PENERAPAN MEKANISME *AUTHENTICATION* DENGAN METODE *OPEN AUTHENTICATION* (OAuth) (STUDI KASUS : belajardisini.com)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Afrianda Cahyapratama
NIM: 125150202111004

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

PENERAPAN MEKANISME *AUTHENTICATION*
DENGAN METODE *OPEN AUTHENTICATION* (OAuth)
(STUDI KASUS : belajardisini.com)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Afrianda Cahyapratama
NIM: 125150202111004

Skripsi ini telah diuji dan dinyatakan lulus pada
25 Agustus 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eko Sakti P., S.Kom., M.Kom.	Adhitya Bhawiyuga, S.Kom, M.S.
NIK: 201102 860805 1 001	NIK: 201405 890720 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 19 Agustus 2016

Afrianda Cahyapratama
NIM: 125150202111004

KATA PENGANTAR

Puji dan syukur penulis panjatkan atas kehadirat Allah SWT yang selalu memberikan limpahan rahmat dan hidayahNya, sehingga penulis dapat menyelesaikan laporan skripsi yang berjudul "PENERAPAN MEKANISME *AUTHENTICATION* DENGAN METODE *OPEN AUTHENTICATION* (OAuth) (STUDI KASUS : belajardisini.com)".

Penulis sangat menyadari bahwa skripsi ini tidak dapat terselesaikan tanpa bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih yang sebesar-besarnya kepada:

1. Bapak Eko Sakti Pramukantoro, S.Kom., M.Kom dan Adhitya Bhawiyuga, S.Kom., M.S selaku dosen pembimbing skripsi yang telah membimbing dan mengarahkan penulis untuk dapat menyelesaikan skripsi ini.
2. Seluruh dosen dan civitas Program Studi Informatika, Universitas Brawijaya, atas dukungan dan kerjasamanya.
3. Bapak H. Moh Arip, S.Kp., M.Kes dan Ibu Sri Susantini, S.Ag. selaku orang tua penulis beserta seluruh anggota keluarga yang selalu memberikan doa dan dorongan semangat sehingga telah banyak membantu dalam kelancaran penulisan skripsi ini.
4. Siti Rizky Amalia, Charlina Mariane, S.E., Fathaillah Liestianto, Amd.Kep., Nena Riski Hariyati, STR.Keb., Rizal Ziadi, S.Pd., Pande Agung Mahariski, S.Ked., Fitria Intan Ningtyas, Deshinta Putri, Mardian Dwi Cahyo, sahabat-sahabat terbaik yang selalu memberikan semangat serta menjadi motivasi tersendiri dalam menyelesaikan kuliah.
5. Melinda Dwi Astari, S.Kg., Baiq Junita Anggun Rarasati, S.H., Wahidha Prika Febriani, S.E., yang banyak memberikan masukan, dorongan semangat serta waktunya untuk mendengarkan segala cerita di masa perkuliahan.
6. Aditya Sudarmadi, S. Agustian Putra, Jodie Putra Kahir, Yazid, Aditya Wahyu Iswarahadi, Putra Aditya dan seluruh teman-teman Program Studi Informatika Universitas Brawijaya angkatan 2012 untuk segala bantuan dan kerjasamanya.

Semoga segala bantuan yang telah diberikan kepada penulis mendapat balasan dari Allah SWT. Penulis menyadari bahwa skripsi ini tidak sempurna karena keterbatasan materi dan pengetahuan yang dimiliki penulis, sehingga penulis menerima saran dan kritikan yang dapat disampaikan melalui *e-mail* penulis. Akhirnya, semoga skripsi ini dapat bermanfaat dan berguna bagi pembaca.

Malang, 19 Agustus 2016

Penulis

afriandacahyapratama@gmail.com

ABSTRAK

Afrianda Cahyapratama. 2016. Penerapan Mekanisme *Authentication* Dengan Metode *Open Authentication* (OAuth). (Studi Kasus : belajardisini.com). Skripsi program studi Informatika/Ilmu Komputer. Fakultas Ilmu Komputer. Universitas Brawijaya, Malang. Pembimbing : Eko Sakti P., S.Kom, M.Kom dan Adhitya Bhawiyuga, S.Kom, M.S.

OAuth merupakan mekanisme yang berperan dalam mengatur hak akses sumber daya yang dilindungi, karena sumber daya ada yang bersifat terbuka dan ada yang terbatas. OAuth menawarkan mekanisme otentikasi untuk mendapatkan hak akses. Pada sistem terdahulu belajardisini.com, web service yang digunakan belum menerapkan mekanisme otentikasi. Oleh karena itu pada penelitian ini diterapkan mekanisme kerja teknologi OAuth 2.0 pada situs belajardisini.com dengan melibatkan adanya server otorisasi yang merupakan sumber daya server itu sendiri dalam melakukan otentikasi dan otorisasi *credential* (*client_id*, *client_secret*, *redirect_uri*, *api_key*) dari *client*. Otentikasi OAuth 2.0 melalui peran server otorisasi akan dilakukan validasi mandat dari *client*, di proses dengan mengeluarkan sebuah halaman otorisasi untuk diarahkan ke halaman utama setiap aplikasi *client*. Otorisasi akhir dari kebenaran mandat milik *client* adalah di hasilkan sebuah kode otorisasi yang bekerja pada URL (*uniform resource locator*) pada setiap aplikasi *client*. Berdasarkan hasil penelitian dapat disimpulkan bahwa mekanisme otentikasi dengan menggunakan OAuth terbukti dapat menjembatani peminta layanan dan pemilik sumber daya dengan adanya mekanisme otentikasi dan otorisasi dari peran serta pengguna dalam mengakses layanan belajardisini.com.

Kata Kunci - OAuth 2.0, Server Otorisasi, Otentikasi dan Otorisasi.

ABSTRACT

Afrianda Cahyapratama. 2016. Implementation of Authentication Mechanism With Open Authentication Methods (OAuth). (Case of Study : belajardisini.com). Thesis of study program Informatics/Computer science. Computer science faculty. Brawijaya University, Malang. Adviser : Eko Sakti P., S.Kom, M.Kom and Adhitya Bhawiyuga, S.Kom, M.S.

OAuth is a mechanism whose role in regulating as a right of access to a protected resource, because there are resources that are open and that are limited. OAuth offer an authentication mechanism to get a permission. In the previous system used by belajardisini.com, the web service have not implemented an authentication mechanism. Therefore, in this study the OAuth 2.0 technology mechanisms is implemented on belajardisini.com site involving the authorization server which is a resource server itself in the authentication and authorization credentials (client_id, client_secret, redirect_uri, api_key) from the client. OAuth 2.0 Authentication mechanism via Authorization Server will validate the role of client credential, in the process by issuing an authorization page (Authorize Request) to be redirected to the main page of each respective client application. Final authorization of the client credential validity is generate an authorization code that works on a URL (uniform resource locator) on each client application. Based on the results of this study concluded that using OAuth authentication mechanism is proven to connect the requester and resource owners with their authentication and authorization mechanisms of the participation of users in accessing services belajardisini.com.

Keywords - OAuth 2.0, Authorization Server, Authentication and Authorization.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Konsep teknologi *web service* pada saat ini dikembangkan untuk mendukung sistem terdistribusi yang beroperasi pada infrastruktur yang berbeda. *Web service* merupakan teknologi yang turunkan dari sebuah aplikasi web yang dapat dipublikasikan, diletakkan, dan dibangkitkan pada *platform* yang berbeda. Berbeda halnya dengan *web page* jika dibandigkan dengan *web service* dari sisi tampilan akan terlihat berbeda, *web page* memiliki desain serta tampilan yang menarik, sedangkan *web service* hanya memiliki desain tampilan yang sangat sederhana, namun hanya menyediakan fungsi-fungsi yang bisa dimanfaatkan oleh pengguna lain untuk membuat aplikasi. Dengan kesuksesan *web service* yang menjadi suatu standar teknologi perangkat lunak, membuka peluang besar untuk pengembangan aplikasi terdistribusi melalui Internet. Saat ini teknologi *web service* sudah semakin berkembang tidak hanya bisa diakses melalui komputer saja, tetapi juga dapat diakses melalui perangkat bergerak, sehingga memungkinkan untuk diciptakannya aplikasi *mobile* yang menggunakan *web service* (Wellem, 2009).

Seiring dengan perkembangan *web service*, sudah terdapat beberapa arsitektur *web service* dan salah satunya yang dibahas dalam penelitian ini adalah arsitektur REST (*Representational State Transfer*). REST memiliki 2 format pertukaran data, yaitu format XML dan format JSON (*Java Script Object Notation*). *Web service* dengan format pertukaran data JSON memiliki performa yang lebih baik karena memiliki ukuran data lebih kecil dibandingkan format XML sehingga dalam proses parsing data menjadi lebih cepat (Nurseotiv, et al., 2009). Dengan REST, komunikasi yang terjadi dapat memudahkan penyedia dan penguna *resource* atau layanan. Namun, *resources* ada yang bersifat terbuka dan ada juga yang di proteksi sehingga untuk dapat mengakses ke *resources* tersebut diperlukan sebuah mekanisme *authentication* untuk dapat mengenali penguna layanan. Begitu juga pada situs belajardisini.com *web service* yang digunakan adalah arsitektur REST sehingga memungkinkan bagi pengguna lain untuk dapat memanfaatkan *resource* atau layanan pada situs tersebut.

Open Authentication atau biasa dikenal dengan singkatan OAuth adalah protokol *authentication* yang berasal dari layanan penyedia API, yang memberikan kuasa kepada seseorang untuk memperoleh hak akses mereka dengan cara menukarkan *credential* (*username dan password*) menjadi *code accses* (Kaur & Aggarwal, 2013). *Authentication* dengan menggunakan OAuth mengatur agar pengguna tetap aman ketika akan menggunakan layanan aplikasi dengan memberikan sebuah kepercayaan kehadiran pihak ketiga (*third party*) yang telah dipercayai sebagai sumber server (*resource server*) yang memiliki kuasa penuh untuk menertibkan dan mengatur segala *credential* yang ada. Sumber server juga merupakan *authorization server* yang melakukan proses otorisasi *client* dalam melakukan permintaan API dengan menjadikan *authorization code* untuk menjadi

code accses (Hardt, 2012). Pada intinya, *OAuth* adalah mekanisme *authentication* untuk aplikasi saat peminta layanan akan mengakses API atas nama pengguna tanpa harus memiliki *username* dan *password*. Sebaliknya, aplikasi mendapatkan *token* yang dapat mereka gunakan dengan mandat aplikasi mereka sendiri untuk membuat panggilan API (asana.com, -).

Situs belajardisini.com juga sudah menerapkan *web service*, akan tetapi dalam situs tersebut belum menerapkan fitur yang menangani proses otentikasi dalam mengakses sumber daya/layanan pada *web service*, sehingga diperlukan sebuah mekanisme otentikasi dan otorisasi untuk dapat menangani peminta layanan yang akan mengakses sumber daya/layanan pada situs tersebut. Oleh karena itu, pada penelitian skripsi ini penulis mencoba untuk menerapkan mekanime *authentication* pada situs website belajardisini.com dengan menggunakan metode *OAuth* (*Open Authentication*), karena hal ini dirasa lebih aman dan sederhana untuk proses otorisasi.

1.2 Rumusan Masalah

Dari pembahasan latar belakang yang telah dijelaskan diatas, maka dapat disusun rumusan masalah dengan rincian sebagai berikut:

1. Menerapkan OAuth pada *web service* belajardisini.com.
2. Bagaimana kinerja sistem OAuth pada *web service* belajardisini.com?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah untuk menerapkan metode *OAuth* dalam mekanisme *authentication* pada *web service* belajardisini.com serta melakukan pengujian sistem setelah dilakukan penerapan mekanisme *authentication* dengan menggunakan metode OAuth.

1.4 Manfaat

Adapun manfaat yang diperoleh dari penelitian yang sudah dilakukan yaitu :

a. Bagi penulis, penelitian ini diharapkan bisa menjadi bekal untuk menambah wawasan dan pengalaman sebagai realisasi dari yang dipelajari selama perkuliahan dengan kenyataan yang sebenarnya serta mendapatkan hasil uji kelayakan penggunaan OAuth dalam mekanisme *authentication*.

b. Bagi akademik, penelitian ini diharapkan dapat mengetahui kemampuan mahasiswa/penulis dalam menemukan solusi dari masalah yang ada sehingga dapat menghasilkan solusi terbaik yang telah ditawarkan oleh penulis dan memberikan kontribusi kepada prodi Informatika/Ilmu Komputer Fakultas Ilmu Komputer Universitas Brawijaya.

c. Bagi pengguna, penelitian ini diharapkan bermanfaat dengan menggunakan OAuth dapat membangun mekanisme *authentication* yang dapat

mengamankan informasi dalam sebuah aplikasi dan dapat dijadikan salah satu alternatif atau media belajar yang bermanfaat.

1.5 Batasan Masalah

Sesuai dengan terpaparnya permasalahan yang terdapat pada latar belakang agar tidak memperluas area bahasan maka diperlukan suatu batasan sebagai berikut :

1. Mekanisme *Authentication* yang digunakan untuk menyelesaikan penelitian ini adalah metode *OAuth* (*Open Authentication*).
2. Pertukaran data dalam format JSON (*Java Script Object Notation*).
3. Menggunakan protokol HTTP. Karena pada penggunaan protokol HTTPS perlu menggunakan serifikat keamanan yang kuat dan terpercaya (*trusted*) untuk dapat mengaktifkan protokol HTTPS pada situs terkait.
4. Untuk aplikasi dari *client* dilakukan implementasi menggunakan media yang berbasis android dengan fitur *sign-in*.

1.6 Sistematika Pembahasan

Penyusunan proposal skripsi ini menggunakan kerangka pembahasan yang tersusun sebagai berikut :

BAB 1 PENDAHULUAN
Bab ini memuat latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan penelitian, sistematika pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN
Pada bab ini memuat tentang pengertian dan konsep teori-teori yang akan digunakan sebagai dasar untuk menunjang penyusunan skripsi.

BAB 3 METODOLOGI
Pada bab ini memuat tentang metode yang digunakan dalam melakukan penelitian skripsi mulai dari tahapan studi literatur, perancangan, implementasi, hingga pengujian hasil dari implementasi.

BAB 4 PERANCANGAN
Pada bab ini memuat tentang gambaran umum arsitektur awal sistem dan perancangan arsitektur baru yang dikembangkan.

BAB 5 IMPLEMENTASI

Pada bab ini memuat tentang arsitektur baru sistem yang telah dikembangkan serta pembahasan hasil implementasi metode OAuth (*Open Authentication*) untuk proses *authentication* pada sebuah *web service*.

BAB 6 PENGUJIAN

Pada bab ini memuat pengujian sistem OAuth berdasarkan implementasi yang telah dilakukan.

BAB 7 PENUTUP

Pada bab ini memuat tentang kesimpulan dan saran sesuai hasil penelitian berdasarkan uraian-uraian dari bab-bab sebelumnya untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas mengenai landasan kepustakaan meliputi dasar teori yang mendasari dari penelitian. Dasar teori merupakan pembahasan teori yang diperlukan untuk memperkaya pengetahuan dalam menyusun penelitian yang diusulkan.

2.1 Landasan Teori

Berikut ini merupakan teori-teori yang digunakan dalam penelitian penerapan mekanisme *authentication*. Adapun penjelasan-penjelasan teori tersebut seperti berikut :

2.1.1 Definisi *Web Service*

World Wide Web Consortium atau biasa dikenal dengan W3C mengartikan *web service* sebagai suatu sistem perangkat lunak yang dirancang untuk meningkatkan performa interoperabilitas dan interaksi antar mesin/sistem pada suatu jaringan (w3.org, 2005). *Web service* memiliki fungsi sebagai suatu fasilitas yang tersedia pada sebuah *website* untuk memberikan layanan (dalam bentuk informasi) kepada sistem lain, sehingga sistem yang lainnya dapat melakukan interaksi dengan sistem yang bersangkutan melalui layanan(*services*) yang telah diberikan oleh suatu sistem yang menyediakan *web service*. *Web service* melakukan penyimpanan data dalam format pesan universal (misal: JSON), sehingga data tersebut bisa diakses oleh sistem lain meskipun dengan *platform*, sistem operasi, maupun bahasa pemrograman yang berbeda.

2.1.2 REST *Web Service*

REST merupakan singkatan dari *Representational State Transfer*. Istilah REST atau RESTful pertama kali diperkenalkan oleh Roy Fielding pada disertasinya di tahun 2000 (Fielding, 2000). REST bukanlah sebuah standar protokol *web service*, melainkan hanya sebuah gaya arsitektur. Ide dasar dari arsitektur REST adalah bagaimana menghubungkan jalur komunikasi antar mesin/aplikasi melalui HTTP sederhana. Sebelum adanya REST, komunikasi antar mesin/aplikasi dilakukan dengan menggunakan beberapa mekanisme atau protokol *middleware* yang cukup kompleks seperti DCE, CORBA, RPC, ataupun SOA.

Arsitektur REST mampu mengeksploitasi berbagai kelebihan dari HTTP yang digunakan untuk kebutuhan web service. HTTP sendiri merupakan sebuah protokol standar di dunia *Worl Wide Web* (WWW) yang berbasis *synchronous request/response*. Protokol tersebut sangat sederhana : *client* mengirimkan *request message* yang mencakup HTTP method yang akan diinvokasi, lokasi *resource* dalam format URI, serta pilihan format pesan (pada dasarnya dapat berupa format apa saja seperti HTML, plain text, XML, JSON ataupun binary), kemudian server akan mengirimkan *response* sesuai dengan spesifikasi yang diminta oleh *client*. Selama ini, yang berfungsi sebagai *client* adalah sebuah *web browser* yang memfasilitasi komunikasi antara mesin dengan manusia. Dengan

adanya REST, aplikasi *client* dapat berupa aplikasi apa saja termasuk aplikasi berbasis *mobile* hanya dengan memanfaatkan HTTP .

Ada beberapa prinsip arsitektur dari REST yang dikutip dari sebuah buku berjudul "*RESTful Java with JAX-RS*" (Burke, 2014):

1. Addressability

Addressability merupakan sebuah ide dimana setiap objek dan resources pada suatu siste dapat dicapai hanya dengan melalui sebuah *unique identifier*. Pada lingkungan REST *addressability* dikelola dengan penggunaan *Uniform Resource Identifier* (URI). Format sebuah URI telah distandarisasi seperti berikut :

```
scheme://host:port/path?queryString#fragment
```

Gambar 2.1 Format URI
sumber: (Burke, 2014)

Scheme merupakan nama protocol yang akan digunakan. Untuk REST *web service*, protocol yang biasa digunakan adalah HTTP atau HTTPS. *Host* merupakan nama domain atau IP *address* dan diikuti sebuah *port* yang bersifat opsional. *Path* merupakan urutan yang akan menjadi jalur untuk menuju file yang akan dituju. *QueryString* merupakan daftar parameter yang direpresentasikan sebagai pasangan nama/nilai. Dan *fragment* merupakan sebuah arahan untuk sumber daya, seperti contoh judul bagian dalam sebuah artikel. Ketika sumber daya utama adalah HTML dokumen, *fragment* sering merupakan *id* atribut dari elemen tertentu. Berikut ini contoh pengaksesan *resource* melalui URI :

```
https://example.com/customers?lastName=Ghifar&zipcode=40293
```

Gambar 2.2 Contoh Akses Resource
Sumber : (Burke, 2014)

2. Constrained & Uniform Interface

Ide dari prinsip ini adalah menyediakan berbagai layanan melaui antarmuka atau pemanggilan *method/procedure* yang seragam. Pada sistem CORBA ataupun SOAP, pengembang *client* harus mengetahui *method* apa saja yang disediakan oleh *web service* server. Pemanggilan *method* tersebut dikenal dengan istilah RPC (*Remote Procedure Call*). Namun pada sistem REST, *method/procedure* yang digunakan untuk layanan apapun hanyalah *method-method* yang disediakan pada HTTP. Istilah yang biasa digunakan untuk menyatakan pronsip *uniform interface* pada REST adalah CRUD (*Create, Read, Update, Delete*). Berikut adalah ulasan detail mengenai *method-method* tersebut:

Tabel 2.1 Tabel Method CRUD

Method	Keterangan	Contoh Resource
GET	Merupakan operasi *read-only* yang digunakan untuk meminta informasi pada server dalam bentuk *query*. Karakteristik dari operasi GET adalah *idempotent* dan *safe*. *Idempotent* berarti sebanyak-banyak apapun operasi ini dilakukan hasilnya akan tetap sama. Sedangkan *safe* berarti ketika operasi ini diinvokasi tetap tidak mengubah *state* di server.	`function user_get($id_param=NULL){ $id = $this->get('id'); if($id===NULL){ $id = $id_param; } if($id === NULL){ $user=$this>Model_user>read($id); if($user){ $this->response($user, REST_Controller::HTTP_OK); }else{ $this->response(['status'=>FALSE, 'error'=>'Data tidak ada'], REST_Controller::HTTP_NOT_FOUND); } } $user=$this->Model_user>read($id); if($user){ $this->response($user, REST_Controller::HTTP_OK); }else{ $this->response(['status'=>FALSE, 'error'=>'Data tidak ada..'], REST_Controller::HTTP_NOT_FOUND); } }`
PUT	Merupakan operasi untuk meminta kepada server agar membuat sebuah *resource* baru.	`public function user_put(){ $user=$this->input>input_stream(); $this->Model_user->update($user); $message=['Level' => $user['level'], 'Email' => $user['email'], 'Nama' => $user['nama'], 'Username' => $user['username'], 'Password' => $user['password'], 'Status' => $user['status'], 'message' => 'Added a resource']; $this->response($message, REST_Controller::HTTP_CREATED); }`
DELETE	Digunakan untuk menghapus *resource* tertentu.	`public function user_delete(){ $id=$this->uri->segment(3); $id = (int) $this->get('id'); if($id===NULL){`

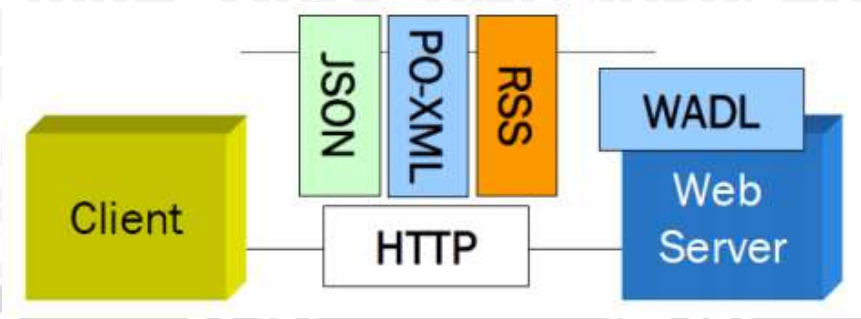
		`$this->response(['status'=>FALSE, 'error'=>'ID cannot be empty'], REST_Controller::HTTP_NOT_FOUND); } $user=$this->Model_user>delete($id); if($user){ $this->response($user, REST_Controller::HTTP_OK); }else{ $this->response(['status'=>FALSE, 'error'=>'Record could not be found'], REST_Controller::HTTP_NOT_FOUND); } }`
POST	Merupakan operasi untuk membuat *resource* baru (PUT) ataupun memodifikasi *resource* yang sudah ada.	`public function user_post(){ $user=array('level'=>$this>input->post('level'), 'email'=>$this->input>post('email'), 'nama'=>$this->input->post('nama'), 'username'=>$this->input->post('username'), 'password'=>$this->input->post('password'), 'status' => $this->input->post('status')); $this->Model_user->insert($user); $message=['Level'=>$user['level'], 'Email'=>$user['email'], 'Nama' =>$user['nama'], 'Username'=>$user['username'], 'Password'=>$user['password'], 'Status'=>$user['status'], 'message'=>'Added a resource']; $this->response($message, REST_Controller::HTTP_CREATED); }`

3. Stateless Communication

Pada lingkungan REST, seperti halnya pada dunia *World Wide Web* (WWW), *stateless* berarti tidak ada *client session data* yang disimpan pada server. Server hanya menyimpan dan mengelola *state* dari resource yang digunakan. Jika terdapat kebutuhan informasi *session-specific*, hal tersebut seharusnya diatur pada sisi *client*. Karakteristik tersebut memberikan kemudahan dari sisi *scalability* suatu server yang berbasis *cluster*. Untuk mengembangkan ukuran *cluster*, yang

perlu dilakukan hanyalah menambah mesin baru serta tidak perlu memikirkan kepemilikan data terhadap *client* tertentu.

Apabila diilustrasikan dengan sebuah gambar sederhana dari sudut pandang *client* dan *server*, RESTful *Web Service* dapat digambarkan seperti berikut:



Gambar 2.3 Skema Client-Server REST Web Service
Sumber : (unpar.ac.id)

2.1.3 Komunikasi Data

JSON adalah singkatan dari *JavaScript Object Notation* yang merupakan objek asli bawaan *JavaScript* (json.org, 2014). JSON didesain sebagai format pesan pertukaran yang human *readable* serta mudah dibaca (*parsing*) oleh program komputer. Pada aplikasi berbasis *JavaScript*, penggunaan JSON sebagai format pesan pertukaran akan membawa dampak performa yang cukup signifikan dibandingkan bila menggunakan XML, karena penggunaan XML melibatkan *library* tambahan untuk membaca data dari *Document Object Model* (DOM) (w3.org, 2005). Berikut ini contoh penulisan sintaks dalam JSON :

```
1  {
2   "namaDepan": "Budi",
3   "namaBelakang": "Subudi",
4   "alamat": [
5        {"namaJalan": "Jl. Sudirman 15A"},
6        {"kota": "Jakarta Selatan"},
7        {"provinsi": "DKI Jakarta"},
8        {"kodePos": 11111}
9
10  ],
11  "nomerTelepon": "021 555-1234"
}
```

Gambar 2.4 Contoh Sintaks JSON
Sumber : (w3ii.com)

Format yang digunakan pada sintaks *JSON* mengadopsi dari sebuah code yang digunakan *JavaScript* untuk membuat sebuah objek. Penjelasan dari sintaks diatas merupakan data *JSON* yang sebelumnya telah ditulis menggunakan kode *JavaScript* yang menyatakan bahwa variable tersebut menggunakan *Object JSON* karena terdapat tanda kurung kurawal "{}". Di dalam tanda kurung kurawal terdapat sebuah atribut dan value, apabila dilihat dari sintaks diatas penulisannya

dipisahkan dengan tanda titik dua ":". Seperti salah satu contohnya yaitu pada atribut "namaDepan" memiliki value "Budi".

2.2 Authentication

Dalam sebuah web application dibutuhkan sebuah mekanisme yang mampu melindungi data/informasi dari pengguna yang tidak memiliki hak untuk mengaksesnya. Mekanisme authentication pada prinsipnya memiliki fungsi yang memberikan kesempatan bagi pengguna dan pemberi layanan dalam proses pengaksesan resource (Agarwal & Leonetti, 2013). Pihak pengguna harus bisa memberikan informasi valid yang dibutuhkan oleh pemberi layanan untuk bisa mendapatkan resource. Sedangkan pihak pemberi layanan harus bisa menjamin bahwa pihak yang tidak memiliki hak akses tidak akan dapat mengakses resource tersebut.

2.3 OAuth (*Open Authentication*)

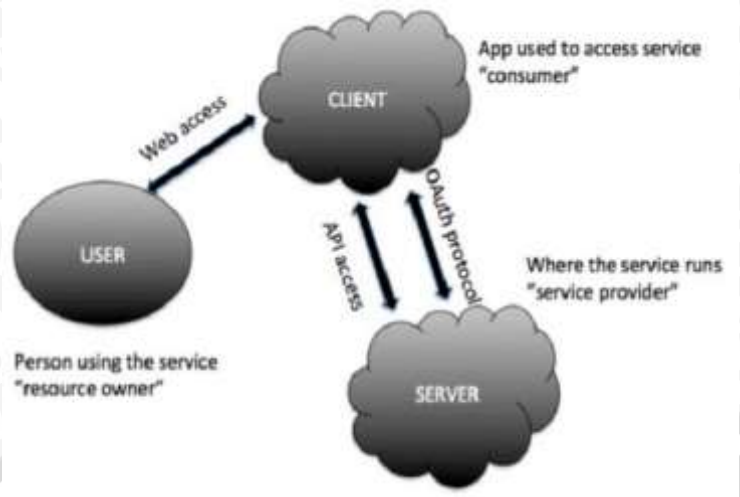
2.3.1 Pengenalan OAuth

OAuth (*Open Authentication*) adalah protokol otentikasi yang memungkinkan pengguna untuk dapat mengakses aplikasi tanpa perlu berbagi identitas mereka (github.com, 2012). Pemilik aplikasi mengintegrasikan *credential* yang dimiliki oleh pengguna dengan teknologi otentikasi yang dimiliki oleh pengembang API (*Application Programming Interface*). OAuth juga akan mengizinkan otentikasi API yang telah di proteksi baik berasal dari desktop maupun aplikasi web dengan menggunakan metode sederhana dan standard. Mengatur aliran data antar aplikasi dan digunakan pada saat pengembang API tersebut ingin mengetahui siapa saja yang sedang terlibat dan melakukan komunikasi dalam sistem (developers.google.com, 2016).

2.3.1.1 *OAuth 1.0*

OAuth 1.0 atau biasa dikenal dengan sebutan RFC 5849 yang diluncurkan pada 4 Desember 2007, lalu dilakukan revisi pada tanggal 24 Juni 2009, dan pada akhirnya diselesaikan pada bulan April 2010 yang memberikan banyak pengaruh pada perkembangan terhadap keamanan web API tanpa harus memberikan *username* dan *password* dari pengguna. Adapun pencipta dan orang yang pertama kali mengenali dari otentikasi OAuth berbasis API ini adalah E. Hammer-Lahav, Ed (E. Hammer-Lahav, 2010).

OAuth 1.0 bekerja dengan melibatkan dari 3 peran yang terdiri dari: *client*, *server*, dan *resource owner*. Ketiga peran tersebut akan terlibat dalam setiap alur kerja OAuth. Versi asli dari spesifikasi yang digunakan yang berbeda istilah untuk peran ini : konsumen (*client*), penyedia layanan (*server*), dan user (*resource owner*). 3 peran yang terlibat aktif ini biasa dikenal dengan sebutan *3-Legged*. Apabila digambarkan dalam sebuah arsitektur akan terlihat seperti berikut ini. Pada gambar 2.4 merupakan conto keterlibatan atas 3 peran aktif pada lalu lintas protokol OAuth 1.0 (Brail & Ramji, 2012).



Gambar 2.5 Skema Alur Transaksi OAuth 1.0
Sumber : (Brail & Ramji, 2012)

2.3.1.2 *OAuth 2.0*

OAuth 2.0 adalah sebuah hasil perkembangan dari protokol OAuth yang pada awalnya dihadirkan pada akhir tahun 2006. Untuk OAuth 2.0 lebih terfokus untuk memberikan kemudahan kepada *client* sebagai pemilik dan pengembang aplikasi dengan memberikan otorisasi khusus di berbagai aplikasi. Pengembangan OAuth berada dalam kekuasaan IETF OAuth WG dan didasarkan pada usulan *Web Resource Authorization Protocol* OAuth. WRAP (*Web Resource Authorization Protocol*) merupakan profil dari OAuth serta memiliki beberapa kemampuan penting yang tidak terdapat pada versi OAuth 1.0. Spesifikasi terbaru dari OAuth disumbangkan kepada IETS OAuth WG serta menjadi dasar dari terciptanya OAuth versi 2.0 (Kaur & Aggarwal, 2013).

Mekanisme kerja OAuth 2.0 memiliki 4 peran utama, seperti yang dijelaskan dibawah ini (Hardt, 2012):

1. *Resource Server*

Resource server (Sumber Daya Server) adalah sumber daya yang dimanfaatkan oleh pengguna yang memiliki API (*Application Programming Interface*) dan dilindungi oleh OAuth. *Resource server* merupakan penyedia API yang memegang dan memiliki kekuasaan pengaturan data seperti data pribadi, foto, video, kontak, atau kalender.

2. *Resource Owner*

Memposisikan dirinya sebagai pemilik dari sumber daya (*Resource Owner*), yang merupakan pemilik dari aplikasi. Pemilik sumber daya mempunyai kemampuan untuk dapat mengakses sumber daya server dengan aplikasi yang sudah ada.

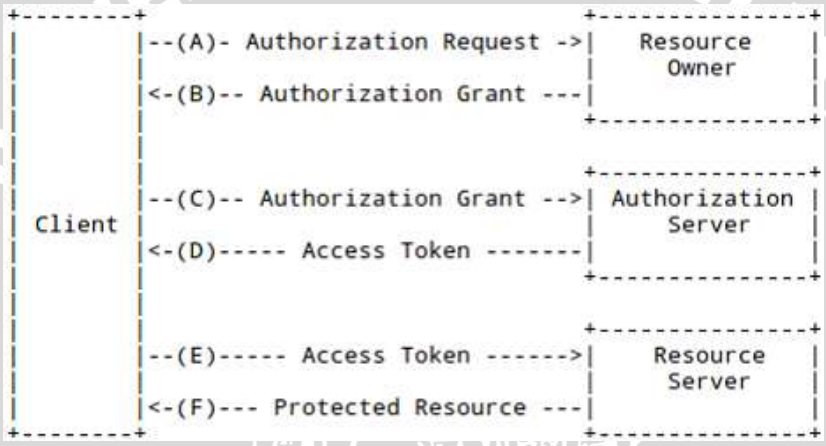
3. *Client*

Merupakan aplikasi yang melakukan permintaan API pada *Resource Server* yang telah diproteksi untuk kepentingan pemilik *Resource Owner* dengan melakukan otorisasi.

4. *Authorization Server*

Authorization Server atau bisa disebut dengan Otorisasi Server mendapat persetujuan dari pemilik sumber daya (*Resource Owner*) dengan melakukan dan memberikan *code accses token* kepada *client* untuk mengakses sumber daya yang diproteksi yang sudah tersedia pada *Resource Server*.

Mekanisme kerja OAuth 2.0 terbentuk dari 4 bagian dengan peran aktif yang terdiri dari : *Client*, *Resource Owner*, *Authorization Server*, *Resource Server* (Kaur & Aggarwal, 2013). Gambar 2.5 menunjukkan mekanisme kinerja OAuth 2.0 (Hardt, 2012).



Gambar 2.6 Mekanisme Alur Kerja OAuth 2.0
Sumber : (Hardt, 2012)

Sedangkan pada tabel 2.2 menunjukkan keterangan dari tugas dan fungsi 4 komponen OAuth yang telah dijelaskan diatas (Hardt, 2012).

Tabel 2.2 Tugas dan Fungsi 4 Komponen OAuth 2.0

	Keterangan
A	*Client* melakukan permintaan untuk untuk proses otorisasi dari *Resource Owner*. Permintaan otorisasi tersebut dapat dilakukan langsung menuju *Resource Owner*, atau jika tidak langsung melalui perantara *Authorization Server*.
B	*Client* mendapatkan persetujuan otorisasi yang merupakan *credential* mewakili otorisasi kepemilikan *client*. Pemberian otorisasi ini bergantung pada metode yang digunakan oleh *client* dan jenis yang didukung oleh *Authorization Server*.

C	*Client* melakukan permintaan *authorization code* dengan melewati proses otentikasi terlebih dahulu kepada *Authorization Server*, *client* akan mendapatkan hibah dalam bentuk otorisasi dari *Authorization Server*.
D	Otorisasi Server (*Authorization Server*) melakukan otentikasi terhadap *client* dan melakukan validasi dengan memberikan otorisasi kepada *client*, jika sesuai dan berlaku, otorisasi server akan membagikan *authorization code*.
E	*Client* melakukan permintaan sumber daya yang telah diproteksi dari *Resource Server*, melakukan tindakan otentikasi dengan menghadirkan *authorization code*.
F	*Resource Server* memvalidasi *authorization code*, jika valid dan sesuai, maka akan dilayani permintaan *client* untuk menggunakan *resource* yang sudah terlindungi.

2.3.2 *Authorization Grant* (Hibah Otorisasi)

Authorization Grant (Hibah Otorisasi) adalah mandat yang mewakili otorisasi sumber daya pemilik (*Resouce Owner*) untuk dapat mengakses sumber daya yang telah diproteksi ketika digunakan oleh *client* untuk mendapatkan *code access token*. Ada beberapa metode hibah otorisasi : *authorization code*, *implicit*, *resource owner password credential*, *client credential* (Kaur & Aggarwal, 2013).

2.3.2.1 *Authorization Code*

Kode otorisasi (*Authorization Code*) diperoleh ketika menggunakan otorisasi server sebagai perantara antara *client* dengan *Resource Owner*. *Client* melakukan permintaan otorisasi langsung dari *Resource Owner*, diarahkan menuju ke otorisasi server melalui *user-agent* (*web browser*) yang nanti pada gilirannya akan mengarahkan pemilik sumber daya ke *client* dengan kode otorisasi (Hardt, 2012).

2.3.2.2 *Implicit*

Hibah otorisasi implicit merupakan hibah otorisasi yang sudah disederhanakan dan dioptimalkan untuk *client* yang diimplementasikan dalam browser dengan menggunakan bahasa *scripting javascript*. Dalam alur otorisasi implisit, *client* diberikan langsung *code access token* sebagai hasil dari proses otorisasi pemilik sumber daya (*Resource Owner*) (Hardt, 2012).

2.3.2.3 *Resource Owner Password Credential*

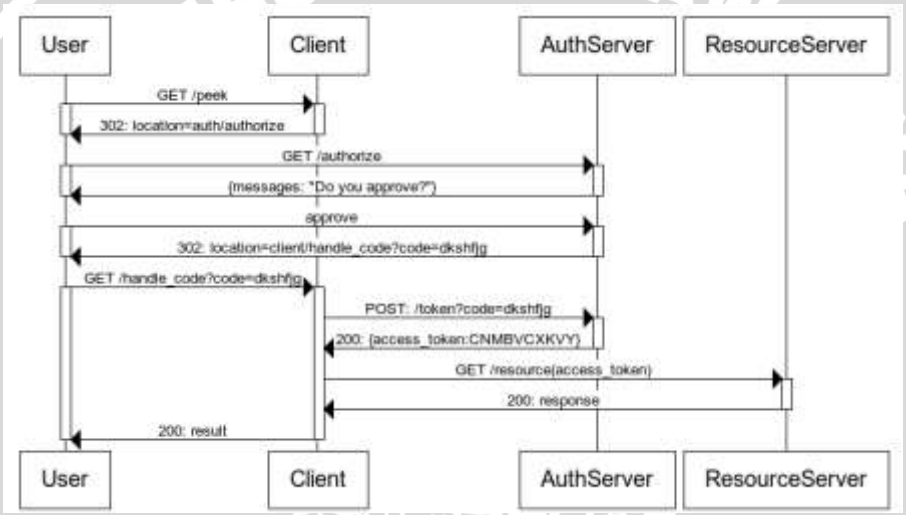
Sumber daya pemilik password credential seperti (*username* dan *password*) dapat digunakan langsung sebagai hibah otorisasi untuk mendapatkan *code access token*. *Credential* yang digunakan berasal atas kepercayaan yang tinggi

antara pemilik sumber daya dan *client*. Jenis hibah ini perlu melakukan akses *client* langsung ke *resource owner credential* yang digunakan untuk melakukan sebuah permintaan dan ditukar dengan *code access token* (Hardt, 2012).

2.3.2.4 *Client Credential*

Client Credential yaitu bentuk lain dari proses otentikasi *client* yang digunakan sebagai hibah otorisasi pada sumber daya yang di proteksi di bawah pengawasan dan pengaturan *client*. Atau ke sumber daya terproteksi yang sebelumnya sudah di atur dengan otorisasi server. *Client credential* digunakan sebagai hibah otorisasi saat *client* bertindak sebagai pemilik sumber daya (*Resource Owner*) atau sebagai peminta akses ke sumber daya yang terproteksi berdasarkan otorisasi yang telah diatur dengan otorisasi server (Hardt, 2012).

2.3.3 Proses *Authentiation*



Gambar 2.7 Diagram Alur Authentication
Sumber : (pivotal.io)

Berdasarkan diagram diatas dapat dijelaskan bahwa *authorization server* melakukan otentikasi pengguna dengan cara apapun yang diperlukan. *Client* dimulai dari proses otorisasi dan memperoleh persetujuan dari *authorization server* untuk bertindak atas nama pengguna. Persetujuan pengguna sangat diperlukan untuk *client* bisa mendapatkan hak akses ke *resource server*.

Pada akhir dapri proses otorisasi, apabila proses dinyatakan berhasil, *authorization server* akan mengeluarkan *authorization code* (kode otorisasi). Lalu *client* akan menjadikan kode tersebut untuk menjadi kunci dalam mengakses *resource server*.

2.3.3.1 *Web Server Application*

Aplikasi web yang berjalan pada sisi server adalah jenis paling umum yang dikembangkan oleh pemilik aplikasi (*Resource Owner*) yang didukung penuh pada OAuth Server. Aplikasi web pada bahasa pemrograman *server-side* seperti php, asp.net, jsp melakukan proses kerjanya pada sisi server yang tidak dipublikasikan

untuk umum (Brail & Ramji, 2012). Proses yang terjadi dan berlaku saat seorang pengguna melakukan kegiatan otentikasi dengan menekan tombol "*Sign-In*" adalah pengguna akan menerima rangkaian *link* pada alamat browser seperti pada gambar 2.8 (Parecki, 2012):

```
http://oauth2server.com/auth?response_type=code&client_
id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos
```

Gambar 2.8 Request Authentication Web Server Application
Sumber : (Parecki, 2012)

Beberapa parameter *query* yang terdapat pada rangkaian *link* alamat *browser* adalah sebagai berikut (Byod, 2012):

1. *Client_ID*

Nilai yang diberikan saat mendaftarkan suatu aplikasi web yang dimiliki oleh *Resource Owner*.

2. *Redirect_Uri*

Lokasi kemana pengguna harus kembali setelah menyetujui akses untuk aplikasi.

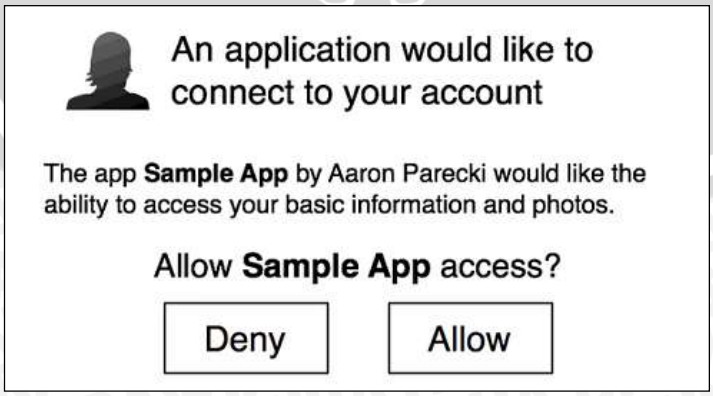
3. *Scope*

Data aplikasi meminta akses menuju dan merujuk pada *web developer* penyedia OAuth. Jika berasal dari google API berarti merujuk ke halaman auth google API contoh : *https://www.googleapis.com/auth/tasks.*

4. *Response_Type*

Adalah kode *server side web application flow* yang mengindetifikasikan sebuah kode otorisasi yang akan dikembalikan lagi ke aplikasi setelah user menerima dan menyetujui permintaan otorisasi.

Pada sisi halaman browser, pengguna akan di berikan suatu halaman pemberian izin hibah, untuk disetujui oleh pengguna agar dapat dilakukan otorisasi oleh OAuth Server (*Authorization Server*). Gambar 2.9 memperlihatkan salah satu contoh permintaan persetujuan dari pengguna (Parecki, 2012).



Gambar 2.9 Authorization Confirm Web Server Application
Sumber : (Parecki, 2012)

Jika pengguna melakukan proses tindakan "*Allow*", maka sistem yang bekerja akan mengarahkan kembali pada aplikasi web dengan sebuah *Auth Code*. *Auth Code* untuk suatu aplikasi web memiliki kode *unique*. Bagi setiap aplikasi jika didaftarkan pada salah satu penerbit OAuth seperti Google akan mendapatkan *auth code* tertentu. Pada gambar 2.10 memperlihatkan contoh mendapatkan *auth code*.

```
http://oauth2client.com/cb?code=AUTH_CODE_HERE
```

Gambar 2.10 Proses Mendapatkan Auth Code
Sumber : (Parecki, 2012)

Berdasarkan yang telah di hasilkan pada gambar-gambar diatas dapat dijelaskan untuk tiap komponennya sebagai berikut (Byod, 2012):

1. *Client_ID*

   Nilai id yang diberikan saat mendaftarkan suatu aplikasi.

2. *Client_Secret*

   Sebuah parameter yang memiliki sifat rahasia diberikan saat mendaftar aplikasi.

3. *Grant_Type*

   Nilai suatu *authorization_code*, mengidentifikasikan penukaran sebuah kode otorisasi pada suatu *access token*.

4. *Code*

   Authorization code yang nantinya akan dikirimkan kepada aplikasi *client*.

5. *Redirect_Uri*

   Lokasi yang telah didaftarkan sebelumnya dan digunakan pada permintaan awal menuju otorisasi.

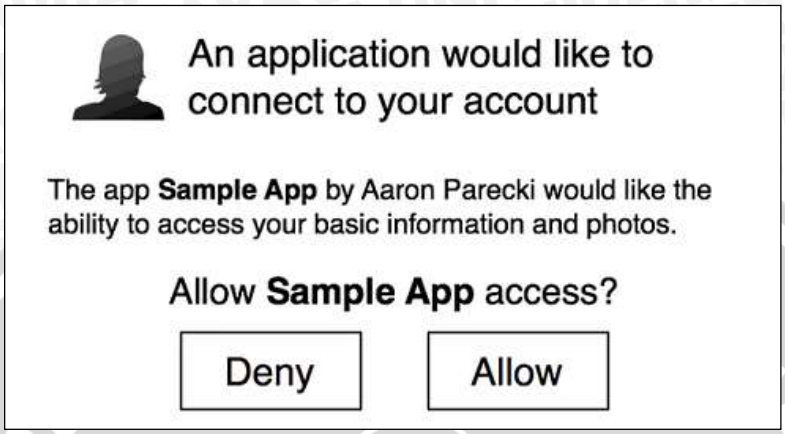
2.3.3.2 *Browser Based Application*

Aplikasi *browser* dijalankan sepenuhnya dalam *browser* setelah memuat suatu kode sumber dari sebuah situs halaman web. Karena seluruh kode sumber tersedia untuk dan pada *browser*, sehingga kerahasiaan dari rahasia *client* tidak digunakan pada aplikasi ini (Parecki, 2012). Proses yang tebentuk ketika seorang pengguna melakukan kegiatan otentikasi dengan menekan perintah tombol "Sign In" yang sudah disediakan pada aplikasi *browser*, seperti yang ditunjukkan pada gambar 2.11 di bawah ini :

```
http://oauth2server.com/auth?response_type=token&clie
nt_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photo
s
```

Gambar 2.11 Request Authentication Browser Based Application
Sumber : (Parecki, 2012)

Kemudian pengguna akan diarahkan pada suatu halaman pemberian izin hibah, agar pengguna dapat melakukan pemberian izin hibah yang akan di proses kembali oleh otorisasi server (*authorization server*). Gambar 2.12 berikut menunjukkan halaman *browser* persetujuan pemberian hibah oleh pengguna (Parecki, 2012).



Gambar 2.12 Authorization Confirm Browser Based Application
Sumber : (Parecki, 2012)

Jika pengguna melakukan tindakan dengan menekan perintah tombol "*Allow*", layar browser akan diarahkan atau *direct* menuju aplikasi dengan menghadirkan sebuah *Access token*. Pada gambar 2.13 adalah proses yang terdapat pada alamat HTTP (*Hypertext Transfer Protocol*) saat pengguna menekan tombol "*Allow*" dengan menampilkan *Access token*.

```
http://oauth2client.com/cb#token=ACCESS_TOKEN
```

Gambar 2.13 Menampilkan Access Token
Sumber : (Parecki, 2012)

2.3.3.3 *Native Application*

Native application merupakan sebuah aplikasi yang diinstal dan dijalankan pada perangkat yang digunakan langsung oleh pengguna, salah satunya pada perangkat *mobile* yang hampir memiliki kesamaan seperti *browser based application*, aplikasi *mobile* juga tidak bisa menjaga kerahasiaan rahasia *client* mereka. Karena itu, aplikasi *mobile* juga diharuskan untuk menggunakan aliran OAuth yang tidak memerlukan rahasia *client* (Parecki, 2012).

Proses yang terbentuk ketika seorang pengguna melakukan kegiatan otentikasi dengan menekan perintah "Sign In" yang tersedia pada aplikasi *mobile* akan mengirim pengguna ke salah satu aplikasi asli dari layanan atau halaman web mobile. Misal pada aplikasi yang menggunakan *platform* android, aplikasi dapat mendaftarkan pola pencocokan URL yang akan meluncurkan aplikasi asli jika URL cocok pola akan dikunjungi (Parecki, 2012). Pada gambar 2.14 akan menunjukan URL untuk otorisasi aplikasi.

```
http://facebook.com/dialog/oauth?response_type=token&cl
ient_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=email
```

Gambar 2.14 Request Authentication Mobile Applicatin

Sumber : (Parecki, 2012)

Setelah itu pengguna akan diarahkan pada suatu halaman untuk melakukan pemberian izin, agar pengguna dapat melakukan pemberian izin yang akan di proses kembali oleh otorisasi server (*authorization server*). Gambar 2.15 berikut menunjukkan halaman *browser* persetujuan pemberian hibah oleh pengguna (Parecki, 2012).

Gambar 2.15 Authorization Confrims Mobile Application

Sumber : (Parecki, 2012)

Jika pengguna melakukan tindakan dengan menekan perintah "*Okay*", tampilan layar akan *direct* atau diarahkan menuju aplikasi dengan menghadirkan sebuah *authorizarion code sehingga a*plikasi *mobile* dapat menggunakan *code access* dari URI dan mulai menggunakannya untuk membuat permintaan API.

BAB 3 METODOLOGI

Pada bab ini dibahas mengenai langkah-langkah dalam melakukan penelitian. Penelitian ini merupakan tipe penelitian implementatif-pengembangan lanjut. Penelitian ini dimulai dari tahapan Studi Literatur yaitu mencari literatur untuk memahami permasalahan. Kemudian dilanjutkan dengan mempelajari arsitektur sistem yang sudah dikembangkan sebelumnya. Setelah itu melakukan implementasi dengan menggunakan metode yang telah diusulkan untuk dapat dilihat hasilnya melalui pengujian dan analisa. Jika sudah sesuai, maka diambil kesimpulan serta saran dari hasil penelitian terrsebut. Diagram alur tahapan dalam proses pengerjaan penelitian ini ditunjukan pada gambar 3.1 sebagai berikut :

Gambar 3.1 Diagram Alur Penelitian

3.1 Studi Literatur

Tahapan penelitian ini membutuhkan studi literatur untuk merealisasikan implementasi proses *authentication* pada *web service* situs halaman *website* belajardisini.com dengan menggunakan metode OAuth (*Open Authentication*). Informasi dan pustaka yang memiliki keterkaitan dengan penelitian ini didapat dari jurnal, buku, situs internet, dosen pembimbing, dan rekan-rekan mahasiswa. Adapun teori-teori yang dipelajari tentang :

1. Komunikasi *web service*
   - Definisi *web service*
   - REST *web service*
   - Komunikasi data
2. *Authentication*
3. OAuth (*Open Authentication*)
   - Pengenalan OAuth
     - OAuth 1.0
     - OAuth 2.0
   - *Authorization Grant* (Hibah Otorisasi)
     - *Authorization code*
     - *Implicit*
     - *Resource owner password credential*
     - *Client credential*
   - Proses *Authentication*
     - *Web server application*
     - *Browser based application*
     - *Native application*

3.2 Perancangan

Pada tahap perancangan dilakukan ketika penulis telah melakukan analisa sistem karena perancangan untuk mengembangkan sistem nantinya akan diikuti dari arsitektur awal sistem yang sudah dikembangkan sebelumnya. Pada tahap perancangan ini penulis terlebih dahulu akan memulai dengan menggambarkan secara umum arsitektur awal sistem dan setelah itu dilakukan perancangan arsitektur sistem yang akan dikembangkan dengan menerapkan mekanisme *authentication* yaitu menggunakan teknologi OAuth.

3.3 Implementasi

Tahap implementasi disini maksudnya adalah mulai mengimplementasikan metode yang sudah ditawarkan oleh penulis pada beberapa pembahasan sebelumnya yaitu OAuth (*Open Authentication*) untuk melengkapi mekanisme *authentication* pada *web service* dari situs belajardisini.com.

3.3.1 Lingkungan Penelitian

Ruang lingkup dari penelitian ini yaitu dilakukan dengan mekanisme simulasi dengan penggunaan server yang dijalankan pada komputer peneliti dengan memanfaatkan *ip public* yang didapat dari *virtual machine* dan *ip* tersebut akan dijadikan domain aplikasi sehingga ketika aplikasi *client* akan mengakses ke server juga akan menggunakan *ip* yang sama untuk dijadikan domain pada saat akan melakukan *request*. Apabila dilihat secara umum, untuk tahap implementasi diawali dengan melakukan implementasi *authorization server* untuk

menambahkan proses *authentication* dengan menerapkan metode OAuth (*Open Authentication*) yang akan diimplementasikan pada sisi server dari sistem yang akan dikembangkan. *Authorization server* nantinya akan memeriksa kebenaran dari *credential* yang dikirimkan oleh *client*. Pada proses implementasi dari sisi server juga ditambahkan fitur untuk membuat token (*token generator*) yang nantinya *token* tersebut akan digunakan untuk meminta layanan yang dilakukan oleh *client*. *Token* yang dihasilkan pada proses tersebut akan langsung masuk kedalam *database* dalam bentuk *string* yang pada tiap proses *generate* tidak akan terdapat *token* yang serupa. Untuk implementasi teknologi OAuth (*Open Authentication*) dari sisi *client* dilakukan implementasi langsung didalam sebuah aplikasi sederhana berbasis *mobile android* yang dijalankan pada sebuah emulator dengan fitur *sign-in* yang akan dikembangkan dengan memanfaatkan *token* yang sudah didapat dari proses *generate token* sebelumnya.

3.4 Pengujian

Tahap pengujian dilakukan setelah proses implementasi sudah di selesaikan. Pengujian pada penelitian ini membahas tentang pengujian dari penerapan metode *OAuth* dalam proses *authentication* yang telah diterapkan pada *web service* dari situs halaman *website* belajardisini.com. Pengujian yang dilakukan bertujuan untuk menguji kinerja sistem setelah dilakukan implementasi dengan menerapkan metode OAuth dan langsung dijalankan dengan menggunakan perangkat *smartphone* android. Parameter kinerja yang menjadi tolak ukur dalam pengujian yaitu sistem OAuth mampu mengelola *credential* yang diberikan oleh *client* untuk bisa ditukarkan menjadi hak akses ketika *client* akan mengakses layanan *web service* melalui persetujuan pengguna. Skenario pengujian akan dilakukan dengan beberapa cara, sehingga berdasarkan pengujian sistem yang telah dilakukan akan dapat diambil kesimpulan.

3.7.1 Pengujian Fungsional

Skenario pada peengujian fungsional dilakukan dengan menggunakan pengujian *black box* yang memiliki tujuan untuk mengetahui kesesuaian hasil berdasarkan kebutuhuan dari penelitian yang dilakukan.

3.7.2 Pengujian HTTP

Pada skenario pengujian ini akan dilakukan dengan menjalankan proses request yang menggunakan protokol HTTP dengan tujuan untuk mengetahui bagaimana hasil komunikasi yang terjadi dari sistem yang telah dikembangkan ketika dijalankan pada request dari protokol HTTP tersebut.

3.7.3 Pengujian Keamanan

Pada tahap pengujian keamanan ini akan dilakukan dengan menggunakan 2 skenario pengujian yaitu dengan melakukan penggunaan kembali *credential client* dan *decompile* APK.

1. Untuk skenario pengujian penggunaan kembali *credential client* akan dilakukan dengan melakukan *capture* proses komunikasi yang terjadi pada *protocol* HTTP yang digunakan dalam sistem yang dikembangkan dengan memanfaatkan *tools wireshark*.
2. Setelah aplikasi *client* dijadikan APK, maka akan dilakukan simulasi dengan melakukan *decompile* APK untuk dapat diketahui apakah *credential client* yang didefinisikan oleh *client* pada kode program masih dapat terlihat atau tidak.

3.5 Penarikan Kesimpulan dan Saran.

Pengambilan kesimpulan dan saran menjadi tahapan terakhir dalam penelitian skripsi ini. Kesimpulan ini dapat diambil ketika semua kebutuhan yang sudah didefinisikan sudah terpenuhi oleh sistem. Kesimpulan ini juga merupakan jawaban dari beberapa rumusan masalah yang telah didefinisikan sebelumnya. Adapun saran merupakan ide atau pikiran yang diberikan oleh pengguna atau *stakeholder* untuk pengembangan sistem selanjutnya.

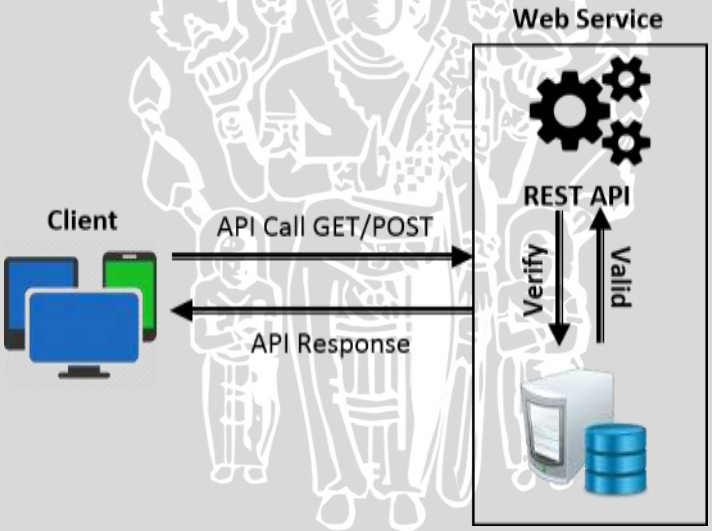
BAB 4 PERANCANGAN SISTEM

Pada bab 4 ini membahas terkait arsitektur sistem, gambaran umum sistem serta perancangan sistem yang dibangun.

4.1 Arsitektur Awal Sistem

Dengan keberadaan API saat ini dapat mengembangkan aplikasi yang baru dengan memanfaatkan sumberdaya yang sudah ada tanpa harus membangun program dari awal. Belajardisini.com merupakan salah satu situs yang menjadi penyedia API sehingga memungkinkan pengguna lain untuk dapat membuat aplikasi sendiri tanpa harus dipusingkan dengan bagaimana proses didalamnya.

Belakangan ini REST sering dibutuhkan, seiring dengan berkembangnya pemrograman *smartphone* yang memanfaatkan data/informasi yang ada pada sebuah server. Begitu juga pada aplikasi web dengan situs belajardisini.com yang sudah menerapkan *web service engine* dengan menggunakan REST. REST API yang sudah dikembangkan pada situs belajardisini.com menggunakan *framework CodeIgniter*. Berikut adalah sebuah analisa gambar arsitektur awal sistem dari belajardisini.com :

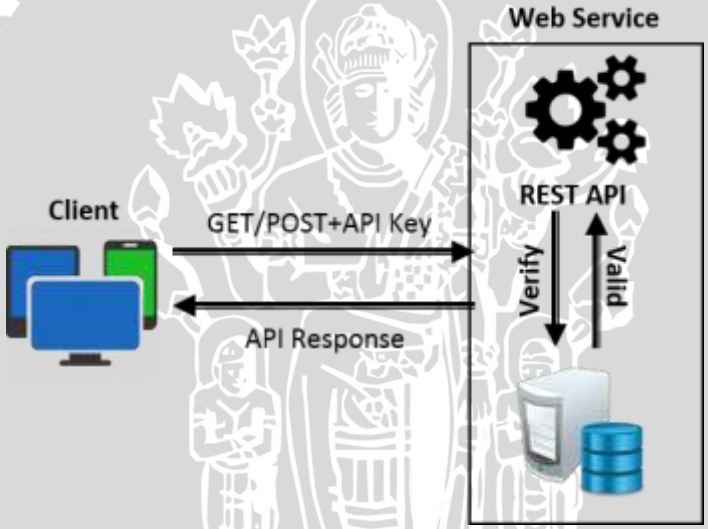


Gambar 4.1 Arsitektur Awal Arsitektur Sistem

Pada gambar 4.1 diatas merupakan gambaran umum sistem awal yang menjelaskan tentang proses pemanggilan API pada *web service* belajardisini.com, dimana pada proses pemanggilannya dilakukan secara langsung tanpa adanya mekanisme pengenalan *client* atau biasa disebut mekanisme *authentication*. Oleh karena itu *client* hanya tinggal memanggil method apa saja yang perlu dipanggil dan disertakan pada saat melakukan *request* API.

4.2 Pengembangan Sistem Dengan OAuth 2.0

Untuk dapat memanfaatkan layanan dan mengakses API dari belajardisini.com, umumnya peminta layanan harus memiliki API *key* terlebih dahulu yang merupakan sebuah kode unik dengan sebuah kombinasi dan biasanya dibentuk secara otomatis oleh program tertentu (tidak dapat menentukan sendiri). Pada situs belajardisini.com belum menerapkan proses *generate* API *key* untuk peminta layanan yang akan memanfaatkan layanannya, sehingga pengguna tidak bisa masuk ke pintu layanan dimana API tersebut disediakan. Fungsi dari API *key* ini sendiri nantinya akan digunakan ketika *client third party* sudah mendapatkan hak aksesnya atau dengan kata lain sudah terotorisasi maka akan disertakan juga API *key* tersebut untuk melengkapi proses *request* yang akan dilakukan oleh client. Sehingga penulis telah mencoba untuk menambahkan proses dimana agar *client* terlebih dahulu meminta API *key* untuk dapat melanjutkan ke proses berikutnya yaitu menggunakan layanan yang tersedia pada situs belajardisini.com. Berikut adalah arsitektur sitem belajardisini.com setelah ditambahkan *request* dengan menggunakan API *key* :

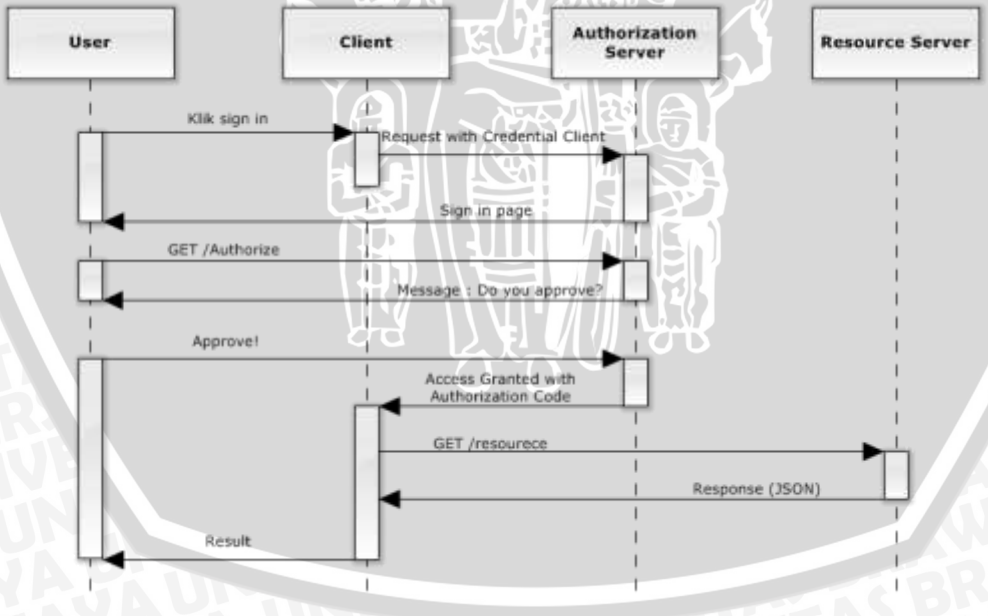


Gambar 4.2 Arsitektur Sistem Dengan API Key

Dengan menggunakan REST, komunikasi yang dibangun bisa menjadi mudah antara penyedia dan peminta sumber daya. Namun, sumber daya ada yang tidak terproteksi dan ada yang sudah terproteksi sehingga untuk melakukan akses ke sumber daya (*resources*) yang terproteksi diperlukan *authentication* untuk mengenali peminta layanan. Pada umumnya variabel *authentication* (*credentials*) yang digunakan adalah *username* dan *password*. Untuk itulah diciptakan OAuth, yang menggunakan *grant access* sebagai alat pengganti *username* dan *password* sehingga aksesnya pun bisa diberikan batas atau bahkan aksesnya bisa di hapus kapanpun.

OAuth (*Open Authentication*) merupakan *authorization framework* yang memungkinkan aplikasi pihak ketiga untuk memperoleh *limited access* ke http service, OAuth ini bertindak sebagai perantara interaksi *approval* antara pemilik sumber daya dan layanan http. Dengan OAuth, aplikasi pihak ketiga tidak perlu melemparkan *username* dan *password* untuk mendapatkan akses ke aplikasi pemilik *resource*. Jadi, fungsi OAuth pada intinya ialah memberikan izin akses kepada aplikasi pihak ketiga untuk mengakses informasi yang terdapat di penyedia layanan tanpa harus membagi keseluruhan data.

Dengan melihat kondisi teknologi saat ini berdasarkan yang sudah dibahas sebelumnya, penulis telah melakukan percobaan untuk menerapkan proses keamanan layanan dengan memanfaatkan metode OAuth (*Open Authentication*) agar penyedia layanan lebih aman dengan adanya proses *authentication* sebelum peminta akses mulai menggunakan layanan yang tersedia. Pada pengembangannya, tipe dari *client* yang digunakan dalam penelitian ini mengadopsi dari *native based app* yang dijalankan pada perangkat yang digunakan oleh pengguna. Proses otentikasi dan otorisasinya semua berjalan pada sisi server dan *client* hanya melakukan permintaan untuk proses otentikasi dan otorisasi yang ketika *credential client* sudah dianggap valid oleh *authorization server* barulah diarahkan pada sebuah halaman *authorize app* yang telah disediakan oleh pemilik layanan untuk melakukan proses otoriasi dan mendapatkan akses. Berikut adalah gambar arsitektur baru sistem setelah di implementasikan OAuth pada *web service* belajardisini.com :



Gambar 4.3 Alur Sistem Dengan OAuth

4.3 Alur Kerja Teknologi OAuth 2.0 Pada Sistem

Teknologi *authentication* yang diterapkan pada prosedur mekanisme *authentication* "sign-in" setiap *client* menggunakan teknologi OAuth 2.0. Pada gambar sebelumnya yaitu gambar 4.3 di atas adalah alur kerja mekanisme *authentication* OAuth 2.0 yang diilustrasikan berjalan pada aplikasi yang sudah diterapkan menggunakan mekanisme *authentication* dengan metode OAuth 2.0 seperti aplikasi yang ada pada situs belajardisini.com.

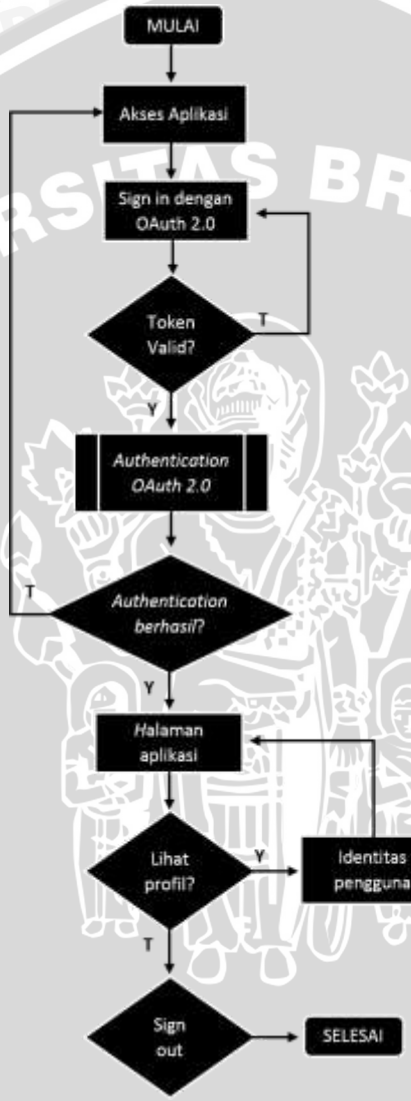
Tabel 4.1 Keterangan Alur Sistem OAuth 2.0

Proses	Keterangan
A	*Client* yang mengakses aplikasi milik *Resource Owner* melalui perantara internet menuju *authorization server* untuk melakukan "*sign-in*" pada aplikasi.
B	*Authorization Server* (Server Otorisasi) melakukan otentikasi *credential* kepemilikan *client*, memberikan persetujuan apakah pemilik sumber daya memberi atau menolak permintaan *client* untuk bisa mengakses layanan.
C	Dengan asumsi *client* mendapatkan kuasa melakukan akses, server otorisasi (*authorization server*) menampilkan halaman *sign-in*.
D	Apabila *client* yang sudah menerima kode otorisasi (*authorization code*) sebelumnya, maka akan mendapatkan perintah yang di terima berupa *authorize app*, persetujuan melakukan otorisasi yang di kirim ke *authorization server*.
E	Jika *client* melakukan otorisasi, *authorization code* akan dihadirkan bersamaan dengan dihantarkannya *client* pada halaman selanjutnya dari aplikasi.
F	Aplikasi *client* sudah berhasil mendapatkan hak akses dan sudah bisa masuk untuk mengakses layanan yang ada.

4.4 *Flowchart* Sistem

Flowchart pada pembahasan ini menggambarkan alur kerja OAuth yang telah dikembangkan baik alur secara umum maupun alur mekanisme otentikasinya.

4.4.1 *Flowchart* Sistem Secara Umum



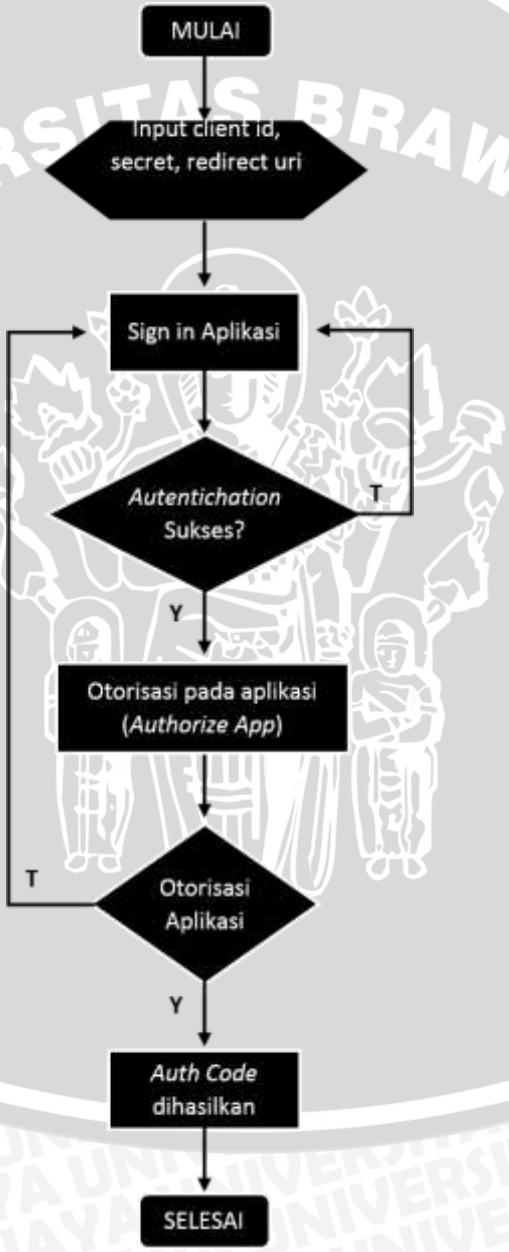
Gambar 4.4 Flowchart Sistem Secara Umum

Prosedur *flowchart* tersebut merupakan gambaran sistem secara umum. Sistem diatas memberikan petunjuk bahwa ada proses *authentication* OAuth yang perlu di lalui oleh pengguna untuk dapat masuk dan menerima akses layanan yang tersedia pada aplikasi. Di bawah ini adalah prosedur kerja dari *flowchart* di atas :

1. Mulai.
2. Masuk kehalaman awal aplikasi.
3. Melakukan "*sign-in*" pada sistem aplikasi.

4. Data *credential client* yang menyertakan *client id*, *client secret*, *redirect uri* dan *api_key* pada saat melakukan *load url* akan di proses menggunakan teknologi *authentication* OAuth 2.0.
5. Di asumsikan data *client* sudah sesuai, *client* dapat menerima layanan yang tersedia pada sistem aplikasi.
6. *Client* dapat memilih akses salah satu pada aplikasi. Dengan melakukan akses lihat profil ataupun perintah "*sign-out*".
7. Selesai

4.4.2 *Flowchart Authentication* OAuth 2.0



Gambar 4.5 Flowchart Sistem Dengan OAuth 2.0

Prosedur *flowchart* tersebut di gunakan untuk memproses *credential* (mandat) yang diberikan oleh *client* apakah masih valid atau tidak, menggunakan prosedur teknologi *authentication* OAuth. Di bawah ini akan di jelaskan prosesnya secara terperinci :

1. Mulai
2. *Client* menyertakan *client id*, *client secret*, *redirect uri* dan *api_key* pada saat melakukan *load url*.
3. *Client* melakukan tindakan "*sign-in*" menuju aplikasi. Jika *client* sesuai maka proses *authentication* dilanjutkan menggunakan teknologi OAuth.
4. Proses *authentication* OAuth dilakukan untuk mengecek kesesuaian dari data *client*, Dengan melakukan otorisasi untuk menghadirkan halaman *Authorize App*.
5. Halaman persetujuan otorisasi pada halaman *Authorize App* merupakan halaman melakukan tindakan persetujuan otoritas untuk mendapatkan *authorization code*.
6. *Client* yang melakukan tindakan persetujuan akan di arahkan menuju halaman utama sistem aplikasi, ditandai dengan keluarnya *authorization code* pada layar.
7. Jika *client* tidak melakukan tindakan persetujuan maka sistem akan kembali kehalaman awal yang berarti *client* tidak mendapatkan hak akses untuk mengambil informasi yang ada pada server.
8. Selesai.

BAB 5 IMPLEMENTASI

5.1 Implementasi Sistem

Implementasi adalah tahapan penerapan di mana sistem telah selesai dirancang setelah melewati tahapan analisis dan perancangan. Dalam penelitian ini, penulis mengimplementasikan aplikasi dari sisi server yang dirancang menggunakan bahasa pemrograman PHP (*Hypertext Preprocessor*) yang bersifat *open source* beserta penggunaan basis data yang melibatkan DBMS (*Database Management System*) MySQL. Dan untuk penerapan dari sisi *client* menggunakan Bahasa pemrograman *Java* karna berjalan pada perangkat *mobile*.

Implementasi sistem berdasarkan hasil analisa sistem yang sudah dilakukan akan dibagi menjadi beberapa bagian seperti berikut :

1. Implementasi OAuth server.
2. Implementasi proses generate API key dan token OAuth.
3. Implementasi proses *authentication* OAuth 2.0 yang melibatkan *authorization server* yang bertindak mengeluarkan halaman *authorize app* (otorisasi aplikasi) kemudian dikembalikan dalam bentuk *authorization code*.

Hal ini dimaksudkan untuk mengetahui proses tahapan kerja penerapan teknologi *authentication* OAuth 2.0 yang melibatkan *authorization server* sebagai pihak yang memproses otorisasi valid atau tidaknya *credential client*.

5.1.1 Spesifikasi Perangkat Lunak dan Perangkat Keras

Perangkat lunak dan perangkat keras yang di gunakan dalam mengembangkan sistem tersebut sangat berpengaruh dan memiliki fungsi penting yang tidak bisa diabaikan. Tabel 5.1 di bawah ini menunjukkan spesifikasi dari perangkat lunak dan perangkat keras yang digunakan dalam mengembangkan sistem.

Tabel 5.1 Keterangan Spesifikasi Perangkat

No.	*Jenis Perangkat*	*Komponen*
1.	Perangkat Keras	a. Processor Intel(R) Core(TM) i5-2450M b. RAM 4.00 GB
2.	Perangkat Lunak	a. Sistem Operasi : Windows7 32-bit b. Server Software : Apache/2.4.7 (Win32) c. PHP Version 5.5.9 d. MySQL 5.0.11 e. Eclipse Versi 4.2.1 f. Browser Google Chrome

5.1.2 Implementasi OAuth Server

Untuk dapat menggunakan mekanisme *authentication* OAuth ini maka terlebih dahulu perlu dilakukan proses pendefinisian *authorization server* berdasarkan kebutuhan OAuth yang di implementasikan langsung pada sisi server dengan beberapa komponen OAuth yaitu *client_id*, *client_secret*, *redirect_uri*, *respon_type*, *scope* dan *rest_key* yang nantinya akan difungsikan bagi *client* dan dijadikan kunci untuk dapat mengakses layanan terpoteksi yang sudah dilengkapi dengan mekanisme *authentication* OAuth. Tabel dibawah adalah potongan kode program dari sisi server yang didefinisikan sebagai *authorization server*.

```
1   <?php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2   require APPPATH . '/libraries/REST_Controller.php';
3   class Oauth extends CI_Controller
4   {
5   function index()
6   {
7               //Get query string parameters
8               $params = array();
9
10              //Client id
11              if ($client_id = $this->input->get('client_id'))
12              {
13                      $params['client_id'] = trim($client_id);
14              }
15              else
16              {
17              $this->_fail('invalid_request',
18              'The request is missing a required parameter,
19              includes an invalid parameter value, or is otherwise malformed.
20              See client_id.', NULL, array(), 400);
21              }
22
23              //Client secret
24              if ($client_secret = $this->input->get('client_secret'))
25              {
26                      $params['client_secret'] = trim($client_secret);
27              }
28              else
29              {
30                      $this->_fail('invalid_request',
31                      'The request is missing a required parameter,
32                      includes an invalid parameter value,
33                      or is otherwise malformed. See client_secret.',
34                      NULL, array(), 400, 'json');
35                      return;
36              }
37
38              //rest_key_name
39              if ($rest_key = $this->input->get('X-API-KEY'))
40              {
41                      $params['rest_key'] = trim($rest_key);
42              }
43              else
```

```
44          {
45                  $this->_fail('invalid_request',
46                  'The request is missing a required parameter,
47                  includes an invalid parameter value,
48                  or is otherwise malformed. See Rest_key.',
49                  NULL, array(), 400);
50          }
51
52          //Client redirect uri
53          if ($redirect_uri = $this->input->get('redirect_uri'))
54          {
55                  $params['redirect_uri'] = trim($redirect_uri);
56          }
57          else
58          {
59                  $this->_fail('invalid_request',
60                  'The request is missing a required parameter,
61                  includes an invalid parameter value,
62                  or is otherwise malformed. See redirect_uri.',
63                  NULL, array(), 400);
64                  return;
65          }
66
67          //Validate the response type
68          if ($response_type = $this->input->get('response_type'))
69          {
70
71                  $response_type = trim($response_type);
72                  $valid_response_types = array('code');
73                  //array to allow for future expansion
74
75                  if ( ! in_array($response_type, $valid_response_types))
76                  {
77                          $this->_fail('unsupported_response_type',
78                          'The authorization server does not support
79                          obtaining an authorization code using this method.
80                          Supported response types are \'' .
81                          implode('\' or ', $valid_response_types) .
82                          '\'.', $params['redirect_uri'], array(), 400);
83                          return;
84                  }
85                  else
86                  {
87                          $params['response_type'] = $response_type;
88
89                  }
90
91
92                  }else
93                  {
94                          $this->_fail('invalid_request',
95                          'The request is missing a required parameter,
96                          includes an invalid parameter value,
97                          or is otherwise malformed. See response_type.',
```

```
98                          NULL, array(), 400);
99                  return;
100                 }
101
102         //Validate client_id and redirect_uri
103         $client_details = $this->oauth_server->validate_client($params
104         ['client_id'], $params['client_secret'], $params['redirect_uri']);
105         //returns object or FALSE
106         print_r($client_details );
107         if ($client_details === FALSE )
108         {
109                  $this->_fail('unauthorized_client',
110                 'The client is not authorized to request an authorization
111                 code using this method.', NULL, array(), 403);
112                 return;
113         }
114         else
115         {
116                 //The client is valid, save the details to the session
117                 $this->session->set_userdata('client_details', $client_details);
118
119                 //$this->session->set_userdata('user_id', trim($client_id));
120
121         }
122
123         // Get and validate the scope(s)
124         if ($scope_string = $this->input->get('scope'))
125         {
126                 $scopes = explode(',', $scope_string);
127                 $params['scope'] = $scopes;
128         }
129         else
130         {
131                 $params['scope'] = array();
132         }
133
134         //Check scopes are valid
135         if (count($params['scope']) > 0)
136         {
137                 foreach($params['scope'] as $s)
138                 {
139                         $exists = $this->oauth_server->scope_exists($s);
140                         if ( ! $exists)
141                         {
142                                 $this->_fail('invalid_scope',
143                                 'The requested scope is invalid, unknown,
144                                 or malformed. See scope \''.$s.'\'.',
145                                 NULL, array(), 400);
146                                 return;
147                         }
148                 }
149         }
150         else
151         {
```

```
152                     $this->_fail('invalid_request',
153                     'The request is missing a required parameter,
154                     includes an invalid parameter value,
155                     or is otherwise malformed. See scope.', NULL, array(), 400);
156                     return;
157             }
158
159             //Save the params in the session
160             $this->session->set_userdata(array('params' => $params));
161
162             //Redirect the user to sign in
163             redirect(site_url(array('oauth', 'sign_in')), 'location');
164     }
```

Gambar 5.1 Potongan Kode Program Authorization Server

Berikut adalah penjelasan berdasarkan potongan kode program diatas. Baris 8 merupakan variabel array yang digunakan untuk menyimpan data. Kode pada baris 11-21 berfungsi untuk menyimpan *client_id* pada variable $params ketika *client* mengirimkan *cient_id* nya. Apabila tidak ada *client_id* yang dikirimkan maka akan menampilkan pesan error seperti yang dituliskan pada baris 17.

Sama halnya dengan kode pada baris 24-36 yang berfungsi untuk menyimpan *client_secret* pada variable $params ketika *client* mengirimkan *cient_secret* nya. Apabila tidak ada *client_secret* yang dikirimkan maka akan menampilkan pesan error seperti yang dituliskan pada baris 30.

Potongan kode pada baris 39-50 juga memiliki fungsi yang sama yaitu menyimpan *rest_key* pada variable $params ketika *client* mengirimkan *rest_key* nya. Apabila tidak ada *rest_key* yang dikirimkan maka akan menampilkan pesan error seperti yang dituliskan pada baris 45.

Pada potongan kode baris 53-65 memiliki fungsi untuk menyimpan *redirect_uri* pada variable $params ketika *client* mengirimkan *redirect_uri* nya, yang dimana fungsi dari *redirect_uri* ini nantinya untuk mengarahkan *client* menuju lokasi API yang diakses ketika proses otorisasi telah berhasil. Apabila tidak ada *redirect_uri* yang dikirimkan maka akan menampilkan pesan error seperti yang dituliskan pada baris 59.

Baris 68-99 memiliki fungsi untuk menyimpan *respon_type*. Jika tidak ada kesamaan isi dari *valid_respon_type* dengan variable *response_type* maka akan menampilkan pesan error seperti pada baris 77. Apabila memiliki kesamaan maka variable *response_type* akan disimpan pada variable $params .seperti yang ditunjukan pada baris 89. Apabila tidak ada *response_type* yang dikirimkan maka akan menampilkan pesan eror seperti yang ada pada baris 93.

Baris 102-119 digunakan untuk memvalidasi *client_id*, *client_secret* dan *redirect_uri* yang dikirimkan oleh *client*. Pada baris 102-105, variable $client_details merupakan variable array yang digunakan untuk menyimpan data hasil pencarian berdasarkan parameter yang dikirimkan oleh *client* antara lain

client_id, *client_secret* dan *redirect_uri* yang nantinya akan ditampilkan. Pada baris 106 dan 108 dijelaskan apabila variable $client_details bernilai *false* maka akan ditampilkan pesan error yang menyatakan bahwa *client* tersebut tidak mendapatkan izin untuk menggunakan layanan. Pada baris 116 dijelaskan apabila variable tidak bernilai *false* (berisi data dari *client*) maka server menyimpan data-data parameter *client_details* dalam *session*.

Pada baris 122-155 memiliki fungsi untuk melakukan validasi *scope* yang dapat di akses oleh *client*. Baris 122 menjelaskan apabila *client* mengirimkan parameter *scope* maka akan dimasukan kedalam variable $params, dan apabila *client* tidak menyertakan parameter *scope* maka akan disimpan kedalam variable $params sebagai array kosong seperti yang dijelaskan pada baris 129. Pada baris 133 dilakukan pengecekan pada *scope* yang terdapat dalam variable $params jika isi *scope* pada variable $params lebih dari 0 maka dilakukan pengecekan pada tiap *scope* yang tersimpan dalam variable $params dengan yang terdapat pada database. Pada baris 138 menjelaskan apabila *scope* tidak terdaftar pada database maka akan menampilkan pesan eror. Baris 150 menjelaskan apabila *scope* yang terdapat pada variable $params berjumlah 0 maka akan menampilkan pesan eror yang menyatakan bahwa *client* tidak mengirimkan parameter *scope*.

Pada baris 158 menjelaskan bahwa server menyimpan variable $params kedalam *session*. Dan pada baris 161 ketika semua proses validasi sudah berhasil maka *client* akan langsung di *redirect* ke *class* oauth dan menjalankan fungsi sign_in.

5.1.3 Implementasi Proses *Generate Token*

Ketika *client* yang bertindak sebagai peminta layanan akan melakukan akses layanan yang ada pada belajardisini.com, *client* terlebih dahulu harus mendapatkan *token* yang berfungsi sebagai kunci untuk mendapatkan hak akses ke penyedia layanan. *Client* nantinya akan mendaftarkan aplikasi yang akan dikembangkan untuk bisa mendapatkan beberapa komponen dari OAuth yang akan digunakan untuk mengakses layanan.

Proses pembuatan *token* ini dijalankan dalam tampilan *web page* yang hanya bisa dilakukan oleh pemilik layanan. Berikut adalah potongan kode program dari sistem untuk proses pembuatan *token* tersebut :

```
1  <body lang="en">
2          <h1>Rest Api Key & OAuth Token Generator</h1>
3          <h2>Please enter your data</h2>
4          <?php echo form_open('oauth/generate_token/'); ?>
5          <?php if($error): ?>
6          <div class="error">
7                  <strong>Please correct the following errors</strong>
8                  <ul>
9                          <?php foreach($error_messages as $e): ?>
10                         <li><?php echo $e; ?></li>
11                         <?php endforeach; ?>
12                 </ul>
```

```
13          </div>
14          <?php endif; ?>
15          <p>
16                  <label for="app_name">app name</label><br>
17                  <input type="text" id="app_name" required="required"
18                  name="app_name" placeholder="app_name">
19          </p>
20
21          <p>
22                  <label for="rest_key">rest api key</label><br>
23                  <input type="text" id="rest_key"  required="required"
24                  readonly="readonly" name="rest_key" placeholder="rest_key"
25                  value="<?php echo $key; ?>" >
26          </p>
27
28          <p>
29                  <label for="client_id">client_id</label><br>
30                  <input type="text" id="client_id"  readonly="readonly"
31                  readonly="readonly"  required="required" name="client_id"
32                  placeholder="client_id"  value="<?php echo
33                  md5("CID".strval(rand(1000,5000)).strval( date('dmyhims')));?>">
34          </p>
35
36          <p>
37                  <label for="client_secret">client_secret</label><br>
38                  <input type="text" id="client_secret"   readonly="readonly"
39                  required="required" name="client_secret"
40                  placeholder="client_secret"  value="<?php echo
41                  md5("CSEC".strval(rand(5000,10000)).strval( date('dmyhims')));?>">
42          </p>
43
44          <p>
45                  <label for="redirect_uri">redirect_uri</label><br>
46                  <input type="text" id="redirect_uri"
47                  value="<?php echo base_url().index_page()."/apis/"; ?>"
48                  readonly="readonly"   required="required" name="redirect_uri"
49                  placeholder="redirect_uri">
50          </p>
51
52          <p>
53                  <input type="submit" name="validate_user" value="Generate">
54          </p>
55
56          <?php echo form_close(); ?>
57  </body>
58  </html>
```

Gambar 5.2 Potongan Kode Program Token Generator

Berikut ini adalah penjelasan mengenai potongan program diatas. Potongan code pada baris 4 berfungsi untuk membuka tag form html untuk token generator nya. Baris 5 berfungsi ketika ada kesalahan/error pada proses generate token tersebut maka akan menampilkan pesan seperti yang terdapat pada baris 6-13. Pada baris 17 berfungsi untuk membuat sebuah inputan nama aplikasi yang akan dibuatkan token. Baris 23 digunakan untuk melakukan generate rest_key. Nilai dari rest_key tersebut didapatkan dari hasil proses fungsi generate_key yang ada pada controller Key. Pada baris 30-33 berfungsi untuk melakukan generate client_id yang di generate dengan menggunakan hash md5. Sama halnya dengan potongan code pada baris 38-41 digunakan untuk melakukan generate client_secret yang di generate dengan menggunakan hash md5. Baris 47 berfungsi untuk menampilkan redirect_uri yang didapatkan dari base_url, index_page dan ditambahkan dengan lokasi api yang akan diakses oleh client.

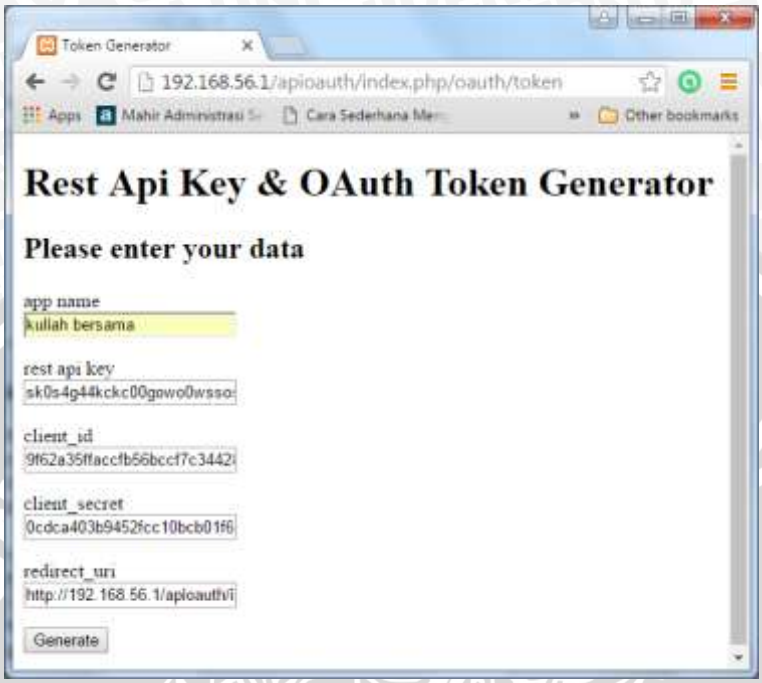
```
1  private function _generate_key()
2  {
3      do
4      {
5          // Generate a random salt
6          $salt = base_convert(bin2hex($this->security->get_random_bytes(64)), 16, 36);
7
8          // If an error occurred, then fall back to the previous method
9          if ($salt === FALSE)
10         {
11             $salt = hash('sha256', time() . mt_rand());
12         }
13         $new_key = substr($salt, 0, config_item('rest_key_length'));
14     }
15     while ($this->_key_exists($new_key));
16     return $new_key;
17 }
```

Gambar 5.3 Potongan Kode Program Fungsi Generate REST Key

Berikut adalah penjelasan mengenai potongan program diatas. Pada Baris 6 memiliki fungsu untuk melakukan kriptografi salt dengan fungsi php bin2hex. Baris 9 memiliki fungsi ketika salt bernilai *false* maka nilai dari salt adalah nilai hash berdasarkan nilai acak *timestamp* akses dari fungsi generate_key. Pada baris 13 ini digunakan untuk menghasilkan kode api dari salt dengan panjang *key* yang sudah diatur sebelumnya pada rest_key_length. Baris 15 memiliki fungsi untuk pengecekan kode yang sama pada database, apabila ada yang sama maka akan dilakukan generate ulang, tetapi apabila tidak ada yang sama maka akan langsung tersimpan dalam database.

Ketika file *token.php* yang berisi kode program diatas dijalankan pada halaman browser maka akan dilakukan proses pembuatan token yang nantinya akan langsung tersimpan pada database sehingga token-token yang sudah di *generate* akan langsung terdaftar dan sudah bisa digunakan oleh *client* untuk dijadikan kunci dalam mengakses layanan. Untuk komponen yang akan di dapatkan oleh *client* nanti adalah *rest api key, client_id, client_secret, redirect_uri*. Untuk *rest api key* di *generate* dengan proses enkripsi menggunakan *sha256*,

sedangkan *client_id* dan *client_secret* di *generate* dengan proses enkripsi menggunakan *md5*. Untuk tiap komponen tersebut dibentuk secara acak untuk tiap kali dilakukan permintaan *generate token*. Berikut adalah gambar ketika *generate token* dijalankan :



Gambar 5.4 Proses Token Generator

Ketika menjalankan file *token.php* maka akan ditampilkan seperti gambar diatas. Pada halaman tersebut akan dimina untuk memasukkan nama aplikasi yang akan menggunakan *token* nantinya. Untuk komponen selanjutnya seperti *rest api key, client_id, client_secret, redirect_uri* akan di *generate* secara acak untuk untuk tiap kali terjadi permintaan *generate token* kecuali pada *redirect_uri* diambil secara default dari lokasi API yang sudah didefinisikan pada server. Ketika tombol *generate* di klik maka semua komponen yang tampil seperti gambar diatas akan langsung tersimpan kedalam database dan siap digunakan.

5.1.4 Implementasi Proses *Authenticaton Client*

Aplikasi yang sudah terdaftar sudah bisa melakukan akses ke penyedia layanan. Dan disini disimulasikan dengan mengimplementasikan sebuah aktivitas *sign-in* menuju penyedia layanan yang dimana proses *sign-in* nantinya akan dilakukan oleh pengguna aplikasi dan pengguna melakukan proses persetujuan apakah pengguna tersebut menyetujui data dirinya dapat diakses oleh aplikasi atau tidak diizinkan. Berikut adalah gambar simulasi *sign-in* yang dilakukan oleh pengguna pada aplikasi :

Gambar 5.5 Tampilan Awal Aplikasi

Gambar 5.6 Tampilan Sign-in OAuth Aplikasi

Ketika pengguna menekan tombol *sign-in* maka proses *authentication credential client* (*client_id* dan *client_secret*) diproses menggunakan teknologi *authentication* OAuth 2.0. Tabel dibawah ini adalah potongan kode program dari sisi *client* yang menunjukkan proses awal *client* saat menekan tombol *sign-in*.

```
1  @SuppressLint("NewApi")
2  public class MainActivity extends Activity {
3
4  private static String DOMAIN = "192.168.56.1";
5  private static String CLIENT_ID = "2bfe9d72a4aae8f06a31025b7536be80";
6  private static String API_KEY =
7  "k408s8ggk000gsggsscwcwc00o8kg0ososg0w84k";
8  private static String CLIENT_SECRET = "9d667c2b7fae7a329f32b6df17926154";
9
```

```
10  private static String REDIRECT_URI = "http://" + DOMAIN
11                  + "/apioauth/index.php/apis/";
12  private static String GRANT_TYPE = "authorization_code";
13  private static String RESPONSE_TYPE = "code";
14  private static String OAUTH_SCOPE = "user";
15  private static String OAUTH_URL = "http://" + DOMAIN
16                  + "/apioauth/index.php/oauth/";
17
18  public void onClick(View arg0) {
19          //TODO Auto-generated method stub
20          web.getSettings().setJavaScriptEnabled(true);
21          web.loadUrl(OAUTH_URL + "?X-API-KEY=" + API_KEY
22                          + "&response_type=" + RESPONSE_TYPE + "&client_id="
23                          + CLIENT_ID + "&client_secret=" + CLIENT_SECRET
24                          + "&scope=" + OAUTH_SCOPE + "&redirect_uri="
25                          + REDIRECT_URI);
26          web.setWebViewClient(new WebViewClient() {
27
28                  boolean authComplete = false;
29                  Intent resultIntent = new Intent();
30
31                  @Override
32                  public void onPageStarted(WebView view, String url,Bitmap
33  favicon) {
34                          super.onPageStarted(view, url, favicon);
35                          view.setVisibility(View.GONE);
36                  }
```

Gambar 5.7 Potongan Kode Program Aplikasi Android

Berdasarkan potongan kode program diatas pada baris 4-14 dilakukan inisialisasi untuk komponen yang akan menjadi pramater dalam proses meminta akses layanan. Dapat dilihat bahwa diawal pembuatan program sudah didefinisikan terlebih dahulu token-token yang sudah di dapat pada saat proses *generate token* yang sudah dijelaskan pada pembahasan sebelumnya. Setelah itu pada baris 20 dilanjutkan untuk proses *load url* berdasarkan semua komponen yang telah didefinisikan diatas untuk dapat masuk ke layanan penyedia API. Dibawah ini adalah *url* yang diakses berdasarkan definisi gambar diatas :

```
192.168.56.1/apioauth/index.php/oauth/?X-API-
KEY=k408s8ggk000gsggsscwcwc00o8kg0ososg0w84k&response_type=c
ode&client_id=2bfe9d72a4aae8f06a31025b7536be80&&client_secre
t=9d667c2b7fae7a329f32b6df17926154&scope=user&redirect_uri=h
ttp://192.168.56.1/apioauth/index.php/apis/
```

Gambar 5.8 Url Load Aplikasi Android

Apabila domain atau *credential client* yang didefinisikan oleh aplikasi saat akan mengakses teknologi OAuth tidak sesuai atau tidak terdaftar pada database server penyedia layanan maka aplikasi akan mendapatkan pesan *error* seperti yang ditunjukan pada gambar berikut :

Gambar 5.9 Tampilan Error Credential Client

OAuth 2.0 melakukan kebijakan memeriksa validnya data dari seorang *client* lewat *authorization server*, sehingga hasil akhir *client* yang terpercaya dapat menerima akses ke aplikasi dengan *authorization server* menghasilkan code access yang muncul pada http (*hypertext transfer protocol*). Apabila dijalankan pada browser akan muncul pada kolom URL. *Code access* yang diberikan oleh *authorization server* adalah jika *client* melakukan tindakan persetujuan "Apakah Melanjutkan Otoritasi Pada Aplikasi ?". Seperti pada gambar 5.11. Pada gambar 5.12 yang muncul dengan kotak berwarna abu-abu di bawah adalah gambar saat *authorization code* dihasilkan yang terletak pada http (*hypertext transfer protocol*).

Gambar 5.10 Tampilan Authorization Request

Gambar 5.11 Tampilan Authorization Code

Setelah proses otorisasi telah berhasil dan *client* juga telah mendapatkan hak aksesnya untuk mengakses layanan maka ketika dilakukan proses *request* berikutnya yang dalam kasus ini dilakukan pengguna menekan tombol *profile* maka akan terjadi *request* dengan mengambil data pengguna yang sedang aktif pada saat itu seperti yang terlihat pada gambar dibawah.

Gambar 5.12 Tampilan Get Profile

BAB 6 PENGUJIAN

Pada bab 6 ini akan membahas mengenai hasil pengujian kinerja sistem dengan parameter keberhasilan yang menjadi tolak ukur pengujian yaitu sistem yang telah dikembangkan mampu mengelola *credential* yang diberikan oleh *client* untuk bisa ditukarkan menjadi hak akses ketika *client* akan mengakses layanan *web service* melalui persetujuan pengguna.

6.1 Pengujian Fungsional

Pengujian fungsional ini dilakukan untuk mengetahui apakah sistem yang dikembangkan sudah sesuai atau belum dengan apa yang dibutuhkan. Beberapa item yang telah dirumuskan dari analisa awal akan menjadi acuan untuk melakukan pengujian fungsional. Pengujian fungsional ini dilakukan dengan menggunakan metode pengujian *Black Box,* karena hanya akan memerlukan konsentrasi terhadap alur jalannya algoritma program saja dan lebih ditekankan untuk mendapatkan kesesuaian antara kinerja sistem yang diharapkan dengan hasil analisa awal.

6.1.1 Pengujian *Black Box*

Tahapan pengujian dimaksudkan untuk mengetahui tingkat kesuksesan dan keberhasilan dari sistem yang sudah dikembangkan berdasarkan sistem sebelumnya. Pengujian dilakukan setelah dilakukan implementasi dengan metode yang telah diusulkan penulis. Pengujian dengan metode *black box* merupakan pengujian yang terfokus pada spesifikasi fungsional dari perangkat lunak, menemukan kesalahan dalam penggunaan struktur data, kesalahan data yang tidak valid, memeriksa dan memastikan hasil yang diinginkan apakah sudah sesuai atau tidak pada sistem.

Adapun tahapan-tahapan dalam melakukan pengujian validasi sistem di tunjukkan seperti pada tabel 6.1 di bawah ini :

Tabel 6.1 Skenario Tahapan Pengujian Black Box

Kasus Uji	Nilai Masukan	Skenario Pengujian	Hasil Yang Diharapkan	Hasil Uji
***Generate token* baru**	Salah	Belum mendaftarkan nama aplikasi untuk *token* yang baru.	Terdapat pesan "*please fill out this field*"	Sukses
	Benar	Syarat pendaftaran diisi dengan lengkap.	*Token* yang telah terdaftar akan langsung masuk dalam database	Sukses
Sign-in	Salah	Pengguna mengisi form *sign-in* tidak	Muncul *dialog box* yang memberikan	sukses

		sesuai dengan data yang ada	informasi bahwa data yang diisi tidak valid	
	Salah	Pengguna tidak mengisi salah satu atau kedua *field* form *sign-in*	Muncul *dialog box* yang memberikan informasi bahwa *field* tidak boleh kosong	Sukses
	Benar	Pengguna mengisi form dengan lengkap dan benar sesuai data yang tersimpan	Pengguna akan diarahkan pada halaman *Authorize App Request*, dan jika menyetujui *authorize app akan mendapat authorization code*	Benar

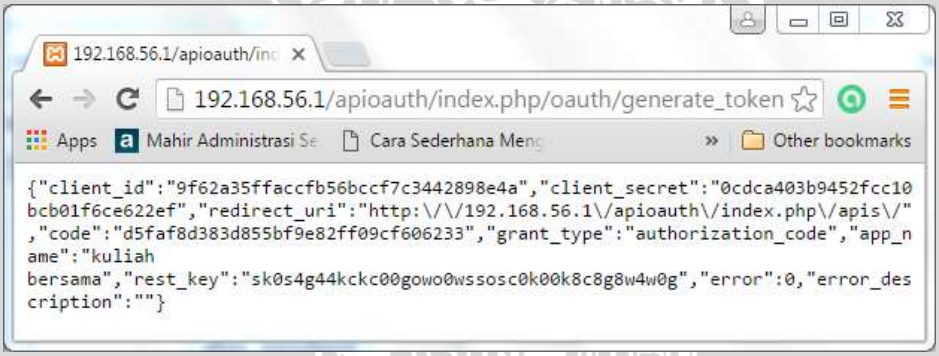
6.1.2 Pengujian *Generate Token*

Generate token ini hanya dapat dilakukan pada halaman browser yang hak aksesnya hanya boleh dilakukan oleh pemilik sumber daya atau bisa dikatakan dari sisi admin. Apabila pada saat akan melakukan *generate token* nama aplikasi yang akan menerima token tersebut tidak diisi maka proses ini tidak akan bisa dilanjutkan ke proses penyimpanan token pada database, karena untuk kebutuhan database agar pemilik sumber daya dapat mengetahui daftar token yang sudah digunakan. Dibawah ini adalah gambar ketika pada saat akan melakukan *generate token* dan tombol *generate* di tekan dengan kondisi nama aplikasi dikosongkan :

Gambar 6.1 Token Genrator Tanpa Nama Aplikasi

Gambar 6.2 Token Genrator Dengan Nama Aplikasi

Apabila nama aplikasi telah diisi dan tombol *generate* di tekan maka semua komponen yang tampil seperti yang terdapat pada gambar 6.2 akan langsung tersimpan kedalam database dan siap digunakan. Halaman pada gambar dibawah ini adalah ketika telah berhasil melakukan pendaftaran aplikasi dan setelah penekan tombol perintah *generate*.



Gambar 6.3 Hasil Token Generator

6.1.3 Pengujian *Sign-in* Dengan *Authentication OAuth 2.0*

Dalam melakukan pengujian *sign-in* ini, penulis mencoba untuk menguji dengan tidak menggunakan *emulator android* namun langsung diterapkan pada sebuah *smartphone* yang menggunakan sistem operasi *android*. Maksud dan tujuannya adalah untuk dapat dilakukan *capture request* dan *response* terhadap komunikasi yang muncul dari *smartphone* dengan penyedia layanan dengan menggunakan *software* pendukung yaitu *Wireshark*.

Berikut adalah aplikasi yang telah mendapatkan *client_id*, *client_secret* dan *redirect_uri* yang sudah terdaftar pada database :

Tabel 6.2 Daftar Client Pada Database

Nama Aplikasi	*Client_id*	*Client_secret*	*Redirect_uri*
Belajar Disini	2bfe9d72a4aae8f06a31025b7536be80	9d667c2b7fae7a329f32b6df17926154	http://192.168.56.1/apioauth/index.php/apis/
Kuliah Bersama	9f62a35ffaccfb56bccf7c3442898e4a	0cdca403b9452fcc10bcb01f6ce622ef	http://192.168.56.1/apioauth/index.php/apis/

Untuk pengujian tahap ini data dari aplikasi yang digunakan adalah aplikasi "Belajar Disini". Skenario pengujian yang akan dilakukan sesuai dengan yang terdapat pada tabel 6.1. Pertama kali aplikasi *mobile* dijalankan akan langsung ditampilkan tombol *sign-in* untuk dapat masuk ke halaman berikutnya seperti gambar dibawah ini :

Gambar 6.4 Tampilan Awal Aplikasi Android

Saat pengguna sudah menekan tombol *sign-in* dan tidak memasukkan data dengan benar seluruh *field* yang tersedia, tidak mengisi atau hanya mengisi salah satu dari *field* yang ada, maka akan muncul pesan peringatan seperti yang terdapat pada gambar 6.5. Dan pada saat pengguna mengisi *field* yang tersedia pada form *sign-in* OAuth dengan data yang tidak benar dan tidak terdapat pada database, di tampilan *smartphone* akan menampilkan pesan dalam bentuk *dialog box* seperti gambar 6.6 di bawah ini :

Gambar 6.5 Pesan Error Aktivitas Sign-in 1

Gambar 6.6 Pesan Error Aktivitas Sign-in 2

Saat pengguna sudah menekan tombol *sign-in* pertama kali dan apabila di capture dari *wireshark* maka akan terlihat sebuah *request* yang mengirimkan seluruh komponen dari OAuth tersebut seperti yang terdapat pada gambar 6.7. Ketika *client* yang bertindak sebagai pengguna melakukan dengan benar pengisian *field* yang ada pada form *sign-in*, maka proses selanjutnya yang terjadi adalah proses *authentication* menggunakan teknologi OAuth 2.0 dimana *credential* milik *client* akan di *otentikasi*. Jika dinyatakan valid, aplikasi akan menghasilkan halaman *authorize app* (persetujuan melakukan otorisasi aplikasi) seperti yang di tunjukkan pada gambar 6.8. Pada gambar di bawah adalah hasil capture dari *wireshark* untuk melihat *request* yang muncul dan *gambar* dimana pengguna telah mengisi semua *field* yang tersedia pada form *sign-in*.

```
  74 http → 39993 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=
  66 39993 → http [ACK] Seq=1 Ack=1 Win=87680 Len=0 TSval=85326 TSecr=131659
 761 GET /apioauth/index.php/oauth/?X-API-KEY=k408s8ggk000gsggsscwcwc00o8kg0ososg0w84k&res
 797 HTTP/1.1 307 Temporary Redirect  (text/html)
  66 39993 → http [ACK] Seq=696 Ack=732 Win=89088 Len=0 TSval=85333 TSecr=131666
 600 GET /apioauth/index.php/oauth/sign_in HTTP/1.1
```

Gambar 6.7 Tampilan Request Dengan Method GET

Gambar 6.8 Tampilan Authorization Confirm

Pengguna yang telah melakukan persetujuan melanjutkan otorisasi aplikasi pada halaman otorisasi, akan di arahkan menuju halaman berikutnya dari aplikasi tersebut. Bersamaan dengan itu informasi *authorization code* akan dihasilkan pada layar yang muncul sesaat setelah itu akan hilang dan secara bersamaan juga *code* tersebut tersimpan pada database server. Ketika semua *credential client* sudah valid maka aplikasi sudah dapat mengakses layanan dengan seizin pengguna. Apabila di *capture* pada *wireshark* nantinya akan muncul *code* tersebut yang disertakan pada saat proses *request*. Dibawah ini menunjukkan hasil *capture* dan gambar yang dihasilkan ketika *credential client* sudah valid :

```
 806 POST /apioauth/index.php/oauth/authorise HTTP/1.1  (application/x-www-form-urlencoded
  66 http → 39994 [ACK] Seq=2013 Ack=2135 Win=65024 Len=0 TSval=132500 TSecr=86147
1233 HTTP/1.1 303 See Other  (text/html)
 779 GET /apioauth/index.php/apis/?code=57e44a1d363d498fd0e8ad193d40ccae&X-API-KEY=k408s8g
 153 Solicit XID: 0xec955e CID: 000100011aa44a4a08edb9d40b8b
 184 Advertise XID: 0xec955e CID: 000100011aa44a4a08edb9d40b8b
 509 HTTP/1.1 200 OK  (application/json)
```

Gambar 6.9 Tampilan Authorization Code Pada Wireshark

Gambar 6.10 Tampilan Authorization Code

6.2 Pengujian HTTP

Skenario pada pengujian ini akan dilakukan dengan menjalankan proses *request* dengan menggunakan protokol HTTP kemudian di lihat hasil komunikasi yang terjadi ketika adanya *request/response* yang terjadi dengan menggunakan *wireshark*.

6.2.1 *Request* HTTP

Request HTTP dilakukan dengan menerapkan protokol HTTP pada aplikasi *client* saat akan mengakses layanan dengan melakukan percobaan pada aplikasi *client* yang dijalankan langsung pada *smarthphone* untuk dapat dilihat komunikasi yang terjadi ketika menggunakan proses *request* HTTP. Seperti yang terlihat pada gambar dibawah yaitu dilakukan sebuah *request* yang membutuhkan peran serta pengguna ketika perintah *sign-in* pada aplikasi *client* dijalankan.

Gambar 6.11 Tampilan Sign-in

Ketika pengguna menekan tombol *sign-in* maka akan terjadi sebuah request yang dimana ketika *client* mengirimkan url load yang sudah didefinisikan pada saat awal pembuatan aplikasi *client* akan di periksa terlebih dahulu oleh *authorization server* untuk dapat dilanjutkan ke proses selanjutnya yaitu otorisasi aplikasi. Gambar dibawah ini adalah url load yang dikirim ketika request awal dilakukan oleh aplikasi *client*.

```
http://192.168.43.6/apioauth/index.php/oauth/?X-API-
KEY=k408s8ggk000gsggsscwcwc00o8kg0ososg0w84k&response_type
=code&client_id=2bfe9d72a4aae8f06a31025b7536be80&&client_s
ecret=9d667c2b7fae7a329f32b6df17926154&scope=user&redirect
_uri=http://192.168.43.6/apioauth/index.php/apis/
```

Gambar 6.12 Tampilan Url Load

Ketika aplikasi *client* mulai dijalankan, maka disaat bersamaan juga dilakukan *capture* komunikasi yang terjadi pada saat adanya *request* yang datang dari aplikasi *client*. Dapat dilihat pada gambar dibawah menandakan bahwa penggunaan protokol HTTP pada proses *request* sebuah aplikasi masih dapat terlihat karena tidak adanya enkripsi yang diterapkan, sehingga masih ada kemungkinan pengguna lain bisa mengambil informasi yang didapat dari hasil capture komunikasi yang terjadi pada *wireshark* untuk digunakan kembali dalam melakukan proses *request*.

Gambar 6.13 Tampilan Request Pada Wireshark

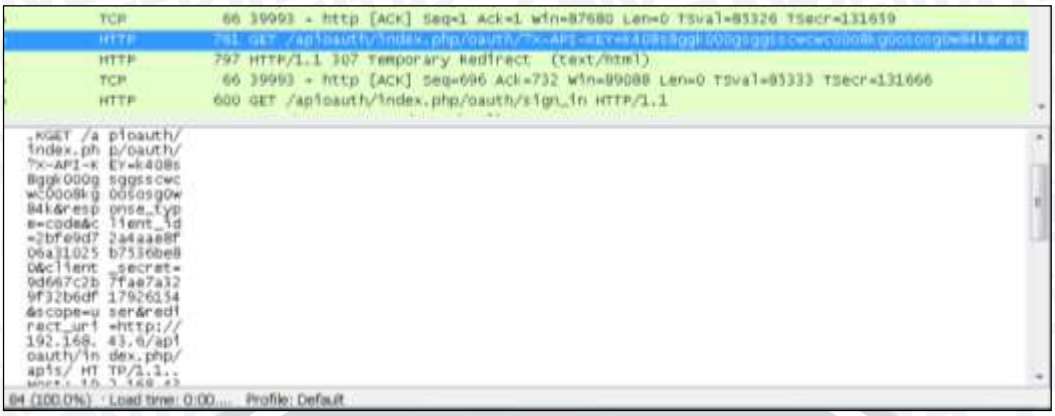
6.3 Pengujian Keamanan

Pada pengujian keamanan ini dilakukan dengan beberapa skenario pengujian yang bertujuan untuk mengetahui apakah kerahasiaan milik *client* masih dapat dijaga atau tidak.

6.3.1 Penggunaan Kembali *Credential Client*

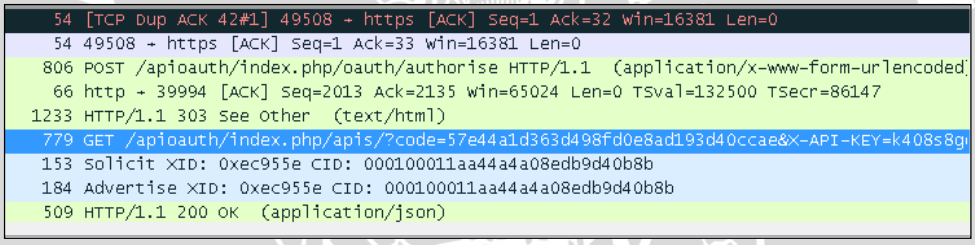
Pada pengujian ini dilakukan sebuah simulasi dimana ketika *authorization code* sudah dihasilkan setelah adanya proses otorisasi dan digunakan kembali oleh aplikasi lain untuk dapat mengakses data yang ada pada server. Pengujian ini dilakukan dengan melakukan sebuah percobaan yang dimulai dari *capture* komunikasi *request* dan *response* yang terjadi dengan menggunakan *wireshark* sampai melakukan percobaan dengan sebuah *tools* jmeter untuk membuat simulasi penggunaan kembali *authorization code* tersebut untuk mendapatkan hak akses.

Tahap awal dalam pengujian ini adalah melakukan *capture* pada *wireshark* untuk melihat *credential client* dan *authorization code* yang telah dihasilkan dari proses otorisasi seperti gambar berikut :



Gambar 6.14 Tampilam Credential Client Pada Wireshark

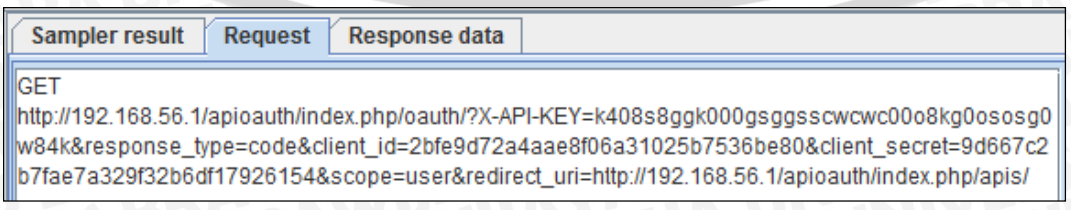
Berdasarkan hasil *capture* yang ada pada gambar diatas telah didapatkan hasil dari proses *request* yang telah dilakukan oleh *client* menuju *server* yaitu berupa *client_id* "2bfe9d72a4aae8f06a31025b7536be80", *client_secret* "9d667c2b7fae7a329f32b6df17926154", *response_type* "*code*", *scope* "*user*",*redirect_uri* "http://192.168.56.1/apioauth/index.php/apis/", dan *X-API-KEY* "k408s8ggk000gsggsscwcwc00o8kg0ososg0w84k".



Gambar 6.15 Tampilan Authorization Code Pada Wireshark

Pada gambar diatas dapat dilihat bahwa *authorization code* yang dihasilkan setelah proses otorisasi *client* adalah "57e44a1d363d498fd0e8ad193d40ccae". Setelah *client* sudah mendapatkan *authorization code* maka *client* tersebut sudah bisa mengakses data pengguna yang ada karena sudah mendapatkan hak akses.

Berdasarkan kegiatan pengujian yang sudah dilakukan pada tahap awal tersebut, tahap berikutnya akan dilakukan simulasi dengan menggunakan *tools* jmeter dalam proses mengakses layanan dan dilakukan secara berurutan mulai dari *request* pertama kemudian *request* kedua. Jadi dilakukan simulasi dengan melakukan proses *request* dalam waktu yang bersamaan sesuai urutannya. seperti yang terlihat pada gambar berikut :



Gambar 6.16 Simulasi Request Akses Menggunakan Jmeter 1

Gambar 6.17 Simulasi Request Akses Menggunakan Jmeter 2

Pada gambar diatas adalah proses request yang disimulasikan pada *tools* jmeter untuk mendapatkan akses ke layanan yang dituju dengan menggunakan *credential client* dan *authorization* yang sama berdasarkan hasil *capture* dengan menggunakan *wireshark* yang sudah dilakukan sebelumnya.

Gambar 6.18 Response Dari Request Akses Menggunakan Jmeter 1

Gambar 6.19 Response Dari Request Akses Menggunakan Jmeter 2

Pada gambar 6.18 dan 6.19 diatas dijelaskan bahwa simulasi dari proses *request* yang dilakukan telah berhasil dengan mengatas namakan bahwa *client* baru yang dilakukan pada simulasi tersebut adalah sama dengan *client* aslinya. Kemudian *response* akhir yang diterima berupa *Access Granted* yang menandakan bahwa *client* baru tersebut telah mendapatkan hak akses untuk masuk kedalam layanan.

Berdasarkan percobaan diatas maka diperlukan sebuah mekanisme yang dapat mengamankan *credential client* tersebut sehingga apabila ada pelaku jahat yang mencoba untuk menggunakannya kembali masih dapat dicegah karena *credential* milik *client* tersebut sudah diamankan dengan salah satu mekanisme yang bisa digunakan yaitu mengubah protocol HTTP menjadi HTTPS karena apabila menggunakan HTTPS semua *request* yang dikirimkan akan dienkripsi terlebih dahulu sehingga mengurangi peluang pelaku jahat untuk mengambil informasi atau data-data yang ada ketika proses transaksi sedang berlangsung.

6.3.2 Decompile APK

Pengujian dengan melakukan *decompile* APK dilakukan untuk mengetahui apakah token yang sudah didefinisikan sebelumnya pada saat proses awal pembuatan aplikasi tersebut masih bisa terlihat atau tidak pada saat dilakukan *decompile*. APK yang sudah di generate akan di *decompile* dengan memanfaatkan *compiler tools* yang sudah disediakan pada situs www.javadecompilers.com

dengan melakukan upload file APK yang akan di *decompile* lalu untuk mengetahui hasilnya. Berikut adalah proses *decompile* APK yang telah dilakukan :

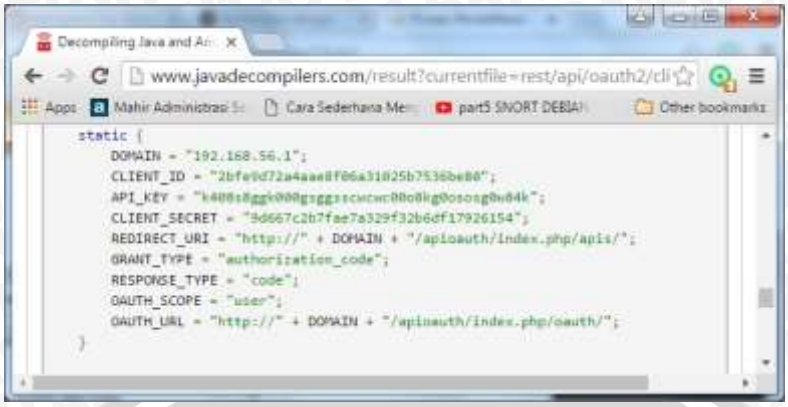
Gambar 6.20 Proses Decompile APK 1

Gambar diatas menunjukan proses awal untuk melakukan *decompile* APK dengan memanfaatkan situs www.javadecompilers.com. Pertama kali membuka situs tersebut akan ditampilkan halaman seperti gambar diatas yang terdapat perintah untuk memilih file APK mana yang akan di *decompile*.

Gambar 6.21 Proses Decompile APK 2

Setelah menekan perintah *Upload and Decompile* yang terdapat pada halaman web tersebut maka akan ditampilkan hasil dari *decompile* yang telah dilakukan, apabila proses *decompile* telah berhasil maka akan ditampilkan pesan seperti yang terlihat pada gambar diatas.

Untuk mengetahui token yang telah didefinisikan pada kode program tersebut apakah masih bisa terlihat atau tidak maka bisa dilakukan langsung dengan cara membuka file *MainActivity.java* yang ada pada folder dari hasil *decompile* seperti gambar berikut :



Gambar 6.22 Hasil Decompile APK

Gambar diatas adalah hasil dari proses *decompile* APK yang sudah dilakukan dengan membuka file *MainActivity.java* yang dapat dilihat bahwa ketika file APK di *decompile* ternyata komponen-komponen OAuth yang menjadi parameter dalam proses meminta akses terhadap layanan masih dapat terlihat dan masih bisa dipergunakan kembali untuk membuat aplikasi yang serupa dengan memanfaatkan komponen-komponen tersebut. Oleh karena itu, untuk mengembangkan sebuah aplikasi yang didalamnya terdapat kode-kode penting yang seharusnya tidak layak untuk diketahui oleh para pengguna lain maka perlu dilakukan proses enkripsi pada kode program yang akan dikembangkan untuk mencegah pengguna jahat yang ingin memanfaatkan kode-kode tersebut dari program yang sudah dikembangkan.

BAB 7 PENUTUP

Berdasarkan perancangan, implementasi, dan pengujian yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut.

7.1 Kesimpulan

Berdasarkan pembahasan dari bab-bab sebelumnya, serta dari hasil analisis hingga pengujian kinerja sistem yang telah dilakukan, maka dapat diambil beberapa kesimpulan sebagai berikut :

1 Dapat di implementasikan teknologi *authentication* OAuth 2.0 pada *web service* yang berperan dalam proses *authentication credential* milik *client* yang merupakan pengguna aplikasi pada aplikasi belajardisini.com. Melalui mekanisme *authentication* OAuth 2.0 yang melibatkan *authorization server*, berperan dalam menghadirkan halaman proses otorisasi aplikasi (*authorize app*). *Authorize App* yang disetujui oleh *client* akan dikembalikan lagi oleh *authorization server* dalam bentuk *authorization code* untuk mendapatkan hak akses menuju layanan. *Authorization code* yang dihasilkan setiap kali *client* berhasil melakukan aktivitas *sign-in* serta menyetujui permintaan otorisasi akan menghasilkan kode unik dalam bentuk *string* yang berbeda dengan yang sebelumnya.

2 Berdasarkan hasil pengujian sistem apabila dilihat dari sisi fungsional, sistem sudah berhasil dijalankan dengan memanfaatkan mekanisme *authentication* dengan sebuah fitur *sign-in* yang menggunakan teknologi OAuth 2.0, namun dilihat dari sisi keamanan *credential client* masih belum dilakukan penerapan mekanisme yang mengatur untuk proses mengamankan *credential client* tersebut.

7.2 Saran

Adapun saran-saran yang dapat penulis berikan setelah melakukan penelitian ini yaitu untuk penelitian, pengembangan dan pembangunan mekanisme *authentication* dengan metode OAuth pada penelitian berikutnya disarankan untuk dilakukan pengembangan dengan mengembangkan sistem yang berjalan pada protokol HTTPS agar proses komunikasi yang terdapat pada sistem bisa lebih aman dengan proses enkripsi yang tersedia dalam penggunaan HTTPS. Disarankan juga untuk penelitian selanjutnya agar melakukan enkripsi kode program pada aplikasi yang akan dikembangkan untuk mengurangi celah bagi pengguna lain yang akan memanfaatkan kode-kode pada program yang akan dikembangkan.

DAFTAR PUSTAKA

Agarwal, T. & Leonetti, M., 2013. Design and Implementation of an IP based authentication mechanism for Open Source Proxy Servers in Interception Mode. *Advanced Computing : An International Journal (ACIJ),* Volume 4, No. 1.

asana.com, -. *Authentication.* [Online] Available at: https://asana.com/developers/documentation/getting-started/auth [Diakses 29 September 2015].

Brail, G. & Ramji, S., 2012. *OAuth-The Big Picture.* [Online] Available at: http://apigee.com [Diakses 29 November 2015].

Burke, B., 2014. *RESTful Java with JAX-RS 2.0.* 2nd penyunt. United States of America: O'Reilly Media.

Byod, R., 2012. *Getting Started With OAuth 2.* 1st penyunt. United States of America: O'Reilly Media.

developers.google.com, 2016. *Using OAuth 2.0 to Access Google APIs.* [Online] Available at: https://developer.google.com/accounts/docs/OAuth2 [Diakses 17 Mei 2016].

E. Hammer-Lahav, E., 2010. The OAuth 1.0 Protocol. *Internet Engineering Task Force (IETF). ISSN : 2070-1721.*

Fielding, R. T., 2000. *Architectural Styles and the Desaign of Network based Software Architectures,* Irvine: Doctor of Philosophy Dissertation in Information and Computer Science, University of California.

github.com, 2012. *OAuth 2.0.* [Online] Available at: https://code.google.com/p/google-api-php-client/wiki/OAuth2 [Diakses 29 November 2015].

Hardt, E. D., 2012. The OAuth 2.0 Authorization Framework. *Internet Enginering Task Force (IETF). ISSN : 2070-1721.*

json.org, 2014. *JSON.* [Online] Available at: http://www.json.org [Diakses 29 November 2015].

Kaur, E. G. & Aggarwal, E. D., 2013. A survey Paper On Social Sign On Protocol OAuth 2.0. *Journal of Engineering, Computer & Applied Sciences (JEC & AS). ISSN : 2319-5606,* Volume 2, No. 26.

Nurseotiv, N., Paulson, M., Reynolds, R. & Izurieta, C., 2009. *Comparison of JSON and XML Data Interchange Formats : A Case Study.* s.l., ICSA 22nd

International Conference on Computer Application in Industry and Engineering.

Parecki, A., 2012. *OAuth 2 Simplified.* [Online] Available at: http://aaronparecki.com/articles/2012/07/29/1/oauth2-simplified [Diakses 29 November 2015].

w3.org, 2005. *W3C Document Object Model.* [Online] Available at: http://www.w3.org/DOM [Diakses 28 November 2015].

w3.org, 2005. *Web Services Architecture.* [Online] Available at: https://www.w3.org/TR/wsSarch/ [Diakses 28 november 2015].

Wellem, T., 2009. Perancangan Prototype Aplikasi Mobile Untuk Pengaksesan Web Service. *Seminar Nasional Informatika.*