Implementasi *Single sign on* (SSO) Menggunakan *Representational State Transfer* (REST) dan *Open Authorization* (OAuth 2.0) (Studi kasus: Universitas Muhammadiyah Magelang)

Afwan Ghiffari[1], Purwono Hendradi*[2]
[1,2]Fakultas Teknik, Teknik Informatika, Universitas Muhammadiyah Magelang,
E-mail: [1]afwanfari00@gmail.com , ***[2] purwonohendradi@gmail.com**

ABSTRAK

Penelitian mengenai implementasi Single Sign On (SSO) di Universitas Muhammadiyah Magelang (UNIMMA) dilakukan dengan tujuan untuk mempermudah akses mahasiswa dan staf ke berbagai sistem layanan web menggunakan satu akun tunggal. Tahapan penelitian meliputi identifikasi masalah, studi literatur, analisis kebutuhan, perancangan, pembuatan prototipe, pengujian, dan implementasi. Penelitian ini difokuskan pada mahasiswa sebagai pengguna utama dan menggunakan Laravel Passport sebagai Server OAuth*, sementara website e-learning (MOCA) dan website administrasi serta rencana studi (KRS) sebagai Client* OAuth*. Arsitektur REST digunakan untuk aplikasi Frontend dan Backend. Protokol SSO yang digunakan adalah* OAuth *2.0, yang lebih aman dan dapat digunakan untuk berbagai keperluan seperti autentikasi dan otorisasi untuk aplikasi web, mobile, dan API.*

Kata Kunci—Single sign on, REST, OAuth 2.0, Perguruan Tinggi

1. PENDAHULUAN

Single sign on (SSO) adalah metode yang memungkinkan pengguna untuk mengakses berbagai sumber daya dan layanan dengan hanya satu proses otentikasi, sehingga tidak perlu melakukan *login* berulang kali. Hal ini terjadi karena proses *login*/otentikasi SSO menggunakan kredensial pengguna untuk mengotentikasi mereka pada semua layanan aplikasi, setelah terlebih dahulu melakukan otentikasi pada layanan berbasis web yang sudah terintegrasi dengan sistem SSO[1].

Universitas Muhammadiyah Magelang (UNIMMA) merupakan institusi yang menggunakan berbagai macam sistem untuk keperluan akademik diantaranya *website* MOCA (moca.unimma.ac.id) yaitu *website e-learning* yang digunakan untuk pembelajaran daring dan KRS (krsol.unimma.ac.id) yang digunakan untuk administrasi, rencana studi, dan keperluan kuliah lainnya. kedua layanan *website* itu sering digunakan oleh mahasiswa, masing-masing *website* memiliki sistem otorisasi yang berbeda[2].

Sebuah studi membahas penerapan SSO di perguruan tinggi Raharja yang memudahkan mahasiswa dan staf akademik untuk mengakses berbagai layanan internet atau sistem web dengan menggunakan satu akun tunggal dan mendapatkan informasi yang diperlukan. [3]. Sedangkan penelitian lainnya menerapkan metode otentikasi dan otorisasi dengan fasilitas *Remote Authentication Dial-In User Service* (RADIUS) sebagai protokol yang memberi izin akses setelah proses otentikasi [4]. Dalam konteks penerapan SSO menggunakan protokol *Open Authorization* (OAuth), pengguna dapat memberikan otorisasi kepada *Client* untuk mengakses data yang dilindungi yang ada di *Server* dengan memberikan token, tanpa perlu memberikan *Username* dan *password*. Protokol OAuth memungkinkan pengguna untuk memberikan akses ke situs pihak ketiga untuk mengakses informasi yang tersimpan pada penyedia layanan lain tanpa harus membagikan semua hak akses atau seluruh data mereka. [5].

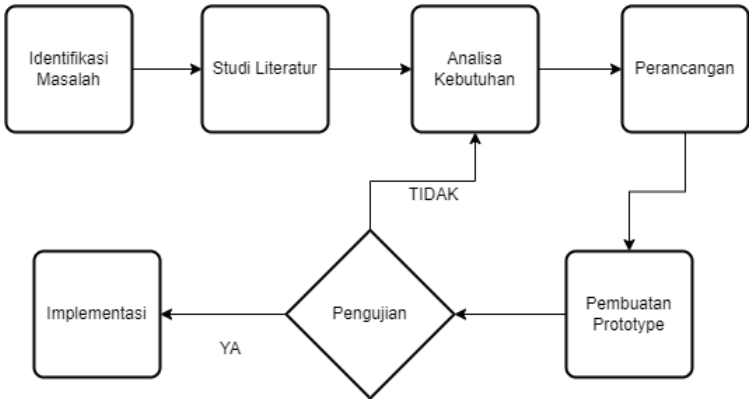
Protokol dan standar digunakan untuk mendukung interoperabilitas aplikasi-aplikasi pada sistem dengan memungkinkan komunikasi antara *Client* dan *Server*. Ada dua protokol yang populer digunakan, yaitu *Representational State Transfer* (REST) dan *Simple Object Access Protocol* (SOAP). REST adalah arsitektur yang digunakan untuk membangun *web services*, sedangkan SOAP adalah protokol yang digunakan untuk bertukar data pada jaringan komputer. [6]

Untuk membangun SSO, beberapa protokol yang umum digunakan antara lain: OAuth 2.0, Protokol ini menyediakan mekanisme untuk aplikasi untuk berbagi informasi autentikasi dan otorisasi tanpa memperlukan *User* untuk memberikan kredensial mereka pada setiap aplikasi.[7] *Security Assertion Markup Language* (SAML), Protokol ini memungkinkan *transfer* data otorisasi antara layanan identitas dan aplikasi. *OpenID Connect* (OIDC), Protokol ini berdasarkan OAuth 2.0 dan menambahkan lapisan identitas pada autentikasi dan otorisasi [8]. Dengan demikian OAuth 2.0 lebih bisa digunakan untuk berbagai *usecase*, seperti autentikasi dan otorisasi untuk aplikasi *web, mobile, dan api*. OAuth 2.0 juga memungkinkan *User* untuk memberikan otorisasi terbatas ke aplikasi lain untuk mengakses data mereka. Ini membantu meningkatkan privasi dan keamanan informasi *User*.

Berdasarkan informasi yang diberikan dalam paragraf tersebut, penelitian selanjutnya akan membahas implementasi SSO dengan menggunakan protokol OAuth 2.0, menggunakan *Laravel Passport* yang dibangun dengan arsitektur REST. Sistem yang akan dirancang menggunakan *Laravel Passport* sebagai *Server* dan arsitektur REST, terdiri dari dua jenis aplikasi, yaitu aplikasi *Backend* yang menggunakan bahasa pemrograman PHP dengan menggunakan *Framework Laravel*, dan aplikasi *Frontend* yang dibangun dengan menggunakan *JavaScript*, *CSS*, dan *HTML*.

2. METODE PENELITIAN

Pada penelitian ini terdapat beberapa tahapan yaitu identifikasi masalah, studi literatur, analisa kebutuhan, Perancangan, Pembuatan Prototype, Pengujian dan Implementasi yang dapat dilihat pada gambar 1 dibawah ini.



Gambar 1. Metodologi Penelitian

2.1. Identifikasi Masalah

Pada tahap pertama penelitian dilakukan proses identifikasi masalah. Berdasarkan informasi tentang objek kebutuhan SSO dapat diidentifikasi menjadi 3 *User* yaitu karyawan, dosen dan mahasiswa. dalam penelitian ini yang menjadi fokus adalah mahasiswa karena merupakan *User* utama dengan jumlah yang banyak. Sedangkan studi literatur dicari dengan menggunakan kata kunci SSO, perguruan tinggi, REST, dan OAuth. rumusan masalah yang harus diselesaikan dalam penelitian ini adalah mengembangkan sistem SSO dengan menggunakan arsitektur REST dan *protocol* OAuth 2.0 untuk mahasiswa unimma

Aplikasi utama untuk mahasiswa di UNIMMA yaitu, aplikasi MOCA dan aplikasi KRS. Kedua aplikasi tersebut berdasarkan fungsinya merupakan aplikasi utama yang selalu diakses oleh mahasiswa. Tujuan dari penelitian ini adalah menggabungkan kedua aplikasi tersebut ke dalam satu sistem *login* untuk mempermudah proses otentikasi.

2.2. Studi Literatur

Pada tahapan ini dilakukan untuk mencari sumber-sumber informasi, konsep-konsep yang mendasar terkait dengan sistem single sign-on, konsep Rest, dan penerapan OAuth 2.0. Materi ini diperoleh dari buku-buku, jurnal, makalah serta artikel-artikel di internet.

Beberapa penerapan dan penelitian tentang SSO sudah pernah dilakukan sebelumnya. Penelitian tentang SSO dengan menggunakan *protocol* OAuth facebook [5]. Penelitian ini melakukan implementasi SSO dengan menggunakan *protocol* OAuth dan memanfaatkan API *login* facebook dan menggunakan php *native*.

Penerapan SSO dengan Google pada *website* menggunakan YII *Framework* telah dilakukan [9]. Penelitian ini menerapkan mekanisme *google sign in* untuk melakukan sign in ke *website* dan tidak mengimplementasikan OAuth 2.0 untuk mengamankan akses ke *Server*.

Penelitian selanjutnya yaitu penerapan SSO di perguruan tinggi juga pernah dilakukan [3]. Pada penelitian ini peneliti hanya menerapkan SSO untuk meningkatkan keamanan serta mempermudah mahasiswa dalam proses *login*. Peneliti juga menggunakan mekanisme google sign in untuk proses autentikasi.

Pada penelitian yang dilakukan oleh Ajie dan kawan-kawan melakukan pengembangan teknologi SSO pada sistem dosen dan sistem informasi kurikulum [10]. Pada penelitian ini penulis menggunakan *Laravel Passport* untuk melakukan integrasi pada sistem dosen dan sistem informasi kurikulum. Metode yang digunakan dalam pengembangan ini adalah metode *prototyping*. Hasil dari penelitian ini adalah berdasarkan pengujian menggunakan *black box* dan dapat disimpulkan bahwa sistem otentikasi berjalan dengan baik

Penelitian yang dilakukan oleh I Kadek tentang implementasi SSO dengan menggunakan REST, Google Identity, dan OAuth 2.0 telah selesai dilakukan. Dalam penelitian ini, pengguna melakukan sign in melalui sistem SSO dengan memasukkan nama pengguna dan kata sandi mereka untuk otentikasi dan otorisasi akses ke aplikasi web. Aplikasi ini menggunakan model *Client-Server* yang dibangun dengan arsitektur REST, Aplikasi ini berkomunikasi melalui antarmuka pemrograman aplikasi (API), di mana token digunakan untuk mengotorisasi pertukaran informasi antara *Backend* dan *Frontend*.[11]

Dengan mempertimbangkan literatur diatas, penelitian SSO akan dibangun dengan menggunakan *protocol* OAuth2.0 pada *Laravel Passport* serta menggunakan arsitektur REST sebagai *Client-Server*. Berdasarkan kajian pada penelitian sebelumnya, penggunaan *Framework* untuk membangun sebuah sistem informasi sangat membantu mempercepat proses implementasi desain pada aplikasi. Dan pada rancangan sistem yang akan dibuat teknologi yang akan digunakan adalah *Laravel Passport* sebagai model OAuth *Server* dan OAuth *Client* serta arsitektur REST yang terdiri dari *Backend* dan *Frontend*.

2.3. Analisa Kebutuhan

Analisa dilakukan setelah data yang dikumpulkan telah lengkap, analisa ini menjabarkan beberapa data pendukung serta membahas dan menyelesaikan permasalahan- permasalahan yang akan diterapkan untuk membangun sistem. Adapun analisa yang dilakukan diantaranya yaitu:

2.3.1. Analisa web service

Sistem SSO dirancang dengan menggunakan model *Client-Server*. Sebuah *Client* merupakan program yang membangun koneksi ke *Server* yang bertujuan untuk mengirimkan satu atau lebih request. Sedangkan *Server* merupakan program yang menerima koneksi-koneksi agar dapat melayani request dengan cara mengirimkan response. Sistem ini juga dibangun dengan menggunakan arsitektur REST yang digunakan untuk mengimplementasikan web service dalam menerapkan konsep perpindahan antar *state*. Pada kasus ini Moca dan Krs adalah web service yang terdapat di Unimma yang akan menjadi Client sistem SSO

2.3.2. Laravel Passport

Laravel adalah sebuah *Framework* php dan dibangun dengan konsep *model view controller* (MVC) yang digunakan untuk pengembangan *website*. *Framework* ini dibangun oleh Taylor otwell pada April 2011[12]. *Laravel Passport* adalah fitur pemrograman yang dibuat oleh *Laravel* untuk membuat aplikasi RESTful API dengan mudah. Ini memfasilitasi pembuatan token OAuth2 dan memudahkan proses autentikasi dan autorisasi pada aplikasi API *Laravel*. *Passport* menyediakan solusi bebas-kendala untuk autentikasi pembuatan aplikasi API dan memastikan bahwa hanya *User* yang sah yang dapat mengakses data pada aplikasi Anda[13]

2.3.3. OAuth 2.0

Protokol OAuth 2.0 merupakan protokol yang banyak digunakan pada SSO dikarenakan kemudahan implementasi dan banyaknya dukungan dari ASP terhadap protokol ini. OAuth merupakan standar terbuka untuk otorisasi yang mengizinkan aplikasi pihak ketiga untuk mengakses resource pada *resource Server* tanpa perlu membagikan kredensial *User* [14]. Protokol OAuth 2.0 memiliki beberapa peran, antara lain *Resource owner, Client, Authorization Server*, dan *Resource Server*. Dalam implementasi SSO menggunakan OAuth 2.0, sebuah aplikasi harus berperan sebagai *Server* OAuth, sementara aplikasi lain harus berperan sebagai *Client* OAuth. *Server* OAuth akan meminta kredensial pengguna dan memberikan token OAuth yang dapat digunakan oleh *Client* OAuth lain untuk mengakses data pengguna terkait. Dengan demikian, pengguna dapat mengakses berbagai aplikasi tanpa perlu melakukan proses autentikasi setiap kali masuk ke aplikasi yang berbeda.[15] adapun penjelasan tiap oauth roles sebagai berikut:

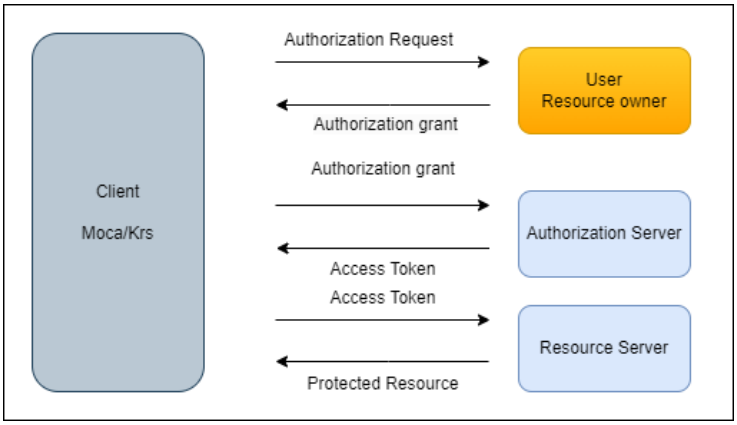
1. Client
   Client adalah aplikasi pihak ketiga yang akan mengakses data dari *Resource Server*. Client dapat berupa Web Apps, Mobile Apps, dll.
2. Resource Owner
   Resource Owner atau user adalah orang yang memiliki resources. Resource owner dapat mengakses data secara langsung ke facebook (contoh) dengan cara login pada facebook.com, namun dalam konteks OAuth bukan akses secara langsung yang dimaksud, tetapi secara tidak langsung dimana user mengakses data melalui *Third party apps*
3. Resource Server

Resource server ibarat rumah bagi resources (data) yang mana bertugas melindungi resources. *Resource Owner* dapat mengakses *resources* melalui *Resource Server* yang diakses dengan menggunakan HTTP Protokol dan disediakan dalam bentuk Web API. Web API harus memastikan bahwa hanya user yang ter-*authentication* dan *authorization* yang dapat mengakeses data. Client harus memberikan *access token* pada HTTP Header ketika ingin merequest suatu data. Yang harus pertama kali Web API lakukan di setiap request yang masuk adalah memeriksa validitas *access token* ke *OAuth Server*. Jika *access token* yang dikirim valid maka client berhak mendapatkan data.

4. Authorization Server
   Terdapat 3 Komponen di dalam OAuth Server.
   - Authentication Component: Komponen ini yang menyediakan login page yang ditampilkan ke user.
   - Consent Component: Komponen ini akan muncul setelah proses authentication component selesai. Komponen ini akan memunculkan page lain yang berisi informasi persetujuan bahwa *Third party apps* akan mengakses beberapa informasi dari user.
   - Token Management: Komponen ini bertugas untuk menjaga token-token yang sudah diberikan oleh OAuth Provider yang tersimpan pada database. Komponen ini juga bertugas memvalidasi token yang di request oleh client.

2.4. Perancangan

perancangan sistem sso mengacu pada studi literatur dan Analisa kebutuhan di unimma



Gambar.2 Workflow SSO

Untuk menerapkan SSO di Unimma menggunakan *protocol* OAuth 2.0, terdapat dua pilihan alur yang dapat digunakan, yaitu *Authorization Code Grant* dan *Implicit Grant*. Alur *Authorization Code Grant* memerlukan lebih banyak langkah daripada *Implicit Grant*, karena membutuhkan pertukaran kode otorisasi untuk mendapatkan *access token*, namun dapat dikelola secara independen oleh *Client*. Sementara itu, *Implicit Grant* hanya digunakan pada aplikasi berbasis *web* dan memberikan *access token* secara langsung saat *URL redirection* diterima oleh *Client*. Alur *Authorization Code Grant* lebih fleksibel dalam pengelolaan *access token* dan cocok digunakan pada aplikasi *web* dan *mobile* karena memudahkan pengembangan aplikasi di berbagai platform yang berbeda. Gambar 2 merupakan desain alur sign in yang diimplementasikan.

Adapun penjelasan setiap langkahnya sebagai berikut:

1. *User* mengakses *Client* (Moca/Krs)
2. *Client* mengarahkan *User* ke authorization *Server* untuk login
3. *User* memasukkan kredensialnya untuk login
4. Authorization *Server* memverifikasi kredensial *User* dan meminta *auth code* untuk memberikan akses token
5. *User* memberikan otorisasi, dan authorization *Server* mengembalikan auth code ke *Client*
6. *Client* menggunakan auth code untuk meminta token dari resource *Server* apakah code tersebut valid atau tidak
7. *Resource Server* menyimpan token dan menggunakannya untuk mengakses Layanan atau *Client*

Dengan alur ini, token tidak dikirimkan langsung ke *Client*, melainkan diteruskan melalui kode otorisasi. Hal ini membuat alur ini lebih aman daripada alur lain yang ditentukan oleh spesifikasi OAuth 2.0, karena token tidak dapat dilihat oleh pihak manapun yang tidak memiliki akses ke *Client*.

Table 1. keterangan Message body

Definisi	Keterangan
Client id	Client id merupakan identitas dari *Third Party Apps*. Client id didapatkan setelah mendaftarkan *Third Party Apps*
Client Secret	pasangan dari client id, bisa disebut juga sebagai password nya
Access Token	digunakan untuk mengakses Resources menggunakan *HTTP Bearer*.
Redirect uri	Digunakan oleh OAuth Server untuk mengembalikan hasil. Redirect URI yang dikirim *Third Party Apps* ke *OAuth Server* harus sama dengan Redirect URI yang didaftarkan ketika *Third Party Apps* mendapatkan Client Id.
Scope	Scope digunakan untuk mendefinisikan informasi apa saja yang akan diminta. Dalam gambar diatas kita ingin meminta data nama, phone, dan birthday.

P-ISSN: 2089-676X
E-ISSN: 2549-0796

```
15    public function getLogin(Request $request)
16    {
17        $request->session()->put("state", $state =  Str::random(40));
18        $query = http_build_query([
19            "client_id" => config("auth.client_id"),
20            "redirect_uri" => config("auth.callback") ,
21            "response_type" => "code",
22            "scope" => config("auth.scopes"),
23            "state" => $state,
24            "prompt" => true
25        ]);
26        return redirect(config("auth.sso_host") .  "/oauth/authorize?" . $query);
27    }
```

Gambar 1. Request Authcode

Pada Gambar 1 menunjukkan penerapan prinsip *stateless* pada REST Karena setiap request mewajibkan *Client* untuk mengirimkan application state, *Server* tidak perlu mengetahui sama sekali apa yang terjadi pada *Client*. semua informasi yang dibutuhkan *Server* untuk melakukan sign in telah ditransmisikan pada *message body*. Sedangkan *identifikasi resource* dilakukan dengan menggunakan *Host* dan *URI* pada request. Request method GET digunakan karena mekanisme sign in akan melakukan pembuatan access dan refresh token untuk kemudian ditransmisikan kembali kepada *Client* melalui pesan response.

```
28    public function getCallback(Request $request)
29    {
30        $state = $request->session()->pull("state");
31
32        // throw_unless(strlen($state) > 0 && $state == $request->state, InvalidArgumentExcept
33
34        $response = Http::asForm()->post(
35            config("auth.sso_host") .  "/oauth/token",
36            [
37                "grant_type" => "authorization_code",
38                "client_id" => config("auth.client_id"),
39                "client_secret" => config("auth.client_secret"),
40                "redirect_uri" => config("auth.callback") ,
41                "code" => $request->code
42            ]
43        );
44        $request->session()->put($response->json());
45        return redirect(route("sso.connect"));
46    }
```

Gambar 2. Exchange authcode with access token

Gambar 2 merupakan request *Client* ke authorization *Server* untuk menukar authcode dengan access token. Setelah mendapatkan akses code selanjutnya kode tersebut akan ditukarkan dengan access token yang nantinya akan digunakan *Client* untuk mengakses *Server*

P-ISSN: 2089-676X
E-ISSN: 2549-0796

```
47      public function connectUser(Request $request)
48      {
49          $access_token = $request->session()->get("access_token");
50          $response = Http::withHeaders([
51              "Accept" => "application/json",
52              "Authorization" => "Bearer " . $access_token
```

Gambar 3 Token Authorization

Gambar 3 merupakan otorisasi akses token. *Resource Server* akan mengecek apakah access code tersebut valid atau tidak apabila valid *Client* akan dilanjutkan ke halaman aplikasi dan apabila tidak valid maka *Client* tidak akan bisa mengakses *Server* dan kembali ke halaman awal

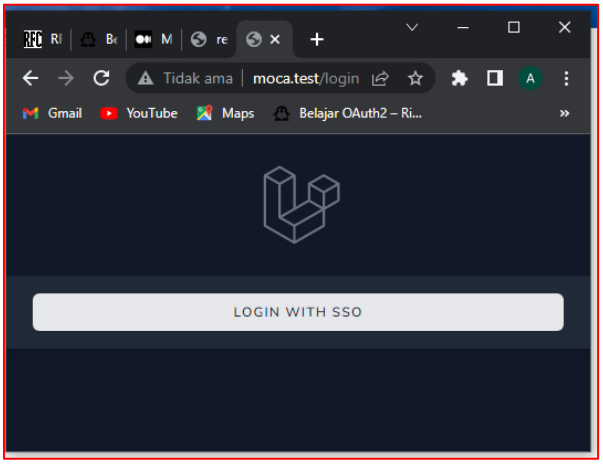
```
:\laragon\www\laravel>php artisan passport:client

Which user ID should the client be assigned to?:
> 1

What should we name the client?:
> MOCA

Where should we redirect the request after authorization? [http://localhost:8000/auth/callback]:
> http://moca.test/oauth/callback

ew client created successfully.
lient ID: 98877aa8-24cf-4760-8d72-79963fbd1799
lient secret: FLTPtmin3skgXiyrEVYpWE9TysBXuV2d0G0r8H2z
```

Gambar 4. Membuat OAuth Client

Gambar 4 merupakah langkah membuat *Oauth Client* pada *laravel passport.* Yaitu mendaftarkan *Client* ke server SSO.

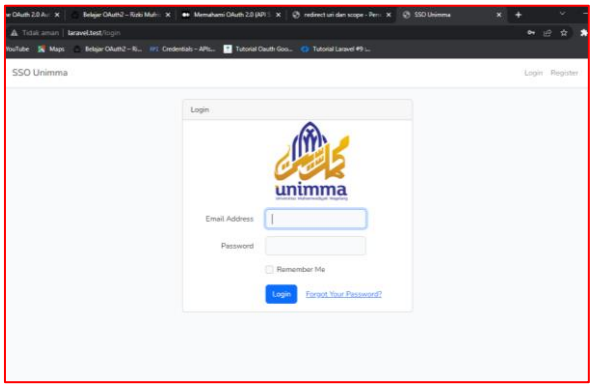
3. HASIL DAN PEMBAHASAN

Mempertimbangkan uraian diatas SSO yang dibutuhkan sesuai dengan permasalahan adalah sso sebagai opsi login untuk mengakses MOCA dan KRS. Berikut adalah gambar

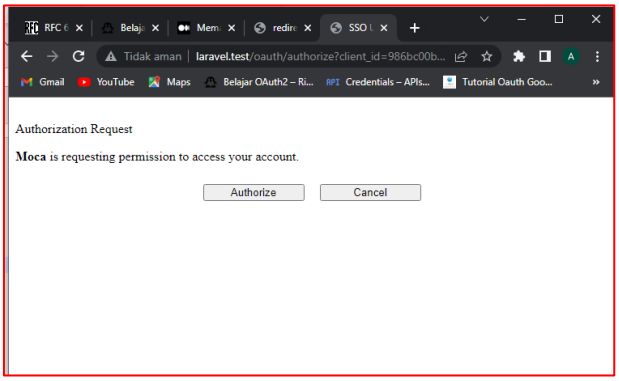


Gambar 5. Antarmuka sign in Moca

P-ISSN: 2089-676X
E-ISSN: 2549-0796

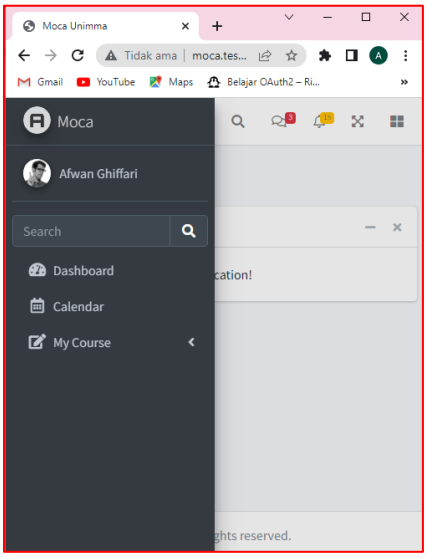


Gambar 6. Tampilan login SSO Oauth Provider



Gambar 7. Authorization Request ke User

Implementasi SSO pada *Frontend web* dilakukan dengan bantuan visual studio code. Antar muka halaman *login* aplikasi web dapat dilihat pada gambar. Pertama saat *User* mengakses Moca dan Krs *User* akan *redirect* ke halaman *sign-in* page. Setelah user menekan tombol login maka akan diarahkan ke halaman SSO Login page seperti pada gambar 6. Selanjutnya *User* harus memasukkan *credential* untuk mengakses aplikasi tersebut. Untuk itu *User* harus mendaftarkan akun terlebih dahulu. Dengan catatan admin sudah mendaftarkan aplikasi moca dan krs ke *Server* SSO. Apabila *User* belum login ke dalam sistem akan menuju ke laman *sign-in*. Tetapi apabila *User* sudah login ke dalam salah satu sistem (moca atau krs) maka hanya akan muncul prompt seperti gambar 7 yang meminta otorisasi. Apabila credential yang dimasukkan *User* valid maka akan mendapatkan *access token* yang nantinya akan digunakan untuk mengakses *Client*.



Gambar 8. Dummy home screen aplikasi Moca

Apabila proses *sign-in* berhasil pengguna akan langsung masuk ke dalam *home screen* seperti yang terlihat pada gambar 8. setiap request yang dikirimkan ke server dilakukan menggunakan protokol HTTPS dengan menyertakan *access token* pada header request yang dikirimkan adalah cara yang umum digunakan untuk memastikan keamanan dan privasi dalam komunikasi antara *client* dan *server*. Dalam protokol HTTPS, setiap permintaan yang dikirimkan antara *client* dan *server* dienkripsi sehingga tidak dapat dibaca oleh pihak lain yang tidak berwenang. Ini mencegah data yang sensitif seperti *access token* dari dicuri atau dimanipulasi oleh pihak yang tidak sah. Sementara itu, menyertakan *access token* pada header request juga penting untuk memastikan bahwa hanya pengguna yang sah yang memiliki akses ke sumber daya yang dilindungi. *Access token* biasanya diberikan kepada pengguna setelah mereka berhasil mengotentikasi dan diotorisasi oleh server. Dengan menyertakan *access token* pada setiap request yang dikirimkan, *server* dapat memverifikasi bahwa pengguna memiliki hak akses yang sesuai sebelum memberikan akses ke sumber daya yang diminta.

4. KESIMPULAN

Implementasi SSO dengan menggunakan protokol OAuth 2.0 berhasil dilakukan pada sistem Moca dan KRS dengan menggunakan *Laravel passport* dan arsitektur REST. Implementasi SSO pada aplikasi *client* mempermudah proses akses pengguna ke sistem karena pengguna tidak perlu melakukan sign in berkali-kali setiap akan menggunakan aplikasi. Dengan hasil penelitian yang diselesaikan, Kesimpulan yang dapat diambil dari penelitian ini adalah sebagai berikut:

- Dalam penelitian ini sistem yang sudah ada bisa mengintegrasikan dan dapat menjawab permasalahan di Unimma
- Pengelolaan menjadi terpusat dalam sebuah SSO Server. *User* tidak perlu login berulang ke setiap *Client* cukup dengan satu akun login

- Dengan memisahkan antara aplikasi dengan SSO Server, maka ketika ada aplikasi baru cukup menambahkan *Oauth Client* baru dan menghubungkannya dengan *Oauth Server*

DAFTAR PUSTAKA

[1] I. P. A. E. D. Udayana, "Integrasi Sistem Single Sign On Pada Sistem Informasi Akademik, Web Information System Dan Learning Management System Berbasis Central Authentication Service," *J. Resist. (Rekayasa Sist. Komputer)*, vol. 1, no. 1, pp. 48–54, 2018, doi: 10.31598/jurnalresistor.v1i1.265.

[2] S. D. Kurniawan *et al.*, "Implementasi Sistem Informasi Pelayanan Akademik (SIPA) Berbasis Web di Politeknik Negeri Ketapang," pp. 752–761.

[3] I. Handayani STMIK Raharja, E. Febriyanto STMIK Raharja, and F. Andhara Khumaida STMIK Raharja, "SATIN-Sains dan Teknologi Informasi SSO (Single Sign On) pada Sistem PESSTA+ Berbasis Yii Framework di Perguruan Tinggi," 2018. [Online]. Available: http://jurnal.stmik-amik-riau.ac.id

[4] T. D. PUTRI, W. SUGENG, and R. KATRI, "Sistem Otentikasi Login Dengan Single Sign-On Untuk Mengakses Banyak Sistem," *MIND J.*, vol. 4, no. 2, pp. 96–110, 2019, doi: 10.26760/mindjournal.v4i2.17-31.

[5] M. Elsera, "Implementasi Single Sign on Pada Web Menggunakan Protocol Oauth Facebook," *Bul. Utama Tek.*, vol. 16, no. 3, pp. 179–185, 2021.

[6] R. Rizal and A. Rahmatulloh, "Restful Web Service Untuk Integrasi Sistem Akademik Dan Perpustakaan Universitas Perjuangan," *J. Ilm. Inform.*, vol. 7, no. 01, p. 54, 2019, doi: 10.33884/jif.v7i01.1004.

[7] T. Waluyo and Sutarman, "Comparative Analysis of the Performance of Single Sign-On Authentication Systems with OpenID and OAuth Protocols," *Int. J. Comput. Inf. Technol.*, vol. 11, no. 3, pp. 100–107, 2022, doi: 10.24203/ijcit.v11i3.277.

[8] W. Li, C. J. Mitchell, and T. Chen, "Oauthguard: Protecting user security and privacy with Oauth 2.0 and Openid connect," *Proc. ACM Conf. Comput. Commun. Secur.*, no. 2, pp. 35–44, 2019, doi: 10.1145/3338500.3360331.

[9] Q. Aini *et al.*, "Penerapan Single Sign On dengan Google pada Website berbasis YII Framework Application Single Sign On with Google the Website Based on Yii Framework," 2018.

[10] H. Ajie, M. Insan Rizky, and M. F. Duskarnaen, "Pengembangan Teknologi Single Sign On Pada Sistem Informasi Dosen dan Sistem Informasi Kurikulum di Universitas Negeri Jakarta," *PINTER J. Pendidik. Tek. Inform. dan Komput.*, vol. 3, no. 1, pp. 32–37, 2019, doi: 10.21009/pinter.3.1.6.

[11] I. K. D. Senapartha, "Implementasi Single Sign-On Menggunakan Google Identity, REST dan OAuth 2.0 Berbasis Scrum," *J. Tek. Inform. dan Sist. Inf.*, vol. 7, no. 2, pp. 307–320, 2021, doi: 10.28932/jutisi.v7i2.3437.

[12] Y. Y. dan H. A. Prasetyo, *Mudah Menguasai Framework Laravel*. 2019.

[13] R. Choirudin, A. Adil, P. Studi, and T. Informatika, "IMPLEMENTASI REST API WEB SERVICE DALAM MEMBANGUN APLIKASI," vol. 18, no. 2, pp. 284–293, 2019.

[14] J. Khan *et al.*, "An Authentication Technique Based on Oauth 2.0 Protocol for Internet of Things (IoT) Network," *2018 15th Int. Comput. Conf. Wavelet Act. Media Technol. Inf. Process. ICCWAMTIP 2018*, pp. 160–165, 2019, doi: 10.1109/ICCWAMTIP.2018.8632587.

[15] B. Martin and A. Quimatio, "Continuous Single-Sign-On (CSSO) method for authentication and authorization in microservices architectures," 2022.