BAB 2
TINJAUAN PUSTAKA

2.1 Profil Perusahaan

Qwords.com Web Hosting adalah perusahaan penyedia jasa layanan *web hosting* (*web hosting provider*) yang melayani registrasi nama domain & *web hosting* sejak Juli 2005. Qwords.com menyediakan jasa layanan registrasi nama domain dan *web hosting* yang sangat murah dan cocok untuk kebutuhan anda.

Qwords.com menyediakan *server* dengan kualitas terbaik dan juga cepat diakses dari seluruh dunia bukan hanya dari Indonesia saja, sehingga mampu memberikan pelayanan yang prima bagi pengunjung yang mengakses situs klien kami.

Berdiri pada 10 Juli 2005, dan mulai menggunakan *brand* Qwords.com sejak 28 Agustus 2005 Qwords.com mulai membuka lapangan kerja baru bagi mahasiswa juga *fresh graduate* yang memiliki kompetensi tinggi pada saat itu, saat ini Qwords.com merupakan salah satu tempat tujuan kerja yang diminati oleh berbagai lulusan perguruan tinggi di Kota Bandung.

Qwords.com melayani salah satu proses bisnis dari perusahaan-perusahaan yang membutuhkan solusi IT dan komunikasi agar perusahaannya lebih berkembang dan juga dikenal oleh seluruh dunia melalui *media* Internet.

Qwords.com memiliki komitmen dalam melayani pelanggan adalah memberikan pelayanan semaksimal mungkin dengan harga terjangkau dan kualitas yang optimal, demi mencapai kepuasan pelanggan kami serta memberikan solusi bagi usahanya.

Dalam enam tahun perjalanannya memberikan layanan registrasi nama domain dan *web hosting*, Qwords.com berkembang amat pesat dengan melayani lebih dari 6000 klien aktif, dan terus berkembang setiap harinya, Qwords.com juga masuk kedalam jajaran 10 besar perusahaan *web hosting* di Indonesia , dan merupakan perusahaan *web hosting* nomor satu di Jawa Barat . [7]

2.1.1 Sejarah Perusahaan

Qwords.com didirikan pada bulan Juli 2005 oleh Rendy Maulana saat ia belajar di Institut Teknologi Bandung (ITB). Berawal dari keinginan memiliki perusahaan yang menunjang aktivitas teman kolasenya (pada saat itu) terutama di pihak internet. Ia memulai dengan membuka layanan pembuatan *website* dan layanan penyimpanan data *website* atau *web hosting*. Sejak saat itu, *Web hosting* adalah bisnis inti Qwords.com. Kami menyediakan ruang sewa untuk menempatkan file situs web dan email yang diakses di Internet. Kami juga menyediakan layanan registrasi untuk nama domain atau alamat untuk mengakses situs anda di internet. Kemudian pada tahun 2007 Qwords membuka kantor pertamanya di Bandung yang beralamat di Cisitu Lama no 48.

Qwords memiliki server pribadinya di Indonesia pada Tahun 2008 yang disimpan di Gedung Cyber Jakarta, didorong oleh peningkatan jaringan bisnis dengan kuat, Qwords kemudian membuka kantor operasional kedua di ibu kota, di Cyber Building 1, Jakarta Selatan, Pada saat itu juga Qwords secara formal berada di bawah naungan PT. Qwords Company International

Seiring berkembangnya perusahaan, untuk meningkatkan jaringan bisnis Qwords membuka kantor ketiga di D.I Yogyakarta di Tahun 2015 disusul tahun berikutnya membuka kantor keempat di Kota Surabaya. Tahun 2016 Qwords menjadi Panitera Nama Domain Internet Indonesia atau "Pengelola Nama Domain Internet Indonesia" (PANDI). Menangani lebih dari 45.000 nasabah aktif dari dalam dan luar negeri.[11]

2.1.2 Visi dan Misi Perusahaan

Visi Qwords.com kedepan adalah menjadi perusahaan yang memberikan layanan IT *Business* kepada perusahaan-perusahaan yang sedang berkembang atau yang telah mapan, guna mempermudah proses bisnis mereka.

Misi Qwords.com adalah memasyarakatkan IT *Business Service* hingga ke level UKM untuk mempermudah proses bisnis dengan harga yang terjangkau dan kualitas yang tinggi. [7]

2.1.3 Logo Perusahaan

Adapun logo perusahaan PT. Qwords Company International ditunjukan pada Gambar 2.1. berikut ini.



Sumber Gambar : https://www.qwords.com/about-qwordscom/penggunaan-merek-dan-logo/

Gambar 2.1. Logo PT Qwords Company International

2.2 Profil PANDI

Pengelola Nama Domain Internet Indonesia (PANDI) adalah Registri Nama Domain Tingkat Tinggi Indonesia (.ID) yang ditetapkan oleh Menteri Komunikasi dan Informatika (Kominfo) Republik Indonesia dengan Surat Keputusan nomor 806 tahun 2014.

PANDI adalah badan hukum berbentuk perkumpulan, beranggotakan individu-individu yang berasal dari *multistakeholder* internet Indonesia. Keanggotaan PANDI mencerminkan keterwakilan dari Pemerintah Republik Indonesia, kalangan akademisi, dan kalangan usaha.

Untuk menjalankan pengelolaan nama domain .ID, setiap empat tahun sekali anggota PANDI memilih Dewan Eksekutif dalam sebuah Rapat Umum Anggota. Dewan Eksekutif kemudian memilih Ketua PANDI yang sekaligus menjabat sebagai Direktur Utama PANDI. Dewan Eksekutif juga menetapkan direktur lainnya dalam Dewan Direktur yang diusulkan oleh Direktur Utama. Dewan Direktur adalah tenaga profesional yang bekerja penuh waktu di PANDI.

PANDI didukung *staff-staff* berkemampuan dan berdedikasi tinggi yang selalu mengembangkan diri mengikuti perkembangan teknologi dan perubahan dalam masyarakat pengguna internet – di Indonesia maupun di dunia internasional.[10]

2.2.1 Sejarah PANDI

PANDI merupakan organisasi nirlaba yang dibentuk pada 29 Desember 2006 oleh pemerintah Republik Indonesia bersama komunitas internet Indonesia. PANDI dibentuk untuk mengelola nama domain .ID secara profesional, akuntabel, dan transparan sesuai dengan kaidah hukum Republik Indonesia.

Pada 29 Juni 2007, Departemen Kominfo RI secara resmi menyerahkan pengelolaan nama domain .ID kepada PANDI. Penyerahan ini tertuang dalam Berita Acara Penyerahan Pengelolaan Domain .id no. BA–343/DJAT/MKOMINFO/6/2007 dari Dirjen Aptel Kominfo Republik Indonesia.[10]

2.2.2 Visi dan Misi PANDI

Visi PANDI adalah menjadi pengelola nama domain internet Indonesia yang terpercaya, aman, stabil dan handal.

Misi PANDI adalah

1. Memastikan sistem layanan registri dan domain *name system* beroperasi dengan baik, stabil, aman, dan terpercaya.
2. Meningkatkan jumlah nama domain .id.
3. Melibatkan partisipasi publik dan bekerja sama dengan pemerintah di dalam proses pembentukan kebijakan.
4. Berperan aktif dalam komunitas *global* dan pembentukan kebijakan tata kelola internet *global*.
5. Memfasilitasi penyelesaian perselisihan nama domain sesuai ketentuan peraturan perundangan yang berlaku.
6. Berperan aktif dalam riset dan pengembangan teknologi yang terkait dengan internet. [12]

2.2.3 Logo PANDI

Adapun logo PANDI ditunjukan pada Gambar 2.2. berikut ini.

Sumber Gambar: https://pandi.id

Gambar 2.2. Logo PANDI

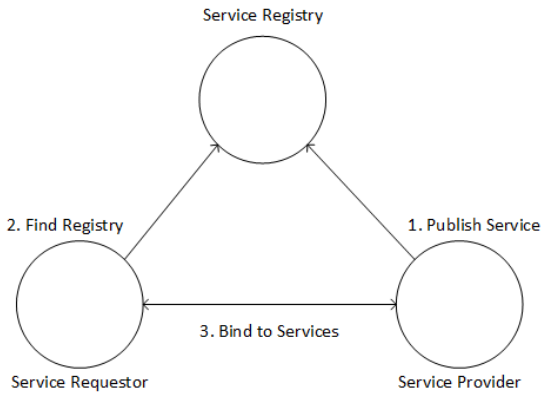
2.3 *Web Service*

Web Service atau API (*Application Programming Interface*) adalah suatu sistem perangkat lunak yang dirancang untuk mendukung *interaction* and *interoperability* antar sistem pada suatu jaringan.

Web service digunakan sebagai suatu fasilitas yang menyediakan layanan (dalam bentuk informasi atau data) kepada sistem lain, sehingga dapat berinteraksi dengan sistem tersebut melalui layanan-layanan yang disediakan. *Web service* menyimpan data informasi dalam format JSON atau XML, sehingga data ini dapat diakses oleh sistem lain walaupun berbeda platform, sistem operasi, dan bahasa pemrograman.[13]

2.3.1 Arsitektur *Web Service*

Secara umum, arsitektur *web service* dapat dilihat pada gambar 2.3 berikut ini



Gambar 2.3 Arsitektur *Web Service*

Pada gambar diatas, ada tiga komponen utama dari *web service* yaitu:

1. ***Service Provider*** adalah penyedia *web service* yang berfungsi menyediakan kumpulan *web service* yang dapat diakses oleh pengguna.
2. ***Service Requestor*** adalah aplikasi yang bertindak sebagai pengguna yang melakukan permintaan layanan (berupa *web service*) ke *service provider*
3. ***Service Registry*** adalah tempat dimana *service provider* mempublikasikan layanannya. Pada arsitektur *web service*, s*ervice registry* bersifat opsional. [14]

2.3.2 Strategi API

Dalam kebanyakan kasus, sangat bermanfaat bagi penyedia API untuk menentukan kasus penggunaan primer untuk API. Ini satu dari pertimbangan penyedia API yang paling penting untuk dibuat. API mungkin ditargetkan untuk penggunaan internal dalam bisnis, atau untuk mitra atau tujuannya adalah membuat data dapat diakses oleh pengembang perorangan di seluruh dunia. Tiga pendekatan utama untuk strategi API diidentifikasi oleh Boyd (2014 & 2015) disebut private, partner dan public. Juga Willmott (2015) menggunakan klasifikasi yang sangat mirip. Pendekatan yang relevan bergantung pada model bisnis penyedia API dan penyedia API lingkungan bisnis beroperasi. Secara teknis API diimplementasikan di bawah pendekatan ini mungkin sama, namun pendekatan yang dipilih menentukan bagaimana penyedia API memprioritaskan berbagai fitur dan pemangku kepentingan API yang berbeda.

1. *Private* API

Private API ditargetkan untuk pengembang yang bekerja dalam internal perusahaan. *Private* API sering digunakan untuk membuat beberapa aplikasi atau kumpulan data dapat diakses dalam internal perusahaan sehingga mudah digunakan kembali di aplikasi lain atau unit bisnis. Dengan cara ini *Private* API dapat digunakan untuk mengurangi duplikat pemrosesan informasi antar departemen. Selain itu *Private* API dapat digunakan sebagai blok bangunan arsitektur *microservices*, yaitu varian yang lebih modern dan lebih ringan dari

pendekatan arsitektur berorientasi layanan yang lebih lama (Bradbury, 2016). Dalam pendekatan ini model aplikasi tradisional dipecah menjadi beberapa layanan kecil yang saling berbicara melalui API. *Microservices* bisa membawa lebih banyak kecepatan dan kelincahan untuk pengembangan perangkat lunak. Aplikasi yang lebih kecil dapat lebih cepat berkembang dan mudah berubah, berkat API ada sedikit independensi bila dibandingkan dengan aplikasi model lama/tradisional.

Private API juga sering digunakan saat ada kebutuhan untuk mengakses data melalui *mobile* seperti aplikasi iPhone atau Android. Banyak profesional API seperti Lane (2012a) merekomendasikan agar organisasi memulai eksperimen mereka dengan *Private* API lalu pindah ke *Partner* dan *Public* API saat mereka telah mengumpulkan pengalaman dalam pengelolaan dan pembangunan API yang dibutuhkan. Pendekatan *Private* API memungkinkan penyedia API untuk melihat masalah dan tantangan terkait dengan API, namun kegagalan hanya akan mempengaruhi penyedia API itu sendiri. Satu perusahaan yang menggunakan pendekatan ini adalah Amazon, salah satu tim mengekspos data dan fungsinya ke tim lain melalui API. Pendekatan ini memungkinkan Amazon untuk tetap terorganisir dan tangkas, sementara ada beberapa tim yang bekerja dengan berbagai layanan infrastruktur sistem diwaktu yang sama.

Operasi dan dukungan memerlukan lebih sedikit sumber daya dalam pendekatan *Private* API daripada yang pendekatan lainnya. Penyedia API harus tetap berusaha membuat dokumentasi API berkualitas tinggi dan memberikan dukungan yang diperlukan kepada pengembang sehingga mereka dapat menggunakan API dan mewujudkan kesempatan yang diberikan oleh API. *Private* API biasanya tidak didokumentasikan secara terbuka.

2. *Partner* API

Dalam pendekatan *Partner* API, penyedia API menggunakan API untuk mengintegrasikan dan mengotomatisasi interaksi dengan mitra-mitranya. Pendekatan ini populer terutama di kalangan bisnis yang beroperasi di lingkungan pasar *business-to-business*. Dokumentasi yang terdefinisi dengan baik akan memudahkan integrasi proses bisnis saat hubungan bisnis baru

sedang dibangun. API mengurangi kebutuhan akan penyesuaian integrasi sistem-ke-sistem dan memudahkan komunikasi antara mitra yang telah dibangun dan didokumentasikan terlebih dahulu dan diimplementasikan menggunakan teknologi yang standar.

Partner API juga memungkinkan untuk memperdalam hubungan dengan mitra dengan mengintegrasikan proses bisnis lebih ketat dan dengan menawarkan data operasional *real time partner*.

Kasus khusus penggunaan untuk *Partner* API adalah faktur otomatis, sistem pemesanan dan pembayaran, produk dan katalog harga dan berbagai jenis pencarian status. Pendekatan *Partner* API biasanya membutuhkan lebih banyak sumber daya dibanding pendekatan *Private* API, karena penyedia API harus mendukung juga eksternal pengembang yang bekerja untuk subkontraktor mitra atau mitranya. Di lingkungan *Partner* API saat ada pengembangan atau fitur baru perubahan tidak boleh merusak integrasi yang ada dengan sistem mitra yang sudah ada. Dokumentasi *Private* API mungkin bersifat publik, namun untuk mendapatkan akses ke API biasanya memerlukan hubungan bisnis dengan penyedia API.

3. *Public* API

Public API didokumentasikan secara terbuka dan dapat diakses oleh pihak manapun yang ingin mendaftar. Pada kasus paling baik *Public* API memungkinkan penyedia API untuk memanfaatkan kreativitas pengembang pihak ketiga yang tak terhitung jumlahnya. *Public* API memberi kesempatan kepada penyedia API untuk mengubah produknya menjadi platform, yang mana pengembang pihak ketiga dapat memperpanjang dan memasukkannya ke dalam aplikasi mereka sendiri. Itu satu alasan mengapa perusahaan online berorientasi platform besar seperti Facebook, Twitter dan Google miliki banyak berinvestasi dalam pengembangan API. Juga penyedia *Software-as-service* yang melayani perusahaan kecil dan menengah biasanya memungkinkan untuk memperluas layanan mereka melalui *Public* API.

Pendekatan *Public* API juga disukai oleh organisasi sektor publik seperti pemerintah kota, karena data yang mereka miliki sering bersifat publik dan harus dibuat semudah diakses untuk semuanya.

Mengembangkan dan mendukung *Public* API memerlukan lebih banyak sumber daya dari penyedia API daripada dua pendekatan sebelumnya. Penyedia API harus memberikan dukungan bagi pengembang yang mungkin tidak sangat akrab dengan produknya. Menurut banyak profesional API seperti Bortenschlager (2015) *Developer Experience* (DX) adalah salah satu faktor terpenting *program* API yang berhasil terutama dalam kasus *Public* API. Istilah pengalaman pengembang mengacu pada total pengembang bekerja dengan API yang diberikan. Pengalaman pengembang adalah sejumlah hal seperti kualitas Dokumentasi API, daya tanggap dukungan teknis, kualitas teknis API, kemudahan memulai dengan pekerjaan pengembangan dan kualitas alat pengembangan perangkat lunak yang tersedia untuk API. Untuk mendapatkan pengalaman pengembang mungkin memerlukan investasi yang signifikan dari pemberi API. Agar berhasil melibatkan pengembang pihak ketiga ke dalam pengembangan produk penyedia API harus memiliki data, yang mengilhami pengembang atau perusahaan dapat memberi insentif seperti pembagian keuntungan kepada pengembang.[4]

2.4 *Representational State Transfer* (REST)

REST adalah filosofi desain yang mendorong kita untuk menggunakan protokol dan fitur yang sudah ada pada *web* untuk memetakan permintaan terhadap sumber daya pada berbagai macam representasi dan manipulasi data di Internet (Scribner, 2009).

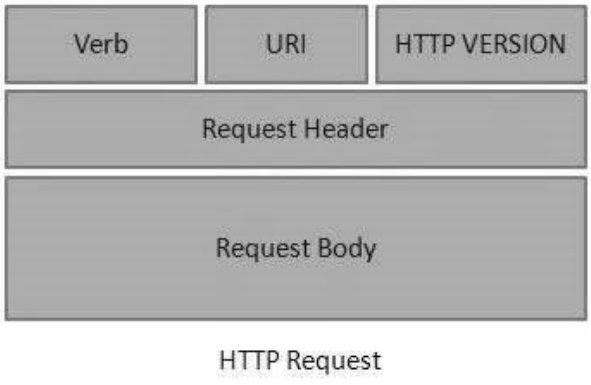
REST adalah gaya arsitektural yang memiliki aturan seperti antar muka yang seragam, sehingga jika aturan tersebut diterapkan pada *web service* akan dapat memaksimalkan kinerja *web service* terutama pada performa, skalabilitas, dan kemudahan untuk dimodifikasi. Pada arsitektur REST data dan fungsi dianggap sebagai sumber daya yang dapat diakses lewat *Uniform Resource Identifier* (URI), biasanya berupa tautan pada *web*.

REST menggunakan protokol HTTP yang bersifat *stateless*. Perintah HTTP yang bisa digunakan adalah fungsi GET, POST, PUT atau DELETE. Hasil yang dikirimkan dari *server* biasanya dalam bentuk format XML atau JSON sederhana tanpa ada protokol pemaketan data, sehingga informasi yang diterima lebih mudah dibaca dan di*parsing* disisi client.

Dalam penerapannya, REST lebih banyak digunakan untuk *web service* yang berorientasi pada *resource*. Maksud orientasi pada sumber daya adalah orientasi yang menyediakan sumber daya sebagai layanannya dan bukan kumpulan-kumpulan dari aktifitas yang mengolah sumber daya itu. Bentuk *web service* menggunakan REST *style* sangat cocok digunakan sebagai *backend* dari aplikasi berbasis *mobile* karena cara aksesnya yang mudah dan hasil data yang dikirimkan berformat JSON sehingga ukuran *file* menjadi lebih kecil. [14]

2.4.1 *HTTP Request* dan *HTTP Response*

Restful *web service* memanfaatkan protokol HTTP sebagai media komunikasi antara klien dan *server*. Klien mengirim pesan berupa Permintaan HTTP dan *server* merespons dalam bentuk HTTP *response*. Teknik ini disebut sebagai *messaging*. Pesan ini berisi data pesan dan *metadata* yaitu informasi tentang pesan itu sendiri. Mari kita lihat pesan HTTP *request* untuk HTTP 1.1 pada Gambar 2.4 berikut ini.



Sumber Gambar: https://www.tutorialspoint.com/restful/restful_quick_guide.htm

Gambar 2.4 HTTP *Request*

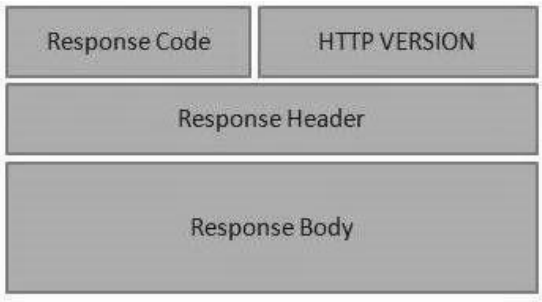
HTTP *Request* memiliki 5 bagian utama:

a. Verb - Mengindikasikan metode HTTP seperti GET, POST, DELETE, PUT, dll.
b. URI - *Uniform Resource Identifier* (URI) untuk mengidentifikasi *resource* pada server.
c. HTTP *Version* - Menunjukkan versi HTTP. Sebagai contoh, HTTP v1.1.
d. *Request Header* - Berisi metadata untuk pesan HTTP *request* sebagai pasangan *keyvalue*. Misalnya, jenis klien (atau *browser*), format yang didukung oleh klien, *format body message*, pengaturan *cache*, dll.
e. *Request Body* - Isi pesan atau representasi *resource*. [17]

Contoh sintaks HTTP *Request*:

```
POST /rest/register_domain
Host: registrarqwords.id
Cache-control: no-cache
```

Mari kita lihat pesan HTTP *Response* untuk HTTP 1.1 pada Gambar 2.5 berikut ini.



Sumber Gambar: https://www.tutorialspoint.com/restful/restful_quick_guide.htm

Gambar 2.5 HTTP *Response*

HTTP *Response* memiliki 4 bagian utama:

a. Kode Status / Respon - Menunjukkan status *server* untuk sumber yang diminta. Misalnya, 404 berarti *resource* tidak ditemukan dan 200 berarti *response* tidak apa-apa.
b. Versi HTTP - Menunjukkan versi HTTP. Misalnya HTTP v1.1.

c. *Response Header*- Berisi *metadata* untuk pesan HTTP *response* sebagai pasangan *keyvalue*. Misalnya, panjang konten, jenis konten, tanggal tanggapan, jenis *server*, dll.
d. *Response Body* - *Response* konten pesan atau representasi r*esource*. [17]

Contoh sintaks HTTP Response (json):

```
{
      "code":"1000",
      "msg":"Domain Telah Berhasil Diaktivasi."
}
```

2.4.2 *Addressing* REST

Addressing mengacu pada mencari sumber daya atau banyak sumber yang tergeletak di *server*. Ini adalah analog untuk menemukan alamat pos seseorang.

Setiap *resource* dalam arsitektur REST diidentifikasi oleh URI (*Uniform Resource Identifier*). Sintaks format URI adalah sebagai berikut:

```
<protocol>://<service-name>/<ResourceType>/<ResourceID>
```

Tujuan URI adalah untuk mencari sumber daya pada *server* yang menghosting layanan *web*. Atribut penting lain dari permintaan adalah *VERB* yang mengidentifikasi operasi yang akan dilakukan pada sumber daya.

Berikut adalah poin penting yang harus dipertimbangkan saat merancang URI:

a. Hindari penggunaan spasi - Gunakan garis bawah (_) atau tanda hubung (-) bila menggunakan nama sumber daya yang panjang. Misalnya, gunakan authorized_users sebagai ganti authorized%20users
b. Gunakan huruf kecil - Meskipun URI tidak sensitif huruf, praktik yang baik adalah menyimpan url dalam huruf kecil saja.
c. Menjaga Kompatibilitas - Sebagai *web service* untuk publik, URI yang pernah dibuat publik harus selalu tersedia. Dalam kasus, URI akan diperbarui, *redirect* URI yang URI lama ke URI baru menggunakan kode HTTP Status, 300.
d. Gunakan HTTP *Verb* - Selalu gunakan HTTP *Verb* seperti GET, PUT dan DELETE untuk melakukan operasi pada *resource*. Tidak baik menggunakan nama operasi di URI. [17]

Contoh URI seperti: https://registrarqwords.id/rest/register_domain dengan *VERB* berupa POST.

2.4.3 HTTP *Verbs*

Seperti yang telah kita bahas di bab-bab sebelumnya bahwa *restful web service* menggunakan banyak HTTP *verbs* untuk menentukan operasi yang akan dilakukan pada sumber daya yang ditentukan. HTTP *verbs* yang paling umum digunakan dapat dilihat pada Tabel 2.1 berikut ini.

Tabel 2.1. HTTP *Verbs*

No	HTTP *Method*, URI and *Operation*
1	GET http://localhost:8080/UserManagement/rest/UserService/users *Gets the list of users.* (*Read Only*)
2	GET http://localhost:8080/UserManagement/rest/UserService/users/1 *Gets the User of* Id 1 (*Read Only*)
3	PUT http://localhost:8080/UserManagement/rest/UserService/users/2 *Inserts User with* Id 2 (*Idempotent*)
4	POST http://localhost:8080/UserManagement/rest/UserService/users/2 *Updates the User with* Id 2
5	DELETE http://localhost:8080/UserManagement/rest/UserService/users/1 *Deletes the User with* Id 1 (*Idempotent*)

No	HTTP *Method*, URI and *Operation*
6	OPTIONS http://localhost:8080/UserManagement/rest/UserService/users *Lists out the supported operations in a web service*. (*Read Only*)
7	HEAD http://localhost:8080/UserManagement/rest/UserService/users *Returns the* HTTP *Header only*, *no Body*. (*Read Only*)

Poin berikut harus dipertimbangkan saat memilih HTTP *Verb* yang akan digunakan.

a. Operasi GET bersifat *read only* dan *safe*
b. Operasi PUT dan DELETE bersifat *idempoten*, yang berarti hasilnya akan selalu sama, tidak peduli berapa kali operasi ini dipanggil.
c. Operasi PUT dan POST hampir sama, dengan perbedaan hanya pada hasil dimana operasi PUT bersifat *idempotent* sedangkan operasi POST dapat menyebabkan hasil yang berbeda.

2.5 Nama Domain

Nama domain adalah nama situs Anda. Nama domain adalah alamat dimana pengguna internet dapat mengakses situs Anda. Nama domain digunakan untuk menemukan dan mengidentifikasi komputer di Internet. Komputer menggunakan alamat IP, yang merupakan rangkaian nomor. Namun, sulit bagi manusia untuk mengingat angka. Karena itu, nama domain dikembangkan dan digunakan untuk mengidentifikasi entitas di Internet daripada menggunakan alamat IP.

Nama domain bisa berupa kombinasi huruf dan angka, dan bisa digunakan dalam kombinasi berbagai ekstensi nama domain, seperti .com, .net dan lainnya.

Nama domain harus didaftarkan sebelum bisa menggunakannya. Setiap nama domain itu unik. Tidak ada dua situs *web* yang memiliki nama domain yang

sama. Jika seseorang mengetik di www.yourdomain.com, itu akan masuk ke situs *web* Anda dan tidak ada yang lain. [36]

2.5.1 TLD

Top Level Domain (TLD) merujuk ke segmen terakhir dari nama domain, atau bagian yang mengikuti segera setelah simbol "titik". TLD terutama dikelompokkan menjadi dua kategori: generic TLD dan TLD spesifik negara atau ccTLD. Contoh beberapa TLD populer termasuk .com, .org, .net, .gov, .biz dan .edu. *The Internet Corporation for Assigned Names and Numbers* (ICANN), adalah entitas yang mengkoordinasikan domain dan alamat IP untuk internet.

Secara historis, TLD mewakili tujuan dan jenis domain. ICANN umumnya sangat ketat untuk membuka TLD baru, namun pada tahun 2010, ADB memutuskan untuk mengizinkan pembuatan berbagai TLDs generik baru dan juga TLD untuk merek dagang khusus perusahaan. TLD juga dikenal sebagai sufiks domain.[37]

2.5.2 Jenis-jenis TLD

1. ccTLD

CcTLD adalah kode negara ekstensi TLD yang dikhususkan untuk setiap negara di dunia. Setiap negara memiliki ekstensi domain mereka sendiri berdasarkan kode negara alfa-2 ISO 3166-1. Ini berarti bahwa semua ccTLD hanya dua karakter, dan semua dua karakter TLDs adalah ccTLD. CcTLD pertama yang didelegasikan adalah AS untuk Amerika Serikat, .UK untuk Inggris, dan .IL untuk Italia pada tahun 1985.

Meskipun beberapa ccTLD dibatasi hanya untuk digunakan oleh warga negara tersebut (contohnya adalah Canada's .CA), banyak yang memiliki kebijakan registrasi terbuka. Beberapa ccTLD bahkan mencap dirinya untuk penggunaan alternatif. Contohnya adalah domain .TV Tuvalu, yang dicap untuk penggunaan *video online*. Karena beberapa ccTLD lebih sering digunakan secara umum, Google dan mesin pencari lainnya menganggapnya sebagai "gccTLD".

Beberapa ccTLD juga tidak mengizinkan registrasi langsung. Sebagai gantinya Anda hanya bisa mendaftarkan domain di tingkat kedua ccTLD, yang dikenal sebagai ccSLD atau kode negara tingkat kedua. Selain itu, beberapa ccTLD menawarkan registrasi langsung dan ccSLD untuk pendaftaran. Contoh ccSLD adalah .CO.UK dan .CO.IN (kedua ccTLD ini juga mengizinkan registrasi langsung pada .UK dan .IN).

Harap dicatat bahwa ada beberapa pengecualian pada peraturan bahwa semua ccTLD didasarkan pada kode negara ISO 1666-1 alpha-2. Salah satu pengecualian tersebut adalah .UK sebagai "GB" adalah kode negara ISO Inggris.[35]

2. gTLD

Generic TLD (gTLD) adalah kategori *Top Level Domain* (TLD) yang mudah dikenali oleh akhiran yang dilampirkan pada nama domain. Ini digunakan oleh sistem nama domain internet (DNS), dengan pengawasan oleh *Internet Assigned Numbers Authority* (IANA), yang sekarang dikendalikan oleh *Internet Corporation* for *Assigned Names and Numbers* (ICANN).

Contoh gTLD yang terkenal adalah com, org, info, net, dan biz. Generic TLD dan terbatas yang dibuat pada awal munculnya DNS memerlukan bukti kelayakan pendaftaran nama domain. TLD ini adalah gov, mil, int dan edu.

Pada tahun 2012 ICANN menerapkan program perluasan gTLD, yang meluncurkan banyak gTLD baru yang dianggap sebagai gangguan, bukan cara untuk membuka Internet dengan kemungkinan kreatif baru. TLD baru, seperti "*ninja*" dan "*unicorn*," adalah contohnya. Esther Dyson, pendiri ICANN, menyatakan bahwa ekspansi ini akan menciptakan lapangan kerja bagi pemasar dan pengacara namun memberikan nilai tambah yang kecil.

2.6 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 – Desember 1999. JSON

merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh *programmer* keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data. [20]

Penulis menggunakan metode JSON dalam pengiriman data yang dilakukan, karena JSON memiliki beberapa kelebihan - kelebihan dibandingkan XML, kelebihan – kelebihan tersebut adalah:

1. Format Penulisan

   Untuk merepresentasikan sebuah struktur data yang rumit dan berbentuk hirarkis penulisan JSON relatif lebih terstruktur dan mudah.

2. Ukuran

   Ukuran karakter yang dibutuhkan JSON lebih kecil dibandingkan XML untuk data yang sama. Hal ini tentu berpengaruh pula pada kecepatan pertukaran data, walaupun tidak signifikan untuk data yang kecil, namun cukup berarti jika koneksi yang digunakan relatif lambat untuk mengakses aplikasi *web* kaya fitur yang memanfaatkan pertukaran data. Di sini JSON lebih unggul dibandingkan XML, kecuali jika data dikompresi terlebih dahulu sebelum dikirimkan, perbedaan JSON dan XML yang telah dikompresi tidaklah signifikan.

3. *Browser Parsing*

   Proses *parsing* merupakan proses pengenalan *token* atau bagian-bagian kecil dalam rangkaian dokumen XML/JSON. Contohnya, terdapat data *text* dalam format JSON. Data tersebut harus di-*parsing* terlebih dahulu sebelum dapat diakses dan dimanipulasi. *Browser parsing* berarti proses parsing yang terjadi pada sisi *client*/*browser*. Melakukan *browser parsing* pada JSON lebih sederhana dibandingkan pada XML, JSON menggunakan *function* JavaScript eval() untuk melakukan *parsing*. Sementara dokumen XML di-*parsing* oleh XMLHttpRequest. Rata-rata survei menobatkan JSON sebagai pemenang jika diadu kecepatan *parsingnya*.

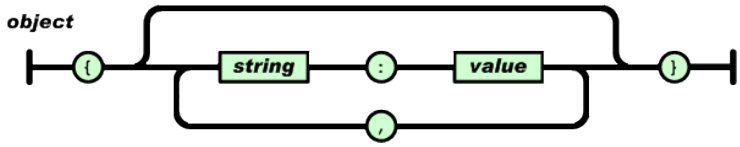
   JSON terbuat dari dua struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel *hash* (*hash* table), daftar berkunci (*keyed list*), atau *associative array*.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai struktur data *universal*. Pada dasarnya, semua bahasa pemrograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. Objek

Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).

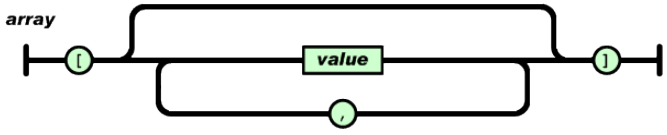


Sumber Gambar: http://json.org

Gambar 2.6 Objek JSON

2. Larik

Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).

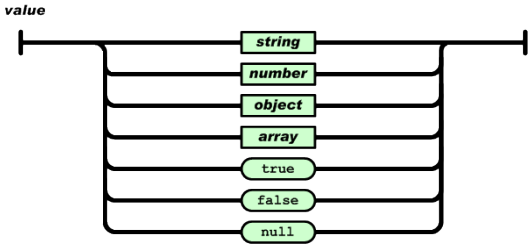


Sumber Gambar: http://json.org

Gambar 2.7 Larik JSON

3. Nilai

Nilai (value) dapat berupa sebuah *string* dalam tanda kutip ganda, atau angka, atau *true* atau *false* atau *null*, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.

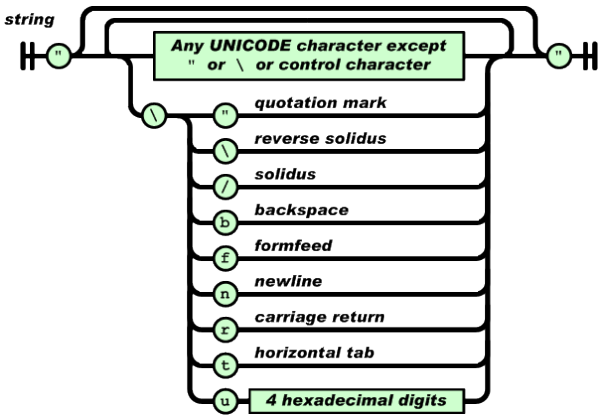


Sumber Gambar: http://json.org

Gambar 2.8 Nilai JSON

4. String

String adalah kumpulan dari nol atau lebih karakter *Unicode*, yang dibungkus dengan tanda kutip ganda. Di dalam *string* dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada *string*. *String* sangat mirip dengan *string* C atau Java.

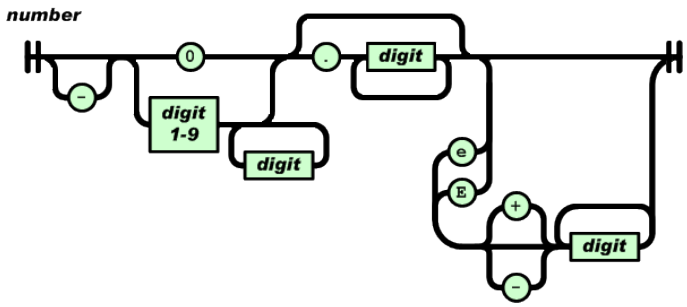


Sumber Gambar: http://json.org

Gambar 2.9 String JSON

5. Angka

Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Sumber Gambar: http://json.org

Gambar 2.10 Angka JSON

Contoh sintaks Json:

```
{
      "code":"1000",
      "msg":"Domain Telah Berhasil Diaktivasi."
}
```

2.7 PHP

PHP atau PHP:*Hypertext Processor* merupakan salah satu bahasa pemrograman di sisi *server* untuk membuat *web* dinamis. Bahasa pemrograman PHP diciptakan oleh Rasmus Lerdorf, pada tahun 1995 Lerdorf membuat sebuah alat untuk yang kemudian disebut "*Personal Home Page*" .

PHP sendiri merupakan bahasa pemrograman yang bebas dipergunakan (*open source*), dan sudah banyak digunakan oleh pengembang website. PHP juga berdiri sebagai *platform* yang mampu dijalankan di berbagai sistem operasi, baik di Windows, UNIX, Linux maupun Mac, dan juga mendukung penggunaan beberapa *database*, salah satunya adalah MySQL.

PHP juga merupakan salah satu bahasa pemrograman *website* yang mempunyai keunggulan selain bahasa pemrograman yang lain misalnya Perl, Microsoft *Active Server Page* (ASP), Java *Server Page* (JSP) atau Allaire Cold Fusion. Berbagai keunggulan dari PHP diantaranya adalah :

a. Performansi yang tinggi, PHP mempunyai performansi yang sangat efisien, dibuktikan dengan banyaknya *website* yang dibangun dengan menggunakan PHP.
b. Dapat digunakan di berbagai DBMS (*Database Management System*). PHP mendukung penggunaan *database* sebagai pendukung aplikasi. Berbagai

DBMS yang bisa terintegrasi dengan PHP diantaranya MySQL, PostgreSQL, mSQL, Oracle, dbm, filpro, Hyperware, Informix, Interbase and Sybase *databases* dan lain-lain. Dan dapat juga dengan menggunakan *Open Database connectivity Standard* (ODBC) sehingga bisa digunakan untuk produk dari Microsoft dan yang mendukung koneksi ODBC.

c. Dibangun dengan kepustakaan yang memadai untuk berbagai penggunaan web. Banyaknya *developer* yang menggunakan PHP sebagai bahasa pemrograman yang digunakan, maka PHP sendiri mendukung kepustakaan yang lengkap misalnya penggunaan pengaturan gambar dengan ekstensi GIF, JPG yang diperkecil atau diperbesar dari bentuk aslinya, pembuatan dokumen PDF, ekspor dan impor dokument EXCEL dan lain-lain
d. Harga yang murah karena sifatnya *open source*. Karena sifatnya *open source* (gratis) maka pengguna bisa bebas memakai maupun mendownload pengembangan produk PHP berbagai versi, misalnya di http://www.php.net
e. Mudah dipelajari dan digunakan. Kode program dari PHP juga merupakan atau didasarkan pada bahasa pemrograman yang lain misalnya bahasa C, Perl atau Java, sehingga dengan mempunyai atau mengetahui bahasa pemrograman tersebut secara otomatis akan mudah menggunakan bahasa pemrograman PHP.
f. Mudah diaplikasikan di beberapa sistem operasi. PHP merupakan bahasa pemrograman yang dapat digunakan di berbagai sistem operasi, misalnya Windows, Unix, Linux, FreeBSD. Sehingga tidak perlu mengubah kode program jika digunakan lintas sistem operasi.
g. Kebebasan pemakaian kode program. Pengembangan aplikasi dengan menggunakan PHP bisa juga digunakan untuk hasil aplikasi yang gratis maupun digunakan untuk hasil aplikasi yang berbayar (diperjual belikan) meskipun bahasa pemrograman PHP sendiri adalah *open source* (gratis).

2.7.1 Penggunaan Kode Program PHP

Penggunaan kode program PHP dalam yang akan diterapkan di bahasa pemrograman konsep penulisannya diawali dengan tag <?php dan diakhiri dengan

tag ?>. Atau bisa juga diawali dengan tag <? Dan diahiri dengan tag ?>. Dan pada setiap akhiran dari penulisan kode program di PHP diakhiri dengan semicolon (;) / titik koma.

Ada beberapa konsep penulisan dari tag php diantaranya :

a. Kode singkat. (*Short Style*)

Contoh :

```
<?  echo”Latihan PHP”;   ?>
```

Kode ini merupakan kode standar yang digunakan oleh para pengembang yang menggunakan php atau biasa disebut dengan SGML (*Standard Generalized Markup Language*). Penggunaan kode ini biasanya juga disebut dengan *tag* pendek (*Short Tag*)

b. Kode dengan XML (XML *Style*)

Contoh :

```
<?php  echo”Latihan PHP”;   ?>
```

Penggunaan *tag* ini dapat digunakan dengan dokumen XML (*Extensible Markup language*). Dan tag ini juga sering digunakan untuk menghindari kode *error* jika *server* tidak mendukung *short tag*.

c. Bentuk SCRIPT (SCRIPT *Style*)

Contoh :

```
<SCRIPT LANGUAGE=’php’> echo “Latihan php”; </SCRIPT>
```

Penggunaan *tag* kode program ini digunakan jika pengembangan menerapkan di dalam JavvaScript atau VBScript, dan bentuk bisa diterapkan di HTML jika *tag* yang lain tidak mendukung.

d. Bentuk ASP (ASP *Style*)

Contoh :

```
<%  echo “Latihan php”;  %>
```

Tag ini berbeda dengan *tag-tag* yang lain, *tag* ini digunakan untuk penerapan di bahasa pemrograman ASP (*Active Server Page*). Dan ini dapat digunakan dengan mengaktifkan pengaturan asp_tags.

2.7.2 Pernyataan Kode Program PHP

Kode program php yang digunakan tidak lepas dari *interpreter* (penerjemah kode program) agar bisa dikenali jika digunakan yaitu dengan menambahkan *tag* sebelum dan sesudah kode program *php* digunakan. Contoh :

```
<?php
echo "Latihan pemrograman PHP";
?>
```

Dimana dalam setiap penulisan kode program tersebut di awali dengan *tag* <?php atau <? Dan diakhiri dengan *tag* ?> . Dan yang perlu diperhatikan juga adalah semicolon (;) tanda titik koma pada setiap akhiran penulisan kode program.

Pada baris ke 3 ada pernyataan *echo*, *echo* digunakan untuk mencetak konten ke dalam *web browser*. Selain *echo*, untuk mencetak juga bisa menggunakan print. Perhatikan contoh :

```
<?php
echo "Latihan pemrograman PHP";
print("Latihan pemrograman PHP");
?>
```

Pada baris ke 2 dan ke 3 mempunyai fungsi yang sama yaitu mencetak ke halaman *web browser*. Pada pernyataan tersebut aturan penulisan jika berbentuk karakter maka penulisan konten harus di dalam tanda petik dua (" konten ") atau tanda petik (' konten ').

Selain menggunakan *echo* pencetakan pada php juga bisa menggunakan kode program di bawah ini :

```
<?="Latihan pemrograman PHP"; ?>
```

Kode program di atas sama dengan kode program di bawah ini :

```
<?php
echo "Latihan pemrograman PHP";
?>
```

2.7.3 *Whitespace* (Baris kosong / baris baru).

Di dalam penulisan kode program php di setiap akhir dari penulisan kode selalu diakhiri dengan titik koma (;). Itu bermaksud bahwa kode program tersebut sudah selesai dan bisa dilanjutkan dengan penulisan kode selanjutnya. Artinya kode program tersebut bisa ditulis satu baris dengan kode sebelumnya yang sudah diakhiri dengan titik koma (;). Perhatikan contoh :

```
<?php
echo "Latihan pemrograman PHP <br>";
echo "Sedang Belajar PHP";
?>
```

Kode diatas bisa juga dituliskan sebagai berikut :

```
<?php
echo "Latihan pemrograman PHP <br>";  echo "Sedang Belajar PHP";
?>
```

Dari contoh di atas kode program bisa dituliskan dalam satu baris ketika kode program sebelumnya sudah diakhiri dengan titik koma (;), hal ini dimaksudkan untuk menghemat ukuran *file* dengan memanfaatkan spasi yang masih kosong.

Pada contoh di atas perhatikan juga tag <br> yang merupakan *tag* HTML yang diterapkan di dalam php. Dan hal ini berlaku untuk semua *tag* di dalam html.

2.7.4 Keterangan (*Comment*) di dalam PHP.

Keterangan yang dimaksudkan di sini adalah konten / isi tidak akan ditampilkan di dalam *web browser* tetapi kode tersebut / keterangan kita tuliskan di dalam kode program (*file*).

Perhatikan contoh :

```
<?php
# echo "Latihan pemrograman PHP <br>";
//echo "Sedang Belajar PHP <br>";
/*  echo "Membuat Komentar <br>";
echo "Membuat Komentar <br>";  */ ?>
```

Ada beberapa cara penulisan untuk komentar diantaranya :

a. Tanda // (*slash* 2 kali) digunakan untuk komentar yang dituliskan dalam satu baris
b. Tanda /* komentar */ digunakan untuk komentar yang dituliskan lebih dari satu baris
c. Tanda # (pagar) digunakan untuk komentar yang dituliskan dalam satu baris.[21]

2.7.5 PHP *Framework*

Framework secara sederhana dapat diartikan kumpulan dari fungsi-fungsi/prosedur-prosedur dan *class-class* untuk tujuan tertentu yang sudah siap digunakan sehingga bisa lebih mempermudah dan mempercepat pekerjaan seorang *programmer*, tanpa harus membuat fungsi atau *class* dari awal. [24]

Ada beberapa alasan mengapa menggunakan *Framework*:

1. Mempercepat dan mempermudah pembangunan sebuah aplikasi *web*.
2. Relatif memudahkan dalam proses maintenance karena sudah ada pola tertentu dalam sebuah *framework* (dengan syarat programmer mengikuti pola standar yang ada)
3. Umumnya *framework* menyediakan fasilitas-fasilitas yang umum dipakai sehingga kita tidak perlu membangun dari awal (misalnya validasi, ORM, *pagination*, *multiple database*, *scaffolding*, pengaturan *session*, *error handling*, dll
4. Lebih bebas dalam pengembangan jika dibandingkan CMS

2.8 CURL

CURL merupakan librari php yang memungkinkan anda untuk mentransfer data melalui berbagai protokol. CURL ini, banyak digunakan sebagai cara untuk mengirim atau meminta data dari satu atau beberapa situs, termasuk didalamnya seperti interaksi API dan oAuth. Pada dasarnya permintaan dengan CURL tidak dibatasi dalam hal apapun, mirip seperti permintaan HTTP dasar, juga bisa untuk upload FTP dan yang lebih lebih kompleks interaksi otentifikasi dengan situs HTTPS tertutup. Nah pada postingan kali ini, kami akan menunjukan beberapa perbedaan sederhana penggunaan CURL antara permintaan GET dan permintaan POST.

Didalam PHP sebenarnya ada fungsi sejenis dan sangat mudah penggunaannya yaitu file_get_contents, tapi cURL memiliki beberapa kelebihan yaitu bisa menggunakan COOKIE dan bisa melakukan POST *method*. Karena ada beberapa *website* yang melindungi kontennya dengan menggunakan *Cookie* dan

POST *method*, namun saya tidak menyarankan untuk mencuri konten menggunakan cURL.

Waktu yang tepat untuk menggunakan cURL adalah ketika kita ingin mengambil sebuah data dari sebuah *website* sebagai contoh mengambil rss *timeline* dari twitter kita, mengambil data dari bmkg untuk diterjemahkan menjadi data yang lebih enak dibaca.

2.8.1 Pengaturan Dasar CURL

Sebelum kita mulai melakukan permintaan isi halaman dengan Curl, kita harus terlebih dahulu tahu *instance* dari Curl – anda dapat melakukan curl dengan memanggil fungsi curl_init(); yang mengembalikan data Curl. Fungsi ini membutuhkan satu *parameter* yang merupakan URL. [18]

a. curl_init = memulai sesi cURL
b. curl_setopt = menentukan opsi-opsi sebelum menjalankan cURL.
c. curl_exec = menjalankaan cURL.
d. curl_close = menutup sesi cURL.

Konfigurasi CURL dapat diatur dengan menggunakan metode curl_setopt(), yang mengambil tiga *parameter* yaitu: sumber Curl, konfigurasi dan nilai, berikut adalah CURLOPT optionnya :

a. CURLOPT_RETURNTRANSFER – Mengembalikan respon sebagai string
b. CURLOPT_CONNECTTIMEOUT – Mengembalikan Jumlah detik yang dihabiskan saat mencoba mengkoneksi
c. CURLOPT_TIMEOUT – Jumlah detik untuk cURL
d. CURLOPT_USERAGENT – Nilai Useragent digunakan untuk permintaan
e. CURLOPT_URL – URL untuk mengirim permintaan ke
f. CURLOPT_POST – Kirim permintaan sebagai POST
g. CURLOPT_POSTFIELDS – Array untuk data dalam permintaan POST

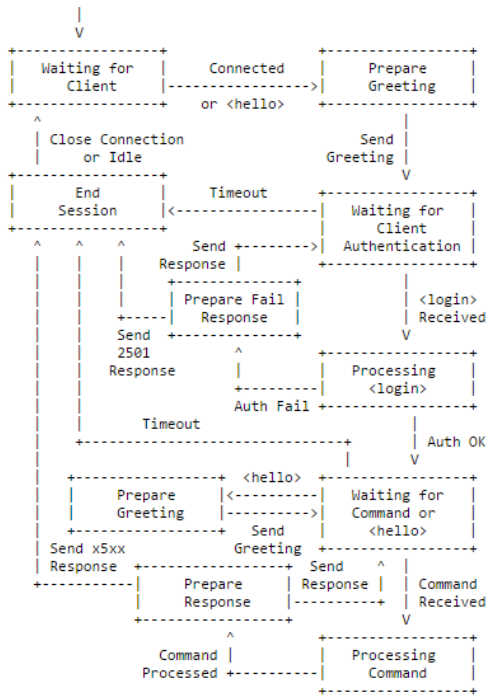
Contoh sintaks CURL:

```
<?php
      $url = ”http://localhost/curl/data.php”;
      $data = array
            (
            “name” => “farham”,
            “age”       => “31”
            );
      $ch = curl_init();
      curl_setopt($ch, CURLOPT_URL, $url);
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
      curl_setopt($ch, CURLOPT_POST, 1);
      curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
      $output = curl_exec($ch);
      curl_close($ch);
      echo $output;
```

2.9 EPP

EPP atau *Extensible Provisioning Protocol* adalah protokol XML stateful yang dapat berlapis terdiri atas beberapa protokol *transport*. Dilindungi dengan menggunakan protokol keamanan-lapisan bawah, klien bertukar identifikasi, otentikasi, dan pilihan informasi, dan kemudian terlibat dalam serangkaian perintah-perintah klien yang diprakarsai pertukaran *response*. Semua perintah EPP adalah *atomic* (tidak ada parsial keberhasilan atau parsial kegagalan) dan dirancang sehingga mereka dapat dibuat *idempoten* (mengeksekusi perintah lebih dari sekali memiliki *net* yang sama berpengaruh pada keadaan sistem seperti berhasil mengeksekusi perintah sekali).

Server EPP harus menanggapi komunikasi yang diinisiasi oleh *client* (yang dapat berupa permintaan sambungan-lapisan bawah atau *service discovery*) dengan meresponse proses *greeting* kepada *client*. Sebuah *server* harus segera menanggapi setiap perintah EPP dengan respon yang terkoordinasi yang menggambarkan hasil dari pengolahan perintah. Berikut ini diagram *server* mesin yang menggambarkan proses pertukaran pesan secara terperinci pada Gambar 2.8:



Sumber Gambar: https://tools.ietf.org/html/rfc5730

Gambar 2.11 EPP *Server State Machine*

2.9.1 Elemen EPP

Pada EPP *Server*, terdapat 5 jenis elemen yaitu:

1. Identifikasi

Semua XML EPP harus dimulai dengan elemen <epp>. elemen ini mengidentifikasi awal elemen protokol EPP dan *namespace* yang digunakan dalam protokol. Elemen <epp> sebagai elemen awal dan Elemen </ epp> sebagai elemen akhir harus diterapkan untuk semua struktur yang dikirim oleh kedua *client* dan *server*.

Contoh elemen awal dan elemen akhir dari EPP:

```
<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
</epp>
```

2. Halo

EPP *client* dapat meminta <*greeting*> dari *server* EPP setiap saat ketika berhasil menjalankan perintah <*login*> dan perintah <*logout*> dengan mengirimkan

<*hello*> ke *server*. Penggunaan ini merupakan elemen penting pada lingkungan di mana *server* tidak bisa mengembalikan <*greeting*> untuk menanggapi *client* yang memulai koneksi.

Contoh elemen Halo:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <hello/>
C:</epp>
```

3. Salam

Server EPP merespon koneksi sukses dan elemen <*hello*> dengan mengembalikan elemen <*greeting*> ke client.

Contoh elemen Salam:

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <greeting>
S:    <svID>Example EPP server epp.example.com</svID>
S:    <svDate>2000-06-08T22:00:00.0Z</svDate>
S:    <svcMenu>
S:      <version>1.0</version>
S:      <lang>en</lang>
S:      <lang>fr</lang>
S:      <objURI>urn:ietf:params:xml:ns:obj1</objURI>
S:      <objURI>urn:ietf:params:xml:ns:obj2</objURI>
S:      <objURI>urn:ietf:params:xml:ns:obj3</objURI>
S:      <svcExtension>
S:        <extURI>http://custom/obj1ext-1.0</extURI>
S:      </svcExtension>
S:    </svcMenu>
S:    <dcp>
```

4. Perintah

Client EPP berinteraksi dengan *server* EPP dengan mengirimkan perintah ke *server* untuk kemudian menerima respon dari *server*.

Contoh elemen Perintah:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <info>
C:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <obj:name>example</obj:name>
C:      </obj:info>
C:    </info>
C:      <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

5. *Response*

Server EPP merespon perintah *client* dengan mengembalikan *output response* kepada client. perintah EPP bersifat *atomic*, sehingga perintah akan selalu mengembalikan *output response* berupa berhasil sepenuhnya atau gagal sepenuhnya, tidak dapat bercampuran.

Contoh elemen respon berhasil sepenuhnya:

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <obj:creData xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:name>example</obj:name>
S:      </obj:creData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

Contoh elemen respon gagal sepenuhnya dengan pesan *error*:

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <response>
S:    <result code="2004">
S:      <msg>Parameter value range error</msg>
S:      <value xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:elem1>2525</obj:elem1>
S:      </value></result>
S:    <result code="2005">
S:      <msg>Parameter value syntax error</msg>
S:      <value xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:elem2>ex(ample</obj:elem2>
S:      </value>
S:      <extValue>
S:        <value xmlns:obj="urn:ietf:params:xml:ns:obj">
S:          <obj:elem3>abc.ex(ample</obj:elem3>
S:        </value>
S:        <reason>Invalid character found.</reason>
S:      </extValue>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

2.9.2 Jenis-jenis Perintah pada EPP

Ada 3 jenis perintah pada EPP yaitu:

2.9.2.1 Perintah Manajemen Sesi

EPP menyediakan 2 perintah untuk manajemen sesi yaitu:

1. Perintah <*login*>

Perintah <*login*> ini digunakan untuk membangun sebuah sesi dengan EPP *Server* dalam menanggapi *greeting* dari *server*. sebuah Perintah <login> harus dikirim ke *server* sebelum perintah EPP lain untuk membangun sesi yang sedang berlangsung. Sebuah *server operator* dapat membatasi jumlah dari gagal masuk mencoba N, 1 <= N <= tak terhingga, setelah satu perintah *login* gagal maka koneksi ke *server* (jika ada) akan ditutup.

Contoh perintah <*login*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <login>
C:      <clID>ClientX</clID>
C:      <pw>foo-BAR2</pw>
C:      <newPW>bar-FOO2</newPW>
C:      <options>
C:        <version>1.0</version>
C:        <lang>en</lang>
C:      </options>
C:      <svcs>
C:        <objURI>urn:ietf:params:xml:ns:obj1</objURI>
C:        <objURI>urn:ietf:params:xml:ns:obj2</objURI>
C:        <objURI>urn:ietf:params:xml:ns:obj3</objURI>
C:        <svcExtension>
C:          <extURI>http://custom/obj1ext-1.0</extURI>
C:        </svcExtension>
C:      </svcs>
C:    </login>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

2. Perintah <*logout*>

Perintah <*logout*> ini digunakan untuk mengakhiri sesi dengan *server* EPP.

Contoh perintah <*logout*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <logout/>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

2.9.2.2 Perintah Query

EPP menyediakan 2 perintah untuk *Query* yaitu:

1. Perintah <*check*>

Perintah <*check*> ini digunakan untuk menentukan apakah sebuah objek dapat ditetapkan dalam repositori atau tidak. Ini memberikan petunjuk yang memungkinkan *client* untuk mengantisipasi keberhasilan atau kegagalan penyediaan objek ketika akan menggunakan perintah <*create*>.

Contoh perintah <*check*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <check>
C:      <obj:check xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <obj:name>example1</obj:name>
C:        <obj:name>example2</obj:name>
C:        <obj:name>example3</obj:name>
C:      </obj:check>
C:    </check>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

2. Perintah <*info*>

Perintah <*Info*> ini digunakan untuk mengambil informasi terkait dengan objek yang ada. Unsur yang dibutuhkan untuk mengidentifikasi suatu objek dan jenis informasi yang terkait dengan obyek secara spesifik

Contoh perintah <*info*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <info>
C:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj">
C:      </obj:info>
C:    </info>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

2.9.2.3 Perintah Perubahan Objek

EPP menyediakan 5 perintah untuk mengubah objek: <*create*> untuk membuat obyek di *server*, <*delete*> untuk menghapus objek dari *server*, <*renew*> untuk memperpanjang validitas periode objek, <*transfer*> untuk mengelola perubahan *client* sponsor/*Registrar* dari sebuah objek, dan <*update*> untuk mengubah informasi terkait dengan objek.

1. Perintah <*create*>

Perintah <*create*> ini digunakan untuk membuat sebuah objek. Sebuah objek dapat dibuat untuk waktu yang tidak terbatas, atau objek dapat dibuat untuk masa berlaku tertentu.

Contoh perintah <*create*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <create>
C:      <obj:create xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:create>
C:    </create>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

2. Perintah <*delete*>

Perintah <*delete*> ini digunakan untuk menghapus sebuah obyek yang sudah ada.

Contoh perintah <*delete*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <delete>
C:      <obj:delete xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:delete>
C:    </delete>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

3. Perintah <*renew*>

Perintah <*renew*> ini digunakan untuk memperpanjang masa berlaku dari objek yang ada.

Contoh perintah <*renew*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <renew>
C:      <obj:renew xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:renew>
C:    </renew>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

4. Perintah <*transfer*>

Perintah <*transfer*> ini digunakan untuk mengelola perubahan *client* Sponsor/Registrar dari obyek yang sudah ada.

Contoh perintah <*transfer*>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <transfer op="request">
C:      <obj:transfer xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:transfer>
C:    </transfer>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

5. Perintah <*update*>

Perintah < *update* > ini digunakan untuk mengubah informasi terkait

dengan objek yang ada. [6]

Contoh perintah <*update*>:

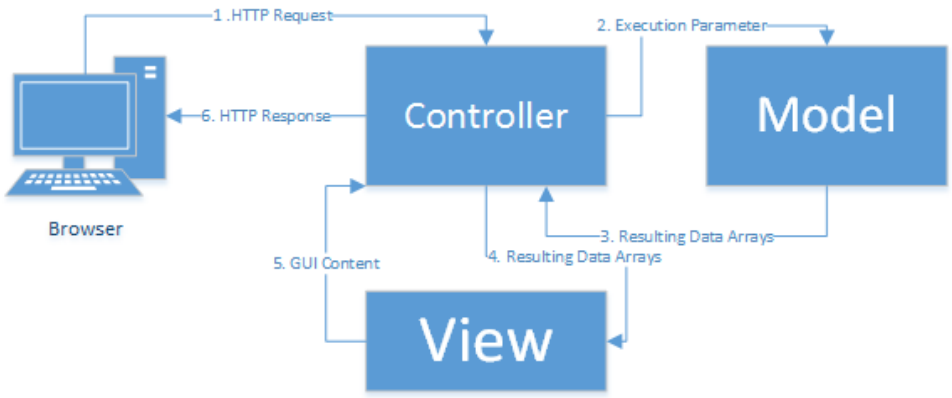
```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <update>
C:      <obj:update xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:update>
C:    </update>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

2.10 MVC (*Model, View, Controller*)

MVC (*Model-View-Controller*) adalah sebuah metode dalam membuat aplikasi dengan memisahkan pengembangan aplikasi berdasarkan komponen utama yang membangun sebuah aplikasi seperti manipulasi data, antarmuka pengguna, dan bagian yang menjadi kontrol utama aplikasi.

1. *Model*, bukan sebuah *database* tetapi merupakan bagian (fungsi-fungsi) yang berhubungan langsung dengan *database* untuk memanipulasi data seperti memasukkan data, pembaruan data, hapus data, dan lain-lain, namun tidak dapat berhubungan langsung dengan bagian *view*.
2. *View*, merupakan bagian yang mengatur tampilan ke pengguna. Pada suatu aplikasi *web* bagian ini biasanya berupa file template HTML, yang diatur oleh controller. *View* berfungsi untuk menerima dan merepresentasikan data kepada pengguna. Bagian ini tidak memiliki akses langsung terhadap bagian *model*.
3. *Controller*, merupakan bagian yang mengatur hubungan antara bagian *model* dan bagian *view*, *controller* berfungsi untuk menerima request dan data dari *user* kemudian menentukan apa yang akan diproses oleh aplikasi. [22]

Cara kerja MVC dapat dilihat di Gambar 2.12 berikut ini.



Gambar 2.12 Cara Kerja MVC

Penjelasan mengenai cara kerja MVC sebagai berikut:

1. *Browser* mengirimkan HTTP *Request* ke *website* yang menggunakan MVC
2. *Controller* menerima HTTP *Request* kemudian mengeksekusi *Model* yang sesuai
3. *Model* menjalankan *query* ke *database* kemudian hasil di kembalikan ke *Controller*
4. *Controller* mengolah hasil *query* dan memanggil *View* yang sesuai untuk ditampilkan
5. *View* mengolah PHP menjadi HTML kemudian dikirim kembali ke *Controller* yang sesuai
6. *Controller* meresponse HTTP dari *Browser*

Sebuah aplikasi yang menggunakan konsep MVC dapat terdiri dari banyak *triads* (istilah untuk satu *model-view-controller*) yang masing-masing bertanggung jawab atas elemen UI yang berbeda.

Dengan menggunakan prinsip MVC suatu aplikasi dapat dikembangkan secara terpisah antara *layer application-logic* dan *presentation*, sehingga dalam sebuah tim pengembangan *website*, seorang programmer bisa berkonsentrasi pada pengembangan *core-system* saja, sedangkan *web designer* bisa berkonsentrasi pada tampilan *web* saja. Walaupun demikian dibutuhkan komunikasi yang baik antara *programmer* dan *designer*. [25]

2.11 CodeIgniter

CodeIgniter adalah sebuah *framework* yang digunakan untuk membuat sebuah aplikasi berbasis *web* yang disusun dengan menggunakan bahasa PHP. Di dalam CI terdapat beberapa macam kelas (*class*) yang berbentuk *library* dan *helper*. Keduanya berfungsi untuk membantu pemrogram (*programmer*) dalam mengembangkan aplikasinya. Codeigniter (CI) adalah *framework* pengembangan aplikasi dengan menggunakan PHP, suatu kerangka untuk bekerja atau membuat program dengan menggunakan PHP yang lebih sistematis. MVC adalah konsep dasar yang harus diketahui sebelum mengenal Codeigniter. MVC adalah singkatan dari *Model View Controller*. MVC sebenarnya adalah sebuah teknik pemrograman yang memisahkan alur bisnis, penyimpanan data dan antarmuka aplikasi atau secara sederhana adalah memisahkan antara desain, data dan proses.[23]

Ada beberapa kelebihan CodeIgniter (CI) dibandingkan dengan *Framework* PHP lain, yaitu:

a. Performa sangat cepat : salah satu alasan tidak menggunakan *framework* adalah karena eksekusinya yang lebih lambat daripada PHP from the *scratch*, tapi Codeigniter sangat cepat bahkan mungkin bisa dibilang codeigniter merupakan *framework* yang paling cepat dibanding framework yang lain.
b. Konfigurasi yang sangat minim (*nearly zero configuration*) : tentu saja untuk menyesuaikan dengan *database* dan keleluasaan *routing* tetap diizinkan melakukan konfigurasi dengan mengubah beberapa *file* konfigurasi seperti database.php atau autoload.php, namun untuk menggunakan codeigniter dengan *setting standard*, anda hanya perlu mengubah sedikit saja *file* pada *folder* config.
c. Banyak komunitas: dengan banyaknya komunitas CI ini, memudahkan kita untuk berinteraksi dengan yang lain, baik itu bertanya atau teknologi terbaru.
d. Dokumentasi yang sangat lengkap : Setiap paket instalasi codeigniter sudah disertai *user guide* yang sangat bagus dan lengkap untuk dijadikan permulaan, bahasanya pun mudah dipahami.

Contoh sintaks *Controller* Codeigniter adalah sebagai berikut:

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access
allowed');
class Home extends CI_Controller {
      public function index(){
            echo 'Index dari controller '; }
      function test(){
            $this->load->model('webmodel');
            echo $this->webmodel->test(); }
      function contact(){
            $data['pesan'] = '';
            $this->load->model('webmodel');
            if($this->input->post('kirim')){
                  $data['pesan'] = $this->webmodel-
>proses_contact($this->input->post());}
            $this->load->view('contact',$data);}
}
```

Contoh sintaks *Model* Codeigniter adalah sebagai berikut:

```
<?php
class Webmodel extends CI_Model {
      function __construct(){
        parent::__construct(); }
      function test(){
            return 'Data untuk test model'; }
      function proses_contact($array){
            $nama = $array['nama'];
            $email= $array['email'];
            $telp = $array['telp'];
            $pesan= $array['pesan'];
            $data = 'Nama :'.$nama.'<br />'.
                        'Email:'.$email.'<br />'.
                        'Telp :'.$telp.'<br />'.
                        'Pesan:'.$pesan;
            return $data; } }
}
```

Contoh sintaks View Codeigniter adalah sebagai berikut:

```
<style type="text/css">
      label.lb{
            display:block;
            line-height:20px;
            margin:0 0 10px 0;
      }
      h1.header{
            font:bold 20px Tahoma,Verdana;
            display:block;
            margin:0 0 20px 0;
            padding:0 0 10px 0;
            border-bottom:#cecece 1px solid;
      }
</style>
<h1 class="header">Kontak Pembaca</h1>
<?php echo $pesan; ?>
<form method="post" name="frm" action="">
<label class="lb">Nama</label>
<input type="text" name="nama" size="50" />
<label class="lb">Email</label>
<input type="text" name="email" size="50" />
<label class="lb">Telp</label>
<input type="text" name="telp" size="30" />
<label class="lb">Kodepos</label>
<input type="text" name="kdpos" size="10" maxlength="6" />
<label class="lb">Pesan</label>
<textarea name="pesan" cols="50" rows="10"></textarea>
<br />
<input type="submit" name="kirim" value="Kirim">
<input type="reset" name="reset" value="Reset">
</form>
```

2.12 HTML

HTML yang merupakan singkatan dari *Hyper Text Markup* Language adalah serangkaian kode program yang merupakan dasar dari representasi visual sebuah halaman *web*. Didalamnya berisi kumpulan informasi yang disimpan dalam *tag-tag* tertentu, dimana *tag-tag* tersebut digunakan untuk melakukan format terhadap informasi yang dimaksud.

Berbagai pengembangan telah dilakukan terhadap kode HTML dan telah melahirkan teknologi-teknologi baru di dalam dunia pemrograman *web*. Kendati demikian, sampai sekarang HTML tetap berdiri kokoh sebagai dasar dari bahasa *web* seperti PHP, ASP, JSP dan lainnya. Bahkan secara umum, mayoritas situs *web* yang ada di Internet pun masih tetap menggunakan HTML sebagai teknologi utama mereka.[26]

Contoh sintaks dasar HTML

```
<span style="font-size: small;"><html>
<head>
<!-ini adalah bagian header-->
<title>judul web</title>
</head>
<body>
<!-tempat menuliskan kode-->
</body>
</html>
</span>
```

2.13 Bootstrap

Bootstrap adalah *framework* HTML dan CSS yang berfungsi untuk situs *web* dan aplikasi web. Bootstrap berisi HTML dan CSS desain berbasis *template* untuk tipografi, bentuk, tombol, navigasi dan komponen antarmuka lainnya, serta opsional JavaScript ekstensi. Hal ini bertujuan untuk memudahkan pengembangan dari *website* dinamis dan aplikasi *web*. Karena ini adalah *framework* HTML & CSS, *front-end* para "*programmer*" lah sering menggunakan keandalan dari bootstrap ini. Tetapi, tak menutup kemungkinan *back-end programmer* juga ikut menggunakan *framework* yang satu ini.

Bootstrap dilengkapi dengan beberapa komponen JavaScript dalam bentuk jQuery *plugin*. Mereka menyediakan elemen antarmuka pengguna tambahan seperti kotak dialog, *tooltips*. Bootstrap juga memperluas fungsi beberapa elemen antarmuka yang ada, termasuk fungsi auto-lengkap untuk bagian masukan. Dalam versi 2.0, plugin JavaScript berikut ini sudah: Modal, Dropdown, Scrollspy, Tab, Tooltip, Popover, Alert, Button, Ciutkan, Carousel dan Typeahead.

Tidak hanya tampilan statis saja yang disediakan bootstrap, anda juga dapat menambahkan tampilan yang dinamis, atau ada sedikit animasinya, sebab di fra*m*ework ini disediakan juga *plugin* Javascript. Kabar bagusnya lagi, *framework* ini mensupport *web responsive*. Jadi, tampilan akan beralih ukurannya tergantung pada resolusi *monitor device* yang di gunakan pengguna di saat mengakses *website* dari *browser*nya.

Bootstrap, awalnya bernama Twitter *Blueprint*, dikembangkan oleh Mark Otto dan Jacob Thornton di Twitter sebagai kerangka kerja untuk mendorong konsistensi di alat internal. Sebelum Bootstrap, berbagai perpustakaan yang

digunakan untuk pengembangan antarmuka, yang menyebabkan inkonsistensi dan beban pemeliharaan tinggi.

Bootstrap kompatibel dengan versi terbaru dari Google Chrome, Firefox, Internet Explorer, Opera , dan Safari browser, meskipun beberapa *browser* ini tidak didukung pada semua *platform*.

Sejak versi 2.0 juga mendukung desain *web* responsif. Ini berarti tata letak halaman *web* menyesuaikan dinamis, dengan mempertimbangkan karakteristik dari perangkat yang digunakan (*desktop*, *tablet*, ponsel).

Dimulai dengan versi 3.0, Bootstrap mengadopsi desain ponsel-pertama filsafat, menekankan desain responsif secara default. Dan pada versi 4.0 alpha release menambahkan Sass dan flexbox dukungan.[27]

Bootstrap adalah *open source* dan tersedia di GitHub. Pengembang didorong untuk berpartisipasi dalam proyek dan membuat kontribusi mereka sendiri untuk platform.[28]

Contoh sintaks dasar Boostrap adalah sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
    <title>Latihan Bootstrap 1</title>
    <meta name="viewport" content="width=device-width,initial-
scale=1.0">
    <link rel="stylesheet" type="text/css" href="bootstrap-
3.1.1-dist/css/bootstrap.min.css">
    <style type="text/css">
        .col
       {
            box-shadow : inset 1px -1px 1px #444, inset -1px
1px 1px #444;;
            background-color: #dedef8;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="row" align="center">
            <div class="col-md-1 col">1</div>
        </div>
        <div class="row" align="center">
            <div class="col-md-2 col">2</div>
        </div>
    </div>
</body>
</html>
```

2.14 JQuery

jQuery adalah JavaScript *library* yang dirancang untuk meringkas kode-kode JavaScript, sehingga dapat menyederhanakan penulisan skrip program, sesuai dengan slogan "*write less, do more*" . jQuery pertama kali dirilis oleh John Resig pada tahun 2006, pada perkembangannya jQuery tidak hanya sebagai *framework* JavaScript, namun memiliki kelebihan antara lain.

1. Kemudahan mengakses dan memanipulasi elemen-elemen HTML.
2. Memanipulasi CSS.
3. Penanganan *event* HTML.
4. Efek-efek JavaScript dan animasi.
5. Memodifikasi elemen HTML DOM.

Sintak dasar jQuery $(selector).action(), tanda $ untuk mendefinisikan jQuery, jQuery *selector* digunakan untuk mendapatkan elemen HTML, action adalah tindakan yang dilakukan jQuery pada elemen ()

Contoh penggunaan jQuery untuk menyembunyikan elemen dengan id "test" sebagai berikut.

```
$("#test").hide()
```

Semua metode jQuery berada di dalam fungsi document.ready() yaitu perintah inisialisasi yang menunjukan dokumen telah siap ditampilkan dan sekaligus menjalankan perintah yang terdapat didalam fungsi.[29]

2.15 *Database* / Basis Data

Database/basis data terdiri dari 2 suku kata, yaitu data dan *base*/basis. Data dapat diartikan sebagai representasi fakta dunia nyata yang mewakili suatu objek, misalnya manusia, hewan, barang, peristiwa, konsep, dan lain sebagainya yang direkam dalam bentuk huruf, teks, simbol, angka, suara, gambar, dan lainnya.

Sedangkan basis/*base* dapat diartikan sebagai tempat berkumpul, sarang atau gudang untuk menyimpan sesuatu. Dengan demikian, basis data/*database* dapat diartikan sebagai tempat berkumpul/menyimpan data – data suatu benda atau kejadian yang saling berhubungan.

Database juga dapat diartikan sebagai kumpulan *file*, arsip yang saling berhubungan yang disimpan di dalam media elektronis. Data yang disimpan tersebut tidak dibiarkan begitu saja, namun dikelola dan diorganisasi yang dikenal dengan *Database Management System* (DBMS). Dengan demikian, data yang tersimpan di dalam database dapat disusun dengan rapi dan terstruktur sehingga memudahkan dalam mendapatkan informasi yang bersangkutan dengan *database*.[30]

2.15.1 Manfaat *Database*

Database sangat bermanfaat untuk mengatasi berbagai masalah yang sering terjadi dalam penyusunan data. Masalah dalam penyusunan data dapat berupa redudansi dan inkonsistensi data atau kesulitan pengaksesan data. Lebih jelas akan dipaparkan manfaat *database* sebagai berikut :

1. Redudansi dan Inkonsistensi Data.

   Redudansi adalah penyimpanan data yang sama dalam dua tempat (tabel) yang berbeda sehingga dapat menyebabkan inkonsistensi data. Inkonsistensi data, yaitu data yang sama dan disimpan pada dua tabel dapat memiliki nilai yang berbeda. Sebagai contoh, dalam database perpustakaan, di mana data nama anggota disimpan di tabel anggota dan tabel transaksi peminjaman (redudansi). Apabila nama anggota pada tabel anggota diubah, maka nama pada tabel transaksi tidak berubah (inkonsistensi data) padahal data tersebut menjelaskan objek yang sama.
2. Kesulitan Pengaksesan data.

   Kesulitan pengaksesan data pada *database* dengan program yang dapat muncul dapat diatasi dengan DBMS. DBMS mampu mengambil data secara langsung dengan bahasa yang sudah *familiar* dan mudah digunakan, misalnya SQL. Sebagai contoh, apabila pada suatu waktu akan dibutuhkan laporan tentang adventori buku yang terbit tahun 2009, sedangkan program untuk itu belum dibuat, maka dengan mudah DBMS menyelesaikannya.
3. Isolasi Data untuk Standarisasi.

Isolasi data untuk standarisasi berkaitan dengan format penyimpanan data tersebut. Sebagai contoh, data disimpan pada file dengan bahasa tertentu, misalnya Java, C/C++,Visual Basic, Pascal, dan lainnya, tentunya hal ini menyulitkan *programmer* untuk mengakses data tersebut. Untuk itu, data dalam suatu *database* harus disimpan dalam format yang sama. Sebagai contoh, dalam format JavaDB/Derby, MySQL, MsQL.

4. *Multiple User* atau Banyak Pemakai.

   Database dirancang agar data diakses oleh banyak pemakai(*user*) dan banyak program yang sama dalam waktu bersamaan maupun berbeda. Hal ini mungkin terjadi dengan *database*, dikarenakan dalam rangka mempercepat daya guna sistem dan mendapatkan responsi waktu yang cepat, beberapa sistem mengizinkan banyak pemakai untuk meng-*update* data secara simultan. Salah satu alasan mengapa basis data dibangun adalah karena nantinya data tersebut akan digunakan oleh banyak orang dalam waktu yang berbeda, diakses oleh program yang sama, tetapi berbeda orang dan waktu. Semua ini mungkin akan terjadi karena data yang diolah dan tidak tergantung dan menyatu dalam program, tetapi terlepas dalam satu kelompok data.

5. Masalah Keamanan.

   Walaupun *database* dibuat untuk diakses oleh banyak pemakai, *database* tetap menjaga keamanan data. Keamanan data dilakukan dengan memberikan hak akses tertentu bagi para *user* tersebut dan tidak semua user diizinkan untuk mengakses *database*. Sebagai misal data transaksi peminjaman buku tidak boleh diakses oleh anggota. Keamanan dapat diatur melalui program yang dirancang oleh *programmer* atau melalui fasilitas keamanan sistem operasi.

6. Masalah Kesatuan Data.

   Database memiliki tabel – tabel yang saling berkaitan/berhubungan. Hubungan antartabel secara teknis dikaitkan/dihubungkan dengan *field* kunci, sehingga dikenal dengan kunci primer (*primary key*) dan kunci tamu (*foreign key*) dalam suatu tabel. Sebagai contoh tabel buku dan tabel transaksi peminjaman buku, secara teknis buku mempunyai *field* kunci, sebagai misal *field* id dan tabel peminjaman juga mempuyai *field* id di mana mempunyai struktur yang sama

dengan *field* id pada tabel buku. Maka *field* id pada tabel buku disebut kunci primer dan *field* id pada tabel peminjaman disebut kunci tamu. Kedua *field* yang disebut kunci inilah yang menghubungkan antar tabel, sehingga dalam database mempunyai kesatuan data yang saling berkaitan.

7. Masalah Kebebasan Data.

   Kebebasan data yang dimaksud adalah kemudahan pembacaan data dalam *database* dapat diakses dengan mudah dalam menampilkan informasi yang dibutuhkan. Sebagai contoh, ketidakbebasan data adalah aplikasi program *database* yang telah dibuat secara khusus, di mana apabila suatu struktur *database* diubah strukturnya, maka *progammer* harus mengubah program aplikasi tersebut untuk mengakses data pada struktur *database* yang baru. Hal ini berbeda dengan database DBMS yang bersifat bebas, di mana data dengan mudah ditampilkan, dibaca, diubah, dan sebagainya dengan perintah –perintah tertentu.

8. Kemandirian Program Data.

   Program aplikasi berbasis database terpisah dengna data yang digunakannya, di mana data disimpan dalam DBMS dengan *format* tertentu. Dengan demikian, program aplikasi dan *database* berdiri sendiri – sendiri, namun antara program aplikasi dan *database* mempunyai hubungan yang erat. Hubungan tersebut adalah program aplikasi dapat mengakses data yang disimpan dalam DBMS. Hal ini dapat meningkatkan pengembangan aplikasi dan produktivitas perawatan.

9. Meningkatkan Kualitas Data.

   Untuk meningkatkan kualitas data dapat dilakukan dengan aturan validasi data melalui program aplikasi. Validasi data adalah penyaringan data – data yang hanya boleh disimpan dalam *database,* misal *field* id hanya boleh menyimpan data dengan *format* A11 (satu karakter diikuti 2 angka), atau hanya menerima input data berupa angka.

10. Akses Data Lebih Baik atau Responsif.

    Dengan menggunakan bahasa *Structured Query Language* (SQL), pengaksesan data jauh lebih baik dan cepat. Hal ini dikarenakan DBMS menggunakan bahasa

SQL yang mempunyai bahasa yang simpel dan mudah digunakan serta telah menjadi standar bahasa pembuatan *database*.

2.15.2 *Database Management System* (DBMS)

Database Management System atau disingkat DBMS adalah perangkat lunak (*software*) yang berfungsi untuk mengelola *database*. Mulai dari membuat *database* itu sendiri, sampai dengan proses yang berlaku dalam *database* tersebut, baik berupa *entry*, edit, hapus, *query* terhadap data, membuat laporan dan lain sebagainya secara efektif dan efisien. Salah satu jenis DBMS yang sangat terkenal saat ini adalah *Relational* DBMS (RDBMS). RDBMS merepresentasikan data dalam bentuk tabel – tabel yang saling berhubungan. Sebuah tabel disusun dalam bentuk baris (*record*) dan kolom (*field*). [29]

Ada tiga kelompok perintah yang digunakan dalam mengelola dan mengorganisasikan data dalam RDBMS, yaitu :

1. *Data Definition Language*.

   Merupakan perintah yang digunakan oleh seorang *Database Administrator* untuk mendefinisikan struktur *database*, baik membuat tabel baru, menentukan struktur penyimpanan tabel, *model* relasi antar tabel dan validasi data.

2. *Data Manipulation Language* (DML).

   Perintah – perintah yang digunakan untuk memanipulasi dan mengambil data pada suatu *database*. Manipulasi yang dapat dilakukan terhadap data adalah :

   a. Penambahan data
   b. Penyisipan data
   c. Penghapusan data
   d. Pengubahan data

   DML merupakan bahasa yang memudahkan pengguna dalam mengakses database. Ada dua jenis DML, yaitu :

   a. Prosedural, mengharuskan pengguna menentukan spesifikasi data apa yang dibutuhkan dan bagaimana cara mendapatkannya. Contoh paket bahasanya adalah dBase III, FoxBase, FoxPro.

b. Non Prosedural, pengguna hanya menentukan data apa yang dibutuhkan tanpa harus tahu cara mendapatkannya. Contoh paket bahasanya diberi nama *Structural Query Language* (SQL).
c. *Data Control Language*. Bagian ini berkenaan dengan cara mengendalikan data, seperti siapa saja yang bisa melihat isi data, bagaimana data bisa digunakan oleh banyak *user*, lebih mengarah ke segi sekuritas data.[31]

2.16 MySQL

MySQL pertama kali dirintis oleh seorang *programmer database* bernama Michael Widenius. MySQL *database server* adalah RDBMS (*Relational Database Management System*) yang dapat menangani data yang ber*volume* besar. Meskipun begitu, tidak menuntut *resource* yang besar. MySQL adalah *database* yang paling populer diantara *database – database* yang lain.

MySQL adalah program *database* yang mampu mengirim dan menerima data dengan sangat cepat dan *multi user*. MySQL memiliki dua bentuk lisensi yaitu *free software* dan *shareware*. Untuk penelitian ini akan menggunakan MySQL yang *free software* karena bebas menggunakan *database* ini untuk keperluan pribadi atau usaha tanpa harus membeli atau membayar lisensi, yang berada di bawah lisensi GNU/GPL (*general public license*), yang dapat didapatkan pada *website* resminya di http://www.mysql.com. MySQL sudah cukup lama dikembangkan beberapa fase penting dalam pengembangan MySQL adalah sebagai berikut :

1. MySQL dirilis pertama kali secara internal pada 23 Mei 1995
2. Versi Windows dirilis pada 8 Januari 1998 untuk Windows 95 dan Windows NT.
3. Versi 3.23 : beta dari Juni 2000, dan dirilis pada Januari 2001.
4. Versi 4.0 : beta dari Agustus 2002, dan dirilis pada Maret 2003 (unions).
5. Versi 4.1 : beta dari bulan Juni 2004, dirilis pada bulan Oktober 2004 (Rtrees dan B-trees, subqueries, prepared statements).
6. Versi 5.0 : beta dari bulan Maret 2005, dirilis pada Oktober 2005.
7. Sun Microsystems Membeli MySQL AB pada tanggal 26 Februari 2008.

8. Versi 5.1 : dirilis 27 November 2008 (*event scheduler*, *partitioning*, *plug – in* API, *row – based replication*, *server log table*).

2.16.1 Kelebihan dan Keuntungan Menggunakan MySQL

Ketika menggunakan sebuah *software* DBMS perlu diperhitungkan apa kebutuhan yang paling penting. Apakah performa, support, fitur – fitur SQL, kondisi keamanan dalam *license* atau harga. Dengan pertimbangan – pertimbangan tersebut, database MySQL memiliki beberapa kelebihan dan keuntungan dibandingkan dengan *database* lain. Beberapa diantaranya adalah :

1. MySQL merupakan *Software* DBMS yang bersifat *Open Source* sehingga bebas digunakan oleh perseorangan atau instansi tanpa harus membeli atau membayar pada pembuatnya.
2. MySQL dapat diakses melalui protokol ODBS (*Open Database Connectivity*) buatan Microsoft. Ini menjadikan MySQL dapat diakses oleh banyak *software*.
3. Semua klien dapat mengakses *server* dalam satu waktu, tanpa harus menunggu yang lain untuk mengakses *database*.
4. *Database* MySQL mengerti bahasa SQL (*Structured Query Language*).
5. *Database* MySQL dapat diakses darimana saja melalui internet dengan hak akses tertentu.
6. MySQL dapat berjalan di berbagai Sistem Operasi seperti Linux, Windows dan Solaris.

2.16.2 SQL

SQL adalah singkatan dari *Structured Query Language*. SQL sering dibaca dengan “Sequel” adalah bahasa terstruktur yang digunakan untuk *query*, mengupdate dan memanipulasi database. Bahasa ini digunakan untuk memuat, mengurutkan dan menyaring suatu data, sehingga dihasilkan data yang spesifik dari sebuah *database*.

2.16.3 SQL dalam MySQL

SQL dalam MySQL dapat dikelompokkan menjadi beberapa kelompok perintah, meskipun sebetulnya pengelompokan ini hanyalah untuk membedakan fungsi – fungsi utamanya saja. Kelompok SQL tersebut adalah seperti berikut ini : *Data Definition Statements* Adalah kelompok SQL untuk melakukan perintah dalam tingkat definisi *database*, tabel dan index. Terdiri dari perintah – perintah *Alter database*, *Alter Table*, *Create Database*, *Create Index, Create Table*, *Drop Database*, *Drop Index*, *Drop Table*, dan *Rename Table*.

1. *Data Manipulation Statements*. Adalah kelompok SQL untuk melakukan perintah manipulasi data. Terdiri dari perintah – perintah *Delete*, *Do*, *Handler, Insert, Load Data InFile, Select, Sub Query, Truncate,* dan *Update*.

   Contoh sintaks:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
 [create_specification] ...
create_specification:
 [DEFAULT] CHARACTER SET [=] charset_name
 | [DEFAULT] COLLATE [=] collation_name
```

2. *Data Manipulation Statements* Adalah kelompok SQL untuk melakukan perintah manipulasi data. Terdiri dari perintah – perintah *Delete*, *Do, Handler, Insert, Load Data InFile, Select, Sub Query, Truncate,* dan *Update*.

   Contoh sintaks:

```
INSERT INTO tbl_name () VALUES();
```

3. MySQL *Utility Statements*. Adalah kelompok SQL bagian *utility*. Terdiri dari perintah – perintah *Describe, Use*, *Explain* dan *Help*.

   Contoh sintaks:

```
HELP 'search_string'
```

4. MySQL *Transactional and Locking Statements*. Adalah kelompok SQL untuk melakukan perintah transaksi dan *locking* (penguncian). Terdiri dari perintah-perintah *Start Transaction, Commit, Rollback., SavePoint, Rollback to SavePoint, Lock Tables, Unlock Tables,* dan *Set Transaction*.[32]

   Contoh sintaks:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE
type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

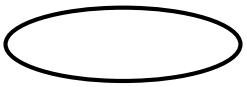
2.17 UML

Menurut Windu Gata, Grace (2013:4), *Unified Modeling Language* (UML) adalah bahasa spesifikasi standar yang dipergunakan untuk mendokumentasikan, menspesifikasikan dan membanngun perangkat lunak. UML merupakan metodologi dalam mengembangkan sistem berorientasi objek dan juga merupakan alat untuk mendukung pengembangan sistem.[33]

Alat bantu yang digunakan dalam perancangan berorientasi objek berbasiskan UML adalah sebagai berikut:

a. *Use Case Diagram*. *Use* case *diagram* merupakan pemodelan untuk kelakuakn (*behavior*) sistem informasi yang akan dibuat. *Use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Simbol-simbol yang digunakan dalam *Use Case Diagram* dapat dilihat pada Tabel 2.2 berikut ini.

Tabel 2.2 Simbol-simbol *Use Case Diagram*

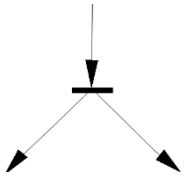
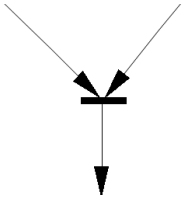
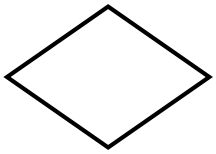
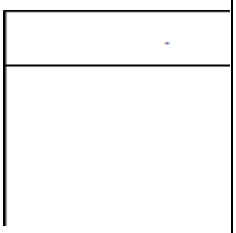
Simbol	Keterangan
	*Use Case* menggambarkan fungsionalitas yang disediakan sistem sebagai unit-unit yang bertukar pesan antar unit dengan aktor, yang dinyatakan dengan menggunakan kata kerja
	*Actor* atau Aktor adalah *Abstraction* dari orang atau sistem yang lain yang mengaktifkan fungsi dari target sistem. Untuk mengidentifikasikan aktor, harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks target sistem.

Simbol	Keterangan
	Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa aktor berinteraksi dengan *Use Case*, tetapi tidak memiliki kontrol terhadap *use case*
	Asosiasi antara aktor dan *use case*, digambarkan dengan garis tanpa panah yang mengindikasikan siapa atau apa yang meminta interaksi secara langsung dan bukannya mengindikasikan data
	Asosiasi antara aktor dan *use case* yang menggunakan panah terbuka untuk mengindikasikan bila aktor berinteraksi secara pasif dengan sistem
<<include>>	*Include*, merupakan di dalam *use case* lain (*required*) atau pemanggilan *use case* oleh *use case* lain, contohnya adalah pemanggilan sebuah fungsi program
<<extends>>	*Extend*, merupakan perluasan dari *use case* lain jika kondisi atau syarat terpenuhi

b. Diagram Aktivitas (*Activity Diagram*) menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Simbol-simbol yang digunakan dalam *activity diagram* dapat dilihat pada Tabel 2.3 berikut ini.

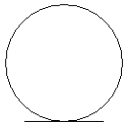
Tabel 2.3. Simbol-simbol Activity Diagram

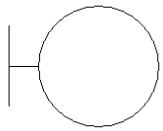
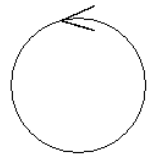
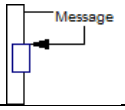
Simbol	Keterangan
	Start Point, diletakkan pada pojok kiri atas dan merupakan awal aktivitas
	End Point, akhir aktivitas
	Activities, menggambar kan suatu proses/kegiatan bisnis

Simbol	Keterangan
	*Fork*/percabangan, digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabung kan dua kegiatan paralel menjadi satu
	*Join* (penggabungan) atau *rake*, digunakan untuk menunjukkan adanya dekomposisi
	*Decision Points*, menggambar kan pilihan untuk pengambilan keputusan, *true* atau *false*
	*Swimlane*, pembagian *activity* diagram untuk menunjukkan siapa melakukan apa

c. Diagram Urutan (*Sequence Diagram*) menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Simbol-simbol yang digunakan dalam *Sequence* d*iagram* dapat dilihat pada Tabel 2.4 berikut ini.

Tabel 2.4 Simbol-simbol *Sequence Diagram*

Simbol	Keterangan
	*Entity Class*, merupakan bagian dari sistem yang berisi kumpulan kelas berupa entitas-entitas yang membentuk

Simbol	Keterangan
	gambaran awal sistem dan menjadi landasan untuk menyusun basis data
	*Boundary Class*, berisi kumpulan kelas yang menjadi *interfaces* atau interaksi antara satu atau lebih aktor dengan sistem, seperti tampilan *form entry* dan form cetak
	*Control class*, suatu objek yang berisi logika aplikasi yang tidak memiliki tanggung jawab kepada entitas, contohnya adalah kalkulasi dan aturan bisnis yang melibatkan berbagai objek
	*Message*, simbol mengirim pesan antar *class*
	*Recursive*, menggambarkan pengiriman pesan yang dikirim untuk dirinya sendiri
	*Activation*, mewakili sebuah eksekusi operasi dari objek, panjang kotak ini berbanding lurus dengan durasi aktivasi sebuah operasi
	Lifeline, garis titik-titik yang terhubung dengan objek, sepanjang *lifeline* terdapat *activation*

d. Diagram Kelas (*Class Diagram*) Merupakan hubungan antar *class* dan penjelasan detail tiap-tiap kelas di dalam model desain dari suatu sistem, juga memperlihatkan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem. *Class* Diagram juga menunjukkan atribut-atribut dan operasi-operasi dari sebuah kelas dan constraint yang berhubungan dengan objek yang dikoneksikan. *Class Diagram* secara khas meliputi : Kelas (*Class*), Relasi *Assosiations*, Generalitation dan *Aggregation*, attribut (*Attributes*), operasi (operation/method) dan *visibility*, tingkat akses objek eksternal kepada suatu

operasi atau attribut. Hubungan antar kelas mempunyai keterangan yang disebut dengan *Multiplicity* atau *Cardinality*. *Multiplicity* pada *class diagram* dapat dilihat pada Tabel 2.5 berikut ini.[34]

Tabel 2.5 *Multiplicity Class Diagram*

Multiplicity	**Penjelasan**
1	Satu dan hanya satu
0..*	Boleh tidak ada atau 1 atau lebih
1..*	1 atau lebih
0..1	Boleh tidak ada, maksimal 1
n..n	Batasan antara. Contoh 2..4 mempunyai arti minimal 2 maksimal 4