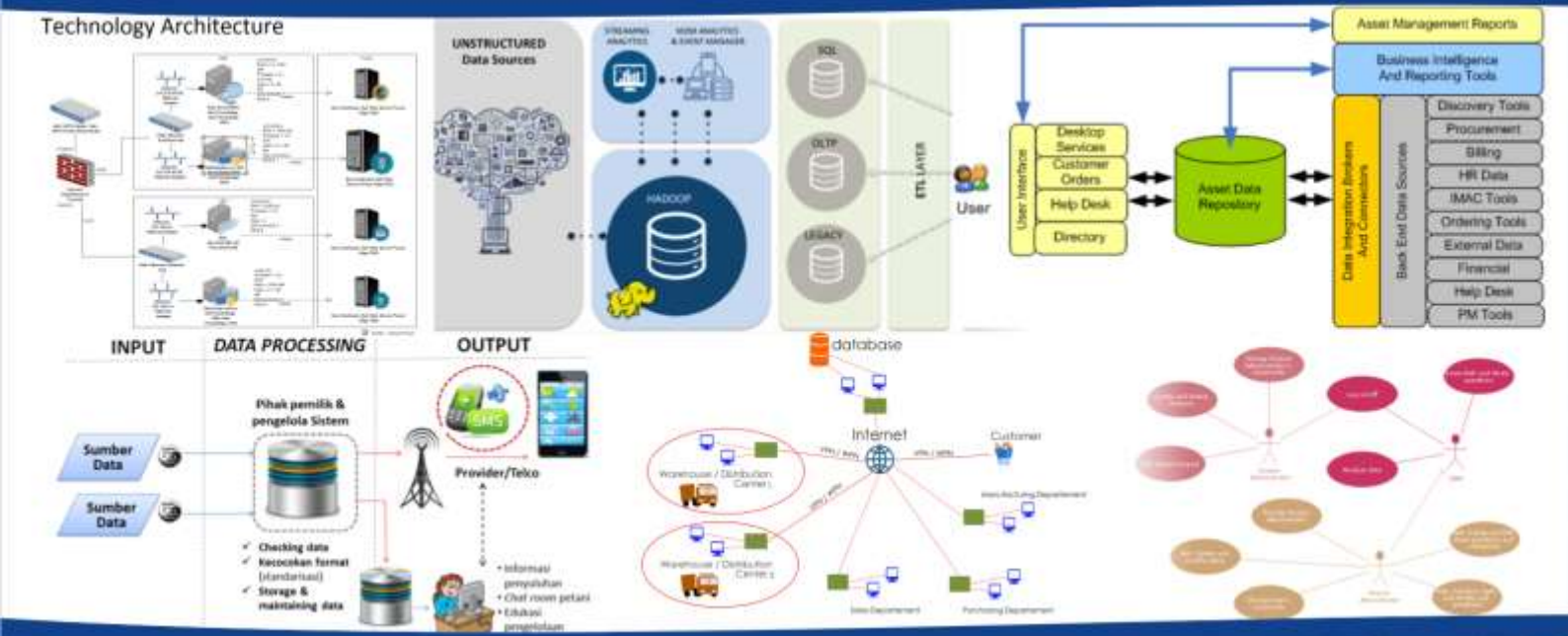
ISSN : 2614-1701 (*Cetak*)
ISSN : 2614-3739 (*Online*)

Volume 4, Nomor 1, Januari 2021

MISI

JURNAL MANAJEMEN INFORMATIKA & SISTEM INFORMASI



Diterbitkan Oleh LPPM STMIK Lombok
Jln. Basuki Rahmat No.105 Praya, Lombok Tengah - NTB
Telp dan Fax (0370) 654310 - e-journal.stmiklombok.ac.id/jsi
email. lppm@stmiklombok.ac.id

DAFTAR ISI

MISI *(Jurnal Manajemen informatika & Sistem Informasi)* Volume 4, No 1, Januari 2021
http://e-journal.stmiklombok.ac.id/index.php/misi

PERANCANGAN SIMPLE STATELESS AUTENTIKASI DAN OTORISASI LAYANAN REST-API BERBASIS PROTOKOL HTTP

I Gusti Ngurah Ady Kusuma

Program Studi Sistem Komputer, Institut Teknologi dan Bisnis STIKOM Bali
Jln. Raya Puputan No. 86 Renon, Denpasar-Bali 80234

ady_kusuma@stikom-bali.ac.id

Abstract

Microservice architecture is a system building architecture that is often used today. The concept is to separate program components/functionality into different applications. Between components using the Hypertext Transfer Protocol (HTTP) protocol to communicate because HTTP supports multiplatform like a website plus the emergence of the REST API scheme. The REST API allows users and services to exchange data between platforms. The problem that arises is that all communications on the HTTP protocol are stateless, where every request received will be executed independently. This includes whether the client has previously been authenticated or not. This will be vulnerable to intruders who have not been authenticated so that the authority cannot be ascertained in accessing a function. Based on these problems, an authentication and authorization scheme is needed that can accommodate a microservice scheme. The method that can be used is to use a unique authorization code that is attached to each service request that is validated by an independent server. The flow of the scheme is described using sequence diagrams and is supported by a conceptual database. This research has produced a simple authentication and authorization method design that can be used as a secure alternative to a microservice architecture.

Keywords : *http, rest-api, stateless, autheticate, authorize*

Abstrak

Arsitektur Microservice merupakan arsitektur pembangunan sistem yang sering digunakan saat ini. Konsepnya adalah memisahkan komponen program / fungsionalitas ke dalam aplikasi yang berbeda. Antar komponen menggunakan protokol Hypertext Transfer Protocol (HTTP) untuk berkomunikasi karena HTTP mendukung multiplatform layaknya sebuah website ditambah lagi munculnya skema REST API. REST API memungkinkan pengguna dan layanan untuk bertukar data antar platform. Permasalahan yang muncul berasal dari semua komunikasi pada protokol HTTP bersifat stateless, yang dimana setiap permintaan yang diterima akan dijalankan secara independen. Termasuk apakah klien sebelumnya sudah diautentikasi atau belum. Hal ini akan rentan terhadap penyusup yang belum terotentikasi sehingga tidak dapat dipastikan otoritasnya dalam mengakses suatu fungsi. Berdasarkan permasalahan tersebut, diperlukan skema otentikasi dan otorisasi yang dapat mengakomodasi skema layanan mikro. Metode yang dapat digunakan adalah dengan pemanfaatan kode otorisasi yang bersifat unik yang ditempelkan pada setiap permintaan layanan yang divalidasi oleh sebuah server independent. Alur dari skemanya digambarkan menggunakan sequence diagram dan didukung dengan konseptual database. Penelitian ini menghasilkan sebuah rancangan metode autentikasi dan otorisasi sederhana yang dapat digunakan sebagai alternatif kemanan pada arsitektur microservice.

Kata kunci : *http, rest-api, stateless, autentikasi, otorisasi*

1. PENDAHULUAN

Kebutuhan akan akses *internet* kian meningkat setiap harinya [1]. Hampir seluruh aplikasi yang digunakan oleh pengguna saat ini memerlukan koneksi *internet* untuk dapat menyediakan layanan yang dibutuhkan oleh pengguna. Layanan seperti media sosial, transportasi hingga penjualan kini lebih memanfaatkan jaringan internet. Pengguna dapat mengakses layanan tersebut dengan menggunakan aplikasi yang bersifat *hybrid* maupun langsung mengakses menggunakan *browser* untuk mengakses *website*. Seperti layanan pengolahan data perkebunan pada penelitian [2] yang menggunakan basis website dalam pembuatannya hingga pengelolaan data yang ditangani oleh *customer relation* seperti pada penelitian [3].

Awal kemunculan aplikasi berbasis *internet* lebih banyak menggunakan arsitektur Monolithic. Arsitektur Monolithic merupakan arsitektur dimana seluruh komponen/fungsionalitas pembangun dari aplikasi tersebut menjadi satu dengan yang lainnya dan tidak terpisah [4]. Arsitektur monolithic ini mengakibatkan ketika mengakses sebuah fungsi pada aplikasi tersebut, maka keseluruhan aplikasi ikut dipanggil sehingga lebih cenderung memberatkan kinerja komputer.

Untuk mengurangi beban pemanggilan aplikasi pada arsitektur Monolithic, muncul konsep arsitektur Microservice. Konsep dari arsitektur microservice sendiri merupakan memisahkan komponen/fungsionalitas program ke dalam aplikasi-aplikasi yang berbeda [5]. Hal ini menjadikan suatu sistem informasi dibangun dari banyak aplikasi yang memiliki fungsionalitas tersendiri. Hal ini memberi keuntungan jika mengakses sebuah fungsi maka kita tidak perlu mengakses fungsi yang lainnya.

Agar dapat bertukar informasi maupun data antar komponen, maka diperlukan skema pertukaran data yang mendukung sifat multiplatform. Hal ini dikarenakan kemungkinan komponen pendukung dari sistem dibangun menggunakan *environment* yang berbeda. Berdasarkan hal tersebut, antar komponen cenderung memanfaatkan *protocol* Hypertext *Transfer Protocol* (HTTP) karena mendukung multiplatform seperti *website* pada umumnya yang dapat diakses menggunakan semua jenis perangkat.

Representational State Transfer Application Programming Interface (REST API) merupakan sebuah konsep pertukaran data berbasis text dengan memanfaatkan *protocol* HTTP [6]. REST API memungkinkan antara pengguna dan layanan dapat bertukar data antar platform. Bahkan antar layanan pun juga dapat bertukar informasi menggunakan konsep REST API. Dengan hal ini, komponen pendukung sebuah sistem informasi dapat bertukar data meskipun antar komponen ditempatkan pada Server yang berbeda lokasi fisiknya.

Pada dasarnya, REST API menggunakan *protocol* HTTP dalam bertukar informasi yang berarti *property* dari HTTP juga diturunkan pada REST API. HTTP merupakan *protocol* yang berbasis text yang dimana dibangun dari skema *request* dan *response*. *Request* dikirimkan oleh *client* kepada penyedia layanan (server) yang kemudian mendapatkan balasan berupa *response [7]*. Semua komunikasi pada *protocol* HTTP bersifat *stateless* yang berarti setiap *request* yang diterima akan dieksekusi secara independent tanpa mengetahui *request* sebelumnya. Termasuk apakah *client* tersebut telah terautentikasi sebelumnya atau tidak. Hal ini tentunya akan rentan terhadap penyusup yang belum terautentikasi sehingga tidak dapat dipastikan otoritasnya dalam mengakses sebuah fungsi pada sistem informasi tersebut. Tentunya hal ini berujung pada integritas dan kemanan data pada sistem informasi tersebut. Proses autentikasi dan otorisasi pada aplikasi dengan arsitektur Monolithic dapat memanfaatkan *session key* [8]. Namun pada arsitektur microservice tidak dapat memanfaatkan ini, dikarenakan *session key* bersifat *local* data pada satu sistem operasi sehingga konsep ini tidak bisa diaplikasikan.

Berdasarkan permasalahan tersebut, diperlukan sebuah perancangan dari skema autentikasi dan otorisasi yang dapat mengakomodir skema microservice. Proses autentikasi pada perancangan yang diusulkan akan dilakukan pada satu komponen yang bertindak sebagai *authority* yang berbeda tempat dari fungsionalitas yang diakses. Ketika pengguna mengakses sebuah fungsi, maka terlebih dahulu dilakukan konfirmasi terhadap otoritas yang dimiliki. Diharapkan dengan perancangan ini, dapat memberikan gambaran mengenai model autentikasi dan otorisasi pada skema REST API yang berbasiskan HTTP yang dapat digunakan dengan skema microservice.

2. TINJAUAN PUSTAKA DAN TEORI

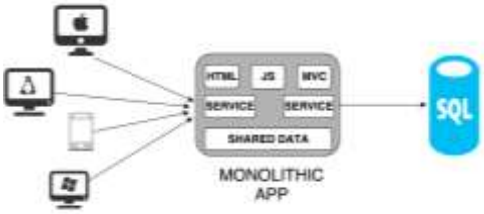
2.1 Tinjauan Pustaka

Terdapat penelitian sebelumnya yaitu [9] yang membangun sebuah website dengan konsep Model View Controller (MVC). Model ini merupakan model yang sangat banyak digunakan dalam pembuatan website saat ini. Pada penelitian tersebut keamanan websitenya memanfaatkan form login dimana pengguna wajib login menggunakan username dan passwordnya masing-masing. Setelah berhasil login maka pengguna akan mendapatkan sebuah PHP *session_id* yang hanya berlaku pada mesin tersebut saja. Pada penelitian [10] dijelaskan bahwa PHP Session cenderung tidak fleksible dan dibangun berorientasi pada satu server saja. Dengan kata lain mendukung penuh konsep dari arsitektur Monolithic namun tidak dengan Microservice. Hal ini dikarenakan microservice dapat terdiri dari banyak mesin server dan jika menggunakan konsep PHP Session, maka pengguna diharuskan login diseluruh layanan yang ada. Hal ini tentunya mengurangi kenyamanan pengguna dalam menggunakan layanan.

Berdasarkan hal tersebut yang menjadi konsep awal dari penelitian ini, yang dimana pengguna hanya perlu login sebanyak satu kali untuk kemudian dapat menggunakan semua layanan layaknya konsep Single Sign-On [11]. Namun dalam penelitian ini memanfaatkan jaringan komunikasi internal untuk proses validasinya antara satu layanan dengan layanan lainnya.

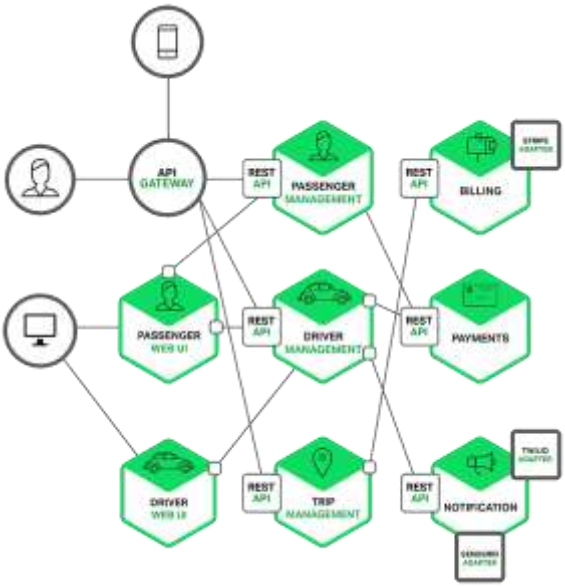
2.2. Arsitektur Microservice

Microservice merupakan arsitektur pembangunan sebuah sistem informasi yang melakukan pemecahan pada setiap komponen penyusun sistem informasi tersebut. Arsitektur ini merupakan pengembangan dari model aplikasi monolithic [5]. Monolithic sendiri merupakan arsitektur pembuatan aplikasi yang dimana seluruh komponennya merupakan satu kesatuan [4]. Gambar 1 merupakan gambaran dari arsitektur monolithic.



Gambar 1. Arsitektur Monolithic

Pada arsitektur monolithic, kita dapat melihat bahwa seluruh komponen penyusun aplikasi tersebut menjadi satu kesatuan. Kekurangan dari arsitektur ini dikarenakan seluruh komponennya menjadi satu, itu berarti keseluruhan komponen akan berada pada satu *environment* dan ketika semakin tinggi keperluan akan aplikasi tersebut maka akan terjadi penurunan performa. Selain hal tersebut, jika misalkan salah satu komponen mengalami gangguan, maka komponen yang lainnya akan terkena dampaknya. Hal ini yang diperbaiki pada konsep arsitektur microservice. Gambar 2 merupakan arsitektur microservice.



Gambar 2. Arsitektur Microservice [12]

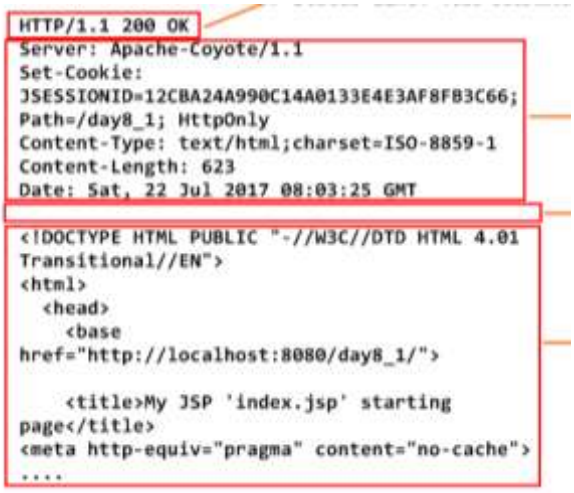
Pada arsitektur microservice, komponen penyusun aplikasi tersebut terpisah namun tetap dapat saling bekerjasama. Arsitektur ini memungkinkan pengembang aplikasi memiliki scalabilitas yang luas serta jika terjadi suatu error maka tidak berimbas pada komponen yang lain.

2.3. Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol merupakan sebuah *protocol* atau standarisasi pengiriman

data antar dua entitas yang sering digunakan ketika mengakses sebuah *website* [7]. Protokol HTTP digunakan pertama kali pada akses *website* sekitar tahun 1990 versi 0.9, kemudian berkembang menjadi versi 1.0 pada tahun 1996 dan menjadi versi 1.1. Pada versi terbaru, HTTP telah mendukung fitur *proxy, cache* serta koneksi *persistent*. Versi 2.0 muncul di tahun 2015 yang dikembangkan oleh Google dan yang terakhir versi 3.0 di tahun 2018 namun masih berupa *draft*.

HTTP merupakan *protocol* dengan format data yang dikirimkan memiliki dua bagian, yaitu header dan message body. Pada bagian header terdapat informasi tambahan mengenai data yang dikirimkan seperti tipe data, status data dan lainnya. Kemudian pada bagian message body adalah isi data sesungguhnya yang dikirimkan. Gambar 3 merupakan arsitektur umum dari *protocol* HTTP.



Gambar 3. Arsitektur pesan HTTP, 3 kotak merah diatas merupakan header dan dibawah merupakan body.

2.4. Hashing dan Salting

Hashing merupakan sebuah metode yang digunakan untuk mengidentifikasi sebuah data dengan menggunakan sebuah formula perhitungan yang menghasilkan sebuah nilai yang bersifat unik [13]. *Hashing* digunakan dalam berbagai aplikasi untuk melakukan identifiaksi data, seperti kecocokan *password* dan kecocokan file pada aplikasi *antivirus*. *Hashing* bukan metode enkripsi, karena *hashing* bersifat *irreversible* atau tidak dapat dikembalikan ke bentuk aslinya. Metode hashing ada banyak yang dapat digunakan, mulai dari CRC-32. SHA-1, SHA-2 dan sebagainya.

Sedangkan metode *Salting* merupakan sebuah metode penambahan sebuah *string* pada data yang akan dilakukan *hashing* [14]. Dengan proses *salting* dapat memberikan perlindungan tambahan terhadap kemungkinan terjadinya proses duplikasi nilai *hashing*. Konsepnya adalah meskipun data dapat diduplikasi, namun nilai *hashing*-nya tidak akan sama karena tidak mengetahui nilai *salting* yang ditambahkan ke dalam data tersebut.

2.5. *Representational State Transfer* (REST) API

Representational State Transfer API (REST API) merupakan standar arsitektur komunikasi yang berbasis website [15]. REST API menggunakan HTTP state header untuk memberikan tanda jenis operasi yang dilakukan. Mulai dari code response 200, 404 dan sebagainya. Kemudian memanfaatkan method state pada request header seperti GET, POST dan lainnya.

3. METODOLOGI PENELITIAN

3.1. Skema Alur Penelitian

Secara garis besar penelitian akan dilakukan dalam tahap dimulai dari studi literatur, desain dan perancangan sistem, dan kemudian melakukan evaluasi terhadap rancangan yang dibuat. Gambar 4 merupakan tahapan dari skema alur penelitian.

Gambar 4. Tahapan penelitian yang dilakukan

3.2. Pengumpulan Data

Data yang digunakan pada penelitian ini mengacu pada kebutuhan data pada proses autentikasi dan otorisasi seperti data username, password dan kode rahasia. Sebagai contoh

layanan yang diakses akan menampilkan data barang yang bersifat data *dummy*.

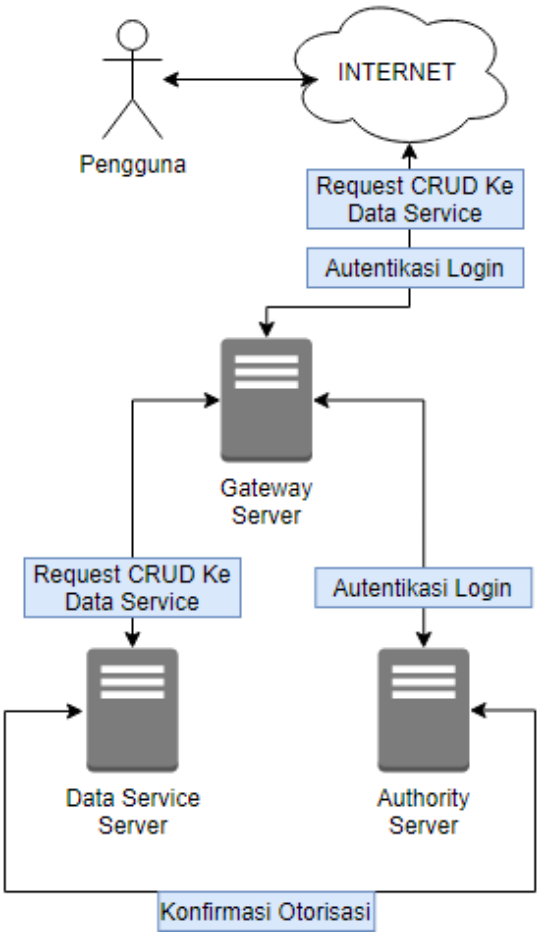
4. HASIL DAN PEMBAHASAN

Dalam penelitian ini menggunakan beberapa tahap sebagai metode penelitiannya. Tahap pertama adalah melakukan analisis dan perancangan skema dari arsitektur server yang akan saling berkomunikasi. Kemudian merancang bagaimana skema autentikasi yang dapat digunakan ketika pengguna melakukan *login* pada layanan aplikasi. Setelah pengguna melakukan autentikasi kemudian dilakukan perancangan mengenai proses otorisasi memastikan bahwa pengguna memiliki hak terhadap layanan yang diakses.

4.1 Rancangan Skema dan Pemetaan Fungsi Server

Pada tahap ini dilakukan design pada arsitektur skema yang diusulkan. Tahap design dilakukan mulai dari tahap perencanaan skema, komponen-komponen pendukung yang digunakan serta scenario pengujian yang dilakukan. Pada tahap ini juga dilakukan implementasi terhadap rancangan pengembangan dalam skala prototipe untuk melihat apakah rancangan yang digunakan bisa berjalan dengan baik. Gambar 5 merupakan desain dari arsitektur Autentikasi dan Otorisasi.

Dengan menggunakan model arsitektur pada gambar 5, memungkinkan untuk melakukan pemisahan antara satu layanan dengan layanan lainnya. Dan tentunya dengan hal ini membantu pengelola layanan untuk mengamankan data *credentials* yang bersifat rahasia seperti data *login* dan *password* dari pengguna. Pada rancangan tersebut, *Authority service* merupakan layanan terpisah dari semua layanan yang disediakan untuk memberikan keamanan pada data pengguna. Pengguna hanya melakukan akses sekali kepada *Authority service* pada saat melakukan *login* sebagai langkah autentikasi pertama kali mengakses layanan. Untuk selanjutnya pengguna dapat mengakses semua layanan yang disediakan tanpa perlu menghubungi *authority service* yang berarti data pengguna lebih jarang ter-*expose* ke publik.



Gambar 5. Rancangan Arsitektur keseluruhan

Untuk selanjutnya setiap pengguna melakukan akses terhadap sebuah layanan, maka pengguna tidak perlu kembali melakukan proses otorisasi layanan kepada *authority service* melainkan masing-masing layanan yang akan melakukan proses *otorisasi* dengan memastikan bahwa akses tersbut bersifat legal. Otorisasi dilakukan dengan cara mencocokan informasi yang disertakan pada *HTTP Header* berupa kode rahasia yang didapatkan ketika berhasil melakukan autentikasi atau login ke system yang diberikan oleh *authority service*. Kode rahasia ini akan kembali dikonfirmasi kepada *authority service* menggunakan jaringan *backbone* dari penyedia layanan. Dengan demikian proses otorisasi ini tidak akan terekspose ke publik termasuk kode rahasia yang dikonfirmasi.

Untuk melindungi arsitektur dan skema yang berjalan dibalik layanan, maka semua akses dari luar akan melalui sebuah *gateway* yang akan bertindak sebagai *proxy* dari layanan. Dengan menggunakan konsep tersebut, pengguna layanan akan mendapatkan asumsi bahwa

layanan dibangun dari satu *host* saja. Ini juga mempermudah pengguna dalam mengakses layanan karena tidak perlu mengakses layanan mulai dari proses autentikasi hingga otorisasi secara beda alamat domain. Cukup dengan mengakses satu alamat layanan. Untuk selanjutnya *gateway* yang akan meneruskan kemana *request* tersebut ditujukan. Tentunya akses *backbone* (antara satu layanan dengan layanan lainnya) tidak perlu melalui gateway melainkan dapat dilakukan secara point-to-point antar layanan yang disediakan. Hal ini termasuk proses otorisasi yang dilakukan ketika pengguna mengakses sebuah layanan.

Skema contoh penggunaan layanan pada penelitian ini adalah pengguna akan melakukan akses terhadap daftar nama barang yang tersedia yang dimana data nama barang tersebut bersifat *confidential* sehingga memerlukan proses otorisasi saat mengaksesnya. Untuk mendapatkan kode otorisasi (kode rahasia) maka pengguna wajib melakukan autentikasi terlebih dahulu.

4.2 Pembentukan Kode Rahasia Password

Kode rahasia dibentuk dengan proses *hashing* yang diberikan *salting* pada data yang akan dibentuk. Nilai *hashing* didapatkan dari penggabungan data berupa nomor identifikasi pengguna, *ip address* pengguna, serta kapan waktu pengguna melakukan proses login. Kemudian data *salting* yang akan ditambahkan adalah sebuah string rahasia yang hanya diketahui oleh *authority server* yang akan menjadi kunci rahasia. Dengan *salting* ini, maka tidak ada yang dapat menduplikasi hasil *hashing* sehingga memberikan keamanan yang tinggi.

Password pengguna nantinya juga akan disimpan menggunakan metode yang sama seperti pembentukan kode rahasia tersebut. Dengan metode ini, administrasi sistem juga tidak dapat melihat password yang digunakan oleh pengguna meskipun memiliki akses ke *database*. Tentunya ini memberikan rasa aman kepada pengguna bahwa tidak satupun mengetahui *password* yang digunakan selain pengguna itu sendiri.

4.3 Rancangan Skema Autentikasi

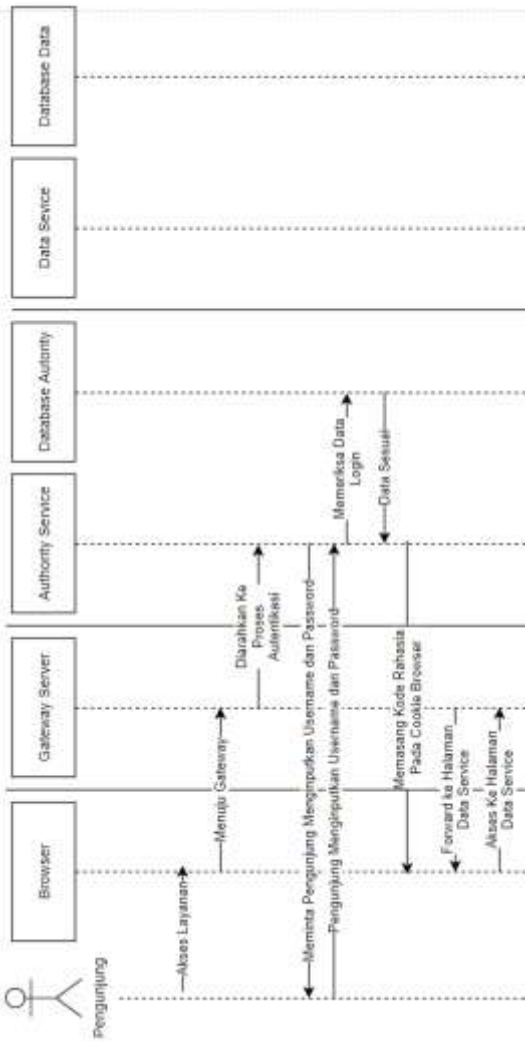
Ketika pengguna ingin melakukan akses terhadap sebuah layanan/service maka pengguna wajib melakukan proses autentikasi. Pada proses autentikasi ini pengguna harus memasukan username beserta password yang dimiliki. Jika sesuai maka Authority Service akan memberikan sebuah kode rahasia yang disimpan pada cookies dari browser dan mengarahkan pengguna menuju halaman Data Service. Gambar 6 skema dari proses autentikasi yang dilakukan.

Pada saat proses akses layanan dari browser menuju *gateway*, akses tentunya dilakukan melalui jaringan terbuka (*open-network*) yang dimana sangat rentan sekali terhadap serangan digital seperti penyadapan data (*sniffing*), perubahan data secara illegal (*tampering*) hingga pemalsuan identitas (*masquerading*). Untuk menghindari dari serangan tersebut, komunkasi antara browser dan *gateway* akan menggunakan Hypertext Transfer Protocol over SSL (HTTPS) sehingga akses yang dilakukan terhindar dari serangan.

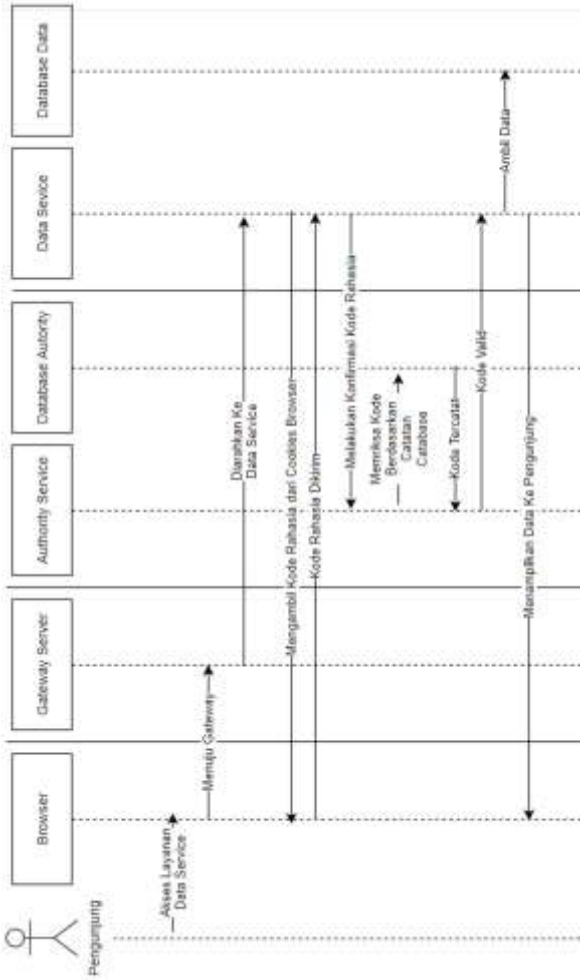
4.4 Rancangan Skema Otorisasi

Pengguna yang telah terautentikasi atau login kini telah mendapatkan kode rahasia yang telah diberikan oleh authority service yang diletakan pada cookies browser dan kini dapat mengakses data service. Akses yang dilakukan bersifat stateless sehingga data service yang berada pada mesin yang berbeda tidak mengetahui apakah sebelumnya pengunjung telah melakukan autentikasi sebelumnya atau tidak. Disinilah peran dari kode rahasia yang dimiliki oleh pengunjung. Data service akan melakukan pemeriksaan pada kode rahasia yang digunakan oleh pengunjung setiap kali pengunjung melakukan request. Pemeriksaan dilakukan dengan cara mengkonfirmasi kebenaran kode yang dimiliki kepada Authority Service. Jika kode yang dimiliki benar maka *request* dikabulkan. Gambar 7 merupakan skema otorisasi yang di lakukan oleh data service.

Sama seperti pada rancangan skema autentikasi yang dilakukan pada 4.2, akses dari browser pengguna kepada *gateway* akan dilakukan dengan protokol HTTPS karena akses akan dilakukan melalui internet. Namun untuk proses otorisasi antara satu layanan dengan layanan lainnya yang dalam hal ini *data service* dan *authority service* hanya melalui standar protokol HTTP tanpa menggunakan SSL. Hal ini dikarenakan akses akan dilakukan melalui jaringan lokal intranet yang tidak terekspos ke publik. Sehingga penggunaan HTTPS tidak diperlukan oleh proses otorisasi.



Gambar 6. Skema Autentikasi yang dilakukan



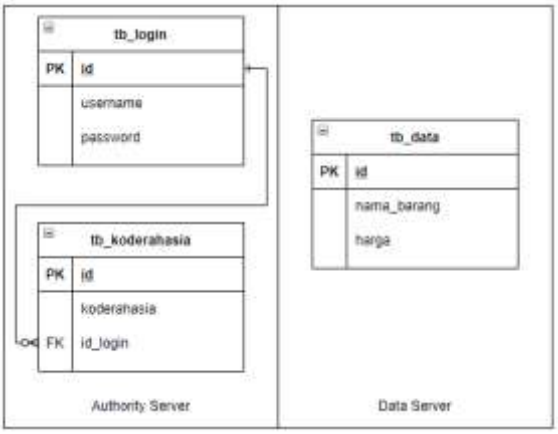
Gambar 7 Skema Rancangan Proses Otorisasi

4.5 Konseptual Database

Berdasarkan skema rancangan yang telah dikerjakan, maka dapat disimpulkan bahwa terdapat 2 database yang terpisah. Database berada pada server authority dan pada data *service*. Database pada *server authority* digunakan untuk menampung informasi login yang digunakan oleh pengunjung beserta kode rahasia yang terverifikasi dibuat oleh authorithy *service*. Pada database di data *service* hanya berisikan simulasi data informasi barang. Gambar 8 merupakan konseptual database yang dimiliki oleh kedua database yang digunakan oleh masing-masing server.

Database yang digunakan adalah Database Relational yang *menggunakan Structured Query Language* (SQL) dalam proses Create, Read, Update dan Delete (CRUD) sebuah data. Dalam hal ini database ditempatkan pada mesin yang sama dengan dimana layanan berada. Database pertama yang terdiri dari tabel tb_login dan

tb_koderahasia dipasangkan pada *authority server* dan database yang terdiri dari tb_data dipasangkan pada Data Service Server.



Gambar 8. Konseptual database yang dimiliki 2 server

Tabel tb_login merupakan tabel yang menyimpan data mengenai informasi login yang dimiliki oleh masing-masing pengguna. Primary Key yang digunakan adalah berupa id dengan nilai Integer. Pemilihan menggunakan id dengan nilai integer ditujukan agar pengguna dapat mengganti username jika diperlukan nantinya tanpa harus kehilangan relasi dengan tabel lainnya dalam hal ini tb_koderahasia. Data username dapat disimpan dalam bentuk varchar dengan kapasitas 25 karakter. Kemudian untuk password disimpan dalam bentuk varchar dengan kapasitas sebesar 255 karakter. Password yang disimpan akan di-*hashing* terlebih dahulu sebelum disimpan dengan menambahkan *salting* pada password yang dimiliki. Hal ini bertujuan agar password tidak mudah dibaca ketika terjadi kebocoran password. Hal ini juga memastikan bahwa admin sekalipun yang memilki akses ke database tidak dapat melihat password yang digunakan. Dengan begitu, *confidentiality* dari password yang digunakan dapat terjamin.

Tabel tb_koderahasia merupakan tabel yang menyimpan mengenai kode rahasia / kode otorisasi yang telah dibuat oleh *authority server*. Kode rahasia ini dibuat setiap kali pengguna melakukan autentikasi login pada layanan. Masing-masing kode rahasia memiliki *relasi one-to-one* kepada data pengguna yang berarti satu kode rahasia akan memiliki satu pengguna. Namun satu pengguna dapat memiliki lebih dari satu kode rahasia. Rancangan ini memungkinkan pengguna dapat mengakses pada perangkat yang berbeda secara bersamaan. Kode rahasia dibuat dengan melakukan *hashing* terhadap kombinasi username, waktu login, dan IP Address yang digunakan untuk mengakses. Hasil hashing tersebut yang digunakan sebagai kode otorisasi yang akan selalu dikonfirmasi setiap pengguna melakukan akses.

Tabel tb_data merupakan tabel yang menyimpan data *dummy* berupa data barang yang akan diakses oleh pengguna. Hal ini digunakan sebagai contoh dari penggunaan rancangan serta nantinya dapat digunakan sebagai pengujian. Setiap pengguna yang akan mengakses data tersebut diwajibkan memiliki kode otorisasi yang valid.

4.6 Uji Coba Simulasi Perancangan

Untuk menganalisa dari apakah perancangan ini bersifat *applicable*, maka dilakukan sebuah simulasi. Simulasi dilakukan dengan menggunakan kemampuan *Virtual Host (vhost)* pada web service Apache dan memanfaatkan module CURL pada PHP versi 7.4 sebagai sarana komunikasi internal *service*. Dibentuk 3 *vhost* yang dimana masing-masing bertindak sebagai *gateway, authority service* dan *data service.*

Pada Gambar 9 merupakan potongan kode dari bagaimana proses pembentukan kode rahasia terjadi. Kode rahasia terbentuk pada saat pengguna berhasil melakukan login dan disimpan pada *database*.

```
1   function login($username, $password,
    $ipaddress = '127.0.0.1') {
2     $dbconn = dbConnection();
3     $username = sanitasiVariabel($dbconn,
    $username);
4     $password = hashSHA256($password);
5     $SQL = "SELECT * FROM tb_login WHERE
    username = '" . $username . "' AND
    password='" . $password . "'";
6     $result = mysqli_query($dbconn, $SQL);
7
8     if (mysqli_num_rows($result) > 0) {
9       $koderahasia = hashSHA256($username
    . $ipaddress . "SALTKEY");
10      $row = $result-
    >fetch_array(MYSQLI_NUM);
11
12      $SQL = "INSERT INTO
    tb_koderahasia(login_id, koderahasia)
    VALUES ('$row[0]','$koderahasia')";
13      if (mysqli_query($dbconn, $SQL)) {
14        //Tulis cookie
15        setcookie("tokenauth",
    $koderahasia);
16        return true;
17      } else {
18        return false;
19      }
20
21    } else {
22      return false;
23    }
```

```
24    mysqli_close($dbconn);
25  }
```

Gambar 9. Potongan Kode Proses Pembentukan Kode Rahasia

Selanjutnya pada Gambar 10 merupakan potongan kode yang menunjukan bagaimana *data service* melakukan konfirmasi keaslian kode rahasia tersebut. Setiap kali pengguna melakukan akses sebuah data maka *data service* akan memeriksa kode rahasia tersebut dan mengkonfirmasinya kepada *authority service*. Gambar 11 merupakan potongan kode dari bagaimana *authority service* melakukan pengujian terhadap kode yang dikonfirmasi.

```
1  if(!isset($_COOKIE['tokenauth'])) {
2      exit('Anda belum login!');
3    } else {
4      $ch = curl_init();
5
6      curl_setopt($ch,
   CURLOPT_URL,"http://auth.simpleauth.lo
   cal/index.php");
7      curl_setopt($ch, CURLOPT_POST, 1);
8      curl_setopt($ch,
   CURLOPT_POSTFIELDS,
   "koderahasia=".$_COOKIE['tokenauth']."
   &validasi=true");
9      curl_setopt($ch,
   CURLOPT_RETURNTRANSFER, true);
10
11     $server_output = curl_exec($ch);
12
13     curl_close ($ch);
14     if ($server_output != "OK") {
15       exit('Anda belum login!');
16     }
17   }
```

Gambar 10. Potongan Kode Proses Verifikasi Oleh Data Service

```
1  function validasiKode($token) {
2      $dbconn = dbConnection();
3      $username =
   sanitasiVariabel($dbconn, $token);
4      $password = hashSHA256($password);
5
6      $SQL = "SELECT * FROM
   tb_koderahasia WHERE koderahasia =
   '$token'";
7      $result = mysqli_query($dbconn,
   $SQL);

8      if (mysqli_num_rows($result) > 0)
9  {
10       return true;
11     } else {
12       return false;
13     }
14     mysqli_close($dbconn);
15   }
```

Gambar 11. Potongan Kode Proses Verifikasi Oleh Authority Service

Berdasarkan simulasi yang dilakukan, proses autentikasi dapat berjalan dengan baik. Pengguna dapat melakukan login menggunakan password yang dimilikinya. Kemudian ketika melakukan proses otorisasi, *data service* dapat melakukan proses otorisasi dengan baik, Ketika pengguna belum login maupun terjadinya ketidaksesuaian dari kode rahasia yang digunakan, website berhasil menutup koneksi dan memunculkan pesan bahwa pengguna belum login.

5. Kesimpulan dan Saran

Berdasarkan perancangan yang telah dilakukan dapat disimpulkan bahwa perancangan ini telah menghasilkan 2 skema rancangan yang dimana skema proses autentikasi beserta skema proses otorisasi. Dalam perancangannya menunjukan bahwa implementasi dilakukan pada 3 buah server yang memiliki peran dan fungsinya masing-masing.

Berdasarkan skema perancangan ini diharapkan dapat diperkaya fiturnya dengan menambahkan double atau mungkin triple otorisasi untuk memastikan keaslian dari kode rahasia yang dimiliki client. Saran selanjutnya adalah melakukan implementasi sesuai dengan perancangan yang telah dilakukan. Dalam menunjang implementasi dan pengujiannya, layanan dapat diimplementasikan berbasis website.

6. UCAPAN TERIMA KASIH

Terima kasih ditujukan kepada Institut Teknologi dan Bisnis STIKOM Bali yang telah memberikan bantuan pendanaan terhadap penelitian ini.

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] S. Sunardi and S. Fadli, "SISTEM INFORMASI PENGOLAHAN DATA KELAPA SAWIT BERBASIS CLIENT-SERVER," *Jurnal Manajemen Informatika dan Sistem Informasi*, no. 2, pp. 23-28%V 1, 2018-08-30 2018.

[3] R. Rohana and K. Imtihan, "Sistem Informasi Keluhan Pelanggan Pada Perusahaan Daerah Air Minum (PDAM) Kabupaten Lombok Tengah," *Jurnal Manajemen Informatika dan Sistem Informasi*, no. 1, pp. 24-30%V 1, 2018-02-07 2018.

[4] E. Axelsson and E. Karlkvist, "Extracting Microservices from a Monolithic Application," 2019.
[5] 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.
[6] 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.
[7] 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.
[8] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen, "SessionShield: Lightweight protection against session hijacking," in *International Symposium on Engineering Secure Software and Systems*, 2011: Springer, pp. 87-100.
[9] A. Rosenbloom, "A simple MVC framework for web development courses," in *Proceedings of the 23rd western canadian conference on computing education*, 2018, pp. 1-3.
[10] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi, "Efficient and flexible discovery of php application vulnerabilities," in *2017 IEEE european symposium on security and privacy (EuroS&P)*, 2017: IEEE, pp. 334-349.
[11] S. Purkayastha, J. W. Gichoya, and S. A. Addepally, "Implementation of a single sign-on system between practice, research and learning systems," *Applied clinical informatics*, vol. 8, no. 1, p. 306, 2017.
[12] D. Boyke. (2016, 1 Oktober 2020). *Microservices, Konsep dan Implementasi (1)* [Online]. Available: https://indosystem.com/blog/microservices-konsep-dan-implementasi/.
[13] S. Long, "A Comparative Analysis of the Application of Hashing Encryption Algorithms for MD5, SHA-1, and SHA-512," in *Journal of Physics: Conference Series*, 2019, vol. 1314, no. 1: IOP Publishing, p. 012210.
[14] N. Sharma, V. Srivastava, and A. Sharma, "An Improved Authentication Security Scheme," *PaperID Authors Title Page No.*, p. 26, 2017.
[15] 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.