**PEMBANGUNAN *SINGLE SIGN-ON* UNTUK AUTENTIKASI PENGGUNA BERBASIS TOKEN**

**Laporan Tugas Akhir**

**Disusun sebagai syarat kelulusan tingkat sarjana**

**Oleh**

**Luqman Arifin Siswanto**

**NIM: 13513024**



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO & INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2018**

**PEMBANGUNAN *SINGLE SIGN-ON* UNTUK AUTENTIKASI PENGGUNA BERBASIS TOKEN**

**Laporan Tugas Akhir**

**Oleh**

**Luqman Arifin Siswanto**

**NIM: 13513024**

**Program Studi Teknik Informatika**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung, 1 Oktober 2018

Mengetahui,

Pembimbing,

Adi Mulyanto, S.T., M.T.

NIP. 19631126 198803 1 002

LEMBAR PERNYATAAN

Dengan ini saya menyatakan bahwa:

1. Pengerjaan dan penulisan Laporan Tugas Akhir ini dilakukan tanpa menggunakan bantuan yang tidak dibenarkan.
2. Segala bentuk kutipan dan acuan terhadap tulisan orang lain yang digunakan di dalam penyusunan laporan tugas akhir ini telah dituliskan dengan baik dan benar.
3. Laporan Tugas Akhir ini belum pernah diajukan pada program pendidikan di perguruan tinggi mana pun.

Jika terbukti melanggar hal-hal di atas, saya bersedia dikenakan sanksi sesuai dengan Peraturan Akademik dan Kemahasiswaan Institut Teknologi Bandung bagian Penegakan Norma Akademik dan Kemahasiswaan khususnya Pasal 2.1 dan Pasal 2.2.

Bandung, 1 Oktober 2018

Luqman Arifin Siswanto

NIM 13513024

ABSTRAK

**PEMBANGUNAN *SINGLE SIGN-ON* UNTUK AUTENTIKASI PENGGUNA BERBASIS TOKEN**

Oleh

LUQMAN ARIFIN SISWANTO

NIM: 13513024

Dengan bergesernya tren pengembangan aplikasi menuju ke arah berbasis internet, kini banyak organisasi memiliki lebih dari satu aplikasi web. Setiap aplikasi umumnya membutuhkan autentikasi user. Namun, tidak efisien apabila tiap aplikasi dalam satu organisasi yang sama memiliki mekanisme autentikasi sendiri-sendiri. *Single sign-on* adalah solusi dari permasalahan autentikasi multi-aplikasi. Berdasarkan analisis, tidak semua kakas *single sign-on* dengan mudah diintegrasi dengan aplikasi tersebut. Beberapa masalah dikaji yang kemudian menghasilkan karakteristik *single sign-on* yang baru.

Dalam tugas akhir ini, dikembangkan sebuah *single sign-on* yang dinamai Minisso, yang terdiri dari *server* Minisso dan *dashboard* Minisso. *Server* bertugas sebagai otak utama sementara *dashboard* berfungsi sebagai pelengkap untuk keperluan manajemen. Minisso diimplementasi sebagai pengembangan dari kakas *single sign-on* yang sudah ada sebelumnya.

Minisso diuji menggunakan skenario uji. Skenario uji tersebut meliputi pengujian fungsionalitas dan pengujian terintegrasi. Hasil pengujian menunjukkan bahwa Minisso berhasil dikembangkan dan dapat diintegrasikan dengan aplikasi yang sudah ada maupun aplikasi yang baru dikembangkan, tanpa memerdulikan bahasa dan teknologi yang digunakan dalam aplikasi (*language-agnostic*). Tahapan penerapan Minisso pada aplikasi dijelaskan pada tugas akhir ini.

Kata kunci: autentikasi, Minisso, *single sign-on,* token

KATA PENGANTAR

Segala puji syukur penulis panjatkan kepada Allah SWT, karena atas rahmat dan hidayah-Nya, laporan tugas akhir yang berjudul “Pembangunan *Single Sign-on* untuk Autentikasi Pengguna Berbasis Token” ini dapat diselesaikan dengan baik. Penulis menyadari bahwa proses pelaksanaan tugas akhir ini memiliki banyak kendala dan tantangan. Namun, berkat bantuan, bimbingan, dan kerjasama dari berbagai pihak, tantangan tersebut dapat dihadapi dengan baik. Oleh karena itu, penulis mengucapkan terima kasih, terutama kepada:

1. Bapak Adi Mulyanto, S.T., M.T. selaku pembimbing penulis atas ilmu yang diberikan selama bimbingan, kritik, dan saran yang diberikan selama proses pengerjaan tugas akhir, serta arahan yang diberikan selama penulis menempuh pendidikan di Teknik Informatika.
2. Bapak Dr.tech. Wikan Danar Sunindyo S.T., M.Sc. dan Bapak Satrio Adi Rukmono, S.T., M.T., selaku dosen penguji atas kritik dan saran membangun yang diberikan dalam seminar dan sidang tugas akhir.
3. Ibu Dr. Fazat Nur Azizah, S.T., M.Sc. dan Ibu Dessi Puji Lestari, S.T., M.Sc., Ph.D., selaku dosen koordinator Tugas Akhir, untuk arahan dan bimbingan selama pengerjaan tugas akhir.
4. Bapak Dr.techn. Saiful Akbar, S.T., M.T., selaku Ketua Program Studi Teknik Informatika ITB, atas dukungan selama pengerjaan tugas akhir maupun selama proses perkuliahan di Teknik Informatika.
5. Seluruh Bapak dan Ibu dosen pengajar Teknik Informatika ITB atas ilmu dan pelajaran yang telah diberikan sehingga tugas akhir ini dapat diwujudkan.
6. Orang tua dan keluarga penulis, yang telah memberikan semangat, dukungan, dan doa sepanjang hidup penulis, khususnya selama perkuliahan dan tugas akhir.
7. Niki Emersan, partner paling suportif sekaligus *proofreader* laporan tugas akhir, terima kasih banyak atas diskusi dan dukungan mental selama pengerjaan tugas akhir ini.
8. Albertus Angga Raharja atas ilmu React-nya yang sangat berguna dalam implementasi tugas akhir.
9. Anggota tim *competitive programming* Ainge: Wiwit Rifa’i, Alfonsus Raditya Arsadjaja, Aufar Gilbran, dan Muhamad Visat Sutarno, di mana penulis belajar konsistensi dan perverensi selama kuliah. Terima kasih atas pengalaman yang berharga hingga ke Singapura, Manila, Hong Kong, dan Beijing.
10. Sekeloagakure: Gazandi Cahyadarma, Afrizal Fikri, Muhammad Azam Iszuhri, atas diskusi faedah dan nirfaedah di berbagai restoran di Bandung.
11. *Men of Culture*: Cliff Jonathan, Calvin Aditya Jonathan, Ben Lemuel Tanasale, Juan Anton, dan Budi Pangestu, teman-teman penulis selama *part-time* di Bukalapak, terima kasih atas selingan NSFW-nya selama jam kerja.
12. Pihak-pihak lain yang tidak dapat disebutkan satu per satu, baik yang akrab, yang asing, akrab yang menjadi asing, maupun asing yang tak disangka menjadi akrab. Semua memberi penulis pengalaman, pelajaran, dan inspirasi. Terima kasih.

Perjalanan panjang telah penulis lalui hingga mencapai tahap ini dengan berbagai rintangan yang tak sesuai harapan. Akhir kata, semoga penulisan laporan tugas akhir ini berguna bagi penelitian-penelitian terkait berikutnya. Penulis terbuka terhadap berbagai masukan yang membangun apabila terdapat kekurangan dalam penulisan laporan tugas akhir ini.

Bandung, Oktober 2018

Penulis

DAFTAR ISI

[LEMBAR PERNYATAAN iii](#_Toc526198600)

[ABSTRAK iv](#_Toc526198601)

[KATA PENGANTAR v](#_Toc526198602)

[DAFTAR ISI vii](#_Toc526198603)

[DAFTAR GAMBAR xi](#_Toc526198604)

[DAFTAR TABEL xii](#_Toc526198605)

[BAB I PENDAHULUAN 1](#_Toc526198606)

[I.1 Latar Belakang 1](#_Toc526198607)

[I.2 Rumusan Masalah 3](#_Toc526198608)

[I.3 Tujuan 4](#_Toc526198609)

[I.4 Batasan Masalah 4](#_Toc526198610)

[I.5 Metodologi 4](#_Toc526198611)

[I.6 Sistematika Pembahasan 5](#_Toc526198612)

[BAB II STUDI LITERATUR 7](#_Toc526198613)

[II.1 Autentikasi dan Autorisasi 7](#_Toc526198614)

[II.2 Sistem Autentikasi Identitas 8](#_Toc526198615)

[II.2.1 *Password* 8](#_Toc526198616)

[II.2.2 *Token* 9](#_Toc526198617)

[II.2.3 *Biometrics* 11](#_Toc526198618)

[II.2.4 Tanda Tangan Digital 11](#_Toc526198619)

[II.2.5 Sertifikat Digital 11](#_Toc526198620)

[II.3 *Multi-Factor Authentication* 12](#_Toc526198621)

[II.4 *Single Sign-On* 12](#_Toc526198622)

[II.5 *Web Single Sign*-*On* 16](#_Toc526198623)

[II.5.1 *Auth0* 16](#_Toc526198624)

[II.5.2 WSO2 *Identity Server* 19](#_Toc526198625)

[II.5.3 *Central Authentication Service (CAS)* 23](#_Toc526198626)

[II.5.4 *Java Open Single Sign-On (JOSSO)* 25](#_Toc526198627)

[II.6 *Single Sign-On* Berbasis Token 26](#_Toc526198628)

[II.6.1 *Cookies* pada Web Browser 28](#_Toc526198629)

[II.6.2 *Cross-Origin Resource Sharing (CORS)* 28](#_Toc526198630)

[II.7 Pekerjaan Terkait 29](#_Toc526198631)

[II.7.1 *CAS++* 29](#_Toc526198632)

[II.7.2 Kristanto, 2011 30](#_Toc526198633)

[BAB III ANALISIS PERMASALAHAN DAN RANCANGAN SOLUSI 32](#_Toc526198634)

[III.1 Analisis Permasalahan 32](#_Toc526198635)

[III.1.1 Auth0 32](#_Toc526198636)

[III.1.2 WSO2 *Identity Server* 33](#_Toc526198637)

[III.1.3 *Central Authentication Server* 34](#_Toc526198638)

[III.1.4 Perbandingan *Web Single Sign-On* 35](#_Toc526198639)

[III.2 Analisis Solusi 36](#_Toc526198640)

[III.2.1 Karakteristik *Web Single Sign-On* 36](#_Toc526198641)

[III.2.2 Kebutuhan Sistem 39](#_Toc526198642)

[III.2.3 Kebutuhan Fungsional Minisso 42](#_Toc526198643)

[III.2.4 Kebutuhan Non-Fungsional Minisso 43](#_Toc526198644)

[III.2.5 Diagram *Use Case* 44](#_Toc526198645)

[III.3 Rancangan Solusi 46](#_Toc526198646)

[III.3.1 Gambaran Umum Solusi 46](#_Toc526198647)

[III.3.2 Desain *Server dan Dashboard* 48](#_Toc526198648)

[III.3.2.1 Diagram Kelas 50](#_Toc526198649)

[III.3.2.2 Diagram *Sequence* 51](#_Toc526198650)

[III.3.3 Pemanfaatan Token 54](#_Toc526198651)

[BAB IV IMPLEMENTASI DAN PENGUJIAN 55](#_Toc526198652)

[IV.1 Implementasi 55](#_Toc526198653)

[IV.1.1 Lingkungan Implementasi 55](#_Toc526198654)

[IV.1.2 Implementasi *Server* Minisso 56](#_Toc526198655)

[IV.1.3 Implementasi *Dashboard* Minisso 59](#_Toc526198656)

[IV.1.4 Implementasi Aplikasi *Service Provider* 60](#_Toc526198657)

[IV.1.4.1 Prasyarat Aplikasi 61](#_Toc526198658)

[IV.1.4.2 Integrasi Sisi *Server* (*Backend*) 61](#_Toc526198659)

[IV.1.4.3 Integrasi Sisi *Browser* (*Front-End*) 63](#_Toc526198660)

[IV.2 Pengujian 63](#_Toc526198661)

[IV.2.1 Pengujian Fungsional KF1 64](#_Toc526198662)

[IV.2.2 Pengujian Fungsional KF2 65](#_Toc526198663)

[IV.2.3 Pengujian Fungsional KF3 67](#_Toc526198664)

[IV.2.4 Pengujian Fungsional KF4 68](#_Toc526198665)

[IV.2.5 Pengujian Fungsional KF5 69](#_Toc526198666)

[IV.2.6 Pengujian Fungsional KF6 70](#_Toc526198667)

[IV.3 Analisis Hasil Uji 70](#_Toc526198668)

[BAB V KESIMPULAN DAN SARAN 71](#_Toc526198669)

[V.1 Kesimpulan 71](#_Toc526198670)

[V.2 Saran 72](#_Toc526198671)

[DAFTAR PUSTAKA 73](#_Toc526198672)

DAFTAR GAMBAR

[Gambar ‎II.1 Sistem yang Mekanisme Autentikasinya tanpa SSO 14](#_Toc526198673)

[Gambar ‎II.2 Sistem dengan SSO 15](#_Toc526198674)

[Gambar ‎II.3 Cara Kerja Auth0 18](#_Toc526198675)

[Gambar ‎II.4 Arsitektur WSO2 *Identity Server* (WSO2, 2015) 20](#_Toc526198676)

[Gambar ‎II.5 Mekanisme WSO2 *Identity Server* 21](#_Toc526198677)

[Gambar ‎II.6 Mekanisme Kerja CAS 23](#_Toc526198678)

[Gambar ‎II.7 Konsep General *Single Sign-On* Berbasis Token 27](#_Toc526198679)

[Gambar ‎III.1 Sistem *Single Sign-On* 40](#_Toc526198680)

[Gambar ‎III.2 *Use Case* Minisso 45](#_Toc526198681)

[Gambar ‎III.3 Arsitektur Minisso 48](#_Toc526198682)

[Gambar ‎III.4 Diagram Kelas Minisso 50](#_Toc526198683)

[Gambar ‎III.5 Diagram *Sequence* KF1 Registrasi User 51](#_Toc526198684)

[Gambar ‎III.6 Diagram *Sequence* KF2 Login User 52](#_Toc526198685)

[Gambar ‎III.7 Diagram *Sequence* KF3 Validasi Token 52](#_Toc526198686)

[Gambar ‎III.8 Diagram *Sequence* KF4 Manajemen Aplikasi *Service Provider* 53](#_Toc526198687)

[Gambar ‎III.9 Diagram *Sequence* KF5 Manajemen User 53](#_Toc526198688)

[Gambar ‎IV.1 Diagram Kelas Implementasi Server Minisso 58](#_Toc526198689)

[Gambar ‎IV.2 Aplikasi *Service Provider* 60](#_Toc526198690)

DAFTAR TABEL

[Tabel ‎III.1 Perbandingan *Web Single Sign-On* 35](#_Toc526198691)

[Tabel ‎III.2 Karakteristik *Single Sign-On* 37](#_Toc526198692)

[Tabel ‎III.3 Kebutuhan Sistem Minisso 41](#_Toc526198693)

[Tabel ‎III.4 Kebutuhan Fungsional Minisso 43](#_Toc526198694)

[Tabel ‎III.5 Kebutuhan Non-Fungsional Minisso 43](#_Toc526198695)

[Tabel ‎IV.1 Lingkungan Implementasi 55](#_Toc526198696)

[Tabel ‎IV.2 Pengujian Fungsionalitas KF1 Registrasi *User* 64](#_Toc526198697)

[Tabel ‎IV.3 Pengujian Fungsionalitas KF2 *Login* *User* 65](#_Toc526198698)

[Tabel ‎IV.4 Pengujian Fungsionalitas KF3 Verifikasi Token 67](#_Toc526198699)

[Tabel ‎IV.5 Pengujian Fungsionalitas KF4 Manajemen Aplikasi 68](#_Toc526198700)

[Tabel ‎IV.6 Pengujian Fungsionalitas KF5 Manajemen User 69](#_Toc526198701)

[Tabel ‎IV.7 Pengujian Fungsionalitas KF6 Menerima *Request* 70](#_Toc526198702)

# PENDAHULUAN

## Latar Belakang

Tren pengembangan perangkat lunak berevolusi dari waktu ke waktu. Di era 1980-an, orang cenderung mengembangkan aplikasi di *desktop*, pengembangan aplikasi berbasis *web* belum populer pada saat itu. Sementara itu, di masa kini, hampir semua aplikasi yang digunakan sudah berbasis *web* dan terkoneksi ke jaringan internet, walaupun tidak dipungkiri aplikasi *desktop* saat ini masih tetap digunakan.

Dengan bergesernya *trend* pengembangan aplikasi menuju ke arah berbasis internet, kini banyak juga yang perusahaan yang mengembangkan SaaS (*Software as a Service*) (Goth, 2008). Produk yang ditawarkan oleh perusahaan dapat diakses secara *real-time* dan langsung melalui internet. Banyak bidang yang menawarkan produknya melalui internet, seperti misalnya *e-commerce*, *banking*, *travel agent*, *e-government*, penyedia jasa infrastruktur, dan lain lain.

Banyak institusi yang menyediakan banyak *service* aplikasi sekaligus. Setiap aplikasi tentu membutuhkan *credential* untuk autentikasi dan autorisasi. Tidak efisien apabila tiap aplikasi dalam satu vendor yang sama memiliki mekanisme autentikasi sendiri-sendiri (Volchkov, 2001). Ini mengakibatkan redundansi *credential* yang sebenarnya tidak diperlukan.

Institusi biasanya memiliki lebih dari 70 aplikasi yang membutuhkan sistem supaya user dapat log-in dan menyimpan credential (nama, email, password) (Novell, 2011). Fakta yang terjadi pada perusahaan ini membuktikan bahwa ada kebutuhan besar terhadap solusi single sign-on supaya mekanisme autentikasi dan autorisasi menjadi efektif.

Setiap aplikasi dalam sebuah sistem yang terintegrasi membutuhkan mekanisme autentikasi supaya menjamin hanya user yang terautentikasi-lah yang mampu mengakses informasi pribadi dan servis dari aplikasi (Goth, 2008). Apabila setiap satu aplikasi menyimpan informasi *credential* berupa email dan password, redundansi akan terjadi. Dalam satu *enterprise*, rata-rata user akan menggunakan antara 5-10 aplikasi sekaligus (Novell, 2011). Sebagai contoh, user kemungkinan besar akan menggunakan banyak servis Google secara bersamaan, seperti Google Maps, Youtube, dan Google Mail. Tanpa *single sign-on*, user harus mengelola tiap password untuk tiap aplikasi.

Di sisi *enterprise*, Google harus menyimpan *credential* masing-masing user untuk tiap aplikasi. Ini tentunya tidak efisien. Bagaimana bisa aplikasi dalam satu vendor, memiliki mekanisme autentikasi yang berbeda. Bagaimana jika user cukup login sekali saja untuk satu vendor (Google), kemudian user bisa mengakses seluruh servis yang disediakan Google? (Goth, 2008).

*Single sign-on* adalah solusi dari permasalahan autentikasi multi-servis. Ide utama *single sign-on* adalah pemusatan autentikasi beberapa servis menjadi *single-entry point* untuk autentikasi (Volchkov, 2001). Dengan *single sign-on*, permasalahan autentikasi akan disentralisasi menjadi satu mekanisme autentikasi. Mekanisme *single sign-on* ini akan mengatur dan memberikan autorisasi kepada *credential* supaya bisa mengakses bagian tertentu pada servis.

Single sign-on adalah mekanisme di mana user cukup mengautentikasi dirinya satu kali saja kemudian user diberi akses ke banyak web servis, tanpa memasukkan banyak *password* (Volchkov, 2001). User diberi akses oleh *server single sign-on* berupa *token* atau *cookies* yang dapat digunakan untuk mengakses beberapa aplikasi *service provider*.

Secara umum, implementasi *single sign-on* melibatkan *server single sign-on* sebagai penyedia identitas yang mengatur autentikasi dan autorisasi dan aplikasi *service provider* sebagai penyedia servis yang bertugas mengkonsumsi autentikasi dari *server single sign-on*. Apabila bicara ke level detil implementasi, ada banyak mekanisme berbeda yang dilakukan oleh SSO yang sudah ada seperti yang dilakukan oleh SAML, Auth0, OAuth, OpenID, dan LastPass.

Dalam praktek di lapangan, tidak seluruh *web single sign-on* dengan mudah diintegrasi dengan aplikasi *service provider*. Beberapa masalah muncul dalam proses pengembangan *single sign-on* dengan aplikasi *service provider*. Masalah yang muncul adalah dokumentasi tidak lengkap, contoh aplikasi yang disediakan tidak berjalan semestinya, server *single sign-on* yang membutuhkan memori besar untuk dijalankan. Sebagai contoh, *single sign-on* milik Apereo, *Central Authentication Service* membutuhkan server dengan RAM minimal 8 GB untuk dijalankan (Apereo, 2015). Sementara, WSO2 tidak dengan mudah diintegrasi karena dukungan dokumentasi yang kurang lengkap dan contoh aplikasi *service provider* pengguna *single sign-on* yang tidak berjalan semestinya. Sementara itu, Auth0 sebagai produk single sign-on lain memiliki banyak keunggulan seperti kemudahan pengembangan, dukungan dokumentasi lengkap, dan dukungan SDK lengkap terhadap banyak bahasa dan *framework*. Namun penggunaan Auth0 untuk lingkup gratis harus menggunakan media penyimpanan basis data *cloud* yang disediakan Auth0 (Pace, 2015). Ini tentunya kurang nyaman bagi developer ketika aplikasi menyimpan data sensitif user. Auth0 menyediakan server *self-hosted* yang memungkinkan developer menyimpan data *user* pada media penyimpanan yang dapat dimodifikasi sendiri, tetapi layanan ini berbayar.

Berdasarkan kelemahan dan kelebihan masing-masing *single sign-on* yang sudah dianalisis kemudian, maka dalam tugas akhir ini diusulkan *web single sign-on* yang baru. *Web single sign-on* yang dibangun merupakan akumulasi poin positif dari *web single sign-on* yang sudah ada, dengan penyempurnaan pada kelemahan dari masing-masing *single sign-on* yang dianalisis.

## Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, didapat perumusan masalah yang akan diselesaikan dalam tugas akhir, yakni sebagai berikut:

1. Bagaimana cara membangun *web single sign-on* sebagai penyempurnaan terhadap *single sign-on* yang pernah ada sebelumnya.
2. Bagaimana karakteristik *web* *single sign-on* yang akan dibangun.

## Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut.

1. Menentukan teknik yang tepat untuk membangun *web single sign-on* sebagai penyempurnaan terhadap *single sign-on* yang sudah ada.
2. Menentukan karakteristik *web* *single sign-on* yang akan dibangun.

## Batasan Masalah

Batasan-batasan dalam pelaksanaan tugas akhir ini perlu didefinisikan agar hasil dari tugas akhir bersifat spesifik dan jelas. Batasan-batasan yang diambil dalam pelaksanaan tugas akhir ini adalah sebagai berikut.

1. Fokus pengerjaan dalam tugas akhir ini adalah *web single sign-on*. Aplikasi *service provider* bukan merupakan fokus dalam tugas akhir.
2. Pengiriman data antar komponen dilakukan melalui protokol HTTP, protokol lain seperti OAuth, SAML, OpenID tidak didukung.

## Metodologi

Metodologi yang digunakan dalam pengerjaan tugas akhir ini meliputi:

1. Analisis masalah. Pada tahap ini, dilakukan analisis terhadap permasalahan berkaitan dengan topik yang diangkat pada tugas akhir ini. Analisis masalah ini meliputi kajian terhadap bermacam model dan desain *single sign-on* yang pernah didesain orang sebelumnya.
2. Pembangunan aplikasi *single sign-on*. Setelah masalah dianalisis dan dilakukan *requirement engineering*, dibangunlah aplikasi *single sign-on* yang sesuai dengan kebutuhan yang diberikan. Aplikasi ini harus mampu memenuhi kebutuhan fungsional yang telah didefinisikan pada tahap analisis masalah sebelumnya. Aplikasi *single sign-on* ini bertindak sebagai sentra pusat autentikasi bagi *service* lain..
3. Pembangunan aplikasi *service provider*. Walaupun pembangunan aplikasi *service provider* bukan merupakan fokus utama dalam pengerjaan tugas akhir ini karena topik sebenarnya adalah *single sign-on*, aplikasi yang bertindak sebagai *service provider* harus tetap dibangun walau dengan fungsionalitas yang bersinggungan langsung dengan *single sign-on* saja. Ini tetap harus dilakukan untuk tujuan pengujian fungsionalitas *single sign-on* nantinya.
4. Evaluasi dan penarikan kesimpulan. Pada tahap ini dilakukan evaluasi terhadap *single sign-on* yang telah dibangun. Setelah itu, dilakukan juga penarikan kesimpulan yang didasari oleh hasil evaluasi pengujian aplikasi single sign-on.

## Sistematika Pembahasan

Penulisan tugas akhir ini dibagi menjadi 5 bab, yaitu bab I Pendahuluan, bab II Studi Literatur, bab III Analisis Permasalahan dan Rancangan Solusi, bab IV Implementasi dan Pengujian, dan bab V Kesimpulan dan Saran. Bab I Pendahuluan memberikan gambaran mengenai tugas akhir ini. Bab I terdiri dari latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan.

Bab II Studi Literatur memaparkan hasil studi literatur terkait autentikasi dan autorisasi, *single sign-on*, beserta kakas dalam pengembangan *single sign-on*.

Bab III Analisis Permasalahan dan Desain Solusi berisi hasil analisis dari permasalahan yang ada pada tugas akhir ini serta usulan solusi yang diajukan untuk memecahkan permasalahan tersebut.

Bab IV Implementasi dan Pengujian membahas implementasi serta hasil pengujian. Pengujian yang dilakukan terkait perbandingan antara kakas yang digunakan satu sama lain atas berbagai parameter.

Bab V Kesimpulan dan Saran menjelaskan mengenai kesimpulan dari tugas akhir ini dan saran untuk mengembangkan lebih lanjut mengenai tugas akhir ini agar menghasilkan hasil yang lebih baik.

# STUDI LITERATUR

## Autentikasi dan Autorisasi

Autentikasi adalah proses bagaimana sistem dapat mengetahui dengan siapa sistem tersebut berkomunikasi (Tanenbaum, 2007). Menurut Neuman (1994), autentikasi adalah verifikasi dari identitas seseorang yang memberikan data beserta data yang diberikan.

Autorisasi adalah teknik yang menentukan proses apa saja yang boleh dilakukan (Tanenbaum, 2007). Definisi lain dari Neuman (1994), autorisasi adalah proses yang menentukan apakah sebuah pihak diperbolehkan untuk melakukan operasi tertentu.

Menurut Halonen (2000), autentikasi dan autorisasi seharusnya memiliki karakteristik sebagai berikut:

1. Kebenaran (*Correctness*)

Hasil dari setiap autentikasi dan autorisasi harus akurat.

1. Kemungkinan untuk anonim dan privasi (*Possibility to anonymity and privacy*)

Autentikasi terhadap identitas hanya dilakukan apabila benar-benar diperlukan. Apabila tidak perlu membuka identitas, maka identitas tersebut tidak perlu dibuka. Jika memungkinkan, autorisasi sebaiknya tidak dilakukan tanpa membuka identitas.

1. Kecepatan (*Speed*)

Proses autentikasi seharusnya dilakukan dengan cepat.

1. Ketahanan terhadap serangan (*Attack resistance*)

Metode autentikasi seharusnya dapat bertahan terhadap berbagai jenis serangan, misalnya serangan *Man-in-the-Middle* (MITM).

1. Murah (*Inexpensiveness*)

Proses autentikasi dan autentikasi seharusnya tidak memerlukan ongkos yang mahal, baik ongkos pengembangan dan pemeliharaan.

1. Kemudahan penggunaan (*User friendliness*)

Metode autentikasi sedapat mungkin mudah digunakan dan dimengerti oleh penggunanya. Pada kondisi ideal, pengguna tidak perlu mengeluarkan banyak usaha ketika melakukan proses autentikasi.

1. Universal (*Universality*)

Metode autentikasi sebaiknya bersifat sama pada semua layanan.

## Sistem Autentikasi Identitas

Autentikasi adalah hal paling mendasar dari keamanan *service* karena autentikasi yang terpercaya dibutuhkan untuk memberi kontrol akses, menentukan siapa yang berhak menerima atau mengubah informasi, dan melawan penyanggahan (*non-repudiation*) (Ford, 1997).

Menurut Ford (1998), ada lima metode yang dapat digunakan dalam proses autentikasi, yaitu sebagai berikut.

1. Sesuatu yang diketahui pengaku (*something the claimant knows*),
2. Sesuatu yang dimiliki pengaku (*something the claimant owns*),
3. Sesuatu tentang siapa pengaku (*something the claimant is*),
4. Pengaku pada tempat atau waktu tertentu (*claimant is at a particular place at particular time*),
5. Autentikasi yang diberikan oleh pihak ketiga yang terpercaya (*authentication is established by a trusted third party*).

### *Password*

*Password* merupakan metode yang paling mudah dan umum digunakan dalam proses autentikasi (Ford, 1998). *Password* pada umumnya diasosiasikan dengan *username*, sehingga metode ini dapat digunakan oleh sistem yang memiliki banyak pengguna. Tetapi, *password* merupakan metode yang paling tidak aman untuk autentikasi dan hal ini merupakan salah satu kerentanan utama dari metode *password*. Namun, kelemahan ini dapat diatasi dengan menggunakan *tool password generator* sehingga kita tidak perlu mengingat *password* yang panjang dan cukup menggunakan *tool password generator* yang diamankan dengan satu *master password* saja.

Kriteria keamanan pada metode *password* terletak hanya pada panjangnya *string password*. Beberapa teknik penyerangan yang dapat dilakukan pada metode *password* adalah *external disclosure*, *guessing*, *communications eavesdropping*, *replay attacks*, dan *host compromise* (Ford, 1998). Oleh karena itu, dewasa ini metode autentikasi *password* sering kali digunakan bersama dengan metode autentikasi lainnya sehingga memiliki tingkat keamanan yang lebih tinggi, atau yang sering disebut *Multi-Factor Authentication* (MFA).

### *Token*

*Token* adalah metode yang digunakan untuk meningkatkan keamanan dari sistem autentikasi (Ford, 1998). Salah satu penerapan metode autentikasi *token* adalah *physical tokens* yang sering digunakan oleh bank. Tujuannya adalah untuk memberikan pengamanan tambahan pada transaksi yang dilakukan. Jika seandainya PIN nasabah bank diketahui oleh pencuri atau pihak yang tidak berhak, maka pencuri tidak dapat melakukan transaksi jika tidak memiliki *token*. Metode pembangkitan *token* juga memerlukan suatu alat fisik dan memperhitungkan variabel waktu sehingga *token* menjadi sulit ditebak oleh penyerang.

Dewasa ini *virtual token* juga sering digunakan sebagai metode autentikasi. Cara pembuatan *virtual token* berbeda dengan cara pembuatan *physical token* yang memerlukan alat atau benda fisik untuk menghasilkan *token*. *Virtual token* merupakan *token* yang dapat dihasilkan secara *virtual* dengan bantuan perangkat lunak atau *software*. Sedangkan untuk representasi datanya, *physical token* dan *virtual token* mungkin saja memiliki representasi data yang sama, misalnya *string* atau angka. Contoh implementasi dari *virtual token* adalah *One-time Password*, atau yang sering disingkat OTP. OTP adalah *virtual token* yang memungkinkan *user* menerima *password* yang dibangkitkan secara acak oleh *provider* OTP dan berlaku hanya dalam jangka waktu tertentu (biasanya 30 detik). Contoh OTP yang secara luas digunakan oleh masyarakat adalah *Google Authenticator* dan *Authy*. Kedua aplikasi tersebut merupakan aplikasi yang berjalan pada *device* *Android* dapat diunduh di *Playstore* secara gratis.

Contoh penggunaan *virtual tokens* pada aplikasi berbasis web adalah *session* dan *cookie*. *Session* disimpan pada *server* dan digunakan untuk menyimpan *state* dari pengguna (Whitehead, 2006). *Session* dapat diumpamakan sebagai kumpulan dari *key* yang memiliki *value* tertentu. Masing-masing *session* memiliki *session* *ID* yang dihasilkan secara otomatis dan berbeda untuk setiap *session*. *Session ID* inilah yang dapat dianggap sebagai *token*.

*Cookie* merupakan kumpulan dari pasangan parameter dan nilainya. *Cookie* dibuat atau dihasilkan oleh *server*, dan diberikan kepada *web browser*. *Cookie* umumkan disimpan pada media penyimpanan lokal milik *device client* dan dapat diakses oleh *web browser*. *Cookie* dapat digunakan untuk memberikan tanda bahwa pengguna sudah melakkan atau mendapatkan sesuatu dari *server*. Dengan kata lain, *cookie* merupakan suatu mekanisme untuk menambahkan *state* pada *HTTP request* (Whitehead, 2006).

Pada umumnya, *session* dan *cookie* dapat dikombinasikan sebagai teknik manajemen *state* pada aplikasi web (Whitehead, 2006). State diperlukan pada aplikasi web untuk mengetahui apakah pengguna pernah mengunjungi web pada waktu sebelumnya atau apakah pengguna sudah melakukan tahapan yang benar untuk sebuah proses pada aplikasi berbasis web. *Server* menyimpan informasi *state* dari pengguna dengan menggunakan *session*. *Cookie* berisi informasi, misalnya *session ID*, yang digunakan untuk mencari *session* milik pengguna pada *server*. *Cookie* dihasilkan oleh *server* dan diberikan kepada *web browser* pada *HTTP response*, kemudian *cookie* dikirimkan kembali oleh *web browser* pada *server* pada *HTTP request* selanjutnya.

### *Biometrics*

Metode *biometrics* menekankan bahwa autentikasi akan mengidentifikasi dan memverifikasi seseorang berdasarkan dari ciri-ciri fisik atau perilaku yang dimilikinya. Metode autentikasi dengan *biometrics* memiliki kekurangan utama yaitu harga atau biaya untuk mengimplementasikannya sangat mahal dibandingkan dengan metode autentikasi lainnya (Ford, 1998). Contoh ciri-ciri fisik yang dapat digunakan pada metode *biometrics* adalah sidik jari (*fingerprint*), retina mata, tulisan tangan, bentuk wajah, dan bentuk gigi. Sedangkan, contoh ciri perilaku yang dapat digunakan adalah kebiasaan gerak seseorang dan suara dalam melafalkan kata-kata kalimat.

Salah satu metode *biometrics* yakni sidik jari memiliki akurasi yang sangat tinggi. Sidik jari manusia merupakan komponen yang identitas manusia yang unik. Dua *fingerprint* yang sama hanya ditemukan sekali dalam setiap 1048 tahun (Jain, 2005).

### Tanda Tangan Digital

Tanda tangan digital atau *digital signatures* adalah tanda tangan yang diberikan pada data digital (Munir, 2005). Tanda tangan berupa nilai yang berdasarkan dari isi dan pengirim pesan. Proses umum pemberian tanda tangan melibatkan kriptografi asimetri dan fungsi *hash*. Tanda tangan digital dapat digunakan oleh penerima pesan untuk memverifikasi bahwa pengirim pesan memiliki kunci privat yang sah dan isi pesan tidak diubah selama proses pengiriman (Ford, 1998). Penerima pesan dapat dianalogikan sebagai aplikasi dan pengirim pesan adalah pengguna, atau sebaliknya. Sebagai metode autentikasi, pengguna juga dapat mengirimkan tanda tangan digitalnya untuk diverifikasi.

### Sertifikat Digital

Sertifikat digital merupakan sertifikat yang dikeluarkan oleh *Certification Authorities* (CA) yang menyatakan hubungan antara kunci publik dan pemiliknya (Ford, 1998). Isi informasi dari sertifikat digital berupa nama organisasi pemilik kunci publik, kunci publik, dan waktu kadaluarsa sertifikat. Sertifikat digital dapat digunakan untuk mengautentikasi seorang pengguna atau sebuah *website* ataupun sebuah institusi. *Certification Authorities* memberikan cara untuk mengirimkan kunci publik beserta informasi pemiliknya pada dunia digital melalui sertifikat digital (Ford, 1998). Namun, pengenalan CA yang sah dan dipercaya untuk mengeluarkan sertifikat digital harus dapat dilakukan oleh pihak penerima pesan. Pengenalan CA yang sah menjadi masalah baru yang ditimbulkan dari penggunaan metode *Certification Authorities* untuk metode autentikasi.

## *Multi-Factor Authentication*

*Multi-Factor Authentication* (MFA) atau autentikasi banyak faktor adalah autentikasi yang menggunakan lebih dari satu faktor atau metode (*Federal Deposit Insurance Corporation*, 2004). Faktor untuk melakukan autentikasi adalah lima metode autentikasi yang sudah dijelaskan pada subbab 2.2. MFA dengan dua metode autentikasi seringkali disebut *two-factor authentication* (2FA).

Semakin banyak metode autentikasi yang digunakan, semakin baik keamanan sebuah sistem (*Federal Deposit Insurance Corporation*, 2004). Tetapi, untuk setiap kelebihan kadangkala terdapat kelemahan yang harus diterima sebagai konsekuensinya. Kelemahannya adalah lamanya waktu yang dibutuhkan untuk autentikasi yang berbanding lurus dengan banyaknya metode autentikasi yang digunakan.

## *Single Sign-On*

*Single sign-on* adalah proses autentikasi *user* yang memperbolehkan seorang pengguna dengan sebuah identitas yang terdaftar untuk mengakses berbagai aplikasi (Pashadilis, 2003). *Single sign-on* disingkat SSO. Menurut He (2007), *single sign-on* dalam definisi yang lebih sederhana adalah proses penggunaan identitas pengguna yang sama secara bersama-sama. *Single sign-on* adalah bagian dari sistem yang memungkinkan *user* mengakses beberapa layanan/aplikasi/*resource* melalui satu titik entri. *Single sign-on* merupakan properti akses kontrol dari beberapa sistem yang berhubungan tetapi independen (Volchkov, 2001). Dengan properti ini, *user* dapat melakukan *log in* menggunakan satu *ID* dan *password* untuk mendapatkan akses ke sistem yang terhubung atau sistem dengan banyak *username* dan *password* yang berbeda. *Single-sign on* biasanya menyimpan satu *set credential*, seperti nama, *e-mail*, dan *password* untuk mengakses beberapa aplikasi. *Single sign-on* mengautentikasi *user*, apakah *user* yang bersangkutan diberikan hak untuk mengakses *resource* yang ada. Pada *back-end*, *single sign-on* sangat membantu untuk mengelola aktivitas *user* sekaligus memonitor akun. Contoh sederhana dari *single sign-on* dapat dicapai lewat penyimpanan *cookie* di *browser* untuk domain tertentu, namun hanya dapat dicapai apabila situs memiliki domain DNS yang sama. Secara umum, *single sign-on* terdiri dari *Identity Provider* (IdP) yang berfungsi untuk mengelola *credential*/*session* milik *user*. Sementara itu *single-sign on* juga memiliki *Service Provider* (SP) yang merupakan konsumer dari autentikasi yang disediakan IdP.

Misalkan dalam sebuah sistem terdiri dari beberapa aplikasi dan sistem tersebut memiliki mekanisme autentikasi dalam *single sign-on*. Penguna harus melakukan *log in* pada setiap aplikasi dan aplikasi akan mengautentikasi pengguna tersebut. Pengguna juga harus mendaftarkan dirinya dan mengingat sejumlah identitas untuk diautentikasi pada sejumlah aplikasi. Hal ini juga dapat menyebabkan *human error* jika pengguna lupa terhadap identitasnya atau mekanisme autentikasi *error* pada sebuah aplikasi.

Pengelolaan pengguna menjadi pekerjaan yang merepotkan jika pengguna dan/atau aplikasi semakin bertambah banyak. Hal ini dapat diperburuk dengan pengelolaan pengguna yang harus dilakukan pada masing-masing aplikasi jika aplikasi-aplikasi tidak menggunakan *single sign-on*.

Pada sistem yang mekanisme autentikasinya tanpa *single sign-on*, aplikasi juga mengalami kerugian yaitu terdapat *overhead* proses autentikasi. *Overhead* pada autentikasi maksudnya adalah proses autentikasi dilakukan berkali-kali pada setiap aplikasi meskipun proses autentikasi tersebut cukup dilakukan sekali oleh sebuah aplikasi saja. Dengan menggunakan *single sign-on*, proses autentikasi cukup dilakukan sekali pada aplikasi yang menangani *single sign-on.*



Gambar ‎II.1 Sistem yang Mekanisme Autentikasinya tanpa SSO

.

Beberapa poin kerugian, yaitu kemungkinan terjadinya *human error*, kerumitan pada pengelolaan pengguna dan *overhead* pada proses autentikasi, dapat terjadi pada sistem yang tidak menggunakan *single sign-on*. Gambar II.1 merupakan contoh sistem yang mekanisme autentikasinya tanpa *single sign-on*. Sistem pada gambar tersebut terdiri dari aplikasi A, B, dan C. Pengguna akan melakukan *log in* terlebih dahulu untuk menggunakan aplikasi yang diinginkannya. Setiap aplikasi memiliki mekanisme autentikasinya sendiri sehingga pengguna harus melakukan *log in* pada setiap aplikasi untuk menggunakannya.

Mekanisme autentikasi pada Gambar II.1 kurang efektif dan efisien jika dibandingkan dengan mekanisme autentikasi pada Gambar II.2. Mekanisme autentikasi pada Gambar II.2 menggunakan *single sign-on* untuk mekanisme autentikasi seluruh aplikasi. Sehingga, pengguna cukup terautentikasi sekali saja untuk menggunakan aplikasi A atau B atau C. Autentikasi dapat dilakukan oleh salah satu aplikasi ataupun mengunakan aplikasi pihak ketiga yang dipercaya oleh masing-masing aplikasi A, B, dan C.



Gambar ‎II.2 Sistem dengan SSO

Berikut adalah kelebihan implementasi *single sign-on*:

1. Satu *password* untuk masuk ke dalam sistem.
2. *Login* otomatis, apabila *user* sudah memiliki *session* sebelumnya.
3. Lebih mudah untuk mengelola *user*.
4. Satu klik untuk mengambil (*revoke*) akses *user* untuk sebagian atau seluruh bagian sistem.

Berikut adalah kekurangan implementasi *single sign-on*:

1. *Single point of failure*. Apabila server *single sign-on* *down*, maka seluruh aplikasi tidak memiliki mekanisme untuk autentikasi *user*. Dapat di-*handle* dengan mereplika *server* *single sign-on* menjadi beberapa replika supaya *availability* semakin besar.
2. *Single point of attack*. Penyerang dapat bertindak sebagai aplikasi yang sah untuk mencuri informasi yang dikirimkan SSO *server* kepada pengguna (Pashadilis, 2003). Sehingga, penyerang dapat mengetahui informasi berharga yang digunakan untuk komunikasi antara SSO *server* dan pengguna.
3. Apabila aplikasi sudah ada dan *single sign-on* akan diintegrasi, *cost* untuk mengintegrasikan *single sign-on* dengan aplikasi yang sudah ada cukup besar. Hal ini biasanya rumit, terutama untuk pengimplementasian SSO pada sebuah sistem lama. Pada kondisi ini, implementasi SSO lebih diarahkan kepada penambahan fitur untuk meningkatkan *user experience* dan efektivitas dan efisiensi dari aplikasi. Akibatnya, implementasi SSO harus memperhatikan fungsionalitas yang sudah ada dan biasa digunakan pada sistem lama.

## *Web Single Sign*-*On*

*Web single sign-on* adalah *single sign-on* yang digunakan pada aplikasi berbasis web (Jasig, 2009). *Web single sign-on* biasanya disingkat *web* SSO. Banyaknya jumlah layanan yang tersedia di Internet menyebabkan ketidaknyamanan terhadap pengguna layanan. Pengguna biasanya harus melakukan *log in* pada beberapa sistem layanan dan masing-masing sistem layanan memerlukan identitas untuk diautentikasi. Jika setiap sistem layanan yang memerlukan identitas yang spesifik dan berbeda, maka pengguna akan mendapatkan kesulitan yang lebih banyak. Salah satu kesultan dalah pengguna harus mengingat identitas yang digunakan pada setiap layanan.

*Web single sign-on* sangat mudah ditemukan dari *web* pada Internet. Dewasa ini, beberapa grup atau komunitas sudah mengembangkan *web single sign-on*. Beberapa produk *web single sign-on* bersifat *open-source*, tetapi ada juga produk yang bersifat komersil. Beberapa grup atau kounitas yang mengembangkan *web single sign-on* adalah:

1. *Auth0 Inc* dengan produknya *Auth0*.
2. WSO2 dengan produknya WSO2 *Identity Server*.
3. *Apereo* dengan produknya *Central Authentication Service*.
4. *Atricore Inc* dengan produknya *Java Open Single Sign-On*.

### *Auth0*

Auth0 adalah layanan *Platform as a Service* (PaaS) yang menyediakan *single sign-on* yang memudahkan *developer* untuk memungkinkan *developer* menambahkan SSO pada aplikasinya secara mudah tanpa memperdulikan bahasa, teknologi, dan protokol yang digunakan dalam autentikasi. Sebagai PaaS, Auth0 menyediakan *platform single sign-on* secara keseluruhan termasuk *dashboard*, media penyimpanan data *user*, dan halaman *login* untuk *single sign-on*. Auth0 dikembangkan oleh *Auth0 Inc* sebuah perusahan yang berpusat di Seattle, Amerika Serikat. Alamat *website* Auth0 adalah <https://auth0.com>. Sementara alamat *website* untuk manajemen *dashboard* Auth0 adalah <https://manage.auth0.com>.

Auth0 dapat digunakan tanpa memerdulikan bahasa apa yang digunakan aplikasi *service provider*. Sejauh ini, banyak *stack* teknologi yang didukung Auth0 baik *regular web application* (*server-side authentication*) seperti Go, Java, Ruby on Rails, dan seterusnya, *single-page application* (*client-side authentication*) seperti NodeJS, AngularJS, dan JQuery, maupun *mobile application* (iOS dan Android) (Pace, 2015). Masing-masing jenis *client server-side*, *client-side*, dan *mobile* memiliki alur autentikasi yang berbeda-beda. Auth0 mendukung lebih dari 65 SDK (*Software Development Kit*) yang dapat digunakan *developer* untuk mengintegrasikan aplikasinya dengan *single sign-on* Auth0. Auth0 juga mendukung protokol SAML, OpenID, dan integrasi dengan banyak penyedia *single sign-on* (*provider*) seperti Slack, Dropbox, Yahoo. Selain itu, dokumentasi Auth0 sangat mudah dimengerti dan diaplikasikan karena banyak tersedia *sample application* yang benar-benar dapat dijalankan. Dengan demikian, Auth0 berkembang dengan sangat pesat, saat ini memiliki 2000+ *enterprise* *customer*, 42 juta lebih *login* dalam satu hari. Cara kerja Auth0 dapat dilihat pada Gambar II.3 di bawah.



Gambar ‎II.3 Cara Kerja Auth0

Cara kerja Auth0 adalah sebagai berikut.

* 1. *Admin*, atau *developer* mendaftarkan aplikasi *service provider* ke *dashboard* Auth0.
  2. *Dashboard* Auth0 akan menyimpan informasi kepada *server* mengenai *developer* dan aplikasi miliknya.
  3. *Developer* mengarahkan (*redirect*) aplikasi *service provider* ke *server* Auth0.
  4. *User* ingin mengakses *resource* di aplikasi *service provider* yang dilindungi oleh autentikasi.
  5. Aplikasi *service provider* mengetahui *user* belum terautentikasi, maka *service provider* mengarahkan *user* ke halaman *login* di *server* Auth0. *Server* Auth0 ini dapat berupa *cloud* Auth0 yang dapat ter-*deploy* sehingga dapat diakses siapapun, ataupun *server* yang berada pada jaringan privat. Untuk layanan gratis, Auth0 hanya menyediakan *server cloud* saja sehingga kita tidak bisa menggunakan *server* Auth0 pada jaringan privat kita sendiri. Ini artinya seluruh data *user* dan aplikasi *service provider* disimpan pada media penyimpanan milik Auth0.
  6. User melakukan autentikasi pada halaman *login server* Auth0 dengan memasukkan *credential*.
  7. Setelah autentikasi pada *server* Auth0 berhasil dilakukan, *server* mengembalikan *user* ke *service provider* beserta *token* yang dapat digunakan untuk identifikasi *user* ketika mengakses *resource* berikutnya.
  8. Ketika *user* membutuhkan *resource* yang membutuhkan autentikasi, aplikasi tidak perlu lagi melakukan *redirect* laman ke *server* Auth0 karena *token* sudah tersimpan pada aplikasi *service provider* untuk digunakan selanjutnya.

### WSO2 *Identity Server*

WSO2 *Identity Server* adalah *web single sign-on open-source* yang dibuat oleh WSO2. WSO *Identity Server* dibangun di atas WSO2 *Carbon*, sebuah framework manajemen identitas yang dibuat oleh WSO2. WSO2 mendukung banyak protokol SSO seperti OpenID Connect, OAuth, SAML SSO, maupun *social authenticator* yang popular seperti *Google*, *Facebook*, *Yahoo*, *Microsoft*, juga mendukung berbagai metode autentikasi lain seperti SMS OTP, Email OTP, Kerberos, dan lain-lain.

Berikut adalah arsitektur WSO2.



Gambar ‎II.4 Arsitektur WSO2 *Identity Server* (WSO2, 2015)

Aplikasi *service provider* yang menggunakan WSO2 *Identity Server* melakukan autentikasi pada sisi aplikasi *service provider*. *Credential* yang dimasukkan oleh *resource owner* (user) akan dikirimkan ke WSO2 *Identity Server* melalui *Inbound Authenticator*. *Inbound Authenticator* berfungsi untuk mengurusi permintaan *user* yang akan melakukan *log in*. Permintaan ini diterima oleh modul *Request Processor* kemudian akan diteruskan ke *Authentication Framework* yang berada pada gambar di tengah atas. Permintaan *user* atas *service provider* dipetakan ke struktur data lokal milik WSO2 (*claim mapping*) yang terstandardisasi menurut WSO2. Kemudian permintaan *login* akan dilanjutkan menurut tipenya. Apabila permintaan *login* tersebut adalah *login* dengan *username*/*password*, maka permintaan akan diteruskan ke *Local Authenticators*, sementara apabila permintaan *login* menggunakan model autentikasi lain seperti SAML, OpenID Connect, dst maka permintaan akan diteruskan *Federated Authenticators*. *Federated Authenticators* bertindak sebagai komunikator antara WSO2 *Identity Server* dengan aplikasi eksternal seperti *Google*, *Facebook*, *Yahoo*, atau penyedia layanan *identity server* lain.

Setelah *Local Authenticators* atau *Federated Authenticators* selesai melakukan validasi terhadap *user*, respon akan dikembalikan ke *Authentication Framework*. *Authentication Framework* akan melakukan pemetaan ulang dari *identity server* ke struktur data internal WSO2 (*claim mapping*), dan dari *claim mapping* ke *service provider*. Respons akan dikembalikan ke *Inbound Authenticator*. Modul *Response Generator* pada *Inbound Authenticator* akan mengembalikan respon *login* kepada *user*. Respon ini bisa berupa penolakan *login* atau penerimaan *login*.



Gambar ‎II.5 Mekanisme WSO2 *Identity Server*

Cara kerja WSO2 *Identity Server* adalah sebagai berikut:

* 1. *Admin*, atau *developer* mendaftarkan aplikasi *service provider* ke *dashboard* WSO2 *Identity Server*.
  2. *Dashboard* WSO2 akan menyimpan informasi kepada *server* mengenai *developer* dan aplikasi miliknya.
  3. *Developer* mengarahkan aplikasi *service provider* ke *single sign-on* ketika membutuhkan autentikasi *user*.
  4. *User* ingin mengakses *resource* di aplikasi *service provider* yang dilindungi oleh autentikasi.
  5. Aplikasi *service provider* mengirimkan *credential user* dan *credential* aplikasi untuk menukarkannya dengan *token* *user* yang bisa digunakan untuk *login* pada *session* berikutnya. *Credential user* diperlukan untuk membedakan suatu *user* dan *user* lain. *Credential* aplikasi diperlukan *server* WSO2 untuk memastikan aplikasi yang meminta *login user* tersebut adalah benar-benar aplikasi yang valid.
  6. Setelah autentikasi di *server* Auth0 berhasil dilakukan, *server* memberikan *token* kepada *service provider* yang dapat digunakan untuk identifikasi *user* ketika mengakses *resource* berikutnya.
  7. Ketika *user* membutuhkan *resource* yang membutuhkan autentikasi, aplikasi tidak perlu lagi *login* ke aplikasi, cukup ke laman ke *server* Auth0 karena *token* sudah tersimpan pada aplikasi *service provider* untuk digunakan selanjutnya.

Secara garis besar, cara kerja WSO2 *Identity Server* kurang lebih mirip dengan Auth0 yang telah dijelaskan pada subbab sebelumnya. WSO2 *Identity Server* menyediakan *dashboard* *developer/admin* sama seperti yang Auth0 miliki. Yang menjadi perbedaan adalah *server* WSO2 IS disediakan secara gratis karena *open-source* dan kita bisa melakukan *self-host* dengan mudah.

Perbedaan lain adalah aplikasi *service provider* tidak di-*redirect* ke *server* milik SSO untuk *login*. *Login* dilakukan pada sisi aplikasi tetapi validasi dilakukan pada sisi *server* WSO2.

### *Central Authentication Service (CAS)*

*Central Authentication Service* (CAS) merupakan *web single sign-on* yang dibuat oleh Apereo. Pada mulanya, CAS hanyalah sebuah protokol *single sign-on* yang dibuat oleh *Yale University*. Kemudian, *Central Authentication Service* diimplementasikan dengan *platform* Java dan mendukung *library* dari *client* untuk Java, PHP, Perl, dan lainnya sehingga CAS menjadi sebuah penyedia *web single sign-on*.

Cara kerja protokol autentikasi CAS hampir mirip dengan cara kerja protokol Kerberos. Kemiripan protokol autentikasi dikarenakan CAS menggunakan Kerberos sebagai model dalam pembuatan mekanisme autentikasi. Cara kerja protokol CAS dapat dilihat pada Gambar II.6.



Gambar ‎II.6 Mekanisme Kerja CAS

Cara kerja protokol CAS adalah sebagai berikut (Jasig, 2009)

1. Pengguna mengakses sebuah aplikasi *web* atau *service provider*.
2. Pengguna tersebut akan di-*redirect* ke CAS *login* URL melalui koneksi HTTPS. Nama aplikasi *web* atau *service provider* diberikan sebagai parameter dari *request*.
3. Pengguna memasukkan ID, yang dapat berupa *username*, dan *password*. Kemudian, CAS *server* melakukan autentikasi terhadap pengguna. Jika autentikasi gagal, aplikasi web tidak akan bisa diakses dan pengguna akan diminta untuk mengisikan ID dan *password* yang benar.
4. Jika autentikasi berhasil, CAS *server* mengarahkan pengguna ke aplikasi *web* yang diinginkan dan menambahkan *ticket* pada parameter URL. CAS *server* membuat sebuah *cookie* yang disebut *ticket granting cookie* (TGC). *Ticket granting cookie* akan digunakan untuk menandakan bahwa pengguna berhasil diautentikasi. Untuk selanjutnya, *ticket granting cookie* dapat digunakan untuk melakukan autentikasi secara otomatis.
5. Aplikasi *web* atau *service provider* akan melakukan validasi keabsahan dari *ticket* dan kepemilikan *ticket* oleh pengguna yang sah. CAS *Server Validate* yang disediakan oleh CAS *server* dapat digunakan untuk memvalidasi *ticket*.
6. Aplikasi *web* dan CAS *server* akan berkomunikasi melalui HTTPS dan *ticket* dan nama aplikasi akan dijadikan sebagai parameter. CAS *server* melakukan pengecekan bahwa *ticket* tersebut sah. Jika validasi berhasil, CAS *server* akan memberikan ID pengguna pada aplikasi.

*Central Authentication Service* terdiri dari dua komponen yaitu CAS *server* dan CAS *client*. CAS *server* menangani *web single sign-on* untuk aplikasi *web*. CAS *server* merupakan sebuah aplikasi *web* dengan *platform* Java.

CAS *client* akan membantu aplikasi *web* untuk menangani modul *web* *single sign-on* dari CAS *server*. CAS *client* akan dijadikan sebagai modul dari aplikasi *web* yang ingin mengunakan *web single sign-on*. CAS *client* harus disesuaikan dengan *platform* atau *framework* dari aplikasi *web*. CAS *client* juga dapat dikonfigurasi atau diimplementasi ulang jika memang diperlukan untuk memberikan dukungan *web single sign-on* pada aplikasi *web*.

Keunggulan dari CAS adalah CAS menyediakan banyak metode autentikasi yang dapat digunakan dengan aplikasi atau komponen lain. Contoh komponen yang digunakan CAS adalah *active directory*, LDAP, DBMS, X.509, SPNEGO, Legacy, dan lain-lain (Jasig, 2009). Metode autentikasi tersebut dapat dikonfigurasi pada CAS *server*.

### *Java Open Single Sign-On (JOSSO)*

JOSSO adalah sebuah infrastruktur *open-source single sign-on* berbasiskan Java EE and Spring yang menyediakan solusi untuk autentikasi dan autorisasi pengguna secara terpusat tanpa melihat *platform* (Atricore, 2010). Alamat URL *website* JOSSO adalah sebagai berikut: <http://josso.org>. JOSSO merupakan sebuah perangkat lunak yang dibuat oleh Atricore Inc.

JOSSO dapat digunakan pada *web server* dengan *platform* yang berbeda-beda, misalnya Jboss, Tomcat, Weblogic, Apache, PHP, dan Microsoft IIS. Meskipun begitu, SSO *gateway* hanya dapat di-*install* pada *web server* yang men-*support* Java, sedangkan SSO *agent* dapat di-*install* pada seluruh *web server*.

Komponen utama JOSSO berjumlah tiga, yaitu SSO *gateway*, SSO *agent*, dan aplikasi *service provider*. Sedangkan, entitas luar yang berinteraksi dengan JOSSO adalah penguna dengan *web browser*-nya, *identity storage resource*, dan *application stage storage resource*.

Penjelasan tiga komponen utama JOSSO adalah sebagai berikut.

1. SSO *gateway* / SSO *server* (*Identity Provider*/IdP)

SSO *gateway* merupakan SSO *server* atau dikenal sebagai *Identity Provider*. SSO *gateway* memiliki fungsi untuk mengelola SSO dari aplikasi *web* yang dapat menggunakan SSO. Pengguna yang dapat menggunakan SSO dapat dikelola melalui SSO *gateway*.

1. SSO *agent*

SSO *agent* bertugas melakukan penanganan *web single sign-on* terhadap aplikasi rekanan sesuai dengan lingkungan eksekusinya. Lingkungan eksekusi dalam prakteknya adalah *web server* dari aplikasi *service provider*. SSO *agent* menggunakan *identity services* yang disediakan oleh SSO *gateway*.

1. Aplikasi *service provider* (SP)

Aplikasi *service provider* atau *partner application* merupakan aplikasi yang menggunakan *web* SSO. Aplikasi *service provider* memiliki ketergantungan atau dependensi terhadap SSO *agent* dan SSO *gateway* untuk menyediakan *web single sign-on* kepada pengguna.

JOSSO mengandalkan komponen *identity provider*, *identity manager*, dan *session manager* untuk menangani *single sign-on* yang dilakukan oleh setiap aplikasi *service provider*. *Session manager* merupakan komponen yang mengelola *session* aplikasi, *identity manager* merupakan komponen yang mengelola *session* dari *user* yang sudah *login*, dan *identiy provider* merupakan komponen yang menyediakan daftar pengguna yang dapat menggunakan SSO.

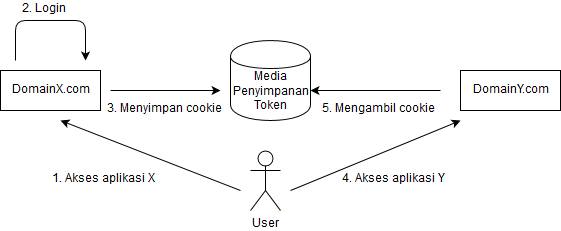
Pada April 2011, versi JOSSO yang terbaru adalah 2.0.2, JOSSO2, mulai versi 2.0.0, merupakan generasi baru dari JOSSO. JOSSO generasi pertama (JOSSO1) dikembangkan hingga versi 1.8.4. Arsitektur JOSSO1 tidak berbeda dengan arsitektur JOSSO2. Namun, JOSSO2 men-*support* SAML2 untuk memberikan *interoperability* dengan sistem SSO pihak ketiga dan aplikasi *Software-as-a-Service* (SaaS). JOSSO2 juga memberikan pengelolaan SSO dengan menggunakan RIA (*Rich Internet Application*).

## *Single Sign-On* Berbasis Token

Teknologi *single sign-on* memungkinkan dua atau lebih aplikasi memiliki mekanisme autentikasi yang terpusat, artinya ketika sebuah user sudah terautentikasi pada aplikasi *service provider* X dan akan mengakses aplikasi Y, user tidak perlu autentikasi lagi karena sudah terautentikasi di aplikasi X. Dengan catatan bahwa aplikasi *service provider* X dan Y berada pada naungan sistem *single sign-on* yang sama.

Pada aplikasi non-SSO, informasi berupa data user yang sudah terautentikasi disimpan pada media penyimpanan *web browser*. Dengan ini, apabila user telah terautentikasi pada sebuah aplikasi, kemudian user mengunjungi aplikasi lagi, *web browser* akan mengambil identitas dari media penyimpanan.

Konsep *single sign-on* berbasis token pada umumnya sama, yakni beberapa aplikasi pada *browser* menggunakan media penyimpanan token yang sama (Pace, 2015). Artinya, ketika user sudah terautentikasi pada sebuah aplikasi, aplikasi lain cukup mengambil data user yang sudah terautentikasi pada media penyimpanan yang sama. Berikut adalah gambar yang menjelaskan konsep general tersebut.



Gambar ‎II.7 Konsep General *Single Sign-On* Berbasis Token

Dalam prakteknya, media penyimpanan token bisa saja dalam variasi yang berbeda-beda, tetapi konsep yang digunakan tetap sama yakni dua domain yang mengakses media penyimpanan yang sama. Berikut adalah kemungkinan variasi media penyimpanan token:

1. *Single sign-on* berada pada domain terpisah yang berbeda dengan domain aplikasi *service provider*. Setiap kali aplikasi akan melakukan login, aplikasi service provider harus melakukan *redirect* ke domain *single sign-on*. *Cookies* yang disimpan merupakan *cookies* milik domain *single sign-on* tersebut.
2. Aplikasi di-*deploy* pada subdomain yang berbeda pada domain yang sama. *Cookies* disimpan pada domain bersama milik subdomain yang berbeda.

### *Cookies* pada Web Browser

*Cookies* adalah serangkaian teks yang disimpan pada *web browser* oleh situs web yang dikunjungi. Pada umumnya, *cookies* menyimpan pengaturan atau preferensi untuk suatu web tertentu, misalkan bahasa yang dipilih, lokasi. Ketika user kembali ke situs web tersebut, *web browser* akan mengirimkan *cookies* yang bersesuaian dengan situs web terkait. Dengan ini, situs dapat menampilkan informasi yang sesuai dengan pengaturan.

Masing-masing *cookies* memiliki atribut seperti alamat apa yang terjadi pada letak *cookies* tersebut, berapa lama *cookies* tersebut valid sebelum kadaluwarsa, dan seterusnya. *Web browser* hanya dapat mengakses *cookies* milik alamat yang memiliki subdomain atau domain yang sama. Misalkan ada sebuah aplikasi dengan URL blog.luqman.com. Aplikasi tersebut dapat membaca cookie yang tersimpan pada blog.luqman.com (subdomain yang sama) dan luqman.com (domain yang sama), tetapi tidak pada [www.luqman.com](http://www.luqman.com) (subdomain berbeda) dan [www.kentang.com](http://www.kentang.com) (domain yang berbeda).

### *Cross-Origin Resource Sharing (CORS)*

*Cross-Origin Resource Sharing* (CORS) adalah mekanisme yang memungkinkan *browser* memberi tahu aplikasi pada origin tertentu diperbolehkan mengakses *resource* terpilih pada origin lain. CORS menggunakan atribut tambahan pada HTTP *header* untuk mengkomunikasikan hal ini. Sebuah aplikasi dikatakan melakukan HTTP *request* *cross-origin* apabila aplikasi tersebut meminta *resource* yang memiliki *origin* berbeda (domain, protokol, port).

Berikut contoh konsep *request* *cross-origin* pada HTTP *request*.

1. [www.luqman.com](http://www.luqman.com) mengakses api.service.com, maka *request* merupakan *cross-origin* karena beda domain.
2. <https://www.luqman.com> mengakses <http://api.luqman.com>, maka *request* merupakan *cross-origin* karena beda protokol.
3. [www.luqman.com:8080](http://www.luqman.com:8080) mengakses [www.luqman.com:1234](http://www.luqman.com:1234), maka *request* merupakan *cross-origin* karena beda port.

Untuk mengatasi aturan CORS yang tidak memperbolehkan suatu *origin* mengakses *resource* dari *origin* lain, *origin* tujuan harus memperbolehkan CORS dengan cara menambahkan atribut *Access-Control-Allow-Origin* pada *path* dan *method* yang dikehendaki.

## Pekerjaan Terkait

### *CAS++*

CAS++ merupakan sistem *open-source single sign-on* yang dikembangkan oleh Ardagna dan krunya. CAS++ merupakan pengembangan dari *Central Authentication Services* (Ardagna, 2006). Pengembangan yang dilakukan pada CAS++ adalah:

1. Menggunakan *identity certificates* dan intertigrasi dengan *security layer* pada Jboss. *Identity certificates* disebut juga sebagai *digital certificates* atau *public key certificates*.
2. Mekanisme autentikasi menggunakan *Public Key Infrastrukture* (PKI).
3. Keamanan pada CAS++ ditingkatkan dengan menggunakan:
   * 1. SSL untuk memberikan jalur komunikasi yang aman terhadap aplikasi web yang berpartisipasi.
     2. X.509 *digital certificates* digunakan untuk pertukaran *credentials*.
4. Menyediakan *user management* secara terpusat sehingga masing-masing aplikasi web tidak perlu melakukan atau menyediakan *user management*. Identitas pengguna akan disimpan secara terpusat pada SSO server.
5. Memiliki layanan yang dapat digunakan untuk aplikasi web berbeda platform.
6. Proses autentikasi menggunakan *username* dan *password* dan diperkuat dengan metode *biometrics* dan *smart card*. Metode *biometrics* yang digunakan adalah sidik jari atau *fingerprint* yang dibaca dengan *fingerprint* *readers*.

Dengan pengembangan yang dilakukan pada CAS++, CAS++ memiliki kelebihan terhadap aspek keamanan (Ardagna, 2006). Keamanan pada CAS++ ditingkatkan dengan menggunakan SSL, X.509 dan metode autentikasi yang menggunakan metode *biometrics* dan *smart card*. CAS++ menjadi lebih mudah untuk digunakan oleh aplikasi web berbeda platform karena menggunakan protokol standar seperti HTTP, SSL, X.509 untuk melakukan pertukaran informasi antara CAS++ server dengan aplikasi web.

### Kristanto, 2011

Kristanto mencoba mengimplementasi web single sign-on pada web pada lingkungan informatika ITB. Masalah timbul ketika aplikasi-aplikasi yang digunakan pada lingkungan informatika ITB seperti *milestone*, *moodle*, *oddyseus* memiliki mekanisme *login*-nya masing-masing.

Kristanto menganalisis karakteristik masing-masing aplikasi, terutama struktur data yang digunakan dalam autentikasi *user* beserta protokol yang digunakan dalam web SSO. Kristanto kemudian mengintegrasikan JOSSO pada aplikasi lingkungan informatika ITB. Media penyimpanan untuk data *user* menggunakan LDAP.

Berikut adalah komponen yang digunakan Kristanto dalam pengembangan SSO pada lingkungan web informatika ITB:

1. SSO *Server* yang merupakan bawaan JOSSO. SSO *server* ini tidak diimplementasi ulang, hanya perlu dikonfigurasi.
2. SSO *Agent* adalah komponen yang menghubungkan aplikasi *service* *provider* dengan SSO *server*. SSO *Agent* mendefinisikan bagaimana aplikasi *login*, *logout*, dan *endpoint* milik SSO *server*. Untuk membuktikan bahwa *single sign-on* tidak peduli pada bahasa client, SSO *Agent* diperlukan sebanyak dua buah, masing-masing menggunakan bahasa Java dan PHP.
3. Aplikasi *service provider*. Terdapat dua buah yakni Java dan PHP dan diletakkan pada *web server* yang berbeda. Aplikasi *service provider* ini melakukan autentikasi melalui SSO *Agent* yang sudah diimplementasi sebelumnya.

Kesimpulan yang didapat adalah *web single sign-on* pada lingkungan informatika ITB dapat dilakukan dengan mengubah sedikit basis data dan beberapa atribut kelas pada aplikasi.

# ANALISIS PERMASALAHAN DAN RANCANGAN SOLUSI

## Analisis Permasalahan

Pada bab ini, akan dipaparkan analisis berdasarkan studi literatur pada bab sebelumnya. Permasalahan yang dipaparan merupakan acuan dari topik Tugas Akhir ini. Setelah analisis permasalahan tersebut, akan dijelaskan rancangan solusi yang akan dibangun.

### Auth0

Dalam tahap studi literatur, di antara *tool web single sign-on* yang dipelajari, Auth0 menyediakan fungsional dan *Software Development Kit* (SDK) yang paling lengkap. Proses integrasi aplikasi dengan *single sign-on* milik Auth0 sangat mudah, hampir tidak ada halangan. Selain itu, Auth0 memiliki dokumentasi *Application Programming Interface* (API) yang jelas dan benar-benar berjalan. Dokumentasi API sangat penting dalam pengembangan perangkat lunak. Penyebabnya adalah karena dalam praktiknya, dokumentasi API merupakan kontrak yang sudah disediakan oleh penyedia API, yang mana *developer* harus mengikuti kontrak yang ada. Selain itu, Auth0 memiliki *dashboard admin* yang bisa digunakan untuk *monitoring user*, *monitoring* aplikasi *service provider*, *monitoring identity provider* eksternal yang terintegrasi dengan SSO Auth0. *Dashboard admin* ini memudahkan *developer* dalam *monitoring* banyak hal. Penggunaan Auth0 sebagai *single sign-on* gratis dibatasi hanya fitur sederhana, untuk penggunaan lebih, Auth0 memungut biaya.

Kelemahan Auth0 adalah Auth0 merupakan perangkat lunak *proprietary* milik Auth0 Inc. Produknya Auth0 tidak *open-source*, artinya tidak bisa dikembangkan lebih lanjut oleh orang umum. Pengembangan Auth0 hanya dapat dilakukan oleh pegawai Auth0 In. Selain itu, dalam penggunaan gratis, *developer* harus me-*redirect* aplikasi *service provider* ke halaman *login* yang di-*deploy* pada *cloud server* milik Auth0. Hal ini berarti seluruh data *user*, data aplikasi *service provider* tersimpan di basis data milik Auth0. Praktik ini tidak aman, terutama apabila data *user* yang disimpan merupakan data sensitif. Ini menyebabkan aplikasi *service provider* milik *developer* akan bergantung secara penuh terhadap ketersediaan *cloud server* milik Auth0. Apabila *cloud server* milik Auth0 *down*, maka aplikasi *service provider* tidak akan dapat melakukan segala hal yang berkaitan dengan manajemen *user* seperti autentikasi atau meminta data dari *user*. Untuk mengatasi permasalahan ini, Auth0 Inc menyediakan Auth0 agar dapat di-*deploy* secara *self-host* di *server* sendiri. Namun kelebihan ini hanya dapat dimanfaatkan bagi *developer* yang menggunakan Auth0 secara berbayar.

### WSO2 *Identity Server*

WSO2 *Identity Server* adalah produk SSO milik WSO2. WSO2 *Identity Server* dapat digunakan secara gratis karena *open-source*. Masyarakat umum secara luas dapat ikut berpartisipasi dalam mengembangkan WSO2 *Identity Server* melalui *pull request* di GitHub. WSO2 juga memungkinkan SSO di-*deploy* pada *server* privat (*self-hosted*). Dengan demikian, basis data dan media penyimpanan data tentang aplikasi *service provider* juga dapat dikontrol secara langsung oleh *developer* sehingga dapat dijamin sendiri keamanan dan *maintenance*/*uptime* *server*-nya. Seperti Auth0, WSO2 *Identity Server* juga memiliki *dashboard* yang digunakan untuk melakukan konfigurasi aplikasi *service provider*. Selain itu, WSO *Identity Server* juga mendukung banyak protokol SSO seperti SAML, OAuth, dan HTTP. WSO *Identity Server* juga menyediakan banyak integrasi dengan *identity provider* lain, seperti Facebook, Google, dan Yahoo.

Kelemahan utama dari WSO2 *Identity Server* adalah dokumentasi yang kurang rapi dan *learning curve* yang cukup tajam. *Sample application* yang sangat dasar untuk SSO dengan protokol SAML dan OAuth milik WSO2 *Identity Server* bahkan tidak bisa dijalankan. Dengan demikian, *developer* yang baru menggunakan teknologi *single sign-on* akan mengalami kesulitan dalam melakukan integrasi terhadap aplikasinya. Dengan tingkat kompleksitas aplikasi yang tinggi seperti ini, dukungan teknis *customer service* hanya dapat diperoleh apabila *developer* men-*subscribe* WSO2 *Identity Server* secara berbayar. WSO2 *Identity Server* (SSO *server*) membutuhkan *resource* memori yang tinggi ketika dijalankan, yakni sekitar 800 MB dan 500 MB *heap size*. Kebutuhan memori yang besar terjadi karena WSO2 *Identity Server* diimplementasi menggunakan bahasa pemrograman Java. WSO2 *Identity Server* menyediakan *cloud server* yang hanya bisa diakses secara berbayar. Selain itu, untuk *monitoring user*, WSO2 *Identity Server* tidak menyediakan *dashboard* secara terpusat dan harus dilakukan dari sisi aplikasi *service provider*. *Dashboard* yang disediakan WSO2 *Identity Server* hanya berfungsi sebagai *monitoring* aplikasi *service provider*, tetapi tidak untuk *monitoring user*.

### *Central Authentication Server*

*Central Authentication Server* (CAS) pada mulanya adalah sebuah protokol untuk *single sign-on*. CAS sekarang telah dikembangkan lebih lanjut menjadi *tool single sign-on* dan aplikasi *client*-nya banyak diimplementasikan menggunakan beberapa bahasa pemrograman. CAS merupakan *project open-source* milik Apereo. Dengan demikian, CAS *server* memungkinkan untuk di-*deploy* secara *self-hosted*. CAS mendukung beberapa protokol seperti CAS, SAML, OAuth, dan OpenID. CAS memiliki dokumentasi yang lengkap pada laman <https://apereo.github.io/cas/4.2.x/index.html>. CAS memiliki *standalone dashboard* (berbeda dari *server*) yang dapat digunakan untuk memonitor aplikasi *service provider*.

Walaupun memiliki dokumentasi yang lengkap, tetapi Apereo, organisasi pemilik *project* CAS, tidak menyediakan *sample application* yang bisa digunakan sebagai petunjuk dalam mengintegrasikan aplikasi dengan *Central Authentication Server*. Kelemahan CAS berikutnya adalah CAS membutuhkan memori besar untuk dijalankan. CAS *server* membutuhkan memori minimal 8 GB dan prosesor minimal 3,00 GHz (Apereo, 2015). Selain itu walaupun CAS memiliki *dashboard* untuk *monitoring* aplikasi *service provider*, CAS tidak memiliki *dashboard* untuk manajemen *user* sehingga *developer* harus membuat aplikasi *client* sendiri. Kelemahan lainnya dari CAS adalah pertukaran informasi antar komponen CAS dilakukan tidak menggunakan SSL yang menyebabkan kerentanan terhadap serangan *Man-in-the-Middle*. Selain itu, bahasa pemrograman *client* yang didukung oleh CAS hanya bahasa pemrograman kuno, seperti Java, .NET, PHP, dan Perl. Bahasa modern seperti Python, Go, Ruby tidak didukung karena CAS merupakan teknologi lama. CAS juga tidak mendukung *single sign-on* untuk *single page application* (SPA) yang menggunakan berbagai *framework* JavaScript seperti React, Redux, dan VueJS yang akhir-akhir ini sedang *trend*.

### Perbandingan *Web Single Sign-On*

Setelah melakukan analisis pada beberapa produk *single sign-on* pada subbab di atas, diperoleh beberapa poin yang dapat dibandingkan antara *web single sign-on* dengan yang lain. Berikut adalah ringkasan tabel perbandingan *web single sign-on* yang telah dianalisis.

Tabel ‎III.1 Perbandingan *Web Single Sign-On*

| No | Tipe | Auth0 | WSO2 IS | CAS |
| --- | --- | --- | --- | --- |
| 1. | Kemudahan integrasi | Mudah | Sulit | Mudah |
| 2. | Dokumentasi | Bagus | Buruk | Bagus |
| 3. | *Sample application* | Ada | Ada, tidak bisa dijalankan | Tidak ada |
| 4. | Dukungan SDK *client* | Lengkap | Lengkap | Kurang lengkap, tidak terkini |
| 5. | Dukungan protokol lain (SAML, OAuth, dll) | Ada | Ada | Ada |
| 6. | *Dashboard* untuk *service* | Ada | Ada | Tidak |
| 7. | *Dashboard provisioning user* | Ada | Tidak | Tidak |
| 8. | Hak cipta | *Proprietary* | *Open-source* | *Open-source* |
| 9. | Tipe *server* | *Cloud* | *Self-hosted* | *Self-hosted* |
| 10. | Mekanisme SSO | *Redirect* | *Redirect* | API |
| 11. | SSL | Bisa | Bisa | Tidak |

## Analisis Solusi

Berdasarkan analisis permasalahan pada subbab sebelumnya, kali ini akan dibahas mengenai analisis solusi yang ditawarkan dalam tugas akhir ini. Secara garis besar, akan dibuat sebuah *single sign-on* baru yang memiliki fungsionalitas gabungan dari *web single sign-on* di atas dan mengurangi/menghapus kelemahan dari *web single sign-on* yang sudah ada.

### Karakteristik *Web Single Sign-On*

Dengan demikian, dalam tugas akhir ini ditawarkan *single sign-on* baru dengan karakteristik sebagai berikut.

Tabel ‎III.2 Karakteristik *Single Sign-On*

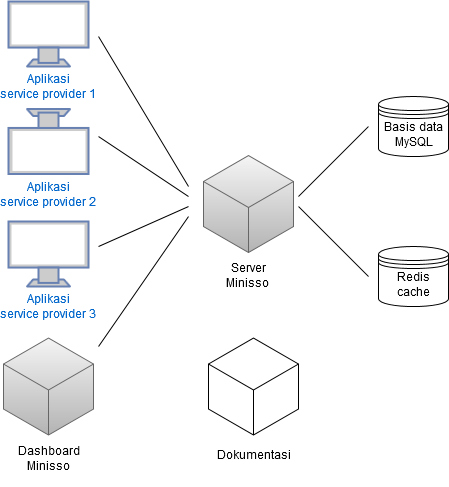
| No | Tipe | Solusi *single sign-on* |
| --- | --- | --- |
| 1. | Kemudahan integrasi | Mudah |
| 2. | Dokumentasi | Bagus |
| 3. | *Sample application* | Ada |
| 4. | Dukungan SDK *client* | Go, HTTP API |
| 5. | Dukungan protokol lain (SAML, OAuth, dll) | Tidak |
| 6. | *Dashboard* untuk *service* | Ada |
| 7. | *Dashboard provisioning user* | Ada |
| 8. | Hak cipta | *Open-source* |
| 9. | Tipe *server* | *Self-hosted* |
| 10. | Mekanisme SSO | API |
| 11. | SSL | Bisa |

Sebagai pengembangan dari *web single sign-on* yang sudah ada sebelumnya, pada tugas akhir ini akan dibangun *web single sign-on* baru dengan karakteristik seperti di atas. Secara general, karakteristik *web single sign-on* yang dibangun merupakan perbaikan dari kelemahan *web single sign-on* yang telah dianalisis pada bab III.1. Berikut adalah penjelasan dari setiap poin karakteristik dari *single sign-on* yang akan dibangun.

1. **Kemudahan integrasi**. *Single sign-on* akan menggunakan protokol HTTP dan JSON. Saat ini, umumnya *web application* menggunakan protokol HTTP untuk berkomunikasi antar *service*. Dengan pembangunan SSO menggunakan protokol *single sign-on*, harapannya adalah *service* dengan teknologi dan bahasa apapun dapat diintegrasi *web* SSO dengan mudah. Diimplementasinya SSO menggunakan protokol HTTP memberikan ruang cerah sehingga SSO yang berkarakter *language-agnostic* dapat dicapai.
2. **Dokumentasi.** Dokumentasi yang akan dibangun haruslah mudah dipahami sehingga *developer* dapat menggunakan *web* SSO ini dengan mudah.
3. **Adanya *sample application*.** Dokumentasi saja tidak cukup untuk sebuah perangkat lunak supaya dirilis secara *open-source*. Perangkat lunak harus memiliki *sample application* yang benar-benar berhasil dijalankan.
4. **Dukungan SDK *client***. Kelebihan dari *web* SSO yang sudah terkenal dan besar adalah mereka memiliki dukungan SDK yang lengkap. Dukungan yang lengkap bisa terjadi karena jumlah tim yang besar dan memiliki basis *developer* yang kuat. Untuk pengerjaan tugas akhir ini, karena keterbatasan sumber daya manusia, *client* hanya didukung untuk bahasa Go saja. Sementara, untuk bahasa lain, SSO yang akan dibangun hanya mendukung integrasi melalui HTTP API.
5. **Dukungan protokol lain (SAML dan OAuth).** Protokol lain seperti SAML dan OAuth tidak didukung dalam *web single sign-on* ini. Web *single sign-on* yang akan dibangun bertujuan untuk melakukan simplifikasi dari teknologi *single sign-on* yang sudah ada dan rumit, menjadi *service* sederhana yang mudah diintegrasikan.
6. ***Dashboard* untuk *service*.** *Single sign-on* harus memiliki *dashboard* yang dapat digunakan untuk mengelola *application service provider*. *Dashboard* ini hanya dapat diakses oleh *admin*.
7. ***Dashboard provisioning user*.** *Single sign-on* harus memiliki *dashboard* yang digunakan untuk pengelolaan *user*.
8. **Hak cipta *open-source*.** *Web single sign-on* akan di-*open-source* ke masyarakat luas supaya dapat dikembangkan dengan bebas.
9. **Tipe *server* *self-hosted***. *Server web single sign-on* dapat dijalankan pada mesin sendiri. Dengan demikian, aplikasi *service provider* yang menggunakan *single sign-on* tidak perlu bergantung pada infrastruktur *cloud server single sign-on.*
10. **Mekanisme SSO.** *Web**single sign-on* akan diimplementasi dalam bentuk API. *Web**single sign-on* dapat menerima *request* dari aplikasi *service provider* untuk *login*/*register* atau manajemen *user*.
11. **SSL.** Seluruhtransport informasi pada sistem SSO dilakukan mengunakan *layer* yang aman sehingga aman dari serangan *Man-in-the-Middle*.

### Kebutuhan Sistem

Sistem yang dimaksud pada subbab ini adalah keseluruhan sistem yang terlibat dalam *single sign-on* termasuk *web single sign-on*, aplikasi *service provider*, *dashboard* untuk manajemen dan *monitoring*, dan dokumentasi. Berikut adalah gambaran secara garis besar antar komponen dalam sistem.



Gambar ‎III.1 Sistem *Single Sign-On*

Aplikasi *service provider* menggunakan *server single sign-on* untuk proses autentikasi. Untuk kemudahan penyebutan, sistem *single sign-on* yang dibangun berikutnya dinamai **Minisso**.

S*erver* Minisso menggunakan basis data persisten MySQL untuk penyimpanan data, dan Redis untuk *cache* sebagai media penyimpanan data sementara yang cepat. *Dashboard* Minisso akan menghubungi server Minisso untuk proses manajemen user dan manajemen service. Sementara itu akan ada dokumentasi *standalone* sebagai pedoman developer yang ingin mengintegrasikan aplikasinya dengan *single sign-on* Minisso.

*Server* Minisso membutuhkan sebuah media penyimpanan data yang persisten untuk menyimpan data data user. Database relasional MySQL dipilih karena MySQL mampu menyimpan data secara persisten.

Selain itu, *server* Minisso juga membutuhkan penyimpanan data temporary untuk menyimpan struktur data yang sifatnya tidak permanen dan akan hangus dalam periode waktu tertentu. Dalam hal ini, Redis *cache* dipilih karena kemampuannya menghapus data secara otomatis setelah masa berlakunya habis (*expired*). Selain itu, Redis memiliki kemampuan *read* dan *write* yang lebih cepat dibandingkan database relasional karena seluruh data pada Redis disimpan pada memori (RAM).

Berdasarkan analisis yang dilakukan sebelumnya, Minissosetidaknya memiliki spesifikasi kebutuhan sistem seperti yang dijabarkan pada tabel III.3.

Tabel ‎III.3 Kebutuhan Sistem Minisso

| Kode | Kebutuhan Sistem Minisso |
| --- | --- |
| KS1 | Ada aplikasi *service* *provider* yang terintegrasi dengan server Minisso. |
| KS2 | Ada dashboard yang berfungsi sebagai monitoring aplikasi dan user yang terdaftar pada *single sign-on* Minisso. |
| KS3 | Ada dokumentasi yang berfungsi sebagai pegangan *developer* dalam mengintegrasikan aplikasinya dengan *single sign-on* Minisso. |
| KS4 | Ada media penyimpanan data persisten dan sementara. |
| KS5 | Ada *server* yang bertugas mengontrol seluruh logika dan kerja *single sign-on*. |
| KS6 | *Developer* mampu mengintegrasikan *single sign-on* dengan mudah. |
| KS7 | Cara kerja *single sign-on* mudah dipahami. |
| KS8 | Aplikasi *service provider* dan *user* dapat dimonitor. |
| KS9 | *Single sign-on* tidak berbayar, mudah dimodifikasi, dan dikembangkan lebih lanjut. |
| KS10 | Setiap informasi yang ditukarkan antar komponen sistem dilakukan secara aman. |
| KS11 | Sistem *single sign-on* ringan dijalankan. |

Berdasarkan kebutuhan sistem di atas, dalam tugas akhir ini dikembangkan dua komponen *single sign-on* yakni:

1. *Server* Minisso, bertugas mengontrol seluruh logika *single sign-on*.
2. *Dashboard* Minisso, sebagai pelengkap *single sign-on*, berfungsi sebagai monitoring aplikasi dan user yang terdaftar pada *single sign-on* Minisso.

Letak kedudukan komponen ini di antara komponen lain dapat dilihat pada gambar III.1. Pada gambar tersebut, komponen yang dibangun merupakan komponen yang diarsir.

### Kebutuhan Fungsional Minisso

Kebutuhan fungsional yang dibahas pada subbab ini mencakup kebutuhan fungsional *server* Minisso dan *dashboard* Minisso. Berdasarkan kebutuhan sistem yang telah diidentifikasi sebelumnya, Minissosetidaknya memiliki spesifikasi kebutuhan fungsional seperti yang dijabarkan pada tabel III.4.

Tabel ‎III.4 Kebutuhan Fungsional Minisso

| Kode | Kebutuhan Fungsional Minisso |
| --- | --- |
| KF1 | *Server* mampu menerima pendaftaran *user* baru. |
| KF2 | *Server* mampu menerima *login user* dan mengembalikan *token*. |
| KF3 | *Server* mampu memberikan detil informasi *user* ketika *token* terbukti *valid*. |
| KF4 | *Dashboard* mampu melakukan manajemen aplikasi *service provider* (*create*, *read*, *update*, *delete*) |
| KF5 | *Dashboard* mampu melakukan manajemen *user*. |
| KF6 | *Server* mampu menerima *request* dari aplikasi *service provider* untuk proses autentikasi *user* untuk keperluan *single sign-on*. |

### Kebutuhan Non-Fungsional Minisso

Berdasarkan kebutuhan sistem yang telah diidentifikasi sebelumnya, Minissosetidaknya memiliki spesifikasi kebutuhan non-fungsional seperti yang dijabarkan pada tabel III.5.

Tabel ‎III.5 Kebutuhan Non-Fungsional Minisso

| Kode | Kebutuhan Non-Fungsional Minisso |
| --- | --- |
| KNF1 | Terdapat dokumentasi yang runtut dan mudah dipahami. |
| KNF2 | Terdapat contoh aplikasi *service provider* yang menggunakan *single sign-on* dan langsung bisa digunakan. |
| KNF3 | *Single sign-on* haruslah *open-source* dan bisa dimodifikasi setiap orang. |
| KNF4 | Transport data dilakukan menggunakan SSL. |
| KNF5 | Kebutuhan memori untuk menjalankan *server single sign-on* rendah. |

### Diagram *Use Case*

Berdasarkan kebutuhan fungsional yang telah dipaparkan pada subbab sebelumnya, maka *use case diagram* dari Minisso adalah seperti pada gambar III.2 berikut. Diagram *use case* Minisso ini mencakup *server* Minisso dan *dashboard* Minisso, yang kedudukannya secara lebih jelas dapat dilihat pada gambar III.1.



Gambar ‎III.2 *Use Case* Minisso

Berikut adalah penjelasan dari masing-masing aktor yang terlibat dalam *use case* Minisso.

1. Aplikasi *service provider*. Merupakan aplikasi *service provider* yang menggunakan Minisso sebagai autentikasi.
2. *Developer*. Adalah entitas yang memiliki akses terhadap *single sign-on*. Karena developer merupakan pemilik *single sign-on*, oleh karena itu *developer* berhak untuk akses manajemen *single sign-on* seperti *monitoring user* dan aplikasi *service provider*.
3. Basis data relasional. Basis data relasional ini digunakan untuk media penyimpanan data permanen seperti data *user*, aplikasi *service provider*, dan menyimpan *log history user*.
4. Redis *cache*. Redis adalah basis data no-SQL non-persisten yang berjalan di RAM, digunakan untuk menyimpan data sementara seperti token untuk keperluan manajemen user. Redis dipilih karena performansinya yang bagus.

## Rancangan Solusi

Pembahasan mengenai rancangan solusi dibagi menjadi dua bagian, yakni gambaran umum solusi dan desain *server/dashboard* Minisso.

### Gambaran Umum Solusi

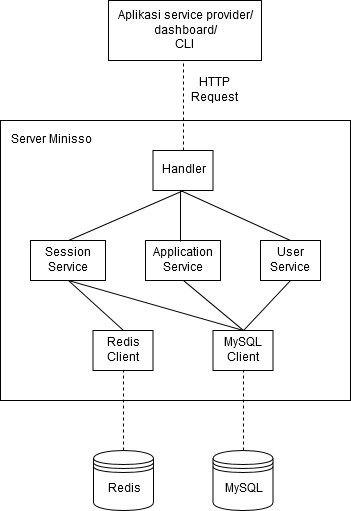
Berdasarkan analisis yang telah dilakukan, akan dibangun sebuah *web single sign-on* dengan karakteristik seperti yang sudah dipaparkan pada subbab III.2.1. Tipe *single sign-on* yang akan digunakan adalah REST API dan dilakukan menggunakan protokol HTTP menggunakan SSL untuk tingkat keamanan yang tinggi.

Kode sumber server Minisso dibuka secara umum (*open-source*) pada laman <https://github.com/luqmanarifin/minisso>. Dengan demikian, seluruh akan bisa menggunakan *server* Minisso untuk mengintegrasikan aplikasinya secara mudah. Dengan di-*open-source*-nya Minisso, artinya *developer* juga bisa mengembangkan Minisso secara bebas. Kelanjutan pengembangan Minisso yang mungkin contohnya adalah pengembangan fungsional SSO, semisal integrasi dengan *identity provider* eksternal, seperti Google atau Yahoo, atau penambahan data atribut *user* sehingga bermanfaat untuk *monitoring user*.

Contoh kemungkinan pengembangan Minisso yang lain adalah pengembangan *client* Minisso dengan bahasa lain, semisal Java, JavaScript, atau Ruby. *Server* Minisso hanya menyediakan API dengan protokol HTTP dengan tujuan supaya mudah diintegrasikan dengan sistem lain yang rata-rata sekarang ini berkomunikasi lewat protokol HTTP. Kelebihan lain menggunakan protokol HTTP adalah *server* tidak perlu mempedulikan bahasa yang digunakan oleh *client*. *Client* Minisso akan bertindak sebagai penghubung (*plug*) antara aplikasi *service provider* dengan *server* Minisso. Tentunya bahasa dan teknologi yang digunakan dalam *client* Minisso seharusnya disesuaikan dengan aplikasi dan *framework* yang digunakan oleh aplikasi *service provider*.

### Desain *Server dan Dashboard*

Dalam subbab ini, akan dijelaskan arsitektur beserta modul-modul yang berperan dalam Minisso.



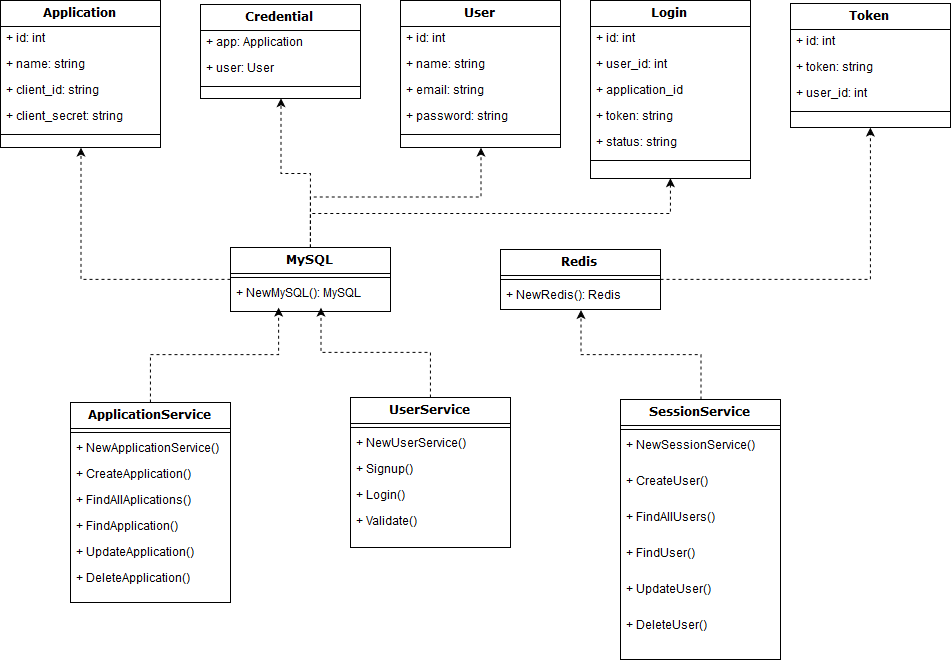
Gambar ‎III.3 Arsitektur Minisso

*Server* Minisso melibatkan modul-modul sebagai berikut.

1. ***Handler***. *Handler* berfungsi untuk menerima HTTP *request* dari alamat Minisso. *Handler* juga bertanggungjawab untuk mentransformasi data dari JSON (tipe data yang dapat dioper dengan mudah di prokotol HTTP) ke tipe data Go sehingga dapat diproses oleh bahasa pemrograman.
2. ***User Service*.** *User service* bertugas memproses seluruh aktivitas yang berkaitan dengan manajemen *user*, termasuk tambah *user*, hapus *user*, ubah *user*, cari *user*.
3. ***Application Service*.** *Application service* bertugas memproses seluruh aktivitas yang berkaitan dengan manajemen aplikasi yang terdaftar pada Minisso, termasuk tambah, hapus, baca, dan ubah aplikasi.
4. ***Session Service***. *Session service* bertugas untuk memproses seluruh proses yang berkaitan dengan manajemen *session user* yang sudah *login*. Cakupan modul ini termasuk *token*, riwayat *login* berhasil/gagal, dan lainnya.
5. **MySQL *client***. MySQL *client* bertugas untuk memproses seluruh *query* pada *database* relasional MySQL.
6. **Redis *client***. Redis *client* bertugas untuk memproses seluruh *query* yang melibatkan Redis *cache*.

#### Diagram Kelas

Arsitektur Minisso di atas menghasilkan diagram kelas sebagai berikut.



Gambar ‎III.4 Diagram Kelas Minisso

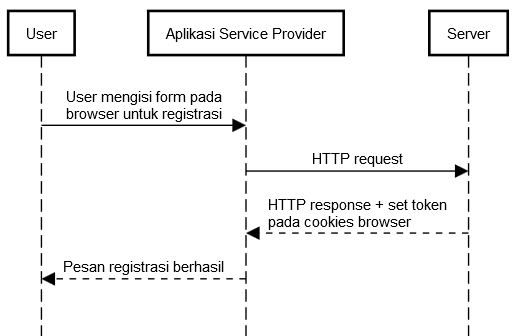
Berikut adalah penjelasan dari kelas yang dibuat:

1. Kelas pada baris atas meliputi *application*, *credential*, *user*, *login*, *token*, merupakan kelas yang merepresentasikan struktur data pada basis data. Seringkali juga disebut kelas model.
2. Kelas MySQL dan Redis merupakan kelas yang berfungsi sebagai *database client* yakni untuk melakukan koneksi pada database MySQL dan Redis.
3. Kelas ApplicationService, UserService, dan SessionService merupakan kelas *service* yang berguna untuk bertanggung jawab menerima HTTP *request* dari aplikasi *service provider*.

#### Diagram *Sequence*

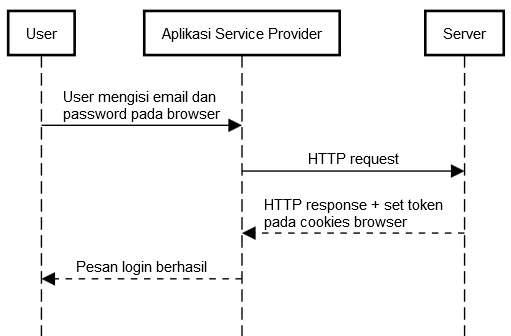
Selain diagram kelas di atas, salah satu komponen yang penting dalam pengembangan Minisso adalah bagaimana mekanisme kerjanya. Dalam subbab ini, akan dibahas alur kerja Minisso untuk tiap kebutuhan fungsional dalam bentuk diagram *sequence*.

1. KF1 server mampu menerima pendaftaran *user* baru



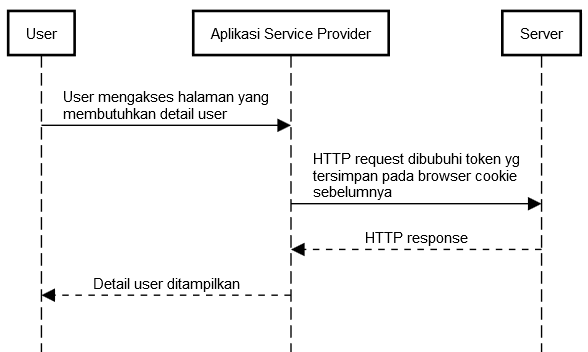
Gambar ‎III.5 Diagram *Sequence* KF1 Registrasi User

1. KF2 server mampu menerima *login* user dan mengembalikan token.



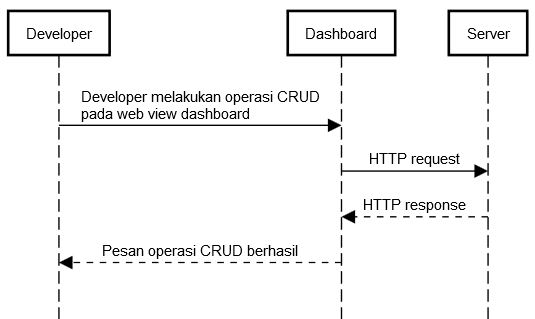
Gambar ‎III.6 Diagram *Sequence* KF2 Login User

1. KF3 server mampu memberikan detil informasi user ketika token terbukti valid



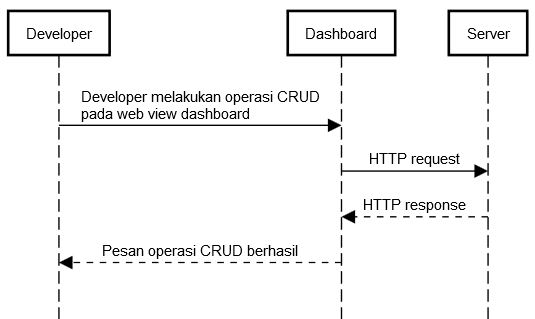
Gambar ‎III.7 Diagram *Sequence* KF3 Validasi Token

1. KF4 *dashboard* mampu melakukan manajemen aplikasi *service provider* berupa operasi CRUD (*create, read, update, delete*).



Gambar ‎III.8 Diagram *Sequence* KF4 Manajemen Aplikasi *Service Provider*

1. KF5 *dashboard* melakukan manajemen *user* berupa operasi CRUD.



Gambar ‎III.9 Diagram *Sequence* KF5 Manajemen User

### Pemanfaatan Token

Token dalam Minisso digunakan pada dua kasus.

1. Untuk **autentikasi user**. Token digunakan sebagai penanda apakah *user* telah login atau belum. Token disimpan sebagai *cookie* pada *browser* sementara dalam jangka waktu yang dapat dikonfigurasi lewat *dashboard*. Untuk memanfaatkan token ini, beberapa aplikasi *service provider* yang menggunakan Minisso secara bersamaan harus di-*deploy* pada domain yang sama, namun dalam subdomain yang berbeda.

Misal, konfigurasi subdomain yang benar antara aplikasi *service provider* A dan B yang sama-sama menggunakan SSO Minisso adalah: a.minisso.com dan b.minisso.com. Contoh konfigurasi domain yang salah adalah a.com dan b.com. Apabila domain berbeda, maka login antar aplikasi tidak dapat dilakukan secara terpusat menggunakan Minisso.

1. Untuk **autentikasi aplikasi *service provider***. Token diambil dari dashboard Minisso setelah mendaftarkan aplikasi *service provider*. Token ini disimpan pada aplikasi *service provider*, dan nanti dibutuhkan setiap kali aplikasi *service provider* berkomunikasi dengan server Minisso.

Detil dan standard format komunikasi antar komponen Minisso terlampir pada **Lampiran A**.

# IMPLEMENTASI DAN PENGUJIAN

Pembahasan pada bab ini akan dibagi menjadi dua bagian. Bagian pertama akan dipaparkan tentang lingkungan dan implementasi. Sedangkan bagian kedua dibahas mengenai pengujian Minisso beserta analisisnya.

## Implementasi

Bagian implementasi ini dibagi menjadi 4 bagian. Bagian pertama dibahas mengenai lingkungan implementasi. Bagian kedua dibahas mengenai implementasi *server* Minisso. Bagian ketiga dibahas mengenai implementasi *dashboard* Minisso. Bagian keempat dibahas mengenai implementasi aplikasi *service provider*.

### Lingkungan Implementasi

Tabel ‎IV.1 Lingkungan Implementasi

| Perihal | Spesifikasi |
| --- | --- |
| Sistem Operasi | Ubuntu 16.04.5 LTS |
| CPU | Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz |
| RAM | 8 GB |
| Bahasa Pemrograman | Go |
| Go Compiler | go1.10.1 |
| MySQL | mysql Ver 14.14 Distrib 5.7.23 |
| Redis | Redis server v=3.0.6 |

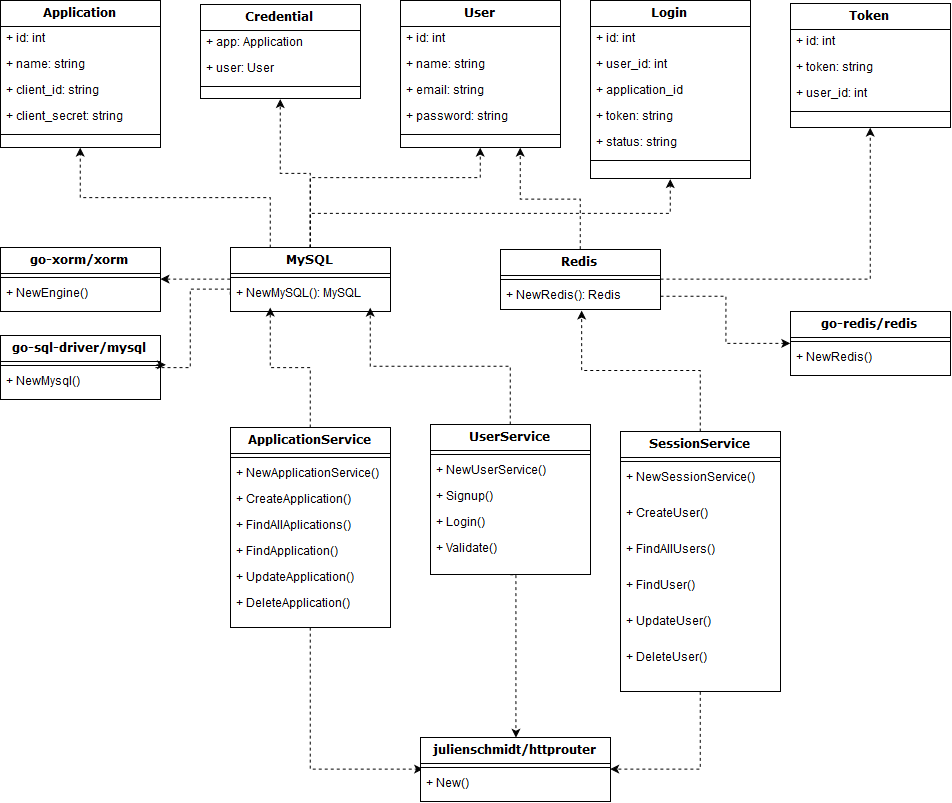
### Implementasi *Server* Minisso

*Server* Minisso diimplementasi menggunakan bahasa pemrograman Go. Bahasa pemrograman Go dipilih karena keutamannya dalam membangun *service*. Pengembangan *service* menggunakan Go memiliki kurva pembelajaran yang rendah. Sementara itu, *service* yang dibuat menggunakan Go lebih mampu menahan lebih banyak *request* (Jackson, 2017). Bahasa Go juga dikompilasi langsung ke bahasa mesin tanpa mesin virtual seperti Java atau interpreter seperti Python, menyebabkan kode yang ditulis dalam bahasa Go dieksekusi dengan cepat. Go juga memiliki *standard library* lengkap yang dikelola dengan baik. Pembuatan *service* dalam Go dapat dilakukan hanya dengan *standard library* saja karena rutinitas untuk membuat *web server* seperti pengambilan *request* HTTP, *parsing* JSON, dan enkripsi sudah didukung oleh *standard library* Go.

Dalam *server* Minisso, *web single sign-on* yang diusulkan dalam tugas akhir ini, terdapat beberapa *package* yang diimplementasi.

1. **Database**. Bertanggungjawab untuk melakukan koneksi ke *database* baik persisten maupun *cache*. *Package database* memiliki beberapa modul.
2. Modul mysql. Berfungsi melakukan *query* ke database relasional mysql.
3. Modul redis. Berfungsi melakukan *query* ke server Redis untuk cache.
4. **Model**. Merupakan *package* yang berisi kelas hasil interpretasi dari tabel dalam basis data relasional. *Package* model memiliki beberapa kelas.
5. *Application*. Merupakan perwujudan dari aplikasi *service provider*.
6. *Login*. Merupakan perwujudan dari *history login* dari masing-masing *user*.
7. *Token*. Merupakan perwujudan dari *token* yang didapat *user* ketika *login* ke dalam aplikasi.
8. *User*. Merupakan perwujudan dari *user* yang diperbolehkan *login* ke dalam aplikasi *service provider* yang menggunakan autentikasi *single sign-on*.
9. **Service**. Merupakan *package* yang berisi modul inti logika dalam *single sign-on*. Setiap modul dalam *package* ini bertugas untuk mengelola langsung request HTTP dari *single sign-on (request handling)*. *Package* *service* berisi beberapa modul.
10. Modul *application*. Berisi kumpulan fungsi dan prosedur yang berkaitan dengan manajemen aplikasi *service provider*.
11. Modul *common*. Berisi kumpulan fungsi bantuan yang digunakan oleh modul lain dalam *package service*.
12. Modul *session*. Berisi kumpulan fungsi yang berkaitan dengan manajemen token dan *session*.
13. Modul *user*. Berisi kumpulan fungsi yang berkaitan dengan manajemen *user*.
14. **Vendor**. Merupakan *package* yang berisi dependensi Minisso terhadap package eksternal, contohnya redis *client*, SQL *client*, httprouter, dan lain-lain.
15. **Main program**. Merupakan program inti yang harus dijalankan supaya supaya Minisso up. Bertugas membuka *file* *environment variable*, melakukan *binding* antara alamat URL dengan fungsi *handler*, dan menjalankan *service web single sign-on* pada *port* tertentu.

Berikut adalah diagram kelas dari *package* yang telah dijelaskan di atas.



Gambar ‎IV.1 Diagram Kelas Implementasi Server Minisso

Diagram kelas implementasi di atas hampir sama seperti diagram kelas desain yang telah dijelaskan pada gambar III.4. Namun pada diagram kelas implementasi ini, ditambahkan beberapa kelas dependensi eksternal milik Go yang digunakan sebagai helper untuk implementasi. Kelas-kelas eksternal tersebut adalah:

1. *go-xorm/xorm*, adalah ORM (*Object Relation Model*) yang mengimplementasi database *client* MySQL dalam bahasa Go.
2. *go-sql-driver/mysql*, adalah kelas pelengkap XORM di atas yang digunakan untuk memperjelas bahwa XORM menggunakan dialek MySQL, bukan basis data relasional lain misal MariaDB, PostgreSQL, dst.
3. *go-redis/redis*, adalah kelas dalam bahasa Go sebagai *client* Redis.
4. *julienschmidt/httprouter*, adalah kelas yang mengimplementasi fungsi HTTP *router* untuk keperluan *binding endpoint* REST API ke masing-masing fungsi *handler*-nya dalam Go.

### Implementasi *Dashboard* Minisso

*Dashboard* Minisso diimplementasi menggunakan React (salah satu *framework* JavaScript) dan Material UI *component*. React dipilih karena popular dalam pengembangan aplikasi *front-end* masa kini. Sementara itu, Material UI merupakan komponen dalam React yang mengimplementasi desain dengan standar Google.

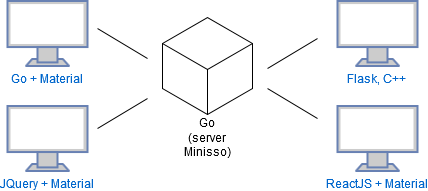
Pada *dashboard* Minisso, ada beberapa package yang diimplementasi.

1. **Assets**. Merupakan kumpulan dependensi eksternal dari *source code*.
2. **Components**. Merupakan mikro-komponen *front-end* general yang digunakan oleh *package layouts* dan *views* dalam mengimplementasi tampilan *dashboard*. Contoh modul dalam *package* ini adalah: Table, CustomInput, Footer, Header, Sidebar.
3. **Layouts**. Merupakan *template* *static dashboard* default yang tidak berubah secara dinamik. Modul yang diimplementasi dalam *package* ini adalah menu yang terletak di samping dashboard (*sidebar menu*).
4. **Routes**. Merupakan *package* yang bertanggung jawab atas mendelegasikan alamat tertentu ke halaman yang benar.
5. **Util**. Merupakan *package* yang berisi kumpulan fungsi dan prosedur pembantu yang digunakan bersama-sama oleh *package* lain.
6. **Variables**. Merupakan *package* yang berisi kumpulan konstanta yang digunakan bersama-sama oleh package lain.
7. **Views**. Merupakan kumpulan laman HTML, JavaScript, CSS, yang akan ditampilkan langsung ke *browser user*. Contoh modul yang diimplementasi adalah: UserPage, ApplicationPage, dan IndexPage. Package *views* dependen terhadap *package* *components*.

### Implementasi Aplikasi *Service Provider*

Dibutuhkan dua atau lebih aplikasi *service provider* untuk menguji Minisso apakah sudah berjalan sesuai yang diharapkan. Namun, dalam pengembangan tugas akhir ini, aplikasi *service provider* bukan merupakan fokus utama.

Aplikasi *service provider* tetap dibutuhkan untuk menguji kemampuan Minisso dalam melakukan integrasi autentikasi beberapa aplikasi. Rancangan desain aplikasi *service provider* adalah sebagai berikut.



Gambar ‎IV.2 Aplikasi *Service Provider*

Walaupun begitu, berikut adalah empat aplikasi *service provider* yang diimplementasi untuk keperluan pengujian Minisso, keempat aplikasi tersebut yakni:

1. Aplikasi berbasis Go (integrasi sisi *server*).
2. Aplikasi berbasis *framework* JavaScript JQuery (integrasi sisi *browser*).
3. Aplikasi berbasis Python (Flask) (integrasi sisi *server*).
4. Aplikasi berbasis *framework* JavaScript ReactJS (integrasi sisi *browser*).

Ditemukan bahwa cara pengintegrasian aplikasi membentuk pola. Cara pengintegrasian aplikasi dengan integrasi pada sisi *server* cenderung berbeda dengan sisi *browser*. Sementara itu, untuk integrasi sisi *server* walaupun *platform* berbeda yakni Go dan Python, caranya cenderung sama. Begitu pula pada integrasi sisi *browser*.

Pada subbab di bawah ini akan dibagi menjadi tiga bahasan yakni prasyarat aplikasi yang harus dilakukan untuk integrasi aplikasi dengan Minisso (berlaku untuk aplikasi sisi *server* dan *browser*), langkah integrasi sisi *server*, dan langkah integrasi sisi *browser*.

#### Prasyarat Aplikasi

Untuk menggunakan *single sign-on* Minisso, aplikasi *service provider* harus terdaftar pada *dashboard* Minisso. Aplikasi yang telah terdaftar pada *dashboard* akan mendapatkan *credential* aplikasi yakni CLIENT\_ID dan CLIENT\_SECRET berupa string random. Credential ini berguna sebagai jaminan bahwa aplikasi *service provider* yang berkomunikasi dengan *single sign-on* Minisso merupakan aplikasi yang terautorisasi untuk mencegah serangan *man-in-the-middle*. Tanpa *credential*, pihak ketiga dapat berpura-pura menjadi aplikasi *service provider* kemudian menghubungi Minisso dan mendapatkan data sensitif user.

#### Integrasi Sisi *Server* (*Backend*)

Yang termasuk dalam golongan aplikasi sisi *server* adalah aplikasi dengan *platform* Ruby on Rails, Go, Java, PHP, dan bahasa pemrograman sisi *server* lainnya. Berikut ini beberapa hal yang perlu diperhatikan untuk mengintegrasi aplikasi dari sisi *server*.

1. Menambah kelas *application* dan *user* dengan properti sebagai berikut.

|  |
| --- |
| type Application struct {  ClientId string  ClientSecret string  }  type User struct {  Id int64  UserId string  FirstName string  LastName string  Picture string  Gender string  Email string  Role string  LatestLogin time.Time  LastIp string  Connection string  CreatedAt time.Time  UpdatedAt time.Time  } |

1. Mengubah implementasi fungsi yang bertugas untuk registrasi, *login*, dan fungsi untuk mengambil detail *user* dengan HTTP *request* ke *server* Minisso. Format *request* sudah dijelaskan pada subbab IV.2.2.2.

Berikut adalah tahapan yang terjadi ketika aplikasi *service provider* sisi *server* berkomunikasi dengan server Minisso.

1. Aplikasi *service provider* mengirimkan HTTP *request* ke *server* Minisso, dengan salah satu format *request* (register, *login*, atau validasi token).
2. Server Minisso memberikan *response* berupa data yang dibutuhkan, atau pesan *error* apabila permintaan tidak sesuai.

#### Integrasi Sisi *Browser* (*Front-End*)

Yang termasuk dalam golongan aplikasi sisi browser adalah aplikasi dengan *platform* JavaScript dengan berbagai varian *framework*-nya seperti React, JQuery, AngularJS, dan lainnya. Berikut ini beberapa hal yang perlu diperhatikan untuk mengintegrasi aplikasi dari sisi *browser*.

1. Mengubah implementasi fungsi yang bertugas untuk registrasi, *login*, dan fungsi untuk mengambil detail *user* dengan HTTP *request* ke *server* Minisso. Format *request* sudah dijelaskan pada subbab IV.2.2.2
2. Menambahkan properti *Access-Control-Allow-Origin* pada HTTP *header* *request* untuk memperbolehkan CORS.

Berikut adalah tahapan yang terjadi ketika aplikasi *service provider* sisi *browser* berkomunikasi dengan *server* Minisso.

1. Aplikasi *service provider* mengirimkan request CORS *preflight*, yakni *request* ke *server* Minisso untuk mengecek apakah protokol CORS dipahami.
2. *Server* Minisso memberikan *response* bahwa protokol CORS bisa dilakukan.
3. Aplikasi *service provider* mengirimkan HTTP *request* ke *server* Minisso, dengan salah satu format *request* (registrasi, *login*, atau validasi token).
4. Server Minisso memberikan *response* berupa data yang dibutuhkan, atau pesan *error* apabila permintaan tidak sesuai.

## Pengujian

Pengujian yang dilakukan merupakan pengujian fungsional yang menggunakan integrasi dari beberapa aplikasi *service provider* yang terhubung dengan *single sign-on* Minisso. Pengujian fungsional merupakan pengujian secara kualitatif berdasarkan kebutuhan fungsional yang sudah dideklarasi sebelumnya. Pengujian *server single sign-on* Minisso dilakukan dengan menggunakan skenario uji. Skenario uji dapat dilihat pada Tabel IV.2 hingga Tabel IV.7. Berikut adalah aplikasi service provider yang digunakan dalam pengujian:

1. Aplikasi berbasis Go (integrasi sisi *server*).
2. Aplikasi berbasis *framework* JavaScript JQuery (integrasi sisi *browser*).
3. Aplikasi berbasis Python (Flask) (integrasi sisi *server*).
4. Aplikasi berbasis *framework* JavaScript ReactJS (integrasi sisi *browser*).

Terdapat enam kebutuhan fungsional yang akan diuji sesuai dengan kebutuhan fungsional pada bab 3. Kebutuhan fungsional yang diuji adalah sebagai berikut.

### Pengujian Fungsional KF1

Kebutuhan fungsionalitas KF1 adalah: mampu menerima pendaftaran *user* baru.

Tabel ‎IV.2 Pengujian Fungsionalitas KF1 Registrasi *User*

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| Kasus normal | 1. Membuka aplikasi 1 2. Registrasi pada aplikasi 1 3. Mengisi form termasuk *email* dan *password* 4. Membuka aplikasi 1 dan 2 | Pengguna baru terdaftar dan langsung *login* pada kedua aplikasi | Sesuai |
| Registrasi user dengan *credential* aplikasi yang keliru | 1. Membuka aplikasi 1 yang setting *credential* aplikasinya keliru 2. Mengisi *form* termasuk *email* dan *password* 3. Registrasi pada aplikasi 1 | *User* gagal terdaftar | Sesuai |
| Registrasi *user* dengan *email* yang sudah pernah didaftarkan sebelumnya | 1. Membuka aplikasi 1 2. Mendaftarkan *user* baru menggunakan email yang telah digunakan sebelumnya 3. Registrasi pada aplikasi 1 | *User* gagal terdaftar karena *email* sudah pernah didaftarkan sebelumnya. | Sesuai |
| Registrasi *user* dengan *email* yang tidak valid | 1. Membuka aplikasi 1 2. Mendaftarkan *user* baru menggunakan *email* yang tidak valid 3. Registrasi pada aplikasi 1 | *User* gagal terdaftar karena *email* tidak valid | Sesuai |
| Registrasi *user* dengan *password* yang terlalu pendek | 1. Membuka aplikasi 1 2. Mendaftarkan user baru menggunakan *password* yang terlalu pendek 3. Registrasi pada aplikasi 1 | *User* gagal terdaftar karena *password* tidak valid | Sesuai |

### Pengujian Fungsional KF2

Kebutuhan fungsionalitas KF2 adalah: mampu menerima *login* *user* dan mengembalikan token.

Tabel ‎IV.3 Pengujian Fungsionalitas KF2 *Login* *User*

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| *Login* normal | 1. Membuka aplikasi 1 2. Mengisi *email* dan *password* yang benar 3. Login pada aplikasi 1 | *Login* *user* berhasil | Sesuai |
| *Login* dengan pasangan *email* dan *password* yang keliru | 1. Membuka aplikasi 1 2. Mengisi *email* dan *password* yang keliru 3. *Login* pada aplikasi 1 | *Login* user gagal karena *password* salah | Sesuai |
| *Login* dengan *email* yang belum pernah terdaftar | 1. Membuka aplikasi 1 2. Mengisi dengan *email* yang belum terdaftar sebelumnya 3. *Login* pada aplikasi 1 | *Login* *user* gagal karena *email* tidak dikenali | Sesuai |
| *Login* pada aplikasi 1, *refresh* aplikasi 2 | 1. Membuka aplikasi 1 2. Mengisi *email* dan *password* yang benar 3. *Login* pada aplikasi 1 4. *Refresh* aplikasi 2 | *User* berhasil *login* di kedua aplikasi | Sesuai |
| *Login* pada aplikasi 2, *refresh* aplikasi 1 | 1. Membuka aplikasi 2 2. Mengisi *email* dan *password* yang benar 3. *Login* pada aplikasi 2 4. *Refresh* aplikasi 1 | *User* berhasil *login* di kedua aplikasi | Sesuai |

### Pengujian Fungsional KF3

Kebutuhan fungsionalitas KF3 adalah: mampu memberikan detil informasi *user* ketika token terbukti valid.

Tabel ‎IV.4 Pengujian Fungsionalitas KF3 Verifikasi Token

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| Verifikasi token yang valid | 1. Membuka aplikasi 1 dalam keadaan belum *login*. 2. Membuka aplikasi 2 dari *browser* lain dan mengambil *cookie*-nya. 3. Meng-*update* *cookie* pada *browser* aplikasi 1. 4. *Refresh* aplikasi 1 | Berhasil *login* pada aplikasi 1 | Sesuai |
| Verifikasi token yang keliru | 1. Membuka aplikasi 1. 2. *Login* dengan mengisi *email* dan *password* yang benar. 3. Ganti *cookie* pada *browser* dengan *cookie* yang salah. 4. *Refresh* aplikasi 1 | Keluar dari aplikasi 1 | Sesuai |

### Pengujian Fungsional KF4

Kebutuhan fungsionalitas KF4 adalah: mampu melakukan manajemen aplikasi *service provider* (*create, read, update, delete*).

Tabel ‎IV.5 Pengujian Fungsionalitas KF4 Manajemen Aplikasi

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| Menambahkan aplikasi baru | 1. Membuka *dashboard* 2. Mengisi *form* penambahan aplikasi 3. *Submit* | Aplikasi berhasil ditambahkan dan berhasil *generate* client\_id dan client\_secret secara random | Sesuai |
| Menampilkan aplikasi | 1. Membuka *dashboard* 2. Membuka daftar aplikasi | Seluruh aplikasi yang terdaftar muncul | Sesuai |
| Mengubah detail aplikasi | 1. Membuka *dashboard* 2. Membuka daftar aplikasi 3. Memilih aplikasi yang akan diubah detailnya 4. Mengubah detail aplikasi | Aplikasi terubah | Sesuai |
| Menghapus aplikasi | 1. Membuka *dashboard* 2. Membuka daftar aplikasi 3. Menghapus salah satu aplikasi | Aplikasi terhapus | Sesuai |

### Pengujian Fungsional KF5

Kebutuhan fungsionalitas KF5 adalah: mampu melakukan manajemen user.

Tabel ‎IV.6 Pengujian Fungsionalitas KF5 Manajemen User

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| Menambahkan *user* baru | 1. Membuka *dashboard* 2. Mengisi *form* penambahan user 3. *Submit* | *User* berhasil ditambahkan | Sesuai |
| Menampilkan *user* | 1. Membuka *dashboard* 2. Membuka daftar *user* | Seluruh *user* yang terdaftar muncul | Sesuai |
| Mengubah detail *user* | 1. Membuka *dashboard* 2. Membuka daftar *user* 3. Memilih *user* yang akan diubah detailnya 4. Mengubah detail *user* | *User* terubah | Sesuai |
| Menghapus *user* | 1. Membuka *dashboard* 2. Membuka daftar *user* 3. Menghapus salah satu *user* | *User* terhapus | Sesuai |

### Pengujian Fungsional KF6

Kebutuhan fungsionalitas KF6 adalah: mampu menerima *request* dari aplikasi *service provider* untuk proses autentikasi user untuk keperluan *single sign-on*.

Tabel ‎IV.7 Pengujian Fungsionalitas KF6 Menerima *Request*

| Deskripsi | Prosedur Pengujian | Kriteria Evaluasi Hasil | Hasil yang Didapat |
| --- | --- | --- | --- |
| Meminta *request* HTTP ke API server Minisso | 1. Membuka Postmanatau aplikasi simulasi HTTP *request* sejenis 2. Mengirim *payload* yang benar sesuai kontrak API Minisso | Status *code* *response* 200 | Sesuai |

## Analisis Hasil Uji

Seluruh hasil pengujian telah dinyatakan berhasil. Minisso dapat diintegrasikan oleh aplikasi yang benar-benar baru diimplementasi, aplikasi yang sudah ada, aplikasi sisi *server*, atau aplikasi sisi *browser*. Ada beberapa langkah yang berbeda dalam mengintegrasi aplikasi sisi *server* dengan sisi *browser*. Implementasi perubahan pada aplikasi yang pernah ada juga berhasil teruji pada seluruh skenario uji pada subbab IV.2.

# KESIMPULAN DAN SARAN

## Kesimpulan

Kesimpulan yang diambil dari pengembangan *single sign-on* yang dinamai Minisso pada tugas akhir ini adalah:

1. *Single sign-on* dapat diimplementasi sendiri dengan mekanisme yang jauh lebih sederhana menggunakan protokol HTTP. Protokol HTTP ini dapat menjamin bahwa server SSO dapat diintegrasikan dengan aplikasi *service provider* mengunakan teknologi dan bahasa apapun (*language-agnostic*).
2. Karakteristik Minisso yang diharapkan sebagai pengembangan dari server *single sign-on lain* adalah: mudah diintegrasikan, memiliki dokumentasi bagi *developer* yang berisi kontrak API, memiliki *sample application* yang dapat dijalankan, memiliki *dashboard* untuk *monitoring*, *open-source,* *server* yang dapat di-*host* di mesin sendiri, dan komunikasi antar komponen SSO dilakukan menggunakan transport yang aman.
3. Kontrak API penting dalam menstandarkan komunikasi server Minisso yang dengan aplikasi *service provider* milik *developer*. Kontrak API ini seperti janji yang harus ditepati dan harus konsisten di antara kedua belah pihak baik *server* Minisso maupun aplikasi *service provider*.
4. Ada perbedaan cara pengintegrasian aplikasi *service provider* berbasis *server side* dengan *browser side*. Aplikasi sisi *server* mampu melakukan request ke *server* Minisso secara langsung, sementara aplikasi sisi *browser* harus melakukan tambahan properti pada HTTP *request header* karena masalah CORS.

## Saran

Pengembangan dan peningkatan yang dapat dilakukan terkait tugas akhir ini adalah:

1. Menambahkan dukungan protokol lain seperti SAML, OAuth, OpenID.
2. Menambahkan tambahan dukungan SDK *client* menggunakan bahasa lain selain Go.
3. Menambahkan integrasi dengan *identity provider* eksternal lain semisal Google *Sign-In*, Facebook, Twitter, GitHub, dan lain-lain.
4. Meningkatkan performansi dengan menggunakan pemrograman paralel. Salah satu alternatifnya adalah menggunakan Go Routine karena server Minisso diimplementasi menggunakan bahasa Go.

DAFTAR PUSTAKA

Adabala, S., Matsunaga, A., Tsugawa, M., Figueiredo, R., & Fortes, J. A. B. *Single Sign-On in In-VIGO: Role-based Access via Delegation Mechanisms Using Short-lived User Identities*. ACIS Laboratory, University of Florida, Gainesville.

Ardagna, Claudio A., et al. (2006). *CAS++: an Open Source Single Sign-On Solution for Secure e-Services*. Crema: University Degli Study di Milano.

Apereo. (2015). Installation Requirements for CAS | Enterprise Single Sign-On for All. <https://apereo.github.io/cas/development/planning/Installation-Requirements.html#servlet-containers>. Diakses pada 25 Juli 2017.

Atricore. (2010). Simplified Identity an Access | About JOSSO. <http://www.josso.org/>. Diakses pada 18 Juli 2018.

Arianezhad, M., Kelley, T., Camp, L. J., & Stebila, D. *Comparative Eye Tracking of Experts and Novices in Web Single Sign-on*. School of Engineering Science, Simon Fraser University, Burnaby, B.C., Canada. Indiana University Bloomington, IN. Science and Engineering Faculty, Queensland University of Technology, Brisbane, Queensland, Australia.

Bauer, L., Bravo-Lillo, C., Fragkaki, E., & Melicher, W. *A Comparison of Users’ Perceptions of and Willingness to Use Google, Facebook, and Google+ Single-Sign-On Functionality*. Carniage Mellon University, Pittsburgh, PA.

Fleury, T., Basney, J., & Welch, V. *Single Sign-On for Java Web Start Applications Using MyProxy*. National Center for Supercomputing Applications.

Federal Deposit Insurance Corporation. Division of Supervision and Consumer Protection. (2004). *Putting and End to Account-Hijacking Identity Theft*. New York: FDIC.

Ford, M. D. (1998). *Identity Authentication and ‘E-Commerce‘*. The Journal of Information, Law, and Technology.

Ford, W., & Baum, M. S. (1997). Secure Electronic Commerce: Building the infrastructur for digital signature and encryption. New Jersey: Prentice Hall, Inc.

Goth, G. (2008). *Single Sign-on and Social Networks*. IEEE Computer Society, IEEE Distributed System Online, 1541-4922. vol. 8, no. 12.

Halonen, T. (2000). *Authentication and Authorization in Mobile Environment*. Finlandia: Helsinki University of Technology.

Harding, P., Johansson, L., & Klingenstein N. (2007). *Dynamic Security Assertion Markup Language Simplifying Single Sign-On*. IEEE Computer Society, IEEE Security & Privacy, 1540-7993/07.

He, Wenli. (2007). *Single Sign On*. University of Iowa.

Huhnlein, D. et al. *Options for Integrating eID and SAML*. ecsec GmbH, Ruhr-Universitat Bochum. Adolf Wurth GmbH & Co. KG.

Horst, T. W. V. D., & Seamons, K. E. (2007). *Simple Authentication for the Web*. Internet Security Research Lab, Brigham Young University, Provo, UT, USA.

Ishii, T., Inoie, A., & Okamoto, M. *Single Sign-on Using Portable IdP on USB Flash Drive*. Kanagawa Institute of Technology.

Jackson, Nic. (2017). *Building Microservices with Go*. Birmingham: Packt Publishing.

Jain, Anil. K. (2005). *On The Uniqueness of Fingerprints*. Dept. of Computer Science and Engineering, Michigan State University.

Jasig. (2009). *Central Authentication Service*. <https://wiki.jasig.org/display/CAS/Home>. Diakses pada 18 Juli 2018.

Kamil, R. (2016). *Smartcity Bandung 2016*. Presentation of Mayor of Bandung.

Kristanto, R. F. (2011). *Otentikasi dan Otorisasi Terpusat pada Aplikasi Web di Lingkungan Informatika ITB*. Program Studi Teknik Informatika, Institut Teknologi Bandung.

Liu, W., Yue Tan, & Enwei Zhang. (2009). *Service Token for Identity Access Management*. Beijing: Tsinghua University.

Munir M.T., Ir. Rinaldi. (2005). *Diktat Kuliah IF5054 Kriptografi*. Bandung: Institut Teknologi Bandung.

Neuman, B. Clifford, & Theodore Ts’o. (1994). Kerberos: An Authenticaton Service for Computer Networks. IEEE Communications Magazine, Vol 32, No 9, p. 33-38.

Novell. (2011). *Investigating Single Sign-on*. Novell Solution Provider. Waltham, MA.

OASIS. (2008). *Security Assertion Markup Language (SAML) V2.0 Technical Overview*.

Pace, Eugenio. (2015). *About Auth0 | The new way to solve Identity*. <https://auth0.com/about>. Diakses pada 24 Juli 2018.

Pashadilis, Andreas dan Chris J. Mitchell. (2003). *Single Sign-On using Trusted Platforms*. London: University of London.

Park, K., Lim, S. S., & Park, K. H. (2008). *Computationally Efficient PKI-Based Single Sign-On Protocol PKASSO for Mobile Devices*. IEEE Transactions on Computers, vol. 57, no. 6.

Rifai, Wiwit. (2018). *Pengembangan Mesin Pencari Citra dengan Penanganan Semantik Kemiripan yang Berbeda*. Bandung: Institut Teknologi Bandung.

Sun, S.-T., & Beznosov, K. *The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems*. Laboratory for Education and Research in Secure Systems Engineering, Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada.

Sun, S.-T., Boshmaf, Y., Hawkey, K., & Beznosov, K. *A Billion Keys, but Few Locks: The Crisis of Web Single Sign-On*. Laboratory for Education and Research in Secure System Engineering (LERSSE), Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada.

Sun, S.-T. et al. (2010). *What Makes Users Refuse Web Single Sign-On? An Empirical Investigation of OpenID*. University of British Columbia, Vancouver, BC, Canada. Dalhousie University, Halifax, NS, Canada.

Sun, S.-T., Pospisil, E., Muslukhov, I., Dindar, N., Hawkey, K., and Beznosov, K. 2013. *Investigating users’perspectives of Web single sign-on: Conceptual gaps and acceptance model*. ACM Trans. Internet Technol. 13, 1, Article 2 (November 2013), 35 pages.

Tanenbaum, & Steen V., (2007). Distributed System: Principles and Paradigms 2nd Edition. Pearson Education. Vrije Universiteit, Amsterdam, The Netherlands.

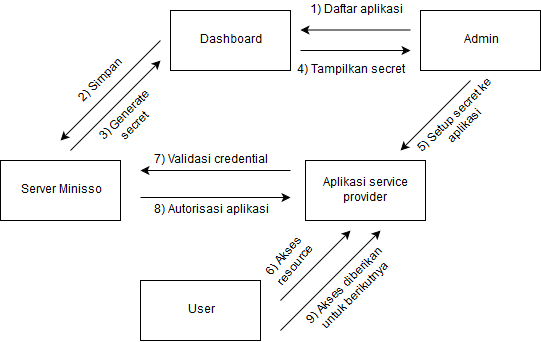
Volchkov, A. (2001). *Revisiting Single Sign-On. A Pragmatic Approach in a New Context*. IEEE Computer Society Magazine IT Pro, 1520-9202/01.

Whitehead, Prof. Jim. (2006). *PHP Session Management*. CMPS.

WSO2 Inc, (2015). *WSO2 Identity Server Documentation | Architecture*. <https://docs.wso2.com/display/IS530/Architecture>. Diakses pada 25 Juli 2018.

Yang, R. Li, G., Lau, W. C., Zhang, K., & Hu, P. *Model-based Security Testing: an Empirical Study on OAuth 2.0 Implementations*. The Chinese University of Hong Kong.

**Lampiran A. Mekanisme Kerja Minisso**

****

Berikut penjelasan mekanisme kerja Minisso sebagai panduan *developer,* termasuk cara mengintegrasikan Minisso dengan aplikasi *service provider* milik *developer*:

1. Admin/*developer* mendaftarkan aplikasi *service provider* pada *dashboard* Minisso dengan mengisi form seperti nama aplikasi, deskripsi, dll.
2. *Dashboard* mengirim informasi ke server.
3. Server mengembalikan *secret* berupa CLIENT\_ID dan CLIENT\_SECRET ke *dashboard*.
4. *Dashboard* menampilkan *secret* berupa data di atas kepada *developer*.
5. *Developer* menerima *secret* kemudian melakukan *setup* CLIENT\_ID dan CLIENT\_SECRET di aplikasi *service provider*. *Secret* ini akan dibutuhkan aplikasi *service provider* setiap kali berkomunikasi dengan server untuk keperluan registrasi, login, dan validasi *user*. Format yang digunakan untuk berkomunikasi antara aplikasi *service provider* dilampikan pada lampiran berikutnya. *Secret* ini sebagai mekanisme keamanan bahwa aplikasi *service provider* yang berhubungan dengan server merupakan aplikasi yang sesungguhnya sethingga mencegah terjadinya serangan *man-in-the-middle*.
6. *User* meminta *resource* kepada aplikasi *service provider*.
7. Aplikasi *service provider* melakukan validasi ke server Minisso.
8. Server Minisso memberikan *response* bahwa aplikasi telah diautorisasi (apabila syarat terpenuhi seperti CLIENT\_ID dan CLIENT\_SECRET benar).
9. Ketika user ingin mengakses *resource* lagi ke aplikasi *service provider*, *resource* dapat diberikan.

**Lampiran B. Kontrak API**

Produk dari tugas akhir ini adalah API (*Application Programming Interface*) yang bisa digunakan untuk integrasi *single sign-on*. Ada tiga *endpoint* yang disediakan Minisso sebagai *single sign-on*. Berikut adalah format kontraknya.

1. /signup

*Endpoint* /signup digunakan untuk mendaftarkan user.

*Endpoint* ini menerima JSON dengan format sebagai berikut.

|  |
| --- |
| {  application: {  client\_id: <CLIENT\_ID>,  client\_secret: <CLIENT\_SECRET>  },  user: {  first\_name: <nama pertama>,  last\_name: <nama belakang>,  picture: <link gambar>,  gender: <gender>,  email: <email>,  password: <password>  }  } |

*Endpoint* ini mengembalikan JSON dengan format *body* sebagai berikut.

|  |
| --- |
| {  data: {  first\_name: <nama pertama>,  last\_name: <nama belakang>,  picture: <link gambar>,  gender: <gender>,  email: <email>,  errors: [<array berisi string error>]  },  meta: {  http\_status: 200  }  } |

Apabila pendaftaran berhasil, endpoint tersebut juga mengembalikan *cookies* pada HTTP *response header* dengan format sebagai berikut.

|  |
| --- |
| minisso=<token> |

1. /login

*Endpoint* /login digunakan untuk *login* *user*.

*Endpoint* ini menerima JSON dengan format sebagai berikut.

|  |
| --- |
| {  application: {  client\_id: <CLIENT\_ID>,  client\_secret: <CLIENT\_SECRET>  },  user: {  email: <email>,  password: <password>  }  } |

*Endpoint* ini mengembalikan JSON dengan format *body* sebagai berikut.

|  |
| --- |
| {  data: {  first\_name: <nama pertama>,  last\_name: <nama belakang>,  picture: <link gambar>,  gender: <gender>,  email: <email>,  errors: [<array berisi string error>]  },  meta: {  http\_status: 200  }  } |

Apabila *login* berhasil, *endpoint* tersebut juga mengembalikan *cookies* pada HTTP *response header* dengan format sebagai berikut.

|  |
| --- |
| minisso=<token> |

1. /validate

*Endpoint* /validate digunakan untuk memvalidasi token *user*.

*Endpoint* ini menerima *request header* dengan format sebagai berikut.

|  |
| --- |
| minisso=<token> |

*Endpoint* ini mengembalikan JSON dengan format *body* sebagai berikut.

|  |
| --- |
| {  data: {  first\_name: <nama pertama>,  last\_name: <nama belakang>,  picture: <link gambar>,  gender: <gender>,  email: <email>,  errors: [<array berisi string error>]  },  meta: {  http\_status: 200  }  } |