Implementasi *Single Sign-On* Menggunakan *Google Identity*, REST dan OAuth 2.0 Berbasis Scrum

http://dx.doi.org/10.28932/jutisi.v7i2.3437

Riwayat Artikel
Received: 22 Februari 2021 | *Final Revision*: 28 Juni 2021 | *Accepted*: 6 Juli 2021

I Kadek Dendy Senapartha[#1]

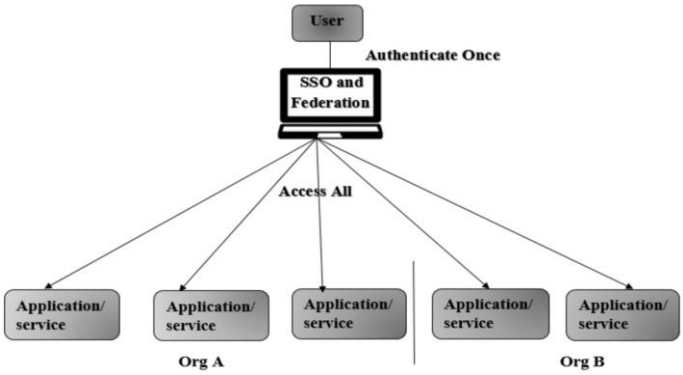
[#] *Program Studi Informatika, Universitas Kristen Duta Wacana*
Jl. Dr. Wahidin Sudirohusodo No.5-25, Kotabaru, Kec. Gondokusuman, Daerah Istimewa Yogyakarta 55224
[1]dendy.prtha@staff.ukdw.ac.id

Abstract —**Single Sign-On (SSO) is a technology that can support user convenience in accessing a system. By using SSO, a user only needs to authenticate once to get access to a system. OAuth 2.0 is one of the protocols that can be implemented on the SSO system. Currently, many Application Service Providers (ASP) support the OAuth 2.0 protocol thus providing convenience in the development of a more standard SSO system. Google Identity is one of the services provided by Google that can be used to build SSO systems using the OAuth 2.0 protocol. Application of the request and response methods provided by the protocol specification OAuth 2.0 and Representational State Transfer (REST) architecture of the system implementation can also make SSO systems more secure. In its implementation, the use of an agile system development methodology with the Scrum framework is used to increase speed and flexibility. The results of this research show that the use of Google Identity, REST, and OAuth 2.0 can provide easy user access, guarantee access validity, accelerate client-server data exchange and simplify the SSO implementation process.**

Keywords— **Google Identity; OAuth 2.0; REST; Scrum; Single Sign-On (SSO).**

I. PENDAHULUAN

Student Relationship Management (SRM) adalah sebuah konsep sistem yang memodifikasi sistem yang ada pada *Customer Relationship Management* (CRM) [1]. Sistem ini menangani dan mengatur hubungan instansi pendidikan dengan antara pelajar/mahasiswa dalam upaya untuk membangun sinergi yang bersifat personal sehingga meningkatkan performa akademiknya [2]. Untuk dapat meningkatkan kenyamanan dan keamanan akses pengguna pada sistem, maka salah satu hal yang dapat dilakukan adalah mempermudah akses pengguna ke dalam sistem sehingga sistem harus mengimplementasi *Single Sign-On* (SSO) sebagai mekanisme otorisasi pada sistem [3].



Gambar 1. Gambaran Umum SSO

SSO merupakan sebuah metode akses kontrol sehingga pengguna hanya perlu melakukan sekali proses *login* untuk dapat mengakses berbagai *resource* dan *services* tanpa perlu meminta pengguna melakukan *login* kembali. Pada Gambar 1 memperlihatkan bagaimana SSO mempermudah akses ke aplikasi atau layanan yang berbeda dengan hanya melakukan satu kali proses autentikasi. SSO hanya akan melakukan permintaan kredensial yang dibutuhkan dan menerima data tersebut untuk melakukan proses autentikasi. Oleh karena itu SSO dapat mempermudah akses pengguna ke aplikasi lain yang ada pada sistem.

Terdapat beberapa tipe arsitektur SSO, dengan properti dan infrastruktur yang berbeda. Tipe-tipe arsitektur SSO tersebut yaitu *Secure Client-Side Credential Caching*, *Secure Server-Side Credential Caching*, SSO dengan *Single Set Credentials*, SSO berbasis *Public Key Infrastructure*, dan SSO berbasis *Token* [4]. *Secure Client-Side Credential Caching* dan *Secure Server-Side Credential Caching* menggunakan beberapa set kredensial, sedangkan SSO berbasis *Public Key Infrastructure* dan SSO berbasis *token* hanya menggunakan satu kredensial. Tipe-tipe arsitektur ini dapat digunakan sesuai dengan situasi dan kondisi yang beragam tergantung kebutuhannya.

Untuk mendukung interoperabilitas aplikasi-aplikasi yang ada pada sistem maka dibutuhkan protokol dan standar yang digunakan untuk komunikasi antara *client* dan *server*. Terdapat dua protokol yang populer digunakan yaitu *Representational State Transfer* (REST) dan *Simple Object Access* (SOAP). REST merupakan sebuah arsitektur untuk membangun *web services* sedangkan SOAP merupakan protokol yang digunakan untuk bertukar data pada jaringan Komputer. Hasil eksperimen yang pernah dilakukan menunjukkan bahwa SOAP memiliki waktu respons yang lebih lambat serta mengonsumsi memori lebih besar dibandingkan REST [5]. Selain itu mekanisme REST lebih sederhana karena interaksi *client* dan *server* yang terjadi lebih sedikit yang berdampak pada kecepatan proses data *client*.

Untuk mempermudah implementasi mekanisme otorisasi dapat menggunakan *services* yang sudah disediakan oleh *Application Service Provider* (ASP). Google merupakan salah satu ASP yang mendominasi pengguna terbanyak untuk layanan mesin pencari [6] dan mail [7]. Salah layanan yang disediakan oleh Google adalah Google Identity [8], yang menyediakan fitur untuk melakukan SSO dengan menggunakan protokol OAuth 2.0. Dengan memanfaatkan layanan ini, pengguna dapat mengakses layanan-layanan yang disediakan oleh Google API seperti Google Calendar, Google Mail, dan lain sebagainya. Namun selain dapat memanfaatkan layanan yang telah disediakan Google, pengguna dapat memanfaatkan layanan Google Identity sebagai server autentikasi dan otorisasi, sehingga integrasi SSO dengan menggunakan Google dimungkinkan.

Protokol OAuth 2.0 merupakan protokol yang banyak digunakan pada SSO dikarenakan kemudahan implementasi dan banyaknya dukungan dari ASP terhadap protokol ini [9]. OAuth merupakan standar terbuka untuk otorisasi yang mengizinkan aplikasi pihak ketiga untuk mengakses *resource* pada *resource* server tanpa perlu membagikan kredensial pengguna. Protokol lain yang dapat digunakan adalah *Security Assertion Markup Language* (SAML) dan OpenID. SAML merupakan protokol berbasis XML sedangkan Open ID merupakan protokol terbaru yang berjalan diatas protokol OAuth 2.0. Hasil analisis penelitian yang pernah dilakukan menunjukan bahwa SAML memiliki batasan interoperabilitas dengan perangkat bergerak seperti *smartphone* atau *tablet pc*, sedangkan OpenID merupakan standar baru yang memiliki tingkat adopsi yang masih rendah di kalangan Enterprise karena dirasa belum matang untuk dijadikan protokol otorisasi yang standar [10]. Dengan demikian, penerapan SSO berbasis protokol OAuth 2.0 dapat digunakan sebagai protokol otorisasi SSO pada sistem.

Beberapa penerapan dan penelitian tentang SSO sudah pernah dilakukan sebelumnya. Penelitian tentang SSO dengan menggunakan protokol OAuth 2.0 untuk mengamankan aplikasi *e-commerce* pernah dilakukan [11]. Penelitian ini melakukan implementasi SSO dengan protokol OAuth 2.0 dengan hanya memanfaatkan layanan LoginRadius sebagai *identity platform* pengguna aplikasi *e-commerce.*

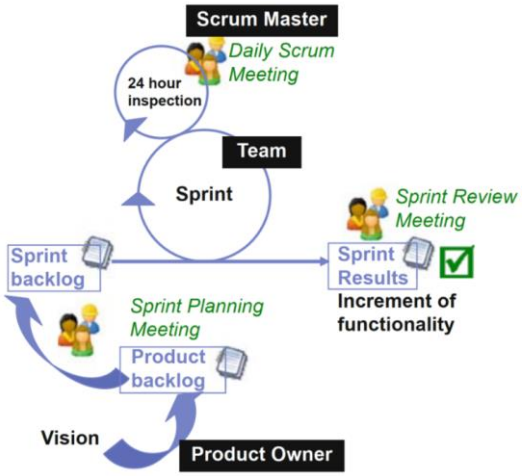
Penerapan SSO dengan Google pada *website* menggunakan kerangka kerja YII pada PHP telah dilakukan [12]. Penelitian ini hanya menerapkan mekanisme Google Sign In untuk melakukan *sign in* ke website dan tidak mengimplementasi protokol OAuth 2.0 secara mandiri untuk mengamankan akses pengguna ke server.

Penelitian dan penerapan SSO dan OAuth 2.0 juga pernah dilakukan namun tanpa menggunakan layanan-layanan ASP [11]. Pada penelitian ini pengguna melakukan *sign in* melalui sistem SSO, dengan memasukan *username* dan *password* untuk melakukan autentikasi dan otorisasi akses ke aplikasi web. Aplikasi ini menggunakan model *client-server* dibangun dengan arsitektur *representational state transfer* (REST) yang terdiri dari dua jenis aplikasi yaitu *backend* berteknologi Java dengan *framework* Spring Boot dan *frontend* dengan teknologi JavaScript dengan *framework* Angular 2.0. Kedua jenis aplikasi ini akan berkomunikasi melalui sebuah *application programming interface* (API) dimana sebuah *token* digunakan untuk mengotorisasi pertukaran informasi antara *backend* dan *frontend*.

Berdasarkan penelitian-penelitian terkait SSO yang pernah dilakukan sebelumnya, maka penelitian selanjutnya dilakukan untuk membahas tentang implementasi SSO dengan menggunakan protokol OAuth 2.0 menggunakan *Google Identity* yang dibangun dengan arsitektur REST. Berdasarkan kajian pada penelitian sebelumnya, penggunaan *framework* untuk membangun sebuah sistem informasi sangat membantu mempercepat proses implementasi desain pada aplikasi. Pada rancangan sistem yang akan dibuat, teknologi yang akan digunakan adalah *Google Identity* sebagai ASP serta arsitektur

REST yang terdiri dari dua jenis aplikasi yaitu *backend* berteknologi Java dengan framework Spring Boot dan dua aplikasi *frontend* dengan teknologi JavaScript dengan *framework* Vue.Js dan Android.

II. METODE PENELITIAN



Gambar 2. Proses Scrum

Metode penelitian adalah langkah-langkah ilmiah untuk mendapatkan data yang akan digunakan untuk tujuan dan kegunaan tertentu [13]. Berdasarkan jenis datanya, terdapat dua macam metode penelitian yaitu kuantitatif dan kualitatif. Metode kuantitatif merupakan metode yang datanya merujuk pada suatu nilai nyata berupa angka, yang nantinya dapat dilakukan analisa. Sedangkan metode penelitian kualitatif adalah metode yang digunakan untuk meneliti kondisi obyek yang alamiah dengan cara melakukan observasi, wawancara, kuesioner, atau analisa dokumen-dokumen. Penelitian bersifat kualitatif dan data yang didapat bukan berupa angka, hasil dari penelitian kualitatif lebih menekankan pada maknanya. Oleh karena itu, dapat disimpulkan penelitian dengan pendekatan metode kualitatif mengacu pada paradigma *post-positivism* [13]. Metode penelitian dengan pendekatan kualitatif digunakan pada penelitian ini karena sistem SSO yang dibangun memiliki tujuan untuk mempermudah dan mengamankan proses autentikasi pada sistem. Dengan demikian maka metode *agile software development* adalah metode yang digunakan pada penelitian ini. Adapun *framework agile software development* yang digunakan adalah *framework Scrum* karena *framework* ini telah mengikuti prinsip-prinsip metodologi *agile* [14]. Penggunaan *framework* ini bertujuan agar sistem yang dikembangkan cepat beradaptasi dengan perubahan kebutuhan sistem yang selalu berkembang [15]. Oleh karena itu maka, siklus hidup *agile* akan digunakan sebagai metode penelitian karena memungkinkan proses implementasi fitur-fitur sistem dilakukan secara bertahap (penambahan pada setiap iterasi / *sprint*). Gambar 2 mendeskripsikan langkah-langkah pada *Scrum*.

Adapun tahap-tahap pengembangan menggunakan *framework* Scrum adalah sebagai berikut [16] :

1) *Merencanakan dan mendefinisikan Product Backlog* : Product Owner adalah orang yang bertanggung jawab menentukan spesifikasi dan mempersiapkan daftar fitur-fitur yang diinginkan pada sistem. Tahapan ini akan menghasilkan *product backlog*, yang kemudian akan dipilih dan dibagi berdasarkan prioritasnya.

2) *Sprint planning* : Tahapan dimana semua anggota tim bertemu untuk menentukan pekerjaan berdasarkan prioritas. Pada tahapan ini akan dilakukan penjabaran mendetail terhadap *product backlog* yang telah diprioritaskan. Masing-masing anggota tim akan mengukur tingkat kompleksitas dan lama pengerjaan fitur yang akan dibuat. Tahapan ini akan menghasilkan Sprint Backlog, yang berisi daftar pekerjaan yang harus selesai dalam satu iterasi sprint.

3) *Daily stand up meeting* : Merupakan mekanisme untuk memonitor performa tahap sprint yang dilakukan selama 2 minggu. Aktivitas yang dilakukan saat *daily stand up meeting* adalah melaporkan kemajuan fitur-fitur yang dikerjakan setiap anggota tim, dan memberitahukan kesulitan-kesulitan yang ditemukan. Waktu sisa pengerjaan fitur akan di perbaharui pada akhir pelaksanaan *daily stand up meeting* untuk mengetahui sisa pekerjaan yang harus diselesaikan dalam satu sprint.

4) *Sprint review* : Pada tahap ini setiap anggota tim yang telah menyelesaikan fitur-fitur pada *sprint backlog* akan mendemonstrasikannya pada seluruh anggota tim dan juga *Product Owner*. Tujuan dari tahap ini untuk menunjukkan perkembangan dan kemajuan dari pengerjaan sistem kepada para *stakeholder*.

5) *Sprint retrospective* : Pada tahap ini seluruh anggota tim akan berkumpul dan mendiskusikan kesulitan-kesulitan teknis dan non-teknis yang dihadapi selama sprint berlangsung. Pada pertemuan ini juga akan mendiskusi apa saja yang perlu ditingkatkan dan menentukan pekerjaan apa saja yang dapat dilanjutkan dan mana yang tidak.

TABEL 1
METODE OPERASI API

Metode Operasi API	Penjelasan
GET	GET digunakan untuk mendapatkan data dari *resource*.
POST	POST digunakan untuk membuat *resource*/data baru.
OPTION	OPTION digunakan untuk menjelaskan pilihan komunikasi untuk mengakses *resource*.
PUT	PUT digunakan untuk melakukan *update* menyeluruh pada *resource*/data.
DELETE	DELETE digunakan untuk menghapus *resource*/data

Sistem SSO dirancang dengan menggunakan model *client-server* dengan menggunakan protokol HTTPS yang bersifat *stateless* [17] dan menggunakan *Transport Layer Security* (TLS) untuk melakukan enkripsi data. Sebuah *client* merupakan program yang membangun koneksi ke *server* yang bertujuan untuk mengirimkan satu atau lebih *request*. Sedangkan *server* merupakan program yang menerima koneksi-koneksi agar dapat melayani *request* dengan cara mengirimkan *response*. Protokol HTTPS merujuk pada standar *Uniform Resource Identifier* (URI) yang sama dengan HTTP untuk mengindikasikan target sumber dan hubungan antar sumber, namun dengan menambahkan lapisan enkripsi TLS untuk mengamankan komunikasi data. URI digunakan sebagai petunjuk sebagian besar komunikasi HTTPS berupa permintaan *request* (GET) untuk mendapatkan representasi dari suatu *resource*. Komunikasi *client-server* terjadi saat *client* mengirimkan *request* ke *server* dalam bentuk *request message*, yang dimulai dengan sebuah *request-line* yang terdiri dari beberapa metode, URI dan versi protokol. Metode operasi HTTPS yang didukung sistem dapat dilihat pada Tabel 1. Adapun format pesan standar yang pada HTTPS adalah sebagai berikut.

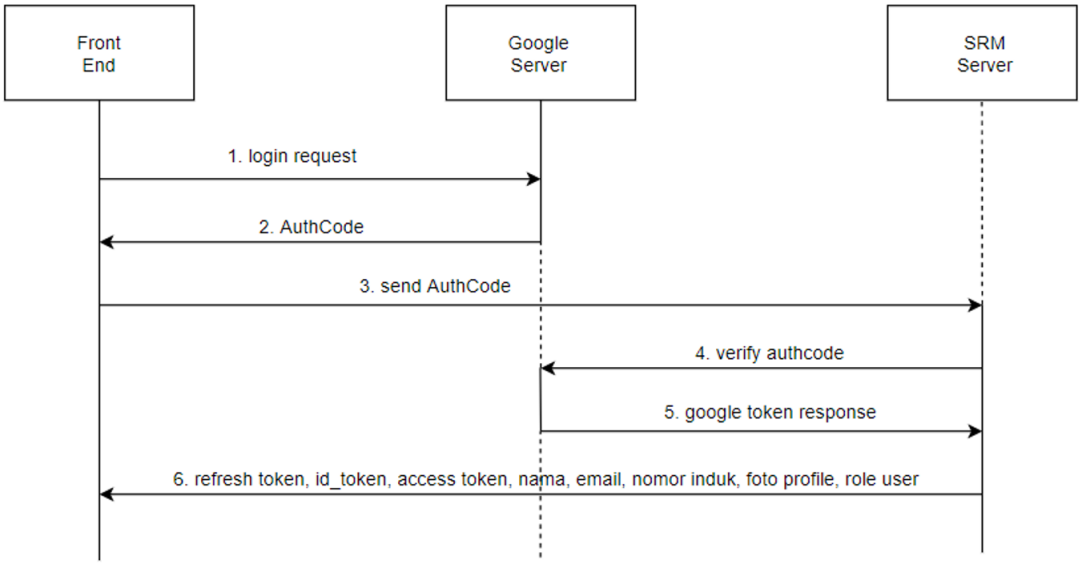
1) *Start line* : Baris pertama yang mengindikasikan apa yang harus dilakukan terhadap *request* atau apa yang terjadi terhadap *response*.
2) *Header fields* : Berisi variabel yang berpasangan berupa nama dan nilai, dipisahkan oleh tanda titik dua (:) untuk mempermudah *parsing*.
3) *Body* : Diberikan setelah *header fields* ditandai dengan baris kosong. *Message body* merupakan sebuah nilai yang sifatnya pilihan, yang dapat diisi atau tidak. Bagian ini berisi data yang dikomunikasikan antara *client* dan *server*. Tidak seperti bagian start line dan header, bagian ini dapat berisi data teks dan terstruktur, atau data biner seperti gambar, video dan audio.

Sistem ini juga dibangun menggunakan arsitektur REST yang bekerja dengan baik pada model sistem *client-server* [18]. REST merupakan arsitektur yang digunakan untuk mengimplementasi *web service* dalam menerapkan konsep perpindahan antar *state*. Untuk membuat *web service* yang memenuhi kriteria arsitektur REST maka aturan-aturan berikut haruslah terpenuhi, yaitu:

1) *Client server*: Sistem harus menerapkan model *client-server* dalam implementasinya.
2) *Stateless*: komunikasi antara *client-server* harus bersifat tidak mendeskripsikan *state* apapun. Setiap *request* dari client pada server harus berisi segala informasi yang dibutuhkan untuk dapat diproses sehingga akan meningkatkan *visibility*, *reliability* dan *scalability* sistem.
3) *Cache-able*: komunikasi menyediakan pilihan bagi *server* untuk melabeli *response* sebagai *cacheable* atau *non-cacheable*. Jika *response* berlabel *cacheable*, maka *client* memiliki hak untuk menyimpan data yang diberikan.
4) *Uniform interface* : Memisahkan antara deklarasi antarmuka *web service* dengan implementasinya. Antarmuka REST memiliki karakter sebagai berikut :
   - Identifikasi *resource* menggunakan URI.
   - Manipulasi *resource* melalui dilakukan dengan menggunakan *request method*.
   - *Message* bersifat *self-descriptive*, sehingga setiap *request* akan diproses berdasarkan isi *message* tersebut.

- *Hypermedia as the engine of application state* (HATEOAS), sehingga *client* harus dapat berinteraksi dengan *server* berdasarkan informasi yang diberikan melalui *hypermedia*.

5) *Layered system* : memungkinkan sebuah sistem terdiri dari beberapa lapisan yang tidak transparan satu sama lain.
6) *Code-on-demand*: menyediakan kemungkinan sebuah *client* untuk menambah kemampuan fungsionalnya dengan cara mengunduh dan mengeksekusi kode dalam bentuk *applets* atau *scripts*. Prinsip ini bersifat *opsional*.



Gambar 3. Diagram sekuensial *sign in*

Untuk desain implementasi SSO menggunakan protokol Oauth 2.0, terdapat dua alur yang dapat digunakan yaitu : *Authorization Code Grant* dan *Implicit Grant*. Pada *Authorization Code Grant*, memiliki jumlah langkah lebih banyak dibandingkan *Implicit Grant* yaitu melakukan pertukaran kode otorisasi untuk mendapatkan *access token* namun dapat dikelola secara independen oleh *client*. Sedangkan *Implicit Grant* digunakan hanya pada aplikasi berbasis *web* sehingga *access token* diberikan secara langsung saat *URL redirection* didapatkan oleh *client*. Oleh karena fleksibilitas pengelolaan access token yang pada alur *Authorization Code Grant*, sangat tepat digunakan pada aplikasi *web* dan *mobile* karena memberikan kemudahan pengembangan aplikasi pada platform yang berbeda. Gambar 3 merupakan desain alur *sign in* yang diimplementasikan. Adapun penjelasan setiap langkahnya sebagai berikut :

1) Permintaan *sign in* dilakukan oleh aplikasi *frontend* ke server Google dengan cara menekan tombol *Google Sign In* pada aplikasi.

2) Apabila proses *sign in* berhasil, *server* Google akan mengirimkan AuthCode untuk mendapatkan *access token*. AuthCode yang didapat bersifat sekali pakai dan memiliki kadaluwarsa maksimum 10 menit [19].

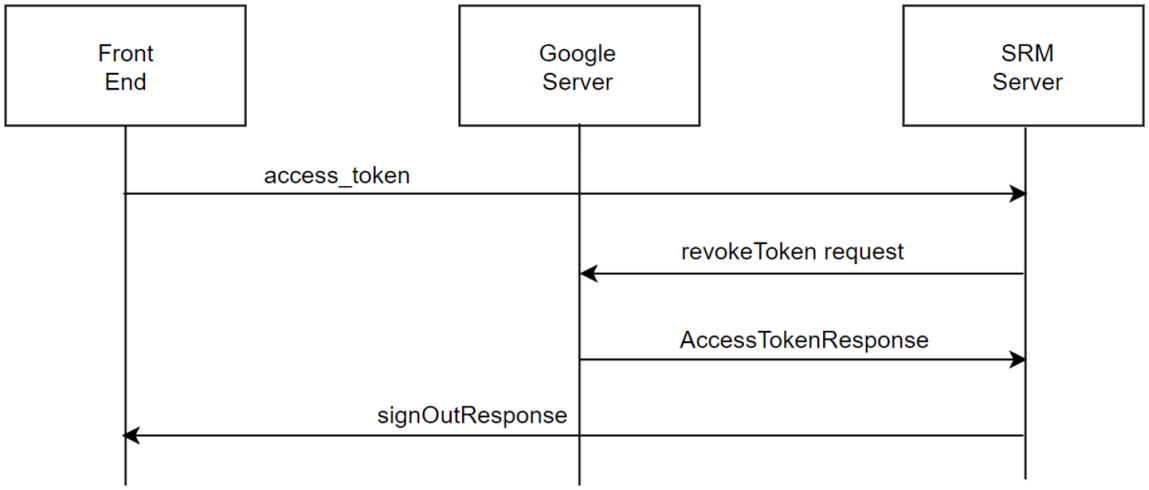
3) AuthCode dikirimkan ke *server* sistem untuk dilakukan verifikasi.

4) *Server* sistem melakukan verifikasi AuthCode ke *server* Google untuk mendapatkan *google token response*.

5) *Server* Google mengembalikan *google token response* yang berisi *access token*, *id token*, dan *refresh token*. Tidak seperti *access token* dan *id token* yang akan selalu ada saat melakukan *sign in*, *refresh token* hanya diberikan sekali saat pengguna *sign in* pertama kali. Oleh karena itu *refresh token* akan disimpan pada basis data *server* sistem. Untuk mendapatkan kembali *refresh token*, pengguna harus melakukan *revoke refresh token* yang lama.

6) *Server* akan mengirimkan *response* ke aplikasi *front end* yang berisi *refresh token*, *id token*, *access token*, nama, email, nomor induk, *foto profile*, *role user*.

p-ISSN : 2443-2210
e-ISSN : 2443-2229



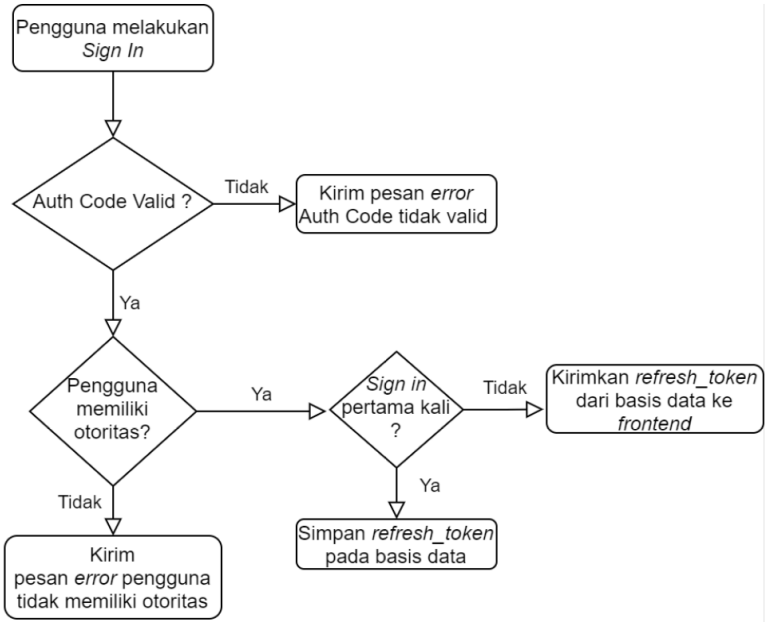
Gambar 4. Diagram sekuensial *revoke refresh token*

Untuk proses *logout* sistem, aplikasi *frontend* dapat melakukan dua pendekatan : melakukan *logout* yang bersifat lokal tanpa melakukan *revoke refresh token*, atau melakukan *logout* secara menyeluruh dengan melakukan *revoke refresh token*. Apabila proses *logout* dilakukan secara menyeluruh, maka akses pengguna ke sistem yang mungkin dilakukan melalui *frontend web* dan juga *frontend* Android akan menjadi tidak sah. Gambar 4 merupakan desain alur *revoke refresh token* yang diimplementasikan. Adapun penjelasan setiap langkahnya adalah sebagai berikut :

1) Saat pengguna menekan tombol *logout* pada aplikasi *frontend*, *access token* akan dikirimkan ke server.
2) *Server* mengirimkan permintaan *revokeToken* ke *server Google.*
3) *Server* Google mengirimkan respon *AccessTokenResponse* ke *server*.
4) *Server* memproses respon yang didapatkan, apabila sah maka *refresh token* pengguna yang tersimpan pada basis data akan dihapus.

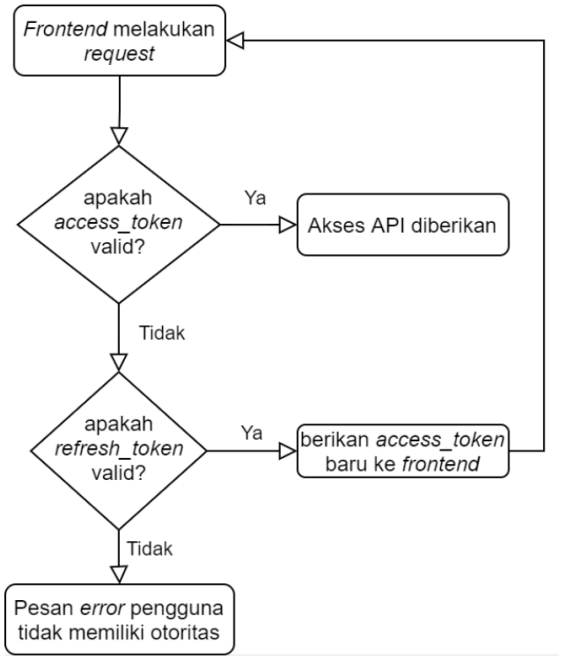
Aplikasi pada sistem ini terdiri 2 jenis yang masing-masing berperan sebagai *client* yang disebut *front-end* dan server yang bisa disebut *back-end*. Pada *client* sendiri akan terdiri dari 2 aplikasi yaitu aplikasi *web* dan juga Android yang masing-masing dikembangkan menggunakan bahasa dan *framework* berbeda. Untuk mempermudah manajemen komunikasi aplikasi Android dengan *server* maka pustaka Retrofit, Dagger dan juga RxJava digunakan. Aplikasi Android juga menggunakan arsitektur *Model-View-ViewModel* (MVVM) untuk mengakomodasi kompleksitas implementasi fitur-fitur yang ada pada aplikasi. Sedangkan pada aplikasi web akan menggunakan bahasa JavaScript dengan *framework* Vue.js, Selain itu, untuk mempermudah manajemen komunikasi jaringan aplikasi terhadap *server* maka pustaka *vue-axios*, *vue-router* dan *vuex* digunakan.

Untuk melakukan implementasi SSO dengan menggunakan Google OAuth 2.0 pada *backend*, terlebih dahulu harus membuat Google OAuth 2.0 Client ID melalui Google API Console. Proses pembuatan dilakukan dengan membuat kredensial melalui menu *Credentials*, dan pilih pembuatan kredensial *OAuth client ID*. Pada saat menu *Create OAuth client ID* ditampilkan, pilih tipe aplikasi *Web application* dan lengkapi nama kredensial yang dibuat.



Gambar 5. Diagram alur proses *sign in*

Setelah kredensial dibuat, gunakan kredensial untuk melakukan verifikasi AuthCode, pembaruan *access token* atau mendapatkan kredensial pengguna Google. Apabila proses verifikasi AuthCode berhasil, maka lakukan pemeriksaan hak akses pengguna. Apabila pengguna melakukan *sign in* untuk pertama kali, maka simpan *refresh token* pada basis data sistem. Apabila tidak, gunakan *refresh token* yang ada pada basis data untuk diberikan pada kepada *frontend*. Proses filter perlu dilakukan untuk setiap *request* yang berasal dari *frontend*. Dengan menggunakan *framework* Java Spring Boot, hal ini dengan mudah dapat dilakukan dengan membuat kelas yang diturunkan dari OncePerRequestFilter. Gambar 5 merupakan diagram alur proses *sign in.*



Gambar 6. Diagram alur proses *refresh access token*

Untuk mempermudah proses integrasi layanan *Google Sign In* dengan aplikasi Android, maka pustaka *Google Play Service API* akan digunakan. Untuk mendapatkan AuthCode yang valid agar digunakan *server*, maka konfigurasi perlu dilakukan pada GoogleSignInClient dengan menggunakan Client ID yang sama dengan aplikasi *backend*. Fungsi untuk melakukan proses otentikasi AuthCode ke *server* melalui internet dengan bantuan pustaka Retrofit dibuat untuk menentukan apakah pengguna dapat masuk ke sistem atau tidak Apabila proses autentikasi berhasil, aplikasi akan menyimpan data *access token*, *id token*, *refresh token*, beserta data-data pribadi lainnya ke memori perangkat. *Access token* akan selalu digunakan setiap melakukan *request* ke API pada *server*. Namun jika *access token* telah kedaluwarsa setelah satu jam [19], maka *refresh token* akan digunakan untuk mendapatkan *access token* baru. Gambar 6 merupakan alur *refresh access token* pada android.

Untuk mempermudah proses integrasi layanan Google *sign in* aplikasi *frontend* web dengan kerangka kerja Vue.js, maka pustaka vue-google-oauth2 (GAuth) akan digunakan. Untuk mendapatkan AuthCode yang valid agar digunakan *server*, maka konfigurasi perlu dilakukan pada GAuth agar menggunakan Client ID yang sama dengan aplikasi *backend*. Mekanisme penyimpanan *token* yang sama seperti pada *frontend* Android diterapkan pada aplikasi ini. Pustaka Axios menyediakan sebuah *interface* untuk menangkap semua anomali yang mungkin terjadi bila terjadi kegagalan karena *access token* telah kedaluwarsa.

Format data sistem SSO yang dibangun pada sistem menggunakan *message body* JSON (*JavaScript Object Notation*) sebagai format *payload request* dan *response*. JSON merupakan adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer [20]. JSON terbentuk dari dua struktur data berpasangan berupa nama/nilai yang strukturnya bersifat universal dan tidak tergantung pada bahasa pemrograman yang digunakan.

Untuk mensimulasikan terjadinya *request* pada aplikasi *backend* dilakukan dengan menggunakan aplikasi Postman. Aplikasi Postman merupakan aplikasi *client* untuk mengembangkan, menguji, dan mendokumentasikan API [21]. Untuk melakukan simulasi *request* API *sign in*, *message body* yang dikirimkan dengan format pada tabel 2 harus terpenuhi. ServerAuthCode didapat apabila pengguna telah berhasil melakukan autentikasi pada server Google. Namun untuk tujuan simulasi, ServerAuthCode juga dapat diperoleh dengan menggunakan contoh program yang telah disediakan oleh Google [8]. Adapun kode HTTP yang digunakan untuk melakukan *request* pada *server* dapat dilihat pada kode program 1.

TABEL 2
MESSAGE BODY REQUEST SIGN IN

Definisi	Keterangan
clientType	Mendefinisikan tipe dari *client*. ClientType dapat bertipe mobile_app atau web_app
serverAuthCode	*Server auth code* didapat dari SSO pada *client*.

Pada kode program 1, menunjukkan penerapan prinsip *stateless* pada REST yang semua informasi yang dibutuhkan *server* untuk melakukan *sign in* telah ditransmisikan pada *message body*. Sedangkan identifikasi *resource* dilakukan dengan menggunakan Host dan URI pada *request*. *Request method* POST digunakan karena mekanisme *sign in* akan melakukan pembuatan *access* dan *refresh token* untuk kemudian ditransmisikan kembali kepada *client* melalui pesan *response*.

```
POST /srm/auth/signin HTTP/1.1
Host: apps.dw.fti.ukdw.ac.id
Content-Type: application/json
Content-Length: 135

{
   "clientType": "mobile_app",
   "serverAuthCode":"4/0AY0e-
g7c6_uG_QShunsk998iQCGVbosjz1ZqCHhO3FoomWE6W6roFCYK3lSOiG
-QdeL9VQ"
}
```

Kode Program 1. Kode HTTP *request sign in*

Kode program 2 merupakan *response* yang akan diberikan saat *request* yang dikirimkan ke *server* berhasil diproses. Dapat diperhatikan pada bagian message body, bahwa *server* mengirimkan informasi *access token*, *id token*, *refresh token*, hak akses dan informasi kredensial pengguna. Apabila pengguna melakukan *sign in* untuk pertama kali, maka *refresh token* akan disimpan pada basis data sistem. Namun apabila pengguna sebelumnya telah melakukan *sign in* dan belum pernah melakukan *revoke token*, makan *refresh token* yang ada pada basis data untuk diberikan pada kepada *frontend*. Informasi yang ada pada nama properti *role* dapat digunakan oleh *frontend* untuk membatasi akses pengguna terhadap fitur-fitur tertentu pada sistem.

```
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Sat, 20 Feb 2021 08:47:21 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type,Authorization
Access-Control-Expose-Headers: Authorization

{
  "data": {
    "accessToken": "ya29. ",
    "idToken": "eyJhbGciOiJS ",
    "refreshToken": "1//0gVYr5 ",
    "fcmToken": null,
    "nomorInduk": "194KE417",
    "nama": "I Kadek Senapartha",
    "email": "dendy.prtha@ti.ukdw.ac.id",
    "imageUrl": "https://lh3.googleusercontent.com/ ",
    "role": "ROLE_DOSEN"
  }
}
```

Kode Program 2. Kode HTTP *response sign in*

TABEL 3
MESSAGE BODY REQUEST REVOKE TOKEN

Definisi	Keterangan
token	Dapat diisi *refresh token* atau *access token* yang akan digunakan untuk me-*revoke access* pengguna. Apabila telah *revoke* berhasil maka *refresh token* atau *access token* yang digunakan tidak akan kadaluwarsa.

```
POST /srm/auth/revoke HTTP/1.1
Host: apps.dw.fti.ukdw.ac.id
Content-Type: application/json
Content-Length: 123

{
  "token":"1//0gVYr52qKjlqwCgYIARAAGBASNwF-
L9IrN9mBeWIQWw2U1VPIL6JAdxoglHSy51V02XMSb1poa_IX1gbDQsRgmN_oyaeI0cCNQdo
"
}
```

Kode Program 3. Kode HTTP *request revoke*

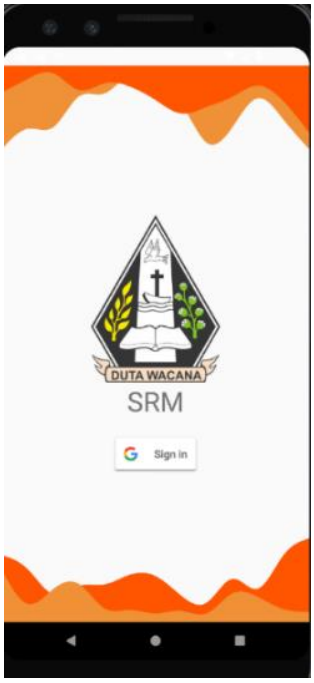
```
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Sat, 20 Feb 2021 08:47:21 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type,Authorization
Access-Control-Expose-Headers: Authorization

{
  "data": true
}
```

Kode Program 4. Kode HTTP *response revoke*

Untuk simulasi API *revoke*, maka *request* pada tabel 3 harus terpenuhi. Dan apabila *request* pada kode program 3 yang dikirimkan ke *server* dan berhasil diproses maka *response* HTTP seperti pada kode program 4 akan diberikan. Properti data pada *response body* mengindikasikan bahwa proses *revoke token* berhasil. Dan dengan demikian *refresh token* yang tersimpan pada basis data sistem menjadi tidak valid dan secara otomatis akan dihapus.

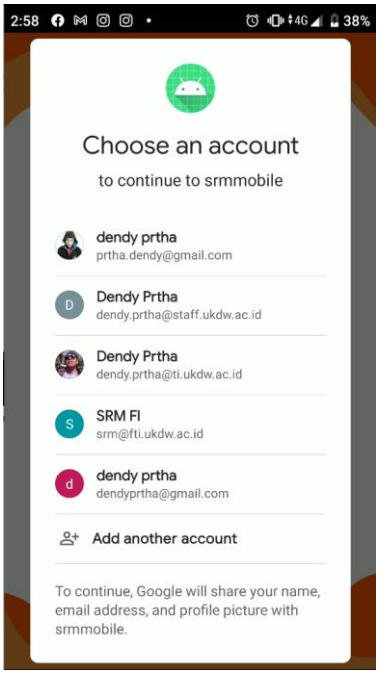
API *revoke* digunakan secara opsional oleh aplikasi *frontend* saat *sign out* dengan cara melakukan *revoke* terhadap semua *token* yang dimiliki *client*, sehingga apabila pengguna melakukan *sign in* pada dua aplikasi *frontend*, yaitu Android atau *web*, maka akses otentikasi pengguna akan menjadi *invalid*. Untuk dapat melakukan akses pada sistem, maka pengguna harus melakukan proses *sign in* kembali dengan mengirimkan AuthCode ke *server* untuk mendapatkan *access token* dan *refresh token* yang baru. Apabila *frontend* tidak menggunakan *revoke* saat *sign out*, *token* dan data kredensial pengguna harus dihapus dari memori.

III. HASIL DAN PEMBAHASAN

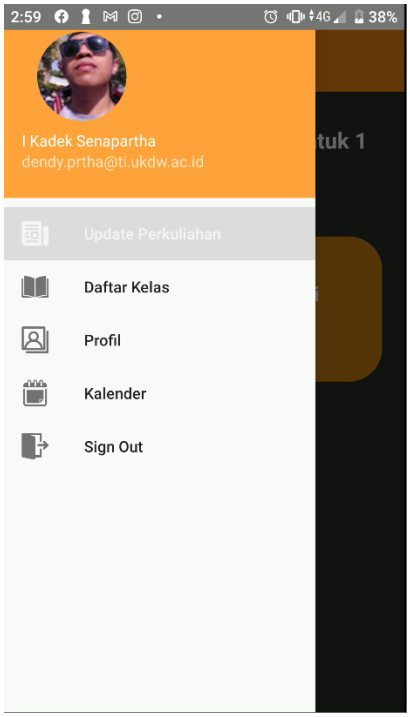


Gambar 7. Antarmuka *sign in* aplikasi Android

Implementasi sistem SSO pada aplikasi *frontend* Android dilakukan dengan menggunakan bantuan Android Studio agar mempermudah proses *debugging*. Hasil implementasi pada *frontend* Android dapat dilihat pada gambar 7. Aplikasi ini diperuntukkan bagi dosen dan mahasiswa yang mana antarmuka *sign in* merupakan tampilan pertama yang akan dilihat oleh pengguna. Apabila tombol *Sign in* ditekan, aplikasi akan menampilkan antarmuka *sign in* Google. Pengguna dapat memasukkan *username* dan *password* Google yang telah terdaftar pada sistem.

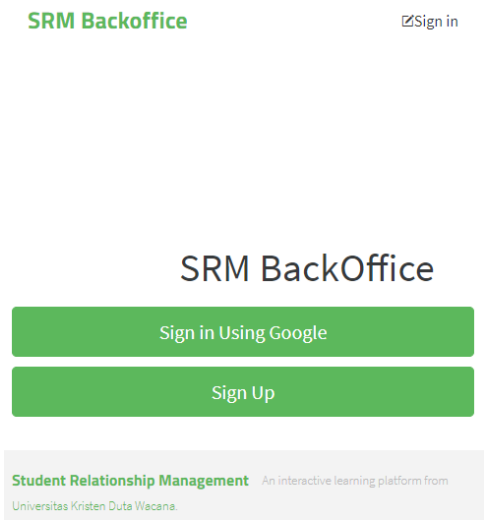


Gambar 8. Tampilan *Google user concent* pada Android



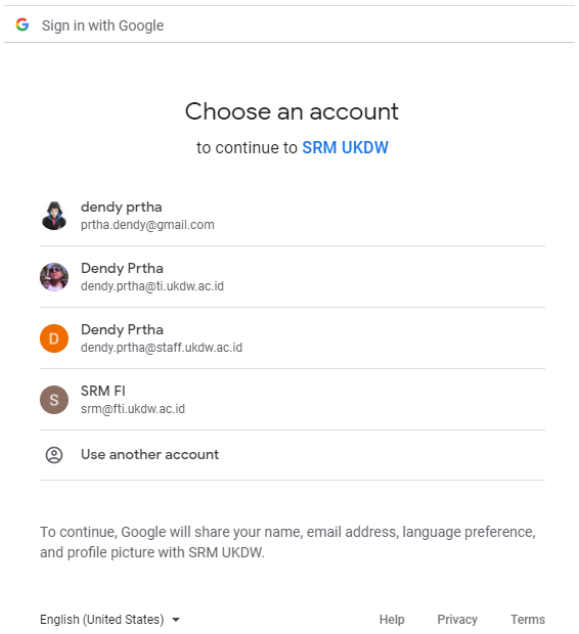
Gambar 9. Tampilan *home screen* aplikasi Android

Apabila pengguna telah menggunakan akun yang telah tersimpan pada perangkat Android, pengguna langsung dapat memilih untuk menggunakannya secara langsung sehingga dengan demikian akan memudahkan proses autentikasi. Aplikasi Android yang telah mengintegrasikan Google API akan langsung menampilkan *Google user concent screen*, untuk memberikan izin akses sistem terhadap data-data yang dibutuhkan. Gambar 8 merupakan tampilan *Google user concent screen* yang dimaksud. Apabila proses *sign in* berhasil, pengguna akan langsung masuk ke dalam *home screen* seperti yang terlihat pada gambar 9. Pada *side menu* aplikasi, pengguna dapat melakukan *logout*, yang akan memanggil API *revoke* dan menghapus *session* yang ada pada perangkat Android.



Gambar 10. Tampilan *sign in* aplikasi web

Implementasi SSO pada aplikasi *frontend web* dilakukan dengan menggunakan bantuan Visual Studio Code untuk mempermudah proses *debugging*. Antarmuka *sign in* aplikasi web dapat dilihat pada gambar 10. Aplikasi *frontend web* diperuntukkan sebagai aplikasi manajemen sistem. Oleh karena itu aplikasi ini hanya dapat diakses oleh pengguna yang memiliki hak akses dosen atau *admin* saja.



Gambar 11. Tampilan *Google user concent* pada aplikasi web

Gambar 12. Tampilan *home screen* aplikasi web.

Apabila tombol s*ign in* ditekan, aplikasi akan menampilkan antarmuka *sign in* Google. Pengguna dapat memasukkan *username* dan *password* yang telah terdaftar pada sistem. Namun apabila akun Google terdaftar pengguna pernah melakukan *sign in* menggunakan pada *browser*, pengguna akan dapat menggunakannya secara langsung untuk mempermudah proses autentikasi. Aplikasi web akan menampilkan *Google user concent screen*, untuk memberikan izin akses sistem terhadap data-data yang dibutuhkan. Gambar 11 merupakan tampilan *Google user concent screen* yang dimaksud. Apabila proses *sign in* berhasil, pengguna akan masuk ke dalam *home screen* seperti yang terlihat pada Gambar 12. Pada *side menu* aplikasi, pengguna dapat melakukan *logout*, yang akan menghapus *session* yang ada pada *browser*.

Selain untuk melakukan *sign in* dan *revoke token*, setiap *request* yang dikirimkan *frontend* ke *server* dilakukan menggunakan protokol HTTPS dengan menyertakan *access token* pada *header request* yang dikirimkan. Penggunaan protokol HTTPS untuk transmisi data akan memberikan lapisan enkripsi sehingga akan membuat sebuah kanal yang aman. Sedangkan metode pengiriman *access token* menggunakan *request header* telah sesuai rekomendasi metode implementasi keamanan OAuth 2.0 [19]. Dengan menggunakan metode ini maka dapat mengantisipasi pengiriman *token* melalui URL sehingga mencegah pengguna lain mengaksesnya melalui *history* penggunaan *browser*. Selain itu apabila *token* dikirimkan menggunakan *message body*, maka akan mengabaikan rekomendasi implementasi arsitektur REST yang menyebabkan *message* menjadi tidak *self-descriptive* karena informasi *access token* tidak memiliki hubungan langsung dengan informasi yang dibutuhkan saat melakukan *request* ke *server*.

IV. Simpulan

Implementasi SSO dengan menggunakan protokol OAuth 2.0 berhasil dilakukan pada sistem SRM dengan menggunakan *Google Identity* dan arsitektur REST. Implementasi SSO pada aplikasi *client* Android dan Web mempermudah proses akses pengguna ke sistem karena pengguna tidak perlu melakukan *sign in* berkali-kali setiap akan menggunakan aplikasi. Implementasi mekanisme *refresh access token* protokol OAuth 2.0 akan menjamin validitas akses pengguna sistem karena akan selalu memeriksa otentikasi, otorisasi dan validitas *access token* yang digunakan *client*. Penggunaan layanan yang disediakan oleh *Google Identity* dapat mempercepat dan mempermudah proses implementasi SSO pada sistem karena pengembang sistem hanya perlu berfokus pada bagaimana mengamankan transmisi data SSO pada sistem. Penggunaan arsitektur REST dengan format data JSON dapat mempercepat komunikasi yang dilakukan antara *client* dan *server* karena mekanisme komunikasi yang sederhana.

Ucapan Terima Kasih

Terima kasih diberikan kepada Fakultas Teknologi Informasi Universitas Kristen Duta Wacana yang telah mendanai pengembangan SSO pada sistem SRM ini. Dengan sistem ini diharapkan dapat meningkatkan layanan universitas kepada mahasiswa.

Daftar Pustaka

[1] Y. D. Handarkho, “Pengembangan Sistem Student Relationship Management (SRM) dan Penerapannya pada Perguruan Tinggi di Indonesia (Studi Kasus MTI UGM Yogyakarta),” pp. 1–13, 2018.

[2] M. Fontaine, “Student Relationship Management (SRM) in Higher Education: Addressing the Expectations of an Ever-Evolving Demographic and

Its Impact on Retention," *J. Educ. Hum. Dev.*, vol. 3, no. 2, pp. 105–119, 2014.
[3] T. Bazaz and A. Khalique, "A Review on Single Sign-on Enabling Technologies and Protocols," *Int. J. Comput. Appl.*, vol. 151, no. 11, pp. 18–25, 2016.
[4] A. Patil, P. R. Pandit, and P. S. Patel, "Analysis of Single Sign-on for Multiple Web Applications," *Electr. Electron. Instrum. Eng.*, vol. 2, no. 8, pp. 4103–4110, 2013.
[5] A. Soni and V. Ranga, "API features individualizing of web services: REST and SOAP," *Int. J. Innov. Technol. Explore. Eng.*, vol. 8, pp. 664–671, 2019.
[6] S. Kemp, "DIGITAL 2020: OCTOBER GLOBAL STATSHOT," 2021. https://datareportal.com/reports/digital-2020-october-global-statshot (accessed Feb. 17, 2021).
[7] [2021] Current Ware Website.[Online]. Tersedia: https://www.currentware.com/best-email-service-providers-2021/
[8] [2021] Google Website. [Online] Tersedia: https://developers.google.com/ (accessed Feb. 17, 2021).
[9] N. Hossain, M. A. Hossain, M. Z. Hossain, M. H. I. Sohag, and S. Rahman, "OAuth-SSO: A Framework to Secure the OAuth-Based SSO Service for Packaged Web Applications," *Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust. 2018*, no. August, pp. 1575–1578, 2018.
[10] N. Naik and P. Jenkins, "An Analysis of Open Standard Identity Protocols in Cloud Computing Security Paradigm," *Proc. - 2016 IEEE 14th Int. Conf. Dependable, Auton. Secur. Comput. DASC 2016, 2016 IEEE 14th Int. Conf. Pervasive Intell. Comput. PICom 2016, 2016 IEEE 2nd Int. Conf. Big Data*, pp. 428–431, 2016.
[11] A. Suhardi, E. Fatkhiyah, and M. Sholeh, "Perancangan dan Implementasi SSO (Single Sign On) Menggunakan Protokol Oauth 2.0," *J. JARKOM*, vol. 5, no. 1, pp. 65–75, 2017.
[12] Q. Aini, U. Rahardja, and R. S. Naufal, "Penerapan Single Sign On dengan Google pada Website berbasis YII Framework," *Sisfotenika*, vol. 8, no. 1, p. 57, 2018.
[13] Sugiyono, *Metode Penelitian Kuantitatif, Kualitatif dan R&D*. Bandung: Alfabeta, 2013.
[14] A. Pham and P.-V. Pham, *Scrum in Action: Agile Software Project Management and Development*. 2011.
[15] T. S. l U. Hansmann, *Agile Software Development Best Practices for Large Software Development Projects*. 2010.
[16] P. Adi, "Scrum Method Implementation in a Software Development Project Management," *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 9, pp. 198–204, 2015.
[17] [2014] Tools IETF website. [Online]. Tersedia: https://tools.ietf.org/html/rfc7230.
[18] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.
[19] [2012] Tools IETF website.[Online]. Tersedia: https://tools.ietf.org/html/rfc6749.
[20] [2021] JSON website. [Online]. Tersedia: https://www.json.org/json-en.html.
[21] [2021] Postman website. [Online]. Tersedia : https://learning.postman.com/.