Arsitektur Infrastruktur Single Sign-On dengan Akses Pengguna yang Masif

Koko Bachrudin[1], I Made Wiryana[2] dan Elyna Fazriyati[1]

[1]Sistem Informasi, Universitas Gunadarma

[2]Manajemen Sistem Informasi, Universitas Gunadarma

E-mail : koko@staff.gunadarma.ac.id, mwiryana@staff.gunadarma.ac.id, efazriyati@staff.gunadarma.ac.id

Abstrak

Terdapat suatu instansi pemerintahan yang memiliki pengguna lebih dari 4 juta ingin mengembangkan sebuah sistem autentikasi yang terpadu dan saling terintegrasi. Instansi ini telah mengembangkan lebih dari sepuluh sistem elektronik untuk menunjang pelayanan. Sepuluh sistem elektronik ini dikembangkan dengan menggunakan metode semi *microservice* dan dibagi menjadi dua bagian yaitu *Frontend* dengan *Client base programing* dan *backend*, komunikasi antar FE dan BE ini juga diperlukan sebuah mekanisme untuk mengetahui wewenang dari pengguna yang telah melakukan autentikasi di sisi *Frontend*. Untuk mengembangkan sistem yang dapat memenuhi kebutuhan ini perlu dibangun sebuah arsitektur *Single Sign-On* (SSO) yang tepat. Penelitian ini berfokus pada pengembangan arsitektur SSO dengan berbasis pada analisis kebutuhan yang telah dilakukan sebelumnya. Penggunaan teori graf menjadikan desain arsitektur menjadi jelas dan tidak terjadi ambiguitas. Penggunaan teori graf juga dapat mengidentifikasi komponen mana yang sifatnya kritis dan menentukan solusi untuk menangani masalah yang ditemukan. Dari hasil pengujian yang dilakukan penggunaan teori graf berhasil dalam mengidentifikasi komponen kritis dari arsitektur yang dikembangkan dan dapat memberikan solusi. Dengan pengembangan arsitektur menggunakan teori graf arsitektur ini telah berhasil diterapkan pada implementasi produksi dengan melayani 4 juta pengguna dengan pengguna konkuren lebih dari 200.000 dan melakukan validasi token lebih dari 28.000 setiap detik.

Kata kunci : *Single Sign-On*, graf , akses pengguna, masif

Pendahuluan

Sistem elektronik menjadi tulang punggung hampir disegala proses bisnis disegala instansi, baik instansi swasta maupun instansi pemerintah. Hampir semua kegiatan instansi akan diarahkan untuk menggunakan sebuah sistem elektronik. Instansi yang besar memiliki sistem elektronik yang tidak hanya satu, namun terdapat banyak sistem elektronik yang saling terhubung maupun tidak saling terhubung. Pendekatan pengembangan berbasiskan *microservice* atau SOA (*Service Oriented Arcitecture*) menyebabkan sistem menjadi terbagi dalam lingkungan yang spesifik [5].

Salah satu fitur yang selalu ada dalam sebuah sistem elektronik itu adalah fitur autentikasi, fitur autentikasi ini digunakan untuk membatasi kewenangan pengguna untuk setiap fitur/data yang bisa di akses. Misalnya pengguna A hanya memiliki kewenangan dalam proses X dan proses Y sedangkan pengguna B hanya dapat mengakses proses Y. Fitur autentikasi ini biasanya tersemat pada setiap sistem elektronik dengan data yang tersimpan di sistem elektronik tersebut, ini menjadikan masalah ketika sistem yang dikembangkan menggunakan data pengguna yang sama namun dengan tujuan yang berbeda.

Praktiknya ada beberapa pendekatan yang dilakukan untuk menangani hal seperti ini. Pendekatan yang pertama adalah melakukan duplikasi data, ini adalah cara yang paling umum digunakan karena tidak terlalu kompleks, namun cara ini memiliki kelemahan yaitu redudansi data yang harus dijaga. Bisa dibayangkan jika terdapat sepuluh sistem yang menggunakan fasilitas autentikasi maka jika terjadi perubahan, maka 10 basis data harus dilakukan perubahan juga.

Pendekatan kedua dengan menggunakan basis data yang sama, cara ini juga bisa digunakan dengan memanfaatkan satu basis data autentikasi yang sama untuk seluruh aplikasi, ini menjadi hal yang sangat mudah. Kelemahan dari sistem ini adalah tidak bisa terjadi integrasi antar sistem elektronik ketika autentikasi, misal pengguna A telah auten-

DOI : http://dx.doi.org/10.32409/jikstik.21.2.3059

tikasi di sistem S_1, maka jika pengguna A tersebut ingin autentikasi di sistem S_2 maka harus melakukan autentikasi lagi.

Pedekatan yang ketiga dengan menggunakan *Single Sign-On* (SSO) adalah metode kontrol akses yang meminta pengguna untuk *login* sekali dan memungkinkan mereka untuk mengakses beberapa sumber daya dan layanan setelah autentikasi berhasil tanpa diminta untuk autentikasi lagi [1]. Cara ini jarang digunakan karena membutuhkan pemahaman teknis yang jelas tentang autentikasi sistem dan pemahaman tentang *acccess control* pada sistem. Pendekatan ini memiliki kelebihan, sistem dapat melakukan *session sharing* pada seluruh sistem yang menggunakan layanan SSO yang sama. Contohnya pengguna A telah autentikasi di sistem S_1, maka jika pengguna A mengakses Sistem S_2 maka sistem akan langsung melakukan autentikasi otomatis tanpa harus melakukan autentikasi ulang. Sistem ini biasanya digunakan di Instansi yang besar dan memiliki sistem yang kompleks yang saling terintegrasi.

Terdapat suatu instansi pemerintahan yang memiliki pengguna lebih dari 4 juta ingin mengembangkan sebuah sistem autentikasi yang terpadu dan saling terintegrasi. Instansi ini telah mengembangkan lebih dari sepuluh sistem elektronik untuk menunjang pelayanan. Sepuluh sistem elektronik ini dikembangkan dengan menggunakan metode *semi microservice* dan dibagi menjadi dua bagian yaitu *Frontend* (FE) dengan *Client base programing* dan *backend* (BE), komunikasi antar FE dan BE ini juga diperlukan sebuah mekanisme untuk mengetahui wewenang dari pengguna yang telah melakukan autentikasi di sisi *Frontend*.

Untuk mengembangkan sistem yang dapat memenuhi kebutuhan ini perlu dibangun sebuah arsitektur SSO yang tepat. Penelitian ini berfokus pada pengembangan arsitektur SSO dengan berbasis pada analisis kebutuhan yang telah dilakukan sebelumnya.

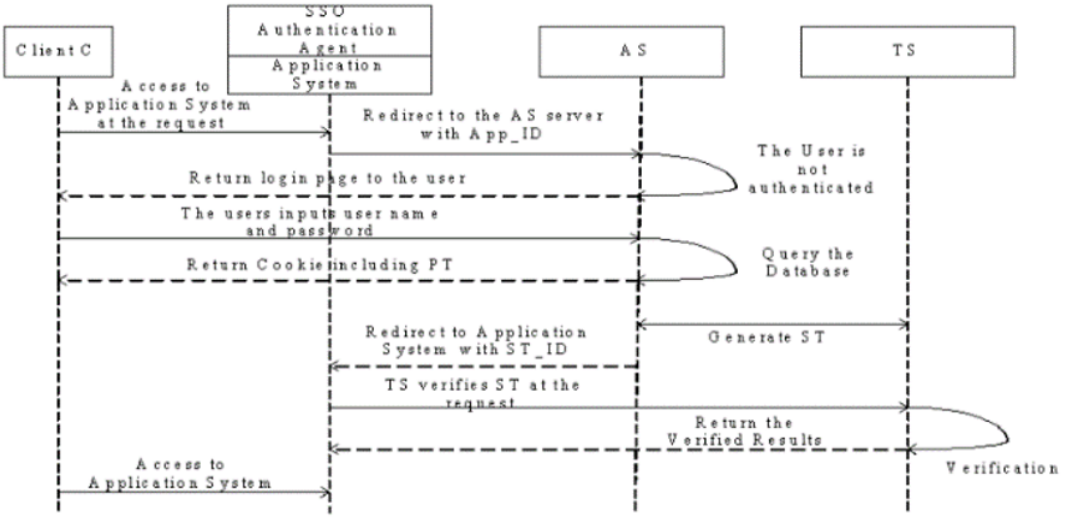
Metode Penelitian

Metode penelitian ini menggunakan empat tahapan yaitu:

1. **Studi Literatur**. Melakukan pengumpulan literatur tentang pengembangan *Single Sign-on*
2. **Pengumpulan data**. Mengumpulkan data berupa hasil analisis kebutuhan dari sistem analis
3. **Desain arsitektur SSO**. Mengembangkan arsitektur berdasarkan pada hasil analis kebutuhan
4. **Pengujian arsitektur SSO**. Pengujian arsitektur SSO dengan menggunakan perangkat lunak SSO.

Model Umum dari Single Sign-on

Secara umum SSO dibagi menjadi tiga jenis model [9] yaitu: *Broker-Based SSO*, *Agent-Based SSO* dan *Gateway-Based SSO*. *Broker-Based SSO* mirip dengan pendekatan menggunakan satu basis data yang diakses oleh banyak sistem elektronik. Sangat cocok untuk pengembangan sistem elektronik dengan skala kecil, yang tidak terdapat kompleksitas tinggi.



Gambar 1: Proses Autentikasi SSO [4]

Agent-Based SSO menggunakan *agent* yang di pasang di setiap sisi baik di sisi *client* maupun di sisi *server*. Model memastikan keamanan koneksi dan SSO, dengan fleksibilitas dan implementasi yang baik. Namun, model memiliki kelemahan besar yaitu informasi autentikasi pengguna disimpan secara lokal, yang meningkatkan risiko terungkapnya. Terlebih lagi jika terdapat aplikasi baru, admin harus mengembangkan program agen yang tepat, yang lebih kompleks dan memiliki beban kerja yang besar untuk pemeliharaan.

Model *Gateway-Based SSO* hampir tidak mengubah sistem aplikasi, selama konfigurasi dan gateway memiliki autentikasi bersama. Pengguna dapat dengan mudah mengenkripsi dan mengirimkan data sistem aplikasi, dan implementasi serta pemeliharaannya relatif sederhana [4]. Gambar 1 menjelaskan tentang proses autentikasi internal si sistem SSO.

Pengumpulan Data

Pada tahap ini didapatkan kebutuhan dari arsitektur sistem SSO sebagai berikut:

R1 Arsitektur Sistem harus mampu melayani pengguna lebih dari empat juta pengguna, dengan asumsi pengguna konkuren 10.000 setiap detik.

R2 Arsitektur Sistem harus mampu melayani komunikasi antara *Frontend* dan *Backend* sistem elektronik.

R3 Arsitektur Sistem harus dapat pulih ketika terjadi masalah.

Data yang digunakan pada penelitian ini adalah data pengguna yang berisi *username* dan *password*. Pengumpulan data ini dapat dirumuskan sebagai berikut:

$$Req = \{R_1, R_2, R_3\}, D = \{u, p\} \quad (1)$$

Di mana *Req* merupakan kebutuhan dari arsitektur sistem SSO yang terdiri dari himpunan R_1, R_2, R_3 dan D merupakan data yang digunakan yang berisi *username* (u) dan *password* (p).

Desain Arsitektur SSO

Berdasarkan analisis kebutuhan untuk model umum SSO yang digunakan adalah *Gateway-Based SSO*. Pada tahapan ini dikembangkan arsitektur berdasarkan kebutuhan yang telah didefinisikan, spesifikasi sistem harus di definisikan. Berikut spesifikasi sistem untuk setiap kebutuhan:

S1 Arsitektur SSO (SSO) harus mendukung *load* konsep *load balancing* (Lb). Di mana *load* akan dibagi secara proporsional kepada beberapa layanan atau mesin.

S2 Arsitektur SSO (SSO) harus mendukung validasi antara *backend* (BE) dan *frontend* (FE).

S3 Arsitektur SSO (SSO) harus mampu melakukan *fail over* (FIP) jika terjadi kegagalan

Solusi dari arsitektur SSO ini dapat dirumuskan sebagai berikut:

$$Sol = \{S_1, S_2, S_3\}$$

Dari solusi yang telah didefinisikan, setiap solusi akan digambarkan ke dalam sebuah graf [8]. Graf ini menjadi panduan dalam mengembangkan arsitektur SSO. Graf memiliki *vertex*, di mana *vertex*-nya adalah komponen yang ada pada solusi tersebut dan *edge*-nya merupakan interkoneksi antar komponen dari setiap solusi. Graf tersebut nantinya dapat mengabarkan *vertex* kritis dari setiap solusi, *vertex* kritis ini dilihat dari nilai *indegree* dan *outdegree* dari *vertex* tersebut.

$$G_{S_x} = (V, E)$$
$$V = \{Lb, BE, FE, SSO\}$$
$$E \subseteq \{\{x, y\} \mid x, y \in V \, dan \, x \neq y\}$$

Pengujian Arsitektur SSO

Metode pengujian menggunakan metode eksperimental, pengujian menggunakan perangkat keras berupa server dengan virtual mesin. Setiap *vertex* yang telah didefinisikan akan menjadi komponen perangkat lunak yang akan dipasang di server, setiap *edge* dari graf yang telah didefinisikan akan menjadi koneksi antar perangkat lunak yang di Instalasi. Pada tahap uji ini, arsitektur diuji apakah akan menjawab kebutuhan yang telah ditentukan.

Analisis dan Diskusi

Graf S1, S2 dan S3

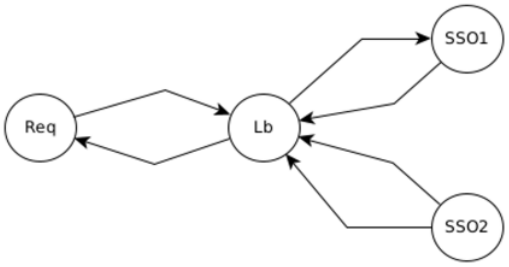
Karena kebutuhan *load balancing*/pembagian beban maka komponen SSO di bagi menjadi dua yaitu SSO_1 dan SSO_2.Gambar 2 merupakan graf yang memvisualisasikan S_1, di mana:

$$V_{S_1} = \{Req, Lb, SSO_1, SSO_2\}$$

dan matrix *Adjacency*-nya sebagai berikut:

$$S_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Graf S_1 dapat dianalisis bahwa derajat dari Lb memiliki nilai *indegree* 3 dan *outdegree* 3, ini menujukan bahwa Lb menjadi titik paling kritis dari arsitektur untuk Solusi S_1. Kegagalan pada Lb dapat menyebabkan sistem tidak bisa berjalan secara semestinya.



Gambar 2: graf S_1

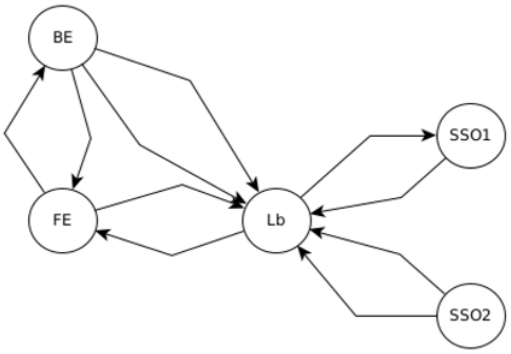
Gambar 3 merupakan graf yang memvisualisasikan S_2 , di mana:

$$V_{S_2} = \{BE, FE, Lb, SSO_1, SSO_2\}$$

dan matrix *Adjacency*-nya sebagai berikut:

$$S_2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Dari graf S_2 dapat dianalisis bahwa dengan pemecahan solusi untuk komunikasi antara BE dan FE menambah beban dari Lb, di mana nilai *indegree* dan *outdegree* dari Lb menjadi masing-masing 4. Tingkat ke kekritisan dari Lb menjadi lebih meningkat dibanding pada graf S_1.



Gambar 3: Arsitektur S_2

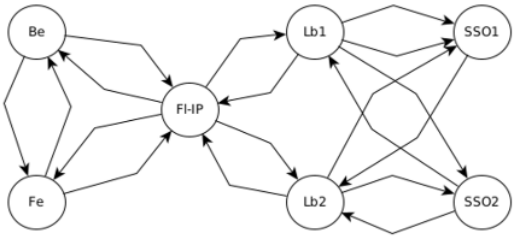
Gambar 3 merupakan graf yang memvisualisasikan S_3 , di mana:

$$V_{S_3} = \{BE, FE, FIP, Lb_1, Lb_2, SSO_1, SSO_2\}$$

dan matrix *Adjacency*-nya sebagai berikut:

$$S_3 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Graf S_3 merupakan bentuk optimal dari permasalahan ke kekritisan dari Lb. Pada graf ini derajat dari Lb *indegree* dan *outdegree* memiliki nilai 3, namun terdapat dua Lb yaitu Lb_1 dan Lb_2, di mana jika gagal maka salah satu Lb maka Lb yang lain dapat digunakan. Kekritisan dari graf ini terletak pada *FIP* , di mana nilai *indegree* dan *outdegree*-nya masing-masing 4. Walaupun bernilai 4 namun *FIP* ini bersifat logis dan secara teknis kemungkinan kegagalannya sangat kecil, kecuali dua node Lb benar-benar gagal., lihat Gambar 4.



Gambar 4: Arsitektur S_3

Implementasi dan Pengujian Arsitektur

Pada pengujian ini digunakan perangkat komputer dengan spesifikasi seperti diperlihatkan Tabel 1:

Tabel 1: Spesifikasi Sistem

Fungsi	CPU (core)	RAM (GB)	*Storage* (GB)
SSO1 dan SSO2	20	32	2456
Lb1 dan Lb2	4	8	125
BE dan FE	4	8	125

Selain penggunaan perangkat keras juga digunakan perangkat lunak untuk menyimulasikan Vertex dari graf yang telah di definisikan seperti diperlihatkan Tabel 2.

Tabel 2: Spesifikasi perangkat lunak

Fungsi	Perangkat Lunak
SSO_1 dan SSO_2	keycloak
Lb_1dan Lb_2	nginx
BE dan *FE*	postman
Floating IP (*FIP*)	keepalived dan gnu/linux

Uji R1

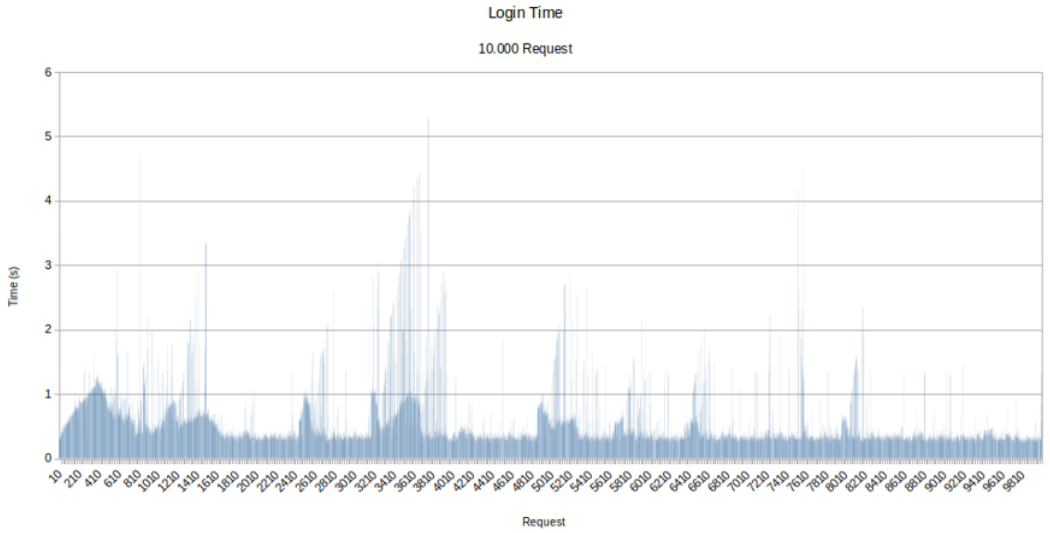
Untuk menguji R1 dari sistem SSO ini disusun sebuah program untuk menyimulasikan *login* pengguna secara konkuren. Algoritma 1 menjelaskan bagaimana program testing berjalan. Algoritma ini ditranslasikan menjadi program dengan menggunakan Bahasa python.

Algoritma 1: Testing Konkuren Pengguna

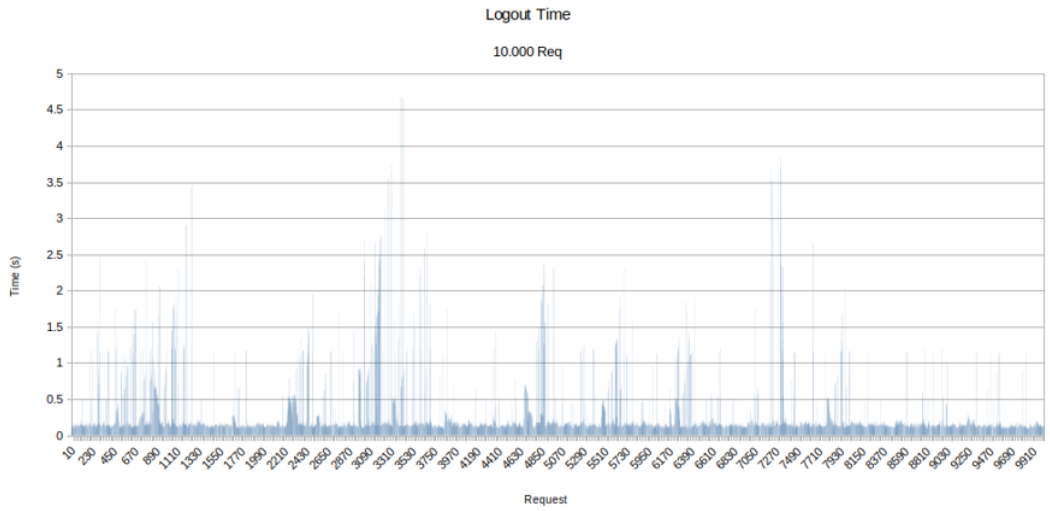
```
Data: Usr≠None, Pas≠None, Url≠None
do in parallel
    T_startLog ← T_sys;
    acceesToken ←
      httpRequest(Url, Usr, Pas);
    if acceesToken ≠ None then
        T_finLog ← T_sys;
        T_processLog ← T_finLog − T_startLog;
        T_startOut ← T_sys;
        httpRequest(Url, accessToken);
        T_finOut ← T_sys;
        T_processLog ← T_finOut − T_startOut;
        write(T_processLog, T_processOut);
    else
        return 0;
return 0;
```

Gambar 5 menggambarkan waktu *login* dalam detik pada saat jumlah *request* tertentu. Pada uji ini digunakan konkurensi sebesar 10.000 *request* setiap detik, pemilihan 10.000 ini mempertimbangkan kemungkinan paling maksimal dari sebuah sistem operasi membuka *socket* untuk melayani *request* yang datang. Dari percobaan ini didapat maksimal waktu yang diperlukan untuk autentikasi adalah 5,28739285469055 detik.

Gambar 6 menggambarkan waktu *logout* setelah autentikasi berhasil dan memberikan *access token*, nilai konkurensinya akan sama dengan proses autentikasi, karena proses ini dijalankan setelah autentikasi berhasil. Dari percobaan ini didapat maksimal waktu yang diperlukan untuk *logout* adalah 4,68681740760803 detik.



Gambar 5: Grafik Waktu Login



Gambar 6: Grafik Waktu Logout

Dari percobaan ini didapatkan juga keseluruhan dari *request* terpenuhi dan tidak ada yang gagal, sehingga dapat disimpulkan arsitektur yang dirancang telah berhasil menjawab kebutuhan R1.

Uji R2

Untuk menguji R2 dari sistem SSO ini disusun sebuah program untuk menyimulasikan koneksi antara *frontend, backend* dan *SSO*. Algoritma 2 menjelaskan proses dari program *frontend*, sedangkan Algoritma 3 menjelaskan proses yang dilakukan oleh sisi *backend*. Untuk kemudahan simulasi di sisi *frontend* digunakan perangkat lunak POSTMAN sedangkan untuk *backend*-nya diprogram menggunakan bahasa Pemrograman GoLang.

Algoritma 2: Frontend

Data: $Usr \neq None, Pas \neq None, Url \neq None,\ Url_{server} \neq None$
$acceesToken \leftarrow GetToken(Url, Usr, Pas)$;
if $acceesT\ oken \neq None$ **then**
    $httpStatus \leftarrow GetAP\ I(Url_{server}, accessToken)$;
    if $httpStatus\ = 200\ then$
        return $Sucsess$;
    else
        return $Failed$;
else
    return 0;

Algoritma 3: Backend

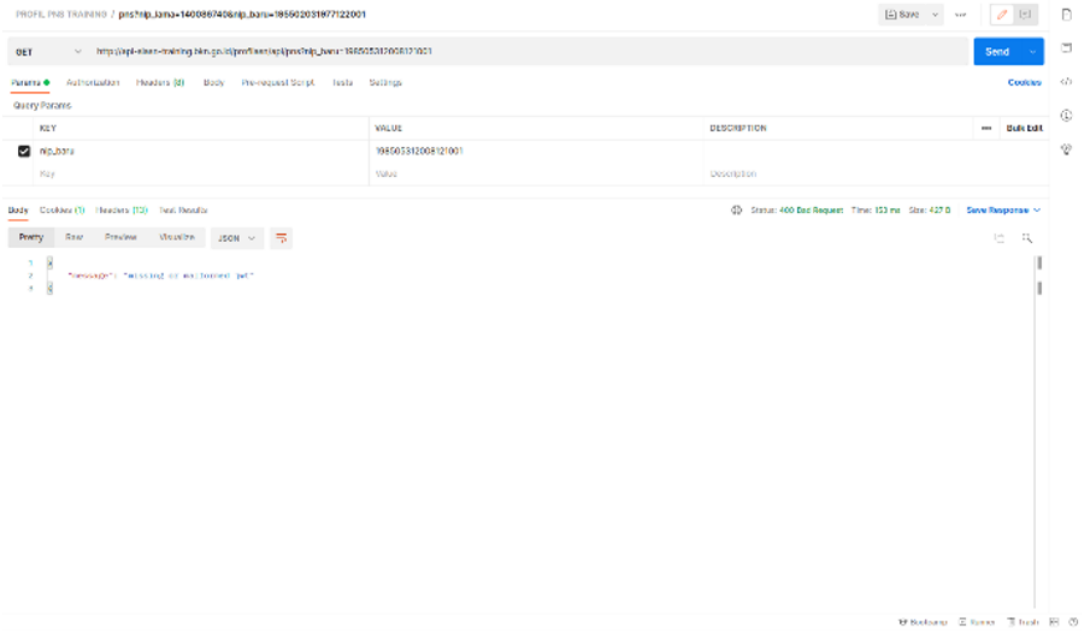
Data: $Usr \neq None, accessToken \neq None, PubKey \neq None$
$acceesToken \leftarrow HttpHeaders(Authentication)$;
if $acceesToken \neq None$ **then**
    $tokenValid \leftarrow ValidToken(Url, accessToken)$;
    if $tokenValid\ is\ True$ **then**
        return 200;
    else
        return 400;
else
    return 400;

Gambar 7 merupakan hasil uji menggunakan perangkat lunak postman ketika sisi *frontend* tidak mengirimkan token dari atau mengirimkan token namun token tidak valid. Ketika mendapati *state* seperti ini akan ditampilkan status 400, di mana pesan ini merupakan pesan balikan dari backend.

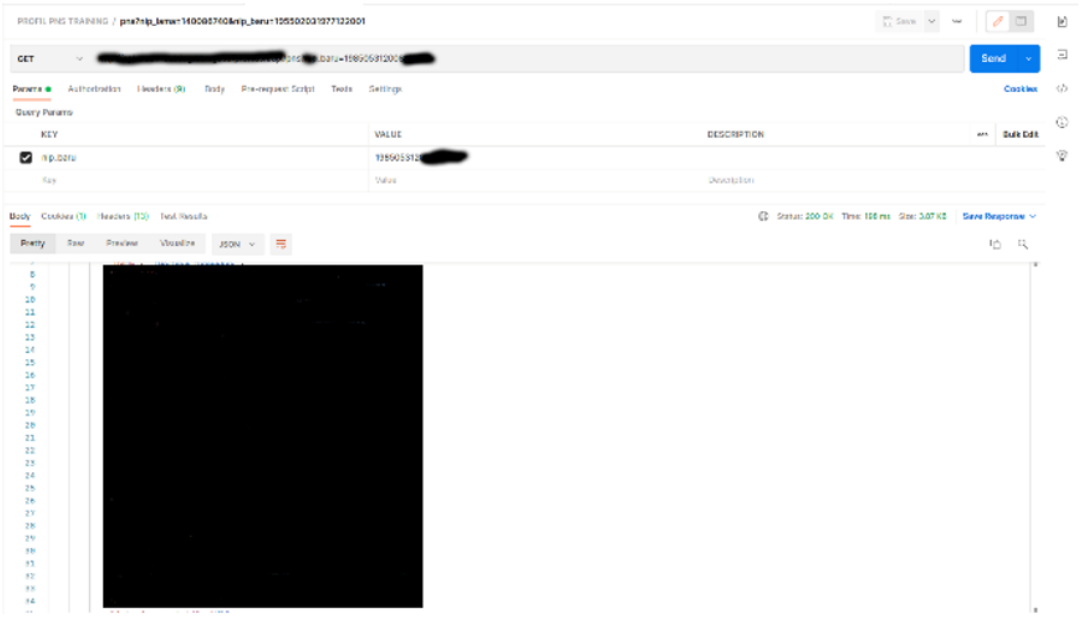
Gambar 8 merupakan hasil uji menggunakan perangkat lunak postman ketika sisi *forntend* mengirimkan token yang didapat dari SSO ke *backend*. Ketika mendapatkan *state* yang sesuai ini maka akan ditampilkan status 200, di mana pesan ini merupakan pesan balikan dari *backend*. Dari uji yang dilakukan ini dapat disimpulkan arsitektur yang dirancang telah berhasil menjawab kebutuhan R2.

Uji R3

Untuk menguji R3 dari sistem SSO ini dilakukan dengan langsung mematikan beberapa komponen sistem, yaitu: Lb_1, Lb_2, SSO_1, SSO_2secara bergantian kemudian mengakses alamat url dari sistem SSO. Gambar 9 menggambarkan pengujian Lb_1 dan Lb_2 dimatikan.



Gambar 7: Client Tidak Mengirim Token

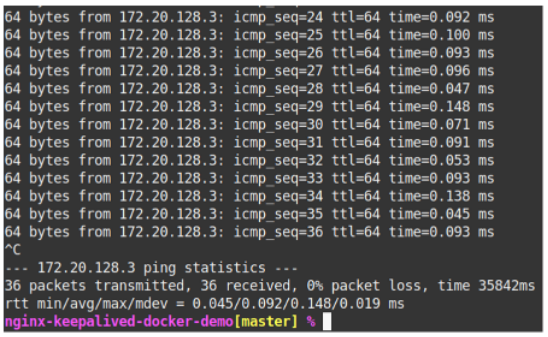


Gambar 8: Client Mengirim Token

Saat salah satu dari *Lb* ini dimatikan kemudian dicoba akses ke *FIP* seperti pada Gambar 11, dapat dilihat sistem tetap berjalan normal walaupun satu dari *Lb* terjadi kegagalan. Gambar 10 menggambarkan pengujian SSO_1 dan SSO_2 dimatikan. Saat salah satu dari *Lb* ini dimatikan kemudian dicoba akses ke *FIP* seperti pada Gambar 11, dapat dilihat sistem tetap berjalan normal walaupun satu dari SSO terjadi kegagalan. Hasil uji ini menggambarkan bahwa arsitektur ini juga menyelesaikan kebutuhan R3 di mana sistem tetap bisa berjalan jika salah satu komponen terjadi kegagalan, walaupun pada pengujian ini tidak dilakukan pengujian *cascading failure*.

Gambar 9: Mematikan Lb_1 atau Lb_2

Gambar 10: Mematikan SSO_1 atau SSO_2



Gambar 11: Uji ping ke *FIP*

Penutup

Penelitian ini telah diimplementasi pada tahap produksi dengan melayani 4 juta pengguna dengan pengguna konkuren lebih dari 200.000 dan melakukan validasi token lebih dari 28.000 setiap detik. Hasil dari implementasi pada tahap produksi ini memperlihatkan seluruh kebutuhan memang terjawab. Dengan implementasi penelitian ini sampai tahap produksi juga memperlihatkan bawa metode graf dapat digunakan dalam menentukan titik kritis dari suatu sistem.

Dari hasil analisis, perancangan, implementasi dan pengujian yang dilakukan pada penelitian ini disimpulkan bahwa arsitektur yang berbasiskan solusi yang ditawarkan dapat memenuhi kebutuhan Instansi. Penggunaan teori graf menjadikan desain arsitektur menjadi jelas dan tidak terjadi ambiguitas. Penggunaan teori graf juga dapat mengidentifikasi komponen mana yang sifatnya kritis dan menentukan solusi untuk menangani masalah yang ditemukan.

Penelitian ini masih jauh dari sempurna, ada be-

berapa aspek yang bisa dikembangkan lagi, seperti pengujian simulasi pada desain graf yang telah jadi menggunakan algoritma tertentu. Selain simulasi pada graf tersebut dapat juga dilakukan verifikasi maupun validasi dari graf solusi. Hal ini akan meningkatkan akurasi dari solusi yang ditawarkan.

Daftar Pustaka

[1] T. Bazaz and Aqeel Khalique, "A review on single sign on enabling technologies and", International Journal of Computer Application, vol. 151(11), pp. 18-25, 10 2016.

[2] Y. Chen, B. Xia, B. Wu and L. Shi, "Design of web service single sign-on based on ticket and assertion", In 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), pp. 297-300, 2011.

[3] A. Kukic, "The Definitive Guide to Single Sign On (SSO)", Auth0, 2017.

[4] Z. Liang, "Design of single sign-on for hybrid architecture based on web service", Journal of Computer Application,vol. 30, p. 3363-3365, 2010.

[5] I. Nadareishvili, R. Mitra, M. McLarty and M. Amundsen, "Microservice Architecture", O'Reilly Media, Inc., 2016.

[6] E. Osmanoglu, "Identity and Access Management", Elsevier Science, Syngress, 2013.

[7] A. G. Revar and M. D. Bhavsar, "Securing user authentication using single sign-on in cloud computing", 2011 Nirma University International Conference on Engineering, pp. 1-4, 2011.

[8] R. J. Wilson, "Introduction to Graph Theory", Fourth edition, Longman, 1996.

[9] X.-e. You and Y. Zhu, "Research and design of web single sign-on scheme", Optimized Design of Web Single Sign-On Based on Authentication Protocol, 06 2012.

[10] X.-e. You and Y. Zhu, "Research and design of web single sign-on scheme", 2012 IEEE Symposium on Robotics and Applications (ISRA), pp. 383-386, 2012.