BAB II
LANDASAN TEORI

2.1 Badan Pengembangan SDM Provinsi Lampung

Badan Pengembangan Sumber Daya Manusia Daerah Provinsi Lampung merupakan Organisasi Perangkat Daerah di Pemerintah Provinsi Lampung, yang mempunyai tugas pokok melaksanakan pengembangan sumber daya manusia aparatur atau pendidikan dan pelatihan di lingkup Pemerintah Provinsi dan Kabupaten/Kota se-Provinsi Lampung, dimana tugas pokok tersebut tertuang didalam PERGUB Lampung Nomor 4 Tahun 2018 tentang perubahan atas peraturan Gubernur Lampung nomor 92 Tahun 2016 tentang kedudukan, susunan organisasi, tugas dan fungsi serta tata kerja Badan Pengembangan sumber daya manusia daerah Provinsi Lampung.

2.2 Jadwal Diklat

Sistem pelatihan mengadopsi sistem SKS di perkuliahan, yang disebut Jam Pelatihan (Jamlat). Umumnya setiap mata pelatihan memiliki 10-12 jamlat per hari mulai dari pukul 08.00 sampai pukul 16.00 atau 18.00 yang diselingi 2 kali *coffee break* dan makan siang.. Contoh Jadwal Pelatihan dapat disusun secara umum sebagai berikut :

HARI KE-	HARI	TANGGAL	WAKTU	MATERI	JP
	Senin		09.00-selesai	Masuk Asrama	-
1	Selasa		07.30-07.45	Persiapan Upacara Pembukaan	-
			07.45-10.00	Upacara Pembukaan	3 JP
			10.15-12.30	Pembekalan Mentor	3 JP
			13.30-15.45	Strategi dan Kebijakan Pegembangan SDM ASN	3 JP
			16.00-18.15	Overview Kebijakan	3 JP

				Diklat	
2	Rabu		05.30-07.00	Pembinaan Kesegaran Jasmani	2 JP
			07.30-07.45	Apel Pagi	-
			07.45-10.00	Dinamika Kelompok	3 JP
			10.15-12.30	Inovasi (Konsep Inovasi)	3 JP
			13.30-15.45	Lanjutan	3 JP
3	Kamis		05.30-07.00	Pembinaan Kesegaran Jasmani	2 JP
			07.30-07.45	Apel Pagi	-
			07.45-10.00	Wawasan Kebangsaan	3 JP
			10.15-12.30	Lanjutan	3 JP
			13.30-15.45	Lanjutan	3 JP
			16.00-18.15	Wawasan Kebangsaan	3 JP
			05.30-07.00	Senam Kesegaran Jasmani	2 JP
4	Jumat		07.30-07.45	Apel Pagi	-
			07.45-10.00	Wawasan Kebangsaan	3 JP
			10.15-12.30	Lanjutan	3 JP
			13.30-15.45	Integritas	3 JP
			16.00-18.15	Lanjutan	3 JP
5.	Sabtu		05.30-07.00	Pembinaan Kesegaran Jasmani	2 JP
			07.45-10.00	Integritas	3 JP
			10.15-12.30	Lanjutan	3 JP
			13.30-15.45	Lanjutan	3 JP
			16.00-18.15	Lanjutan	3 JP

2.3 Android

Menurut Nazruddin Safaat H (Pemrograman aplikasi *mobile smartphone* dan *tablet* PC berbasis *android* 2012:1) android adalah sebuah sistem operasi pada handphone yang bersifat terbuka dan berbasis pada sistem operasi *Linux*. *Android* bisa digunakan oleh setiap orang yang ingin menggunakannya pada perangkat mereka. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang akan digunakan untuk bermacam peranti bergerak. Awalnya, *Google Inc*. membeli *Android Inc*., pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan *Android*, dibentuklah *Open Handset Alliance*, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk *Google, HTC, Intel, Motorola, Qualcomm, T-Mobile,* dan *Nvidia.* Pada saat perilisan perdana *Android*, 5 November 2007.

2.4 PHP: Hypertext Preprocessor

PHP adalah bahasa pemrograman *script server-side* yang didesain untuk pengembangan web. Selain itu, PHP juga bisa digunakan sebagai bahasa pemrograman umum. PHP di kembangkan pada tahun 1995 oleh Rasmus Lerdorf, dan sekarang dikelola oleh The PHP Group. Situs resmi PHP beralamat di http://www.php.net. PHP disebut bahasa pemrograman *server side* karena PHP diproses pada komputer server. Hal ini berbeda dibandingkan dengan bahasa pemrograman *client-side* seperti *JavaScript* yang diproses pada web browser (*client)*.

2.5 Pengertian *SQL*

Menurut A.S. Rosa dan Shalahuddin. M(2014:46): "*SQL* (*Structured Query Language*) adalah bahasa yang digunakan untuk mengelola data pada *Relation* DBMS *(Database Management System)*.". Menurut Muhammad Sadeli (2014:10) "MySQL merupakan *Database* yang menghubungkan script php menggunakan perintah *query* dan *escaps character* yang sama dengan php. MySQL mempunyai

tampilan *client* yang mempermudah anda dalam mengakses *database* dengan kata sandi untuk mengizinkan proses yang bisa anda lakukan."

2.6 Basis Data

Basis data atau *database* merupakan kumpulan data satu dengan data lainnya yang tersimpan dalam satu tempat penyimpanan luar dan membutuhkan suatu perangkat lunak untuk menjalankannya (Sutanta, 2004).

2.7 Android Studio

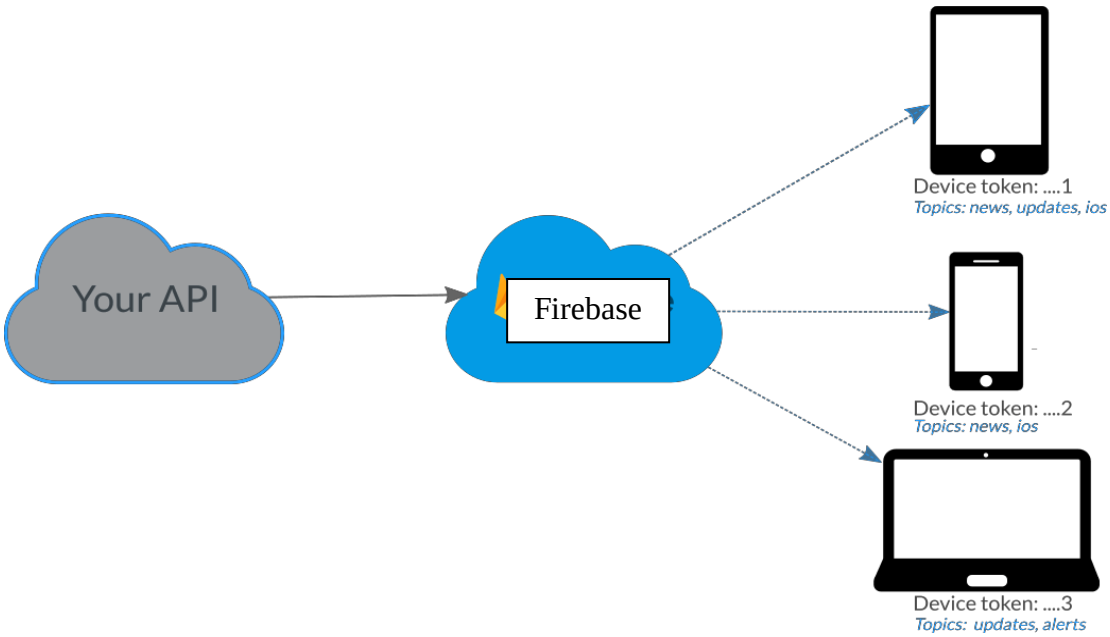
Android Studio adalah sebuah *IDE* untuk *Android Development* yang diperkenalkan *google* pada acara *Google I/O* 2013. *Android studio* merupakan pengembangan dari *Eclipse IDE,* dan dibuat berdasarkan IDE Java populer, yaitu IntelliJ IDEA. *Android Studio* merupakan IDE resmi untuk pengembangan aplikasi Android. Sebagai pengembangan dari *Eclipse, Android Studio* mempunyai banyak fitur-fitur baru dibandingkan dengan *Eclipse IDE*. Berbeda dengan *Eclipse* yang menggunakan *Ant, Android Studio* menggunakan *Gradle* sebagai *build environment*.

2.8 Push Notification

Push notification merupakan salah satu fitur yang ditawarkan oleh *Firebase Cloud Messaging* yang di gunakan untuk interaksi ke user. Implementasi dari salah satu fitur yang dimiliki oleh *Firebase Cloud Messaging* ini mencakup 2 komponen utama untuk mengirim dan menerima pesan, komponen yang pertama adalah lingkungan tepercaya seperti *Cloud Functions for Firebase* atau *server* aplikasi yang akan digunakan untuk membuat, menargetkan, dan mengirim pesan. Komponen yang kedua adalah aplikasi klien *iOS, Android,* atau *Web (JavaScript)* yang menerima pesan.

Notifikasi akan memudahkan pengguna untuk mendapakan informasi baik informasi pengingat ataupun informasi lainnya. Beberapa teknologi yang menyediakan tools notifikasi salah satunya adalah *Firebase Cloud Messaging* (FCM). Menurut mursito (2017) *Firebase* adalah sebuah penyedia layanan berupa *database realtime* dan *backend* yang data digunakan pada

berbagai *platform*. *Backend* sendiri adalah sebuah bagian dalam kode aplikasi yang berhubungan langsung dengan isi *database*. Dengan *firebase,* pengembang aplikasi tidak perlu membuat *backend* sendiri melainkan memakai *API* yang telah di sediakan oleh *firebase* sehingga pengembangan aplikasi dapat di persingkat. *Firebase* dikembangkan dengan menggunakan *database MongoDB* sehingga *firebase* menggunakan tipe *database NoSQL*. Karena memakai tipe *database NoSQL* maka struktur *database* dari *firebase* bersifat fleksibel dan cepat sehingga cocok untuk di gunakan pada aplikasi berbasis *mobile*.



Gambar 2.1 Arsitektur *Firebase Cloud Messaging*

2.9 Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement,* membuat analisis & desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek (Rosa A.S dan M. Shalahuddin, 2014).

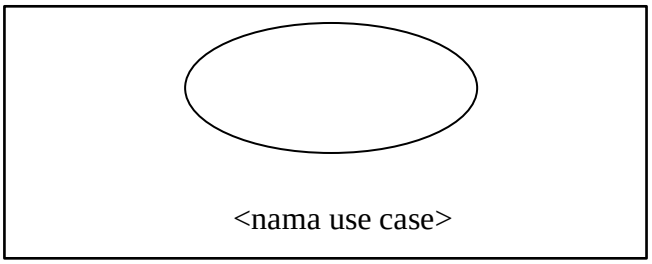
2.9.1. Diagram UML

2.9.1.1. Use Case Diagram

Diagram yang menggambarkan interaksi anatara sistem dengan eksternal sistem dan pengguna. Dengan kata lain, secara garis menggmbarkan siapa yang akan menggunakan sistem dan dengan cara apa pengguna mengharapkan untuk berinteraksi dengan sistem (Whitten, 2004). Dalam *use case diagram* memiliki pemodelan sebagai berikut (Whitten, 2004) :

1. *Use case*

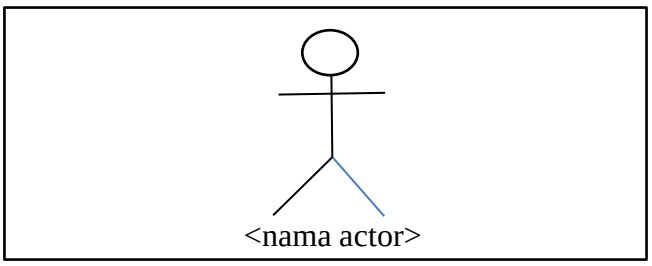
   Use case merupakan urutan langkah-langkah yang secara tindakan saling terkait (scenario), baik otomatis maupun secara manual.



Gambar 2.2. Use Case (Whitten, 2004)

2. *Actor* (Pelaku)

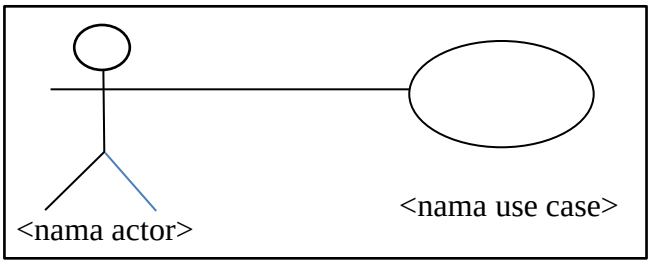
   Actor merupakan pengguna yang perlu berinteraksi dengan sistem untuk pertukan informasi.



Gambar 2.3 Actor (Whitten, 2004)

3. *Relationship* (Hubungan)

   Pada diagram *use case*, *relationship* digambarkan sebuah garis antara dua symbol, pemaknaan *relationship* berbeda-beda tergantung bagaimana garis tersebut digambar dan tipe symbol apa yang digunakan untuk menghubungkan garis tersebut.

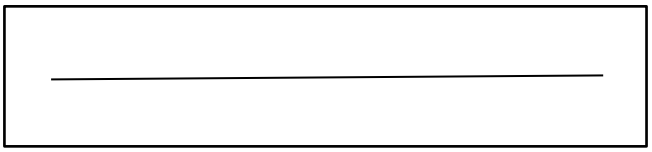


Gambar 2.4 Relationship (Whitten, 2004)

Berikut ini adalah perbedaan di antara *relationship* yang ada pada sebuah diagram *use case.*

a. *Association*

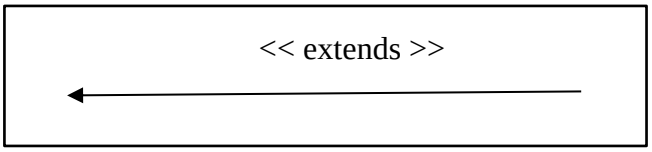
Association merupakan *relationship* antara *actor* dengan *use case* dimana terjadi interkasi di antara mereka.



Gambar 2.5 Association (Whitten, 2004)

b. *Extends*

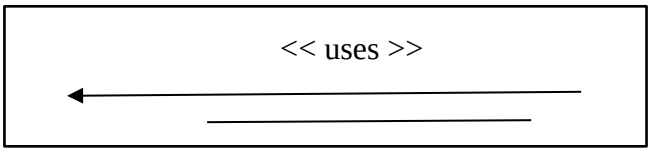
Extends use case merupakan *use case* yang terdiri dari langkah yang terekstraksi dari *use case* yang lebih kompleks untuk menyederhanakan masalah dank arena itu memperluas fungsinya.



Gambar 2.6 Extends (Whitten, 2004)

c. *Uses (includes)*

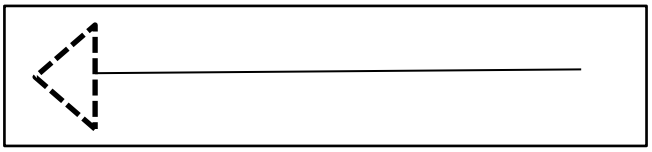
Hubungan *uses* menggambarkan bahwa satu *use case* seluruhnya meliputi fungsionalitas dari *use case* lainnya.



Gambar 2.7 Uses (Whitten, 2004)

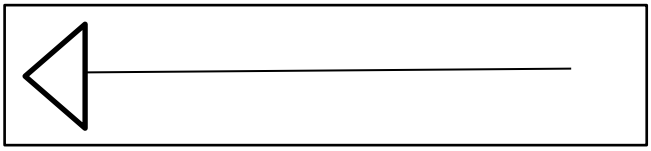
d. *Depends on*

Terkadang suatu *use case* memiliki ketergantungan pada *use case* yang lainnya yang bertujuan untuk menentukan urutan dalam pengembangan *use case*, ketergantungan ini dimodelkan menggunkan *depends on relationship.*



Gambar 2.8 Depends on (Whitten, 2004)

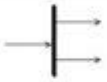
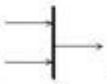
e. Hubungan *inheritance* terjadi ketika dua atau lebih *actor* menggunakan *use case* yang sama



Gambar 2.9 Inheritance (Whitten, 2004)

2.9.1.2. Activity Diagram

Activity diagram secara garis digunakan untuk menggambarkan rangkaian aliran altifitas baik proses bisnis atau *use case* (Whitten, 2004). *Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* merupakan *state* diagram khusus, dimana sebagian besar *state* adlah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing).*

Simbol	Keterangan
	Start Point
	End Point
	Activities
	Fork (Percabangan)
	Join (Penggabungan)
	Decision
Swimlane	Sebuah cara untuk mengelompokkan activity berdasarkan Actor (mengelompokkan activity dalam sebuah urutan yang sama)

Gambar 2.10 Komponen Activity Diagram (Whitten, 2004)

2.9.1.3. Sequence Diagram

Sequence Diagram secara grafis menggambarkan bagaiamana *object* berinteraksi dengan satu sama lain melalui pesan pada eksekusi sebuah *use case* atau operasi. Diagram ini mengilustrasikan bagaimana pesan terkirim dan diterima diantara *object* dan *sequence* (ruang waktu) (Whitten, 2004).

	*Object*
	*Actor*
	Lifeline
	Message
	Message (Call)
	*Message (Return)*
	Activation

Gambar 2.11 Komponen Sequence Diagram (Whitten, 2004)

2.10 Black Box Testing

Black box testing adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Jadi dianalogikan seperti kita melihat suatu kotak hitam, kita hanya bisa melihat penampilan luarnya saja, tanpa tahu ada apa di balik bungkus hitamnya. Sama seperti pengujian *black box*, mengevaluasi hanya dari tampilan luarnya *(interface)*, fungsionalitasnya tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detilnya (hanya mengetahui input dan output).

Pengujian *black box* adalah metode pengujian perangkat lunak yang menguji fungsionalitas aplikasi yang bertentangan dengan struktur internal atau kerja. Pengetahuan khusus dari kode aplikasi atau struktur internal dan pengetahuan pemrograman pada umumnya tidak diperlukan. Uji kasus dibangun di sekitar spesifikasi dan persyaratan, yakni aplikasi apa yang seharusnya dilakukan. Menggunakan deskripsi eksternal perangkat lunak, termasuk spesifikasi, persyaratan, dan desain untuk menurunkan uji kasus. Tes ini dapat menjadi fungsional atau non-fungsional, meskipun biasanya fungsional. Perancang uji memilih input yang valid dan tidak valid dan menentukan output yang benar, tidak ada pengetahuan tentang struktur internal benda uji itu.

Metode uji dapat diterapkan pada semua tingkat pengujian perangkat lunak antara lain unit, integrasi, fungsional, sistem dan penerimaan. Ini biasanya terdiri dari kebanyakan jika tidak semua pengujian pada tingkat yang lebih tinggi, tetapi juga bisa mendominasi unit pengujian juga.

Pengujian pada *black box* berusaha menemukan kesalahan antara lain fungsi-fungsi yang tidak benar atau hilang, kesalahan *interface*, kesalahan dalam struktur data atau akses database eksternal, kesalahan kinerja, dan inisialisasi dan kesalahan terminasi.

2.11.1 Teknik *Black Box Testing*

Teknik khas *Black Box Testing* meliputi:

1. *Decision Table*

   Decision Table adalah cara yang tepat belum kompak untuk model logika rumit, seperti diagram alur dan *if-then-else* dan *switch*, laporan kasus, kondisi mengaitkan dengan tindakan untuk melakukan, tetapi dalam banyak kasus melakukannya dengan cara yang lebih elegan.

2. *All Pairs Testing*

   All-pairs testing atau *pairwise testing* adalah metode pengujian perangkat lunak kombinatorial bahwa untuk setiap pasangan parameter masukan ke sistem (biasanya, sebuah algoritma perangkat lunak), tes semua kombinasi yang mungkin diskrit parameter tersebut. Menggunakan vektor uji dipilih dengan cermat, hal ini dapat dilakukan jauh lebih cepat daripada pencarian lengkap semua kombinasi dari semua parameter, dengan *parallelizing* pengujian pasangan parameter. Jumlah tes biasanya O (nm), dimana n dan m adalah jumlah kemungkinan untuk masing-masing dua parameter dengan pilihan yang paling.

3. *State Transition Table*

   Dalam teori automata dan logika sekuensial, *state transition table* adalah tabel yang menunjukkan apa yang negara (atau negara dalam kasus robot terbatas *non deterministic*) suatu semi automaton terbatas atau mesin finite state akan pindah, berdasarkan kondisi saat ini dan masukan lainnya. Sebuah tabel negara pada dasarnya adalah sebuah tabel kebenaran di mana beberapa input adalah kondisi saat ini, dan output termasuk negara berikutnya, bersama dengan keluaran lain. *state transition table* adalah salah satu dari banyak cara untuk menentukan mesin negara, cara lain menjadi diagram negara, dan persamaan karakteristik.

4. *Equivalence Partitioning*

   Equivalence partitioning adalah pengujian perangkat lunak teknik yang membagi data masukan dari unit perangkat lunak menjadi beberapa partisi data dari mana *test case* dapat diturunkan. Pada prinsipnya, uji kasus dirancang untuk menutupi setiap partisi minimal sekali.

5. *Boundary Values Analysis*

Boundary value analysis merupakan suatu teknik pengujian perangkat lunak di mana tes dirancang untuk mencakup perwakilan dari nilai-nilai batas. Nilai-nilai di tepi sebuah partisi kesetaraan atau sebesar nilai terkecil di kedua sisi tepi. Nilai dapat berupa rentang masukan atau keluaran dari komponen perangkat lunak. Karena batas-batas tersebut adalah lokasi umum untuk kesalahan yang mengakibatkan kesalahan perangkat lunak mereka sering dilakukan dalam kasus-kasus uji.

2.11.2 Kelebihan dan kekurangan *Black Box Testing*

Kelebihan dari pengujian *Black Box* antara lain:

a. Spesifikasi program dapat ditentukan di awal.
b. Dapat digunakan untuk menilai konsistensi program.
c. Testing dilakukan berdasarkan spesifikasi.
d. Tidak perlu melihat kode program secara detail.

Dan kekurangan dari pengujian *Black Box* adalah:

Bila spesifikasi program yang dibuat kurang jelas dan ringkas, maka akan sulit membuat dokumentasi setepat mungkin.