PERANCANGAN DAN IMPLEMENTASI SSO (SINGLE SIGN ON) MENGGUNAKAN PROTOKOL OAUTH 2.0

Agus Suhardi[1], Erfanti Fatkhiyah[2], Muhammad Sholeh[3]
Program Studi Teknik Informatika, Fakultas Teknologi Industri
Institut Sains & Teknologi AKPRIND Yogyakarta
Email : [1]agus.suhardii@gmail.com, [2] erfanti@akprindl.ac.id, [3]muhash@akprind.ac.id

ABSTRACT

Web-built applications generally require users to login before making a transaction, so users have to log in repeatedly to be able to perform transactions on different servers so it is not efficient because it must do the same. In the application database will also have the same data that is the user name and password on different servers, so that will lead to data redundancy.

This study discusses how to separate user information with data on different servers, with SSO (Single Sign On) users can perform once login to be able to access different server. SSO-based web applications using the OAuth 2.0 protocol can provide Authorization between data servers and user servers, so user information is not duplicated on either a server or another server.

Keywords : SSO, OAuth 2.0, Authorization Server, Spring, Java, Angular 2

INTISARI

Aplikasi yang dibangun berbasis web pada umumnya menuntut pengguna aplikasi untuk login sebelum melakukan transaksi, dengan demikian pengguna harus melakukan login secara berulang-ulang untuk dapat melakukan transaksi pada server yang berbeda sehingga hal ini tidak efisien karena harus melakukan hal yang sama. Pada basis data aplikasi juga akan memiliki data yang sama yaitu nama pengguna dan kata sandi pada server yang berbeda, sehingga akan mengakibatkan redudansi data.

Penelitian ini membahas tentang bagaimana memisahkan informasi pengguna dengan data pada server yang berbeda, dengan SSO (Single Sign On) pengguna dapat melakukan sekali login untuk dapat mengakses server yang berbeda. Aplikasi web berbasis SSO menggunakan protokol OAuth 2.0 dapat memberi Authorization antara server data dan server pengguna, sehingga informasi pengguna tidak terduplikasi baik pada satu server maupun server lainnya.

Kata kunci : SSO, OAuth 2.0, Authorization Server, Spring, Java, Angular 2

PENDAHULUAN

Penggunaan internet sebagai layanan informasi dan komunikasi dapat memberikan kemudahan bagi penggunanya untuk melakukan transaksi dan komunikasi tanpa harus melakukan tatap muka secara langsung. Seorang pengguna internet dapat dengan mudah mencari dan memanfaatkan layanan-layanan yang diberikan oleh sebuah situs *web* hanya dengan mengakses internet saja.

Aplikasi *web* yang diakses dapat memberikan informasi yang berbeda pada setiap pengguna meskipun situs yang diakses sama, sebagai contoh *facebook* dan *gmail*. Meskipun dua orang pengguna *facebook* mengakses *url* yang sama seperti *https://web.facebook.com* tetapi hasil yang ditampilkan akan berbeda, Untuk mengetahui pengguna yang mengakses biasanya situs *web* akan melakukan *authentication* yaitu sebuah cara untuk memverifikasi apakah pengguna yang mengakses berhak atau tidak.

Authentication dapat dikenali dengan memberikan kerahasiaan informasi seperti *password, pin,* atau *private key*. Dengan memberikan salah satu kerahasiaan informasi tersebut situs *web* dapat mengenali identitas pengunjung misalnya *authentication* menggunakan proses *login* dengan memasukan *username* dan *password* yang benar maka situs *web* akan mengenali pengunjung tersebut.

Authentication memerlukan sebuah kerahasiaan dari seorang pengguna aplikasi *web* agar dapat dikenali, maka setiap kali mengakses situs *web* yang memerlukan *authentication.* Pengguna harus memasukan *username* dan *password* untuk mengakses setiap situs *web* tersebut. Proses ini akan mempersulit pengguna aplikasi karena setiap

kali mengakses situs *web* yang berbeda pengguna harus melakukan *login* ulang jika mengakses *server* yang berbeda, untuk mengatasi hal tersebut dapat menggunakan *authorization*.

Authorization merupakan proses memberikan hak akses kepada pihak ketiga untuk dapat melakukan akses data yang dimiliki, proses ini dapat dilakukan jika telah melakukan *authentication* agar informasi yang diberikan sesuai dengan pengguna yang telah melakukan proses *login*. Sebagai contoh: penggunaan *authorization* pada google. Dengan memiliki *account* google seorang pengguna dapat mengakses setiap *server* yang diberikan oleh google tanpa harus *login* berulang-ulang, seperti mengakses *gmail*, google+, *youtube*, dan *server* lainnya yang disediakan oleh google. Selain dapat mengakses *server* milik google, pengguna juga dapat melakukan *authentication*ke situs *web* lain dengan hanya melakukan *login* dari google terlebih dahulu. Situs lainnya yang menyediakan *authorization* adalah *facebook*, *twitter*, *linkIn*, dan lainnya.*ProtocolOAuth 2.0 (Open Authorization)* merupakan *protocol authentication*yang bersumber dari layanan penyedia *API (Application Programming Interface)* yang memberikan kuasa kepada orang untuk mendapatkan akses mereka dengan menukarkan *username* dan *password* menjadi *token*. Dengan menggunakan *protocol* ini, aplikasi pihak ketiga dapat mengakses data dari aplikasi yang telah dibangun.

TINJAUAN PUSTAKA

Saat ini aplikasi web memaksa pengguna mengingat beberapa kredensial otentikasi (nama pengguna dan kata sandi) untuk setiap aplikasi menjadikan tugas tidak praktis Mengingat beberapa kredensial pengguna menggunakan kembali kata kunci yang sama, memilih kata sandi yang lemah, atau menyimpan daftar semua nama pengguna dan kata sandi. Mengelola banyak kredensial otentikasi mengganggu pengguna dan melemahkan keamanan sistem autentikasi. Sistem Single-Sign-On (SSO) memungkinkan satu nama pengguna dan kata sandi untuk digunakan pada web yang berbeda aplikasi. Bagi pengguna, sistem SSO membantu menciptakan apa yang disebut identitas federasi. Dengan demikian Aplikasi berbasis SSO bisa melakukan Manajemen identitas yang akan menguntungkan baik pengguna maupun penyedia aplikasi. Pengguna hanya mengingat satu nama pengguna dan satu kata sandi, pengguna tidak perlu melakukan proses pendaftaran ulang yang berlebihan pada aplikasi yang berbeda.

Penelitian ini dikembangkan dari beberapa literature pustaka referensi aplikasi Diantaranya oleh Kridalukmana & Satoto (2014). Pengguna aplikasi akan dibantu untuk menyimpan hanya satu informasi login untuk masuk ke sebuah aplikasi. Tidak mengubah fungsi login di setiap aplikasi adalah batas yang telah ditetapkan saat mengelola kredensial. Untuk alasan ini, secara tidak langsung pendekatan single-sign-on digunakan dalam penelitian ini. Sementara itu, aplikasi akan diintegrasikan menggunakan model integrasi presentasi di lingkungan berbasis web. Menggunakan CodeIgniter Framework, portal single-sign-on telah dikembangkan untuk menangani single login ke aplikasi yang ada di Universitas Diponegoro. Sebagai hasil dari penelitian ini, peneliti menemukan bahwa dengan menggunakan pendekatan single-sign-on secara tidak langsung hampir semua variabel sesi dari aplikasi domain sekunder dapat berjalan dengan baik di bawah aplikasi domain utama kecuali aplikasi sekunder yang menggunakan kunci sesi variabel dinamis. Keadaan ini akan berdampak pada sebuah Proses keluar saat pengguna keluar dari aplikasi domain utama.

Penelitian selanjutnya dilakukan oleh Ionita (2012). Penelitian ini menyajikan implementasi beberapa solusi open source seperti OpenID, NTLM (NT LAN Manager) di PHP atau Central Authentication Service. Untuk aplikasi yang menggunakan OpenID sebuah toko online yang memiliki halaman administrasi, akses mana yang dibatasi dengan kredensial OpenID, selanjutnya dijamin dengan memfilter ID ini di database MySQL untuk mencegah akses ke orang yang memiliki kredensial OpenID yang valid. Untuk demonstrasi SSO yang diaktifkan oleh NTLM situs web lain digunakan untuk akses yang telah diblokir ke halaman. Untuk mengakses halaman ini diperlukan kredensial otentikasi Windows yang valid. Baik situs online shop dan demonstrasi untuk NTLM ditulis dalam PHP. Selain itu, sistem yang menggunakannya diimplementasikan juga menggunakan PHP. Untuk mengimplementasikan demo CAS, dua servlet Tomcat telah digunakan untuk tujuan demonstrasi, penyaringan tambahan dilakukan oleh server ApacheDS.

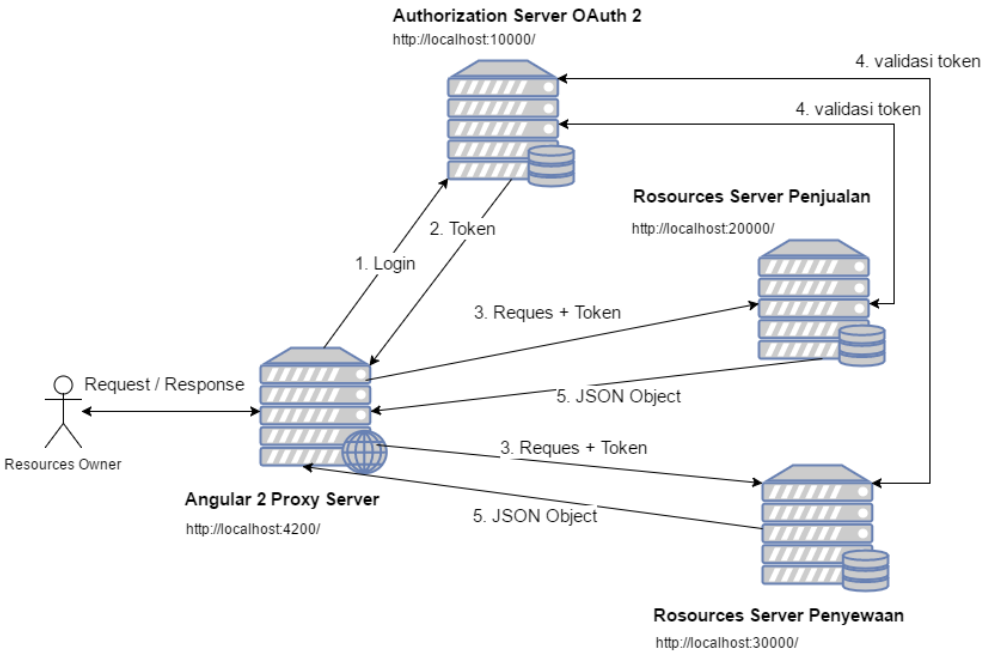
Berikutnya ditinjau dari penelitian serupa yang di lakukan oleh Subhash dan kahate(2016) yang membedakan metode akses data pada SSO. Metode kontrol akses yang memungkinkan pengguna masuk sekaligus dan mendapatkan akses ke sumber daya beberapa sistem perangkat lunak tanpa diperkuat untuk masuk lagi. Karena berbagai aplikasi dan sumber daya mendukung mekanisme otentikasi yang berbeda, single sign-on harus menerjemahkan secara internal dan menyimpan kredensial yang berbeda dibandingkan dengan yang digunakan untuk otentikasi awal. penelitian ini memberikan gagasan yang jelas tentang kerangka kerja yang berbeda yang digunakan untuk SSO.

Berbeda dengan penelitian SSO sebelumnya yang menggunakan Protokol OpenID dan Bahasa pemrograman PHP, aplikasi ini menggunakan dua Bahasa pemrograman yaitu TypeScript yang digunakan sebagai *front end*pada *angular 2.0*dan JAVA dengan framework Spring Boot sebagai *back end*. Sedangkan Protokol SSO yang digunakan adalah Protokol OAuth 2.0, sedangkan basis data yang digunakan adalah MariaDB

PEMBAHASAN

Aplikasi SSO yang dibangun menggunakan Bahasa pemrograman JAVA berbasis web dengan framework Spring boot, sebagai Authorization aplikasi ini menggunakan protocol OAuth 2. Hasil dari aplikasi ini memiliki 4 server dan masing-masing server akan dipisahkan dengan port yang berbeda-beda yang akan terhubung ke proxy. Pemisahan server dengan port ini di lakukan karena aplikasi di uji pada satu buah Komputer, jika dalam kasus sebenarnya tentu saja aplikasi ini dapat dipisahkan dengan server yang berbeda.

Gambar 1 merupakan gambaran arsitektur sistem yang dibangun, Berikut adalah pembahasan spesifik dari alur jalannya aplikasi dengan pengguna yang sudah melakukan pendaftaran



Gambar 1 Desain Sistem Aplikasi

1. Login

Resource Owner (pengguna) mengaksess angular 2 untuk melakukan login dengan memasukan nama pengguna dan kata sandi yang benar, pada bagian ini Angular 2 akan memilih Authorization Server sebagai tempat Resource Owner di simpan.

Untuk menentukan permintaan yang sesuai dari permintaan pengguna, angular 2 menggunakan proxy yang sudah disisipkan pada angular 2. Gambar 2 merupakan

Source Code pengaturan proxy pada angular 2 yang menyatakan target url pada nomor 3, 7, dan 11 :

```
{
  "/oauth/**": {
    "target": "http://localhost:10000",
    "secure": false
  },
  "/penjualan/api/**" : {
    "target": "http://localhost:20000",
      "secure": false
  },
  "/penyewaan/api/**" : {
    "target": "http://localhost:30000",
    "secure": false
  }
}
```

Gambar 2 Source Code pengaturan Proxy pada angular 2

2. Token

Proses pengecekan nama pengguna dan kata sandi akan di lakukan oleh *Authorization server OAuth 2* yang dikirim dari *Angular 2 Proxy Server* di masukan oleh pengguna(*Resources Owner*) pada form yang login. Gambar 3 menunjukan Source Code untuk mengecek nama pengguna dan kata sandi yang dikirim ke Authorization Server melalui form pada angular 2

```
let urlToken =
"/oauth/token?grant_type=password&username="+username+"&password="+password;
```

Gambar 3 Source Code pengecekan nama pengguna dan kata sandi

Response dari Authorization Server jika nama pengguna dan kata sandi benar adalah *true* atau *false*, kemudian akan dikirim token sebagai respon jika nama pengguna dan kata sandi benar. Gambar 4 merupakan Source Code pengolahan respon yang di terima dari Authorization Server.

```
return this.http.post(urlToken, body: null, options).toPromise()
  .then( onfulfilled: hasil => {
    let data = hasil.json();
    console.log("Access Token : "+data.access_token);
    localStorage.setItem( key: "access_token", data.access_token);
    localStorage.setItem( key: "userInfo", JSON.stringify( value: {username : username, fullname: username}));
    return true;
  })
  .catch( onrejected: hasil => {
    console.log(hasil);
    return false;
  });
}
```

Gambar 4 Source Code pengolahan response

Token yang telah didapat akan di simpan ke penyimpanan browser agar mudah diakses, Gambar 5 merupakan Source Code untuk menyimpan ke Local Stored sebagai penyimpanan Browser, metode ini akan menyimpan dua informasi pengguna yang telah melakukan proses login yaitu pada baris 42 akan menyimpan informasi pengguna dan pada baris 43 akan menyimpan token yang telah di dapat dari Authorization server

```
logOut(): void {
  localStorage.removeItem( key: "userInfo");
  localStorage.removeItem( key: "access_token");
}
```

Gambar 5 Source Code penyimpanan Browser

Token yang telah didapat akan di kelolah agar dapat di gunakan pada saat yang diperlukan, Gambar 6 merupakan Source Code untuk mengelolah token yang telah didapat dari Authorization Server. Pada baris 1 sampai 4 menunjukan library yang di perlukan untuk mengelolah token sedangkan baris selanjutnya adalah proses untuk mengelolah token yang telah di dapat

```
import {Injectable} from '@angular/core';
import {Http, RequestOptions, Headers} from "@angular/http";
import 'rxjs/add/operator/toPromise';
import {tokenNotExpired} from "angular2-jwt";
@Injectable()
export class AuthService {
  constructor(private http: Http) { }
  sudahLogin(): boolean {
    return tokenNotExpired( tokenName: "access_token");
  }
  login (username : string, password : string) : Promise<boolean> {
    let urlToken = "/oauth/token?grant_type=password&username="+username+"&password="+password;
    let basicAuth = "Basic YXBsaWthc2lqcnphc0xpa2FzaTEyMw==";
    let headers = new Headers({ 'Authorization': basicAuth });
    let options = new RequestOptions({ headers: headers });
    return this.http.post(urlToken, body: null, options).toPromise()
      .then( onfulfilled: hasil => {
        let data = hasil.json();
        console.log("Access Token : "+data.access_token);
        localStorage.setItem( key: "access_token", data.access_token);
        localStorage.setItem( key: "userInfo", JSON.stringify( value: {username : username, fullname: username}));
        return true;
      })
      .catch( onrejected: hasil => {
        console.log(hasil);
        return false;
      });
  }
  logOut(): void {
    localStorage.removeItem( key: "userInfo");
    localStorage.removeItem( key: "access_token");
  }
  getUserInfo() {
    return JSON.parse(localStorage.getItem( key: "userInfo"));
  }
```

Gambar 6 Source Code Pengolahan token

3. Request + Token

Setelah berhasil mendapatkan token, pengguna dapat mengakses *Resource Server*. Request dapat di lakukan ke *Resource Server* dengan menyertakan token yang telah didapat. Sebagai contoh Gambar 7 merupakan potongan Source Code yang akan melakukan request ke Resources Server penyewaan. Pada baris 5 akan mendapat koneksi ke resources server, baris 6 menunjukan fungsi untuk mendapatkan data dari resources server, pada baris 15 adalah sebua method untuk menyimpan data yang diambil dari form yang di input oleh pengguna, dan untuk menghapus data berada pada baris ke 37

```
@Injectable()
export class SewaService {
  constructor(private authHttp: AuthHttp) { }
  getSewa():Promise<any>{
    return this.authHttp.get( url: "penyewaan/api/sewa/").toPromise()
      .then( onfulfilled: hasil=>{
        return hasil.json();
      }).catch( onrejected: error=>{
        console.log(error);
        return error;
      });
  }
  simpan(user) : Promise<boolean> {
    let url = "penyewaan/api/sewa/save";
    if(user.id != null){
      return this.authHttp.post(url, user).toPromise()
        .then( onfulfilled: hasil => {
          return true;
        })
        .catch( onrejected: error => {
          console.log(error);
          return false;
        });
    } else {
      return this.authHttp.post(url, user).toPromise()
        .then( onfulfilled: hasil => {
          return true;
        })
        .catch( onrejected: error => {
          console.log(error);
          return false;
        });
    }
  }
```

Gambar 7 Source Code request Resource Serper Penyewaan

```
  }
}
delete(user):Promise<boolean>{
  let url = "penyewaan/api/sewa/delete";
  return this.authHttp.post(url, user).toPromise()
    .then( onfulfilled: sukses=>{return true;})
    .catch( onrejected: error=>{return false;});
}
}
```

Gambar 7 Source Code request Resource Serper Penyewaan (lanjutan)

4. Validasi Token

Reources Server penyewaan akan menerima request dan token yang telah dikirim dari Server Angular 2, Sebelum *Request* di proses oleh *Resources Server*, token akan di melakukan validasi terlebih dahulu dengan mengakses *Authorization Server*. Gambar 8 merupakan Source Code validasi token yang mengarah ke Resources Server

```
security.oauth2.resource.jwt.key-uri=http://localhost:10000/oauth/token_key
```

Gambar 8 Source Code Validasi Token

Hasil dari permintaan tersebut berupa token jika diakses dengan URL GET menggunakan web browser akan tampil seperti gambar 9

Gambar 9 Hasil Response Token

5. JSON Object

Resource Server akan mengembalikan *response* sesuai permintaan jika *token* yang dikirim benar. Gambar 10 merupakan *Source coderesponse* yang dapat diberikan oleh *Resource Server,* Source Code ini di tulis menggunakan java dengan Spring Framework. Pada baris 3 sampai baris ke 12 menunjukan paket dan library yang diperlukan untuk melakukan proses akses API, baris 18 adalah URL yang dapat diakses sources pada gambar ini, untuk mengakses basis data berada pada baris 22, baris 25 dan 26 untuk mengakses seluruh data pada tabel, untuk menyimpan data berada pada baris 30 sampai 33 dan untuk menghapus berada pada baris 37 sampai 40

```
1  package com.gmail.at.agussuhardii.skripsi.api.penjualan.controller;
2
3  import com.gmail.at.agussuhardii.skripsi.api.penjualan.dao.BarangDao;
4  import com.gmail.at.agussuhardii.skripsi.api.penjualan.entity.BarangEntity;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.data.domain.Page;
7  import org.springframework.data.domain.Pageable;
8  import org.springframework.http.HttpStatus;
9  import org.springframework.security.access.prepost.PreAuthorize;
10 import org.springframework.web.bind.annotation.*;
11
12 import javax.validation.Valid;
13
14 /**
15  * Created by Agus Suhardi on 5/2/2017.
16  */
17 @RestController
18 @RequestMapping("penjualan/api/barang")
19 public class BarangController {
20
21     @Autowired
22     private BarangDao barangDao;
23
```

Gambar 10 Source Code Response Authoruzation Penjualan

```
20
21     @Autowired
22     private BarangDao barangDao;
23
24     //    @PreAuthorize("hasAnyAuthority('USER_VIEW')")
25     @GetMapping("/")
26     public Page<BarangEntity> getAll(Pageable pageable) { return barangDao.findAll(pageable); }
29
30     @PreAuthorize("hasAnyAuthority('USER_EDIT')")
31     @PostMapping("/save")
32     @ResponseStatus(HttpStatus.CREATED)
33     public void userBaru(@RequestBody @Valid BarangEntity data) { barangDao.save(data); }
36
37     @PreAuthorize("hasAnyAuthority('USER_EDIT')")
38     @PostMapping("delete")
39     @ResponseStatus(HttpStatus.CREATED)
40     public void delete(@RequestBody String kode) { barangDao.delete(kode); }
43 }
```

Gambar 10 Source Code Response Authoruzation Penjualan (Lanjutan)

Resources Server akan mengakses basis data jika fungsi yang di minta adalah fungsi untuk mengakses basis data seperti gambar 11 Merupakan fungsi untuk menyimpan data ke dalam Basi data Resources Server Penjualan.

```
@PreAuthorize("hasAnyAuthority('USER_EDIT')")
```

```
@PostMapping("/save")
@ResponseStatus(HttpStatus.CREATED)
public void userBaru(@RequestBody @Valid BarangEntity data) {
barangDao.save(data);
}
```

Gambar 11 Souces Code Fungsi simpan

Basis data yang di gunakan adalah MariaDB dengan pengaturan akses berada pada application.properties seperti pada gambar 12. akses token pada resources server ini dilakukan pada code baris 9, sedangkan baris lainnya adalah pengaturan untuk basis data

```
1  spring.datasource.url=jdbc:mysql://localhost/skripsi_resources_server_penjualan
2  spring.datasource.username=root
3  spring.datasource.password=
4  spring.jpa.generate-ddl=true
5  spring.jpa.show-sql=true
6  spring.jackson.serialization.indent_output=true
7  server.port=20000
8  #remote token
9  security.oauth2.resource.jwt.key-uri=http://localhost:10000/oauth/token_key
10
```

Gambar 12 Source Code koneksi basis data

KESIMPULAN

Berdasarkan hasil implementasi *Sigle-Sign-On* menggunakan *OAuth 2*, maka dapat diambil kesimpulan sebagai berikut :

1. *Single-Sign-On* menggunakan protokol *OAuth 2* merupakan cara memudahkan pengguna untuk mengakses aplikasi tanpa harus *login* secara berulang.
2. Dengan menggunakan *Single-Sign-On* pengembang dapat dengan mudah menambah atau menghapus *Resources Server* tanpa mengganggu pengguna.
3. Aplikasi akan lebih mudah dikembangkan karena komunikasi server menggunakan *Rest API*

DAFTAR PUSTAKA

Alphy, B. (2016). *Java Crash Course – The Complete Beginner's Course to Learn Java Programming in 21 Clear-Cut Lessons.* CreateSpace Independent Publishing Platform.

Angular. (2017, 8 24). Diambil kembali dari TutorialsPoint: https://www.tutorialspoint.com

Ionita, M. G. (2012). Secure Single Sign-On using CAS and OpenID. *Journal of Mobile*, 160. Retrieved from http://www.jmeds.eu/

James, S. (2007, 12). *Web Single Sign-On Systems*. Retrieved from Computer Science & Engineering: https://cse.wustl.edu

JSON. (2017). Diambil kembali dari Introducing JSON: http://www.json.org/

JWT. (2017, 8 24). Diambil kembali dari JWT: https://jwt.io/introduction/

Kridalukmana, R., & Satoto, K. I. (2014). Implementation of Indirect Single Sign-On Approach to Integrate Web-Based Applications. *IJCSI International Journal of Computer Science Issues*, 21. Diambil kembali dari http://www.ijcsi.org/

MariaDB. (2017, 8 24). Diambil kembali dari MariaDB.org - Supporting continuity and open collaboration: https://mariadb.org/

maven. (2017, 8 24). Diambil kembali dari apache maven project: https://maven.apache.org

nodejs. (2017, 8 24). Diambil kembali dari Node.js: https://nodejs.org/

RESTful. (2017, 8 24). Diambil kembali dari Oracle help center: http://docs.oracle.com

Richer, J., & Sanso, A. (2017). *OAuth 2 in Action.* Manning Publications.

Subhash, B. K., & S.A., P. K. (2016). Different Framework for Single Sign On (SSO). *International Journal of Computer Science and Mobile Computing*, 53. Diambil kembali dari www.ijcsmc.com

Totty, B., & Gourley, D. (2002). *HTTP: The Definitive Guide.* O'Reilly Media.

UML. (2017, 8 24). Diambil kembali dari tutorialsPoint: https://www.tutorialspoint.com

Walls, C. (2014). *Spring in Action, 4th Edition.* Manning Publications.