BAB II

LANDASAN TEORI

2.1. Pengertian dan Tujuan Belajar

Belajar merupakan perubahan tingkah laku atau penampilan, dengan serangkaian kegiatan misalnya dengan membaca, mengamati, mendengarkan, meniru dan lain sebagainya. Juga belajar akan lebih baik, jika subjek belajar mengalami atau melakukannya, jadi tidak hanya bersifat verbalistik. Disamping definisi tersebut, ada beberapa pengertian lain dan cukup banyak, baik dilihat secara mikro maupun secara makro, dilihat dalam arti luas ataupun terbatas atau khusus. Dalam pengertian luas, belajar dapat diartikan sebagai kegiatan psiko-fisik menuju ke perkembangan pribadi seutuhnya. Kemudian dalam arti sempit, belajar dimaksudkan sebagai usaha penguasaan materi ilmu pengetahuan yang merupakan sebagian kegiatan menuju terbentuknya kepribadian seutuhnya. Relevan dengan ini, ada pengertian bahwa belajar adalah "penambahan pengetahuan". (Sardiman, 2005)

Didalam belajar terdapat banyak faktor yang mempengaruhi, salah satunya adalah faktor psikologis. Ada beberapa faktor psikologis dalam belajar diantaranya: faktor motivasi, konsentrasi, reaksi pemahaman, orginasasi, dan ulangan. Selain itu masih ada faktor-faktor lain diantaranya: perhatian, minat, fantasi, faktor ingin tahu, dan sifat kreatif. (Sardiman, 2005)

Sehingga dapat diambil kesimpulan bahwa suatu pembelajaran harus dibuat menarik agar minat, motivasi dan sifat kreatif dari pembelajar dapat muncul yang pada akhirnya materi-materi ajar yang disampaikan dapat diterima dengan baik.

2.1.1. Tes Hasil Belajar

Menurut Purwanto (2008) tes hasil belajar atau *achievement test* adalah tes yang dipergunakan untuk menilai hasil-hasil pelajaran yang telah diberikan oleh guru kepada murid-muridnya, atau oleh dosen kepada mahasiswa, dalam jangka waktu tertentu. Dengan kata lain pemberian nilai oleh pengajar kepada pembelajar.

2.1.2. Penskoran

Menurut Purwanto (2008) penskoran adalah suatu proses perubahan jawaban-jawaban tes menjadi angka-angka. Angka-angka tersebut kemudian diubah menjadi nilai-nilai melalui suatu proses pengolahan tertentu. Penggunaan simbol untuk menyatakan nilai-nilai tersebut dapat berupa angka, seperti misalnya dengan rentangan 0 sampai 10, 0 sampai 100, dan ada pula yang berupa huruf A, B, C, D, dan E.

Purwanto (2008) menjelaskan bahwa terdapat perbedaan antara penskoran (*scoring*) dan penilaian (*grading*). Penskoran berarti proses pengubahan prestasi menjadi angka-angka, sedangkan dalam penilaian angka-angka tersebut diproses dalam hubungannya dengan kedudukan personal subjek ajar yang memperoleh angka-angka tersebut dalam skala tertentu. Misalnya skala tentang baik-buruk, bisa diterima atau tidak diterima, dinyatakan lulus atau tidak lulus.

2.1.3. Pemberian Peringkat

Menurut Silverius (1991) skor yang dipakai untuk membuat peringkat itu diurutkan mulai dari yang tertinggi hingga yang menurun sampai kepada yang terendah. Apabila terdapat skor yang sama, maka peringkatnya ditetapkan dengan

menjumlahkan angka peringkat berturut-turut sebanyak yang memiliki skor yang sama tersebut, lalu hasilnya dibagi dibagi dengan banyaknya yang memiliki skor sama tersebut. Contoh kedudukan skor dapat dilihat pada tabel 2.1.

Tabel 2.1. Kedudukan skor dalam peringkat

No. Urut	Skor	Peringkat
1	93	1
2	89	3
3	89	3
4	89	3
5	71	5

Keterangan tabel diatas adalah skor 89 menempati urutan ke-2, ke-4 dan ke-3. Supaya adil, nomor urut mereka dijumlahkan yaitu 2 + 3 + 4 = 9, kemudian dibagi banyaknya yang memiliki skor tersebut dalam hal ini 3. Jadi diperoleh 9 : 3 = 3.

2.2. Problem Based Learning

Menurut Kelly (2007) *Problem-based Learning* (PBL) adalah sebuah metode yang pertama kali ditemukan pada bidang ilmu pengetahuan medis dan pertama kali diperkenalkan pada tahun 1969, kemudian menjadi populer di disiplin ilmu lain seperti pendidikan, psikologi, dan bisnis (Coombs and Eden, 2004) dan juga pada bidang ilmu sains (Bell et all., 2002).

Menurut Kelly (2007) *problem-based learning* menggeser fokus pendidikan dari yang berpusat pada guru ke pengajaran dan pembelajaran yang berpusat pada siswa, dimana siswa membangun sendiri suatu makna dengan menghubungkan konsep-konsep dan ide-ide dari pengetahuan yang didapat. Tan (2004) dalam Kelly (2007) mengungkapkan bahwa PBL merupakan pendekatan alternatif dalam pengajaran dan pembelajaran yang mengharuskan keterlibatan

pembelajar secara aktif. Lebih jauh lagi menurut Harlen (2006) dalam Kelly (2007) PBL dapat digambarkan dalam aspek lain yaitu konstruktivisme yaitu pembelajaran lewat interaksi sosial, yang memungkinkan siswa didik untuk mencerna ide orang lain dan dan menggunakannya untuk memahami suatu hal.

Menurut Tan (2004) dalam Kelly (2007) dengan menggunakan masalah pada dunia nyata daripada memfokuskan pada suatu konten, siswa diberikan kesempatan untuk benar-benar belajar bagaimana belajar. Secara prinsip menurut White (2002) dalam Kelly (2007) PBL membalik pendidikan tradisional dengan meletakkan masalah lebih dulu dan menggunakannya untuk memotivasi pembelajaran. Dengan menggunakan masalah pada dunia nyata, PBL dapat membuat siswa melihat relevansi yang biasa mereka lewatkan pada konteks lainnya. Tujuan utama dari PBL adalah siswa dapat belajar lebih baik, mengerti apa yang mereka pelajari, dan mengingat lebih lama dengan bekerja sama dalam sebuah kelompok.

Dalam PBL terdapat dua fokus utama untuk menciptakan permasalahan. Pertama, masalah harus membangkitkan konsep dan prinsip-prinsip yang relevan terhadap domain masalah. Karena itu, proses dimulai dengan mengidentifikasi konsep utama dan prinsip-prinsip yang harus siswa pelajari. Kedua, masalah tersebut harus “nyata”. Dua alasan kenapa harus nyata. Pertama, siswa diberikan kesempatan untuk menjelajah semua dimensi dari masalah yang mana pembuatan masalah yang kompleks yang terdiri dari berbagai informasi merupakan hal yang sangat sulit. Kedua, masalah yang nyata terindikasi lebih meningkatkan pembelajaran karena terdapat banyak terdapat kesamaan dari konteks masalah tersebut dengan dunia nyata. (Savery, 2001)

2.3. Simulation Based Learning

Menurut Kneebel & Nestel (2005) dalam Grant (2007) *simulation-based learning* didefinisikan sebagai reproduksi dari sebagian aspek realita untuk dapat dimengerti dengan lebih baik, dimanipulasi atau memprediksi tingkah laku sebenarnya. Terminologi lain yang dikemukaan Gaba (2007) dalam Grant (2007) tentang simulasi diantaranya: Pengganti atau pembuatan ulang dari sebuah realita, suatu aktifitas yang menirukan realita, sebuah lingkungan yang dapat dikontrol, suatu pengalaman yang diarahkan, atau tugas yang interaktif.

Secara singkat Hamburger dalam Kindley (2002) mendefinisikan *simulation-based learning* sebagai "*learning by doing*" atau "belajar dengan praktek langsung". Dalam simulasi, pembelajar memilih dan merakit berbagai pengalaman sebagai respon untuk pertanyataan atau rangsangan lain. Pada simulasi aturan secara umum bahwa subjek dapat juga dikatakan berhasil dengan mengkombinasikan berbagai aksi berbeda dalam urututan yang bervariasi selama hasil akhir yang diperoleh dapat diterima.

Menurut Kindley (2002) terdapat tiga komponen untuk proses simulasi yang baik yaitu:

1. Model yang dikembangkan atau *storyline* dari lingkungan yang baik, sehingga dapat memacu keluaran yang sukses.
2. Pengkondisian yang dapat menyebabkan pembelajar dapat melakukan kesalahan.
3. Mentor simulasi untuk pembelajar.

Salas & Burke (2002) dalam Grant (2007) menyarankan bahwa simulasi harus dirancang dengan sedemikian rupa sehingga meliputi fitur instruksional,

pemandu pengalaman, pengukuran performa, diagnosa *feedback*, dan kecocokan dengan lingkungan yang sedang disimulasi.

Menurut Kneenobe (2006) dalam Grant (2007) atribut dari simulasi yaitu: praktek yang berulang-ulang dalam lingkungan yang aman, panduan dari seoarang pakar jika diperlukan, relevansi dengan praktek klinis yang sebenarnya, belajar dengan yang lain (pembelajar) dalam sebuah konteks realita, dukungan lingkungan yang berorientasi pembelajar.

Kindley (2002) memaparkan tiga tipe tujuan yang ditawarkan oleh *simulation-based learning* yaitu:

1. Simulasi pembelajaran yang ditujukan untuk mengembangkan respon (*response behaviour*).
2. Simulasi pembelajaran yang ditujukan untuk membantu menentukan keputusan.
3. Simulasi pembelajaran yang ditujukan untuk membantu mengakses informasi dan fasilitas manajemen pengetahuan.

Davidovitch (2006) menilai bahwa pembelajaran menggunakan simulator memiliki beberapa keuntungan. Simulator dikarakterkan sebagai alat yang dapat menimbulkan pengalaman praktis dan penerimaan respon secara langsung dari sistem yang sedang dipelajari untuk merangsang pengguna membuat aksi dan keputusan.

Menurut Kindley (2002) beberapa tipe dari *simulation-based learning* yaitu:

1. *Activity Simulations*: Contohnya “bagaimana menerbangkan pesawat” atau “bagaimana mengoperasikan kendaraan ini”. Kunci simulasi ini ada pada

aktifitas pekerjaan yang lebih membutuhkan penanganan informasi secara ahli, pembuatan alasan atau keputusan, dan kemampuan motorik untuk dapat berhasil.

2. *Soft Skill Simulations*: Pada simulasi tipe ini objek bukan sebuah alat melainkan individu atau grup. Sebagai contoh sales training, customer service, dan kepelatihan.
3. *Process Simulations*: Bagaimana pengeboran kilang minyak? bagaimana membuat *widgets*? Itu merupakan salah contoh tipe simulasi jenis ini. Tipe ini umumnya memerlukan model dari proses sebagai dasar dari pengalaman belajar. Pembelajaran dimulai dengan mengikuti materi dan aksi pada materi melalui sebuah proses.
4. *Business Simulations*: Simulasi ini menargetkan masalah seperti strategi kompetitif atau keputusan finansial. Simulasi ini membantu pengguna untuk belajar bagaimana membuat keputusan dan akan mengilustrasikan hasil akhir dari pendekatan yang berbeda-beda dalam bentuk "bagaimana jika".
5. *Software Simulations*: Ini merupakan tipe simulasi yang berhubungan dengan instruksional penggunaan teknologi khususnya software komputer. Contohnya adalah "Bagaimana menggunakan spreadsheet" atau "Bagaimana melakukan mail merge".
6. *Product Simulations*: Simulasi ini merupakan review dari produk tertentu dengan maksud untuk lebih mengenalkan pembelajar dengan komponen dan fungsi-fungsinya.
7. *Casual (Diagnostic) Simulations*: Simulasi yang ditujukan untuk pencarian masalah, troubleshooting, dan analisa akar masalah untuk membantu

pembelajar mengembangkan solusi. Sebagai contoh dari industri elektronik dan pesawat terbang. Peralatan yang tidak berfungsi memerlukan penanganan yang memiliki ciri-ciri sendiri untuk memperbaikinya.

2.4. World Wide Web (Web)

Menurut W3C (2004) *World Wide Web* (lebih dikenal dengan istilah *web* atau www) adalah ruang informasi di mana hal-hal menarik, disebut sebagai sumber (*resource*), diidentifikasi oleh pengenal global yang disebut *Uniform Resource Identifier* (URI).

World Wide Web diciptakan oleh Tim Berners-Lee pada tahun 1989 yang saat itu adalah ilmuan dari CERN (Badan Riset Nuklir Eropa). Tim Berners-Lee juga mencipatakan *HyperText Markup Language* (HTML) sebuah teknologi untuk bertukar informasi melalui dokumen yang memiliki *hypertext*. Bersamaan dengan HTML, dia juga menciptakan protokol komunikasi yang digunakan untuk mengirim informasi melalui *web*, protokol tersebut disebut *HyperText Transfer Protocol* (HTTP). (Deitel, 2008)

Ilustrasi singkat bagaimana seorang user mengakses konten pada *web* adalah sebagai berikut.

1. User membuka *web browser*
2. User memasukkan alamat URL dari *web* yang ingin dikunjungi pada *web browser*, misal *http://example.com/*
3. *Web Browser* kemudian menghubungi server yang bertanggung jawab terhadap alamat google.com dengan mengirimkan HTTP *request*.
4. Server yang bertanggung jawab terhadap alamat google.com kemudian menjawab dengan mengirimkan HTTP *response*

5. Browser menerima respon dari server kemudian mengintepretasikan HTML yang diterima untuk ditampilkan kepada user.

Aplikasi berbasis *web* disusun atas beberapa komponen teknologi yang saling mendukung. Komponen-komponen tersebut diantaranya: URL, protokol HTTP, HTML, *web browser* dan *web server*.

2.4.1. URI (Uniform Resource Identifier)

Menurut Berners-Lee (2005) *Uniform Resource Identifier* (URI) merupakan sebuah pengidentifikasi yang terdiri urutan karakter yang cocok dengan aturan-aturan tertentu yang digunakan untuk mengakses suatu *resource.*

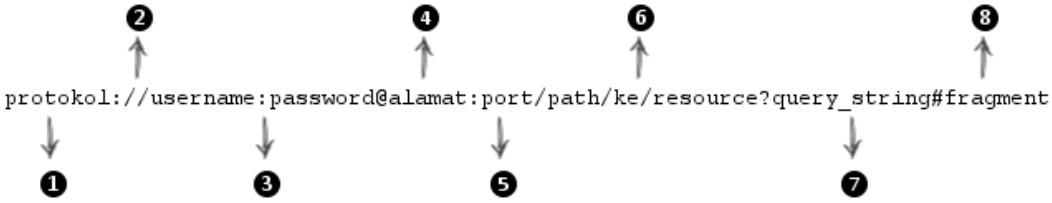
Selain URI terdapat juga istilah *Uniform Resource Locator* (URL) dan *Uniform Resource Name* (URN). Banyak terdapat kebingungan tentang hal ini, secara sederhana URI dapat berupa URL atau juga dapat berupa URN. Analogi sederhananya URI diibaratkan sebuah mobil, dimana pada mobil terdapat kategori seperti sedan (URL) atau truk (URN). Jadi jika suatu URL valid maka URL tersebut dapat disebut juga URI yang valid. Hal yang sama berlaku untuk URN. Berikut ini adalah contoh beberapa URL.

1. http://www.ietf.org/rfc/rfc2396.txt
2. ftp://ftp.is.co.za/rfc/rfc1808.txt
3. https://www.example.com/
4. mailto:johnsmith@example.com
5. http://en.wikipedia.org/wiki/Request_for_Comments#cite_note-6

Menurut Moats (1997) URN difungsikan sebagai media yang tetap, tidak terikat lokasi yang digunakan untuk pengidentifikasi *resource*. Skema URN dimaksudkan agar *resource* yang ditunjuk meskipun tidak ada atau sudah tidak

tersedia skema tersebut tetap unik secara global dan bersifat tetap. Berikut ini adalah beberapa contoh dari URN.

1. urn:isbn:0451450523
2. urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJUQCZO5C
3. urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66



Gambar 2.1. Bagian-bagian dari URL

Karena URN jarang digunakan pada aplikasi berbasis *web*, maka selanjutnya penulis menitikberatkan penjelasan hanya pada URL. Zalewski (2012) memaparkan 8 (delapan) komponen lengkap dalam sebuah URL seperti yang terlihat pada gambar 2.1.

Bagian-bagian URI antara nomor satu sampai dengan delapan memiliki dua sifat yaitu wajib dan pilihan. Tabel berikut menjelaskan bagian-bagian yang terdapat pada gambar 2.1.

Tabel 2.2. Bagian-bagian dan Keterangan dari URI

Nomor	Keterangan	Sifat
1	Nama protokol atau skema yang digunakan dan selalu diakhiri dengan titik dua, contoh *http*, *https*, *mailto* atau *ftp*. Nama protokol atau skema bersifat *case-insensitive*.	Wajib
2	// (*double slash*) indikator dari sebuah URL yang memiliki hirarki.	Pilihan
3	Data otentikasi yang berisi username dan password.	Pilihan
4	Alamat server dari *web* yang ingin dibuka. Dapat berupa *hostname* atau alamat IP.	Wajib
5	Nomor port yang digunakan oleh server. Untuk HTTP standar yang digunakan adalah 80, sedangkan untuk HTTP*S* adalah 443.	Pilihan

Tabel 2.2. Bagian-bagian dan Keterangan dari URI

Nomor	Keterangan	Sifat
6	Alamat hirarki path UNIX ke *resource* yang akan dibuka.	Wajib
7	Parameter *Query String*.	Pilihan
8	Pengidentifikasi *fragment*, biasa digunakan untuk menunjuk bagian tertentu pada *resource* (jika resource berupa dokumen *HTTP*).	Pilihan

Terdapat beberapa karakter yang sudah dipesan (*reserved*) sehingga jika ingin digunakan pada sebuah URL maka perlu dilakukan URL *encoding*. URL encoding merupakan sebuah proses untuk mengubah karakter-karakter tertentu dengan tanda persen (%) diikuti dengan dua digit heksadesimal yang merepresentasikan nilai ASCII dari karakter tersebut. Karakter-karakter yang telah *reserved* pada URL yaitu: **:**, /, **?**, #, **[**, **]**, dan **@** . Sebagai contoh "/" memiliki nomor ASCII 47 (2F dalam heksadesimal) jadi hasil *encoding* karakter tersebut adalah %2F.

2.4.2. Protokol HTTP

Seperti yang telah dijelaskan sebelumnya bahwa protokol HTTP merupakan protokol yang digunakan untuk mengirimkan informasi melalui *web*. Versi paling mutakhir protokol HTTP adalah HTTP versi 1.1. Protokol ini distandardisasi dan dikembangkan oleh Internet Engineering Task Force (IETF) dan World Wide Web Consortium (W3C).

HTTP merupakan protokol yang berada pada sisi *application layer*. Untuk detail komunikasi antar jaringan HTTP menyerahkannya pada protokol TCP/IP. Sehingga dalam terminologi jaringan protokol HTTP berjalan diatas TCP, HTTP menggunakan TCP (*Transmission Control Protocol*) untuk mengirimkan data dan TCP sendiri berjalan diatas *layer* IP (*Internet Protocol*). (Gourley, 2002)

Koneksi HTTP bersifat *connectionless* dan *stateless*. *Connectionless* berarti *server* tidak selalu menahan koneksi untuk setiap klien, dengan kata lain setelah selesai mengirimkan respon maka *server* akan langsung menutup koneksi. Kondisi ini menyebabkan HTTP menjadi *stateless* sehingga *server* selalu menganggap *request* yang datang dari *klien* sebagai *request* baru meskipun datang dari klien yang sama. Hal ini menyebabkan aplikasi berbasis *web* perlu mengimplementasikan *session* yang salah satunya memanfaatkan fitur HTTP *Cookie*.

Komunikasi antara klien dan *server* pada protokol HTTP dilakukan dengan menggunakan mode teks bukan dengan format biner. Sehingga hanya dengan utilitas seperti *telnet* sudah dapat dilakukan komunikasi dengan *server*. Berikut penulis contohkan bagaimana mengambil halaman utama dari website http://ianawww.vip.icann.org/ menggunakan *telnet*. Penulis menggunakan *telnet* pada sistem operasi Ubuntu Linux, namun perintah tersebut juga dapat digunakan pada hampir semua sistem operasi.

```
GET /domains/example HTTP/1.1
Host: ianawww.vip.icann.org

HTTP/1.1 200 OK
Date: Sat, 19 Jan 2013 02:29:01 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Fri, 04 Jan 2013 01:17:22 GMT
Vary: Accept-Encoding
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

[----- Penulis potong sampai disini -----]
```

Pada perintah diatas penulis menginstruksikan telnet untuk melakukan koneksi ke ianawww.vip.icann.org pada *port* 80 (*port* umum untuk *web server*). Setelah koneksi terjadi penulis mengirimkan berturut-turut kombinasi karakter berikut:

1. "GET / HTTP/1.1" diikuti karakter CRLF (*Carriage Return & Line Feed*, pada *keyboard* umumnya dinamakan tombol *Enter*).
2. "Host: ianawww.vip.icann.org" diikuti dua kali CRLF.
3. Karena *server* telah menerima dua kali karakter CRLF (baris kosong) maka *server* berasumsi *header* yang dikirimkan klien telah selesai dan waktunya untuk membalas *request* tersebut berupa HTTP *response*. Hal tersebut terlihat ketika aplikasi *telnet* menampilkan *string* HTTP/1.1 200 OK yang merupakan suatu format HTTP *response* jika suatu *resource* tersedia.

2.4.3. HTTP Method

Menurut Gourley (2002) *HTTP method* merupakan suatu aksi yang dilakukan oleh klien kepada server agar melakukan sesuatu pada *resource* tertentu. *Method* yang umum digunakan adalah GET, POST, dan HEAD. *Syntax* penulisan pada HTTP *method* (dapat disebut juga *request*) adalah sebagai berikut:

```
<method> <request-URL> <versi>
<header>

<request-body>
```

Sedangkan format untuk *response* (disebut juga HTTP *response* atau *status-code*) yang dikirimkan oleh *server* adalah sebagai berikut:

```
<versi> <status-code> <kalimat-status>
<header>

<response-body>
```

Tabel 2.3 menunjukkan detail penjelasan format untuk *syntax* HTTP *method* sedangakn tabel 2.4 menunjukkan detail penjalasan untuk format respon yang dikirim *server*. Setiap pesan HTTP baik itu berupa *request* atau *response* selalu diawali dengan baris pembuka. Setiap *request* (HTTP *Method*) dapat dianalogikan mengatakan *apa yang akan dilakukan* kepada *server* sedangkan

analogi untuk *response* adalah selalu mengatakan *apa yang telah terjadi* pada *server*. Analogi tersebut tercermin dalam *syntax* komunikasi seperti yang tertera pada tabel 2.3 dan 2.4.

Tabel 2.3. Format HTTP Request

Bagian	**Keterangan**
`<method>`	Aksi yang akan dikirimkan dapat berupa GET, HEAD, POST, PUT, DELETE, TRACE, atau OPTIONS.
`<request-URL>`	*Path* ke *resource* yang ingin diminta. Ditulis dengan format *path* absolut dengan diawali "/".
`<versi>`	Versi protokol *HTTP* yang digunakan. Saat ini versi yang paling banyak digunakan adalah *HTTP* 1.1. Sebelumnya juga terdapat *HTTP* versi 1.0 dan 0.9.

`<header>`	Kosong atau lebih header tambahan, setiap header diawali dengan nama, diikuti titik dua (:), diikuti *whitespace* (pilihan), diikuti nilai *header* tersebut, diikuti karakter CRLF dan harus diakhiri karakter CRLF. Contohnya adalah sebagai berikut (Karakter [CRLF] dalam kenyataan tentu tidak terlihat):
	
	`Accept: text/html, */*[CRLF]`
	`Accept-Encoding: gzip, deflate[CRLF]`
	`[CRLF]`

`<request-body>`	Data yang dikiriman ke *server*. Umumnya hanya digunakan untuk method POST.

Tabel 2.4. Format HTTP *Response*

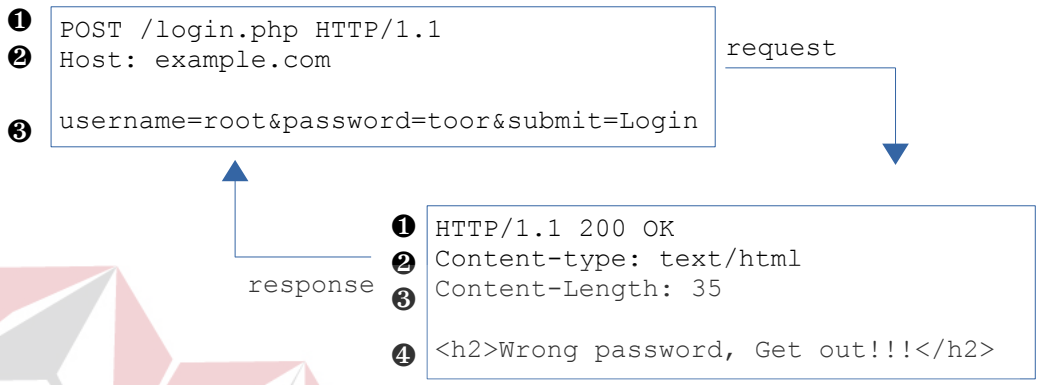
Bagian	**Keterangan**
`<versi>`	Versi protokol *HTTP* yang digunakan.
`<status-code>`	Kode status dari respon HTTP. Kode yang umum adalah 200, 301, 302, 401, 403 atau 404.

`<header>`	Format *header* sama dengan *request*. Contoh *header response* adalah sebagai berikut:
	
	`Content-Type: image/jpeg[CRLF]`
	`Content-Encoding: gzip[CRLF]`
	`Content-Length: 1234[CRLF]`
	`[CRLF]`

`<string-status>`	*String* status yang ditampilkan bervariasi dan berhubungan juga dengan *status-code*. Sebagai contoh jika *status-code*-nya adalah 200 kalimat status yang ditampilkan adalah

Tabel 2.4. Format HTTP *Response*

Bagian	Keterangan
	"OK", bila *status-code*-nya adalah 404 maka *string* status yang ditampilkan adalah "Not Found".
`<response-body>`	*Resource* yang diminta oleh *klien*. Untuk *method* tertentu seperti HEAD maka tidak ada *response-body* yang dikirimkan.



Gambar 2.2 Contoh proses HTTP POST

Pada gambar 2.2 penulis contohkan proses *HTTP* POST yang dikirimkan pada *server* dengan alamat URL /login.php pada sebuah domain dengan nama *example.com*. Pada gambar tersebut baik proses *request* ataupun *response* selalu diawali oleh baris pembuka seperti yang ditandai dengan nomor 1.

2.4.4. HTTP Status Code

Pada protokol HTTP setiap *response* yang dikirim oleh *server* pasti memiliki *status-code*. Fielding (1999) menjabarkan pembagian *status-code* untuk *response* seperti ditunjukkan pada tabel 2.5.

Tabel 2.5. Pembagian *Status-Code*

Kode	Keterangan
`1xx` *Informational*	Berisi status kode sementara untuk *response*. Diterapkan mulai HTTP versi 1.1.
`2xx` *Successful*	Berisi status yang menandakan bahwa *request* dari *klien* berhasil diterima dan dimengerti.

Tabel 2.5. Pembagian *Status-Code*

Kode	Keterangan
`3xx` *Redirection*	Berisi status yang menandakan bahwa *user agent* (*web browser*) perlu melakukan aksi lebih lanjut untuk menuntaskan *request*.
`4xx` *klien Error*	Berisi status yang menandakan bahwa *request* dari klien tidak dimengerti oleh server atau *klien* tidak berhak mengakses *resource* yang dituju.
`5xx` *Server Error*	Berisi status yang menandakan bahwa *server* mengerti *request* yang dikirim *klien* tetapi *server* tidak dapat memprosesnya karena suatu kesalahan tertentu pada sisi *server*.

Jika dipetakan secara menyeluruh maka jumlah *HTTP status-code* tersebut cukup banyak. Namun dalam kenyataannya hanya beberapa saja yang sering digunakan. Kode-kode status yang sering digunakan diataranya dapat dilihat pada tabel 2.6.

Tabel 2.6. Daftar HTTP *Status-Code* yang sering digunakan.

Kode	Keterangan
`200 OK`	Menandakan bahwa *request* berhasil.
`206 Partial Content`	Menandakan *server* mendukung pengambilan *resource* melalui GET secara tidak penuh (partial). *klien* harus menyertakan *header* bernama *Range* untuk menentukan berapa dan mulai dari mana *bytes* yang ingin diambil. Aplikasi-aplikasi *download-manager* umumnya bergantung pada fitur ini.
`301 Moved Permanently`	Menandakan bahwa *URL* yang diminta telah dipindah secara permanen *request* pada masa datang selayaknya menggunakan URL baru yang telah diberikan *server* pada *header Location*.
`302 Found`	Menandakan bahwa *URL* yang diminta dipindahkan secara sementara ke *URL* baru yang diberikan *server* pada *header Location*.
`304 Not Modified`	Menandakan bahwa *resource* yang diminta *klien* lewat proses GET belum mengalami perubahan sejak akses sebelumnya. Jadi *klien* (*web browser*) seharusnya cukup mengambil *resource* tersebut dari *cache* internal.

Tabel 2.6. Daftar HTTP *Status-Code* yang sering digunakan.

Kode	Keterangan
`400 Bad Request`	Menandakan bahwa *request* yang dikirim tidak dimengerti oleh *server*.
`401 Unauthorized`	Menandakan bahwa *resource* yang ingin diambil memerlukan proses otentikasi. Server harus memberikan tambahan *header* bernama *WWW-Authenticate* dalam responnya.
`403 Forbidden`	Menandakan bahwa *server* mengerti request yang diminta tapi tidak dapat memenuhinya. Hal ini dapat terjadi karena beberapa hal salah satunya yaitu masalah *file permission* disisi *server*.
`404 Not Found`	Menandakan bahwa *URL* dari *resource* yang diminta oleh *klien* tidak ditemukan oleh *server*.
`500 Internal Server Error`	Menandakan bahwa terjadi suatu kesalahan pada sisi *server* sehingga tidak dapat memenuhi *request* dari
`502 Bad Gateway`	Menandakan bahwa terjadi kesalahan saat *server* utama meneruskan *request* ke *upstream server*. Hal ini terjadi jika *server* utama berfungsi sebagai *proxy* atau *gateway*.
`503 Service Unavailable`	Menandakan bahwa *server* tidak dapat memenuhi *request* dari *klien* karena mengalami *overload* atau sedang *server* sedang dalam masa perawatan.

2.4.5. HTTP Authentication

Untuk melindungi *resource* tertentu pada *server* dari pihak yang tidak berkepentingan maka diperlukan mekanisme tertentu yang disebut HTTP Authentication. Menurut Franks (1999) HTTP *Authentication* merupakan mekanisme otentikasi sederhana yang digunakan *server* untuk menguji *klien* dengan cara meminta *klien* untuk memberikan informasi otentikasi. Terdapat dua mode otentikasi yaitu: *Basic Authentication* dan *Digest* Authentication. Alur kerja untuk keduanya sama yaitu sebagai berikut:

1. Klien *(Web Browser)* mengirimkan request ke URL yang diproteksi.
2. Server merespon dengan mengirimkan *status-code* “401 Unauthorized”

dengan tambahan header `WWW-Authenticate`.

3. *Web Browser* menampilkan window dialog untuk memasukkan username dan password.
4. *Web Browser* mengirimkan username dan password yang dienkripsi ke dalam header `Authorization`.
5. Server mengambil data otentikasi yang ada pada *header* `Authorization` dan kemudian mencocokkan dengan database internal server.
6. Jika otentikasi berhasil maka *server* akan merespon dengan *status-code* 200 OK, sedangkan jika salah maka *server* kembali menampilkan *401 Unauthorized*.

A. Basic Authentication

Basic Authentication merupakan model otentikasi yang mengharuskan *klien* untuk mengirimkan username dan password untuk setiap *realm*. *String* yang diberikan oleh *server* untuk mengidentifikasi *resource* yang diproteksi itulah yang disebut dengan *realm*. (Franks, 1999)

Username dan password yang dikirim oleh *klien* harus dipisahkan dengan karakter titik dua ":" dan dikirim dalam bentuk *string Base64*. Sebagai contoh jika username adalah “stikom” dan password adalah “surabaya” maka susunannya adalah “stikom:surabaya”. Bentuk *Base64* dari *string* tersebut adalah “c3Rpa29tOnN1cmFiYXlh”. Sehingga untuk setiap request yang memerlukan otorisasi *klien* wajib menyertakan *header* berikut:

```
Authorization: Basic  c3Rpa29tOnN1cmFiYXlh
```

Berikut penulis ilustrasikan proses bagaimana mengakses sebuah *resource* pada URL *http://example.com/top-secret/list.txt* yang diproteksi menggunakan

mekanisme HTTP *Basic Authentication*. Username dan Password yang digunakan adalah “mib:universe”.

1. Klien melakukan *request* tanpa otorisasi.

```
GET /top-secret/list.txt HTTP/1.1
Host: example.com
```

2. Server merespon dengan status-code 401 Unauthorized dan memberitahu klien bahwa mekanisme yang digunakan adalah *Basic Authentication*.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="MIB Top Secret List"
Content-Type: text/html
Content-Length: 1234

[--Pesan error dari server ditampilkan disini--]
```

3. klien melakukan request ulang dengan data otorisasi.

```
GET /top-secret/list.txt HTTP/1.1
Host: example.com
Authorization: Basic bWliOnVuaXZlcnNl
```

4. Server merespon dengan *status-code* 200 OK.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 1234

[--Isi dari file--]
```


B. Digest Authentication

Pada *basic authentication* informasi otorisasi dikirim menggunakan teks biasa yang di-*encode* menggunakan Base64, sehingga orang lain dalam jaringan yang dilalui paket tersebut dapat melihat isi dari username dan password hanya dengan melakukan proses *decode* isi dari header *Authorization*. Digest authentication mencoba untuk memecahkan masalah ini, pada model otentikasi digest kombinasi yang digunakan tidak hanya username dan password tetapi juga suatu nilai yang disebut *nonce*. Kombinasi string pada digest authentication

dienkripsi menggunakan algoritma enkripsi satu arah MD5. Sehingga nilai dari string enkripsi tersebut tidak dapat didekripsi. (Franks, 1999)

Selain ketiga kombinasi tersebut untuk menghasilkan string MD5 *checksum* yang benar maka diperlukan beberapa kombinasi lain untuk digabungkan dengan kombinasi sebelumnya. Franks (1999) memaparkan dalam RFC 2617 tentang cara bagaimana suatu MD5 checksum untuk digest authentication dihasilkan. Langkah-langkahnya adalah sebagai berikut.

1. Hasilkan MD5 hash dari kombinasi "`username:realm:password`". Masukkan hash tersebut kedalam A1.
2. Hasilkan MD5 hash dari kombinasi "*HTTP* `Method:`*URL*". Masukkan hash tersebut kedalam A2.
3. Hasilkan MD5 hash dari kombinasi "`A1:nonce:nc:cnonce:qop:A2`" Masukkan hash tersebut kedalam nilai *response*.

Keterangan dari berbagai kombinasi nilai tersebut diatas dapat dilihat pada tabel 2.7.

Tabel 2.7. Keterangan kombinasi pada digest authentication

Kode	**Keterangan**
username	Nama pengguna yang digunakan oleh user.
realm	*String* yang diberikan oleh *server* untuk mengidentifikasi *resource* yang diproteksi.
password	Kata kunci yang digunakan oleh user.
HTTP Method	Tipe request yang digunakan, contoh GET atau POST.
URL	Alamat *URL* dari *resource* yang diproteksi. Contoh /private/secret.txt.
nonce	String unik yang dihasilkan *server* setiap menampilkan response *401 Unauthorized*.
nc (nonce-count)	Nomor heksadesimal untuk menghitung jumlah *request* yang telah dilakukan, contoh nc=00000001.
cnonce (klien-nonce)	String nonce yang dihasilkan oleh *klien* yang digunakan

Tabel 2.7. Keterangan kombinasi pada digest authentication

Kode	Keterangan
	baik oleh *server* atau *klien* untuk menghindari *plaintext choosen attack*.

Berikut penulis ilustrasikan bagaimana proses digest authentication pada *resouce* dengan alamat URL http://example.com/area51/doc.txt. Username dan password yang digunakan masing-masing adalah “ufo” dan “moon”.

1. Klien melakukan request tanpa otorisasi.

```
GET /area51/doc.txt HTTP/1.1
Host: example.com
```

2. Server merespon dengan status-code 401 Unauthorized dan memberitahu *klien* bahwa mekanisme yang digunakan adalah *Digest Authentication*.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="Digest Auth Test",
                        qop="auth",
                        nonce="1f36b89738b2483cc107bc32b59bb626",
                        opaque="ea9420fab6ff939b0e0a65bb597737ff"
Content-Type: text/html
Content-Length: 1234

[--Pesan error dari server ditampilkan disini--]
```

3. Klien melakukan request ulang dengan data otorisasi.

```
GET /top-secret/list.txt HTTP/1.1
Host: example.com
Authorization: Digest username="ufo",
                      realm="Digest Auth Test",
                      nonce="1f36b89738b2483cc107bc32b59bb626",
                      uri="/private/",
                      response="a22b45564fdf003e687cc72adde50bcc",
                      opaque="ea9420fab6ff939b0e0a65bb597737ff",
                      qop=auth,
                      nc=00000001,
                      cnonce="32157b05c0d22c8b"
```

4. Jika hasil kalkulasi *checksum* yang dihasilkan *server* sama dengan nilai yang ada pada *response*, maka *server* merespon dengan *status-code* 200 OK.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 1234
```

```
[--Isi dari file--]
```

HTTP *authentication* bukanlah solusi untuk membuat sebuah aplikasi yang benar-benar aman. Karena pada kenyatannya semua *header* dan *body* dari paket HTTP tersebut dapat dilihat ketika melintasi suatu jaringan.

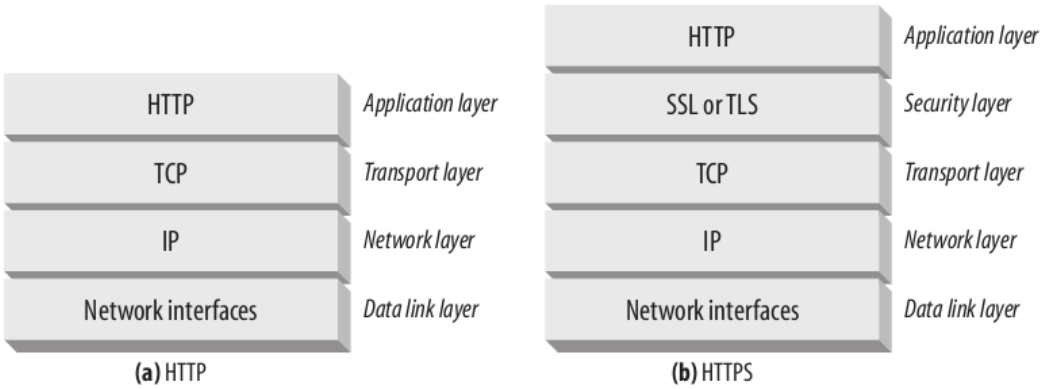
Secara umum metode ini sangat jarang digunakan pada aplikasi yang diperuntukkan kepada publik (internet). Model otentikasi ini banyak digunakan pada lingkungan organisasi yang hanya memanfaatkan jaringan intranet. (Stuttard, 2011).

2.3.6. HTTPS

Protokol HTTP menggunakan TCP biasa untuk melakukan mekanisme pengiriman, yang berarti tidak terenkripsi dan oleh karena itu rawan untuk disabotase oleh penyerang. HTTPS pada dasarnya adalah protokol yang berada pada *aplication layer* sama dengan HTTP namun ia dilewatkan melalui mekanisme pengiriman yang aman yaitu *Secure Socket Layer* (SSL). Mekanisme ini melindungi privasi dan integritas data yang dilewatkan melalui jaringan, sehingga mengurangi resiko kemungkian terjadinya *interception attack*. Sebagai catatan mekanisme SSL telah digantikan oleh yang lebih baru disebut *Transport Layer Security* (TLS). (Stuttard, 2011)

HTTPS merupakan mekanisme yang paling umum digunakan untuk mengamankan HTTP. HTTPS pertama kali diperkenalkan oleh perusahaan Netscape Corporation dan saat ini telah didukung oleh semua *web browser* dan *server*. Untuk mengakses halaman *web* yang menggunakan protokol HTTPS maka cara penulisan skemanya diawali dengan *https://* bukan *http://*. Sebagai contoh *https://www.example.com/.* Ketika menggunakan HTTPS semua data pada *request*

dan *response* dienkripsi sebelum data tersebut dikirimkan ke jaringan. HTTPS bekerja dengan menyediakan sebuah metode pengiriman kriptografi pada susunan *security layer*. (Gourley, 2002)



Gambar 2.3. Susunan l*ayer* HTTP dan HTTPS (Gourley, 2002)

2.3.7. HTTP Cookie

HTTP *Cookie* bukan merupakan bagian dari standar protokol HTTP yang distandardisasi pada dokumen RFC 2616, tapi *cookie* merupakan salah satu protokol tambahan yang sangat penting dalam eksistensi *web*. *Cookie* memperbolehkan server untuk mengirimkan informasi singkat dengan format "*name=value"* kepada *web browser* melalui *header* "*Set-Cookie"* dan untuk mendapatkan informasi itu kembali *server* mengharapkan *web browser* mengirimkan ulang dengan nama *header* "*Cookie"*. *Cookie* merupakan cara yang paling populer untuk melacak *session* dan otentikasi *user* pada aplikasi berbasis *web*. (Zalewski, 2012).

Berikut ini adalah contoh header yang dikirimkan oleh server.

```
Set-Cookie: session_id=b61ec238558b59c3eaf7bb870b850df5
```

Sedangkan contoh dibawah ini adalah header *cookie* yang dikirimkan klien kepada *server*.

```
Cookie: session_id=b61ec238558b59c3eaf7bb870b850df5
```

Stuttard (2011) memaparkan selain nilai aktual, sebuah *cookie* juga dapat membawa atribut tambahan yang sifatnya pilihan. Atribut tambahan (tabel 2.8) ini digunakan untuk memberitahu *web browser* bagaimana memperlakukan *cookie* yang dikirim.

Tabel 2.8. Atribut tambahan pada cookie

Atribut	**Keterangan**
`expires`	Tanggal dimana sebuah *cookie* masih dianggap valid. Jika nilai ini digunakan akan menyebabkan *web browser* menyimpan *cookie* ke sebuah penyimpanan permanen dan akan digunakan ketika setiap ada *request* sampai masa *expires* tiba. Contoh penggunaan *expires*: `expires=Tue, 15 Jan 2013 21:47:38 GMT`

`domain`	Nama *domain* (atau *sub domain*) yang valid untuk *cookie.* Ini harus sama atau masih satu sub dengan *domain parent* asal dari *cookie* yang diterima. Contoh penggunaan domain:
	
	`domain=example.com`
	`domain=.example.com`
	`domain=foo.example.com`

`path`	URL *path* yang valid untuk *cookie*. Contoh penggunaan *path*:
	
	`path=/`
	`path=/member`

`secure`	Jika atribut ini dipakai maka *cookie* hanya boleh dikirimkan lewat jalur HTTPS bukan HTTP.
`HttpOnly`	Jika atribut ini dipakai maka *cookie* tidak dapat diakses lewat *client-side script* seperti *javascript*.

Berikut contoh lengkap susunan dari sebuah cookie yang dikirm oleh server.

```
Set-Cookie: SID=123abc; domain=.foo.com; path=/; expires=Wed, 13-
Jan-2021 22:23:01 GMT; secure; HttpOnly
```

2.5. HTML

Menurut Raggett (1999) HTML (*HyperText Markup Language*) merupakan bahasa yang digunakan untuk melakukan publikasi halaman dalam *World Wide*

Web.

HTML versi paling terakhir adalah HTML 4.01. W3C sebagai organisasi yang melakukan standardisasi HTML menyertakan alternatif bahasa lain yaitu XHTML. XHTML merupakan reformulasi HTML yang disesuaikan dengan pola penulisan XML. Semua fitur yang ada pada HTML 4.01 juga didukung oleh XHTML. (Kennedy, 2006).

Untuk melakukan format tampilan pada halaman HTML harus dipergunakan aturan khusus disebut *tag*. *Tag* memberitahu *web browser* bagaimana menampilkan atau memperlakukan informasi yang ada dalam tag tersebut. (Kennedy, 2006).

Gambar 2.4. Contoh sederhana halaman HTML

Berikut contoh sebuah halaman HTML sederhana dengan penggunaan beberapa tag untuk melakukan format penulisan. Output dari contoh penggunaan tag-tag HTML dibawah ini ditunjukkan oleh gambar 2.4.

```
<html>
```

```
<head><title>Sejarah STIKOM</title></head>
<body>
<h1>Sejarah STIKOM</h1>
<img src="gambar/stikom.png" alt="logo stikom" align="left"
style="margin: 0 20px 10px 0;" /><p>Di tengah kesibukan derap
Pembangunan Nasional, kedudukan informasi semakin penting. Hasil
suatu pembangunan sangat ditentukan oleh materi informasi yang
dimiliki oleh suatu negara. Kemajuan yang dicitakan oleh suatu
pembangunan akan lebih mudah dicapai dengan <strong>kelengkapan
informasi</strong>.</p>
</body>
</html>
```

2.5.1. HTML Form

Tujuan HTML *form* adalah untuk untuk mengirimkan data yang diinputkan pengguna kepada *server* pemroses. Menurut Kennedy (2006), *form* terdiri dari satu atau lebih input teks, tombol, *multiple-choice checkbox*, *combobox*, dan *image map*, semua diletakkan didalam *tag* <form>. Komponen dari *form* ditunjukkan pada tabel 2.9.

Tabel 2.9. Komponen pada *tag* <form>

Komponen	**Keterangan**
Tag pembuka	<form>
Atribut	*accept, action, charset, class, dir, enctype, id, lang, method, name, onClick, onDblClick, onKeyDown, onKeyPress,*
Atribut	*onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReset, onSubmit, style, target, title*
Isi	*Tag-tag* yang digunakan untuk mendapatkan *input* dari pengguna.
Tag penutup	</form>
Digunakan pada	Blok <body></body>

Dua atribut yang paling sering digunakan pada tag pembuka form adalah atribut *action* dan *method*. Menurut Kennedy (2006), atribut *action* digunakan untuk menentukan URL dari aplikasi pada sisi server yang akan menerima dan memproses data yang ada pada *form*. Contoh sebuah atribut *action* adalah sebagai

berikut.

```
<form action="http://example.com/login.php">
...
</form>
```

Nilai dari atribut *action* dapat berupa URL lengkap dengan protokol dan domain (http://example.com) atau berupa URL relatif tanpa protokol dan domain.

Atribut berikutnya yang juga penting adalah atribut *method*. Atribut ini digunakan untuk menentukan metode yang digunakan *web browser* dalam mengirimkan data kepada *server* pemroses. Terdapat dua metode pada atribut *method* yaitu: GET dan POST. Jika atribut *method* tidak dituliskan maka *web browser* akan menggunakan metode GET. Berikut contoh penggunaan atribut *method*. (Kennedy, 2006)

```
<form action="http://example.com/login.php" method="POST">
...
</form>
```

Sebuah *form* tidak akan berguna tanpa ada inputan data didalamnya. Untuk menambahkan inputan data digunakan beberapa *tag* lain. *Tag* yang peruntukannya digunakan didalam *form* adalah *tag* <input>, <select>, <option>, <button> dan <textarea>. Dari beberapa *tag* tersebut tag <input> adalah yang paling sering digunakan. Tabel 2.10 menunjukkan komponen penyusun dari *tag* <input>.

Tabel 2.10. Komponen pada *tag* <input>

Komponen	**Keterangan**
Tag pembuka	<input>
Atribut	*accept, accesskey, align, alt, border , checked, class, dir, disabled, id, lang, maxlength, name, notab , onBlur, onChange, onClick, onDblClick, onFocus, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelect, size, src, tabindex, taborder , title, type, usemap, value*
Isi	Tidak ada
Tag penutup	</input> atau />

Tabel 2.10. Komponen pada *tag* <input>

Komponen	**Keterangan**
Digunakan pada	Blok <form></form>

Sebuah *tag input* dapat direpresentasikan secara berbeda-beda ketika ditampilkan pada *web browser*. Hal tersebut tergantung nilai dari atribut *type*, dimana atribut ini menentukan tipe atau fungsi dari suatu input. Macam-macam nilai dari *tag input* yaitu: *text*, *password*, *radio*, *checkbox*, *button*, *submit*, *image*, *reset* dan *hidden*. Berikut contoh penggunaan *tag input* untuk membuat sebuah *form login,* output dari contoh berikut dapat dilihat pada gambar 2.5.

```
<h2>Login</h2>
<form action="http://example.com/login.php" method="post">
<label for="uname">Username</label><br/>
<input type="text" name="uname" id="uname" /><br/>
<label for="upass">Username</label><br/>
<input type="password" name="upass" id="upass" /><br/>
<input type="submit" name="btnlogin" id="btnlogin"
value="LOGIN" />
</form>
```

Gambar 2.5. Contoh penggunaan tag *input* pada *form*

Dari contoh diatas ketika user menekan tombol *LOGIN* maka *web browser* akan mengirim HTTP *request* bertipe POST kepada server. Menekan tombol tersebut sama halnya dengan mengirim sebuah HTTP *request* seperti berikut (diasumsikan *password* yang diketik adalah “surabaya”).

```
POST /login.php HTTP/1.1<CRLF>
Host: example.com<CRLF>
Content-Type: application/x-www-form-urlencoded<CRLF>
Content-Length: 42<CRLF>
<CRLF>
uname=stikom&upass=surabaya&btnlogin=LOGIN<CRLF><CRLF>
```

Dapat dilihat bahwa data inputan yang dikirimkan oleh *web browser* dalam bentuk teks biasa sehingga orang lain yang memiliki akses terhadap jaringan yang dilalui dapat dengan mudah melihat isi dari data yang dikirimkan. Karena itulah diciptakan protokol HTTPS agar komunikasi antara pengguna dan *server* tidak bisa dilihat secara langsung karena terenkripsi.

Jika pada contoh sebelumnya *method* diubah dari POST menjadi GET maka inilah HTTP *request* yang dikirimkan oleh *web browser*.

```
GET /login.php?uname=stikom&upass=surabaya&btnlogin=LOGIN
HTTP/1.1<CRLF>
Host: example.com<CRLF><CRLF>
```

Pada metode GET data yang dikirimkan akan ditambahkan pada URL ditandai dengan adanya tanda tanya '?'. Menurut Robinson (2004) data setelah tanda '?' disebut variabel pencari dan dimasukkan kedalam *environment variable* bernama QUERY_STRING. Aplikasi pada sisi *server* dapat mengambil data yang dikirim lewat *environment variable* tersebut.

2.5.2. Hyperlink

Hyperlink atau sering disebut link, merupakan fitur yang digunakan untuk dapat berpindah dari satu bagian ke bagian lain dalam halaman itu sendiri, ke halaman lain dalam satu koleksi lokal, atau halaman lain yang ada di internet. Hyperlink atau disebut juga *anchor* ditandai dengan tag <a>. (Kennedy, 2006).

Tabel 2.11. Komponen pada tag anchor

Komponen	**Keterangan**
Tag pembuka	<a>

Tabel 2.11. Komponen pada tag anchor

Komponen	Keterangan
Atribut	*accesskey, charset, class, coords, dir, HRef, hreflang, id, lang, name, onBlur, onClick, onDblClick, onFocus, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, rel, rev, shape, style, tabindex, target, title, type*
Isi	Tag-tag yang dapat digunakan untuk presentasi ke user. Diantaranya: , <label>, <img>, dan .
Tag penutup	
Digunakan pada	Blok teks

Atribut yang utama pada tag link adalah *href*, dimana atribut ini berisi URL tujuan ketika link tersebut diklik. Berikut contoh potongan kode penggunaan tag *anchor*. Output dari potongan kode dibawah ini dapat dilihat pada gambar 2.6.

```
<a href="http://www.stikom.com">Masuk web STIKOM Surabaya</a><br/>
<p>Klik logo Google untuk ke google.com (New Window).</p>
<a href="http://www.google.com"><img src="gambar/google.png"
/></a>
<p>Klik logo Yahoo untuk ke yahoo.com.</p>
<a href="http://www.google.com"><img src="gambar/yahoo.png" /></a>
```

Gambar 2.6. Contoh penggunaan hyperlink

2.5.3. Frame

Menurut Zalewski (2012) *frame* merupakan bentuk markup yang

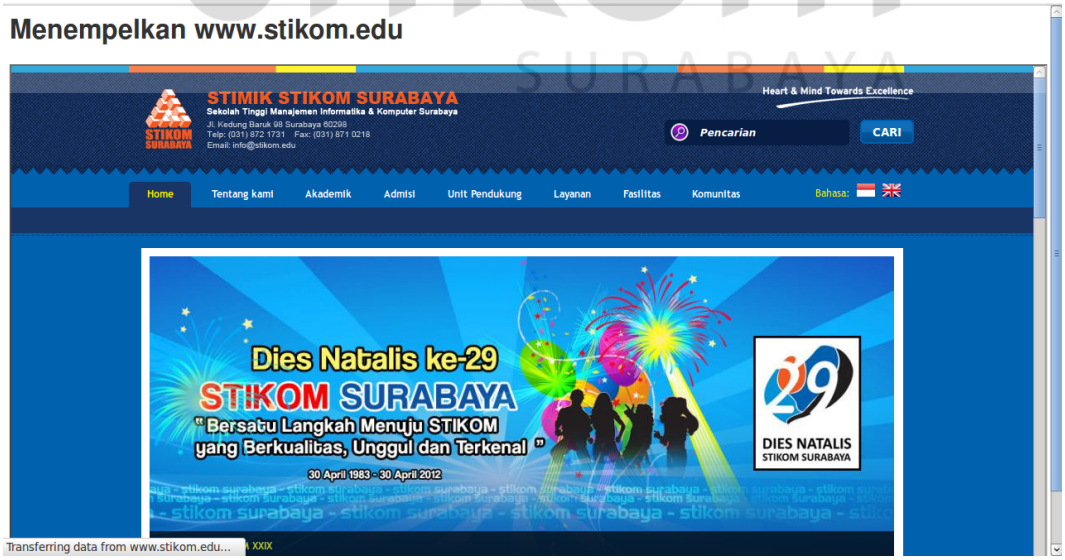
memperbolehkan isi dari suatu halaman HTML untuk ditampilkan dalam area kotak, secara singkat dapat disebut menempelkan halaman lain. *Web browser* modern mendukung beberapa penggunaan *tag frame* namun yang paling umum digunakan adalah *inline frame* atau *iframe*. Contoh penggunaan *tag iframe* adalah sebagai berikut.

```
<iframe src="http://example.com/"></iframe>
```

Kennedy (2006) menjelaskan beberapa komponen-komponen *iframe* seperti yang tertera pada tabel 2.12.

Tabel 2.12. Komponen pada *tag iframe*

Komponen	**Keterangan**
Tag pembuka	<iframe>
Atribut	*align, class, frameborder, height, id, longdesc, marginheight, marginwidth, name, scrolling, src, style, title, width*
Isi	Tidak ada.
Tag penutup	</iframe>
Digunakan pada	Blok *body*



Gambar 2.7. Penggunaan *iframe*

Berikut penulis contohkan penggunaan *iframe* dimana halaman yang ditempelkan dari *website* yang beralamat di http://www.stikom.edu/. Gambar 2.7 menunjukkan output dari contoh yang dimaksud.

```
<html>
<head><title>iframe stikom.edu</title></head>
<body>
<h1>Menempelkan www.stikom.edu</h1>
<iframe   src="http://www.stikom.edu"   border="0"   width="100%"
height="100%"></iframe>
</body>
</html>
```

2.5.4. Javascript

Javascript adalah sebuah bahasa *scripting* yang dapat menggunakan fitur-fitur yang ada pada *web browser*. Javascript dapat dipanggil pada sebuah halaman lewat suatu kode blok atau sebagai statemen tunggal yang dilampirkan pada tag-tag tertentu. *Web browser* yang mendukung penggunaan javascript akan melakukan interpretasi dan sebuah aksi atas statemen javascript yang diberikan untuk melakukan berbagai hal diantaranya: menambahkan suatu tampilan pada halaman, mengontrol tampilan, validasi dan manipulasi elemen *form*, dan melakukan hal-hal komputasi umum. (Kennedy, 2006).

Javascript dapat dijalankan dengan beberapa cara diantanya adalah melalui tag <script></script>. Daftar komponen-komponen pada *tag script* ditunjukkan pada tabel 2.13.

```
<script>
      // ini adalah code javascript
      alert('Hello World');
</script>
```

Cara yang lain adalah dengan menyertakan *event handler* langsung pada tag tertentu. Daftar *event handler* untuk javascript dan *tag* yang pendukungnya ditunjukkan pada Tabel 2.14.

```
<button onclick="alert('Hello World');">Click Me</button>
```

Tabel 2.13. Komponen-komponen pada *tag script*

Komponen	**Keterangan**
Tag pembuka	<script>
Atribut	*charset, defer, language, src, type*
Tag penutup	</script>
Isi	Kode yang dijalankan
Digunakan pada	head, body

Tabel 2.14. Event Handler

Event	**Tag**
onAbort	<img>
onBlur	<a>, <area>, <body>, <button>, <frameset>, <input>, <label>, <select>, <textarea>
onChange	<input>, <select>, <textarea>
onClick	Hampir semua *tag*
onDblClick	Hampir semua *tag*
onError	<img>
onFocus	<a>, <area>, <body>, <button>, <frameset>, <input>, <label>, <select>, <textarea>
onKeyDown	Hampir semua *tag*
onKeyPress	Hampir semua *tag*
onKeyUp	Hampir semua *tag*
onLoad	<body>, <frameset>, <img>
onMouseDown	Hampir semua *tag*
onMouseMove	Hampir semua *tag*
onMouseOut	Hampir semua *tag*
onMouseOver	Hampir semua *tag*
onMouseUp	Hampir semua *tag*
onReset	<form>
onSelect	<input>, <textarea>
onSubmit	<form>
onUnload	<body>, <frameset>

A. Variabel pada Javascript

Menurut McFarlan (2012) varibel merupakan suatu cara untuk menyimpan informasi untuk dapat dimanipulasi dan digunakan pada beberapa waktu kemudian. Untuk mendeklarasikan variabel digunakan kata kunci *var*. Berikut contoh deklarasi variabel pada *javascript*.

```
var kampus;
kampus = "STIKOM";

// atau langsung mengisi nilai
var kampus = "STIKOM";
```

McFarland (2012) juga menjelaskan bahwa terdapat tiga tipe data dasar pada *javascript* yaitu *number*, *string*, dan *boolean*. *Javascript* tidak memiliki cara khusus untuk mendeklarasikan masing-masing tipe data. Semua pendeklarasian menggunakan kata kunci *var*.

B. Fungsi pada Javascript

Fungsi pada *javascript* merupakan sekumpulan langkah-langkah pemrogaman yang disiapkan diawal *script*. Sekumpulan langkah-langkah pemrogaman tersebut tidak dijalankan ketika dibuat, tetapi disimpan pada memori *web browser* dan baru akan dijalankan ketika langkah-langkah tersebut dipanggil. (McFarland, 2012)

Javascript mengenal dua tipe fungsi yaitu fungsi *built-in* dan fungsi *custom*. Fungsi *built-in* merupakan fungsi-fungsi yang sudah disediakan oleh *web browser* dan untuk menggunakannya dapat langsung dipanggil. Beberapa fungsi *built-in* diantaranya: *alert, prompt* dan *isNaN*.

Fungsi *custom* merupakan suatu fungsi yang dibuat sendiri oleh *developer* dan tidak tersedia secara langsung pada *web browser*. Contoh fungsi yang dibuat

sendiri adalah sebagai berikut. Gambar 2.8 menunjukkan ketika fungsi *ucapkan_salam* dipanggil.

```
<script>
function ucapkan_salam(nama) {
      alert('Halo ' + nama + ', apa kabar?');
}
// panggil fungsi untuk menjalankan
ucapkan_salam('Mahasiswa');
</script>
```

Gambar 2.8. Contoh pemanggilan fungsi

C. Document Object Model (DOM)

Menurut Zalewski (2012) Document Object Model (DOM) merupakan sebuah susunan *tree* yang berisi semua elemen HTML, metode-metode spesifik dari tag tersebut dan propertinya (disebut juga atribut), dan data CSS. Perhatikan kode HTML berikut.

```
<html>
<head>Surabaya dan Yogyakarta</head>
<body>
<h2>Surabaya</h2>
<p id="desc_surabaya">Kota Surabaya adalah ibukota Provinsi Jawa
Timur, Indonesia.</p>

<h2>Yogyakara</h2>
<p id="desc_yogyakarta">Kota Yogyakarta adalah salah satu kota
besar di Pulau Jawa.</p>

<p>
  <a href="#" onclick="ganti_deskripsi('surabaya', 'Kota
Pahlawan');">Ganti Surabaya</a><br/>
  <a href="#" onclick="ganti_deskripsi('yogyakarta', 'Kota
Gudeg');">Ganti Yogyakarta</a><br/>
  <a href="#" onclick="kembali_awal();">Kembali Awal</a>
</p>
<script type="text/javascript">
```

```
  // global variabel yang digunakan pada fungsi kembali_awal
  var surabaya_awal =
document.getElementById('desc_surabaya').innerHTML;
  var yogyakarta_awal =
document.getElementById('desc_yogyakarta').innerHTML;

  // pencarian berdasarkan ID
  function ganti_deskripsi(kota, deskripsi_baru) {
    var keterangan_kota = document.getElementById('desc_' + kota);

    // ganti keterangan kota
    keterangan_kota.innerHTML = deskripsi_baru;

    // ganti warna dan tebal
    if (kota == 'surabaya') {
      keterangan_kota.style.color = 'red';
    } else {
      keterangan_kota.style.color = 'green';
    }
    keterangan_kota.style.fontWeight = 'bold';
  }

  // fungsi untuk mengembalikan deskripsi ke awal
  function kembali_awal() {
    var surabaya = document.getElementById('desc_surabaya');
    var yogyakarta = document.getElementById('desc_yogyakarta');

    // kembalikan deskripsi awal
    surabaya.innerHTML = surabaya_awal;
    yogyakarta.innerHTML = yogyakarta_awal;

    // kembalikan style awal
    surabaya.style.color = '';
    surabaya.style.fontWeight = '';
    yogyakarta.style.color = '';
    yogyakarta.style.fontWeight = '';
  }
</script>
</body>
</html>
```

Javascript menyediakan beberapa cara dalam mengakses elemen dalam DOM. Pada contoh potongan kode diatas penulis menggunakan sebuah *method* dengan nama *getElementById* untuk mengakses sebuah element dalam struktur DOM. Setelah referensi obyek dari elemen didapat penulis dengan bebas dapat mengubah isi dan properti dari elemen tersebut. Dalam contoh diperlihatkan ketika tombol link “Ganti Surabaya” atau “Ganti Yogyakarta” diklik maka isi dan warna dari tulisan berubah. Gambar 2.9 menunjukkan *output* yang dimaksud.

Gambar 2.9. Contoh manipulasi DOM dengan Javascript

Method getElementById bukan satu-satunya cara dalam mengakses DOM. Terdapat beberapa *method* untuk mengakses DOM diantaranya ditunjukkan pada tabel 15.

Tabel 15. Metode mengakses elemen pada DOM

Method	**Keterangan**
`document.getElementById`	Parameter: Atribut ID dari tag Output: *Single element*
`document.getElementsByTagName`	Parameter: Nama tag Output: *array of element*
`document.body.children`	Parameter: Index dari elemen Output: *single* atau *array of element*
`document.atribut_dari_name`	Parameter: Atribut *name* dari *tag*

Tabel 15. Metode mengakses elemen pada DOM

Method	Keterangan
	Output: *single* atau *array of element* Hanya berlaku pada tag <form>, <img>, <embed>, <object>, dan <applet>.

2.5.5. Entity Encoding

Menurut Zalewski (2012) e*ntity encoding* adalah suatu cara untuk memperbolehkan penggunaan karakter-karakter secara aman dengan menambahkan skema prefix *ampersand* (&) dan diakhiri dengan tanda titik koma. Terdapat beberapa nama entitas yang sudah disediakan oleh spesifikasi HTML untuk menggantikan karakter-karakter tertentu yang telah dipakai HTML sebagai struktur kode. Beberapa entitas tersebut diantaranya adalah:

1. *>* menggantikan tanda lebih besar (>)
2. *<* menggantikan tanda lebih kecil (<)
3. *"* menggantikan tanda petik dua (")

Entity encoding menjadi penting karena jika karakter-karakter terentu tidak diubah entitasnya maka dapat mengacaukan struktur halaman HTML yang dibangun. Perhatikan contoh berikut.

```
<img src="logo/einstein.png" title="Einsten Mengatakan: "Imagination is more important than knowledge."" alt="Einstein Quote" />
```

Potongan kode HTML diatas tidak valid karena, HTML *parser* dalam hal ini *web browser* menganggap setelah *title="Einsten Mengatakan: "* adalah properti yaitu *Imagination*, *is*, *important*, *than*, *knowledge*. Sedangkan properti-properti tersebut sebenarnya tidak ada dalam definisi HTML atau skema DTD

yang dibuat. *Entity encoding* mengatasi masalah tersebut dengan mengganti petik dua dengan nama entitas *"*

```
<img src="logo/einstein.png" title="Einsten Mengatakan: &quot;Imagination is more important than knowledge.&quot;" alt="Einstein Quote" />
```

Selain entitas nama juga dimungkinkan memasukkan entitas karakter ASCII atau karakter *Unicode* dengan notasi *&#nomor*. *Nomor* dapat berupa bentuk angka desimal atau heksadesimal jika ditambahkan dengan *prefix 'x'*. Berikut beberapa contoh penggunaan notasi tersebut.

1. : menggantikan tanda titik dua
2. (menggantikan tanda kurung buka
3. (menggantikan tanda kurung tutup

Bagi *attacker* karakter-karakter tersebut dapat digunakan untuk memasukkan suatu perintah yang tidak dikehendaki oleh developer. Perhatikan contoh berikut.

```
<a href="javascript&#58;alert&#40;1&#41;&#59;"></a>
```

Contoh potongan kode diatas ekuivalen sama dengan kode berikut.

```
<a href="javascript:alert(1)"></a>
```

Selain pada tag HTML, perlu diperhatikan juga masalah *encoding* pada *javascript*. Karena menurut Zalewski (2012) *javascript* mendukung karakter *encoding* berbasis *backslashes* yang dapat digunakan untuk melakukan *escaping* pada tanda petik, HTML *markup*, dan masalah lain yang berhubungan dengan penyisipan teks. Beberapa cara untuk memasukkan karakter *encoding* pada javascript adalah dengan notasi dalam petik *\nomor*. Dimana nomor dapat berupa bilangan oktal, heksadesimal, atau 16-bit heksadesimal. Berikut beberapa contoh *encoding javascript* menggunakan penomoran heksadesimal.

1. '\x41' menggantikan karakter 'A' (ASCII 65)
2. '\x22' menggantikan karakter petik dua (ASCII 34)
3. '\x3A' menggantikan karakter titik dua (ASCII 58)

2.6. Aplikasi pada Sisi Server

Pada aplikasi berbasis web sekarang ini masih terdapat banyak konten statik. Namun, sebagian besar isi yang disajikan kepada pengguna sebenarnya dihasilkan secara dinamis. Ketika pengguna mengirim *request* pada sebuah konten dinamis, *server* merespon dengan membuatnya saat itu juga (*on-the-fly*) dan setiap pengguna mungkin menerima respon yang berbeda-beda.

Ketika *web browser* dari pengguna melakukan *request* pada sebuah konten dinamis, normalnya *web browser* tidak hanya meminta salinan dari *resource* tersebut. Secara umum, *web browser* juga mengirim berbagai parameter bersama *request* tersebut. Parameter-parameter ini, memungkinkan aplikasi pada sisi server untuk menghasilkan konten yang sesuai dikhusukan untuk masing-masing pengguna secara unik.

Seperti perangkat lunak pada umumnya, aplikasi berbasis web juga menggunakan berbagai macam teknologi pada sisi server untuk mendukung fungsionalitasnya beberapa diantaranya:

1. Bahasa *Scritping* seperti PHP, VBScript (ASP) dan Perl.
2. Platform aplikasi *web* seperti ASP.NET dan Java.
3. *Web server* seperti Apache, IIS dan Netscape Enterprise.
4. *Database server* seperti MS-SQL, Oracle dan MySQL.
5. Dan komponen *backend* lain seperti *filesystem*, *web service* berbasis SOAP dan *directory service*.

(Stuttard, 2011)

2.7. Database

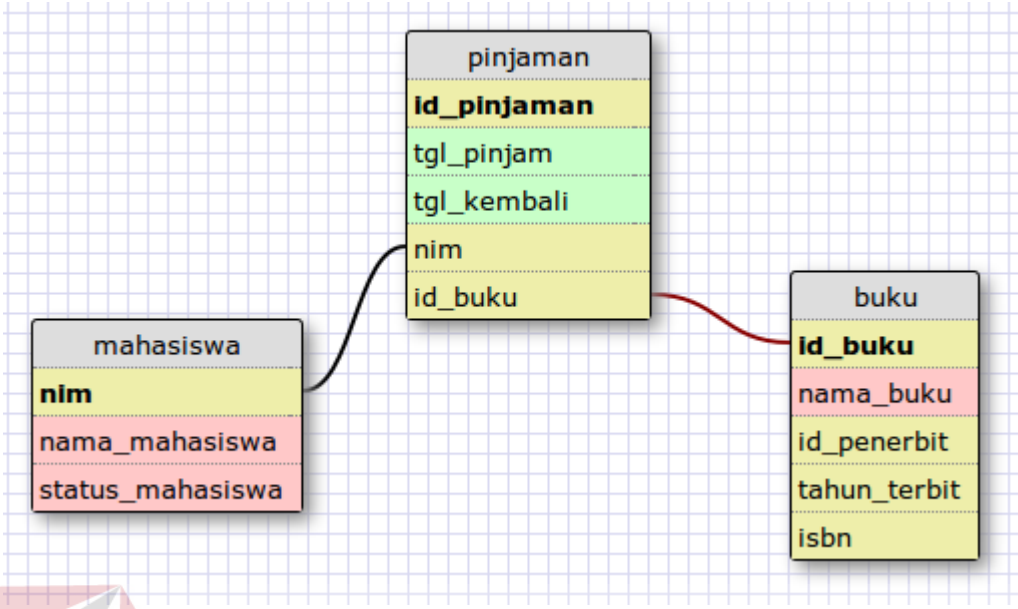
Database adalah suatu koleksi yang terdiri dari data tetap yang digunakan oleh sistem aplikasi dan diatur oleh sebuah *database management system* (DBMS). DBMS dapat juga disebut *database server* adalah sebuah kumpulan program yang memperbolehkan pengguna untuk membuat dan mengelola *database*. (Van der Lans, 2007)

2.7.1. Structured Query Language

Menurut van der Lans (2007) *Structured Query Language* (SQL) adalah perintah yang digunakan untuk memformulasikan statemen-statemen yang diproses oleh sebuah *database server*. Suatu statemen SQL terdiri dari kumpulan klausa-klausa SQL tertentu. Tabel 2.16 menunjukkan beberapa klausa SQL yang umum digunakan.

Tabel 2.16. Klausa-klausa umum SQL

Statemen	Keterangan
SELECT	Digunakan untuk mendapatkan data dari tabel.
UPDATE	Digunakan untuk mengubah nilai pada suatu baris pada tabel.
DELETE	Digunakan untuk menghapus suatu baris pada tabel.
FROM	Digunakan untuk memilih tabel yang menjadi target dari statemen (Umumnya SELECT dan DELETE).
WHERE	Digunakan untuk membatasi hasil dari suatu statemen.
ORDER BY	Digunakan untuk mengurutkan hasil dari statemen SELECT berdasarkan kolom tertentu.
GROUP BY	Digunakan untuk mengelompokkan hasil statemen SELECT berdasarkan kolom tertentu.
JOIN	Digunakan untuk memilih lebih dari satu tabel yang akan menjadi target dari statemen. Bentuk lain dari join adalah LEFT JOIN, INNER JOIN, dan RIGHT JOIN.
UNION	Digunakan untuk menggabungkan hasil dari dua atau lebih statemen SELECT.



Gambar 2.10. Contoh struktur tabel

Berikut contoh perintah SQL yang digunakan untuk mendapatkan *nomor pinjaman*, *nama peminjam*, *nama buku*, *tanggal pinjam*, dan *tanggal kembali* berdasarkan struktur tabel pada gambar 2.10.

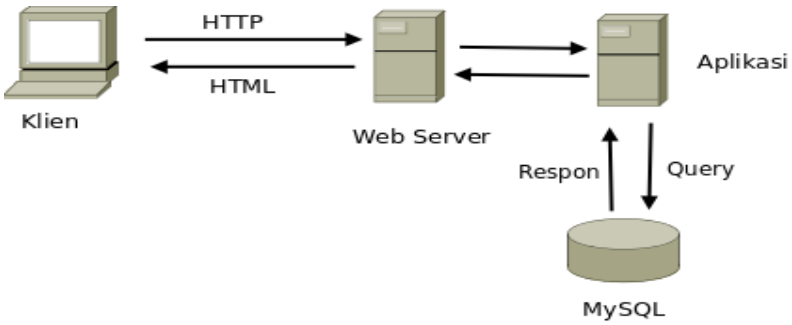
```
SELECT pjn.id_pinjaman, mhs.nama_mahasiswa, bk.nama_buku,
pjn.tgl_pinjam, pnj.tgl_kembali
FROM pinjaman pjn LEFT JOIN mahasiswa mhs ON mhs.nim=pjn.nim
LEFT JOIN buku bk ON bk.id_buku=pjn.id_buku
ORDER BY pjn.tgl_kembali
```

2.7.2. MySQL

MySQL merupakan *database server* yang bersifat *open source*. MySQL salah satu elemen utama dalam susunan perangkat lunak yang sering disebut dengan istilah LAMP yaitu Linux, Apache, MySQL, PHP/Python/Perl yang banyak digunakan untuk pengembangan aplikasi berbasis web. (Oracle, 2012)

Untuk melakukan proteksi data, MySQL menyediakan sistem hak akses yang dapat digunakan untuk membatasi akses seorang pengguna hanya terbatas pada semua *database* (*user privilege*), database tertentu (*database privilege*), tabel

tertentu (*table privilege*) atau kolom tertentu saja (*column privilege*). Hampir sama dengan *database server* lain pengaturan hak akses menggunakan perintah GRANT dan REVOKE. (Van der Lans, 2007)



Gambar 2.11. MySQL dalam arsitektur aplikasi web

Pada MySQL terdapat sebuah tabel istimewa bernama *INFORMATION_SCHEMA*. Tabel tersebut memberikan akses ke *database metadata*. *Metadata* merupakan "data tentang sebuah data", seperti nama *database* dan tabel, tipe data dari kolom, atau informasi hak akses. *INFORMATION_SCHEMA* adalah *metadata* dari database, tempat untuk menyimpan semua informasi tentang database lain yang dikelola oleh MySQL *server*. (Oracle, 2012)

2.8. Hacking

Hacker adalah suatu istilah untuk orang atau kelompok yang menulis kode program dan juga yang mengeksploitasi kelemahannya. Meskipun dua kelompok ini memiliki tujuan yang berbeda namun keduanya menggunakan cara yang mirip untuk memecahkan masalah. Suatu proses yang dilakukan *hacker* untuk menemukan sebuah solusi atas masalah dengan cara yang cerdas dan diluar intuisi inilah yang disebut dengan istilah *hacking*. (Erickson, 2008)

Sehingga dalam *web hacking* semua proses yang melibatkan bagaimana sebuah halaman *website* terbentuk dan bekerja dapat dimanfaatkan oleh *hacker* untuk dieksploitasi. Hal-hal yang dapat dieksploitasi oleh *hacker* pada *web* sangat banyak beberapa diantaranya meliputi mekanisme input data, lalu lintas *traffic* jaringan, *database server*, *web server*, protokol HTTP, kelemahan pada *web browser* dan tentu saja celah keamanan pada aplikasi *web* itu sendiri.

2.9. Ancaman Keamanan Web

Masalah utama yang mengancam aplikasi berbasis *web* adalah klien dapat mengirim berbagai macam input. Hal ini dikarenakan klien berada diluar kontrol dari aplikasi. Sebuah aplikasi harus menganggap semua input yang dikirimkan klien adalah berbahaya. Untuk itu perlu dilakukan berbagai cara agar *attacker* tidak dapat memasukkan input yang berbahaya yang dapat menyebabkan aplikasi dapat diambil alih. Masalah utama tentang input tersebut muncul karena beberapa hal diantaranya:

1. Pengguna dapat dengan mudah melakukan interfensi pada setiap data yang dikirimkan antara klien dan server, termasuk parameter *request* (*query string*), *cookie*, HTTP *header*. Setiap kontrol *security* yang diterapkan pada sisi klien (*web browser*) contohnya validasi input, dapat dengan mudah dilalui.
2. Pengguna dapat mengirimkan *request* dalam berbagai urutan dan dapat mengirimkan berbagai parameter dari yang aplikasi inginkan. Asumsi yang dibuat developer tentang bagaimana pengguna seharusnya mengirimkan data ke aplikasi dapat dijadikan sebuah celah.
3. Pengguna tidak hanya dibatasi menggunakan *web browser* ketika mengakses

aplikasi berbasis *web*. Banyak tersedia *tools* yang beroperasi didalam *web browser* (plugin) atau program independen yang dapat membantu melakukan penyerangan pada aplikasi berbasis web. *Tools-tools* dapat membuat dan melakukan *request* yang sangat besar untuk mempercepat menemukan celah keamanan.

Selain masalah utama tersebut faktor-faktor berikut dapat memperuncing masalah keamanan pada aplikasi berbasis *web*.

1. Lemahnya Kesadaran akan *Security* – Meskipun kesadaran akan *security* (keamanan) aplikasi *web* sudah meningkat beberapa tahun terakhir namun area tersebut belum matang seperti keamanan jaringan atau sistem operasi.
2. Pengembangan secara *Custom* – Kebanyakan aplikasi *web* dibuat oleh perusahaan itu sendiri atau melakukan *outsource* kepada pihak luar. Ketika berbagai bagian kode disatukan hal inilah yang dapat menyebabkan celah keamanan.
3. Kesederhanaan yang Menipu – *Platform* pengembangan *web* saat ini memungkinkan programmer pemula untuk membuat aplikasi *web* dalam waktu singkat. Tapi tentu ada perbedaan antara kode aplikasi yang berfungsi dengan kode aplikasi yang aman. Banyak aplikasi *web* yang dibuat oleh developer yang kurang paham dan minim pengalaman tentang keamanan, dimana mereka tidak tahu mana kode yang mungkin akan menimpulkan celah keamanan.
4. Permintaan Fungsionalitas yang Meningkat – Beberapa tahun lalu developer mengimplementasikan login hanya dengan penebakan username dan password. Website modern menambahkan fungsi-fungsi seperti *password*

recovery, kekuatan password, petunjuk password, dan opsi-opsi lain seperti *remember me* sehingga user langsung login ketika berkunjung lagi di waktu.

(Stuttard, 2011)

Masalah utama ditambah dengan faktor-faktor yang disebutkan sebelumnnya sangat berperangaruh terhadap keamanan aplikasi. Hal itu terbukti dari studi yang dilakukan Stuttard (2011) dimana dia melakukan uji coba terhadap lebih dari 100 aplikasi web dan menemukan celah keamanan sebagai berikut:

1. *Broken Authentication* (62%), Kategori ini mencakup kerentanan berbagai cacat dalam mekanisme login aplikasi, yang dapat memungkinkan penyerang (*attacker*) untuk menebak password yang lemah, meluncurkan serangan brute force, atau bypass login.
2. *Broken Access Control* (71%), Hal ini melibatkan kasus-kasus dimana aplikasi gagal untuk melindungi data dan fungsionalitasnya, berpotensi memungkinkan penyerang untuk melihat data sensitif pengguna lain yang ada pada server atau melakukan tindakan diluar hak aksesnya.
3. *SQL Injection* (32%), Kerentanan ini memungkinkan penyerang untuk mengirimkan input untuk mengganggu interaksi aplikasi dengan *back-end database*. Seorang penyerang mungkin dapat mengambil data penting dari aplikasi, mengganggu logikanya, atau mengeksekusi perintah pada server database itu sendiri.
4. *Cross-Site Scripting* (94%), Kerentanan ini memungkinkan seorang penyerang untuk menargetkan pengguna lain dari aplikasi, berpotensi mendapatkan akses ke data mereka, melakukan tindakan yang tidak sah atas nama mereka, atau melakukan serangan lain terhadap mereka.

5. Kebocoran Informasi (78%). Hal ini melibatkan kasus di mana sebuah aplikasi menggali informasi sensitif yang berguna bagi penyerang dalam mengembangkan serangan terhadap aplikasi, melalui cacat penanganan kesalahan atau perilaku lainnya.
6. *Cross-Site Request Forgery* (98%). Cacat ini berarti bahwa pengguna aplikasi dapat didorong untuk melakukan tindakan yang tidak diinginkan pada aplikasi dalam konteks pengguna. Kerentanan ini memungkinkan situs web berbahaya dikunjungi oleh pengguna (korban) untuk berinteraksi dengan aplikasi tersebut untuk melakukan tindakan yang pengguna tidak inginkan.

2.10. SQL Injection

Menurut OWAPS (2011) SQL *injection* adalah suatu teknik penyerangan yang memasukkan atau "menginjeksi" sebuah *query* SQL melalui input data dari klien ke aplikasi.

SQL adalah *interpreted language* dan aplikasi berbasis *web* umumnya menyusun statemen SQL tersebut digabungkan dengan input data yang diberikan oleh pengguna. Jika langkah tersebut tidak dilakukan dengan cara yang aman, kemungkinan besar aplikasi akan memiliki celah SQL *injection*. (Stuttard, 2011)

Perhatikan query berikut, query tersebut digunakan untuk proses pencarian sebuah *record* pada tabel user. Dimana jika *record* ditemukan maka asumsi dari aplikasi adalah *query* valid.

```
SELECT * FROM user WHERE username='admin' AND password='pas123'
```

Data *username* dan *password* diambil dari inputan yang diberikan oleh pengguna. Sekilas tidak ada yang salah dari struktur query diatas. Namun apabila

username yang dimasukkan adalah admin'-- maka query tersebut akan menjadi:

```
SELECT * FROM user WHERE username='admin'--' AND password='pas123'
```

Dalam SQL tanda "--" berarti komentar jadi karakter setelah tanda tersebut otomatis akan diabaikan. Pada akhirnya query yang dimasukkan username admin'-- sama saja dengan query berikut:

```
SELECT * FROM user WHERE username='admin'
```

Sehingga jika pada tabel user terdapat *username* dengan nama "admin" maka query tersebut valid dan pasti mengembalikan *record*.

Salah satu klausa SQL yang juga sangat sering digunakan untuk menggali data memanfaatkan celah SQL injection adalah UNION. Klausa UNION digunakan untuk menggabungkan dua atau lebih hasil dari statemen SELECT kedalam satu hasil. Untuk memperjelas penggunaan UNION dalam SQL *injection* penulis mengilustrasikan penggunaan sebuah tabel bernama produk dan beberapa perintah untuk mengambil informasi-informasi server MySQL.

Tabel 2.17. Contoh isi tabel produk

produk_id	produk_nama	produk_harga	kategori_id
1	Silver King	9000	2
2	Kat Kit	5000	2
3	Pepsodol	6000	1
4	Ciptadol	5000	1
5	Mak Ozone	7500	3

Jika pada aplikasi terdapat sebuah halaman yang menampilkan daftar produk dengan kategori id 2 dan urut berdasarkan harga terendah, maka *query* yang dijalankan adalah sebagai berikut:

```
SELECT * FROM produk WHERE kategori_id=2 ORDER BY produk_harga ASC
```

Keluaran dari *query* SQL diatas adalah dua buah *record* seperti yang

ditunjukkan pada tabel 2.18.

Tabel 2.18. Query untuk kategori dengan id 2 dan urut harga terendah

produk_id	produk_nama	produk_harga	kategori_id
2	Kat Kit	5000	2
1	Silver King	9000	2

Jika kategori pada aplikasi tersebut diambil dari suatu inputan pengguna dan developer tidak melakukan validasi input, *attacker* dapat menginjeksi *query* SQL dengan menambahkan perintah-perintah lain. Dalam contoh ini penulis menginjeksi *query* dengan menambahkan perintah-perintah untuk melihat versi, nama user, dan nama database yang sedang digunakan.

```
UNION SELECT NULL, CONCAT('VERSI: ', VERSION(), ' | USER: ',
USER(), ' | DATABASE: ', DATABASE()), NULL, NULL --
```

Jika digabungkan dengan *query* awal maka *server database* MySQL akan mengeksekusi *query* berikut.

```
SELECT * FROM produk WHERE kategori_id=2 UNION SELECT NULL,
CONCAT('VERSI: ', VERSION(), ' | USER: ', USER(), ' | DATABASE: ',
DATABASE()), NULL, NULL -- ORDER BY produk_harga ASC
```

Statemen “ORDER BY produk_harga ASC” tidak akan dieksekusi karena berada setelah tanda “--” yang berarti dianggap sebuah komentar oleh MySQL. Output dari *query* yang telah diinjeksi ditunjukkan pada tabel 2.19.

Tabel 2.19. Hasil *query* setelah injeksi

produk_id	produk_nama	produk_harga	kategori_id
2	Kat Kit	5000	2
1	Silver King	9000	2
NULL	VERSI: 5.5.8-log | USER: root@localhost | DATABASE: test	NULL	NULL

2.11. Platform Facebook

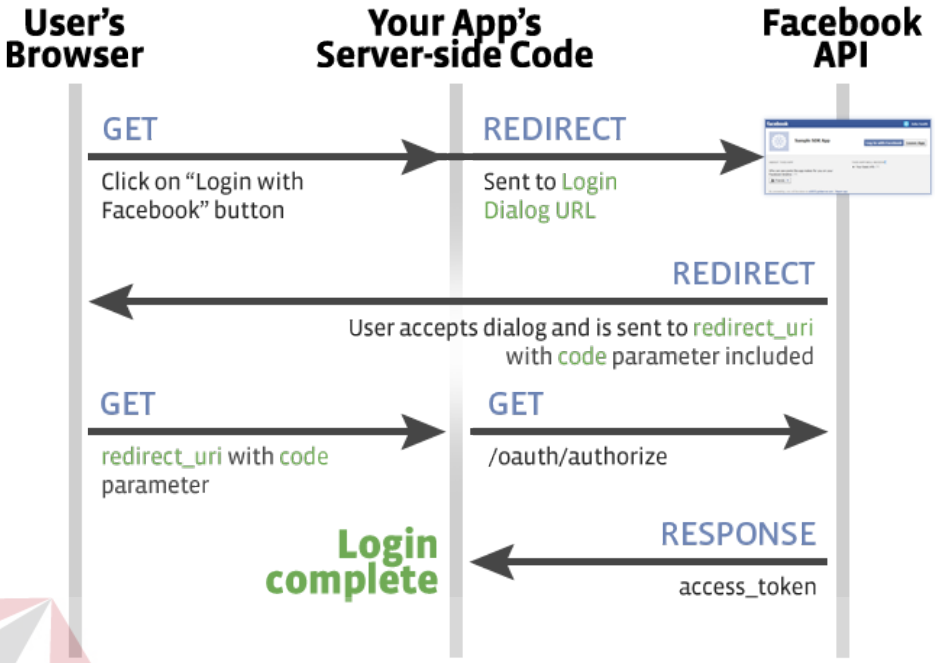
Facebook adalah sebuah situs jejaring sosial yang mulai beroperasi pada

tahun 2004 yang memiliki misi untuk membuat dunia lebih terbuka dan terkoneksi. Orang menggunakan Facebook untuk tetap terkoneksi dengan teman dan keluarga, untuk mengetahui apa yang sedang terjadi di dunia, dan untuk membagikan apa-apa yang bermanfaat bagi mereka. Per Maret 2012 jumlah pengguna Facebook mencapai 901 juta. (Facebook, 2012)

Platform Facebook memberikan kemudahan kepada developer untuk membuat aplikasi dan website yang bersifat sosial yang terintegrasi dengan Facebook dan menggapai jutaan pengguna. (Facebook, 2012)

Pada *platform* Facebook terdapat fitur yang bernama *Social Plugins*, dimana fungsi dari *social plugin* adalah untuk mempermudah developer atau pemilik *website* mengintegrasikan layanan dan juga membagikan konten-konten kepada teman melalui *news feed* atau *timeline*. *Social Plugin* terdiri dari beberapa komponen yaitu:

1. *Like Button*, pengguna dapat membagikan ke teman mereka sebuah halaman web atau konten tertentu dengan hanya sekali klik.
2. *Subscribe Button*, pengguna dapat mengikuti publik figur favorit mereka dalam sekali klik.
3. *Comments Plugin*, pengguna dapat dengan mudah memberikan komentar pada artikel, melihat komentar yang paling relevan, dan membagikan komentar tersebut kepada teman di Facebook.
4. *Single Sign On Registration Plugin*, memperbolehkan pengguna untuk login tanpa harus mengisi *form* registrasi atau mengingat username dan password lainnya.



Gambar 2.12. Alur server-side login pada Facebook (Facebook, 2012)

2.11.1. Facebook Login

Facebook menyediakan beberapa arsitektur untuk menjembatani autentikasi antara website pihak ketiga dengan server Facebook. Arsitektur yang disediakan yaitu: Client-side JavaScript SDK, Native Device Login dan Server-side Login. Dalam Tugas Akhir ini penulis menggunakan arsitektur Server-side Login. Alur server-side login diilustrasikan pada gambar 2.12.

2.11.2. Facebook Graph API

Graph API merupakan inti dari Facebook Platform dimana dengan menggunakan Graph API developer diperbolehkan melakukan *read* dan *write* data pada Facebook. Graph API merupakan *low-level* API berbasiskan protokol HTTP yang dapat digunakan untuk melakukan query data, *post* cerita baru, check-ins atau hal-hal lain yang mungkin dapat dilakukan oleh aplikasi. Berikut ini adalah

contoh Graph API *request* ke Server Facebook.

```
GET /774635482?fields=id%2Cname HTTP/1.1
Host: graph.facebook.com
Connection: close

HTTP/1.1 200 OK
Content-Type: text/javascript; charset=UTF-8
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Date: Thu, 11 Oct 2012 17:28:02 GMT
Connection: close
Content-Length: 48

{"id":"774635482","name":"Christopher Blizzard"}
```

Dalam contoh diatas request yang dilakukan beripe GET, dimana fungsinya adalah untuk membaca data. Path dari request tersebut adalah / 7743635482, yang merupakan lokasi dari sebuah resource pada graph. Data yang dikembalikan dari server Facebook berupa JSON.

2.11.3. Facebook Open Graph

Open Graph menyediakan sebuah cara untuk melakukan penamaan atau *mapping* pada konten aplikasi dan kegiatan apa saja yang dapat oleh pengguna pada konten tersebut – dengan mendefinisikan apa yang disebut *Actions* dan *Objects*.

Actions merupakan interaksi antara pengguna yang dilakukan pada aplikasi. Pengembang dapat mengimplementasikan salah satu *built-in actions* yang telah didefinisikan oleh Facebook atau dapat membuat sendiri *actions* yang merepresentasikan keunikan yang dilakukan pengguna pada aplikasi yang dibuat. Contoh beberapa *built-in action* diantaranya: *Read, Follow*, *Like*, *Watch*, dan *Listen*.

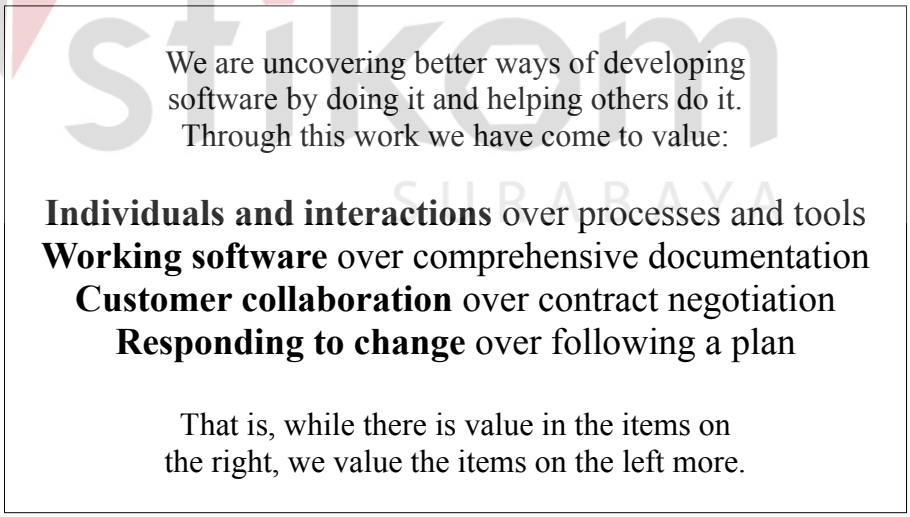
Objects merupakan target untuk aksi yang diambil pengguna dalam aplikasi yang dibuat. Pengembang dapat mengimplementasikan salah satu *built-in*

object atau membuat sendiri *object* yang merepresentasikan suatu hal pada aplikasi. Contoh dari *object built-in* yang telah disediakan Facebook diantaranya: *Article*, *Blog*, *Book*, *Video*, dan *Website*.

2.12. Metode Pengembangan

2.12.1. Pengembangan Perangkat Lunak Secara Agile

Secara umum pengembangan perangkat lunak menggunakan *agile* menawarkan pendekatan secara profesional yang meliputi aspek manusia, organisasi, dan teknologi dalam proses pengembangannya. Secara khusus, gambaran utama dari pengembangan secara *agile* yaitu: yang pertama adalah dengan mendeskripsikan *Agile Manifesto* dan implementasinya, dan yang kedua adalah mengimplementasikan salah satu metode *agile* yang dapat membuat tim menyelesaikan tugas pengembangan dengan kualitas tinggi. (Hazzan, 2008)



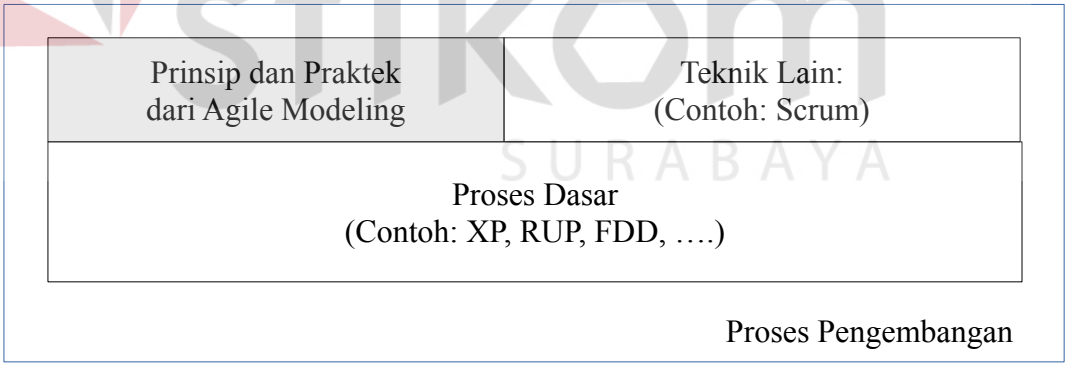
Gambar 2.13 Agile Manifesto

Agile Manifesto merupakan hasil formulasi dari tujuh belas pengembang perangkat lunak yang dilakukan pada bulan Februari tahun 2001 bertempat di Wasatch Mountain, Utah. Penerapan *Agile Manifesto* dapat dilakukan dengan

menggunakan salah satu metode *agile*. Beberapa metode *agile* diantaranya: *Extreme Programming*, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development dan lainnya. Isi dari *Agile Manifesto* dapat dilihat pada gambar 2.13.

2.12.2. Agile Modeling

Agile Modeling (AM) merupakan salah satu penerapan metode *agile*. Menurut Ambler (2012) *Agile Modeling* (AM) merupakan metodologi berbasis praktek untuk pemodelan dan dokumentasi yang efektif dari sistem perangkat lunak. Secara singkat AM adalah kumpulan dari nilai-nilai, prinsip-prinsip, dan praktek-praktek untuk pemodelan perangkat lunak yang dapat diaplikasikan pada proyek pengembangan perangkat lunak secara efektif dan dalam waktu singkat. Amber (2012) mengilustrasikan posisi dari *Agile Modeling* seperti ditunjukkan pada gambar 2.14.



Gambar 2.14 *Agile Modeling* membantu proses lain

Nilai-nilai yang ada pada *Agile Modeling* adalah sebagai berikut:

1. Komunikasi: Penting untuk memiliki komunikasi yang efektif antara tim pengembang dan *stackholder*.
2. Kesederhanaan: Berusaha untuk mengembangkan solusi sesederhana

mungkin.

3. Umpan balik: Dapatkan umpan balik lebih cepat dan lebih sering.
4. Keberanian: Keberanian untuk mencoba teknik baru dan dan tetap pada pendirian.
5. Kerendahan hati: Berusaha sadar bahwa anda tidak selalu mengetahui semuanya dengan demikian pihak lain dapat membantu menambahkan nilai pada pengembangan proyek.

2.12.3. Agile Model-Driven Development (AMDD)

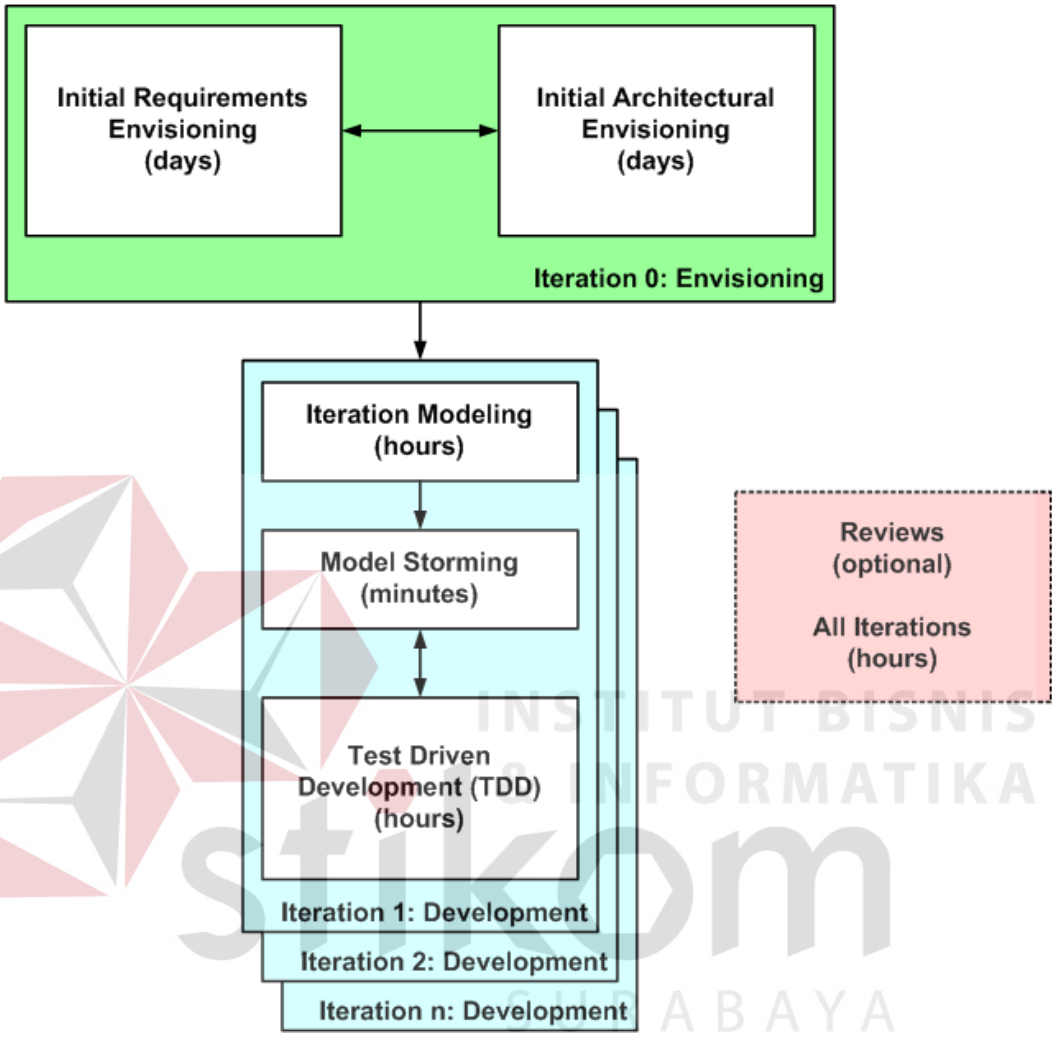
Menurut Ambler (2012) AMDD merupakan versi *agile* dari *Model Driven Development* (MDD). MDD adalah sebuah pendekatan dalam pengembangan perangkat lunak yang mana pembuatan model secara ekstensif (menyeluruh) dilakukan sebelum mulai menulis kode. Perbedaan dengan AMDD adalah pada AMDD pembuatan model tidak dilakukan secara ekstensif melainkan cukup membuat model yang dapat membuat pengembangan segera dijalankan. Ambler (2012) mengilustrasikan AMDD pada sebuah diagram yang ditunjukkan pada gambar 2.15.

Langkah-langkah yang dilakukan dalam pengembangan menggunakan *Agile Model Driven Development* yaitu: *Envisioning*, *Iteration Modeling*, *Model Storming* dan Implementasi via *Test Driven Development* (TDD). Lifecyle dalam pengembangan menggunakan AMDD ditunjukkan oleh gambar 2.16.

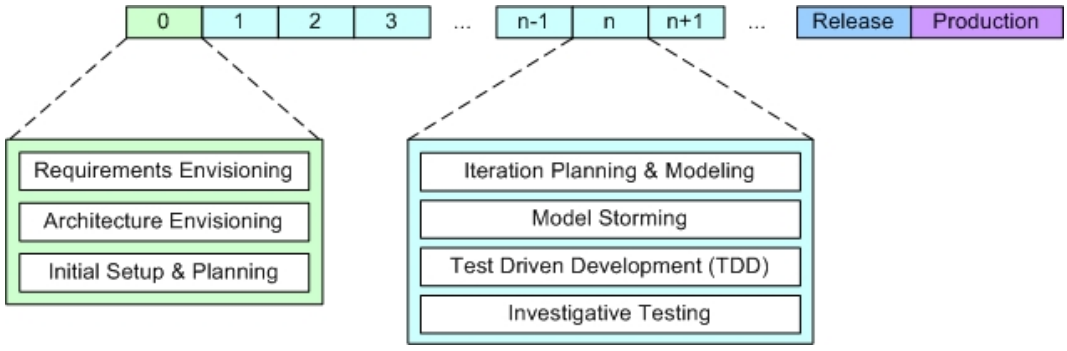
A. Envisoining

Proses *envisioning* dilakukan untuk mendapatkan gambaran umum dari sistem yang akan dibuat dan arsitektur seperti apa yang dapat digunakan.

E*nvisioning* dibagi dalam dua fase yaitu: Pemodelan Kebutuhan Awal dan Pemodelan Arsitektur Awal.



Gambar 2.15. Diagram AMDD (Ambler, 2012)



Gambar 2.16. AMDD Lifecycle (Ambler, 2012)

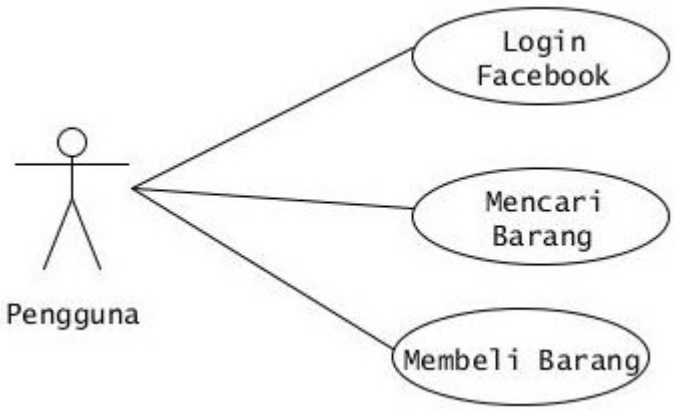
A.1. Pemodelan Kebutuhan Awal

Pemodelan kebutuhan awal mengidentifikasi *high-level requirements* dan juga cakupan dari rilis (apa yang sistem perlu lakukan). Tahap ini dilakukan pada iterasi ke-0. Secara singkat menurut Ambler (2012) yang dilakukan pada pemodelan kebutuhan awal adalah sebagai berikut:

1. Membuat *usage model*
2. Membuat *domain model*
3. Membuat *user interface model* (UI)

a. Usage Model

Usage Model digunakan untuk menggambarkan bagaimana pengguna berinteraksi dengan sistem. Hal ini penting untuk dapat memetakan kebutuhan pengguna. *Usage model* dapat berupa *user stories* atau *use case*. Contoh dari *user stories* dan sebuah *use case* ditunjukkan masing-masing oleh tabel 2.21 dan gambar 2.17.



Gambar 2.17. Contoh use case

Tabel 2.20 Contoh *user stories*

ID	User Stories
U01	Sebagai pengguna, Saya dapat login menggunakan akun Facebook.
U02	Sebagai pengguna, Saya dapat mencari barang berdasarkan nama, harga dan kategori.
U03	Sebagai pengguna, Saya dapat melakukan pembelian barang.

b. Domain Model

Domain model merupakan konseptual *model* yang menggambarkan entitas bisnis dan relasinya. Model ini sebaiknya dibuat dengan sangat sederhana asalkan dapat memuat informasi yang cukup untuk dapat menggambarkan proses yang ada. *Domain model* dapat digambarkan menggunakan notasi pemodelan apapun, sebagai contoh dapat digunakan UML *class diagram* sederahana seperti ditunjukkan pada gambar 2.18.

Gambar 2.18. Contoh *domain model*

c. User Interface Model

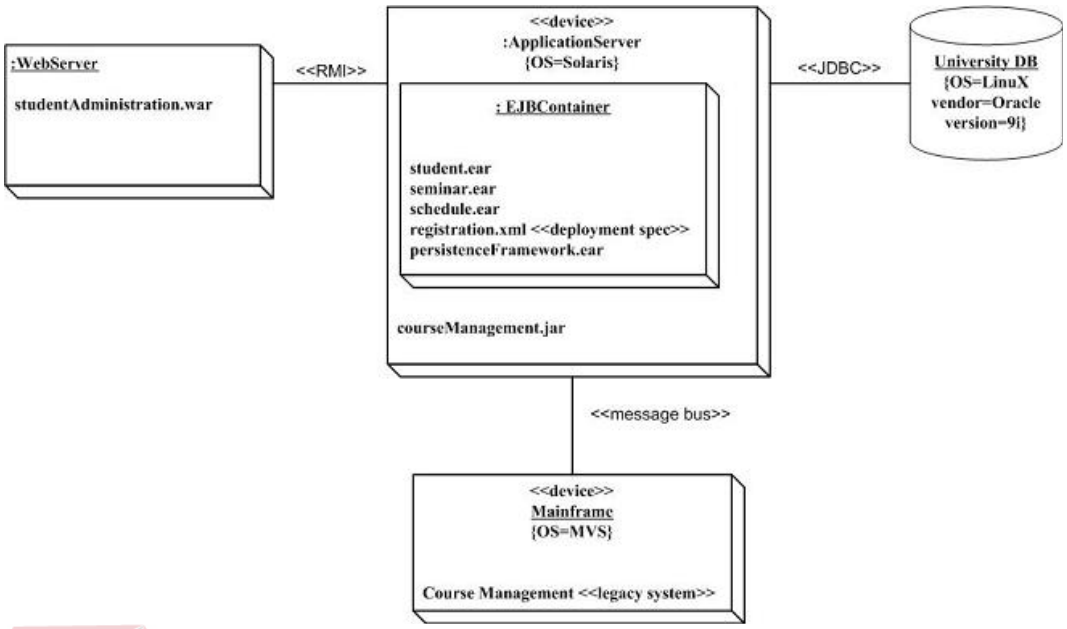
User interface model merupakan tahap pembuatan prototipe untuk desain antar muka dari aplikasi yang dibuat. Dapat digunakan sebuah sketsa untuk menggambarkan antar muka yang akan dibuat. Gambar 2.19 menunjukkan sketsa antar muka.

Gambar 2.19 Contoh sketsa antar muka

A.2. Pemodelan Arsitektur Awal

Menurut Ambler (2012) pada awal pengembangan, pengembang harus mempunyai gambaran seperti apa sistem yang akan dibangun. Apakah aplikasi yang dibangun merupakan aplikasi mainframe, aplikasi .NET, aplikasi J2EE atau yang lainnya. Untuk itu pengembang biasanya melakukan diskusi dan kemudian melakukan sketsa tentang arsitektur untuk sistem tersebut.

Pemodelan arsitektur berjalan paralel dengan tahap pemodelan kebutuhan awal. Sehingga pada tahap ini yang difokuskan adalah arsitektur perangkat keras dan perangkat lunak yang akan digunakan. Arsitektur yang dibuat kemungkinan dapat berubah seiring berjalannya waktu untuk itu pembuatan arsitektur pada tahap ini tidak perlu terlalu detail dan cukup sedikit dokumentasi jika diperlukan. Arsitektur dapat digambarkan dalam bentuk UML *deployment diagram* seperti yang ditunjukkan pada gambar 2.20.

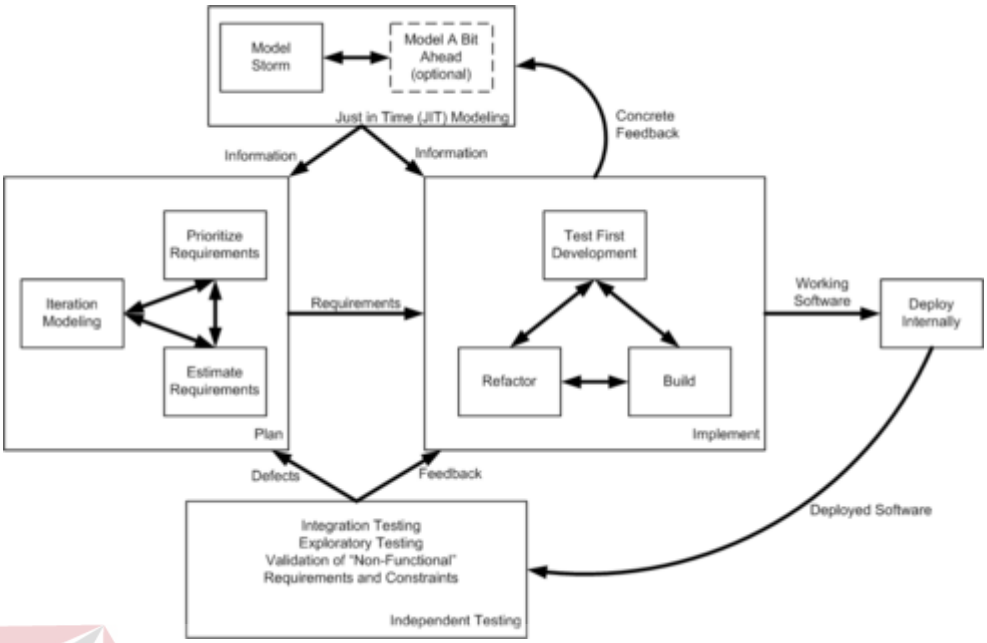


Gambar 2.20. Contoh *deployment diagram (Ambler, 2012)*

B. Iterasi Pemodelan

Pada tahap ini pengembang *agile* harus menyusun estimasi jadwal dan pekerjaan yang akan dilakukan pada setiap iterasi. Untuk melakukan estimasi yang tepat maka pengembang harus memahami pekerjaan seperti apa untuk mengimplementasikannya, dan itulah tujuan pemodelan pada tahap ini.

Pemodelan yang akan dibuat dapat dirancang berdasarkan informasi yang tersedia dari sketsa antar muka dan *user stories* yang telah dibuat. Berdasarkan informasi tersebut tim dapat melakukan estimasi dan menentukan prioritas fitur-fitur mana yang akan diselesaikan lebih dulu. Seperti tahap *agile* yang lain pemodelan pada tahap ini tidak perlu terlalu detail karena akan lebih disempurnakan saat *model storming* atau *Test-Driven Development* (TDD). Alur dari iterasi pemodelan ditunjukkan oleh gambar 2.21.



Gambar 2.21 Diagram iterasi pemodelan (Ambler, 2012)

C. Model Storming

Menurut Ambler (2012) *model storming* adalah *Just In Time* (JIT) *modeling* yang artinya pengembang mengidentifikasi masalah yang akan diselesaikan, jika dalam tim maka pengembang mengajak rekan yang dapat membantu, tim tersebut kemudian mengeksplorasi masalah dan kemudian kembali masing-masing melanjutkan pekerjaan seperti sebelumnya. Ambler (2012) menjelaskan bahwa *model storming* merupakan kegiatan yang mendadak dan berlangsung dalam hitungan menit, rata-rata lima sampai sepuluh menit. Pada sesi *model storming* terdapat dua tahap yaitu:

1. Analisis *Model Storming*: pengembang akan melakukan sesi untuk menganalisis kebutuhan. Dapat digunakan sketsa antar muka untuk menggambarkan apa yang akan dibuat dan dapat dibuat pula sebuah model karena *model storming* merupakan bagian dari iterasi.
2. Desain *Model Storming*: pengembang *agile* tidak selalu langsung menulis

kode begitu mengetahui *requirment* yang dibutuhkan. Tetapi pengembang akan melakukan *model storming* terlebih dahulu karena merupakan hal yang menyangkut arsitektur dan desain. Pengembang dapat menggunakan *sequence diagram*, CRC *cards* atau *flow chart* untuk mendesain pada *model storming*.

D. Implementasi via Test-Driven Development

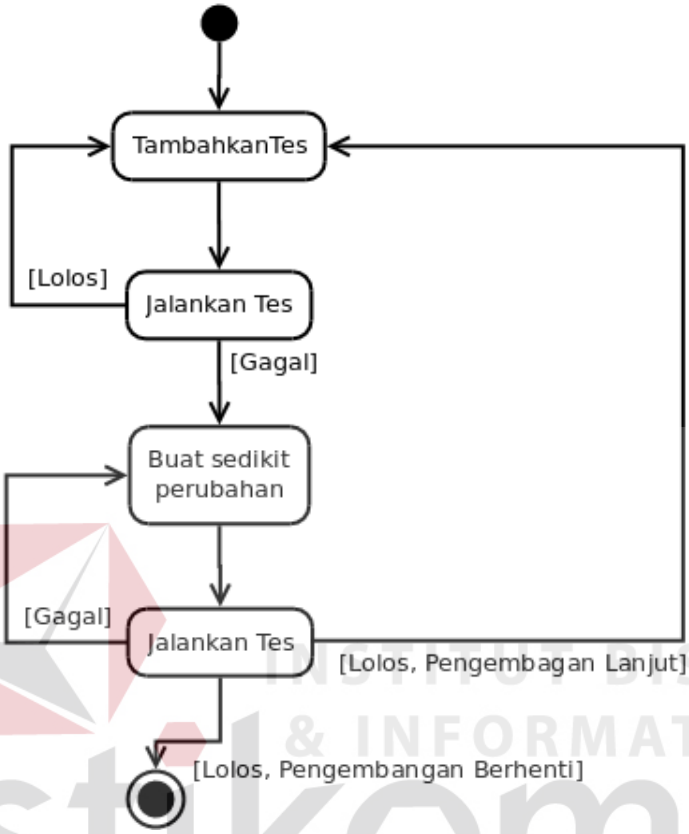
Spesifikasi yang telah dibuat pada *iteration modeling* dan *model storming* dikerjakan menggunakan pendekatan TDD. Dimana tes akan ditulis dulu berdasarkan model yang dibuat kemudian baru menulis kode aktual yang lengkap.

D.1. Test-Driven Development (TDD)

Menurut (Beck 2003; Astel 2003) dalam Ambler (2012) Test-Driven Development (TDD) adalah sebuah pendekatan evolusioner dalam pengembangan yang mengkombinasikan *test-first development* dimana anda menulis sebuah tes sebelum anda sepenuhnya menulis kode yang akan diperuntukkan untuk memenuhi tes dan *refactoring* tersebut. Menurut Fowler (1999) dalam Ambler (2012) *refactoring* adalah sebuah cara dalam melakukan restrukturisasi (perubahan) pada kode.

Secara singkat Ambler (2012) mendeskripsikan TDD sebagai kombinasi dari *refactoring* dan *Test-First Development* (TFD). Langkah-langkah pada TFD adalah menambahkan sebuah tes secara cepat, dibuat cukup untuk membuat sebuah kode akan gagal. Langkah berikutnya adalah menjalankan tes untuk memastikan apakah kode gagal atau tidak. Jika gagal maka ubah (*refactor*) bagian kode tersebut hingga dapat lolos. Jika menambahkan fungsionalitas baru pada

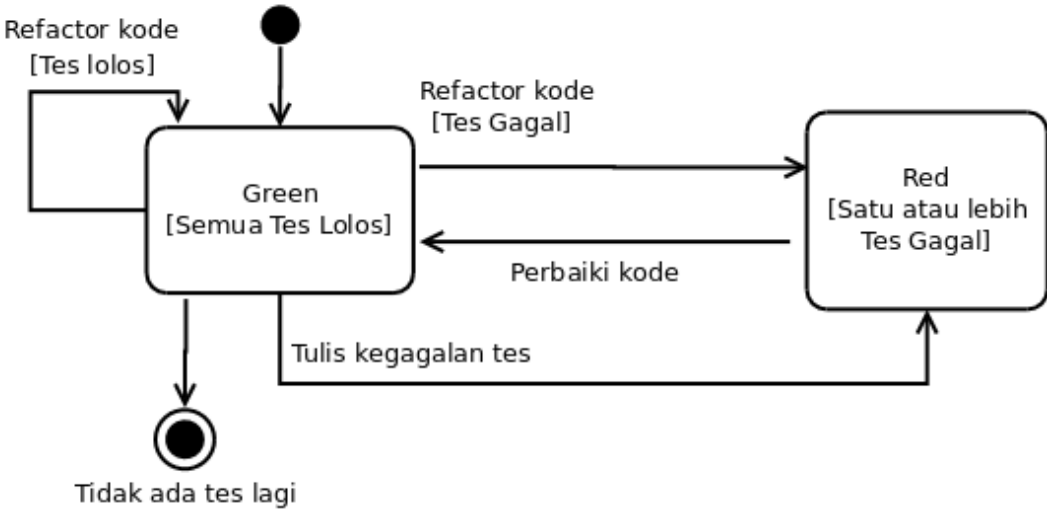
kode maka buat dulu tes dari fungsionalitas tersebut kemudian jalankan tes, *refactor* bagian kode hingga lolos tes. Lakukan berulang kali, ilustrasi dari alur tersebut ditunjukkan pada gambar 2.22.



Gambar 2.22. Test First Development (TFD)

Menurut Jeffries dalam Beck (2002) tujuan utama TDD adalah untuk menciptakan kode yang bersih dan bekerja. Beck (2002) mengemukakan sebuah istilah yang disebut *red/green/refactor*. Maksud dari istilah tersebut adalah:

1. *Red* – Tulis tes kecil yang tidak dapat bekerja, atau bahkan mungkin tidak dapat di-*compile*.
2. *Green* – Buatlah tes berhasil secepatnya, perbaiki semua kesalahan yang diperlukan selama proses tersebut.
3. *Refactor* – Eliminasi semua duplikasi yang tercipta dalam proses membuat tes itu berhasil.



Gambar 2.23. Alur red/green/refactor

Menurut Ambler (2012) keuntungan yang signifikan pada TDD adalah membantu pengembang mengambil beberapa langkah kecil saat menulis *software*. Karena hal tersebut terbukti lebih produktif daripada mencoba menulis kode langsung dalam jumlah yang banyak. Implikasi lain adalah kesalahan akan lebih mudah ditemukan dan diperbaiki.

D.2. Unit Testing Framework

Dalam TDD proses pengetesan dilakukan secara otomatis oleh sebuah *tools* pembantu yang disebut *Unit Testing Framework* (*Unit Testing Tools*). Tanpa bantuan *unit testing framework* maka proses TDD hampir mustahil dilakukan. Tersedia banyak *unit testing framework* yang bersifat *open source* diantaranya JUnit untuk Java, PHPUnit dan SimpleTest untuk PHP, dan NUnit untuk .NET.