BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

Pada bab 2 akan dibahas tentang tinjauan pustaka dan dasar teori yang digunakan dalam pembuatan Skripsi ini.

2.1 Tinjauan Pustaka

Nursyamsi (2009), implementasi sistem *Single Sign On*, dimana teknologi yang digunakan berbasis *Java* dan *Java Open Source Single Sign On (JOSSO)*. Abdurrahman, Luthfi (2012), implementasi sistem *Single Sign On*. Sistem yang dibuat menggunakan teknologi *OpenAm* dan *OpenLDAP* dengan otentikasi *Kerberos*. Yosrinal (2014), perancangan dan implementasi *Resource Server* dan *Authorization Server*, sistem ini dibuat dengan menggunaka teknologi otentikasi *OAuth 2.0*. Kemudian Wijaya Kusuma, Ignatius, Sanjoyo Adi, Puspaningtyas (2012), membuat sistem otentikasi *Single Sign On* menggunakan algoritma *Diffie-Hellman*, sistem yang dibuat menggunakan *database parallel* dengan menggunakan *RMI (Remote Method Invoication)*.

Sedangkan pada naskah ini akan merancang dan membuat aplikasi *Single Sign On*, dimana untuk teknologi otentikasinya menggunakan *OAuth 2*. Adapun perbandingan tinjauan pustaka tersebut dapat di lihat pada Tabel 2.1:

Tabel 2. 1 Tinjauan Pustaka

Nama Peneliti	Topik	Metode / Teknologi	Objek	Hasil Keluaran
Nursyamsi (2009)	Single Sign On berbasis Java	*Java Open Source Single Sign On (JOSSO)*	Login web	otentikasi
Abdurrahman, Luthfi (2012),	Sistem Single Sign On	*OpenAm, OpenLDAP dan Kerberos*	Login web	otentikasi
Yosrinal (2014)	Resource Server dan Authorization Server	*OAuth 2.0*	Login web	otentikasi
Wijaya Kusuma, Ignatius, Sanjoyo Adi, Puspaningtyas (2012)	sistem otentikasi Single Sign On Server	*Algoritma Diffie-Hellman*	Login web	otentikasi
Luthfiati, Nisrina (2017)	*Aplikasi Chatting berbasis Web*	*Websocket dan Node Js*	Aplikasi chatting	Realtime aplikasi chatting
Kristiawan Adi Lutvianto (2019)	Aplikasi Single Sign On	*Node Js, Golang dan OAuth 2.0*	Login web	Aplikasi layanan otentikasi

2.2 Dasar Teori

2.2.1 Pengenalan OAuth

OAuth (Open Authorization) adalah protokol otorisasi standar terbuka yang memungkinkan pengguna mengakses aplikasi tanpa perlu berbagi *password* mereka. Pemilik aplikasi mengintegrasikan credential milik pengguna dengan teknologi otentikasi yang berasal dari penerbit *API (Application Programming Interface)*. *OAuth* juga mengizinkan otorisasi *API* yang sudah terproteksi yang berasal dari desktop ataupun aplikasi web melalui metode sederhana dan standard.

Mengatur lalu lintas data antar aplikasi dan digunakan saat penerbit *API* ingin mengetahui siapa yang terlibat dan berkomunikasi di dalam sistem.

OAuth 1.0 (juga dikenal sebagai *RFC 5849*), diterbitkan pada tanggal 4 Desember 2007, direvisi pada tanggal 24 Juni 2009, dan diselesaikan pada bulan April 2010 yang memberikan pengaruh penting pada perkembangan keamanan *web API* tanpa harus pengguna berbagi *username* dan *password* mereka. Adapun pencipta dan pengagas utama dari otentikasi *OAuth* berbasis *API* ini adalah E. Hammer-Lahav, Ed.

OAuth 2 adalah *project* lanjutan dari protokol *OAuth* yang asal mulanya diciptakan pada akhir tahun 2006. *OAuth 2* lebih menekankan pada kemudahan *client* sebagai pemilik dan pengembang aplikasi dengan memberikan otorisasi khusus di berbagai aplikasi. *OAuth* berada dalam pengembangan *IETF OAuth WG* dan didasarkan pada usulan *WRAP OAuth*. *WRAP (Web Resource Authorization Protocol)* adalah profil *OAuth* yang memiliki sejumlah kemampuan penting yang tidak tersedia di versi *OAuth* sebelumnya. Spesifikasi terbaru dari *OAuth* disumbangkan kepada *IETS OAuth WG* dan merupakan dasar dari terciptanya *OAuth* versi a. 2.

Dengan banyaknya aplikasi *web* yang mengapdopsi kolaborasi penggunaan jaringan sosial, pengembang aplikasi memiliki kesempatan untuk menghubungkan pengguna dengan aplikasi dimanapun mereka berada. Yang dapat meningkatkan efektivitas dan efisiensi terjadinya transaksi pengolahan data didalam sistem tersebut. *OAuth* menyediakan kemampuan terhubung dengan sistem aplikasi secara aman, sehingga pengguna tidak perlu menyerahkan sandi

akun pentingnya. Ada beberapa fungsionalitas menguntungkan yang tersedia pada *OAuth* meliputi:

1. Mendapatkan akses ke grafik social media seperti teman yang berasal dari *Facebook*, orang - orang yang mengikuti *(following)* di *Twitter*, ataupun list *contact* yang berasal dari *Google Contact*.
2. Berbagi informasi mengenai kegiatan seorang pengunjung pada sebuah web aplikasi dengan menandai postingan mereka di *Facebook* ataupun *tweet* dari *twitter* mereka.
3. Mengakses penggunaan *Google Docs* atau akun *Dropbox* untuk menyimpan data online mereka dengan berbagai *file* pilihan

2.2.2 **OAuth 2.0**

Terdapat 4 peran utama dalam mekanisme kerja *OAuth 2* yakni :

1. Resource Server

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

2. Resource Owner

   Memposisikan diri sebagai pemilik sumber daya *(Resource Owner)*, yang merupakan pemilik dari aplikasi. Pemilik sumber daya memiliki kemampuan untuk mengakses sumber daya server dengan aplikasi yang sudah tersedia.

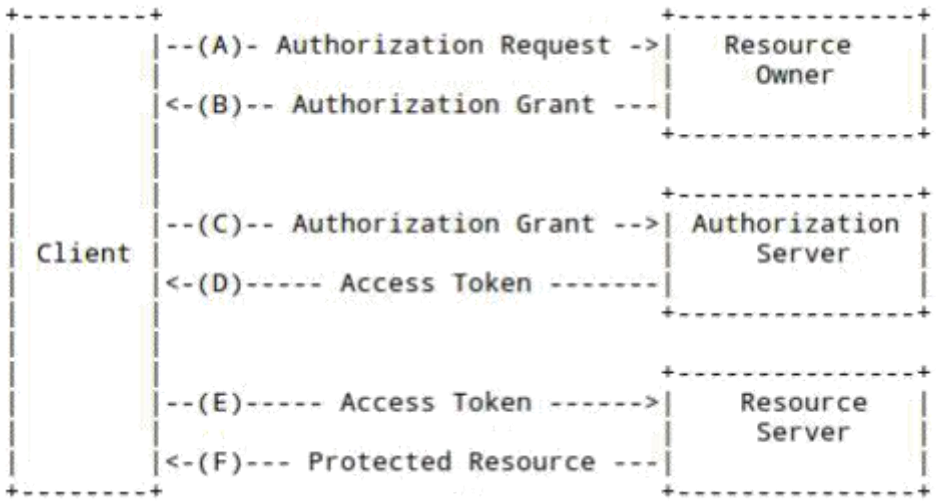
3. Client

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

4. Authorization Server

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

Mekanisme kerja *OAuth 2* terbentuk dari peran aktif 4 (empat) bagian yang terdiri dari : *Client, Resource Owner, Authorization Server, Resource Server*. Gambar 2.5 menunjukkan mekanisme kerja OAuth 2, sedangkan pada tabel 2.1 menunjukkan keterangan dari tugas dan fungsi 4 (empat) komponen OAuth.



Gambar 2. 1 Mekanisme Kinerja OAuth 2

Tabel 2. 2 Tugas dan Fungsi 4 (empat) Komponen OAuth 2

	Keterangan
A	*Client* melakukan permintaan otorisasi dari *Resource Owner*. Permintaan otorisasi dapat dilakukan langsung menuju *Resource Owner*, atau jika tidak langsung melalui perantara *Authorization Server*.
B	*Client* mendapatkan persetujuan otorisasi yang merupakan *credential* mewakili otorisasi kepemilikan *client*. Pemberian otorisasi ini tergantung pada metode yang digunakan oleh *client* dan jenis yang didukung oleh *Authorization Server*.
C	*Client* melakukan permintaan akses token dengan otentikasi kepada *Authorization Server*, *client* mendapatkan penyajian hibah dan bentuk otorisasi dari *Authorization Server*.
D	Otorisasi Server *(Authorization Server)* melakukan otentikasi kepada *client* dan memvalidasi pemberian otorisasi kepada *client*, jika sesuai dan berlaku, otorisasi server membagikan akses token.
E	*Client* melakukan permintaan sumber daya yang sudah diproteksi dari *Resource Server*, melakukan tindakan otentikasi dengan menghadirkan akses token.
F	*Resource Server* memvalidasi akses token, jika valid dan sesuai, akan melayani permintaan *client* untuk menggunakan aplikasi yang sudah terlindungi.

2.2.3 Hibah Otorisasi (Authorization Grant)

Hibah otorisasi adalah mandat mewakili sumber daya pemilik otorisasi (Resouce Owner) untuk mengakses sumber daya yang dilindungi yang digunakan oleh client (pengunjung) untuk mendapatkan akses token. Ada beberapa metode hibah otorisasi :authorization code, implicit, resource owner password credential.

2.2.4 Authorization Code

Kode otorisasi diperoleh dengan menggunakan otorisasi server sebagai perantara antara client dan resource owner. Client melakukan permintaan otorisasi langsung dari resource owner, diarahkan menuju ke otorisasi server melalui user-agent (web browser) yang pada gilirannya mengarahkan pemilik sumber daya ke client dengan kode otorisasi.

2.2.5 Implicit

Hibah otorisasi implisit adalah hibah otorisasi yang disederhanakan dan dioptimalkan untuk client yang diimplementasikan dalam browser menggunakan bahasa scripting javascript. Dalam aliran otorisasi implicit, client diberikan langsung akses token sebagai hasil dari otorisasi pemilik sumber daya (resource owner) yang kemudian digunakan untuk mendapatkan akses token.

2.2.6 Resource Owner Password Credential

Sumber daya pemilik password credential seperti (username dan password) dapat digunakan langsung sebagai hibah otorisasi untuk mendapatkan akses token. Credential yang digunakan berdasarkan kepercayaan tingkat tinggi antara pemilik sumber daya dan client. Jenis hibah ini membutuhkan akses client langsung ke

pemilik sumber daya credential yang digunakan untuk satu permintaan dan ditukar dengan akses token.

2.2.7 Client Credential

Kredensial klien adalah bentuk lain dari otentikasi klien yang digunakan sebagai hibah otorisasi terbatas pada sumber daya yang dilindungi di bawah pengawasan dan pengaturan klien. Atau ke sumber daya terlindungi yang sebelumnya telah di atur dengan server otorisasi. Kredensial klien digunakan sebagai otorisasi hibah ketika klien bertindak sebagai pemilik sumber daya (resource owner) atau meminta akses ke sumber daya yang dilindungi berdasarkan otorisasi yang sudah diatur dengan server otorisasi (resource server).

2.2.8 Access Token

Access Token adalah credential yang digunakan untuk mengakses sumber daya yang terlindungi. Akses token adalah kumpulan string yang mewakili suatu kuasa yang diberikan kepada client. Setiap token menyatakan batasan dan jangka waktu akses, yang diberikan oleh Resource Owner (pemilik sumber daya), yang berasal dari Resource Server (sumber server) dan Authorization Server (otorisasi server). Token dapat menunjukkan pengenal yang digunakan untuk mengambil informasi otorisasi atau mungkin data diri yang diverifikasi yaitu berupa tanda string yang terdiri dari beberapa rangkaian data. Akses token dapat memiliki format yang berbeda baik dalam struktur dan metode pemanfaatan.

2.2.9 Refresh Token

Refresh token adalah token credential yang digunakan untuk mendapatkan token akses yang baru. Merefresh kembali akses token yang dikeluarkan otorisasi

server (authorization server) dan digunakan untuk mendapatkan akses token terbaru ketika akses token tidak menjadi sah atau akses token memiliki ruang lingkup yang lebih pendek yang diterbitkan oleh authorization server. Refresh token merupakan string yang mewakili otorisasi yang diberikan kepada client oleh pemilik sumber daya. Tidak seperti akses token, refresh token hanya digunakan untuk otorisasi server dengan tidak mengirimkan ke pemilik sumber daya (resource owner).

2.2.10 **Jenis Client Profil**

OAuth 2 mendefenisikan beberapa jenis *client profile* penting sebagai berikut :

1. *Server Side Web Application*

   Adalah salah satu dari *client profile OAuth 2* yang berjalan pada sisi web server. Aplikasi web yang telah diakses oleh pemilik sumber daya ataupun pengguna, melakukan permintaan *API (Application Programming Interface)* memanggil penggunaan satu bahasa program yang berjalan pada sisi server.

2. *Client Side User Agent Based Application*

   Adalah jenis *client profile* yang berjalan pada *web browser* pengguna, dimana *client* mendapatkan hak akses kode aplikasi ataupun permintaan *API*. Aplikasi dapat berupa *javascript* yang terdapat di dalam sebuah halaman web bisa berupa *browse extensio*n, atau menggunakan teknologi plugin seperti Flash.

3. *Resource Owner Password Flow*

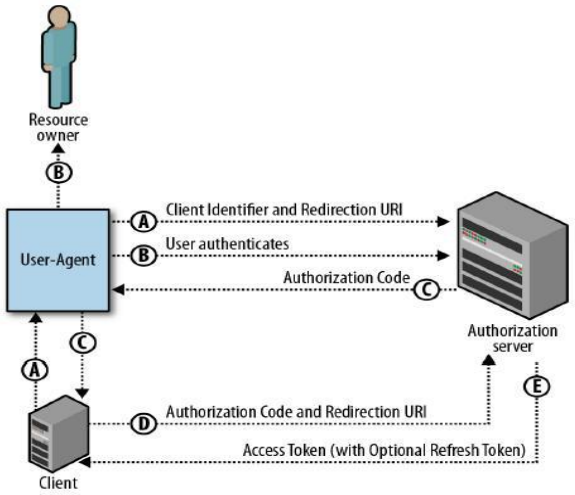
   Adalah jenis *client profile* yang dimana r*esource owner* (pemilik sumber daya) memiliki kepercayaan tingkat tinggi terhadap *client*. Kelemahan dari

client profile berikut adalah otorisasi server harus berhati - hati saat mengijinkan hibah ke dalam aliran sistem yang bekerja, mengingat kepercayaan penuh yang sudah diberikan kepada *client.*

4. *Workflow Client Profile*

Workflow Client Profile pada *OAuth 2* berdasarkan pada mekanisme kerja yang berjalan pada sisi *client side* ataupun *server side*. Di bawah ini adalah model kerja yang otentikasi OAuth 2 yang berjalan pada sisi *server side, client side,* r*esource owner password flow* dan *client credential.*

5. *Server Side Web Application Flow*



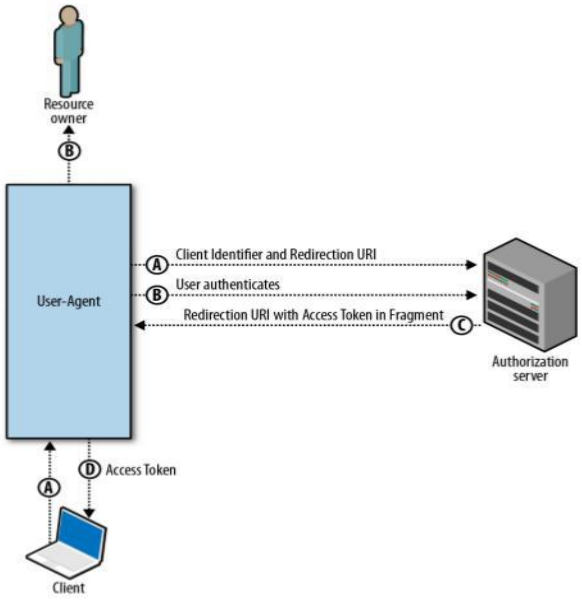
Gambar 2. 2 Server Side Web Application Flow

Tabel 2. 3 Keterangan Server Side Web Application Flow

	Keterangan
A	*Client* memulai aliran menuju aplikasi pemilik sumber daya *(Resource Owner)* lewat perantara *user-agent* langsung menuju titik akhir otorisasi.
B	*Otorisasi Server* mengotentikasi *Resource Owner* dan menetapkan apakah pemilik sumber daya memberikan atau menolak permintaan akses *client.*

C	Dengan asumsi *client* mendapatkan kuasa melakukan akses, *otorisasi server* mengarahkan ulang *user-agent* kembali ke *client* menggunakan *redirect url* (*redirect url* termasuk kode otorisasi).
D	*Client* meminta akses token dari *otorisasi server* termasuk didalamnya kode otorisasi (*code authorization*) yang sudah diterima sebelumnya. Ketika melakukan permintaan, *client* mengotentikasi dengan *server otorisasi*. *Client* termasuk *url direction* digunakan untuk memperoleh kode otorisasi untuk verifikasi.
E	*Otorisasi server* mengotentikasi *client*, memvalidasi kode otorisasi dan memastikan *url redirect* di terima sesuai dengan *url* yang digunakan untuk mengarahkan *client*, jika valid *otorisasi server* merespon kembali dengan *akses token* dan opsional *refresh token.*

6. Client Side Web Application Flow

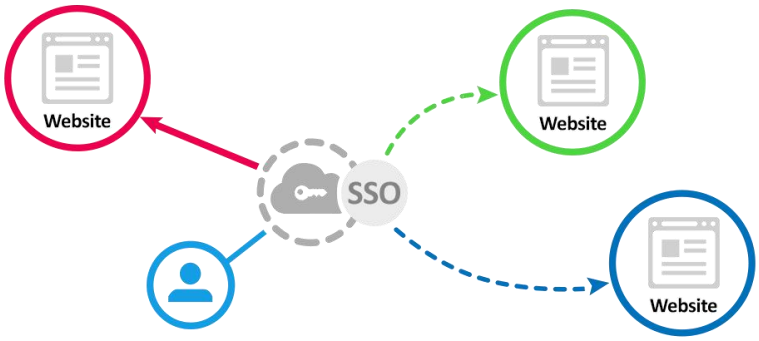


Gambar 2. 3 Client Side User Web Application Flow

Tabel 2. 4 Keterangan Cient Side Web Application Flow

	Keterangan
A	*Client* memulai aliran dengan mengarahkan pemilik sumber daya (*Resource Owner*) lewat perantara *user-agent* menuju titik akhir otorisasi. *Otorisasi Server* akan mengirim kembali ke *user-agent* setelah akses diberikan atau ditolak.
B	*Otorisasi Server* mengotentikasi pemilik sumber daya, dan menetapkan apakah pemilik sumber daya memberikan atau menolak permintaan akses *client*.
C	Dengan asumsi pemilik sumber daya memberikan akses, *otorisasi server* mengarahkan ulang *user-agent* kembali ke *client* menggunakan *redirect url* sebelumnya. Pengalihan *url* termasuk akses token di dalam *fragmen url*
D	*User-agent* mengikuti instruksi pengalihan dengan membuat permintaan sumber daya *client web hosting* dengan tetap mempertahankan informasi *fragment* lokal agar *user-agent* dapat mengeksekusi *scripts* yang disediakan web-host sumber daya *client* lokal yang mengektrak akses token dan lolos ke *client*.

2.2.11 Single Sign-On (SSO)



Gambar 2. 4 Arsitektur Single Sign On

Single Sign On atau sering di singkat SSO merupakan teknologi login terpusat dimana sistem - sistem yang berbeda dalam satu perusahaan besar dapat terintegrasi dengan satu user Account yang valid.

Sebagai contoh, perusahaan besar seperti google, dimana kita bisa mengkases berbagai produk google dengan sistem yang berbeda beda hanya dengan satu akun google yang valid, secara garis besar single sign on memudahkan dari sisi user untuk bisa mengakases berbagai produk dari google dengan satu akun, sehingga user hanya perlu mengingat satu akun saja untuk dapat mengakses semua produk - produk dari google misal blogger, google drive, google+, google mail, google playstore, dan masih banyak lagi produk google yang tidak bisa saya sebutkan satu persatu.

Pada umumnya SSO memiliki beberapa keuntungan, antara lain :

1. Pengguna tidak perlu mengingat banyak username dan password. Cukup dengan satu credential, sehingga pengguna cukup melakukan proses

otentikasi sekali saja untuk mendapatkan izin akses terhadap semua layanan aplikasi yang tersedia di dalam jaringan.

2. Kemudahan pemrosesan data. Jika setiap layanan aplikasi memiliki data pengguna masing-masing, maka pemrosesan data pengguna (penambahan, pengurangan, perubahan) harus dilakukan pada setiap aplikasi yang ada. Sedangkan dengan menggunakan sistem SSO, cukup hanya melakukan sekali pemrosesan pada server database backend-nya. Hal ini menyatakan bahwa penggunaan sistem SSO meningkatkan efisiensi waktu dan kepraktisan dalam memproses data.
3. Tidak perlu membuat data pengguna yang sama di setiap aplikasi Karena setiap layanan aplikasi dalam jaringan dapat terhubung langsung dengan server database backend ini, maka hanya dengan sekali saja meng- input data kedalam database, credential pengguna akan valid di seluruh layanan aplikasi.
4. Menghemat biaya untuk pemeliharaan password. Ketika harus melakukan reset password karena pengguna lupa pada password-nya, pengelola layanan tidak perlu menghabiskan waktu dan bandwith untuk menemukan data credential pengguna.

Selain mendatangkan keuntungan, sistem SSO juga dapat mendatangkan kerugian, antara lain:

1. Pentingnya kesadaran pengguna untuk merahasiakan data credential dan menjaga keadaan login-nya. Bila masih dalam keadaan login, pengguna yang tidak sah dapat memakai mesin yang ditinggalkan pengguna sahnya.

2. Kerumitan mengimplementasikan sistem SSO kedalam sebuah jaringan yang heterogen dan multiplatform, sehingga banyak pengelola layanan jaringan kurang begitu giat dalam mengimplementasikannya.
3. Kelemahan dalam hal keamanan. Jika password sistem pengelola layanan jaringan diketahui oleh orang yang tidak berhak, maka orang tersebut dapat melakukan perubahan terhadap semua data yang ada didalam sistem.
4. Titik Kegagalan Tunggal (Single point failure). Karena setiap layanan aplikasi bergantung kepada sistem Single Sign-On, sistem ini dapat menjadi suatu titik kegagalan bila tidak dirancang dengan baik. Kondisi apapun yang dapat menyebabkan sistem SSO padam, mengakibatkan pengguna tidak dapat mengakses seluruh layanan aplikasi yang dilindungi oleh sistem SSO tersebut.

2.2.12 Go Programming Language (GoLang)

Go atau sering juga disebut GoLang adalah bahasa pemrograman yang free and open source dari Google. Bahasa ini dibuat pada tahun 2007 oleh Robert Griesemer, Rob Pike, dan Ken Thompson. Meskipun demikian Go pertama kali dirilis ke publik sebagai proyek open source pada tanggal 10 November 2009. Setelah desain dan pengembangan yang berlangsung selama bertahun-tahun, Go versi stabil (versi 1) akhirnya dirilis pada tanggal 28 Maret, 2012.

Go muncul dari rasa frustasi orang-orang Google terhadap bahasa pemrograman sistem yang telah ada karena menjadi terlalu sulit. Dalam memilih bahasa pemrograman sistem, ada tiga faktor yang perlu diperhatikan yaitu kompilasi yang efisien, eksekusi yang efisien, dan kemudahan dalam

pengembangan. Ketiga faktor tersebut tidak tersedia di satu bahasa. Programmer akhirnya lebih memilih kemudahaan di atas keamanan dan efisiensi dengan mengganti pilihan bahasanya ke Python dan JavaScript dibanding C++ atau Java. Go merupakan usaha untuk menyatukan kebutuhan-kebutuhan tadi kedalam satu bahasa yang memiliki kemudahan namun dengan kompilasi dan eksekusi yang efisien. Berikut beberapa kelebihan dari bahasa Go:

- *Compiled* dan tidak menggunakan intermediary code.
- *Concurrency at language level*, bahasa Go mendukung concurrency dan komunikasi antar thread/proses tanpa perlu memahami kompleksitas implementasinya (*mutex*, *synchronization*).
- *Garbage Collected*, artinya programmer tidak perlu menghapus alokasi memori secara manual, sehingga tidak akan terjadi *memory leak*.
- *Built in String and Map*, artinya string dan array asosiatif merupakan tipe data yang didukung penuh oleh bahasa, bukan berupa library.
- *No Type Hierarchy*/*Inheritance*, artinya tidak terdapat hirarki tipe data, hanya terdapat Interface, selama tipe data mengimplementasikan fungsi tertentu, maka polymorphism dapat terjadi, pemeriksaan ini dilakukan saat compile bukan pada saat *Runtime*. Inheritance dapat diatasi dengan *Embedding* (*Nameless Composition*).
- *Small Memory Footprint*, artinya program yang dibuat dengan bahasa ini menggunakan RAM relatif sedikit.
- *Zeroed Memory*, artinya setiap tipe data selalu memiliki default value 0.

- *No Unsafe Low Level Code*, artinya program yang dibuat tidak dapat menggunakan *source code* yang tidak aman (direct access ke memori).

2.2.13 NODE JS

Nodejs dikembangan dari engine javaScript yang dibuat oleh Google untuk Browser Chrome / Chronium (V8) ditambah dengan libUV serta beberapa 6 pustaka internal lainnya. Dengan menggunakan Nodejs semua pengembangan akan dilakukan dengan javaScript, baik pada sisi client maupun server.

Pengembangan aplikasi dengan menggunakan Node.js dapat dilakukan secara moduler yaitu dengan memisahkan berbagai komponen kedalam pustaka (library). Pustaka tersebut dapat dikelola dengan npm yang terdapat di Node.js. Pada dasarnya, Node.JS sebuah runtime environment dan script library. Sebuah runtime environment adalah sebuah software yang berfungsi untuk mengeksekusi, menjalankan dan mengimplementasikan funsi-fungsi serta cara kerja inti dari suatu bahasa pemograman. Sedangkan script library adalah kumpulan, kompilasi atau bank data berisi skrip/kode-kode pemograman. (Equan Pr.2013). Berikut kelebihan dari Node Js:

- Node.js menggunakan bahasa pemrograman JavaScript yang diklaim sebagai bahasa pemrograman yang paling populer dan banyak dikenal oleh masyarakat luas.
- Node.js mampu menangani ribuan koneksi bersamaan dengan penggunaan resource minimum untuk setiap prosesnya
- Node.js sangat diandalkan terutama untuk membuat aplikasi real-time

- Node.js adalah project open source, sehingga siapapun dapat melihat struktur kode dan juga dapat berkontribusi untuk pengembangannya
- Penggunaan JavaScript di sisi server dan juga client meminimalisir ketidakcocokan antar dua sisi lingkungan pemrograman, seperti terkait komunikasi data yang mana menggunakan struktur JSON yang sama di kedua sisi, validasi form yang sama yang dapat dijalankan di sisi server dan client, dan sebagainya.
- Database NoSQL seperti MongoDB dan CouchDB mendukung langsung Javascript sehingga interfacing dengan database ini akan jauh lebih mudah.
- Node.js memakai V8 yang selalu mengikuti perkembangan standar ECMAScript (nama standar resmi dari JavaScript, Namun JavaScript yang lebih dikenal dalam implementasinya), sehingga tidak perlu ada kekhawatiran bahwa browser tidak mendukung fitur-fitur di Node.js.

2.2.14 POSTGRESQL

PostgreSQL adalah sebuah Object Relational Management System (ORDBMS) yang dikembangkan oleh Berkeley Computer Science Development. PostgreSQL juga menawarakan tambahan-tambahan yang cukup signifikan yaitu class, type, function dan lain-lain. PostgreSQL adalah sebuah sistem basis data yang disebarluaskan secara bebas menurut Perjanjian lisensi BSD. Piranti lunak ini merupakan salah satu basis data yang paling banyak digunakan saat ini, selain MySQL dan Oracle. PostgreSQL menyediakan fitur yang berguna untuk replikasi basis data. Fitur-fitur yang disediakan PostgreSQL antara lain DB Mirror, PGPool, Slony, PGCluster, dan lain-lain. Berikut kelebihan dari PostgreSQL:

- Terdapat fitur OO: Mempunyai fitur OO berarti juga keunggulan dari PostgreSQL untuk dapat mendefinisikan tabel-tabel dan mewarisi table yang lain untuk dapat digunakan. Fitur yang berguna untuk mewarisi tabel, dan tipe data (tipe data array) akan menjadikan PostgreSQL lebih praktis dalam penyimpanan item data dalam jumlah banyak di dalam satu recordnya.
- Memiliki Arstitektur Multiproses: Arsitektur multiproses (forking) yang dimiliki oleh PostgreSQL menjadikan PostgreSQL mempunyai stabilitas yang tinggi. Hal ini memungkinkan PostgreSQL bekerja lebih powerfull dikarenakan seluruh daemon tidak akan mati meskipun mengalami satu proses anak mati. Hal ini memang dahulu sering terjadi namun setelah melalui berbagai macam pengembangan, tidak perlu lagi khawatir mengenai arsitektur multiproses yang telah dikembangkan di dalamnya.
- Memiliki Kecepatan Meski Dalam Load Tinggi: PostgreSQL didapuk memiliki kecepatan yang tinggi bahkan hingga mengalahkan kecepatan MySQL dalam hal query dengan klausa JOIN dengan tingkat yang kompleks meskipun di dalam kondisi load yang tinggi/ memiliki jumlah koneksi simultan yang besar. Hal ini disebabkan karena PostgreSQL telah mendukung locking level di bagian yang lebih rendah, yakni pada locking level row.
- Memiliki Tipe Data Geometri: PostgreSQL juga akan semakin memudahkan pengguna dikarenakan database system ini telah support berbagai tipe data geometri. Tipe data geometri yang disupport nya yakni seperti data titik, garis, lingkaran dan polygon yang seringkali berguna untuk aplikasi-aplikasi tertentu (aplikasi ilmiah dan sebagainya).

- Dapat Mendefinisikan Field Sebagai Array: Tidak hanya itu saja, PostgreSQL juga dapat menerjemahkan / mendefinisikan field yang biasanya belum dapat terdefinisi otomatis sebagai array.
- Memiliki Rule: Rule merupakan suatu tindakan custom yang dapat didefinisikan sebagai tereksekusi pada saat terdapat sebuah tabel yang di Delete, Update maupun di Insert pada database system tersebut.
- Mampu Menampung Data Spasial: Seperti yang telah diketahui, penyimpanan mengenai data spasial bukanlah suatu hal yang mudah dan simple. Penyimpanan yang perlu disediakan untuk data-data spasial perlu ruang yang cukup besar. PosgreSQL memiliki keunggulan dalam hal tersebut sehingga tidak dapat dipungkiri lagi bahwa database system ini dapat digunakan dalam membuat situs yang dengan basis untuk pemetaan dan lain sebagainya misalnya Web GIS.

2.2.15 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkaan dan dibuat (*generate*) oleh computer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman JavaScript, (Standar ECMA-262 Edisi ke 3, 1999). JSON terbuat dari dua struktur yaitu:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), table hash (*hash table*), daftar kunci (*keyed list*), atau *associative array.*

2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan Bahasa, hal ini dinyatakan seagai larik (*array*), vector (*vektor*), daftar (*list*), atau urutan (*sequence*).