

BAB II

LANDASAN TEORI

2.1. Sistem

Menurut McLeod, sistem adalah sekelompok elemen yang terintegrasi dengan maksud yang sama untuk mencapai suatu tujuan (McLeod,2001).

Menurut O'brien, sistem adalah sekelompok elemen yang terintegrasi, bekerja sama untuk mencapai suatu tujuan bersama dengan menerima *input* serta menghasilkan *output*. Sistem memiliki tiga komponen atau fungsi dasar yang saling berinteraksi, yaitu : (O'brien,2006)

1. *Input*

Merupakan proses memasukan data ke dalam sistem untuk selanjutnya diproses oleh sistem.

2. Pemrosesan

Merupakan proses pengolahan data yang merubah *input* menjadi *output*.

3. *Output*

Merupakan proses menampilkan data yang telah diolah untuk selanjutnya ditampilkan.

2.2. Data dan Informasi

Data adalah fakta atau observasi mentah, yang biasanya mengenai fenomena fisik atau transaksi bisnis. Informasi adalah data yang telah diubah menjadi konteks yang berarti dan berguna bagi para pemakai akhir tertentu (O'brien,2006).

Ciri-ciri informasi (Davis,1995) adalah sebagai berikut :

1. Benar atau salah sebuah informasi sangat tergantung pada penerimanya.

2. Baru, bila penerima informasi sama sekali belum pernah menerima informasi tersebut.
3. Tambahan, bila penerima informasi merasa informasi yang diterima sebelumnya belum lengkap atau benar.
4. Koreksi, bila informasi yang diterima sebelumnya salah.
5. Penegasan, bila informasi tersebut dibutuhkan sekali lagi oleh penerima.

2.3. Sistem Informasi

Sistem Informasi merupakan sistem yang mempunyai kemampuan untuk mengumpulkan informasi dari berbagai sumber dan menggunakan berbagai media untuk menampilkan informasi tersebut (McLeod,2001).

Komponen-komponen yang ada dalam sistem informasi menurut O'brien adalah sebagai berikut (O'brien,2006) :

1. Sumber daya sistem informasi
2. Sumber daya manusia
3. Sumber daya *hardware*
4. Sumber daya *software*
5. Sumber daya data
6. Sumber daya jaringan

Aktivitas-aktivitas yang ada dalam sistem informasi menurut O'brien adalah sebagai berikut :

1. Input sumber daya data
Input biasanya berbentuk proses pemasukan data seperti pencatatan dan pengubahan.
2. Pemrosesan data menjadi informasi
Pemrosesan data ini mengatur, menganalisis, dan memanipulasi data, hingga mengubahnya menjadi informasi bagi pengguna akhir.
3. *Output* produk informasi

Informasi yang merupakan hasil dari pemrosesan data akan dikirimkan ke pemakai akhir dan disediakan dalam bentuk *output*.

4. Penyimpanan sumber daya data

Penyimpanan adalah proses dimana informasi disimpan untuk kemudian digunakan kembali.

5. Pengendalian kinerja sistem

Sistem informasi harus menghasilkan umpan balik mengenai proses *input*, pemrosesan, *output*, dan penyimpanan. Kemudian, aktivitas sistem harus disesuaikan agar informasi yang dihasilkan tepat bagi para pengguna akhir.

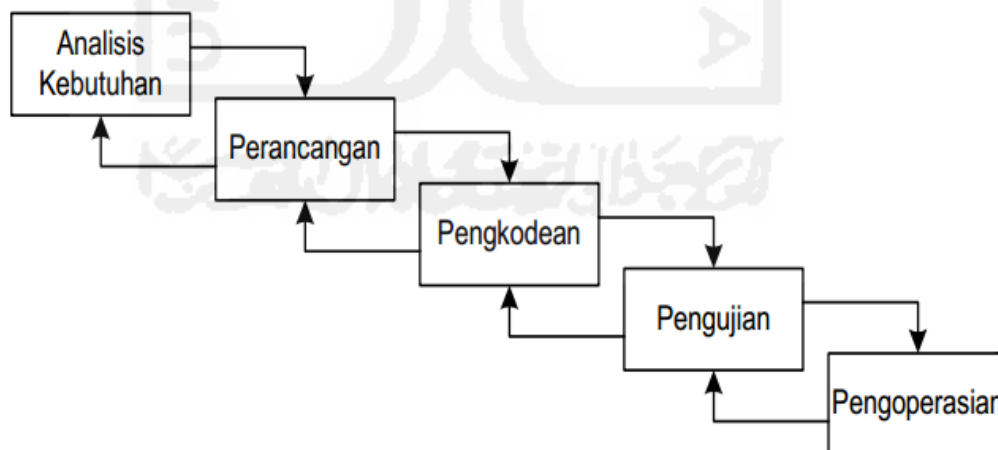
2.4. Metode *Waterfall*

Pada tahun 1960-an dan 1970-an, proyek pengembangan perangkat lunak merupakan pekerjaan yang sangat memakan biaya dan waktu karena pengembangan perangkat lunak ini difokuskan pada perencanaan dan pengendalian (Basili dan Musa, 1991). Kemunculan model air terjun adalah untuk membantu mengatasi kerumitan yang terjadi akibat proyek-proyek pengembangan perangkat lunak (Boehm, 1976). Seperti yang terlihat pada gambar 2.1, sebuah model air terjun memacu tim pengembang untuk memerinci apa yang seharusnya perangkat lunak lakukan (mengumpulkan dan menentukan kebutuhan sistem) sebelum sistem tersebut dikembangkan. Kemudian, model ini memungkinkan pemecahan misi pengembangan yang rumit menjadi beberapa langkah logis (desain, kode, pengujian dan seterusnya) dengan beberapa langkah yang pada akhirnya akan menjadi produk akhir yang siap pakai. Untuk memastikan bahwa sistem bisa dijalankan, setiap langkah akan membutuhkan validasi, masukan, dan kriteria yang ada.

Paradigma *Entry-Task-Validation* (ETVX) adalah kunci karakteristik dari model air terjun dengan beberapa keuntungan. Paradigma ini memungkinkan pelacakan yang lebih akurat dengan kemajuan proyek dan dapat mengidentifikasi dengan jelas apabila terjadi kesalahan. Hal ini juga memaksa organisasi untuk mengembangkan sistem perangkat lunak yang lebih rapi dan mudah diatur. Pendekatan struktural ini sangat penting bagi organisasi dengan proyek-proyek

yang sangat besar dan rumit. Kebutuhan hal ini kemudian menciptakan sistem dokumentasi yang selanjutnya digunakan untuk mencoba dan mempertahankan sistem (Davis, 1988).

Pada akhirnya, pendekatan ini membuat perangkat lunak yang lebih besar, mudah diatur dan selesai tepat pada waktunya tanpa biaya yang berlebihan. Pengalaman beberapa dekade terdahulu menunjukkan bahwa model air terjun sangat berguna dan berharga. Banyak para pengembang besar, apalagi yang telah berdiri lebih dahulu dan terlibat dalam pengembangan proses ini, telah menggunakannya. Kelompok tersebut terdiri dari perusahaan komersial, kontraktor pemerintah, dan kelompok pemerintah lainnya. Walaupun berbagai nama telah diberikan di setiap langkah proses, metodologi dasarnya tetap tidak berubah. Oleh karena itu, tahapan-tahapan kebutuhan sistem kadang disebut analisis sistem, analisis, dan pengumpulan kebutuhan data konsumen, atau analisis kebutuhan pengguna. Tahap pengembangan juga dapat dibagi menjadi beberapa desain tingkat tinggi dan desain tingkat terperinci, tahapan implementasi yang biasa disebut *code* dan *debug* yang merupakan tahapan pengujian yang juga mencakup uji tingkat komponen, uji tingkat produk, dan uji tingkat sistem.



Gambar 2.1 Metode *Waterfall*

2.5. Global Positioning System (GPS)

Global Positioning System (GPS) merupakan sistem navigasi menggunakan lebih dari 24 satelit MEO (*Medium Earth Orbit* atau *Middle Earth Orbit*) yang mengelilingi bumi sehingga penerima-penerima sinyal dipermukaan bumi dapat menangkap sinyalnya. Satelit mengorbit pada ketinggian 12.000 mil di atas bumi dan mampu mengelilingi bumi dua kali 24 jam. Satelit GPS secara kontinyu mengirimkan sinyal radio digital yang mengandung data lokasi satelit dan waktu, pada penerima yang berhubungan. Satelit GPS dilengkapi dengan jam atom yang mempunyai ketepatan waktu satu per satu juta detik. Berdasarkan informasi ini, stasiun penerima mengetahui berapa lama waktu yang digunakan untuk mengirim sinyal sampai kepada penerima di bumi. Semakin lama waktu yang digunakan untuk sampai ke penerima, berarti semakin jauh posisi satelit dari stasiun penerima. (Abidin, 2007).

2.5.1 Fungsi GPS

GPS memiliki fungsi antara lain :

1. Militer

GPS digunakan untuk keperluan perang, seperti menuntun arah bom, atau mengetahui posisi pasukan berada. Dengan cara ini maka kita bisa mengetahui mana teman mana lawan untuk menghindari salah target, ataupun menentukan pergerakan pasukan.

2. Navigasi

GPS banyak juga digunakan sebagai alat navigasi seperti kompas. Beberapa jenis kendaraan telah dilengkapi dengan GPS untuk alat bantu navigasi, dengan menambahkan peta, maka bisa digunakan untuk memandu pengendara, sehingga pengendara bisa mengetahui jalur mana yang sebaiknya dipilih untuk mencapai tujuan yang diinginkan.

3. Sistem Informasi Geografis

Untuk keperluan Sistem Informasi Geografis, GPS sering juga diikutsertakan dalam pembuatan peta, seperti gambar mengukur jarak perbatasan, ataupun sebagai referensi pengukuran.

4. Sistem Pelacakan Kendaraan

Kegunaan lain GPS adalah sebagai pelacak kendaraan, dengan bantuan GPS pemilik kendaraan/pengelola armada bisa mengetahui ada dimana saja kendaraan/aset bergerakanya berada saat ini. Bahkan saat ini, GPS dengan ketelitian tinggi bisa digunakan untuk memantau pergerakan tanah, yang ordenya hanya MM dalam setahun. Pemantauan pergerakan tanah berguna untuk memperkirakan terjadinya gempa, baik pergerakan vulkanik ataupun tektonik.

2.5.2 Cara Kerja GPS

Satelit GPS mengelilingi bumi dua kali sehari pada orbit yang sangat presisi dan memancarkan signal informasi ke GPS *receiver* yang ada di bumi. GPS *receiver* menerima informasi ini dan menggunakan *triangulation* / persegitigaan untuk menghitung lokasi pengguna yang tepat. GPS *receiver* selalu melakukan perbandingan antara waktu saat signal dipancarkan oleh sebuah satelit dengan waktu saat signal informasi diterima. Perbedaan waktu memberitahukan GPS *receiver* seberapa jauh keberadaan satelit pada saat ini, dengan melakukan pengukuran jarak dari beberapa satelit yang lain, maka GPS *receiver* dapat dengan mudah mengetahui dimana posisi pengguna yang saling tepat dan menampilkan data di depan layar alat penerima data GPS.

Sebuah GPS *receiver* harus mengunci signal dari paling sedikit 3 satelit untuk melakukan perhitungan posisi dua dimensi atau 2D (*latitude* dan *longitude*) dan melihat pergerakannya. Dengan empat setalit atau lebih di dalam sebuah penampakan, GPS *receiver* bisa melihat dan mengetahui posisi tiga dimensi atau 3D (*latitude*, *longitude* dan *altitude*). Jika posisi pengguna telah didapatkan, maka alat GPS bisa langsung menghitung formasi lain seperti kecepatan, posisi, jalan, jarak perjalanan, dan lain-lainnya (Abidin,2007).

2.5.3 Sumber Error Signal

Faktor-faktor yang dapat menurunkan signal GPS dan mempengaruhi akurasi data, antara lain (Abidin,2007) :

1. *Ionosphere dan Troposphere Delay*

Signal satelit menjadi lambat saat melewati atmosfer ataupun masalah cuaca buruk seperti hujan atau badai.

2. *Signal Multipath*

Kesalahan ini muncul ketika signal GPS diputuskan oleh objek seperti gedung tinggi atau permukaan batu besar sebelum diterima oleh GPS receiver.

3. *Receiver Clock Error*

Sebuah jam yang tertanam dalam GPS receiver tidak seakurat seperti jam atomic yang dipasang di dalam satelit GPS sehingga bisa mengakibatkan timing error atau kesalahan waktu

4. *Orbital Error* atau *Empiris Error*

Merupakan ketidakakuratan informasi lokasi satelit

5. Lokasi yang tidak terjangkau satelit

Lokasi yang tidak terjangkau satelit biasanya lokasi-lokasi seperti di dalam ruangan, di bawah air, ataupun di bawah tanah.

6. *Satelit geometry / shading*

Merupakan keadaan dimana satelit diletakkan di lokasi yang memiliki sudut lebih luas dengan sudut lainnya. Kesalahan ini dihubungkan langsung dengan posisi satelit pada saat kapanpun. Geometri yang kurang ideal diakibatkan oleh satelit-satelit yang diletakkan pada sebuah garis kelompok yang berjarak terlalu dekat.

2.6. Google API (*Application Programming Interface*)

Google API (*Application Programming Interface*) dapat dikatakan bagian dari Framework Google. Google menyediakan berbagai API yang sangat berguna bagi pengembang web maupun aplikasi desktop untuk memanfaatkan berbagai fitur yang disediakan oleh Google seperti misalnya: AdSense, Search Engine, Translation maupun YouTube (Jimmy, 2012).

API secara sederhana bisa diartikan sebagai kode program yang merupakan antarmuka atau penghubung antara aplikasi atau web yang dibuat dengan fungsi-

fungsi yang dikerjakan. Misalnya dalam hal ini Google API berarti kode program (yang disederhanakan) yang dapat ditambahkan pada aplikasi atau web untuk mengakses/menjalankan/memanfaatkan fungsi atau fitur yang disediakan Google.

Ada banyak API yang disediakan oleh Google, beberapa diantara API tersebut adalah :

1. Language API: untuk memanfaatkan fitur translation yang dimiliki Google.
2. Earth API: memanfaatkan fitur yang ada pada Google Earth.
3. Javascript API.
4. Maps API: memanfaatkan fitur yang ada pada Google Maps.
5. Search API: memanfaatkan fitur pencarian pada Google Search.
6. Visualization API: membuat grafik maupun chart dengan Google API.
7. YouTube API: memanfaatkan fitur yang ada pada YouTube misalnya untuk pencarian video.

2.7. Codeigniter

Codeigniter adalah aplikasi *open source* yang berupa *framework* dengan model *Model, View, Controller* (MVC) untuk membangun sistem yang dinamis dengan menggunakan PHP. Codeigniter memudahkan developer untuk membuat aplikasi web dengan cepat dan mudah dibandingkan dengan membuatnya dari awal (Upton, 2007). Codeigniter dirilis pertama kali pada 28 Februari 2006. Versi stabil terakhir adalah versi 2.1.3.

2.7.1 Framework

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

Ada beberapa alasan mengapa menggunakan *framework*, yaitu (Upton, 2007) :

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*, hingga *error handling*).
4. Lebih bebas dalam pengembangan jika dibandingkan *Content Management System* (CMS).

2.7.2 Design Pattern: Model, View, Controller (MVC)

Model View Controller merupakan suatu konsep yang cukup populer dalam pembangunan aplikasi web, berawal pada bahasa pemrograman *Small Talk*, MVC memisahkan pengembangan aplikasi berdasarkan komponen utama yang membangun sebuah aplikasi seperti manipulasi data, *user interface*, dan bagian yang menjadi kontrol aplikasi. Terdapat 3 jenis komponen yang membangun suatu MVC pattern dalam suatu aplikasi yaitu (Upton, 2007) :

1. *View*, merupakan bagian yang menangani *presentation logic*. Pada suatu aplikasi web bagian ini biasanya berupa file template HTML, yang diatur oleh *controller*. *View* berfungsi untuk menerima dan merepresentasikan data kepada *user*. Bagian ini tidak memiliki akses langsung terhadap bagian *model*.
2. *Model*, biasanya berhubungan langsung dengan *database* untuk memanipulasi data (*insert, update, delete, search*), menangani validasi dari bagian *controller*, namun tidak dapat berhubungan langsung dengan bagian *view*.
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.

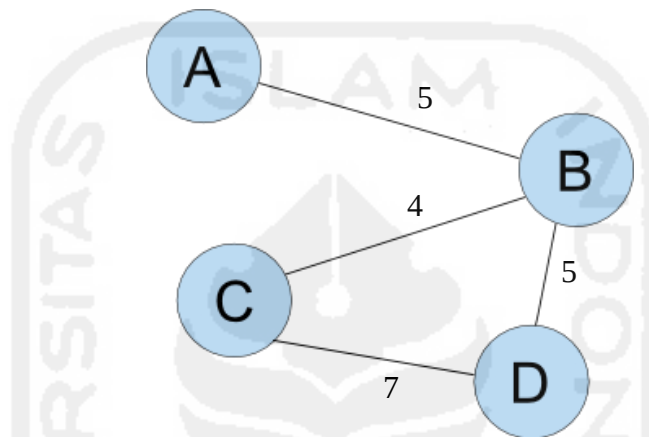
Dengan menggunakan prinsip MVC suatu aplikasi dapat dikembangkan sesuai dengan kemampuan pengembangnya, yaitu *programmer* yang menangani bagian *model* dan *controller*, sedangkan *designer* yang menangani bagian *view*, sehingga penggunaan arsitektur MVC dapat meningkatkan *maintanability* dan organisasi kode. Walaupun demikian dibutuhkan komunikasi yang baik antara *programmer* dan *designer* dalam menangani variabel-variabel yang akan ditampilkan.

Ada beberapa kelebihan Codeigniter (CI) dibandingkan dengan *Framework* PHP lain,

1. 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.
2. 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 pengaturan standar, anda hanya perlu mengubah sedikit saja file pada folder *config*.
3. Banyak komunitas: dengan banyaknya komunitas CI ini, memudahkan kita untuk berinteraksi dengan yang lain, baik itu bertanya atau teknologi terbaru.
4. Dokumentasi yang sangat lengkap : Setiap paket instalasi Codeigniter sudah disertai *user guide* yang sangat bagus dan lengkap untuk dijadikan permulaan, bahasanya pun mudah dipahami.

2.8. Jalur Terpendek

Pada dasarnya permasalahan pencarian jalur terpendek antar kota/lokasi merupakan pencarian jalur terpendek antar titik yang telah diketahui koordinatnya. Dengan mengetahui konsep pencarian jalur terpendek antar titik, untuk selanjutnya dapat diterapkan pada pencarian jalur terpendek pada berbagai kota/lokasi yang ingin diketahui. (Andhy, 2014).



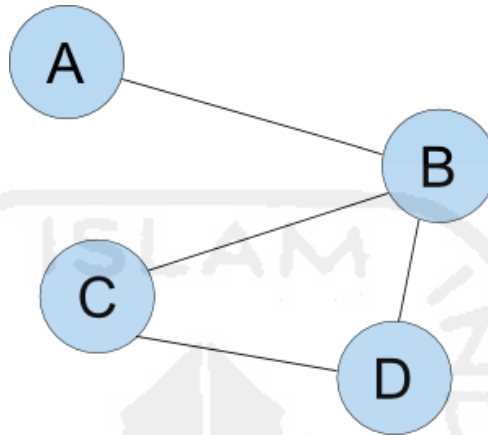
Gambar 2.2 Ilustrasi Graf Jalur Terpendek Tidak Berarah Namun Memiliki Bobot

Tabel 2.1 Tabel Jarak Antar Titik

Titik	A	B	C	D
A	0	5	-	-
B	5	0	4	5
C	-	4	0	7
D	-	5	7	0

Terdapat dua jenis kasus yang bisa diturunkan dari Gambar 2.2. Kasus pertama adalah mengetahui jarak antar node yang ditujukan dengan garis penghubung antar titik. Kasus yang kedua adalah dengan mengetahui koordinat titik saja. Gambar 2.2 merupakan jenis kasus pertama yaitu dengan mengetahui jarak

antar titik. Sedangkan gambar 2.3 merupakan jenis kasus yang kedua, yaitu dengan mengetahui titik koordinatnya saja. Untuk kasus pertama, penyelesaian cenderung lebih mudah karena jarak antar titik telah diketahui pada Tabel 2.1.



Gambar 2.3 Ilustrasi Graf Jalur Terpendek Tidak Memiliki Arah dan Tidak Memiliki Bobot

Tabel 2.2 Koordinat Titik Antar Kota/Lokasi

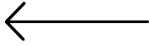
	X	Y
A	20	50
B	10	25
C	18	10
D	50	40

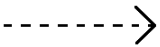
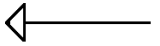
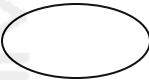
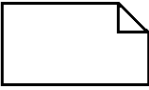
Sedangkan untuk kasus kedua yang telah ditunjukkan pada gambar 2.3, misalnya titik yang telah ditentukan mempunyai koordinat sebagaimana pada Tabel 2.2.

2.9. Use Case Diagram

Use Case adalah komponen-komponen utama yang menggerakkan model proses tersebut. Sebuah *use case* dapat didefinisikan sebagai satu bagian fungsionalitas yang memberi seorang pengguna sebuah hasil serupa nilai. Semua pengembangan *use case* dapat digabungkan ke dalam sebuah model *use case* yang menggambarkan fungsionalitas lengkap dari sistem. Model *use case* dapat disamakan dengan spesifikasi fungsional dalam sebuah model proses pengembangan perangkat lunak tradisional. *Use case* dikembangkan bersama pengguna dan dibentuk dalam UML (*Unified Modeling Language*). *Use case* menyajikan kebutuhan-kebutuhan untuk perangkat lunak dan digunakan pada keseluruhan model proses (Simarmata, 2010).

Tabel 2.3 Notasi *Use Case Diagram*

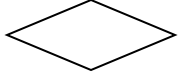
No	Simbol	Nama	Keterangan
1		<i>Actor</i>	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan <i>use case</i>
2		<i>Dependency</i>	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (<i>independent</i>) akan mempengaruhi elemen yang bergantung padanya
3		<i>Generalization</i>	Hubungan dimana objek anak (<i>descendent</i>) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (<i>ancestor</i>)

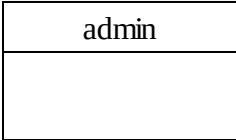
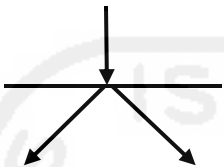
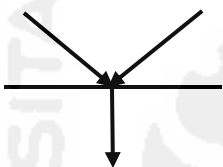
4		<i>Include</i>	Menspesifikasikan bahwa <i>use case</i> sumber secara eksplisit
5		<i>Extend</i>	Menspesifikasikan bahwa <i>use case</i> target memperluas perilaku dari <i>use case</i> sumber pada suatu titik yang diberikan
6		<i>Association</i>	Apa yang menghubungkan objek satu dengan objek lainnya
7		<i>System</i>	Menspesifikasikan paket yang menampilkan sistem secara terbatas
8		<i>Use Case</i>	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang dapat menghasilkan suatu nilai yang terukur bagi aktor
9		<i>Collaboration</i>	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah elemen-elemennya (sinergi)
10		<i>Note</i>	Elemen fisik yang ada saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi

2.10. Activity Diagram

Diagram aktivitas merupakan suatu jenis diagram khusus dari *statechart* diagram yang mempresentasikan *state-state* dari transisi-transisi yang terjadi pada akhir operasi-operasi. Diagram aktivitas dapat digunakan untuk mempresentasikan *state-states* serta transisi-transisi yang bersifat sinkrin, sementara *statechart* diagram digunakan untuk mempresentasikan keadaan asinkron. Implementasi diagram aktivitas sangat mirip dengan implementasi untuk *statechart diagram-state*, Start dalam diagram aktivitas mempresentasikan awal operasi, dan *state* End dalam diagram aktivitas merepresentasikan akhir operasi, dan *state-state* yang berada di antaranya merupakan *state-state* yang harus diimplementasikan ke dalam bentuk kode-kode bahasa pemrograman. Karena diagram aktivitas bersifat sinkron, jika operasi dimulai semua aksi akan berjalan dengan pendefinisian yang relatif mudah. John Satzinger menyatakan (Nugroho, 2009) dalam buku *System Analysis and Design in Changing World* yang menyatakan bahwa “Activity Diagram adalah sebuah diagram alur kerja yang menjelaskan berbagai kegiatan pengguna (atau sistem), orang yang melakukan masing-masing aktivitas, dan aliran sekuensial dari aktivitas-aktivitas tersebut”.

Tabel 2.4 Notasi Activity Diagram

No	Simbol	Nama	Keterangan
1		<i>Start State</i>	Menunjukkan dimana aktivitas atau aliran kerja akan dimulai
2		<i>End State</i>	Menunjukkan dimana aktivitas atau aliran kerja berakhir
3		<i>Activities</i>	Menunjukkan aktivitas atau kegiatan dalam aliran dalam kerja
4		<i>Decision</i>	Menunjukkan dimana sebuah kondisi atau

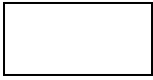
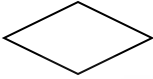
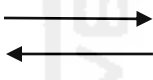
			keputusan perlu dibuat dalam suatu aliran kerja
5		<i>Swimlane</i>	Menunjukkan siapa yang berperan atau bertanggung jawab dalam sebuah aktivitas
6		<i>Synchronization Fork</i>	Menunjukkan dua atau lebih langkah dalam aliran kerja
7		<i>Synchronization Join</i>	

2.11. Flowchart

Flowchart atau bagan alir adalah suatu bagan yang menggambarkan atau mempresentasikan suatu algoritma atau prosedur untuk menyelesaikan masalah (Sismoro, 2005). Fungsi dari *flowchart* adalah untuk mengevaluasi jalannya suatu program sebelum direalisasikan dalam bentuk sintaks program sehingga kesalahan alur dalam sintaks dapat diminimalisir. *Flowchart* terbagi menjadi dua, yaitu *flowchart system* dan *flowchart program*. Dalam penelitian ini penulis menggunakan *flowchart program*, yaitu bagan yang menggambarkan urutan logika dari suatu prosedur pemecahan masalah.

Tabel 2.5 Notasi *Flowchart*

No	Simbol	Nama	Keterangan
1		<i>terminal symbol</i>	menunjukkan awal dan akhir dari program
2		<i>preparation symbol</i>	memberikan nilai awal pada suatu variabel atau <i>counter</i>

3		<i>processing symbol</i>	menunjukkan pengolahan aritmatika dan pemindahan data
4		<i>input/output symbol</i>	menunjukkan proses <i>input</i> atau <i>output</i>
5		<i>decision symbol</i>	mewakili operasi perbandingan logika
6		<i>predefined symbol</i>	proses yang ditulis sebagai subprogram, yaitu prosedur/fungsi
7		<i>connector symbol</i>	penghubung pada halaman yang sama
8		<i>off page connector symbol</i>	penghubung pada halaman yang berbeda
9		<i>flow line symbol</i>	arah proses

2.12. **Black Box Testing**

Terfokus pada apakah unit program memenuhi kebutuhan (*requirement*) yang disebutkan dalam spesifikasi. Pada *black box testing*, cara pengujian hanya dilakukan dengan menjalankan atau mengeksekusi unit atau modul, kemudian diamati apakah hasil dari unit itu sesuai dengan proses bisnis yang diinginkan (Al Fatta, 2007). Metode ujicoba *black box* memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba *black box* memungkinkan pengembang *software* untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba *black box* bukan merupakan alternatif dari ujicoba *white box*, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode *white box*.

Uji coba *black box* berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan *interface*
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi

Tabel 2.6 Contoh Tabel Pengujian *Black Box*

No	Skenario pengujian	Test case	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
1	Mengosongkan semua isian data <i>log in</i> , lalu langsung mengklik tombol 'Masuk'	Nama: - Kata sandi: -	Sistem akan menolak akses login dan menampilkan pesan "Mohon isi dulu nama admin dan kata sandi"	Sesuai harapan	Valid
2	Hanya mengisi data nama <i>admin</i> dan mengosongkan data kata sandi, lalu langsung mengklik tombol 'Masuk'.	Nama: admin Kata sandi: -	Sistem akan menolak akses <i>log in</i> dan menampilkan pesan "Mohon isi dulu salah satu data yang masih kosong"	Sesuai harapan	Valid

3	Hanya mengisi data kata sandi dan mengosongkan data nama <i>admin</i> , lalu langsung mengklik tombol 'Masuk'.	Nama: - Kata sandi: 123	Sistem menerima dan masuk ke dalam sistem	Tidak sesuai harapan	Tidak Valid
---	--	----------------------------	---	----------------------	-------------

2.13. Algoritma Dijkstra

Algoritma Dijkstra adalah algoritma yang ditemukan oleh Edger Dijkstra. Algoritma ini bertujuan untuk menemukan jalur terpendek berdasarkan bobot terkecil dari satu titik ke titik lainnya. Misalkan titik menggambarkan gedung dan garis menggambarkan jalan, maka algoritma Dijkstra melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik (Siang, 2006).

Misalkan :

$$V(G) = \{ V_1, V_2, \dots, V_n \}.$$

$$L = \text{Himpunan titik-titik } \in V(G) \text{ yang sudah terpilih (titik permanen) dalam jalur path terpendek.}$$

$$D(j) = \text{Jumlah bobot path terkecil } V_1 \text{ ke } V_j.$$

$$w(i,j) = \text{Bobot garis dari titik } V_i \text{ ke titik } V_j.$$

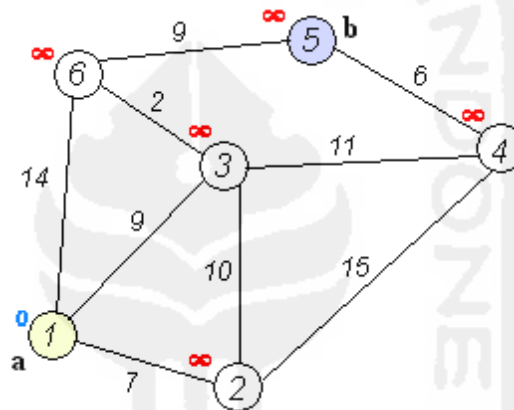
$$w^*(1,j) = \text{Jumlah bobot path terkecil dari } V_1 \text{ ke } V_j.$$

Algoritma Dijkstra untuk mencari path terpendek adalah sebagai berikut :

1. Inisialisasi : $L = \{ \}$; $V = (V_2, V_3, \dots, V_n)$
2. Untuk $i = 2, \dots, n$, lakukan $D(j) = W(1,j)$
3. Selama $V_n \in L$ (V_n belum merupakan titik permanen), lakukan :

- a. Pilih titik $V_k \in V-L$ (tidak titik permanen) dengan $D(k)$ terkecil.
 $L = L \cup \{ V_k \}$ (jadikan V_k menjadi titik permanen)
- b. Untuk setiap $V_j \in V-L$ lakukan :
 Jika $D(k) + W(k,j) < D(j)$ dengan $D(k) + W(k,j)$

Menurut algoritma di atas, jalur terpendek dari titik v_1 ke v_n adalah melalui titik-titik dalam L secara berurutan,, dan jumlah bobot path terkecilnya adalah $D(n)$.



Gambar 2.4 Contoh Keterhubungan Antar Titik Dalam Algoritma Dijkstra

Pertama-tama tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap sesuai pada gambar 2.4. Inilah urutan logika dari algoritma Dijkstra:

Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi). Set semua node “belum terjamah” dan set node awal sebagai “node keberangkatan”

Dari node keberangkatan, pertimbangkan node tetangga yang belum terjamah dan hitung jaraknya dari titik keberangkatan. Sebagai contoh, jika titik

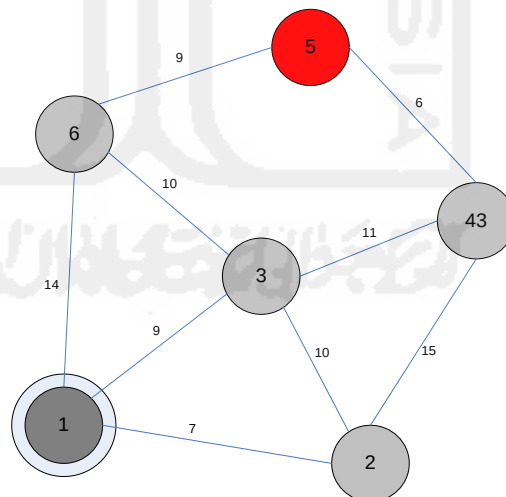
keberangkatan A ke B memiliki bobot jarak 6 dan dari B ke node C berjarak 2, maka jarak ke C melewati B menjadi $6+2=8$. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru.

Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah terjamah sebagai “node terjamah”. Node terjamah tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.

Set “node belum terjamah” dengan jarak terkecil (dari node keberangkatan) sebagai “node keberangkatan” selanjutnya dan lanjutkan dengan kembali ke step 3

Dibawah ini penjelasan langkah per langkah pencarian jalur terpendek secara rinci dimulai dari node awal sampai node tujuan dengan nilai jarak terkecil.

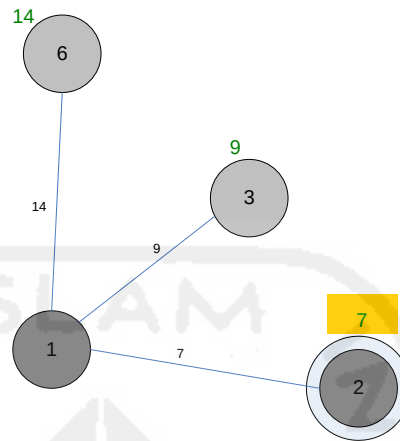
Gambar 2.5 merupakan langkah 1 Dijkstra. Node awal 1, Node tujuan 5. Setiap edge yang terhubung antar node telah diberi nilai



Gambar 2.5 Contoh Kasus Dijkstra - Langkah 1

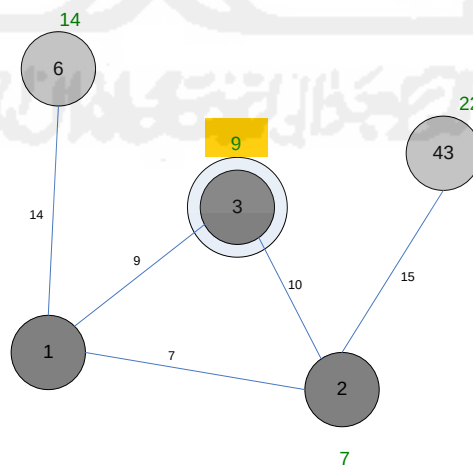
Pada gambar 2.6 Dijkstra melakukan kalkulasi terhadap node tetangga yang terhubung langsung dengan node keberangkatan (node 1), dan hasil yang

didapat adalah node 2 karena bobot nilai node 2 paling kecil dibandingkan nilai pada node lain, nilai = 7 ($0+7$).



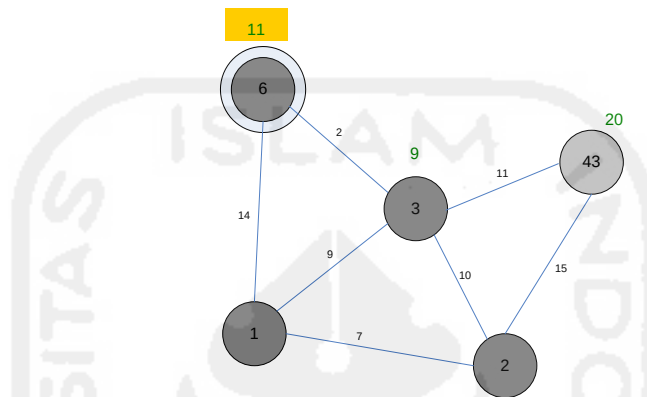
Gambar 2.6 Contoh Kasus Dijkstra - Langkah 2

Pada gambar 2.7 node 2 diset menjadi node keberangkatan dan ditandai sebagai node yang telah terjamah. Dijkstra melakukan kalkulasi kembali terhadap node-node tetangga yang terhubung langsung dengan node yang telah terjamah. Dan kalkulasi Dijkstra menunjukkan bahwa node 3 yang menjadi node keberangkatan selanjutnya karena bobotnya yang paling kecil dari hasil kalkulasi terakhir, nilai 9 ($0+9$).



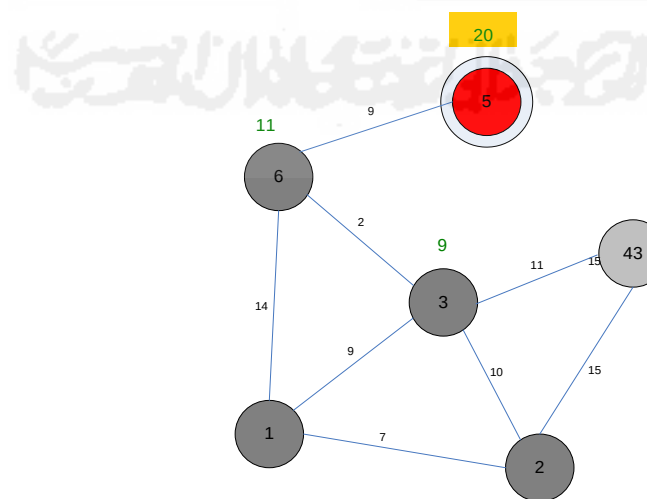
Gambar 2.7 Contoh Kasus Dijkstra - Langkah 3

Pada gambar 2.8 perhitungan berlanjut dengan node 3 ditandai menjadi node yang telah terjamah. Dari semua node tetangga belum terjamah yang terhubung langsung dengan node terjamah, node selanjutnya yang ditandai menjadi node terjamah adalah node 6 karena nilai bobot yang terkecil, nilai 11 ($9+2$).



Gambar 2.8 Contoh Kasus Dijkstra - Langkah 4

Pada gambar 2.9 node 6 menjadi node terjamah, Dijkstra melakukan kalkulasi kembali, dan menemukan bahwa node 5 (node tujuan) telah tercapai lewat node 6. Jalur terpendeknya adalah 1-3-6-5, dan nilai bobot yang didapat adalah 20 ($11+9$). Bila node tujuan telah tercapai maka kalkulasi Dijkstra dinyatakan selesai.



Gambar 2.9 Contoh Kasus Dijkstra - Langkah 5