BAB II
LANDASAN TEORI

II.1 Pariwisata

Pariwisata menurut KBBI adalah yang berhubungan dengan perjalanan untuk rekreasi, pelancongan, turisme [**8**].

II.2 Sistem Informasi

Sistem adalah kumpulan komponen dimana masing-masing komponen memiliki fungsi yang saling berinteraksi dan saling tergantung serta memiliki satu kesatuan yang utuh untuk bekerja mencapai tujuan tertentu [**9**]. Sedangkan informasi adalah data yang sudah diolah ke dalam sebuah formulir bentuk yang bermanfaat dan dapat digunakan untuk manusia [**10**].

Jika disimpulkan, definisi dari sistem informasi adalah sekumpulan komponen yang memiliki fungsi yang saling ketergantungan dalam satu kesatuan utuh dalam bentuk satu data yang telah diolah dalam sebuah bentuk yang dapat digunakan untuk manusia mencapai tujuan tertentu.

Dari pendapat lain, dapat dijelaskan bahwa sistem informasi merupakan seperangkat prosedur yang terorganisasi dengan sistematik yang jika dilaksanakan akan menyediakan informasi yang dapat dimanfaatkan dalam proses pembangunan keputusan [**11**].

II.3 *Data Mining*

Data mining adalah serangkaian proses untuk menggali nilai tambah berupa informasi yang selama ini tidak diketahui secara manual dari suatu basis data dengan melakukan penggalian pola-pola dari data dengan tujuan untuk memanipulasi data menjadi informasi yang lebih berharga yang diperoleh dengan cara mengekstraksi dan mengenali pola yang penting atau menarik dari data yang terdapat dalam basis data [**12**].

Definisi lain dari data *mining* adalah teknik yang digunakan untuk menemukan pola data yang berguna dari banyak data [**13**]. Adapun tahapan *data mining* sebagai berikut [**3**]:

1. *Data Selection*

   Dalam penelitian ini dilakukan tahapan *selection,* dengan memisahkan atribut tempat wisata dan aktivitas.

1. *Data Cleaning*

   Proses menghilangkan *noise* atau data yang tidak konsisten, dalam penelitian ini dilakukan proses data *cleaning* untuk menghilangkan data yang satu transaksinya hanya memiliki satu *item* atau tidak memiliki keterkaitan dengan *item* lainnya.

2. *Data mining*

   Pada proses ini, peneliti mencari pola atau informasi menarik dari data yang telah tersedia menggunakan teknik ataupun metode yang digunakan. Dalam penelitian ini metode yang digunakan adalah *association rule* dan menggunakan algoritma *apriori.*

3. *Interpretation/Evaluation*

   Pola informasi yang dihasilkan harus mudah dimengerti dan diperiksa apakah bertentangan dengan fakta atau hipotesis yang ada sebelumnya.

II.4 *Association Rule*

Association rule merupakan suatu prosedur untuk mencari hubungan antar-*item* dalam suatu *dataset* [**3**]. Association rule (aturan asosiasi) akan menemukan pola tertentu untuk mengasosiasikan data yang satu dengan data yang lain. Untuk mencari association rule dari suatu kumpulan data, tahap pertama yang harus dilakukan adalah mencari frequent itemset terlebih dahulu. Frequent itemset adalah sekumpulan item yang sering muncul secara bersamaan. Setelah semua pola frequent itemset ditemukan, barulah mencari aturan assosiatif atau aturan keterkaitan yang memenuhi syarat yang telah ditentukan [**3**].

Jika diasumsikan bahwa barang yang dijual di swalayan adalah semesta, maka setiap barang akan memiliki boolean variabel yang akan menunjukkan keberadaannya atau tidak barang tersebut dalam satu transaksi atau satu keranjang belanja. Pola boolean yang didapat digunakan untuk menganalisa barang yang dibeli secara bersamaan. Pola tersebut dirumuskan dalam sebuah association rule.

Sebagai contoh konsumen biasanya akan membeli komputer dan *software* antivirus pada saat yang sama yang ditujukkan sebagai berikut [**3**]:

Komputer → *software* antivirus [*support* = 2, *confidence* = 60%]

Association rule diperlukan suatu variabel ukuran yang ditentukan sendiri oleh *user* untuk menentukan batasan sejauh mana atau sebanyak apa *output* yang diinginkan *user*. *Support* dan *confidence* adalah sebuah ukuran kepercayaan dan kegunaan suatu pola yang telah ditemukan. Nilai *support* 2 menunjukkan bahwa keseluruhan dari total transaksi konsumen membeli komputer dan *software* antivirus secara bersamaan yaitu sebanyak 2. Sedangkan *confidence* 60% yaitu menunjukkan bila konsumen membeli komputer dan pasti membeli *software* antivirus sebesar 60% [**3**].

II.5 Algoritma *Apriori*

Algoritma *apriori* adalah salah satu algoritma yang melakukan pencarian *frequent itemset* dengan menggunakan teknik *association rule*. Algortima *apriori* menggunakan pengetahuan frekuensi atribut yang telah diketahui sebelumnya untuk memproses informasi selanjutnya. Pada algoritma *apriori*, menentukan kandidat yang mungkin muncul dengan cara memperhatikan *minimum support* dan *minimum confidence*. *Support* adalah nilai pengunjung atau persentase kombinasi sebuah item dalam database [**3**].

Nilai *support* sebuah *item* diperoleh dengan rumus sebagai berikut: [**3**]

$$Support = (\mathrm{A} \cap \mathrm{B}) = \frac{\sum \text{Jumlah Transaksi yang Mengandung A dan B}}{\sum \text{Total Transaksi}} * 100\% \quad (2.1)$$

Sedangkan *confidence* adalah nilai kepastian yaitu kuatnya hubungan antar *item* dalam sebuah apriori. *Confidence* dapat dicari setelah pola frekuensi munculnya sebuah item ditemukan. Rumus untuk menghitung *confidence* adalah sebagai berikut [**3**]:

$$Confidence = \frac{\sum \text{Jumlah transaksi yang mengandung A dan B}}{\sum \text{Transaksi yang mengandung A}} * 100\% \quad (2.2)$$

Proses utama yang dilakukan dalam algoritma *apriori* untuk mendapat *frequent itemset* yaitu [**3**]:

1. *Join* (Penggabungan)

Proses ini dilakukan dengan cara pengkombinasian item dengan item yang lainnya hingga tidak dapat terbentuk kombinasi lagi.

2. *Prune* (Pemangkasan)

Proses pemangkasan yaitu hasil dari item yang telah dikombinasikan kemudian dipangkas dengan menggunakan minimum support yang telah ditentukan oleh *user*. Pada iterasi ke-k akan ditemukan semua itemset yang memiliki k item, disebut dengan k-itemset. Tiap iterasi terdiri dari dua tahap, yaitu :

a. Gunakan *frequent* (k-1) itemset untuk membangun kandidat *frequent* k- *itemset*.

b. Gunakan *scan database* dan pencocokan pola untuk mengumpulkan hitungan untuk kandidat *itemset*.

Adapun langkah-langkah proses perhitungan *association rule* dengan algoritma *apriori* sebagai berikut [**3**]:

1. Sistem men-*scan database* untuk mendapat kandidat 1-*itemset* (himpunan *item* yang terdiri dari 1 *item*) dan menghitung nilai *support*-nya. Kemudian nilai *support*-nya tersebut dibandingkan dengan minimum *support* yang telah ditentukan. Jika nilainya lebih besar atau sama dengan *minimum support*, maka *itemset* tersebut termasuk *large itemset*.

2. *Itemset* yang tidak termasuk dalam *large itemset* tidak diikutkan dalam iterasi selanjutnya (di pangkas).

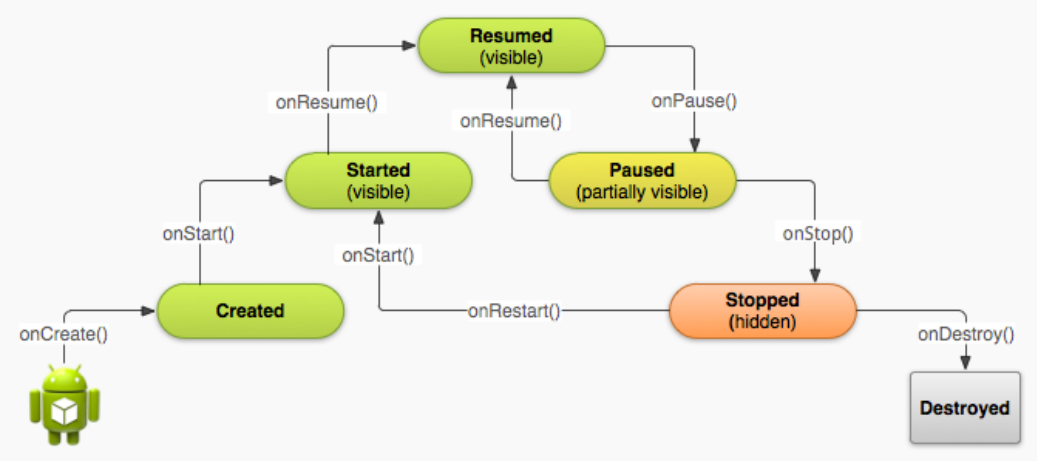
3. Pada iterasi kedua, sistem akan menggunakan hasil *large itemset* pada iterasi pertama (L1) untuk membentuk kandidat *itemset* kedua (L2). Pada iterasi selanjutnya sistem akan menggunakan hasil *large itemset* pada iterasi sebelumnya (Lk-1) untuk membentuk kandidat itemset berikut (Lk). Sistem akan menggabungkan (*join*) Lk-1 dengan Lk-1 untuk mendapatkan Lk. Seperti pada iterasi sebelumnya sistem akan menghapus (memangkas) kombinasi *itemset* yang tidak termasuk dalam *large itemset*.

4. Setelah dilakukan operasi *join*, maka pasangan *itemset* baru hasil proses *join* tersebut dihitung *support*-nya.
5. Proses pembentuk kandidat yang terdiri dari proses penggabungan dan pemangkasan akan terus dilakukan hingga himpunan kandidat *itemset*-nya *null*, atau sudah tidak ada lagi kandidat yang akan terbentuk.
6. Setelah itu, dari hasil *frequent itemset* tersebut dibentuk *association rule* yang memenuhi nilai *support* dan *confidence* yang telah ditentukan.
7. Pada pembentukan associaton rule, nilai yang sama dianggap sebagai satu nilai.
8. Association rule yang terbentuk harus memenuhi nilai minimum yang telah ditentukan.
9. Untuk setiap large itemset L, cari himpunan bagian L yang tidak kosong. Untuk setiap himpunan bagian tersebut, dihasilkan rule dengan bentuk aB(L-a) jika support-nya (a) lebih besar dari minimum support.

II.6 Android

Android adalah sistem operasi smartphone layar sentuh seperti iOS iPhone dan OS BlackBerry yang dalam pengembangannya dipimpin oleh google. Sistem operasi ini bersifat Open Source dan dikembangkan berdasarkan kernel Linux yang disematkan pada gadget, baik itu handphone atau tablet [**14**]. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2015.

Di dalam android terdapat activity dimana komponen ini memberi interaksi antara user dan aplikasi yang dibangun melalui user interface. Activity ini memiliki Siklus hidup yang dinamakan Android Life Cycle. Flowchart siklus hidup tersebut dapat dilihat pada Gambar II.1



Gambar II.1 *Android Life Cycle*

Gambar di atas menjelaskan mengenai sebuah ilustrasi sederhana dari siklus hidup *activity* yang dinyatakan sebagai langkah piramida. Hal ini menunjukkan bagaimana untuk setiap *callback* yang digunakan untuk mengambil aktivitas langkah menuju *state* atas kemudian dilanjutkan ada metode *callback* yang mengambil langkah menurun. Kegiatan ini juga dapat kembali ke keadaan dari *state pause* dan *stop*. Siklus hidup *activity* ini menjadi sangat penting ketika kita hendak membuat aplikasi berbasis android. Gunanya adalah agar aplikasi yang dibangun berjalan dengan baik. Berikut penjelasan pada masing-masing *state*:

1. *onCreate*(), Method ini dipanggil ketika *activity* pertama kali dibuat.
2. *onStart*(), Method ini dipanggil ketika sebuah *activity* tampil ke pengguna.
3. *onResume*(), Method ini dipanggil ketika *activity* yang berjalan pada saat itu dihentikan sementara (*paused*) dan *activity* sebelumnya dijalankan kembali(*resumed*). (Hasil dari method *OnRestart*())
4. *onPause*(), Method ini dipanggil ketika *activity* di hentikan sementara (pause) dan berikutnya ketika dijalankan kembali akan berada dalam posisi resume dan memanggil method OnResume()
5. *onStop*(), Method ini dipanggil ketika *activity* tidak lagi tampak kepada pengguna
6. on*Destroy*(), Method ini dipanggil sebelum *activity* dihancurkan (*destroy*) oleh sistem (baik secara manual maupun untuk kepentingan pelonggaran memori).

Banyak sekali fitur yang dapat dimanfaatkan pada platform android ini. Diantaranya adalah *Location Based Services* (LBS) . *Location Based Services* (LBS) dapat dimanfaat untuk layanan yang menggunakan informasi geografis dalam memberikan informasi lokasi kepada pengguna, sebagai petunjuk posisi atau lokasi piranti *mobile* pengguna berada, dan menemukan rute jalan sesuai permintaan. LBS dapat digambarkan sebagai satu layanan yang berada pada pertemuan tiga teknologi yaitu: *Geographic Information System* (GIS), *Internet Service*, dan *Mobile Devices* [**15**].

II.7 *Structure Analysis and Design*

Struktur analisis dan desain adalah sebuah metodologi yang di gunakan pada rekayasa perangkat lunak untuk mendeskripsikan sistem ke arah fungsional. Pendekatan ini memecahkan masalah-masalah dalam aktivitas bisnis menjadi bagian-bagian kecil yang dapat disatukan kembali menjadi satu kesatuan yang utuh untuk memecahkan masalah [**16**].

Tools yang dapat digunakan pada pendekatan analisis pengembangan sistem secara terstruktur adalah:

1. ERD (*Entity Relationship Diagram*) adalah satu model jaringan yang menggunakan susunan data yang disimpan dalam sistem. ERD merupakan model jaringan data yang menekankan pada struktur dan hubungan antar data [**17**].
2. Diagram Konteks adalah diagram yang menggambarkan sebuah sistem secara menyeluruh yang selanjutnya akan lebih didetailkan oleh DFD. Konteks diagram juga biasa disebut sebagai DFD level 0.
3. Data *Flow* Diagram atau biasa di singkat DFD merupakan serangkaian diagram yang menggambarkan kegiatan-kegiatan yang ada dalam satu sistem. Teknik pembangunan DFD dimulai dengan menggambarkan sistem secara global dan dilanjutkan dengan analisis masing-masing bagian. Pada awalnya digambarkan konteks diagram yang menggambarkan sebuah sistem secara menyeluruh yang akan diinvestigasi. Konteks diagram tersebut dapat dikatakan sebagai DFD

level 0. Analisis sistem yang lebih detail selanjutnya dapat dilakukan dengan menggambarkan DFD level 1,2 dan seterusnya.

4. Spesifikasi Proses merupakan tabel yang berisi keterangan deskripsi dari semua proses yang terdapat di DFD. Logika proses harus dituliskan secara jelas baik menggunakan bahasa deskriptif atau *pseudo code* (tidak boleh campuran.
5. Kamus Data (*Data Dictionary*) merupakan fakta tentang data dan kebutuhan-kebutuhan informasi dari sistem informasi. Dengan menggunakan *data dictionary*, analis sistem dapat mendefinisikan data yang mengalir dalam sistem dengan lengkap [**18**].

II.8 *Object Oriented Analysis and Design*

Analisis dan desain berorientasi objek adalah cara baru dalam memikirkan satu masalah dengan menggunakan model yang dibuat menurut konsep sekitar dunia nyata. Tujuan dari analisis berorientasi objek adalah untuk mengembangkan model yang menggambarkan perangkat lunak komputer karena bekerja untuk memenuhi seperangkat persyaratan yang ditentukan user [**19**]. *Tools* yang dapat digunakan pada pendekatan analisis pengembangan sistem secara objek dapat menggunakan UML.

Unified Modelling Language (UML) adalah sebuah bahasa yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menggunakan *class* dan *operation object* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa bahasa berorientasi objek [**20**]. Dalam membangun block UML ada 3 hal yang harus diperhatikan, yaitu *object* (memodelkan konsep), *relationship* (mengkoneksikan *object*), dan *diagram* (*grouping* yang saling mengkoneksikan antara *object* dan *relationship.* Diagram yang umum dipakai dalam analisis dan desain adalah:

1. Use Case Diagram

Use Case Diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *Use case* merepresentasikan sebuah interaksi antara aktor

dengan sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa use case yang di-*include* akan dipanggil setiap kali use case yang meng-*include* dieksekusi secara normal. Sebuah use case dapat di- include oleh lebih dari satu use case lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang serupa. Sebuah use case juga dapat meng-*extend usecase* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar use case menunjukkan bahwa use case yang satu merupakan spesialisasi dari yang lain.

Dasar menentukan sebuah *use case* adalah *use case* merupakan sesuatu yang menyediakan beberapa hasil terukur kepada pengguna atau sistem eksternal. *Use case* harus memiliki sangat jelas kriteria lulus / gagal. Pengembang, tester, penulis teknis, dan pengguna harus secara eksplisit tahu apakah sistem memenuhi kasus penggunaan atau tidak. Setiap bagian dari *use case* yang memenuhi tes sederhana ini mungkin menjadi kandidat yang baik untuk *use case* [**21**].

2. *Use Case Scenario*

Sebuah diagram yang menunjukkan *use case* dan aktor mungkin menjadi titik awal yang bagus, tetapi tidak memberikan detail yang cukup untuk desainer sistem untuk benar-benar memahami persis bagaimana sistem dapat terpenuhi. Cara terbaik untuk mengungkapkan informasi penting ini adalah dalam bentuk penggunaan *use case scenario* berbasis teks per u*se case*nya. Berikut adalah dasar format penulisan *use case scenario* [**21**]. Dasar pembangunan *use case scenario* dapat dilihat pada Tabel II-1 Dasar Pembangunan Use Case Scenario

Tabel II-1 Dasar Pembangunan Use Case Scenario

Use Case Name	Berisi nama dari *Use case* yang akan digunakan
Goal In Context	Menjelaskan apa yang aktor coba untuk dapatkan dari *Use case*
Description	Menjelaskan gambaran dari *Use case*
Related Use Case	Daftar *Use case* yang berhubungan dengan *Use case* tersebut
Successful End Condition	Kondisi *Use case* jika berhasil
Failed End Condition	Kondisi *Use case* jika gagal
Actors	Daftar aktor yang dapat mengakses *Use case*
Trigger	Aktifitas yang dilakukan untuk mengawali *Use case*

Main Flow	Step	Action
	1	Deskripsi urutan aksi dari aktifitas *Use case*
	2	
	3	
Extension	Step	Branching Action
	2.1	Deskripsi urutan aksi lain selain urutan aksi utama
	2.2	

3. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan *Output* tertentu. Diawali dari apa yang men-trigger aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *Output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, message akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message [**20**]. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk objek *boundary*, *controller* dan *persistent entity*.

4. *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) satu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta

hubungan satu sama lain seperti pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok:

1. Nama dan *stereotype*
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut:

1. *Private,* tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected,* hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public,* dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface,* yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*. *Class* memiliki tipe-tipe *relationship,* diantaranya :

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain.
2. Agregasi, yaitu hubungan yang menyatakan bagian terdiri atas dimana ketika satu class di share atau direferensikan kepada objek yang ada di class lain.
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Komposisi, yaitu jenis relasi class diagram yang kuat dimana jika sebuah class tidak bisa berdiri sendiri dan harus merupakan bagian dari class yang lain, maka class tersebut memiliki relasi Composition terhadap class tempat dia bergantung tersebut. Sebuah relationship composition digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.

5. Depedensi, salah satu jenis relasi class diagram yang lemah dimana objek dalam suatu class akan bekerja sangat singkat dengan objek yang ada pada *class* lain.

5. Activity Diagram

   Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing aliran berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Sebuah aktivitas dapat direalisasikan oleh satu use case atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa object *swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu [**20**].

II.9 Web Service

Web Service adalah sekumpulan *application logic* beserta objek-objek dan metode-metode yang dimilikinya yang terletak di satu server yang terhubung ke internet [**22**]. Tujuan dari teknologi ini adalah untuk memudahkan beberapa aplikasi atau komponennya untuk saling berhubungan dengan aplikasi lain dalam sebuah organisasi maupun diluar organisasi menggunakan standar yang tidak terikat platform (*platform-neutral*) dan tidak terikat akan bahasa pemrograman yang digunakan (*language-neutral*). Hal tersebut dapat terjadi karena penggunaan XML standar yang didukung oleh banyak perusahaan besar di dunia, yang digunakan untuk bertukar data. Selain XML, terdapat pula format pertukaran data lain yaitu JSON.

II.10 JSON (Javascript Object Notation)

JSON (JavaScript Object Notation) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generated*) oleh komputer [**23**]. JSON terbuat dari dua struktur yaitu:

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

JSON menggunakan bentuk sebagai berikut:

1. Objek

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

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

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

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

II.11 Pengujian *Black Box*

Metode pengujian *blackbox* fokus pada keperluan penelusuran kesalahan fungsional dari *software*. Uji coba *blackbox* 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.

Teknik pengujian black box terdiri dari 10 jenis diantaranya Equivalence Partitioning, Boundary Value Analysis/Limit Testing, Comparison Testing, Sample Testing, Robustness Testing, Behavior Testing, Requirement Testing, Performance Testing, Endurance Testing, Cause-Effect Relationship Testing. Salah satunya yang akan dibahas adalah Equivalence partitioning.

Equivalence partioning merupakan metode ujicoba blackbox yang membagi domain input dari program menjadi beberapa kelas data dari kasus ujicoba yang dihasilkan. Kasus uji penanganan single yang ideal menemukan sejumlah kesalahan (misalnya : kesalahan pemrosesan dari seluruh data karakter) yang merupakan syarat lain dari suatu kasus yang dieksekusi sebelum kesalahan umum diamati.