APLIKASI PENCATATAN INVENTORI BERBASIS WEBSITE DENGAN SKEMA AUTENTIKASI DAN OTORISASI STATELESS SEDERHANA

Oleh
I Gusti Ngurah Ady Kusuma
Program Studi Sistem Komputer, Institut Teknologi dan Bisnis STIKOM Bali; Denpasar, Bali.
Email: ady_kusuma@stikom-bali.ac.id

Abstrak

Internet merupakan kebutuhan utama saat ini dalam mendapatkan informasi hingga melakukan manajemen proses bisnis. Hal ini dimungkinkan karena adanya perkembangan pesat pada konten internet seperti website. Website saat ini sudah sangat dinamis dan sebagaian besar telah digunakan sebagai aplikasi manajemen perusahaan seperti perusahaan dagang untuk mencatat inventori. Website awalnya dibangun menggunakan konsep monolitik yang memiliki kekurangan pada sisi pemeliharaan sehingga beralih ke konsep microservice. Namun hal ini mendatangkan permasalahan baru pada proses autentikasi dan otorisasinya. Penelitian ini membahas bagaimana mengadopsi autentikasi dan otorisasi pada konsep monolitik ke microservice dengan mempertahankan sifat stateless pada protokol HTTP. Metode yang digunakan adalah dengan membangun sebuah authority server dan memanfaatkan REST API dalam komunikasi antar layanan. Penelitian ini menunjukan bahwa konsep sederhana ini dapat diimplementasikan dan berdampak pada peningkatan jumlah data terkirim dan adanya penambahan waktu pemrosesan, namun tidak signifikan, masih kurang dari 35ms.

Kata Kunci: Microservice, REST API, Stateless HTTP, Website, Autentikasi dan Otorisasi

PENDAHULUAN

Akses internet menjadi hal yang sangat dibutuhkan dalam hampir semua aspek kehidupan saat ini. Tentunya kebutuhan ini mengakibatkan meningkatnya penggunaan internet setiap harinya [1]. Jika ditelusuri, hampir semua aplikasi atau layanan digital saat ini memerlukan internet agar dapat digunakan. Baik dalam bidang transportasi, hiburan, hingga pengelolaan bisnis memerlukan akses internet agar dapat berjalan optimal. Pengguna layanan untuk dapat menggunakan layanan tersebut memerlukan perangkat tambahan pada perangkat pintar mereka, seperti aplikasi yang harus diunduh yang bersifat *hybird* atau hanya sekedar aplikasi *browser* yang bisa digunakan untuk mengakses sebuah laman *website*.

Website merupakan sekumpulan informasi yang dituangkan ke bentuk digital dan dimuat dalam beberapa halaman yang dimana informasi yang digunakan bersumber dari pemilik website itu sendiri. *Website* merupakan layanan yang bersifat universal dan dapat diakses dengan mudah tanpa perlunya spesifikasi khusus dalam mengaksesnya. Pengguna cukup menggunakan aplikasi *browser* pada perangkat yang dimiliki, seperti Chrome, Firefox dan sebagainya, untuk dapat mengakses informasi secara cepat. Keuntungan tersebut merupakan hal yang baik, karena hanya berbekal akses internet dan perangkat pintar yang memiliki aplikasi *browser* didalamnya, pengguna dapat mengakses informasi secara cepat dan instan [2].

Sebelum munculnya *website* dengan kemampuan yang dinamis dalam menghadirkan informasi, *website* dibangun dengan menggunakan konsep statis. Konsep statis yang dimaksud adalah, pemilik website membangun *website*-nya dengan membuat dokumen Hypertext Markup Language (HTML) kemudian mengunggah dokumen

tersebut ke internet. Jika dikemudian hari pemilik ingin melakukan perubahan pada *website*-nya, maka pemilik web harus melakukan perubahan dengan mengubah kode HTML yang dituliskan sebelumnya, dan mengunggahnya kembali [3]. Hal ini cukup memberatkan karena web menjadi susah untuk diperbarui. Namun saat ini konsep tersebut telah berubah menjadi dinamis. Dinamis yang dimaksudkan adalah bahwa pemilik tidak perlu mengganti atau mengunggah ulang dokumen HTML, namun cukup mengganti bagian informasi yang akan diperbarui pada basis data (*database*) yang digunakan. Perubahan ini juga bersifat langsung, karena web yang bersifat dinamis menampilkan informasi berdasarkan data yang disimpan di basisdata. Kemampuan ini memberikan dampak positif karena menjadikan web dapat dioperasikan seperti aplikasi *desktop* seperti pada penelitian [4].

Seiring dengan penggunaan website yang tidak hanya digunakan sebagai sumber informasi digital, melainkan juga digunakan sebagai alat untuk membantu proses bisnis yang tentunya membutuhkan keamanan untuk melindungi data didalamnya, maka website memerlukan skema autentikasi. Autentikasi adalah sebuah proses yang bertujuan untuk memastikan bahwa seseorang yang sedang menggunakan website adalah orang yang sesuai terdaftar pada sistem [5]. Autentikasi umumnya dilakukan dengan meminta sesorang untuk memasukan informasi mengenai dirinya (berupa nama pengguna atau *username*) beserta kata sandia atau *password* yang hanya diketahui oleh pemilik akun. Ketita proses validasi berhasil dilakukan, maka selanjutnya aplikasi akan mengijinkan orang tersebut untuk mengakses informasi.

Kenyataannya adalah, sebuah proses bisnis sebuah usaha bisa saja memiliki banyak petugas di dalamnya yang turut mengakses aplikasi. Misalkan pada sebuah perusahaan bisa terdiri dari kasir, kepala gudang, akuntan dan pemilik usaha, yang dimana masing-masing pengguna tersebut memiliki hak akses yang berbeda. Untuk mengatasi permasalahan ini, aplikasi membutuhkan sebuah mekanisme otorisasi. Otorisasi merupakan sebuah mekanisme untuk memastikan bahwa seseorang pengguna aplikasi hanya dapat mengakses fitur aplikasi secara terbatas [6]. Ini bertujuan agar antar pengguna tidak dapat melakukan aksi yang berada diluar kawasannya. Seperti kasir tidak dapat mengubah angka penggajian yang diatur oleh akuntan, ataupun kepala gudang tidak dapat melakukan penjualan untuk menghindari *fraud*.

Secara umum, sebuah aplikasi berbasis website dibangun dengan menggunakan arsitektur monolitik. Arsitektur ini mengusung semua fitur maupun proses bisnis dikerjakan dan dilaksanakan oleh sebuah proses. Itu berarti arsitektur monolitik merupakan arsitektur yang menggabungkan seluruh komponen atau fungsionalitas aplikasi ke dalam satu proses yang tidak terpisahkan [7]. Secara proses inisialisasi aplikasi, arsitektur ini memberikan kesederhanaan dalam perancangannya sehingga aplikasi dapat dibangun dengan waktu yang lebih singkat. Hal ini karena pengembang aplikasi hanya perlu fokus membangun satu aplikasi yang nantinya akan menangani semua proses bisnisnya. Setiap fitur ataupun fungsionalitas aplikasi diakses terpusat pada satu aplikasi.

Untuk dapat mengimplementasikan proses autentikasi dan otorisasi pada sebuah website dengan arsitektur monolitik cukup sederhana. Pertama pengguna akan diminta untuk melakukan validasi autentikasi dengan menginputkan *username* dan *password*. Setelah berhasil tervalidasi, pengguna akan diberikan sebuah kode rahasia yang dipasangkan pada *browser* menggunakan *cookies* dan juga menyimpan kode rahasia ini pada sisi aplikasi dan disebut sebagai *session key* [8]. Sehingga setiap pengguna melakukan request kepada website, kode rahasia tersebut akan selalu terkirim, dan aplikasi akan memastikan kode tersebut benar dan cocok terhadap *session key* yang disimpan. Jika *session key* benar dan

terdaftar, maka aplikasi akan memberikan layanan-layanan apa saja yang berhak diakses serta memastikan pengguna hanya mengakses layanan tersebut yang merupakan mekanisme dari otorisasi.

Pada arsitektur monolitik seluruh *request* dari semua pengguna akan ditangani oleh satu aplikasi dan sangat bergantung pada performa aplikasi tersebut. Ketika sebuah *request* diterima oleh aplikasi, maka aplikasi akan membuka aplikasi secara keseluruhan, meskipun mungkin hanya sebagian kode yang dibutuhkan. Ini tentunya berakibat pada tingginya beban aplikasi, dan berpotensi menggangu fungsionalitas lainnya ketika pengguna aplikasi meningkat.

Selain itu, arsitektur dari monolitik ini memiliki kelemahan pada sisi pemeliharaan. Ketika terjadi permasalahan pada aplikasi, atau hanya ingin sekedar melakukan pembaruan aplikasi, maka penyedia layanan harus memadamkan semua layanan atau fungsionalitas aplikasi. Itu karena, semua fungsionalitas berada pada satu aplikasi, sehingga untuk memperbarui sebuah fungsionalitas, maka fungsionalitas lainnya ikut terganggu.

Berdasarkan permasalahan tersebut, maka munculah sebuah arsitektur aplikasi yang baru yang disebut dengan *microservices*. Konsep dari arsitektir ini adalah memisahkan sebuah layanan (*service*) aplikasi menjadi bagian kecil (*micro*) dan terpisah antar fungsionalitas ataupun komponennya [9]. Hal ini bertujuan untuk mengurangi beban aplikasi yang hanya ditopang oleh satu proses. Selain itu ketika melakukan pemeliharaan atau pembaruan aplikasi, maka cukup salah satu dari fungsionalitas aplikasi saja dipadamkan. Sehingga fungsionalitas lainnya tetap berjalan.

Ketika menggunakan arsitektur microservices, pengambang aplikasi dapat melakukan pemecahan aplikasi ke dalam bentuk-bentuk yang lebih kecil. Masing-masing aplikasi kecil ini nantinya berdiri secara mandiri dan memiliki fungsionalitasnya sendiri. Hal ini menjadikan aplikasi lebih mudah dalam pengembangan dan pemeliharaan. Tentunya karena masing-masing fungsionalitas berdiri secara mandiri, maka dibutuhkan sebuah protokol untuk dapat menjembatani komunikasi antar komponen.

Komunikasi antar komponen aplikasi merupakan hal wajib yang diperlukan dalam arsitektur microservice. Selain itu komunikasi ini dibutuhkan juga kemampuannya untuk dapat menghubungkan aplikasi yang multiplatform. Hal ini diperlukan karena masing-masing komponen aplikasi dapat dibangun di lingkungan sistem operasi yang berbeda, atau mungkin menggunakan *compiler* yang berbeda juga. Untuk dapat mengakomodir hal tersebut, arsitektur microservice lebih banyak mengandalkan protokol Hypertext Transfer Protocol atau HTTP. HTTP berbasis teks dalam pengiriman datanya sehingga lebih universal dapat digunakan pada multiplatform aplikasi.

Salah satu metode yang dapat digunakan untuk berkomunikasi antar komponen dengan berlandaskan protokol HTTP adalah metode Representational State Transfer API atau dikenal dengan nama REST API. REST API merupakan konsep dari pertukaran data yang berbasis teks dengan menyertakan *state* dari setiap komunikasi yang dijalin [10]. REST API tidak hanya mendukung komunikasi antar pengguna dengan layanan, namun juga antar layanan, fungsionalitas atau komponen pun dapat berkomunikasi tanpa adanya campur tangan dari developer. Dengan hal ini, komponen pendukung dapat saling bekerjasama meskipun komponennya diletakan di lokasi server yang berbeda.

Mayoritas komunikasi yang dibangun diatas protokol HTTP merupakan metode yang bersifat *stateless* dengan memanfaatkan skema *request* dan *response* [11]. Dikarenakan konsep dari *stateless* yang dimana semua *request* yang dikrimkan merupakan *request* yang berdirisendiri tanpa mengetahui *request* sebelumnya. Hal ini juga termasuk, aplikasi

juga tidak dapat mengetahui apakah *request* tersebut sudah terautentikasi sebelumnya atau tidak. Pada arsitektur monolitik, pengembang aplikasi dapat memanfaatkan *session key* yang tertanam pada *cookies browser* pengguna dan mencocokan dengan *session key* yang tersimpan di memori server. Namun pada arsitekur microservice hal ini tidak dapat langsung diadopsi karena komponen lainnya tidak dapat mengakses memori dari komponen yang menangani autentikasi.

Berdasarkan hal tersebut, diperlukan sebuah metode yang dapat digunakan untuk dapat mengimplementasikan arsitektur microservice dengan menggunakan konsep autentikasi yang serderhana. Penelitian ini membahas bagaimana cara menerapkan proses autentikasi dan otorisasi tersebut pada arsitektur microservice dengan contoh kasusnya adalah mengakses data inventori barang. Dimana yang dapat mengakses data inventory hanya yang sudah di validasi autentikasinya dan otorisasinya. Harapannya adalah, melalui penelitian ini dapat menggambarkan bagaimana teknik sederhana yang dapat dilakukan.

LANDASAN TEORI

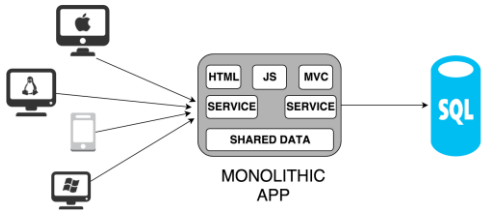
Penelitian ini berlandasakan teori-teori yang sudah ada sebelumnya. Hal ini diperlukan untuk mendukung apa yang dilakukan dalam penelitian ini. Landasan teori yang digunakan dalam penelitian ini adalah sebagai berikut

1. Arsitektur Microservice

Arsitektir pengembangan aplikasi microservice merupakan arsitektur pengembangan aplikasi yang memecah fungsionalitas program ke dalam aplikasi yang mandiri dan berjalan sendiri [9]. Arsitektur ini merupakan arsitektur pengembangan aplikasi monolitik.

Monolitik merupakan arsitektur yang dibangun dengan menggabungkan seluruh komponen atau fungsionalitas aplikasi ke dalam satu aplikasi besar dengan satu proses utama yang berjalan [7]. Hal ini mengakibatkan saat pengguna aplikasi mengakses sebuah fungsionalitas aplikasi, maka keseluruhan aplikasi akan di unggah terlebih dahulu ke memori sebelum nantinya pengguna dapat memiliki fungsionalitas yang akan digunakan. Tentunya cara ini akan membebankan komputer dalam menjalankan aplikasinya. Hal ini juga memberikan dampak saat adanya pemeliharaan aplikasi atau pembaruan aplikasi, maka aplikasi secara keseluruhan harus dihentikan prosesnya, kemudian diganti dengan aplikasi yang baru. Jika aplikasi harus terus berjalan setiap saat karena transaksi terjadi secara terus menerus, maka pembaruan aplikasi ataupun pemeliharaannya akan menghentikan proses bisnis dan memungkinkan adanya kerugian yang harus ditanggung. Gambar 1 merupakan skema arsitektur monolitik.

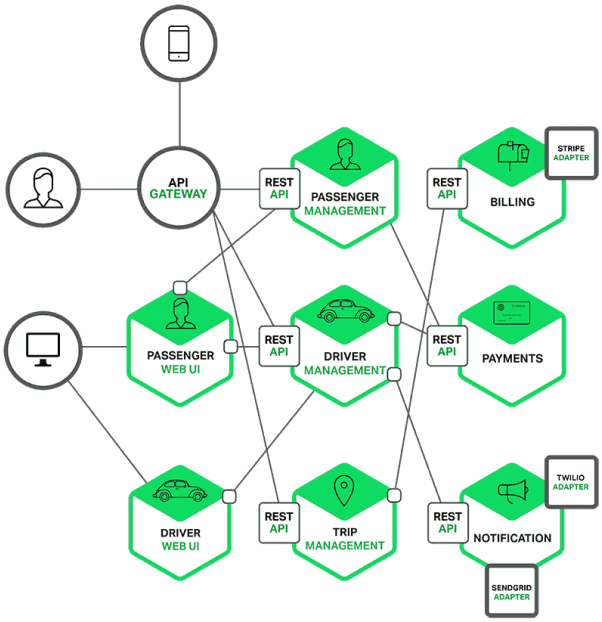
Gambar 1. Arsitektur Monolitik



Sumber gambar dari penelitian [6]

Arsitektur microservice merupakan pengembangan dari arsitektur monolitik. Arsitektur ini, komponen penyusun aplikasi dipisahkan kedalam aplikasi yang berbeda, namun tetap salig terhubung dan dapat bertukar data yang bertujuan untuk menjalin Kerjasama antar komponen [9]. Arsitektur ini dapat memberikan kelebihan pada saat melakukan pemeliharaan atau pembaruan aplikasi. Saat hal tersebut diperlukan, maka cukup dengan menonaktifkan salah satu komponen saja sehingga tidak menghentikan aplikasi secara keseluruhan. Tentunya dengan hal ini, proses bisnis lainnya tidak akan terganggu. Selain itu jika salah satu komponen mengalami kegagalan, tidak akan berimbas pada komponen lainnya. Gambar 2 merupakan arsitektur dari microservice.

Gambar 2. Arsitektur Microservice



Sumber gambar dari penelitian [6]

2. Hypertext Transfer Protocol (HTTP)

HTTP merupakan sebuah protokol pertukaran data dengan berbasiskan pesan data dalam bentuk teks, yang dimana protokol ini merupakan dasar komunikasi saat mengakses sebuah website [11]. Dalam implementasinya, protokol HTTP memiliki dua bagian dalam mengatur pesan yang dikirimkan, yaitu *header* dan *message body*. Bagian *header* merupakan dimana seluruh informasi mengenai *request* atau *response* dicantumkan, seperti metode akses GET, POST dan lainnya, status komunikasi dan sebagainya. Kemudian pada *message body* merupakan data utama yang dikirimkan. Gambar 3 merupakan data pada arsitektur HTTP. Berkat kesederhanaan dari bentuk data protokol HTTP, protokol ini digunakan oleh komunikasi layanan website dan menjadikan layanan website dapat diakses dari perangkat apasaja selama memiliki akses internet pada perangkat tersebut. Selain itu perangkat tersebut juga wajib memiliki aplikasi browser untuk dapat membuka layanan dari laman website yang diakses tersebut.

Gambar 3. Skema Data pada Protokol HTTP

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie:
JSESSIONID=12CBA24A990C14A0133E4E3AF8FB3C66;
Path=/day8_1; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 623
Date: Sat, 22 Jul 2017 08:03:25 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
  <head>
    <base
href="http://localhost:8080/day8_1/">

    <title>My JSP 'index.jsp' starting
page</title>
<meta http-equiv="pragma" content="no-cache">
....
```

Sumber gambar dari penelitian [6]

3. Representational State Transfer (REST) API

REST API merupakan pengembangan dari metode pengiriman data menggunakan HTTP. Pada dasarnya metode ini berlandasakan HTTP, namun dengan memodifikasi *header* pesan [12]. REST API memanfaatkan nilai HTTP *state header* untuk memberikan informasi mengenai operasi yang dilakukan serta memberikan status operasi yang dilakukan.

METODE PENELITIAN

Terdapat beberapa tahap dalam penelitian ini dalam membangun aplikasi pencatatan inventori ini.

1. Perancangan Metode

Tahap ini bertujuan untuk menggambarkan secara umum bagaimana metode autentikasi dan otorisasi dimplementasikan dalam arsitektur microservice tersebut.

2. Implementasi Metode

Tahap ini mewujudakan rancangan dari metode autentikasi dan otorisasi yang sudah dibahas dalam perancangan. Sebagai contoh prosesnya adalah menggunakan skema dimana aplikasi menampilkan informasi data inventori barang kepada penggunaan yang sudah tervalidasi.

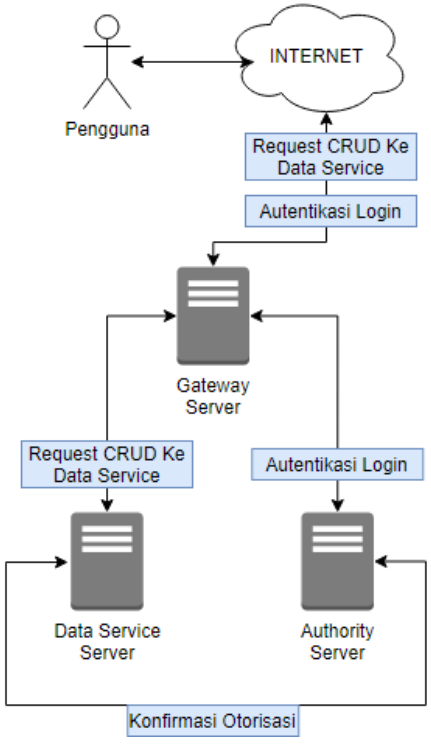
3. Evaluasi

Tahap ini melakukan pengujian dan menganalisis bagaimana cara kerja metode yang dibahas dalam penelitian ini.

HASIL DAN PEMBAHASAN

Pada tahap awal dari penelitian ini adalah merancangan bagaimana arsitektur komponen layanan secara keseluruhan, termasuk bagaimana pengguna dapat mengakses aplikasi. Berdasarkan konsep *stateless* dengan microservice, maka komponen dipecah menjadi tiga bagian yang diasumsikan ditempatkan pada server yang berbeda, dimana nantinya pengguna hanya mengakses satu server *gateway* yang setelahnya akan diteruskan ke komponen yang bersangkutan. Gambar 4 merupakan arsitektur dasar pada penelitian ini.

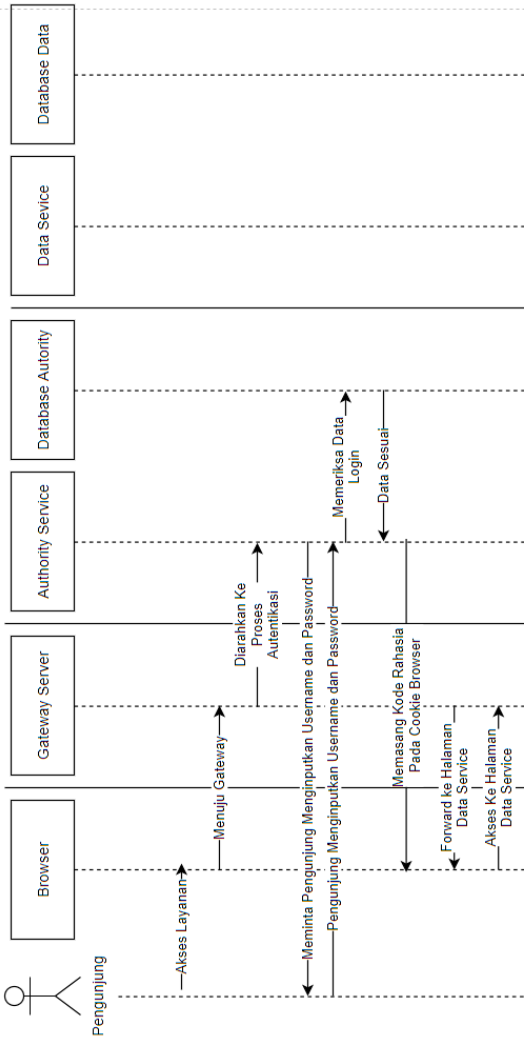
Gambar 4. Arsitektur aplikasi pada penelitian ini



Sumber gambar dari penelitian [6]

Selanjutnya untuk proses autentikasi, pengguna aplikasi nantinya harus menghubungi layanan otoritas (*authority service*) melalui *gateway* dan melakukan validasi terlebih dahulu. Setelah tervalidasi, pengguna akan diberikan kode rahasia yang akan digunakan nanti saat mengakses data ke layanan yang lainnya. Gambar 5 merupakan skema autentikasi yang digunakan.

Gambar 5. Skema Autentikasi

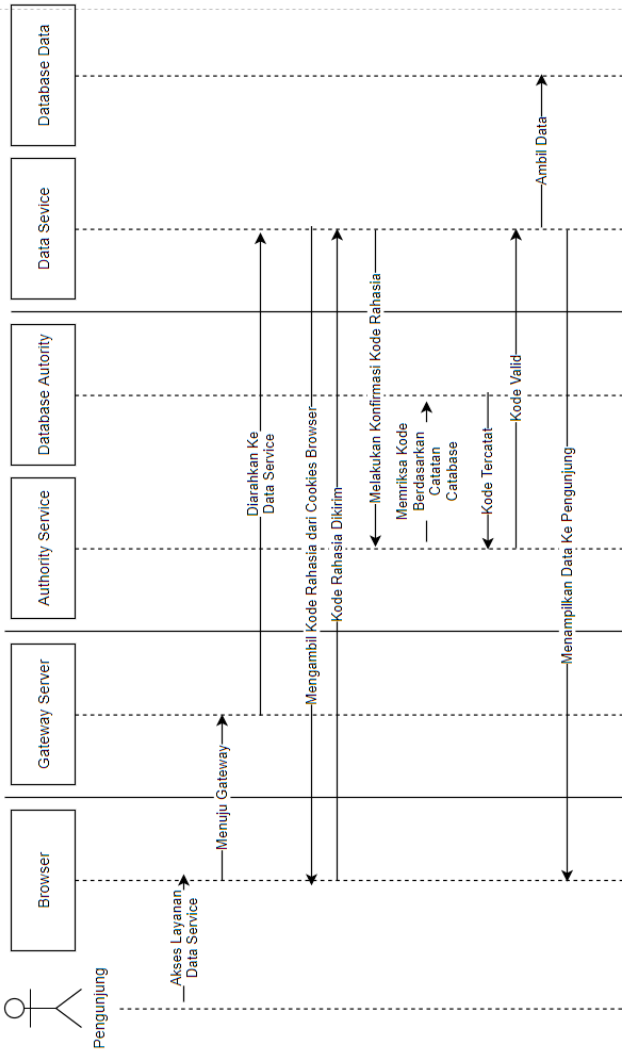


Sumber gambar dari penelitian [6]

Setelah pengguna mendapatkan kode rahasia, kemudian pengguna dapat melakukan akses kepada layanan yang dibutuhkan, dalam penelitian ini dicontohkan mengakses data inventori barang. Akses layanan inventori barang diakses melalui *gateway* dengan menyertakan kode rahasia yang didapat saat autentikasi. Saat server layanan menerima request dari pengguna, maka layanan tersebut akan melakukan konfirmasi terhadap kode rahasia yang disertakan kepada *authority server*

apakah kode tersebut benar atau tidak. Jika benar, maka layanan akan memeberikan data yang dibutuhkan. Gambar 6 merupakan skema dari otorisasi.

Gambar 6. Skema Otorisasi



Sumber gambar dari penelitian [6]

Setelah tahap perancangan, kemudian dilanjutkan dengan tahap implementasi. Dimana pada tahap ini dilakukan penulisan kode program sesuai dengan perancangan yang telah dilakukan. Berikut pseudocode yang digunakan untuk membangun fungsi utama dari penelitian ini, yaitu autentikasi dan otorisasi pada Tabel 1.

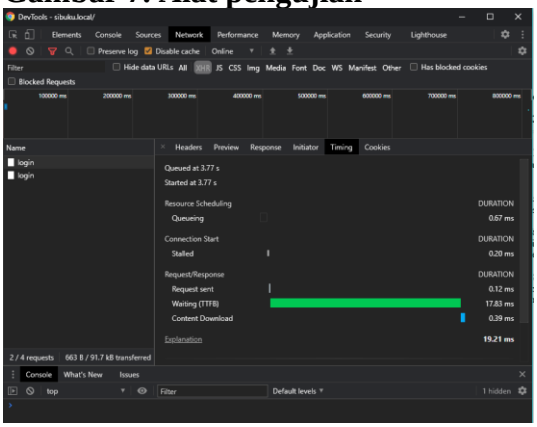
Tabel 1. Pseudocode Aplikasi

```
1  function login($username, $password) {
2    $dbconn = dbConnection();
3  $username = sanitasiVariabel($dbconn,
   $username); $password = hashSHA256
4  ($password);
5  $SQL = "SELECT * FROM tb_login
6  WHERE username = '" . $username . "'
   AND password='" . $password . "'";
     $result = mysqli_query($dbconn,
7  $SQL); if (mysqli_num_rows($result) >
   0) {//Buat KODE RAHASIA $kode
8  rahasia = hashSHA256($username .
9  "SALTKEY");$row = $result-
10 >fetch_array(MYSQLI_NUM); $SQL
11 = "INSERT INTO tb_kode
12 rahasia(login_id, kode rahasia)
   VALUES ('$row[0]','$kode rahasia')";
13     if (mysqli_query($dbconn, $SQL))
   { //Tulis cookie
14       setcookie("kode rahasiaauth",
   $kode rahasia);
         return true;
       } else {return false; }
15   } else {
16     return false;
17   }
18   mysqli_close($dbconn);
19  }
20  function validasiKode rahasia($kode
21 rahasia) {
22   $dbconn = dbConnection();
23   $username =
24 sanitasiVariabel($dbconn, $kode
25 rahasia);
26   $password =
27 hashSHA256($password);
28   $SQL = "SELECT * FROM
29 tb_koderahasia WHERE koderahasia =
30 '$kode rahasia'";
31   $result = mysqli_query($dbconn,
   $SQL);if (mysqli_num_rows($result) >
32 0) {return true;} else {return false;}
33 mysqli_close($dbconn);}
```

Kemudian setelah diimplementasikan, dilakukan pengujian terhadap implementasi

yang telah dilakukan. Pengujian dilakukan dengan mengamati dari besarnya overhead data untuk proses otorisasi dan otentikasi yang dikirimkan dan response waktu yang diperlukan dalam melakukan proses validasi login, validasi kode rahasia dan penampilan data barang. Pengujian dilakukan dengan membaca catatan aktifitas (log file) dari aplikasi dan juga developer tools pada browser. Gambar 7 merupakan gambaran alat pengujian.

Gambar 7. Alat pengujian



Pengujian dilakukan sebanyak 5 kali untuk masing-masing dari fungsi yang diakses. Tabel 2 merupakan rangkuman pengujian yang dilakukan.

Tabel 2. Rangkuman Pengujian

Fungsi Yang Diujikan	Pengujian Ke-	Response time (ms)	Besar Overhead Data Transfer (Byte)
Validasi Login	1	19.21	332
	2	18.57	332
	3	20.8	332
	4	19.23	332
	5	17.25	332
Validasi Kode rahasia	1	18.55	593
	2	17.55	593
	3	18.78	593
	4	18.56	593
	5	18.55	593
Pengambilan Data	1	30.4	1888
	2	30.45	1888
	3	31.24	1888
	4	31.2	1888
	5	30.8	1888

Dari pengujian yang dilakukan terlihat bahwa response time yang diberikan oleh sistem masih tergolong cepat, terutama pada proses fungsi Pengambilan Data. Yang dimana pada proses pengambilan data terjadi komunikasi sebanyak 3 pasang titik, yaitu dari pengguna ke service gateway, service gateway ke service kelola data, dan service kelola data ke service otorisasi. Sehingga fungsi ini sangat wajar jika dilihat dari ketiga fungsi diatas memiliki response time yang lebih lama. Namun meskipun terlama, rata-rata untuk pengambilan data membutuhkan waktu yang cepat yaitu 30,818 ms, masih cukup cepat untuk diaplikasikan.

Dari sisi besarnya overhead data, atau bandwidth data total yang diperlukan ketika menggunakan metode ini bersifat statis dan tidak berubah dari pengujian semua fungsi pengujian. Overhead data pada proses pengambilan data cukup besar yaitu 1888 byte melebihi 1 KiloBytes. Hal ini dikarenakan bandwidth jaringan tidak hanya digunakan untuk keperluan pengiriman data saja, melainkan untuk melakukan validasi kode rahasia antara service. Yang dimana proses validasi kode rahasia dilakukan setiap pengguna melakukan request pengambilan data.

PENUTUP

Kesimpulan

Berdasarkan dari penelitian yang dilakukan dapat disimpulkan bahwa konsep otorisasi dan otentikasi yang diusulkan tidak terlalu menurunkan performa layanan keseluruhan. Kecepatan response layanan masing terjaga dibawah 35 ms, tergolong masih cukup cepat. Kemudian dari sisi overhead data tergolong besar, dan memiliki kemungkinan untuk membebani traffic jaringan karena adanya komunikasi antara service untuk melakukan proses validasi kode rahasia.

Dengan demikian konsep dari otorisasi ini dapat diimplementasikan pada kasus nyata namun dengan mempertimbangkan jumlah service yang tersusun serta frekuensi akses pada layanan tersebut.

Saran

Berdasarkan konsep penelitian yang diusulkan ini diharapkan dapat diperkaya fiturnya dengan menambahkan metode validasi menggunakan *hashing* dengan kunci asimetris, sehingga *request* terhadapt *authority server* bisa lebih dioptimalkan.

DAFTAR PUSTAKA

[1] D. M. Dozier, H. Shen, K. D. Sweetser, and V. Barker, "Demographics and Internet behaviors as predictors of active publics," *Public Relations Review*, vol. 42, no. 1, pp. 82-90, 2016/03/01/ 2016.

[2] M. Mashadi, E. Nurachmad, and M. Mulyana, "Analisis Deskriptif Penilaian Website Perguruan Tinggi," *JAS-PT (Jurnal Analisis Sistem Pendidikan Tinggi Indonesia)*, vol. 3, no. 2, pp. 97-106, 2019.

[3] W. R. Jiang and J. H. Yan, "Implementation of static web-pages generator using JavaScript," in *Applied Mechanics and Materials*, 2011, vol. 39: Trans Tech Publ, pp. 588-591.

[4] I. M. Sudibya, I. M. P. K. P. SS, I. G. Suardika, and I. G. N. A. Kusuma, "Sistem Informasi Pendataan Penduduk Pendatang Kecamatan Tegallalang," *E-JURNAL JUSITI: Jurnal Sistem Informasi dan Teknologi Informasi*, vol. 8, no. 2, pp. 175-185, 2019.

[5] I. Riadi, R. Umar, and I. Busthomi, "Optimasi Keamanan Autentikasi dari man in the middle attack (MiTM) menggunakan teknologi blockchain," *Journal Information Engineering and Educational Technology) ISSN*, vol. 2549, p. 869X, 2020.

[6] I. G. N. A. Kusuma, "PERANCANGAN SIMPLE STATELESS AUTENTIKASI DAN OTORISASI LAYANAN REST-API BERBASIS PROTOKOL HTTP," *Jurnal Manajemen Informatika dan Sistem Informasi*, vol. 4, no. 1, pp. 78-87, 2021.

[7] E. Axelsson and E. Karlkvist, "Extracting Microservices from a Monolithic Application," 2019.

[8] M. A. Faisal, S. J. I. Ismail, and P. Periyad, "Membangun Server Video Conference Menggunakan Jitsi," *eProceedings of Applied Science*, vol. 7, no. 6, 2021.

[9] D. Guaman, L. Yaguachi, C. C. Samanta, J. H. Danilo, and F. Soto, "Performance evaluation in the migration process from a monolithic application to microservices," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 2018: IEEE, pp. 1-8.

[10] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of REST API for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 154-167, 2016.

[11] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen, "Analysis of http requests for anomaly detection of web attacks," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, 2014: IEEE, pp. 406-411.

[12] S. Sohan, F. Maurer, C. Anslow, and M. P. Robillard, "A study of the effectiveness of usage examples in REST API documentation," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2017: IEEE, pp. 53-61.

HALAMAN INI SENGAJA DIKOSONGKAN

ISSN 2798-3471 (Cetak)
ISSN 2798-3641 (Online)