangkan pada server autorisasi myITS SSO, dengan menggunakan metode pengujian terhadap *input* serta *output* berdasarkan skenario. Berikut ini adalah kriteria pengujian yang akan dilakukan:
   - Pengujian CIBA Menggunakan Mode Token *Push*
   - Pengujian CIBA Menggunakan Mode Token *Poll*
   - Pengujian CIBA Menggunakan Mode Token *Ping*
   - Pengujian Mendapatkan *Protected Resource*
   - Pengujian Autentikasi Berperan Sebagai Akun Yang Berbeda

6. **Penyusunan Buku Tugas Akhir**
   Pada tahap ini dilakukan penyusunan buku yang berisi dokumentasi pembuatan serta 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 dan sistematika penulisan tugas akhir.

2. **Bab II. Tinjauan Pustaka**
   Bab tinjauan pustaka berisi penjelasan mengenai dasar teori yang mendukung pengerjaan tugas akhir.
3. **Bab III. Analisis dan Perancangan**
   Bab ini berisi tentang desain sistem, rancangan basis data, diagram kasus penggunaan, diagram aktivitas dan rancangan, diagram sekuensial, antarmuka pengguna.
4. **Bab IV. Implementasi**
   Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa tampilan antarmuka yang telah dibuat dan dapat berfungsi untuk mengakomodir kebutuhan fungsional yang ada.
5. **Bab V. Uji Coba dan Evaluasi**
   Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.
6. **Bab VI. Kesimpulan dan Saran**
   Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab tinjauan pustaka berisi penjelasan teori yang berkaitan dengan implementasi perangkat lunak. Penjelasan tersebut bertujuan untuk memberikan gambaran mengenai sistem yang akan dibangun dan berguna sebagai pendukung dalam pengembangan perangkat lunak.

2.1 Penelitian Terkait

Pada sub bab ini akan dibahas tentang penelitian terkait yang menjadi rujukan tugas akhir ini.

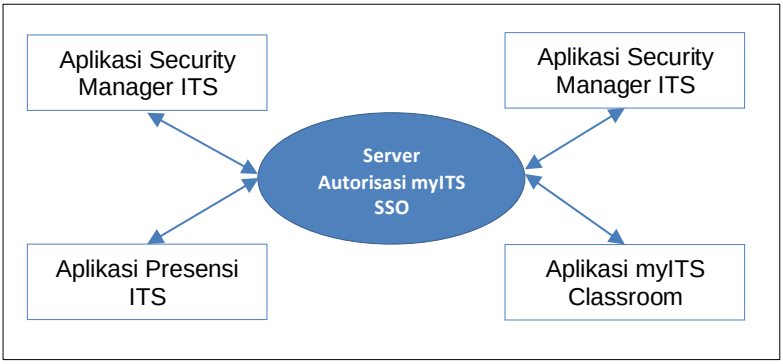
Server Autorisasi MyITS Single Sign-On

Sistem *Single Sign-On* (SSO) merupakan sistem yang dapat mengakomodir proses bisnis autentikasi menggunakan satu akun pada banyak aplikasi. Aplikasi yang memanfaatkan SSO dapat menghindari proses pendaftaran akun sehingga dapat terhindar dari manajemen akun yang pada umumnya merupakan hal yang rumit. Tanpa menggunakan SSO, aplikasi harus mengatur akun pengguna, dan dapat terjadi duplikasi akun antar aplikasi lain karena pengguna diharuskan untuk mendaftar pada setiap aplikasi yang ingin digunakan.

Institut Teknologi Sepuluh Nopember (ITS) memiliki SSO yang dinamakan dengan myITS SSO. Aplikasi yang berjalan pada lingkungan ITS dapat memanfaatkan myITS SSO sehingga pengguna seperti sivitas akademika dapat terhubung pada masing-masing aplikasi, tanpa harus melakukan pendaftaran. Dapat dilihat pada Gambar 2.1 terdapat aplikasi yang ada pada lingkungan ITS menggunakan myITS SSO.

Protokol yang digunakan pada myITS SSO adalah *OpenID Connect* (OIDC) *Authorization Code*. Protokol ini menggunakan mekanisme *redirect* yaitu ketika aplikasi ingin melakukan autentikasi, aplikasi tersebut akan dialihkan ke halaman myITS SSO untuk mengisi *username* dan *password*. Apabila proses autentikasi berhasil, maka myITS SSO akan mengalihkan kembali

pengguna ke halaman aplikasi awal. Protokol OIDC dibahas lebih lanjut pada subbab 2.3.



Gambar 2.1 Komunikasi Aplikasi Klien dan Server Autorisasi

Perbandingan Protokol *Authorization Code* dan CIBA

Pada subbab ini akan dilakukan perbandingan antara protokol OIDC *Authorization Code* yang digunakan pada server autorisasi myITS SSO, dan CIBA. Atribut yang dibandingkan ada lima, yaitu: mode token yang dimiliki oleh protokol, mekanisme alur autentikasi, entitas yang bekerja pada protokol, tipe *Consumption Device*, dan tipe *Relying Party*. Perbandingan kedua protokol ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Perbandingan Protokol *Authorization Code* Dengan CIBA

Atribut	*Authorization Code*	CIBA
Mode Token	**1** – Request token.	**3** – *Push*, *ping*, dan *poll*.
Mekanisme	*Redirect* antar situs.	Komunikasi langsung tanpa *redirect*.
Entitas	**4** – *Authorization Server*, *Consumption Device*,	**4** – *Authorization Server*, *Consumption Device*,

	Authentication Device dan *Relying Party*. *Authentication Device* dan *Consumption Device* merupakan perangkat yang sama.	*Authentication Device* dan *Relying Party*. *Authentication Device* dan *Consumption Device* merupakan perangkat yang terpisah.
Tipe Perangkat (*Consumption Device*)	Merupakan *device* yang tidak *input constrained*, harus dapat dioperasikan dengan mudah oleh pengguna.	Dapat merupakan *device* yang *input constrained*.
Tipe Aplikasi (*Relying Party*)	Harus dioperasikan oleh pengguna yang mengendalikan *Authentication Device*.	Dapat dioperasikan oleh pengguna lain/ pengguna yang berbeda dari yang mengendalikan *Authentication Device*.

2.2 OAuth 2.0

OAuth 2.0 merupakan kerangka kerja yang berperan sebagai lapisan autorisasi untuk memberi akses data yang terproteksi ke aplikasi klien eksternal [5] [6]. Sebagai contoh suatu aplikasi klien dapat mengakses data akun Facebook seperti *friend list*, *date of birth*, *address*, dan *timeline*. Aplikasi klien dapat mengakses data tersebut dengan cara pengguna atau pemilik akun Facebook memberi persetujuan melalui server *OAuth 2.0* yang disediakan oleh Facebook. *OAuth 2.0* mendefinisikan empat entitas yang berperan dalam proses autorisasi [6]:

1. *Resource Owner*
   Resource owner adalah pemilik data yang terproteksi [7]. Pada umumnya, *resource owner* adalah pengguna dan ialah yang memberikan persetujuan ke aplikasi klien untuk

mengakses data yang dimiliki. Namun, akses data yang diberikan ke aplikasi klien belum tentu sepenuhnya. Akses data dibatasi dengan *scope* yang ter-registrasi. Sebagai contoh aplikasi klien hanya dapat *read* data pribadi, tetapi tidak bisa *write*.

2. *Client Application*
   Client application atau aplikasi klien adalah aplikasi eksternal yang membutuhkan data dari *resource server* [7].
   Sebelum mendapatkan akses data, *resource owner* harus menyetujui permintaan akses tersebut.
3. *Resource Server*
   Resource server menyimpan data-data pengguna yang bersifat tidak terbuka untuk umum [7].
4. *Authorization Server*
   Authorization server atau server autorisasi berperan memverifikasi identitas pengguna dan mengeluarkan token [7].

Agar aplikasi klien dapat mengakses data yang terproteksi, setiap *request* harus disertakan token yang disebut *access token*. Metode untuk mendapatkan token bervariasi. *OAuth 2.0* mendefinisikannya sebagai *grant type* atau alur. *OAuth 2.0* memiliki empat *grant type* yang umum digunakan. *Grant type* tersebut terdiri dari *Authorization Code*, *Implicit*, *Resource Owner Password Credentials*, dan *Client Credentials* [8]. Yang paling umum digunakan serta yang diimplementasikan di myITS SSO adalah *Authorization Code*. Alur *Authorization Code* dapat dilihat pada Gambar 2.4. Penjelasan alur kerja *Authorization Code* adalah sebagai berikut, urut berdasarkan nomor:

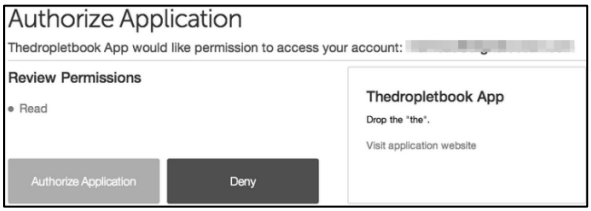
1. *User Authorization Request*
   Pada tahap pertama, aplikasi klien akan mengirim *request* autorisasi ke server autorisasi untuk memulai proses autorisasi. Tahap ini bertujuan agar server autorisasi mengetahui:
   - Identitas aplikasi klien

- Metode pengiriman token
- Hak akses aplikasi klien terhadap data yang tersimpan di *resource server*
- Metode mengembalikan pengguna ke halaman aplikasi klien

2. *User Authorizes Application*

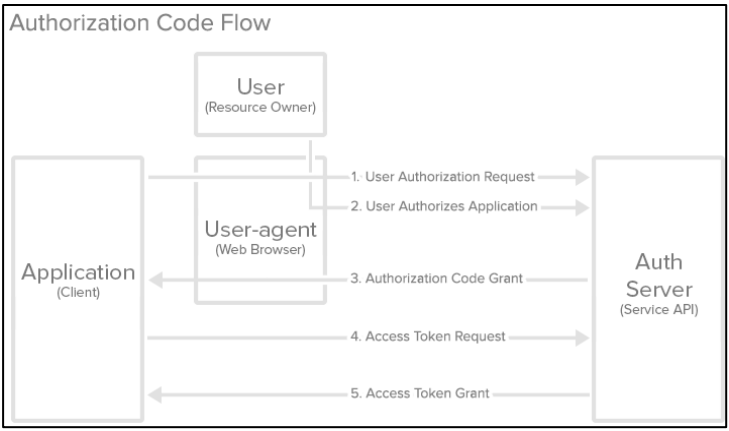
   Pada tahap kedua, pengguna akan diberikan formulir *login* yang harus diisi sesuai kredensial agar server autorisasi dapat memastikan identifikasi pengguna. Formulir *login* hanya berlaku jika pengguna belum dalam kondisi terautentikasi seperti yang dicontohkan pada Gambar 2.2. Setelah itu, akan muncul *dialog* untuk meminta persetujuan dan izin dari pengguna bahwa aplikasi klien akan mengakses data pengguna yang tersimpan di *resource server*. Isi dari *dialog* adalah nama aplikasi klien, dan *scope* data yang dapat diakses. Pengguna dapat mengizinkan atau menolak permintaan akses. Dapat dilihat pada Gambar 2.3 aplikasi klien yang bernama "Thedropletbook App" akan mengakses data pengguna dengan *scope read*.

Gambar 2.2 Formulir *Login*



Gambar 2.3 *Dialog* Aplikasi Klien Meminta Izin Akses Data

3. *Authorization Code Grant*
   Jika pengguna telah mengizinkan aplikasi klien untuk mengakses data, maka server autorisasi akan mengirim *authorization code* dengan cara *redirect* ke *Uniform Resource Identifier* (URI) aplikasi klien. *Authorization code* akan digunakan untuk mendapatkan *access token*.
4. *Access Token Request*
   Tahap ini akan meminta *access token* ke server autorisasi.
5. *Access Token Grant*
   Server autorisasi akan membalas dengan *access token*. *Access token* memiliki waktu kedaluwarsa sehingga dapat hangus di masa akan datang.



Gambar 2.4 Alur *Authorization Code* [5]

2.3 OpenID Connect

Permasalahan yang muncul pada kerangka kerja *OAuth 2.0* adalah tidak terdefinisinya prosedur untuk melakukan autentikasi untuk membuktikan bahwa pengguna adalah pemilik akun yang sesungguhnya. Alur yang tersedia pada *OAuth 2.0* dikhususkan untuk autorisasi atau memberi hak akses [2].

Untuk menanggulangi masalah tersebut, muncul protokol *OpenID Connect* (OIDC). OIDC merupakan *extension* dari *OAuth 2.0* dengan alur yang serupa [1]. Aplikasi klien dapat memverifikasi identitas pengguna berdasarkan hasil autentikasi yang telah dilaksanakan oleh server autorisasi, dan mendapatkan informasi dasar atau *user info* akun pengguna berdasarkan *scope* yang ditentukan.

Karakterisik yang membuat OIDC unik adalah konsep *ID Token* yang dimilikinya. *ID Token* didefinisikan sebagai kartu identitas yang telah dilakukan *digital signing* sehingga memiliki format JSON Web Token (JWT) [9].

Ketika pengguna berhasil melakukan autentikasi di server autorisasi, maka pengguna akan mendapatkan *ID Token* dengan format yang dapat dilihat pada Tabel 2.2 [9].

Tabel 2.2 *Claim* Pada *ID Token*

Nama Properti/ *Claim*	Deskripsi	Wajib
iss	*Issued Identifier* mendefinisikan identifikasi server autorisasi yang memberikan *ID Token*	YA
sub	Subject Identifier mendefinisikan identifikasi akun pengguna	YA
aud	*Audience* mendefinisikan identifikasi aplikasi klien yang dikhususkan untuk mendapatkan *ID Token*	YA
exp	*Expiration time* mendefinisikan waktu kedaluwarsa agar token tidak diproses lanjut	YA
iat	*Issued at* mendefinisikan waktu *ID Token* diberikan	YA
auth_time	*Authentication time* mendefinisikan waktu pengguna melakukan autentikasi	TIDAK
nonce	*Nonce* mendefinisikan nilai untuk mengikat *session* aplikasi klien dengan *ID Token* agar tidak terjadi permintaan autentikasi yang berulang-ulang	TIDAK
acr	*Authentication Context Class Reference* mendefinisikan aturan bisnis yang harus dipenuhi ketika melakukan autentikasi	TIDAK
amr	*Authentication Methods References* mendefinisikan metode autentifikasi yang dilakukan.	TIDAK
azp	*Authorized party* mendefinisikan identifikasi aplikasi klien dan hanya digunakan ketika *audience* berbeda dengan *authorized party*.	TIDAK

ID Token dapat terdiri dari *claim* lain yang telah terdefinisi pada spesifikasi *OpenID Connect Core 1.0* [9]. Contoh *ID Token* dapat dilihat pada Kode Sumber 2.1.

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjhM
bDh2UlU3UmxTc2ZlcXZrZGVueG5YZjlnRlNwUXk0b3d
waVY4dlRFbE0ifQ.

eyJpc3MiOiJodHRwczpcL1wvZGV2LW15Lml0cy5hYy5p
ZCIsInN1YiI6IjIwREEyNjFDLUNBMkUtNDMyRS04Mj
UzLUUzN0JEQjNGNkNBNyIsImF1ZCI6IkIzMEU4MjlFL
TNFMUUtNEE5NS05ODRFLTBBNzlENTY2QTIyMSIsI
mlhdCI6MTU4ODQzMzU4MywiZXhwIjoxNTg4NDM3M
TgzLCJhdXRoX3RpbWUiOjE1ODg0MzM1ODMsIm5vb
mNlIjoiNDc5NThlNTVjMGE0OWNmZjQ4MzIzNGM1Y
2QyMjA2NmYifQ.

u9VZh6Rhk_iKzu8oTUD2KCN8GDdtus1aNmP_I96JtU35
RjA3yG76pzdkx79-kND8Rmm0-0PT70d7UEx0vObf-
imZfe0Ow8-
gUOfutO0SBBA4AOKj6JhmA2xVUmMrFaIp8atw3mp0J
P6AYgoffd_IEeyfzT5S33We0Z7d804TUl8EegC_--
PbUF8YZSE8h1DwnyRhQstVv03ESYvsIkGv1wy3aaOCQ
ph-D4LUIL3ermnSuilmbCwNIuELqPnJf-
OYV61M92vFa5kxCcQs7u6sM9-
_n4bNUV6qQwBwXdRUKphEyqpo8Xouev6JoESqS_XM
CDJ9jaBzLUS3tNrv3q4mvA

Header

Payload

Signature

Kode Sumber 2.1 Contoh *ID Token*

ID Token pada Kode Sumber 2.1 memiliki bentuk *decoded* yang dapat dilihat pada Kode Sumber 2.2 and Kode Sumber 2.3. *ID Token* ini memiliki bentuk JWT yang menggunakan algoritma RS256 dengan suatu *key id* yang dapat digunakan untuk melakukan verifikasi *signature*. *Public key* dengan *key id "8Ll8vRU7RlSsfeqvkdenxnXf9gFSpQy4owpiV8vTElM"* dapat diambil dari *endpoint* server autorisasi myITS SSO *"https://dev-my.its.ac.id/.well-known/jwks.json"*. Pihak yang mengeluarkan *ID Token* ini adalah server autorisasi myITS SSO dengan URI "*https://dev-my.its.ac.id*". *ID Token* ini ditujukan ke aplikasi klien dengan identifikasi "*B30E829E-3E1E-4A95-984E-0A79D566A221*" dan identifikasi akun pengguna "*20DA261C-*

CA2E-432E-8253-E37BDB3F6CA7". *ID Token* ini disusun pada tanggal 02 Mei 2020 pukul 22:33:03 atau "*1588437183*" dalam *unix timestamp*.


```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid":
"8Ll8vRU7RlSsfeqvkdenxnXf9gFSpQy4owpiV8vTElM"
}
```


Kode Sumber 2.2 Header *ID Token*


```
{
  "iss": "https://dev-my.its.ac.id",
  "sub": "20DA261C-CA2E-432E-8253-E37BDB3F6CA7",
  "aud": "B30E829E-3E1E-4A95-984E-0A79D566A221",
  "iat": 1588433583,
  "exp": 1588437183,
  "auth_time": 1588433583,
  "nonce": "47958e55c0a49cff483234c5cd22066f"
}
```


Kode Sumber 2.3 *Payload ID Token*

ID Token dapat dimanfaatkan oleh aplikasi klien dalam beberapa skenario seperti [10]:

- Mengatur *session* aplikasi klien untuk membedakan keadaan terautentikasi dan tidak terautentikasi
- Memperan akun pengguna lain dengan memodifikasi *claim* yang terdapat pada *ID Token* [11]

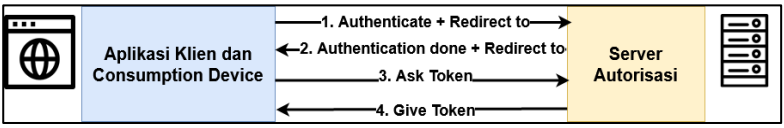
2.4 Client Initiated Backchannel Authentication

OpenID Connect memungkinkan RP untuk melakukan autentikasi pengguna dari berbagai macam aplikasi klien. RP dapat memulai alur dengan mengirim *request* autorisasi ke *authorization*

endpoint dengan interaksi pengguna pada CD [12]. Interaksi ini dapat diartikan bahwa pengguna mengisi kredensial akun seperti *username* dan *password* pada browser yang bertindak sebagai AD. Mekanisme ini membutuhkan *redirect* antara aplikasi klien dan server autorisasi. Alur ini dinamakan sebagai alur autentikasi tradisional. Akan tetapi, ada berbagai kasus penggunaan yang tidak mengakomodir alur dimana kasus penggunaan memiliki CD dan AD yang terpisah sehingga mekanisme *redirection* tidak dapat dilakukan.

Sebagai contoh kasus penggunaan yang tidak dapat diakomodir oleh alur autentikasi tradisional adalah pada sistem *Point of Sales* (POS). Terdapat konsumen yang ingin membayar belanjaan pada perangkat POS di kasir. Konsumen akan *scan* kode QR atau mengisi nomor *handphone* pada perangkat POS agar aplikasi di perangkat tersebut dapat mengidentifikasi akun konsumen. Perangkat POS tersebut bertindak sebagai CD dan aplikasi yang berjalan pada perangkat POS adalah RP atau aplikasi klien. Untuk dapat melanjutkan pembayaran, konsumen harus memberi akses pada aplikasi di AD. Apabila hak akses sudah diberikan, maka pembayaran dapat diselesaikan. Ilustrasi sistem POS menggunakan CIBA dapat dilihat pada Gambar 2.6.

Dapat dilihat pada Gambar 2.5 alur autentikasi yang membutuhkan *redirect* karena aplikasi klien/ CD merupakan perangkat yang sama dengan AD.



Gambar 2.5 *Redirect* Antar Situs Pada Alur Tradisional

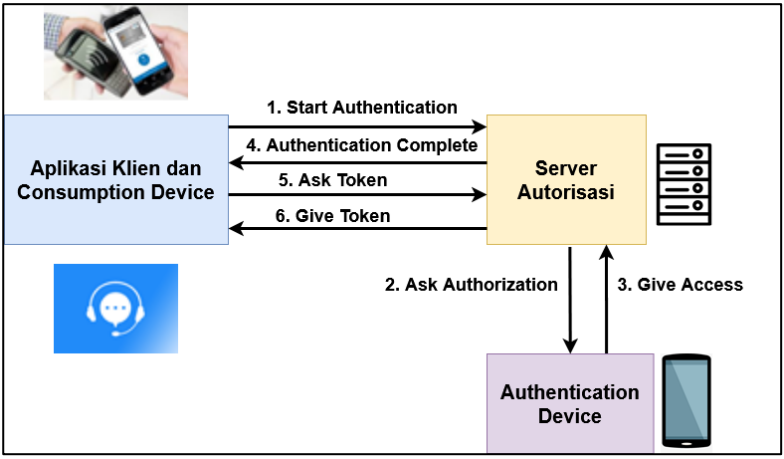
Client Initiated Backchannel Authentication (CIBA) adalah alur autentikasi baru yang memiliki entitas terpisah. Flow CIBA memiliki empat entitas yang terdefinisi pada Tabel 2.3 [13].

Tabel 2.3 Entitas Pada Protokol CIBA

Nama Entitas	Deskripsi
OpenID Provider (OP)	Server autorisasi yang mengimplementasi OIDC dan CIBA. Kata OP dan server autorisasi dapat digunakan secara bergantian karena memiliki makna yang sama [13].
Relying Party (RP)	Aplikasi klien yang menggunakan layanan CIBA. Kata RP dan aplikasi klien dapat digunakan secara bergantian karena memiliki makna yang sama [13].
Consumption Device (CD)	Perangkat yang digunakan untuk mengoperasikan dan menjalankan aplikasi klien [13].
Authentication Device (AD)	Perangkat yang digunakan untuk melakukan autentikasi dan autorisasi terhadap permintaan akses data yang terproteksi oleh RP. Pada umumnya, perangkat ini berupa *smartphone* dan dipegang oleh pengguna [13].

Pada alur CIBA, RP mendapatkan identifikasi akun pengguna yang dapat berupa *user id* atau *email* agar dapat menginisiasi alur autentikasi. Inisiasi alur autentikasi dapat dilakukan tanpa interaksi pengguna yang menggunakan CD. Alasan CIBA dinamakan sebagai *decoupled flow* adalah karena proses autentikasi dan autorisasi tidak perlu melakukan *redirect*, seperti pada umumnya pada alur *Authorization Code* dan *Implicit*. Proses *redirect* ini dibutuhkan untuk mendapatkan *access token* dari OP. *Redirect* dilakukan karena CD dan AD merupakan perangkat yang sama, dan tidak terpisah. Namun pada CIBA, AD dan CD dapat berupa perangkat yang berbeda dan terpisah secara fisik. Oleh karena itu pengambilan *access token* dapat dilakukan

menggunakan 3 mode yang tersedia pada CIBA, yaitu: *poll*, *ping*, dan *push* [14]. Pengambilan *access token* dilakukan ketika pengguna pada AD telah melakukan autorisasi atau memberikan hak akses. *Access token* digunakan oleh RP agar memungkinkan untuk mendapatkan data yang terproteksi. Alur CIBA ini dapat dilihat pada Gambar 2.6.



Gambar 2.6 Alur Autentikasi Dan Autorisasi Dengan Perangkat Terpisah

2.4.1 Pendaftaran Aplikasi Klien dan Server Autorisasi

Aplikasi klien yang ingin menggunakan protokol CIBA harus melakukan pendaftaran pada server autorisasi. Pendaftaran adalah proses pencatatan aplikasi klien oleh server autorisasi agar server autorisasi dapat mengidentifikasi aplikasi klien tersebut [3]. Metadata yang harus diberikan oleh aplikasi klien dapat dilihat pada Tabel 2.4.

Tabel 2.4 Metadata Pendaftaran Aplikasi Klien

Nama Metadata	Deskripsi
backchannel_token_delivery_mode	**Mode token yang ingin digunakan. Salah satu nilai dari: *poll, ping,* atau *push*.**
backchannel_client_notification_endpoint	***Endpoint* aplikasi klien untuk menerima notifikasi dan token.**
backchannel_authentication_request_signing_alg	**Algoritma yang digunakan oleh aplikasi klien untuk melakukan *digital signing* pada *request* autentikasi.**
backchannel_user_code_parameter	**Menentukan penggunaan *user code* pada aplikasi klien.**

Sedangkan, agar server autorisasi dapat mendukung protokol CIBA [3], dapat menggunakan metadata yang dapat dilihat pada Tabel 2.5.

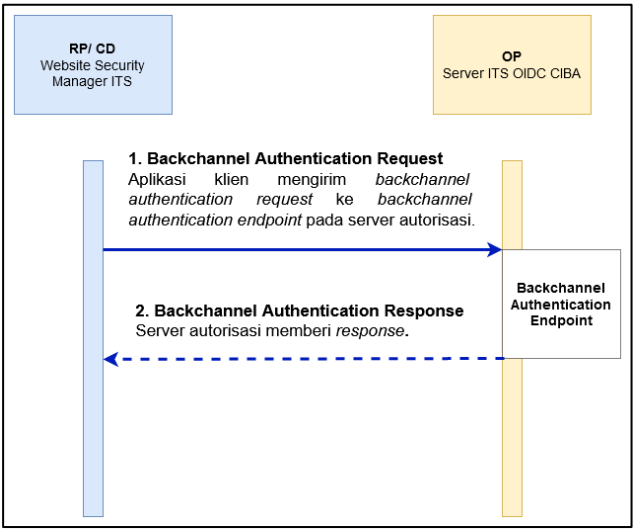
Tabel 2.5 Metadata Pendaftaran Server Autorisasi

Nama Metadata	Deskripsi
backchannel_token_delivery_modes_supported	Mode token yang didukung oleh server autorisasi. Mengandung setidaknya satu diantara *poll*, *ping*, dan *push*.
backchannel_authentication_endpoint	URI autentikasi CIBA.
backchannel_authentication_request_signing_alg_values_supported	Kumpulan algoritma yang digunakan untuk *signed request*.

backchannel_user_code_parameter_supported	Menentukan penggunaan *user code* pada server autorisasi.

2.4.2 Inisiasi *Request* Autentikasi

Pada tahap pertama, RP akan mengirim *backchannel authentication request* ke *backchannel authentication endpoint* yang dimiliki oleh OP. Pada Gambar 2.7 dapat dilihat bahwa ada aplikasi klien Security Manager yang mengirim *request* autentikasi ke Server Autorisasi ITS.



Gambar 2.7 Aplikasi Klien Mengirim *Request* Autentikasi

Pengiriman *request* memiliki parameter yang berbeda tergantung dengan mode token yang digunakan. Berikut adalah contoh *backchannel authentication request* untuk setiap mode pengambilan token.

2.4.2.1 *Request* Autentikasi Dengan Aplikasi Klien Yang Terdaftar Menggunakan Mode Token *Poll*

Pada *request* ini terdapat tujuh paramater yang bisa dikirim yang dapat dilihat pada Tabel 2.6.

Tabel 2.6 Parameter *Request* Autentikasi Menggunakan Mode *Poll*

Nama Parameter	Deskripsi Parameter	Wajib
scope	Hak akses yang akan dimiliki oleh RP.	YA
login_hint_token	Berisi *token* yang mengidentifikasi *end-user* yang ingin diautentikasi.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).
id_token_hint	Berisi *token* yang sebelumnya didapatkan dari OP yang mengidentifikasi *end-user* yang ingin diautentikasi.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).
login_hint	Berisi identifikasi *end-user* yang ingin diautentikasi. Nilai dapat berupa *email address*, *phone number*, atau *account ID*.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).
binding_message	Pesan atau kode yang akan dimunculkan pada CD dan AD untuk mengikat *session*	TIDAK
user_code	Kode rahasia yang dimiliki oleh pengguna.	TIDAK
requested_expiry	Nilai untuk memberi jangka kedaluwarsa penggunaan *authentication request id*.	TIDAK

2.4.2.2 *Request* Autentikasi Dengan Aplikasi Klien Yang Terdaftar Menggunakan Mode *Ping* dan *Push*

Pada *request* ini terdapat delapan paramater yang bisa dikirim yang dapat dilihat pada Tabel 2.7. Aplikasi klien yang menggunakan *mode* token *ping* dan *push* harus menyertakan *client notification token* sebagai bentuk keamanan [3]. Aturan tersebut bermanfaat untuk mencegah komunikasi yang berasal dari server autorisasi asing. Penjelasan *client notification token* akan diperdalam pada subbab 2.4.4.2.

Tabel 2.7 Parameter *Request* Autentikasi Menggunakan Mode *Ping* Dan *Push*

Nama Parameter	**Deskripsi Parameter**	**Wajib**
scope	Hak akses yang akan dimiliki oleh RP	YA
client_notification_token	Berisi *token* yang disusun oleh RP yang akan digunakan oleh OP sebagai *bearer token* ketika OP mengeksekus*i notification endpoint*.	YA
login_hint_token	Berisi *token* yang mengidentifikasi *end-user* yang ingin diautentikasi.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).
id_token_hint	Berisi *token* yang sebelumnya didapatkan dari OP yang mengidentifikasi *end-user* yang ingin diautentikasi.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).

Nama Parameter	Deskripsi Parameter	Wajib
login_hint	Berisi identifikasi *end-user* yang ingin diautentikasi. Nilai dapat berupa *email address*, *phone number*, atau *account ID*.	YA (Pilih salah satu antara login_hint_token, id_token_hint, login_hint).
binding_message	Pesan atau kode yang akan dimunculkan pada CD dan AD untuk mengikat *session.*	TIDAK
user_code	Kode rahasia yang dimiliki oleh pengguna.	TIDAK
requested_expiry	Nilai untuk memberi jangka kedaluwarsa penggunaan *authentication request id.*	TIDAK

2.4.2.3 Validasi *Request* Autentikasi

Request autentikasi yang berasal dari aplikasi klien harus divalidasi oleh server autorisasi [3]. Langkah validasi yang dilakukan oleh server autorisasi dapat dilihat pada Tabel 2.8.

Tabel 2.8 Langkah Validasi *Request* Autentikasi

Langkah	Deskripsi
1	Server autorisasi melakukan autentikasi aplikasi klien menggunakan metode autentikasi yang sudah terdaftar. Contoh: *Client Secret Basic*, *Client Secret Post*, dan *Client Secret JWT*.
2	Jika merupakan *signed request* maka JWT yang terdapat pada parameter *request* harus divalidasi.

3	Memastikan bahwa setiap parameter yang wajib terpenuhi.
4	Memastikan bahwa akun pengguna yang disertakan pada parameter *hint* dapat teridentifikasi.
5	Jika *hint* tidak valid, maka harus mengembalikan *response* dengan *error unknown_user_id*.
6	Memastikan bahwa setiap parameter memiliki format yang sesuai.
7	Mengabaikan parameter yang tidak diketahui atau tidak sesuai dengan spesifikasi.

Jika ditemukan *error* saat proses validasi, maka server autorisasi mengembalikan *response* dengan format yang dapat dilihat pada Tabel 2.9 [3].

Tabel 2.9 Format *Response Error* Autentikasi

Nama Parameter	Deskripsi	Wajib
error	Kode *error* yang mendefinisikan *error* yang terjadi.	YA
error_description	Deskripsi *error* yang terjadi.	TIDAK
error_uri	*Link* yang dapat digunakan oleh *developer* untuk melihat penjelasan *error* agar dapat menentukan langkah untuk memperbaiki *error* yang terjadi.	TIDAK

Kode *error* yang digunakan dapat dilihat pada Tabel 2.10 [3].

Tabel 2.10 Kode *Error* Autentikasi

Kode	Deskripsi	Status HTTP

invalid_request	Ada parameter yang tidak lengkap.	400
invalid_scope	*Scope* yang diminta tidak valid.	400
expired_login_hint_token	Token telah kedaluwarsa.	400
unknown_user_id	Akun pengguna tidak ditemukan.	400
unauthorized_client	Aplikasi klien tidak berhak menggunakan protokol tersebut.	400
missing_user_code	Parameter *user code* tidak lengkap.	400
invalid_user_code	Parameter *user code* salah.	400
invalid_binding_message	Parameter *binding message* salah.	400
invalid_client	Autentikasi aplikasi klien tidak berhasil.	401
access_denied	Server autorisasi tidak mengizinkan aplikasi klien untuk menggunakan protokol tersebut.	403

Jika tidak terjadi *error* pada proses validasi, maka server autorisasi mengembalikan *response* dengan format yang dapat dilihat pada Tabel 2.11 [3].

Tabel 2.11 Format *Response* Autentikasi Yang Berhasil

Nama Parameter	Deskripsi	Wajib
auth_req_id	Identifikasi yang bersifat unik untuk merepresentasikan sesi *request* autentikasi.	YA

expires_in	Waktu kedaluwarsa *auth_req_id*.	YA
interval	Minimal jangka waktu dalam detik untuk melakukan token *polling*. Khusus mode token *poll.*	TIDAK

2.4.2.4 *Signed Request* Autentikasi

Bentuk *request* autentikasi yang telah dijelaskan pada subbab 2.4.2.1 dan 2.4.2.2 adalah *non signed request*. *Non signed request* memiliki arti bahwa parameter yang terdapat pada *request* tidak melalui proses *digital signing*. Sebagai keamanan tambahan, CIBA memiliki spesifikasi *signed request* yang digunakan untuk autentikasi [3].

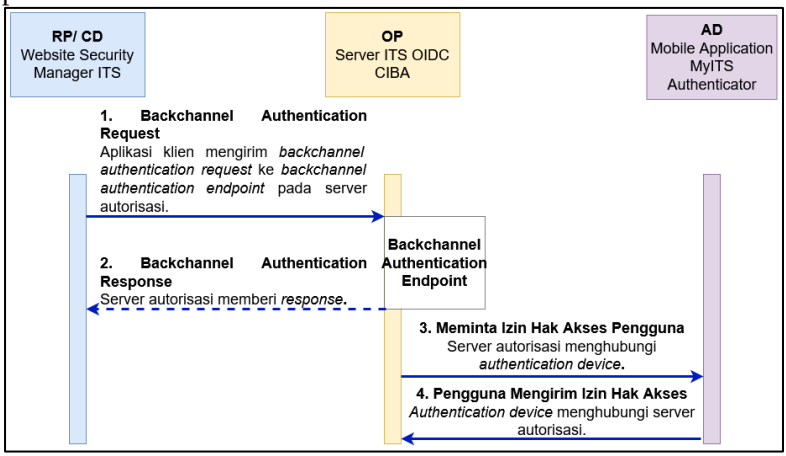
Pada *signed request*, parameter-parameter akan melalui proses *digital signing* yang akan menghasilkan JWT dengan serialisasi *JSON Web Signature* (JWS). Tidak ada penghapusan parameter yang telah terdefinisi pada Tabel 2.6 dan Tabel 2.7. Namun parameter tersebut akan dijadikan sebagai *payload* untuk menghasilkan JWT. Jadi, JWT yang dihasilkan akan memili *claims* dengan parameter pada Tabel 2.6, Tabel 2.7, dan Tabel 2.12.

Tabel 2.12 *Claims* Tambahan *Signed Request*

Nama *Claim*	Deskripsi	Wajib
aud	*Audience* mendefinisikan identifikasi server autorisasi.	YA
iss	*Issuer* mendefinisikan identifikasi aplikasi klien menggunakan *client_id*.	YA
exp	Waktu kedaluwarsa JWT.	YA
iat	Waktu saat JWT disusun.	YA
nbf	*Not before at* mendefiniskan waktu saat JWT bisa diproses.	YA
jti	Identifikasi unik untuk *request* ini.	YA

2.4.3 Pemberian Hak Akses Oleh Pengguna

Jika validasi *request* autentikasi berhasil seperti yang telah dijelaskan pada subbab 2.4.2.3, OP akan mengirim permintaan izin hak akses ke AD [3]. Proses autorisasi akan didelegasikan ke AD. Pengguna melakukan autorisasi di AD dan hasil autorisasi dikirimkan ke OP. Pengguna dapat pilihan untuk mengizinkan atau tidak mengizinkan aplikasi klien. Proses autorisasi dapat dilihat pada Gambar 2.8.



Gambar 2.8 Autorisasi Dilakukan Pada Perangkat *Authentication Device*

2.4.4 Pengambilan Token Oleh Aplikasi Klien

Hak akses yang diberikan oleh pengguna memungkinkan RP untuk melakukan pengambilan token ke OP.

CIBA memiliki tiga mode untuk pengambilan token. Parameter *request* yang digunakan oleh mode token *poll* dan *ping* dapat dilihat pada Tabel 2.13. Mode token *push* tidak membutuhkan *request* token karena token akan dikirim oleh OP [3].

Tabel 2.13 Parameter *Request* Token

Nama Parameter	Deskripsi	Wajib
grant_type	Memiliki nilai "urn:openid:params:grant-type:ciba"	YA
auth_req_id	Nilai unik yang didapatkan ketika melakukan *request* autentikasi.	YA

Ketika terjadi *error* saat melakukan *request* token, maka OP mengembalikan *response* dengan format yang dapat dilihat pada Tabel 2.14 [3].

Tabel 2.14 Format *Response Error* Token

Nama Parameter	Deskripsi	Wajib
error	Kode *error* yang mendefinisikan *error* yang terjadi.	YA
auth_req_id	Identifikasi *request*.	YA
error_description	Deskripsi *error* yang terjadi.	TIDAK

Kode *error* yang digunakan dapat dilihat pada Tabel 2.15 [3].

Tabel 2.15 Kode *Error* Token

Kode	Deskripsi	Status HTTP
authorization_pending	Pengguna belum memberi izin hak akses.	400
slow_down	Kecepatan *polling* harus dipelankan.	400

expired_token	*auth_req_id* telah kedaluwarsa.	400
transaction_failed	Server autorisasi mengalami gangguan.	400
access_denied	Pengguna menolak izin hak akses.	403

Jika pengambilan token berhasil, maka aplikasi klien akan mendapatkan *response* dengan format yang dapat dilihat pada Tabel 2.16 [3].

Tabel 2.16 Format *Response* Token Yang Berhasil

Nama Parameter	Deskripsi	WAJIB
auth_req_id	Identifikasi yang bersifat unik untuk merepresentasikan sesi *request* autentikasi.	YA
access_token	Nilai *access token*.	YA
token_type	Tipe *access token*. Contoh: Bearer.	YA
expires_in	Waktu kedaluwarsa *access token*.	YA
id_token	Nilai *ID Token*.	YA
refresh_token	Nilai *refresh token*.	TIDAK

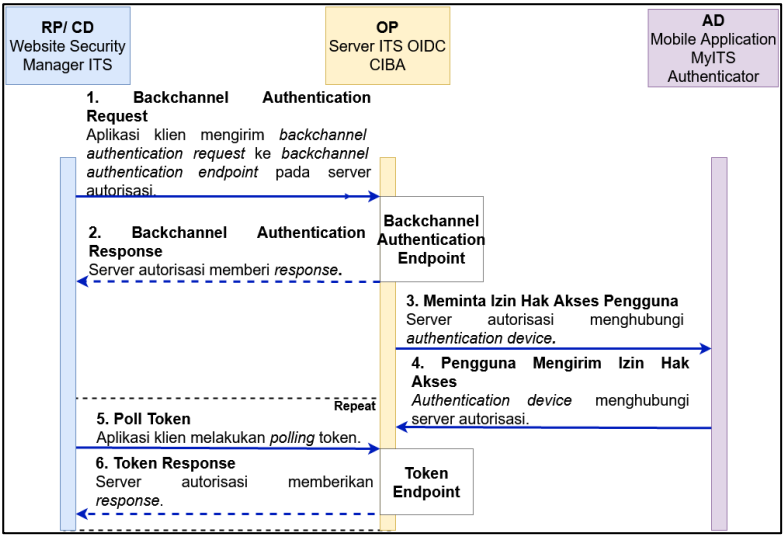
Ketiga mode token CIBA adalah *poll*, *ping*, dan push. Penjelasan ketiga mode tersebut adalah sebagai berikut.

2.4.4.1 Mode Token *Poll*

Polling adalah proses melakukan HTTP *request* secara terus menerus dalam interval yang sudah ditentukan. Setelah RP mendapatkan *response* dari *backchannel authentication endpoint* yang dimiliki oleh OP, RP akan melakukan *polling* terhadap token *endpoint* [3].

Saat proses *polling*, dapat terjadi kemungkinan pengguna belum mengirim izin hak akses. Pada skenario tersebut, aplikasi

klien dapat melakukan *polling* berkali-kali sampai pengguna memberi atau menolak hak akses. Proses *polling* dapat dilihat pada Gambar 2.9.

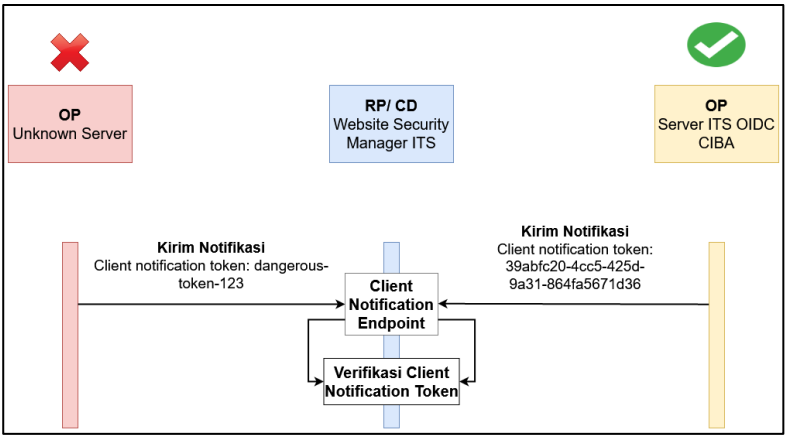


Gambar 2.9 Proses Mode Token *Poll*

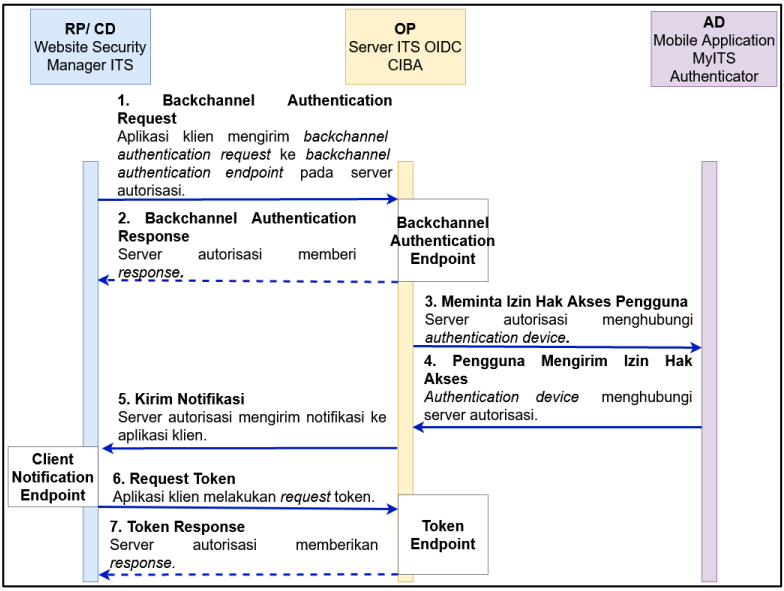
2.4.4.2 Mode Token *Ping*

Pada mode *ping*, setelah proses autorisasi sukses dilakukan pada AD, OP akan mengirim notifikasi ke *client notification endpoint* yang telah dibuat ketika mendaftarkan aplikasi klien [3]. Pengiriman notifikasi oleh OP akan disertakan *client notification token* pada *header request*. RP akan memeriksa *client notification token* untuk memastikan bahwa OP yang mengirim notifikasi adalah server autorisasi yang terpercaya. RP harus memiliki mekanisme untuk menyimpan *client notification token* yang dibuat ketika melakukan *request* autentikasi agar bisa dicocokkan. Mekanisme pemeriksaan *client notification token* dapat dilihat pada Gambar 2.10. Notifikasi yang dikirim bertujuan untuk memberitahu RP bahwa token siap untuk diambil. RP akan mengeksekusi token *endpoint* untuk mengambil token. Proses

pengambilan token menggunakan mode *ping* dapat dilihat pada Gambar 2.11.



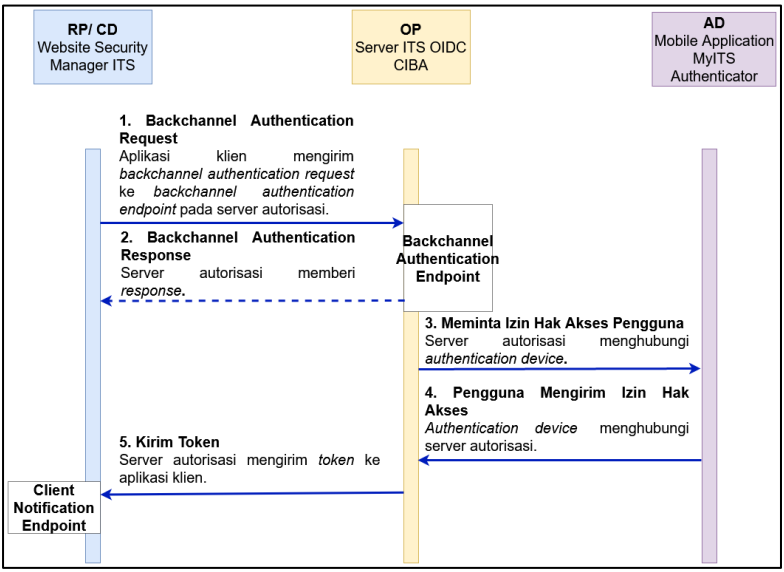
Gambar 2.10 Pemeriksaan *Client Notification Token*



Gambar 2.11 Proses Mode Token *Ping*

2.4.4.3 Mode Token *Push*

Pada mode *push*, setelah proses autorisasi sukses dilakukan pada AD, OP akan mengirim token ke RP menggunakan client *notification endpoint* yang telah dibuat ketika mendaftarkan aplikasi klien [3]. Sama halnya seperti mode token *ping*, OP akan menyertakan *client notification token* yang harus divalidasi oleh RP. Selain melakukan validasi *client notification token*, mode token *push* harus mencocokkan hash dari *access token* dan *auth_req_id* dengan atribut *at_hash* dan *urn:openid:params:jwt:claim:auth_req_id* yang terdapat didalam *ID Token*. Proses pengambilan token pada mode token *push* dapat dilihat pada Gambar 2.12.



Gambar 2.12 Proses Mode Token *Push*

2.5 JSON Web Token

JSON Web Token atau JWT adalah suatu standar untuk mengirim suatu data yang dikemas dalam format yang terstruktur

agar originalitas data tetap terjamin [15]. JWT dapat diverifikasi dan dipercaya karena proses penyusunan telah melalui prosedur *digital signing* [16]. Data yang dikemas dalam JWT direpresentasikan sebagai *claims* atau fakta. Berdasarkan spesifikasi JWT yang mengacu pada dokumen RFC7519, *claims* yang telah terdaftar dapat dilihat pada Tabel 2.17 [17]:

Tabel 2.17 *Claims* pada JWT yang telah ter-registrasi sebagai standar RFC7519

Nama *Claim*	**Deskripsi**
iss	*Issuer* mendefinisikan identitas pihak yang mendistribusikan JWT.
sub	*Subject* mendefinisikan identitas subyek JWT. Sebagai contoh User ID pengguna system *Single Sign On.*
aud	*Audience* mendefinisikan pihak yang akan menerima JWT.
exp	*Expiration time* mendefinisikan waktu kedaluarsa agar JWT tidak diproses lanjut.
nbf	*Not before at* mendefiniskan waktu saat JWT bisa diproses.
iat	*Issued at* mendefinisikan waktu JWT diberikan.
jti	*JWT ID* mendefinisikan identitas JWT.

Meskipun telah terdaftar *claims* yang dijadikan sebagai standar, pihak pengguna JWT dapat mendefiniskan *claims* sesuai dengan kebutuhan sistem.

2.5.1.1 Elemen JWT

Prosedur penyusunan JWT yang menggunakan serialisasi *JSON Web Signature* (JWS) terbagi menjadi tiga elemen. Ketiga elemen tersebut adalah *header, payload,* dan *signature* [18]. Elemen *header* dan *payload* masing-masing akan dilakukan *Base64 URL-safe encode,* lalu digabung dengan hasil *signature* dengan pemisah direpresentasikan oleh karakter ".". Sebagai contoh dapat melihat JWT pada Kode Sumber 2.4.


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiI1MTE2MTAwMTAzIiwibmFtZSI6IkFkaXN0
eWEgQXpoYXIiLCJkcHQiOiJpbmZvcm1hdGljcyIsImlhd
CI6MTUxNjIzOTAyMn0.

mXDNvzjFWTw6kPrs71Oh4BMqZbWicasziV0ewbUux0o
TMFDk0hj3sFFSFc5Jc8oU5UEY4eSyoI4-tryskVAn-
iSALMuP3_7cvlZl-
ZHqQAe4Zd5NMNbMbDkArwkFw5mr4PqgCwZyBGUY
aTSbxdFjkt1xTOtmJJ6dmJliV-83b32_0-0WIDLmozpxH-
o1yGKhlZot4kXaih3ZeZW_Nz-
gpzk6Cy58Hnks_tSb_0Ir5nltS8_z4E88JqwJJLy_38K8sn_P
gD6AHTXJP9yxi5XYpjIteBPSPsdBT9f8vRqAhhqbIipr-
DYxbZw5Z3d9AHzYdLjke1rpBKTUNCAI7ioZpA
```


Heade — *Payload* — *Signature*

Kode Sumber 2.4 Contoh penyusunan JWT menunjukkan tiga elemen: *header, payload,* dan *signature*

Skenario penggunaan JWT pada kali ini akan mengarah ke sistem *autentikasi* dan *autorisasi* sehingga isi dari JWT akan berkaitan dengan fakta-fakta akun pengguna.

a. *Header*

Seluruh JWT memiliki *header* yang berisi *claims*. *Claims* yang terdapat pada header mendefinisikan tipe algoritma yang digunakan untuk menyusun JWT, tipe media, dan

tipe konten [19]. Menggunakan *tool jwt.io, header* JWT telah disusun sebagai berikut [20].

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

Kode Sumber 2.5 *Header* JWT menggunakan algoritma RS256

b. *Payload*

Pada elemen *payload,* terdapat *claims* yang mendefinisikan fakta-fakta akun pengguna [19]. Diantaranya adalah: *sub, name, address,* dan *email*. Pihak yang menggunakan JWT dapat menyisipkan *claims* sesuai dengan kebutuhannya. *Claims* yang tidak tercatat pada dokumen RFC7519 [17] atau yang telah dijelaskan pada Tabel 2.17 disebut sebagai *private* dan *public claims*.

- *Private claims*: fakta yang didefinisikan oleh pengguna JWT. Fakta yang digunakan bersifat *ad hoc* untuk kasus pengguna yang spesifik. Oleh karena itu, pengguna JWT harus melakukan pencegahan tidak terjadi tabrakan dengah *claims* yang sudah ada.
- *Public claims*: fakta yang didefinisikan oleh pengguna JWT dan teregistrasi pada situs IANA JSON Web Token Claims Registry [21]. Oleh karena itu, tabrakan terhadap *claims* yang sudah ada dapat terminimalisir.

Menggunakan *tool* jwt.io tersusun *payload* sebagai berikut [20]:

```
{
  "sub": "5116100103",
  "name": "Adistya Azhar",
  "dpt": "informatics",
  "iat": 1516239022
}
```

Kode Sumber 2.6 *Payload* JWT yang berisi fakta akun pengguna

c. *Signature*

Pembuatan *signature* melalui proses *encode header*, *encode payload*, dan sandi rahasia atau *private key* tergantung dengan algoritma yang digunakan [19]. Elemen *signature* digunakan oleh pengguna JWT untuk memastikan bahwa JWT yang diterima berisi data yang asli, dan berasal dari pihak yang terpercaya.

Berikut adalah contoh *signature* yang disusun menggunakan *tool jwt.io* [20].


```
mXDNvzjFWTw6kPrs71Oh4BMqZbWicasziV0ewb
Uux0oTMFDk0hj3sFFSFc5Jc8oU5UEY4eSyoI4-
tryskVAn-iSALMuP3_7cvlZl-
ZHqQAe4Zd5NMNbMbDkArwkFw5mr4PqgCwZy
BGUYaTSbxdFjkt1xTOtmJJ6dmJliV-83b32_0-
0WIDLmozpxH-o1yGKhlZot4kXaih3ZeZW_Nz-
gpzk6Cy58Hnks_tSb_0Ir5nltS8_z4E88JqwJJLy_38K
8sn_PgD6AHTXJP9yxi5XYpjIteBPSPsdBT9f8vRq
AhhqbIipr-
DYxbZw5Z3d9AHzYdLjke1rpBKTUNCAI7ioZpA
```


Kode Sumber 2.7 *Signature* JWT menggunakan algoritma RS256

Private key yang digunakan untuk menyusun elemen *signature* pada Kode Sumber 2.7 dapat dilihat pada Kode Sumber 2.8.


```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAnzyis1ZjfNB0bBgKFMSvvkTtwlvBsaJq7S5wA+kzeVOVpVWw
kWdVha4s38XM/pa/yr47av7+z3VTmvDRyAHcaT92whREFpLv9cj5lTeJSibyr/Mr
m/YtjCZVWgaOYIhwrXwKLqPr/11inWsAkfIytvHWTxZYEcXLgAXFuUuaS3uF9gEi
NQwzGTU1v0FqkqTBr4B8nW3HCN47XUu0t8Y0e+lf4s4OxQawWD79J9/5d3Ry0vbV
3Am1FtGJiJvOwRsIfVChDpYStTcHTCMqtvWbV6L11BWkpzGXSW4Hv43qa+GSYOD2
QU68Mb59oSk2OB+BtOLpJofmbGEGgvmwyCI9MwIDAQABAoIBACiARq2wkltjtcjs
kFvZ7w1JAORHbEufEO1Eu27zOIlqbgyAcAl7q+/1bip4Z/x1IVES84/yTaM8p0go
amMhvgry/mS8vNi1BN2SAZEnb/7xSxbflb70bX9RHLJqKnp5GZe2jexw+wyXlwaM
+bclUCrh9e1ltH7IvUrRrQnFJfh+is1fRon9Co9Li0GwoN0x0byrrngU8Ak3Y6D9
D8GjQA4Elm94ST3izJv8iCOLSDBmzsPsXfcCUZfmTfZ5DbUDMbMxRnSo3nQeoKGC
0Lj9FkWcfmLcpGlSXTO+Ww1L7EGq+PT3NtRae1FZPwjddQ1/4V905kyQFLamAA5Y
lSpE2wkCgYEAy1OPLQcZt4NQnQzPz2SBJqQN2P5u3vXl+zNVKP8w4eBv0vWuJJF+
hkGNnSxXQrTkvDOIUddSKOzHHgSg4nY6K02ecyT0PPm/UZvtRpWrnBjcEVtHEJNp
bU9pLD5iZ0J9sbzPU/LxPmuAP2Bs8JmTn6aFRspFrP7W0s1Nmk2jsm0CgYEAyH0X
+jpoqxj4efZfkUrg5GbSEhf+dZglf0tTOA5bVg8IYwtmNk/pniLG/zI7c+GITc9B
BwfMr59EzBq/eFMI7+LgXaVUsM/sS4Ry+yeK6SJx/otIMWtDfqxsLD8CPMCRvecC
2Pip4uSgrl0MOebl9XKp57GoaUWRWRHqwV4Y6h8CgYAZhI4mh4qZtnhKjY4TKDjx
QYufXSdLAi9v3FxmvchDwOgn4L+PRVdMwDNms2bsL0m5uPn104EzM6w1vzz1zwKz
5pTpPI0OjgWN13Tq8+PKvm/4Ga2MjgOgPWQkslulO/oMcXbPwWC3hcRdr9tcQtn9
Imf9n2spL/6EDFId+Hp/7QKBgAqlWdiXsWckdE1Fn91/NGHsc8syKvjjk1onDcw0
NvVi5vcba9oGdElJX3e9mxqUKMrw7msJJv1MX8LWyMQC5L6YNYHDfbPF1q5L4i8j
8mRex97UVokJQRRA452V2vCO6S5ETgpnad36de3MUxHgCOX3qL382Qx9/THVmbma
3YfRAoGAUxL/Eu5yvMK8SAt/dJK6FednGcM3JEFNplmtLYVLWhkIlNRGDwkg3I5K
y18Ae9n7dHVueyslrb6weq7dTkYDi3iOYRW8HRkIQh06wEdbxt0shTzAJvvCQfrB
jg/3747WSsf/zBTcHihTRBdAv6OmdhV4/dD5YBfLAkLrd+mX7iE=
-----END RSA PRIVATE KEY-----
```


Kode Sumber 2.8 *Private Key* Yang Digunakan Untuk Menyusun JWT

2.5.1.2 Penggunaan JWT

Berikut adalah beberapa kasus penggunaan dimana JWT dapat berguna:

a. Autorisasi

Skenario yang paling umum saat menggunakan JWT adalah pembatasan hak akses *resource* pada suatu *server*. Setelah pengguna berhasil autentikasi, maka untuk

meminta *resource* kedepannya dapat menggunakan token JWT. Kebenaran JWT akan menjamin pengguna untuk selalu mendapatkan layanan dari *server*.

b. Penukaran Informasi
Bertukar informasi melalui internet dengan berbagai pihak membutuhkan keamanan agar tehindar dari peretas yang dapat merubah isi data secara tersembunyi. Oleh karena itu, dengan menggunakan JWT pihak yang menerima informasi dapat memverifikasi bahwa data yang diterima adalah asli, tidak terjadi korupsi data, dan data berasal dari pihak yang dipercaya. Proses ini mungkin karena adanya *digital signing* yang dilakukan oleh pihak yang mendistribusikan JWT.

2.6 Prinsip Pemrograman Berorientasi Obyek

Pengembangan protokol CIBA pada myITS SSO akan mengikuti prinsip pemrograman berorientasi obyek agar kode sumber mudah dikembangkan, dipelihara, dapat dipahami dengan mudah, serta tidak merusak bagian kode sumber yang lain. Oleh karena itu, prinsip SOLID digunakan selama masa pengembangan protokol CIBA.

2.6.1.1 Penjelasan Prinsip SOLID

Prinsip SOLID terdiri dari lima elemen. Kelima elemen tersebut adalah [22]:

1. Prinsip *Single Responsibility*
Prinsip *single responsibility* memiliki arti bahwa obyek harus memiliki satu tanggung jawab yang spesifik [22]. Memiliki lebih dari satu tanggung jawab akan mengakibatkan kesulitan untuk melakukan *refactoring* yang tanpa tidak sengaja merusak bagian yang lain. Dengan tanggung jawab yang terpisah, maka tingkat ketergantungann akan rendah karena tanggung jawab kelas tidak terpusat menjadi satu.

Dapat dilihat pada Kode Sumber 2.9 terdapat dua kelas yang memiliki tanggung jawab spesifik. Kelas *User* untuk manajemen *user*, dan Kelas *User Repository* untuk akses ke basis data.

```
1.  class User {
2.      public function getName() { }
3.      public function getEmail() { }
4.  }
5.
6.  class UserRepository {
7.      public function find($id) { }
8.      public function save(User $user) { }
9.  }
```

Kode Sumber 2.9 Kelas *User* dan *User Repository* Memiliki Tanggung Jawab Yang Berbeda

2. Prinsip *Open-Closed*

   Prinsip *open-closed* memiliki arti terbuka untuk pengembangan lanjutan tetapi tertutup untuk modifikasi kode sumber yang sudah ada [22]. *Developer* yang akan mengerjakan sistem tidak boleh memodifikasi kode sumber yang sudah ada namun yang harus dilakukan adalah mencari cara bagaimana kelas bisa dikembangkan secara terpisah (*extend*). Sebagai contoh Kode Sumber 2.10 dalam sistem yang dapat menghasilkan nota, jika *developer* ingin menambahkan fitur pengiriman nota yang baru, hanya perlu membuat kelas yang mengimplementasikan abstraksi *Delivery Interface*. Tidak perlu mengubah class yang sudah ada.

   Dapat dilihat pada Kode Sumber 2.10 terdapat kelas yang memiliki tanggung jawab untuk mengirim nota melalui *email*.

```
1.  class EmailInvoiceDelivery implements DeliveryInterface {
2.    public function send(Invoice $invoice) { }
3.  }
```

Kode Sumber 2.10 Kelas Yang Mengakomodir Fitur Baru Mengirim Nota Melalui *Email*

3. Prinsip *Liskov Substitution*

Prinsip *liskov substitution* memiliki arti bahwa obyek yang berasal dari abstraksi yang sama dapat ditukar tanpa mempengaruhi perilaku obyek yang membutuhkannya. Abstraksi pada bahasa pemrograman PHP dapat menggunakan *interface*. *Interface* memberikan definisi struktur suatu kelas, lalu kelas dapat mengimplementasikan *interface* tersebut dan memiliki perilaku khusus [22].

Dapat dilihat pada Kode Sumber 2.11 terdapat tiga kelas yang dapat saling ditukar, karena berasal dari abstraksi yang sama.

```
1.  class Greeter {
2.    public function sayHello(HelloInterface $hello) {
3.      echo $hello->getHello();
4.    }
5.  }
6.
7.  $greeter = new Greeter();
8.  $greeter->sayHello(new EnglishHello());
9.  $greeter->sayHello(new IndonesianHello());
10. $greeter->sayHello(new CantoneseHello());
```

Kode Sumber 2.11 Kelas *Engish Hello*, *Indonesian Hello*, Dan *Cantonese Hello* Dapat Saling Ditukar Karena Berasal Dari Abstraksi Yang Sama

4. Prinsip *Interface Segregation*

Suatu kelas yang mengimplementasi suatu abstraksi harus memasukkan semua fungsi yang telah terdefinisi. Pada banyak kasus, ada fungsi yang memang tidak dibutuhkan oleh kelas tersebut. Ini berkontradiksi dengan prinsip *single responsibility*. Oleh karena itu, prinsip *interface segregation*

menghasilkan *decoupling* dengan cara memisahkan abstraksi sesuai tanggung jawab yang berlaku [22].
Dapat dilihat pada Kode Sumber 2.12 terdapat dua abstraksi sesuai dengan tanggung jawab yang dimiliki.

```
1.  interface LogWriterInterface {
2.     public function write($message);
3.  }
4.
5.
6.  interface LogReaderInterface {
7.     public function read();
8.  }
```

Kode Sumber 2.12 Tanggung Jawab Untuk Menyimpan Dan Membaca *Log* Dapat Dijadikan Abstraksi Yang Terpisah

5. Prinsip *Dependency Inversion*
   Prinsip *dependency inversion* menyatakan bahwa:
   a. Modul harus bergantung pada abstraksi
   b. Abstraksi tidak boleh bergantung pada detail, tetapi detail bergantung pada abstraksi

   Sebagai contoh, ketika ada kelas yang bergantung pada modul level rendah seperti *input-output* pada komputer akan terjadi masalah jika prosedur *input-output* akan diubah. Awalnya *input* berasal dari *keyboard*, lalu diganti menggunakan *joystick*.
   Dengan menggunakan prinsip *liskov substitution*, *developer* dapat menukar obyek yang berasal dari abstraksi yang sama. Oleh karena itu, perlu adanya lapisan abstrak berupa *interface*, dan kelas yang mulanya membutuhkan *input-output* harus bergantung pada lapisan abstraksi. Itulah yang disebut *inverting dependency* atau merubah ketergantungan yang mulanya terhadap implementasi menjadi abstraksi [22].
   Dapat dilihat pada Kode Sumber 2.13, Kode Sumber 2.14, dan Kode Sumber 2.15 terdapat kelas *Computer* yang mengaplikasikan prinsip *dependency inversion* dengan

cara menerima abstraksi *Input Interface* dan *Output Interface* melalui *constructor* kelas.

```
1.  class Computer {
2.    protected $input;
3.    protected $output;
4.
5.    public function __construct(
6.      InputInterface $input,
7.      OutputInterface $output
8.    ) {
9.      $this->input = $input;
10.     $this->output = $output;
11.   }
12. }
```

Kode Sumber 2.13 Kelas *Computer* Yang Bergantung Pada Abstraksi Untuk Melakukan *Output-Input* Agar Dapat Dilakukan *Dependency Inversion*

```
1.  class KeyboardInput implements InputInterface {
2.   public function getInputEvent() { }
3.  }
4.
5.  class JoystickInput implements InputInterface {
6.     public function getInputEvent() { }
7.  }
```

Kode Sumber 2.14 Kelas Untuk Melakukan *Input* Pada Komputer Dapat Direpresentasikan Melalui *Keyboard* Dan *Joysyick*

```
1.  class MonitorOutput implements OutputInterface {
2.    public function render() { }
3.  }
4.
5.  class TerminalOutput implements OutputInterface {
6.    public function render() { }
7.  }
```

Kode Sumber 2.15 Kelas Untuk Melakukan *Output* Pada Komputer Dapat Direpresentasikan Melalui *Terminal* Dan Layar

2.7 Representational State Transfer Application Programming Interface (REST API)

REST API merupakan standar arsitektur yang digunakan untuk komunikasi pada lingkungan layanan web. Pada umumnya, REST memanfaatkan protokol HTTP ketika digunakan untuk komunikasi [23].

2.7.1.1 Cara kerja REST API

Data yang diabstraksikan dalam bentuk REST API dinamakan sebagai *resource*. Obyek apapun dapat dijadikan sebagai *resource*, seperti dokumen atau *file*, kumpulan berita, mahasiswa aktif dan lain-lain. REST menggunakan *resource identifier* untuk mengidentifkasikan suatu *resource* ketika berinteraksi dengan komponen lain. Ini berguna agar *resource* yang dituju tidak terjadi duplikat atau salah tujuan. *Resource identifier* pada umumnya bersifat unik [23].

Proses untuk mendapatkan *resource* dapat memanfaatkan metode HTTP yang terdefinisi berdasarkan dokumen RFC2616 [24]. Adapun, metode HTTP dalam penggunaan REST adalah sebagai berikut beserta contoh *request* yang dibangun dengan *tool reqbin* [25]:

a. GET: digunakan untuk mengambil *resource*
   Dapat dilihat pada Kode Sumber 2.16 contoh *request* menggunakan metode GET.

```
GET /api/students/011-224-6983
HTTP/1.1
Host: example.com
```

Kode Sumber 2.16 *Request* Untuk Mencari Data Mahasiswa Menggunakan *Resource Identifier*

b. PUT: digunakan untuk memperbaruhi *resource*
   Dapat dilihat pada Kode Sumber 2.17 contoh *request* menggunakan metode PUT.

```
PUT /api/students/011-224-6983 HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 64

{
        "name": "M. Adistya Azhar",
        "age": 25,
        "height": 165
}
```

Kode Sumber 2.17 *Request* Untuk Memperbaruhi Data Mahasiswa. Dapat Diperhatikan Bahwa Terdapat *Resource Identifier* Beserta *Content Body*

c. POST: digunakan untuk membuat *resource* baru
   Dapat dilihat pada Kode Sumber 2.18 contoh *request* menggunakan metode POST.

```
POST /api/students HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 60

{
        "name": "Adistya Azhar",
        "age": 20,
        "height": 165
}
```

Kode Sumber 2.18 *Request* Untuk Membuat Data Mahasiswa Baru

d. DELETE: digunakan untuk menghapus resource
   Dapat dilihat pada Kode Sumber 2.19 contoh *request* menggunakan metode DELETE.

```
DELETE /api/students/011-224-6983 HTTP/1.1
Host: example.com
```

Kode Sumber 2.19 *Request* Untuk Menghapus Data Mahasiswa

2.8 Kerangka Kerja Phalcon

Phalcon merupakan kerangka kerja untuk membuat aplikasi *web* menggunakan bahasa pemrograman PHP. Kerangkat kerja ini dibangun menggunakan bahasa pemrograman C yang kemudian menjadi *extension* yang bisa dimuat dalam lingkungan PHP. Dengan dibangunnya Phalcon menggunakan bahasa pemrograman C, dapat dipastikan bahwa Phalcon memiliki performa yang tinggi.

Phalcon menawarkan pengalaman pengembangan yang menyenangkan, karena *developer* dapat menggunakan obyek yang hanya dibutuhkan sehingga menghasilkan sumber kode yang *decoupled*. Kerangka kerja ini menyediakan fitur yang komplit. Diantaranya adalah sebagai berikut:

- Modul *dependency injection*
- Modul *session*
- Modul keamanan seperti Access Control List, dan Crypt untuk membatasi hak akses dan enkripsi data
- HTTP *router*
- Autoloader
- Object Relational Mapper, akses basis data dan *cache*
- *Template engine* yang bernama Volt
- Internasionalisasi agar aplikasi *web* bisa dioperasikan dalam berbagai bahasa

2.9 Microsoft SQL Server

Microsoft SQL Server adalah suata basis data relasional yang dikembangkan oleh perusahaan Microsoft. SQL Server memiliki fungsi utama yaitu untuk menyimpan dan mengambil data yang diminta oleh aplikasi lain [26].

Operasi yang jalan pada SQL Server akan melalui protokol yang dinamakan Tabular Data Stream (TDS). TDS berada pada lapisan aplikasi yang digunakan untuk bertukar data antara server basis data dan klien. SQL Server mendukung berbagai macam tipe data, diantaranya adalah tipe data primitive seperti *integer, float, decimal, char, varchar, binary,* dan *text* [26].

Integrasi SQL Server pada lingkungan bahasa pemrograman PHP dapat dilakukan dengan memanfaatkan *ODBC Driver 17,* ekstensi *php_sqlsrv,* dan ekstensi *php_pdo_sqlsrv* [27].

2.10 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) adalah platform berbasis *cloud* yang memberikan layanan pengiriman pesan notifikasi untuk perangkat Android, iOS dan aplikasi *web*.

Kemampuan utama yang dimiliki oleh FCM antara lain adalah:

- Mengirim pesan notifikasi atau pesan data yang ditampilkan kepada pengguna.

- Mendistribusikan pesan ke aplikasi klien yang menargetkan satu perangkat, grup perangkat, atau perangkat yang berlangganan pada suatu topik.
- Mengirim pesan dari aplikasi klien yang berupa notifikasi, *chat*, atau pesan lain dari perangkat ke server FCM melalui slauran koneksi FCM.

[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN

Pada bab ini akan dibahas tahap analisis kebutuhan dari sistem yang akan dibangun, serta perancangan sistem dan aplikasinya. Pada bagian bab ini juga dibahas mengenai analisis permasalahan yang akan diselesaikan, dan selanjutnya dibahas mengenai perancangan program secara umum untuk memberikan gambaran awal mengenai aplikasi yang akan dibangun. Bab ini disusun dengan menggunakan pendekatan berorientasi obyek (*object oriented approach)* yang direpresentasikan dalam diagram-diagram *Unified Modelling Language* (UML).

3.1 Analisis

Bagian analisis mengenai sistem dibagi menjadi beberapa subbagian, antara lain: domain permasalahan yang menerangkan permasalahan pokok sistem, deskripsi umum sistem, dan spesifikasi kebutuhan sistem.

3.1.1 Domain Permasalahan

Domain permasalahan yang diangkat dalam pengerjaan tugas akhir ini meliputi pengimplementasian protokol *Client Initiated Backchannel Authentication* (CIBA) pada server autorisasi myITS Single Sign-On (myITS SSO), serta secara khusus pengimplementasian pendaftaran aplikasi klien agar memungkinkan untuk menggunakan alur CIBA, layanan pengambilan dan pemberian *access token* berdasarkan mode token *poll*, *ping*, dan *push*, dan *session binding* antara *Consumption Device* (CD) dan *Authentication Device* (AD).

3.1.1.1 Domain Permasalahan CIBA Pada Server Autorisasi

Domain permasalahan utama yang diangkat dalam pengerjaan tugas akhir ini adalah bagaimana mengimplentasikan protokol CIBA pada server autorisasi myITS SSO sesuai standar yang berlaku. CIBA merupakan alur *OpenID Connect* yang

memungkinkan aplikasi klien untuk mendapatkan token dari server autorisasi dengan pemberian hak akses oleh pengguna dari perangkat yang terpisah.

Pada umumnya, sever autorisasi dengan protokol yang sudah terstandarisasi memudahkan pihak lain yang membutuhkan layanan autorisasi untuk berkomunikasi dengan cara yang terorganisir. Jika salah satu pihak server maupun klien tidak mengikuti standar protokol, maka akan menyebabkan pertukaran komunikasi terhenti karena terganggunya proses identifikasi tujuan komunikasi. Oleh karena itu, implementasi CIBA pada server autorisasi myITS SSO harus sesuai dengan standar yang telah tertulis pada spesifikasi CIBA [3].

3.1.1.2 Domain Permasahalan Pendaftaran Aplikasi Klien

Penggunaan aplikasi pendukung perkerjaan dalam lingkungan organisasi cenderung meningkat. Seiring dengan bertingkatnya jumlah aplikasi diperlukan basis data akun pengguna yang tersentralisasi sehingga hanya dibutuhkan satu akun pengguna untuk mengakses seluruh aplikasi. Departemen IT dalam suatu organisasi dapat menggunakan sistem *Single Sign-On* untuk menangani basis data akun pengguna yang tersentralisasi.

Hal yang serupa telah dilakukan oleh myITS SSO. MyITS SSO mampu memberikan layanan autorisasi dan autentikasi yang terpusat, sehingga aplikasi yang terdapat pada lingkunan Institut Teknologi Sepuluh Nopember (ITS) dapat digunakan oleh seluruh sivitas akademika ITS dengan mudah. Namun, pendekatan ini tidak terbuka untuk aplikasi umum.

Oleh karena itu, diperlukan pendaftaran aplikasi klien agar dapat menggunakan protokol yang tersedia pada myITS SSO. Secara spesifik pada tugas akhir ini dibutuhkan pendaftaran aplikasi klien agar dapat menggunakan layanan protokol CIBA.

3.1.1.3 Domain Permasalahan Pengambilan *Access Token*

Aplikasi klien yang terhubung dengan myITS SSO dapat mengakses data yang terproteksi dengan cara menyertakan *access*

token. Pada protokol CIBA harus dicermati bahwa pengeluaran token dapat dilalui menggunakan tiga mode yaitu *push*, *ping*, dan *poll*. Masing-masing mode memiliki karakteristik yang unik. Sebagai contoh mode *push* dan *ping* memiliki lapisan keamanan tambahan yang dinamakan sebagai *client notification token*. Lapisan ini digunakan oleh aplikasi klien agar dapat memverifikasi bahwa token atau notifikasi berasal dari server autorisasi yang dipercaya. Sedangkan pada mode *poll* tidak membutuhkan *client notification token* tetapi memiliki prosedur melakukan token request berkali-kali yang dapat memakan sumber daya server.

Server autorisasi yang mendukung protokol CIBA harus setidaknya mendukung satu mode pengambilan token, dan aplikasi klien yang terdaftar menggunakan hanya satu mode pengambilan token yang didukung oleh server autorisasi seperti yang dapat dilihat pada Tabel 3.1.

Oleh karena itu, diperlukan implementasi server autorisasi CIBA yang mampu mendukung tiga mode pengambilan token agar aplikasi klien dapat mendaftar sesuai dengan kebutuhan dan kemampuan yang dimiliki.

Tabel 3.1 Contoh Kasus Mode Pengambilan Token Yang Dapat Dan Tidak Dapat Digunakan

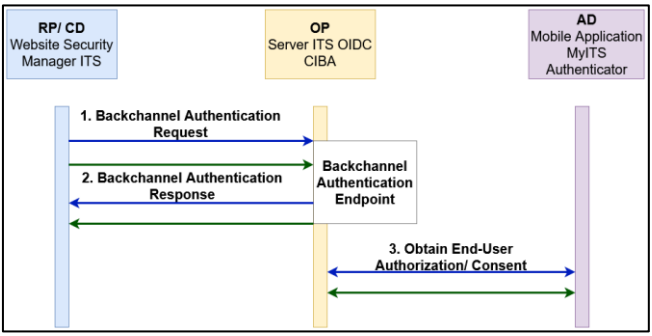
Entitas CIBA	Mode Token	Valid
Server Autorisasi XYZ	push, ping	YA
Aplikasi Klien A	push	YA
Aplikasi Klien B	poll	TIDAK. OP (Server Autorisasi XYZ) tidak mendukung.
Aplikasi Klien C	push, ping	TIDAK. Hanya boleh satu mode.

3.1.1.4 Domain Permasalahan *Session Binding* Antara *Consumption Device* dan *Authentication Device*

Aplikasi klien yang menginisiasi CIBA berada pada perangkat *Consumption Device* (CD), dan aplikasi yang menerima permintaan hak akses berada pada *Authentication Device* (AD). Dikarenakan proses autentikasi dan autorisasi berjalan pada perangkat yang saling terpisah menimbulkan celah keamanan. Celah keamanan ini menimbulkan:

1. Permintaan hak akses dari bermacam aplikasi klien sehingga pengguna AD tidak dapat membedakan sesi aplikasi klien yang sesungguhnya.
2. Permintaan hak akses dari satu aplikasi klien tetapi terjadi permintaan hak akses yang menumpuk, sehingga pengguna AD tidak dapat mengidentifikasi permintaan hak akses yang sesungguhnya.

Pada Gambar 3.1 terdapat *request* CIBA dari aplikasi klien Security Manager ITS yang terjadi dua kali secara bersamaan (direpresentasikan dengan panah warna biru dan hijau) akan membingungkan pengguna AD karena pengguna AD bisa kesulitan untuk membedakan permintaan hak akses mana yang valid.



Gambar 3.1 Ilustrasi Inisiasi *Request* CIBA Yang Saling Tumpang Tindih

Oleh karena itu, dibutuhkan implementasi server autorisasi CIBA yang dapat menangani *session binding* menggunakan *binding message* yang dapat diandalkan agar pengguna RP dan AD dapat mengidentifikasi permintaan hak akses yang sesungguhnya.

3.1.2 Deskripsi Umum Perangkat Lunak

Pada tugas akhir ini akan dibangun sebuah perangkat lunak yang dapat mengakomodir protokol CIBA yang berjalan di lingkunan ITS menggunakan server autorisasi myITS SSO.

Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI) ITS merupakan organisasi dimana salah satu tugasnya adalah memberi layanan *customer service* oleh staf DPTSI untuk menangani keluhan yang dialami sivitas akademika. Pada umumnya, keluhan yang terjadi dapat berupa *error* yang muncul ketika menggunakan sebuah aplikasi. Agar keluhan tersebut dapat ditangani dengan baik, maka staf DPTSI sebaiknya dapat melihat secara langsung *error* yang terjadi. Satu-satunya cara agar *error* dapat dilihat secara langsung adalah dengan mengoperasikan akun pelapor. DPTSI memiliki aplikasi internal dengan nama Security Manager, yang dapat mengatur akun pengguna, aplikasi klien, *provider* server autorisasi, dan lain-lain. Pada aplikasi Security Manager, staf DPTSI dapat melakukan *login* atas pengguna yang dipilih, yang dinamakan dengan fitur *impersonate*. Dikarenakan akun pengguna yang bersifat sensitif, diperlukan prosedur agar pemilik akun dapat memberi akses yang dilakukan oleh staf DPTSI. Disinilah protokol CIBA dapat berjalan. Sebelum staf DPTSI dapat *impersonate* sebagai akun pelapor keluhan, aplikasi Security Manager akan menginisiasi CIBA. Saat permintaan hak akses muncul pada AD dan diberi hak akses oleh pelapor, maka staf DPTSI dapat melanjutkan *impersonation*. Sebaliknya jika pelapor tidak memberi hak akses, maka staf DPTSI tidak dapat melanjutkan *impersonation*. Pemberian hak akses oleh pengguna akan memberi tanda bahwa proses CIBA dapat diselesaikan, sehingga aplikasi klien yaitu Security Manager memungkinkan

untuk mendapatkan token dari server autorisasi. Disaat Security Manager sudah mendapatkan token, maka staf DPTSI dapat menyelesaikan fase *impersonation.*

Entitas yang berjalan pada lingkungan CIBA ada empat, yaitu *OpenID Provider* atau server autorisasi, *Authentication Device*, *Relying Party* atau aplikasi klien, dan *Consumption Device*. Pembagian kerja entitas dan aplikasi CIBA pada tugas akhir ini dapat dilihat pada Tabel 3.2 dan Tabel 3.3.

Pada tugas akhir ini akan dibangun dua aplikasi di sisi server agar dapat berjalan pada lingkungan server autorisasi CIBA. Adapun kedua aplikasi tersebut adalah:

1. *Library* Server Autorisasi CIBA.
2. Aplikasi Server Autorisasi CIBA.

Dua aplikasi lainnya akan dibangun oleh Andika Andra dan dapat dibaca pada laporan tugas akhir yang berjudul "Implementasi Protokol Client Initiated Backchannel Authentication (CIBA) di Sisi Klien pada MyITS Single Sign-On" [28], yaitu:

1. Aplikasi Security Manager ITS sebagai aplikasi klien.
2. Aplikasi myITS Authenticator sebagai *Authentication Device*.

Tabel 3.2 Pembagian Kerja Entitas CIBA pada Tugas Akhir

Entitas	Muhammad Adistya Azhar	Andika Andra
Server Autorisasi	✓	×
Aplikasi Klien	×	✓
Authentication Device	×	✓
Consumption Device	×	✓

Tabel 3.3 Pembagian Kerja Aplikasi CIBA Pada Tugas Akhir

Nama Aplikasi	Muhammad Adistya Azhar	Andika Andra
Library Server Autorisasi CIBA	✓	×
Aplikasi Server Autorisasi CIBA	✓	×
Aplikasi Security Manager ITS	×	✓
Aplikasi myITS Authenticator	×	✓

Perbedaan tugas akhir ini dengan judul "Implementasi Protokol Client Initiated Backchannel Authentication (CIBA) di Sisi Server pada MyITS Single Sign-On" dan tugas akhir Andika Andra dengan judul "Implementasi Protokol Client Initiated Backchannel Authentication (CIBA) di Sisi Klien pada MyITS Single Sign-On" [28] terletak pada *concern* yang dimilikinya. Tugas akhir ini menekankan *concern* pada sisi server, yaitu untuk memberikan layanan CIBA. Adapun layanan tersebut terdiri dari:

1. Prosedur alur autentikasi menggunakan tiga alur CIBA antara lain *push*, *ping*, dan *poll*.
2. Pemodelan struktur basis data agar dapat menyimpan data aplikasi klien, *access token*, sesi autentikasi, dan akun pengguna.
3. Mekanisme komunikasi pesan dengan aplikasi klien dan *Authentication Device*.

Tugas akhir Andika Andra [28] memiliki *concern* membangun aplikasi klien agar dapat memanfaatkan layanan CIBA. Adapun *concern* tersebut terdiri dari:

1. Prosedur memodelkan *request HTTP* agar dapar mengonsumsi layanan alur CIBA dengan mode token *push*, *ping*, dan *poll*.
2. Mekanisme untuk mengatur tingkah laku aplikasi klien apabila server autorisasi CIBA mengembalikan pesan *error*.
3. Mekanisme untuk dapat menerima pesan dari server autorisasi CIBA.

3.1.2.1 *Library* Server Autorisasi CIBA

Library server autorisasi CIBA adalah *library* yang akan dikembangkan untuk mengakomodir protokol CIBA. *Library* ini merupakan hasil pengembangan lanjutan dari *library Bshaffer OAuth 2.0*. *Library Bshaffer OAuth 2.0* memiliki fungsi-fungsi yang dapat mengakomodir alur seperti *Authorization Code*, *Client Credentials*, *Implicit*, dan *User Credentials*.

Proses bisnis *library* server autorisasi meliputi:

1. Inisiasi request autentikasi yang menentukan batasan hak akses serta identifikasi pengguna yang dituju (*authentication request*).
2. Pengiriman dan penerimaan izin hak akses dari pengguna yang dituju (*obtaining end-user consent*).
3. Pembuatan token menggunakan mode *push*, *ping*, dan *poll* (*issuing token*).
4. Pemberian akses data yang terproteksi menggunakan token.

3.1.2.2 Aplikasi Server Autorisasi CIBA

Aplikasi server autorisasi CIBA merupakan aplikasi yang menggunakan *library* server autorisasi CIBA. Aplikasi ini akan dibangun menggunakan kerangka kerja Phalcon. Aplikasi ini akan memanggil fungsi-fungsi yang terdapat pada *library* server autorisasi CIBA. Aplikasi ini tidak menyimpan logika kode untuk memenuhi protokol CIBA, melainkan *library* server autorisasi CIBA yang melapisi logika CIBA. Alasan menggunakan pendekatan ini adalah:

1. Agar tidak terjadi *hard coupling* atau ketergantungan secara langsung antara kerangka kerja dan *library*.
2. Agar dapat menggunakan *library* server autorisasi pada kerangka kerja yang berbeda. Tidak hanya pada kerangka kerja Phalcon saja.

3.1.3 Spesifikasi Kebutuhan Perangkat Lunak

Berdasarkan uraian mengenai cakupan-cakupan subdomain yang telah dijelaskan pada subbab 3.1.1 dan deskripsi umum pada subbab 3.1.2, maka terdapat beberapa spesifikasi perangkat lunak yang harus dipenuhi dalam pembangunan aplikasi ini agar sistem mampu mengakomodasi serta menyelesaikan permasalahan yang menjadi domain utama dalam tugas akhir ini. Spesifikasi kebutuhan aplikasi ini dibagi menjadi dua, yaitu kebutuhan fungsional dan kebutuhan nonfungsional.

3.1.3.1 Spesifikasi Kebutuhan Fungsional

Kebutuhan fungsional berisi semua kebutuhan utama yang harus dipenuhi oleh aplikasi agar sistem mampu berjalan bahkan bekerja dengan baik. Kebutuhan fungsional secara tidak langsung mendeskripsikan fungsionalitas dasar perangkat lunak. Daftar kebutuhan fungsional dapat dilihat pada Tabel 3.4.

Tabel 3.4 Daftar Kebutuhan Fungsional Perangkat Lunak

No	Kebutuhan Fungsional	Deskripsi
1	CIBA menggunakan mode token *push*	Pengguna dapat memulai CIBA menggunakan mode token *push*.
2	CIBA menggunakan mode token *ping*	Pengguna dapat memulai CIBA menggunakan mode token *ping*.
3	CIBA menggunakan mode token *poll*	Pengguna dapat memulai CIBA

No	Kebutuhan Fungsional	Deskripsi
		menggunakan mode token *poll*.
4	Autentikasi berperan sebagai akun yang berbeda	Pengguna dapat memilih akun untuk memulai autentikasi yang berperan sebagai akun yang berbeda.

3.1.3.2 Spesifikasi Kebutuhan Nonfungsional

Kebutuhan nonfungsional mendeskripsikan batasan dan karakteristik pada sistem dalam lingkungan operasional. Daftar kebutuhan nonfungsional dapat dilihat pada Tabel 3.5.

Tabel 3.5 Daftar Kebutuhan Nonfungsional Perangkat Lunak

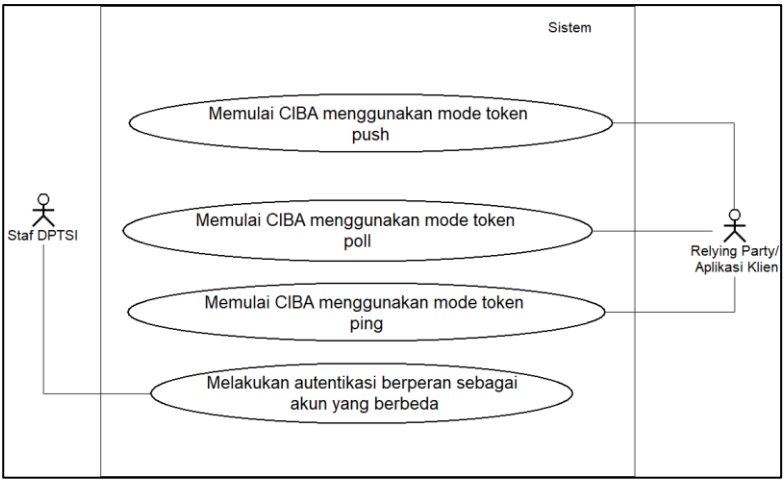
No	Kebutuhan Nonfungsional	Deskripsi
1	Menampilkan pesan *error* yang *developer friendly*.	Sistem harus dapat menampikan pesan *error* sehingga *developer* yang menggunakan sistem dapat mengetahui kesalahan yang terjadi.
2	Membatasi hak akses kepada aplikasi klien dan pengguna.	Sistem harus dapat membatasi hak akses aplikasi klien dan pengguna agar data yang bersifat rahasia tidak terbongkar.

3.2 Perancangan

Tahap perancangan dalam subbab ini dibagi menjadi beberapa bagian yaitu perancangan skenario kasus penggunaan, arsitektur, kelas, basis data, dan antarmuka.

3.2.1 Skenario Kasus Penggunaan

Dalam subbab ini akan dibahas secara rinci masing-masing kasus penggunaan sistem yang akan dibangun dengan menggunakan diagram-diagram untuk mempermudah deskripsi setiap kasus. Pada tugas akhir ini terdapat empat kasus penggunaan seperti yang ditunjukkan pada Gambar 3.2 dan Tabel 3.6.



Gambar 3.2 Diagram Kasus Penggunaan Sistem

Tabel 3.6 Deskripsi Kasus Penggunaan Aplikasi

No	Kode	Kebutuhan Fungsional	Deskripsi
1	UC-001	CIBA menggunakan mode token *push*	Pengguna dapat memulai CIBA menggunakan mode token *push*.

No	Kode	Kebutuhan Fungsional	Deskripsi
2	UC-002	CIBA menggunakan mode token *ping*	Pengguna dapat memulai CIBA menggunakan mode token *ping*.
3	UC-003	CIBA menggunakan mode token *poll*	Pengguna dapat memulai CIBA menggunakan mode token *poll*.
4	UC-004	Autentikasi berperan sebagai akun yang berbeda	Pengguna dapat memilih akun untuk memulai autentikasi yang berperan sebagai akun yang berbeda.

3.2.1.1 Kasus Penggunaan CIBA Menggunakan Mode Token *Push* (UC-001)

Pada kasus penggunaan ini, pengguna dapat menjalankan CIBA menggunakan mode token *push*. Kasus penggunaan ini bekerja sama dengan proses *impersonate* sebagai tanda sistem mendapatkan izin hak akses oleh pengguna sivitas akademika.

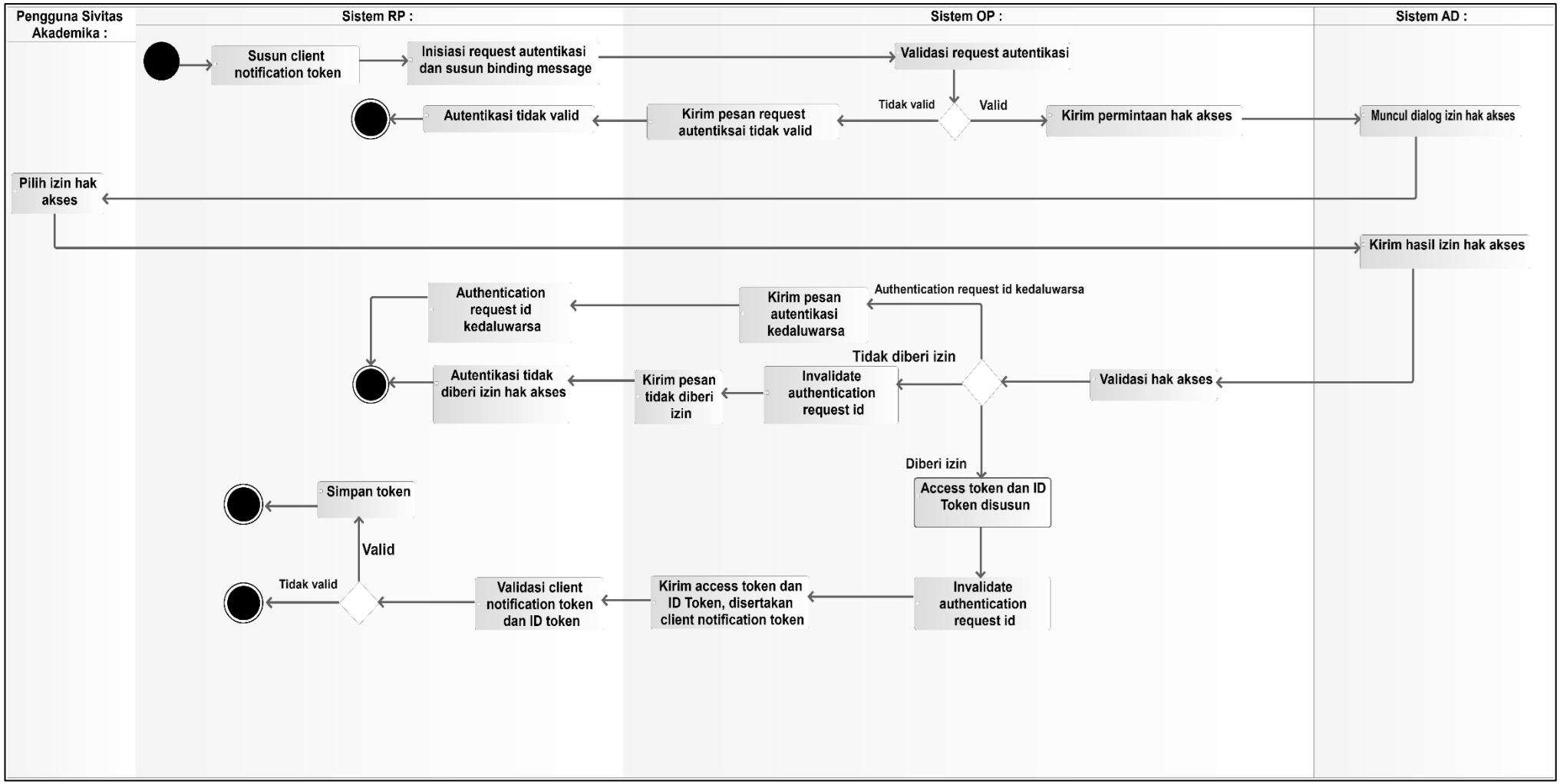
Pada mode token *push*, terdapat komunikasi langsung dari sistem OP ke sistem RP dalam pemberian hasil autentikasi dan token, berbeda dengan mode token *poll* dan *ping* yang harus mengeksekusi token *endpoint*. Kasus penggunaan ini memiliki karakteristik yang serupa dengan Kasus Penggunaan CIBA Menggunakan Mode Token *Ping* (UC-003), karena sistem OP memiliki prosedur untuk memberitahu sistem RP bahwa token dapat diambil. Oleh karena itu, mode token *push* dan *ping* melakukan validasi *request* yang terdiri dari *client notification token* dan *ID Token* untuk mencegah komunikasi yang berasal dari sistem OP yang tidak terpercaya.

Tabel 3.7 Rincian Kasus Penggunaan CIBA Menggunakan Mode Token *Push*

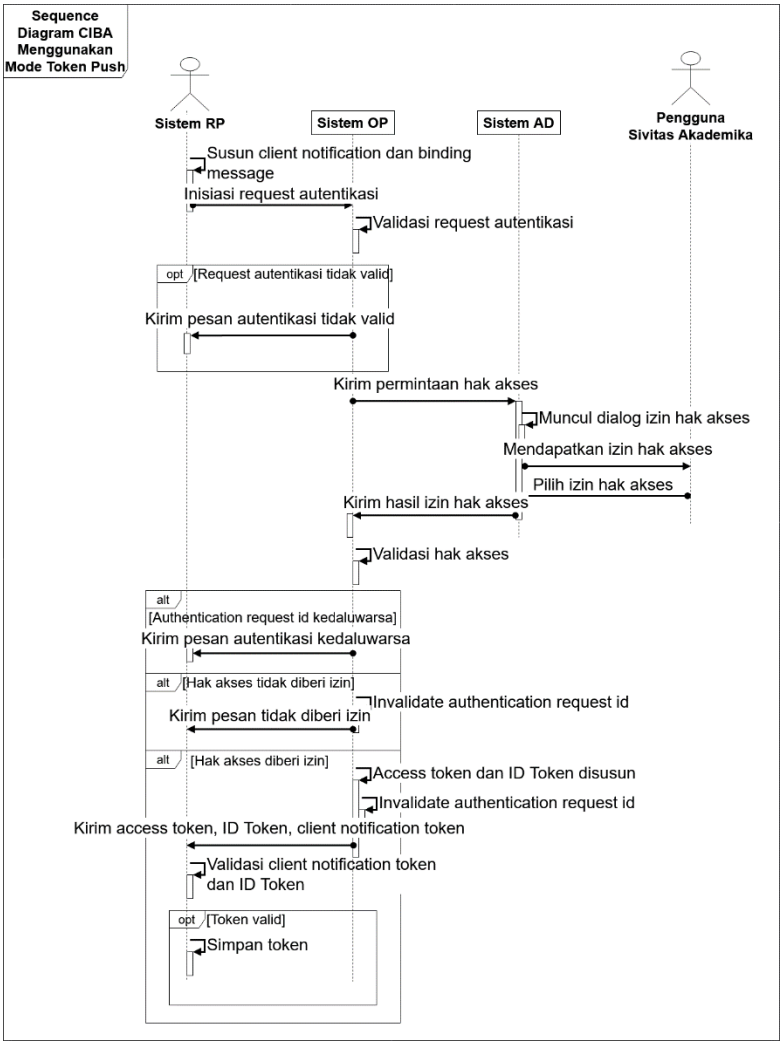
Nama	CIBA Menggunakan Mode Token *Push*
Nomor	UC-001
Deskripsi	Kasus penggunaan ini digunakan sebelum melakukan *impersonation* sebagai tanda bahwa proses *impersonation* diperbolehkan oleh pengguna sivitas akademika.
Tipe	Fungsional
Aktor	Sistem RP
Kondisi Awal	Terdapat Aplikasi Klien dan Server Autorisasi yang terdaftar menggunakan protokol CIBA.
Alur Normal	1. Sistem RP menyusun *client notification token*. 2. Sistem RP menginisasi *request* autentikasi dan menyusun *binding message*. 3. Sistem OP melakukan validasi terhadap *request* yang masuk. 4. Sistem OP mengirim permintaan persetujuan hak akses ke sistem AD. 5. Sistem AD memunculkan *dialog* izin hak akses. 6. Persetujuan hak akses diizinkan oleh pengguna sivitas akademika. 7. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 8. Sistem OP melakukan validasi hak akses. 9. Sistem OP menyusun token. 10. Sistem OP melakukan *invalidate authentication request id*.

	11. Sistem OP mengirim token ke sistem RP melalui *client notification endpoint* disertakan *client notification token*. 12. Sistem RP melakukan validasi *client notification token* dan *ID Token*. 13. Sistem RP menyimpan token. 14. Kasus penggunaan berakhir.
Eksepsi	3a. *Request* tidak valid. 3a.1. Terjadi salah satu skenario diantara berikut dalam *request* autentikasi: metode autentikasi aplikasi klien salah atau tidak terdaftar, tidak ditemukan akun pengguna yang dituju, *scope* tidak didukung, *binding message* tidak sesuai format, atau *request* parameter yang wajib tidak lengkap. 3a.2. Sistem OP mengirim *request* ke sistem RP bahwa *request* autentikasi tidak valid. 3a.3. *Request* autentikasi diberhentikan. 8a. Persetujuan hak akses tidak diizinkan oleh pengguna sivitas akademika. 8a.1. Sistem OP melakukan *invalidate authentication request id*. 8a.2. Sistem OP mengirim hasil autentikasi ke sistem RP bahwa pengguna sivitas akademika tidak memberi izin hak akses. 8b. Pengguna sivitas akademika terlalu lama untuk memberi izin hak akses sehingga *authentication request id* kedaluwarsa 8b.1. Sistem OP mengirim *request* ke sistem RP bahwa *authentication request id* telah kedaluwarsa. 8b.2. *Request* autentikasi diberhentikan.

	12a. *Client notification token* tidak valid 12.1 *Request* autentikasi diberhentikan. 12b. *ID Token* tidak valid 12b.1 *Request* autentikasi diberhentikan.
Alur Alternatif	-
Kondisi Akhir	Sistem RP mendapatkan token.



Gambar 3.3 Aktivitas Diagram Kasus Penggunaan CIBA Menggunakan Mode Token *Push*



Gambar 3.4 Diagram Sekuensial CIBA Menggunakan Mode Token *Push*

3.2.1.2 Kasus Penggunaan CIBA Menggunakan Mode Token *Poll* (UC-002)

Pada kasus penggunaan ini, pengguna dapat menjalankan CIBA menggunakan mode token *poll*. Kasus penggunaan ini bekerja sama dengan proses *impersonate* sebagai tanda sistem mendapatkan izin hak akses oleh pengguna sivitas akademika.
Kasus penggunaan ini memiliki karakteristik yang serupa dengan mode token *ping* bahwa kedua mode token tersebut mengambil token melalui token *endpoint*. Yang membuat mode token *poll* menonjol adalah prosedur eksekusi token *endpoint* berulang-ulang sampai mendapatkan token. Hal ini dapat mengakibatkan server menjadi kewalahan, oleh karena itu terdapat pengecekan jarak waktu *polling*.

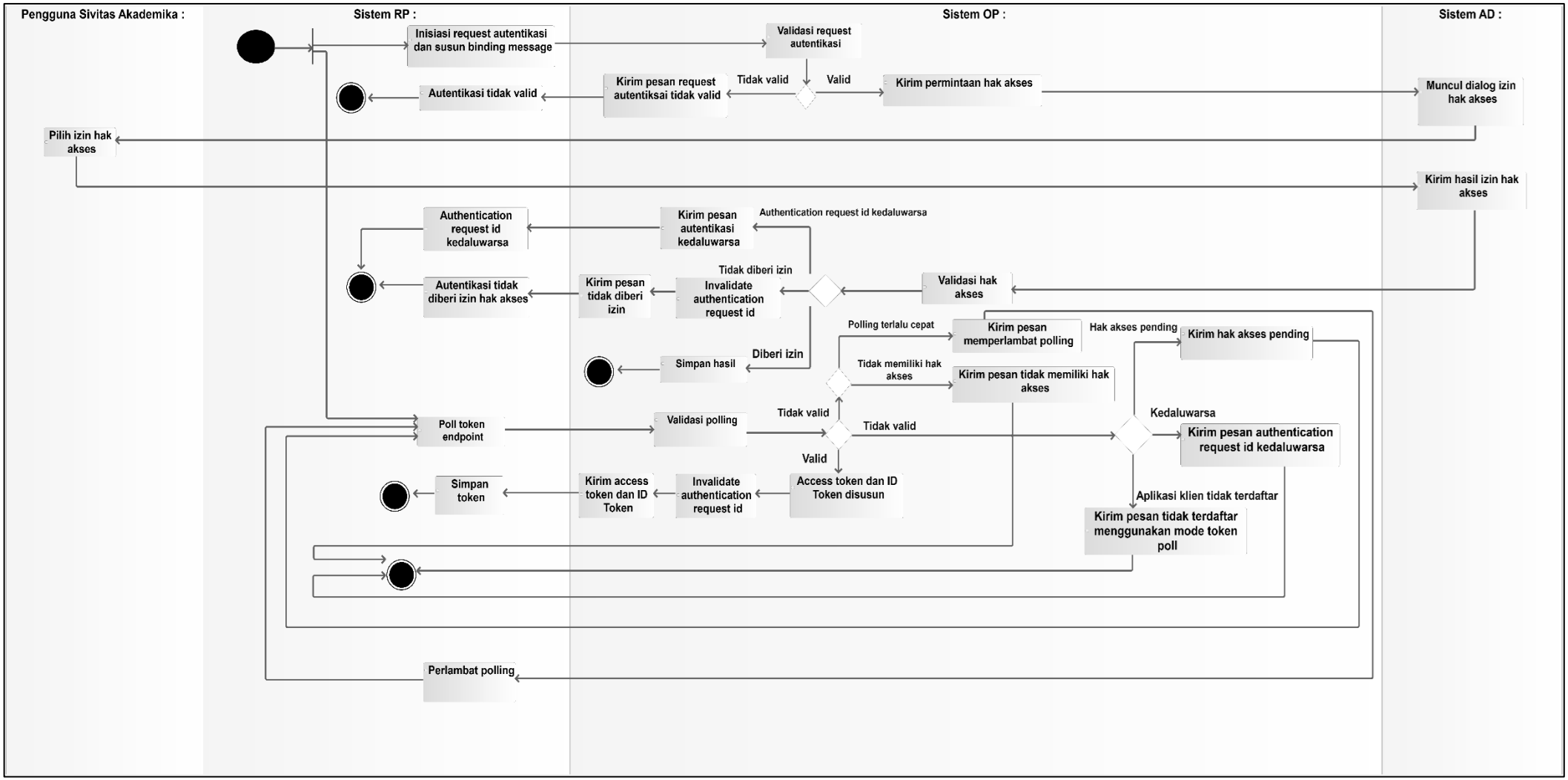
Tabel 3.8 Rincian Kasus Penggunaan CIBA Menggunakan Mode Token *Poll*

Nama	CIBA Menggunakan Mode Token *Poll*
Nomor	UC-002
Deskripsi	Kasus penggunaan ini digunakan sebelum melakukan *impersonation* sebagai tanda bahwa proses *impersonation* diperbolehkan oleh pengguna sivitas akademika.
Tipe	Fungsional
Aktor	Sistem RP
Kondisi Awal	Terdapat Aplikasi Klien dan Server Autorisasi yang terdaftar menggunakan protokol CIBA.
Alur Normal	1. Sistem RP menginisiasi *request* autentikasi dan menyusun *binding message*. 2. Sistem OP melakukan validasi terhadap *request* yang masuk.

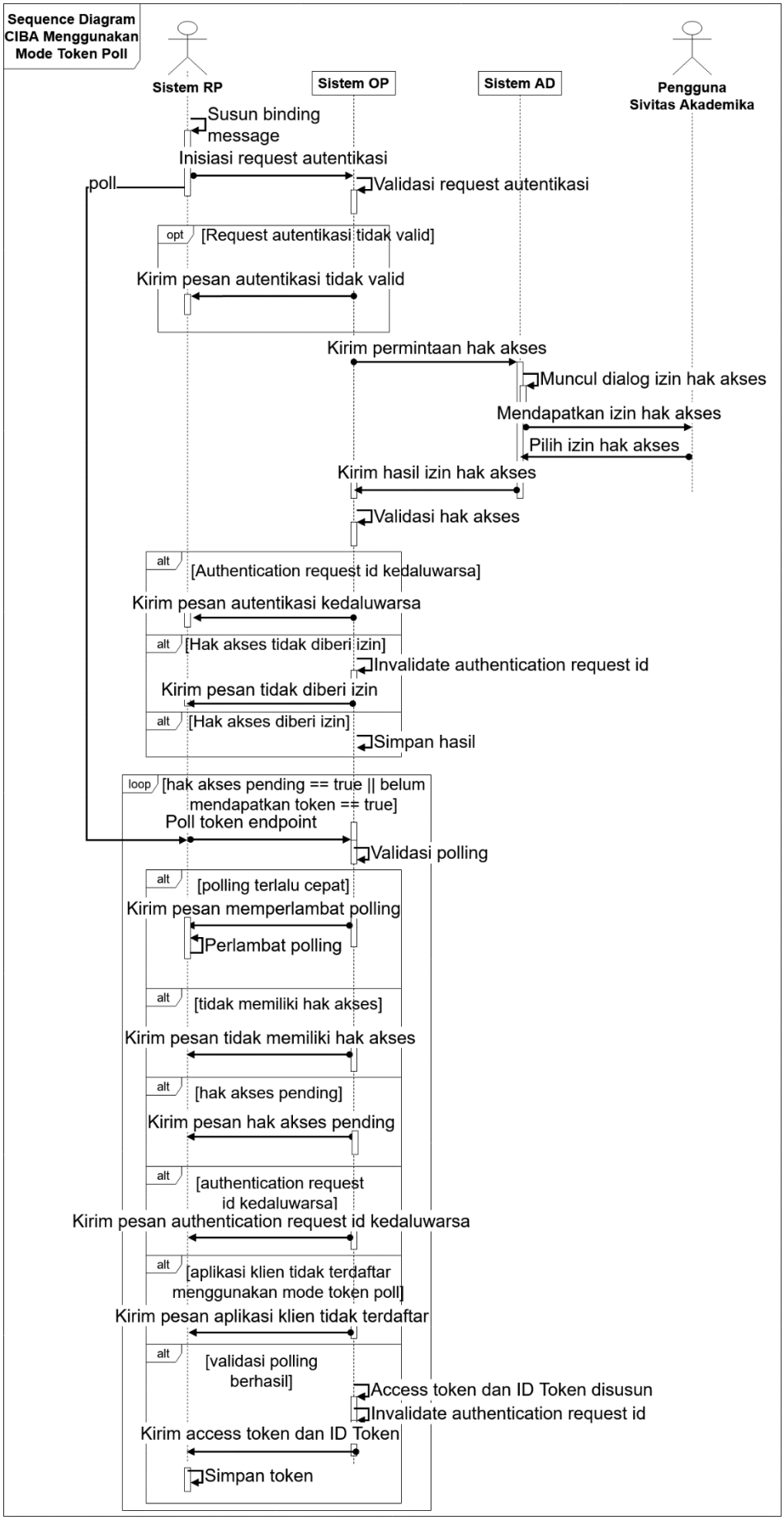
	3. Sistem OP mengirim permintaan persetujuan hak akses ke sistem AD. 4. Sistem AD memunculkan *dialog* izin hak akses. 5. Persetujuan hak akses diizinkan oleh pengguna sivitas akademika. 6. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 7. Sistem OP melakukan validasi hak akses, dan menyimpan hasil. 8. Sistem RP melakukan *polling* ke token *endpoint*. 9. Sistem OP melakukan validasi *polling*. 10. Sistem OP menyusun token. 11. Sistem OP melakukan *invalidate authentication request id*. 12. Sistem OP mengirim token ke sistem RP. 13. Sistem RP menyimpan token. 14. Kasus penggunaan berakhir.
Eksepsi	2a. *Request* tidak valid. 2a.1. Terjadi salah satu skenario diantara berikut dalam *request* autentikasi: metode autentikasi aplikasi klien salah atau tidak terdaftar, tidak ditemukan akun pengguna yang dituju, *scope* tidak didukung, *binding message* tidak sesuai format, atau *request* parameter yang wajib tidak lengkap. 2a.2. Sistem OP mengirim *request* ke sistem RP bahwa *request* autentikasi tidak valid. 2a.3. *Request* autentikasi diberhentikan. 9a. Persetujuan hak akses tidak diizinkan oleh pengguna sivitas akademika. 9a.1. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP.

	9a.2. Sistem OP melakukan *invalidate authentication request id*. 9a.3. Sistem OP mengirim hasil autentikasi ke sistem RP bahwa pengguna sivitas akademika tidak memberi izin hak akses. 9b. Pengguna sivitas akademika terlalu lama untuk memberi izin hak akses sehingga *authentication request id* kedaluwarsa 9b.1. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 9b.2. Sistem OP mengirim *request* ke sistem RP bahwa *authentication request id* telah kedaluwarsa. 9b.3. *Request* autentikasi diberhentikan. 9c. Sistem RP tidak terdaftar menggunakan mode token *poll* 9c.1. Sistem OP mengirim *request* ke sistem RP pemberitahuan tidak terdaftar mode token *poll*. 9c.2. *Request* autentikasi diberhentikan. 9d. Sistem RP tidak punya izin hak akses 9d.1. Sistem OP mengirim *request* ke sistem RP berupa pemberitahuan tidak memiliki hak akses. 9d.1 *Request* autentikasi diberhentikan. 9e. Sistem RP menggunakan *authentication request id* yang kedaluwarsa 9e.1. Sistem OP mengirim *request* berupa pemberitahuan *authentication request id* telah kedaluwarsa ke sistem RP. 9e.2. *Request* autentikasi diberhentikan.

Alur Alternatif	9f. Sistem RP terlalu cepat melakukan *polling* 9f.1. Sistem OP mengirim *request* berupa pemberitahuan untuk memperlambat *polling* ke sistem RP. 9f.2 Sistem RP memperlambat kecepatan *polling* dengan cara memperbesar jarak waktu *polling*. 9f.3. Kembali ke alur normal langkah 8. 9g. Izin hak akses masih *pending* 9g.1. Sistem OP kirim pesan hak akses *pending* ke sistem RP. 9g.2. Kembali ke alur normal langkah 8.
Kondisi Akhir	Sistem RP mendapatkan token.



Gambar 3.5 Aktivitas Diagram Kasus Penggunaan CIBA Menggunakan Mode Token *Poll*



Gambar 3.6 Diagram Sekuensial CIBA Menggunakan Mode Token *Poll*

3.2.1.3 Kasus Penggunaan CIBA Menggunakan Mode Token *Ping* (UC-003)

Pada kasus penggunaan ini, pengguna dapat menjalankan CIBA menggunakan mode token *ping*. Kasus penggunaan ini bekerja sama dengan proses *impersonate* sebagai tanda sistem mendapatkan izin hak akses oleh pengguna sivitas akademika.

Kasus penggunaan ini memiliki karakteristik yang sama dengan mode token *push* bahwa sistem OP dapat berkomunikasi secara langsung dengan sistem RP untuk memberitahu hasil autentikasi. Mode token ini memiliki keunggulan dari mode token *poll* karena eksekusi token *endpoint* hanya dilakukan satu kali.

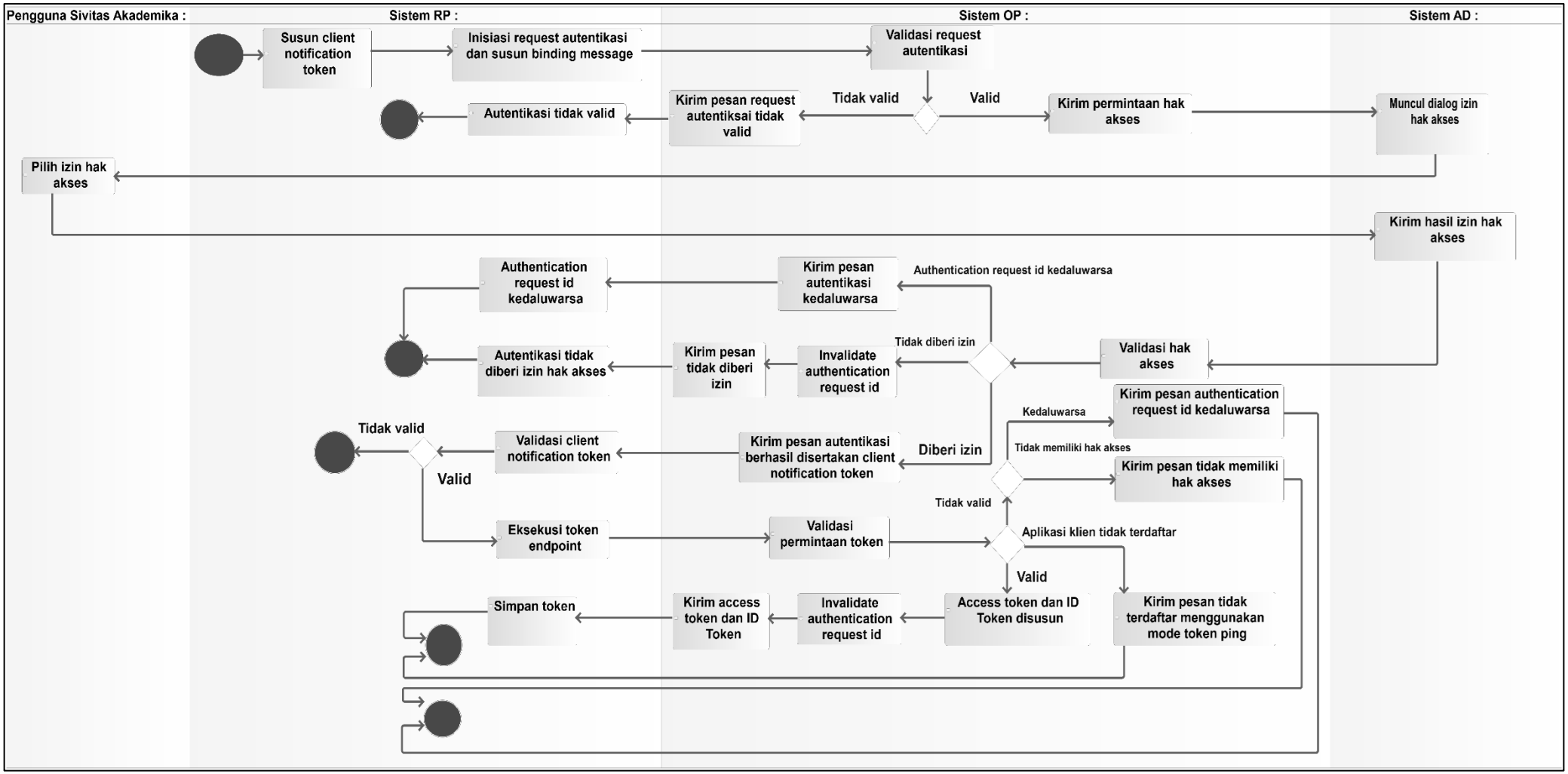
Tabel 3.9 Rincian Kasus Penggunaan CIBA Menggunakan Mode Token Ping

Nama	CIBA Menggunakan Mode Token *Ping*
Nomor	UC-003
Deskripsi	Kasus penggunaan ini digunakan sebelum melakukan *impersonation* sebagai tanda bahwa proses *impersonation* diperbolehkan oleh pengguna sivitas akademika.
Tipe	Fungsional
Aktor	Sistem RP
Kondisi Awal	Terdapat Aplikasi Klien dan Server Autorisasi yang terdaftar menggunakan protokol CIBA.
Alur Normal	1. Sistem RP menyusun *client notification token*. 2. Sistem RP menginisiasi *request* autentikasi dan menyusun *binding message*. 3. Sistem OP melakukan validasi terhadap *request* yang masuk. 4. Sistem OP mengirim permintaan persetujuan hak akses ke sistem AD.

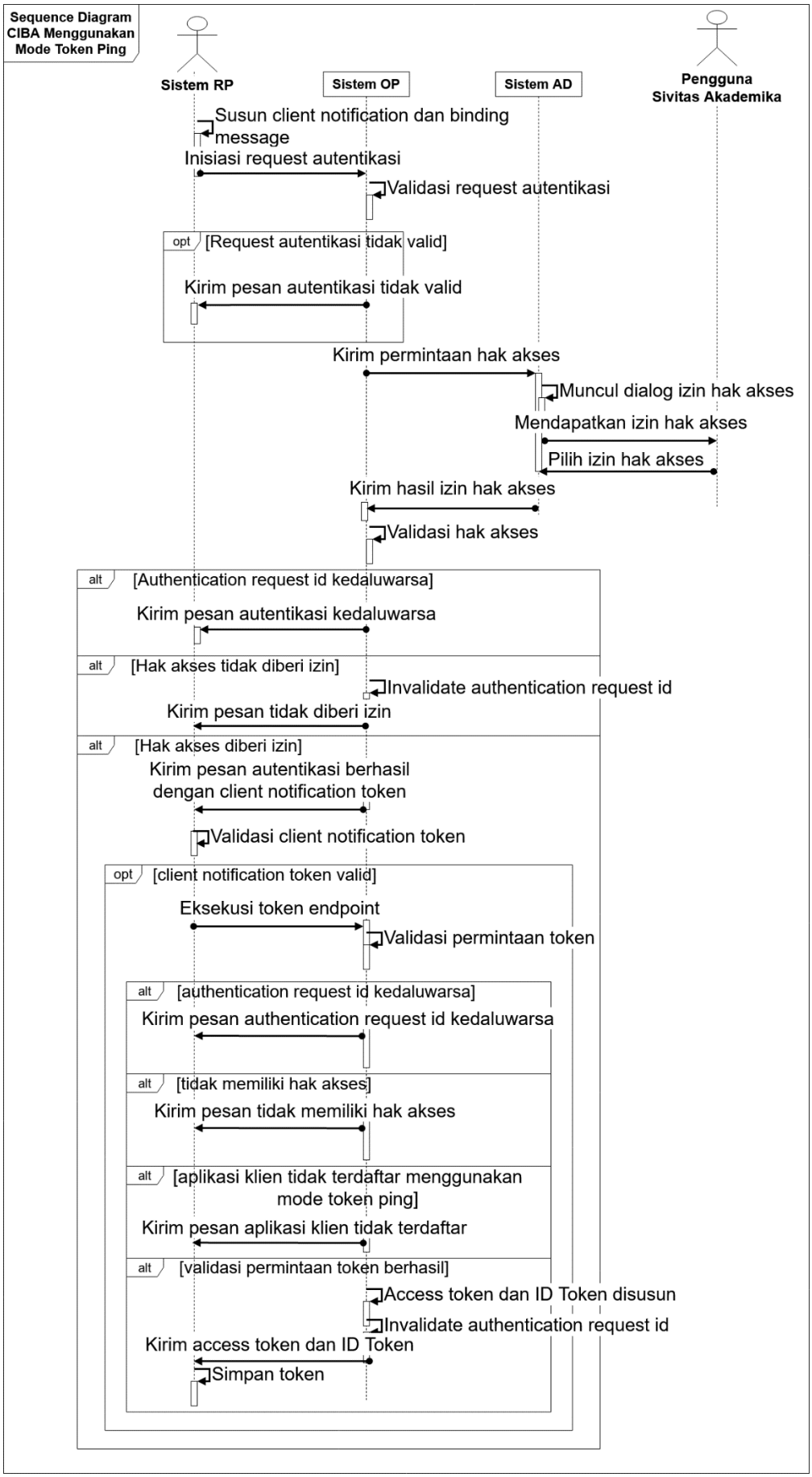
	5. Sistem AD memunculkan *dialog* izin hak akses. 6. Persetujuan hak akses diizinkan oleh pengguna sivitas akademika. 7. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 8. Sistem OP melakukan validasi hak akses, dan menyimpan hasil. 9. Sistem OP mengirim hasil autentikasi ke sistem RP melalui *client notification endpoint* disertakan *client notification token*. 10. Sistem RP melakukan validasi *client notification token*. 11. Sistem RP melakukan *request* ke token *endpoint*. 12. Sistem OP melakukan validasi permintaan token. 13. Sistem OP menyusun token. 14. Sistem OP melakukan *invalidate authentication request id*. 15. Sistem OP mengirim token ke sistem RP. 16. Sistem RP menyimpan token. 17. Kasus penggunaan berakhir.
Eksepsi	3a. *Request* tidak valid. 3a.1. Terjadi salah satu skenario diantara berikut dalam *request* autentikasi: metode autentikasi aplikasi klien salah atau tidak terdaftar, tidak ditemukan akun pengguna yang dituju, *scope* tidak didukung, *binding message* tidak sesuai format, atau *request* parameter yang wajib tidak lengkap. 3a.2. Sistem OP mengirim *request* ke sistem RP bahwa *request* autentikasi tidak valid. 3a.3. *Request* autentikasi diberhentikan.

	8a. Persetujuan hak akses tidak diizinkan oleh pengguna sivitas akademika. 8a.1. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 8a.2. Sistem OP melakukan *invalidate authentication request id*. 8a.3. Sistem OP mengirim hasil autentikasi ke sistem RP bahwa pengguna sivitas akademika tidak memberi izin hak akses. 8b. Pengguna sivitas akademika terlalu lama untuk memberi izin hak akses sehingga *authentication request id* kedaluwarsa 8b.1. Sistem AD mengirim *request* hasil izin persetujuan hak akses ke sistem OP. 8b.2. Sistem OP mengirim *request* ke sistem RP bahwa *authentication request id* telah kedaluwarsa. 8b.3. *Request* autentikasi diberhentikan. 10a. *Client notification token* tidak valid 10a.1 *Request* autentikasi diberhentikan. 12a. Sistem RP tidak terdaftar menggunakan mode token *ping* 12a.1. Sistem OP mengirim *request* ke sistem RP pemberitahuan tidak terdaftar mode token *ping*. 12a.2. *Request* autentikasi diberhentikan. 12b. Sistem RP tidak punya izin hak akses 12b.1. Sistem OP mengirim *request* ke sistem RP berupa pemberitahuan tidak memiliki hak akses.

	12b.2 *Request* autentikasi diberhentikan. 12c. Sistem RP menggunakan *authentication request id* yang kedaluwarsa 12c.1. Sistem OP mengirim *request* berupa pemberitahuan *authentication request id* telah kedaluwarsa ke sistem RP. 12c.2. *Request* autentikasi diberhentikan.
Alur Alternatif	-
Kondisi Akhir	Sistem RP mendapatkan token.



Gambar 3.7 Aktivitas Diagram Kasus Penggunaan CIBA Menggunakan Mode Token *Ping*



Gambar 3.8 Diagram Sekuensial CIBA Menggunakan Mode Token *Ping*

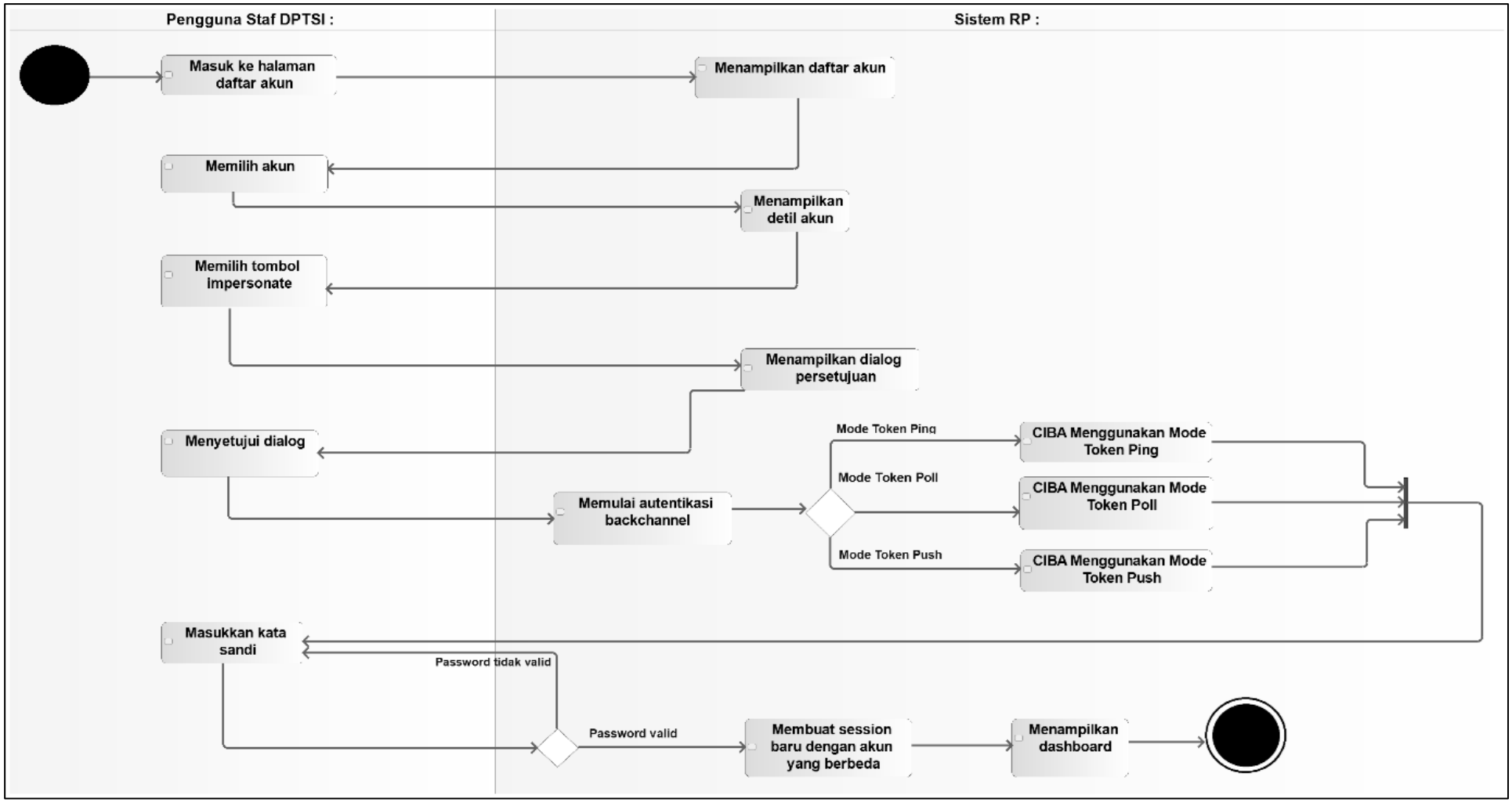
3.2.1.4 Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004)

Pada kasus penggunaan ini, pengguna dapat melakukan autentikasi atas akun yang berbeda, namun harus mendapatkan izin hak akses oleh pemilik akun yang dipilih. Untuk mendapatkan izin hak akses, kasus penggunaan ini dapat memanfaatkan Kasus Penggunaan CIBA Menggunakan Mode Token *Push* (UC-001), Kasus Penggunaan CIBA Menggunakan Mode Token *Poll* (UC-002), dan Kasus Penggunaan CIBA Menggunakan Mode Token *Ping* (UC-003). Kasus penggunaan ini diinisiasi oleh aktor staf DPTSI dan melakukan *include* dari salah satu kasus penggunaan diantara tiga berikut: Kasus Penggunaan CIBA Menggunakan Mode Token *Push* (UC-001), Kasus Penggunaan CIBA Menggunakan Mode Token *Poll* (UC-002), dan Kasus Penggunaan CIBA Menggunakan Mode Token *Ping* (UC-003).

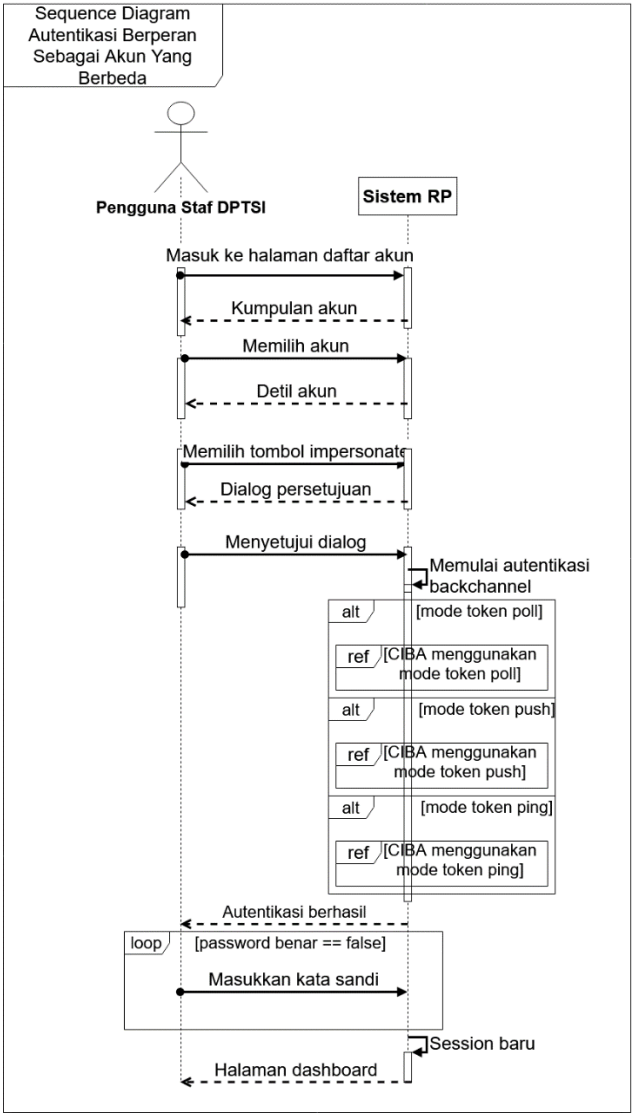
Tabel 3.10 Rincian Kasus Penggunaan Melakukan Autentikasi Berperan Sebagai Akun Yang Berbeda

Nama	Melakukan Autentikasi Berperan Sebagai Akun Yang Berbeda
Nomor	UC-004
Deskripsi	Kasus penggunaan ini digunakan untuk berperan sebagai akun yang berbeda. Pada umumnya digunakan ketika ada sivitas akademika yang memiliki keluhan *error* sehingga pengguna Staf DPTSI dapat masuk ke akun pengguna sivitas akademika untuk melihat masalahnya secara langsung.
Tipe	Fungsional
Aktor	Staf DPTSI
Kondisi Awal	Terdapat akun pengguna yang ingin diperan.
Alur Normal	1. Pengguna masuk ke halaman daftar akun pengguna. 2. Sistem menampilkan daftar akun pengguna.

	3. Pengguna memilih akun pengguna. 4. Sistem menampikan detil akun pengguna. 5. Pengguna memilih tombol *impersonate*. 6. Sistem menampilkan *dialog* persetujuan. 7. Pengguna menyetujui *dialog* persetujuan. 8. Sistem memulai autentikasi *backchannel*. 9. Pengguna memasukkan kata sandi. 10. Sistem membuat *session* baru dengan akun yang berbeda. 11. Sistem menampilkan *dashboard*. 12. Kasus penggunaan berakhir.
Eksepsi	6a. Pengguna tidak meyetujui *dialog*. 6a.1. Kasus penggunaan dihentikan.
Alur Alternatif	8a. Eksekusi CIBA menggunakan mode token *push* 8a.1. *Include* Kasus Penggunaan CIBA Menggunakan Mode Token *Push* (UC-001) 8b. Eksekusi CIBA menggunakan mode token *poll* 8b.1. *Include* Kasus Penggunaan CIBA Menggunakan Mode Token *Poll* (UC-002) 8c. Eksekusi CIBA menggunakan mode token *ping* 8c.1. *Include* Kasus Penggunaan CIBA Menggunakan Mode Token *Ping* (UC-003) 9d. Password yang diberikan salah 9d.1. Kembali ke langkah normal nomor 9.
Kondisi Akhir	Pengguna masuk dengan sesi baru dengan akun yang berbeda.



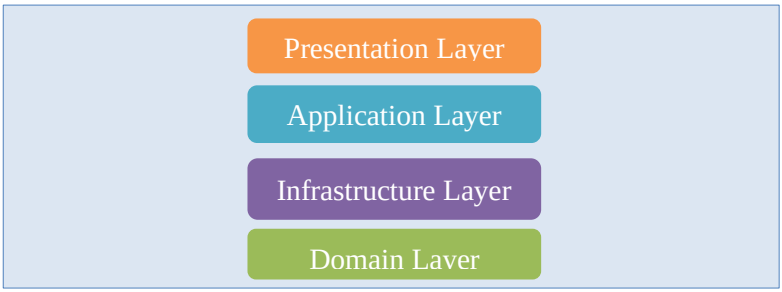
Gambar 3.9 Aktivitas Diagram Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda



Gambar 3.10 *Sequence Diagram* Autentikasi Berperan Sebagai Akun Yang Berbeda

3.2.2 Perancangan Arsitektur

Pengerjaan tugas akhir ini akan menggunakan arsitektur dengan lapisan *presentation*, *application*, *infrastructure* dan *domain*. Arsitektur aplikasi server autorisasi CIBA yang dikembangkan menggunakan kerangka kerja Phalcon akan berperan pada lapisan *presentation* dan *infrastructure*. Aplikasi ini digunakan untuk memberi layanan protokol CIBA dengan cara membuka akses melalui rute yang didefiniskan pada konfigurasi kerangka kerja.



Gambar 3.11 Lapisan Arsitektur Perangkat Lunak

Sedangkan *library* server autorisasi CIBA yang dibangun menggunakan bahasa pemrograman PHP dan terlepas dari kerangka kerja Phalcon memiliki empat lapisan, yaitu *presentation*, *application*, *infrastructure*, dan *domain*. Kelas yang terdapat pada *library* ini memiliki tanggung jawab yang lebih spesifik pada setiap lapisan yang akan dibahas di subbab 3.2.3.1. Pada umumnya, kerangka kerja aplikasi server autorisasi akan memanggil fungsi-fungsi *library* server autorisasi CIBA di lapisan *presentation*.

Fungsi dari setiap lapisan dijelaskan sebagai berikut.

3.2.2.1 *Presentation Layer*

Lapisan ini merupakan lapisan yang bertanggung jawab dengan tampilan antarmuka. Komponen yang dinamakan sebagai

Controller dan *View* berada pada lapisan ini. Semua *request* yang masuk ke sistem akan melalui suatu rute yang tersambung ke *Controller*. Validasi *request* agar parameter yang dibutuhkan sesuai dengan peraturan dilakukan oleh lapisan ini. *Controller* lalu meneruskannya ke lapisan *application* yang memiliki logika aplikasi. *Controller* dapat menampilkan antarmuka dalam bentuk HTML atau data mentah dalam bentuk JSON.

3.2.2.2 *Application Layer*

Lapisan ini merupakan penghubung antara lapisan *presentation* dan *infrastructure*. Pada umumnya, logika dari perangkat lunak ditangani oleh lapisan ini.

3.2.2.3 *Infrastructure Layer*

Lapisan ini memiliki tanggung jawab untuk menangani kebutuhan yang berkaitan dengan sistem eksternal perangkat lunak. Sebagai contoh operasi basis data yang membutuhkan koneksi, pemanggilan API melalui HTTP, komunikasi menggunakan *message broker* dan lain-lain. Kode yang membutuhkan sumber sarana infrastruktur diletakkan pada lapisan ini.

3.2.2.4 Domain Layer

Lapisan ini terdiri dari obyek yang membungkus aturan proses bisnis yang murni, dan abstraksi yang berkaitan dengan aturan proses bisnis. Pada umumnya, lapisan ini tidak memiliki ketergantungan secara langsung terhadap infrastruktur.

3.2.3 Perancangan Kelas

Pada subbab ini dijelaskan mengenai rancangan kelas untuk aplikasi yang akan dibangun. Perancangan kelas akan dibagi menjadi dua bagian sesuai dengan jumlah aplikasi yang dibangun, yaitu perancangan kelas *library* server autorisasi CIBA dan aplikasi server autorisasi yang menyediakan layanan CIBA. Sebagai catatan, saat fase perancangan dan implementasi protokol

CIBA, kata *Ciba* memiliki makna yang sama dengan CIBA, yang dapat ditemukan pada nama kelas, fungsi, dan atribut, karena menggunakan konvensi penamaan *pascal case*.

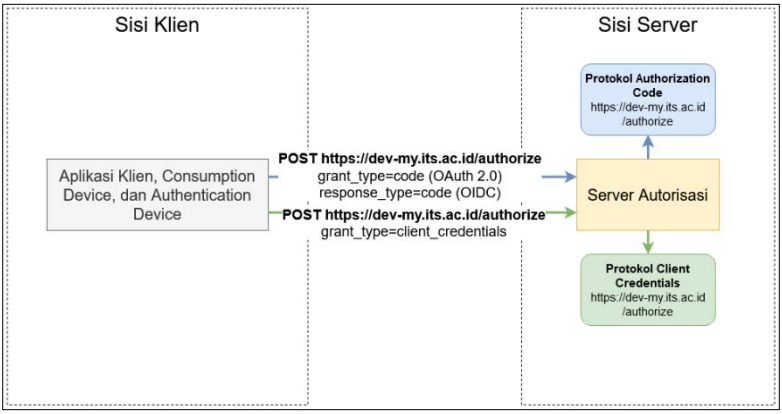
3.2.3.1 Perancangan Kelas *Library* Server Autorisasi

Perancangan kelas *library* server CIBA dibuat sedemikian rupa sehingga *library* server CIBA tidak mengalami *breaking changes* atau kerusakan yang disebabkan oleh penambahan kelas, fungsi, dan atribut. Perlu digarisbawahi bahwa *library* server CIBA akan dikembangkan melalui *library Bshaffer OAuth 2.0* yang sudah memiliki fungsi untuk menangani protokol seperti *Authorization Code, User Credentials*, dan *Client Credentials*. Oleh karena itu, penambahan protokol CIBA tidak boleh menghambat fungsi-fungsi yang sudah tertanam. Apabila penambahan protokol CIBA menghambat protokol yang lain, maka proses bisnis pada server autorisasi myITS SSO akan mengalami kerusakan.

Mekanisme komunikasi klien dan server tanpa protokol CIBA dapat dilihat pada Gambar 3.12. Terdapat pemisahan sisi klien dan server. Sisi klien akan mengonsumsi layanan autentikasi dan autorisasi yang disediakan oleh server autorisasi. Dapat dilihat bahwa server autorisasi myITS SSO menerapkan protokol *Authorization Code* dan *Client Credentials*. Kedua protokol tersebut memiliki *endpoint* yang sama, yaitu "*https://dev-my.its.ac.id/authorize"*. Meskipun *endpoint* yang digunakan sama, server autorisasi dapat membedakan protokol yang dituju, yaitu dengan menggunakan parameter *grant_type* dan *response_type*. Parameter *grant_type* digunakan apabila protokol yang dituju merupakan alur dari kerangka kerja *OAuth 2.0*. Sedangkan parameter *response_type* digunakan apabila protokol yang dituju merupakan alur dari kerangka kerja OIDC. Dapat dilihat nilai dari parameter *grant_type* dan *response_type* pada Tabel 3.11. Dengan menggunakan kedua parameter tersebut untuk membedakan protokol yang dituju, maka protokol yang diterapkan dapat terhindar dari *breaking changes*.

Tabel 3.11 Nilai Parameter *grant_type* dan *response_type*

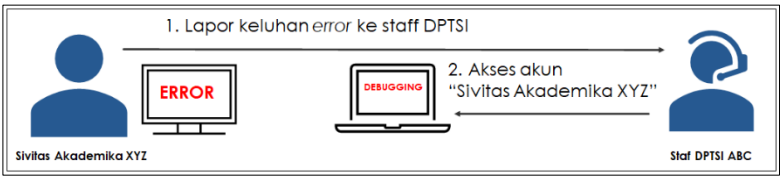
Protokol	**Nilai *grant_type***	**Nilai *response_type***
Authorization Code	*code*	*code*
Client Credentials	*client_credentials*	-
User Credentials	*password*	-



Gambar 3.12 Server Autorisasi Tanpa Protokol CIBA

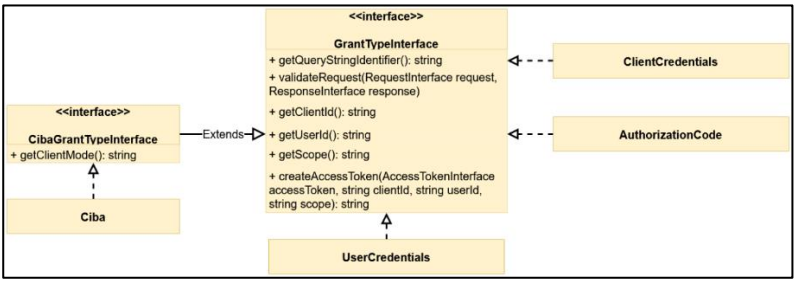
Proses bisnis yang berjalan di DPTSI ITS, *customer care*, dapat dilihat pada Gambar 3.13. Aktor sivitas akademika (pelapor) melapor keluhan *error* yang dialami kepada staf DPTSI. Agar staf DPTSI dapat mengetahui *error* yang dialami oleh sivitas akademika, staf DPTSI harus melakukan *debugging* dengan cara masuk ke akun pelapor. Tanpa ada protokol CIBA, staf DPTSI dapat langsung masuk ke akun pelapor tanpa izin hak akses oleh pelapor. Autentikasi yang digunakan untuk masuk ke akun pelapor menggunakan protokol *Authorization Code*. Proses ini dapat diperbaiki agar lebih etis dengan menerapkan protokol CIBA, agar

saat mengakses akun pelapor, pelapor dapat memberi izin hak akses.



Gambar 3.13 Proses Bisnis Tanpa CIBA

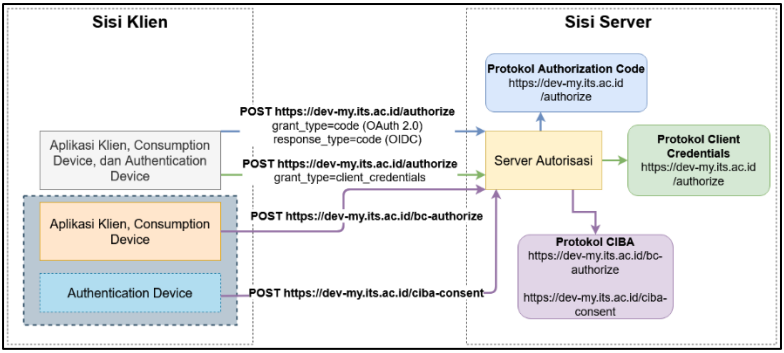
Penambahan protokol CIBA dapat memanfaatkan abstraksi (pada bahasa pemrograman PHP abstraksi direpresentasikan menggunakan *interface*) yang disediakan oleh *library Bshaffer OAuth 2.0*. Dapat dilihat pada Gambar 3.14 bahwa terdapat abstraksi yang dinamakan dengan *Grant Type Interface*. Abstraksi ini berisi fungsi-fungsi umum yang dapat menangani alur autentikasi dan autorisasi. Alur autentikasi dan autorisasi memiliki berbagai macam tipe seperti *Authorization Code*, *User Credentials*, *Client Credentials*, dan CIBA. Dengan memanfaatkan abstraksi *Grant Type Interface*, masing-masing implementasi alur autentikasi dan autorisasi dapat berjalan tanpa menghambat protokol lainnya. Keunggulan menggunakan abstraksi telah dijelaskan subbab 2.6.1.1.



Gambar 3.14 Perancangan Abstraksi dan Kelas *Grant Type*

Dengan menerapkan protokol CIBA, mekanisme komunikasi untuk menggunakan protokol CIBA dapat diakses

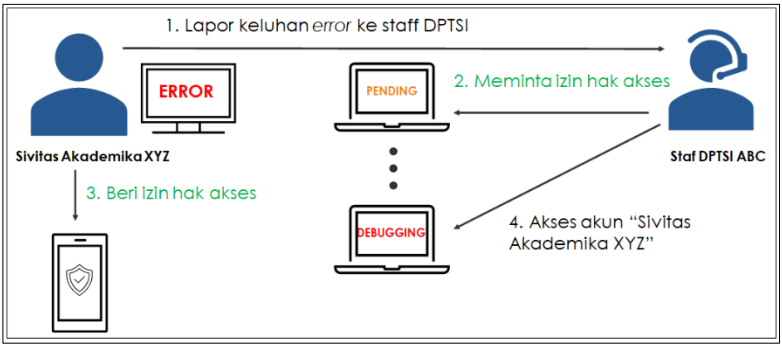
menggunakan *endpoint* yang berbeda, yaitu "*https://dev-my.its.ac.id/bc-authorize*". Pendekatan ini dilakukan karena protokol CIBA tidak menyediakan parameter *grant_type* atau *response_type*. Sehingga dengan memiliki *endpoint* tersendiri, server autorisasi dapat mengetahui bahwa *request* yang masuk ke *endpoint "https://dev-my.its.ac.id/bc-authorize"* memiliki tujuan untuk menggunakan CIBA. Selain *endpoint* autentikasi yang berbeda, CIBA menambahkan mekanisme untuk memberi izin hak akses. Proses ini dapat diakses menggunakan *endpoint "https://dev-my.its.ac.id/ciba-consent"* yang digunakan oleh AD. Sistem dengan protokol CIBA dapat dilihat pada Gambar 3.15. Perbedaan yang signifikan terletak pada mekanisme autentikasi antara aplikasi klien dengan server autorisasi, dan pemberian izin hak akses antara AD dan server autorisasi.



Gambar 3.15 Server Autorisasi Dengan Protokol CIBA

Akhirnya proses bisnis *customer care* yang berjalan di DPTSI ITS, dapat memanfaatkan protokol CIBA. Protokol CIBA memiliki peran untuk mendapatkan hak akses dari pelapor. Pemberian hak akses menjamin bahwa pelapor memiliki pengetahuan bahwa ada entitas yang dapat mengakses akunnya, dengan batasan yang sudah terdefinisi, sehingga keamanan dan privasi tetap terjamin. Pada Gambar 3.16, staf DPTSI meminta izin hak akses, dan sivitas akademika dapat memberi izin hak akses

pada AD yang dimilikinya. Apabila hak akses sudah diberikan, maka proses *debugging* oleh staf DPTSI dapat dimulai. Ini berbeda dengan proses bisnis tanpa CIBA, dimana hak akses dari pelapor tidak berkontribusi pada proses mengakses akun.



Gambar 3.16 Proses Bisnis Dengan CIBA

Langkah perancangan kelas *library* server CIBA akan dimulai dengan mengobservasi *library* yang sudah ada yaitu *library Bshaffer OAuth 2.0*, lalu menghasilkan kelas yang didapatkan dari pengetahuan observasi.

3.2.3.1.1 Observasi Awal Struktur *Library Bshaffer OAuth 2.0*

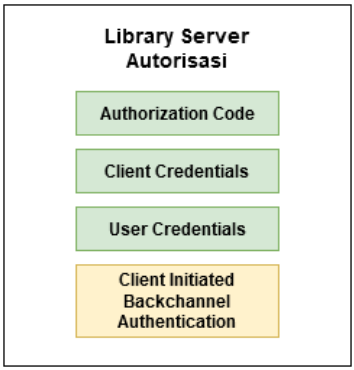
Pertama, akan dibahas fitur yang tersedia pada *library Bshaffer OAuth 2.0*. Fitur yang tersedia pada *library* tersebut dapat dilihat pada Tabel 3.12. *Library* ini tidak menyediakan fitur yang mendukung alur CIBA. Oleh karena itu, diperlukan perancangan yang matang agar dapat mempermudah proses implementasi.

Tabel 3.12 Fitur Pada *Library Bshaffer OAuth 2.0*

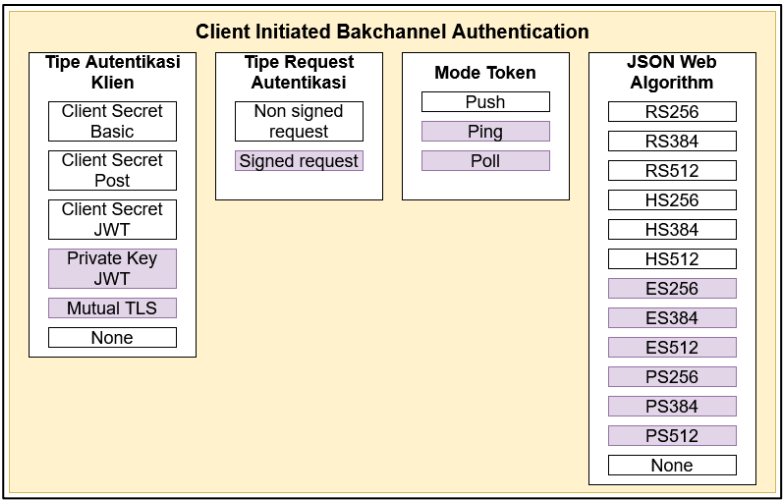
Nama Fitur	Status
Alur *Authorization Code OAuth 2.0*	Tersedia
Alur *Authorization Code OIDC*	Tersedia

Alur *Client Credentials*	Tersedia
Alur *User Credentials*	Tersedia
Alur CIBA Mode Token *Poll*	Tidak Tersedia
Alur CIBA Mode Token *Ping*	Tidak Tersedia
Alur CIBA Mode Token *Push*	Tidak Tersedia

Pada Gambar 3.17 dan Gambar 3.18 terdapat rinci mekanisme yang berkerja sama untuk menghasilkan protokol CIBA. Mekanisme tersebut terdiri dari tipe autentikasi klien, tipe *request* autentikasi, mode token, dan JSON Web Algorithm (JWA). Spesifikasi CIBA memberi fleksibilitas untuk dapat memilih masing-masing mekanisme yang tersedia sehingga tidak harus mengimplementasi seluruh mekanisme yang ada. Sebagai contoh apabila untuk tipe autentikasi klien ingin mengimplementasi *client secret basic* protokol CIBA tetap valid. Mekanisme yang diwarnai dengan warna ungu adalah mekanisme yang menghasilkan protokol *Financial Grade API CIBA (*FAPI CIBA) [29]. Protokol FAPI CIBA merupakan protokol yang berasal dari CIBA dan memiliki fondasi yang sama. Namun hanya memiliki mekanisme kemanan yang berbeda. FAPI CIBA dapat dikatan sebagai protokol CIBA yang telah memenuhi parameter yang sudah ditentukan seperti menggunakan tipe autentikasi klien *private key JWT*, memanfaatkan *signed request*, tidak menggunakan mode token *push*, dan menggunakan algoritma PS256 atau ES256 keatas. Tugas akhir ini tidak mengimplementasi FAPI CIBA.



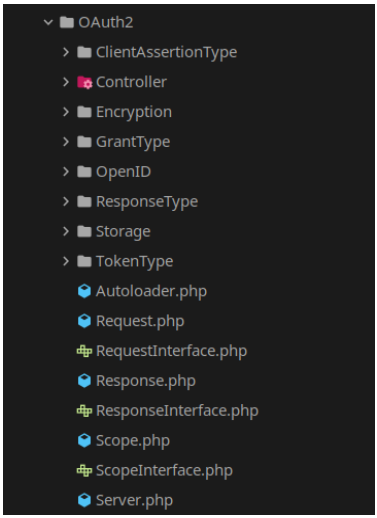
Gambar 3.17 Protokol Pada Library Server Autorisasi



Gambar 3.18 Perincian Protokol CIBA

Selanjutnya, akan dibahas struktur *library Bshaffer OAuth 2.0* yang dapat dilihat pada Gambar 3.19. *Library* tersebut memiliki direktori utama dengan nama *OAuth2*. Guna mengobservasi struktur *library* adalah untuk mengetahui fungsi masing-masing *folder*, konvensi pemrograman, struktur kode

sumber, alur pemanggilan fungsi, dan ketergantungan setiap lapisan sehingga pengetahuan *domain* akan bertambah. Oleh karena itu, dengan mengetahui struktur dan cara kerja *library* secara mendalam akan mempermudah penambahan protokol CIBA. Penjelasan masing-masing *folder* dapat dilihat pada Tabel 3.13.



Gambar 3.19 Struktur Direktori *Library Bshaffer OAuth 2.0.*

Tabel 3.13 Penjelasan *Folder* Pada Direktori *OAuth2*

Nama Folder	Deskripsi
ClientAssertionType	*Folder* yang mengelompokkan tipe autentikasi klien.
Controller	*Folder* yang mengelompokkan layanan protokol OAuth 2.0 dan OIDC. Dilakukan validasi *request* yang masuk sebelum didelegasikan ke lapisan selanjutnya.

Encryption	*Folder* yang mengelompokkan algoritma enkripsi dan *digital signing*.
GrantType	*Folder* yang mengelompokkan alur protokol.
OpenID	*Folder* yang mengelompokkan komponen-komponen yang spesifik ke OpenID.
ResponseType	*Folder* yang mengelompokkan bentuk obyek *response*.
Storage	*Folder* yang mengelompokkan tanggung jawab penyimpanan dan pengambilan data.
TokenType	*Folder* yang mengelompokkan tipe token yang digunakan pada HTTP *request*.

Berdasarkan penjabaran *folder* pada Tabel 3.13, tidak terlihat secara jelas lapisan *presentation*, *application*, *infrastructure*, dan *domain* yang terdapat pada Gambar 2.2. Namun, masing-masing lapisan tersebut telah terdefinisi secara implisit pada *library Bshaffer OAuth2*. Sebagai contoh lapisan *presentation* terdapat pada *folder Controller*, *Token Type*, dan *Client Assertion Type*, lapisan *application* terdapat pada *folder Grant Type*, dan *Response Type*, lapisan *infrastructure* terdapat pada folder *Repositories* yang terdefinisi diluar *library*, dan lapisan *domain* terdapat pada *folder Storage* yang berupa abstraksi, dan tipe data *array* yang bersifat primitif untuk melakukan *mapping* obyek dari basis data. Pemetaan *namespace* kelas sesuai lapisan yang digunakan sebagai contoh dapat dilihat pada Tabel 3.14.

Tabel 3.14 Pemetaan *Namespace Library* Sesuai Lapisan Arsitektur (Hanya Diambil Sebagian)

Lapisan	Namespace
Presentation	• *OAuth2\Controller* • *OAuth2\OpenID\Controller*

	• *OAuth2\TokenType* • *Kelas OAuth2\Request.php* • *Kelas OAuth2\Response.php* • *OAuth2\ClientAssertionType*
Application	• *OAuth2\GrantType* • *OAuth2\OpenID\GrantType* • *OAuth2\ResponseType* • *OAuth2\OpenID\ResponseType*
Infrastructure	• *MyIts\Oidc\Repositories* (implementasi abstraksi *storage*)
Domain	• *OAuth2\Storage* (abstraksi *storage*) • *OAuth2\OpenID\Storage* (abstraksi *storage*)

3.2.3.1.1.1 Kelas Dan Abstraksi Pada *Folder OAuth2*

Penjelasan kelas dan abstraksi yang terdapat pada *folder OAuth2* dapat dilihat pada Tabel 3.15.

Tabel 3.15 Penjelasan Kelas Dan Abstraksi Pada *Folder OAuth2*

Nama *File*	Deskripsi
Request.php	Kelas yang merepresentasikan obyek *request* HTTP.
RequestInterface.php	Abstraksi yang menyediakan fungsi umum untuk merepresentasikan obyek *request* HTTP.
Response.php	Kelas yang merepresentasikan obyek *response* HTTP.

ResponseInterface.php	Abstraksi yang menyediakan fungsi umum untuk merepresentasikan obyek *response* HTTP.
Scope.php	Kelas yang mengakomodir manajemen *scope*.
Server.php	Kelas yang merepresentasikan server autorisasi sebagai obyek utama. Kelas ini merupakan *entry point* layanan server autorisasi.

3.2.3.1.1.2 Kelas Dan Abstraksi Pada *Folder Client Assertion Type*

Penjelasan kelas dan abstraksi yang terdapat *folder Client Assertion Type* dapat dilihat pada Tabel 3.16.

Tabel 3.16 Penjelasan Kelas Dan Abstraksi Pada *Folder Client Assertion Type*

Nama *File*	Deskripsi
HttpBasic.php	Kelas yang menangani autentikasi klien menggunakan skema *Basic Authentication.*
ClientAssertionTypeInterface.php	Abstraksi yang menyediakan fungsi umum untuk melakukan autentikasi klien.

3.2.3.1.1.3 Kelas Dan Abstraksi Pada *Folder Controller*

Penjelasan kelas dan abstraksi yang terdapat pada *folder Controller* dapat dilihat pada Tabel 3.17. Telah diambil dua kelas dan dua abstraksi sebagai contoh.

Tabel 3.17 Penjelasan Kelas Dan Abstraksi Pada *Folder Controller*

Nama *File*	Deskripsi
AuthorizeController.php	Kelas *controller* untuk menangani autorisasi klien.
AuthorizeControllerInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani autorisasi klien.
TokenController.php	Kelas *controller* untuk menangani pembuatan token.
TokenControllerInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani pembuatan token.

3.2.3.1.1.4 Kelas Dan Abstraksi Pada *Folder Encryption*

Penjelasan kelas dan abstraksi yang terdapat pada *folder Encryption* dapat dilihat pada Tabel 3.18.

Tabel 3.18 Penjelasan Kelas Dan Abstraksi Pada *Folder Encryption*

Nama *File*	Deskripsi
Jwt.php	Kelas yang menangani penyusunan JWT.
FirebaseJwt.php	Kelas yang menangani penyusunan JWT.
EncryptionInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani penyusunan data dengan algoritma tertentu.

3.2.3.1.1.5 Kelas Dan Abstraksi Pada *Folder Grant Type*

Penjelasan kelas dan abstraksi yang terdapat pada *folder Grant Type* dapat dilihat pada Tabel 3.19. Folder *Grant Type* hanya menyediakan satu jenis abstraksi yaitu *GrantTypeInterface.php*, sehingga pembuatan alur baru dapat menggunakan abstraksi tersebut. Telah diambil dua kelas dan satu abstraksi sebagai contoh.

Tabel 3.19 Penjelasan Kelas Dan Abstraksi Pada *Folder Grant Type*

Nama *File*	Deskripsi
AuthorizationCode.php	Kelas yang memiliki proses bisnis alur *Authorization Code.*
ClientCredentials.php	Kelas yang memiliki proses bisnis alur *client credentials.*
GrantTypeInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani proses bisnis alur OAuth 2.0 dan OIDC.

3.2.3.1.1.6 Kelas Dan Abstraksi Pada *Folder OpenID Controller*

Penjelasan kelas dan abstraksi yang terdapat pada *folder OpenID Controller* dapat dilihat pada Tabel 3.20. Kelas *controller* masing-masing memiliki abstraksi sendiri karena memiliki tanggung jawab yang spesifik sehingga tidak bisa dijadikan sebagai satu abstraksi dengan fungsi-fungsi umum.

Tabel 3.20 Penjelasan Kelas Dan Abstraksi Pada *Folder OpenID Controller*

Nama *File*	Deskripsi
AuthorizeController.php	Kelas *controller* untuk menangani autorisasi klien.
AuthorizeControllerInterface.php	Abstraksi yang menyediakan fungsi umum

	untuk menangani autorisasi klien.
UserInfoController.php	Kelas *controller* untuk menangani informasi dasar pengguna.
UserInfoControllerInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani informasi dasar pengguna.

3.2.3.1.1.7 Kelas Pada *Folder OpenID Grant Type*

Penjelasan kelas yang terdapat pada *folder OpenID Grant Type* dapat dilihat pada Tabel 3.21. Dapat dilihat bahwa ada satu kelas yaitu *AuthorizationCode.php* karena *library Bshaffer OAuth2* hanya dapat menangani alur *Authorization Code* jika menggunakan OIDC. Kelas itu pun hasil ekstensi dari kelas *AuthorizationCode.php* yang telah dijelaskan pada subbab 3.2.3.1.1.5 karena memiliki fungsi umum yang sama.

Tabel 3.21 Penjelasan Kelas Pada *Folder OpenID Grant Type*

Nama *File*	Deskripsi
AuthorizationCode.php	Kelas yang memiliki proses bisnis alur OIDC *authorization code*.

3.2.3.1.1.8 Kelas Dan Abstraksi Pada *Folder OpenID Response Type*

Penjelasan kelas yang terdapat pada *folder OpenID Response Type* dapat dilihat pada Tabel 3.22.

Tabel 3.22 Penjelasan Kelas Dan Abstraksi pada *folder OpenID Response Type*

Nama *File*	Deskripsi
AuthorizationCode.php	Kelas yang menangani *response* dalam bentuk *authorization code*.
AuthorizationCodeInterface.php	Abstraksi yang menyediakan fungsi umum

	untuk menangani *response* dalam bentuk *authorization code*.
IdToken.php	Kelas yang menangani *response* dalam bentuk *ID Token*.
IdTokenInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani *response* dalam bentuk *ID Token*.

3.2.3.1.1.9 Abstraksi Pada *Folder OpenID Storage*

Penjelasan abstraksi yang terdapat pada *folder OpenID Storage* dapat dilihat pada Tabel 3.23. Telah diambil dua abstraksi sebagai contoh.

Tabel 3.23 Penjelasan Abstraksi pada *folder OpenID Storage*

Nama *File*	Deskripsi
AuthorizationCodeInterface.php	Abstraksi yang menyediakan fungsi umum untuk mengambil dan menyimpan *authorization code*.
UserClaimsInterface.php	Abstraksi yang menyediakan fungsi umum untuk mengambil dan menyimpan fakta pengguna.

3.2.3.1.1.10 Kelas Dan Abstraksi Pada *Folder Response Type*

Penjelasan kelas dan abstraksi yang terdapat pada *folder Response Type* dapat dilihat pada Tabel 3.24. Telah diambil dua kelas dan dua abstraksi sebagai contoh. Masing-masing implementasi kelas memiliki abstraksi sendiri karena memiliki tugas yang spesifik.

Tabel 3.24 Penjelasan Kelas Dan Abstraksi Pada *Folder Response Type*

Nama File	Deskripsi
AccessToken.php	Kelas yang menangani *response* dalam bentuk *access token*.
AccessTokenInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani *response* dalam bentuk *access token*.
AuthorizationCode.php	Kelas yang menangani *response* dalam bentuk *authorization code*.
AuthorizationCodeInterface.php	Abstraksi yang menyediakan fungsi umum untuk menangani *response* dalam bentuk *authorization code*.

3.2.3.1.1.11 Abstraksi Pada *Folder Storage*

Penjelasan abstraksi yang terdapat pada *folder Storage* dapat dilihat pada Tabel 3.25. Telah diambil dua abstraksi sebagai contoh.

Tabel 3.25 Penjelasan Abstraksi Pada *Folder Storage*

Nama File	Deskripsi
ScopeInterface.php	Abstraksi yang menyediakan fungsi umum untuk mengambil dan menyimpan *scope*.
UserCredentialsInterface.php	Abstraksi yang menyediakan fungsi umum untuk mengambil dan menyimpan kredensial pengguna.

3.2.3.1.1.12 Kelas Dan Abstraksi Pada *Folder Token Type*

Penjelasan kelas dan abstraksi yang terdapat pada *folder Token Type* dapat dilihat pada Tabel 3.26. *Folder Token Type* hanya menyediakan satu abstraksi karena tanggung jawab *Token Type* adalah untuk menangani pengambilan token dari *request* HTTP sehingga dapat disatukan sebagai fungsi umum.

Tabel 3.26 Penjelasan Kelas Dan Abstraksi Pada *Folder Token Type*

Nama File	Deskripsi
Bearer.php	Kelas yang menangani pengambilan tipe token *bearer* dari *header* HTTP *request*.
TokenTypeInterface	Abstraksi yang menyediakan fungsi umum untuk mengekstrak token dari HTTP *request*.

3.2.3.1.2 Perancangan Kelas Untuk Akomodir CIBA Setelah Melakukan Observasi

Berdasarkan hasil observasi terhadap struktur *library Bshaffer OAuth2* didapatkan pengetahuan bahwa:

a. *Client Assertion Type* merupakan kelas yang menangani autentikasi aplikasi klien.
b. *Controller* merupakan kelas yang menerima permintaan klien untuk meneruskannya ke proses alur CIBA yang lebih spesifik.
c. *Encryption* merupakan kelas yang menangani enkripsi data.
d. *Grant Type* merupakan kelas yang membungkus proses bisnis alur protokol.
e. *Response Type* merupakan kelas yang menangani pembuatan obyek entitas (*access token*, *JWT access token*, *ID Token* dan lain-lain) OAuth 2.0 dan OIDC yang digunakan sebagai *response*.
f. *Storage* merupakan kelas yang menangani pencarian dan penyimpanan data terhadap suatu basis data.

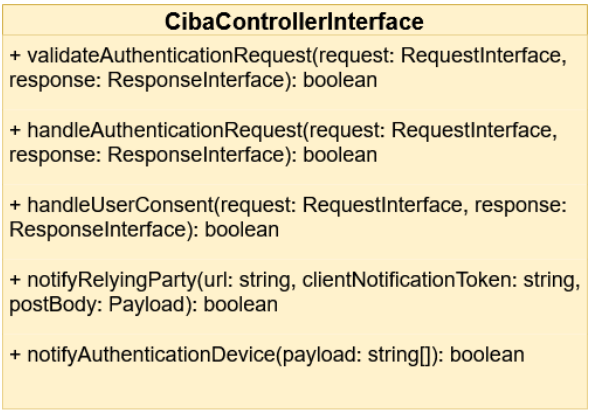
g. *Token Type* merupakan kelas yang menangani pengambilan token dari *request* HTTP.

Sehingga didapatkan rancangan kelas yang akan digunakan untuk mengakomodir protokol CIBA sebagai berikut. Kelas-kelas berikut berawal dari suatu abstraksi sehingga tidak menimbulkan ketergantungan secara langsung terhadap implementasi. Sebagai contoh jika *developer* ingin mengganti sistem basis data, yang awal mulanya berupa relasional menjadi non relasional, maka yang perlu dilakukan adalah membuat kelas implementasi baru dari abstraksi *storage* yang sudah disediakan. Dengan melalui cara tersebut, tidak akan menimbulkan kerusakan pada perlikau *library* dan sistem.

Berikut adalah perancangan abstraksi yang didapat setelah melakukan observasi.

3.2.3.1.2.1 Abstraksi *Controller Ciba Controller Interface*

Abstraksi ini digunakan untuk memfasilitasi proses bisnis autentikasi menggunakan CIBA. Untuk mempermudah *developer* menggunakan *library* yang dibuat, maka *Ciba Controller Interface* memiliki fungsi umum yang akan meneruskan proses bisnis ke lapisan yang lebih spesifik. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.20.



Gambar 3.20 Fungsi Abstraksi *Ciba Controller Interface*

Penjelasan fungsi pada *Ciba Controller Interface* dapat dilihat pada Tabel 3.27.

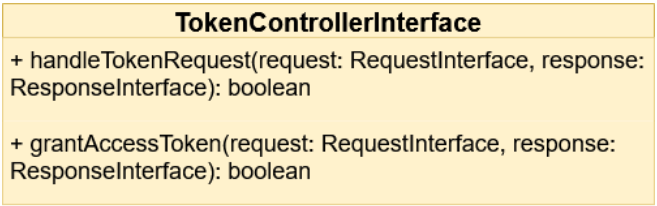
Tabel 3.27 Penjelasan Fungsi Abstraksi *Ciba Controller Interface*

Nama Fungsi	Deskripsi
validateAuthenticationRequest	Fungsi ini digunakan untuk melakukan validasi *request* autentikasi yang masuk. Ini bertujuan untuk mengetahui parameter yang tidak lengkap, atau yang tidak sesuai aturan. Peraturan validasi dapat dilihat pada subbab 2.4.2.3.
handleAuthenticationRequest	Fungsi ini digunakan untuk menangani *request* autentikasi yang masuk. Ini merupakan fungsi *entry point* untuk melayani protokol CIBA.

handleUserConsent	Fungsi ini digunakan untuk menangani pemberian izin hak akses oleh pengguna yang berasal dari AD.
notifyRelyingParty	Fungsi ini digunakan untuk mengirim pesan ke RP.
notifyAuthenticationDevice	Fungsi ini digunakan untuk mengirim pesan ke AD.

3.2.3.1.2.2 Abstraksi *Controller Token Controller Interface*

Abstraksi ini digunakan untuk memfasilitasi proses permintaan dan pembuatan token pada protokol CIBA. Jika mengacu pada subbab 3.2.3.1.1.3 telah tersedia *Token Controller Interface* dengan fungsi-fungsi umum yang memadai. Oleh karena itu, proses permintaan dan pembuatan token pada protokol CIBA dapat menggunakan *Token Controller Interface* yang sudah ada. Hanya saat fase implementasi memiliki perilaku yang berbeda. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.21.



Gambar 3.21 Fungsi Abstraksi *Token Controller Interface*

Penjelasan fungsi abstraksi *Token Controller Interface* dapat dilihat pada Tabel 3.28.

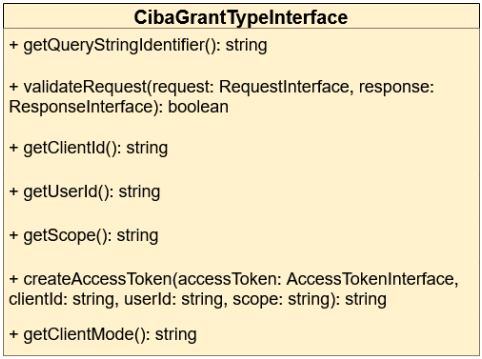
Tabel 3.28 Penjelasan Fungsi Abstraksi *Token Controller Interface*

Nama Fungsi	Deskripsi

handleTokenRequest	Fungsi ini digunakan sebagai *entry point* untuk menangani permintaan token.
grantAccessToken	Fungsi ini digunakan untuk memutuskan pemberian token.

3.2.3.1.2.3 Abstraksi *Grant Type Ciba Grant Type Interface*

Abstraksi ini digunakan untuk melapisi proses bisnis alur CIBA. Jika mengacu pada subbab 3.2.3.1.1.5 telah tersedia abstraksi *Grant Type Interface* sehingga dapat dimanfaatkan. Namun abstraksi *Grant Type Interface* tidak menyediakan fungsi untuk mengetahui mode pengambilan token yang digunakan oleh aplikasi klien. Oleh karena itu perancangan abstraksi *Ciba Grant Type Interface* berupa *extension* dari abstraksi *Grant Type Interface*. *Extension* tersebut berupa penambahan satu fungsi, yaitu *getClientMode* yang digunakan untuk mengetahui mode pengambilan token aplikasi klien. Fungsi yang terdapat pada abstraksi ini selain *getClientMode* merupakan warisan dari abstraksi *Grant Type Interface*. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.22.



Gambar 3.22 Fungsi Abstraksi *Ciba Grant Type Interface*

Penjelasan fungsi abstraksi *Ciba Grant Type Interface* dapat dilihat pada Tabel 3.29.

Tabel 3.29 Penjelasan Fungsi Abstraksi *Ciba Grant Type Interface*

Nama Fungsi	Deskripsi
getQueryStringIdentifier	Fungsi ini digunakan untuk mendapatkan identifikasi *grant type* atau alur yang digunakan.
validateRequest	Fungsi ini untuk melakukan validasi *request* yang masuk.
getClientId	Fungsi ini digunakan untuk mendapatkan identifikasi aplikasi klien.
getUserId	Fungsi ini digunakan untuk mendapatkan identifikasi pengguna.
getScope	Fungsi ini digunakan untuk mendapatkan *scope* yang diminta.
createAccessToken	Fungsi ini digunakan untuk menyusun token.
getClientMode	Fungsi ini digunakan untuk mendapatkan mode token aplikasi klien.

3.2.3.1.2.4 Abstraksi *Response Type Ciba Interface*

Abstraksi ini digunakan untuk menyusun *response* dan pembuatan obyek *authentication request* CIBA. Obyek tersebut digunakan untuk mencatat sesi yang terikat pada protokol CIBA. Jika mengacu pada subbab 3.2.3.1.1.10 telah tersedia abstraksi *Response Type Interface*, namun fungsi-fungsi pada abstraksi tersebut tidak memadahi untuk protokol CIBA sehingga dibuat abstraksi *Ciba Interface*. Fungsi yang tersedia pada *Ciba Interface* dapat dilihat pada Gambar 3.23.

```
CibaInterface
+ createAuthenticationRequest(authReqId: string, clientId: string, providerId:
string, userId: string, addedSecondsToExpire: integer, mode: string,
clientNotificationToken: string, hint: string, bindingMessage: string, scope:
string, idToken: string): string [ ]

+ generateAuthReqId(): string
```

Gambar 3.23 Fungsi Abstraksi *Ciba Interface*

Penjelasan fungsi abstraksi *Ciba Interface* dapat dilihat pada Tabel 3.30.

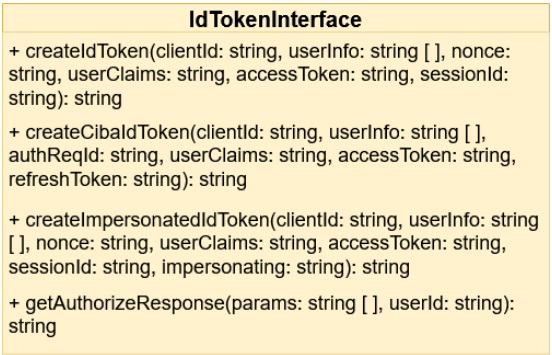
Tabel 3.30 Penjelasan Fungsi Abstraksi *Ciba Interface*

Nama Fungsi	Deskripsi
createAuthenticationRequest	Fungsi ini digunakan untuk menyusun sesi autentikasi CIBA.
generateAuthReqId	Fungsi ini digunakan untuk menyusun identitas *request* autentikasi.

3.2.3.1.2.5 Abstraksi *Response Type Id Token Interface*

Abstraksi ini digunakan untuk menyusun *ID Token* CIBA. *ID Token* dapat dimanfaatkan oleh aplikasi klien sebagai sesi autentikasi pengguna. *ID Token* disusun menjadi format JWT. Seperti yang telah dijelaskan pada subbab 3.2.3.1.1.8 telah tersedia abstraksi *ID Token Interface*. Namun, fungsi pada abstraksi *ID Token Interface* tidak memadahi sehingga dibuat satu fungsi khusus untuk menyusun *ID Token* CIBA dengan nama *createCibaIdToken*. Selanjutnya ditambahkan fungsi *createImpersonatedIdToken* untuk menyusun *ID Token* kasus *impersonation*. Abstraksi *ID Token Interface* merupakan *extension* dari abstraksi *Response Type Interface* sehingga mewarisi fungsi dari abstraksi *Response Type Interface*, yaitu fungsi

getAuthorizeResponse. Fungsi yang tersedia pada abstraksi ini dapat dilihat pada Gambar 3.24.



Gambar 3.24 Fungsi Abstraksi *Id Token Interface*

Penjelasan fungsi abstraksi *Id Token Interface* dapat dilihat pada Tabel 3.31.

Tabel 3.31 Penjelasan Fungsi Abstraksi *ID Token Interface*

Nama Fungsi	Deskripsi
createIdToken	Fungsi ini digunakan untuk menyusun *ID Token* pada alur *Authorization Code*.
createCibaIdToken	Fungsi ini digunakan untuk menyusun *ID Token* pada alur CIBA.
createImpersonatedIdToken	Fungsi ini digunakan untuk menyusun *ID Token* pada alur *Authorization Code* kasus *impersonation.*
getAuthorizeResponse	Fungsi ini digunakan untuk menyusun *response redirect URI* pada alur *Authorization Code*.

3.2.3.1.2.6 Abstraksi *Storage Ciba Interface*

Abstraksi ini digunakan untuk mencari dan mengambil sesi *request autentikasi* CIBA pada suatu basis data. Fungsi pada abstraksi ini telah dibuat menjadi fungsi yang umum, sehingga jika terjadi perubahan basis pada sistem, maka perilaku perangkat lunak tetap berjalan dengan normal. Sebagai contoh jika data CIBA tersimpan pada suatu basis data yang bekerja menggunakan *file system*, dan *developer* ingin meningkatkan performa menggunakan basis data yang bersifat *in-memory*, maka hanya perlu membuat implementasi yang bergantung pada abstraksi ini. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.25.

CibaInterface

+ getAuthReqId(authReqId: string): string [ ]

+ setAuthReqId(authReqId: string, clientId: string, providerId: string, userId: string, expires: datetime, clientNotificationToken: string, hint: string, bindingMessage: string, scope: string, idToken: string): boolean

+ expireAuthReqId(authReqId: string): boolean

+ setConsent(authReqId: string, consent: boolean): boolean

+ setLastRequestedTokenAt(authReqId: string, time: datetime): boolean

Gambar 3.25 Fungsi Abstraksi *Ciba Interface*

Penjelasan fungsi abstraksi *Id Token Interface* dapat dilihat pada Tabel 3.31.

Tabel 3.32 Penjelasan Fungsi Abstraksi *Ciba Interface*

Nama Fungsi	Deskripsi
getAuthReqId	Fungsi ini digunakan untuk mencari data *request* autentikasi.
setAuthReqId	Fungsi ini digunakan untuk menyimpan data *request* autentikasi.

expireAuthReqId	Fungsi ini digunakan untuk mengatur data *request* autentikasi menjadi kedaluwarsa.
setConsent	Fungsi ini digunakan untuk menyimpan izin hak akses *request* autentikasi.
setLastRequestedTokenAt	Fungsi ini digunakan untuk menyimpan waktu terakhir sesi *request* autentikasi memanggil token *endpoint*. Nilai waktu terakhir digunakan oleh server autorisasi pada mode token *poll* untuk mengetahui kecepatan *polling*, agar server autorisasi dapat memberi tahu RP untuk *polling* lebih pelan jika terlalu cepat.

3.2.3.1.2.7 Abstraksi *Storage Access Token Interface*

Library *Bshaffer OAuth2* telah menyediakan abstraksi untuk menyimpan dan mengambil *access token* pada suatu basis data. Fungsi yang tedapat pada abstraksi ini telah memadahi untuk digunakan pada protokol CIBA. Oleh karena itu abstraksi ini dapat digunakan tanpa melakukan perubahan apapun. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.26. Abstraksi ini memiliki tiga fungsi, namun yang digunakan pada protokol CIBA hanya satu, yaitu fungsi *setAccessToken*.

```
AccessTokenInterface
+ getAccessToken(oauthToken: string): string[ ]

+ setAccessToken(oauthToken: string, clientId: string, userId:
string, expires: int, scope: string): boolean

+ unsetAccessToken(accessToken: string): boolean
```

Gambar 3.26 Fungsi Abstraksi *Access Token Interface*

Penjelasan fungsi abstraksi *Access Token Interface* dapat dilihat pada Tabel 3.33.

Tabel 3.33 Penjelasan Fungsi Abstraksi *Access Token Interface*

Nama Fungsi	Deskripsi
getAccessToken	Fungsi ini digunakan untuk mengambil *access token* dari basis data.
setAccessToken	Fungsi ini digunakan untuk menyimpan *access token* ke basis data.
unsetAccessToken	Fungsi ini digunakan untuk menghapus *access token* dari basis data.

3.2.3.1.2.8 Abstraksi *Infrastructure Notification Transport Interface*

Library Bshaffer OAuth2 tidak menyediakan abstraksi untuk berkomunikasi dengan sistem eksternal. Oleh karena itu dibuat pengelompokkan berupa *folder* yang dinamakan *Infrastructure*.

Abstraksi ini digunakan untuk mengirim data ke sistem eksternal melalui protokol pilihan. Sebagai contoh pada tugas akhir ini, abstraksi *Notification Transport Interface* digunakan untuk mengirim pesan ke AD dan RP menggunakan protokol *Firebase* dan HTTP. Fungsi pada abstraksi ini dapat dilihat pada Gambar 3.27.

NotificationTransportInterface
+ send(payload: string[], to: string): boolean

Gambar 3.27 Fungsi Abstraksi *Notification Transport Interface*

Penjelasan fungsi abstraksi *Notification Transport Interface* dapat dilihat pada Tabel 3.34.

Tabel 3.34 Penjelasan Fungsi Abstraksi *Notification Transport Interface*

Nama Fungsi	Deskripsi
send	Fungsi ini digunakan untuk mengirim data ke sistem eksternal.

3.2.3.2 Perancangan Aplikasi Server Autorisasi CIBA

Perancangan aplikasi server autorisasi dibuat sedemikian rupa sehingga *kelas* yang dihasilkan pada *library* server autorisasi dapat digunakan dengan mudah pada aplikasi ini. Oleh sebab itu, kelas yang dihasilkan pada aplikasi server autorisasi adalah dalam bentuk kelas *controller* yang terletak pada lapisan *presentation*. Kelas *controller* ini merupakan *entry point* server autorisasi CIBA dan memiliki URI yang dapat dieksekusi oleh aplikasi klien.

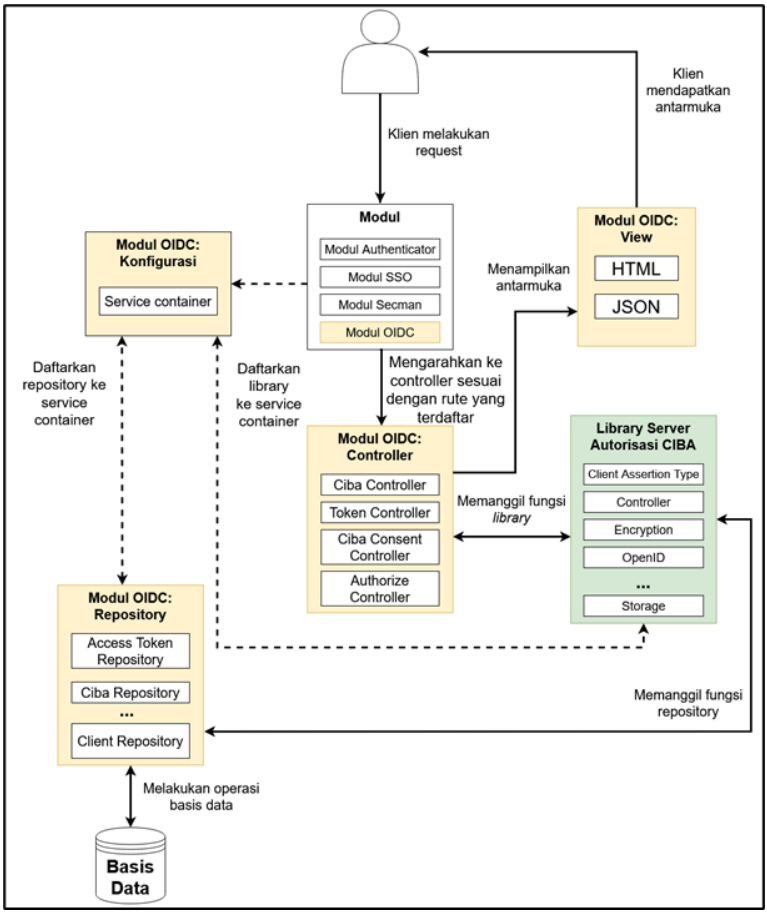
3.2.3.2.1 Observasi Awal Struktur Aplikasi Server Autorisasi

Aplikasi server autorisasi berada dalam proyek yang dinamakan dengan *php-oidc*. *Php-oidc* adalah proyek yang dibangun menggunakan kerangka kerja Phalcon. Proyek ini terdiri dari beberapa modul, yaitu modul *authenticator*, *oidc*, *secman*, dan *sso* yang bisa dilihat pada Tabel 3.35. *Php-oidc* merupakan proyek yang memiliki struktur *multi module*, yang bisa diartikan sebagai proyek besar dengan proyek-proyek spesifik didalamnya. Aplikasi server autorisasi berada pada modul *oidc*.

Tabel 3.35 Modul Pada Proyek *Php-Oidc*

Nama Modul	Deskripsi
Authenticator	Aplikasi untuk mendukung *Authentication Device.*
Oidc	Aplikasi yang berperan sebagai server autorisasi. Aplikasi ini memberi layanan autentikasi dan autorisasi dengan alur *Authorization Code* dan CIBA.
Secman	Aplikasi klien *Security Manager* yang mengatur keamanan server autorisasi..
Sso	Aplikasi klien yang memberi antarmuka server autorisasi.

Cara kerja modul *oidc* dapat dilihat pada Gambar 3.28. Klien akan melakukan *request* yang ditujukan ke modul *oidc*. Modul *oidc* lalu akan meneruskan *request* yang masuk ke *controller* yang bersangkutan. Pada *controller* terdapat fungsi-fungsi yang akan menggunakan *library* server autorisasi CIBA. Untuk memudahkan pembangunan obyek *library*, modul *oidc* dapat memanfaatkan fitur *service container* dan *dependency injection*. Fungsi *controller* akan memanggil fungsi pada *library* server autorisasi CIBA. Pada *library* tersebut, sudah dilakukan *dependency injection* sehingga kelas *repository* yang terdapat pada modul *oidc* dapat digunakan oleh *library*. *Library* dapat melakukan proses bisnis CIBA lebih lanjut seperti validasi autentikasi, penyusunan *authentication request id* dan penyusunan *access token*. Lalu jika tugas pada *library* sudah selesai, *controller oidc* dapat mengembalikan pesan ke klien dalam bentuk HTML atau JSON.



Gambar 3.28 Hubungan *Library* Server Autorisasi CIBA Dengan Aplikasi Server Autorisasi CIBA

Modul *oidc* memiliki struktur *folder* yang dapat dilihat pada Tabel 3.36.

Tabel 3.36 Struktur *Folder* Modul Oidc

Nama Folder	Deskripsi
Config	Folder yang mengelompokkan konfigurasi seperti *service container*, *rute URI*, *private key*, dan *localization*.
Controllers	Folder yang mengelompokkan tipe kelas *controller*.
Domain	Folder yang mengelompokkan tipe kelas *domain event*.
Exceptions	*Folder* yang mengelompokkan tipe kelas *exception* yang menjelaskan *error* spesifik.
Models	*Folder* yang mengelompokkan tipe kelas *model* untuk melakukan akses ke basis data. Namun *model* tidak digunakan karena modul *oidc* telah melakukan migrasi dari *model* ke *raw SQL query* menggunakan *library PHP Document Object* (PDO).
Repositories	*Folder* yang mengelompokkan tipe kelas *repository* untuk melakukan akses ke basis data.
Usecases	*Folder* yang mengelompokkan tipe kelas *use case* atau *application service*.
Views	*Folder* yang mengelompokkan antarmuka dalam bentuk HTML.

3.2.3.2.1.1 *File* Pada *Folder Config*

Penjelasan *file* yang terdapat pada *folder Config* dapat dilihat pada Tabel 3.37.

Tabel 3.37 Penjelasan *File* Pada *Folder Config*

Nama File	Deskripsi
Keys/login_private_key.pub	*File* yang menyimpan *private key*.

Lang/en.php	*File* yang menyimpan terjemahan kata dalam bahasa Inggris.
Lang/id.php	*File* yang menyimpan terjemahan kata dalam bahasa Indonesia.
Config.php	*File* yang menyimpan konfigurasi sistem seperti alamat dan *port* basis data.
Routing.php	*File* yang menyimpan rute URI.
Services.events.php	*File* yang menyimpan *obyek domain event* pada *service container*.
Services.php	*File* yang menyimpan obyek umum pada *service container*. *Library* server autorisasi CIBA akan dikonfigurasi pada *file* ini agar dapat tersimpan pada *service container*.

3.2.3.2.1.2 Kelas Pada *Folder Controllers*

Penjelasan kelas yang terdapat pada *folder Controllers* dapat dilihat pada Tabel 3.38.

Tabel 3.38 Penjelasan Kelas Pada *Folder Controllers*

Nama Kelas	Deskripsi
Authorize Controller	Kelas *controller* untuk menangani autorisasi pengguna.
Configuration Controller	Kelas *controller* untuk menangani *OpenID Connect Discovery*.
Consent Controller	Kelas *controller* untuk menangani hak akses aplikasi klien.
Login Controller	Kelas *controller* untuk menangani autentikasi pengguna.

Token Controller	Kelas *controller* untuk menangani permintaan *access token*.

3.2.3.2.1.3 Kelas Pada *Folder Domain*

Penjelasan kelas yang terdapat pada *folder Domain* dapat dilihat pada Tabel 3.39.

Tabel 3.39 Penjelasan Kelas Pada *Folder Domain*

Nama Kelas	Deskripsi
Events/Login Successful Event	*Event* yang dilakukan *dispatch* ketika pengguna berhasil autentikasi.

3.2.3.2.1.4 Kelas Pada *Folder Exceptions*

Penjelasan kelas yang terdapat pada *folder Exceptions* dapat dilihat pada Tabel 3.40.

Tabel 3.40 Penjelasan Kelas Pada *Folder Exceptions*

Nama Kelas	Deskripsi
Update Primary Email And Activate Account Failed	*Error* yang muncul ketika gagal memperbarui akun.

3.2.3.2.1.5 Kelas Pada Folder Repositories

Penjelasan kelas yang terdapat pada *folder Repositories* dapat dilihat pada Tabel 3.41. Diambil lima kelas *repository* sebagai contoh.

Tabel 3.41 Penjelasan Kelas Pada *Folder Repositories*

Nama Kelas	Deskripsi

Access Token Repository	Kelas *repository* yang menangani penyimpanan dan pencarian *access token*.
Authorization Code Repository	Kelas *repository* yang menangani penyimpanan dan pencarian *authorization code*.
Client Repository	Kelas *repository* yang menangani penyimpanan dan pencarian aplikasi klien.
User Claims Repository	Kelas *repository* yang menangani penyimpanan dan pencarian fakta akun pengguna.
Users Repository	Kelas *repository* yang menangani penyimpanan dan pencarian akun pengguna.

3.2.3.2.1.6 Kelas Pada *Folder Usecases*

Penjelasan kelas yang terdapat pada *folder Usecases* dapat dilihat pada Tabel 3.42.

Tabel 3.42 Penjelasan Kelas Pada *Folder Usecases*

Nama Kelas	Deskripsi
Login Successful	Kelas *usecase* yang menangani logika aplikasi untuk mencatat waktu autentikasi akun pengguna. Kelas *usecase* ini berperan sebagai *event handler* atas kejadian *event*

	"Events/Login Successful Event".
Update Primary Email And Activate Account	Kelas *usecase* yang menangani logika aplikasi untuk memperbarui data pengguna.

3.2.3.2.1.7 Antarmuka Pada Folder Views

Penjelasan antarmuka yang terdapat pada *folder Views* dapat dilihat pada Tabel 3.43.

Tabel 3.43 Penjelasan Antarmuka Pada *Folder Views*

Nama Antarmuka	Deskripsi
Base.volt	Antarmuka yang menampilkan struktur *layout* halaman.
Login.volt	Antarmuka yang menampilkan formulir *login*.
Logout.volt	Antarmuka yang menampilkan formulir *logout*.
User_cosent.volt	Antarmuka yang menampilkan formulir hak akses.

3.2.3.2.2 Perancangan Kelas Untuk Akomodir CIBA Setelah Melakukan Observasi

Setelah melakukan observasi terhadap aplikasi server autorisasi yang terletak pada modul *oidc*, didapatkan rancangan kelas yang akan digunakan untuk mengakomodir protokol CIBA sebagai berikut.

3.2.3.2.2.1 Kelas *Controller Authorize Controller*

Pada modul *oidc* sudah terdapat kelas ini sebelum pengerjaan tugas akhir dimulai. Kelas ini memiliki tugas untuk

menangani hak akses aplikasi klien dan pengguna pada alur *Authorization Code*. Aplikasi klien harus terdaftar menggunakan alur ini, dan jika aplikasi klien membutuhkan izin hak akses dari pengguna, maka pengguna harus menyetujui untuk melanjutkan proses autorisasi. Namun agar dapat memenuhi Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004), akan dilakukan modifikasi terhadap fungsi *handleAuthorizeAction* pada kelas ini yang dapat dilihat pada Tabel 3.44.

Tabel 3.44 Penjelasan Fungsi Kelas *Authorize Controller*

Nama Fungsi	Deskripsi
handleAuthorizeAction	Fungsi ini digunakan untuk menangani autorisasi aplikasi klien dan pengguna.

Modifikasi yang akan dilakukan adalah penambahan *claim "act"* pada *ID Token* yang disusun ketika melakukan autentikasi. ID Token yang mulanya memiliki *payload* yang dapat dilihat pada Gambar 3.29 akan memiliki *"act"* yang dapat dilihat pada Gambar 3.30. Claim *"act"* memiliki arti sebagai aktor. Sistem akan membaca suatu *ID Token* dengan *claim "act"* sebagai identitas pengguna yang melakukan *impersonation*, sedangkan *claim "sub"* memiliki arti sebagai identitas pengguna yang dilakukan *impersonation*. Pendekatan ini dipilih karena sistem tetap dapat menggunakan alur *Authorization Code* sehingga proses autentikasi tetap aman, dan tidak perlu membuat protokol baru.

```
{
  "iss": "https://dev-my.its.ac.id",
  "sub": "20DA261C-CA2E-432E-8253-E37BDB3F6CA7",
  "aud": "B30E829E-3E1E-4A95-984E-0A79D566A221",
  "iat": 1588433583,
  "exp": 1588437183,
  "auth_time": 1588433583,
  "nonce": "47958e55c0a49cff483234c5cd22066f"
}
```


Gambar 3.29 *Payload ID Token* Sebelum Modifikasi


```
{
  "iss": "https://dev-my.its.ac.id",
  "act": "30FA464D-037C-4511-B3DD-6A7B730DAF7C",
  "sub": "20DA261C-CA2E-432E-8253-E37BDB3F6CA7",
  "aud": "B30E829E-3E1E-4A95-984E-0A79D566A221",
  "iat": 1588433583,
  "exp": 1588437183,
  "auth_time": 1588433583,
  "nonce": "47958e55c0a49cff483234c5cd22066f"
}
```


Gambar 3.30 *Payload ID Token* Setelah Modifikasi

3.2.3.2.2.2 Kelas *Controller Login Controller*

Kelas *Login Controller* digunakan untuk melayani autentikasi beserta menampilkan formulir *login*. Kelas ini memiliki fungsi yang dinamakan dengan *handleLoginAction*. Fungsi ini digunakan untuk masuk ke sistem myITS SSO dan tidak menggunakan alur CIBA tetapi alur *Authorization Code*. Fungsi ini dapat dilihat pada Tabel 3.45.

Tabel 3.45 Penjelasan Fungsi Kelas *Login Controller*

Nama Fungsi	Deskripsi
handleLoginAction	Fungsi ini digunakan untuk melayani autentikasi.

3.2.3.2.2.3 Kelas *Controller Ciba Controller*

Kelas *Ciba Controller* digunakan untuk melayani aplikasi klien yang ingin melakukan autentikasi *backchannel*. Kelas ini memiliki fungsi yang dinamakan dengan *handleAuthenticate Action*. Fungsi ini dapat dilihat pada Tabel 3.46.

Tabel 3.46 Penjelasan Fungsi Kelas *Ciba Controller*

Nama Fungsi	Deskripsi	URI
handleAuthenticateAction	Fungsi ini digunakan untuk melayani autentikasi *backchannel*.	https://dev-my.its.ac.id/bc-authorize

3.2.3.2.2.4 Kelas *Controller Ciba Consent Controller*

Kelas *Ciba Consent Controller* digunakan untuk melayani pemberian hak akses oleh pengguna pada *Authentication Device*. Kelas ini memiliki fungsi yang dinamakan dengan *handleConsentAction*. Fungsi ini dapat dilihat pada Tabel 3.47.

Tabel 3.47 Penjelasan Fungsi Kelas *Ciba Consent Controller*

Nama Fungsi	Deskripsi	URI
handleConsentAction	Fungsi ini digunakan untuk melayani pemberian hak akses.	https://dev-my.its.ac.id/ciba-consent

3.2.3.2.2.5 Kelas *Controller Token Controller*

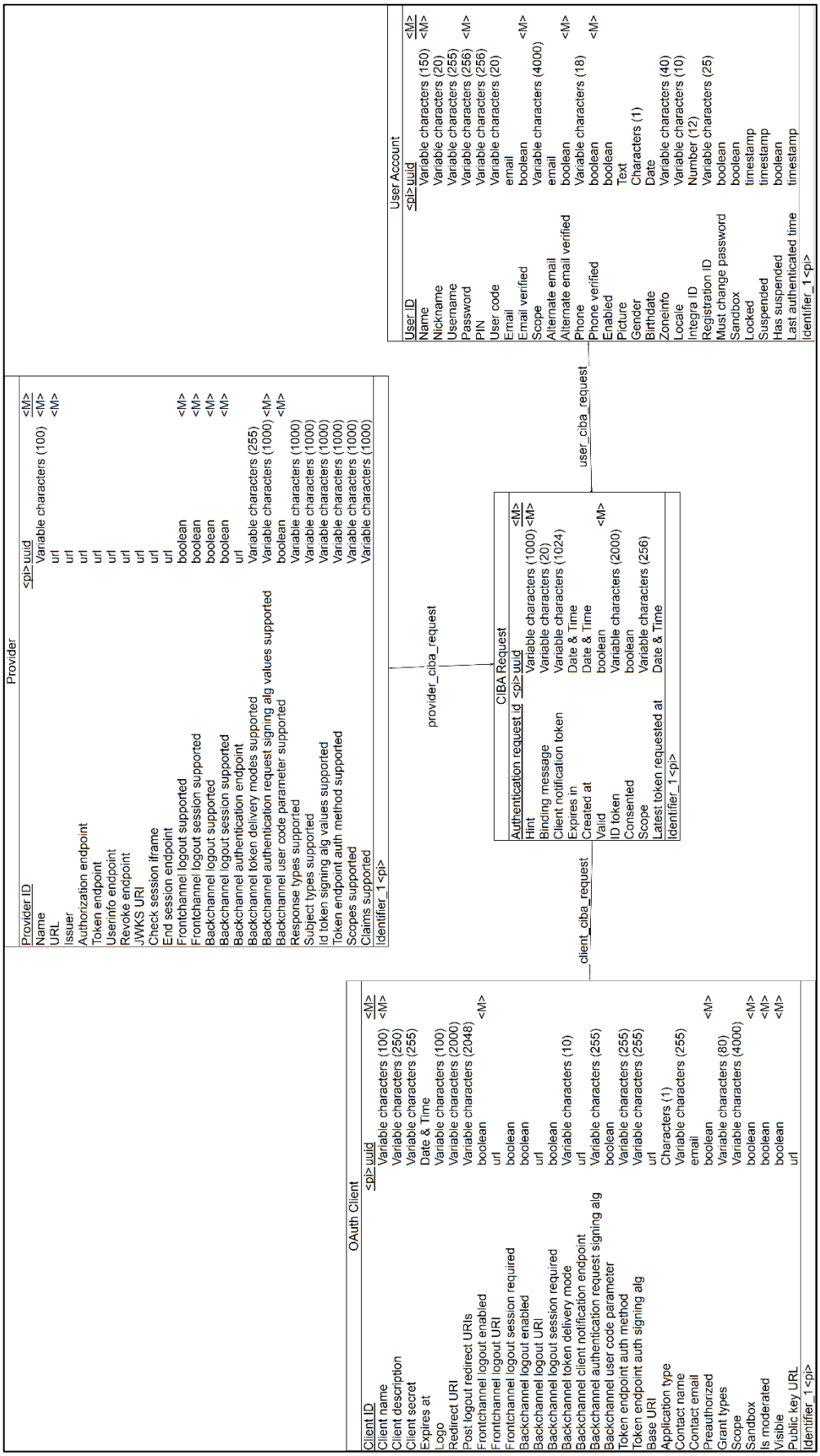
Kelas *Token Controller* digunakan untuk melayani pengambilan *access token*. Kelas ini dapat menangani pengambilan *access token* untuk alur CIBA dan *Authorization Code*. Kelas ini memiliki fungsi yang dinamakan dengan *handleTokenAction*. Fungsi ini dapat dilihat pada Tabel 3.48.

Tabel 3.48 Penjelasan Fungsi Kelas *Token Controller*

Nama Fungsi	Deskripsi	URI
handleTokenAction	Fungsi ini digunakan untuk melayani pengambilan *access token*.	https://dev-my.its.ac.id/token

3.2.4 Perancangan Basis Data

Pada subbab ini dijelaskan mengenai rancangan basis data untuk aplikasi yang akan dibangun. Basis data yang digunakan dalam sistem yang akan dibangun dalam tugas akhir ini menggunakan Microsoft SQL Server 2017. Diagram *Conceptual Data Model* dapat dilihat pada Gambar 3.31. Terdapat lima tabel yang digunakan pada server autorisasi. Lima tabel tersebut akan dijelaskan sebagai berikut.



Gambar 3.31 Struktur Basis Data CIBA

3.2.4.1 Tabel *OAuth Client*

Tabel *OAuth Client* adalah tabel yang menyimpan data aplikasi klien yang terdaftar menggunakan server autorisasi. Pada tugas akhir ini, tabel *OAuth Client* mendefinisikan aplikasi klien yang terdaftar menggunakan protokol CIBA. Sebagai catatan, tabel *OAuth Client* sudah terdapat dalam sistem myITS SSO namun agar dapat berjalan pada protokol CIBA, telah terjadi penambahan lima kolom baru. Kolom tambahan dapat dilihat pada Tabel 3.49.

Tabel 3.49 Kolom Tabel *OAuth Client*

Nama Kolom	Tipe Data	Deskripsi
Backchannel token delivery mode	*Varchar(10)*	Mode token yang digunakan oleh aplikasi klien.
Backchannel client notification endpoint	*Url*	*Client notification endpoint* yang digunakan oleh aplikasi klien.
Backchannel Authentication request signing alg	*Varchar(255)*	Algoritma yang digunakan untuk *signed request* oleh aplikasi klien.
Backchannel user code parameter	*Boolean*	Konfigurasi untuk menggunakan *use code*.
Public key URL	*Url*	*Public key endpoint* yang digunakan oleh aplikasi klien.

3.2.4.2 Tabel *Provider*

Tabel *Provider* adalah tabel yang menyimpan data *provider* atau server autorisasi. Pada tugas akhir ini, tabel *Provider* mendefinisikan server autorisasi yang memiliki layanan CIBA.

Sebagai catatan, tabel *Provider* sudah terdapat dalam sistem myITS SSO namun agar dapat berjalan pada protokol CIBA, telah terjadi penambahan empat kolom baru. Kolom tambahan dapat dilihat pada Tabel 3.50

Tabel 3.50 Kolom Tabel *Provider*

Nama Kolom	Tipe Data	Deskripsi
Backchannel authentication endpoint	*Url*	*Endpoint* untuk autentikasi *backchannel* yang digunakan oleh server autorisasi.
Backchannel token delivery modes supported	*Varchar(255)*	Mode token yang didukung oleh server autorisasi.
Backchannel authentication request signing alg values supported	*Varchar(1000)*	Algorithma yang didukung oleh server autorisasi untuk *signed request*.
Backchannel user code parameter supported	*Boolean*	Konfigurasi untuk menggunakan *use code*.

3.2.4.3 Tabel *CIBA Request*

Tabel *CIBA Request* adalah tabel yang menyimpan data *request* CIBA. Setiap *request* yang memulai autentikasi backchannel akan mendapatkan *authentication request id* yang merupakan sesi autentikasi dan tercatat pada tabel ini. Tabel ini merupakan tabel baru yang ditambahkan pada server autorisasi myITS SSO. Tabel ini memiliki sebelas kolom. Penjelasan kolom pada tabel ini dapat dilihat pada Tabel 3.51.

Tabel 3.51 Kolom Tabel *CIBA Request*

Nama Kolom	Tipe Data	Deskripsi
Authentication request id	*Uuid*	Identifikasi unik untuk merepresentasikan sesi autentikasi.
Hint	*Varchar(1000)*	Identifikasi akun pengguna yang ingin diautentikasi.
Binding message	*Varchar(20)*	Kode atau pesan untuk melakukan *session binding.*
Client notification token	*Varchar(1024)*	Token untuk autentikasi server autorisasi.
Expires in	*Date & Time*	Waktu kedaluwarsa.
Created at	*Date & Time*	Waktu terjadinya autentikasi.
Valid	*Boolean*	Keberlakuan sesi autentikasi.
ID Token	*Varchar(2000)*	*ID Token* yang didapatkan.
Consented	*Boolean*	Hak akses yang diberikan oleh pengguna.
Scope	*Varchar(256)*	*Scope* yang diminta oleh aplikasi klien.
Latest token requested at	*Date & Time*	Waktu terakhir permintaan token menggunakan suatu *authentication request id* oleh aplikasi klien.

3.2.4.4 Tabel User Account

Tabel *User Account* adalah tabel yang menyimpan data akun pengguna. Sebagai catatan, tabel *User Account* sudah terdapat dalam sistem myITS SSO namun agar dapat berjalan pada protokol CIBA, telah terjadi penambahan satu kolom baru. Kolom tersebut dapat dilihat pada Tabel 3.52.

Tabel 3.52 Kolom Tabel *User Account*

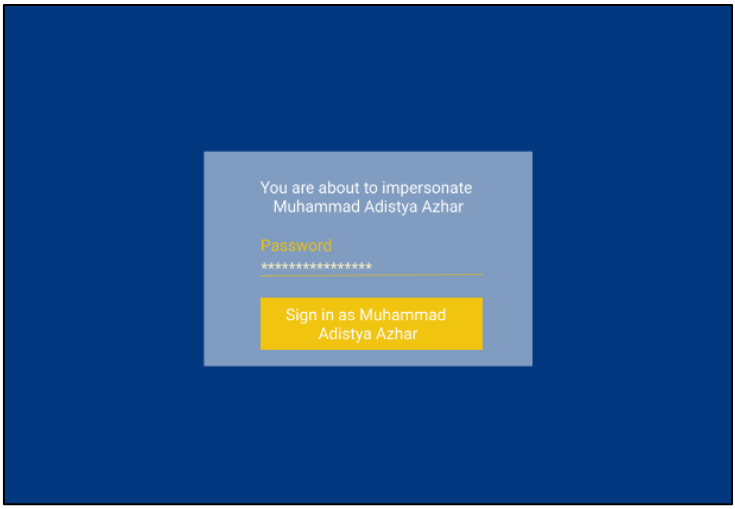
Nama Kolom	Tipe Data	Deskripsi
User code	*Varchar(20)*	Kode rahasia yang dimiliki oleh pengguna.

3.2.5 Perancangan Antarmuka

Tahap perancangan antarmuka dalam subbab ini membuka perancangan antarmuka dari sistem. Perancangan antarmuka ini bertujuan untuk memberikan gambaran proses pengembangan aplikasi mengenai tampilan antarmuka aplikasi.

3.2.5.1 Halaman Formulir Kata Sandi

Halaman ini digunakan untuk memasukkan kata sandi pengguna yang ingin berperan sebagai akun yang berbeda. Halaman ini ditampilkan pada Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004) pada subbab 3.2.1.4. Perancangan antarmuka ini dapat dilihat pada Gambar 3.32.



Gambar 3.32 Antarmuka Formulir Kata Sandi

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi dari perancangan aplikasi yang meliputi *library* server autorisasi CIBA dan aplikasi server autorisasi CIBA yang menggunakan *library* server autorisasi CIBA.

4.1 Lingkungan Implementasi

Dalam membangan aplikasi ini digunakan beberapa perangkat pendukung baik perangkat keras maupun perangkat lunak. Lingkungan pembangunan dijelaskan sebagai berikut.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang dipakai dalam pembuatan ini memiliki spesifikasi sebagai berikut.

- Prosesor Intel® Core™ i5-7200U
- Memori RAM 12 GB
- Sistem Operasi Ubuntu 18.04

4.1.2 Lingkungan Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan untuk membuat aplikasi ini yakni sebagai berikut.

- Jetbrains PhpStorm sebagai IDE
- Microsoft SQL Server sebagai basis data
- Redis sebagai basis data
- Power Designer v15 sebagai aplikasi pemodelan UML
- DrawIO UML sebagai aplikasi pemodelan UML
- Modelio UML sebagai aplikasi pemodelan UML
- Nginx 1.14 sebagai *website server*
- Xdebug sebagai *tool debugger*

4.1.3 Implementasi Library Server Autorisasi CIBA

Dalam subbab ini dijelaskan implementasi kelas pada *library* server autorisasi CIBA.

4.1.3.1 Implementasi Kelas

Dalam subbab ini dijelaskan implementasi kelas yang berasal dari abstraksi yang telah didapatkan pada subbab 3.2.3.1.2. Implementasi kelas yang dihasilkan akan dikelompokkan sesuai dengan rumusan permasahalan. Pada implementasi *library* server autorisasi CIBA, rumusan masalah "Bagaimana mengimplementasi alur CIBA pada server autorisasi myITS Single Sign-On" akan dibahas pada subbab 4.1.3.1.1, "Bagaimana cara mengimplementasi pengambilan *access token* menggunakan mode *poll*, *ping*, dan *push*" akan dibahas pada subbab 4.1.3.1.2, dan "Bagaimana mengimplementasi *session binding* antara CD dan AD" akan dibahas pada 4.1.3.1.3.

4.1.3.1.1 Penerapan Alur CIBA

Alur CIBA dimulai dengan *request* autentikasi dari aplikasi klien yang ditujukan ke server autorisasi. Di fase ini, server autorisasi dapat melanjutkan prosedur pengiriman hak akses, pemberian token dan sebagainya jika *request* autentikasi telah memenuhi syarat. Pada tugas akhir ini, penerapan alur CIBA menghasilkan empat kelas sebagai berikut.

4.1.3.1.1.1 Kelas *Ciba Controller*

Kelas *Ciba Controller* adalah kelas yang memiliki tanggung jawab untuk menangani *request* autentikasi, mengirim pesan notifikasi ke aplikasi klien, dan mengirim pesan hak akses ke *Authentication Device*. Kelas *Ciba Controller* akan bergantung pada abstraksi *Ciba Controller Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.1. Kelas ini memiliki lima fungsi umum sama halnya dengan abstraksi *Ciba Controller Interface*.

4.1.3.1.1.1.1 Fungsi *Handle Authentication Request*

Fungsi pertama yang akan dijelaskan adalah fungsi *handleAuthenticationRequest*. Fungsi ini memiliki tugas untuk menangani *request* autentikasi yang berasal dari aplikasi klien. Fungsi ini dapat dilihat pada Kode Sumber 4.1, dan Kode Sumber 4.2. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 9, *library* melakukan validasi *request* autentikasi. Metadata pada obyek *request* akan disimpan pada variabel *$this->cibaRequest* dan metadata aplikasi klien pada variabel *$this->clientDetail*.
2. Pada baris 11, *library* menyusun *authentication request id* sebagai bentuk identifikasi *request* yang masuk.
3. Pada baris 14 sampai 20, *library* menyusun *ID Token* jika mode token yang digunakan adalah *poll* atau *ping*. *ID Token* untuk mode token *push* tidak akan disusun saat ini, karena *ID Token* pada mode token *push* membutuhkan *hash access token* yang berarti bahwa *access token* harus disusun terlebih dahulu.
4. Pada baris 22 sampai 32, *library* membuat *response* CIBA yang merepresentasikan *request* CIBA saat ini. *Response* yang dibuat akan disimpan ke basis data. Pada bagian ini, telah terjadi kontrak *session binding* antara *aplikasi klien* dengan *Authentication Device* karena *binding message* telah teridentifikasi dan tersimpan.
5. Pada baris 34 sampai 44 *library* mengirim permintaan hak akses ke *Authentication Device*. Pesan keberhasilan *request* autentikasi berada pada variabel *$response* yang dapat dikirim ke aplikasi klien.
6. Baris 46 sampai 47 berupa pesan *error* jika *request* autentikasi tidak berhasil.

```
1.  public function handleAuthenticationRequest(RequestInterface $re
    quest, ResponseInterface $response)
2.    {
3.      try {
4.        if (!$this-
    >validateAuthenticationRequest($request, $response)) {
5.          return false;
6.        }
7.      } catch (Exception $e) {
8.        throw $e;
9.      }
10.
11.     $authReqId = $this->cibaResponse->generateAuthReqId();
12.     $idToken = null;
13.
14.     if ($this-
    >clientDetail['backchannel_token_delivery_mode'] !== self::PUSH_ME
    THOD) {
15.       $idToken = $this->idToken->createCibaIdToken(
16.         $this->clientDetail['client_id'],
17.         $this->userDetail,
18.         $authReqId
19.       );
20.     }
21.
22.     $authenticationRequest = $this->
23.     cibaResponse->createAuthenticationRequest($authReqId,
24.       $this->clientDetail['client_id'],
25.       $this->clientDetail['provider_id'],
26.       $this->cibaRequest['hint'],
27.       $this->cibaRequest['requested_expiry'],
28.       $this->clientDetail['backchannel_token_delivery_mode'],
29.       $this->cibaRequest['client_notification_token'],
30.       $this->cibaRequest['hint'],
31.       $this->cibaRequest['binding_message'],
32.       $this->cibaRequest['scope'], $idToken);
33.
34.     if ($authenticationRequest) {
35.       $response->addParameters($authenticationRequest);
36.
```

Kode Sumber 4.1 Fungsi *Handle Authentication Request* (1)

```
37.         $payload = $this-
>buildAuthDevicePayload($authReqId,
38.             $this->clientDetail['client_name'],
39.             $this->cibaRequest['binding_message']);
40.
41.         $this->notifyAuthenticationDevice($payload);
42.
43.         return true;
44.     }
45.
46.     $response-
>setError(500, 'internal_server', 'An error occurred');
47.     return false;
48.  }
```

Kode Sumber 4.2 Fungsi *Handle Authentication Request* (2)

4.1.3.1.1.1.2 Fungsi *Validate Authentication Request*

Fungsi kedua yang akan dijelaskan adalah *validateAuthenticationRequest*. Fungsi ini memiliki tugas untuk melakukan validasi *request* autentikasi yang masuk. Fungsi ini dapat dilihat pada Kode Sumber 4.3. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 terdapat variabel *$ signedRequestAndPassedValidation* yang digunakan sebagai penanda *request* autentikasi merupakan *signed request*. Saat ini nilainya masih *null* karena *library* belum tahu bentuk *request* yang masuk.
2. Pada baris 5 sampai 10, *library* melakukan pemeriksaan tipe autentikasi yang digunakan oleh aplikasi klien. Gunanya adalah untuk mengetahui apakah aplikasi klien dapat menggunakan layanan CIBA, dan juga bentuk *request* (*signed* atau *non-signed*). Jika gagal, maka *library* akan mengembalikan nilai *false*.

3. Pada baris 12 sampai 16, *library* melakukan pemeriksaan parameter *request*. Peraturan pemeriksaan telah dijelaskan pada subbab 2.4.2.3. Jika gagal, maka *library* akan mengembalikan nilai *false*.
4. Pada baris 18, *library* mengembalikan nilai *true* sebagai bentuk tanda berhasil.

```
1.  public function validateAuthenticationRequest(RequestInterface $request, ResponseInterface $response)
2.  {
3.      $signedRequestAndPassedValidation = null;
4.
5.      if (! $clientAuthMethod = $this->
6.      checkClientAuthenticationMethod($request,
7.      $response,
8.      $signedRequestAndPassedValidation)) {
9.          return false;
10.     }
11.
12.     if (is_null($signedRequestAndPassedValidation)
13.     && ! $authenticationRequestParameters
14.     = $this->checkAuthenticationRequestParameters($request, $response)) {
15.         return false;
16.     }
17.
18.     return true;
19. }
```

Kode Sumber 4.3 Fungsi *Validate Authentication Request*

4.1.3.1.1.1.3 Fungsi *Handle User Consent*

Fungsi ketiga yang akan dijelaskan adalah *handleUserConsent*. Fungsi ini memliki tugas untuk menangani pemberian hak akses oleh pengguna yang berasal dari *Authentication Device*. Fungsi ini dapat dilihat pada Kode Sumber

4.4, Kode Sumber 4.5, Kode Sumber 4.6, Kode Sumber 4.7, dan Kode Sumber 4.8. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 6, *library* melakukan validasi dengan memeriksa parameter *authentication request id* dan hak akses yang diberikan. Jika tidak valid maka *library* akan mengembalikan nilai *false*.
2. Pada baris 8 sampai 22, *library* memeriksa *authentication request id* adalah valid dan tersimpan di basis data. Jika tidak valid, *library* akan mengembalikan nilai *false* dan pesan *error*.
3. Pada baris 24 sampai 27, *library* melakukan validasi *authentication requet id* yang telah didapatkan. Validasi gunanya untuk memastikan bahwa *authentication reques id* tidak kedaluwarsa, dan belum pernah digunakan.
4. Pada baris 29 sampai 31, *library* mengambil data aplikasi klien berdasarkan *client id* yang diberikan.
5. Pada baris 33 sampai 37, *library* memastikan bahwa *client id* yang tercatat pada *authentication request id* adalah sama dengan *client id* yang tercatat pada aplikasi klien. Gunanya untuk mencegah penggunaan *authentication request id* yang bukan milik aplikasi klien.
6. Pada baris 39, *library* mendapatkan *client notification token* milik aplikasi klien.
7. Pada baris 42 sampai 50, *library* memeriksa hak akses yang diberikan oleh *Authentication Device*. Jika hak akses tidak sesuai format, maka *library* akan mengembalikan pesan *error*.
8. Pada baris 51, *library* menyimpan hak akses yang diberikan.
9. Pada baris 54 sampai 69, *library* memeriksa mode token yang digunakan adalah *push*. Jika aplikasi klien tidak diberi hak akses, maka *library* akan mengembalikan pesan *error* dengan cara mengirim pesan ke aplikasi klien melalui *client notification endpoint*.

10. Pada baris 70 sampai 96, *library* memeriksa mode token yang digunakan adalah *push*. Jika aplikasi klien diberi hak akses, maka *library* akan menyusun *access token* dan *ID Token* untuk dikirim ke aplikasi klien. Pesan ini dikirim melalui *client notification endpoint*.
11. Pada baris 99 sampai 100, *library* menyimpan *authentication request id* sebagai kedaluwarsa agar tidak dapat digunakan kedepannya.
12. Pada baris 103 sampai 113, *library* memeriksa mode token yang digunakan adalah *ping*. *Library* akan mengirim pesan notifikasi ke aplikasi klien melalui *client notification endpoint* agar aplikasi klien dapat meminta token. Aplikasi klien baru dapat tahu status autentikasi dan hak akses ketika mengeksekusi token *endpoint*.
13. Pada baris 115 sampai 116, *library* mengembalikan pesan. Aplikasi klien baru dapat tahu status autentikasi dan hak akses ketika mengeksekusi berulang-ulang (*polling)* token *endpoint*.

```
1.  public function handleUserConsent(RequestInterfa
    ce $request, ResponseInterface $response)
2.      {
3.          if (! $consentValid = $this
4.              -
    >checkUserConsentValidation($request, $response)
    ) {
5.              return false;
6.          }
7.
8.          try {
9.              $authReqIdData = $this
10.                 ->cibaStorage
11.                 ->getAuthReqId($request-
    >request('auth_req_id'));
12.             if ($authReqIdData === null || !$aut
    hReqIdData) {
13.                 $this-
    >setInvalidGrantError($response);
14.
15.                 return false;
16.             }
17.         }
18.         catch (Exception $e) {
19.             $this-
    >setInvalidGrantError($response);
20.
21.             return false;
22.         }
23.
24.         if (! $authReqIdValid =
25.             $this-
    >checkAuthReqIdValidation($response, $authReqIdD
    ata)) {
26.             return false;
27.         }
28.
```

Kode Sumber 4.4 Fungsi *Handle User Consent* (1)

```
29.         $clientData = $this
30.             ->clientStorage
31.             -
    >getClientDetails($authReqIdData['client_id']);

32.
33.         if ($clientData['client_id'] !== $authRe
    qIdData['client_id']) {
34.             $this-
    >setInvalidGrantError($response);
35.
36.             return false;
37.         }
38.
39.         $clientNotificationEndpoint = $clientDat
    a['backchannel_client_notification_endpoint'];

40.
41.
42.         $consentStatus = self::USER_CONSENT_MAPP
    ING[$request->request('user_consent')];
43.
44.         if (! in_array($request-
    >request('user_consent'),
45.         [self::USER_CONSENT_ACCEPT, self::USER_C
    ONSENT_DENY])) {
46.             $response-
    >addParameters(['auth_req_id' => $authReqIdData[
    'auth_req_id']]);
47.             $response-
    >setError(400, 'transaction_failed', 'The user_c
    onsent is invalid.');
48.
49.             return false;
50.         }
51.         $this->cibaStorage-
    >setConsent($authReqIdData['auth_req_id'], $cons
    entStatus);
```

Kode Sumber 4.5 Fungsi *Handle User Consent* (2)

```
52.
53.         switch ($clientData['backchannel_token_deli
    very_mode']) {
54.             case self::PUSH_METHOD:
55.                 if ($request-
    >request('user_consent') === self::USER_CONSENT_DEN
    Y) {
56.                     $body = [
57.                         'auth_req_id' => $authReqId
    Data['auth_req_id'],
58.                         'error' => 'access_denied',

59.                         'error_description' => 'The
     end-user denied the authorization request.'
60.                     ];
61.
62.                     $this->notifyRelyingParty(
63.                         $clientNotificationEndpoint
    ,
64.                         $authReqIdData['client_noti
    fication_token'],
65.                         $body,
66.                         403
67.                     );
68.
69.                 } else if ($request-
    >request('user_consent') === self::USER_CONSENT_ACC
    EPT) {
70.                     $token = $this->cibaGrantType-
    >createAccessToken(
71.                         $this->accessToken,
72.                         $clientData['client_id'],

73.                         $authReqIdData['user_id'],
```

Kode Sumber 4.6 Fungsi *Handle User Consent* (3)

```
74.                         $authReqIdData['scope']

75.                     );
76.
77.                     $idToken = $this->idToken-
    >createCibaIdToken(
78.                         $clientData['client_id'
    ],
79.                         $authReqIdData['user_id
    '],
80.                         $authReqIdData['auth_re
    q_id'],
81.                         $token['access_token']
    ? $token['access_token'] : null,
82.                         $token['refresh_token']
     ? $token['refresh_token'] : null
83.                     );
84.
85.                     $pushResponse = [
86.                         'auth_req_id' => $authR
    eqIdData['auth_req_id'],
87.                         'access_token' => $toke
    n['access_token'],
88.                         'expires_in' => $token[
    'expires_in'],
89.                         'id_token' => $idToken

90.                     ];
91.
92.                     $this-
    >notifyRelyingParty(
93.                         $clientNotificationEndp
    oint,
94.                         $authReqIdData['client_
    notification_token'],
95.                         $pushResponse
96.                     );
97.                 }
98.
```

Kode Sumber 4.7 Fungsi *Handle User Consent* (4)

```
99.                 $this->cibaStorage-
   >expireAuthReqId($authReqIdData['auth_req_id']);

100.                $response-
   >addParameters(['message' => $this-
   >statusTexts[200]]);
101.
102.                break;
103.            case self::PING_METHOD:
104.                $body = [
105.                    'auth_req_id' => $authReqIdDat
   a['auth_req_id']
106.                ];
107.                $this->notifyRelyingParty(
108.                    $clientNotificationEndpoint,

109.                    $authReqIdData['client_notific
   ation_token'],
110.                    $body
111.                );
112.
113.                $response-
   >addParameters(['message' => $this-
   >statusTexts[200]]);
114.                break;
115.            case self::POLL_METHOD:
116.                $response-
   >addParameters(['message' => $this-
   >statusTexts[200]]);
117.                break;
118.            default:
119.                break;
120.        }
121.
122.        return false;
123.    }
```

Kode Sumber 4.8 Fungsi *Handle User Consent* (5)

4.1.3.1.1.1.4 Fungsi *Notify Relying Party*

Fungsi keempat yang akan dijelaskan adalah *notifyRelyingParty*. Fungsi ini memiliki tugas untuk mengirim pesan ke aplikasi klien. Pada mode token *push*, fungsi ini digunakan untuk mengirim token ke aplikasi klien. Sedangkan pada mode token *ping*, fungsi ini digunakan untuk mengirim pesan notifikasi ke aplikasi klien sebagai tanda bahwa token dapat diambil. Fungsi ini dapat dilihat pada Kode Sumber 4.9. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 5 sampai 9, *library* menyusun *payload* yang akan dikirim ke aplikasi klien.
2. Pada baris 11, *library* mengirim pesan ke aplikasi klien dengan infrastruktur yang sudah terinjeksi.

```
1.  public function notifyRelyingParty
2.  ($url, $client_notification_token,
3.  $post_body = null, $statusCode = 200)
4.    {
5.      $payload = [
6.        'client_notification_token' => $client_notification_token,
7.        'post_body' => $post_body,
8.        'status_code' => $statusCode
9.      ];
10.
11.     return $this->rpInfrastructure->send($payload, $url);
12.   }
```

Kode Sumber 4.9 Fungsi *Notify Relying Party*

4.1.3.1.1.1.5 Fungsi *Notify Authentication Device*

Fungsi kelima yang akan dijelaskan adalah *notifyAuthenticationDevice*. Fungsi ini memiliki tugas untuk mengirim pesan hak akses ke *Authentication Device*. Pesan ini berupa permintaan hak akses. Fungsi ini dapat dilihat pada Kode Sumber 4.10. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* akan menyusun nama topik yang digunakan sebagai rute penyampaian pesan.
2. Pada baris 4, *library* akan mengirim pesan dengan *payload* dan nama topik ke *Authentication Device*.

```
1. public function notifyAuthenticationDevice
2. ($payload)
3. {
4.    $topicName = $this->getTopicName($this-
   >userDetail['user_id']);
5.    $this->authenticationDeviceTransport-
   >send($payload, $topicName);
6. }
```

Kode Sumber 4.10 Fungsi *Notify Authentication Device*

4.1.3.1.1.2 Kelas *Authorize Controller*

Kelas *Authorize Controller* adalah kelas yang memiliki tanggung jawab untuk menangani autorisasi aplikasi klien. Kelas ini berjalan pada alur *Authorization Code*. Namun pada pengerjaan tugas akhir ini, kelas *Authorize Controller* digunakan untuk memenuhi Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004). Implementasi yang dilakukan adalah memodifikasi fungsi *buildAuthorizeParameters*. Fungsi *buildAuthorizeParameters* akan dipanggil oleh fungsi *handleAuthorizeRequest* pada obyek yang sama. Fungsi *handleAuthorizeRequest* tidak dilakukan modifikasi.

4.1.3.1.1.2.1 Fungsi *Build Authorize Parameters*

Fungsi pertama yang akan dijelaskan adalah *buildAuthorizeParameters*. Salah satu tugas yang dilakukan oleh fungsi ini adalah menyusun *ID Token* ketika melakukan autorisasi menggunakan *Authorization Code*. Fungsi ini dapat dilihat pada Kode Sumber 4.11 dan Kode Sumber 4.12. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 4, *library* mengambil data aplikasi klien berdasarkan *client id* pada basis data.
2. Pada baris 6 sampai 7, *library* menyimpan nilai yang menentukan penggunaan *frontchannel logout* dan *backchannel logout* pada aplikasi klien ini.
3. Pada baris 9, *library* menyusun dan menyimpan *session id* pada sesi autorisasi ini.
4. Pada baris 11 terdapat *statement* yang akan tereksekusi jika alur yang digunakan membutuhkan *ID Token*. Karena alur yang digunakan adalah *Authorization Code*, maka alur ini membutuhkan *ID Token*.
5. Pada baris 12, *library* memeriksa jika aplikasi klien menggunakan *frontchannel logout* atau *backchannel logout*. Jika membutuhkannya, maka penyusunan ID Token akan disisipkan *session id* yang disusun pada baris 9.
6. Pada baris 13 sampai 16, *library* memeriksa *parameter* "*impersonated_sub*" pada *request* HTTP. Nilai dari *"impersonated_sub"* adalah identifkasi akun pengguna yang dilakukan *impersonation*. Nilai ini merupakan *user id* akun pengguna. Jika terdapat parameter *"impersonated_sub"* maka *library* memanggil fungsi *createImpersonatedIdToken* pada kelas *ID Token Response Type*. Jika tidak tedapat parameter "*impersonated_sub*" maka *library* memanggil fungsi *createIdToken* pada kelas *ID Token Response Type*. Kelas *ID Token Response Type* dijelaskan pada subbab 4.1.3.1.1.4.
7. Pada baris 18, *library* menyusun *ID Token* untuk aplikasi klien yang tidak terdaftar menggunakan *frontchannel logout* atau *backchannel logout*. Ini merupakan kebalikan dari yang dijelaskan pada langkah 5. Perbedaanya adalah jika tidak menggunakan *frontchannel logout* atau *backchannel*

logout, *ID Token* tidak memiliki *claim session id*. Penyusunan *ID Token* serupa dengan yang dijelaskan pada 6.

```
1.  protected function buildAuthorizeParameters($request, $res
    ponse, $user_id, $user_agent)
2.    {
3.
4.      $clientData = $this->clientStorage->getClientDetails($this-
    >getClientId());
5.
6.      $frontchannelLogoutSessionRequired = $clientData['front
    channel_logout_session_required'];
7.      $backchannelLogoutSessionRequired = $clientData['backc
    hannel_logout_session_required'];
8.
9.      $session_id = $this-
    >setSessionID($user_agent, $user_id, $clientData['client_id']);
10.
11.     if ($this->needsIdToken($this->getScope()) && $this-
    >getResponseType() == self::RESPONSE_TYPE_AUTHORIZATI
    ON_CODE) {
12.         if ($frontchannelLogoutSessionRequired || $backchanne
    lLogoutSessionRequired) {
13.             if ($request->query('impersonated_sub')) {
14.                 $params['id_token'] = $this-
    >responseTypes['id_token']-
    >createImpersonatedIdToken($this-
    >getClientId(), $user_id, $this-
    >nonce, null, null, $session_id, $request-
    >query('impersonated_sub'));
15.             } else {
16.                 $params['id_token'] = $this-
    >responseTypes['id_token']->createIdToken($this-
    >getClientId(), $user_id, $this->nonce, null, null, $session_id);
17.             }
```

Kode Sumber 4.11 Fungsi *Build Authorize Parameters* (1)

```
18.         } else {
19.             if ($request->query('impersonated_sub')) {
20.                 $params['id_token'] = $this-
>responseTypes['id_token']-
>createImpersonatedIdToken($this-
>getClientId(), $user_id, $this->nonce, null, null, null, $request-
>query('impersonated_sub'));
21.             } else {
22.                 $params['id_token'] = $this-
>responseTypes['id_token']->createIdToken($this-
>getClientId(), $user_id, $this->nonce);
23.             }
24.         }
25.     }
26.
27.     $params['nonce'] = $this->nonce;
28.
29.     return $params;
30. }
```

Kode Sumber 4.12 Fungsi *Build Authorize Parameters* (2)

4.1.3.1.1.3 Kelas *Ciba Response Type*

Kelas *Ciba Response Type* adalah kelas yang memiliki tanggung jawab menyusun *authentication request id* yang merepresentasikan suatu sesi autentikasi pada protocol CIBA. Kelas *Ciba Response Type* akan bergantung pada abstraksi *Ciba Response Type Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.4. Kelas ini memiliki dua fungsi umum sama halnya dengan abstraksi *Ciba Response Type Interface*.

4.1.3.1.1.3.1 Fungsi *Create Authentication Request*

Fungsi pertama yang akan dijelaskan adalah *createAuthenticationRequest*. Fungsi ini memiliki tugas untuk menyusun *authentication request*. Fungsi ini dapat dilihat pada Kode Sumber 4.13. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 4 sampai 5, *library* menyimpan waktu kedaluwarsa sebagai 120 detik dari waktu saat ini jika tidak diberikan nilai secara eksplisit.
2. Pada baris 8, *library* menambahkan waktu kedaluwarsa.
3. Pada baris 10, *library* menyusun *authentication request* dan menyimpan ke basis data.
4. Pada baris 12 sampai 15, *library* mengembalikan pesan *authentication request* dan disisipkan nilai interval lima detik jika mode token adalah *poll*.
5. Pada baris 17, *library* mengembalikan pesan *authentication request* tanpa nilai interval.
6. Pada baris 20, *library* mengembalikan pesan nilai *false* sebagai tanda gagal.

```
1.  public function createAuthenticationRequest
2.  ($authReqId, $client_id, $provider_id, $user_id, $addedSecond
    sToExpire, $mode, $client_notification_token = null, $hint = nul
    l, $binding_message = null, $scope = null, $id_token = null)
3.    {
4.      if (! $addedSecondsToExpire) {
5.        $addedSecondsToExpire = 120;
6.      }
7.
8.      $expires = $addedSecondsToExpire + time();
9.
10.     $response = $this->cibaStorage-
    >setAuthReqId($authReqId, $client_id, $provider_id, $user_id,
    $expires, $client_notification_token, $hint, $binding_message,
    $scope, $id_token);
11.
12.     if ($response) {
13.       if ($mode === self::POLL_METHOD) {
14.         $defaultInterval = 5;
15.         return $this-
    >createResponse($authReqId, $expires, $defaultInterval);
16.       }
17.       return $this->createResponse($authReqId, $expires);
18.     }
19.
20.     return false;
21.   }
```

Kode Sumber 4.13 Fungsi *Create Authentication Request*

4.1.3.1.1.3.2 Fungsi *Generate Authentication Request Id*

Fungsi kedua yang akan dijelaskan adalah *generateAuthenticationRequestId*. Fungsi ini memiliki tugas untuk menyusun *authentication request id* sebagai nilai yang unik. Fungsi ini dapat dilihat pada Kode Sumber 4.14. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 12, *library* menyusun *authentication request id* dengan nilai yang unik. Nilai

ini berupa format UUID (*Universally Unique Identifier*).

2. Pada baris 12, *library* mengembailkan nilai *authentication request id*.

```
1.  public function generateAuthReqId()
2.  {
3.    $uuid = sprintf( '%04x%04x-%04x-%04x-%04x-
    %04x%04x%04x',
4.        mt_rand( 0, 0xffff ),
5.        mt_rand( 0, 0xffff ),
6.        mt_rand( 0, 0xffff ),
7.        mt_rand( 0, 0x0fff ) | 0x4000,
8.        mt_rand( 0, 0x3fff ) | 0x8000,
9.        mt_rand( 0, 0xffff ),
10.       mt_rand( 0, 0xffff ),
11.       mt_rand( 0, 0xffff )
12.   );
13.
14.   return strtoupper($uuid);
15. }
```

Kode Sumber 4.14 Fungsi *Generate Authentication Request Id*

4.1.3.1.1.4 Kelas *ID Token* Response Type

Kelas *ID Token Response Type* memiliki tanggung jawab untuk menyusun *response* dalam bentuk *ID Token*. Kelas *ID Token Response Type* akan bergantung pada abstraksi *ID Token Response Type Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.5. Kelas ini memiliki empat fungsi umum sama halnya dengan abstraksi *ID Token Response Type Interface*.

4.1.3.1.1.4.1 Fungsi *Get Authorize Response*

Fungsi pertama yang akan dijelaskan adalah *getAuthorizeResponse*. Fungsi ini memiliki tugas untuk menyusun *redirect URI* untuk mendapatkan *authorization response*. Fungsi ini digunakan pada alur *Authorization Code* dan tidak digunakan

pada alur CIBA. Fungsi ini dapat dilihat pada Kode Sumber 4.15. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 4 sampai 5, *library* menyusun struktur data untuk menyimpan *query parameter* yang digunakan sebagai *redirect URI*.
2. Pada baris 8 sampai 9, *library* mengambil data akun pengguna yang digunakan sebagai *claim* atau fakta pada *ID Token*.
3. Pada baris 11 sampai 14, *library* menyusun *ID Token*.
4. Pada baris 17, *library* mengembalikan pesan dalam bentuk *redirect URI* dan *ID Token*.

```
1.  public function getAuthorizeResponse($params, $userInfo =
    null)
2.  {
3.     // build the URL to redirect to
4.     $result = array('query' => array());
5.     $params += array('scope' => null, 'state' => null, 'nonce' =>
    null);
6.
7.     // create the ID Token.
8.     list($user_id, $auth_time) = $this-
    >getUserIdAndAuthTime($userInfo);
9.     $userClaims = $this->userClaimsStorage-
    >getUserClaims($user_id, $params['scope']);
10.
11.    $id_token = $this-
    >createIdToken($params['client_id'], $userInfo, $params['non
    ce'], $userClaims, null);
12.    $result["fragment"] = array('id_token' => $id_token);
13.    if (isset($params['state'])) {
14.       $result["fragment"]["state"] = $params['state'];
15.    }
16.
17.    return array($params['redirect_uri'], $result);
18. }
```

Kode Sumber 4.15 Fungsi *Get Authorize Response*

4.1.3.1.1.4.2 Fungsi *Create ID Token*

Fungsi kedua yang akan dijelaskan adalah *createIdToken*. Fungsi ini memiliki tugas untuk menyusun *ID Token*. Fungsi ini digunakan pada alur *Authorization Code* dan tidak digunakan pada alur CIBA. Fungsi ini dapat dilihat pada Kode Sumber 4.16. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengambil data akun pengguna.
2. Pada baris 5 sampai 11, *library* menyusun struktur data *ID Token*.
3. Pada baris baris 14 sampai 28, *library* menambahkan *claim* berbeda jika dibutuhkan.
4. Pada baris 30¸ *library* menyusun *ID Token* dengan cara menggunakan *algoritma asimetris*.

```
1.  public function createIdToken($client_id, $userInfo, $nonce
    = null, $userClaims = null, $access_token = null, $session_id =
    null)
2.    {
3.        list($user_id, $auth_time) = $this-
    >getUserIdAndAuthTime($userInfo);
4.
5.        $token = array(
6.            'iss'       => $this->config['issuer'],
7.            'sub'       => $user_id,
8.            'aud'       => $client_id,
9.            'iat'       => time(),
10.           'exp'       => time() + $this->config['id_lifetime'],
11.           'auth_time'  => $auth_time,
12.       );
13.
14.       if ($session_id) {
15.           $token['sid'] = $session_id;
16.       }
17.
18.       if ($nonce) {
19.           $token['nonce'] = $nonce;
20.       }
21.
22.       if ($userClaims) {
23.           $token += $userClaims;
24.       }
25.
26.       if ($access_token) {
27.           $token['at_hash'] = $this-
    >createHash($access_token, $client_id);
28.       }
29.
30.       return $this->encodeToken($token, $client_id);
31.   }
```

Kode Sumber 4.16 Fungsi *Create ID Token*

4.1.3.1.1.4.3 Fungsi *Create Ciba ID Token*

Fungsi ketiga yang akan dijelaskan adalah *createCibaIdToken*. Fungsi ini memiliki tugas untuk menyusun *ID Token*. Fungsi ini digunakan pada alur CIBA. Perbedaan fungsi ini dengan fungsi *createIdToken* pada subbab 4.1.3.1.1.4.2 adalah *claim* yang disisipkan pada *ID Token*. *Claim* pada *ID Token* CIBA memiliki tambahan "*urn:openid:params:jwt:claim:auth_req_id"* yang menyimpan nilai *authentication request id*, "*urn:openid:params:jwt:claim:rt_hash"* yang menyimpan *hash refresh token*, dan "*at_hash*" yang menyimpan *hash access token*. *Claim* tersebut wajib pada mode token *push*.

Mekanisme fungsi *hash* yang dapat dilihat pada Kode Sumber 4.17 adalah sebagai berikut:

1. Gunakan algorima *hashing* sesuai dengan nilai "*alg"* yang terdapat pada *header ID Token*. Sebagai contoh jika nilai "*alg"* adalah RS256 atau HS256 maka gunakan algoritma *hashing* SHA256, jika nilai *"alg"* adalah RS512 atau HS512 maka gunakan algoritma *hashing* SHA512 dan seterusnya.
2. Masukkan nilai *access token* atau *refresh token* sebagai nilai yang akan dilakukan *hash*.
3. Lalu pisah hasil hash menjadi dua bagian, dan ambil bagian kiri.
4. Hasil *access token* atau *refresh token hash* akan dilakukan *base64url encode*.

```
1.  protected function createHash($token, $client_id = null)
2.  {
3.      // maps HS256 and RS256 to sha256, etc.
4.      $algorithm = $this->publicKeyStorage-
    >getEncryptionAlgorithm($client_id);
5.      $hash_algorithm = 'sha' . substr($algorithm, 2);
6.      $hash = hash($hash_algorithm, $token, true);
7.      $at_hash = substr($hash, 0, strlen($hash) / 2);
8.
9.      return $this->encryptionUtil-
    >urlSafeB64Encode($at_hash);
10. }
```

Kode Sumber 4.17 Fungsi *Hash*

Fungsi *createCibaIdToken* dapat dilihat pada Kode Sumber 4.18. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengambil data akun pengguna.
2. Pada baris 5 sampai 13, *library* menyusun struktur data *ID Token*.
3. Pada baris baris 14 sampai 21, *library* menambahkan *claim* berbeda jika dibutuhkan.
4. Pada baris 23, *library* menyusun *ID Token* dengan cara menggunakan *algoritma asimetris*.

```
1.  public function createCibaIdToken($client_id, $userInfo, $au
    th_req_id, $access_token = null, $refresh_token = null)
2.  {
3.     list($user_id, $auth_time) = $this-
    >getUserIdAndAuthTime($userInfo);
4.
5.     $token = array(
6.        'iss'       => $this->config['issuer'],
7.        'sub'       => $user_id,
8.        'aud'       => $client_id,
9.        'iat'       => time(),
10.       'exp'       => time() + $this->config['id_lifetime'],
11.       'auth_time'  => $auth_time,
12.       'urn:openid:params:jwt:claim:auth_req_id' => $auth_req_i
    d,
13.    );
14.
15.    if ($access_token) {
16.       $token['at_hash'] = $this-
    >createHash($access_token, $client_id);
17.    }
18.
19.    if ($refresh_token) {
20.       $token['urn:openid:params:jwt:claim:rt_hash'] = $this-
    >createHash($refresh_token, $client_id);
21.    }
22.
23.    return $this->encodeToken($token, $client_id);
24. }
```

Kode Sumber 4.18 Fungsi *Create Ciba Id Token*

4.1.3.1.1.4.4 Fungsi *Create Impersonated ID Token*

Fungsi keempat yang akan dijelaskan adalah *createImpersonatedIdToken*. Fungsi ini memiliki tugas untuk menyusun *ID Token* pada alur *Authorization Code* dengan kasus *impersonation*. Perbedaanya dengan fungsi *createIdToken* dan *createCibaIdToken* adalah *claim* yang tersusun pada *ID Token ini*. *ID Token* yang digunakan untuk *impersonation* memiliki *claim*

"*act*" dengan nilai identifikasi akun pengguna yang melakukan *impersonation*. Sedangkan *claim "sub"* adalah identifikasi akun pengguna yang dilakukan *impersonation*. Fungsi ini dapat dilihat pada Kode Sumber 4.19. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengambil data akun pengguna.
2. Pada baris 5 sampai 13, *library* menyusun struktur data *ID Token*.
3. Pada baris baris 15 sampai 29, *library* menambahkan *claim* berbeda jika dibutuhkan.
4. Pada baris 31, *library* menyusun *ID Token* dengan cara menggunakan *algoritma asimetris*.

```
1.  public function createImpersonatedIdToken($client_id, $user
    Info, $nonce = null, $userClaims = null, $access_token = null, $s
    ession_id = null, $impersonating = null)
2.    {
3.      list($user_id, $auth_time) = $this-
    >getUserIdAndAuthTime($userInfo);
4.
5.      $token = array(
6.        'iss'      => $this->config['issuer'],
7.        'act'      => $user_id,
8.        'sub'      => $impersonating,
9.        'aud'      => $client_id,
10.       'iat'      => time(),
11.       'exp'      => time() + $this->config['id_lifetime'],
12.       'auth_time'  => $auth_time,
13.     );
14.
15.     if ($session_id) {
16.       $token['sid'] = $session_id;
17.     }
18.
19.     if ($nonce) {
20.       $token['nonce'] = $nonce;
21.     }
22.
23.     if ($userClaims) {
24.       $token += $userClaims;
25.     }
26.
27.     if ($access_token) {
28.       $token['at_hash'] = $this-
    >createHash($access_token, $client_id);
29.     }
30.
31.     return $this->encodeToken($token, $client_id);
32.   }
```

Kode Sumber 4.19 Penjelasan Fungsi *Create Impersonated ID Token*

4.1.3.1.1.5 Kelas Utama *Server*

Kelas utama *Server* memiliki tanggung jawab untuk membangun obyek kelas-kelas pada *library* server autorisasi CIBA. Prosedur menggunakan *library* ini adalah dengan cara membuat *obyek* dari kelas S*erver*, lalu kelas *Server* memiliki fungsi yang akan memanggil fungsi dari kelas lain. Sebagai contoh jika ingin memulai *request* autentikasi, maka kelas *Server* akan membangun obyek dari kelas *Ciba Controller*, lalu memanggil fungsi *handle Authentication Request* dari kelas *Ciba Controller*. Pada umumnya, kelas *Server* akan melapisi fungsi-fungsi utama pada *library* server autorisasi CIBA. Pada penerapan alur CIBA, telah terimplementasi tiga fungsi sebagai berikut.

4.1.3.1.1.5.1 Fungsi *Handle Authorize Request*

Fungsi pertama yang akan dijelaskan adalah *handleAuthorizeRequest*. Fungsi ini memiliki tugas untuk menangani Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004). Fungsi ini menggunakan alur *Authorization Code*. Fungsi ini dapat dilihat pada Kode Sumber 4.20. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyimpan obyek *response* yang digunakan untuk mengembalikan pesan ke klien.
2. Pada baris 4, *library* mengambil obyek *Authorize Controller* dan memanggil fungsi *handleAuthorizeRequest*. Kelas *Authorize Controller* telah dijelaskan pada subbab 4.1.3.1.1.2.
3. Pada baris 6, *library* mengembalikan obyek *response* sebagai pesan. Obyek *response* menyimpan data seperti *status code* dan *message.*

```
1.  public function handleAuthorizeRequest(RequestInterface $
    request, ResponseInterface $response, $is_authorized, $user_i
    d = null, $user_agent )
2.  {
3.     $this-
    >response = is_null($response) ? new Response() : $response
    ;
4.     $this->getAuthorizeController()-
    >handleAuthorizeRequest($request, $this-
    >response, $is_authorized, $user_id, $user_agent);
5.
6.     return $this->response;
7.  }
```

Kode Sumber 4.20 Fungsi *Handle Authorize Request*

4.1.3.1.1.5.2 Fungsi *Handle Ciba Request*

Fungsi kedua yang akan dijelaskan adalah *handleCibaRequest*. Fungsi ini memiliki tugas untuk menangani *request* autentikasi yang berasal dari aplikasi klien dengan alur CIBA. Fungsi ini dapat dilihat pada Kode Sumber 4.21. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyimpan obyek *response* yang digunakan untuk mengembalikan pesan ke klien.
2. Pada baris 4, *library* mengambil obyek *Ciba Controller* dan memanggil fungsi *handleAuthenticationRequest*. Kelas *Ciba Controller* telah dijelaskan pada subbab 4.1.3.1.1.1.
3. Pada baris 6, *library* mengembalikan obyek *response*.

```
1.  public function handleCibaRequest(RequestInterface $requ
    est, ResponseInterface $response = null)
2.  {
3.     $this-
    >response = is_null($response) ? new Response() : $respons
    e;
4.     $value = $this->getCibaController()-
    >handleAuthenticationRequest($request, $this->response);
5.
6.     return $this->response;
7.  }
```

Kode Sumber 4.21 Fungsi *Handle Ciba Request*

4.1.3.1.1.5.3 Fungsi *Handle Ciba User Consent*

Fungsi ketiga yang akan dijelaskan adalah *handleCibaUserConsent*. Fungsi ini memiliki tugas untuk menangani pemberian hak akses oleh pengguna. Fungsi ini dapat dilihat pada Kode Sumber 4.22. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyimpan obyek *response* yang digunakan untuk mengembalikan pesan ke klien.
2. Pada baris 4, *library* mengambil obyek *Ciba Controller* dan memanggil fungsi *handleCibaUserConsent*. Kelas *Ciba Controller* telah dijelaskan pada subbab 4.1.3.1.1.1.
3. Pada baris 6, *library* mengembalikan obyek *response*.

```
1.  public function handleCibaUserConsent(RequestInterface $re
    quest, ResponseInterface $response = null)
2.  {
3.     $this-
    >response = is_null($response) ? new Response() : $response;

4.     $value = $this->getCibaController()-
    >handleUserConsent($request, $this->response);
5.
6.     return $this->response;
7.  }
```

Kode Sumber 4.22 Fungsi *Handle Ciba User Consent*

4.1.3.1.2 Penerapan Pengambilan *Access Token*

Prosedur pengambilan *access token* pada protokol CIBA memiliki tiga mode yang berbeda yaitu *push*, *ping*, dan *poll*. Pada mode *push*, aplikasi klien tidak perlu mengeksekusi token *endpoint* sehingga token diberikan secara langsung ke *aplikasi klien*. Pemberian *access token* pada mode token *push* telah dijelaskan pada subbab 4.1.3.1.1.1.3, sehingga subbab ini menjelaskan pengambilan *access token* pada mode token *ping* dan *poll*. Pada tugas akhir ini, penerapan pengambilan *access token* menghasilkan empat kelas sebagai berikut.

4.1.3.1.2.1 Kelas *Ciba Token Controller*

Kelas *Ciba Token Controller* adalah kelas yang memiliki tanggung jawab untuk mengakomodir permintaan *access token*. Kelas *Ciba Token Controller* akan bergantung pada abstraksi *Token Controller Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.2. Kelas ini memiliki dua fungsi umum sama halnya dengan abstraksi *Token Controller Interface*.

4.1.3.1.2.1.1 Fungsi *Handle Token Request*

Fungsi pertama yang akan dijelaskan adalah *handleTokenRequest*. Fungsi ini memiliki tugas untuk menangani

request token yang berasal dari aplikasi klien. Aplikasi klien yang mengeksekusi token *endpoint* akan dilayani oleh fungsi ini. Penanganan pemberian token bagi aplikasi klien yang menggunakan mode token *ping* dan *poll* disatukan pada fungsi ini. Fungsi ini dapat dilihat pada Kode Sumber 4.23, Kode Sumber 4.24, Kode Sumber 4.25, dan Kode Sumber 4.26. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 5, *library* memeriksa kebenaran *request* token yang masuk. Pemeriksaan meliputi adanya parameter wajib seperti yang telah dijelaskan pada subbab 2.4.4. Jika tidak valid, *library* akan mengembalika nilai *false* dan pesan *error*.
2. Pada baris 6 sampai 7, *library* mengambil *grant type* dan *client id* yang digunakan oleh aplikasi klien.
3. Pada baris 9, *library* akan memeriksa *scope* yang diminta oleh aplikasi klien. *Scope* yang diminta oleh aplikasi klien harus terdaftar dan didukung oleh server autorisasi. Permintaan *scope* yang tidak valid adalah *scope* yang tidak terdaftar. Jika permintaan *scope* tidak valid, *library* akan mengembalikan nilai *false* dan pesan *error*.
4. Pada baris 10 sampai 11, *library* mengambil data aplikasi klien, dan *authentication request id*.
5. Pada baris 14 sampai 18, *library* memeriksa bahwa *authentication request id* masih valid dan belum kedaluwarsa. *Authentication request id* yang tidak valid adalah yang kedaluwarsa dan sudah digunakan sebelumnya.
6. Pada baris 20, *library* memeriksa metode token yang digunakan oleh aplikasi klien adalah *poll*. Pada baris 21 sampai 26, *library* akan menghitung jarak *request* token dari saat ini dengan waktu terakhir *request* token. Hitungan ini digunakan untuk mengetahui apakah *request* token terlalu cepat, mengingat bahwa mode token *poll* memiliki interval *polling*. Jika *interval* terlalu cepat, maka *library* akan mengembalikan nilai *false* dan pesan *error*.

7. Pada baris 28, *library* memperbarui nilai waktu terakhir *request* token dan disimpan di basis data.
8. Pada baris 31, *library* menaikkan batas waktu *request* menjadi lebih dari 30 detik sebagai upaya memulai *long polling*.
9. Pada baris 33, *library* memulai *long polling* dengan cara menggunakan *while loop*.
10. Pada baris 35 sampai 45, *library* memeriksa pemberian hak akses. Jika sudah melebihi 30 detik, maka *library* akan mengembalikan nilai *false* dan pesan *error*. Jika belum ada pemberian hak akses *library* akan *idle* selama 1 detik.
11. Pada baris 46 sampai 49, *library* memeriksa pemberian hak akses. Jika hak akses tidak diberikan maka *library* akan mengembalikan nilai *false* dan pesan *error*.
12. Pada baris 50 sampai 51, *library* memeriksa pemberian hak akses. Jika hak akses diberikan maka *library* akan keluar dari *long polling*.
13. Pada baris 56, *library* akan menyusun *access token*. Jika mode token yang digunakan adalah *ping* maka *library* tidak perlu melalui prosedur *long polling* yang telah dijelaskan pada langkah 9 sampai 12 sehingga langsung masuk ke langkah ini. Pada baris 57, *library* akan menyimpan *authentication request id* sebagai kedaluwarsa agar tidak dapat digunakan lagi.
14. Pada baris 59 sampai 67, *library* mengembalikan pesan *access token* dan nilai *true*.

```
1.  public function handleTokenRequest
2.  (RequestInterface $request,
3.  ResponseInterface $response)
4.  {
5.          if ($grantAccessToken = parent::grantAcc
    essToken($request, $response)) {
6.              $grantType = $grantAccessToken-
    >getGrantType();
7.              $clientId = $grantAccessToken-
    >getClientId();
8.
9.              if ($requestedScope = $this-
    >checkScope($request, $response, $grantType , $c
    lientId)) {
10.                 $client = $this->clientStorage-
    >getClientDetails($this-
    >authReqIdData['client_id']);
11.                 $authReq = $this->cibaStorage-
    >getAuthReqId($request-
    >request('auth_req_id'));
12.                 $now = time();
13.
14.                 if ($authReq['expires_in'] < $no
    w || !$authReq['valid']) {
15.                     $response-
    >setError(400, 'expired_token', "The auth_req_id
     has expired.");
16.
17.                     return false;
18.                 }
19.
20.                 if ($client['backchannel_token_d
    elivery_mode'] === self::POLL_METHOD) {
21.                     $reqInterval = $now - $authR
    eq['latest_token_requested_at'];
```

Kode Sumber 4.23 Fungsi *Handle Token Request* (1)

```
22.
23.                     if (($authReq['latest_token_
    requested_at']) && $reqInterval < $this-
    >configs['polling_interval']) {
24.                         $response-
    >setError(400, 'slow_down', 'Polling interval is
     too fast.');
25.
26.                         return false;
27.                     } else {
28.                         $this->cibaStorage-
    >setLastRequestedTokenAt($request-
    >request('auth_req_id'), $now);
29.                     }
30.
31.                     set_time_limit (32);
32.                     $start = microtime(true);

33.                     while (true)
34.                     {
35.                         $userConsent = $this-
    >cibaStorage->getAuthReqId($request-
    >request('auth_req_id'));
36.                         if ($userConsent['consen
    ted'] === null) {
37.                             $timeout = 30;
38.                             if (microtime(true)
    - $start >= $timeout) {
39.                                 $response-
    >setError(400, 'authorization_pending', 'The aut
    horization request is still pending as the end-
    user hasn\'t yet been authenticated.');
40.
41.                                 return false;

42.                             }
43.
```

Kode Sumber 4.24 Fungsi *Handle Token Request* (2)

```
44.                                 sleep(1);
45.                                 continue;
46.                             } else if ($userConsent[
    'consented'] === "0") {
47.                                 $response-
    >setError(403, 'access_denied', 'The end-
    user denied the authorization request.');
48.
49.                                 return false;
50.                             } else if ($userConsent[
    'consented'] === "1") {
51.                                 break;
52.                             }
53.                         }
54.                     }
55.
56.                     $token = parent::createAccessTok
    en($grantType, $clientId, $requestedScope);
57.                     $this->cibaStorage-
    >expireAuthReqId($request-
    >request('auth_req_id'));
58.
59.                     $response-
    >setStatusCode(200);
60.                     $response-
    >addParameters($token);
```

Kode Sumber 4.25 Fungsi *Handle Token Request* (3)

```
61.                 $response->addHttpHeaders(array(
62.                     'Cache-Control' => 'no-store',
63.                     'Pragma' => 'no-cache',
64.                     'Content-Type' => 'application/json'
65.                 ));
66.
67.                 return true;
68.             }
69.         }
70.
71.         return false;
72. }
```

Kode Sumber 4.26 Fungsi *Handle Token Request* (4)

4.1.3.1.2.1.2 Fungsi *Grant Access Token*

Fungsi kedua yang akan dijelaskan adalah *grantAccessToken*. Fungsi ini memiliki tugas untuk menentukan keputusan pemberian token yang diminta oleh aplikasi klien. Fungsi ini dapat dilihat pada Kode Sumber 4.27, Kode Sumber 4.28, Kode Sumber 4.29, dan Kode Sumber 4.30. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 5 sampai 9, *library* menambahkan *header allow* metode POST dan OPTIONS.
2. Pada baris 11 sampai 15, *library* memeriksa metode HTTP yang masuk. Jika menggunakan metode selain POST maka *library* akan mengembalikan pesan *error*.
3. Pada baris 18 sampai 21, *library* memeriksa parameter *grant type*. Jika tidak ditemukan, *library* akan mengembalikan pesan *error*.
4. Pada baris 24 sampai 27, *library* memeriksa apakah *grant type* yang digunakan aplikasi klien didukung

oleh server autorisasi. Jika tidak didukung oleh server autorisasi, *library* akan mengembalikan pesan *error*.

5. Pada baris 32 sampai 49, *library* melakukan validasi *request* token. Validasi meliputi pemeriksaan status daftar aplikasi klien dan status *authentication request id*. Jika aplikasi klien tidak terdaftar atau *authentication request id* kedaluwarsa, maka server autorisasi tidak akan memperboleh pengambilan token.
6. Pada baris 59, *library* mengembalikan pesan bahwa server autorisasi telah memutuskan untuk memperboleh pengambilan token.

```
1.  public function grantAccessToken
2.  (RequestInterface $request,
3.  ResponseInterface $response)
4.      {
5.          if (strtolower($request-
    >server('REQUEST_METHOD')) === 'options') {
6.              $response-
    >addHttpHeaders(array('Allow' => 'POST, OPTIONS'
    ));
7.
8.              return null;
9.          }
10.
```

Kode Sumber 4.27 Fungsi *Grant Access Token* (1)

```
11.         if (strtolower($request->server('REQUEST_METHOD')) !== 'post') {
12.             $response->setError(405, 'invalid_request', 'The request method must be POST when requesting an access token', '#section-3.2');
13.             $response->addHttpHeaders(array('Allow' => 'POST, OPTIONS'));
14.
15.             return null;
16.         }
17.
18.         if (!$grantTypeIdentifier = $request->request('grant_type')) {
19.             $response->setError(400, 'invalid_request', 'The grant type was not specified in the request');
20.
21.             return null;
22.         }
23.
24.         if (!isset($this->grantTypes[$grantTypeIdentifier])) {
25.             $response->setError(400, 'unsupported_grant_type', sprintf('Grant type "%s" not supported', $grantTypeIdentifier));
26.
27.             return null;
28.         }
29.
30.         $grantType = $this->grantTypes[$grantTypeIdentifier];
31.
```

Kode Sumber 4.28 Fungsi *Grant Access Token* (2)

```
32.         if (!$grantType instanceof ClientAssert
    ionTypeInterface) {
33.             if (!$this->clientAssertionType-
    >validateRequest($request, $response)) {
34.                 return null;
35.             }
36.             $clientId = $this-
    >clientAssertionType->getClientId();
37.         }
38.
39.         if (!$grantType-
    >validateRequest($request, $response)) {
40.             return null;
41.         }
42.
43.         if ($grantType instanceof ClientAsserti
    onTypeInterface) {
44.             $clientId = $grantType-
    >getClientId();
45.         } else {
46.             if (!is_null($storedClientId = $gra
    ntType-
    >getClientId()) && $storedClientId != $clientId
    ) {
47.                 $response-
    >setError(400, 'invalid_grant', sprintf('%s doe
    sn\'t exist or is invalid for the client', $gra
    ntTypeIdentifier));
48.
49.                 return null;
50.             }
51.         }
52.
```

Kode Sumber 4.29 Fungsi *Grant Access Token* (3)

```
53.         if (!$this->clientStorage-
    >checkRestrictedGrantType($clientId, $grantTypeId
    entifier)) {
54.             $response-
    >setError(400, 'unauthorized_client', 'The grant
    type is unauthorized for this client_id');
55.
56.             return false;
57.         }
58.
59.         return new GrantAccessTokenResponse($requ
    est, $response, $grantType, $clientId);
60.     }
```

Kode Sumber 4.30 Fungsi *Grant Access Token* (4)

4.1.3.1.2.2 Kelas *Ciba Grant Type*

Kelas *Ciba Grant Type* adalah kelas yang melapisi proses bisnis CIBA. Kelas *Ciba Grant Type* akan bergantung pada abstraksi *Ciba Grant Type Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.3. Kelas ini memiliki tujuh fungsi umum sama halnya dengan abstraksi *Ciba Grant Type Interface*.

4.1.3.1.2.2.1 Fungsi *Validate Request*

Fungsi pertama yang akan dijelaskan adalah *validateRequest*. Fungsi ini memiliki tugas untuk melakukan validasi yang menentukan pemberian *access token*. Fungsi ini dapat dilihat pada Kode Sumber 4.31, Kode Sumber 4.32, dan Kode Sumber 4.33. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 5 sampai 8, *library* memeriksa parameter *authentication request id*. Jika parameter tersebut tidak ada, maka *library* akan mengembalikan nilai *false* dan pesan *error*.

2. Pada baris 12 sampai 15, *library* mengambil data *authentication request id* dari basis data. Jika tidak ditemukan, maka *library* mengembalikan nilai *false* dan pesan *error*.
3. Pada baris 18 sampai 21, *library* mengambil data aplikasi klien. Jika tidak ditemukan, maka *library* akan mengembalikan nilai *false* dan pesan *error*.
4. Pada baris 24, *library* mengatur mode token yang digunakan oleh aplikasi klien. Pada baris 26 sampai 29, *library* akan mengembalikan pesan *error* jika mode token yang digunakan adalah *push.* Karena validasi token untuk mode token *push* tidak dilakukan pada fungsi ini.
5. Pada baris 32 sampai 35, *library* mengembalikan nilai *false* dan pesan *error* jika hak akses belum diberikan.
6. Pada baris 38 sampai 41, *library* mengembalikan nilai *false* dan pesan *error* jika hak akses tidak diberikan.
7. Pada baris 44 sampai 45, *library* mengembalikan pesan *error* jika tidak ada nilai kedaluwarsa pada *authentication request id*.
8. Pada baris 48 sampai 51, *library* mengembalikan nilai *false* dan pesan *error* jika *authentication request id* kedaluwarsa.
9. Pada baris 54 sampai 56, *library* mengembalikan nilai *false* dan pesan *error* jika *authentication request id* sudah pernah digunakan sebelumnya.
10. Pada baris 66, *library* mengembalikan nilai *true* sebagai tanda validasi berhasil.

```
1.  public function validateRequest
2.  (RequestInterface $request,
3.  ResponseInterface $response)
4.      {
5.          if (!$request-
    >request('auth_req_id')) {
6.              $response-
    >setError(400, 'invalid_request', 'Missing parame
    ter: "auth_req_id" is required');
7.
8.              return false;
9.          }
10.
11.         $auth_req_id = $request-
    >request('auth_req_id');
12.         if (!$authReqId = $this->storage-
    >getAuthReqId($auth_req_id)) {
13.             $response-
    >setError(400, 'invalid_grant', 'auth_req_id does
    n\'t exist or is invalid for the client');
14.
15.             return false;
16.         }
17.
18.         if (!$clientDetails = $this-
    >clientStorage-
    >getClientDetails($authReqId['client_id'])) {
19.             $response-
    >setError(400, 'invalid_client', 'client not exis
    t');
20.
21.             return false;
22.         }
23.
24.         $this-
    >setClientMode($clientDetails['backchannel_token_
    delivery_mode']);
```

Kode Sumber 4.31 Fungsi *Validate Request* (1)

```
25.
26.         if ($this-
    >getClientMode() === self::PUSH_METHOD) {
27.             $response-
    >setError(400, 'unauthorized_client', 'The Clien
    t is not authorized as it is configured in Push
    Mode');
28.
29.             return false;
30.         }
31.
32.         if ($authReqId['consented'] === null) {

33.             $response-
    >setError(403, 'authorization_pending', "The end
    -
    user hasn't consented the authorization request.
    ");
34.
35.             return false;
36.         }
37.
38.         if ($authReqId['consented'] === self::US
    ER_CONSENT_DENY_STATUS) {
39.             $response-
    >setError(403, 'access_denied', "The end-
    user denied the authorization request.");
40.
41.             return false;
42.         }
43.
44.         if (!isset($authReqId['expires_in'])) {

45.             throw new \Exception('Storage must r
    eturn auth_req_id with a value for "expires_in".
    ');
```

Kode Sumber 4.32 Fungsi *Validate Request* (2)

```
46.         }
47.
48.         if ($authReqId['expires_in'] < time()) {
49.             $response->setError(403, 'expired_token', "The auth_req_id has expired.");
50.
51.             return false;
52.         }
53.
54.         if ($authReqId['valid'] === "0") {
55.             $response->setError(403, 'expired_token', "The auth_req_id has expired.");
56.
57.             return false;
58.         }
59.
60.         if (!isset($authReqId['auth_req_id'])) {
61.             $authReqId['auth_req_id'] = $auth_req_id;
62.         }
63.
64.         $this->authReqId = $authReqId;
65.
66.         return true;
67.     }
```

Kode Sumber 4.33 Fungsi *Validate Request* (3)

4.1.3.1.2.2.2 Fungsi *Get Client Mode*

Fungsi kedua yang akan dijelaskan adalah *getClientMode*. Fungsi ini memiliki tugas untuk mendapatkan mode token yang digunakan oleh aplikasi klien. Fungsi ini dapat dilihat pada Kode Sumber 4.34. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengembalikan mode token yang digunakan oleh aplikasi klien.

```
1.  public function getClientMode()
2.  {
3.    return isset($this->clientMode) ? $this-
    >clientMode : null;
4.  }
```

Kode Sumber 4.34 Fungsi *Get Client Mode*

4.1.3.1.2.2.3 Fungsi *Get Query String Identifier*

Fungsi ketiga yang akan dijelaskan adalah *getQueryStringIdentifier*. Fungsi ini memiliki tugas untuk mendapatkan format identifikasi CIBA. Tujuannya adalah untuk membedekan kelas *grant type* CIBA dengan kelas *grant type* yang lain seperti *Authorization Code*. Fungsi ini dapat dilihat pada Kode Sumber 4.34. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* akan mengembalikan *query string identifier*.

```
1.  public function getQueryStringIdentifier()
2.  {
3.    return 'urn:openid:params:grant-type:ciba';
4.  }
```

Kode Sumber 4.35 Fungsi *Get Query String Identifier*

4.1.3.1.2.2.4 Fungsi *Get Client Id*

Fungsi keempat yang akan dijelaskan adalah *getClientId*. Fungsi ini memiliki tugas untuk mendapatkan identifikasi aplikasi klien. Fungsi ini dapat dilihat pada Kode Sumber 4.36. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengembalikan *client id*.

```
1.  public function getClientId()
2.  {
3.     return $this->authReqId['client_id'];
4.  }
```

Kode Sumber 4.36 Fungsi *Get Client Id*

4.1.3.1.2.2.5 Fungsi *Get Scope*

Fungsi kelima yang akan dijelaskan adalah *getScope*. Fungsi ini memiliki tugas untuk mendapatkan *scope* yang diminta oleh aplikasi klien. Fungsi ini dapat dilihat pada Kode Sumber 4.37. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengembalikan *scope*.

```
1.  public function getScope()
2.  {
3.     return isset($this->authReqId['scope']) ? $this-
    >authReqId['scope'] : null;
4.  }
```

Kode Sumber 4.37 Fungsi *Get Scope*

4.1.3.1.2.2.6 Fungsi *Get User Id*

Fungsi keenam yang akan dijelaskan adalah *getUserId*. Fungsi ini memiliki tugas untuk mendapatkan identifikasi akun pengguna. Fungsi ini dapat dilihat pada Kode Sumber 4.38. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* mengembalikan *user id*.

```
1.  public function getUserId()
2.  {
3.     return isset($this->authReqId['user_id']) ? $this-
    >authReqId['user_id'] : null;
4.  }
```

Kode Sumber 4.38 Fungsi *Get User Id*

4.1.3.1.2.2.7 Fungsi *Create Access Token*

Fungsi ketujuh yang akan dijelaskan adalah *createAccessToken*. Fungsi ini memiliki tugas untuk menyusun *access token*. Fungsi ini dapat dilihat pada Kode Sumber 4.39. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 5, *library* menyusun *access token*.
2. Pada baris 7 sampai 10, *library* menyisipkan *ID Token* dan menyimpan *authentication request id* sebagai kedaluwarsa jika mode token adalah *ping* atau *poll*.
3. Pada baris 13, *library* mengembalikan *access token* sebagai tanda berhasil.

```
1.  public function createAccessToken
2.  (AccessTokenInterface $accessToken, $client_id,
3.  $user_id, $scope)
4.  {
5.      $token = $accessToken-
    >createAccessToken($client_id, $user_id, $scope);
6.
7.      if ($this->getClientMode() !== null && $this-
    >getClientMode() !== 'push') {
8.          $authReqIdData = $this->storage->getAuthReqId($this-
    >authReqId['auth_req_id']);
9.          $token['id_token'] = $authReqIdData['id_token'];
10.         $this->storage->expireAuthReqId($this-
    >authReqId['auth_req_id']);
11.     }
12.
13.     return $token;
14. }
```

Kode Sumber 4.39 Fungsi *Create Access Token*

4.1.3.1.2.3 Kelas *Curl Relying Party Notification Transport*

Kelas *Curl Relying Party Notification Transport* adalah kelas yang bertanggung jawab untuk mengirim pesan ke aplikasi klien. Kelas *Curl Relying Party Notification Transport* akan

bergantung pada abstraksi *Notification Transport Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.7. Kelas ini memiliki satu fungsi umum sama halnya dengan abstraksi *Notification Transport Interface*.

4.1.3.1.2.3.1 Fungsi *Send*

Fungsi pertama yang akan dijelaskan adalah *send*. Fungsi ini memiliki tugas untuk mengirim pesan notifikasi ke aplikasi klien bahwa aplikasi klien dapat melakukan pengambilan token. Fungsi ini dapat dilihat pada Kode Sumber 4.40. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyusun *header authorization* yang memiliki nilai *client notification token*.
2. Pada baris 7, *library* menginisialisasi obyek *curl*. *Library* dapat mengetahui rute tujuan aplikasi klien karena aplikasi klien telah mendaftarkan *client notification endpoint* yang direpresentasikan dengan variable *$to*.
3. Pada baris 9 sampai 16, *library* menyusun data *body*.
4. Pada baris 19 sampai 25, *library* melakukan konfigurasi *curl*.
5. Pada baris 27, *library* mengeksekusi *curl*.
6. Pada baris 29 sampai 33, *library* mengembalikan nilai *data* dari aplikasi klien jika berhasil, dan nilai *null* jika gagal.

```
1.  public function send($payload, $to)
2.    {
3.      $headers = array(
4.        'Authorization: Bearer ' . $payload['client_notification_t
    oken']
5.      );
6.
7.      $ch = curl_init($to . '?XDEBUG_SESSION_START=XDEBUG_
    ECLIPSE');
8.
9.      if ($payload['post_body'] != null) {
10.
11.        $post_body = json_encode($payload['post_body']);
12.        curl_setopt($ch, CURLOPT_POST, 1);
13.        curl_setopt($ch, CURLOPT_POSTFIELDS, $post_body);

14.
15.        $headers[] = "Content-Type: application/json";
16.        $headers[] = 'Accept: application/json';
17.      }
18.
19.      header(sprintf('HTTP/%s %s %s', 2, $payload['status_co
    de'], $this->statusTexts[$payload['status_code']]));
20.      curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
21.      curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
22.      curl_setopt($ch, CURLOPT_HEADER, 0);
23.      curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
24.      curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
25.      curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
26.
27.      $output = curl_exec($ch);
28.
29.      if ($output === false) {
30.        return null;
31.      }
32.      curl_close($ch);
33.      return $output;
34.  }
```

Kode Sumber 4.40 Fungsi *Send*

4.1.3.1.2.4 Kelas Utama *Server*

Pada penerapan pengambilan *access token*, kelas *Server* menyediakan dua fungsi sebagai berikut.

4.1.3.1.2.4.1 Fungsi *Handle Token Request*

Fungsi pertama yang akan dijelaskan adalah *handleTokenRequest*. Fungsi ini digunakan untuk melayani pengambilan *access token* untuk alur selain CIBA. Fungsi ini dapat dilihat pada Kode Sumber 4.41. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyimpan obyek *response* yang digunakan untuk mengembalikan pesak ke klien.
2. Pada baris 4, *library* mengambil obyek *Token Controller* dan memanggil fungsi *handleTokenRequest*. Kelas *Token Controller* telah dijelaskan pada subbab 3.2.3.1.2.2.
3. Pada baris 6, *library* mengembalikan obyek *response*.

```
1. public function handleTokenRequest(RequestInterface $req
   uest, ResponseInterface $response = null)
2. {
3.    $this-
   >response = is_null($response) ? new Response() : $response
   ;
4.    $this->getTokenController()-
   >handleTokenRequest($request, $this->response);
5.
6.    return $this->response;
7. }
```

Kode Sumber 4.41 Fungsi *Handle Token Request*

4.1.3.1.2.4.2 Fungsi *Handle Ciba Token Request*

Fungsi kedua yang akan dijelaskan adalah *handleCibaTokenRequest*. Fungsi ini digunakan untuk melayani pengambilan *access token* untuk alur CIBA. Fungsi ini dapat

dilihat pada Kode Sumber 4.42. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyimpan obyek *response* yang digunakan untuk mengembalikan pesak ke klien.
2. Pada baris 4, *library* mengambil obyek *Ciba Token Controller* dan memanggil fungsi *handleTokenRequest*. Kelas *Ciba Token Controller* telah dijelaskan pada subbab 4.1.3.1.2.1.
3. Pada baris 6, *library* mengembalikan obyek *response*.

```
1. public function handleCibaTokenRequest(RequestInterface $
   request, ResponseInterface $response = null)
2. {
3.   $this-
   >response = is_null($response) ? new Response() : $response;

4.   $this->getCibaTokenController()-
   >handleTokenRequest($request, $this->response);
5.
6.   return $this->response;
7. }
```

Kode Sumber 4.42 Fungsi *Handle Ciba Token Request*

4.1.3.1.3 Penerapan Session Binding

Penggunaan *session binding* pada CIBA sangat penting agar pengguna pada *Authentication Device* dapat membedakan permintaan hak akses yang dapat terjadi berulang-ulang atau untuk membedakan hak akses dari aplikasi klien yang valid. Pada tugas akhir ini penerapan *session binding* melibatkan dua kelas. Kelas pertama adalah *Ciba Controller* yang telah dijelaskan pada subbab 4.1.3.1.1.1.1. Kelas kedua adalah *Firebase Request Consent Notification Transport* yang akan dijelaskan sebagai berikut.

4.1.3.1.3.1 Kelas *Firebase Request Consent Notification Transport*

Kelas *Firebase Request Consent Notification Transport* adalah kelas yang memiliki tanggung jawab untuk berkomunikasi dengan *Authentication Device* menggunakan *Firebase Cloud Messaging*. Kelas *Firebase Request Consent Notification Transport* akan bergantung pada abstraksi *Notification Transport Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.7. Kelas ini memiliki satu fungsi umum sama halnya dengan abstraksi *Notification Transport Interface*. Kelas ini berbeda dengan kelas *Curl Relying Party Notification Transport* yang dijelaskan pada subbab 4.1.3.1.2.3 karena kelas ini digunakan untuk mengirim pesan ke *Authentication Device* menggunakan *Firebase Cloud Messaging*, sedangkan kelas *Curl Relying Party Notification Transport* digunakan untuk mengirim pesan ke *aplikasi klien*.

4.1.3.1.3.1.1 Fungsi *Send*

Fungsi pertama yang akan dijelaskan adalah *send*. Fungsi ini memiliki tugas untuk mengirim pesan ke *Authentication Device* menggunakan *Firebase Cloud Messaging*. Pada permasalahan *session binding*, fungsi ini dapat mengirim kode atau *binding message* yang disusun oleh aplikasi klien, dan disimpan oleh server autorisasi, lalu diteruskan oleh fungsi ini untuk dikirim ke *Authentication Device*. Sehingga yang terjadi adalah pengguna pada *Authentication Device* dapat mencocokkan *kode* yang tampil pada perangkatnya dengan kode yang terdapat pada *Consumption Device*. Meskipun perangkatnya terpisah, pengguna *Consumption Device* dapat mengkonfirmasi ke pengguna *Authentication Device* bahwa kode yang muncul memiliki format yang seharusnya. Jika kode yang muncul pada *Authentication Device* berbeda, maka itu merupakan *session* yang berbeda sehingga pengguna pada *Authentication Device* dapat menolak hak akses. Fungsi ini dapat dilihat pada Kode Sumber 4.43. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 8, *library* menyusun parameter yang digunakan sebagai rute tujuan pesan. Rute ini diidentifikasi sebagai topik pada *Firebase*. Topik ini memiliki format sebagai "*/topics/ciba_[user_id_pengguna]*" sehingga hanya terkirim ke pengguna yang dipilih.
2. Pada baris 10, *library* menginisialisasi *curl* menggunakan alamat *Firebase Cloud Messaging*.
3. Pada baris 11 sampai 20, *library* melakukan konfigurasi *curl*.
4. Pada baris 21, *library* mengeksekusi *curl*.
5. Pada baris 22, *library* mengakhiri obyek *curl*.

```
1.  public function send($payload, $to)
2.  {
3.      if (! is_array($to)) {
4.          $to = ['to' => $to];
5.      }
6.
7.      $payload = array_merge($to, $payload);
8.      $jsonPayload = json_encode($payload);
9.
10.     $ch = curl_init('https://fcm.googleapis.com/fcm/send');
11.     curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
12.     curl_setopt($ch, CURLINFO_HEADER_OUT, true);
13.     curl_setopt($ch, CURLOPT_POST, true);
14.     curl_setopt($ch, CURLOPT_POSTFIELDS, $jsonPayload);
15.
16.     curl_setopt($ch, CURLOPT_HTTPHEADER, array(
17.         'Content-Type: application/json',
18.         'Content-Length: ' . strlen($jsonPayload),
19.         'Authorization: ' . "key={$this->options['key']}")
20.     );
21.     $result = curl_exec($ch);
22.     curl_close($ch);
23. }
```

Kode Sumber 4.43 Fungsi *Send*

4.1.4 Implementasi Aplikasi Server Autorisasi CIBA

Dalam subbab ini dijelaskan implementasi kelas pada aplikasi server autorisasi CIBA.

4.1.4.1 Implementasi Kelas

Dalam subbab ini dijelaskan implementasi kelas yang berasal dari perancangan kelas yang telah didapatkan pada subbab 3.2.3.2. Kelas pada aplikasi server autorisasi bentuknya adalah *controller* sehingga fungsi-fungsi yang terdapat pada *controller* akan memanggil fungsi-fungsi yang terdapat pada *library* server autorisasi yang telah dijelaskan pada subbab 4.1.3.

Kerangka kerja Phalcon memiliki beberapa konvensi yang digunakan pada fase pengembangan.

1. Konvensi pertama adalah penamaan fungsi *controller*. Nama fungsi harus disertakan dengan kata *action* agar Phalcon dapat mengetahui fungsi yang dituju berupa fungsi *controller*. Sebagai contoh *handleAuthenticationAction*, *handleLogoutAction*, dan *handleVerifyUserAction*.
2. Konvensi kedua adalah mendaftarkan direktori kelas berserta *namespace* yang digunakan. Tujuannya adalah agar kelas yang digunakan dapat diketahui oleh modul *autoloader* Phalcon. Fungsi ini dapat dilihat pada Kode Sumber 4.44.
3. Konvensi ketiga adalah mendaftarkan rute URI *controller*. Tujuannya adalah agar klien dapat mengakses fungsi yang disediakan oleh Phalcon melalui URI. Dapat dilihat pada Kode Sumber 4.45
4. Konvensi keempat adalah mendaftarkan obyek pada *service container*. Gunanya adalah agar pembuatan dan penghapusan obyek ditangani oleh *service container* sehingga bisa memanfaatkan konsep seperti *dependency injection*. Dapat dilihat pada Kode Sumber 4.46, Kode Sumber 4.47, dan Kode Sumber 4.48 mendaftarkan *library* server autorisasi CIBA pada *service container*. Pada kode

sumber tersebut, baris 1 memberikan nama obyek sebagai "*oidc".* Pada baris 4 sampai 13, dilakukan konfigurasi *library* seperti konfigurasi pendukung *openid connect,* alamat URI autorisasi server, dan konfigurasi pendukung *frontchannel/ backchannel logout*. Pada baris 16 sampai 31, dilakukan pendaftaran *repository* yang digunakan oleh *library*. Salah satu *repository* yang digunakan adalah *Ciba Repository* yang akan dijelaskan pada subbab 4.1.4.1.3.3. Pada baris 33 sampai 35 dilakukan konfigurasi *grant type* yang didukung oleh *library,* yaitu *Client Credentials, Authorization Code,* dan CIBA. Pada baris 37 sampai 40, dibangun obyek *Server* dengan menambahkan *grant type* yang telah dikonfigurasi. Pada baris 44 sampai 45, dilakukan konfigurasi obyek *Firebase Notification Transport* yang digunakan untuk mengirim pesan ke *Authentication Device*. Pada baris 47, *service container* mengembalikan obyek *Server* jika ada kelas yang membutuhkan obyek *Server*.

```
1.  public function registerAutoloaders(DiInterface $di = null)
2.  {
3.      $loader = new Loader();
4.
5.      $loader->registerNamespaces([
6.          'MyIts\Oidc\Controllers' => __DIR__ . '/controllers',
7.          'MyIts\Oidc\Controllers\Api' => __DIR__ . '/controllers/Ap
    i',
8.          'MyIts\Oidc\Repositories' => __DIR__ . '/repositories',
9.          'MyIts\Oidc\Models' => __DIR__ . '/models',
10.         'MyIts\Oidc\Domain\Events' => __DIR__ . '/domain/event
    s',
11.         'MyIts\Oidc\UseCases' => __DIR__ . '/usecases',
12.         'MyIts\Oidc\Exceptions' => __DIR__ . '/exceptions'
13.     ]);
14.
15.     $loader->register();
16. }
```

Kode Sumber 4.44 Fungsi Pendaftaran Direktori Kelas *Controller*

```
1.  $router->addPost('/bc-authorize', [
2.      'namespace' => 'MyIts\Oidc\Controllers',
3.      'module' => 'oidc',
4.      'controller' => 'Ciba',
5.      'action' => 'authenticate'
6.  ]);
7.
8.  $router->addPost('/ciba-consent', [
9.      'namespace' => 'MyIts\Oidc\Controllers',
10.     'module' => 'oidc',
11.     'controller' => 'cibaconsent',
12.     'action' => 'handle'
13. ]);
```

Kode Sumber 4.45 Fungsi Mendaftarkan Rute URI *Controller*

```
1.  $di->setShared('oidc', function () use ($di) {
2.      $config = $di->get('config');
3.
4.      $configParams['use_openid_connect'] = true;

5.      $configParams['issuer'] = $config->oidc-
    >url;
6.      $configParams['allow_implicit'] = true;
7.      $configParams['allow_public_clients'] = fals
    e;
8.      $configParams['allow_session_management'] =
    true;
9.      $configParams['frontchannel_logout_supported
    '] = true;
10.     $configParams['frontchannel_logout_session_s
    uppoorted'] = true;
11.     $configParams['backchannel_logout_supported'
    ] = true;
12.     $configParams['backchannel_logout_session_su
    ppoorted'] = true;
13.     $configParams['jwt_extra_payload_direct_impe
    rsonation_callable'] = 'createImpersonateExtraPa
    yload';
```

Kode Sumber 4.46 Fungsi Mendaftarkan Obyek Pada *Service Container* (1)

```
14.
15.
16.     $storage = array(
17.         'access_token' => new RedisAccessTokenRe
    pository(),
18.         'authorization_code' => new RedisAuthori
    zationCodeRepository(),
19.         'client_credentials' => new ClientCreden
    tialsRepository(),
20.         'client' => new ClientRepository(),
21.         'refresh_token' => new RedisRefreshToken
    Repository(),
22.         'user_credentials' => new UserCredential
    sRepository(),
23.         'user_claims' => new UserClaimsRepositor
    y(),
24.         'public_key' => new PublicKeyRepository(
    ),
25.         'jwt_bearer' => new JwtBearerRepository(
    ),
26.         'scope' => new ScopeRepository(),
27.         'session_state' => new SessionStateRepos
    itory(),
28.         'session' => new RedisSessionRepository(
    ),
29.         'jwk' => new JwkRepository(),
30.         'client_initiated_backchannel_authentica
    tion' => new CibaRepository()
31.     );
32.
33.     $clientCredentialsGrantType = new ClientCred
    entials($storage['client_credentials']);
34.     $oidcAuthCodeGrantType = new AuthorizationCo
    de($storage['authorization_code']);
```

Kode Sumber 4.47 Fungsi Mendaftarkan Obyek Pada *Service Container* (2)

```
35.     $cibaGrantType = new OpenIDCibaGrantType($st
    orage['client_initiated_backchannel_authenticati
    on'], $storage['client']);
36.
37.     $server = new OAuth2\Server($storage, $confi
    gParams);
38.     $server-
    >addGrantType($clientCredentialsGrantType);
39.     $server-
    >addGrantType($oidcAuthCodeGrantType);
40.     $server->addGrantType($cibaGrantType);
41.
42.     $firebaseCloudMessageKey = 'suatu-kode-
    rahasia-dari-firebase-cloud-messaging;
43.
44.     $firebaseNotificationTransport = new Firebas
    eRequestConsentNotificationTransport(['key' => $
    firebaseCloudMessageKey]);
45.     $server-
    >addNotificationTransport($firebaseNotificationT
    ransport);
46.
47.     return $server;
48. });
```

Kode Sumber 4.48 Fungsi Mendaftarkan Obyek Pada *Service Container* (3)

Implementasi kelas yang dihasilkan akan dikelompokkan sesuai dengan rumusan permasalahan. Pada implementasi aplikasi server autorisasi, rumusan masalah "Bagaimana mengimplementasi prosedur pendaftaran aplikasi klien agar dapat menggunakan alur CIBA" akan dibahas pada subbab 4.1.4.1.1, "Bagaimana mengimplementasi alur CIBA pada server autorisasi myITS Single Sign-On" dan "Bagaimana mengimplementasi *session binding* antara CD dan AD" akan dibahas pada subbab 4.1.4.1.2, dan "Bagaimana cara mengimplementasi pengambilan *access token* menggunakan mode *poll*, *ping*, dan *push*" akan dibahas pada subbab 4.1.4.1.2.3.

4.1.4.1.1 Penerapan Prosedur Pendaftaran Aplikasi Klien

Prosedur pendaftaran aplikasi klien pada protokol CIBA telah dibebaskan kepada *developer* sistem untuk menerapkan sesuai kebutuhan server autorisasi. Pada server autorisasi myITS SSO, pendaftaran aplikasi dapat dilakukan melalui sistem Security Manager yang digunakan oleh DPTSI. Namun, sistem Security Manager bertindak sebagai aplikasi klien, oleh karena itu penjelasan pendaftaran aplikasi klien menggunakan Security Manager diluar cakupan tugas akhir ini.

Tetapi sebagai panduan yang dapat digunakan untuk menerapkan pendaftaran aplikasi klien, subbab ini akan menjelaskan struktur basis data sebagai pendekatan implementasi pendaftaran aplikasi klien.

Mengacu pada subbab 3.2.4.1 yang menjelaskan tabel *OAuth Client* pada basis data relasional, *developer* dapat memanfaatkan *metadata* CIBA sebagai berikut. Dapat dilihat pada Tabel 4.1, terdapat aplikasi klien dengan nama Security Manager. Aplikasi ini telah terdaftar menggunakan alur CIBA yang memiliki identifikasi *client id* dan *client secret*. Server autorisasi CIBA pada myITS SSO mendukung tiga mode token yaitu *push, ping* dan *poll*. Oleh karena itu aplikasi ini dapat memilih salah satu diantara tiga mode, dan terpilih mode token *ping*. Dikarenakan aplikasi ini menggunakan mode token *ping*, maka aplikasi harus menyediakan *client notification endpoint* agar server autorisasi dapat mengirim pesan notifikasi ke aplikasi klien. Aplikasi ini telah memiliki *client notification endpoint* dengan nilai URI. Server autorisasi CIBA pada myITS SSO telah mendukung *signed request*, maka aplikasi klien dapat menggunakan *signed request* yang terdaftar menggunakan algoritma RS256. Agar *signed request* dapat diverifikasi oleh server autorisasi, maka aplikasi klien harus menyediakan *public key*. Server autorisasi dapat meminta *public key* melalui *metadata public key URI* yang telah terdaftar. Aplikasi ini tidak menggunakan *user code* sehingga didapatkan nilai *false* pada metadata *backchannel user code parameter*.

Tabel 4.1 Penerapan Pendaftaran Aplikasi Klien

Nama Metadata	Nilai
Client Id	b8b434e3-2d83-464c-8a6e-eb5983970b28
Client Name	Security Manager
Client Secret	271f8cb9-71d5-4812-b81f-b939b099f87a
Backchannel token delivery mode	*Ping*
Backchannel client notification endpoint	https://dev-my.its.ac.id/secman/notification
Backchannel Authentication request signing alg	RS256
Backchannel user code parameter	*false*
Public key URL	https://dev-my.its.ac.id/secman/ciba/public-key

4.1.4.1.2 Penerapan Alur CIBA dan *Session Binding*

Peneran alur CIBA dan *session binding* bekerja sama dikarenakan langkah kerja *session binding* berawal dari *request* autentikasi *backchannel* yang disertakan dengan *binding message*.

4.1.4.1.2.1 Kelas Ciba Controller

Kelas *Ciba Controller* digunakan untuk melayani aplikasi klien yang ingin melakukan autentikasi *backchannel*. *Request* HTTP yang masuk pada *controller* ini akan diteruskan ke *library* server autorisasi CIBA untuk menangani logika yang lebih spesifik. Obyek library server autorisasi CIBA merupakan hasil instansiasi kelas *Server* yang telah dijelaskan pada subbab 4.1.3.1.1.4.4.

4.1.4.1.2.1.1 Fungsi *Handle Authentication Action*

Kelas *Ciba Controller* hanya memiliki satu fungsi yaitu *handleAuthenticationAction*. *Request* yang masuk pada fungsi ini adalah sesuai peraturan yang telah dijelaskan pada subbab 0. Fungsi ini dapat dilihat pada Kode Sumber 4.49. Langkah kerja fungsi ini adalah sebagai berikut.

1. Pada baris 3 sampai 4, *controller* akan membuat obyek *request* dan *response*. Obyek *request* digunakan untuk menyimpan *header* dan parameter *body* pada *request* HTTP. Parameter *body* pada *request* telah dijelaskan pada subbab 2.4.2.1 dan 2.4.2.2 sehingga *library* server autorisasi CIBA dapat mengidentifikasi bahwa *request* ini adalah permintaan autentikasi menggunakan CIBA. Obyek *response* digunakan untuk mengembalikan pesan dari *library* server autorisasi CIBA.
2. Pada baris 6, *controller* memanggil fungsi *handleCibaRequest* pada obyek *Server*, dan disertakan dengan parameter *request* dan *response*. Obyek ini sudah terdaftar pada *service container*.
3. Pada baris 7, *controller* akan mengembalikan pesan yang didapatkan dari library server autorisasi CIBA.

```
1.  public function handleAuthenticationAction()
2.   {
3.          $request = Request::createFromGlobals();
4.          $response = new \OAuth2\Response();
5.
6.          $this->serverCiba-
    >handleCibaRequest($request, $response);
7.          $response->send();
8.
9.          return;
10.  }
```

Kode Sumber 4.49 Fungsi *Handle Authentication Request Action*

Fungsi ini dapat diakses melalui rute URI yang terdefinisi pada Kode Sumber 4.50 dengan parameter yang telah dijelaskan pada subbab 2.4.2.1 dan 2.4.2.2.

```
POST https://dev-my.its.ac.id/bc-authorize
```

Kode Sumber 4.50 URI *Backchannel Authentication*

4.1.4.1.2.2 Kelas *Ciba Consent Controller*

Kelas *Ciba Consent Controller* digunakan untuk melayani pemberian hak akses oleh pengguna *Authentication Device*. Kelas ini hanya memiliki satu fungsi yaitu *handleConsentAction*.

4.1.4.1.2.2.1 Fungsi *Handle Consent Action*

Fungsi *handleConsentAction* memiliki tugas untuk melayani pemberian hak akses oleh pengguna *Authentication Device*. Hak akses yang diberikan dapat berupa penolakan atau pembolehan. Fungsi ini dapat dilihat pada Kode Sumber 4.51. Langkah kerja fungsi ini adalah sebagai berikut.

1. Pada baris 3 sampai 4, *controller* akan membuat obyek *request* dan *response*. Obyek *request* digunakan untuk menyimpan *header* dan parameter *body* pada *request* HTTP. Prosedur *request* HTTP untuk pemberian hak

akses tidak memiliki standar pada spesifikasi CIBA, oleh karena itu didapatkan implementasi yang dapat dilihat pada Tabel 4.2.

2. Pada baris 6, *controller* memanggil fungsi *handleCibaUserConsent* pada obyek *Server*, dan disertakan dengan parameter *request* dan *response*. Obyek ini sudah terdaftar pada *service container*.
3. Pada baris 7, *controller* akan mengembalikan pesan yang didapatkan dari library server autorisasi CIBA.

Tabel 4.2 Parameter HTTP Pemberian Hak Akses

Nama Parameter	Deskripsi	Wajib
auth_req_id	*Authentication request id* yang berkaitan dengan sesi CIBA ini.	YA
user_consent	Hak akses yang diberikan oleh pengguna. Nilai dapat berupa "*accept*" yang memiliki arti pengguna mengizinkan aplikasi klien dan "*deny*" yang memiliki arti pengguna tidak mengizinkan aplikasi klien.	YA

```
1.  public function handleConsentAction()
2.  {
3.      $request = Request::createFromGlobals();
4.      $response = new \OAuth2\Response();
5.
6.      $this->serverCiba-
    >handleCibaUserConsent($request, $response);
7.      $response->send();
8.      return;
9.  }
```

Kode Sumber 4.51 Fungsi *Handle Consent Action*

Fungsi ini dapat diakses melalui rute URI yang terdefinisi pada Kode Sumber 4.52 dengan parameter yang telah dijekaskan pada Tabel 4.2.

```
POST https://dev-my.its.ac.id/ciba-consent
```

Kode Sumber 4.52 URI Pemberian Hak Akses

4.1.4.1.2.3 Kelas *Authorize Controller*

Kelas *Authorize Controller* digunakan untuk menangani hak akses aplikasi klien dan pengguna pada saat menggunakan alur *Authorization Code*. Kelas *controller* ini memiliki satu fungsi yang dinamakan dengan *handleAuthorizeAction*. Fungsi ini digunakan saat ada pengguna yang melakukan kasus penggunaan *impersonation*.

4.1.4.1.2.3.1 Fungsi *Handle Authorize Action*

Fungsi ini memanfaatkan obyek *Server* yang telah dijelaskan pada subbab 4.1.3.1.1.5. Fungsi ini dapat dilihat pada Pseudocode 4.1. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 8, terjadi pembuatan obyek *request* yang mengandung *parameter* dari klien, *response* yang digunakan untuk pengembalikan pesan ke klien, *server* sebagai obyek dari *library* server

autorisasi CIBA, *session* sebagai status autentikasi pengguna, dan *loginUrl/ loginAsUrl* yang menyimpan nilai URI formulir *login*.

2. Pada baris 10, dilakukan validasi oleh *library*. Pada baris 12, jika validasi menghasilkan *error*, maka pesan *error* dikembalikan ke klien.
3. Pada baris 17, dilakukan pengambilan *query parameter* yang terdapat pada URI.
4. Pada baris 19, jika pengguna belum terautentikasi, maka dialihkan ke halaman *login*.
5. Pada baris 23, dilakukan pemeriksaan sesi autentikasi pengguna. Jika pengguna telah terautentikasi dan terindikasi bahwa ada *parameter* "*impersonated_sub*" maka akan dialihkan ke halaman *login* khusus *impersonation*.
6. Pada baris 27, jika pengguna telah terautentikasi maka akan memanggil fungsi *handleAuthorizeRequest* pada obyek *Server*. Lalu klien mendapatkan pesan hasil dari pemanggilan fungsi tersebut.

```
1.  function handleAuthorizeAction()
2.    {
3.      request = GET request object
4.      response = CREATE response object
5.      server = GET server object
6.      session = GET session object
7.      loginUrl = GET loginUrl object
8.      loginAsUrl = GET loginAsUrl object
9.
10.     result = server.validateAuthorizeRequest(request, respon
    se)
11.
12.     if result == false
13.       response.send()
14.       return false
15.     endif
16.
17.     requestParameters = request.geAllQueryParameters();
18.
19.     if session.Authenticated() == false
20.       return response.redirect(loginUrl)
21.     endif
22.
23.     if session.Authenticated() == true AND requestParameter
    s['impersonated_sub'] AND session.impersonatingSession() =
    = false
24.       return response.redirect(loginAsUrl)
25.     endif
26.
27.     if session.impersonatingSession() == true || session.Authe
    nticated()
28.       server.handleAuthorizeRequest(request, response)
29.       response.send()
30.     endif
31.
32.   }
```

Pseudocode 4.1 Fungsi *Handle Authorize Action*

Fungsi ini dapat diakses melalui URI yang dapat dilihat pada Kode Sumber 4.53 dengan parameter yang dapat dilihat pada Tabel 4.3.

```
POST https://dev-my.its.ac.id/authorize
```

Kode Sumber 4.53 URI *Authorize*

Tabel 4.3 Parameter Autorisasi

Nama Parameter	Deskripsi
scope	Hak akses yang akan dimiliki oleh RP
response_type	Menentukan alur autorisasi yang digunakan. Diisi dengan nilai "*code*".
client_id	Identifikasi aplikasi klien.
redirect_uri	URI yang digunakan ketika server autorisasi mengalihkan kembali ke halaman aplikasi klien.
impersonated_sub	Identifikasi akun pengguna yang ingin diperan.

4.1.4.1.2.4 Kelas *Login Controller*

Kelas *Login Controller* digunakan untuk menangani autentikasi pengguna menggunakan alur *Authorization Code*. Kelas ini memiliki fungsi yang dinamakan *handleLoginAction*. Fungsi ini digunakan saat ada pengguna yang melakukan Kasus Penggunaan Autentikasi Berperan Sebagai Akun Yang Berbeda (UC-004).

4.1.4.1.2.4.1 Fungsi *Handle Login Action*

Fungsi ini memiliki tugas untuk melakukan autentikasi terhadap data pengguna yang masuk yaitu dengan cara memastikan bahwa data pengguna yang diberikan oleh klien cocok dengan

yang tersimpan pada basis data. Selain itu, memiliki tugas untuk menampilkan halaman *login*. Fungsi ini dapat dilihat pada Pseudocode 4.2. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 7, terjadi pembuatan obyek *request* yang mengandung *parameter* dari klien, *response* yang digunakan untuk pengembalikan pesan ke klien, *loginUrl* yang menyimpan nilai URI formulir *login*, dan *view* sebagai obyek yang menyimpan antarmuka HTML.
2. Pada baris 9, *library* memeriksa metode *request* yang masuk. Apabila metodenya adalah *POST* maka ini *request* yang berasal dari klien dengan data pengguna.
3. Pada baris 10 sampai 11, dilakukan pengambilan data *request* dan dekripsi konten. Hasil dekripsi berupa struktur data dengan nilai *email/ username,* dan *password*.
4. Pada baris 13, dilakukan autentikasi dengan data pengguna yang didekripsi.
5. Pada baris 15, apabila proses autentikasi berhasil, maka klien akan dialihkan ke *dashboard.* Jika gagal karena data pengguna salah, maka dialihkan ke halaman *login* untuk melakukan autentikasi ulang.
6. Pada baris 21 sampai 27, dilakukan penampilan antarmuka HTML sesuai dengan mode yang terpilih, yaitu mode *impersonation* atau tanpa *impersonation.*

```
1.  function handleLoginAction()
2.  {
3.     request = GET request object
4.     response = CREATE response object
5.     loginUrl = GET loginUrl object
6.     authorizeUrl = GET authorizeUrl object
7.     view = GET view object
8.
9.     if request.method() == 'POST'
10.       content = request['content']
11.       credentials = decrypt(content)
12.
13.       userAuthenticated = authenticate(credentials)
14.
15.       if userAuthenticated == true
16.          return response.redirect(authorizeUrl)
17.       else
18.          return response.redirect(loginUrl)
19.       endif
20.
21.    else if request.method() == 'GET'
22.       if request['impersonated_sub']
23.          return view.pick('login-as')
24.       else
25.          return view.pick('login')
26.       endif
27.    endif
28. }
```

Pseudocode 4.2 Fungsi *Handle Login Action*

Fungsi ini dapat diakses melalui dua URI. URI pertama digunakan untuk mengirim data pengguna untuk melanjutkan proses *login*, sedangkan URI kedua digunakan untuk menampilkan halaman *login*. URI pertama dapat dilihat pada Kode Sumber 4.54, sedangkan URI kedua dapat dilihat pada Kode Sumber 4.55. Parameter yang dapat dikirim pada URI pertama dapat dilihat pada Tabel 4.4.

```
POST https://dev-my.its.ac.id/signin
```

Kode Sumber 4.54 URI *Login*

Tabel 4.4 Parameter *Login*

Nama Parameter	Deskripsi
content	Hasil enkripsi *username, password, device method* dan *password state*.

```
GET https://dev-my.its.ac.id/signin
```

Kode Sumber 4.55 URI Halaman *Login*

4.1.4.1.2.5 Kelas *Ciba Repository*

Kelas *Ciba Repository* adalah kelas yang memiliki tanggung jawab untuk melakukan komunikasi ke basis data. Segala hal yang bergantung pada pencarian dan penyimpanan data akan ditangani oleh kelas ini. Kelas *Ciba Repository* akan bergantung pada abstraksi *Ciba Storage Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.6. Kelas ini memiliki lima fungsi umum sama halnya dengan abstraksi *Ciba Storage Interface*. Namun pada subbab ini hanya digunakan empat fungsi. Fungsi kelima akan dijelaskan pada subbab 4.1.4.1.3.3.1 karena berhubungan dengan pengambilan *access token*.

4.1.4.1.2.5.1 Fungsi *Get Authentication Request Id*

Fungsi pertama yang akan dijelaskan adalah *getAuthenticationRequestId*. Fungsi ini memiliki tugas untuk mengambil *authentication request id* dari basis data. Fungsi ini dapat dilihat pada Kode Sumber 4.56. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* menyusun SQL yang akan digunakan untuk mengambil *authentication request id*.
2. Pada baris 5, *library* menyusun parameter SQL.

3. Pada baris 7, *library* mengeksekusi SQL.
4. Pada baris 9 sampai 12, *library* melakukan konversi tipe data menjadi *unix timestamp*.
5. Pada baris 14, *library* mengembalikan nilai *authentication request id*.

```
1.  public function getAuthReqId($auth_req_id)
2.  {
3.      $sql = \sprintf('SELECT * FROM %s WHERE auth_req_id = C
    ONVERT(uniqueidentifier, :auth_req_id) ', $this-
    >tables['ciba_request_table']);
4.
5.      $params = \compact('auth_req_id');
6.
7.      $authReqId = $this->getDb()-
    >fetchOne($sql, \Phalcon\Db::FETCH_ASSOC, $params);
8.
9.      if ($authReqId) {
10.         $authReqId['expires_in'] = strtotime($authReqId['expires
    _in']);
11.         $authReqId['latest_token_requested_at'] = strtotime($aut
    hReqId['latest_token_requested_at']);
12.     }
13.
14.     return $authReqId;
15. }
```

Kode Sumber 4.56 Fungsi *Get Authentication Request Id*

4.1.4.1.2.5.2 Fungsi *Set Authentication Request Id*

Fungsi kedua yang akan dijelaskan adalah *setAuthenticationRequestId*. Fungsi ini memiliki tugas untuk menyimpan *authentication request id* pada basis data. Fungsi ini dapat dilihat pada Kode Sumber 4.57 dan Kode Sumber 4.58. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 8 sampai 11, *library* menginisialisasi nilai awal parameter SQL.

2. Pada baris 13 sampai 17, *library* menyusun SQL untuk menyimpan atau memperbarui *authentication request id*.
3. Pada baris 19, *library* menyusun parameter SQL.
4. Pada baris 21, *library* mengeksekusi SQL.
5. Pada baris 23 sampai 27, *library* mengembalikan nilai *true* sebagai tanda berhasil dan *false* sebagai tanda gagal.

```
1.  public function setAuthReqId
2.  ($auth_req_id, $client_id, $provider_id,
3.  $user_id, $expires,
4.  $client_notification_token = null, $hint = null,
5.  $binding_message = null, $scope = null,
6.  $id_token = null)
7.  {
8.     $expires = date('Y-m-d H:i:s', $expires);
9.     $valid = 1;
10.    $created_at = date('Y-m-d H:i:s');
11.    $consented = null;
12.
13.    if ($this->getAuthReqId($auth_req_id)) {
14.       $sql = sprintf('UPDATE %s SET client_id = :client_id, provi
    der_id = :provider_id, user_id = :user_id, hint = :hint, binding_
    message = :binding_message, client_notification_token = :clien
    t_notification_token, expires_in = :expires_in, created_at = :cre
    ated_at, valid = :valid, id_token =:id_token, scope =:scope, cons
    ented =:consented WHERE auth_req_id = :auth_req_id', $this-
    >tables['ciba_request_table']);
```

Kode Sumber 4.57 Fungsi *Set Authentication Request Id* (1)

```
15.     } else {
16.         $sql = sprintf('INSERT INTO %s (auth_req_id, client_id, pro
    vider_id, user_id, hint, binding_message, client_notification_tok
    en, expires_in, created_at, valid, id_token, scope, consented) VA
    LUES (:auth_req_id, :client_id, :provider_id, :user_id, :hint, :bind
    ing_message, :client_notification_token, :expires, :created_at, :v
    alid, :id_token, :scope, :consented)', $this-
    >tables['ciba_request_table']);
17.     }
18.
19.     $params = compact('auth_req_id', 'client_id', 'provider_id', 'u
    ser_id', 'hint', 'binding_message', 'client_notification_token', 'ex
    pires', 'created_at', 'valid', 'id_token', 'scope', 'consented');
20.
21.     $result = $this->getDb()->execute($sql, $params);
22.
23.     if ($result) {
24.         return true;
25.     }
26.
27.     return false;
28. }
```

Kode Sumber 4.58 Fungsi *Set Authentication Request Id* (2)

4.1.4.1.2.5.3 Fungsi *Expire Authentication Request Id*

Fungsi ketiga yang akan dijelaskan adalah *expireAuthenticationRequestId*. Fungsi ini memiliki tugas untuk menyimpan *authentication request id* sebagai kedaluwarsa pada basis data. Tujuannya disimpan sebagai kedaluwarsa adalah agar *authentication request id* tidak dapat digunakan untuk memberi hak akses atau mengambil token pada masa akan datang. Fungsi ini dapat dilihat pada Kode Sumber 4.59. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3 sampai 6, *library* menyusun parameter dan SQL.
2. Pada baris 8, *library* mengeksekusi SQL.

3. Pada baris 10 sampai 14, *library* mengembalikan nilai *true* sebagai tanda berhasil, dan nilai *false* sebagai tanda gagal.

```
1.  public function expireAuthReqId($auth_req_id)
2.  {
3.      $valid = 0;
4.      $sql = sprintf('UPDATE %s SET valid = :valid WHERE auth_r
    eq_id = :auth_req_id', $this->tables['ciba_request_table']);
5.
6.      $params = compact('auth_req_id', 'valid');
7.
8.      $result = $this->getDb()->execute($sql, $params);
9.
10.     if ($result) {
11.         return true;
12.     }
13.
14.     return false;
15. }
```

Kode Sumber 4.59 Fungsi *Expire Authentication Request Id*

4.1.4.1.2.5.4 Fungsi *Set Consent*

Fungsi keempat yang akan dijelaskan adalah *setConsent*. Fungsi ini memiliki tugas untuk menyimpan hak akses yang diberikan oleh pengguna *Authentication Device* pada basis data. Fungsi ini dapat dilihat pada Kode Sumber 4.60. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 4 sampai 6, *library* menyusun parameter dan SQL.
2. Pada baris 7, *library* mengeksekusi SQL.
3. Pada baris 10 sampai 14, *library* mengembalikan nilai *true* sebagai tanda berhasil, dan nilai *false* sebagai tanda gagal.

```
1.  public function setConsent
2.  ($auth_req_id, $consent)
3.    {
4.      $sql = sprintf('UPDATE %s SET consented = :consent WH
    ERE auth_req_id = :auth_req_id', $this-
    >tables['ciba_request_table']);
5.
6.      $params = compact('auth_req_id', 'consent');
7.      $result = $this->getDb()->execute($sql, $params);
8.
9.      if ($result) {
10.       return true;
11.     }
12.
13.     return false;
14.   }
```

Kode Sumber 4.60 Fungsi *Set Consent*

4.1.4.1.3 Penerapan Pengambilan *Access Token*

Penerapan pengambilan *access token* menghasilkan satu kelas *controller* dan dua *repository*. Penjelasannya adalah sebagai berikut.

4.1.4.1.3.1 Kelas *Token Controller*

Kelas *Token Controller* menangani pengambilan *access token* yang diminta oleh *aplikasi klien*. Fungsi yang menanganinya adalah *handleTokenAction*.

4.1.4.1.3.1.1 Fungsi *Handle Token Action*

Fungsi *handleTokenAction* memliki tugas untuk melayani aplikasi klien yang ingin mengambil *access token*. Fungsi ini mengakomodir pengambilan *access token* untuk berbagai macam *grant type* atau alur. Diantaranya adalah CIBA dan *Authorization Code*. Fungsi ini dapat dilihat pada Kode Sumber 4.61. Langkah kerja fungsi ini adalah sebagai berikut.

1. Pada baris 3 sampai 4, *controller* akan membuat obyek *request* dan *response*. Obyek *request* digunakan untuk

menyimpan *header* dan parameter *body* pada *request* HTTP. Parameter *body* pada *request* telah dijelaskan pada subbab 2.4.4 sehingga *library* server autorisasi CIBA dapat mengidentifikasi bahwa *request* ini adalah permintaan pengambilan *access token* menggunakan salah satu diantara mode token yang tersedia pada CIBA, yaitu *poll*, *ping* dan *push*. Obyek *response* digunakan untuk mengembalikan pesan dari *library* server autorisasi CIBA.

2. Pada baris 6 sampai 7, *controller* akan memeriksa alur yang digunakan, jika alurnya menggunakan CIBA maka akan masuk ke *statement* ini. *Controller* akan memanggil fungsi *handleCibaTokenRequest* disertakan dengan parameter *request dan response* yang dimiliki oleh *library* server autorisasi CIBA.
3. Pada baris 8 sampai 9, *controller* akan memanggil fungsi *handleTokenRequest* jika alur adalah selain CIBA.
4. Pada baris 12, *controller* akan mengembalikan pesan yang didapatkan dari *library* server autorisasi CIBA.

```
1.  public function handleTokenAction()
2.  {
3.      $request = Request::createFromGlobals();
4.      $response = new Response();
5.
6.      if ($request-
    >request('grant_type') === self::GRANT_TYPE_CIBA) {
7.          $this->serverCiba -
    >handleCibaTokenRequest($request, $response);
8.      } else {
9.          $this->serverCiba -
    >handleTokenRequest($request, $response);
10.     }
11.
12.     return $response->send();
13. }
```

Kode Sumber 4.61 Fungsi *Handle Token Action*

Fungsi ini dapat diakses melalui rute URI yang terdefinisi pada Kode Sumber 4.62 URI Pengambilan Token dengan parameter yang telah dijelaskan pada subbab 2.4.4.

```
POST https://dev-my.its.ac.id/token
```

Kode Sumber 4.62 URI Pengambilan Token

4.1.4.1.3.2 Kelas *Access Token Repository*

Kelas *Access Token Repository* adalah kelas yang memiliki tanggung jawab untuk berkomunikasi ke basis data. Kelas *Access Token Repository* akan bergantung pada abstraksi *Access Token Storage Interface* yang telah dijelaskan pada subbab 3.2.3.1.2.7. Kelas ini memiliki tiga fungsi yang umum, namun pada tugas akhir ini, lebih khususnya pada penerapan pengambilan *access token*, hanya menggunakan satu fungsi, yaitu fungsi *setAccessToken*. Fungsi lainnya digunakan pada alur yang berbeda dari CIBA. Implementasi kelas *repository* ini tidak diharuskan menggunakan basis data tertentu seperti SQL. Oleh karena itu,

implementasi kelas ini menggunakan basis data Redis. Alasan menggunakan Redis untuk menyimpan *access token* adalah karena *access token* cenderung menumpuk pada basis data, sehingga jika menggunakan basis data relasional akan terjadi penurunan performa yang signifikan. Dengan menggunakan Redis, sistem dapat memanfaatkan *in memory caching* yang dimiliki oleh Redis sehingga pencarian *access token* akan jauh lebih cepat dibandingkan dengan basis data yang menyimpan ke *disk*.

4.1.4.1.3.2.1 Fungsi *Set Access Token*

Fungsi pertama yang akan dijelaskan adalah *setAccessToken*. Fungsi ini memiliki tugas untuk menyimpan *access token* yang berhasil disusun ke suatu basis data. Fungsi ini dapat dilihat pada Kode Sumber 4.63. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 3, *library* akan menyusun waktu kedaluwarsa dari *unix timestamp* menjadi *datetime string*.
2. Pada baris 5 sampai 6, *library* akan menyusun parameter dan mengubah struktur data *access token* menjadi *JSON string*.
3. Pada baris 8, *library* akan mengambil obyek Redis yang digunakan untuk berkomunikasi ke basis data.
4. Pada baris 10 sampai 11, *library* menyusun *key* yang digunakan sebagai identifikasi di Redis dan menyimpan *access token*.
5. Pada baris 13, *library* mengembalikan nilai *true* sebagai tanda berhasil.

```
1.  public function setAccessToken($access_token, $client_id, $us
    er_id, $expires, $scope = null)
2.  {
3.      $expires = date('Y-m-d H:i:s', $expires);
4.
5.      $params = compact('access_token', 'client_id', 'user_id', 'expi
    res', 'scope');
6.      $jsonData = json_encode($params);
7.
8.      $redis = $this->getRedis();
9.
10.     $key = 'access_token:' . $access_token;
11.     $redis->setex($key, 3600, $jsonData);
12.
13.     return true;
14. }
```

Kode Sumber 4.63 Fungsi *Set Access Token*

4.1.4.1.3.3 Kelas Ciba Repository

Melanjutkan kelas *ciba repository* yang telah dijelaskan pada subbab 4.1.4.1.2.5, kelas ini memiliki fungsi kelima yang digunakan untuk menyimpan waktu terakhir permintaan token. Fungsi tersebut dijelaskan sebagai berikut.

4.1.4.1.3.3.1 Fungsi *Set Last Requested Token At*

Fungsi kelima pada kelas *ciba repository* yang akan dijelaskan adalah *setLastRequestedTokenAt*. Fungsi ini memiliki tugas untuk menyimpan waktu terakhir meminta token menggunakan suatu *authentication request id* pada basis data. Tujuannya adalah ketika menggunakan mode token *poll*, server autorisasi harus mampu mengkalkulasi jarak waktu antara setiap permintaan token yang masuk. Jika permintaan token terlalu cepat, maka server autorisasi harus mengirim pesan untuk memperpelan permintaan. Nilai *last requested token at* dapat membantu server autorisasi untuk mengetahui terakhir kali suatu *authentication request id* digunakan untuk meminta token. Fungsi ini dapat dilihat

pada Kode Sumber 4.64. Langkah kerja fungsi ini adalah sebagai berikut:

1. Pada baris 4 sampai 6, *library* menyusun parameter dan SQL.
2. Pada baris 9, *library* mengeksekusi dan mengembalikan nilai *true* sebagai tanda berhasil, dan nilai *false* sebagai tanda gagal.

```
1.  public function setLastRequestedTokenAt
2.  ($auth_req_id, $time)
3.    {
4.      $time = date('Y-m-d H:i:s', $time);
5.
6.      $sql = sprintf("UPDATE %s SET latest_token_requested_at
    = :time WHERE auth_req_id = :auth_req_id", $this-
    >tables['ciba_request_table']);
7.      $params = compact('auth_req_id', 'time');
8.
9.      return $this->getDb()->execute($sql, $params);
10.   }
```

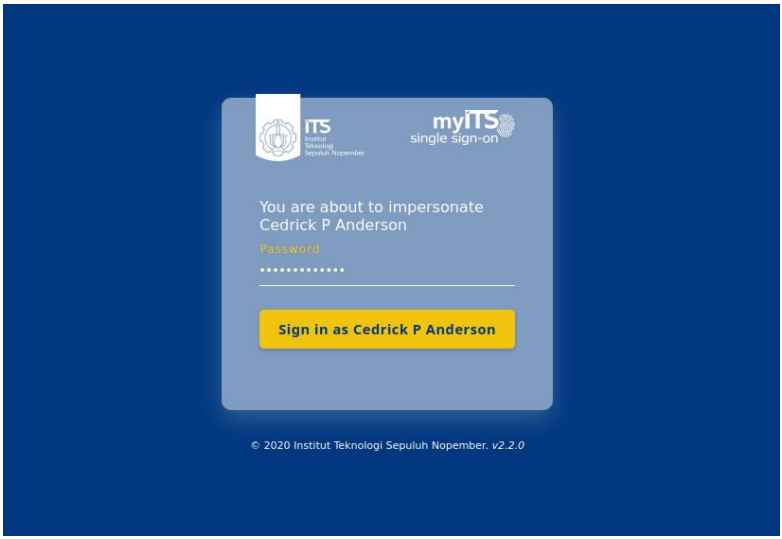
Kode Sumber 4.64 Fungsi *Set Last Requested Token At*

4.1.5 Implementasi Antarmuka

Pada subbab ini dijelaskan mengenai tampilan antarmuka pengguna untuk sistem yang dibangun dalam tugas akhir ini. Antarmuka yang dibangun berbasis *web* dengan menggunakan teknologi *template engine* volt yang disediakan oleh Phalcon.

4.1.5.1 Antarmuka Halaman Formulir Kata Sandi

Antarmuka hamalan formulir kata sandi merupakan halaman yang ditampilkan pada layar saat pengguna melakukan *impersonation*. Dalam antarmuka ini, pengguna dapat memasukkan kata sandi untuk melanjutkan proses *impersonation*. Tampilan antarmuka ini dapat dilihat pada Gambar 4.1.



Gambar 4.1 Antarmuka Formulir Kata Sandi

BAB V
UJI COBA DAN EVALUASI

Bab ini membahas tentang rangkaian pengujian dan evaluasi perangkat lunak yang dilakukan sesuai hasil implementasi. Pengujian dilakukan untuk menguji fungsionalitas secara keseluruhan berjalan sesuai keinginan atau tidak. Pengujian dilakukan dengan beberapa macam skenario. Pembahasan pada bab ini meliputi lingkungan pengujuan, skenario dan hasil pengujian, dan evaluasi.

5.1 Lingkungan Pengujian

Lingkungan pengujian sistem yang digunakan adalah berupa satu unit komputer pengguna yang berperan sebagai klien, dan sebuah unit *virtual machine* yang berperan sebagai server. Adapun rincian masing-masing lingkungan pengujian dapat dilihat pada Tabel 5.1 dan Tabel 5.2.

Tabel 5.1 Lingkungan Uji Ciba Komputer Pengguna

Spesifikasi	Deskripsi
CPU	Intel® Pentium® G4560
RAM	8 GB
Sistem Operasi	Windows 10
Perangkat Lunak	Postman dan Mozila Firefox

Tabel 5.2 Uji Coba Lingkungan *Virtual Machine*

Spesifikasi	Deskripsi
CPU	Intel® Xeon® CPU E5-2690 v4
RAM	2 GB
Sistem Operasi	Debian GNU/Linux 9.4
Web Server	Nginx 1.10.3

5.2 Skenario Pengujian

Bagian ini menjelaskan proses uji coba yang digunakan untuk menguji fungsionalitas server autorisasi CIBA. 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 perangkat lunak tanpa memandang struktur internalnya.

Server autorisasi CIBA dikembangkan dengan cara mengikuti spesifikasi yang telah dikeluarkan oleh organisasi yang bernama OpenID. Organisasi OpenID menyediakan *conformance testing* yang dapat digunakan untuk melakukan uji coba implementasi server autorisasi yang telah dikembangkan. *Conformance testing* ini bertujuan untuk mengetahui apakah sistem yang dikembangkan sudah sesuai standar spesifikasi atau tidak.

Namun pada saat pengembangan tugas akhir ini, belum ada *conformance testing* yang dikembangkan OpenID untuk alur CIBA. Salah satu *conformance testing* yang telah disediakan oleh OpenID adalah untuk *Financial Grade API CIBA* (FAPI CIBA). Sedangkan alur CIBA dan FAPI CIBA memiliki perbedaan yang signifikan [29]. FAPI CIBA menerapkan mekanisme pengamanan yang berbeda, sebagai contoh pada FAPI CIBA tidak mendukung autentikasi klien dengan skema *Basic Authentication*, FAPI CIBA menggunakan skema *Private Key JWT* dan *Mutual Transport Layer Security* (MTLS), namun penerapan CIBA pada tugas akhir ini menggunakan skema *Basic Authentication*. Selain itu FAPI CIBA harus menggunakan algoritma PS256 atau ES256 untuk menyusun JWT yang menggunakan serialisasi JWS, sedangkan pada penerapan CIBA pada tugas akhir ini menggunakan algoritma RS256. Sehingga jika menggunakan platform *conformance testing* FAPI CIBA yang disediakan oleh OpenID, hasil *conformance testing* akan gagal.

Sehingga yang bisa dilakukan untuk uji coba server autorisasi CIBA adalah untuk mengambil *test case* yang dimiliki

oleh *conformance testing* FAPI CIBA dan menggunakannya pada aplikasi yang dapat mengirim *request* HTTP. Aplikasi yang bisa mengirim request HTTP adalah Postman. Pada proses uji coba, pengujian dilakukan dengan menjalankan pemanggilan REST API menggunakan aplikasi Postman. Uji coba menggunakan Postman akan tetap sesuai dengan spesifikasi CIBA. *Test case* yang diambil pada *conformance testing* FAPI CIBA akan tetap valid karena CIBA dan FAPI CIBA merupakan alur yang memiliki fondasi yang sama, tetapi hanya berbeda pada mekanisme pengamanan yang digunakan.

Aplikasi klien yang digunakan pada kumpulan kasus uji ini telah terdaftar dengan nilai *client id* *"*EC19D04C-D5B9-4791-B211-D70E715297EC*"* dan *client secret* "e1f25104dac2612696c88ffc93afce1c". Kedua nilai tersebut akan menghasilkan nilai yang digunakan sebagai *authorization header* dengan skema *basic access authentication*. Rumus *basic access authentication* dapat dilihat pada Pseudocode 5.1. Nilai *basic access authentication* menggunakan *client id* dan *client secret* aplikasi klien ini adalah "RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDOmUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj".

```
1.  basicAuthValue = base64Encode(clientId + ":" + clientSecret)
```

Pseudocode 5.1 Rumus *Basic Access Authentication*

5.2.1 Kasus Pengujian CIBA Menggunakan Mode Token *Push*

Pada kasus ini, server autorisasi akan diujui dengan melakukan CIBA menggunakan mode token *push*.

Tabel 5.3 Kasus Pengujian CIBA Menggunakan Mode Token *Push*

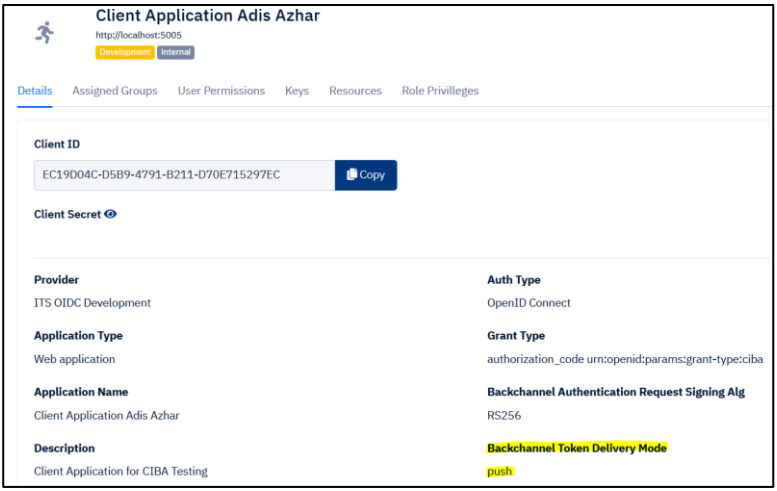
ID	**UJ-001**

Nama Skenario	Pengujian CIBA Menggunakan Mode Token *Push*
Tujuan Pengujian	Menguji server autorisasi untuk melayani klien menggunakan alur CIBA mode token *push*.
Skenario 1	Menggunakan *non signed request*
Kondisi Awal	Klien sudah terdaftar, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi. 3. Klien mendapatkan *response* dari server autorisasi berupa *authentication request id*. 4. Klien menyusun parameter *request* pemberian hak akses menggunakan *authentication request id* yang didapatkan. 5. Klien mengirim *request* pemberian hak akses. 6. Klien mendapatkan *response* dari server autorisasi beruka *acknowledgement*. 7. Klien mendapatkan token yang dikirim oleh server autorisasi.
Hasil yang diharapkan	Mendapatkan token dari server autorisasi.
Hasil yang diperoleh	Mendapatkan token dari server autorisasi.
Hasil Pengujian	Berhasil
Skenario 2	Memanggil token *endpoint*
Kondisi Awal	Klien sudah mendapatkan *authentication request id*.

Langkah Pengujian	1. Klien menyusun parameter *request* token. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena klien terdaftar menggunakan mode token *push*.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena klien terdaftar menggunakan mode token *push*.

Pada skenario pertama, *request* autentikasi yang disusun menggunakan aplikasi Postman dapat dilihat pada Kode Sumber 5.1. Klien sudah terdaftar menggunakan mode token *push* dan dapat dilihat pada Gambar 5.1. Server autorisasi mengembalikan *response* yang terdapat *authentication request id*. *Authentication request id* ini akan valid selama 120 detik. *Response* ini dapat dilihat pada Kode Sumber 5.2. *Request* pemberian hak akses dapat dilihat pada Kode Sumber 5.3. *Token* yang dikirim oleh server autorisasi dapat dilihat pada Kode Sumber 5.4. Server autorisasi mengirim token melalui *client notification endpoint* yang dimiliki oleh aplikasi klien. Khusus pengujian menggunakan mode token *push*, dibuat aplikasi yang memiliki *client notification endpoint* agar dapat menampilkan *token* yang dikirim oleh server autorisasi. Skenario ini berhasil dilakukan.

Pada skenario terakhir, dilakukan permintaan token dengan cara mengirim *request* token ke token *endpoint*. Uji coba yang berhasil memiliki tanda pesan *error* dari server autorisasi, karena klien yang terdaftar menggunakan mode token *push* tidak dilayani pada token *endpoint*. Pesan *error* dapat dilihat pada Kode Sumber 5.5. Skenario ini berhasil dilakukan.



Gambar 5.1 Aplikas Klien Terdaftar Menggunakan Mode token *Push*

```
POST /bc-authorize? HTTP/1.1
Host: dev-my.its.ac.id
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Content-Type: application/x-www-form-urlencoded

scope=openid email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-2020&requested_expiry=120
```

Kode Sumber 5.1 *Request* Autentikasi

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Mon, 11 May 2020 13:17:39 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive

{"auth_req_id":"6A60FC96-D751-4373-AC37-
FD1E5A70D5EF","expires_in":1589205637}
```

Kode Sumber 5.2 *Response* Autentikasi

```
POST /ciba-consent? HTTP/1.1
Host: dev-my.its.ac.id
Content-Type: application/x-www-form-urlencoded
Content-Type: application/x-www-form-urlencoded

auth_req_id=6A60FC96-D751-4373-AC37-
FD1E5A70D5EF&user_consent=accept
```

Kode Sumber 5.3 *Request* Pemberian Hak Akses

{
"AccessToken":"24810ada4977cce6c214f489335d220eec92aabd",
"TokenType":"Bearer",
"ExpiresIn":3600,
"IdToken":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjhMb
Dh2UlU3UmxTc2ZlcXZrZGVueG5YZjlnRlNwUXk0b3dwaVY4dlRFbE0
ifQ.eyJpc3MiOiJodHRwczpcL1wvZGV2LW15Lml0cy5hYy5pZCIsInN1
YiI6IjJEMDJBMTVFLTJFQTEtNEU5RS04MzI3LTI3Q0U1QkZERTI5Qi
IsImF1ZCI6IkVDMTlEMDRDLUQ1QjktNDc5MS1CMjExLUQ3MEU3
MTUyOTdFQyIsImlhdCI6MTU4OTIwNTU0OCwiZXhwIjoxNTg5MjA5
MTQ4LCJhdXRoX3RpbWUiOjE1ODkyMDU1NDgsInVybjpvcGVuaWQ
6cGFyYW1zOmp3dDpjbGFpbTphdXRoX3JlcV9pZCI6IjZBNjBGQzk2L
UQ3NTEtNDM3My1BQzM3LUZEMUU1QTcwRDVFRiIsImF0X2hhc2g
iOiJ6alA4emMyMElRSEF4anltVUh6TFZnIiwidXJuOm9wZW5pZDpwY
XJhbXM6and0OmNsYWltOnJ0X2hhc2giOiJVWDBoVFJtUTBXVHBsT
HliVXRjekZBIn0.Wd5DxvdDsy5Lu1pO3DYBoicx6ZV46mSNvlf5kzQw7
mZjcqjm2IhlDw75sgM1O0vKXJbSkOh6YyLX5FWYK9MIfgLBIBMc5k
DVyIqefE2ltx7XequHgmZ1iLJEEPcr1zwUAnnl_t_y4cnsbzqBgIiOgod3_4
oNh-q8CY8WKZrePKe29lNyvzhHL6De6-
tVT8QFbDl241yTx96xEIinrGu8Xra_gTRQh7vLltHLxIaJBk3aKFqhtartIe
X6AM9CNFjfcw724iIx_0Ug_qzsYjrQcDczai2StJcX0uD29OqFqqsaErnJ4
BSpfWDowvLKcrMp_ViiPHKQ0UH8M4w8yymurA"
}

Kode Sumber 5.4 *Access Token* Dari Server Autorisasi

HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Mon, 11 May 2020 14:39:44 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"unauthorized_client","error_description":"The Client is not authorized as it is configured in Push Mode"}

Kode Sumber 5.5 *Response* Berupa Pesan *Error* Dari Server Autorisasi

5.2.2 Kasus Pengujian CIBA Menggunakan Mode Token *Poll*

Pada kasus ini, server autorisasi akan diujui dengan melakukan CIBA menggunakan mode token *poll*.

Tabel 5.4 Kasus Pengujian CIBA Menggunakan Mode Token *Poll*

ID	**UJ-002**
Nama Skenario	Pengujian CIBA Menggunakan Mode Token *Poll*
Tujuan Pengujian	Menguji server autorisasi untuk melayani klien menggunakan alur CIBA mode token *poll*.
Skenario 1	Menggunakan *non signed request*
Kondisi Awal	Klien sudah terdaftar, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi. 3. Klien mendapatkan *response* dari server autorisasi berupa *authentication request id*. 4. Klien menyusun parameter *request* pemberian hak akses menggunakan *authentication request id* yang didapatkan. 5. Klien mengirim *request* pemberian hak akses. 6. Klien menyusun parameter *request* token. 7. Klien mengirim *request* token. 8. Klien mendapatkan *response* dari server autorisasi berupa token.

Hasil yang diharapkan	Mendapatkan token dari server autorisasi.
Hasil yang diperoleh	Mendapatkan token dari server autorisasi.
Hasil Pengujian	Berhasil
Skenario 2	Menggunakan *client id* yang tidak terdaftar saat melakukan *request* autentikasi.
Kondisi Awal	Klien tidak terdaftar
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena klien tidak ditemukan atau tidak terdaftar.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena klien tidak ditemukan atau tidak terdaftar.
Hasil Pengujian	Berhasil
Skenario 3	Menggunakan *client id* yang berbeda saat melakukan *request* token.
Kondisi Awal	Klien sudah mendapatkan *authentication request id.*
Langkah Pengujian	1. Klien menyusun parameter *request* token dengan *client id* yang berbeda. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *client id* tidak sesuai dengan *client id* yang terikat pada *authentication request id*.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *client id* tidak sesuai dengan *client id* yang terikat pada *authentication request id*.

Hasil Pengujian	Berhasil
Skenario 4	Menggunakan *authentication request id* yang berbeda saat melakukan *request* token.
Kondisi Awal	Klien sudah mendapatkan *authentication request id.*
Langkah Pengujian	1. Klien menyusun parameter *request* token dengan *authentication request id* yang berbeda. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *authentication request id* tidak ditemukan atau milik klien lain.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *authentication request id* tidak ditemukan atau milik klien lain.
Hasil Pengujian	Berhasil
Skenario 5	Memanggil token *endpoint* dengan cepat.
Kondisi Awal	Klien sudah mendapatkan *authentication request id* dan klien hak akses masih *pending*.
Langkah Pengujian	1. Klien menyusun parameter *request* token dengan *authentication request id* yang memiliki status hak akses *pending.* 2. Klien mengirim *request* token sebanyak dua kali dalam jangka kurang (interval) dari lima detik.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena interval *polling* terlalu cepat.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena interval *polling* terlalu cepat.
Hasil Pengujian	Berhasil

Skenario 6	Menggunakan *binding message* yang Panjang
Kondisi Awal	Klien sudah terdaftar, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *binding message* tidak valid
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *binding message* tidak valid
Hasil Pengujian	Berhasil
Skenario 7	Melakukan *request* autentikasi tanpa *user code*
Kondisi Awal	Klien sudah terdaftar menggunakan *user code*, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *user code* tidak ada
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *user code* tidak ada
Hasil Pengujian	Berhasil
Skenario 8	Melakukan *request* autentikasi dengan *user code* yang salah
Kondisi Awal	Klien sudah terdaftar menggunakan *user code*, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi.

Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *user code salah*
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *user code salah*
Hasil Pengujian	Berhasil

Pada skenario pertama, *request* autentikasi yang disusun menggunakan aplikasi Postman dapat dilihat pada Kode Sumber 5.6. Klien sudah terdaftar menggunakan mode token *poll* dan dapat dilihat pada Gambar 5.2. Server autorisasi mengembalikan *response* yang terdapat *authentication request id*. *Authentication request id* ini akan valid selama 120 detik. *Response* ini dapat dilihat pada Kode Sumber 5.7. *Request* pemberian hak akses dapat dilihat pada Kode Sumber 5.8. Lalu disusun *request* token yang dapat dilihat pada Kode Sumber 5.9. Server autorisasi mengembalikan *response* token yang dapat dilihat pada Kode Sumber 5.10. Skenario ini berhasil dilakukan.

Pada skenario kedua, didapatkant *client id* dengan nilai "974e4bc3-a748-41c1-a74d-7b95ca1e66c0". *Client id* ini tidak terdaftar pada server autorisasi. Sehingga jika dilakukan *request* autentikasi menggunakan *client id* tersebut, server autorisasi tidak akan melayani. *Request* autentikasi dapat dilihat pada Kode Sumber 5.11 dan *response* dapat dilihat pada Kode Sumber 5.12. Skenario ini berhasil dilakukan.

Pada skenario ketiga, telah dilakukan *request* autentikasi menggunakan *request* parameter yang sama pada Kode Sumber 5.6. Namun ketika melakukan *request* token, parameter *client id* diubah menjadi "0ebfea68-e667-4c0b-b29d-043f861a67e3" dari yang awal mulanya "EC19D04C-D5B9-4791-B211-D70E715297EC". Ini menyebabkan *client id* berbeda dengan yang digunakan saat *request* autentikasi. Server autorisasi tidak akan memberikan klien token karena terdeteksi *client id* yang berbeda dengan *client id* yang terikat pada *authentication request id*. *Request* token dan *response* token berupa pesan *error* dapat dilihat

pada Kode Sumber 5.13 dan Kode Sumber 5.14. Skenario ini berhasil dilakukan.

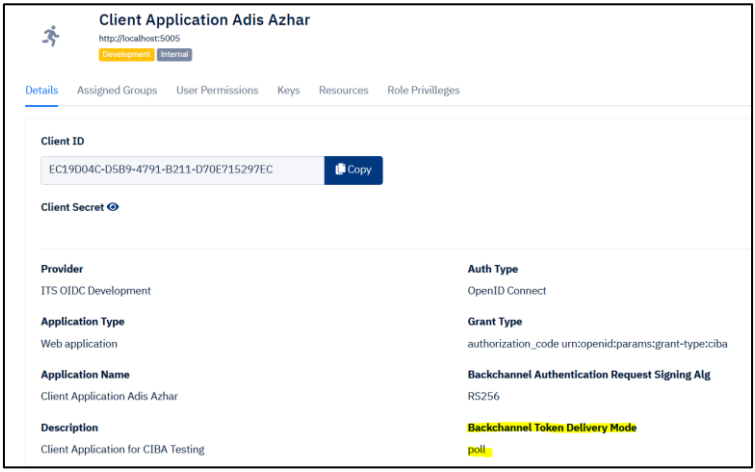
Pada skenario keempat, telah dilakukan *request* autentikasi menggunakan *request* parameter yang sama pada Kode Sumber 5.6 dan menghasilkan *authentication request id* dengan nilai "05C1A26C-3116-4714-B88B-0846EB0574F7". Namun ketika melakukan *request* token, parameter *authentication request id* diubah menjadi "6ed95ace-7210-4325-874f-e835cdfb4280" dari yang awal mulanya "05C1A26C-3116-4714-B88B-0846EB0574F7". Server autorisasi tidak akan memberikan klien token karena *authentication request id* tidak tercatat atau *authentication request id* merupakan milik klien lain. *Request* token dan *response* token berupa pesan *error* dapat dilihat pada Kode Sumber 5.15 dan Kode Sumber 5.16. Skenario ini berhasil dilakukan.

Pada skenario kelima, telah dilakukan *request* autentikasi menggunakan *request* parameter yang sama pada Kode Sumber 5.6 dan menghasilkan *authentication request id* dengan nilai "900EC5EF-F313-48C6-871A-37C54B01626E". Lalu disusun *request* token menggunakan *authentication request id* tersebut. Keadaan *authentication request id* ini belum diberi hak akses agar server autorisasi aktif dalam *long polling* yaitu memberi *response* sampai ada pemberian hak akses atau ketika sudah *timeout* (30 detik). Lalu *request* token ini dikirim sebanyak dua kali dalam jangka waktu kurang dari lima detik. Lima detik adalah nilai *default* jarak waktu *polling* yang dimiliki oleh server autorisasi. *Request* token dan *response* token berupa pesan *error* dapat dilihat pada Kode Sumber 5.17 dan Kode Sumber 5.18. Skenario ini berhasil dilakukan.

Pada skenario keenam, telah dilakukan *request* autentikasi menggunakan parameter *binding message* dengan panjang lebih dari sepuluh karakter. *Request* ini dapat dilihat pada Kode Sumber 5.19. Server autorisasi mengembalikan pesan *error* karena *binding message* terlalu panjang. *Response* ini dapat dilihat pada Kode Sumber 5.20.

Pada skenario ketujuh, telah dilakukan *request* autentikasi tanpa menggunakan parameter *user code*. Aplikasi klien ini terdaftar menggunakan *user code* sehingga *request* autentikasi diwajibkan menyertakan *user code*. *Request* ini dapat dilihat pada Kode Sumber 5.21. Server autorisasi mengembalikan pesan *error* karena *user code* tidak ada. *Response* ini dapat dilihat pada Kode Sumber 5.22.

Pada skenario terakhir, telah dilakukan *request* autentikasi menggunakan parameter *user code* yang salah. *User code* adalah sandi rahasia yang dimiliki oleh pengguna. *Request* ini dapat dilihat pada Kode Sumber 5.23. Server autorisasi mengembalikan pesan *error* karena *user code* salah. *Response* ini dapat dilihat pada Kode Sumber 5.24.



Gambar 5.2 Aplikasi Klien Terdaftar Menggunakan Mode Token *Poll*

```
POST /bc-authorize? HTTP/1.1
Host: dev-my.its.ac.id
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Content-Type: application/x-www-form-urlencoded

scope=openid email&client_notification_token=77a554a6-9a4e-45b9-8643-
8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-2020&requested_expiry=120
```

Kode Sumber 5.6 Skenario 1 - *Request* Autentikasi

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Mon, 11 May 2020 17:31:46 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive

{"auth_req_id":"49E5708D-BD44-485B-813D-
ECC3B21C7F2B","expires_in":1589218425,"interval":5}
```

Kode Sumber 5.7 Skenario 1 - *Response* Autentikasi

```
POST /ciba-consent? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept: */*
Host: dev-my.its.ac.id
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

auth_req_id=49E5708D-BD44-485B-813D-
ECC3B21C7F2B&user_consent=accept
```

Kode Sumber 5.8 Skenario 1 - *Request* Pemberian Hak Akses

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=49E5708D-BD44-485B-813D-
ECC3B21C7F2B
```

Kode Sumber 5.9 Skenario 1 - *Request* Token

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Mon, 11 May 2020 17:32:17 GMT
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{"access_token":"10baff1a73bc97a83d257279f903201dd0577801","expire
s_in":3600,"token_type":"Bearer","scope":"openid
email","refresh_token":"bf54356100050f968690016f16b6f0722a23b62b","
id_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjhMbD
h2UlU3UmxTc2ZlcXZrZGVueG5YZjlnRlNwUXk0b3dwaVY4dlRFbE0if
Q.eyJpc3MiOiJodHRwczpcL1wvZGV2LW15Lml0cy5hYy5pZCIsInN1Yi
I6IjJEMDJBMTVFLTJFQTEtNEU5RS04MzI3LTI3Q0U1QkZERTI5QiIsI
mF1ZCI6IkVDMTlEMDRDLUQ1QjktNDc5MS1CMjExLUQ3MEU3MT
UyOTdFQyIsImlhdCI6MTU4OTIxODMwNSwiZXhwIjoxNTg5MjIxOTA
1LCJhdXRoX3RpbWUiOjE1ODkyMTgzMDUsInVybjpvcGVuaWQ6cGF
yYW1zOmp3dDpjbGFpbTphdXRoX3JlcV9pZCI6IjQ5RTU3MDhELUJE
NDQtNDg1Qi04MTNELUVDQzNCMjFDN0YyQiJ9.YqtDdP4QlN2so21j
f0_6CDS4Hlwt459YKANaDSD8PI24CW8dTsJ52q6w7y_x86YRexPicJ0l
R5kOlIFHEm1WOf2CK_PmviwchSYZ5J3T-
xS8wA2G8EcC_odhjr6R3uszMwfbB1sBtU1PRfBWnpzAwqH0tDx8lv675
Fdq0H8-eLtghYHtOPkQYu_bUwXPEHI5dO8kwPGRm_7EQBl-
TVhNqQwJs8RUYlyVZsi8Esul5ESya7dExyDVN1axREp-
o1FGvgJRcuyIJb96YWQQ5Z_hjLZM_mFXsWxYwscd1-
y3tRVmu9vvLKLeRLkwpJRBi4gr28fUw2MML7hySMKhcdydXQ"}
```

Kode Sumber 5.10 Skenario 1 - *Response* Token (1)

```
POST /bc-authorize? HTTP/1.1
Host: dev-my.its.ac.id
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
OTc0ZTRiYzMtYTc0OC00MWMxLWE3NGQtN2I5NWNhMWU2NmM
wOmUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Content-Type: application/x-www-form-urlencoded

scope=openid email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-2020&requested_expiry=120
```

Kode Sumber 5.11 Skenario 2 - *Request* Autentikasi Menggunakan *Client Id* Yang Tidak Terdaftar

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.10.3
Date: Mon, 11 May 2020 17:01:30 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"invalid_client","error_description":"invalid client credentials,
unknown client, no client authentication included, or unsupported
authentication method"}
```

Kode Sumber 5.12 Skenario 2 – *Response* Autentikasi Berupa Pesan *Error* Dari Server Autorisasi

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=476404D4-E92C-41E2-9D02-
545AADC1490C
```

Kode Sumber 5.13 Skenario 3 - *Request* Token

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Mon, 11 May 2020 18:56:38 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"invalid_grant","error_description":"auth_req_id doesn't exist or is
invalid for the client"}
```

Kode Sumber 5.14 Skenario 3 – *Response* Token Berupa Pesan *Error* Dari Server Autorisasi

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=6ed95ace-7210-4325-874f-e835cdfb4280
```

Kode Sumber 5.15 Skenario 4 - *Request* Token

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Mon, 11 May 2020 19:17:29 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"invalid_grant","error_description":"auth_req_id doesn't exist or is
invalid for the client"}
```

Kode Sumber 5.16 Skenario 4 - *Response* Token Berupa Pesan *Error* Dari Server Autorisasi

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=900EC5EF-F313-48C6-871A-37C54B01626E
```

Kode Sumber 5.17 Skenario 5 - *Request* Token

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Tue, 12 May 2020 11:29:15 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"slow_down","error_description":"Polling interval is too fast."}
```

Kode Sumber 5.18 Skenario 5 – *Response* Token Berupa Pesan *Error* Dari Server Autorisasi

```
POST /bc-authorize HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

scope=openid%20email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=binding-message-yang-
panjang&requested_expiry=120
```

Kode Sumber 5.19 Skenario 6 – *Request* Autentikasi Dengan *Binding Message* Panjang

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Content-Type: application/json

{"error":"invalid_binding_message","error_description":"The binding
message is invalid."}
```

Kode Sumber 5.20 Skenario 6 – *Response* Autentikasi Berupa Pesan *Error* Dari Server Autorisasi

```
POST /bc-authorize HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

scope=openid%20email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-2020&requested_expiry=120
```

Kode Sumber 5.21 Skenario 7 – *Request* Autentikasi Tanpa *User Code*

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Fri, 19 Jun 2020 08:41:28 GMT
Content-Type: application/json

{"error":"missing_user_code","error_description":"User code is required
but was missing from the request."}
```

Kode Sumber 5.22 Skenario 7 – *Response* Autentikasi Berupa Pesan *Error* Dari Server Autorisasi

```
POST /bc-authorize HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

scope=openid%20email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-
2020&requested_expiry=120&user_code=4321
```

Kode Sumber 5.23 Skenario 8 – *Request* Autentikasi Dengan *User Code* Salah

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Fri, 19 Jun 2020 09:34:05 GMT
Content-Type: application/json

{"error":"invalid_user_code","error_description":"User code was invalid"}
```

Kode Sumber 5.24 Skenario 8 – *Response* Autentikasi Berupa Pesan *Error* Dari Server Autorisasi

5.2.3 Kasus Pengujian CIBA Menggunakan Mode Token *Ping*

Pada kasus ini, server autorisasi akan diujui dengan melakukan CIBA menggunakan mode token *ping*.

Tabel 5.5 Kasus Pengujian CIBA Menggunakan Mode Token *Ping*

ID	**UJ-003**
Nama Skenario	Pengujian CIBA Menggunakan Mode Token *Ping*
Tujuan Pengujian	Menguji server autorisasi untuk melayani klien menggunakan alur CIBA mode token *ping*.
Skenario 1	Menggunakan *non signed request*
Kondisi Awal	Klien sudah terdaftar, memiliki *client id* dan *client secret*
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. 2. Klien mengirim *request* autentikasi. 3. Klien mendapatkan *response* dari server autorisasi berupa *authentication request id*. 4. Klien menyusun parameter *request* pemberian hak akses menggunakan *authentication request id* yang didapatkan. 5. Klien mengirim *request* pemberian hak akses. 6. Klien menyusun parameter *request* token. 7. Klien mengirim *request* token. 8. Klien mendapatkan *response* dari server autorisasi berupa token.

Hasil yang diharapkan	Mendapatkan token dari server autorisasi.
Hasil yang diperoleh	Mendapatkan token dari server autorisasi.
Hasil Pengujian	Berhasil
Skenario 2	Menggunakan lebih dari satu *hint* saat melakukan *request* autentikasi.
Kondisi Awal	Klien sudah terdaftar, memiliki *client id* dan *client secret*.
Langkah Pengujian	1. Klien menyusun parameter *request* autentikasi. *Request* autentikasi diberi paremeter *hint* lebih dari satu. Yaitu *login hint* dan *login hint token*. 2. Klien mengirim *request* autentikasi.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena ditemukan parameter *hint* lebih dari satu.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena ditemukan parameter *hint* lebih dari satu.
Hasil Pengujian	Berhasil
Skenario 3	Memanggil token *endpoint* dengan *authentication request id* yang kedaluwarsa.
Kondisi Awal	Klien memiliki *authentication request id* yang kedaluwarsa.
Langkah Pengujian	1. Klien menyusun parameter *request* token. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Sever autorisasi memberi pesan *error* karena *authentication request id* kedaluwarsa.

Hasil yang diperoleh	Sever autorisasi memberi pesan *error* karena *authentication request id* kedaluwarsa.
Hasil Pengujian	Berhasil
Skenario 4	Memanggil token *endpoint* dengan kondisi pengguna belum memberi hak akses.
Kondisi Awal	Klien memiliki *authentication request id*.
Langkah Pengujian	1. Klien menyusun parameter *request* token. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena *authentication request id* belum diberi hak akses.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena *authentication request id* belum diberi hak akses.
Hasil Pengujian	Berhasil
Skenario 5	Memanggil token *endpoint* dengan kondisi pengguna tidak memberi hak akses.
Kondisi Awal	Klien memiliki *authentication request id* yang tidak diberi hak akses.
Langkah Pengujian	1. Klien menyusun parameter *request* token. 2. Klien mengirim *request* token.
Hasil yang diharapkan	Server autorisasi memberi pesan *error* karena klien tidak diberi hak akses.
Hasil yang diperoleh	Server autorisasi memberi pesan *error* karena klien tidak diberi hak akses.
Hasil Pengujian	Berhasil

Pada skenario pertama, *request* autentikasi yang disusun menggunakan aplikasi Postman dapat dilihat pada Kode Sumber

5.25. Klien sudah terdaftar menggunakan mode token *ping* dan dapat dilihat pada Gambar 5.3. Server autorisasi mengembalikan *response* yang terdapat *authentication request id*. *Authentication request id* ini akan valid selama 120 detik. *Response* ini dapat dilihat pada Kode Sumber 5.26. *Request* pemberian hak akses dapat dilihat pada Kode Sumber 5.27. *Request* token dapat dilihat pada Kode Sumber 5.28. *Response* token dapat dilihat pada Kode Sumber 5.29. Skenario ini berhasil dilakukan.

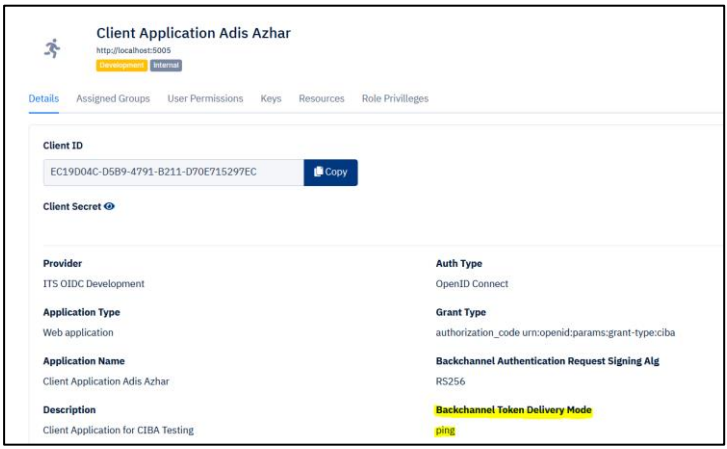
Pada skenario kedua, *request* autentikasi disusun menggunakan dua parameter *hint* yaitu *login hint* dan *login hint token*. Parameter *hint* berisi nilai identifikasi akun pengguna. *Request* autentikasi dan *response* autentikasi berupa pesan *error* dapat dilihat pada Kode Sumber 5.30 dan Kode Sumber 5.31. Skenario ini berhasil dilakukan.

Pada skenario ketiga, *request* token disusun menggunakan *authentication request id* yang didapat ketika melakukan *request* autentikasi pada skenario 1, yaitu *authentication request id* dengan nilai "0F65D708-AE91-4D48-A0C6-8CEC0802D42A". *Authentication request id* ini sudah kedaluwarsa karena saat dilakukan uji coba ini, sudah melewati 120 detik sejak *authentication request id* diterbitkan. Klien mengirim *request* token dan mendapatkan *response* berupa pesan *error* dari server autorisasi. *Request* dan *response* dapat dilihat pada Kode Sumber 5.32 dan Kode Sumber 5.33. Skenario ini berhasil dilakukan.

Pada skenario keempat, klien memiliki *authentication request id* dengan nilai "4DF7A626-D90B-4ABC-97F8-567CF41AAD9D". *Authentication request id* ini belum memilili hak akses karena klien belum membolehkan atau menolak hak akses. Ketika dikirim *request* token, server autorisasi mengembalikan *response* berupa pesan error. *Request* dan *response* dapat dilihat pada Kode Sumber 5.34 dan Kode Sumber 5.35. Skenario ini berhasil dilakukan.

Pada skenario terakhir, klien memiliki *authentication request id* dengan nilai "28C8531A-B248-4AED-A880-622D5CE87AF2". *Request* penolakan hak akses terhadap

authentication request id ini dikirim. Lalu klien mengirim *request* token dan mendapatkan pesan *error* dari server autorisasi. *Request* penolakan hak akses, token dan *response* token dapat dilihat pada Kode Sumber 5.36, Kode Sumber 5.37, dan Kode Sumber 5.38. Skenario ini berhasil dilakukan.



Gambar 5.3 Aplikas Klien Terdaftar Menggunakan Mode Token *Ping*

```
POST /bc-authorize HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VD
OmUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

scope=openid%20email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-2020&requested_expiry=120
```

Kode Sumber 5.25 Skenario 1 - *Request* Autentikasi

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Tue, 12 May 2020 12:09:32 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive

{"auth_req_id":"0F65D708-AE91-4D48-A0C6-
8CEC0802D42A","expires_in":1589285492}
```

Kode Sumber 5.26 Skenario 1 - *Response* Autentikasi

```
POST /ciba-consent HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: dev-my.its.ac.id

auth_req_id=0F65D708-AE91-4D48-A0C6-
8CEC0802D42A&user_consent=accept
```

Kode Sumber 5.27 Skenario 1 - *Request* Pemberian Hak Akses

```
POST /token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=0F65D708-AE91-4D48-A0C6-
8CEC0802D42A
```

Kode Sumber 5.28 Skenario 1 - *Request* Token

HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Tue, 12 May 2020 12:09:47 GMT
Content-Type: application/json
Cache-Control: no-store

{"access_token":"4718534d0979f450ce39d97c03ed901195c49e81","expires_in":3600,"token_type":"Bearer","scope":"openid email","refresh_token":"6cf5490808e7060b4f5c33b2064be10e3e9a626c",
"id_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjhMbDh2UlU3UmxTc2ZlcXZrZGVueG5YZjlnRlNwUXk0b3dwaVY4dlRFbE0ifQ.eyJpc3MiOiJodHRwczpcL1wvZGV2LW15Lml0cy5hYy5pZCIsInN1YiI6IjJEMDJBMTVFLTJFQTEtNEU5RS04MzI3LTI3Q0U1QkZERTI5QiIsImF1ZCI6IkVDMTlEMDRDLUQ1QjktNDc5MS1CMjExLUQ3MEU3MTUyOTdFQyIsImlhdCI6MTU4OTI4NTM3MiwiZXhwIjoxNTg5Mjg4OTcyLCJhdXRoX3RpbWUiOjE1ODkyODUzNzIsInVybjpvcGVuaWQ6cGFyYW1zOmp3dDpjbGFpbTphdXRoX3JlcV9pZCI6IjBGNjVENzA4LUFFOTEtNEQ0OC1BMEM2LThDRUMwODAyRDQyQSJ9.GCSv4Ydi9pgpm7HbpAl4w44HlnupA8-PADM2xAGCFwkxoviuSVkkhxgyMbqU2aBqVX9FJAwqw7H1mBlbtyK8P4lyltptAXmjRftVs7x7LbtONcf8x5n-oEAzLp9-GZjBgNTYwlbD64XFzPwW1wr5gsmIaaV-nV5xhbZGWNetmnQ3biYgY5v5W3YIgkQO__n9GrlTlW50eL-2Ne1HUa6_SN_aFWVCWSMzLs6TESenQARyVcabWu6uRDVaZW6up4048DYlO50NEbcY3iQbw36IsYjJUUU_d-uu7k2DYYKTSTT05LZdfhQnyaVigOfB8kBMEuXbLADuN3HDG0S6cOYyjQ"}

Kode Sumber 5.29 Skenario 1 - *Response* Token

```
POST /bc-authorize HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

scope=openid%20email&client_notification_token=77a554a6-9a4e-45b9-
8643-8f1aaca1a286&login_hint=2D02A15E-2EA1-4E9E-8327-
27CE5BFDE29B&binding_message=aa-
2020&requested_expiry=120&login_hint_token=2D02A15E-2EA1-4E9E-
8327-27CE5BFDE29B
```

Kode Sumber 5.30 Skenario 2- *Request* Autentikasi

```
HTTP/1.1 400 Bad Request
Server: nginx/1.10.3
Date: Tue, 12 May 2020 12:20:34 GMT
Content-Type: application/json

{"error":"invalid_request","error_description":"The request is missing a
required parameter, includes an invalid parameter value, includes a
parameter more than once, contains more than one of the hints, or is
otherwise malformed."}
```

Kode Sumber 5.31 Skenario 2 - *Response* Autentikasi

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=0F65D708-AE91-4D48-A0C6-
8CEC0802D42A
```

Kode Sumber 5.32 Skenario 3 - *Request* Token

```
HTTP/1.1 403 Forbidden
Server: nginx/1.10.3
Date: Tue, 12 May 2020 12:39:34 GMT
Content-Type: application/json

{"error":"expired_token","error_description":"The auth_req_id has
expired."}
```

Kode Sumber 5.33 Skenario 3 - *Response* Token

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VDO
mUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=4DF7A626-D90B-4ABC-97F8-
567CF41AAD9D
```

Kode Sumber 5.34 Skenario 4 - *Request* Token

```
HTTP/1.1 403 Forbidden
Server: nginx/1.10.3
Date: Tue, 12 May 2020 13:40:25 GMT
Content-Type: application/json

{"error":"authorization_pending","error_description":"The end-user hasn't
consented the authorization request."}
```

Kode Sumber 5.35 Skenario 4 - *Response* Token

```
POST /ciba-consent? HTTP/1.1
Host: dev-my.its.ac.id
Content-Type: application/x-www-form-urlencoded
Content-Type: application/x-www-form-urlencoded

auth_req_id=28C8531A-B248-4AED-A880-
622D5CE87AF2&user_consent=deny
```

Kode Sumber 5.36 Skenario 5 - *Request* Penolakan Hak Akses

```
POST /token? HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
RUMxOUQwNEMtRDVCOS00NzkxLUIyMTEtRDcwRTcxNTI5N0VD
OmUxZjI1MTA0ZGFjMjYxMjY5NmM4OGZmYzkzYWZjZTFj
Host: dev-my.its.ac.id

grant_type=urn%3Aopenid%3Aparams%3Agrant-
type%3Aciba&auth_req_id=28C8531A-B248-4AED-A880-
622D5CE87AF2
```

Kode Sumber 5.37 Skenario 5 - *Request* Token

```
HTTP/1.1 403 Forbidden
Server: nginx/1.10.3
Date: Tue, 12 May 2020 13:46:43 GMT
Content-Type: application/json
Cache-Control: no-store

{"error":"access_denied","error_description":"The end-user denied the
authorization request."}
```

Kode Sumber 5.38 Skenario 5 - *Response* Token

5.2.4 Kasus Pengujian Mendapatkan Protected Resource

Pada kasus ini, server autorisasi akan diuji dengan melayani klien yang membutuhkan *protected resource* atau data yang terproteksi dengan cara memeriksa *access token* yang disertakan pada *authorization header*. *Access token* ini hasil dari *request* CIBA yang berhasil.

Tabel 5.6 Kasus Pengujian Mendapatkan *Protected Resource*

ID	**UJ-004**
Nama Skenario	Pengujian Mendapatkan *Protected Resource*
Tujuan Pengujian	Menguji server autorisasi untuk melayani klien yang meminta *protected resource*
Skenario 1	Menggunakan *access token* yang tidak kedaluwarsa
Kondisi Awal	Memiliki *access token* dari *request* CIBA
Langkah Pengujian	1. Menyusun parameter *request user info.* 2. Mengirim *request user info.*
Hasil yang diharapkan	Mendapatkan data pengguna.
Hasil yang diperoleh	Mendapatkan data pengguna.
Skenario 2	Menggunakan *access token* yang kedaluwarsa

Kondisi Awal	Memiliki *access token* dari *request* CIBA yang kedaluwarsa.
Langkah Pengujian	1. Menyusun parameter *request user info.* 2. Mengirim *request user info.*
Hasil yang diharapkan	Tidak mendapatkan data pengguna.
Hasil yang diperoleh	Tidak mendapatkan data pengguna.

Pada skenario pertama, didapatkan *access token* dari *request* CIBA yang dapat dilihat pada Kode Sumber 5.39. *Access token* ini digunakan sebagai nilai *authorization header* pada *request user info*. *Request* ini dapat dilihat pada Kode Sumber 5.40. *Request* ini dikirim dan diapatkan *user info* yang dapat dilihat pada Kode Sumber 5.41.

Pada skenario terakhir, didapatkan *access token* dari *request* CIBA. *Access token* yang digunakan harus dalam status kedaluwarsa. Oleh karena itu, diambil *access token* dari Kasus Pengujian CIBA Menggunakan Mode Token *Ping* karena pada saat uji coba skenario ini, *access token* tersebut sudah kedaluwarsa. *Access token* dapat dilihat pada Kode Sumber 5.29. *Request* ini dapat dilihat pada Kode Sumber 5.42. *Request* ini dikirim dan diapatkan pesan *error* dari server autorisasi yang menyatakan bahwa *access token* tidak valid, dapat dilihat pada Kode Sumber 5.43.

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Fri, 15 May 2020 14:50:04 GMT
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{"access_token":"b213828cb2c57d48e5602baa009dfd15e9813ec5","expire
s_in":3600,"token_type":"Bearer","scope":"openid
email","refresh_token":"257fcae04b79191adbfaf1498ec0d078eeac6f2c","id
_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjhMbDh2
UlU3UmxTc2ZlcXZrZGVueG5YZjlnRlNwUXk0b3dwaVY4dlRFbE0ifQ.e
yJpc3MiOiJodHRwczpcL1wvZGV2LW15Lml0cy5hYy5pZCIsInN1YiI6Ij
JEMDJBMTVFLTJFQTEtNEU5RS04MzI3LTI3Q0U1QkZERTI5QiIsImF
1ZCI6IkVDMTlEMDRDLUQ1QjktNDc5MS1CMjExLUQ3MEU3MTUy
OTdFQyIsImlhdCI6MTU4OTU1NDE2MiwiZXhwIjoxNTg5NTU3NzYyL
CJhdXRoX3RpbWUiOjE1ODk1NTQxNjIsInVybjpvcGVuaWQ6cGFyYW
1zOmp3dDpjbGFpbTphdXRoX3JlcV9pZCI6IkE2QzFFN0U2LTdDODktN
DM4MC05NzI0LUZFNDYxRDJBOTAwRSJ9.rxj2baZo-
pPBWhpCniei_K7QDPu7B1ssDDNAT7sW13txflbt9w8DnpwBYge28pk9
_i8_j06xos01gIYC7f6pNzvulyfeztPk4EkGLF0-
FSQLOnG6zqB28Om1WjOW-
i0p2adpPVWNl2OE58V6pJcR7kVZO_dJgBddoLrAjw0rW8p9ffUjDMVq
8D1DewLvE_9gWmpDnhFnYhsxg1Ui8RTnIzmZnUHH52PqvmJYzY1O
Bih_1cC96OpbE9RpzW0NU5zFWtmbf3D2wJd_EUwgOvL6fTbsVLXfT
VSLPEw5V9w--8cz-
h0ukTxk_YIeFulEPeMDJXzV3qpEDouAQs76N0lAIg"}
```

Kode Sumber 5.39 Skenario 1 - *Access Token* CIBA

```
POST /userinfo?schema=openid HTTP/1.1
Authorization: Bearer b213828cb2c57d48e5602baa009dfd15e9813ec5
Host: dev-my.its.ac.id
Content-Length: 0
```

Kode Sumber 5.40 Skenario 1 - *Request User Info*

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Fri, 15 May 2020 15:03:35 GMT
Content-Type: application/json
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache

{"email":"myemail@its.ac.id","email_verified":"0","alternate_email":"mya
lternateemail@example.com","alternate_email_verified":"1","sub":"2D02
A15E-2EA1-4E9E-8327-27CE5BFDE29B"}
```

Kode Sumber 5.41 Skenario 1 - *Response User Info*

```
POST /userinfo?schema=openid HTTP/1.1
Authorization: Bearer 4718534d0979f450ce39d97c03ed901195c49e81
Host: dev-my.its.ac.id
```

Kode Sumber 5.42 Skenario 2 - *Request User Info*

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.10.3
Date: Fri, 15 May 2020 15:10:54 GMT
Content-Type: application/json
WWW-Authenticate: Bearer realm="Service", error="invalid_token",
error_description="The access token provided is invalid"

{"error":"invalid_token","error_description":"The access token provided is
invalid"}
```

Kode Sumber 5.43 Skenario 2 - *Response User Info*

5.2.5 Kasus Pengujian Autentikasi Berperan Sebagai Akun Yang Berbeda

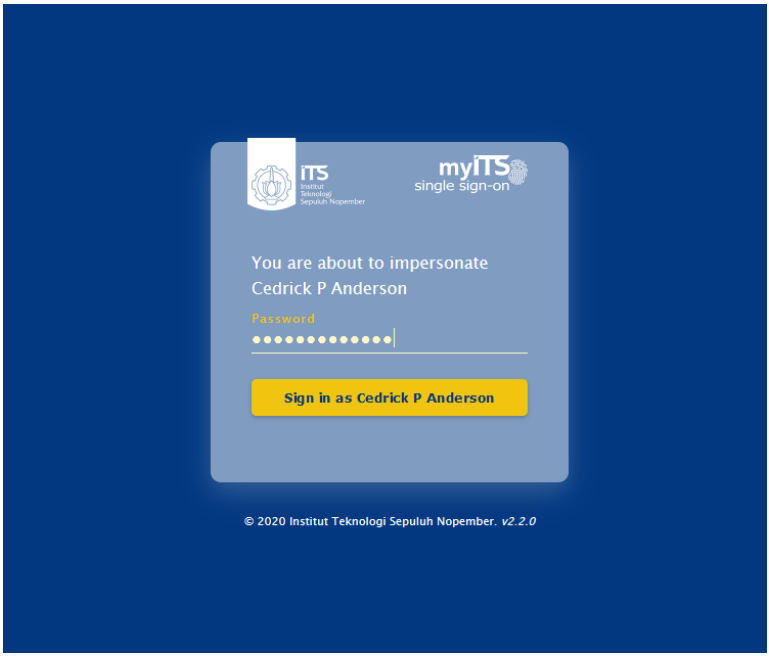
Pada kasus ini, server autorisasi akan diujui dengan melakukan autentikasi berperan sebagai akun yang berbeda. Uji coba ini tidak dilakukan menggunakan aplikasi Postman tetapi menggunakan *browser*. Pendekatan menggunakan *browser* dipilih

karena prosedur ini terdapat *redirect* antar halaman, sehingga lebih mudah dilakukan melalui *browser*.

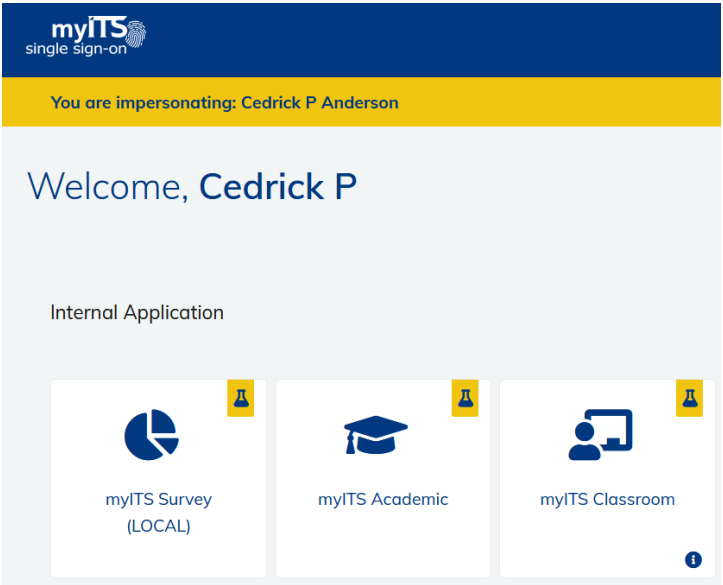
Pada uji coba ini, pengguna memasukkan kata sandi akun asli yang dimilikinya. Halaman ini dapat dilihat pada Gambar 5.4. Ketika pengguna memilih tombol "*Sign in as*" server autorisasi akan memproses permintaan tersebut. Lalu saat sudah selesai memproses, pengguna dialihkan ke halaman dashboard dengan sesi baru. Halaman ini dapat dilihat pada Gambar 5.5.

Tabel 5.7 Kasus Pengujian Autentikasi Berperan Sebagai Akun Yang Berbeda

ID	**UJ-005**
Nama Skenario	Pengujian Autentikasi Berperan Sebagai Akun Yang Berbeda
Tujuan Pengujian	Menguji server autorisasi untuk melayani autentikasi berperan sebagai akun yang berbeda
Kondisi Awal	Pengguna sudah terautentikasi dan memilih akun yang ingin diperan.
Langkah Pengujian	1. Pengguna memasukkan password akun asli. 2. Pengguna memilih tombol "*Sign in as*".
Hasil yang diharapkan	Server autorisasi membuat sesi baru dengan akun pengguna yang berbeda.
Hasil yang diperoleh	Server autorisasi membuat sesi baru dengan akun pengguna yang berbeda.
Hasil Pengujian	Berhasil



Gambar 5.4 Halaman Formulir Kata Sandi



Gambar 5.5 Halaman *Dashboard* Dengan Sesi Baru

5.3 Evaluasi

Pada subbab ini dijelaskan hasil dari pengujian pada subbab sebelumnya.

5.3.1 Evaluasi Fungsionalitas Sistem

Evaluasi ini adalah hasil dari pengujian kasus uji pada subbab 5.2. Hasil dinyatakan dalam status terpenuhi atau tidak. Hasil tersebut ditunjukkan pada Tabel 5.8.

Hasil evaluasi yang ditunjukkan pada Tabel 5.8 terpenuhi semua sesuai skenario uji coba. Ketika ada kondisi yang ganjil, sistem akan menampilkan pesan dalam bentuk *error* sehingga pengembang sistem dapat mengetahui alasan terjadinya *error* tersebut, seperti pada skenario "Memanggil token *endpoint* dengan *authentication request id* yang kedaluwarsa" pada uji coba Kasus Pengujian CIBA Menggunakan Mode Token *Ping*.

Tabel 5.8 Evaluasi Kasus Pengujian Fungsionalitas Sistem

No.	Kode Kasus Pengujian	Skenario		Terpenuhi
1	UJ-001	1	✓	YA
		2	✓	
2	UJ-002	1	✓	YA
		2	✓	
		3	✓	
		4	✓	
		5	✓	
		6	✓	
		7	✓	
		8	✓	
3	UJ-003	1	✓	YA
		2	✓	
		3	✓	
		4	✓	
		5	✓	
4	UJ-004	1	✓	YA
		2	✓	
5	UJ-005			YA

5.3.2 Evaluasi Implementasi Sistem

Evaluasi ini membahas tantangan yang dihadapi saat melakukan eksplorasi, implementasi server autorisasi CIBA dan sinkronisasi dengan klien.

Langkah implementasi CIBA dimulai dengan menentukan bentuk *deployment* sistem. Yaitu mekanisme agar sistem dapat berjalan pada lingkungan *development* sampai dengan *production*. Mekanisme *deployment* pada lingkungan *development* tidak berbeda jauh dengan lingkungan *production*. Yaitu dengan menggunakan perangkat lunak *web server* Nginx, basis data Microsoft SQL Server dan Redis.

Pada fase implementasi CIBA pada lingkungan *development*, sistem CIBA berjalan pada perangkat keras laptop. Pada

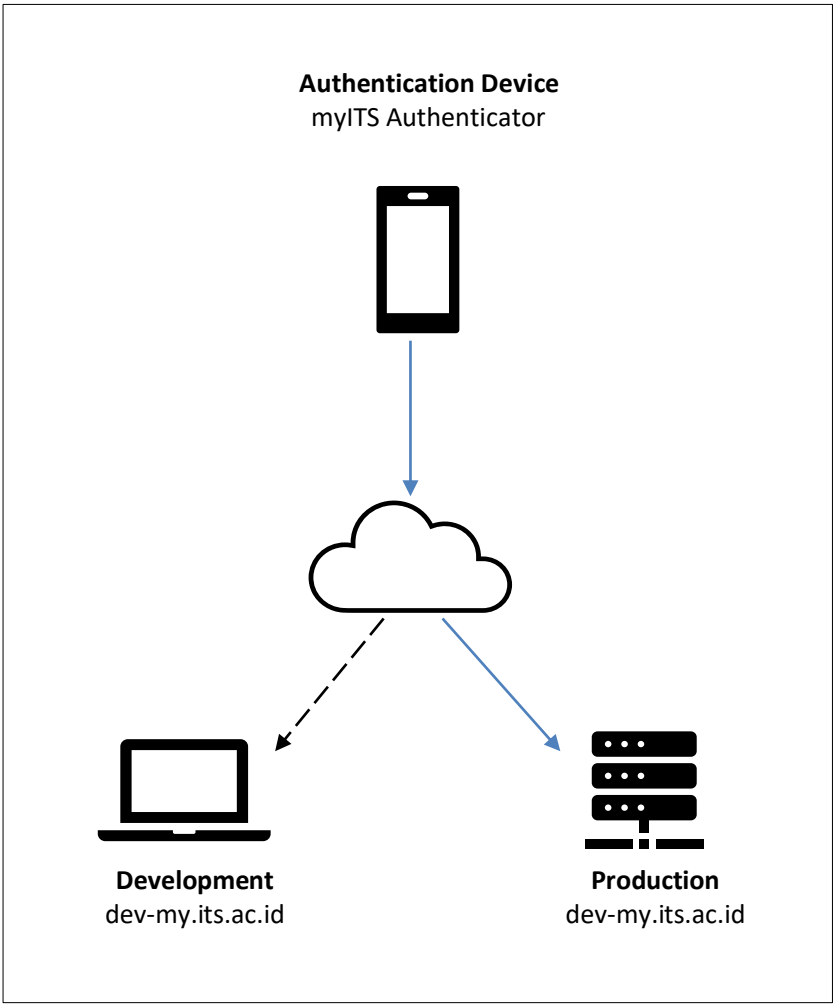
umumnya, sistem yang bertindak sebagai server dapat diakses menggunakan *Internet Protocol Address* (*IP Address*) sebagai alamat perangkat. Sistem CIBA pada lingkungan *development* memanfaatkan fitur *local Domain Name System* (DNS). DNS menghubungkan *Uniform Resource Locator* (URL) dengan *IP Address,* sehingga sistem CIBA dapat diakses menggunakan nama *domain*.

Nama *domain* server CIBA adalah "*dev-my.its.ac.id"* dan digunakan pada lingkungan *development* dan *production*. Pendekatan ini digunakan dengan mempertimbangkan waktu yang dibutuhkan untuk melakukan konfigurasi ulang ketika berpindah lingkungan. Dengan menggunakan nama *domain* yang sama saat *development* dan *production,* maka dapat menghindari waktu yang terbuang untuk mengubah URL pada sistem server dan klien. Sistem URL ini adalah alamat yang digunakan untuk berkomunikasi dengan server, sehingga apabila URL yang dituju salah, maka proses komunikasi tidak dapat dimulai.

Perbedaan yang signifikan pada lingkungan *development* dan *production* adalah pembatasan komunikasi yang dimilikinya. Pada lingkungan *development,* server CIBA hanya dapat diakses melalui perangkat keras laptop yang menjalankan server. Namun pada lingkungan *production,* siapa pun dapat mengakses karena terbuka menggunakan *public IP Address.*

Tantangan muncul saat ingin menghubungkan server autorisasi dengan *Authentication Device* (AD), yang merupakan aplikasi Android. Proses bisnis ini adalah saat pengguna pada AD ingin memberi hak akses. Dapat dilihat pada Gambar 5.6, AD menggunakan nama *domain* "*dev-my.its.ac.id"* untuk dapat berkomunikasi dengan server CIBA. Namun ketika AD mengirim *request* HTTP ke *domain* ini, ternyata menuju ke server CIBA pada lingkungan *production.* Ini disebabkan oleh kemampuan AD yang hanya dapat berkomunikasi dengan *public IP Address*. Sehingga, apabila ada perubahan kode sumber pada lingkungan *development* yang dapat mempengaruhi komunikasi AD dan server autorisasi, maka perubahan ini harus disimpan

menggunakan *version control git*. Ini menjadi proses yang rumit karena terjadi secara repetitif. Selain itu, proses *debugging* menjadi sulit karena pada lingkungan *production*, server hanya dapat diakses melalui terminal, sehingga apabila ingin melihat *error* diharuskan untuk membuka *file log*.



Gambar 5.6 Skema Komunikasi *Authentication Device* Dengan Server Autorisasi

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan yang diperoleh selama pengerjaan tugas akhir ini berdasarkan hasil pengujian dan hal lainnya yang telah dilakukan. Selain itu, juga terdapat beberapa saran terhadap tugas akhir ini untuk pengembangan kedepannya.

6.1 Kesimpulan

Berdasarkan pengerjaan tugas akhir ini dan hasil yang didapatkan pada tahap uji coba aplikasi, dapat diambil beberapa kesimpulan sebagai berikut.

1. Protokol CIBA server yang dibangun pada tugas akhir ini dapat mengakomodir proses bisnis yang terdiri dari pendaftaran aplikasi klien, alur autentikasi *backchannel* pada myITS SSO, pengambilan *access token* serta *session binding* antar perangkat.
2. Protokol CIBA server yang dibangun pada tugas akhir ini telah memenuhi spesifikasi protokol CIBA dengan melayani permintaan autentikasi dengan parameter yang sesuai, serta tidak melanjutkan permintaan autentikasi jika parameter tidak sesuai yang didukung oleh pesan *error* yang deskriptif.
3. *Library* CIBA server dapat digunakan pada server autorisasi myITS SSO tanpa menghambat protokol yang sudah tertanam, dengan memanfaatkan prinsip pemrograman obyek SOLID. Salah satu protokol diantaranya adalah *Authorization Code*.
4. *Test case* yang diambil dari *conformance testing* FAPI CIBA dapat diaplikasikan pada protokol CIBA karena berasal dari fondasi yang sama.

6.2 Saran

Terdapat beberapa saran terkait tugas akhir ini yang diharapkan membuat tugas akhir ini menjadi lebih baik. Saran-saran tersebut antara lain:

1. Penambahaan kode sumber baru pada *library* server autorisasi CIBA sehingga library versi baru dapat digunakan untuk mendukung protokol *Financial Grade API Client Initiated Backchannel Authentication* (FAPI CIBA) agar dapat memaksimalkan pengamanan jika berurusan dengan data finansial.
2. Dapat menangani pendaftaran aplikasi klien secara dinamis agar dapat memudahkan klien untuk memulai menggunakan CIBA.
3. Penambahan *unit testing* pada kode sumber *library* agar dapat mengetahui *breaking changes* setiap saat ada perubahan kode sumber.

GLOSARIUM

Authentication Device	Perangkat yang digunakan untuk melakukan autentikasi dan autorisasi terhadap permintaan akses data yang terproteksi oleh RP. Pada umumnya, perangkat ini berupa *smartphone* dan dipegang oleh pengguna.
Claim	Properti yang terdapat pada *JSON Web Token* yang berupa suatu fakta.
Consumption Device	Perangkat yang digunakan untuk mengoperasikan dan menjalankan aplikasi klien. Dapat dalam bentuk *web browser*.
Endpoint	Suatu alamat yang bisa diakses oleh aplikasi klien.
Grant Type	Metode pemberian akses ke resources (sumber data) yang dilindungi dengan berbagai cara dan keamanan data yang berbeda. Contoh *grant type*: CIBA, *Authorization Code*, dan *Client Credentials.* Kata *grant type*, protokol, dan alur dapat saling ditukar.
HTTP	Singkatan dari *Hyper Text Transfer Protocol* yang merupakan protokol untuk bertukar data.
JSON Web Token	Suatu format data yang melalui proses penyusunan menggunakan *digital signing* sehingga terstruktur dan originalitas data tetap terjamin.
OAuth 2.0	Kerangka kerja yang beperan sebagai lapisan autorisasi untuk memberi akses data yang terproteksi ke aplikasi klien eksternal

OpenID Connect	Lapisan autentikasi yang dibangun menggunakan fondasi OAuth 2.0.
Relying Party	Aplikasi klien yang menggunakan layanan CIBA. Kata *Relying Party* (RP) dan aplikasi klien dapat digunakan secara bergantian karena memiliki makna yang sama.
Scope	Mekanisme untuk memberi batas hak akses terhadap aplikasi klien.
Server Autorisasi	Server yang mengimplementasi OIDC dan CIBA. Kata *OpenID Provider* (OP) dan server autorisasi dapat digunakan secara bergantian karena memiliki makna yang sama.
URI	Sebuah untaian karakter yang digunakan untuk mengidentifikasi suatu layanan di internet.

DAFTAR PUSTAKA

[1] N. Sakimura, N. J. Bradley, P. Identity, M. Jones, M. B. d. Medeiros, G. C. Mortimore and S. , "OpenID Connect Core 1.0," OpenID Connect, 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html. [Accessed 02 Juli 2019].

[2] K. Kasey, "Understanding OAuth 2.0 and OpenID Connect," 26 April 2018. [Online]. Available: https://blog.runscope.com/posts/understanding-oauth-2-and-openid-connect. [Accessed Maret 2020].

[3] G. Fernandez, Telefonica, F. Walter, A. Nennker, D. T. AG, D. Tonge, Moneyhub, B. Campbell and P. Identity, "OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0 draft-03," 22 Januari 2020. [Online]. Available: https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html. [Accessed 3 April 2020].

[4] K. W. Dwiastini, Implementasi Otentikasi SSO: Single Sign On dan Otorisasi Role Based Access Control Menggunakan Standar OpenID Connect, Surabaya, 2018.

[5] M. Anicas, "An Introduction to OAuth 2," Digitalocean, 2014.

[6] D. H. Ed. and Microsoft, "The OAuth 2.0 Authorization Framework," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed 8 Mei 2020].

[7] D. H. Ed. and Microsoft, "Section 1.1. Roles: The OAuth 2.0 Authorization Framework," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749#section-1.1. [Accessed 8 Mei 2020].

[8] H. D. Ed. and Microsoft, "Section 1.3. Authorization Grant: The OAuth 2.0 Authorization Framework," 2012.

[Online]. Available: https://tools.ietf.org/html/rfc6749#section-1.3. [Accessed 8 Mei 2020].

[9] N. Sakimura, NRI, J. Bradley, P. Identity, M. Jones, Microsoft, B. d. Medeiros, Google, C. Mortimore and Salesforce, "OpenID Connect Core 1.0 ID Token," 8 November 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html#IDToken. [Accessed 15 Maret 2020].

[10] Connect2id, "OpenID Connect explained - Cool ID Token Uses," [Online]. Available: https://connect2id.com/learn/openid-connect#cool-id-token-uses. [Accessed 01 Maret 2020].

[11] Connect2id, "Identity and access token impersonation in Connect2id server 4.0," Connect2id, 12 Februari 2017. [Online]. Available: https://connect2id.com/blog/connect2id-server-4-0#impersonation. [Accessed 1 Maret 2020].

[12] G. F. Rodriguez, F. Walter, A. Nennker, D. Tonge and B. Campbell, "OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0 draft-02," 16 Januari 2016. [Online]. Available: https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html. [Accessed 10 November 2019].

[13] G. Fernandez, Telefonica, F. Walter, A. Nennker, D. T. AG, D. Tonge, Moneyhub, B. Campbell and P. Identity, "OpenID Connect Client Initiated Backchannel Authentication Flow Terminology," 22 Januari 2020. [Online]. Available: https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html#rfc.section.2. [Accessed 15 Maret 2020].

[14] G. Fernandex, Telefonica, F. Walter, A. Nennker, D. T. AG, D. Tonge, Moneyhub, B. Campbell and P. Identity, "Poll, Ping and Push Modes," 22 Januari 2020. [Online].

Available: https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html#rfc.section.5. [Accessed 01 Maret 2020].

[15] S. Peyrott, "JWT Handbook: Introduction," in *JWT Handbook*, Auth0 Inc., 2018, p. 5.

[16] Auth0, "Introduction to JSON Web Tokens - What is the JSON Web Token structure?," Auth0. [Online]. [Accessed 15 Maret 2020].

[17] M. Jones, Microsoft, J. Bradley, P. Identity, N. Sakimura and NRI, "JSON Web Token (JWT) - Registered Claim Names," Mei 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519#section-4.1. [Accessed 01 Maret 2020].

[18] "JWT Handbook: JSON Web Tokens in Detail," in *JWT Handbook*, Auth0 Inc., 2018, p. 23.

[19] S. Peyrott, "JWT Handbook: JSON Web Tokens in Detail - The Header, The Payload, JSON Web Signatures," in *JWT Handbook*, Auth0 Inc., 2018, pp. 24-25, 30.

[20] Auth0, "JWT Debugger," [Online]. Available: https://jwt.io/. [Accessed 2020 Maret 22].

[21] J. Bradley, B. Capmbell, M. B. Jones and C. Mortimore, "JSON Web Token (JWT) - IANA Registry," 02 Maret 2020. [Online]. Available: https://www.iana.org/assignments/jwt/jwt.xhtml. [Accessed 22 Maret 2020].

[22] K. Wilson, "SOLID Design Principles," in *The Clean Architecture In PHP*, 2015, pp. 31-59.

[23] M. Rouse, C. Bedell, E. Hannan and S. Wilson, "RESTful API (REST API)," Tech Target, Juni 2019. [Online]. Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API. [Accessed 22 Maret 2020].

[24] R. Fielding, U. Irvine, C. W. J. Mogul, C. H. Frystyk, W. M. L. Masinter, X. P. Leach, M. T. Berners-Lee and W3C/MIT, "Hypertext Transfer Protocol -- HTTP/1.1," Juni 1999. [Online]. Available: https://www.ietf.org/rfc/rfc2616.txt. [Accessed 22 Maret 2020].

[25] "Post HTTP Request Online," Reqbin, [Online]. Available: https://reqbin.com/#pills-req-raw. [Accessed 22 Maret 2020].

[26] I. Ben-Gan, D. Sarka and R. Wolter, Inside Microsoft SQL Server 2005: T-SQL Programming, Microsoft Press, 2006.

[27] "System Requirements for the Microsoft Drivers for PHP for SQL Server," Microsoft, 1 Januari 2020. [Online]. Available: https://docs.microsoft.com/en-us/sql/connect/php/system-requirements-for-the-php-sql-driver?view=sql-server-ver15. [Accessed 27 April 2020].

[28] A. Andra, Implementasi Protokol Client Initiated Backchannel Authentication (CIBA) di Sisi Klien pada MyITS Single Sign-On, Surabaya, 2020.

[29] D. Tonge, Moneyhub, J. Heenan, Authlete, T. Lodderstedt, Yes, B. Campbell and Ping Identity, "Financial-grade API: Client Initiated Backchannel Authentication Profile," 15 Agustus 2019. [Online]. Available: https://openid.net/specs/openid-financial-api-ciba-ID1.html. [Accessed 19 Mei 2020].

[30] Auth0, "Introduction to JSON Web Tokens - When should you use JSON Web Tokens?," Auth0, [Online]. Available: https://jwt.io/introduction/. [Accessed 15 Maret 2020].

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Penulis, Muhammad Adistya Azhar, lahir di Trenggalek, 17 Juli 1999. Penulis menyelesaikan pendidikan dasar di Colonel Light Gardens Primary School South Australia (2005-2012), kemudian untuk pendidikan menengah pertama di Pasadena High School South Australia (2012-2014), dan jenjang menengah atas di SMAN 5 Surabaya (2014-2016). Setelah tamat sekolah menengah atas, penulis melanjutkan pendidikan sarjana di departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS).

Selama berada di dunia akademi kampus, penulis aktif sebagai asisten dosen pemrograman berorientasi obyek, pemrograman berbasis kerangka kerja, dan arsitektur perangkat lunak. Selain itu, penulis menekuni unit kegiatan mahasiswa ITS Debate Society mewakili ITS dalam berbagai kompetisi debat bahasa Inggris tingkat provinsi dan nasional, *runner up* Ma-Chung University Debate Competition 2018, *top speakers* STTAL Debate Competition 2018, *top speakers* National University Debating Championship Kopertis VII 2018, dan administrator laboratorium Rekayasa Perangkat Lunak. Penulis juga merupakan penerima beasiswa unggulan Bank Indonesia tahun 2018-2019 dan 2019-2020. Penulis memiliki pengalaman di *tech industry* sebagai *software engineer* di perusahaan Suitmedia dan Tokopedia (Developer Camp) pada tahun 2019.

Penulis dalam menyelesaikan Pendidikan sarjana mengambil rumpun mata kuliah Rekayasa Perangkat Lunak serta memiliki minta di bidang arsitektur dan desain perangkat lunak, sistem terdistribusi, dan komputasi awan. Untuk komunikasi, penulis dapat dihubungi melalui surel: aa@adisazhar.com.